CAN DEEP REINFORCEMENT LEARNING SOLVE ERDOS-SELFRIDGE-SPENCER GAMES?

Maithra Raghu
Google Brain and Cornell University
{maithrar}@gmail.com

Alex Irpan
Google Brain

Jacob Andreas
University of California, Berkeley

Robert Kleinberg
Cornell University

Quoc V. Le
Google Brain

Jon Kleinberg
Cornell University

ABSTRACT

Deep reinforcement learning has achieved many recent successes, but our understanding of its strengths and limitations is hampered by the lack of rich environments in which we can fully characterize optimal behavior, and correspondingly diagnose individual actions against such a characterization. Here we consider a family of combinatorial games, arising from work of Erdos, Selfridge, and Spencer, and we propose their use as environments for evaluating and comparing different approaches to reinforcement learning. These games have a number of appealing features: they are challenging for current learning approaches, but they form (i) a low-dimensional, simply parametrized environment where (ii) there is a linear closed form solution for optimal behavior from any state, and (iii) the difficulty of the game can be tuned by changing environment parameters in an interpretable way. We use these Erdos-Selfridge-Spencer games not only to compare different algorithms, but test for generalization, make comparisons to supervised learning, analyse multiagent play, and even develop a self play algorithm.

1 INTRODUCTION

Deep reinforcement learning has seen many remarkable successes over the past few years Mnih et al. (2015); Silver et al. (2017). But developing learning algorithms that are robust across tasks and policy representations remains a challenge. Standard benchmarks like MuJoCo and Atari provide rich settings for experimentation, but the specifics of the underlying environments differ from each other in multiple ways, and hence determining the principles underlying any particular form of sub-optimal behavior is difficult. Optimal behavior in these environments is generally complex and not fully characterized, so algorithmic success is generally associated with high scores, typically on a copy of the training environment making it hard to analyze where errors are occurring or evaluate generalization.

An ideal setting for studying the strengths and limitations of reinforcement learning algorithms would be (i) a simply parametrized family of environments where (ii) optimal behavior can be completely characterized and (iii) the environment is rich enough to support interaction and multiagent play.

To produce such a family of environments, we look in a novel direction – to a set of two-player combinatorial games with their roots in work of Erdos and Selfridge (Erdos & Selfridge, 1973), and placed on a general footing by Spencer (1994). Roughly speaking, these *Erdos-Selfridge-Spencer (ESS) games* are games in which two players take turns selecting objects from some combinatorial structure, with the feature that optimal strategies can be defined by potential functions derived from conditional expectations over random future play.

These ESS games thus provide an opportunity to capture the general desiderata noted above, with a clean characterization of optimal behavior and a set of instances that range from easy to very hard as we sweep over a simple set of tunable parameters. We focus in particular on one of the best-known games in this genre, *Spencer's attacker-defender game* (also known as the "tenure game"; Spencer, 1994), in which — roughly speaking — an *attacker* advances a set of pieces up the levels of a board, while a *defender* destroys subsets of these pieces to try prevent any of them from reaching the final level (Figure 1). An instance of the game can be parametrized by two key quantities. The first is the number of levels K, which determines both the size of the state space and the approximate length of the game; the latter is directly related to the sparsity of win/loss signals as rewards. The second quantity is a *potential function* ϕ, whose magnitude characterizes whether the instance favors the defender or attacker, and how much "margin of error" there is in optimal play.

The environment therefore allows us to study learning by the defender and attacker, separately or concurrently in multiagent and self-play. In the process, we are able to develop insights about the robustness of solutions to changes in the environment. These types of analyses have been long-standing goals, but they have generally been approached much more abstractly, given the difficulty in characterizing step-by-step optimality in non-trivial environments such as this one. Because we have a move-by-move characterization of optimal play, we can go beyond simple measures of reward based purely on win/loss outcomes and use supervised learning techniques to pinpoint the exact location of the errors in a trajectory of play.

The main contributions of this work are thus the following:

1. We develop these combinatorial games as environments for studying the behavior of reinforcement learning algorithms in a setting where it is possible to characterize optimal play and to tune the underlying difficulty using natural parameters.
2. We show how reinforcement learning algorithms in this domain are able to learn generalizable policies in addition to simply achieving high performance, and through new combinatorial results about the domain, we are able to develop strong methods for *multiagent* play that enhance generalization.
3. Through an extension of our combinatorial results, we show how this domain lends itself to a subtle self-play algorithm, which achieves a significant improvement in performance.
4. We can characterize optimal play at a move-by-move level and thus compare the performance of a deep RL agent to one trained using supervised learning on move-by-move decisions. By doing so, we discover an intriguing phenomenon: while the supervised learning agent is more accurate on individual move decisions than the RL agent, the RL agent is better at playing the game! We further interpret this result by defining a notion of *fatal mistakes*, and showing that while the deep RL agent makes more mistakes overall, it makes fewer fatal mistakes.

In summary, we present learning and generalization experiments for a variety of commonly used model architectures and learning algorithms. We show that despite the superficially simple structure of the game, it provides both significant challenges for standard reinforcement learning approaches and a number of tools for precisely understanding those challenges.

2 ERDOS-SELFRIDGE-SPENCER ATTACKER DEFENDER GAME

We first introduce the family of Attacker-Defender Games (Spencer, 1994), a set of games with two properties that yield a particularly attractive testbed for deep reinforcement learning: the ability to continuously vary the difficulty of the environment through two parameters, and the existence of a closed form solution that is expressible as a *linear model*.

An Attacker-Defender game involves two players: an attacker who moves pieces, and a defender who destroys pieces. An instance of the game has a set of *levels* numbered from 0 to K, and N pieces that are initialized across these levels. The attacker's goal is to get at least one of their pieces to level K, and the defender's goal is to destroy all N pieces before this can happen. In each turn, the attacker proposes a partition A, B of the pieces still in play. The defender then chooses one of the sets to destroy and remove from play. All pieces in the other set are moved up a level. The game ends when either one or more pieces reach level K, or when all pieces are destroyed. Figure 1 shows one turn of play.

With this setup, varying the number of levels K or the number of pieces N changes the difficulty for the attacker and the defender. One of the most striking aspects of the Attacker-Defender game is that it is possible to make this trade-off precise, and en route to doing so, also identify a *linear optimal policy.*

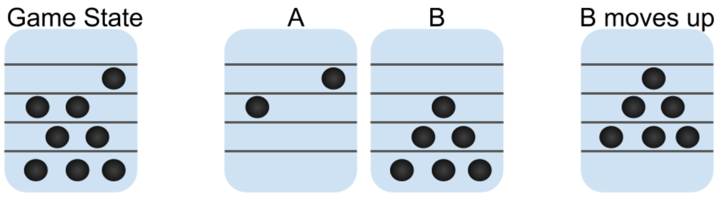


Figure 1: One turn in an ESS Attacker-Defender game. The attacker proposes a partition A, B of the current game state, and the defender chooses one set to destroy (in this case A). Pieces in the remaining set (B) then move up a level to form the next game state.

We start with a simple special case — rather than initializing the board with pieces placed arbitrarily, we require the pieces to all start at level 0. In this special case, we can directly think of the game's difficulty in terms of the number of levels K and the number of pieces N.

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.*

There is a simple proof of this fact: the defender simply always destroys the larger one of the sets A or B. In this way, the number of pieces is reduced by at least a factor of two in each step; since a piece must travel K steps in order to reach level K, and $N < 2^K$, no piece will reach level K.

When we move to the more general case in which the board is initialized at the start of the game with pieces placed at arbitrary levels, it will be less immediately clear how to define the "larger" one of the sets A or B. We therefore describe a second proof of Theorem 1 that will be useful in these more general settings. This second proof, due to Spencer (1994), uses Erdos's probabilistic method and proceeds as follows.

