Handling Continuous Space Security Games with Neural Networks

Nitin Kamra^{1*}, Fei Fang^{2†}, Debarun Kar^{1}, Yan Liu^{1}, Milind Tambe^{1}
University of Southern California^{1}, Harvard University^{2}
{nkamra, dkar, yanliu.cs, tambe}@usc.edu^{1}, fangf07@seas.harvard.edu^{2}

Abstract

Despite significant research in Security Games, limited efforts have been made to handle game domains with continuous space. Addressing such limitations, in this paper we propose: (i) a continuous space security game model that considers infinite-size action spaces for players; (ii) OptGradFP, a novel and general algorithm that searches for the optimal defender strategy in a parametrized search space; (iii) OptGradFP-NN, a convolutional neural network based implementation of OptGradFP for continuous space security games; (iv) experiments and analysis with OptGradFP-NN. This is the first time that neural networks have been used for security games, and it shows the promise of applying deep learning to complex security games which previous approaches fail to handle.

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]. In SSGs, the defender (referred to as “she”) perpetually defends a set of targets with a limited number of resources, and the adversary (referred to as “he”) is able to surveil and learn the defender's strategy and then plan an attack based on this information. Exact and approximate approaches have been proposed to find the optimal defender strategy in SSGs, which maximizes her expected utility given that the attacker will best respond to the defender's strategy [Kiekintveld *et al.*, 2009; Amin *et al.*, 2016].

Recently, there has been an increasing interest in SSGs for green security domains such as protecting wildlife [Yang *et al.*, 2014; Kar *et al.*, 2015; Fang *et al.*, 2015], fisheries [Haskell *et al.*, 2014] and forests [Johnson *et al.*, 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). Most previous works discretize the area into grid cells and restrict the players' actions to discrete sets [Yang *et al.*, 2014; Haskell *et al.*, 2014]. 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, especially when there are multiple defender resources. While [Johnson *et al.*, 2012] addresses continuous space in security games in the domain of forest protection, it focuses on a special case: the objective of the defender is to compute a strategy to minimize the trespassing distance of the adversaries who are located at the boundary of the forest area and therefore the problem can be reduced to a one-dimensional problem.
Our major contributions are as follows:

- a continuous space security game model which considers infinite action spaces over two-dimensional continuous areas with asymmetric target distribution.
- OptGradFP, a novel and general algorithm which leverages recent advances in policy learning and game theoretic fictitious play to optimize parametrized policies in continuous spaces.
- OptGradFP-NN, an application of OptGradFP using convolutional neural networks (CNNs) to represent the players' mixed strategies.

Finally, we conduct experiments and analysis with OptGradFP-NN and demonstrate the superiority of our approach against comparable approaches such as StackGrad [Amin *et al.*, 2016] and Cournot Adjustment (CA) [Fudenberg and Levine, 1998]. Our analysis shows that standard approaches either (i) take very aggressive steps for both players and never settle to a good strategy (CA), or (ii) makes aggressive best response updates for the opponent and only soft steps for the defender, thus leading to poor average defender rewards (StackGrad). In contrast, our approach effectively allows both players to take soft steps and takes past strategies into account through a replay memory, thereby eventually converging to a good average response for both players. We show for the first time, the promise of applying deep learning to complex continuous domain security games which previous approaches fail to handle.

[*]This work was partially supported by USC Viterbi Graduate PhD fellowship.

[†]This work was partially supported by Harvard Center for Research on Computation and Society fellowship.

2 Preliminaries and Related Work

We use small letters (x) to denote scalars, bold small letters ($\boldsymbol{x}$) to denote vectors, capitals (X) to denote matrices and bold capitals ($\boldsymbol{X}$) to denote random vectors. $\mathbb{R}$ represents the set of real numbers. $\mathbf{0}_n$ and $\mathbf{1}_n$ are vectors of size n of zeros and ones respectively (we will sometimes skip n if the size is evident from context). I_n is the identity matrix of size $n \times n$. 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 $\boldsymbol{b}$. The notation $[n]$ is the set of natural numbers uptil n i.e. $\{1, 2, \ldots, n\}$. $\mathcal{N}(\mu, \nu^2)$ 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: $\text{logit}(\boldsymbol{x}) \triangleq \log \frac{\boldsymbol{x}}{\mathbf{1}-\boldsymbol{x}}\ \forall \boldsymbol{x} \in [\mathbf{0}, \mathbf{1}]$. 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). 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. We use the term mixed strategy and policy interchangeably in the rest of the paper.

