Large Action Spaces are Problematic in Deep Reinforcement Learning

Marc Zeller
Osnabrück University
Institue of Cognitive Science
mazeller@uni-osnabrueck.de

Robin Gratz
Osnabrück University
Institue of Cognitive Science
rgratz@uni-osnabrueck.de

Tim Niklas Witte
Osnabrück University
Institue of Computer Science
wittet@uni-osnabrueck.de

***Abstract*—Deep Q networks (DQNs) and policy gradient algorithms such as Proximal Policy Optimization (PPO) are the state-of-the-art approaches in Deep Reinforcement Learning. However, these fail with increasing size of the action space. To show this behavior, a Candy Crush gym was developed which is parameterizable in the field size and number of candies. This paper demonstrates that new approaches such as the Decision Transformer are able to achieve stable performance results independent of the action space size.**

1. Introduction

Being able to let a computer learn how to play and master games on a (sub-)human level has always been a core interest in reinforcement learning (RL). The Arcade Learning Environment (ALE) [1], emerged to be the perfect baseline to test and improve RL approaches on 2600 Atari games. The huge and possibly continuous action spaces of these games became the next challenge that was tackled by the combination of RL with Deep Artificial Neural Networks termed deep reinforcement learning (DRL), which showed to be better suited for these kinds of problems. For more complex games however, popular DRL approaches still often fall short.

In the following, we will show how state-of-the-art DRL approaches such as DQN and PPO struggle with increasing action spaces in contrast to a Decision Transformer. This is done by introducing a custom made gym for the game Candy Crush on which the three approaches are run.

2. Candy Crush

Reaching an optimal score in Candy Crush is an NP-Hard Problem [2] which means that solving it takes exponentially longer with increasing problem complexity. In this case this is due to the action space growing exponentially when increasing the field size and/or the number of different candy types in the game.

2.1. Gameplay

To form a better intuition of the problem we will shortly address how the game is played. Figure 1 shows a typical gameboard of 8x8 fields size and 6 different types of candies. In each move one of these candies is chosen, as well as an action where to move it {top, right, bottom, left}. The candy at the chosen position is then swapped with the candy at the destination of the chosen action. If this results in either one of the two candies being combined with other candies of the same type in certain formations (e.g. three or more in a row, an L- or T-shape) the player will receive points depending on the formation, the involved candies will be removed, and random candies will fall from the top to fill up the empty positions. If no combination is achieved, the swap gets reverted and the player receives a negative reward.

Figure 1. Example gameboard with field size 8x8 and 6 different candies.

2.2. Problem Setup

To be able to simulate the algorithms playing the game, a new Candy Crush environment implementing the OpenAI gym structure [3] was developed and parameterized in a way which would allow for the field size and number of candies to easily be changed.

The three algorithms {DQN, PPO, Decision Transformer} are then run on this gym for different combinations of field size and number of candies respectively, to show how well they perform with regard to the growing action space. The settings include all combinations of the field sizes {5x5, 6x6 ,7x7, 8x8} and numbers of candies {4, 5, 6}.

3. Algorithms

3.1. DQN

Deep Q-networks (DQN) were a breakthrough in reinforcement learning. With the increasing state and action spaces needed to play the Atari games in the ALE [1] it became too computationally expensive to store the Q-values inside a Q-table. There is also the problem of how to store Q-values for continuous problems like steering a car on a track. DQN solves these problems by using a neural network as a function approximator for the q-value, making Q-tables obsolete.

In this experiment, we use an extension of the DQN called Double DQN (DDQN) [4] which tackles the overestimation bias. This overestimation is caused by the DQN maximizing for the action with the highest Q-value and so neglecting the exploration of states with presumed lesser values which could be essential to bring the estimate back closer to the real Q-value. DDQN fights this by separating the steps of calculating the Q-value and choosing the next action onto two networks, where the target network estimates the Q-value and the model network which action to choose. This way both networks potentially overestimate for different state-action pairs which balances the final result.

