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.

Additional Results from Section 5

Here in Figure 11 we show results of training the attacker agent using the alternative parametrization given with Theorem 3 with PPO and A2C (DQN results were very poor and have been omitted.)

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

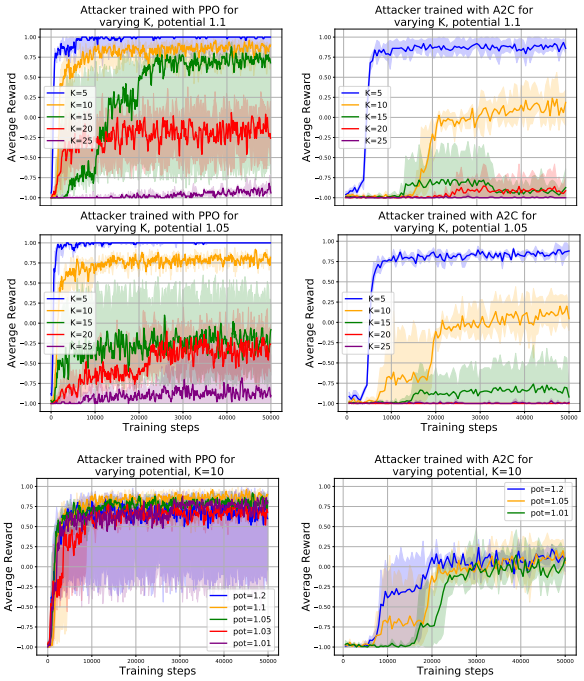

Figure 11. Performance of PPO and A2C on training the attacker agent for different difficulty settings. DQN performance was very poor (reward < -0.8 at $K = 5$ with best hyperparams). We see much greater variation of performance with changing K, which now affects the sparseness of the reward as well as the size of the action space. There is less variation with potential, but we see a very high performance variance with lower (harder) potentials.

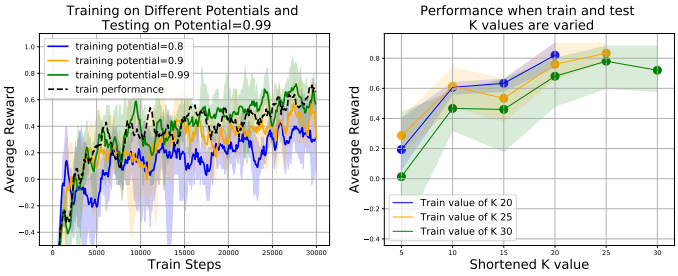

Figure 12. On the left we train on different potentials and test on potential 0.99. We find that training on harder games leads to better performance, with the agent trained on the easiest potential generalizing worst and the agent trained on a harder potential generalizing best. This result is consistent 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.

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.

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 $2^{K^2/4}$. □

8.2. Comparison to Random Search

Inspired by the work of (Mania et al., 2018), we also include the performance of random search in Figure **??**

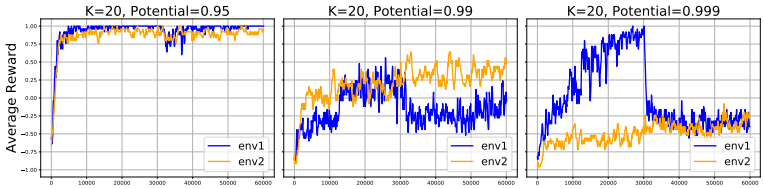

Figure 13. Defender agent demonstrating catastrophic forgetting when trained on environment 1 with smaller K and environment 2 with larger K.

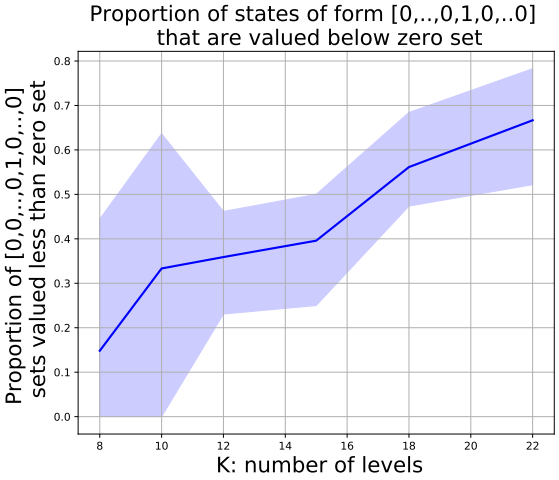

Figure 14. Figure showing proportion of sets of form $[0, ..., 0, 1, 0, ..., 0]$ that are valued less than the null set. Out of the $K + 1$ possible one hot sets, we determine the proportion that are not picked when paired with the null (zero) set, and plot this value for different K.