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

An attacker best responds to a defender strategy if he chooses a strategy that maximizes his expected utility. The optimal defender strategy in SSGs is the (mixed) strategy 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 strategy is the same as the strategy for the defender in any Nash equilibrium (NE). In most previous work on SSGs, discrete actions for the players are considered, even if the game setting is over continuous space [Amin *et al.*, 2016; Gan *et al.*, 2017]. There are a few exceptions [Johnson *et al.*, 2012; Fang *et al.*, 2013; Yin *et al.*, 2014], but the solution techniques often rely on exploitable spatio-temporal structures of the problem and cannot be generalized to handle continuous spaces as has been handled in this paper.

2.2 Fictitious play in normal form games

Fictitious play (FP) is a learning rule where each player plays best response 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 $\boldsymbol{X} \sim p(\boldsymbol{x}|\boldsymbol{\theta})$, the gradient of the expected value of $f(\cdot)$ with respect to a policy parameters 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 $\boldsymbol{x}_i \sim p(\boldsymbol{X}|\boldsymbol{\theta})$, and computing $\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X})] \approx f(\boldsymbol{x}_i) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}_i|\boldsymbol{\theta})$. 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 vector 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 parametrize 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 Continuous Space Security Game

We begin by describing the forest protection game model. Though we consider it as an example domain, the game model is general and can also represent other domains such as wildlife and fishery protection.

Game model: We assume a circular forest with radius 1.0 (i.e. all lengths are normalized with respect to the forest radius), with a prespecified but arbitrary tree distribution. All locations are represented in cylindrical coordinates with the forest center as origin. We assume n lumberjacks who can collaborate to plan their wood chopping locations. We will use the word adversary or opponent interchangeably to refer to the lumberjacks as a group. The defender has m forest guards, which can be allocated to various forest locations to ambush the trespassing lumberjacks.

State representation: One way of specifying the game state (S) is via locations of all trees. This leads to a variable state-size, dependent on the number of trees in the forest. Since a variable length representation is hard to process and we are mostly concerned with the relative density of trees over the forest, we instead summarize the forest state S as a 120×120 matrix containing a grayscale image of the forest. This makes the input representation for the defender and opponent policies invariant to the number of trees in the forest and also allows our approach to be used for planning strategies by using satellite images of a forest as input. An example input in RGB is shown in figure 1a (input to the players is a grayscale version).

Defender action: The defender picks m locations, one for each guard to remain hidden, and ambush lumberjacks

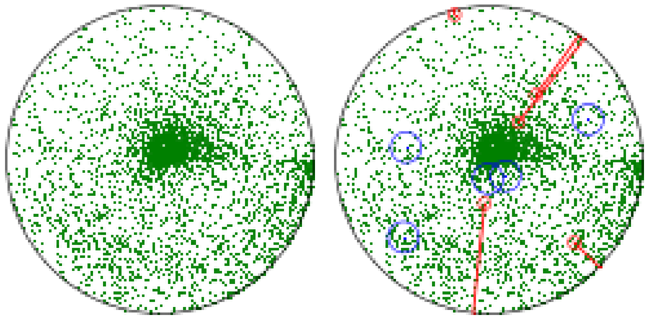
(a) (b)

Figure 1: (a) State representation of the forest as a 120×120 image (converted to grayscale for players), 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).

with wood. The defender's action a_D is a set of m distances $\boldsymbol{d} \in [0,1]^m$ and angles $\boldsymbol{\theta} \in [0, 2\pi]^m$ i.e. $a_D \in \mathbb{R}^{m\times 2}$. The cylindrical coordinates $(\boldsymbol{d}, \boldsymbol{\theta})$ specify the guards' positions in the forest.

Adversary action: Following [Johnson *et al.*, 2012], we assume that lumberjacks trespass the forest boundary and move straight towards the forest center. They can stop at any point along this straight line, cut wood in a radius R_l around the stopping point and come back to their starting location. The model is justified because lumberjacks generally wish to avoid one another as much as possible [Johnson *et al.*, 2012]. Since the lumberjack trajectories are fully specified by their stopping coordinates, the adversary's action is to decide all stopping points. The opponent's action a_O is a set of n distances $\boldsymbol{\rho} \in [0,1]^n$ and angles $\boldsymbol{\phi} \in [0, 2\pi]^n$ i.e. $a_O \in \mathbb{R}^{n\times 2}$. The cylindrical coordinates $(\boldsymbol{\rho}, \boldsymbol{\phi})$ define the wood chopping locations of all lumberjacks.