Additionally, the approach of Dueling networks is integrated, making it a Dueling DDQN (DDDQN) [5]. This approach introduces the Advantage into the network which is the difference between the current action value and the mean of all action values in this state. Learning these differently allows for a more stable learning, since the same value function can be used for all actions in the same state, only the advantage for each action has to be learned separately and added to the result.

Our DDDQN consists of two 2D convolutional layers with 16 and 32 filters respectively, a 3x3 kernel, 2x2 stride, a tanh activation function and same padding to enable it to work on small field sizes. These are followed by a flattening layer which feeds in one dense layer for the state value and one for the advantages of the action values. These are then combined for the returned Q-value. Adam is used as the optimizer with a learning rate of 0.0001 together with the MSE loss function.

For the training process, two of such networks are instantiated, one as the model and one as the target network. In each episode, a batch of 16 samples is generated and stored in the replay buffer. The model network, responsible for choosing the best action, is then randomly trained on these samples and the epsilon (scaling exploration and exploitation) is then reduced from the starting value of 1, by multiplying it by 0.999. Every 50 episodes the target network is updated by obtaining the weights from the model network.

3.2. PPO

The Proximal Policy Optimization (PPO) algorithm is a Actor-Critic method which was thought up by the OpenAI team in 2017 [6]. It is an on-policy reinforcement learning algorithm which can, after applying some minor changes, be used in environments with either discrete or continuous action spaces. PPO works by first collecting a batch of experiences through interaction with the environment and then using this batch to update its policy. After this update, the old batch is disregarded and a new batch of experiences is collected using the updated policy. The PPO algorithm makes sure that, during the update process, the old policy does not stray too far from the current policy. This is achieved by relying on a specialized clipping in the objective function which removes the incentive for the new policy to get far from the old policy.

The version of PPO implemented in this project features Generalized Advantage Estimation (GAE) [7] instead of the usual discounted rewards and further uses the PPO2 objective functions. The main differences between PPO and PPO2 are that, for PPO2, the value function is being clipped as well and that the advantages are being normalized. Sadly, the OpenAI team did not further elaborate on these changes [8].

Both the actor and the critic network consist of 3 dense layers of decreasing size $[512, 256, 64]$ followed by a single output layer. For every combination of field sizes and candy numbers, the PPO algorithm was run for a total of 10 update steps with each batch update containing 50 trajectories. Each of these trajectories was limited to a maximum of 100 steps. A learning rate of 0.0025 was used in conjunction with the Adam optimizer. For GAE, the λ value was set to 0.9.

3.3. Decision Transformer

The desired return, the state, and the action leading to this state are passed through to the Decision Transformer (DT) as shown in Figure 2. Based on this input, the DT outputs the probabilities of how likely each action is to achieve the desired return. The action with the highest probability is then selected, executed in the environment, and the next state is received. Similar to an autoregressive next word prediction task, this executed action, this next state, and a desired return are appended to the input sequence [9].

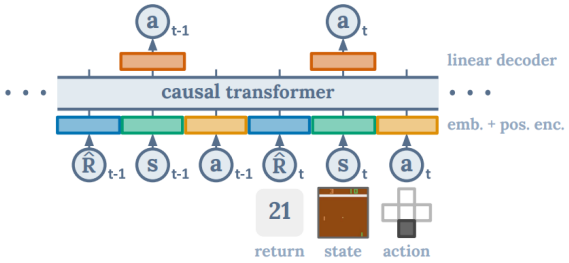

Figure 2. Decision Transformer architecture [9].

Overall, the DT processes a sequence of triples consisting of the desired return, the current state and the action leading to this state. If the number of triples is below the maximum number of triples of the input sequence, then a

padding must be applied. In other words, the input sequence is filled up with triples containing the desired return of 0, a state, and an action both indicating *none* [9]. Take, for example, an environment in which 10 actions are available. The 11th action indicates the *none* action used for padding. The same applies to the number of candies.

