Machine Learning in Interacting Multi-agent Systems

by

Nitin Kamra

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Science)

August 2021

Acknowledgements

I would first like to thank my advisor, Prof Yan Liu, for constantly guiding me through this journey and providing valuable feedback on my work. Without her excellent supervision and persistent mentoring, I would not have been able to complete this thesis. I would also like to thank my PhD committee members, Prof Bistra Dilkina and Prof Ashutosh Nayyar, for finding time from their hectic schedule to provide supervision, support and insightful comments for my thesis. During this journey, I have also had a fair opportunity to collaborate with excellent researchers who helped me improve in my field and become better at doing research. I would like to express my gratitude to Prof Milind Tambe, Prof Fei Fang, Prof Nora Ayanian and Prof Satish Kumar Thittamaranahalli for providing me their time and guidance during various stages of my PhD research. I am also grateful to my friends, labmates and colleagues Palash Goyal, Umang Gupta, Wolfgang Hoenig, Artem Molchanov, Michael Tsang, Sirisha Rambhatla and Ayush Jain for engaging with me in countless discussions, debates and brainstorming sessions which provided me a lot of inspiration towards new ideas. Last, but not the least, I express my gratitude to my family, especially my mother who helped me immensely to carry on with my PhD.

Table of Contents

List Of Tables

List Of Figures

Abstract

Making predictions and learning optimal behavioral strategies are important problems in many domains such as traffic prediction, pedestrian tracking, financial investments and security systems. These systems often consist of multiple agents interacting with each other in complex ways, which makes both the above tasks very challenging in nature. In this thesis, I study and propose methods to advance the state-of-the-art for several multi-agent learning problems. The first work on trajectory prediction presents a relational model involving a fuzzy decision making attention mechanism for multi-agent trajectory prediction. Our approach shows significant performance gains over many existing state-of-the-art predictive models in diverse domains such as human crowds, US freeway traffic and various physics datasets. The second work focuses on computing nash equilibrium strategies in spatial security games with continuous action spaces. We present OptGradFP, a novel and general model-free learning 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. The third work introduces DeepFP, a model-based strategy learning algorithm which addresses several challenges with OptGradFP and improves upon it. We demonstrate stable convergence to Nash equilibrium on several classic games and also apply our methods to a large forest security domain thereby demonstrating the robustness of the computed strategies against adversarial exploitation. Finally, my last work focuses on placing multiple resources to protect and cover geographical spaces. We propose the Coverage Gradient Theorem and combine it with existing genetic algorithms and my previous algorithm, DeepFP, to improve existing benchmarks for spatial coverage domains.

Chapter 1

Introduction

1.1 Learning in multi-agent systems

Multi-agent systems are ubiquitous today and arise in almost all practical domains like traffic trajectory prediction [141, 81], pedestrian tracking in crowds [1, 9], path planning problems [107], infrastructure security [124, 106, 16, 7], game AI [115] etc. These systems are characterized by a set of agents/entities each with their own separate goals. The policy required by each agent to achieve their goal is not independent of other agents' policies. Hence, they all co-exist and interact in a common environment while affecting each other's policies in order to achieve their own goals. Devising models to capture such interaction between multiple agents is the primary focus of this manuscript.

While there exist a plethora of interesting multi-agent system problems, we will be focusing on a select few in this thesis due to vastness of the domain. Most conventional research in multi-agent systems focuses on design, planning and performance for systems with multiple interacting entities. However the advent of machine learning has given rise to a new set of challenging problems which focus on settings with multiple goal-oriented agents interacting with each other and learning autonomously in the presence of other agents. We will keep our focus on multi-agent learning problems in this manuscript. The key theme behind my work will be to devise models which can

learn about interactions in a multi-agent or multi-entity system and either make future predictions or devise actionable strategies to behave optimally in such systems.

1.2 Challenges in multi-agent learning

We begin by characterizing learning problems in multi-agent systems. While there could be many potential ways to characterize them based on different criterion, for the purpose of this work we will characterize them into three broad categories as described in the upcoming sections.

1.2.1 Multi-agent prediction

Prediction problems require an algorithm to make predictions in a system comprising of multiple agents interacting with each other. While for specific simple applications of interest, one could potentially hard-code a prediction system with handcrafted rules, such an approach does not often scale to large and more complex practical systems. Hence, learning becomes a key component of such a prediction algorithm. Since the agents are generally acting autonomously with their own goals, with potentially limited sensing and observation capabilities, the key challenge for the prediction algorithm in this setting is to learn to detect changes in agents' behavior resulting from interactions with other agents and learn to model the effects of such changes. However, modeling interactions between agents can often be challenging because of the following reasons:

- Interaction between agents often changes over time and it is hard to infer precisely when two agents are interacting with each other.
- Changes to agents' behaviors resulting from interactions can be quite complex to model in general. In domains involving humans, these interactions often have a fuzzy nature to them. For instance, a person driving a car on a freeway might reason along these lines: "The car in front of me is slowing down so I should also step on the brake lightly to avoid tailing the car closely", without ever precisely quantifying the degree of *slowing down*, *braking lightly* or

following closely. Characterizing such interactions can require building a learnable attention mechanism which should be able to take fuzzy decision making into account.

1.2.2 Multi-agent control

Multi-agent control problems comprise of multiple agents co-existing in a common environment and each trying to achieve its own goal. Each agent needs to learn a policy which accomplishes its goal in the presence of the other agents while accounting for the effects of their actions on the agent under focus. In such cases, the agents learn concurrently and their ever-changing policies often dictate changes in the behaviors of other agents. Hence, learning to control in a multi-agent setting is a more challenging problem than its corresponding single-agent counterpart. This is primarily due to the following reasons:

- While in a single-agent learning scenario, the optimal policy is often deterministic, this is no longer true in a multi-agent learning setting. When multiple agents co-exist, each agent might need to randomize his/her strategy if there exists any other agent in the environment with a goal conflicting with that of this agent. Such stochastic strategies are necessary to prevent exploitation by other adversarial agents.

- Secondly, no agent can optimize a stationary objective. Any agent's objective often depends on all other agents' strategies and requires all agents to learn strategies which achieve equilibrium in some sense e.g., Nash equilibrium. This way all agents are forced to learn strategies which are pareto-optimal and no player has any incentive to deviate from his/her strategy while the others stick to their respective strategies. While there have been significant advances in single-agent reinforcement learning and control [122, 92, 91], multi-agent control still suffers from the problem of non-stationary objectives and an agent's strategy learning can often go around in circles because the other agents learning in tandem can counteract its learning [53].

Further, while there have been recent works which target multi-agent reinforcement learning in games with discrete action spaces [115, 88, 22, 96], the problem still remains unsolved in continuous action spaces where it is much harder to approximate and learn probability densities flexibly.

1.2.3 Differentiable modeling of multi-agent systems

Credit allocation is one of the most important problems in multi-agent learning. To be able to learn policies for all agents jointly, one often requires a predictive model of the multi-agent system which can allocate credit to each agent's individual action for any given objective. However, when multiple agents are interacting in a common environment and an event happens due to their joint actions, it is often hard to allocate credit to their individual actions for the event. When learning optimal (or pareto-optimal) policies in continuous action spaces, such a credit allocation model often boils down to having a continuous and differentiable reward prediction model of the multi-agent system, in which backpropagation can then allow for credit assignment. However, learning such differentiable models comes with its own set of challenges in terms of accuracy and performance guarantees. While it is not always possible to simplify the design of differentiable reward models, we will consider the problem of designing differentiable reward models for the specific domain of multi-resource spatial coverage and tackle some of the common challenges which make the reward models non-differentiable in this domain.

1.2.4 Working in continuous action spaces

This thesis focuses on prediction, control and designing differentiable reward models for interacting multi-agent systems. We focus on games and settings with continuous action spaces in this manuscript. Before moving forward it is important to justify this decision.

While there have been recent works which target multi-agent reinforcement learning in games with discrete action spaces [115, 88, 22, 96], the problem still remains unsolved in continuous action spaces where it is much harder to approximate and learn probability densities flexibly.

More specifically, we consider Stackelberg Security Games (SSGs), which have been extensively used to model defender-adversary interaction in protecting important infrastructure targets such as airports, ports, and flights [106, 16, 7]. Recently, there has been an increasing interest in SSGs for green security domains such as protecting wildlife [69, 134], fisheries [46] and forests [63]. Unlike infrastructure protection domains which have discrete locations, green security domains are categorized by continuous action spaces (e.g., a whole conservation area needs protection) for placing resources, which makes it hard to approximate and learn stochastic strategies flexibly.

Notably, many previous works, especially in spatial security game domains [142, 46, 36, 139] have chosen to discretize the state and action spaces involved to find equilibrium strategies. However, note that in reality an attacker 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 growing number of players' resources) [124]. For instance, consider a coarse discretization of a 2D forest domain into a 100×100 grid. Let us assume we wish to cover this forest with 10 drones. Note that this discretization already gives us an intractable number of joint drone placements ($(100 \times 100)^{10} = 10^{40}$). While column generation and double oracle based approaches can somewhat improve computation efficiency, the memory and runtime requirement still remains high [139]. Hence, in our work we have chosen to keep the action spaces of all agents (or their placed resources) continuous and with this choice we are able to benefit from continuity of the agents' action spaces and from gradient-based methods.

1.3 Research contributions of thesis

Our contributions for this manuscript are briefly summarized below:

Multi-agent prediction: Our first contribution is to address the problem of predicting trajectories of multiple agents interacting with each other. Trajectory prediction for scenes with multiple agents and entities is a challenging problem in numerous domains such as traffic prediction,

pedestrian tracking and path planning. We present a general architecture to address this challenge which models the crucial inductive biases of motion, namely, inertia, relative motion, intents and interactions. Specifically, we propose a relational model to flexibly model interactions between agents in diverse environments. Since fuzzy representations without precise quantification enter routinely into human interactions and decision making processes [24], we posit that a model learning to predict trajectories of interacting agents can benefit from embedded fuzzy decision making capabilities. At the core of our model lies a novel attention mechanism, namely Fuzzy Query Attention (FQA). It models pairwise attention to decide about when two agents are interacting by learning keys and queries which are combined with a dot-product structure to make continuous-valued (fuzzy) decisions. It also simultaneously learns how the agent under focus is affected by the influencing agent given the fuzzy decisions. We demonstrate significant performance gains over existing state-of-the-art predictive models in five domains: (a) trajectories of human crowd, (b) US freeway traffic, (c) object motion and collisions governed by Newtonian mechanics, (d) motion of charged particles under electrostatic fields, and (e) NBA sports data, thereby showing that FQA can learn to model very diverse kinds of interactions. Our experiments show that our model derives its strength from fuzzy decision making and the fuzzy decisions made over time are highly predictive of interactions even when all other input features are ignored. Lastly, we show that our architecture supports adding human knowledge in the form of fuzzy decisions, which can provide further gains in prediction performance.

Multi-agent control: Our next set of contributions is in the field of Stackelberg Security Games (SSGs). We provide a novel approach for solving security games based on reinforcement learning, fictitious play and deep learning. This approach extends the existing toolkit to handle complex settings such as general games with continuous action spaces. 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. Our experimental analysis with OptGradFP demonstrates the superiority of our approach

against comparable approaches such as StackGrad [3] and Cournot Adjustment (CA) [34]. Next we present DeepFP, which addresses the weaknesses of OptGradFP. The key novelties of DeepFP are: (a) It represents players' approximate best responses via state-of-the-art generative neural networks which are highly expressive implicit density approximators with no shape assumptions on players' action spaces, (b) Since implicit density models cannot be trained directly, it also uses a game-model network which is a differentiable approximation of the players' payoffs given their actions, and trains these networks end-to-end in a model-based learning regime, and (c) DeepFP allows replacing these networks with domain-specific oracles if available. This allows working in the absence of gradients for player/(s) and exploit techniques from research areas like mathematical programming to compute best responses. DeepFP addresses the lack of representational power of OptGradFP via flexible implicit density approximators. Further, its model-based training proceeds without any likelihood estimates and hence does not yield $-\infty$ log-likelihoods in any parts of the action space, thereby converging stably. Moreover, unlike OptGradFP, DeepFP is an off-policy algorithm and trains significantly faster by directly estimating expected rewards using the game model network instead of replaying previously stored games.

Differentiable modeling of multi-agent systems: Lastly, we contribute to building of differentiable reward models in multi-resource spatial coverage domains. Allocation of multiple resources for efficient spatial coverage is an important component in many practical systems, e.g., robotic surveillance, mobile sensor networks and green security domains. Most conventional solution approaches either: (a) rely on exploiting spatio-temporal structure of specific coverage problems, or (b) use genetic algorithms when targeting general coverage problems where no special exploitable structure exists. We instead propose the *coverage gradient theorem*, which provides a gradient estimator for a broad class of spatial coverage objectives using a combination of Newton-Leibniz theorem and implicit boundary differentiation. This allows differentiable credit assignment for the placement of different resources towards a given coverage objective. We also propose a tractable framework to approximate the coverage objectives and their gradients using spatial discretization.

Hence, we keep the resource allocations amenable to gradient-based optimization thereby leading to faster, scalable and more directed ways of search and optimization for multi-resource coverage problems. By combining our framework with existing optimization methods, we demonstrate successful applications on both surveillance and green security spatial coverage domains.

Chapter 2

Survey of Related Work

Learning in multi-agent systems has been a long standing research challenge in numerous practical domains. Since a complete literature review is out of scope of this thesis, we instead choose a few key practical domains to study and focus here on recent works in these domains.

Specifically, study of interaction is a common problem when a model needs to predict trajectories of multiple agents, e.g., in pedestrian trajectory prediction or predicting trajectories of vehicles around a self-driving vehicle. Hence, we will choose multi-agent trajectory prediction as the specific domain to study interaction modeling between multiple agents.

Similarly, studying interaction between more than one agent and learning optimal policies can happen in a vast variety of domains. We choose to focus on a narrow subset, namely, Stackelberg Security Games (SSGs) because of its practical importance in modeling protection of critical targets like forests, airports, wildlife etc. For the same domain, we shall also study optimal resource placement for spatial coverage for credit allocation in multi-agent systems.

Hence, we will primarily focus on a literature review of multi-agent trajectory prediction, Stackelberg Security Games (and the surrounding game theoretic constructs) and multi-resource spatial coverage domains. The rest of this chapter discusses these domains, cites the most recent related work and contrasts the work presented in this manuscript to the related work.

2.1 Multi-agent trajectory prediction

Multi-agent trajectory prediction is a well-studied problem spanning across many domains such as modeling human interactions for navigation, pedestrian trajectory prediction, spatio-temporal prediction, multi-robot path planning, traffic prediction, etc. While earlier work on trajectory prediction focused on simple social force based models, more recent line of work has focused on using graph recurrent neural network architectures.

2.1.1 Social Force based models

Early work on predicting trajectories of multiple interacting agents dates back to more than two decades starting from Helbing and Molnar's social force model [51] aimed at modeling behavior of pedestrians in crowds. The key idea is that the motion of pedestrians can be described as if they would be subject to *social forces*. The model assumes three types of social forces: (a) A term for accelerating towards the desired direction of motion, (b) Terms to maintain a certain safe distance from other pedestrians and boundaries, and (c) A term to model attractive effects. This early model results in non-linearly coupled Langevin equations and was empirically shown to be capable of reasonably describing self-organization of pedestrian behavior.

The model was later extended and applied to videos recorded from birds-eye view at busy locations for multi-people tracking [98] and also linked with an energy-based formulation [141]. These models primarily assumes that attractive and repulsive social forces govern the behavior of agents in vicinity of each other and magnitude coefficients of such forces were often learnt from data. However, such models are often simple and do not fare well in complex scenarios like traffic trajectory prediction or complex physics domains modeling. In such scenarios models with a more flexible inductive bias are often required, e.g. deep neural networks and their variants.

2.1.2 Graph recurrent neural network based models

Due to the growing success being enjoyed by deep recurrent models like RNNs and LSTMs [55] in sequence prediction, recurrent neural networks with LSTM-based interaction modeling have recently become predominant for multi-agent trajectory prediction [89]. These models contain a general deep neural network to model interactions between agents and specialized pooling or attention mechanisms to aggregate the effect of interactions from multiple agents on a single agent.

To aggregate influence of multiple interactions, various pooling mechanisms have been proposed for both human crowds modeling [1, 40] and for predicting future motion paths of vehicles from their past trajectories [23]. Specifically, Alahi *et al.* [1] model trajectories of agents using LSTMs and pool the internal LSTM hidden states of agents in a discretized neighborhood of an agent of focus to model the effects of neighboring agents onto the focused agent. While discretizing the neighborhood is a simple method, it often introduces inaccuracies in summarizing hidden states which was later improved by [40] with a direct mean-pooling mechanism. They also introduced a generative adversarial network (GAN) based architecture for diverse trajectory sample generation. The idea was also augmented to that of convolutional social pooling by [23] and applied to vehicle trajectory prediction on freeways.

Many state-of-the-art models have also incorporated attention mechanisms to predict motion of human pedestrians in crowds. [129] proposed an end to end deep learning model to learn the motion patterns of humans using different navigational modes including a soft attention mechanism. The architecture is extendable to handle multiple modes of movements (e.g. pedestrians, bikers and buses) simultaneously. [132] propose *Social Attention* to capture the relative importance of each neighboring agent onto an agent of focus when navigating in a crowd, while [29] propose a combination of soft-attention and hard-wired attention in order to map trajectory information from the local neighbourhood of a pedestrian to its future positions.

For a review and benchmark of different approaches in this domain, we refer the interested reader to [9].

2.1.3 Other learning based methods

Other recent works in traffic modeling have used deep autoencoder models with additional scene context for trajectory prediction [81] and learnt safe driving policies using reinforcement learning via model-predictive control [52]. Many recent works have also studied trajectory prediction for particles in mechanical and dynamical systems where the goal is to learn the underlying laws of physics which govern the motion of particles either by using learnable network architectures [17, 73] or via topological invariants enforced by Hamiltonian Dynamics [90]. Other works have employed hierarchical neural network encoder-decoder models for predicting trajectories of soccer and basketball players [148, 57, 119, 147] and designed specialized models suited for predicting trajectories in multi-robot path planning [107]. More recently, additional architectures have been proposed inspired form the psychological *Theory of Mind* for predicting actions of agents trained with reinforcement learning [102].

2.1.4 Comparison with our FQA model

A recurring theme in many of the above works is to view the agents/entities as nodes in a graph while capturing their interactions via the graph edges. Since graph neural networks can be employed to learn patterns from graph-structured data [44, 8], the problem reduces to learning an appropriate variant of graph neural networks to learn the interactions and predict the trajectories of all agents [123]. Recent works have devised different variants of graph networks, e.g. with direct edge-feature aggregation [44, 8], edge-type inference [73], modeling spatio-temporal relations [61], and attention on edges between agents [131] to predict multi-agent trajectories in diverse settings.

Our work (Fuzzy Query Attention [68]) assumes an underlying graph-based representation but differs from the above literature in the use of a novel attention mechanism to capture interactions

between agents. Our attention mechanism learns keys and queries to make fuzzy decisions about when and how two agents are interacting, and further models the effects of the interaction. The learnt fuzzy decision variables are highly predictive of interactions between pairs of agents and our architecture also allows incorporating human-knowledge in the form of hard-coded fuzzy decisions.

2.2 Learning to control in multi-agent games

Next we consider the case where multiple agents in a multi-agent learning are individually trying to learn optimal policies to meet their own goals. This setting naturally forays us into the realms of game theory. However, using reinforcement learning to learn optimal policies can be a challenging task when multiple agents interact with each other simultaneously because from the perspective of any single agent, the environment appears to be non-stationary. We will first describe the state-of-the-art in reinforcement learning for when a single agent interacts with a stationary environment over time. Next we will focus on extensions to the multi-agent setup and mitigating the non-stationarity induced by the presence of multiple agents. Finally we will review the specific case of Stackelberg Security Games which will be the domain under consideration in this manuscript.

2.2.1 Single-agent reinforcement learning

Reinforcement learning (RL) is the classic learning paradigm which allows an agent to interact with an environment repeatedly and optimize its behavioral policy over time towards achieving a designated goal. Traditionally, reinforcement learning is based on the framework of Markov Decision Processes (MDPs) [13], which assumes a set of states $\mathcal{S}$ available to the agent in which the agent can take an action from its pre-designated action set $\mathcal{A}$. This leads to the environment transitioning to a new state as determined by its fixed (but generally unknown to the agent) transition distribution $\mathcal{T}$ and the agent achieving a reward for its action. By collecting rewards

over multiple state-action trajectories being executed in the environment, the agent eventually needs to learn to take actions which lead to a better total reward over a trajectory. Further extensions of MDPs dis-allow the agent from receiving the full state, rather they only allow access to a restricted transformation of the state i.e. an observation. Such setting is called a Partially Observable MDPs or POMDPs [5, 64]. For the purpose of this manuscript we will primarily focus on techniques for learning in MDPs and leave the POMDP extensions to future work.

There are primarily two kinds of methods in single-agent reinforcement learning: (a) Model-free RL and (b) Model-based RL. The former class of methods rely on learning either optimal value functions or optimal policies or both directly without learning a model for the environment's transition distribution. The latter class of methods rely on learning a model of the environment's transition distribution and subsequently using it to learn optimal behavioral policies.

2.2.1.1 Value-function based Model-free RL

A key idea in reinforcement learning is that of a value function Q^π, with $Q^\pi(s, a)$ denoting the long-term utility of taking an action a in state s and following the policy π thereafter. By learning the optimal value-function Q^*, one can always extract the best action in any given state, a.k.a. the optimal policy as $\pi^*(s) = \arg\max_a Q^*(s, a)$. While value-based methods had already existed for a few decades for single agent games, e.g., Q-learning [137], they were recently revived again with the advent of deep learning. The first successful application of this technique was the DQN [92], which used a flexible deep neural network as a function approximator in Q-learning to store the Q^* function. Mnih *et al.* [92] additionally employed a target neural network and a replay memory to stabilize the convergence of Q-learning under function approximation. This was tested on a suite of 51 Atari 2600 games, in majority of which it outperformed human-level scores by learning directly from high-dimensional sensory inputs.

The DQN algorithm has since been researched further and evolved significantly by many subsequent works. Instead of directly approximating the Q^* function, Wang *et al.* [136] approximate

the advantage of taking a specific action over the average value function of a state in the form of a dueling neural network architecture. Van Hasselt *et al.* [128] show that the DQN algorithm overestimates action values in general and can lead to a significant learning bias. They rectify it by proposing the Double-DQN algorithm which decouples the learning of the optimal Q^* function from the actual gameplay by using the target Q network to select actions and the optimal Q network to evaluate the selected actions in the Q-learning update rule. Schaul *et al.* [110] analyze the mechanism used by the replay memory of DQN to sample experiences and propose a modified replay mechanism which prioritizes drawing out experiences which have a larger error between their actual and predicted Q-value to reduce the error on such experience trajectories faster. These advances along with those in distributional RL [12] have been recently combined into the Rainbow DQN [54] which now provides state-of-the-art performance on the Atari 2600 benchmark, amongst value-based model-free methods, both in terms of data efficiency and final performance. Extensions to POMDPs have also been proposed by employing recurrent neural networks like LSTMs to approximate the Q-function, namely DRQN [47].

2.2.1.2 Policy based Model-free RL

The other key idea in model-free RL is to directly learn the optimal policy π. At its most basic level, this can be done by calculating the gradient of the desired long-term expected return w.r.t. the policy parameters and performing gradient ascent on the parameters. The policy is usually stored by a deep neural network and the gradients are computed using the policy gradient theorem [122], which is often also called the REINFORCE rule.

However, a direct application of raw policy gradients can provide very high variance gradient estimators. This is often rectified by employing value-functions as baselines to reduce the variance of vanilla policy gradient methods. Doing so results in actor-critic methods where the policy being learnt is often termed as the actor and the value-function as the critic. A popular example of

such approaches is the A3C algorithm [91] which additionally parallelizes the execution of multiple actor policies for faster data collection and accelerated learning.

Another way to employ policy gradients with controlled variance is by devising auxiliary approximations to the desired long-term return such that the gradients of these auxiliary loss functions admit more principled estimators with lower variance, such as in the Trust Region Policy Optimization (TRPO) algorithm [112] and its more popularly used variant called the Proximal Policy Optimization (PPO) algorithm [113]. These algorithms also have been rigorously tested and employed in many domains [48] and their variants proposed for: (a) scalability, using approximate factorizations [138], (b) robustness, using double Q-learning [35], and (c) sample-efficiency, using experience replay [135]. One of the most popularly employed actor-critic algorithms is the Soft Actor-Critic (SAC) proposed by Haarnoja *et al.* [42] which uses a stochastic actor with an augmented maximum-entropy objective to improve the brittle convergence properties of actor-critic methods. Other recent works explore additional relationships and equivalences between value-based and policy-based methods [95, 38, 39, 111].

2.2.1.3 Model-based RL

The final class of single-agent RL methods rely on learning a model of the environment's transition distribution and subsequently using it to learn optimal behavioral policies. One of the simplest and earliest algorithm of this class called Dyna [121] does exactly the following: it learns a model of the environment and uses it to compute the optimal policy using direct value or policy iterations with bellman updates in the tabular setting. These two steps can be iteratively repeated. More elaborately, if a fully accurate environment model is available then one can use tree search to search for the optimal action. This optimal action can be learnt by a policy and the learnt policy neural network can be used in conjunction with the tree search to guide it to more useful actions as done by the ExIt algorithm [4]. Other works use learnt generative models of the environment

to extract compressed features as inputs to the policy network or directly train the policy entirely inside of an environment generated by its own internal world model [41].

However, learning a full environment model can be both cumbersome and infeasible in large practical settings. Further, using a learnt model directly is generally highly erroneous since the errors in learning the model often also propagate into the optimal policy computation procedure. Hence, more recent models employ more indirect ways of acquiring and using learnt models. For instance, Nagabandi *et al.* [93] propose to use learnt neural network based models to bootstrap model-free policies which can then be fine-tuned using classical model-free policy gradient methods.

At times, model-based methods can be used more directly with model-free methods to reduce the sample complexity of learning RL policies. The I2C model [103] uses a learnt model to perform imagination rollouts of how a certain state can pan out in the future under various sequence of actions. A summary of these imagination rollouts encoded by a recurrent neural network is then provided as an input to the standard model-free policy neural network. This allows a more robust usage of a learnt model and the model-free policy network can learn to ignore inaccurate rollouts or some parts of them when the learnt model is not very accurate. Recently, Feinberg *et al.* [27] proposed model-based value expansion (MVE), which controls for uncertainty in the model by allowing imagination upto a fixed depth. This improves value estimation, and in turn, reduces the sample complexity of learning. Buckman *et al.* [15] introduce the stochastic ensemble value expansion (STEVE) model as another method to combine model-based approaches with model-free learning in such a way that errors in the model do not degrade performance. STEVE dynamically interpolating between model rollouts of various horizon lengths for each individual example and ensures that the model is only utilized when doing so does not introduce significant errors. This approach outperforms model-free baselines on continuous control benchmarks with an order-of-magnitude increase in sample efficiency.

Other approaches to using model-based methods in conjunction with model-free methods include Model-Ensemble Trust-Region Policy Optimization (ME-TRPO) [79] which features ensembles of

models and Model-Based Meta-Policy-Optimization (MB-MPO) [18] which uses meta-learning to prevent policy degradation due to model errors.

2.2.2 Multi-agent reinforcement learning

Reinforcement learning in multi-agent systems is studied under the framework of Markov games [85] which extends the notion of a Markov Decision Process to incorporate multiple agents with their own distinct objectives. Learning in multi-agent systems can be an extremely challenging problem since many temporal difference approaches which are feasible in a single-agent setting no longer apply in the multi-agent setting. This is because from the perspective of any single agent, the environment is no longer stationary when multiple agents are learning in tandem [53]. Consequently a plethora of evolutionary dynamics can result depending on how the different agents learning together affect each other. For a comprehensive survey, we refer the reader to [14] and focus only on the recent advances in this section.

If the agents in a multi-agent system can be guaranteed to be purely cooperative or purely adversarial by nature, this knowledge can often significantly affect the design of learning algorithms for them. Learning in more general mixed cooperative-competitive settings is still possible but can require more complex techniques. We will survey each of these settings separately below.

2.2.2.1 Learning in cooperative multi-agent games

Purely cooperative games are generally easiest to learn in since all agents share the same objective and the underlying challenge is to learn to communicate for inducing cooperation or induce cooperation directly via actions. Some of the earliest attempts at learning to communicate were CommNet [118] and Differentiable Inter-Agent Learning (DIAL) which consisted of multiple agents learning to communicate amongst themselves alongside their policy via end-to-end learning of protocols in complex environments.

While communication is generally an effective way for multi-agent cooperation, broadcasting information amongst all agents on predefined communication channels can be impairing and may even slow down learning. [22] presents the TarMAC model for targeted multi-agent communication, where agents learn both what messages to send and whom to address them to while performing cooperative tasks in partially-observable environments. Jiang and Lu [62] present an attentional communication model that learns when communication is needed and how to integrate shared information for large-scale cooperative decision making.

