Policy Learning for Continuous Space Security Games using Neural Networks

Nitin Kamra[1], **Umang Gupta**[1], **Fei Fang**[2], **Yan Liu**[1], **Milind Tambe**[1]
University of Southern California[1], Carnegie Mellon University[2]
{nkamra, umanggup, yanliu.cs, tambe}@usc.edu[1], feifang@cmu.edu[2]

Abstract

A wealth of algorithms centered around (integer) linear programming have been proposed to compute equilibrium strategies in security games with discrete states and actions. However, in practice many domains possess continuous state and action spaces. In this paper, we consider a continuous space security game model with infinite-size action sets for players and present a novel deep learning based approach to extend the existing toolkit for solving security games. Specifically, we present (i) OptGradFP, a novel and general algorithm that searches for the optimal defender strategy in a parameterized continuous search space, and can also be used to learn policies over multiple game states simultaneously; (ii) OptGradFP-NN, a convolutional neural network based implementation of OptGradFP for continuous space security games. We demonstrate the potential to predict good defender strategies via experiments and analysis of OptGradFP and OptGradFP-NN on discrete and continuous game settings.

1 Introduction

Stackelberg Security Games (SSGs) have been extensively used to model defender-adversary interaction in protecting important infrastructure targets such as airports, ports, and flights (Tambe 2011; Rosenfeld and Kraus 2017; Cermák et al. 2016; Basilico et al. 2017). In SSGs, the defender (referred to as "she") perpetually defends a set of targets with limited resources. The adversary (referred to as "he") can surveil and learn the defender's strategy and plan an attack based on this information. Exact and approximate approaches have been proposed to maximize the defender's expected utility in SSGs given that the adversary will best respond to her strategy (Conitzer and Sandholm 2006; Kiekintveld et al. 2009; Amin, Singh, and Wellman 2016). Most approaches rely on linear programming (LP) and mixed integer linear programming (MILP) which do not scale well to large-scale and complex security games, despite techniques such as column generation and cutting planes (Tambe 2011).

Recently, there has been an increasing interest in SSGs for green security domains such as protecting wildlife (Kar et al. 2015; Wang, Zhang, and Zhong 2017), fisheries (Haskell et al. 2014) and forests (Johnson, Fang, and Tambe 2012). Unlike infrastructure protection domains which have discrete locations, green security domains are categorized by continuous spaces (e.g., a whole conservation area needs protection). Previous works mostly discretize the area into grid cells and restrict the players' actions to discrete sets (Yang et al. 2014; Haskell et al. 2014; Gan et al. 2017) to find the equilibrium strategy using LP or MILP. However, a coarse discretization may lead to low solution quality, and a fine-grained discretization would make it intractable to compute the optimal defender strategy using mathematical programming based techniques, especially when there are multiple defender resources. Other approaches handle continuous space by exploiting spatio-temporal structure of the game (Fang, Jiang, and Tambe 2013; Yin, An, and Jain 2014) or by numerically solving differential equations for special cases (Johnson, Fang, and Tambe 2012), which cannot be extended to general settings.

In this paper, we provide a novel approach for solving security games based on policy learning, fictitious play and deep learning. This approach extends the existing toolkit to handle complex settings such as general games with continuous spaces. We make the following major contributions:

- We present OptGradFP, a novel and general algorithm which considers continuous space parameterized policies for two-player zero-sum games and optimizes them using policy gradient learning and game theoretic fictitious play.
- We provide a continuous space security game model for forest protection, which incorporates infinite action sets over two-dimensional continuous areas and asymmetric target distributions. Existing approaches based on MILP or differential equations fail to handle such games.
- We provide a convolutional neural network based implementation of OptGradFP (called OptGradFP-NN), which after learning on various game states, shifts computation in security games from online to offline, by predicting good defender strategies on previously unseen states.

Our experimental analysis with OptGradFP and OptGradFP-NN demonstrates the superiority of our approach against comparable approaches such as StackGrad (Amin, Singh, and Wellman 2016) and Cournot Adjustment (CA) (Fudenberg and Levine 1998). Our approach gives a good strategy for both players, even when the baselines fail to converge.

2 Preliminaries

We use small letters (x) to denote scalars, bold small letters ($\boldsymbol{x}$) to denote vectors, capitals (X) to denote random variables and bold capitals ($\boldsymbol{X}$) to denote random vectors. $\mathbb{R}$ represents the set of real numbers. Saying $\boldsymbol{x} \in [\boldsymbol{a}, \boldsymbol{b}]$ implies that all corresponding elements of $\boldsymbol{x}$ are $\geq$ those of $\boldsymbol{a}$ and $\leq$ those of $\boldsymbol{b}$. $\mathcal{N}(\mu, \nu^2)$ is the normal distribution with mean μ and variance ν^2.

The sigmoid function $\frac{1}{1+\exp(-\boldsymbol{z})}$ is denoted by $\sigma(\boldsymbol{z})$. The logit function is defined as: $\text{logit}(\boldsymbol{x}) \triangleq \log \frac{\boldsymbol{x}}{\boldsymbol{1}-\boldsymbol{x}}$ $\forall \boldsymbol{x} \in [\boldsymbol{0}, \boldsymbol{1}]$. Note that the sigmoid and logit functions are inverses of each other i.e. $\sigma(\text{logit}(\boldsymbol{x})) = \boldsymbol{x}$.

2.1 Stackelberg Security Games

A Stackelberg Security Game (SSG) (Kiekintveld et al. 2009; Korzhyk et al. 2011) is a leader-follower game between a defender and an adversary (a.k.a. opponent). Given a game state (locations of targets), an action or a pure strategy of the defender is to allocate the resources to protect a subset of targets in a feasible way (e.g., assign each resource to protect one target). A pure strategy of the adversary is to attack a target. The mixed strategy of a player is a probability distribution over the pure strategies. A player's policy is a mapping from the game state to a mixed strategy.

The payoff for a player is decided by the game state and joint action of both players, and the expected utility function is defined as the expected payoff over all possible states and joint actions given the players' policies. In this paper, we restrict ourselves to zero-sum games while deferring investigation of general-sum games to future work.

An attacker best responds to a defender policy if he chooses a policy that maximizes his expected utility, given the defender's policy. The optimal defender policy in SSGs is one that maximizes her expected utility, given that the attacker best responds to it and breaks ties in favor of the defender. In zero-sum SSGs, the optimal defender policy is the same as the defender policy in any Nash Equilibrium (NE).

2.2 Fictitious Play in Normal Form Games

Fictitious play (FP) is a learning rule where each player best responds to the empirical frequency of their opponent's play. It converges to a NE under various settings including two-player zero-sum games (Fudenberg and Levine 1998).