For any attacker strategy, assume the defender plays randomly. Let T be a random variable for the number of pieces that reach level K. Then $T = \sum T_i$ where T_i is the indicator that piece i reaches level K. But then $E[T] = \sum E[T_i] = \sum_i 2^{-K}$: as the defender is playing randomly, any piece has probability $1/2$ of advancing a level and $1/2$ of being destroyed. As all the pieces start at level 0, they must advance K levels to reach the top, which happens with probability 2^{-K}. But now, by choice of N, we have that $\sum_i 2^{-K} = N2^{-K} < 1$. Since T is an integer random variable, $E[T] < 1$ implies that the distribution of T has nonzero mass at 0 - in other words there is some set of choices for the defender that guarantees destroying all pieces. This means that the attacker does not have a strategy that wins with probability 1 against random play by the defender; since the game has the property that one player or the other must be able to force a win, it follows that the defender can force a win. This completes the proof.

Now consider the general form of the game, in which the initial configuration can have pieces at arbitrary levels. Thus, at any point in time, the state of the game can be described by a K-dimensional vector $S = (n_0, n_1, ..., n_K)$, with n_i the number of pieces at level i.

Extending the argument used in the second proof above, we note that a piece at level l has a $2^{-(K-l)}$ chance of survival under random play. This motivates the following *potential function* on states:

Definition 1. Potential Function*: Given a game state $S = (n_0, n_1, ..., n_K)$, we define the* potential *of the state as $\phi(S) = \sum_{i=0}^{K} n_i 2^{-(K-i)}$.*

Note that this is a *linear* function on the input state, expressible as $\phi(S) = w^T \cdot S$ for w a vector with $w_l = 2^{-(K-l)}$. We can now state the following generalization of Theorem 1, again due to Spencer (1994).

Theorem 2 (Spencer (1994))**.** *Consider an instance of the Attacker-Defender game that has K levels and N pieces, with pieces placed anywhere on the board, and let the 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.

One way to prove part (a) of this theorem is by directly extending the proof of Theorem 1, with $E[T] = \sum E[T_i] = \sum_i 2^{-(K-l_i)}$ where l_i is the level of piece i. After noting that $\sum_i 2^{-(K-l_i)} = \phi(S_0) < 1$ by our definition of the potential function and choice of S_0, we finish off as in Theorem 1.

This definition of the potential function gives a natural, concrete strategy for the defender: the defender simply destroys whichever of A or B has higher potential. We claim that if $\phi(S_0) < 1$, then this strategy guarantees that any subsequent state S will also have $\phi(S) < 1$. Indeed, suppose (renaming the sets if necessary) that A has a potential at least as high as B's, and that A is the set destroyed by the defender. Since $\phi(B) \leq \phi(A)$ and $\phi(A) + \phi(B) = \phi(S) < 1$, the next state has potential $2\phi(B)$ (double the potential of B as all pieces move up a level) which is also less than 1. In order to win, the attacker would need to place a piece on level K, which would produce a set of potential at least 1. Since all sets under the defender's strategy have potential strictly less than 1, it follows that no piece ever reaches level K.

For $\phi(S_0) \geq 1$, Spencer (1994) proves part (b) of the theorem by defining an optimal strategy for the attacker, using a greedy algorithm to pick two sets A, B each with potential ≥ 0.5. For our purposes, the proof from Spencer (1994) results in an intractably large action space for the attacker; we therefore (in Theorem 3 later in the paper) define a new kind of attacker — the *prefix-attacker* — and we prove its optimality. These new combinatorial insights about the game enable us to later perform multiagent play, and subsequently self-play.

3 Related Work

The Atari benchmark (Mnih et al., 2015) is a well known set of tasks, ranging from easy to solve (Breakout, Pong) to very difficult (Montezuma's Revenge). Duan et al. (2016) proposed a set of continuous environments, implemented in the MuJoCo simulator Todorov et al. (2012). An advantage of physics based environments is that they can be varied continuously by changing physics parameters (Rajeswaran et al., 2016), or by randomizing rendering (Tobin et al., 2017). DeepMind Lab (Beattie et al., 2016) is a set of 3D navigation based environments. OpenAI Gym (Brockman et al., 2016) contains both the Atari and MuJoCo benchmarks, as well as classic control environments like Cartpole (Stephenson, 1909) and algorithmic tasks like copying an input sequence. The difficulty of algorithmic tasks can be easily increased by increasing the length of the input. Our proposed benchmark merges properties of both the algorithmic tasks and physics-based tasks, letting us increase difficulty by discrete changes in length or continuous changes in potential.

4 Deep Reinforcement Learning on the Attacker-Defender Game

From Section 2, we see that the Attacker-Defender games are a family of environments with a difficulty knob that can be continuously adjusted through the start state potential $\phi(S_0)$ and the number of levels K. In this section, we describe a set of baseline results on Attacker-Defender games that motivate the exploration in the remainder of this paper. We set up the Attacker-Defender environment as follows: the game state is represented by a $K + 1$ dimensional vector for levels 0 to K, with coordinate l representing the number of pieces at level l. For the defender agent, the input is the concatenation of the partition A, B, giving a $2(K + 1)$ dimensional vector. The start state S_0 is initialized randomly from a distribution over start states of a certain potential.

4.1 Training a Defender Agent on Varying Environment Difficulties

We first look at training a defender agent against an attacker that randomly chooses between (mostly) playing optimally, and (occasionally) playing suboptimally for exploration (with the *Disjoint Support Strategy*, described in the Appendix.) Recall from the specification of the potential function, in Definition 1 and Theorem 2, that the defender has a *linear optimal policy*: given an input partition A, B, the defender simply computes $\phi(A) - \phi(B)$, with $\phi(A) = \sum_{i=0}^{K} a_i w_i$, where a_i is the number of pieces in A at level i; $\phi(B) = \sum_{i=0}^{K} b_i w_i$, where b_i is the number of pieces in B at level i; and $w_i = 2^{-(K-i)}$ is the weighting defining the potential function.

When training the defender agent with RL, we have two choices of difficulty parameters. The potential of the start state, $\phi(S_0)$, changes how close to optimality the defender has to play, with