Sunehag *et al.* [120] study the problem of credit assignment in cooperative multi-agent games. They address this problem by training individual agents with a value decomposition network architecture (VDN), which learns to decompose the team's joint value function into agent-wise value functions and further explore the role of incorporating weight sharing, role information and information channels. Rashid *et al.* present QMIX [104] which employs a network to estimate joint action-values as a complex non-linear combination of per-agent values that condition only on local observations. The structure enforces that the joint-action value is monotonic in the per-agent values, which allows tractable maximisation of the joint action-value in off-policy learning. However, in doing so VDN and QMIX address only a fraction of factorizable multi-agent RL tasks due to assuming structural constraints in their reward factorization like additivity and monotonicity. [116] introduces QTRAN which provides another value function decomposition to alleviate such structural constraints and transforms the original joint action-value function into an easily factorizable one, while maintaining the same optimal actions.

2.2.2.2 Learning in adversarial multi-agent games

While learning to cooperate requires agents to learn to communicate, in adversarial settings, agents need to learn to outsmart their opponents. In such cases, while sometimes a deterministic policy might be optimal, e.g. in the game of Go, at other times no agent can stick to a deterministic

strategy since they might be highly exploited by his opponents and hence agents often require stochastic policies.

Some of the first recent attempts to solving adversarial games with two players are based on the idea of scaling fictitious play to large domains. [49] proposed Fictitious Self-Play (FSP), a machine learning framework that implements fictitious play in a sample-based fashion for large domains and presents experiments in imperfect-information poker games demonstrating the convergence of FSP to approximate Nash equilibria. The authors' later work Neural Fictitious Self-Play (NFSP) [50] introduces a more scalable end-to-end approach to learning approximate Nash equilibria with deep reinforcement learning. This was also the first time, where a learnt strategy in Limit Texas Holdem Poker approached the performance of state-of-the-art superhuman algorithms. Finally, the idea of self-play was scaled to solve games like Chess, Shogi and most importantly Go by combining it with deep neural networks and tree search in the AlphaGo algorithm [115]. The approach used value networks to evaluate board positions and policy networks to select moves, where these neural networks were trained by a combination of reinforcement learning from self-play and Monte Carlo tree search. AlphaGo was able to achieve a 99.8% win-rate against other Go programs and defeated multiple international human Go grandmasters.

Alternatives to fictitious play have also been proposed where the two competing players use specialized update rules to shape their own learning and also their opponent's learning. Foerster *et al.* present Learning with Opponent-Learning Awareness (LOLA) [33], where each agent shapes the anticipated learning of the other agents in the environment by including an update term that account for the impact of one agent's policy on the anticipated parameter updates of the other agents. The LOLA learning rule showed desirable behavior in certain games like the emergence of tit-for-tat strategy in the iterated prisoners' dilemma game and being robust against exploitation by higher order gradient-based methods. However, [83] showed that while experimentally successful, LOLA agents can exhibit behaviour directly at odds with convergence. The authors then presented Stable Opponent Shaping (SOS), a method to interpolate between LOLA and a stable variant called

LookAhead, which converges locally to equilibria in all differentiable games while also shaping the learning of opponents and consistently matching or outperforming LOLA. Lockhart *et al.* present a more direct approach, namely exploitability descent [86], to compute approximate equilibria in two-player zero-sum extensive-form games, by direct policy optimization against worst-case opponents. The key idea is to drive both players to force the exploitability of their strategy to converge asymptotically to zero, thereby sending the joint policies to a Nash equilibrium.

Finally, Vinyals *et al.* recently proposed AlphaStar [133], an augmented variant of self-play which involved a diverse tournament of continually adapting strategies and counter-strategies, each represented by deep neural networks. AlphaStar stood at grandmaster level and above 99.8% of officially ranked human players in the immensely complex human e-sport of StarCraft [108].

2.2.2.3 Learning in general multi-agent games

While learning in cooperative and competitive games admit algorithms based on the reward structure of the game, it is much harder to find successful algorithms to learn in any general multi-agent game. One of the earliest studies of exploring with the DQN algorithm for multi-agent settings is [125] . However, the experimentation in this work was restricted to only one of the Atari 2600 benchmark games, namely Pong, and featured simple results for extending single-agent DQN to multi-agent settings which may not work for more complex games. Later Foerster *et al.* introduced a stabilized version of experience replay for deep multi-agent RL [31] built on top of DQN. The major ideas were to: (a) use a multi-agent variant of importance sampling to decay obsolete data, and (b) condition each agent's value function on the age of the data sampled from the replay memory in order to alleviate the non-stationairty issue in multi-agent deep RL. However, such simplified heuristics do not necessarily suffice to solve any general multi-agent game.

Lanctot *et al.* presented a complex approach namely, Policy Space Response Oracles (PSRO) [80] as a unified game-theoretic approach to multi-agent RL. The algorithm was based on approximate best responses to mixtures of policies generated using deep reinforcement learning, and empirical

game-theoretic analysis to compute meta-strategies for policy selection. The algorithm generalized existing ones such as independent RL, iterated best response, double oracle and fictitious play and was tested in settings with discrete action spaces. Lowe *et al.* presented MADDPG [88], a multi-agent Actor-Critic for mixed cooperative-competitive environments that considers action policies of other agents and utilizes an ensemble of policies for each agent that leads to more robust multi-agent policies. [60] improves upon this with another actor-critic algorithm that trains decentralized policies in multi-agent settings, using centrally computed critics that share an attention mechanism which selects relevant information for each agent at every timestep. This approach applies to cooperative settings with shared rewards, individualized agent rewards, adversarial settings as well as settings that do not provide global states.

Despite all previous approaches, learning in general multi-agent games remains an open area of study. More recent works study mechanics of n-player differentiable games in general where there are multiple interacting losses. Balduzzi *et al.* [6] develop a new variant of gradient adjustment called Symplectic Gradient Adjustment (SGA), a new algorithm for finding stable fixed points in general games. The key idea is to decompose the second-order dynamics of games into two components: (a) The first related to potential games, which reduce to gradient descent on an implicit function, and (b) The second related to Hamiltonian games, a class of games that obey a conservation law, akin to those in mechanical systems. Foerster *et al.* have also recently presented the Bayesian Action Decoder (BAD) for multi-agent RL [32]. They introduce the new, public belief MDP, in which the action space consists of all deterministic partial policies, and exploits the fact that an agent acting only on this public belief state can still learn to use its private information if the action space is augmented to be over all partial policies mapping private information into environment actions. BAD surpasses all state-of-the-art approaches on the challenging, cooperative partial-information card game Hanabi in the two-player setting.

Finally, many multi-agent RL approaches also focus on settings with partial observability [96, 117] and on model-based multi-agent RL where agents learn models for other agents. For a comprehensive survey of the latter, we refer the interested reader to [2].

2.3 Stackelberg Security Games

Stackelberg Security Games (SSGs) are a sub-class of games played between two agents, namely, a defender and an attacker. The defender perpetually defends a set of targets with a limited set of resources. The targets can be discrete, e.g., entry points at an airport or continuous, e.g., tree density in a protected forest. The resources to be placed, e.g., checkpoints at airports or surveillance drones for forests, are generally assumed to be discrete and finite. The attacker is allowed to surveil the defender's resource placement strategy for an indefinite period of time. The attacker can then choose to attack a target (or a set of targets) based on the acquired information.

SSGs are an important sub-class of multi-agent games due to their practical utility in security domains and have been extensively used to model defender-adversary interaction in protecting important infrastructure targets such as airports, ports and flights [106, 16, 7].

2.3.1 Approaches to solving SSGs with discrete targets

SSGs are leader-follower games and the associated solution concept with such games is characterized as a Stackelberg Equilibrium. The solution concept characterizes the defender's strategy to commit to, such that the defender's expected utility is maximized assuming that the attacker will best respond to his/her strategy.

Some of the early results on leader-follower games come from [21] where the authors study how to compute optimal strategies to commit to under both pure and mixed strategy regimes. The authors provide both positive results in the form of efficient algorithms when the set of targets and resources are discrete and negative NP-hardness results otherwise. Additional algorithms for

large security games have been developed by [70] by assuming compact models of security games, which allow improvements in both memory and run-time compared to the best known algorithms for solving general Stackelberg games.

Most known approaches for solving SSGs with discrete targets 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 [124].

2.3.2 SSGs with continuous target densities

Recently, there has been an increasing interest in SSGs for green security domains such as for protecting wildlife [69, 134] and fisheries [46] and for devising patrol strategies to protect forest areas [63]. Unlike infrastructure protection domains which have discrete locations, green security domains are categorized by continuous spaces (e.g., a whole conservation area needs protection).

Since green security domains generally involve protecting areas, a key idea is to visualize the security game as happening on a plane (called SGP [36]) and exploit the geometry of the underlying plane by discretizing the target area into grid cells. While computing a Stackelberg equilibrium of an SGP is NP-hard even for zero-sum games, the authors of [36] are able to develop a polynomial-time approximation scheme for zero-sum SGPs with this approximation. Other previous works also discretize the target area into grid cells and restrict the players' actions to discrete sets [142, 46] to find the equilibrium strategy using linear programming (LP) or mixed-integer programming (MIP). [26] focuses on protecting mobile targets that lead to a continuous set of strategies for the players. They discretize the strategy space for the defender to employ an efficient linear-program-based solution along with a heuristic method of equilibrium refinement for improved robustness. [143] propose SCOUT-C which also discretizes the defender's action space into bins and employs linear programming to solve for a strategy efficiently. However, discretization suffers from certain key issues:

1. A fine-grained discretization makes it intractable to compute the optimal defender strategy using mathematical programming based techniques, especially when there are multiple defender resources [124].

2. While a coarse discretization of the target domain might scale better, it leads to a low solution quality for the computed strategy.

Other approaches handle continuous space by exploiting spatio-temporal structure of the game. [11] study a class of security games targets and resources moving on a real line. They provide an algorithm which runs in time polynomial in the input size, and poly-logarithmic in the number of possible resource placement locations. Later, [10] extended the work beyond a one-dimensional line to spatio-temporal graphs. However, since finding an optimal defender strategy is NP-hard on general graphs, the authors proposed an LP relaxation of the problem along with a rounding technique to obtain an approximate solution. [63] frame the problem of setting up patrols to maximize a safe forest area and numerically solve differential equations assuming spherical symmetry and uniform tree density. A key drawback of all these approaches is that they all assume special structure in the game and cannot be extended to general security game settings.

2.3.3 Fictitious Play based approaches

Since Stackelberg Equilibrium coincides with Nash Equilibrium (NE) in zero-sum security games and in some structured general-sum games [75], many general algorithms for finding mixed strategy Nash Equilibrium also apply to such security games. One of the earliest approaches to finding equilibria in continuous action spaces has been the Cournot adjustment strategy which has recently been applied with gradient-based methods by [3], however Cournot adjustment is known to suffer from convergence issues [65].

Fictitious Play (FP) is another classic algorithm studied in game theory and involves players repeatedly playing the game and best responding to each other's history of play [28]. FP converges

to a Nash equilibrium (NE) for specific classes of discrete action games [76] and is a viable practical algorithm for solving many zero-sum security games [20]. FP has laso been extended to continuous time games by [114].

More recent variants of FP like Stochastic Fictitious Play have been proven to converge under much more diverse settings of zero-sum games, potential games and super-modular games [56]. Perkins and Leslie [99] study Stochastic Fictitious Play to continuous action spaces and study the limiting behaviour of SFP using the associated smooth best response dynamics on the space of finite signed measures. They show that SFP converges to an equilibrium point in single population negative definite games, two-player zero-sum games and N-player potential games, under the assumption of Lipschitz continuous rewards. Leslie and Collins have proposed another variant called Generalized Weakened Fictitious Play (GWFP) which is known to converge under more diverse settings under reasonable regularity assumptions over underlying domains [82].

2.3.4 Fictitious Play in continuous action spaces

While FP applies to discrete action games with exact best responses, it does not trivially extend to continuous action games with arbitrarily complex best responses. Variants of FP either require explicit maximization of value functions over the action set as performed in Fictitious Self-Play [49, 50] or maintaining complex hierarchies of players' past joint strategies as in PSRO [80]. These are only feasible with finite and discrete action sets (e.g. poker) and does not generalize to continuous action spaces.

Since it is challenging to maintain distributions over continuous action spaces, recent works in multi-agent reinforcement learning [88] often assume explicit families of distributions for players' strategies which may not span the space of strategies to which NE distributions belong. More recently update rules which modify gradient descent using second-order dynamics of multi-agent games have been proposed [6].

Our work mitigates these issues by maintaining flexible densities via replay memories and by using implicit function approximators with strong representational power to approximate best responses.

2.4 Optimal resource allocation for spatial coverage

Next we focus on the allocation of multiple resources for efficient spatial coverage, which forms an important component of many practical systems, e.g., robotic surveillance, mobile sensor networks and green security domains. This problem will help us tackle the differentiable credit assignment challenge in multi-agent systems.

Traditional methods used to solve multi-resource surveillance problems often make simplifying assumptions to devise tractable solution techniques. While we will survey these methods briefly, we will primarily focus on addressing a broad class of spatial coverage problems, where special spatial-temporal structure or symmetries cannot be exploited to efficiently allocate resources for coverage.

2.4.1 Potential field methods

One of the earliest methods in this field deploys resources for coverage via construction of potential fields [58]. The fields are constructed such that each resource is repelled by both obstacles and by other resources, thereby forcing the network of resources to spread itself throughout the target domain to be covered. [101] extends this potential field method to maximize the area coverage of a domain via mobile sensors with the constraint that each sensor node has at least K neighbors in order to ensure good network coverage. If the target domain has uniform target density and the covering resources are assumed to have infinite coverage fields, then one can employ voronoi tessellation based methods [25]. Notably, these approaches make simplifying assumptions on the target domain to be covered or on the covering resources and focus on exploiting the resulting

symmetry structures. Other approaches to coverage and allocation often discretize the domain to be covered and employ specialized decompositions, for instance, Kong *et al.* employ the Boustrophedon decomposition [74] in case of a robot coverage problem.

2.4.2 Discretization based approaches

Another set of exact and approximate approaches proposed in green security game domains to compute strategies against a best responding attacker relies on discretizing the target area into grid cells and restrict the players' actions to discrete sets to find optimal allocations using linear programming (LP) or mixed-integer programming (MIP) [142, 26, 46, 143]. However as pointed out in section 2.3.2, discretization based approaches which directly discretize the action spaces suffer from intractability of computation. This problem becomes more severe as the number of resources to be placed becomes larger since the size of the discretized action space grows exponentially with the number of resources to be placed.

2.4.3 Genetic algorithm based optimization

In such cases, one has to rely on undirected exploration methods such as particle swarm optimization and genetic algorithms. Nazif *et al.* [94] propose a mechanism for covering an area by means of a group of homogeneous agents through a single-query roadmap. [109] proposes an algorithm for autonomous deployment of micro-aerial vehicles for cooperative surveillance satisfying motion constraints, environment constraints and localization constraints via particle swarm optimization. [126] proposes a regional service coverage maximization algorithm which solves the problem heuristically using a genetic algorithm. [77] present a solution to the problem of optimal placement of sensors for monitoring a spatial road network based on an iterative genetic algorithm for the optimization of a scalar metric computed from the spatial integration of the sensor influence wave. Similarly, [140] proposes an evolutionary approach for vacuum cleaner coverage of a cleaning area.

However, since the coverage problem is generally combinatorially hard, such undirected search methods also do not scale well as the number of resources to be placed grows larger.

2.4.4 Gradient based optimization methods

To address this, recent works in spatial coverage domains have focused on incorporating advances from deep learning and reinforcement learning. For instance, Pham *et al.* [100] focus on multi-UAV coverage of a field of interest using a model-free reinforcement learning method. Kamra *et al.* [67] have proposed DeepFP, a fictitious play based method to solve green security games in continuous action spaces, which relies on neural networks to provide a differentiable approximation to the coverage objectives. While efficient, these require approximating discontinuous and complex multi-resource coverage objectives using continuous and smooth neural network approximators, which can lead to subsequent inaccuracies in resource placements.

2.4.5 Comparison with our Coverage Gradient Theorem based framework

Our current work differs from the above in that we propose the *coverage gradient theorem*, which provides a gradient estimator for a broad class of spatial coverage objectives using a combination of Newton-Leibniz theorem and implicit boundary differentiation. This alleviates the need to use function approximators like neural networks to approximate gradients of the coverage objectives. We further propose a tractable framework to approximate the coverage objectives and their gradients using spatial discretization of only the target domain, but not the allocated positions of the resources. Hence, we keep the resource allocations amenable to gradient-based optimization thereby leading to faster, scalable and more directed ways of search and optimization for multi-resource coverage problems.

Chapter 3

Preliminaries, Datasets and Game Domains

In this chapter we will cover some basic preliminaries which will aid in covering the rest of the work in a more structured manner. We will also provide a detailed description of datasets used for multi-agent trajectory prediction and the various game domains used for Stackelberg Security games and for optimal resource allocation.

3.1 Preliminaries

3.1.1 Notation

We start by describing some notation useful for the rest of the manuscript. We shall denote the set of real numbers by $\mathbb{R}$ and expectation with respect to a random variable by $\mathbb{E}$. Given vectors $\boldsymbol{x}, \boldsymbol{a}$ and $\boldsymbol{b}$, 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.

3.1.2 Activation functions

The sigmoid function $\frac{1}{1+\exp(-\boldsymbol{z})}$ is denoted by $\sigma(\boldsymbol{z})$ and is a popular activation function used in neural network architectures. 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. $\sigma(\text{logit}(\boldsymbol{x})) = \boldsymbol{x}$.

Another useful activation function is the Rectified Linear Unit (often abbreviated as ReLU) and is given by $relu(\boldsymbol{z}) = \max(\boldsymbol{z}, \boldsymbol{0})$ [37]. It is often used to mitigate vanishing and exploding gradient problems in neural networks. Please refer to [45] for a detailed analysis of the ReLU function.

3.1.3 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 for this distribution is given by:

$$p_{ln}(X; \mu, \nu) = \frac{1}{\sqrt{2\pi}\nu} \frac{1}{x(1-x)} e^{-\frac{(\text{logit}(x)-\mu)^2}{2\nu^2}} \tag{3.1}$$

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 $x = \sigma(\nu\epsilon + \mu)$.

3.1.4 Two player games

We consider a two-player game with continuous action sets for players 1 and 2. We will often use the index $p \in \{1, 2\}$ for one of the players and $-p$ for the other player. U_p denotes the compact, convex action set of player p. We denote the probability density for the mixed strategy of player p at action $u_p \in U_p$ as $\sigma_p(u_p) \geq 0$ s.t. $\int_{U_p} \sigma_p(u_p) du_p = 1$. We denote player p sampling an action $u_p \in U_p$ from his mixed strategy density σ_p as $u_p \sim \sigma_p$. We denote joint actions, joint action sets and joint densities without any player subscript i.e. as $u = (u_1, u_2), U = U_1 \times U_2$ and $\sigma = (\sigma_1, \sigma_2)$ respectively.

Each player has a bounded and Lipschitz continuous reward function $r_p : U \to \mathbb{R}$. For zero-sum games, $r_p(u) + r_{-p}(u) = 0 \ \forall u \in U$. With players' mixed strategy densities σ_p and σ_{-p}, the expected reward of player p is:

$$\mathbb{E}_{u\sim\sigma}[r_p] = \int_{U_p} \int_{U_{-p}} r_p(u)\sigma_p(u_p)\sigma_{-p}(u_{-p})du_p du_{-p}.$$

The best response of player p against player $-p$'s current strategy σ_{-p} is defined as the set of strategies which maximizes his expected reward:

$$BR_p(\sigma_{-p}) := \arg\max_{\sigma_p} \left\{ \mathbb{E}_{u\sim(\sigma_p,\sigma_{-p})}[r_p] \right\}.$$

A pair of strategies $\sigma^* = (\sigma_1^*, \sigma_2^*)$ is said to be a Nash equilibrium if neither player can increase his expected reward by changing his strategy while the other player sticks to his current strategy. In such a case both these strategies belong to the best response sets to each other:

$$\sigma_1^* \in BR_1(\sigma_2^*) \text{ and } \sigma_2^* \in BR_2(\sigma_1^*).$$

3.1.5 Stackelberg Security Games

A Stackelberg Security Game (SSG) [70, 75] is a two-player leader-follower game between a defender and an adversary (a.k.a. attacker). 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. 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 manuscript, we will primarily focus on 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).

3.1.6 Fictitious Play

Fictitious play (FP) is a learning rule where each player best responds to the empirical frequency of their opponent's play. Let the density function corresponding to the empirical distribution of player p's previous actions (a.k.a. belief density) be $\bar{\sigma}_p$. Then fictitious play involves player p best responding to his opponent's belief density $\bar{\sigma}_{-p}$:

$$BR_p(\bar{\sigma}_{-p}) := \arg\max_{\sigma_p} \left\{ \mathbb{E}_{u \sim (\sigma_p, \bar{\sigma}_{-p})}[r_p] \right\}.$$

Repeating this procedure for both players is guaranteed to converge to the Nash equilibrium (NE) densities for both players for certain classes of games including two-player zero-sum games [34].

3.1.7 Policy Gradient Theorem

The policy gradient theorem is a popular tool used in reinforcement learning to calculate the gradients of an expected utility function with respect to policy paramters. According to the policy gradient theorem [122], 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

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X})] = \mathbb{E}_{\boldsymbol{X}}[f(\boldsymbol{X}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{X}|\boldsymbol{\theta})] \tag{3.2}$$

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 work.

3.1.8 Multi-resource spatial coverage problems

In this section, we formally introduce notation and definitions for multi-resource spatial coverage problems.

Multi-resource spatial coverage: Spatial coverage problems comprise of a target space $Q \subset \mathbb{R}^d$ (generally $d \in \{2, 3\}$) and a set of m resources. **Action**: An action $u \in \mathbb{R}^{m \times \hat{d}}$ is the placement of all m resources in an appropriate coordinate system of dimension $\hat{d}$. **Coverage**: When placed, each resource covers (often probabilistically) some part of the target space Q. Let $\text{cvg} : q \times u \to \mathbb{R}$ be a function denoting the coverage of a target point $q \in Q$ due to action u. We do not assume a specific form for the coverage cvg and leave it to be defined flexibly, to allow many different coverage applications to be amenable to our framework. **Reward**: The scalar coverage reward due to action u is defined as: $r(u) = \int_Q \text{cvg}(q, u) \, \text{imp}(q) \, dq$, where $\text{imp}(q)$ denotes the importance of the target point q. The objective is to optimize the placement reward r w.r.t. action u.

While the above description suffices for single player games, it can be easily extended to multi-agent games with a set of agents (or players). In such a case, the solution concept is to compute the mixed strategy Nash equilibria for all players. For brevity, we provide the extended notation and the Nash equilibria concepts associated with it in section 3.1.9 for the interested reader.

3.1.9 Extended notation for multi-agent spatial coverage games

Here we discuss the notation for multi-agent spatial coverage games more extensively.

Multi-agent multi-resource spatial coverage: Spatial coverage problems comprise of a target space $Q \subset \mathbb{R}^d$ (generally $d \in \{2, 3\}$) and a set of agents (or players) P with each agent $p \in P$ having m_p resources. We will use the notation $-p$ to denote all agents except p i.e. $P \backslash \{p\}$.

Actions: An action $u_p \in \mathbb{R}^{m_p \times d_p}$ for agent p is the placement of all its resources in an appropriate coordinate system of dimension d_p. Let U_p denote the compact, continuous and convex action set of agent p.

Mixed strategies: We represent a mixed strategy i.e. the probability density of agent p over its action set U_p as $\sigma_p(u_p) \geq 0$ s.t. $\int_{U_p} \sigma_p(u_p) du_p = 1$. We denote agent p sampling an action $u_p \in U_p$ from his mixed strategy density as $u_p \sim \sigma_p$.

Joints: Joint actions, action sets and densities for all agents together are represented as $u = \{u_p\}_{p \in P}, U = \times_{p \in P} \{U_p\}$ and $\sigma = \{\sigma_p\}_{p \in P}$ respectively.

Coverage: When placed, each resource covers (often probabilistically) some part of the target space Q. Let $\text{cvg}_p : q \times u \to \mathbb{R}$ be a function denoting the utility for agent p coming from a target point $q \in Q$ due to a joint action u for all agents. We do not assume a specific form for the coverage utility cvg_p and leave it to be defined flexibly, to allow many different coverage applications to be amenable to our framework.

Rewards: Due to the joint action u, each player achieves a coverage reward $r_p : u \to \mathbb{R}$ of the form $r_p(u) = \int_Q \text{cvg}_p(q, u) \, \text{imp}_p(q) \, dq$, where $\text{imp}_p(q)$ denotes the importance of the target point q for agent p. With a joint mixed strategy σ, player p achieves expected utility: $\mathbb{E}_{u \sim \sigma}[r_p] = \int_U r_p(u) \sigma(u) du$.

Objectives: In single-agent settings, the agent would directly optimize his expected utility w.r.t. action u_p. But in multi-agent settings, the expected utilities of agents depend on other agents' actions and hence cannot be maximized with a deterministic resource allocation due to potential

exploitation by other agents. Instead agents aim to achieve Nash equilibrium mixed strategies $\sigma = \{\sigma_p\}_{p \in P}$ over their action spaces.

Nash equilibria: A joint mixed strategy $\sigma^* = \{\sigma_p^*\}_{p \in P}$ is said to be a Nash equilibrium if no agent can increase its expected utility by changing its strategy while the other agents stick to their current strategy.

Two-player settings: While our proposed framework is not restricted to the number of agents or utility structure of the game, we focus on single-player settings and zero-sum two-player games in our work. An additional concept required by fictitious play in two-player settings is that of a best response. A best response of agent p against strategy σ_{-p} is an action which maximizes his expected utility against σ_{-p}:

$$br_p(\sigma_{-p}) \in \arg\max_{u_p} \left\{ \mathbb{E}_{u_{-p} \sim \sigma_{-p}}[r_p(u_p, u_{-p})] \right\}.$$

The expected utility of any best response of agent p is called the exploitability of agent $-p$:

$$\epsilon_{-p}(\sigma_{-p}) := \max_{u_p} \left\{ \mathbb{E}_{u_{-p} \sim \sigma_{-p}}[r_p(u_p, u_{-p})] \right\}.$$

Notably, a Nash equilibrium mixed strategy for each player is also their least exploitable strategy.

3.2 Datasets

In this section we introduce our datasets used for multi-agent trajectory prediction tasks. We chose datasets from multiple diverse domains, e.g., human crowds, freeway traffic, physics and sports analytics.

3.2.1 ETH-UCY dataset

The ETH-UCY dataset [9] is a human crowds dataset with medium interaction density. It comprises of video clips from scenes with multiple pedestrians walking past each other, at times in groups. The dataset offers dynamics like solo walking, walking in pairs and groups and collision avoidance between humans. We sampled about 3400 scenes at random from the dataset for our experiments and used total time-series length as 20 steps following prior work [1, 40].

3.2.2 Collisions dataset

Collisions is a synthetic physics dataset with balls moving on a friction-less 2D plane. The plane also contains invisible boundary walls and fixed visible circular landmarks. The dataset is based around modeling newtonian mechanics and contains about 9500 scenes with time series of length $T = 25$ steps each. The collisions between balls preserve momentum and energy, while collisions of agents with walls or immobile landmarks only preserve energy but not momentum of moving agents. This dataset is specifically challenging for most algorithms since majority of the time balls move in straight lines and collisions with walls are somewhat rare events. Even rarer are inter-ball collisions which happen very infrequently in the data and are hence hard for most models to learn about.

3.2.3 NGsim dataset

NGsim [19] is a traffic dataset with vehicles moving at high speeds. It contains data from four distinct freeways in Los Angeles, California. We extract only the global x and y coordinates of vehicles from two of these freeways, namely, the US-101 and the i-80 freeway. Since this dataset features very high agent density per scene (ranging in several thousands), we chunked the freeways with horizontal and vertical lines into sub-sections to restrict the number of vehicles in a sub-scene

to less than 15. We sampled about 3500 sub-scenes from the resulting chunks and set the time series length to $T = 20$ steps.

3.2.4 Charges dataset

Charges is a simulated physics dataset generated as specified by [73]. It contains data of positive and negative charges moving under other charges' electric fields and colliding with bounding walls. The coulombic forces feature dense attractive and repulsive interactions which may at times lead to the charges exhibiting very complex oscillatory behavior which is hard to model for most existing predictive models. The dataset contains 3600 scenes with time series length $T = 25$.

3.2.5 NBA dataset

The NBA [148] dataset is a sports analytics dataset with basketball player trajectories. We sampled about 7500 scenes with time series length $T = 30$. This dataset features complex goal-oriented motion heavily dictated by agents' intentions. It has been included to highlight limitations of interaction modeling approaches and to understand limitations of multi-agent trajectory prediction models.

3.3 Game Domains

Throughout this manuscript we will be using a variety of different small and large game domains to demonstrate applications of our algorithms. In this section, we describe the game domains in detail.

3.3.1 Rock-Paper-Scissors (RPS)

Rock-Paper-Scissors game is a small classical stateless, zero-sum game with two players. Each player has an action set comprising of three discrete actions: {Rock, Paper, Scissors}. Both

players simultaneously choose their actions and receive rewards as shown in Figure 3.1. This game will serve as a pedagogical example to demonstrate convergence of our algorithms to the Nash Equilibrium (NE), and get interesting insights into their behavior at times. It is well-known that the Nash Equilibrium of this game is when both players play each action with a probability $\frac{1}{3}$. In such a case, the expected utility for each player at the Nash Equilibrium is 0.