Rewards: A lumberjack is considered ambushed if his path comes within R_g distance from any guard's location. If a lumberjack gets ambushed, he fails in cutting the trees and gets a penalty $-r_{pen}$. The total utility for the opponent ($r_O \in \mathbb{R}$) equals to the total number of trees cut by the lumberjacks. The total utility for the defender is $r_D = -r_O$.

Game play: Given prespecified tree locations, a single run of the game proceeds as follows: (1) The defender gives m guard locations and the adversary gives n wood chopping locations, (2) The game simulator computes and returns rewards for the defender and opponent. By playing the game multiple times, the defender gets rewards and uses this information to optimize her strategy. A full game has been visualized in figure 1b.

4 Policies and Utilities

Policies: A player's policy (or mixed strategy) is a probability distribution over the player's actions given the game state (S). The defender maintains a learnable policy π_D parametrized by weights $\boldsymbol{w_D}$, from which she can sample the guards' positions. She also maintains an estimate of the adversary's policy π_O parametrized by $\boldsymbol{w_O}$, which helps her learn her own optimal policy. Note that the opponent's real policy is the best response to the defender's deployed policy (not the same as π_O). We use the symbols π_D, π_O to denote the policies, 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).

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

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

Note that the integral over S can be removed if we only require strategies over a specific state (forest), but our method allows learning policies over multiple states if needed.

Both the defender and the opponent want to maximize their utility functions. Yet, their approaches differ since the defender has to deploy her policy first, without knowing the opponent's policy. The opponent gets to observe the defender's policy and he can use this information to react with a best response to the defender's deployed policy. Hence the problem faced by the defender is essentially as follows:

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

The opponent's problem is simpler:

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

We approach these problems by taking a gradient optimization based approach. The gradient of J_D w.r.t. the defender parameters $\boldsymbol{w_D}$ can be found using the policy gradient theorem (see section 2.3) as:

$$\boldsymbol{\nabla_{w_D} J_D} = \mathbb{E}_{S,a_D,a_O}[\nabla_{w_D}\pi_D(a_D|S;\boldsymbol{w_D})\, r_D] \tag{6}$$

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) using the following unbiased estimator:

$$\boldsymbol{\nabla_{w_D} J_D} \approx \frac{1}{B}\sum_{i=1}^{B} \nabla_{w_D}\pi_D(a_D^i|S^i;\boldsymbol{w_D})\, r_D^i \tag{7}$$

The gradient for the opponent objective w.r.t. $\boldsymbol{w_O}$ can be similarly estimated as:

$$\boldsymbol{\nabla_{w_O} J_O} \approx \frac{1}{B}\sum_{i=1}^{B} \nabla_{w_O}\pi_O(a_O^i|S^i;\boldsymbol{w_O})\, r_O^i \tag{8}$$

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.

5 OptGradFP: Optimization with policy gradients and fictitious play

We propose our algorithm OptGradFP to solve security game models. Our algorithm leverages the recent advances in