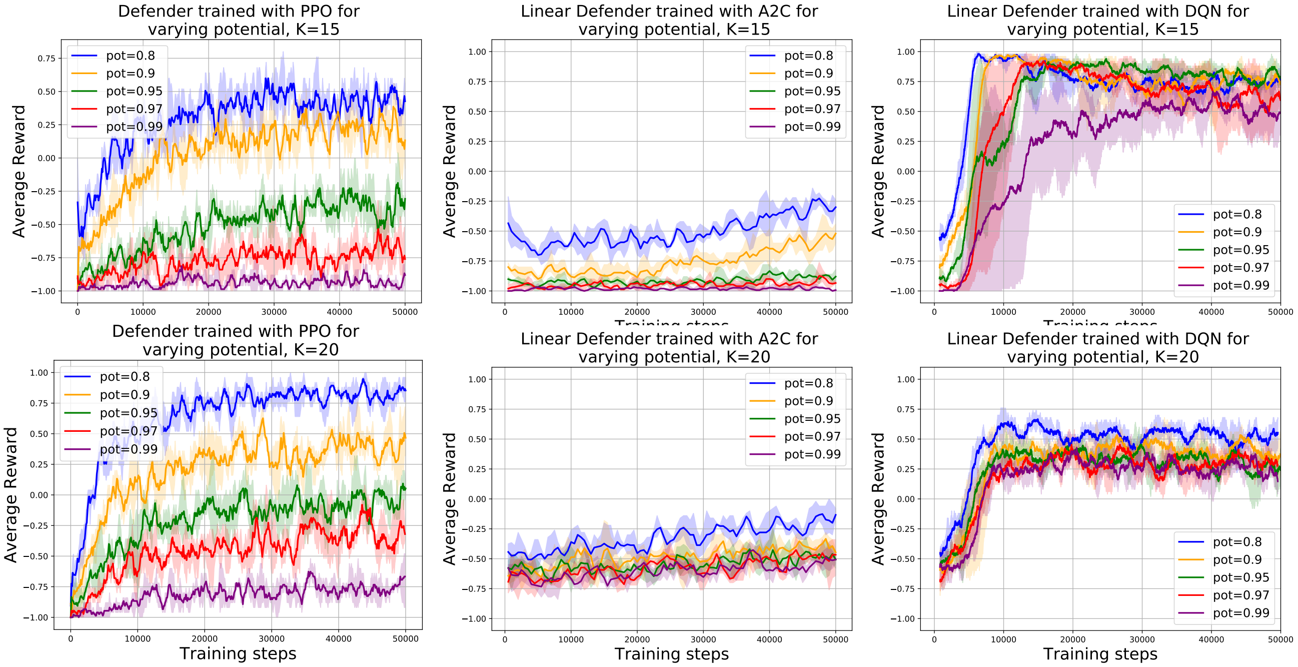


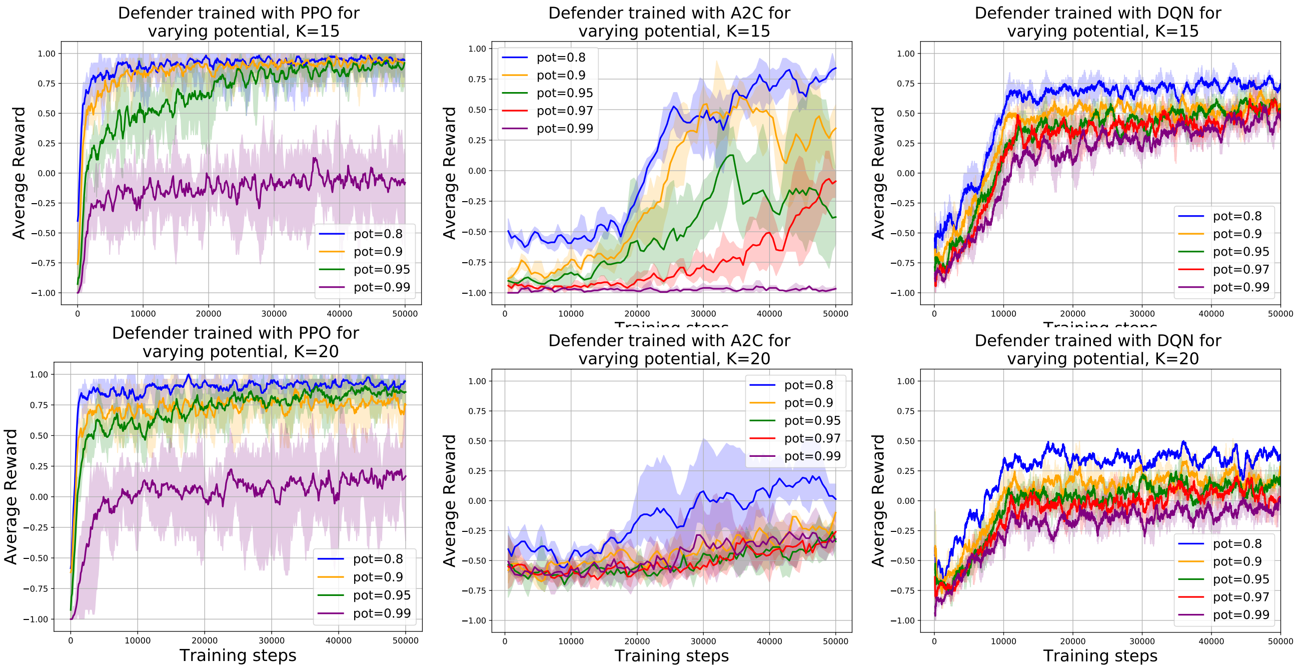
Figure 2: Training a linear network to play as the defender agent with PPO, A2C and DQN. A linear model is theoretically expressive enough to learn the optimal policy for the defender agent. In practice, we see that for many difficulty settings and algorithms, RL struggles to learn the optimal policy and performs more poorly than when using deeper models (compare to Figure 3). An exception to this is DQN which performs relatively well on all difficulty settings.

values close to 1 giving much less leeway for mistakes in valuing the two sets. Changing K, the number of levels, directly affects the sparsity of the reward, with higher K resulting in longer games and less feedback. Additionally, K also greatly increases the number of possible states and game trajectories (see Theorem 4 in the Appendix).

4.1.1 Evaluating Deep RL

As the optimal policy can be expressed as a linear network, we first try training a linear model for the defender agent. We evaluate Proximal Policy Optimization (PPO) (Schulman et al., 2017), Advantage Actor Critic (A2C) (Mnih et al., 2016), and Deep Q-Networks (DQN) (Mnih et al., 2015), using the OpenAI Baselines implementations (Hesse et al., 2017). Both PPO and A2C find it challenging to learn the harder difficulty settings of the game, and perform better with deeper networks (Figure 2). DQN performs surprisingly well, but we see some improvement in performance variance with a deeper model. In summary, while the policy can theoretically be expressed with a linear model, empirically we see gains in performance and a reduction in variance when using deeper networks (c.f. Figures 3, 4.)

Having evaluated the performance of linear models, we turn to using deeper neural networks for our policy net. (A discussion of the hyperparameters used is provided in the appendix.) Identically to above, we evaluate PPO, A2C and DQN on varying start state potentials and K. Each algorithm is run with 3 random seeds, and in all plots we show minimum, mean, and maximum performance. Results are shown in Figures 3, 4. Note that all algorithms show variation in performance across different settings of potentials and K, and show noticeable drops in performance with harder difficulty settings. When varying potential in Figure 3, both PPO and A2C show larger variation than DQN, though PPO mostly matches or beats DQN in performance. When varying K, PPO shows less variation than DQN. A2C shows the greatest variation and worst performance out of all three methods.