2.3 Policy Gradient Theorem

According to the policy gradient theorem (Sutton et al. 1999), given a function $f(\cdot)$ and a random variable $\boldsymbol{X} \sim p(\boldsymbol{x}|\boldsymbol{\theta})$ whose distribution is parameterized by parameters $\boldsymbol{\theta}$, the gradient of the expected value of $f(\cdot)$ with respect to $\boldsymbol{\theta}$ can be computed as

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X})] = \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{X}|\boldsymbol{\theta})] \tag{1}$$

We can approximate the gradient on the right-hand side by sampling B samples $\{\boldsymbol{x}_i\}_{i=1:B} \sim p(\boldsymbol{X}|\boldsymbol{\theta})$, and computing $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X})] \approx \frac{1}{B} \sum_{i=1}^{B} f(\boldsymbol{x}_i) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}_i|\boldsymbol{\theta})$. The only requirement for this to work is that the density $p(\boldsymbol{x}_i|\boldsymbol{\theta})$ should be computable and differentiable w.r.t. $\boldsymbol{\theta}$ for all $\boldsymbol{x}$. We will use the policy gradient theorem to compute the gradients of the defender and opponent utilities w.r.t. their policy parameters in our algorithm.

2.4 Logit-normal Distribution

Logit-normal is a continuous distribution with a bounded support. A random variable $X \in [0, 1]$ is said to be distributed according to a logit-normal distribution if $\text{logit}(X)$ is distributed according to a normal distribution. The density function is given by:

$$p_{ln}(X; \mu, \nu) = \frac{1}{\sqrt{2\pi}\nu} \frac{1}{x(1-x)} e^{-\frac{(\text{logit}(x)-\mu)^2}{2\nu^2}} \tag{2}$$

Unlike the normal distribution, logit-normal distribution does not have analytical expressions for its mean and standard deviation. But we can still parameterize the distribution by using the mean (μ) and standard deviation (ν) of the underlying normal distribution. If $X \sim p_{ln}(X; \mu, \nu)$, a sample of X can be drawn by sampling $\epsilon \sim \mathcal{N}(0, 1)$ and then outputting $x = \sigma(\nu\epsilon + \mu)$.

3 Game Models

We will demonstrate our algorithm on two domains:

- **Rock-Paper-Scissors**: A small stateless zero-sum game with three discrete actions. This serves as a pedagogical running example to demonstrate convergence of our algorithm to the Nash Equilibrium (NE), and get interesting insights into its behavior.
- **Forest Security Game**: A continuous state, zero-sum security game with continuous actions space for both players. While this game model is the focus of the paper and is used to illustrate our algorithm, the algorithm is general and is also applicable to other domains such as wildlife, fishery protection etc.

3.1 Rock-Paper-Scissors (RPS)

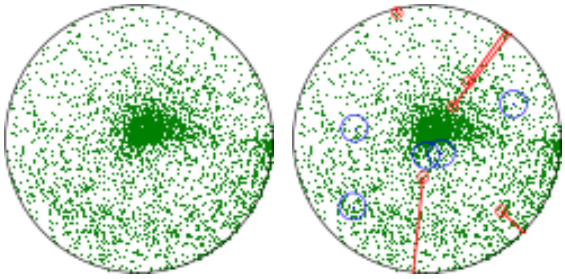
Rock-Paper-Scissors game is a classical stateless, zero-sum game with two players each having an action set: {Rock, Paper, Scissors}. Both players simultaneously choose their actions and receive rewards as shown in Figure 1.

p1 \ p2	rock	paper	scissor
rock	0,0	-1,1	1,-1
paper	1,-1	0,0	-1,1
scissor	-1,1	1,-1	0,0

Figure 1: Rewards for Rock-Paper-Scissor Game

3.2 Forest Security Game

Game model: We assume a circular forest with radius 1.0, with an arbitrary tree distribution. All locations are represented in cylindrical coordinates with the forest center as origin. The opponent (a.k.a. adversary) has n lumberjacks to chop trees in the forests. The defender has m forest guards to ambush the trespassing lumberjacks.



(a) (b)

Figure 2: (a) Forest state visualization as 120×120 image (actual state used is grayscale), and (b) Forest game with 5 guards and 5 lumberjacks visualized. Trees are green dots, guards are blue dots (blue circles show radius R_g) and lumberjacks are red dots (red circles show radius R_l).

State representation: One way of specifying the game state (s) is via number and location of all trees. This leads to a variable state-size, depending on the number of trees. Variable length representations are hard to process for most gradient-based optimization algorithms and we are mostly concerned with the relative density of trees over the forest, so we instead summarize the forest state s as a 120×120 matrix containing a grayscale image of the forest. This makes the defender and opponent policies invariant to the total number of trees in the forest and additionally allows our approach to be used for learning policies with satellite images of forests. An example input in RGB is shown in figure 2a (players' input is a grayscale version).

Defender action: The defender picks m locations, one for each guard to remain hidden, and ambush lumberjacks. The defender's action $a_D \in \mathbb{R}^{m \times 2}$ is a set of m distances $\boldsymbol{d} \in [0,1]^m$ and angles $\boldsymbol{\theta} \in [0, 2\pi]^m$ specifying the cylindrical coordinates of the guards' positions.

Opponent action: Following (Johnson, Fang, and Tambe 2012), we assume that lumberjacks cross the boundary and move straight towards the forest center. They can stop at any point on their path, chop trees in a radius R_l around the stopping point and exit back from their starting location. Since lumberjack trajectories are fully specified by their stopping coordinates, the opponent's action is to decide all stopping points. The opponent's action $a_O \in \mathbb{R}^{n \times 2}$ is a set of n distances $\boldsymbol{\rho} \in [0,1]^n$ and angles $\boldsymbol{\phi} \in [0, 2\pi]^n$ specifying the cylindrical coordinates of all chopping locations.

Rewards: A lumberjack is considered ambushed if his path comes within R_g distance from any guard's location. An ambushed lumberjack gets a penalty $-r_{pen}$ and loses all chopped trees. The total utility for the opponent ($r_O \in \mathbb{R}$) is sum of the number of trees cut by the lumberjacks and the total ambush penalty incurred. The total utility for the defender is $r_D = -r_O$.

Game play: In a single gameplay: (1) A game state is revealed, (2) Defender gives m guard locations and adversary gives n wood chopping locations, (3) Game simulator returns rewards for players. A full game is shown in figure 2b.

4 Policies and Utilities

Policies: Conventionally, a player's mixed strategy is a probability distribution over the player's actions given the game state (s). Most previous work in computational game theory focuses on how to compute a mixed strategy given a specific game state. Inspired by the recent advances in reinforcement learning, we focus on an understudied concept in games: a player's policy. A player's policy is a mapping from game states to mixed strategies. The concept of policy can help a player model different mixed strategies for different states that might be encountered in a game domain. The defender maintains a learnable policy π_D parameterized by weights $\boldsymbol{w_D}$, from which she can sample the guards' positions, given any game state. She also maintains an estimate of the adversary's policy π_O parameterized by $\boldsymbol{w_O}$, which helps her learn her own policy. Note that in case of Rock-Paper-Scissors, the finally learnt π_O will also be the opponent's Nash Equilibrium policy. However in SSGs like the forest game, a rational opponent will play a best response to the defender's deployed policy (computable separately without same parameterization as that of π_O).

