Can Deep Reinforcement Learning Solve Erdos-Selfridge-Spencer Games?

Maxime Bourliatoux & Maxence Monfort

31 January 2019

This is a summary of the paper *Can Deep Reinforcement Learning Solve Erdos-Selfridge-Spencer Games?* by Maithra Raghu, Alex Irpan, Jacob Andreas, Robert Kleinberg, Quoc Le and Jon Kleinberg. Here is the Github where you can find the implementation of the environments and some methods to train the agents : `https://github.com/Gafpix/dqn-attack-defense`

Introduction

In general, it seems that deep reinforcement learning is more powerful than the reinforcement learning. It is the case for some environments like MuJoCo environments which are environments based on physics interactions or Atari environments which are the environments to play all old Atari games. Indeed, there were many successes due to deep reinforcement learning. However, we still clearly don't know the limitations of this method. The goal of this paper is to consider an environment which would be easy to customize, easy to represent in order to test deep reinforcement learning and easy to compare to the optimal policy of the game. In order to do this, they considered a series of games created by Erdos, Selfridge and Spencer. The reason why they considered these games are that they are easy to represent since they represent a low-dimensional environment, there exists an optimal policy which is easy to generalize and there exist some parameters to customize the difficulty of the game.

1 Erdos-Selfridge-Spencer Attacker Defender Game

In this paper, they focus on the game called "the tenure game". This game can be played with 2 players: the attacker and the defender. Each game is defined by a number of rows (K + 1) and the number of pieces on each row. The principle is simple: each turn, the attacker has to create 2 groups of pieces, then the defender must remove all the pieces from one group and all the pieces from the other group are moved up to the upper row. The goal of the defender is to remove all the pieces from the game and the goal of the attacker is to make at least one piece go to the top row. These are the rules of "the tenure game". However, another quantity was brought to the game: the potential. This is a function which can characterize whether the current situation favors the attacker or the defender.

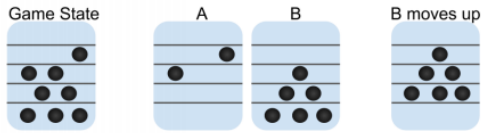

Figure 1: One turn of the tenure game

With these rules, it is clear that varying K which represents the number of rows minus 1 has an impact on the difficulty for the attacker and the defender. Indeed, if K is rather big, then the game will last longer and that also means that there are more possibilities to commit errors.

In order to explain all the concepts of the game, especially the potential function, we will first consider a game where all pieces are placed on the first row. In this case, the following theorem applies:

Theorem 1. Consider an instance of the Attacker-Defender game with K levels and N pieces, with all N pieces starting at level 0. Then if $N < 2^K$, the defender can always win.

A proof of this theorem is rather simple. Indeed, each turn, the attacker proposes a partition of the pieces and the defender has to remove one of them. That means that each turn, if the defender removes the largest group, the number of pieces N is reduced by more than 2: $N_1 < 2^{-1}N_0$. We can easily spot the recurrence so $N_K < 2^{-K}N_0$. Considering that $N_0 < 2^K$, $N_K < 1$ so $N_K = 0$. If the defender always removes the largest group, he will always win if the conditions are respected. However, this only applies because all the pieces are located on the same row, so they all have the same amount of rows between their row and the top row. That is why there exists a more complex proof of this theorem based on Erdos's probabilistic method.

The simple proof was from the defender's point of view. Let us now consider the attacker strategy. Let T be a random variable for the number of pieces that reach the top row. Each turn, each piece is either removed or moved up to the upper row with a probability of 2^{-1}. By recurrence, it appears that each piece has a probability of 2^{-K} to reach the top row. That means that $E[T_i] = 2^{-K}$ where T_i is a random variable for the piece i to reach the top row. It is obvious that $E[T] = \sum E[T_i] = \sum 2^{-K}$. As there are N pieces at the beginning of the game, $E[T] = N \cdot 2^{-K}$. We know that $N < 2^K$ so i appears that $E[T] < 1$. As T represents the number of pieces that reach the top level, it is an integer random variable, it means that $P(T = 0) \neq 0$. That means that, no matter which strategy is chosen by the attacker, there exists a set of choices that guarantees the defender to win. That allows to prove the theorem.