For each triple, a stack of convolutional layers is extracting an embedding of the state and fully connected layers are used for an embedding of the action. These two embeddings are concatenated together with the desired return of the triple. In the following, the resulting vector for each triple is passed through a multi-head-attention layer. The output sequence of this layer is flattened and passed through a fully connected layer with a softmax activation function for action determination (output) [9].

4. Evaluation

Each of the three algorithms is run on the gym with the described settings including all combinations of field sizes {5x5, 6x6, 7x7, 8x8} and numbers of candies {4, 5, 6}.

4.1. DQN

Each combination was run for 1000 episodes. In Figures 3 and 4 you can see the average rewards and scores over this process. It becomes evident that larger field sizes seem to benefit the performance of the agent. This can however be attributed to the larger combinations of multiple candies giving a higher reward. With very small fields some of these are impossible, others very hard to accomplish. Comparing the effects of the number of candies in the game, one can see how the performance declines with increasing complexity.

Even with the ongoing training and decline of epsilon, seen in Figure 5, shifting to an increasing exploitation of better states, the performance of the algorithm doesn't seem to improve significantly, especially looking at the more complex combinations.

4.2. PPO

As one can see in Figures 6 and 7, the average rewards per episode seem to quickly stagnate after only a couple of update steps. This applies to all candy and field size combinations that were tested over the course of the project. Furthermore, the overall score per episode never seems to significantly increase over the duration of the training. These results were to be expected though as PPO still struggles with these problem types. Apart from these aspects which are mainly concerned with the algorithm's performance, one can again see that, due to the way the Candy Crush environment functions, large field sizes generally seem to result in higher scores.

In addition to the Candy Crush gym environment, this implementation of PPO was also tested on a discrete action space version of LunarLander-v2 in which it achieved good results, further indicating that the increased action space size itself may be responsible for the poor performance.