Algorithm 1: OptGradFP

```
Data: Learning rates $(\alpha_D, \alpha_O)$, decays $(\beta_D, \beta_O)$,
      training rates $(f_D, f_O)$
Result: Parameters $\boldsymbol{w_D}$
Initialize policy parameters $\boldsymbol{w_D}$ and $\boldsymbol{w_O}$ randomly;
Fill replay memories $memD$, $memO$ of size $E$ with
  randomly played games;
for $ep$ in $\{0, \dots, ep_{max}\}$ do
    Get game state $S$;
    Sample $a_D = (d, \theta) \sim \pi_D(\cdot|S; \boldsymbol{w_D})m, a_O =$
      $(\rho, \phi) \sim \pi_O(\cdot|S; \boldsymbol{w_O})$;
    Execute actions $(a_D, a_O)$ and get rewards $(r_D, r_O)$;
    Store sample $\{S, a_D, a_O, r_D, r_O\}$ in memD, memO;
    if $ep \,\%\, f_D == 0$ then
        Get samples $\{S^i, a_D^i, a_O^i, r_D^i, r_O^i\}_{i \in [E]}$ from
          memD;
        Replay all $E$ games $S^i, \tilde{a}_D^i, a_O^i$ with
          $\tilde{a}_D^i \sim \pi_D(\cdot|S; \boldsymbol{w_D})$ to obtain rewards $\tilde{r}_D^i, \tilde{r}_O^i$;
        $\boldsymbol{\nabla_{w_D} J_D} = \frac{1}{E} \sum_{i=1}^{E} \nabla_{\boldsymbol{w_D}} \pi_D(\tilde{a}_D^i | S^i; \boldsymbol{w_D}) \, \tilde{r}_D^i$;
        $\boldsymbol{w_D} := \boldsymbol{w_D} + \frac{\alpha_D}{1 + ep\, \beta_D} \boldsymbol{\nabla_{w_D} J_D}$;
    if $ep \,\%\, f_O == 0$ then
        Get samples $\{S^i, a_D^i, a_O^i, r_D^i, r_O^i\}_{i \in [E]}$ from
          memO;
        Replay all $E$ games $S^i, a_D^i, \tilde{a}_O^i$ with
          $\tilde{a}_O^i \sim \pi_O(\cdot|S; \boldsymbol{w_D})$ to obtain rewards $\tilde{r}_D^i, \tilde{r}_O^i$;
        $\boldsymbol{\nabla_{w_O} J_O} = \frac{1}{E} \sum_{i=1}^{E} \nabla_{\boldsymbol{w_O}} \pi_O(\tilde{a}_O^i | S^i; \boldsymbol{w_O}) \, \tilde{r}_O^i$;
        $\boldsymbol{w_O} := \boldsymbol{w_O} + \frac{\alpha_O}{1 + ep\, \beta_O} \boldsymbol{\nabla_{w_O} J_O}$;
```

policy gradient learning [Sutton *et al.*, 1999] and those from game theoretic fictitious play [Heinrich *et al.*, 2015; Heinrich and Silver, 2016], to find the optimal defender parameters $\boldsymbol{w_D}$ which maximize her utility.

The pseudocode for OptGradFP has been provided in algorithm 1. The algorithm computes the optimal policy from the defender side. It essentially gets a game state S and it generates responses from the player's current policy estimates π_D, π_O to receive rewards r_D, r_O. It keeps storing all these game samples in separate replay memories $memD$ and $memO$ for the defender and the opponent. The replay memories are circular buffers of length E and once full, they overwrite the least recently stored sample in order to accommodate new incoming samples.

Every few episodes (f_D for defender, f_O for opponent), the algorithm retrieves all samples stored in the replay memory for a player, replays all the 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 is essentially a soft update towards the best response by changing the player's policy parameters ($\boldsymbol{w_D}$ or $\boldsymbol{w_O}$) in a way that increases their expected reward. We employ learning rate decay to take larger steps initially and obtain a finer convergence towards the end. Yet, it is not a soft update towards a best response to the other player's current policy, but rather towards a best response to an *average* of the other player's current and previous policies. This is because we compute the policy gradient for a player using all E samples in the player's replay memory, out of which only f_D (or f_O) samples are drawn from the other player's current policy and the rest are from the other player's previous policies ($E \gg f_D, f_O$). But all E samples employ the player's current policy, since the player replays all games with his/her current policy before using them to make the policy gradient update. This is an approximation to fictitious play where both players react with a best response to the other player's *average* strategy.

Note that replaying 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.

6 OptGradFP-NN: OptGradFP with neural networks

For our OptGradFP implementation, 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 action spaces are bounded and continuous).

To represent these distributions 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 $a_O = (\boldsymbol{\rho}, \boldsymbol{\phi})$. Though any function approximators can be used for this purpose, we use two convolutional neural networks to generate the means and standard deviations for each player, owing to their recent success in image processing and computer vision applications [Krizhevsky *et al.*, 2012; Zeiler and Fergus, 2014].

Defender policy representation: The defender neural network parametrized 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 joint probability of taking action $a_D = (\boldsymbol{d}, \boldsymbol{\theta})$ is given by:

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

where the product is over all m elements of the vector.

The defender neural network takes an image of size 120×120 as input. First hidden layer is a convolutional layer with 32 filters of size 16×16 and strides 8×8. The second hidden layer is convolutional with 16 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 $32m$ 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].