We shall now consider a more general form of the game: the pieces are no longer all located on the bottom row. The game can now be represented by a vector of dimension (K+1) $S = (n_0, n_1, ..., n_K)$ where n_i is the number of pieces on row i. Using the Theorem 1, the chance to reach the top level for a piece at level l is $2^{-(K-l)}$. To understand this result, we only have to apply the second proof of the theorem to a game where K would be equal to (K-l), allowing the piece to be placed at level 0. This observation allows us to define the potential function.

Definition 1. Potential function: Given a game state $S = (n_0, n_1, ..., n_K)$, we define the potential of the state as $\phi(S) = \sum n_i \cdot 2^{-(K-i)}$

We can notice that this is a linear function, expressible as $\phi(S) = w^T \cdot S$ where w is a vector with $w_l = 2^{-(K-l)}$. Thanks to this function, we can now consider a more general form of the first theorem.

Theorem 2. Consider an instance of the Attacker-Defender game that has K levels and N pieces, with pieces placed anywhere on the board, and let initial state be S_0. Then:

a. If $\phi(S_0) < 1$, the defender can always win.

b. If $\phi(S_0) \geq 1$, the attacker can always win.

In order to prove the first part of this theorem, we simply have to go back to the proof of the first theorem. Indeed, we noticed that $E[T] = \sum E[T_i] = \sum_i 2^{-(K-l_i)}$ where l_i is the level of piece i. What is more, this last sum is equal to $\phi(S_0)$ which is less or equal to 0. The result is the same as the first theorem: the defender can always win.

For the second part of the theorem, we have to use the result of the third theorem we will see later: if $\phi(S_0) >= 1$, there always exist 2 partitions A, B where $\phi(A) \geq 0.5$ and $\phi(B) \geq 0.5$. This result is very strong. Indeed, the defender must remove one of the 2 groups. However, all the pieces from the other group are moved up, which means that all levels are increased by one. Let us consider that B was the group that was removed, then $new\phi(A) = \sum 2^{-(K-(l_i+1))} = 2 \cdot \phi(A)$. As $\phi(A) \geq 0.5$, $new\phi(A) \geq 1$. By recurrence, the attacker can always win.

2 Deep Reinforcement Learning on the Attacker-Defender Game

In order to implement this game and train the defender, we had to code an environment. To do this, we used the Gym package. The environment only has 2 parameters: K which has an influence on the length of the game, which means the length between 2 rewards as the reward only appears at the end of a game, and the initial potential since the optimal policy for a defender is really efficient when the potential of the game is less or equal than 1.

A game is represented as a vector of dimension (K+1), it is named "game_state" in the code. We also chose to set a "state" variable in the code which is the concatenation of A and B. Based on the Gym implementation, there is an action space, which is in this case either 0 or 1 (0 erases A and 1 erases B), and an observation space, which is in this case a vector of dimension (2*K + 2).

Concerning the methods, there is a "reset" method that resets the game once it is over and randomly places pieces to get a new game with the potential set to create the environment. There is also a "check" method that returns the reward, which means -1 if the attacker wins and 1 if the defender wins.

The most important function is the "step" function that allows the algorithm to perform a step. In this function, the defender erases one of the two groups proposed by the attacker and the attacker chooses 2 new groups. To efficiently train the defender, the attacker will mostly follow the optimal strategy: create 2 groups of nearly equal potential. However, there is also a small probability for the attacker to choose 2 groups with a set difference to make the policy learned by the defender generalizable, it is called the Disjoint Support Strategy.

Once the environment is set, we can now define 2 policies. The first policy is a simple linear policy. The second policy is a deep network containing 2 hidden layers with 300 neurons each. 3 different algorithms were used: Proximal Policy Optimization (PPO), Advantage Actor Critic (A2C) and Deep Q-Networks (DQN). Here are the results for these 3 algorithms on both policies with different parameters. We also produced results based on our defender environment, these results can be seen at the end of this report.