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 3.1: Rewards for Rock-Paper-Scissor Game

3.3.2 Concave-Convex game

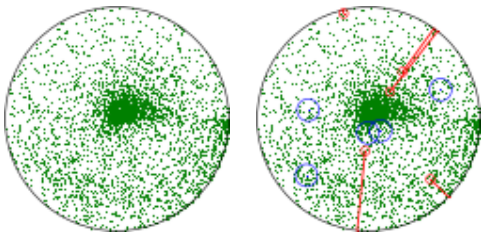
This is a small zero-sum game with continuous action spaces for the player where traditional fictitious play is known to converge. Two players 1 and 2 with scalar actions $x, y \in [-2, 2]$ respectively play to maximize their rewards: $r_1(x, y) = -2x^2 + 4xy + y^2 - 2x - 3y + 1$ and $r_2(x, y) = -r_1(x, y)$. The game is concave w.r.t. x and convex w.r.t. y and admits a pure strategy NE which can be computed using standard calculus. The NE strategies are $x = 1/3, y = 5/6$, the expected equilibrium rewards are $r_1^* = -r_2^* = -7/12$ and the best responses of players to each others' average strategies are $BR_1(\bar{y}) = \bar{y} - 1/2$ and $BR_2(\bar{x}) = 3/2 - 2\bar{x}$.

3.3.3 Cournot game

It is a classic game [28] with two competing firms (1 and 2) producing a quantity ($q_1 \geq 0$ and $q_2 \geq 0$ resp.) of a product. The price of the product is $p(q_1, q_2) = a - q_1 - q_2$ and the cost of manufacturing quantity q is $C(q) = cq$, where $c, a > 0$ are constants. Reward for a firm p is $R_p(q_1, q_2) = (a - q_1 - q_2)q_p - cq_p$, $p \in \{1, 2\}$ and the best response against the competing firm's choice can be analytically computed as q_{-p} is $BR_p(q_{-p}) = \frac{a-c-q_{-p}}{2}$. The NE strategy can be

computed as $q_1^* = q_2^* = \frac{a-c}{3}$. We use $a = 2$ and $c = 1$ for our experiments so that $q_1^* = q_2^* = 1/3$. Note that this is an example of a non zero-sum game, however traditional fictitious play is known to converge here and we use this game domain to establish the same about our extensions to fictitious play.

3.3.4 Forest Security Game



(a) (b)

Figure 3.2: (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).

We next introduce a continuous state, zero-sum security game with continuous actions spaces for both players.

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 attacker (a.k.a. adversary) has n lumberjacks to chop trees in the forests. The defender has m forest guards to ambush the trespassing lumberjacks.

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 attacker 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 color is shown in figure 3.2a (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 [63], 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 (attacker'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 $r_D = -r_O$, thereby making the game zero-sum.

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 3.2b.

3.3.5 Single-agent Areal Surveillance

A single agent, namely the defender (D), allocates m areal drones with the i^{th} drone D_i having three-dimensional coordinates $u_{D,i} = (p_{D,i}, h_{D,i}) \in [-1,1]^2 \times [0,1]$ to surveil a two-dimensional forest $Q \subset [-1,1]^2$ of arbitrary shape and with a known but arbitrary tree density $\rho(q)$. Consequently, $u_D \in \mathbb{R}^{m\times 3}$.

Each drone has a downward looking camera with a circular lens and with a half-angle θ such that at position $(p_{D,i}, h_{D,i})$, the drone D_i sees the set of points $S_{D,i} = \{q \mid ||q - p_{D,i}||_2 \leq h_{D,i} \tan\theta\}$. A visualization of this problem with $m = 2$ drones is shown for a sample forest in Figure 3.3a.

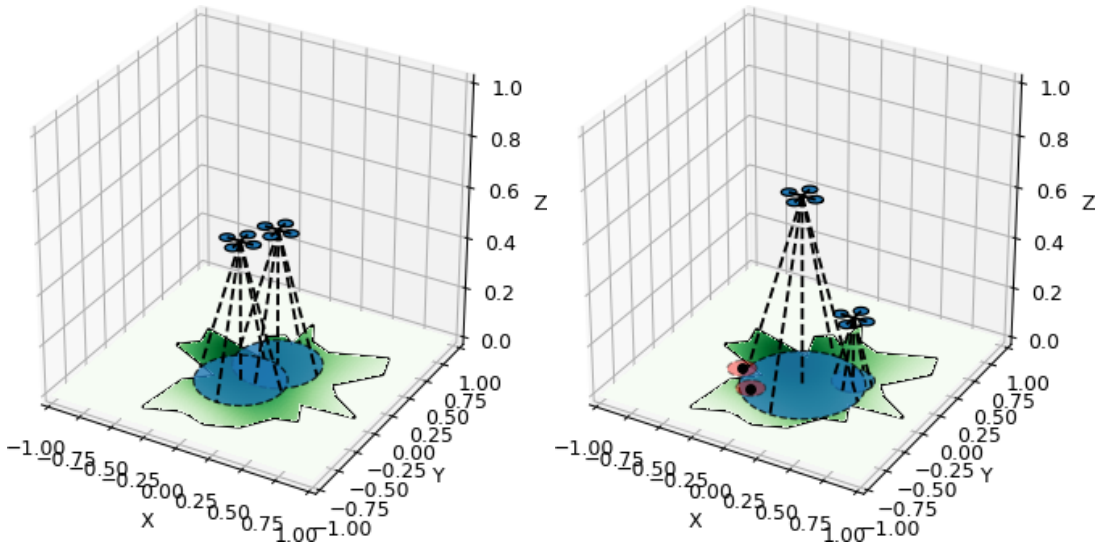
We assume a probabilistic model of coverage with a point q being covered by drone D_i with probability $P_H(h_{D,i}) = e^{K(h_{opt} - h_{D,i})} \left(\frac{h_{D,i}}{h_{opt}}\right)^{Kh_{opt}}$ if $q \in S_{D,i}$ and 0 otherwise. With multiple drones, the probability of a point q being covered can then be written as: $\text{cvg}(q, u_D) = 1 - \prod_{i|q \in S_{D,i}} \bar{P}_H(h_{D,i})$ where $\bar{P}_H$ stands for $1 - P_H$. Hence, the reward function to be maximized is: $r_{D,1p}(u_D) = \int_Q \left(1 - \prod_{i|q \in S_{D,i}} \bar{P}_H(h_{D,i})\right) \rho(q) dq$ with the tree density $\rho(q)$ being the importance of target point q (subscript $1p$ denotes one agent).

Note that in the above domain, drones provide best probabilistic coverage at a height h_{opt}. By increasing their height, a larger area can be covered at the cost of deterioration in coverage probability. Further, the defender can increase coverage probability for regions with high tree density by placing multiple drones to oversee them; in which case, the drones can potentially stay at higher altitudes too.

While the above description suffices for single player games, it can be easily extended to multi-agent games with a set of agents (or players).

3.3.6 Two-agent Adversarial Coverage

Two agents, namely the defender D and the attacker A, compete in a zero-sum game. The defender allocates m areal drones with the same coverage model as in section 3.3.5. The attacker controls n lumberjacks each with ground coordinates $u_{A,j} \in [-1, 1]^2$ to chop trees in the forest Q. Consequently, $u_A \in \mathbb{R}^{n \times 2}$. Each lumberjack chops a constant fraction κ of trees in a radius R_L around its coordinates $u_{A,j}$. We denote the area covered by the j-th lumberjack as $S_{A,j} = \{q \mid \|q - p_{A,j}\|_2 \leq R_L\}$. A visualization of this problem with $m = n = 2$ is shown for a sample forest in Figure 3.3b.



(a) (b)

Figure 3.3: (a) Areal surveillance example with an arbitrary forest and $m = 2$ drones, (b) Adversarial coverage example with $m = 2$ drones and $n = 2$ lumberjacks (red circles).

A drone can potentially catch a lumberjack if its field of view overlaps with the chopping area. For a given resource allocation $u = (u_D, u_A)$, we define $I_j = \{i \mid \|p_{A,j} - p_{D,i}\|_2 \leq R_L + h_{D,i} \tan\theta\}$ as the set of all drones which overlap with the j-th lumberjack. The areal overlap $\alpha_{ij} = \int_{S_{D,i} \cap S_{A,j}} dq$ controls the probability of the j-th lumberjack being caught by the i-th drone: $P_C(h_{D,i}, \alpha_{ij}) = P_H(h_{D,i})\, P_A(\alpha_{ij})$ where P_H is the same as that in section 3.3.5 and captures the effect of drone's height on quality of coverage, while $P_A(\alpha_{ij}) = 1 - \exp\left(-\frac{K_a \alpha_{ij}}{\pi R_L^2}\right)$ captures the effect of areal overlap on probability of being caught. Hence, the reward achieved by the j-th lumberjack can be computed as: $r_{A,j}(u_D, u_{A,j}) = \kappa \int_{S_{A,j} \cap Q} \rho(q) dq$ with probability $\prod_{i \in I_j} \bar{P}(h_{D,i}, \alpha_{ij})$, and $-\kappa \int_{S_{A,j} \cap Q} \rho(q) dq$ otherwise i.e. the number of trees chopped if the j-th lumberjack is not caught by any drone or an equivalent negative penalty if it is caught. Hence, the total agent rewards are: $r_{A,2p}(u_D, u_A) = -r_{D,2p}(u_D, u_A) = \sum_j r_{A,j}(u_D, u_{A,j})$ (subscript $2p$ denotes two-agent). Both agents are expected to compute the mixed-strategy Nash equilibria over their respective action spaces.

This domain adds additional interactions due to overlaps between defender and attacker's resources[1]. Hence, these surveillance and adversarial coverage domains form a challenging set of evaluation domains with multiple trade-offs and complex possibilities of coverage involving combinatorial interactions between the players' resources.

For both these domains, we use the following constants: $\theta = \frac{\pi}{6}$, $h_{opt} = 0.2$, $K = 4.0$, $R_L = 0.1$, $K_a = 3.0$, $\kappa = 0.1$. However, note that these values only serve as practical representative values. The techniques that we introduce in the subsequent chapters are not specific to the above probabilistic capture models or specific values of game constants, but rather apply to a broad class of coverage problems where the agents act by placing resources with finite coverage fields and agents' rewards are of the form: $r_p(u) = \int_Q f_p(u, q)dq$.

[1]In reality, lumberjacks might act independent of each other and lack knowledge of each others' plans. By allowing them to be placed via a single attacker and letting them collude, we tackle a more challenging problem and ensure that not all of them get caught by independently going to strongly covered forest regions.

Chapter 4

Multi-agent Trajectory Prediction with Fuzzy Query Attention

4.1 Introduction

Predicting trajectories of multiple agents in motion is a key challenge in many domains. The capability is especially useful for predicting paths of vehicles in traffic [141, 81], tracking pedestrians or humans in crowds [1, 9] and robotic path planning [107] (see figure 4.1). However, predicting trajectories of multiple agents is challenging because their mutual interaction complicates their behavior and leads to significant changes in their otherwise goal-oriented motion.

In order to model multi-agent settings with complex underlying interactions, several recent works based on graphs and graph neural networks have achieved significant success in prediction performance [123, 73]. But modeling interactions between two agents is challenging because it is not a binary true/false variable but is rather fuzzy[1] by nature. For instance, a person driving a car on a freeway might reason along these lines: "The car next to me is turning to switch lanes so I should also step on the brake lightly to avoid tailing the other car closely", wherein the decisions *turning*, *braking lightly* and *tailing closely* are all continuous-valued in nature (see figure 4.2). Since such fuzzy representations enter routinely into human interactions and decision

[1]We use the word *fuzzy* in this work to represent continuous-valued decisions over their discrete-valued boolean counterparts and not necessarily to fuzzy logic.

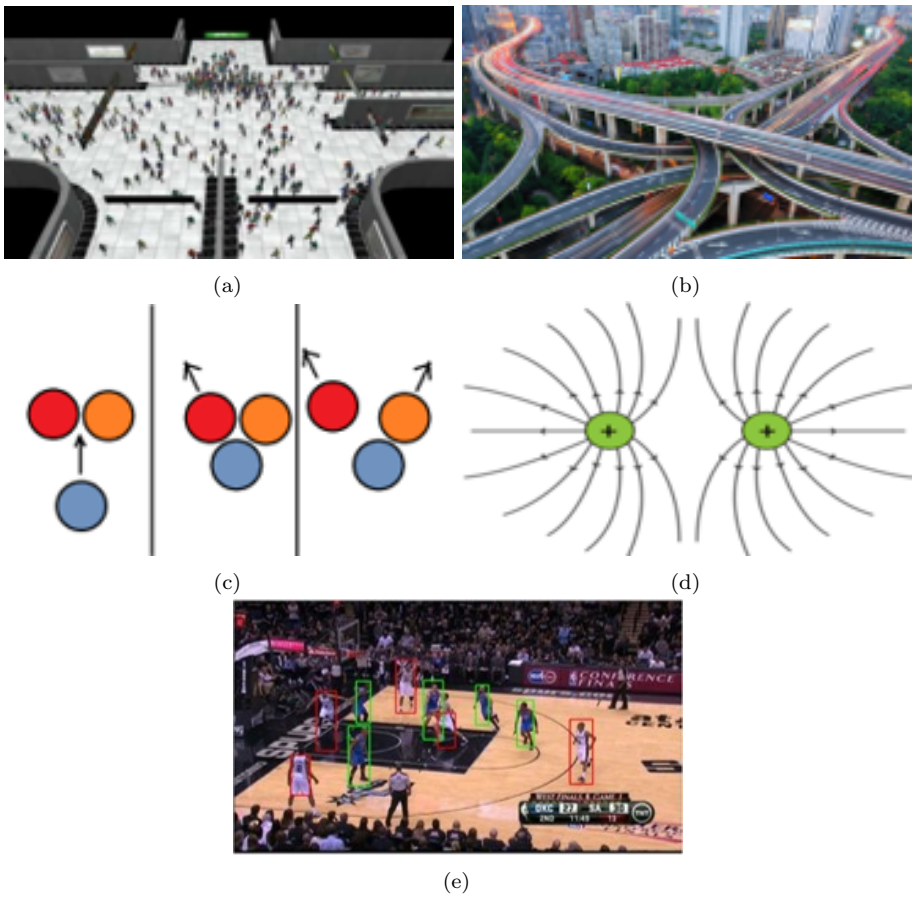


Figure 4.1: Several domains requiring multi-agent trajectory prediction: (a) Human crowds, (b) Freeway traffic, (c) Physical objects, (d) Charged particles, and (e) Sports analytics

making, we posit that learning to predict trajectories of interacting agents can benefit from fuzzy (continuous-valued) decision making capabilities.

In this chapter we present a general architecture to address the problem of multi-agent trajectory prediction by modeling the crucial inductive biases of motion, namely, inertia, relative motion, intents and interactions. Specifically, we propose a relational model to flexibly model interactions between agents in diverse environments with a novel Fuzzy Query Attention (FQA) mechanism to solve the aforementioned challenge. FQA models pairwise attention to decide about when two agents are interacting by learning keys and queries which are combined with a dot-product structure to make continuous-valued (fuzzy) decisions. It also simultaneously learns how the agent under focus is affected by the influencing agent given the fuzzy decisions. We demonstrate

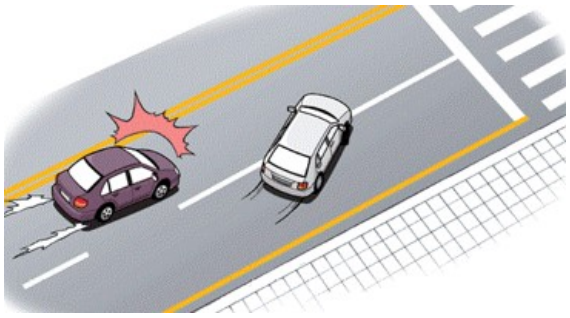


Figure 4.2: Humans exhibit fuzzy decision making routinely

significant performance gains over existing state-of-the-art predictive models in several domains: (a) trajectories of human crowd, (b) US freeway traffic, (c) object motion and collisions governed by Newtonian mechanics, (d) motion of charged particles under electrostatic fields, and (e) basketball player trajectories, thereby showing that FQA can learn to model very diverse kinds of interactions. Our experiments show that the fuzzy decisions made over time are highly predictive of interactions even when all other input features are ignored. Our architecture also supports adding human knowledge in the form of fuzzy decisions, which can provide further gains in prediction performance.

4.2 Fuzzy Query Attention model

4.2.1 Problem Formulation

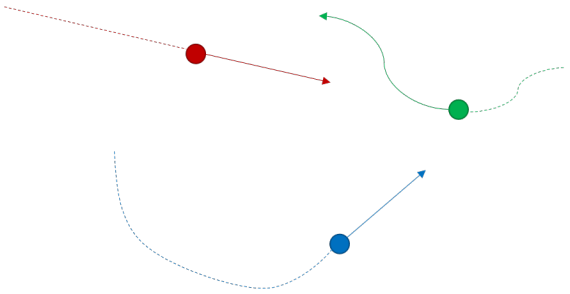


Figure 4.3: Multi-agent trajectory prediction problem setup

Following previous work [1, 73], we assume a given scene which has been pre-processed to obtain the spatial coordinates $p_i^t = (x_i^t, y_i^t)$ of all agents $i \in 1 : N$ at a sequence of time-steps

$t \in 1:T$. The task is to observe all agents from time 1 to T_{obs}, infer their motion characteristics and ongoing interactions and predict their positions for time-steps $T_{obs}+1$ to T (see figure 4.3). In all subsequent text, $p^t = \{p_1^t, p_2^t, \ldots, p_N^t\}$ represents the set of positions of all agents at time t, while $p_i = [p_i^1, p_i^2, \ldots, p_i^T]$ represents the sequence of positions of a single agent i at all time-steps. v is used to denote velocity, tilde symbol ($\tilde{\cdot}$) on the top to denote intermediate variables and hat symbol ($\hat{\cdot}$) on the top for predicted quantities or unit vectors (will be clear from context).

4.2.2 Design Principles

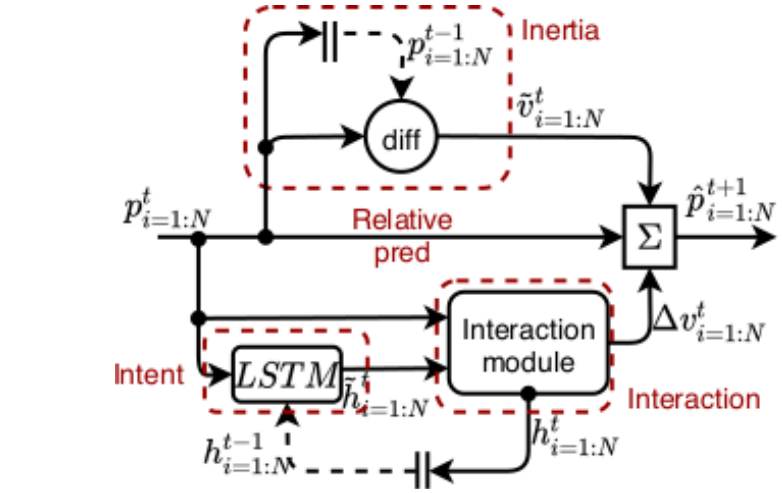
Our architecture incorporates the following crucial inductive biases required for motion prediction:

- **Inertia**: Most inanimate entities move with constant velocity until acted upon by external forces. This also acts as a good first-order approximation for animate agents for short time-intervals, e.g., pedestrians walk with nearly constant velocities unless they need to turn or slow down to avoid collisions.

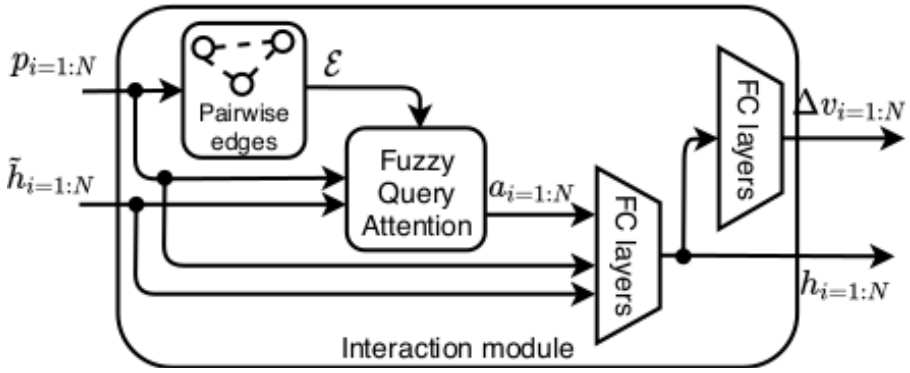
- **Motion is relative**: Since motion between two agents is relative, one should use agents' relative positions and velocities while predicting future trajectories (*relative observations*) and should further make predictions as offsets relative to the agents' current positions (*relative predictions*).

- **Intent**: Unlike inanimate entities, animate agents have their own intentions which can cause deviations from inertia and need to be accounted for in a predictive model.

- **Interactions**: Both inanimate and animate agents can deviate from their intended motion due to influence by other agents around them and such interaction needs to be explicitly modeled.



(a) Overall prediction architecture



(b) Interaction module

Figure 4.4: Multi-agent prediction architecture using Fuzzy Query Attention at time t: (a) Overall architecture takes positions (p) of all agents, computes a first-order estimate of velocity ($\tilde{v}$) and incorporates effects of interactions between agents via a correction term (Δv) thereby predicting the positions at the next time-step ($\hat{p}^{t+1}$); (b) the Interaction module generates pairwise edges between agents ($\mathcal{E}$) and uses the FQA module to account for interactions and generate the aggregate effect (a) for each agent which is used to update their LSTM state (h) and predict the velocity correction (Δv).

4.2.3 Prediction Architecture

The overall prediction architecture (Figure 4.4a) takes the spatial positions of all agents i.e. $p^t_{i=1:N}$ as input at time t. We use the observed positions for $t \leq T_{obs}$ and the architecture's own predictions from the previous time-step for $t > T_{obs}$. We predict each agent's position at the next time-step $\hat{p}^{t+1}_i$ as an offset from its current position p^t_i to capture the *relative prediction* inductive bias. We further break each offset into a first-order constant velocity estimate $\tilde{v}^t_i$ which accounts for the *inertia* inductive bias and a velocity correction term Δv^t_i which captures agents' *intents* and

inter-agent *interactions* (see eq 4.1). The first-order estimate of velocity ($\tilde{v}_i^t$) is made by a direct difference of agents' positions from consecutive time steps (eq 4.2). To capture agents' intents, an LSTM module is used to maintain the hidden state (h_i^{t-1}) containing the past trajectory information for the i^{th} agent. The learnable weights of the LSTM are shared by all agents. To compute the correction term (Δv_i^t), a preliminary update is first made to the LSTM's hidden state using the incoming observation for each agent. This preliminary update captures the deviations from inertia due to an agent's own intentional acceleration or retardation (eq 4.3). The intermediate hidden states $\tilde{h}_i^t$ and the current positions of all agents are further used to infer the ongoing interactions between agents, aggregate their effects and update the hidden state of each agent to h_i^t while also computing the correction term for the agent's velocity via an interaction module (eq 4.4).

$$\hat{p}_i^{t+1} = p_i^t + \tilde{v}_i^t + \Delta v_i^t, \qquad \forall i \in 1:N \tag{4.1}$$

$$\text{(Inertia): } \tilde{v}_i^t = p_i^t - p_i^{t-1}, \quad \forall i \in 1:N \tag{4.2}$$

$$\text{(Agent's Intents): } \tilde{h}_i^t = \text{LSTM}(p_i^t, h_i^{t-1}), \ \forall i \in 1:N \tag{4.3}$$

$$\text{(Interactions): } h^t, \Delta v^t = \text{InteractionModule}(p^t, \tilde{h}^t) \tag{4.4}$$

Since computation in all sub-modules happens at time t, we drop the superscript t from here on.

4.2.4 Interaction Module

The interaction module (Figure 4.4b) first creates a graph by generating directed edges between all pairs of agents (ignoring self-edges)[2]. The edge set $\mathcal{E}$, the positions and the states of all agents are used to compute an attention vector a_i for each agent aggregating all its interactions with other agents via the Fuzzy Query Attention (FQA) module (eq 4.5). This aggregated attention along with each agent's current position and intermediate hidden state is processed by subsequent

[2]We also show experiments with edges based on distance-based cutoffs as previous work [17] has found this heuristic useful for trajectory prediction.

fully-connected layers to generate the updated state h_i (which is fed back into the LSTM) and the velocity correction Δv_i for each agent (eqs 4.6 and 4.7).

$$a = \text{FQA}(p, \tilde{h}, \mathcal{E}) \tag{4.5}$$

$$h_i = FC_2(ReLU(FC_1(p_i, h_i, a_i))), \quad \forall i \in 1:N \tag{4.6}$$

$$\Delta v_i = FC_4(ReLU(FC_3(h_i))), \quad \forall i \in 1:N \tag{4.7}$$

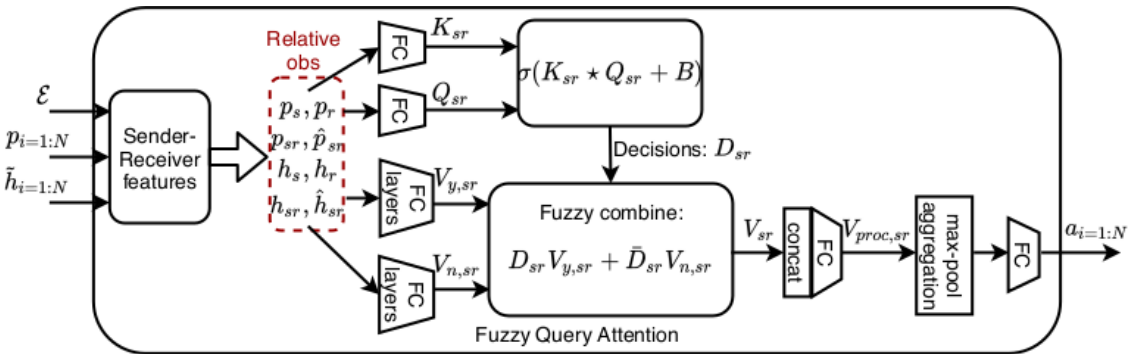


Figure 4.5: FQA module generates keys (K_{sr}), queries (Q_{sr}) and responses ($V_{y,sr}, V_{n,sr}$) from sender-receiver features between agent pairs, combines the responses according to the fuzzy decisions (D_{sr}), and aggregates the concatenated responses into a vector (a) per agent.

4.2.5 Fuzzy Query Attention

The FQA module views the graph edges as sender-receiver ($s - r$) pairs of agents. At a high level, it models the aggregate effect of the influence from all sender agents onto a specific receiver agent (Figure 4.5). To do so, we build upon the key-query-value based self-attention networks introduced by Vaswani *et al.* [130]. FQA first generates independent features: p_s, p_r, h_s and h_r for the senders and receivers by replicating p and h along each edge. It also generates relative features: $p_{sr} = p_s - p_r$ (relative displacement), $h_{sr} = h_s - h_r$ (relative state), $\hat{p}_{sr} = p_{sr}/\|p_{sr}\|$ (unit-vector along p_{sr}) and $\hat{h}_{sr} = h_{sr}/\|h_{sr}\|$ (unit-vector along h_{sr}) to capture the *relative observations* inductive bias. These features $f_{sr} = \{p_s, p_r, p_{sr}, \hat{p}_{sr}, h_s, h_r, h_{sr}, \hat{h}_{sr}\}$ are combined by single fully-connected layers to generate n keys $K_{sr} \in \mathbb{R}^{n \times d}$ and queries $Q_{sr} \in \mathbb{R}^{n \times d}$ of dimension d each for every $s - r$ pair

(eqs 4.8 and 4.9), which are then combined via a variant of dot-product attention to generate fuzzy[3] decisions $D_{sr} \in \mathbb{R}^n$ (eq 4.10):

$$K_{sr} = FC_5(f_{sr}^{\perp}), \qquad \forall (s,r) \in 1:N, s \neq r \tag{4.8}$$

$$Q_{sr} = FC_6(f_{sr}^{\perp}), \qquad \forall (s,r) \in 1:N, s \neq r \tag{4.9}$$

$$D_{sr} = \sigma(K_{sr} \star Q_{sr} + B) = \sigma\left(\sum_{dim=1} K_{sr} \odot Q_{sr} + B\right), \qquad \forall (s,r) \in 1:N, s \neq r \tag{4.10}$$

where $\odot$ represents element-wise product, $B \in \mathbb{R}^n$ is a learnable bias parameter, σ stands for the sigmoid activation function and $\perp$ stands for the detach operator[4]. As a consequence of this formulation, $D_{sr} \in [0,1]^n$ can be interpreted as a set of n continuous-valued decisions capturing the interaction between agents s and r. These can now be used to select the receiving agent's response to the current state of the sending agent. For this, the sender-receiver features are parsed in parallel by two-layer neural networks (with the first layer having a ReLU activation) to generate yes-no responses $V_{y,sr}, V_{n,sr} \in \mathbb{R}^{n \times d_v}$ corresponding to D_{sr} being 1 (*yes*) or 0 (*no*) respectively (eqs 4.11 and 4.12). Though all the $s - r$ features can be used here, our preliminary experiments showed that including only a subset of features (h_s and p_{sr}) gave comparable results and led to considerable saving in the number of parameters, so we only use this subset of features to generate the yes-no responses. These responses are then combined using a *fuzzy if-else* according

[3]Note that the word *fuzzy* represents continuous-valued decisions over their discrete-valued boolean counterparts and not fuzzy logic.