Opponent policy representation: The opponent neural network similarly parametrized by weights $\boldsymbol{w_O}$ takes as S as input and outputs means $(\boldsymbol{\mu_\rho}(S; \boldsymbol{w_O}) \in \mathbb{R}^n, \boldsymbol{\mu_\phi}(S; \boldsymbol{w_O}) \in \mathbb{R}^n)$ and standard deviations $(\boldsymbol{\nu_\rho}(S; \boldsymbol{w_O}) \in \mathbb{R}^n, \boldsymbol{\nu_\phi}(S; \boldsymbol{w_O}) \in \mathbb{R}^n)$ for two n-dimensional gaussians. For clarity we will skip writing $(S; \boldsymbol{w_O})$ with these parameters. Each opponent action coordinate is then a logit-normal distribution and the joint probability of taking action $a_O = (\boldsymbol{\rho}, \boldsymbol{\phi})$ is:

$$\pi_O(\boldsymbol{\rho}, \boldsymbol{\phi} | S; \boldsymbol{w_O}) = \prod_{i \in [n]} p_{ln}(\rho_i; \mu_{\rho,i}, \nu_{\rho,i}) \\ p_{ln}\left(\frac{\phi_i}{2\pi}; \mu_{\phi,i}, \nu_{\phi,i}\right) \quad (10)$$

where the product is over all n elements of the vector.

The opponent neural network is similar to the defender network, with the only difference being the number of hidden units in the fully-connected dense layers. The fully-connected hidden layer has $32n$ 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.

Finally, note that even though we assumed all elements of a_D (resp. a_O) to be independent logit-normal distributions, the means and standard deviations for the underlying normal distributions are computed jointly via the convolutional neural networks and are dependent on each other. This allows the defender and the opponent to plan coordinated moves.

Hyperparameters: Our OptGradFP implementation uses a replay memory size of $E = 1000$ samples, maximum episodes $ep_{max} = 10000$, learning rates $\alpha_D = \alpha_O = 10^{-5}$, training rates $f_D = f_O = 50$ and decays $\beta_D = \beta_O = 0.045$. The architectures of all neural networks involved and all algorithm hyperparameters were chosen by doing informal grid searches within appropriate intervals. For more information on choosing convolutional neural network architectures, refer to [Ullah and Petrosino, 2016].

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 sequentially respond to each other's policy with their best response policies. 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 *et al.*, 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 compare our results against those from CA and also with a version of StackGrad in our experiments.

Note that StackGrad uses a best response computation for the opponent in the original paper (approximated by a parametrized softmax distribution). Since it is hard to compute the best response to any policy analytically for our forest domain, we use the following approximation to emulate the opponent's best response: we play multiple games with random actions for the opponent while drawing the defender's actions from its current policy. The random action which gets the highest reward against the defender's policy is chosen as the best response action for the opponent.

We present our results for $m = 8$ guards and $n = 8$ lumberjacks where the numbers have been chosen to provide appropriate coverage of the forest (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). These are just representative values and our algorithm works well for any values of these parameters.

7.1 Reward curves

Figure 2 shows a plot of the average reward achieved by the defender on the E replayed games before every training iteration. In OptGradFP, these average rewards measure the utility of the defender's current policy against the opponent's average policy. For CA and StackGrad, the average rewards measure the defender's utility against the opponent's current policy (no stored history for CA and StackGrad). We observe that StackGrad starts with a random value of average reward for the defender (due to random initialization) and thereafter goes down. The curve mostly stays around the same average reward while displaying consistent oscillations. CA on the other hand, jumps up to a higher average reward value than random and oscillates around it. Finally, OptGradFP smoothly rises and saturates at an average reward value higher than all other baselines.

7.2 Opponent's final utility

Another indicator of performance is the utility achieved by opponent's final policy after the defender has fixed her own policy. The opponent's maximum utility was computed approximately (computing the actual value is extremely prohibitive), by sampling 100 random opponent actions and 100 actions from the defender's final policy. 10000 games were played with each combination of the defender's and opponent's actions and the opponent action which led to the maximum reward for the opponent (averaged over all 100 defender actions) was assumed to be the opponent's final action.

Table 1 gives the opponent's maximum utility after ep_{max} episodes for each algorithm. OptGradFP clearly gives the least utility to the opponent as opposed to other baselines. Further, StackGrad dominates CA which suggests that though CA quickly finds a best response to the opponent's estimated current strategy, it's policy does not provide appropriate coverage of the forest region. After the policy has been deployed, the opponent can still find high utility places to attack.