We use the symbols $\pi_D(w_D), \pi_O(w_O)$ to denote policies, $\pi_D(\cdot|s; w_D), \pi_O(\cdot|s; w_O)$ to denote mixed strategies for the state s, and expressions $\pi_D(a_D|s; \boldsymbol{w_D}), \pi_O(a_O|s; \boldsymbol{w_O})$ to denote the probability of a certain action (a_D or a_O) drawn from the policy (π_D or π_O) given a state s. We sometimes skip writing w_D or w_O to promote clarity. Note that with our policy representation, functions of a policy (e.g. utilities) can be directly written as functions of the policy weights.

Utilities: The utilities of the defender and the opponent (J_D and $J_O = -J_D$ respectively) are the expected rewards obtained given the players' policies:

$$\begin{aligned} J_D(\boldsymbol{w_D}, \boldsymbol{w_O}) &= \mathbb{E}_{s,a_D,a_O}[r_D(s, a_D, a_O)] \\ &= \int_s \int_{a_D} \int_{a_O} P(s)\pi_D(a_D|s; \boldsymbol{w_D})\pi_O(a_O|s; \boldsymbol{w_O}) \\ &\qquad r_D(s, a_D, a_O)\, ds\, da_D\, da_O \end{aligned} \tag{3}$$

Note that the integral over s can be removed if we only require mixed strategies for a given state, but our method also allows learning policies over multiple states if needed.

Both the defender and the opponent want to maximize their utilities. In SSGs, the defender has to deploy her policy first, without knowing the opponent's policy. The problem faced by defender is to compute:

$$\boldsymbol{w_D^*} \in arg \max_{\boldsymbol{w_D}} \min_{\boldsymbol{w_O}} J_D(\boldsymbol{w_D}, \boldsymbol{w_O}) \tag{4}$$

The opponent observes the defender's policy and he can use this information to react with a best response to the defender's deployed policy:

$$\boldsymbol{w_O^*} \in arg \min_{\boldsymbol{w_O}} J_D(\boldsymbol{w_D^*}, \boldsymbol{w_O}) \tag{5}$$

However, to reach a Nash Equilibrium, both players face a symmetric problem to find a policy in the set of best responses (BR) to the other player's current policy:

$$\pi_D^* \in BR_D(\pi_O^*) \tag{6}$$

$$\pi_O^* \in BR_O(\pi_D^*) \tag{7}$$

Note that Nash and Stackelberg Equilibrium policies (and policy weights) may not be unique. From here on, we use *best response* to denote any policy which belongs to the best response set and *optimal policy* (or weights) to denote any policy (or weights) belonging to the set of policies which optimizes the players' utilities.

Since, it is known that every Nash Equilibrium is also a Stackelberg Equilibrium for two-player zero-sum games (Fudenberg and Levine 1998), we propose a common algorithm to solve both types of games. We approach these problems by taking a gradient-based optimization approach. The gradient of J_D w.r.t. the defender parameters $\boldsymbol{w_D}$ can be found using the policy gradient theorem (section 2.3) as:

$$\boldsymbol{\nabla_{w_D} J_D} = \mathbb{E}_{s,a_D,a_O}[r_D \nabla_{w_D} \log \pi_D(a_D|s;\boldsymbol{w_D})] \quad (8)$$

The exact computation of the above integral is prohibitive, but it can be approximated from a batch of B on-policy samples (w.r.t. π_D) as pointed out in section 2.3. The gradient for the opponent objective w.r.t. $\boldsymbol{w_O}$ can be computed similarly. Ideally one can use even a single sample to get an unbiased estimate of the gradients, but such an estimate has a very high variance. Hence, we use a small batch of *i.i.d.* samples to compute the gradient estimate.

Lastly, we point out that gradient-based optimization only finds locally optimum points in the parameterized search space, so the term *optimal* from here on would refer to a local optimum of the objective functions under consideration, when optimized in a parameterized weight space.

5 OptGradFP: Optimization with Policy Gradients and Fictitious Play

We propose our algorithm OptGradFP to solve security game models. Our algorithm leverages the advances in policy gradient learning (Sutton et al. 1999) and those from game theoretic fictitious play (Heinrich, Lanctot, and Silver 2015; Heinrich and Silver 2016), to find the optimal defender parameters $\boldsymbol{w_D}$ which maximize her utility. Policy gradient theorem (Sutton et al. 1999) provides a way to make soft updates to current policy parameters to get new policies. Fictitious play involves best responding to the average of the other players' policies uptil now.

OptGradFP (algorithm 1) aims to approximate the Nash Equilibrium policies for the players. It maintains estimates of players' policies π_D, π_O and samples n_s actions from each policy in every episode. The game state, and the sampled actions (s, a_D, a_O) are stored in a replay memory. The replay memory stores samples from all past policies of the players and helps to emulate approximate fictitious play.

Every episode, the algorithm randomly samples a mini-batch of size n_b from the replay memory, containing actions of both players from all their policies uptil then. To train a player, it then plays games by resampling that player's actions for those samples from his/her current policy (while keeping the other player's actions the same), and improves the player's policy using the policy gradient update.

Note that the policy gradient update made this way is approximately a soft update towards the best response to the other player's *average* policy. We employ learning rate decay to take larger steps initially and obtain a finer convergence towards the end.

Also, playing all games with the player's current policy before the policy gradient step is required since policy gradients require on-policy sampling. If a game simulator, which allows playing games by restoring arbitrary previous states is not available, importance sampling can be a viable substitute for this step.

Finally observe that OptGradFP can learn to find the optimal policies for a single game state s, if the game simulator always gives out that state. However, it can also learn to generalize over multiple input states, if the same simulator gives it many different states s while sampling. Also, our algorithm is very generic in the sense that it does not require computing any best response functions specific to any game, but rather learns directly from samples.