[4]The detach operator acts as identity for the forward-pass but prevents any gradients from propagating back through its operand. This allows us to learn feature representations only using responses while the keys and queries make useful decisions from the learnt features.

to decisions D_{sr} and their complements $\bar{D}_{sr} = 1 - D_{sr}$ to generate the final responses $V_{sr} \in \mathbb{R}^{n \times d_v}$ (eq 4.13):

$$V_{y,sr} = FC_8(ReLU(FC_7(p_{sr}, h_s))), \qquad \forall (s,r) \in 1:N, s \neq r \tag{4.11}$$

$$V_{n,sr} = FC_{10}(ReLU(FC_9(p_{sr}, h_s))), \qquad \forall (s,r) \in 1:N, s \neq r \tag{4.12}$$

$$\text{(Fuzzy if-else): } V_{sr} = D_{sr} V_{y,sr} + \bar{D}_{sr} V_{n,sr}, \qquad \forall (s,r) \in 1:N, s \neq r \tag{4.13}$$

The n final responses generated per agent pair ($\in \mathbb{R}^{n \times d_v}$) are then concatenated ($\in \mathbb{R}^{n d_v}$) and final responses from all senders are aggregated on the respected receivers by dimension-wise max-pooling to accumulate effect of all interactions on the receiver agents (eqs 4.14 and 4.15). Since max-pooling loses information while aggregating, we pre-process the final responses to increase the dimensions and retain more information followed by subsequent post-processing after aggregation to reduce the number of dimensions again (eqs 4.14 and 4.16):

$$V_{proc,sr} = FC_{11}(\text{concat}(V_{sr})) \tag{4.14}$$

$$V_{proc,r} = \text{maxpool}_{s:(s-r)\in\mathcal{E}} V_{proc,sr} \tag{4.15}$$

$$a_r = FC_{12}(V_{proc,r}), \qquad \forall r \in 1:N. \tag{4.16}$$

4.2.6 Strengths of FQA

While originally motivated from multi-head self-attention [130], FQA differs from it significantly in many aspects. Firstly, FQA generalizes self-attention to pairwise-attention which attends to an ordered pair (sender-receiver) of entities and captures the interaction effects of the sender on the receiver. This allows application to multi-agent settings. Secondly, FQA has a learnable bias B to improve modeling power (explained below). Further, though the original matrix-dot-product structure of self-attention requires a large memory to fit even for regular batch sizes e.g. 32,

our simpler row-wise dot-product structure fits easily on a single GPU (12GB) for all datasets, while still retaining the strong performance of the dot-product attention structure. Moreover, we learn the sender-receiver features by backpropagating only through the responses (V_{sr}) while features are detached to generate the keys and queries. This additionally allows us to inject human knowledge into the model via handcrafted non-learnable decisions, if such decisions are available (see experiments in section 4.3.4).

What kinds of decisions can FQA learn?: Since keys and queries are linear in the senders' and receivers' states and positions, the decision space of FQA contains many intuitive decisions important for trajectory prediction, e.g.:

1. *Proximity*: FQA can potentially learn a key-query pair to be p_{sr} each and the corresponding bias as $-d_{th}^2$, then the decision $D = \sigma(p_{sr}^T p_{sr} - d_{th}^2)$ going to zero reflects if agents s and r are closer than distance d_{th}. Note that such decisions would not be possible without the learnable bias parameter B, hence having the bias makes FQA more flexible.

2. *Approach*: Since a part of the state h_i can learn to model velocity of agents v_i internally, FQA can potentially learn a key-query pair of the form $K_{sr} = v_{sr}, Q_{sr} = \hat{p}_{sr}, B = 0$ to model $D = \sigma(v_{sr}^T \hat{p}_{sr} + 0)$ which tends to 0 when the agents are directly approaching each other. While we do not force FQA to learn such human-interpretable decisions, our experiments show that the fuzzy decisions learnt by FQA are highly predictive of interactions between agents (section 4.3.4).

4.2.7 Training

FQA and all our other baselines are trained to minimize the mean-square error in predicting next time-step positions of all agents. Since some datasets involve agents entering and exiting the scene freely between frames, we input binary masks to all models for each agent to determine the presence of agents in the current frame and control updates for agents accordingly (masks

not shown in figures to avoid clutter). All models are trained with the Adam optimizer [72] with batch size 32 and an initial learning rate of 0.001 decaying multiplicatively by a factor $\gamma = 0.8$ every 5 epochs. All models train for at least 50 epochs after which early stopping is enabled with a max patience of 10 epochs on validation set mean-square error and training is terminated at a maximum of 100 epochs. Since we test the models by observing T_{obs} (kept at $\frac{2T}{5}$ for all datasets) time-steps and make predictions until the remaining time T, we followed a dynamic schedule allowing all models to see the real observations for T_{temp} time-steps followed by $T - T_{temp}$ of its own last time-step predictions. During training, T_{temp} is initialized to T and linearly decayed by 1 every epoch until it becomes equal to T_{obs}. We found this dynamic burn-in schedule employed during training to improve the prediction performance for all models.

4.3 Experiments

We perform multi-agent trajectory prediction on different datasets used previously in the literature with a diverse variety of interaction characteristics[5]. For datasets with no provided splits, we follow a $70 : 15 : 15$ split for training, validation and test set scenes. We used the following datasets (detailed descriptions available in section 3.2):

1. **ETH-UCY** [9]: A human crowds dataset with medium interaction density. We sampled about 3400 scenes at random from the dataset and set $T = 20$ following prior work [1, 40].

2. **Collisions**: Synthetic physics data with balls moving on a friction-less 2D plane, fixed circular landmarks and boundary walls. The collisions between balls preserve momentum and energy, while collisions of agents with walls or immobile landmarks only preserve energy but not momentum of moving agents. Contains about 9500 scenes with $T = 25$.

3. **NGsim** [19]: US-101 and i-80 freeway traffic data with fast moving vehicles. Since this dataset features very high agent density per scene (ranging in several thousands), we chunked the

[5]Code for implementing FQA can be found at `https://github.com/nitinkamra1992/FQA.git`

freeways with horizontal and vertical lines into sub-sections to restrict the number of vehicles in a sub-scene to less than 15. We sampled about 3500 sub-scenes from the resulting chunks and set $T = 20$.

4. **Charges** [73]: Physics data with positive and negative charges moving under other charges' electric fields and colliding with bounding walls. Contains 3600 scenes with $T = 25$ involving dense attractive and repulsive interactions.

5. **NBA** [148]: Sports dataset with basketball player trajectories. We sampled about 7500 scenes with $T = 30$. This dataset features complex goal-oriented motion heavily dictated by agents' intentions. It has been included to highlight limitations of interaction modeling approaches.

4.3.1 Baselines

We compare our FQA architecture with state-of-the-art baselines:

1. **Vanilla LSTM** [VLSTM]: An LSTM preceeded and followed by fully-connected neural network layers is used to predict the offset without considering interactions.

2. **Social LSTM** [SLSTM] [1]: Recurrent architecture which models interactions by discretizing space around each agent and aggregating neighbors' latent states via a social pooling mechanism.

3. **GraphSAGE** [GSAGE] [44]: Graph neural networks with node features to model interactions between agents. We use feature-wise max-pooling for aggregating the messages along the edges.

4. **Graph Networks** [GN] [8, 123]: Graph neural networks with node features, edge features and global features to model interactions between agents. We adapt the Encoder→RecurrentGN→ Decoder architecture from [123].

5. **Neural Relational Inference** [NRI] [73]: Uses graph neural networks to model interactions between agents and additionally infers edges between agents using variational inference.

6. **Graph Attention Networks** [GAT] [131]: Follows an aggregation style similar to GraphSAGE, but weighs messages passed from all sender agents via a learnt attention mechanism.

We provide the model architectures and hyperparameters of our baselines and those of FQA in this section. All our experiments were done on systems with Ubuntu 16.04 and all models trained using either Nvidia Titan X or Nvidia GeForce GTX 1080 Ti GPUs. All code was written in Python 3.6 with neural network architectures defined and trained using PyTorch v1.0.0.

4.3.1.1 Vanilla LSTM

The Vanilla LSTM model embeds each p_i^t to a 32-dimensional embedding vector using a fully-connected layer with ReLU activation. This vector is fed along with the previous hidden states to an LSTM with state size 64, whose output is again processed by a fully-connected layer to generate the 2-dimensional offset for next-step prediction.

4.3.1.2 Social LSTM

We adapted the code from `https://github.com/quancore/social-lstm` which directly reproduces the original authors' model from [1]. We kept the initial embedding size as 20, the LSTM's hiddden size as 40, the size of the discretization grid as 4 and the discretization neighborhood size as 0.5[6].

4.3.1.3 Neural Relational Inference

We adapted the authors' official repository from `https://github.com/ethanfetaya/NRI`. The input dimension was kept as 2 for positional coordinates and the number of edge types as 3 (since setting it to 2 gave worse results). The encoder employed the MLP architecture with hidden layers of sizes 32 and no dropout, while the GRU-based RNN architecture was used for the decoder

[6]This neighborhood size is also the same as the distance cutoff used in section 4.3.4.

with hidden state of size 32 and no dropout. The variance of the output distribution was set to 5×10^{-5}.

4.3.1.4 Graph Networks

While the original repository for Graph Networks is written in TensorFlow (`https://github.com/deepmind/graph_nets`), we translated the repository into PyTorch and adapted models similar to those employed by [123, 8]. We employed a vertex-level encoder followed by a recurrent Graph Network based on GRU-style recurrence followed by a Graph Net decoder. The vertex-level encoder transforms 2-dimensional positional input at each time step to a 10-dimensional node embedding. An input graph is constructed from these node embeddings having all pairwise edges and dimensions $10, 1$ and 1 respectively for the node, edge and global attributes. This input graph along with a previous state graph (with dimensions $45, 8$ and 8 for node, edge and global state attributes) was processed using a GRU-style recurrent Graph Network to output the updated state graph of the same dimensions ($45, 8$ and 8 for node, edge and global state attributes respectively). This new state graph was processed by a feedforward graph-network as prescribed in [8] to output another graph whose node features of dimensions 2 were treated as offsets for the next time step prediction. All update networks both in the encoder and the decoder (for node, edge and global features) used two feedforward layers with the intermediate layer having latent dimension 32 and a ReLU activation. While the original work proposes to use sum as the aggregation operator, we found summing to often cause the training to divergence since different agents have neighborhoods of very diverse sizes ranging from 0 to about 40 at different times in many of our datasets. Hence we used feature-wise mean-pooling for all aggregation operators.

4.3.1.5 GraphSAGE, Graph Attention Networks and Fuzzy Query Attention

Since GraphSAGE (GSAGE) [44] and Graph Attention Networks (GAT) [131] were not originally prescribed for a multi-agent trajectory prediction application, we used their update and aggregation

styles in our own FQA framework to replace the FQA sub-module in our Interaction module described in Section 4.2. For all three methods the input size and the output size was 2, while the hidden state dimension of the LSTM shared by all agents was 32. The dimension of the aggregated attention for each agent a_i^t was also set to 32 for all three methods. All the three methods involved the FC_1, FC_2, FC_3 and FC_4 layers described in section 4.2 and had the output sizes $48, 32, 16$ and 2 respectively.

GSAGE: GraphSAGE [44] directly embeds all sender latent vectors h_s into 32-dimensional embeddings via two fully-connected layers each with a RELU activation and with the intermediate layer of dimensions 32. The output embeddings were aggregated into the receiver nodes via feature-wise max-pooling to generate a_i^t.

GAT: GAT performs a similar embedding of sender hidden states using a similar embedding network as GSAGE but aggregates them via feature-wise max-pooling after weighing the embeddings with 8 attention head coefficients generated as proposed in [131] and finally averages over the 8 aggregations. We used 8 attention heads to match the number of FQA's decisions.

FQA: FQA used 8 query-key pairs for all datasets leading to 8 decisions. The dimension for keys and queries was set to 4, while the dimension for yes-no responses was kept as 6. Consequently the dimension of learnt bias vector B was also 8 and the sizes of the fully-connected layers $FC_5, FC_6, FC_7, FC_8, FC_9, FC_{10}, FC_{11}$ and FC_{12} were $32, 32, 33, 48, 33, 48, 32$ and 32 respectively.

4.3.2 Prediction results

Table 4.1: Prediction error metrics for all methods on all datasets

Model	ETH-UCY	Collisions	NGsim	Charges	NBA
VLSTM	0.576 ± 0.002	0.245 ± 0.001	5.972 ± 0.065	0.533 ± 0.001	6.377 ± 0.053
SLSTM	0.690 ± 0.013	0.211 ± 0.002	6.453 ± 0.153	0.485 ± 0.005	6.246 ± 0.048
NRI	0.778 ± 0.027	0.254 ± 0.002	7.491 ± 0.737	0.557 ± 0.008	5.919 ± 0.022
GN	0.577 ± 0.014	0.234 ± 0.001	5.901 ± 0.238	0.508 ± 0.006	5.568 ± 0.032
GSAGE	0.590 ± 0.011	0.238 ± 0.001	5.582 ± 0.082	0.522 ± 0.002	5.657 ± 0.018
GAT	0.575 ± 0.007	0.237 ± 0.001	6.100 ± 0.063	0.524 ± 0.004	6.166 ± 0.052
FQA (ours)	**0.540 ± 0.006**	**0.176 ± 0.004**	**5.071 ± 0.186**	**0.409 ± 0.019**	**5.449 ± 0.039**

For all models, we report the Root Mean Square Error (RMSE) between ground truth and our predictions over all predicted time steps for all agents on the test set of every dataset in Table 4.1. The standard deviation is computed on the test set RMSE over five independent training runs differing only in their initial random seed. Our model with $n = 8$ decisions outperforms all the state-of-the-art baselines on all benchmark datasets (on many by significant margins). This shows that FQA can accurately model diverse kinds of interactions. Specifically, we observe that all models find it difficult to model sparse interactions on the Collisions data, while FQA performs significantly better with lower errors presumably due to its fuzzy decisions being strongly predictive of when two agents are interacting (more detail in section 4.3.4). Further, though GAT also uses an attention mechanism at the receiver agents to aggregate messages, FQA outperforms GAT on all datasets showing a stronger inductive bias towards modeling multi-agent interactions for trajectory prediction.

As a side note, we point out that SLSTM [1] and NRI [73] both of which model interactions are often outperformed by VLSTM which does not model interactions. While surprising at first, we found that this has also been confirmed for SLSTM by prior works, namely, Social GAN [40] which has common co-authors with SLSTM, and also independently by the TrajNet Benchmark paper [9]. We believe that this is because both methods introduce significant noise in the neighborhood of agents: (a) SLSTM does this by aggregating agents' hidden states within discretized bins which can potentially lose significant motion specific information, and (b) NRI infers many spurious edges during variational edge-type inference (also shown by [84]).

4.3.3 Ablations

To show that it is indeed the fuzzy decisions attention mechanism which lends our model its strength, we present several ablations of our model.

Modeling only inertia: We first remove the velocity correction term (Δv_i^t) and only retain the constant velocity estimate (inertia) to show that both intention and interaction modeling are

indeed required for accurate prediction. We call this model FQA$_{inert}$ and Table 4.2 shows the stark deterioration in performance after the removal of velocity correction term.

Modeling only inertia and agent intention: We next drop only the interaction module by setting all attention vectors $a_{i=1:N}$ to 0, while keeping the constant velocity estimate and the intentional motion LSTM (eqs 4.2,4.3) intact. The resulting RMSEs shown as FQA$_{NoIntr}$ in Table 4.2 capture the severe drop in performance on all datasets, thereby showing that a major chunk of improvement indeed comes from modeling the interactions.

Removing decision making of FQA: To demonstrate that the strength of the interaction module comes from FQA's decision making process, we next replaced all sub-modules between the inputs of the FQA module uptil V_{sr} in figure 4.5 with fully-connected layers with equivalent number of learnable parameters so that responses V_{sr} are directly produced from input features without any fuzzy decisions. We call this variant FQA$_{NoDec}$ and show the deterioration in performance from loss of decision making in Table 4.2. It is clear that while FQA$_{NoDec}$ outperforms FQA$_{inert}$ and FQA$_{NoIntr}$ because it models interactions with at least a simple neural network, substituting the decision making mechanism has reduced FQA to the same or worse level of performance as other baselines on most benchmark datasets.

Table 4.2: Prediction error metrics with ablations and augmentations

Model	ETH-UCY	Collisions	NGsim	Charges	NBA
FQA$_{inert}$	0.576 ± 0.000	0.519 ± 0.000	6.159 ± 0.000	0.778 ± 0.000	13.60 ± 0.000
FQA$_{NoIntr}$	0.549 ± 0.006	0.236 ± 0.0003	5.756 ± 0.152	0.523 ± 0.001	6.038 ± 0.044
FQA$_{NoDec}$	0.539 ± 0.006	0.234 ± 0.001	5.616 ± 0.163	0.505 ± 0.007	5.518 ± 0.049
GN$_{dce}$	0.572 ± 0.020	0.227 ± 0.002	**5.714 ± 0.155**	0.451 ± 0.004	**5.553 ± 0.010**
GSAGE$_{dce}$	0.579 ± 0.011	0.231 ± 0.001	5.901 ± 0.099	0.456 ± 0.005	5.898 ± 0.048
GAT$_{dce}$	0.571 ± 0.006	0.232 ± 0.001	5.936 ± 0.124	0.460 ± 0.008	5.938 ± 0.021
FQA$_{dce}$	**0.532 ± 0.002**	**0.175 ± 0.004**	5.814 ± 0.170	**0.416 ± 0.001**	5.733 ± 0.033
FQA$_{hk}$	0.541 ± 0.002	0.177 ± 0.006	4.801 ± 0.215	0.396 ± 0.007	5.457 ± 0.084

4.3.4 Understanding fuzzy decisions of FQA

Distance-based cutoff for edges: To check if FQA can learn decisions to reflect proximity between agents, we replaced our edge generator to produce edges with a distance-based cutoff so it outputs a directed edge between agents s and r only if $\|p_s^t - p_r^t\|_2 \leq d_{thresh}$. The threshold d_{thresh} was found by a crude hyperparameter search and was set to $d_{thresh} = 0.5$ in the normalized coordinates provided to all models. We show prediction errors for FQA and other baselines namely GN, GSAGE and GAT[7] by providing them distance-constrained edges instead of all edges (*dce* variants) in Table 4.2. While *dce* variants of baselines show improvement in prediction errors on most datasets, FQA only shows minor improvements on Collisions which has sparse density of interactions, while the performance degrades on the other datasets with dense interactions. This suggests that FQA is indeed able to model proximity between agents even from a fully-connected graph, if the dataset is sufficiently dense in the number of interactions per time-step and does not require aiding heuristics, while other baselines do not necessarily extract this information and hence benefit from the heuristic.

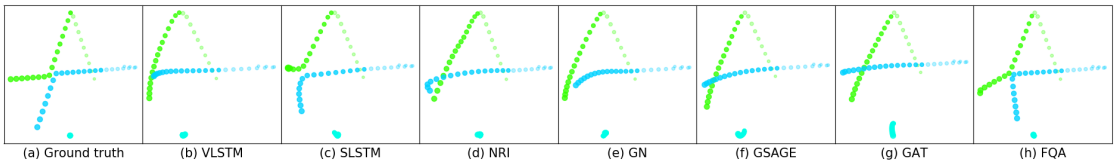
Table 4.3: Predict collisions from FQA decisions

τ	1	2	3	Recurrent
Accuracy	95.55%	95.48%	95.35%	95.75%
AUROC	0.854	0.866	0.870	0.907

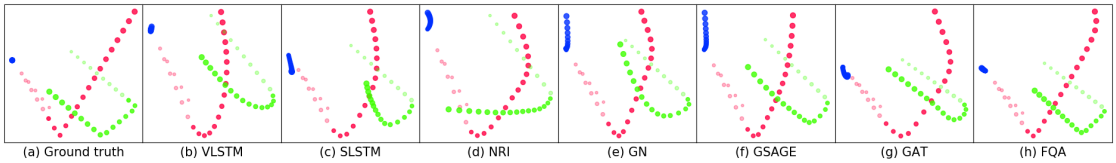
Predicting interactions from decisions: To investigate if the decisions capture inter-agent interactions well, we present an experiment to predict when a collision happens between two agents on the Collisions dataset[8] from only the 8 agent-pair decisions D_{sr}^t. Since collisions are sparse, we present the prediction accuracy and the area under the ROC curve on a held-out test set in Table 4.3 for various classifiers trained to predict collisions between agents using different horizon of time-steps (τ) of the input decisions. Note that we do not even use the agents' positions,

[7]SLSTM already uses a neighborhood size of 0.5 for discretization, while NRI infers edges internally via variational inference.

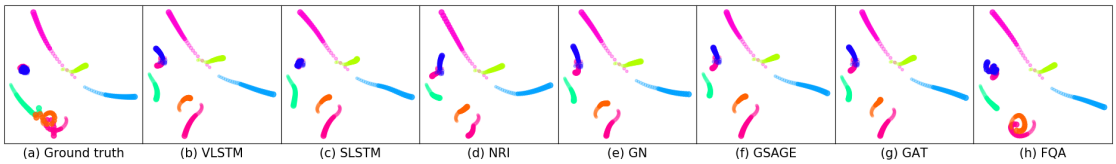
[8]This is the only synthetic dataset for which the ground truth of interactions is available.



(a) Collisions data: FQA models sparse interactions like inter-agent collisions well.



(b) Collsions data: FQA models stationary fixed landmarks well (blue) and predicts sharp collisions with walls.



(c) Charges data: Complex swirling in opposite charges (see pink and orange trajectories) accompanied by high accelerations; No model except FQA is able to predict such complex motion.

Figure 4.6: Predicted trajectories from all models shown with circles of radii increasing with time. The lighter shades show the observed part uptil T_{obs} while the darker shades show the predictions till T.

velocities or the FQA responses (V_{sr}) as inputs to the predictors. Yet, the decision-trajectories alone are sufficient to predict collisions with a surprisingly high accuracy and AUROC, which strongly indicates that FQA's decisions are accurately capturing inter-agent interactions.

Including human-knowledge in FQA: Next we show that one can also add fuzzy decisions to FQA, which are intuitive for humans but might be hard to infer from data. To this end, we add an additional fixed decision $D = \sigma(\tilde{v}_{sr}^T \hat{p}_{sr})$ to FQA which should tend to 0 (no) when two agents are directly approaching each other, while leaving the corresponding yes-no responses learnable (we call this FQA$_{hk}$). While Table 4.2 shows no significant improvement on most datasets, presumably since the information captured by this decision is already being captured by the model, we do observe a significant decrease in RMSE on the NGsim dataset compared to Table 4.1. This is because our chunking procedure on NGsim eliminates a few neighbors of the agents at sub-scene boundaries and consequently certain interaction effects become harder to capture from data. So adding this human-knowledge directly as a decision improves performance. Hence, FQA allows

the designer to augment the model with human-knowledge decisions as hints, which can improve performance and are ignored if not useful.

Visualization: Next we visualize the trajectories predicted by FQA and other baselines. Figures 4.6a and 4.6b show inter-agent collisions and those between agents and boundaries respectively. Due to agents' small sizes, inter-agent collisions are sparse events and only FQA learns to model them appropriately while the other baselines ignore them. Further FQA models the trajectories of agents faithfully and all collisions sharply while other baselines sometimes predict curved trajectories and premature soft collisions in empty space without any real interaction. We further observe from the pink and orange charges in Figure 4.6c, that it is hard to model chaotic swirling of nearby opposite charges due to high accelerations resulting from coulombic forces and that FQA comes closest to being an accurate model.

Next we show additional visualization from all models on all datasets (other than NBA). The visualizations clearly demonstrate the strong inductive bias of FQA for multi-agent trajectory prediction.

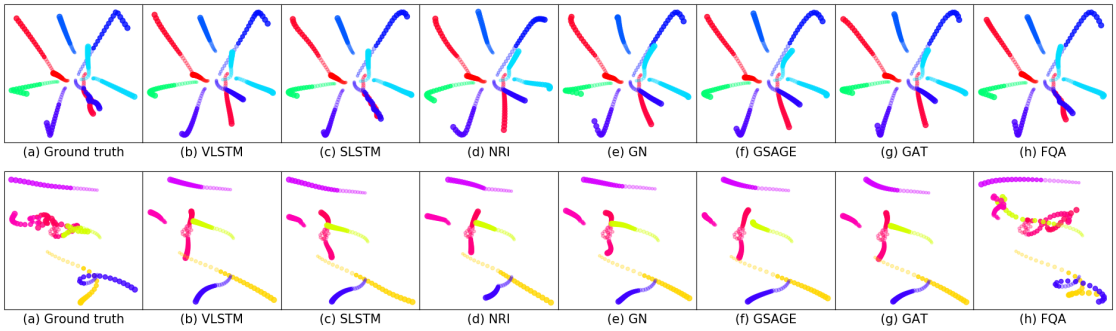


Figure 4.7: Predicted trajectory visualization from various models on Charges dataset.

Limitations: Finally, we point out that FQA (and all baselines) have a high RMSE on the NBA dataset (w.r.t. the relative scale of values in the dataset). This is because the NBA dataset comprises of many sudden intent dependent events or otherwise motions with many valid alternatives that cannot be predicted in the long term[9].

[9]Note that FQA is still the most accurate trajectory predictor amongst our baselines on the NBA dataset.

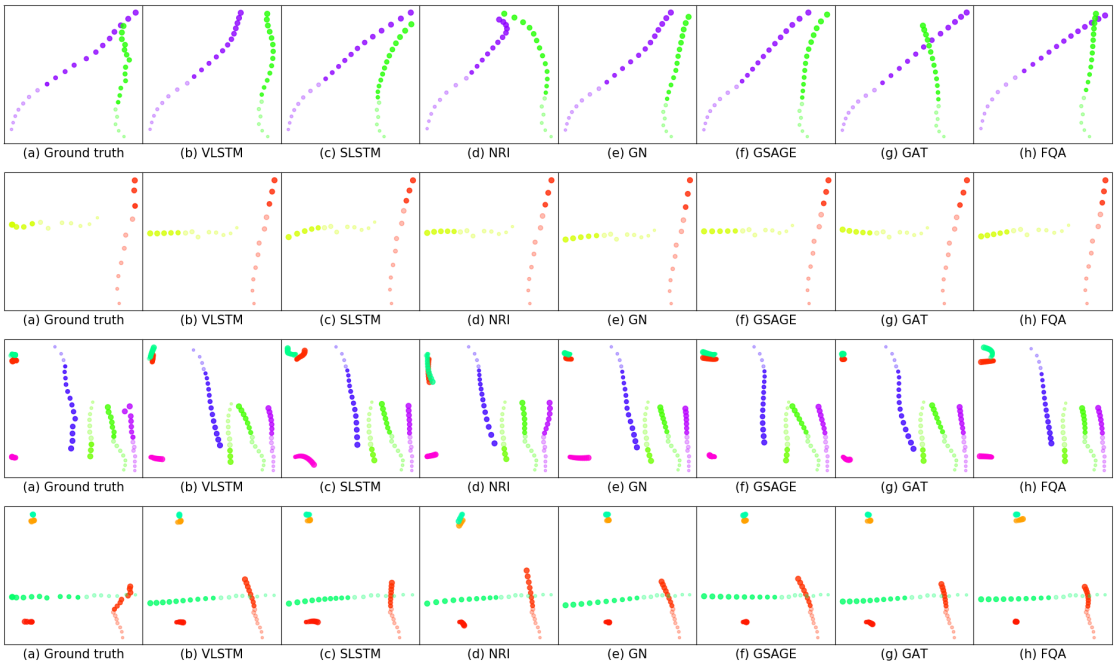


Figure 4.8: Predicted trajectory visualization from various models on ETH-UCY dataset.

With visualizations on the NBA dataset we highlight when our setup and most interaction modeling approaches may not be useful for trajectory prediction. Figure 4.11 shows a scene from the NBA dataset with the ball trajectory being green and the team players being blue and red trajectories. A blue player carries the ball and passes it to a teammate at the corner of the field after the observation period ($2T/5$) ends, which turns all the red player trajectories towards that corner (ground truth). Such passes and consequent player motions are heavily intent dependent and quite unpredictable. Most methods e.g. FQA instead predicate an equally valid alternative in which the original blue player carries the ball towards the basket. NBA dataset comprises of many such intent dependent sudden events or otherwise motions with many valid alternatives which cannot be predicted in the long term ($3T/5$). For such datasets, we recommend making shorter length predictions or including visual observations for making predictions instead of just trajectory data. Figure 4.12 shows three other cases where a player chooses to counter-intuitively pass (or not pass) the ball after the observation period ends. Most methods, especially FQA, predict an

equally valid and often more likely alternative of not passing the ball or passing it in a direction more logically deducible from only trajectory data.