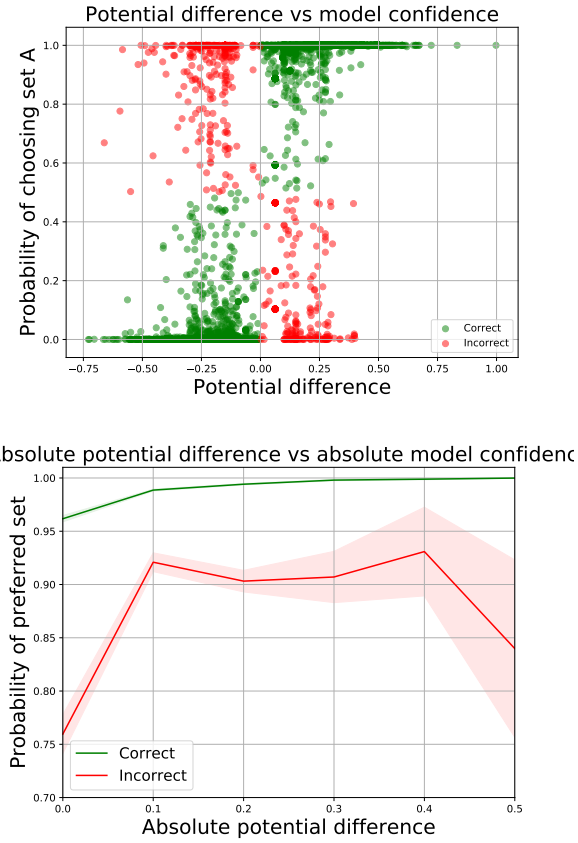

Figure 15. Confidence as a function of potential difference between states. The top figure shows true potential differences and model confidences; green dots are moves where the model prefers to make the right prediction, while red moves are moves where it prefers to make the wrong prediction. The right 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.

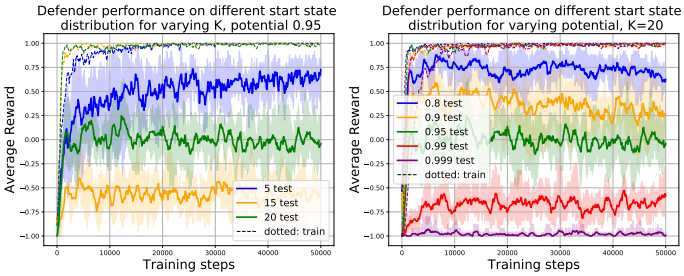

Figure 16. Change in performance when testing on different state distributions

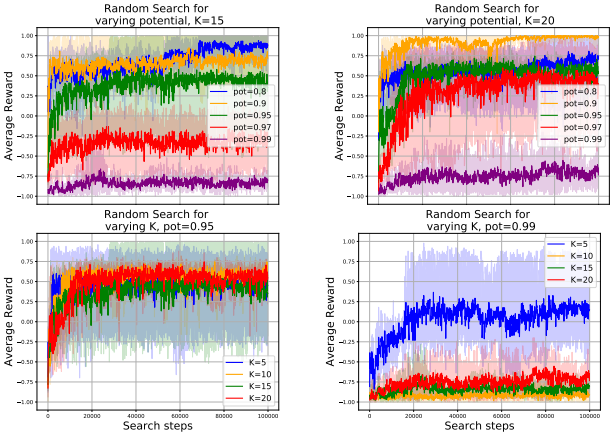

Figure 17. Performance of random search from (Mania et al., 2018). The best performing RL algorithms do better.