Algorithm 1: OptGradFP

Data: Learning rates (α_D, α_O), decays (β_D, β_O), batch size (n_b), sample size (n_s), episodes (ep_{max})
Result: Parameters $\boldsymbol{w_D}$
Initialize policy parameters $\boldsymbol{w_D}$ and $\boldsymbol{w_O}$ randomly;
Create replay memory mem of size $E = ep_{max} \times n_s$;
for *ep in* $\{0, \ldots, ep_{max}\}$ **do**
  `/* Sample states and actions */`
  for n_s *times* **do**
    Obtain game state s;
    Get $a_D \sim \pi_D(\cdot|s;\boldsymbol{w_D}), a_O \sim \pi_O(\cdot|s;\boldsymbol{w_O})$;
    Store $\{s, a_D, a_O\}$ in `mem`;
  `/* Train Defender */`
  Draw n_b samples $\{s^i, a_D^i, a_O^i\}$ from `mem`;
  Play n_b games $s^i, \tilde{a}_D^i, a_O^i$ with $\tilde{a}_D^i \sim \pi_D(\cdot|s^i;\boldsymbol{w_D})$ to obtain rewards $\tilde{r}_D^i, \tilde{r}_O^i$;
  $\boldsymbol{\nabla_{w_D} J_D} = \frac{1}{n_b}\sum_{i=1}^{n_b} \tilde{r}_D^i \nabla_{\boldsymbol{w_D}} \log \pi_D(\tilde{a}_D^i|s^i;\boldsymbol{w_D})$;
  $\boldsymbol{w_D} := \boldsymbol{w_D} + \frac{\alpha_D}{1+ep\,\beta_D}\boldsymbol{\nabla_{w_D} J_D}$;
  `/* Train Opponent */`
  Draw n_b samples $\{s^i, a_D^i, a_O^i\}$ from `mem`;
  Play n_b games $s^i, a_D^i, \tilde{a}_O^i$ with $\tilde{a}_O^i \sim \pi_O(\cdot|s^i;\boldsymbol{w_O})$ to obtain rewards $\tilde{r}_D^i, \tilde{r}_O^i$;
  $\boldsymbol{\nabla_{w_O} J_O} = \frac{1}{n_b}\sum_{i=1}^{n_b} \tilde{r}_O^i \nabla_{\boldsymbol{w_O}} \log \pi_O(\tilde{a}_O^i|s^i;\boldsymbol{w_O})$;
  $\boldsymbol{w_O} := \boldsymbol{w_O} + \frac{\alpha_O}{1+ep\,\beta_O}\boldsymbol{\nabla_{w_O} J_O}$;

6 OptGradFP-NN: OptGradFP with Neural Networks

Since OptGradFP does not depend on policy representation, we can choose it freely according to domain so long as it is differentiable w.r.t. its parameterization. For RPS, we simply maintain the defender and opponent policies as 3×1 vectors i.e. $\pi_D = [\pi_{D1}, \pi_{D2}, \pi_{D3}], \pi_O = [\pi_{O1}, \pi_{O2}, \pi_{O3}]$. Since this is a stateless game, there is no distinction between policy and mixed strategy.

For the forest game, we assume each element of the defender's and opponent's actions (a_D, a_O) to be distributed

independently according to logit-normal distributions. Our choice of logit-normal distribution meets the requirement of a continuous distribution, differentiable w.r.t. its parameters and having bounded support (since our players' actions are bounded and continuous).

To represent them, we need to generate the means and standard deviations of the underlying normal distributions for each element of $a_D = (\boldsymbol{d}, \boldsymbol{\theta})$ and $a_O = (\boldsymbol{\rho}, \boldsymbol{\phi})$. While having a mean and variance would suffice to represent a mixed strategy, we are aiming to find policies that map input states represented by images to mixed strategies. Hence, we use convolutional neural networks (CNNs) to map the input images (states) to means and standard deviations for each player, owing to their recent success in image processing and computer vision applications (Krizhevsky, Sutskever, and Hinton 2012; Zeiler and Fergus 2014).

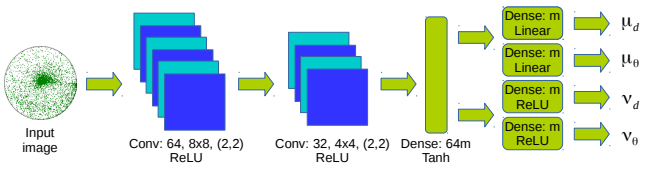


Figure 3: Defender's policy represented via a CNN

Defender policy representation: The defender neural network parameterized by weights $\boldsymbol{w_D}$ takes as input an image s of the forest tree locations and outputs means $(\boldsymbol{\mu_d}(s; \boldsymbol{w_D}) \in \mathbb{R}^m, \boldsymbol{\mu_\theta}(s; \boldsymbol{w_D}) \in \mathbb{R}^m)$ and standard deviations $(\boldsymbol{\nu_d}(s; \boldsymbol{w_D}) \in \mathbb{R}^m, \boldsymbol{\nu_\theta}(s; \boldsymbol{w_D}) \in \mathbb{R}^m)$ for two m-dimensional gaussians. For clarity we will skip writing $(s; \boldsymbol{w_D})$ with these parameters. Each defender action coordinate is then a logit-normal distribution and the probability of taking action $a_D = (\boldsymbol{d}, \boldsymbol{\theta})$ is given by:

$$\pi_D(\boldsymbol{d}, \boldsymbol{\theta}|s) = \prod_{i \in [m]} p_{ln}(d_i; \mu_{d,i}, \nu_{d,i}) p_{ln}\left(\frac{\theta_i}{2\pi}; \mu_{\theta,i}, \nu_{\theta,i}\right) \quad (9)$$

where p_{ln} is the logit-normal distribution and the product is over all m elements of the vector. The defender's policy network is shown in Figure 3.

Opponent policy representation: The opponent neural network is similarly parameterized by weights $\boldsymbol{w_O}$ outputs means $(\boldsymbol{\mu_\rho} \in \mathbb{R}^n, \boldsymbol{\mu_\phi} \in \mathbb{R}^n)$ and standard deviations $(\boldsymbol{\nu_\rho} \in \mathbb{R}^n, \boldsymbol{\nu_\phi} \in \mathbb{R}^n)$ for two n-dimensional gaussians. The probability of action $a_O = (\boldsymbol{\rho}, \boldsymbol{\phi})$ is similar to equation (9).

The network architectures for both players are provided in the appendix. Finally, though all elements of a_D (resp. a_O) are from independent logit-normal distributions, the means and standard deviations for the underlying normal distributions are computed jointly via the CNNs, and allow the players to plan coordinated moves for their resources.

7 Experiments and Results

We now present experiments against several baselines. Cournot Adjustment (CA), one of the early techniques used to optimize players' policies, makes the defender and the opponent respond to each other's policy with their best responses. This method can converge to the Nash Equilibrium for certain classes of games (Fudenberg and Levine 1998). Another method called StackGrad was recently proposed (Amin, Singh, and Wellman 2016). It uses a best response computation for the opponent's updates, and a policy gradient update similar to ours for the defender (but no fictitious play). We also augmented StackGrad with fictitious play (using replay memory), and call it StackGradFP. We compare our results against CA, StackGrad and StackGradFP in our experiments. For more details of the baselines and hyperparameters of all algorithms, refer to the appendix.

For forest game, we present results for $m = 8$ guards and $n = 8$ lumberjacks where the numbers provide appropriate forest coverage (since fewer guards leave too much open space). We set the ambush penalty $r_{pen} = 10$, guard radius $R_g = 0.1$ and lumberjack radius $R_l = 0.04 < R_g$ (since guards can scout lumberjacks from long distances).