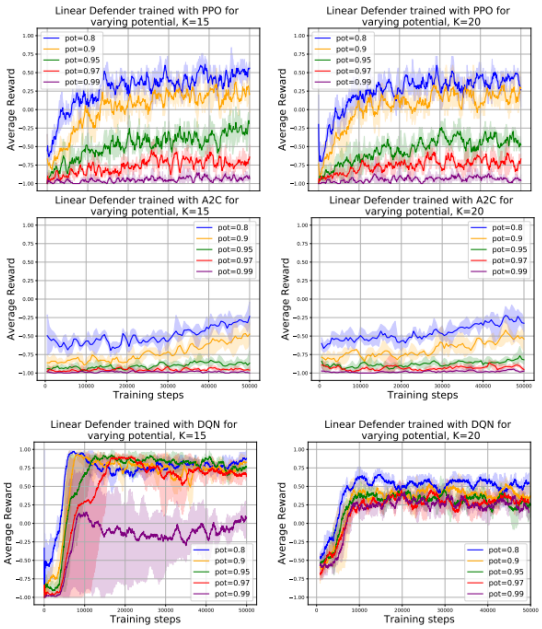

Figure 2: Training a linear network to play as the defender agent

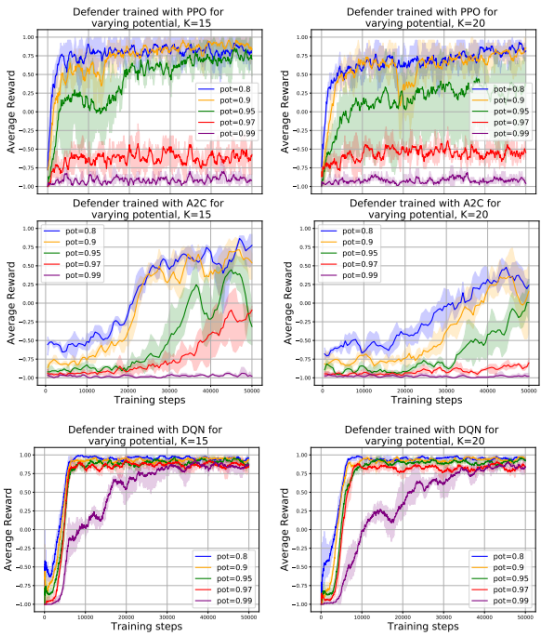

Figure 3: Training a deep network to play as the defender agent

According to these results, we can see that DQN performs relatively well. What is more, the results are better with a deep network policy. On the opposite, the results of A2C are rather bad. In theory, the optimal policy can be expressed with a linear model. However, the results clearly show that a deep network is better, especially on more difficult games where it seems that PPO and A2C stays at -1 for a potential of 0.99. The authors then chose to plot the maximum, the minimum and the mean performance using the deep network policy. Once again, it appears that DQN is better than the others since the variation is way smaller than in PPO and A2C.

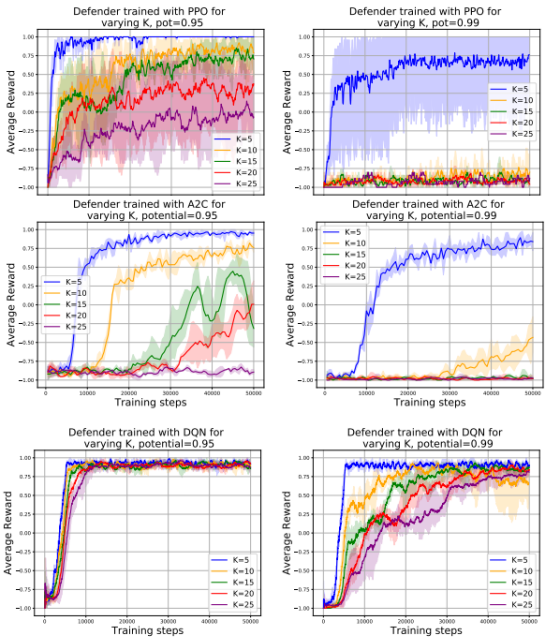

Figure 4: Training a defender on varying K

3 Generalization in Reinforcement Learning and Multiagent Learning

As mentioned before, the goal is also that the defender can win against every kind of strategy. To evaluate this generalization, the defender is trained against an attacker only playing optimally and is tested against an attacker playing optimally and another attacker playing using the Disjoint Support Strategy. The results show that the defender is good against an attacker playing using the optimal strategy. However, it is worse against an attacker playing worse than the optimal attacker. That confirms that the defender has to be trained on both strategies to perform well.