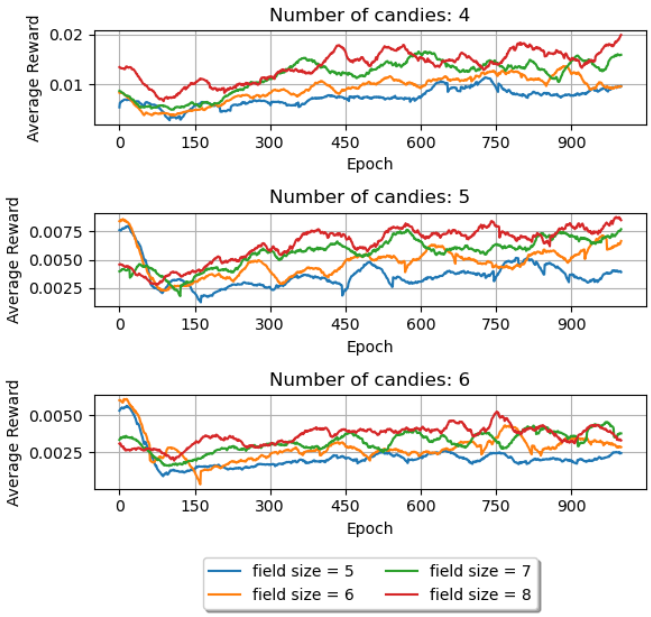

Figure 3. DQN: Average Rewards

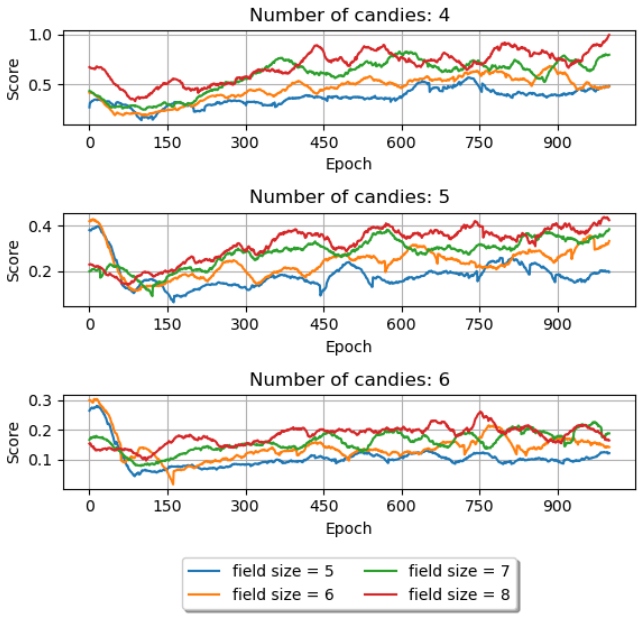

Figure 4. DQN: Scores

4.3. Decision Transformer

The DT was trained on different variants of the Candy Crush environment as mentioned in section 2.2. As shown in Figure 8, the number of candies and the field size affect the convergence speed of the model. Due to limited computing

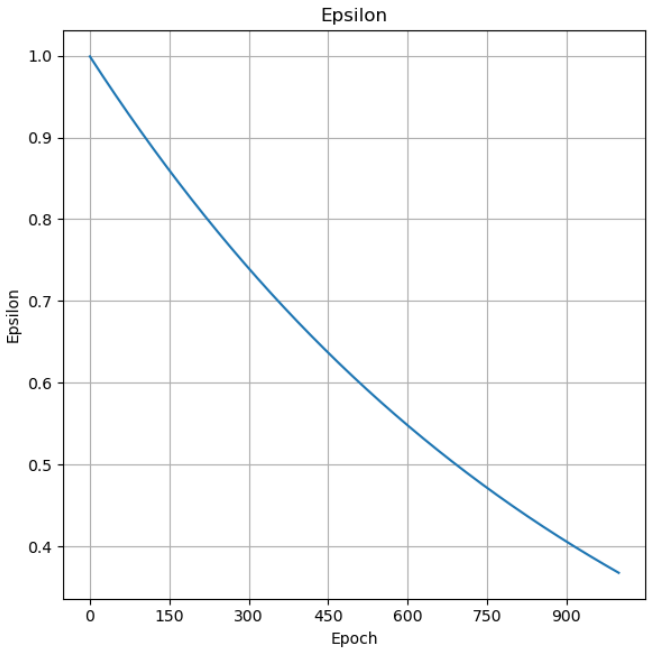

Figure 5. DQN: Epsilon

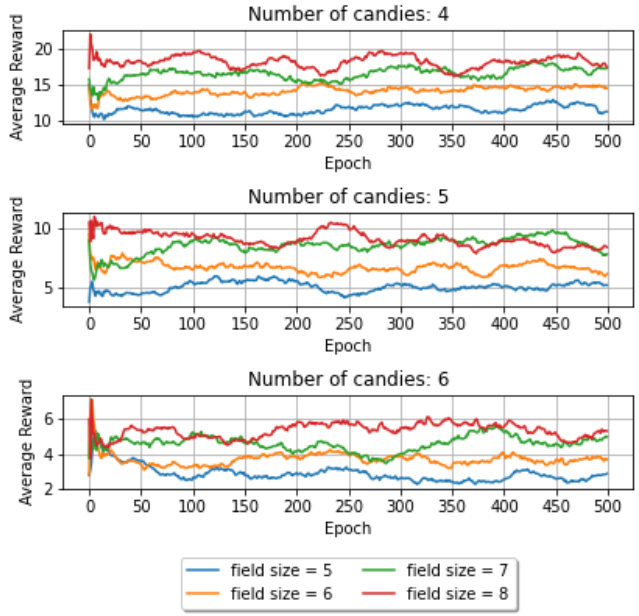

Figure 6. PPO: Average Rewards

resources, the DT was only trained for 20 epochs on each variant. The training dataset consists of 1,000,000 training samples. The model has about 60,000 parameters. A length of 10 time steps was chosen as the sequence length. A lower field size results in a faster training speed and vice versa. The same applies to the number of candies. The increasing