7.1 Rock-Paper-Scissors Results

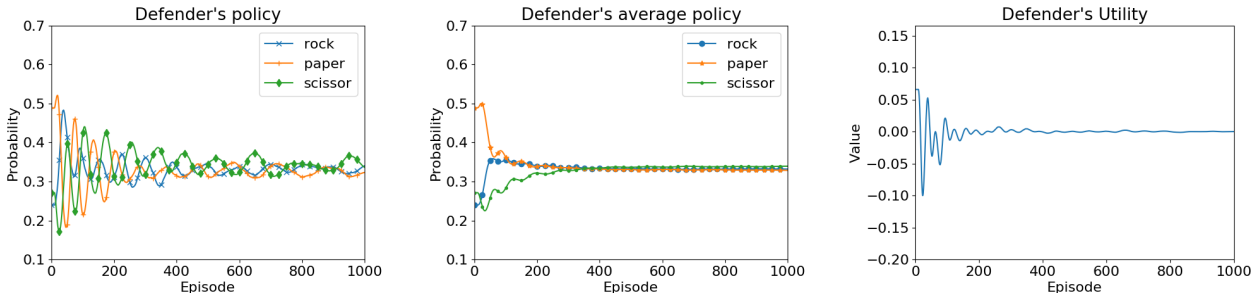
Figure 4 shows the defender's statistics as a function of the number of episodes, when OptGradFP is applied. Note from figure 4a, that the final policy of defender comes close to $\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ and converges slowly while oscillating around it. The oscillations are because of minibatch sampling from the replay memory and become smaller with larger batch sizes. A faster convergence is achieved by the average policy of defender (figure 4b) and we recommend computing the average policies if feasible. Note that average policies are easily computable in small settings like RPS, but in continuous domains like the forest game, there is no clear way of computing average policies and hence we will stick to the parameterized policy in such cases. The defender's utility also converges to the Nash Equilibrium value = 0 as shown in figure 4c. These results demonstrate the convergence of OptGradFP. Results on RPS with other baselines have been shown in the appendix.

7.2 Forest Security Game Results

Learned policy on a single state: We show a visualization of the players' final mixed strategies in figure 5, when trained only on one randomly chosen forest state. The visualizations were generated by sampling 1000 locations for each guard (blue dots) and each lumberjack (red dots) from each algorithm's final strategies. Note that training strategies on a single forest state does not require a neural network, since we only need to learn specific values of $\boldsymbol{\mu_d}, \boldsymbol{\mu_\theta}, \boldsymbol{\nu_d}, \boldsymbol{\nu_\theta}$ as opposed to a mapping for every state s.

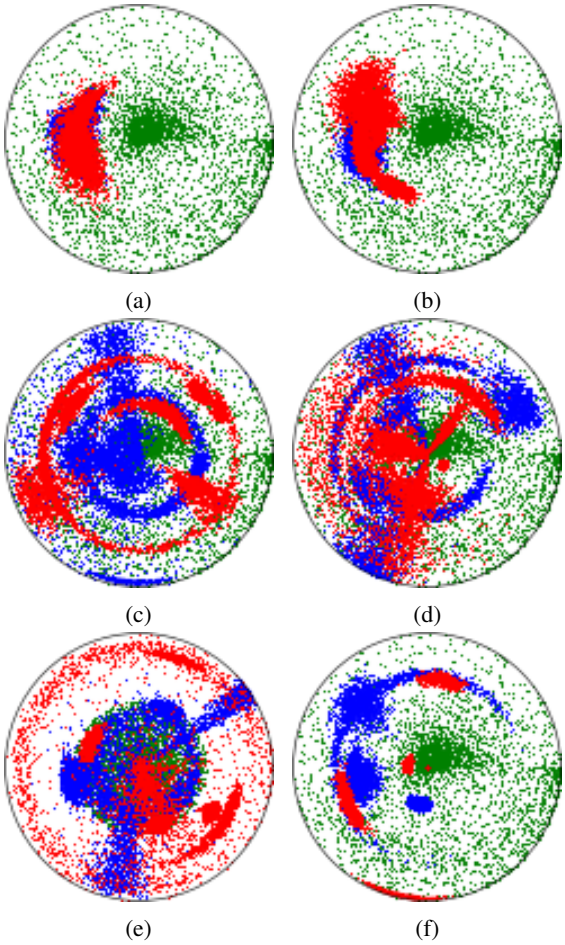
Clearly CA and StackGrad lead to highly concentrated strategies for the defender and the opponent (figures 5a, 5b). In fact, they do not generally converge and keep oscillating. However, OptGradFP and StackGradFP (figures 5d, 5c) converge well and give well-spread out strategies that provide appropriate coverage of the forest for both players.

Note that both OptGradFP and StackGradFP contain a few guards forming circular-band shaped densities centered around the origin, which generally provide reasonable protection for the forest's dense center. CA and StackGrad find



(a) (b) (c)

Figure 4: (a) Defender's policy, (b) Defender's average policy, (c) Defender's utility



(a) (b) (c) (d) (e) (f)

Figure 5: Visualization of players' policies. The blue and red dots show sampled positions for guards and lumberjacks respectively: (a) CA, (b) StackGrad, (c) StackGradFP, (d) OptGradFP, (e) OptGradFP on a forest with a central core, and (f) OptGrad.

local regions to guard, and leave enough space for the lumberjacks to chop wood without getting ambushed. Note that placing the bands close to the forest center would leave a huge area to be chopped by the lumberjacks. Also, placing the guards at the boundary would distribute them sparsely and lumberjacks would be able to come and go unambushed. OptGradFP and StackGradFP find reasonable middle ground by inferring good radii to place the guards.

We also show the mixed strategy found by OptGradFP for a forest containing a symmetric central core of trees similar to (Johnson, Fang, and Tambe 2012). It covers the core with 6 out of 8 guards forming a dense ring, after which the remaining guards take blob-shaped densities since their presence/absence does not matter (local minima). This is similar to the circular bands proposed as the optimal patrol strategy for a uniform tree density.

Opponent's best response utility: Another performance indicator is the utility achieved by opponent's final best response strategy after the defender fixes her strategy (see appendix for opponent's best response computation). Table 1 gives the opponent's best response utility (OBU) for a single forest state. OptGradFP and StackGradFP provide much better utility than alternatives. CA and StackGrad do not converge and hence their utility values keep fluctuating but are in general much higher than those of OptGradFP and StackGradFP. CA and StackGrad have a high opponent's best response utility which is in agreement with our observation that they find local regions to guard, and leave the lumberjacks lots of space to chop wood without getting ambushed.

Algorithm	OBU
CA	661.0 ± 92.7
StackGrad	596.0 ± 74.3
StackGradFP	399.4 ± 8.5
OptGradFP	398.2 ± 5.2

Table 1: Opponent's best response utility ($\pm$ std. error of mean).