Figure 5: Performance of the defender against an unseen strategy

The way to avoid this kind of problem is to train an attacker in order to train the defender against this attacker. However, there quickly appears a difficulty to implement the environment. Indeed, this game was chosen because it was easy to implement. However, how can we simply represent the action space for the attacker. To solve this problem, we will use the following theorem.

Theorem 3. For any Attacker-Defender game with K levels, start state S_0 and $\phi(S_0) \geq 1$, there exists a partition A,B such that $\phi(A) \geq 0.5$, $\phi(B) \geq 0.5$, and for some l, A contains pieces of level $i > l$, and B contains all pieces of level $i < l$.

To prove this theorem, we need to consider A_l which is the set of all pieces from the top level to the (l-1) level. The series is decreasing since the number of pieces is also decreasing. We know that $\phi(A_K) = 0$ and that $\phi(A_0) = \phi(S_0) \geq 1$. According to these last remarks, there exists an l such that $\phi(A_l) < 0.5$ and $\phi(A_{l-1}) \geq 0.5$.

First case: $\phi(A_{l-1}) = 0.5$. Then $\phi(B) = \phi(S_0) - \phi(A_{l-1}) \geq 0.5$.
Second case: $\phi(A_{l-1}) > 0.5$. Then, $\phi(A_{l-1}) = \sum 2^{-(K-(l+i))} = \sum 2^i \cdot 2^{-(K-l)}$, so $\phi(A_{l-1})$ is an integer multiple of $2^{-(K-l)}$.

The same applies to $\phi(A_l)$. Let $\phi(A_l) = n \cdot 2^{-(K-l)}$ and $\phi(A_{l-1}) = m \cdot 2^{-(K-l)}$. We can deduce that level l has m - n pieces, so we can move k ¡ m-n for A_{l-1} to A_l such that the potential of the new set is larger or equal to 0.5.

Thanks to this theorem, we can now define the action space as a discrete space of length (K+1). The attacker outputs a level l and the environment assigns all pieces before level l to A, all pieces after level l to B and splits level l among A and B to fit the theorem 3.

Using this environment, we can now plot the different results we already plot for the defender training using PPO, A2C and DQN. However, the results for DQN were so poor that they were not given. As for the defender environment, the results based on our attacker environment can be seen at the end of the report.

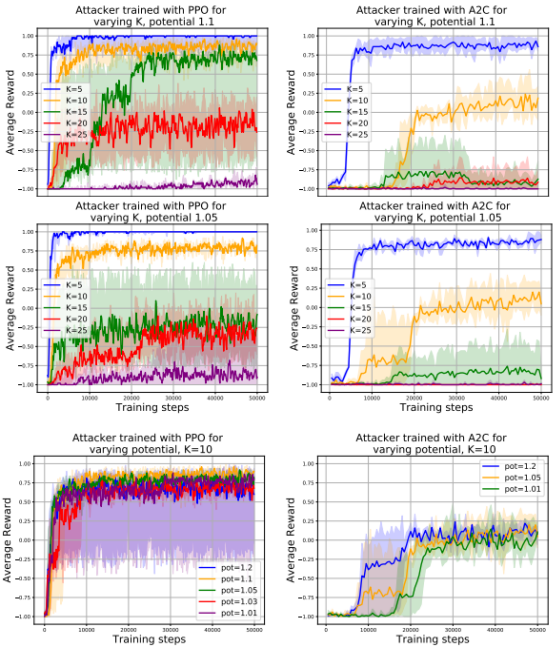

Figure 6: Training a linear network to play as the attacker agent

It appears that the variance clearly increases when we raise the value of K.

The authors then tried learning through multiagent play. We can see that the results are worse than when the attacker learns trough a single agent. The most important thing is that the attacker even loses against the defender when the potential is 1.1 for K=15, which is not the case through single agent learning.