Defender trained with PPO for varying potential, K=15

Defender trained with A2C for varying potential, K=15

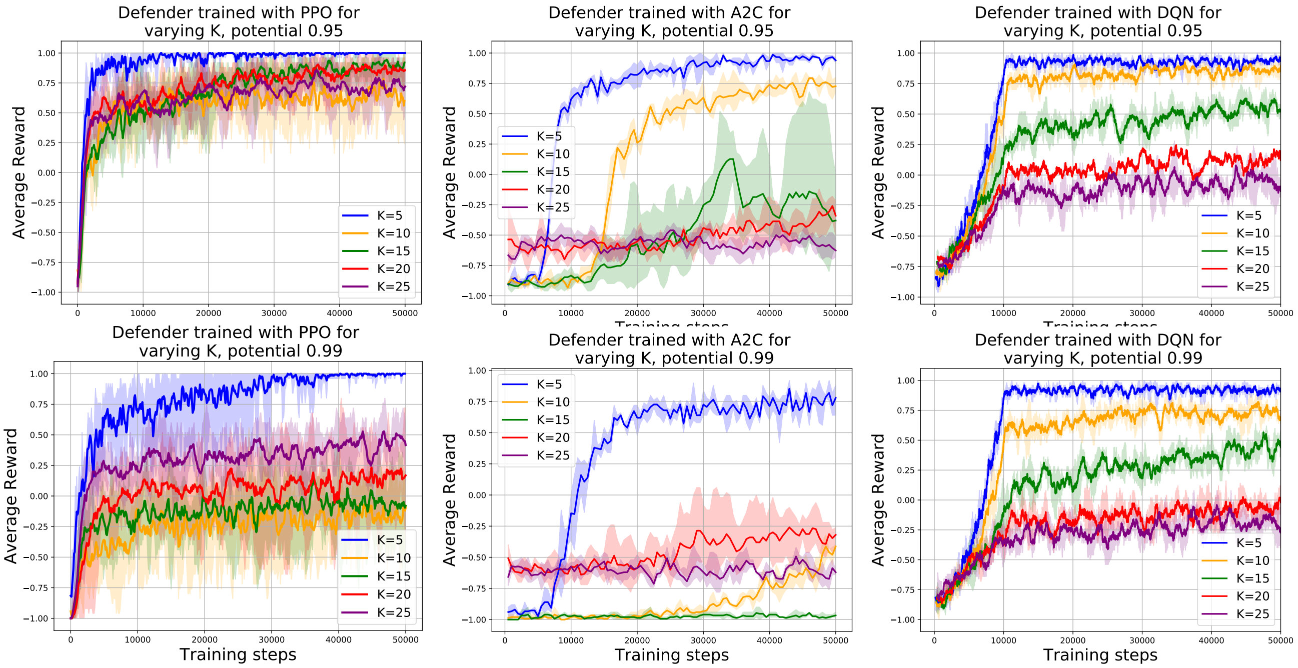
Defender trained with DQN for varying potential, K=15

Defender trained with PPO for varying potential, K=20

Defender trained with A2C for varying potential, K=20

Defender trained with DQN for varying potential, K=20

Figure 3: Training defender agent with PPO, A2C and DQN for varying values of potentials and two different choices of K with a deep network. Overall, we see significant improvements over using a linear model. For smaller K, DQN performs relatively consistently across different potential values, though not quite matching PPO – left and right panes, row 2. A2C tends to fare worse than both PPO and DQN.


Defender trained with PPO for varying K, potential 0.95

Defender trained with A2C for varying K, potential 0.95

Defender trained with DQN for varying K, potential 0.95

Defender trained with PPO for varying K, potential 0.99

Defender trained with A2C for varying K, potential 0.99

Defender trained with DQN for varying K, potential 0.99

Figure 4: Training defender agent with PPO, A2C and DQN for varying values of K and two different choices of potential (top and bottom row) with a deep network. All three algorithms show a noticeable variation in performance over different difficulty settings, though we note that PPO seems to be more robust to larger K (which corresponds to longer episodes). A2C tends to fare worse than both PPO and DQN.

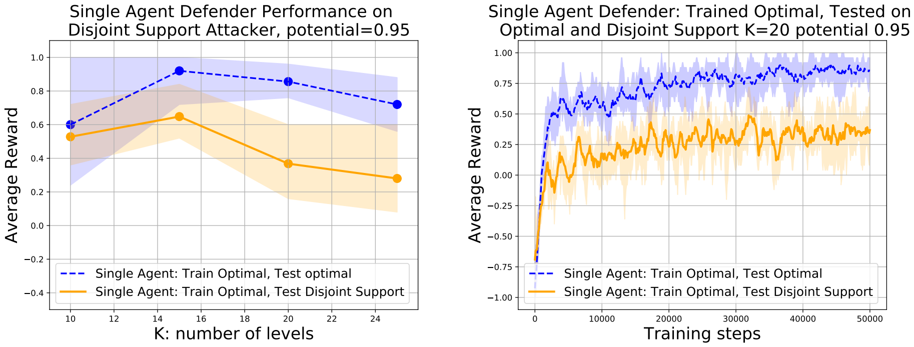


Figure 5: Plot showing overfitting to opponent strategies. A defender agent is trained on the optimal attacker, and then tested on (a) another optimal attacker environment (b) the disjoint support attacker environment. The left pane shows the resulting performance drop when switching to testing on the same opponent strategy as in training to a different opponent strategy. The right pane shows the result of testing on an optimal attacker vs a disjoint support attacker during training. We see that performance on the disjoint support attacker converges to a significantly lower level than the optimal attacker.

5 Generalization in RL and Multiagent Learning

In the previous section we trained defender agents using popular RL algorithms, and then evaluated the performance of the trained agents on the environment. However, noting that the Attacker-Defender game has a known optimal policy that works perfectly in any game setting, we can evaluate our RL algorithms on *generalization*, not just performance.

We take a setting of parameters, start state potential 0.95 and $K = 5, 10, 15, 20, 25$ where we have seen the RL agent perform well, and change the procedural attacker policy between train and test. In detail, we train RL agents for the defender against an attacker playing optimally, and test these agents for the defender on the disjoint support attacker. The results are shown in Figure 5 where we can see that while performance is high, the RL agents fail to generalize against an opponent that is theoretically easier. This leads to the natural question of how we might achieve stronger generalization in our environment.

5.1 Training an Attacker Agent

One way to mitigate this overfitting issue is to set up a method of also training the attacker, so that we can train the defender against a learned attacker, or in a multiagent setting. However, determining the correct setup to train the attacker agent first requires devising a tractable parametrization of the action space. A naive implementation of the attacker would be to have the policy output how many pieces should be allocated to A for each of the $K + 1$ levels (as in the construction from Spencer (1994)). This can grow very rapidly in K, which is clearly impractical. To address this, we first prove a new theorem that enables us to parametrize an optimal attacker with a much smaller action space.

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 $i < l$.*

Proof. For each $l \in \{0, 1, \ldots, K\}$, let A_l be the set of all pieces from levels K down to and excluding level l, with $A_K = \emptyset$. We have $\phi(A_{i+1}) \leq \phi(A_i)$, $\phi(A_K) = 0$ and $\phi(A_0) = \phi(S_0) \geq 1$. Thus, there exists an l such that $\phi(A_l) < 0.5$ and $\phi(A_{l-1}) \geq 0.5$. If $\phi(A_{l-1}) = 0.5$, we set $A_{l-1} = A$ and B the complement, and are done. So assume $\phi(A_l) < 0.5$ and $\phi(A_{l-1}) > 0.5$

Since A_{l-1} only contains pieces from levels K to l, potentials $\phi(A_l)$ and $\phi(A_{l-1})$ are both integer multiples of $2^{-(K-l)}$, the value of a piece in level l. Letting $\phi(A_l) = n \cdot 2^{-(K-l)}$ and $\phi(A_{l-1}) = m \cdot 2^{-(K-l)}$, we are guaranteed that level l has $m - n$ pieces, and that we can move $k < m - n$ pieces from A_{l-1} to A_l such that the potential of the new set equals 0.5. $\square$