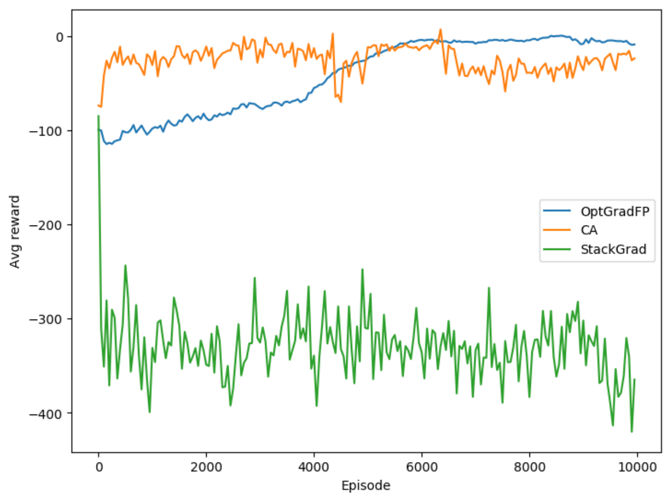

Figure 2: Average reward for E replayed games before every training iteration

	CA	StackGrad	OptGradFP
Max Util	567.05	518.34	**499.15**

Table 1: Maximum average utility of the opponent.

7.3 Learned defender policy

We show a visualization of the defender's final policy for each algorithm in figure 3. Though [Johnson *et al.*, 2012] proposed circular bands as the optimal patrol policy for a uniform tree density, the same still holds for radially symmetric tree densities. Our tree distribution is close to being radially symmetric, and hence the optimal defender policy should be in the form of circular bands centered at the origin. Figure 3c shows the OptGradFP defender policy, and as expected it contains mostly circular bands centered around the origin. The other algorithms find local regions to guard, and leave the lumberjacks lots of space to chop wood without getting ambushed. This is in agreement with the maximum average utility values for the opponent in table 1. One important point to note is the placement of the circular bands. It is easy to see that placing the bands too close to the forest center would leave a huge area to be chopped by the lumberjacks. Also, placing the guards at the boundary would make them very sparsely distributed and most lumberjacks would be able to come and go unambushed. Our algorithm finds a reasonable middle ground by inferring good radii to place the guards.

7.4 Replay memory

We also explored effects of not using fictitious play in OptGradFP. To do this, we use a small replay memory of size $E = f_D = f_O$, only containing games sampled from current policies of both players. This is equivalent to making only policy gradient updates for both players.

The utility achieved by opponent's best response policy was 555.58, which is only slightly better than CA and worse than both StackGrad and OptGradFP. The resulting defender distribution is shown in figure 3d. The defender policy is not well spread out anymore, since the method does not have memory of opponent's previous steps. The results resemble those of CA, since policy gradient update is a small step towards the best response (like in CA). Hence we can approximate CA with OptGradFP using $E = f_D = f_O$.

In general, keeping a large replay memory $(E \gg f_D, f_O)$ gave us less fluctuation and smoother convergence properties, by allowing the policy gradient update to approximate the best response to the other player's *average* policy better. At the same time, a large replay memory led to the replaying time of games becoming a bottleneck for every training step. Hence there exists a trade-off between smooth convergence vs. computation time, which needs tuning to balance the two.