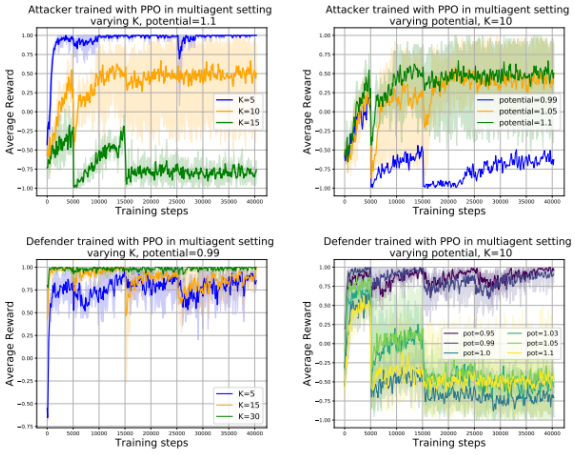

Figure 7: Performance in multiagent setting

The same test is applied to the defender. Indeed, a defender was trained against the optimal attacker and another defender was trained in the multiagent setting. They were both tested on an

attacker that only uses the Disjoint Support Strategy. The following figure shows that the multiagent defender generalizes better against a defender playing an unseen strategy.

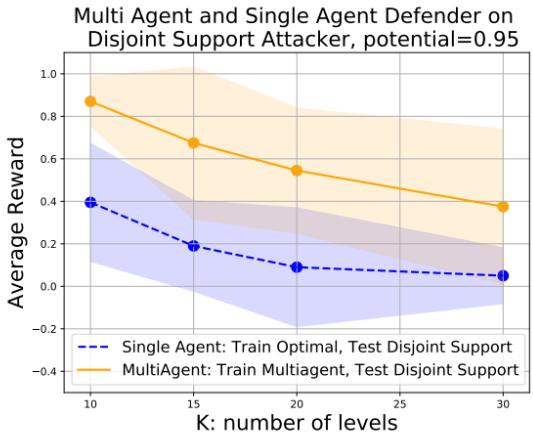

Figure 8: Performance of the defender against an unseen strategy

4 Training with Self Play

The next step is to use the same network for the defender and the attacker. In order to do this, a common policy had to be found. In fact, the attacker and the defender can both take into arguments 2 partitions A and B and compare them. For the defender, it will simply indicate which partition to delete. For the attacker, it will allow him to find a more balanced partition using binary search. With this method, both an attacker and a defender can be trained simultaneously using the same neural network.

Algorithm 1 Self Play with Binary Search

```
initialize game
repeat
    Partition game pieces at center into A, B
    repeat
        Input partition A, B into neural network
        Output of neural network determines next binary
        search split
        Create new partition A, B from this split
    until Binary Search Converges
    Input final partition A, B to network
    Destroy larger set according to network output
until Game Over: use reward to update network parame-
ters with RL algorithm
```

Figure 9: Algorithm that enables training by self-play

DQN was used because, according to their results, it was the best algorithm. The results are extremely good since none of the average rewards of the previous methods were as high as in these results. We can also remark that changing K or the potential doesn't have a huge impact on the results.

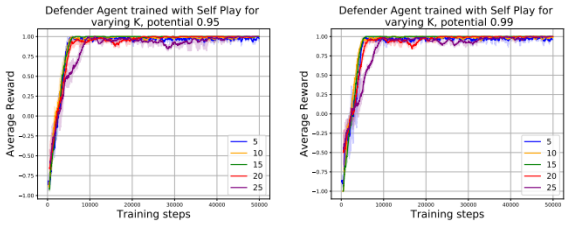

Figure 10: Performance of a defender trained by self-play

5 Supervised Learning vs Reinforcement Learning

The ending point of this paper is to compare supervised learning to reinforcement learning. Indeed, as it is mentioned in the introduction, one of the reasons why this environment was considered was because of the fact that, each turn, we perfectly know what is the best move.