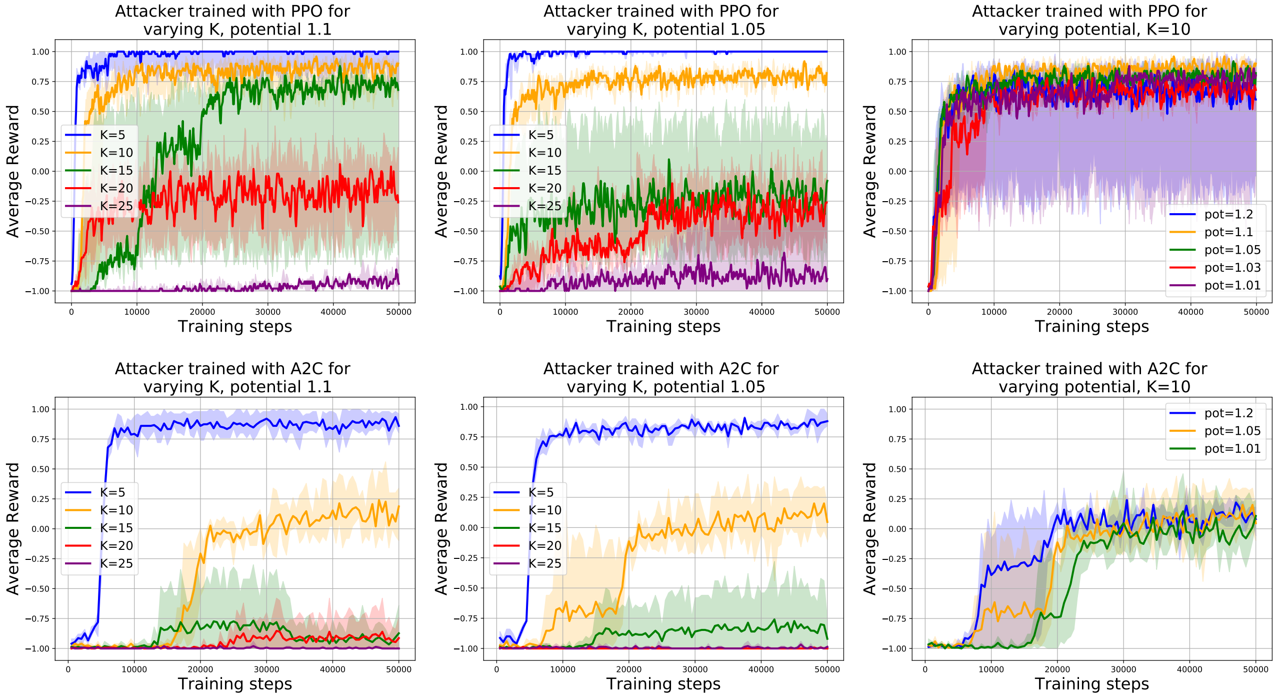


Figure 6: Performance of PPO and A2C on training the attacker agent for different difficulty settings. DQN performance was very poor (reward < -0.8 at $K = 5$ with best hyperparams). We see much greater variation of performance with changing K, which now affects the sparseness of the reward as well as the size of the action space. There is less variation with potential, but we see a very high performance variance (top right pane) with lower (harder) potentials.

This theorem gives a different way of constructing and parametrizing an optimal attacker. The attacker outputs a level l. 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 keep the potentials of A and B as close as possible. Theorem 3 guarantees the optimal policy is representable, and the action space is linear instead of exponential in K.

With this setup, we train an attacker agent against the optimal defender with PPO, A2C, and DQN. The DQN results were very poor, and so we show results for just PPO and A2C. In both algorithms we found there was a large variation in performance when changing K, which now affects both reward sparsity and action space size. We observe less outright performance variability with changes in potential for small K but see an increase in the variance (Figure 6).

5.2 Learning through Multiagent Play

With this attacker training, we can now look at learning in a multiagent setting. We first explore the effects of varying the potential and K as shown in Figure 7. Overall, we find that the attacker fares *worse* in multiagent play than in the single agent setting. In particular, note that in the top left pane of Figure 7, we see that the attacker loses to the defender even with $\phi(S_0) = 1.1$ for $K = 15$. We can compare this to Figure 6 where with PPO, we see that with $K = 15$, and potential 1.1, the single agent attacker succeeds in winning against the optimal defender.

5.3 Single Agent and Multiagent Generalization Across Opponent Strategies

Finally, we return again to our defender agent, and test generalization between the single and multiagent settings. We train a defender agent in the single agent setting against the optimal attacker, and test on a an attacker that only uses the Disjoint Support strategy. We also test a defender trained in the multiagent setting (which has never seen any hardcoded strategy of this form) on the Disjoint Support attacker. The results are shown in Figure 8. We find that the defender trained as part of a

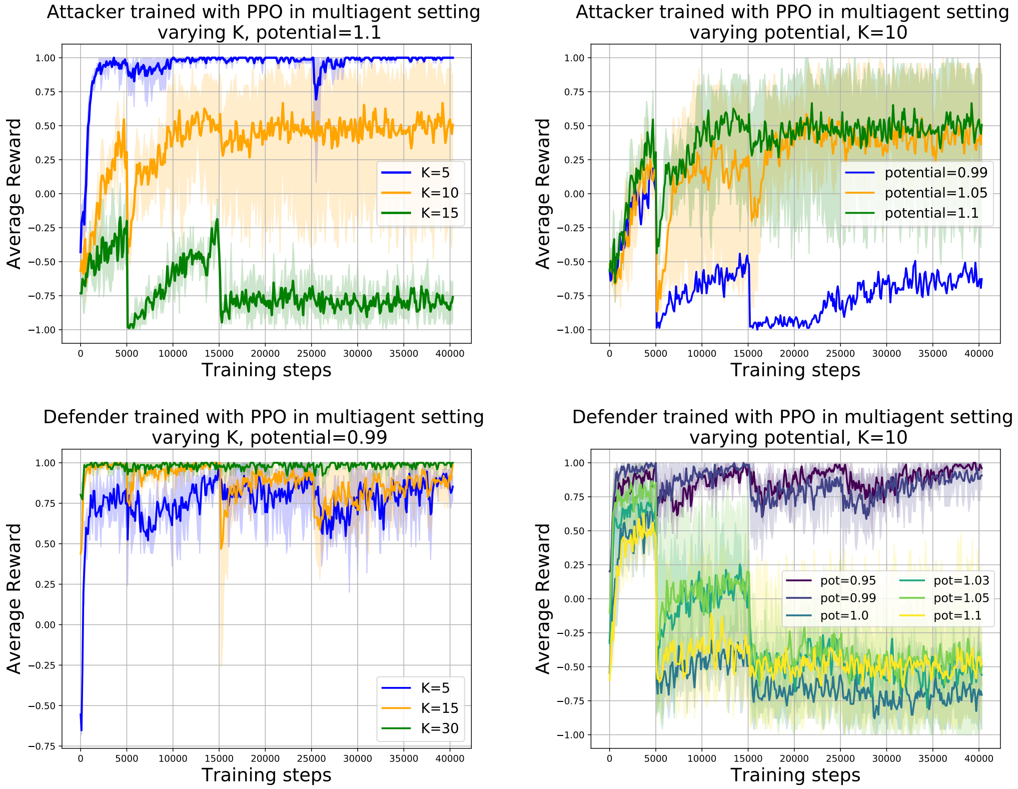


Figure 7: Performance of attacker and defender agents when learning in a multiagent setting. In the top panes, solid lines denote attacker performance. In the bottom panes, solid lines are defender performance. The sharp changes in performance correspond to the times we switch which agent is training. We note that the defender performs much better in the multiagent setting: comparing the top and bottom left panes, we see far more variance and lower performance of the attacker compared to the defender performance below. Furthermore, the attacker loses to the defender for potential 1.1 at $K = 15$, despite winning against the optimal defender in Figure 6. We also see (right panes) that the attacker has higher variance and sharper changes in its performance even under conditions when it is guaranteed to win.

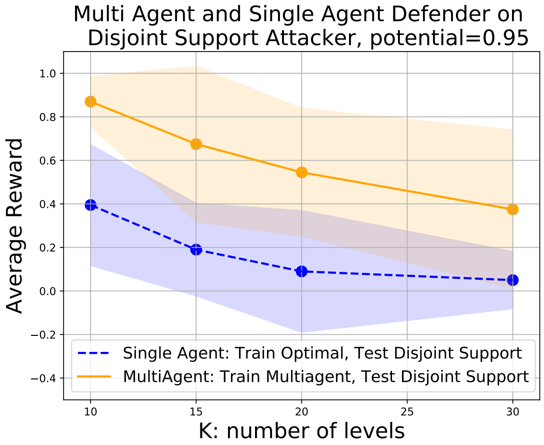


Figure 8: Results for generalizing to different attacker strategies with single agent defender and multiagent defender. The figure single agent defender trained on the optimal attacker and then tested on the disjoint support attacker and a multiagent defender also tested on the disjoint support attacker for different values of K. We see that multiagent defender generalizes better to this unseen strategy than the single agent defender.