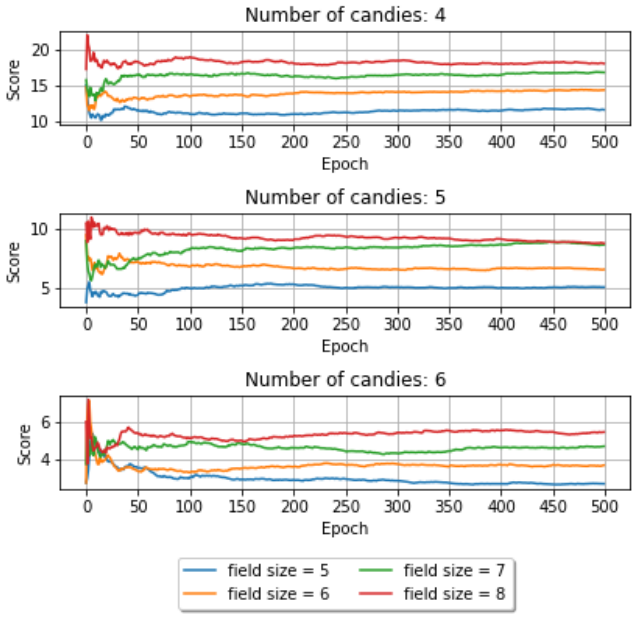

Figure 7. PPO: Scores

field size, the input size, and the number of actions increases. In other words, the problem complexity is increased. An increase in the number of available candies also increases the problem complexity because the model learns additional patterns based on these *new* candies.

Figure 9 represents the effects of different chosen desired rewards on the performance based on the different variants of the Candy Crush environment as mentioned in section 2.2. As mentioned above, the sequence length was set to 10 time steps. During each time step, the received reward was logged. At the end of a sequence, the environment was reset. In order to reduce the noise during measuring, this procedure was repeated 100 times and the average reward was determined. The difference between this average reward and the corresponding desired reward is shown in Figure 9.

With increasing field size, candy combinations leading to higher rewards (≥ 0.75) occur more often in the training data. The training data is generated by choosing a random action: If this action achieves a reward, then it will be added to the dataset. Otherwise, it will be discarded with a probability of 90%. The training dataset is not balanced. Therefore, choosing a random action that achieves a reward of 1.5 is unlikely - such trajectories rarely occur in the training data. Due to candy combinations leading to higher rewards occurring more often in the training data, the model is able to learn the specific candy combinations (patterns) better. An increase in the amount of available candy leads to more candy combinations which must be learned.

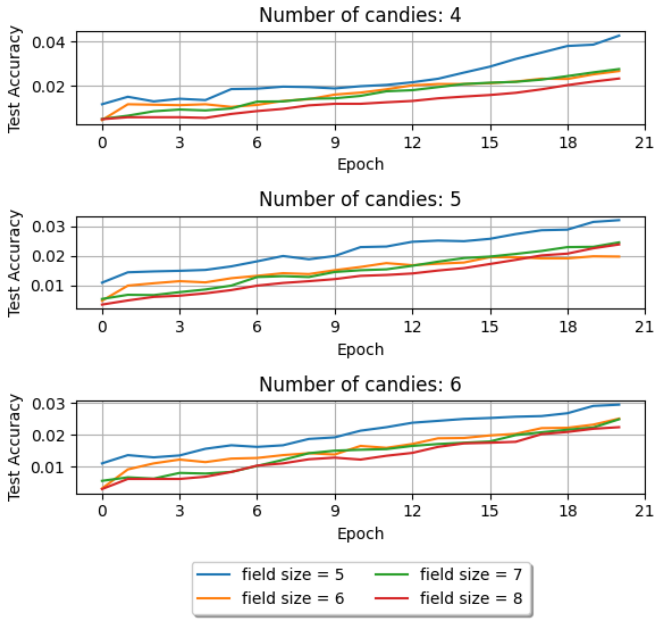

Figure 8. Accuracy on the test dataset during training of the DT on different variants of the Candy Crush environment.