Replay memory: To emphasize the crucial role of fictitious play, we removed fictitious play from OptGradFP (we call this OptGrad). This means using a small replay memory ($E = n_s = n_b$), containing games sampled only from players' current strategies. On a single state, the utility achieved by opponent's best response strategy was 481.14, which is

slightly better than CA and StackGrad, but worse than OptGradFP and StackGradFP. The resulting strategies (figure 5f) are not well spread-out anymore, since the method does not have history of previous steps (similar to StackGrad).

In general, having a replay memory and large batch size ($n_b \gg n_s$) gave us smoother convergence properties by better approximating the best response to the other player's *average* strategy. However, having a large sample size requires playing more games and becomes a bottleneck for every training step. The trade-off between good approximation to fictitious play vs. computation time requires careful balancing to achieve fast convergence.

Computation time: The time for computing the defender's mixed strategy on a single forest state using Algorithm 1 is shown in Table 2. Clearly OptGradFP is slower than OptGrad, CA and StackGrad (because they lack game replays). However, it is faster than StackGradFP since it does not require best response computation, unlike StackGradFP. OptGrad is faster than CA and StackGrad for the same reason. In the example domain of forest protection as well as other security games, computing best responses (even approximately) is quite complex, often domain-dependent and computationally expensive. Replacing it with a policy gradient step provides significant speedup.

Algorithm	Computation time $\pm$ Std. dev. (in secs)
CA	8263.2 ± 76.4
StackGrad	5338.3 ± 120.1
OptGrad	3522.9 ± 98.3
StackGradFP	18426.5 ± 190.8
OptGradFP	12257.6 ± 187.2

Table 2: Computation time for all algorithms (in seconds).

Training on multiple forest states: Finally, we show that OptGradFP can learn to predict good defender strategies on unseen forest states, once trained on multiple forest states. For this we trained the CNN policies (section 6) using OptGradFP on 1000 randomly generated forest states. Then we tested the learnt defender policies on 10 new forest states which were not present in the training set. A good defender strategy for each of the 10 test states was also computed independently using OptGradFP (without the CNN) using Algorithm 1 to compare against the strategies predicted by the learnt CNN policy.

The opponent's best response utility (OBU) on each test forest state is given in Table 3. We observe slightly higher opponent utilities for predicted strategies than the ones directly computed, but the predicted strategies are fairly competitive given that the neural network never saw those states in training. Further it also predicts strategies very similar to those found independently for each forest state (see appendix, Figure 6). This shows that in practice our algorithm can train neural networks to learn about the structure of the problem domain and predict defender strategies with low opponent utilities on unseen states.

Lastly, though independent training on each state requires about $\approx$ 12250 seconds (Table 2) and jointly training on 1000 states took about 7 days (i.e. 170.1 hours), the prediction time on a new state (after training) is only about 90 ms on average, thereby shifting the computation of strategies from online to mostly offline.

State	OBU (predicted)	OBU (computed)
0	414.4 ± 7.7	375.6 ± 7.5
1	179.0 ± 3.8	126.5 ± 3.9
2	394.1 ± 7.8	383.9 ± 8.0
3	283.2 ± 6.6	224.9 ± 5.6
4	263.0 ± 5.4	241.8 ± 5.2
5	400.0 ± 8.2	297.5 ± 6.7
6	317.7 ± 6.9	232.3 ± 5.0
7	340.9 ± 7.4	278.0 ± 5.8
8	264.0 ± 5.2	190.7 ± 4.2
9	462.0 ± 9.6	451.5 ± 9.9

Table 3: Opponent's best response utilities $\pm$ std. error of mean for predicted strategies and independently computed strategies.

8 Discussion

Why not discretize? Some previous works (Yang et al. 2014; Haskell et al. 2014; Gan et al. 2017; Xu et al. 2014) discretize the state and action spaces to find equilibrium strategies, but the attacker in particular, may not attack only at discretized locations, which invalidates discretized solutions in real settings.

Further, the computation after discretization can still be intractable (esp. with multiple player resources). For instance, even a coarse discretization of the forest game for 8 guards and 8 lumberjacks with angular grid size = 10 degree (36 bins) and radial grid size = 0.1 (10 bins), gives an intractable number of pure strategies ($(36 \times 10)^8 \approx 2.82 \times 10^{20}$) for just the defender on a single forest state. While column generation and double oracle based approaches can somewhat improve computation efficiency, the memory and runtime requirement still remains high (Xu et al. 2014).

Additionally, with discretization, the computation cost would be paid independently for each individual game state. In contrast, using our approach, the computation cost for a new game instance after the neural network is trained, is much lower than using a discretization-based approach.

Comparing all algorithms Since StackGrad plays aggressive best responses for the opponent, the lumberjacks keep jumping to far-off locations. The defender's policy gradient (PG) is a soft step and never catches up to the lumberjacks. OptGrad updates both players with a soft PG step and hence outperforms StackGrad, but without replay memory, neither of them converges.

After adding replay memory, both OptGradFP and StackGradFP make the players respond to each other's average strategies. Even when the opponent changes its strategy aggressively (in StackGradFP), responding to the average of its strategies helps the defender converge. Both have similar performance, however OptGradFP dominates because of its lower computation time.

Limitations Gradient-based approaches rely on availability of non-zero gradients throughout the state-action spaces for both players, which may not always apply. In such cases, the algorithm can sometimes stagnate prematurely if the gradient of the utility w.r.t. the policy parameters becomes zero. Hence, gradient-based approaches sometimes require careful initialization to compute good mixed strategies for a given state.

9 Conclusion

We have presented a neural network based approach to address security games with continuous state and action spaces. Our novel algorithm OptGradFP represents policies by parameterizing in continuous space and learns the parameters using fictitious play and policy gradients. Our approach is generic and can train the defender's policy over multiple distinct game states. This allows learning a generalized model for the defender's policy offline and predict good defender strategies on previously unseen game states.

Acknowledgments

This research was supported in part by NSF Research Grant IIS-1254206, MURI grant W911NF-11-1-0332, Harvard Center for Research on Computation and Society fellowship and USC Viterbi Graduate PhD fellowship.

References

Amin, K.; Singh, S.; and Wellman, M. P. 2016. Gradient methods for stackelberg security games. In *UAI*, 2–11.

Basilico, N.; Celli, A.; De Nittis, G.; and Gatti, N. 2017. Coordinating multiple defensive resources in patrolling games with alarm systems. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 678–686.

Cermák, J.; Bošanský, B.; Durkota, K.; Lisý, V.; and Kiekintveld, C. 2016. Using correlated strategies for computing stackelberg equilibria in extensive-form games. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, 439–445.

Conitzer, V., and Sandholm, T. 2006. Computing the Optimal Strategy to Commit to. In *Proc. of the ACM Conference on Electronic Commerce (ACM-EC)*, 82–90.