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 parameters with RL algorithm
```

multiagent setting generalizes noticeably better than the single agent defender. We show the results over 8 random seeds and plot the mean (solid line) and shade in the standard deviation.

6 Training with Self Play

In the previous section, we showed that with a new theoretical insight into a more efficient attacker action space paramterization (Theorem 3), it is possible to train an attacker agent. The attacker agent was parametrized by a neural network different from the one implementing the defender, and it was trained in a multiagent fashion. In this section, we present additional theoretical insights that enables training by *self-play*: using a single neural network to parametrize both the attacker and defender.

The key insight is the following: both the defender's optimal strategy and the construction of the optimal attacker in Theorem 3 depend on a primitive operation that takes a partition of the pieces into sets A, B and determines which of A or B is "larger" (in the sense that it has higher potential). For the defender, this leads directly to a strategy that destroys the set of higher potential. For the attacker, this primitive can be used in a binary search procedure to find the desired partition in Theorem 3: given an initial partition A, B into a prefix and a suffix of the pieces sorted by level, we determine which set has higher potential, and then recursively find a more balanced split point inside the larger of the two sets. This process is summarized in Algorithm 1.

Thus, by training a single neural network designed to implement this primitive operation — determining which of two sets has higher potential — we can simultaneously train both an attacker and a defender that invoke this neural network. We use DQN for this purpose because we found empirically that it is the quickest among our alternatives at converging to consistent estimates of relative potentials on sets. We train both agents in this way, and test the defender agent on the same attacker used in Figures 2, 3 and 4. The results in Figure 9 show that a defender trained through self play significantly outperforms defenders trained against a procedural attacker.

7 Supervised Learning vs RL

Aside from testing the generalization of RL, the Attacker-Defender game also enables us to make a comparison with Supervised Learning. The closed-form optimal policy enables an evaluation of the ground truth on a *per move* basis. We can thus compare RL to a Supervised Learning setup, where we classify the correct action on a large set of sampled states. To carry out this test in practice, we first train a defender policy with reinforcement learning, saving all observations seen to a dataset. We then train a supervised network (with the same architecture as the defender policy) on this dataset to classify the optimal action. We then test the supervised network to determine how well it can play.

We find an interesting dichotomy between the proportion of correct moves and the average reward. Unsurprisingly, supervised learning boasts a higher proportion of correct moves: if we keep count of the ground truth correct move for each turn in the game, RL has a lower proportion of correct moves compared to supervised learning (Figure 10 left pane). However, in the right pane of Figure 10, we see that RL is better at playing the game, achieving higher average reward for all difficulty settings, and significantly beating supervised learning as K grows.

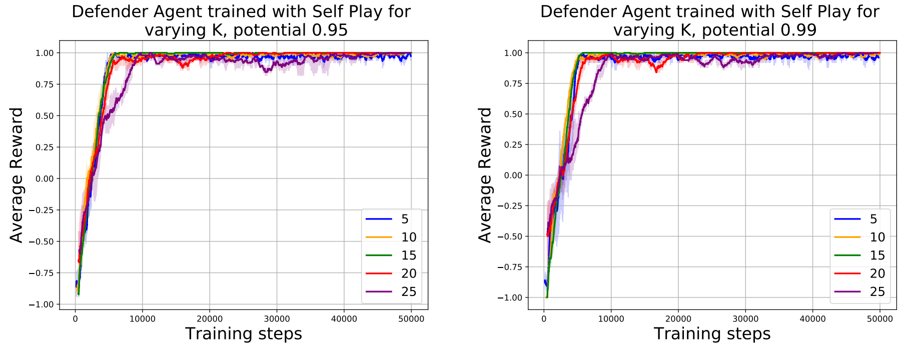


Figure 9: We train an attacker and defender via self play using a DQN. The defender is implemented as in Figures 4, 3, and the same neural network is used to train an attacker agent, performing binary search according to the Q values on partitions of the input space A, B. We then test the defender agent on the same procedural attacker used in Figures 4, 3, and find that self play shows markedly improved performance.

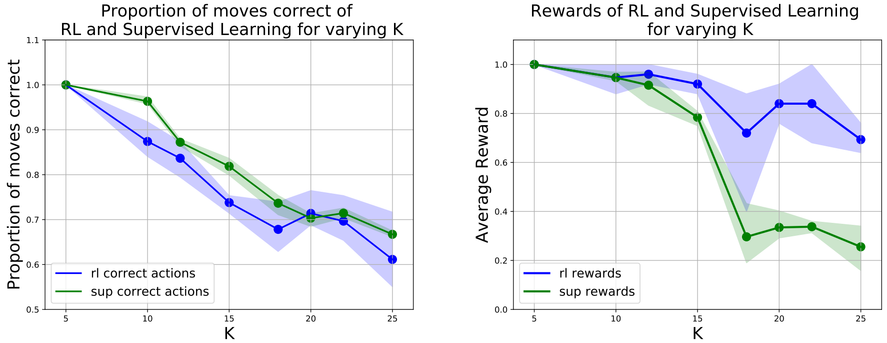


Figure 10: Plots comparing the performance of Supervised Learning and RL on the Attacker Defender Game for different choices of K. The left pane shows the proportion of moves correct for supervised learning and RL (according to the ground truth). Unsurprisingly, we see that supervised learning is better on average at getting the ground truth correct move. However, RL is better at playing the game: a policy trained through RL significantly outperforms a policy trained through supervised learning (right pane), with the difference growing for larger K.

This contrast forms an interesting counterpart to recent findings of Silver et al. (2017), who in the context of Go also compared reinforcement learning to supervised approaches. A key distinction is that their supervised work was relative to a heuristic objective, whereas in our domain we are able to compare to provably optimal play. This both makes it possible to rigorously define the notion of a mistake, and also to perform more fine-grained analysis as we do in the remainder of this section.