For such datasets, we recommend making shorter length predictions or including visual observations in the input instead of just trajectory data to account better for strong intent-dependencies. Alternatively, FQA being primarily designed to target interactions, can be combined with stronger models for modeling intents, e.g., hierarchical policy networks [148] to improve performance on intent-driven prediction setups.

4.4 Summary

In this chapter, we have presented a general architecture designed to predict trajectories in multi-agent systems while modeling the crucial inductive biases of motion, namely, inertia, relative motion, intents and interactions. Our novel Fuzzy Query Attention (FQA) mechanism models pairwise interactions between agents by learning to make fuzzy (continuous-valued) decisions. We demonstrate significant performance gains over existing state-of-the-art models in diverse domains thereby demonstrating the potential of FQA. We further provide ablations and empirical analysis to understand the strengths and limitations of our approach.

FQA additionally allows including human-knowledge in the model by manually inserting known decisions (when available) and learning their corresponding responses. This could be useful for debugging models in practical settings and at times aligning the model's decisions to human expectations. Our architecture relies only on trajectory data and hence can be employed in conjunction to or alternatively as part of visual processing pipelines for trajectory prediction. It can be successfully incorporated in deep learning pipelines for predicting traffic trajectories around self-driving autonomous vehicles, predicting motion of pedestrians on roads etc. Note that while FQA is primarily designed to target interactions, it can be combined with stronger models for modeling intents, e.g., hierarchical policy networks [148] to improve performance on intent-driven

prediction setups e.g. in sports analytics for predicting valid or alternative strategies for basketball players.

This is only a starting point to incorporate fuzzy logic in deep learning models and we believe that many other application domains can benefit from embedding fuzzy operators in the model. In the future, we plan to investigate the potential of more complex symbolic and fuzzy reasoning modules in multi-agent trajectory prediction and other application domains.

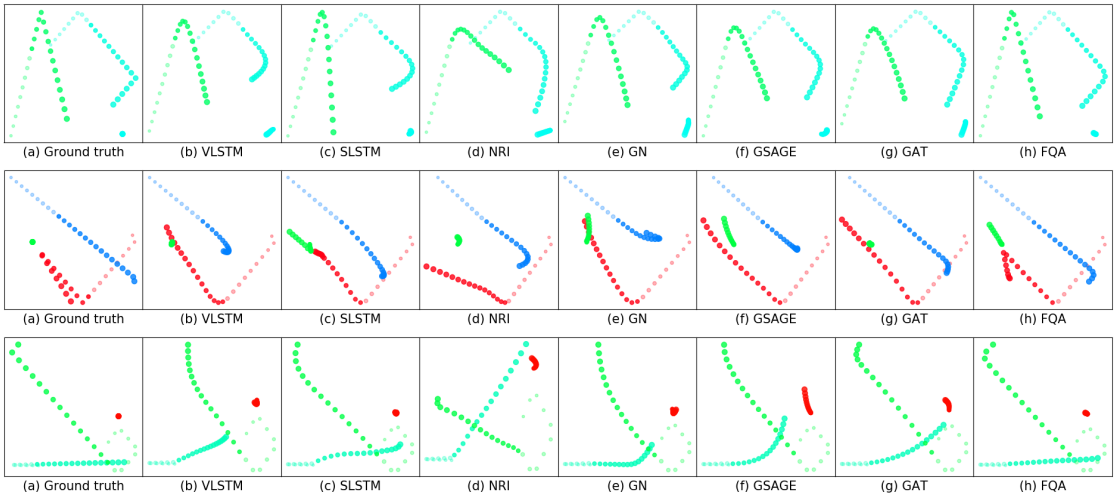


Figure 4.9: Predicted trajectory visualization from various models on Collisions dataset.

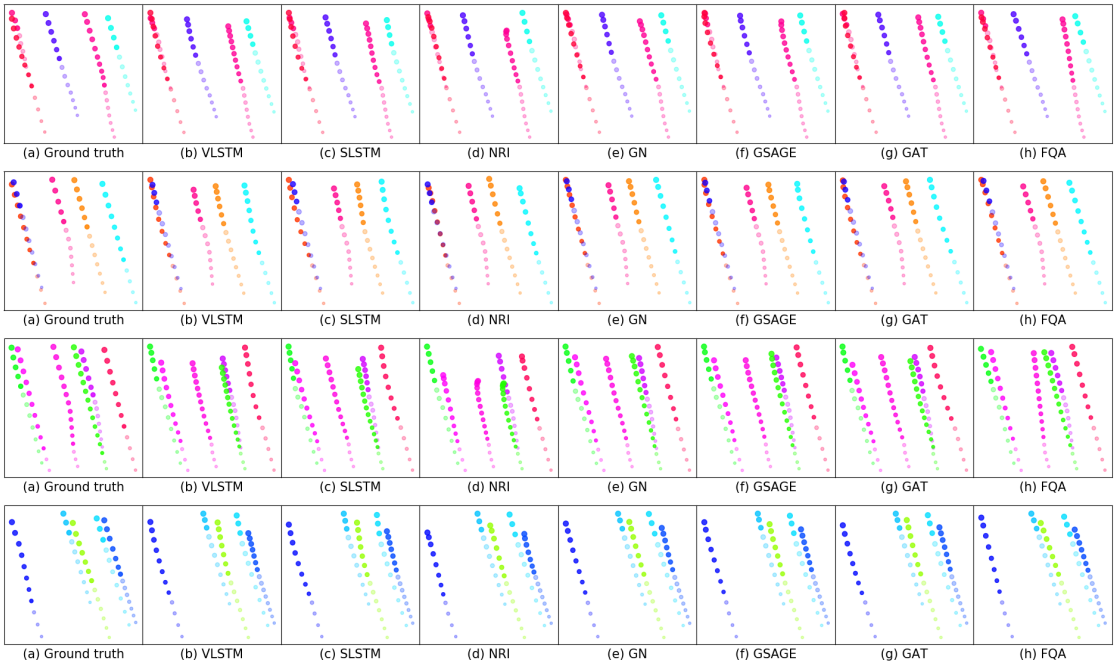


Figure 4.10: Predicted trajectory visualization from various models on NGsim dataset.

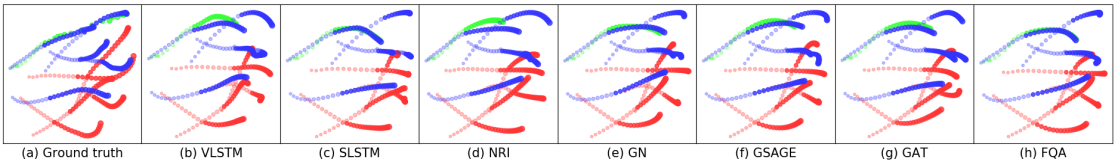


Figure 4.11: NBA data: Green agent is the ball, while the 5 players in each team are colored blue and red. The pass between blue team players is unpredictable and heavily intention dependent.

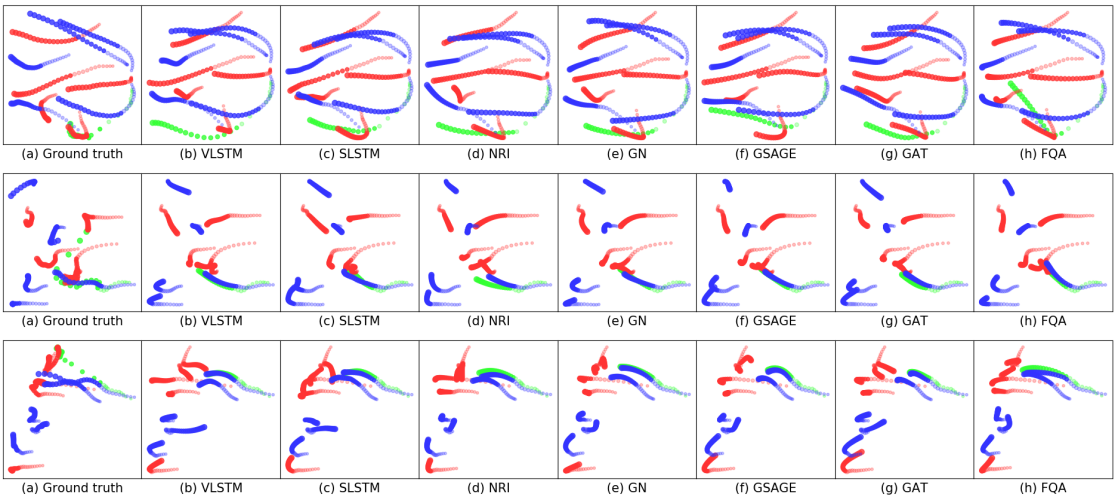


Figure 4.12: Predicted trajectory visualization from various models on the NBA dataset.

Chapter 5

Policy Learning for Continuous Space Security Games using Neural Networks

5.1 Introduction

Stackelberg Security Games (SSGs) are two-player leader-follower games. 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. In this chapter, 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 [3] and Cournot Adjustment (CA) [34]. Our approach gives a good strategy for both players, even when the baselines fail to converge.

5.2 Preliminaries

Notation: 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.

We will demonstrate our algorithm on two domains:

- **Rock-Paper-Scissors**: A small stateless zero-sum game with three discrete actions. Please see section 3.3.1 for details of the game. This game 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**: We also introduce 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. Please see section 3.3.4 for the detailed game model.

5.3 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:

$$\begin{aligned} J_D(\boldsymbol{w_D}, \boldsymbol{w_O}) &= \mathbb{E}_{s,a_D,a_O}[r_D(s, a_D, a_O)] \\ &= \int_s \int_{a_D} \int_{a_O} P(s)\pi_D(a_D|s; \boldsymbol{w_D})\pi_O(a_O|s; \boldsymbol{w_O}) \\ &\qquad r_D(s, a_D, a_O)\; ds\, da_D\, da_O \end{aligned} \tag{5.1}$$

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:

$$\boldsymbol{w}_{\boldsymbol{D}}^{*} \in arg \max_{\boldsymbol{w_D}} \min_{\boldsymbol{w_O}} J_D(\boldsymbol{w_D}, \boldsymbol{w_O}) \tag{5.2}$$

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:

$$\boldsymbol{w}_{\boldsymbol{O}}^{*} \in arg \min_{\boldsymbol{w_O}} J_D(\boldsymbol{w}_{\boldsymbol{D}}^{*}, \boldsymbol{w_O}) \tag{5.3}$$

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:

$$\pi_D^* \in BR_D(\pi_O^*) \tag{5.4}$$

$$\pi_O^* \in BR_O(\pi_D^*) \tag{5.5}$$

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 [34], 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 3.1.7) as:

$$\boldsymbol{\nabla_{w_D} J_D} = \mathbb{E}_{s,a_D,a_O}[r_D \nabla_{w_D} \log \pi_D(a_D|s;\boldsymbol{w_D})] \tag{5.6}$$

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) as pointed out in section 3.1.7. 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.

5.4 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 [122] and those from game theoretic fictitious play [49, 50], to find the optimal defender parameters $\boldsymbol{w_D}$ which maximize her utility. Policy gradient theorem [122] 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.

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 $\tilde{r}_D^i, \tilde{r}_O^i$;
    $\boldsymbol{\nabla_{w_O} J_O} = \frac{1}{n_b} \sum_{i=1}^{n_b} \tilde{r}_O^i \nabla_{\boldsymbol{w_O}} \log \pi_O(\tilde{a}_O^i|s^i; \boldsymbol{w_O})$;
    $\boldsymbol{w_O} := \boldsymbol{w_O} + \frac{\alpha_O}{1+ep\,\beta_O} \boldsymbol{\nabla_{w_O} J_O}$;

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 minibatch 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.

5.5 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 [78, 146].

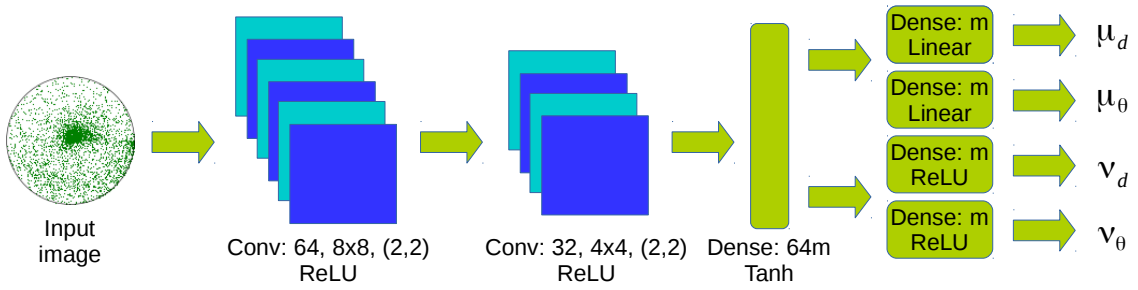


Figure 5.1: Defender's policy represented via a CNN

5.5.1 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:

$$\pi_D(\boldsymbol{d}, \boldsymbol{\theta}|s) = \prod_{i\in[m]} p_{ln}(d_i; \mu_{d,i}, \nu_{d,i}) p_{ln}\left(\frac{\theta_i}{2\pi}; \mu_{\theta,i}, \nu_{\theta,i}\right) \tag{5.7}$$

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 5.1.

5.5.2 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 (5.7).

5.5.3 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 [92]. 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.

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.

5.6 Experiments and Results

We now present experiments against several baselines.

5.6.1 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 [34]. Another method called StackGrad was recently proposed [3]. 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.

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).

5.6.2 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 5.1. 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.

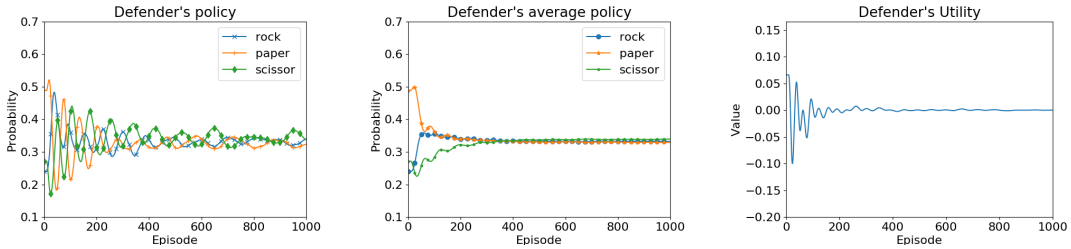
	CA	StackGrad	StackGradFP	OptGradFP
ep_{max}	400	400	400	400
n_s	50	50	25	25
n_b	50	50	250	250
E	50	50	10000	10000
$\alpha_{\{D,O\}}$	$5e-6$	$5e-6$	$1e-5$	$5e-4$
$\beta_{\{D,O\}}$	$\frac{9}{ep_{max}}$	$\frac{9}{ep_{max}}$	$\frac{9}{ep_{max}}$	$\frac{9}{ep_{max}}$

Table 5.1: Hyperparameters

5.6.3 Results

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).

5.6.4 Rock-Paper-Scissors Results



(a) (b) (c)

Figure 5.2: (a) Defender's policy, (b) Defender's average policy, (c) Defender's utility

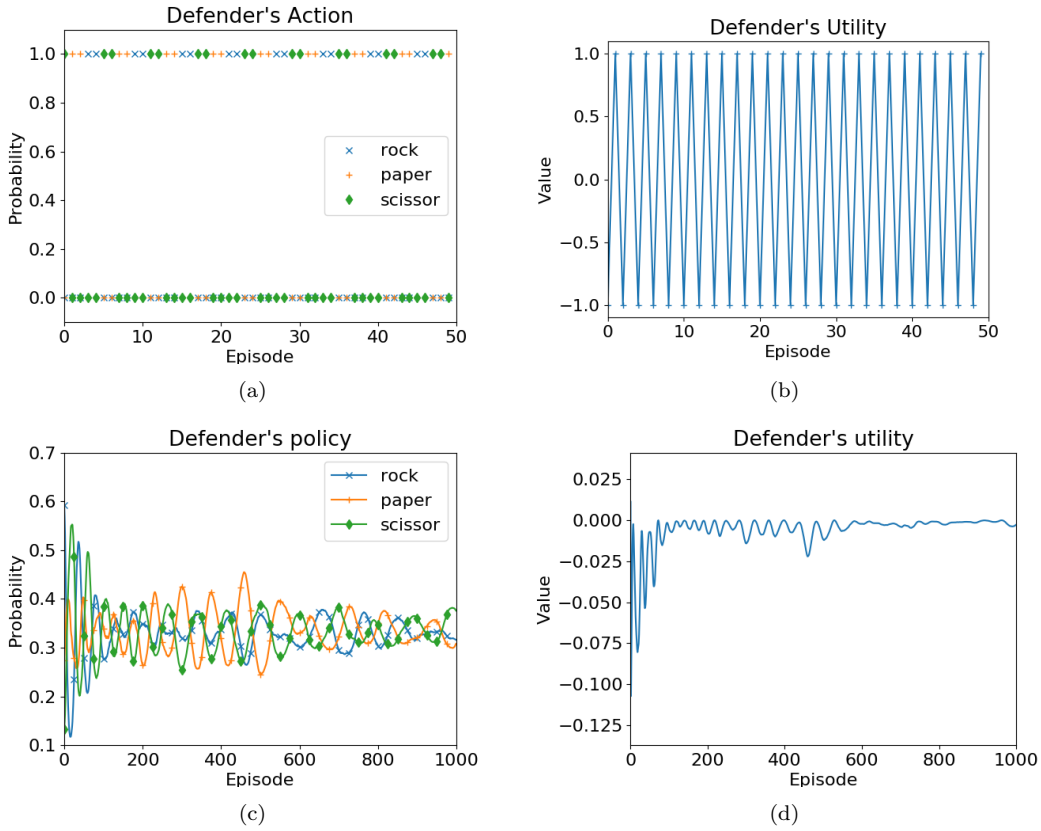


Figure 5.3: Results of CA and StackGrad on Rock-Paper-Scissors: (a) Defender's actions with CA on RPS, (b) Defender's utility with CA on RPS, (c) Defender's policy with StackGrad on RPS, (d) Defender's utility with StackGrad on RPS.

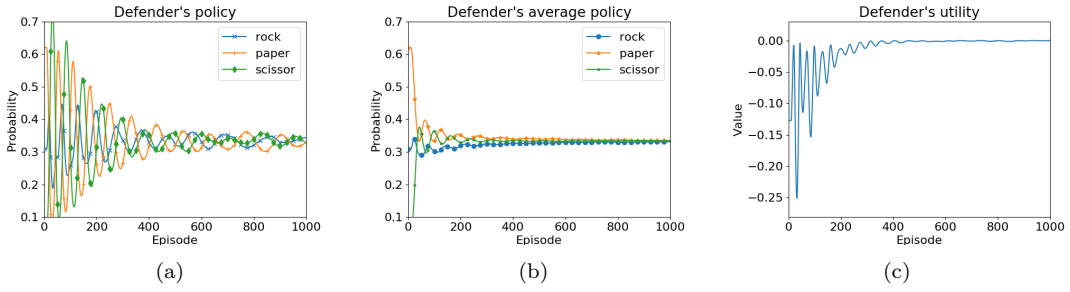


Figure 5.4: Results of StackGradFP on Rock-Paper-Scissors: (a) Defender's policy at each episode, (b) Defender's average policy at each episode, and (c) Defender's utility at each episode.

Figure 5.2 shows the defender's statistics as a function of the number of episodes, when OptGradFP is applied. Note from figure 5.2a, 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 5.2b) 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 5.2c. These results demonstrate the convergence of OptGradFP. Figures 5.3 and 5.4 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 have used $n_s = 50$ and $n_b = 500$ for all baselines.

5.6.5 Forest Security Game Results

5.6.5.1 Learned policy on a single state

We show a visualization of the players' final mixed strategies in figure 5.5, 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 5.5a, 5.5b). In fact, they do not generally converge and keep oscillating. However,

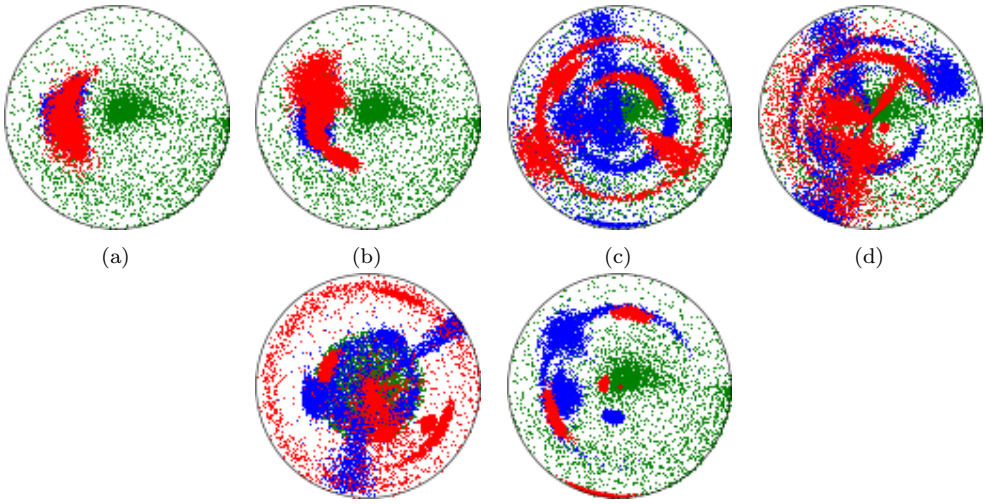


Figure 5.5: Visualization of players' policies. The blue and red dots show sampled positions for guards and lumberjacks respectively: (a) CA, (b) StackGrad, (c) StackGradFP, (d) OptGradFP, (e) OptGradFP on a forest with a central core, and (f) OptGrad.

OptGradFP and StackGradFP (figures 5.5d, 5.5c) 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 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 [63]. 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.

5.6.5.2 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. The opponent's final best response utility for the forest game can be 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 used $k = 25$ 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. Table 5.2 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 5.2: Opponent's best response utility ($\pm$ std. error of mean).

5.6.5.3 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 5.5f) 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.

5.6.5.4 Computation time

The time for computing the defender's mixed strategy on a single forest state using Algorithm 1 is shown in Table 5.3. 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 5.3: Computation time for all algorithms (in seconds).

5.6.5.5 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 5.5) 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 5.4. 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. The predicted strategies and the independently generated strategies for 4 randomly chosen test states are visualized in Figure 5.6. 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.

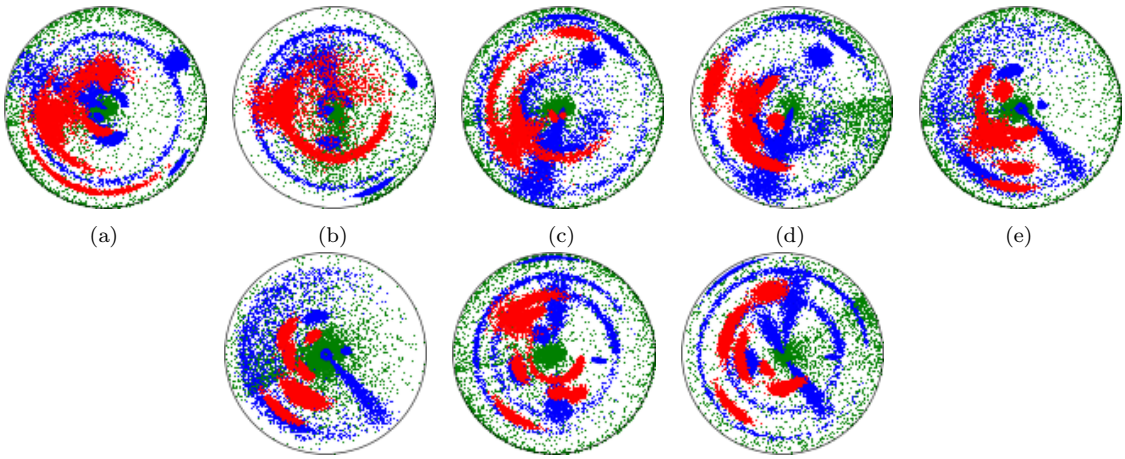


Figure 5.6: Visualization of players' strategies on randomly chosen test states (defender: blue, opponent: red): (a) Predicted: 1, (b) Computed: 1, (c) Predicted: 7, (d) Computed: 7, (e) Predicted: 8, (f) Computed: 8, (g) Predicted: 9, and (h) Computed: 9.

Lastly, though independent training on each state requires about ≈ 12250 seconds (Table 5.3) 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 5.4: Opponent's best response utilities $\pm$ std. error of mean for predicted strategies and independently computed strategies.

5.6.6 Comparing all algorithms

Now we briefly summarize our findings and compare all algorithms in terms of their overall performance. We note that 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. On the other hand, OptGrad updates both players with a soft PG step and hence outperforms StackGrad, but without replay memory, neither of them converges.

After adding a 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. Hence, both algorithms exhibit similar performance, however OptGradFP dominates because of its lower computation time.

5.7 Discussion

5.7.1 Why not discretize?

Some previous works [142, 46, 36, 139] 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 [139].

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.

5.7.2 Limitations of gradient-based methods

During our experiments, we noted certain key limitations of our method and other baselines. Gradient-based approaches rely on availability of non-zero gradients throughout the state-action spaces for both players, which may not always apply for all games. In such cases, the algorithm can sometimes stagnate prematurely if the gradient of the utility w.r.t. the policy parameters becomes zero. Hence, we point out that gradient-based approaches either require careful initialization to compute good mixed strategies for a given state or additional means of mitigating zero-gradient scenarios.

5.8 Summary

In this chapter, 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.

Chapter 6

DeepFP for Finding Nash Equilibrium in Continuous Action Spaces

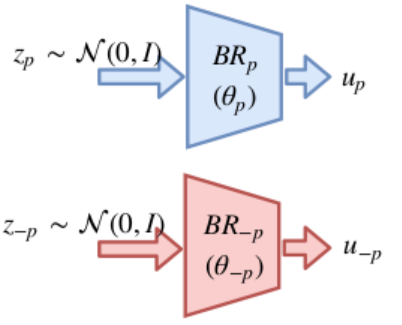
6.1 Introduction

In this chapter, we present DeepFP, an approximate fictitious play algorithm for two-player games with continuous action spaces. DeepFP addresses the lack of representational power of OptGradFP since it represents players' approximate best responses via state-of-the-art generative neural networks which are highly expressive implicit density approximators with no shape assumptions on players' action spaces. Since implicit density models cannot be trained directly, it also uses a game-model network which is a differentiable approximation of the players' payoffs given their actions, and trains these networks end-to-end in a model-based learning regime. Further, DeepFP allows replacing these networks with domain-specific oracles if available. This allows working in the absence of gradients for player/(s) and exploit techniques from research areas like mathematical programming to compute best responses.

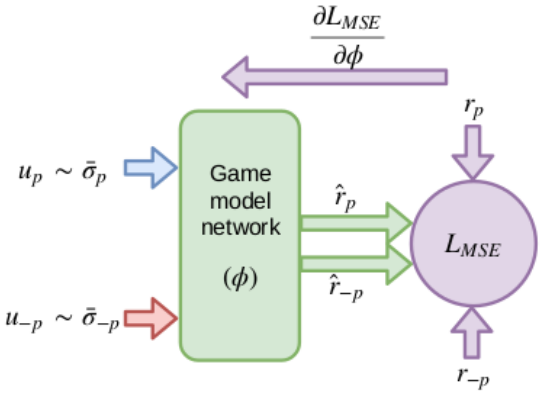
Further, our model-based training proceeds without any likelihood estimates and hence does not yield $-\infty$ log-likelihoods in any parts of the action space, thereby converging stably. Moreover, unlike OptGradFP, DeepFP is an off-policy algorithm and trains significantly faster by estimating expected rewards using the game model network instead of replaying stored games.

6.2 Deep Fictitious Play

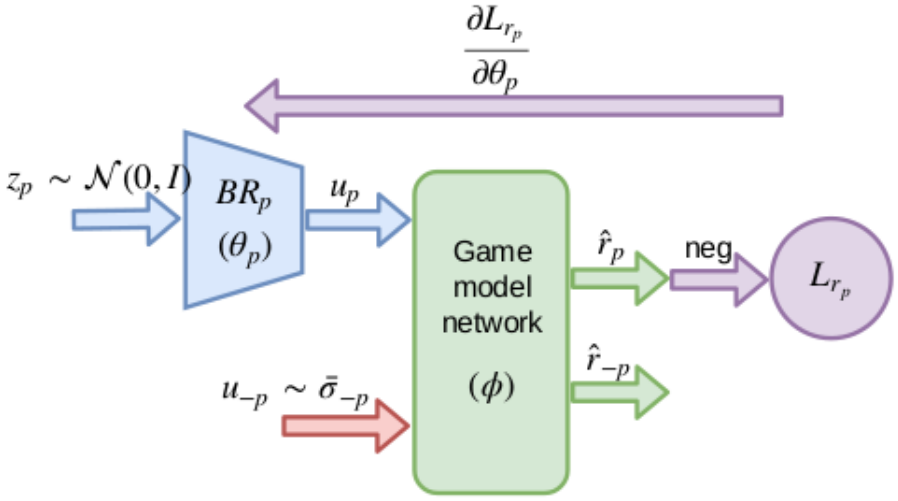
From here on, we use the two-player game model introduced in section 3.1.4. To compute NE for a game, we introduce an approximate realization of fictitious play in high-dimensional continuous action spaces, which we call Deep Fictitious Play (DeepFP). Let the density function corresponding to the empirical distribution of player p's previous actions (a.k.a. belief density) be $\bar{\sigma}_p$. Since fictitious play involves player p repeatedly best responding to his opponent's belief density $\bar{\sigma}_{-p}$, extending the procedure to continuous action spaces requires approximations for two essential ingredients: (a) belief densities over players' actions, and (b) best responses for each player.