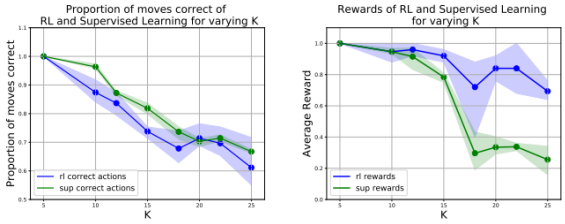

Figure 11: Comparison of correct moves and average reward between RL and SL

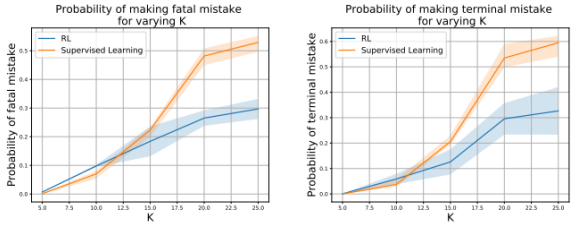

Figure 12: Comparison of mistakes between RL and SL

The first results show that the defender trained using supervised learning plays more correct moves than the defender trained using reinforcement learning. However, it appears that this last defender has a higher average reward than the first defender. How can the reward be higher since the first defender plays more correct moves? In order to understand why, they decided to consider 2 types of mistakes. The first mistake is called a terminal mistake. A defender is making a terminal mistake when $\phi(A) \geq 0.5$, $\phi(B) < 0.5$ and the defender chooses to erase B. That means that the defender could have been in a situation where the potential of the game would have been less than 1: in a situation where the defender can always win. However, it is worse than that since the defender is now in a situation where the attacker can always win. The second mistake is called a fatal mistake. A defender is making a fatal mistake if the potential of the situation is less than 1, $\phi(A) \geq 0.5$ and the defender

chooses to erase B. That means that the defender was in a situation where this defender could always win and it is now in a situation where the attacker can always win.

The following figure shows that, despite the fact that the defender trained using supervised learning plays more correct moves than the other defender, it also makes more fatal and terminal mistakes when K raises.

Conclusion

This paper shows that, even if the environment was simple, the tuning of the parameters has a major impact on the difficulty of the game. In fact, the self play methods performed well but that was not the case at all for the multiagent setting concerning the attacker. The comparison between reinforcement learning and supervised learning showed that, despite the correct moves performed by the defender trained using supervised learning, the defender trained using reinforcement learning performed better since it does not make as many mistakes as the first defender.

Appendix

5.1 Generalization Phenomena

In the paper, they trained the defenders on varying K and varying potentials. However, this environment has been chosen because there is an optimal policy which is generalizable for any K and any potential. What would be interesting is to see if it is also the case using reinforcement learning. It appears that, as the input is a $2 \cdot (K+1)$ dimensional vector, it is not fair to train the defender with K_1 and test it on $K_2 > K_1$. However, it should be possible if $K_2 < K_1$. Indeed, it only has to complete the vector with zeros. Here are the results observed using this process.

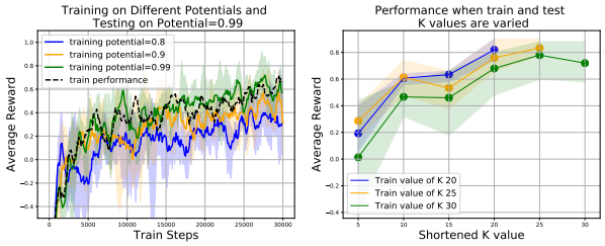

Figure 13: Performance on testing on inferior potential and K

We can see that when $K_2 << K_1$, the performance is rather bad. For example, training with $K_1 = 30$ and testing with $K_2 = 5$ only results in having a reward of nearly 0 : that means that only a half of the games played are won. Concerning the potential, we can see that the average reward is higher when the training potential is higher.

5.2 Understanding Model Failures

In this game, there is a simple strategy for the defender in order to win when one of the 2 groups is the zero set : the defender has to destroy the other set and the game is won. However, according to the following figure, it seems that the agent does not always apply this simple strategy.

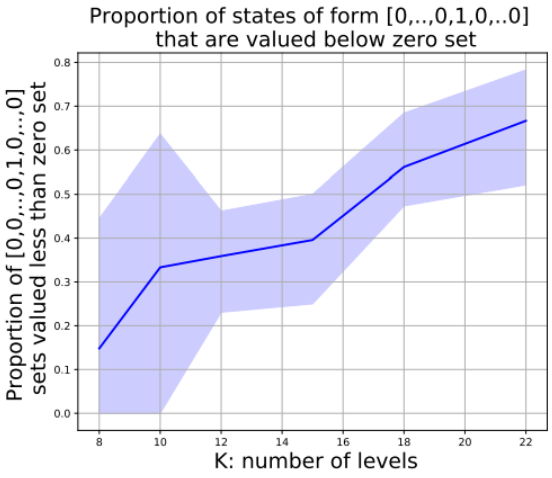

Figure 14: Proportion of states that are valued below zero set

We can clearly see that the proportion of states of form [0,...,0, 1, 0 ...0] grows with K.
We can also check if the predictions are correct since we always know if the action was optimal or not for the defender. In the following figure, the potential difference is $\phi(A) - \phi(B)$. That means that if the difference is positive, the defender should pick A and otherwise, B.