Specifically, how is it that the RL agent is achieving higher reward in the game, if it is making more mistakes at a per-move level? To gain further insight into this, we categorize the per-move mistakes into different types, and study them separately. In particular, suppose the defender is presented with a partition of the pieces into two sets, where as before we assume without loss of generality that $\phi(A) \geq \phi(B)$. We say that the defender makes a *terminal mistake* if $\phi(A) \geq 0.5$ while $\phi(B) < 0.5$, but the defender chooses to destroy B. Note that this means the defender now faces a forced loss against optimal play, whereas it could have forced a win with optimal play had it destroyed A. We also define a subset of the family of terminal mistakes as follows: we say that the defender makes a *fatal mistake* if $\phi(A) + \phi(B) < 1$, but $\phi(A) \geq 0.5$ and the defender chooses to destroy B. Note that a fatal mistake is one that converts a position where the defender had a forced win to one where it has a forced loss.

In Figure 11, we see that especially as K gets larger, reinforcement learning makes terminal and fatal mistakes at a much lower rate than supervised learning does. This suggests a basis for the different in performance: even if supervised learning is making fewer mistakes overall, it is making more mistakes in certain well-defined consequential situations.

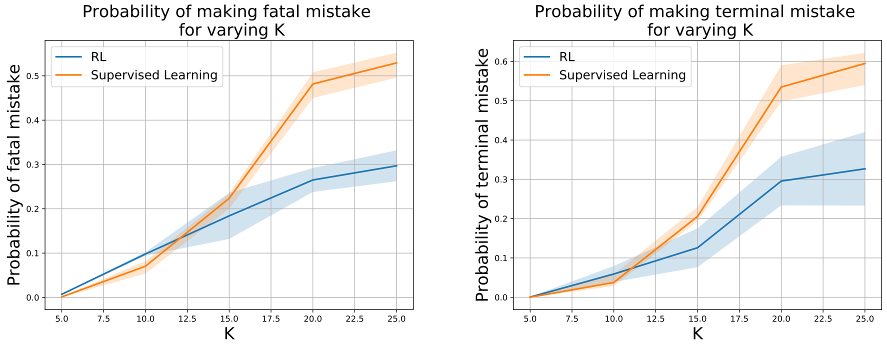


Figure 11: Figure showing the frequencies of different kinds of mistakes made by supervised learning and RL that would cost the game. The left pane shows the frequencies of fatal mistakes: where the agent goes from a winning state (potential < 1) to a losing state (potential ≥ 1). A superset of this kind of mistake, terminal mistakes, look at where the agent makes the wrong choice (irrespective of state potential), destroying (wlog) set A with $\phi(A) < 0.5$, instead of B, with $\phi(B) \geq 0.5$. In both cases we see that RL makes significantly fewer mistakes than supervised learning, particularly as difficulty increases.

8 Conclusion

In this paper, we have proposed Erdos-Selfridge-Spencer games as rich environments for investigating reinforcement learning, exhibiting continuously tunable difficulty and an exact combinatorial characterization of optimal behavior. We have demonstrated that algorithms can exhibit wide variation in performance as we tune the game's difficulty, and we use the characterization of optimal behavior to evaluate generalization over raw performance. We provide theoretical insights that enable multiagent play and, through binary search, self play. Finally, we compare RL and Supervised Learning, highlighting interesting tradeoffs between per move optimality, average reward and fatal mistakes. We also develop further results in the Appendix, including an analysis of catastrophic forgetting, generalization across different values of the game's parameters, and a method for investigating measures of the model's confidence. We believe that this family of combinatorial games can be used as a rich environment for gaining further insights into deep reinforcement learning.

REFERENCES

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pp. 1329–1338, 2016.

Paul Erdos and John Selfridge. On a combinatorial game. *Journal of Combinatorial Theory*, 14: 298–301, 1973.

Christopher Hesse, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015. ISSN 0028-0836. URL `http://dx.doi.org/10.1038/nature14236`.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv preprint arxiv:1602.01783*, 2016.

Aravind Rajeswaran, Sarvjeet Ghotra, Sergey Levine, and Balaraman Ravindran. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arxiv:1707.06347*, 2017.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct 2017. ISSN 0028-0836. URL `http://dx.doi.org/10.1038/nature24270`.

Joel Spencer. Randomization, derandomization and antirandomization: Three games. *Theoretical Computer Science.*, 131:415–429, 09 1994.

Andrew Stephenson. Lxxi. on induced stability. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 17(101):765–766, 1909.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *arXiv preprint arXiv:1703.06907*, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Appendix

Additional Definition and Results from Section 4

Note that for evaluating the defender RL agent, we initially use a slightly suboptimal attacker, which randomizes over playing optimally and with a *disjoint support strategy*. The disjoint support strategy is a suboptimal verison (for better exploration) of the prefix attacker described in Theorem 3. Instead of finding the partition that results in sets A, B as equal as possible, the disjoint support attacker greedily picks A, B so that there is a potential difference between the two sets, with the fraction of total potential for the smaller set being uniformly sampled from $[0.1, 0.2, 0.3, 0.4]$ at each turn. This exposes the defender agent to sets of uneven potential, and helps it develop a more generalizable strategy.

To train our defender agent, we use a fully connected deep neural network with 2 hidden layers of width 300 to represent our policy. We decided on these hyperparameters after some experimentation with different depths and widths, where we found that network width did not have a significant effect on performance, but, as seen in Section 4, slightly deeper models (1 or 2 hidden layers) performed noticeably better than shallow networks.

Other Generalization Phenomena

Generalizing Over Different Potentials and K