Fang, F.; Jiang, A. X.; and Tambe, M. 2013. Optimal patrol strategy for protecting moving targets with multiple mobile resources. In *AAMAS*, 957–964.

Fudenberg, D., and Levine, D. K. 1998. *The theory of learning in games*, volume 2. MIT press.

Gan, J.; An, B.; Vorobeychik, Y.; and Gauch, B. 2017. Security games on a plane. In *AAAI*, 530–536.

Haskell, W.; Kar, D.; Fang, F.; Tambe, M.; Cheung, S.; and Denicola, E. 2014. Robust protection of fisheries with compass. In *IAAI*.

Heinrich, J., and Silver, D. 2016. Deep reinforcement learning from self-play in imperfect-information games. *CoRR* abs/1603.01121.

Heinrich, J.; Lanctot, M.; and Silver, D. 2015. Fictitious self-play in extensive-form games. In *ICML*, 805–813.

Johnson, M. P.; Fang, F.; and Tambe, M. 2012. Patrol strategies to maximize pristine forest area. In *AAAI*.

Kar, D.; Fang, F.; Fave, F. D.; Sintov, N.; and Tambe, M. 2015. "a game of thrones": When human behavior models compete in repeated stackelberg security games. In *AAMAS*.

Kiekintveld, C.; Jain, M.; Tsai, J.; Pita, J.; Ordóñez, F.; and Tambe, M. 2009. Computing optimal randomized resource allocations for massive security games. In *AAMAS*, 689–696.

Korzhyk, D.; Yin, Z.; Kiekintveld, C.; Conitzer, V.; and Tambe, M. 2011. Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *JAIR* 41:297–327.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In Pereira, F.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *NIPS*. 1097–1105.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Rosenfeld, A., and Kraus, S. 2017. When security games hit traffic: Optimal traffic enforcement under one sided uncertainty. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 3814–3822.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; Mansour, Y.; et al. 1999. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, 1057–1063.

Tambe, M. 2011. *Security and Game Theory: Algorithms, Deployed Systems, Lessons Learned*. New York, NY: Cambridge University Press.

Wang, B.; Zhang, Y.; and Zhong, S. 2017. On repeated stackelberg security game with the cooperative human behavior model for wildlife protection. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, 1751–1753.

Xu, H.; Fang, F.; Jiang, A. X.; Conitzer, V.; Dughmi, S.; and Tambe, M. 2014. Solving zero-sum security games in discretized spatio-temporal domains. In *AAAI*, 1500–1506.

Yang, R.; Ford, B.; Tambe, M.; and Lemieux, A. 2014. Adaptive resource allocation for wildlife protection against illegal poachers. In *AAMAS*.

Yin, Y.; An, B.; and Jain, M. 2014. Game-theoretic resource allocation for protecting large public events. In *AAAI*, 826–833.

Zeiler, M. D., and Fergus, R. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*, 818–833.

10 Appendix

10.1 Baseline Implementations

Note that the actual specification of CA and StackGrad cannot directly work in the same domain as OptGradFP. To overcome this situation, we implemented CA, StackGrad and StackGradFP in a way similar to OptGradFP. All the baselines maintain a parameterized strategy representation for both players (π_D and π_O). Each algorithm samples n_s actions for both players in every episode and store them in a replay memory. Since CA and StackGrad lack fictitious play, their replay memory is small and can only contain actions sampled from the current strategy. OptGradFP and StackGradFP both maintain long replay memories containing all previous strategy samples.

For soft policy updates, we use policy gradient updates (like in OptGradFP) on n_b-size batches drawn from the replay memory. However, to emulate best responses we do not actually compute best responses since that would make the implementation specific to the domain. Instead, we generate new randomly initialized neural network strategies and train them multiple times with the soft gradient step on n_b-size batches of the other player's actions drawn from the replay memory. This approximately replicates a best response. If a generic implementation is not required, this step can also be replaced by game-specific best-response functions.

Brief descriptions of update rules for all baselines follow: **CA**: Makes the defender and the opponent best respond to each other's strategy. **StackGrad**: Uses best response update for the opponent, and policy gradient update similar to ours for the defender (but no fictitious play). **StackGradFP**: Same as StackGrad, except it uses a policy gradient update with fictitious play for the defender (i.e. via a replay memory like in OptGradFP).

10.2 Opponent's Best Response Utility

The opponent's final best response utility for the forest game was computed approximately (computing the actual value is extremely prohibitive), by sampling k random opponent actions and k actions from the defender's final strategy. k^2 games were played with each combination of the defender's and opponent's actions and the opponent action which led to the maximum reward (averaged over all k defender actions) was used to compute the opponent's final utility. We use $k = 25$ for all algorithms. Due to such evaluation, the opponent's final action can be very different from that obtained using π_O, and it allows us to test our learnt defender strategy without restraining the opponent's final strategy shape to logit-normal distribution thereby giving a more robust estimate of performance.

10.3 Hyperparameters

OptGradFP for Rock-Paper-Scissors uses maximum episodes $ep_{max} = 1000$, sample size $n_s = 50$, batch size $n_b = 500$, learning rates $\alpha_D = \alpha_O = 0.1$, and decays $\beta_D = \beta_O = \frac{9}{ep_{max}}$. The baselines' hyperparameters for Rock-Paper-Scissors are the same as for OptGradFP (except for E which is equal to n_s for CA and StackGrad). The forest game's hyperparameters for the single forest state case are summarized in Table 4. OptGradFP-NN for multiple forest states uses the same parameters except $ep_{max} = 20000$ and $E = 500000$. The architectures of all neural networks presented earlier and all algorithm hyperparameters were chosen by doing informal grid searches within appropriate intervals.

	CA	StackGrad	StackGradFP	OptGradFP
ep_{max}	400	400	400	400
n_s	50	50	25	25
n_b	50	50	250	250
E	50	50	10000	10000
$\alpha_{\{D,O\}}$	$5e-6$	$5e-6$	$1e-5$	$5e-4$
$\beta_{\{D,O\}}$	$\frac{9}{ep_{max}}$	$\frac{9}{ep_{max}}$	$\frac{9}{ep_{max}}$	$\frac{9}{ep_{max}}$