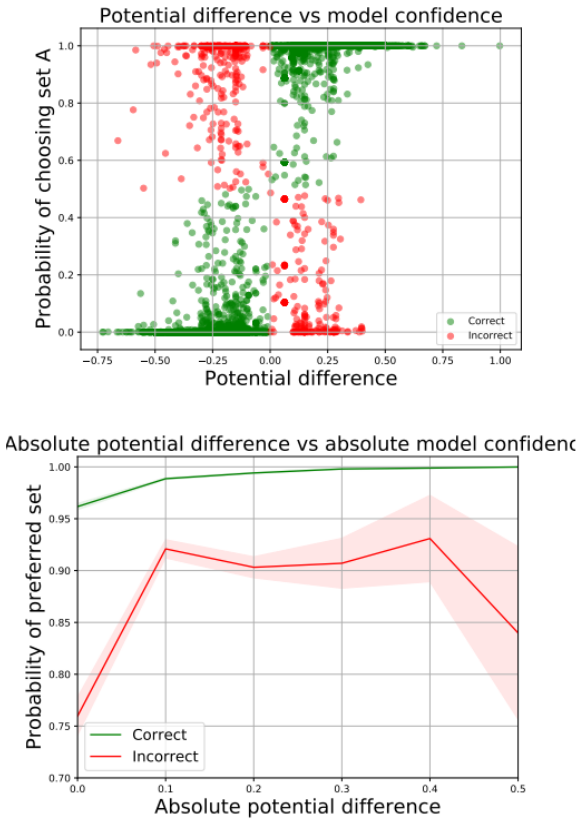

Figure 15: Potential difference vs Model Confidenc

Last but not least, we saw that it was better in order to generalize to train on a difficult potential. That is why they considered the number of states for a given K where $\phi(S_0) = 1$. To get an approximation, they used the following theorem :

Theorem 4. The number of states with potential 1 for a game with K levels grows like $2^{\Theta(K^2)}$ (where $0.25 \cdot K^2 \leq \Theta(K^2) \leq 0.5 \cdot K^2$)

In order to prove this theorem, we will first show that $\Theta(K^2) \leq 0.5 \cdot K^2$. For each s_i in the state of the game, the potential on the row is $2^{-(K-i)}$. To reach 1, it appears that s_i can take a value up to $2^{(K-i)}$. By multiplying all of these together, we have an upper bound that equals $2^{0.5 \cdot K^2}$.

In order to prove that $\Theta(K^2) \geq 0.25 \cdot K^2$, we will assume for convenience that K is a power of 2. Then, we look at the set of positive integer solutions of the following system :

$$a_{j-1} \cdot 2^{1-j} + a_j \cdot 2^{-j} = \frac{1}{K}$$

where j ranges over all even numbers between $log(K) + 1$ and K. As the equations don't share any variables, the number of solutions is $\Pi_j \frac{2^{j-1}}{K}$ which is roughly $2^{0.25 \cdot K^2}$.

Our results

Based on the environments we coded and that you can find on the Github that was provided at the beginning of the report, we tried to reproduce the results that were provided in the original paper. Here are the results concerning the training of the defender using DQN and PPO :