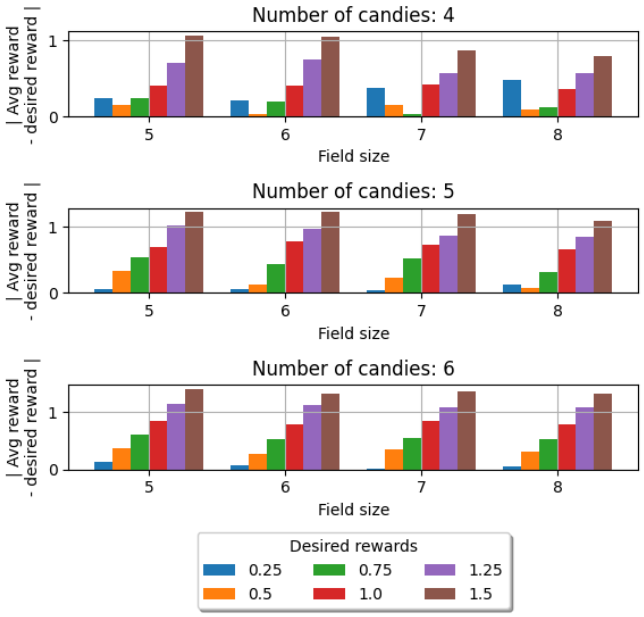

Figure 9. Difference between desired reward and average reward on different configurations of the Candy Crush environment.

5. Further Work

5.1. Candy Crush gym

Further development on the Candy Crush gym itself might lead to new and interesting changes in the results as the current version still includes some restrictions compared to the original application. Sampling actions and then checking whether or not they are valid also added a lot of computational effort and could be avoided in newer versions.

5.2. Algorithms

For improvements on the DQN one could e.g. draw inspiration from the other approaches in the Rainbow DQN Paper [10]. Regarding PPO, multithreaded training which would allow for the execution of multiple environments in parallel could aid in the overall process. Even though the development of this was started, the feature was eventually scrapped over the course of the project in order to shift the focus to other topics. Furthermore, some work has been done in order to include other algorithms such as discrete action space versions of both SAC and TD3 but had to be abandoned due to a lack of time. The inclusion of these approaches may yield some interesting information about the impact of large action spaces on their performance.

5.3. General

Due to the previously mentioned time constraints, a lot of further research can still be done regarding the project. It would for example be interesting to see how different parameter changes, i.e. batch size, amount of epochs etcetera potentially affect the overall performance of the proposed algorithms.

6. Conclusion

DQNs pursue a regression approach by trying to approximate the Q-value function. This becomes especially challenging when there are large action and state spaces as the resulting loss per update is potentially very small compared to less complex environments. This leads to poor overall performance and undesirable results. PPO's success on the other hand is based on encountering as many action combinations for all individual states as possible in order to maximize its learning efficiency regarding the policy. This in turn also becomes problematic in scenarios where there is both a big state and action space. Thus PPO, as mentioned previously, also performs subpar in the Candy Crush gym environment. Decision Transformers on the other hand yield significantly better results as they approach the problem in a different way. Instead of being dependent on state action pairs and approximation the value function through regression, DTs follow an approach more akin to classification tasks. This leads to a better and more stable learning process in scenarios with a large action space which outperforms the other proposed algorithms by a large margin.

References

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[2] Toby Walsh. Candy crush is np-hard. *CoRR*, abs/1403.1911, 2014.

[3] https://www.gymlibrary.dev/.

[4] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Mar. 2016.

[5] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1995–2003, New York, New York, USA, 20–22 Jun 2016. PMLR.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[7] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2015.

[8] https://stable-baselines.readthedocs.io/en/master/modules/ppo2.html.

[9] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021.

[10] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning, 2017.