(a) Sampling actions from the best response network



(b) Learning game model network parameters ϕ



(c) Learning best response network parameters θ_p

Figure 6.1: Neural network models for DeepFP; Blue color denotes player p, red denotes his opponent $-p$, green shows the game model network and violet shows loss functions and gradients.

6.2.1 Approximating belief densities

Representing belief densities compactly is challenging in continuous action spaces. However with an appropriate approximation to Fictitious Play, one can get away with a representation which only requires sampling from the belief density but never explicitly calculating the density at any point in the action space. Our DeepFP is one such approximation and hence we maintain the belief density $\bar{\sigma}_p$ of each player p via a non-parameterized population based estimate i.e. via a simple memory of all actions played by p so far. Directly sampling u_p from the memory gives an unbiased sample from $\bar{\sigma}_p$.

6.2.2 Approximating best responses

Computing exact best responses is intractable for most games. But when the expected reward for a player p is differentiable w.r.t. the player's action u_p and admits continuous and smooth derivatives, approximate best responses are feasible. One way is to use the gradient of reward to update the action u_p iteratively using gradient ascent till it converges to a best response. Since the best response needs to be computed per iteration of FP, employing inner iterations of gradient descent can be expensive. However since the history of play for players doesn't change too much between iterations of FP, we expect the same of best responses. Consequently we approximate best responses with function approximators (e.g., neural networks) and keep them updated with a single gradient ascent step (also done by [30]). We propose a *best response network* for each player p which maps an easy to sample d_p-dimensional random variable $Z_p \in \mathbb{R}^{d_p}$ (e.g. $Z_p \sim \mathcal{N}(0, I_{d_p})$) to the player's action u_p. By learning an appropriate mapping $BR_p(z_p; \theta_p)$ parameterized by weights θ_p, it can approximate any density in the action space U_p (figure 6.1a). Note that this is an implicit density model i.e. one can draw samples of u_p by sampling $z_p \sim \mathcal{P}_{Z_p}(\cdot)$ and then computing $BR_p(z_p; \theta_p)$, but no estimate of the density is explicitly available. Further, best response networks maintain stochastic best responses since they lead to smoother objectives for gradient-based

optimization. Using them is common practice in policy-gradient and actor-critic based RL since deterministic best responses often render the algorithm unstable and brittle to hyperparameter settings (also shown by [42]).

To learn θ_p we need to approximate the expected payoff of player p given by $\mathbb{E}_{(u_p \sim BR_p(\cdot;\theta_p), u_{-p} \sim \bar{\sigma}_{-p})}[r_p]$ as a differentiable function of θ_p. However a differentiable game model is generally not available a priori, hence we also maintain a *game model network* which takes all players' actions i.e. $\{u_p, u_{-p}\}$ as inputs and predicts rewards $\{\hat{r}_p, \hat{r}_{-p}\}$ for each player. This can either be pre-trained or learnt simultaneously with the best response networks directly from gameplay data (figure 6.1b). Coupled with a shared game model network, the best response networks of players can be trained to approximate best responses to their opponent's belief densities ($\bar{\sigma}_{-p}$) (figure 6.1c). The training procedure is discussed in detail in Section 6.2.3.

When the expected reward is not differentiable w.r.t. players' actions or the derivatives are zero in large parts of the action space, DeepFP can also employ approximate best response oracle (BRO_p) for player p. The oracle can be a non-differentiable approximation algorithm employing Linear Programming (LP) or Mixed Integer Programming (MIP) and since it will not be trained, it can also be deterministic. In many security games, Mixed-integer programming based algorithms are proposed to compute best responses and our algorithm provides a novel way to incorporate them as subroutines in a deep learning framework, as opposed to most existing works which require end-to-end differentiable policy networks and cannot utilize non-differentiable solutions even when available.

6.2.3 DeepFP

Algorithm 2 shows the DeepFP pseudocode. DeepFP randomly initializes any best response networks and game model network (if needed) and declares an empty memory (`mem`) to store players' actions and rewards [lines 1-2].

Algorithm 2: DeepFP

Data: `max_games`, batch sizes (m_1, m_2, m_G), memory size E, game simulator and oracle BRO_p for players with no gradient
Result: Final belief densities $\bar{\sigma}_p^*$ in `mem` $\forall$ players p

1 Initialize all network parameters $(\theta_1, \theta_2, \phi)$ randomly;
2 Create empty memory `mem` of size E;
3 **for** *game* $\in \{1, \ldots,$ *max_games*$\}$ **do**
    `/* Obtain best responses */`
4   **for** *each player p* **do**
5   **if** *grad_avlbl(p)* **then**
6   Sample $z_p \sim \mathcal{N}(0, I)$;
7   Approx. best response $u_p = BR_p(z_p; \theta_p)$;
8   **else**
9       $u_p = BRO_p(\bar{\sigma}_{-p})$ with $\bar{\sigma}_{-p}$ from `mem`;
    `/* Play game and update memory */`
10 Play with $u = \{u_1, u_2\}$ to get $r = \{r_1, r_2\}$;
11 Store sample $\{u, r\}$ in `mem`;
    `/* Train shared game model net */`
12 **if** *grad_avlbl(p) for any* $p \in \{1, 2\}$ **then**
13   Draw samples $\{u^i, r^i\}_{i=1:m_G}$ from `mem`;
14    $\phi :=$ Adam.min$(L_{MSE}, \phi, \{u^i, r^i\}_{i=1:m_G})$;
    `/* Train best response nets */`
15 **for** *each player p with grad_avlbl(p)* **do**
16   Draw samples $\{u^i\}_{i=1:m_p}$ from `mem`;
17    $\theta_p :=$ Adam.min$(L_{r_p}, \theta_p, \{u^i_{-p}\}_{i=1:m_p})$;

Then it iteratively makes both players best respond to the belief density of their opponent. This best response can be computed per player p via a forward pass of the best response network BR_p or via a provided oracle BRO_p or if gradients are not available [lines 4-9]. The best response moves and the rewards obtained by playing them are stored in `mem` [lines 10-11]. Samples from *exact* belief density $\bar{\sigma}$ of both players are available from `mem`.

The game model network is also trained simultaneously to learn a differentiable reward model of the game [lines 12-14]. It takes all players' actions u as input and predicts the game rewards $\hat{r}(u; \phi)$ for all players. Its parameters ϕ can be learnt by minimizing the mean square error loss over a minibatch of samples $\{u^i\}_{i=1:m_G}$ from `mem`, using any optimizer (we use Adam [72]):

$$L_{MSE}(\phi) = \frac{1}{2m_G} \sum_{p \in \{1,2\}} \sum_{i=1}^{m_G} (\hat{r}_p(u^i; \phi) - r_p^i)^2.$$

The advantage of estimating this differentiable reward model independent of playing strategies is that it can be trained from the data in replay memory without requiring importance sampling, hence it can be used as a proxy for the game simulator to train the best response networks. An alternative could be to replay past actions of players using the game simulator as done by [66], but it is much slower (see section 6.3.2).

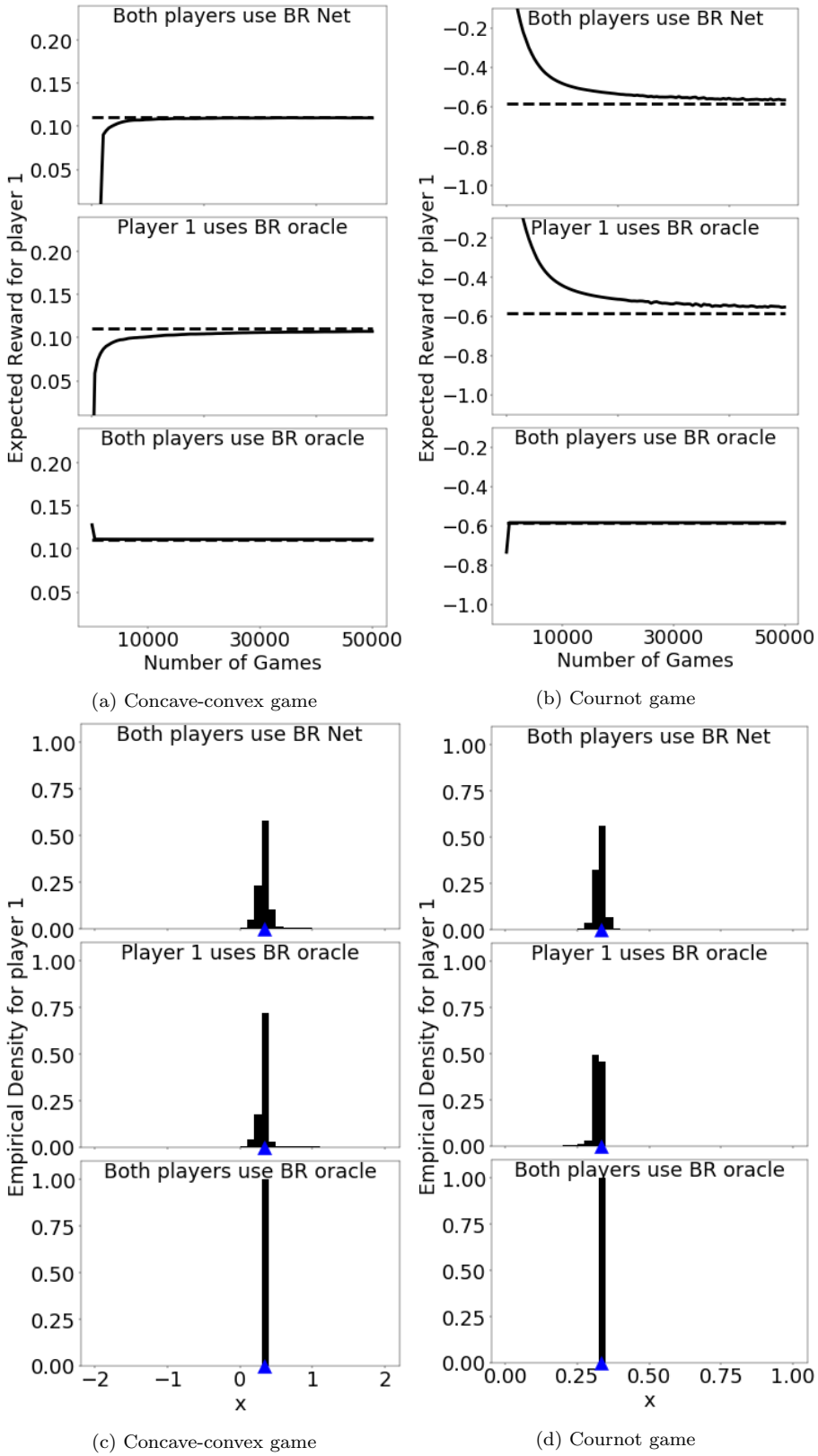
Finally each player updates their best response network to keep it a reasonable approximation to the best response to his opponent's belief density [lines 15-17]. For this, each player p maximizes his expected predicted reward $\hat{r}_p$ (or minimizes expected $-\hat{r}_p$) against the opponent's belief density $\bar{\sigma}_{-p}$ (see figure 6.1c) using any optimizer (we use Adam):

$$L_{r_p}(\theta_p) = -\mathbb{E}_{(z_p \sim \mathcal{N}(0,I), u_{-p} \sim \bar{\sigma}_{-p})}[\hat{r}_p(BR_p(z_p; \theta_p), u_{-p}; \phi)].$$

The expectation is approximated using a minibatch of samples $\{u_{-p}^i\}_{i=1:m_p}$ drawn from `mem` and $\{z_p^i\}_{i=1:m_p}$ independently sampled from a standard normal distribution. In this optimization, ϕ is held constant and the gradient is only evaluated w.r.t. θ_p and the updates applied to the best response network. In this sense, the game model network acts like a critic to evaluate the best responses of player p (actor) against his opponent's belief density $\bar{\sigma}_{-p}$ similar to actor-critic methods [91]. However, unlike actor-critic methods we train the best response and the game model networks in separate decoupled steps which potentially allows replacing them with pre-trained models or approximate oracles, while skipping their respective learning steps.

6.2.4 Connections to Boltzmann actor-critic and convergence of DeepFP

DeepFP is closely related to the Boltzmann actor-critic process proposed by Generalized Weakened Fictitious Play (GWFP) [82], which converges to the NE under certain assumptions. But it differs in two crucial aspects: (i) GWFP requires assuming explicit probability densities and involves weakened ϵ-best responses which are updated via a Boltzmann actor-critic process. Since we store the empirical belief densities and best responses as implicit densities, a Boltzmann-style strategy update is infeasible, (ii) GWFP also requires the ϵ-best responses to eventually become exact (i.e. when $\lim_{n\to\infty} \epsilon_n \to 0$). Since we are approximating stochastic best responses via generative neural networks (or approximate oracles), the assumption may not always hold exactly. Nevertheless, with our approximate oracle and one-step gradient update-based best response networks, we empirically observed that DeepFP converges for multiple games with continuous reward functions wherever GWFP converges. At convergence, the belief density $\bar{\sigma}^*$ in `mem` is a non-parametric approximation to a NE density for both players.



(a) Concave-convex game

(b) Cournot game

(c) Concave-convex game

(d) Cournot game

Figure 6.2: DeepFP on simple games under three settings: When both players learn BR nets (top), player 1 uses BR oracle (mid), and when both players use BR oracle (bottom); (a) and (b) Expected reward of player 1 converges to the true equilibrium value (shown by dashed line) for both games; (c) and (d) Final empirical density for player 1 approaches NE strategy for both games (shown by blue triangle on horizontal axis).

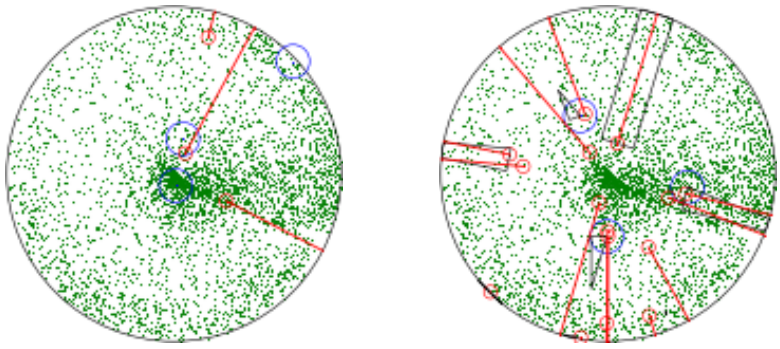
6.3 Experimental Evaluation

6.3.1 Simple games

We first evaluate DeepFP on two simple games, namely Concave-Convex game and the Cournot game introduced in sections 3.3.2 and 3.3.3 respectively. These games are designed such that traditional fictitious play is known to converge on them, and we use these potential sanity checks to demonstrate convergence of DeepFP to a nash equilibrium.

Figure 6.2 shows the results of DeepFP on these games and its convergence to the NE for all variants i.e. when both, exactly one, or no player uses the best response oracle. Note that both players using the best response oracle (bottom case in all subfigures) is the same as exact fictitious play and converges very fast as opposed to other cases (top and mid in all subfigures) since the latter variants require estimating the best responses from repeated gameplay.

6.3.2 Forest protection game



(a) (b)

Figure 6.3: Forest game with trees (green dots), guards (blue dots), guard radii R_g (blue circles), lumberjacks (red dots), lumberjack chopping radii R_l (red circles), lumberjacks' paths (red lines) and black polygons (top weighted capture-sets for guards): (a) With m=n=3, (b) Best response oracle for 3 guards and 15 lumberjacks.

For a large application of DeepFP, we choose the forest protection game as proposed by [66] and presented in section 3.3.4. We denote the Defender as D and Adversary as A. A full game example is shown in figure 6.3a. In our experiments we use the following settings for the game: $r_{pen} = 4.0, R_g = 0.1, R_l = 0.04$.

6.3.2.1 Approximate best response oracle

Note that if guards' locations do not overlap significantly with those of lumberjacks then changing them by a small amount does not affect the rewards for either player since no extra lumberjacks are ambushed. Hence, the gradient of reward w.r.t. defender's parameters ($\nabla_{\theta_D} r$) ≈ 0 over most of the action space. But the gradients for the adversary are continuous and non-zero because of the dense tree distribution. Hence we apply DeepFP to this game with a best response network for the adversary and an approximate domain-specific best response oracle for the defender. Devising a defender's best response to the adversary's belief distribution is non-trivial for this game. So we propose a greedy approximation to the best response (see algorithm 3). We define a capture-set for a lumberjack location l as the set of all guard locations within a radius R_g from any point on the trajectory of the lumberjack. The algorithm involves creating capture-sets for lumberjack locations l encountered so far in `mem` and intersecting these capture-sets to find those which cover multiple lumberjacks. Then it greedily allocates guards to the top m such capture-sets one at a time, while updating the remaining capture-sets simultaneously to account for the lumberjacks ambushed by the current guard allocation. We illustrate an oracle best response in figure 6.3b.

Algorithm 3: Approximate best response oracle

Data: mem, batch size m_D, game simulator, m, n
Result: Guard assignments approximating $BRO_D(\bar{\sigma}_A)$
1 Draw batch of adversary actions $\{u_A^i\}_{i=1:m_D}$ from $\bar{\sigma}_A$ (stored in mem);
2 Extract all $m_D \times n$ lumberjack locations $l \in \{u_A^i\}_{i=1:m_D}$;
/* Capture-set for each lumberjack */
3 Initialize empty capture-set list S;
4 **for** $l \in \{u_A^i\}_{i=1:m_D}$ **do**
5 Create a capture-set $s(l)$ (approximated by a convex polygon) i.e. as the set of all guard locations which are within radius R_g from any point on the trajectory of the lumberjack stopping at l;
6 Query reward $w(l)$ of ambushing at l (using simulator);
7 Append (s, w, l) to S.
/* Output max reward capture-sets */
8 Find all possible intersections of sets $s \in S$ while assigning a reward $w' = \sum_j w_j$ and lumberjacks $l' = \cap_j l_j$ to $s' = \cap_j s_j$ and append all new (s', w', l') triplets to S;
9 Pop the top m maximum reward sets in S one at a time and assign a single guard to each, while updating all remaining sets' weights to remove lumberjacks covered by the guard allotment;
10 Output the guard assignments.

Our algorithm involves the following approximations:

1. *Mini-batch approximation*: Since it is computationally infeasible to compute the best response to the full set of actions in mem, we best-respond to a small mini-batch of actions sampled randomly from mem to reduce computation (line 1).

2. *Approximate capture-sets*: Initial capture-sets can have arbitrary arc-shaped boundaries which can be hard to store and process. Instead, we approximate them using convex polygons for simplicity (line 5). Doing this ensures that all subsequent intersections also result in convex polygons.

3. *Bounded number of intersections*: Finding all possible intersections of capture-sets can be reduced to finding all cliques in a graph with capture-sets as vertices and pairwise intersections as edges. Hence it is an NP-hard problem with complexity growing exponentially with the number of polygons. We compute intersections in a pairwise fashion while adding the newly intersected polygons to the list. This way the k^{th} round of intersection produces uptil all

$k+1$-polygon intersections and we stop after $k=4$ rounds of intersection to maintain polynomial time complexity (implemented for line 8, but not shown explicitly in algorithm 3).

4. *Greedy selection*: After forming capture-set intersections, we greedily select the top m sets with the highest rewards (line 9).

6.3.2.2 Baselines

Since the forest protection game involves arbitrary tree density patterns, the ground truth equilibria are intractable. So we evaluate DeepFP by comparing it with OptGradFP [66] and to another approximate discrete linear programming method (henceforth called DLP).

DLP baseline: We propose DLP which discretizes the action space of players and solves a linear programming problem to solve the game approximately (but only for small m and n). The DLP method discretizes the action space in cylindrical coordinates with 20 radial bins and 72 angular bins, which gives a joint action space of size $(72 \times 20)^{m+n}$. For even a single guard and lumberjack, this implies about 2 million pure strategies. Hence, though DLP gives the approximate ground truth for m=n=1 due to our fine discretization, going beyond m or $n > 1$ is infeasible with DLP. The DLP baseline proceeds in two steps:

1. We generate $72 \times 20 = 1440$ cylindrically discretized bins and compute a matrix $R \in \mathbb{R}^{1440\times1440}$ where R_{ij} characterizes the defender's reward with a guard in the i-th bin and a lumberjack in the j-th bin. Each entry R_{ij} is computed by averaging the game simulator's reward over 20 random placements of the guard and lumberjack inside the bins.

2. Next we solve the following optimization problem for the defender:

$$\sigma^*, \chi^* = \arg \max_{\sigma \geq 0, \chi} \chi$$
$$s.t.\ \sigma^T R_{:j} \geq \chi \ \ \forall j$$
$$\sum_{i=1}^{1440} \sigma_i = 1$$

Note that χ represents the defender's reward, σ_i is the i-th element of $\sigma \in [0,1]^{1440}$ i.e. the probability of placing the guard in the i-th bin and $R_{:j}$ is the j-th column of R corresponding to the adversary taking action j. The above problem maximizes the defender's reward subject to the constraints that σ has all non-negative elements summing to 1 (since it's a distribution over all bins) and the defender's reward χ is least exploitable regardless of the adversary's placement in any bin j. Solving it gives us the optimal defender distribution σ^* over all bins to place the guard and the equilibrium reward for the defender χ^* when m=n=1.

6.3.2.3 Hyperparameters

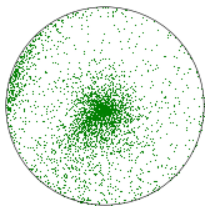
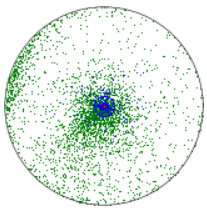
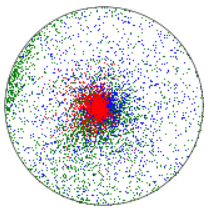
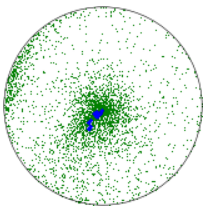
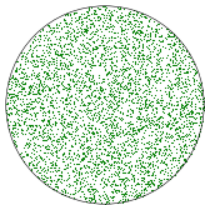
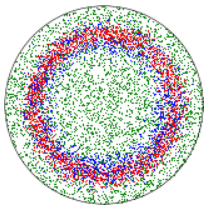
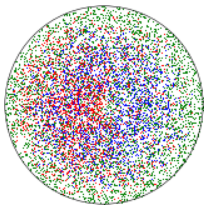
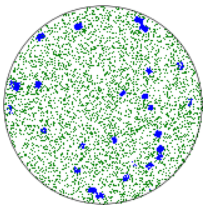
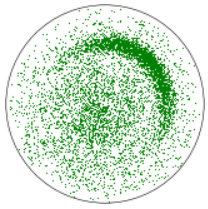
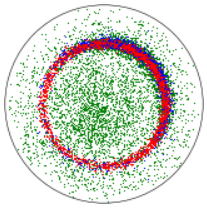
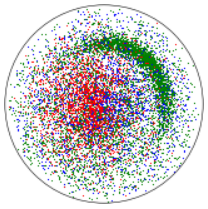
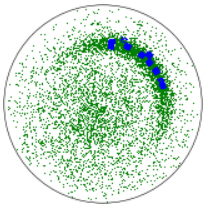
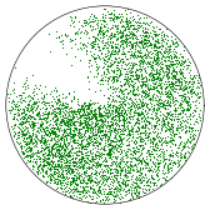
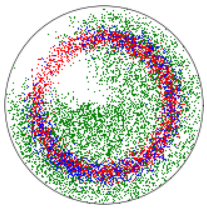
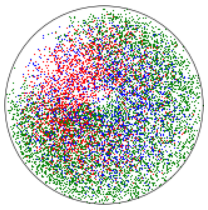
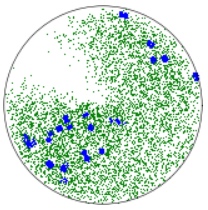
We set `max_games` $= E = 40000$ to provide enough iterations to DeepFP and OptGradFP for convergence. The batch sizes for DeepFP are set to $m_D = 3$ (kept small to have a fast oracle), $m_A = 64, m_G = 128$ (large for accurate gradient estimation). Additional algorithmic parameters and neural network architectures can be found in section A.1 in the appendix.

6.3.2.4 Exploitability analysis

Since direct computation of the ground truth equilibrium is infeasible for a forest, we compare all methods by evaluating the exploitability of the defender's final strategy as NE strategies are least exploitable. For this, we designed an evolutionary algorithm to compute the adversary's best response to the defender's final strategy. It maintains a population (size 50) of adversary's

actions and iteratively improves it by selecting the best 10 actions, duplicating them four-fold, perturbing the duplicate copies with gaussian noise (whose variance decays over iterations) and re-evaluating the population against the defender's final strategy. This evolutionary procedure is independent of any discretization or neural network and outputs the adversary action which exploits the defender's final strategy most heavily. We denote the reward achieved by the top action in the population as the exploitability ϵ and report the exploitability of the defender's strategy averaged across 5 distinct runs of each method (differing only in the initial seed). Since rewards can differ across forests due to the number of trees in the forest and their distribution, the exploitability of each forest can differ considerably. Also, since the evolutionary algorithm requires $150K - 300K$ game plays per run, it is quite costly and only feasible for a single accurate post-hoc analysis rather than using it to compute best responses within DeepFP.

Table 6.1: Results on four representative forests for m=n=1. Green dots: trees, blue dots: guard locations sampled from defender's strategy, red dots: lumberjack locations sampled from adversary's strategy. The exploitability metric shows that DLP which is approximately the ground truth NE strategy is the least exploitable followed by DeepFP, while OptGradFP's inflexible explicit strategies make it heavily exploitable.

Forest structure	DeepFP	OptGradFP	DLP (approx. ground truth)
 F1	 $\epsilon = 88.57 \pm 23.1$	 $\epsilon = 174.08 \pm 21.04$	 $\epsilon = 95.60 \pm 10.82$
 F2	 $\epsilon = 16.90 \pm 0.13$	 $\epsilon = 17.09 \pm 0.39$	 $\epsilon = 16.38 \pm 0.86$
 F3	 $\epsilon = 88.72 \pm 24.09$	 $\epsilon = 115.02 \pm 0.86$	 $\epsilon = 72.95 \pm 1.21$
 F4	 $\epsilon = 30.72 \pm 1.65$	 $\epsilon = 32.21 \pm 0.52$	 $\epsilon = 23.97 \pm 0.64$

6.3.2.5 Single resource case

Table 6.1 shows results on four representative forests when m=n=1. We observe that both DLP and DeepFP find strategies which intuitively cover dense regions of the forest (central forest patch for F1, nearly the whole forest for uniform forest F2, dense arch of trees for F3 and ring for the forest F4 with a tree-less sector). On the uniform forest F2, the expected NE strategy is a ring at

a suitable radius from the center, as outputted by DeepFP. However, DLP has a fine discretization and is able to sense minute deviations from uniform tree structure induced due to the sampling of trees from a uniform distribution, hence it forms a circular ring broken and placed at different radii. A similar trend is observed on F4. On F3, DeepFP finds a strategy strongly covering the dense arch of trees, similar to that of DLP. Note that sometimes DeepFP even finds less exploitable strategies than DLP (e.g. on F1), since DLP while being close to the ground truth still involves an approximation due to discretization. Overall, as expected DLP is in general the least exploitable method and is the closest to the NE, followed by DeepFP. OptGradFP is more exploitable than DeepFP for nearly uniform tree densities (F2 and F4) and heavily exploitable for forests with concentrated tree densities (F1 and F3), since unlike DeepFP, it is unable to approximate arbitrary strategy shapes.

6.3.2.6 Multiple resource case

Since DLP cannot scale for m or $n > 1$, we compute the strategies and exploitability for m=n=$\{2, 3\}$ on F3 in table 6.3 for DeepFP and OptGradFP only. We consistently observe that DeepFP accurately covers the dense forest arch of F3 and OptGradFP spreads both players out more uniformly (due to explicit strategies). For m=n=3 case, DeepFP also allots a guard to the central patch of F3. Overall, DeepFP is substantially less exploitable than OptGradFP. Table 6.2 shows more experiments for DeepFP and OptGradFP with m,n>1. We see that DeepFP is able to cover regions of importance with the players' resources but OptGradFP suffers from the zero defender gradients issue due to logit-normal strategy assumptions which often lead to sub-optimal results and higher exploitability.