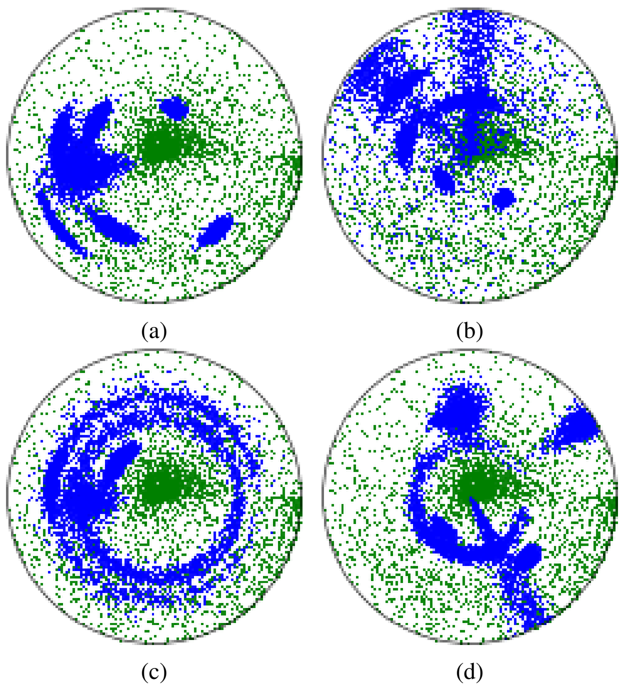

Figure 3: Visualization of the defender's policy. The blue dots show sampled positions for the guards. The locations with many blue dots are the regions where the distribution is concentrated at: (a) Defender policy: CA, (b) Defender policy: StackGrad, (c) Defender policy: OptGradFP and (d) After removing replay memory.

8 Conclusion and Future Work

In this paper, we present for the first time, a neural network based approach to address continuous space security games that previous approaches fail to handle. Our novel approach OptGradFP represents the defender's strategy by parametrizing it in continuous space and training neural networks using fictitious play and policy gradients to learn the parameters. While we have only trained on a single state, note that our approach OptGradFP-NN is generic enough to train the defender's policy over multiple distinct game states. Experiments to evaluate this are a promising direction for future work. Generalizing the model to handle arbitrary forest shapes is also a challenge to be addressed in the future.

References

[Amin *et al.*, 2016] Kareem Amin, Satinder Singh, and Michael P Wellman. Gradient methods for stackelberg security games. In *UAI*, pages 2–11. AUAI Press, 2016.

[Fang *et al.*, 2013] Fei Fang, Albert Xin Jiang, and Milind Tambe. Optimal patrol strategy for protecting moving targets with multiple mobile resources. In *AAMAS*, pages 957–964, 2013.

[Fang *et al.*, 2015] Fei Fang, Peter Stone, and Milind Tambe. When security games go green: Designing defender strategies to prevent poaching and illegal fishing. In *IJCAI*, 2015.

[Fudenberg and Levine, 1998] Drew Fudenberg and David K Levine. *The theory of learning in games*, volume 2. MIT press, 1998.

[Gan *et al.*, 2017] Jiarui Gan, Bo An, Yevgeniy Vorobeychik, and Brian Gauch. Security games on a plane. In *AAAI*, pages 530–536, 2017.

[Haskell *et al.*, 2014] William Haskell, Debarun Kar, Fei Fang, Milind Tambe, Sam Cheung, and Elizabeth Denicola. Robust protection of fisheries with compass. In *IAAI*, 2014.

[Heinrich and Silver, 2016] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *CoRR*, abs/1603.01121, 2016.

[Heinrich *et al.*, 2015] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *ICML*, pages 805–813, 2015.

[Johnson *et al.*, 2012] Matthew P. Johnson, Fei Fang, and Milind Tambe. Patrol strategies to maximize pristine forest area. In *AAAI*, 2012.

[Kar *et al.*, 2015] Debarun Kar, Fei Fang, Francesco Delle Fave, Nicole Sintov, and Milind Tambe. "a game of thrones": When human behavior models compete in repeated stackelberg security games. In *AAMAS*, 2015.

[Kiekintveld *et al.*, 2009] Christopher Kiekintveld, Manish Jain, Jason Tsai, James Pita, Fernando Ordóñez, and Milind Tambe. Computing optimal randomized resource allocations for massive security games. In *AAMAS*, pages 689–696, 2009.

[Korzhyk *et al.*, 2011] Dmytro Korzhyk, Zhengyu Yin, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. Stackelberg vs. nash in security games: An extended investigation of interchangeability, equivalence, and uniqueness. *JAIR*, 41:297–327, 2011.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *NIPS*, pages 1097–1105. 2012.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Sutton *et al.*, 1999] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

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

[Ullah and Petrosino, 2016] Ihsan Ullah and Alfredo Petrosino. About pyramid structure in convolutional neural networks. In *IEEE 2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1318–1324, 2016.

[Yang *et al.*, 2014] Rong Yang, Benjamin Ford, Milind Tambe, and Andrew Lemieux. Adaptive resource allocation for wildlife protection against illegal poachers. In *AAMAS*, 2014.

[Yin *et al.*, 2014] Yue Yin, Bo An, and Manish Jain. Game-theoretic resource allocation for protecting large public events. In *AAAI*, pages 826–833, 2014.

[Zeiler and Fergus, 2014] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.