Table 4: Hyperparameters

10.4 Neural Network Architectures

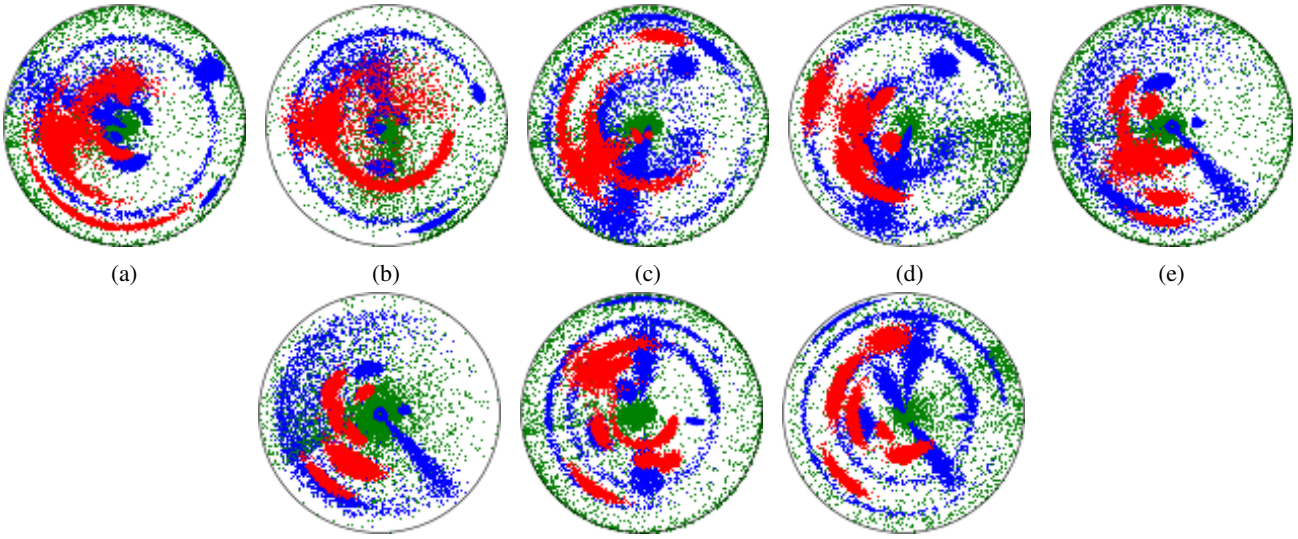
The defender neural network takes an image of size 120×120 as input. First hidden layer is a convolutional layer with 64 filters of size 8×8 and strides 2×2. The second hidden layer is convolutional with 32 filters of size 4×4 and strides 2×2. Both convolutional layers have `relu` activations and no pooling. Next layer is a fully-connected dense layer with $64m$ units (where m = number of guards) and `tanh` activation. Lastly we have four parallel fully-connected dense output layers one each for $\boldsymbol{\mu_d}, \boldsymbol{\nu_d}, \boldsymbol{\mu_\theta}$ and $\boldsymbol{\nu_\theta}$. These four layers have m units each, with the layers for means having `linear` activations and those for standard deviations having `relu` activations. We add a fixed small bias of 0.1 to the outputs of the standard deviation layers to avoid highly concentrated or close to singular distributions. We also clip all gradients to stay in the range $[-0.5, 0.5]$ to avoid large weight updates and potential divergence (Mnih et al. 2015). The opponent neural network is also similar to the defender network, except that the fully-connected hidden layer has $64n$ units (where n = number of lumberjacks) and the four output layers for $\boldsymbol{\mu_\rho}, \boldsymbol{\nu_\rho}, \boldsymbol{\mu_\phi}$ and $\boldsymbol{\nu_\phi}$ have n units each.

10.5 Baseline Results on Rock-Paper-Scissors

Figures 7 and 8 show results for CA, StackGrad and StackGradFP on the Rock-Paper-Scissors game. Note that CA and StackGrad do not use fictitious play and hence mostly keep oscillating, whereas StackGradFP converges to the Nash Equilibrium (both final policy and average policy). We use $n_s = 50$ and $n_b = 500$ for all baselines.

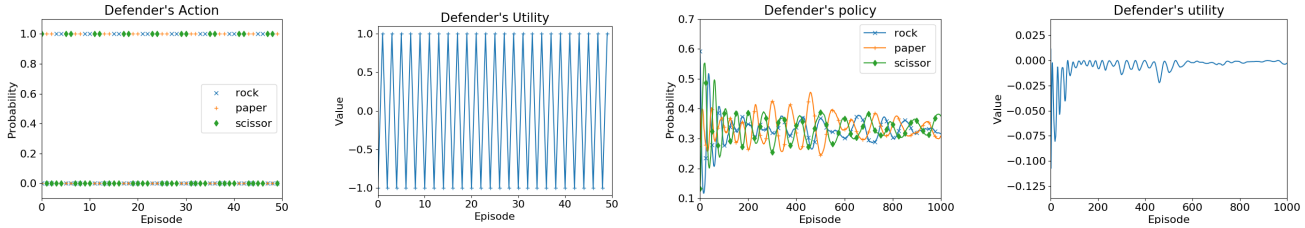
10.6 Policy Visualization for Multiple Forest States

In section 7.2, we trained a CNN on 1000 randomly generated forest states. The predicted strategies and the independently generated strategies for 4 randomly chosen test states are visualized in Figure 6.



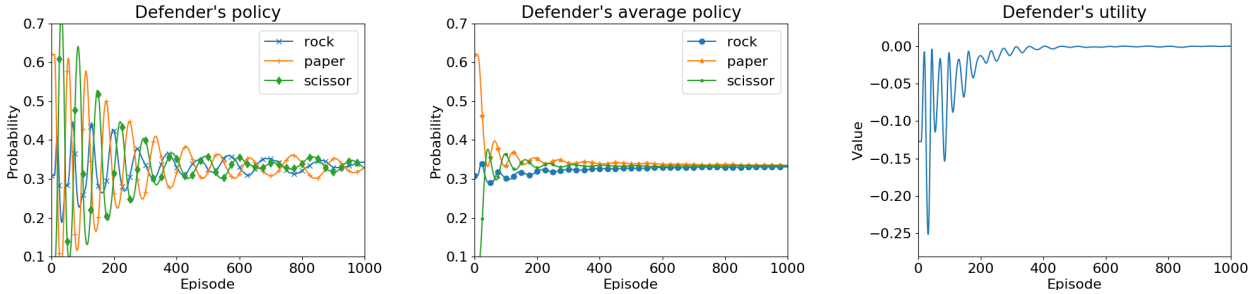
(a) (b) (c) (d) (e) (f) (g) (h)

Figure 6: Visualization of players' strategies on randomly chosen test states (defender: blue, opponent: red): (a) Predicted: 1, (b) Computed: 1, (c) Predicted: 7, (d) Computed: 7, (e) Predicted: 8, (f) Computed: 8, (g) Predicted: 9, and (h) Computed: 9.



(a) (b) (c) (d)

Figure 7: Results of CA and StackGrad on Rock-Paper-Scissors: (a) Defender's actions with CA on RPS, (b) Defender's utility with CA on RPS, (c) Defender's policy with StackGrad on RPS, (d) Defender's utility with StackGrad on RPS.



(a) (b) (c)

Figure 8: Results of StackGradFP on Rock-Paper-Scissors: (a) Defender's policy at each episode, (b) Defender's average policy at each episode, and (c) Defender's utility at each episode.