In the main text we examined how our RL defender agent performance varies as we change the difficulty settings of the game, either the potential or K. Returning again to the fact that the Attacker-Defender game has an expressible optimal that generalizes across all difficulty settings, we might wonder how training on one difficulty setting and testing on a different setting perform. Testing on different potentials in this way is straightforwards, but testing on different K requires a slight reformulation. our input size to the defender neural network policy is $2(K + 1)$, and so naively changing to a different number of levels will not work. Furthermore, training on a smaller K and testing on larger K is not a fair test – the model cannot be expected to learn how to weight the lower levels. However, testing the converse (training on larger K and testing on smaller K) is both easily implementable and offers a legitimate test of generalization. We find (a subset of plots shown in Figure 12) that when varying potential, training on harder games results in better generalization. When testing on a smaller K than the one used in training, performance is inverse to the difference between train K and test K.

Catastrophic Forgetting and Curriculum Learning

Recently, several papers have identified the issue of catastrophic forgetting in Deep Reinforcement Learning, where switching between different tasks results in destructive interference and lower performance instead of positive transfer. We witness effects of this form in the Attacker-Defender games. As in Section 8, our two environments differ in the K that we use – we first try training on a small K, and then train on larger K. For lower difficulty (potential) settings, we see that this curriculum learning improves play, but for higher potential settings, the learning interferes catastrophically, Figure 13

Understanding Model Failures

Value of the Null Set

The significant performance drop we see in Figure 5 motivates investigating whether there are simple rules of thumb that the model has successfully learned. Perhaps the simplest rule of thumb is learning the value of the null set: if one of A, B (say A) consists of only zeros and the other (B) has some pieces, the defender agent should reliably choose to destroy B. Surprisingly, even this simple rule of thumb is violated, and even more frequently for larger K, Figure 14.

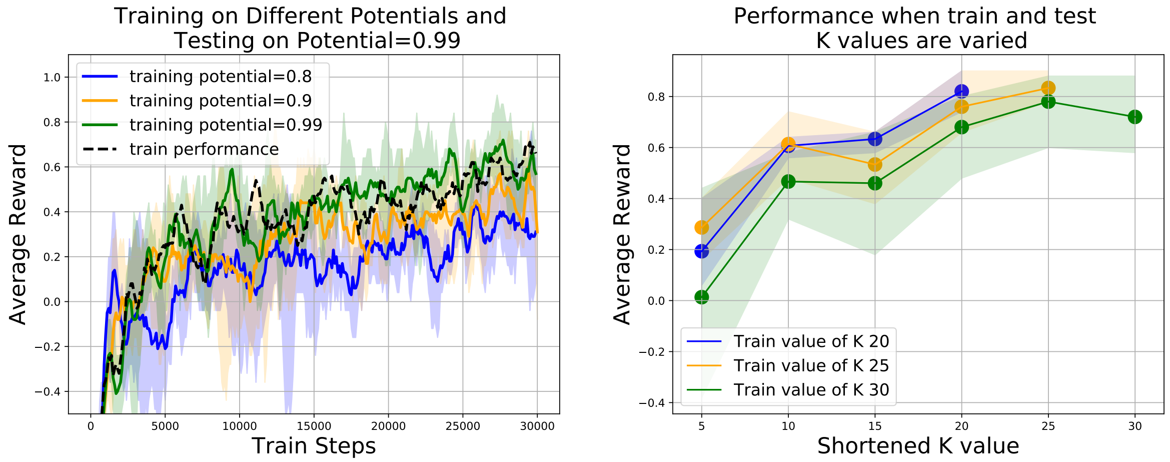


Figure 12: On the left we train on different potentials and test on potential 0.99. We find that training on harder games leads to better performance, with the agent trained on the easiest potential generalizing worst and the agent trained on a harder potential generalizing best. This result is consistent across different choices of test potentials. The right pane shows the effect of training on a larger K and testing on smaller K. We see that performance appears to be inversely proportional to the difference between the train K and test K.

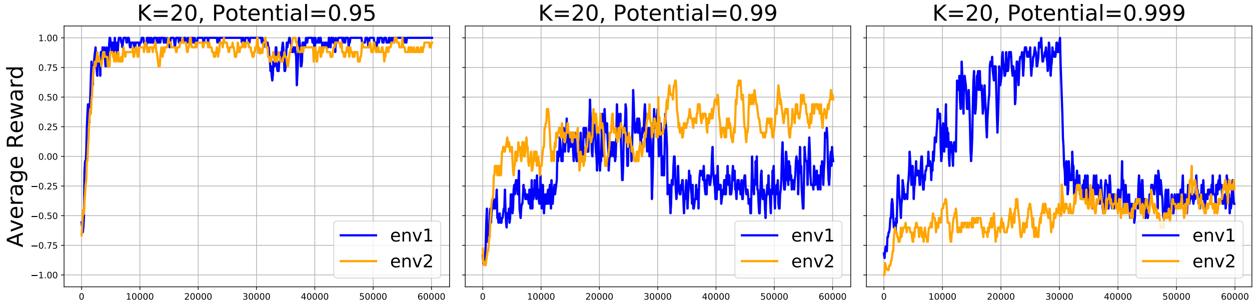


Figure 13: Defender agent demonstrating catastrophic forgetting when trained on environment 1 with smaller K and environment 2 with larger K.

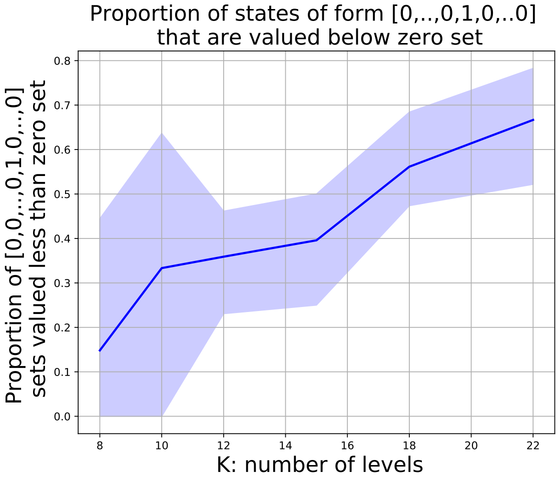


Figure 14: Figure showing proportion of sets of form $[0, ..., 0, 1, 0, ..., 0]$ that are valued less than the null set. Out of the $K + 1$ possible one hot sets, we determine the proportion that are not picked when paired with the null (zero) set, and plot this value for different K.

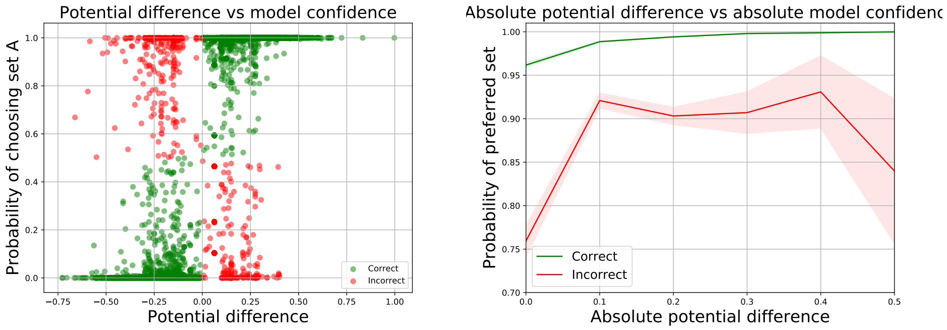


Figure 15: Confidence as a function of potential difference between states. The top figure shows true potential differences and model confidences; green dots are moves where the model prefers to make the right prediction, while red moves are moves where it prefers to make the wrong prediction. The right shows the same data, plotting the *absolute* potential difference and absolute model confidence in its preferred move. Remarkably, an increase in the potential difference associated with an increase in model confidence over a wide range, even when the model is wrong.

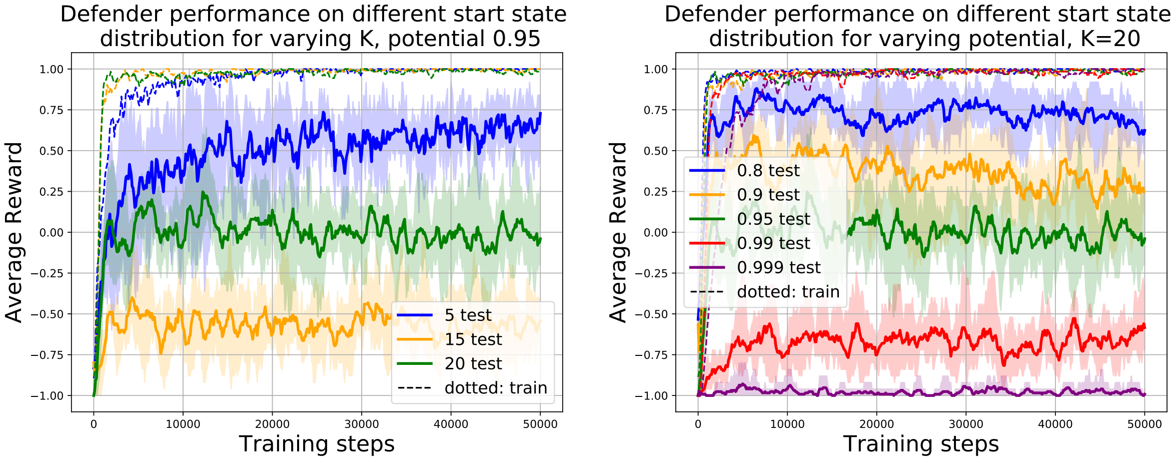


Figure 16: Change in performance when testing on different state distributions

Model Confidence

We can also test to see if the model outputs are well *calibrated* to the potential values: is the model more confident in cases where there is a large discrepancy between potential values, and fifty-fifty where the potential is evenly split? The results are shown in Figure 15.

8.1 Generalizing across Start States and Opponent Strategies

In the main paper, we mixed between different start state distributions to ensure a wide variety of states seen. This begets the natural question of how well we can generalize across start state distribution if we train on purely one distribution. The results in Figure 16 show that training naively on an 'easy' start state distribution (one where most of the states seen are very similar to one another) results in a significant performance drop when switching distribution.

In fact, the amount of possible starting states for a given K and potential $\phi(S_0) = 1$ grows super exponentially in the number of levels K. We can state 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* $0.25K^2 \leq \Theta(K^2) \leq 0.5K^2$ *)*

We give a sketch proof.

Proof. Let such a state be denoted S. Then a trivial upper bound can be computed by noting that each s_i can take a value up to $2^{(K-i)}$, and producting all of these together gives roughly $2^{K/2}$.

For the lower bound, we assume for convenience that K is a power of 2 (this assumption can be avoided). Then look at the set of non-negative integer solutions of the system of simultaneous equations

$$a_{j-1}2^{1-j} + a_j 2^{-j} = 1/K$$

where j ranges over all even numbers between $\log(K)+1$ and K. The equations don't share any variables, so the solution set is just a product set, and the number of solutions is just the product $\prod_j (2^{j-1}/K)$ where, again,j ranges over even numbers between $\log(K)+1$ and K. This product is roughly $2^{K^2/4}$. □