6.3.2.7 Effect of memory size

In algorithm 2, we stored and best responded to all games in the replay memory. Figure 6.4a shows the expected reward ($\mathbb{E}[r_A]$) achieved by the adversary's final strategy against the defender's

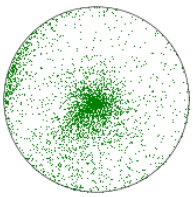
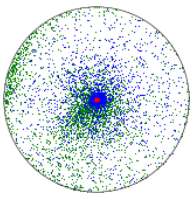
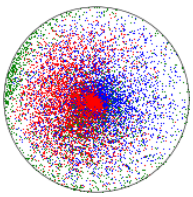
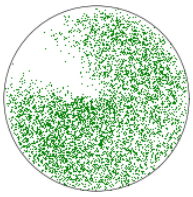
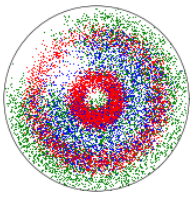
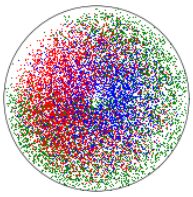
 F1	 DeepFP (m=n=2) $\epsilon = 153.21 \pm 50.87$	 OptGradFP (m=n=2) $\epsilon = 212.92 \pm 27.95$
 F4	 DeepFP (m=n=2) $\epsilon = 53.70 \pm 3.85$	 OptGradFP (m=n=2) $\epsilon = 49.00 \pm 3.68$

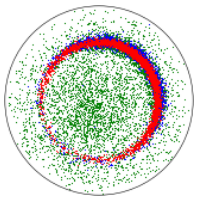
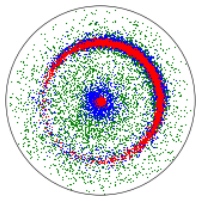
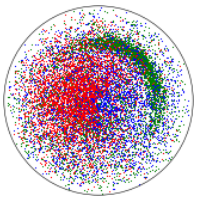
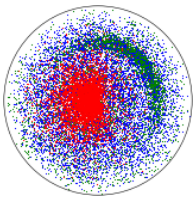
Table 6.2: More results on forests F1 and F4 for m=n=2.

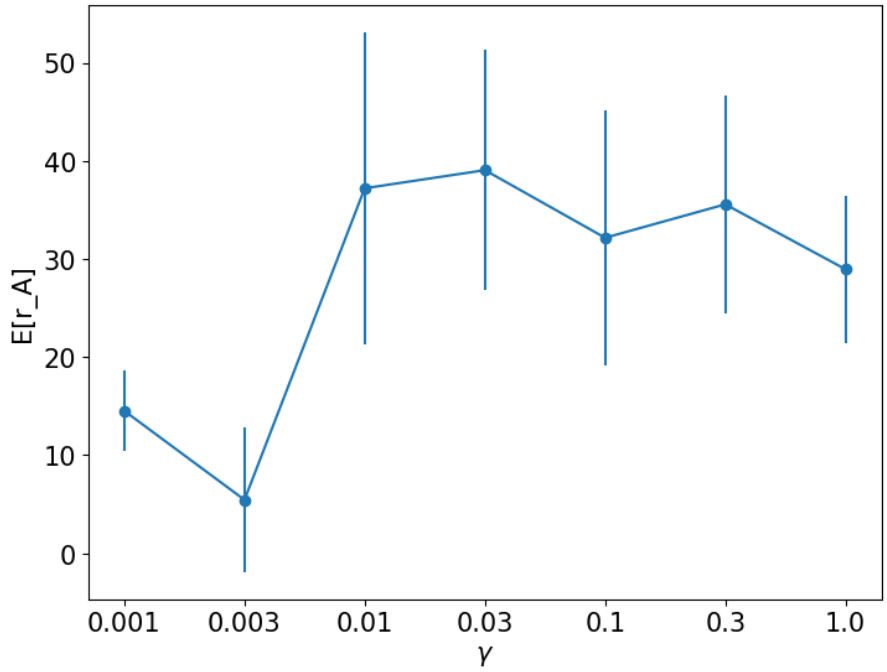
final strategy, when the replay memory size E is varied as a fraction γ of `max_games`. Only the most recent γ fraction of `max_games` are stored and best responded to, and the previous ones are deleted from `mem`. We observe that DeepFP is fairly robust to memory size and even permits significantly small replay memories (upto 0.01 times `max_games`) without significant deterioration in average rewards.

6.3.2.8 Running time analysis

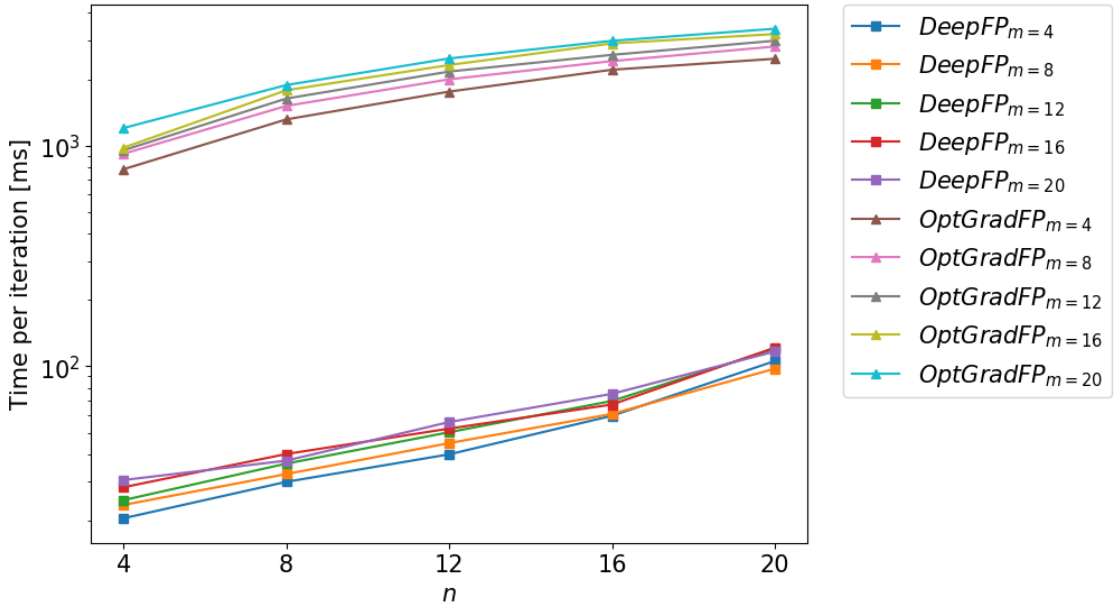
Given the same total number of iterations, we plot the time per iteration for DeepFP and OptGradFP in figure 6.4b with increasing m and n (y-axis has log scale). OptGradFP's training time increases very fast with increasing m and n due to high game replay time. With our approximate best-response oracle and estimation of payoffs using the game model network, DeepFP is orders of magnitude faster. For a total 40K iterations, training DeepFP takes about 0.64 ± 0.34 hrs (averaged over values of m and n) as opposed to 22.98 ± 8.39 hrs for OptGradFP.

Table 6.3: Results on forest F3 for m=n=$\{2, 3\}$. Green dots: trees, blue dots: guard locations sampled from defender's strategy, red dots: lumberjack locations sampled from adversary's strategy. DeepFP is always less exploitable than OptGradFP.

DeepFP (m=n=2)	DeepFP (m=n=3)	OptGradFP (m=n=2)	OptGradFP (m=n=3)
			
$\epsilon = 135.49 \pm 15.24$	$\epsilon = 137.53 \pm 8.63$	$\epsilon = 186.58 \pm 23.71$	$\epsilon = 190.00 \pm 23.63$



(a) Adversary's average reward with memory size E as a fraction of total games played. Even for a 1% fraction of memory size i.e. $\gamma = 0.01$, the average rewards are close to $\gamma = 1$ case.



(b) Time per iteration vs. players' resources. DeepFP is orders of magnitude faster than OptGradFP (y-axis has log scale).

Figure 6.4

6.3.2.9 Limitations of gradient-based algorithms

Like most gradient-based optimization algorithms, DeepFP and OptGradFP can sometimes get stuck in local nash equilibria. To study the issue of getting stuck in locally optimal strategies we show experiments with another forest F5 in Table 6.4. F5 has three dense tree patches and very sparse and mostly empty other parts. The optimal defender's strategy computed by DLP for m=n=1 is shown in C1. In such a case, due to the tree density being broken into patches, gradients for both players would be zero at many locations and hence both algorithms are expected to get stuck in locally optimal strategies depending upon their initialization. This is confirmed by configurations C2, C3, C4 and C5 which show strategies for OptGradFP and DeepFP with m=n=1 covering only a single forest patch. Once the defender gets stuck on a forest patch, the probability of coming out of it is small since the tree density surrounding the patches is negligible. However, with more resources for the defender and the adversary m=n=3, DeepFP is mostly able to break out of the stagnation and both players eventually cover more than a single forest patch (see C7), whereas OptGradFP is only able to cover additional ground due to random initialization of the 3 player resources but otherwise remains stuck around a single forest patch (see C6). DeepFP is partially able to break out because the defender's best response does not rely on gradients but rather come from a non-differentiable oracle. This shows how DeepFP can break out of local optima even in the absence of gradients if a best response oracle is provided, however OptGradFP relies purely on gradients and cannot overcome such situations.

6.4 Summary

In this chapter, we have presented DeepFP, an approximate fictitious play algorithm for games with continuous action spaces. DeepFP implicitly represents players' best responses via generative neural networks without prior shape assumptions and optimizes them using a learnt game-model network with gradient-based training. It can also utilize approximate best response oracles whenever

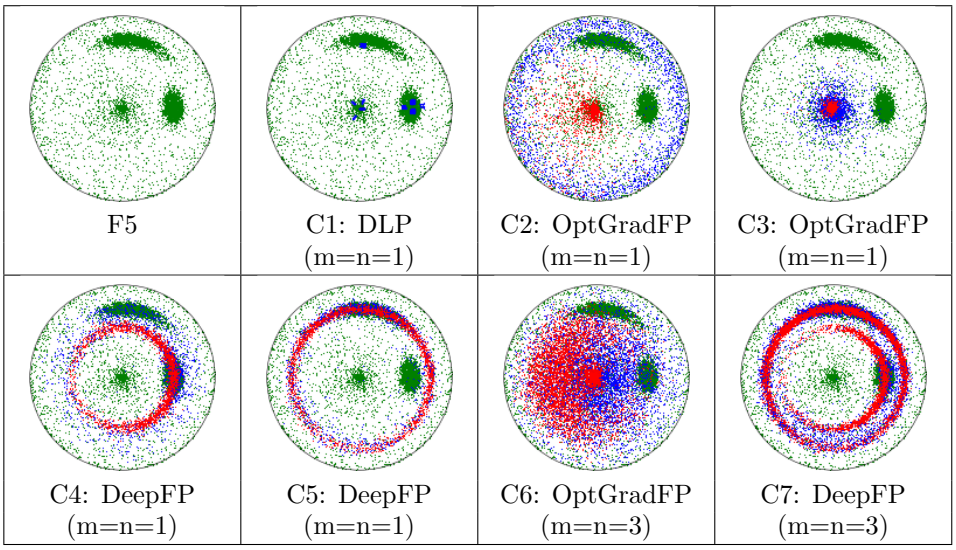


Table 6.4: Demonstrating getting stuck in locally optimal strategies.

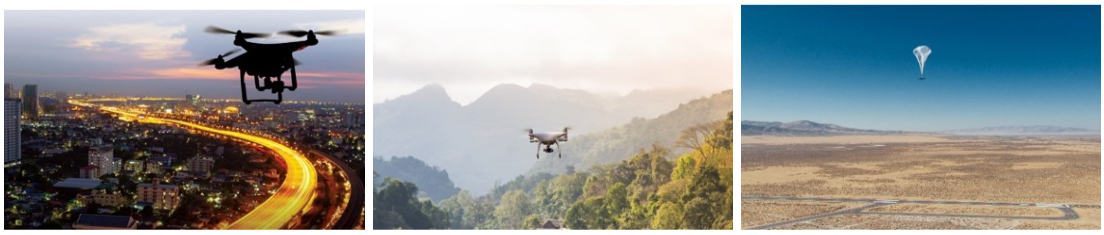
available, thereby harnessing prowess in approximation algorithms from discrete planning and operations research. DeepFP provides significant speedup in training time and scales well with growing number of resources.

DeepFP can be easily extended to multi-player applications, with each player best responding to the joint belief density over all other players using an oracle or a best response network. Like most gradient-based optimization algorithms, DeepFP and OptGradFP can sometimes get stuck in local nash equilibria. While DeepFP gets stuck less often than OptGradFP, principled strategies to mitigate local optima for gradient-based equilibrium finding methods remains an interesting direction for future work.

Chapter 7

Gradient-based Optimization for Multi-resource Spatial Coverage Problems

7.1 Introduction



(a) (b) (c)

Figure 7.1: Several domains requiring multi-resource spatial coverage: (a) Robotic surveillance, (b) Green security, and (c) Mobile sensor networks

Allocation of multiple resources for efficient spatial coverage is an important component in many practical systems, e.g., robotic surveillance, mobile sensor networks and green security domains (see figure 7.1). Surveillance tasks and sensor node placements generally involve assigning resources e.g. drones or sensors, each of which can monitor physical areas, to various points in a target domain such that a loss function associated with coverage of the domain is minimized [105]. Alternatively, green security domains follow a leader-follower game setup between two agents, where a defender defends a continuous target density in a geographical area (e.g. trees in a protected forest) with

limited resources to be placed, while an attacker plans an attack after observing the defender's placement strategy using its own resources [124].

Traditional methods used to solve multi-resource surveillance problems often make simplifying assumptions thereby leading to potential field methods [58], discretization based approaches [74] and voronoi tessellation based methods [25]. Similarly, many exact and approximate approaches have been proposed to maximize the defender's expected utility in green security domains against a best responding attacker [70, 3, 142, 46, 63, 59]. Notably, these approaches focus on exploiting certain specific spatio-temporal or symmetry structures of the domain under consideration.

While the existing work in this domain spans many sub-domains, in this chapter we focus on addressing a broad class of spatial coverage problems, where special spatial-temporal structure or symmetries cannot be exploited to efficiently allocate resources for coverage. In such cases, one has to rely on undirected exploration methods such as particle swarm optimization [94, 109] and genetic algorithms [126, 77, 140] for finding near-optimal placements for resources. However, since the coverage problem is generally combinatorially hard, such undirected search methods do not scale well as the number of resources to be placed grows larger.

Contributions: To address the above challenges, we propose the *coverage gradient theorem*, which provides a gradient estimator for a broad class of spatial coverage objectives using a combination of Newton-Leibniz theorem and implicit boundary differentiation. This alleviates the need to use function approximators like neural networks to approximate gradients of the coverage objectives. We further propose a tractable framework to approximate the coverage objectives and their gradients using spatial discretization of only the target domain, but not the allocated positions of the resources. Hence, we keep the resource allocations amenable to gradient-based optimization thereby leading to faster, scalable and more directed ways of search and optimization for multi-resource coverage problems. By combining our framework with existing optimization methods, we demonstrate successful applications on both surveillance and green security spatial coverage domains.

7.2 Methods

7.2.1 Multi-resource spatial coverage problems

We adopt the multi-resource spatial coverage model as defined in sections 3.1.8 and 3.1.9. We will be using the surveillance game (section 3.3.5) and the adversarial coverage game (section 3.3.6) as our application domains for this work.

To maximize coverage, the key idea behind our solution approach is to obtain the gradient of the expected coverage reward of the agent/(s) w.r.t. the agents' actions. This can then be used to perform direct gradient ascent to arrive at a (locally) optimal action or as a part of other global search algorithms.

7.2.2 Differentiable approximation for coverage objectives

First, we propose a method to approximate coverage objectives and their gradients w.r.t. agents' actions. Consider an objective of the form:

$$r(u) = \int_Q f(u, q)\, dq \tag{7.1}$$

where u denotes actions of one or more agents having multiple resources to place at their disposal and q is any point in the target domain Q. We assume that the action u has m components with u_i representing the location of i-th resource ($i \in [m]$) and $u_{\setminus i}$ representing the locations of all resources other than i. Note that the $\text{imp}(q)$ function has been subsumed into $f(u, q)$ in this formulation.

We are interested in computing the gradient: $\frac{\partial r}{\partial u_i}$. However, this is a hard problem since: (a) $r(u)$ involves integration over arbitrary (non-convex shaped) target domains which does not admit a closed-form expression in terms of elementary functions and hence cannot be differentiated with autograd libraries like PyTorch and TensorFlow, and (b) most resources have a finite coverage area,

outside of which the coverage drops to zero. This often makes the function $f(u, q)$ discontinuous w.r.t. q given a fixed u especially at the coverage boundaries induced by the resources' coordinates, for e.g., drones have a circular probabilistic coverage area governed by their height and camera half-angle θ, outside which the coverage probability suddenly drops to zero.

Theorem 1 (Coverage Gradient Theorem)**.** *Let the objective function be as shown in eq 7.1:* $r(u) = \int_Q f(u, q)\, dq$. *Denoting the set of points covered by the i-th resource as* S_i*, the interior of a set with* $in(\cdot)$ *and the boundary with* $\delta(\cdot)$*, the gradient of* $r(u)$ *w.r.t. the i-th resource's location* u_i *is given by:*

$$\frac{\partial r(u)}{\partial u_i} = \int_{in(Q \cap S_i)} \frac{\partial f(u, q)}{\partial u_i}\, dq + \int_{Q \cap \delta S_i} \big(f(u, q) - f(u_{\setminus i}, q)\big) \frac{\partial q_{Q \cap \delta S_i}}{\partial u_i}^T n_{q_{Q \cap \delta S_i}}\, dq \quad (7.2)$$

Proof. We begin by observing that while function f can be potentially discontinuous in q across resources' coverage boundaries due to finite coverage fields of resources, $r(u)$ integrates over $q \in Q$ thereby removing the discontinuities. Hence, instead of directly taking the derivative w.r.t. a particular resource's location u_i inside the integral sign, we first split the integral into two parts - over the i-th resource's coverage area S_i and outside it:

$$r(u) = \int_{Q \cap S_i} f(u, q)\, dq + \int_{Q \setminus S_i} f(u, q)\, dq \quad (7.3)$$

Splitting the integral at the boundary of the discontinuity allows us to explicitly capture the effect of a small change in u_i on this boundary. Denoting the interior of a set with $\text{in}(\cdot)$ and the boundary with $\delta(\cdot)$, the derivative w.r.t. u_i can be expressed using the Newton-Leibniz formula as:

$$\begin{aligned}\frac{\partial r(u)}{\partial u_i} = &\int_{\text{in}(Q\cap S_i)} \frac{\partial f(u,q)}{\partial u_i}\, dq \\ &+ \int_{\delta(Q\cap S_i)} f(u,q) \frac{\partial q_{\delta(Q\cap S_i)}}{\partial u_i}^T n_{q_{\delta(Q\cap S_i)}}\, dq \\ &+ \int_{\text{in}(Q\setminus S_i)} \frac{\partial f(u_{\setminus i},q)}{\partial u_i}\, dq \\ &+ \int_{\delta(Q\setminus S_i)} f(u_{\setminus i},q) \frac{\partial q_{\delta(Q\setminus S_i)}}{\partial u_i}^T n_{q_{\delta(Q\setminus S_i)}}\, dq,\end{aligned} \tag{7.4}$$

where $\frac{\partial q_{\delta(Q\cap S_i)}}{\partial u_i}$ denotes the boundary velocity for $\delta(Q\cap S_i)$ and $n_{q_{\delta(Q\cap S_i)}}$ denotes the unit-vector normal to a point q on the boundary $\delta(Q\cap S_i)$ (similarly for $\delta(Q\setminus S_i)$). Since $f(u_{\setminus i},q)$ does not depend on u_i, we can set $\frac{\partial f(u_{\setminus i},q)}{\partial u_i} = 0$. Next observe that the boundaries can be further decomposed as: $\delta(Q\cap S_i) = (\delta Q\cap S_i)\cup(Q\cap \delta S_i)$ and similarly $\delta(Q\setminus S_i) = (\delta Q\setminus S_i)\cup(Q\cap\delta S_i)$. However since u_i does not change the boundary of the target domain δQ, we have:

$$\frac{\partial q_{\delta Q\cap S_i}}{\partial u_i} = 0, \quad \forall q\in \delta Q\cap S_i \tag{7.5}$$

$$\frac{\partial q_{\delta Q\setminus S_i}}{\partial u_i} = 0, \quad \forall q\in \delta Q\setminus S_i \tag{7.6}$$

Further on the boundary of S_i, the following unit-vectors normal to the boundary are oppositely aligned:

$$n_{q_{\delta(Q\setminus S_i)}} = -n_{q_{\delta(Q\cap S_i)}} \;\; \forall q\in Q\cap\delta S_i. \tag{7.7}$$

Substituting the above results, we can simplify the gradient expression in eq 7.4 to:

$$\frac{\partial r(u)}{\partial u_i} \quad = \quad \int_{\text{in}(Q \cap S_i)} \frac{\partial f(u,q)}{\partial u_i}\, dq \;+\; \int_{Q \cap \delta S_i} \big(f(u,q) \;-\; f(u_{\setminus i}, q)\big) \frac{\partial q_{Q \cap \delta S_i}}{\partial u_i}^T n_{q_{Q \cap \delta S_i}}\, dq \quad (7.8)$$

□

Note that the first term in eq 7.2 corresponds to the change in f inside the coverage area of resource i due to a small change in u_i, while the second term elegantly factors-in the effects of movement or shape change of the coverage area boundary due to changes in u_i (e.g. when a drone moves or elevates in height). This allows us to mitigate the discontinuities due to finite coverage fields of resources. While we show the general result here, the term $\frac{\partial q_{Q \cap \delta S_i}}{\partial u_i}^T n_{q_{Q \cap \delta S_i}}$ can be simplified further using implicit differentiation of the boundary of S_i, which depends on the particular game under consideration. We show the simplification for our example domains in the next section.

7.2.3 Implicit boundary differentiation for gradient simplification

The term $\frac{\partial q_{Q \cap \delta S_i}}{\partial u_i}^T n_{q_{Q \cap \delta S_i}}$ from eq 7.2 can be simplified further using implicit differentiation of the boundary of S_i. In our example domains, the coverage boundaries induced by all resources (drones or lumberjacks) are circular. With the location of i-th drone as $u_i = \{p_i, h_i\}$ and for the j-th lumberjack as $u_j = p_j$, the boundaries are given as:

$$\delta S_i = \{q \,|\, ||q - p_i||_2 = h_i \tan\theta\} \quad \text{for drones, and}$$

$$\delta S_j = \{q \,|\, \|q - p_j\|_2 = R_L\} \quad \text{for lumberjacks}$$

We illustrate the calculation of the $\frac{\partial q_{Q\cap\delta S_i}}{\partial u_i}^T n_{q_{Q\cap\delta S_i}}$ term for a drone below and the calculation follows similarly for lumberjacks. Any point $q \in Q \cap \delta S_i$ satisfies:

$$||q - p_i||_2 = h_i \tan\theta$$

Differentiating this boundary implicitly w.r.t. p_i and w.r.t. h_i gives:

$$\left(\frac{\partial q}{\partial p_i}^T - I_2\right) \frac{q - p_i}{||q - p_i||_2} = 0, \text{ and}$$
$$\frac{\partial q}{\partial h_i}^T \frac{q - p_i}{||q - p_i||_2} = \tan\theta.$$

Noting that the outward normal n_q at any point $q \in Q \cap \delta S_i$ is given by $\frac{q-p_i}{||q-p_i||_2}$, we now have:

$$\begin{aligned}\frac{\partial q}{\partial u_i}^T n_q &= \left\{ \left(\frac{\partial q}{\partial p_i}^T n_q\right)^T, \frac{\partial q}{\partial h_i}^T n_q \right\} \\ &= \left\{ \left(\frac{q - p_i}{||q - p_i||_2}\right)^T, \tan\theta \right\}\end{aligned}$$

7.2.4 Discretization-based Approximation Framework

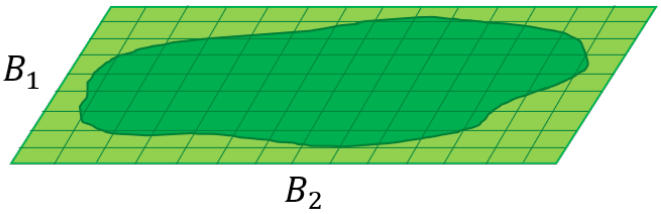
While we now have a general form for $r(u)$ and $\frac{\partial r}{\partial u}$, both forms comprise of non closed-form integrals over the target domain Q or its subsets. While evaluating r and $\frac{\partial r}{\partial u}$ in practice, we adopt a discretization based approach to approximate the integrals. Given a target domain $Q \subset \mathbb{R}^d$ with $d \in \{2, 3\}$, we discretize the full $\mathbb{R}^d$ space into $B_1, \ldots, B_d$ bins respectively in each of the d dimensions (see figure 7.2a).

Approximating spatial maps: All spatial maps i.e. functions over the target domain Q (e.g. $f(u, q)$), are internally represented as *real tensors* of dimension d with size: $(B_1, \ldots, B_d)$ (see figure 7.2b).

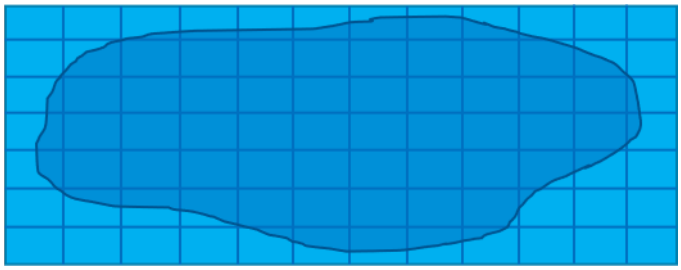
Approximating sets: All geometric shapes (or sets of points) including S_i for all resources

(e.g., the circular coverage areas of drones and lumberjacks) and the target domain Q itself (e.g., the irregular shaped forest) are converted to *binary tensors* each of dimension $d+1$ with size: $(B_1, \ldots, B_d, 3)$. The final dimension of length 3 denotes interior, boundary and exterior of the geometric shape respectively, i.e. a binary tensor T has $T_{b_1,\ldots,b_d,0} = 1$ if the bin at index $(b_1, \ldots, b_d)$ is inside the geometric shape, $T_{b_1,\ldots,b_d,1} = 1$ if the bin is on the boundary of the geometric shape and $T_{b_1,\ldots,b_d,2} = 1$ if the bin is outside the geometric shape (see figure 7.2c).

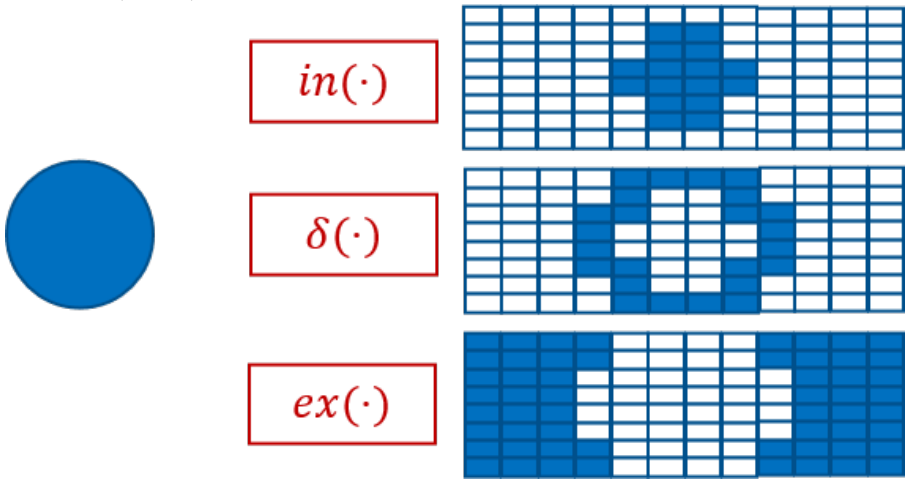
Approximating operators: Doing the above discretization requires an efficient function for computing the *binary tensors* associated with the $\text{in}(\cdot)$ and the $\delta(\cdot)$ operators. This is performed by our efficient divide-and-conquer shape discretizer, which is presented in section B.2 in the appendix for brevity. The other set operations are approximated as follows: (a) set intersections are performed by element-wise *binary tensor* products, (b) integrals of spatial maps over geometric sets are approximated by multiplying (i.e. masking) the *real tensor* corresponding to the spatial map with the *binary tensor* corresponding to the geometric set followed by an across-dimension sum over the appropriate set of axes.



(a) Discretize the target space $\mathbb{R}^d$ (but not action space) into bins



(b) Approximate all spatial maps e.g., $f(u, q)$ as real tensors of shape (B_1, B_2)



(c) Approximate all sets e.g., spatial coverage field S_i of each resource and the target domain Q as binary tensors of shape $(B_1, B_2, 3)$

Figure 7.2: Illustration of spatial discretization-based framework for 2-D target domains.

Scaling: While our discretized bins growing exponentially with dimension d of the target domain may come off as a limitation, our method still scales well for most real-world coverage problems since they reside on two or three-dimensional target domains. Note that unlike previous approaches which discretize the target domain and simultaneously restrict the agents' actions to discrete bins [142, 46], we do not discretize the actions u of agents. Hence, we do not run into

intractability induced by discretizing high-dimensional actions of agents owning multiple resources and we keep u amenable to gradient-based optimization.