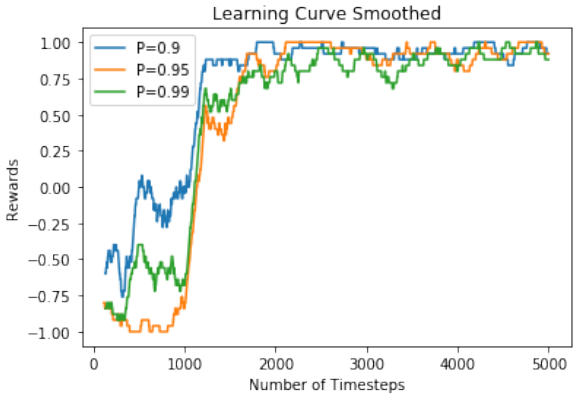

Figure 16: Our defender DQN training with K=5 and different P

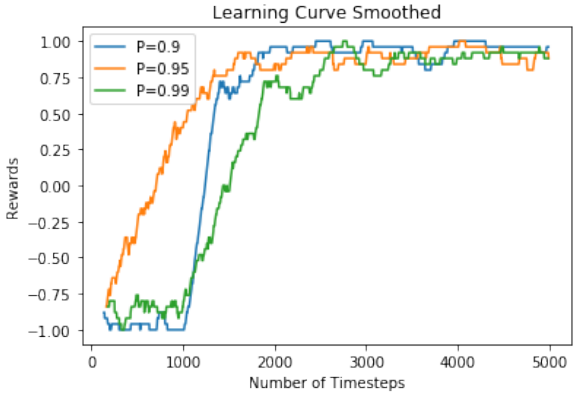

Figure 17: Our defender DQN training with K=10 and different P

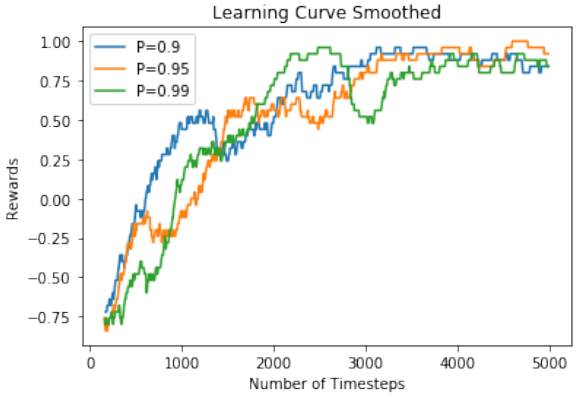

Figure 18: Our defender PPO training with K=10 and different P

We can see in these graphs that, for example, for $K = 5$, the agent learns faster for an initial potential of 0.99 than for an initial potential of 0.95. However, according to the original paper, that should be the opposite. That is because, in the original paper, it corresponds to the mean reward whereas here, we only show the results for one occurence.

We also had some results concerning the training of the attacker using DQN and PPO:

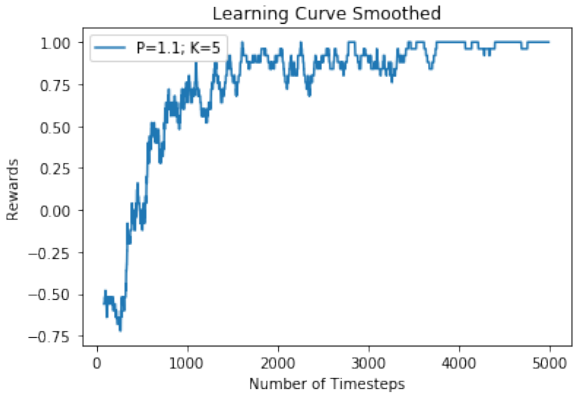

Figure 19: Our attacker PPO training with K=5 and P=1.1

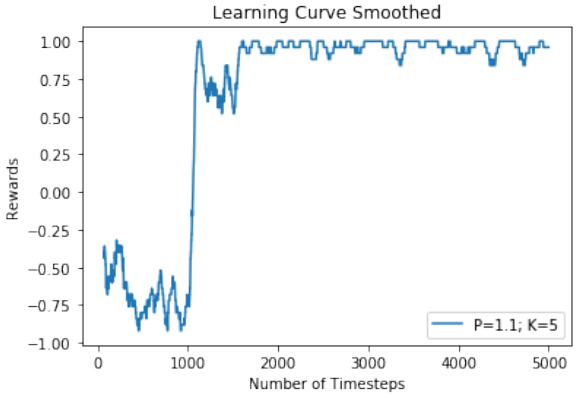

Figure 20: Our attacker DQN training with K=5 and P=1.1

We can see that for all results, the rewards converges to 1, which shows that the agent wins nearly every game. However, due to a lack of memory, we couldn't get results for a greater K than 10.