Using the framework: Our proposed framework essentially acts as an autograd module for $r(u)$ differentiable w.r.t. input u, which provides both the forward and the backward calls (i.e. evaluation and gradients). Hence, it can now be used for direct gradient-based optimization solutions to multi-resource coverage problems. We describe our solution approaches in the next section.

7.2.5 Solution Approaches

For the single agent surveillance domain, we compare the following solution approaches:

1. *Genetic algorithm* [*gen*]: We run a genetic algorithm as shown in algorithm 4 to search for near-optimal resource allocations (with population size $K = 6$ and $max\_itr = 1000$).

2. *Gradient ascent* [*ga*]: We perform gradient ascent on a differentiable approximation to the coverage objective $r_D(u_D)$, thereby converging at a (locally) optimal value of u_D:

   (a) *Neural nets* [*_nn*]: We train feedforward neural networks to approximate the coverage objective and its gradients.

   (b) *Graph neural nets* [*_gnn*]: We train graph neural networks to approximate the coverage objective and its gradients.

   (c) *Our framework* [*_diff*]: We use our spatial discretization based framework and the coverage gradient theorem to approximate the coverage objective and its gradients.

3. *Augmented genetic algorithm* [*agen*]: We augment the genetic algorithm as shown in algorithm 4, line 11 by having an inner-loop which performs gradient ascent on all population members in every iteration of the algorithm. We use population size $K = 6$, $max\_itr = 1000$ and 100 inner-loop gradient ascent iterations. We again have the three variants: [*_nn*], [*_gnn*] and [*_diff*] based on where the gradients come from.

For two-agent adversarial games, we employ the DeepFP algorithm [67], which is based on fictitious play. Briefly summarized in algorithm 5, it obtains a differentiable approximation to the reward functions $r_{D,2p}$ and $r_{A,2p}$, creates an empty memory to store a non-parametric representation of the agents' mixed strategies $\sigma = (\sigma_D, \sigma_A)$ and initializes best responses for both agents randomly [lines 1-3]. Then it alternatively updates: (a) the agents' strategies, by storing the current best responses in the memory [line 5], and (b) the best responses, by maximizing each agent p's differentiable reward function against a batch of samples drawn from the other agent's strategy σ_{-p} [lines 6-8]. We point the readers to [67] for details of the algorithm. In our implementation, we used a modified version of the DeepFP algorithm to apply it to our setting. The modifications made and the reasons behind them have been described in detail in section 7.2.6. The DeepFP hyperparameters used can be found in section B.1 in the appendix. Again we use neural nets [*_nn*], graph neural nets [*_gnn*] and our approximation framework [*_diff*] to obtain the gradients of the coverage objective and compare these variants empirically.

7.2.6 Modifications to DeepFP

Dealing with zero gradients: In the two-agent game (example 2), the attacker's reward depends on the locations of its resources, but the defender's reward solely depends on overlaps with the attacker's resources. In absence of such overlap, the gradient of $r_{D,2p}$ w.r.t. $u_{D,i}$ becomes 0. Hence, we use the reward from the one-agent game (example 1) as an intrinsic reward for the defender similar to how RL algorithms employ intrinsic rewards when extrinsic rewards are sparse [97]. Then the reward function for the defender becomes: $\tilde{r}_{D,2p}(u_D, u_A) = r_{D,2p}(u_D, u_A) + \mu r_{D,1p}(u_D)$. We use a small $\mu = 0.001$ to not cause significant deviation from the zero-sum structure of the game and yet provide a non-zero gradient to guide the defender's resources in the absence of gradients from $r_{D,2p}$.

Mitigating sub-optimal local optima in best responses: During our preliminary experiments, we observed that learning to optimize resource locations or mixed strategies using

Algorithm 4: A Genetic Algorithm for Resource Allocation in Spatial Coverage Problems

Result: Final action u

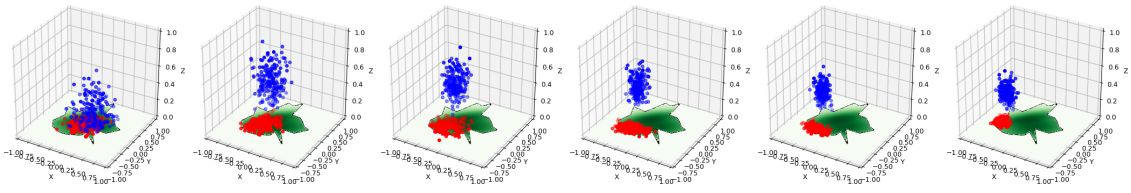
Required: Coverage reward $r(u)$ (a.k.a. fitness function);
Initialize a population of K actions $u_{1:K}$ each $\in \mathbb{R}^{m \times d}$;
for $itr \in \{1, \ldots, max\_itr\}$ **do**
    `/* Evaluate population members */`
    Compute fitness $r(u_i)$ of population member u_i $\forall i \in 1:K$;
    `/* Ranking */`
    Sort all population members in decreasing order of fitness;
    `/* Cross-over */`
    Copy the top $K/3$ fittest population members;
    Make a shuffled copy of these top $K/3$ members;
    Between each pair of the original and shuffled copies, swap the corresponding resource placements with probability 0.5 generating 2 new members per pair;
    Discard the bottom $2K/3$ population and replace them with the newly crossed-over copies;
    `/* Perform mutation */`
    Randomly perturb the coordinates of the newly generated $2K/3$ copies by appropriate amounts (we use uniform random numbers between $[-0.1, 0.1]$ per coordinate);
    `/* Perform inner-loop gradient ascent if augmented genetic algorithm */`
    In the augmented genetic algorithm variant, apply a fixed number of gradient ascent iterations to each population member using gradients from a differentiable approximation $\hat{r}(u)$ to $r(u)$;
Return $\arg\max_{u \in u_{1:K}} \hat{r}(u)$;

Algorithm 5: DeepFP

Result: Final strategies σ_D, σ_A in `mem`

Obtain a differentiable approximation $\hat{r} = (\hat{r}_D, \hat{r}_A)$ to the reward functions: ($r_{D,2p}$, $r_{A,2p}$);
Initialize best responses (br_D, br_A) randomly;
Create empty memory `mem` to store $\sigma = (\sigma_D, \sigma_A)$;
for $game \in \{1, \ldots, max\_games\}$ **do**
    `/* Update strategies */`
    Update σ by storing best responses $\{br_D, br_A\}$ in `mem`;
    `/* Update best responses */`
    for *agent* $p \in \{D, A\}$ **do**
        Draw samples $\{u^i_{-p}\}_{i=1:bs}$ from σ_{-p} in `mem`;
        $br_p := \max_{u_p} \frac{1}{bs} \sum_{i=1}^{bs} \hat{r}_p(u_p, u^i_{-p})$;

purely gradient-based optimization can easily get stuck in local minima. While multiple re-runs in single-agent games can generate a reasonably good local minimum, in multi-agent games where the loss functions of agents are non-stationary due to changes in the other agents' mixed strategies, this leads to agents getting stuck in very sub-optimal local best responses. DeepFP maintains stochastic best responses to partially alleviate this issue, but doesn't completely mitigate it (for an example,



(a) Iter 0 (b) Iter 200 (c) Iter 400 (d) Iter 800 (e) Iter 1300 (f) Iter 1900

Figure 7.3: A sample sequence of iterations for DeepFP with $m = n = 1$ to demonstrate the attacker's best responses getting stuck in non-stationary local minima generated due to eventual adaptation by the defender; The drone (blue dots sampled from the defender's stochastic best response) eventually drives the lumberjack (red dots) into a corner from where it cannot cross over to other parts of the forest, because gradient-based optimization cannot jump over walls of high loss values.

see Figure 7.3). While computing a global best response at every iteration of DeepFP can be costly (often infeasible), in practice it suffices to have a discontinuous exploration technique available in the best response update step. Hence, we propose a simple population-based approach wherein, motivated by [87], we maintain a set of K deterministic best responses $br_p^k(\sigma_{-p})$, for $p \in \{D, A\}$ and $\forall k \in [K]$. During the best response optimization step for agent p [lines 6-8 in algorithm 5], we optimize the K best responses independently and play the one which exploits agent $-p$ the most. After the optimization step, the top $\frac{K}{2}$ best responses are retained while the bottom half are discarded and freshly initialized with random placements for the next iteration. This allows retention and further refinement of the current best responses over subsequent iterations, while discarding and replacing the ones stuck due to the opponent exploiting them. Since best responses get re-ranked every iteration, neither agent can excessively exploit a best response and cause the opponent to get stuck, because the opponent just switches to a different best response from its population in subsequent iterations.

7.3 Experiments

In our experiments on both our application domains, we differentiably approximate rewards using the following variants: (a) feedforward neural networks [*nn*], (b) graph neural networks [*gnn*], and (c) our approximation framework [*diff*]. For the *nn* and *gnn* baselines, we trained neural

networks, one per forest and per value of m (and n for two-agent games), to predict the reward of the defender (and attacker in case of two-agent game) by minimizing the MSE loss using the Adam optimizer. The neural networks take as input the action u_D of the defender (and u_A also for two-agent game) and output a prediction for the reward $\hat{r}_{D,1p}$ ($\hat{r}_{D,2p}$ and $\hat{r}_{A,2p}$ for two-agent game). Please see section B.1 in appendix for network architectures and hyperparameters. We also represent best responses with the following variants: (a) stochastic best response nets [*brnet*] as originally done by DeepFP, and (b) our deterministic evolutionary population [*popK*] with K being the population size (see section 7.2.6 for why this modification is useful). We use $d = 2$ dimensional forests and discretize them into $B_1 = B_2 = 200$ bins per dimension for a total of $40K$ bins when using our framework.

Table 7.1: Maximum reward averaged across forest instances achieved for Areal Surveillance domain.

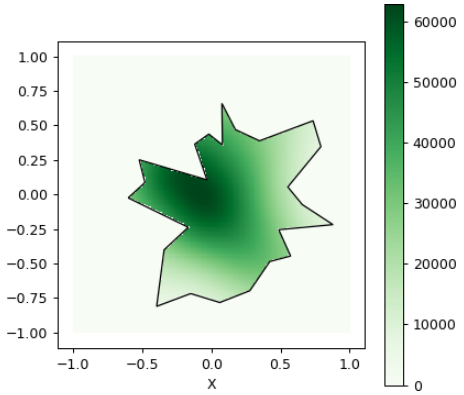
	$m = 1$	$m = 2$	$m = 4$	$m = 8$
gen	**9378.46**	16061.02	24857.09	33749.89
	± 660.27	±940.34	±1593.90	±2949.36
ga_diff	9364.07	**16086.24**	**25109.58**	**34364.64**
	±660.55	**± 923.84**	**± 1552.05**	**± 3168.55**
ga_nn	9337.57	14308.12	19211.01	19127.45
	±680.45	±1070.00	±2233.19	±2498.12
ga_gnn	9291.36	14082.38	19075.09	19657.22
	±665.65	±1073.62	±1378.36	±2346.17
agen_diff	**9374.36**	**16091.18**	**25122.13**	**34792.45**
	± 660.56	**± 927.46**	**± 1555.55**	**± 2924.52**
agen_nn	9351.67	14348.55	19236.34	19563.83
	±674.55	±1057.19	±2229.72	±2378.31
agen_gnn	9307.41	14207.96	19652.45	20286.63
	±676.78	±1044.29	±1712.13	±2339.48

7.3.1 Results on Areal Surveillance domain

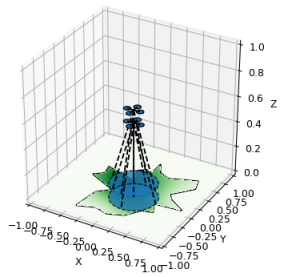
We show the experiment results achieved by using all methods: *gen*, *ga_diff*, *ga_nn*, *ga_gnn*, *agen_diff*, *agen_nn* and *agen_gnn* for different values of $m \in \{1, 2, 4, 8\}$ over 5 different forest instances differing in shape and tree density. The maximum true reward $r_{D,1p}$ achieved by all methods averaged over all the forest instances is summarized in Table 7.1. It is clear that *agen_diff*

always achieves the maximum true reward for nearly all values of m (except $m = 1$ due to stochasticity of genetic algorithms). This is because *gen* only performs undirected global search, while the *ga* variants perform only directed local optimization with gradient ascent. The *agen* variants are the only ones which combine the undirected global search of genetic algorithms with local optimization of gradient-based optimization and hence outperform other baselines. Figure 7.4 shows the final locations computed for a randomly chosen forest and with $m = 2$ for all methods. Amongst the *diff*, *nn* and *gnn* variants, the *diff* variants always outperform the other two since our approximation framework is quite precise while neural networks become more inaccurate at approximating the coverage objective and its gradients, especially as m increases and the objective becomes combinatorially harder to approximate. This is also reflected in the plots of true reward achieved vs training iterations shown in Figure 7.5 for simple gradient ascent (*ga*) variants. Since *diff* variants are unbiased approximators of the true reward[1], the true reward continues to increase till convergence for *diff*. For *nn* and *gnn* variants, the true reward increases initially but eventually goes down as the defender action u_D begins to overfit the potentially inaccurate approximations made by *nn* and *gnn*.

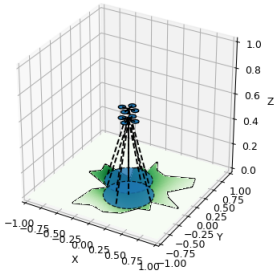
[1]The only bias in *diff* is the discretization bin sizes, which can be made arbitrarily small in principle.



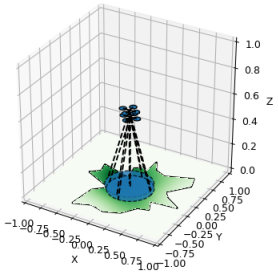
(a) Forest tree density



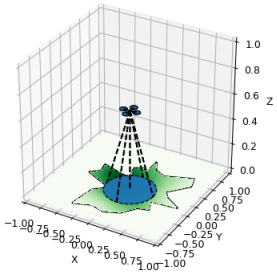
(b) Action found via *gen*



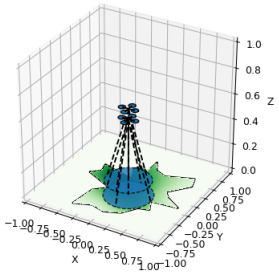
(c) Action found via *ga_diff*



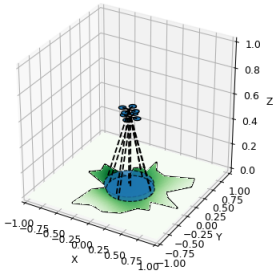
(d) Action found via *ga_nn*



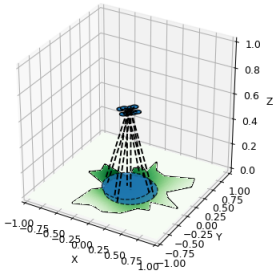
(e) Action found via *ga_gnn*



(f) Action found via *agen_diff*

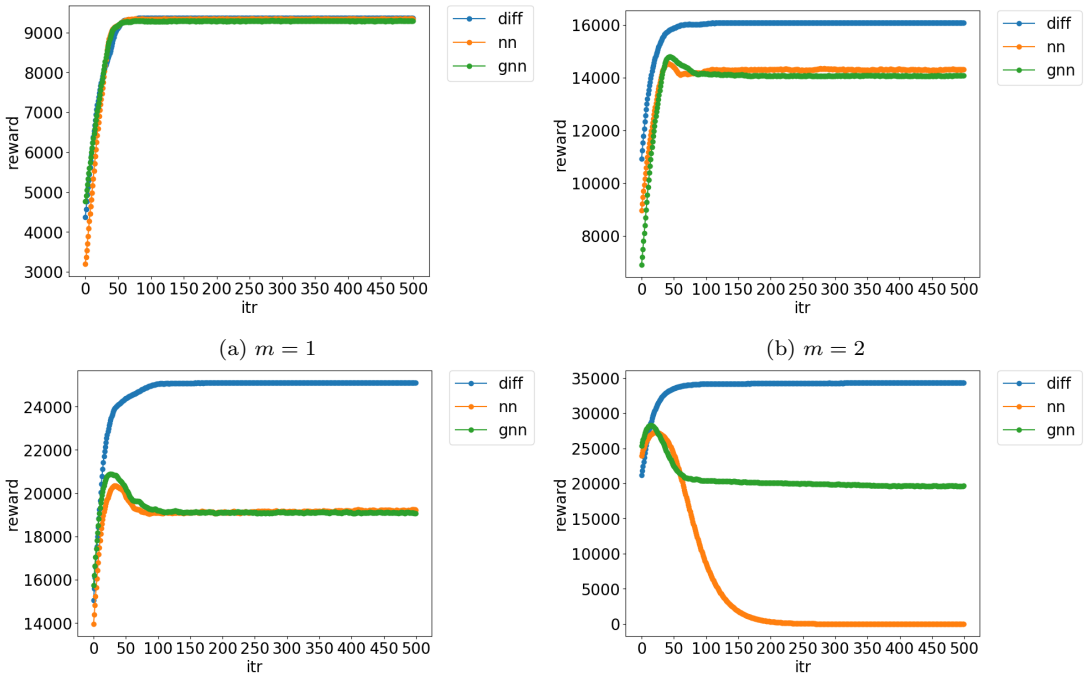


(g) Action found via *agen_nn*



(h) Action found via *agen_gnn*

Figure 7.4: Visualizing final actions for a randomly chosen forest with $m = 2$.



(a) $m = 1$ (b) $m = 2$

(c) $m = 4$ (d) $m = 8$

Figure 7.5: Plots of true reward achieved by *diff*, *nn* and *gnn* variants over gradient ascent iterations for $m \in \{1, 2, 4, 8\}$.

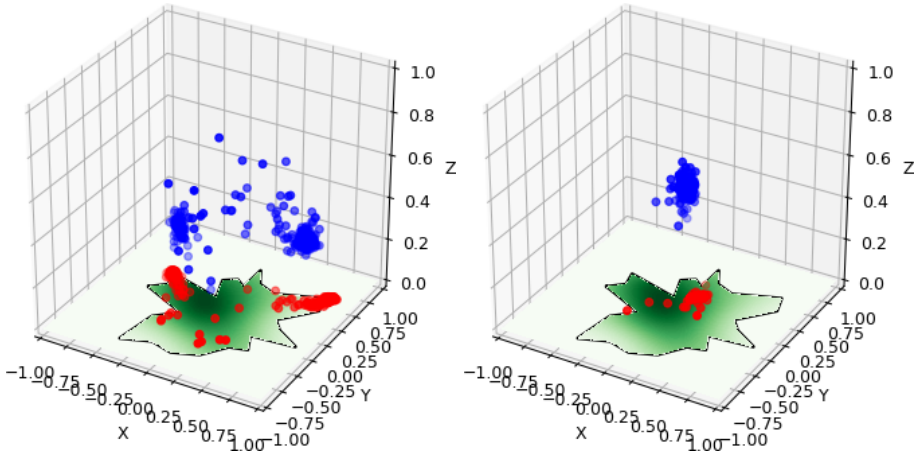
7.3.2 Results on Adversarial Coverage game

We implemented different variants of DeepFP with variations of differentiable reward models in $\{nn, gnn, diff\}$ along with variations of best responses in $\{brnet, pop4\}$. We measured the exploitability $\epsilon_D(\sigma_D)$ of the defender strategy found by all methods to compare them against each other. To compute the exploitability of the defender strategy found by any variant of DeepFP, we froze the defender strategy σ_D and directly maximized $\mathbb{E}_{u_D \sim \sigma_D}[\hat{r}_A(u_D, u_A)]$ w.r.t. u_A with $\hat{r}_A$ being approximated by *diff*. This is a single-agent objective and can be directly maximized with gradient ascent. We perform 30 independent maximization runs to avoid reporting local maxima and report the best of them as the exploitability. Note that nash equilibrium strategies are the least exploitable strategies, hence the lower the value of $\epsilon_D(\sigma_D)$ found, the closer σ_D is to the nash equilibrium strategy.

Table 7.2 shows the exploitability values for different variants of DeepFP. We observe that the exploitability when best responses are approximated by a population-based variant with $K = 4$ is always lower than that of stochastic best response networks employed by original DeepFP. Further, with few agent resources $m = n = 1$, the exploitability across *diff, nn* and *gnn* is nearly similar but the disparity increases for larger number of agent resources and *diff* dominates over *nn* and *gnn* with less exploitable defender strategies. Notably, the original DeepFP (*nn + brnet*) is heavily exploitable while our proposed variant (*diff + popK*) is the least exploitable. In Figure 7.6, we show a visualization of the points sampled from the defender and attacker's strategies for $m = n = 2$ case on the same forest from Figure 7.4a. The visualization confirms that *diff + popK* covers the dense core of the forest with the defender's drones so the attacking lumberjacks attack only the regions surrounding the dense core, while *nn + brnet* drones often gets stuck and concentrated in a small region thereby allowing lumberjacks to exploit the remaining dense forest.

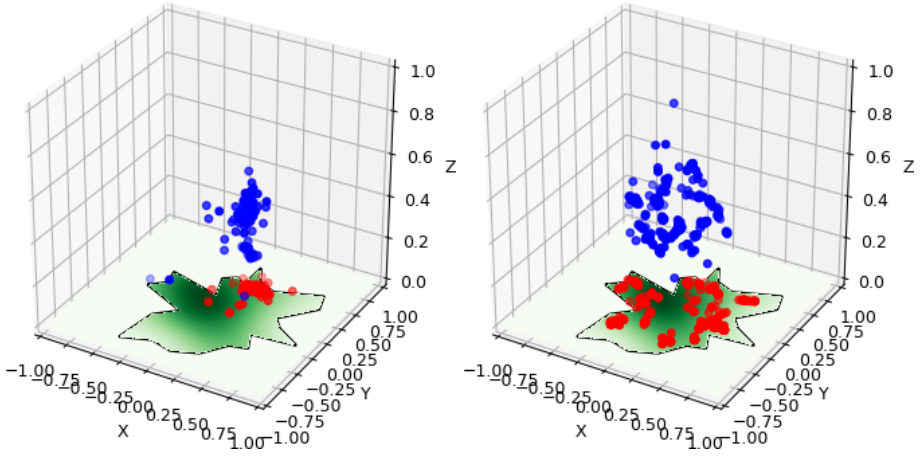
Table 7.2: Exploitability of the defender from DeepFP variants averaged across forest instances.

$\epsilon_D(\sigma_D)$	m=n=1	m=n=2	m=n=4
		brnet	
diff	209.78	399.95	559.36
(ours)	±49.94	±57.70	±164.21
nn	203.92	323.00	787.53
	±54.67	±39.55	±194.82
gnn	204.55	307.74	597.23
	±50.72	±62.67	±125.01
		pop4 (ours)	
diff	116.41	**141.09**	**141.54**
(ours)	±15.02	± **13.90**	± **26.60**
nn	**113.61**	208.23	339.31
	± **6.92**	±22.76	±116.77
gnn	113.99	176.25	172.30
	±13.74	±15.21	±34.08



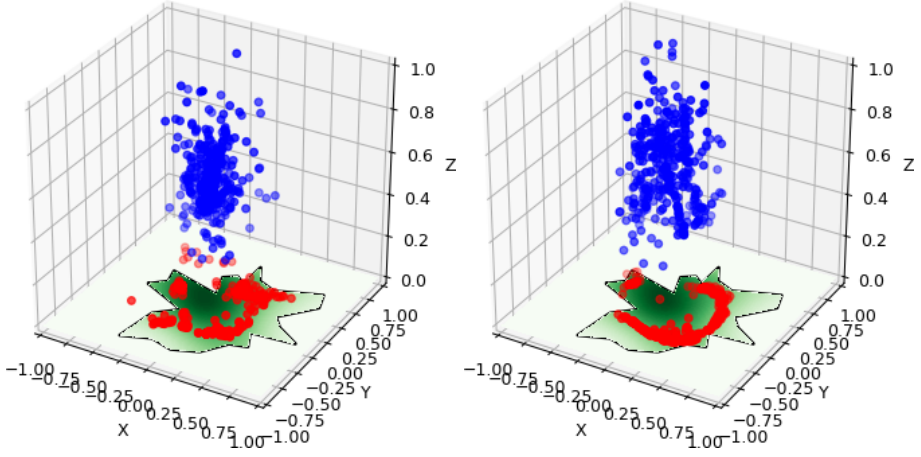
(a) Strategy for *diff* + *brnet*

(b) Strategy for *nn* + *brnet*



(c) Strategy for *gnn* + *brnet*

(d) Strategy for *diff* + *pop4*



(e) Strategy for *nn* + *pop4*

(f) Strategy for *gnn* + *pop4*

Figure 7.6: Visualizing final strategies found via *diff*, *nn* and *gnn* with best responses of the form *brnet* and *pop4* on a randomly chosen forest with $m = n = 2$. The blue (red) dots are sampled from the defender's (attacker's) strategy for the 2 drones (lumberjacks).

Finally since the number of population members K is an important hyperparameter for our proposed approach, we show the effect on defender's exploitability by increasing K in Table 7.3. As expected, the exploitability decreases when using larger population sizes due to better exploration and finding more optimal (local) best responses while running DeepFP. Increasing K also reduces the variance of our metrics considerably. However using large population sizes also directly increases the computational burden and hence we have used $K = 4$ in all our experiments as a reasonable trade-off between achieving better metrics and having manageable run-times.

Table 7.3: Exploitability of defender for $m = n = 2$ averaged across forest instances with increasing population size K.

Variant	$\epsilon_D(\sigma_D)$
brnet	399.9488 ± 57.7006
pop1	348.9498 ± 98.4338
pop2	189.8122 ± 73.6444
pop4	141.0912 ± 13.8966
pop6	$\mathbf{127.9152 \pm 12.8323}$

7.4 Summary

In this chapter, we propose the Coverage Gradient Theorem to directly compute the gradients of a large class of multi-resource spatial coverage objectives. We also provide a tractable and scalable spatial discretization-based framework to approximate the resulting gradient expressions. By augmenting existing approaches with our approximation framework, we show improved performance in both single-agent and adversarial two-agent multi-resource spatial coverage problems.

One of the key limitations of the approximation framework is to approximate the integrals using discretization. While this scales well for two or three dimensional target domains, it is harder to scale if we are working with target spaces of larger dimensions and one needs to explore alternative but less accurate methods, e.g., sampling. However, while it is much more manageable to store discretized shape tensors in GPU memory, while working with samples from geometric shapes and defining operators on them is generally harder. Further, while our framework scales linearly

with the number of resources for single player games, the size of the spatial maps and binary tensors involved depends on the number of bins chosen per dimension of the target domain. This number can be large if a fine-grained discretization is being used or the target space is huge and can require multiple GPUs in parallel to store the full forward and backward models. To obtain the best trade-off between memory and parallelization on GPUs, working on scalable adaptive sampling-based frameworks is a promising next step for future research.

Chapter 8

Conclusion

8.1 Summary of current work

In this thesis, I have presented several key multi-agent learning problems, namely, multi-agent prediction, multi-agent control and multi-agent credit allocation and proposed solutions to advance the state-of-the-art for them. Since studying these challenging problems arising in multi-agent systems in general without any specific focus is hard, we chose specific domains to study each problem.

Specifically, we first studied interaction modeling via the multi-agent trajectory prediction problem which occurs extensively in human crowds, traffic modeling, physics and sports analytics domains. We analyzed the key inductive biases required for motion prediction and presented an architecture which incorporates all the required inductive biases. The primary focus was on capturing interactions amongst multiple agents and being able to represent continuous-valued fuzzy decision making via the architecture. To address this, we designed a novel attention mechanism called the Fuzzy Query Attention (FQA) which provides our architecture and its superior performance capabilities over other competing baselines in many diverse domains.

Next we studied multi-agent interaction in order to learn individual policies for the players. We proposed the algorithms OptGradFP and DeepFP for two-player adversarial stackelberg

security games. OptGradFP aims to compute optimal defender strategies for spatial security games with continuous action spaces. It is a novel and general model-free learning algorithm which implements approximate fictitious play. DeepFP is a model-based strategy learning algorithm which addresses several challenges present in OptGradFP and improves upon it. We demonstrated stable convergence to Nash equilibrium on several classic games and also applied our methods to a large forest security domain thereby demonstrating the robustness of the computed strategies against adversarial exploitation.

Lastly, we analyzed the problem of credit allocation in multi-agent systems. We focused on learning optimal spatial coverage with continuous and differentiable reward prediction models of a multi-resource system, in which backpropagation can allow for credit assignment during placement. While it is not always possible to simplify the design of differentiable reward models, we considered the problem of designing differentiable reward models for the specific domain of multi-resource spatial coverage and tackled some of the common challenges which make the reward models non-differentiable in this domain. Here we introduced the *coverage gradient theorem*, which provides a gradient estimator for a broad class of spatial coverage objectives using a combination of Newton-Leibniz theorem and implicit boundary differentiation. This allowed differentiable credit assignment for the placement of different resources towards a given coverage objective. We also proposed a tractable framework to approximate the coverage objectives and their gradients using spatial discretization.

8.2 New challenges

The proposed approaches in this thesis also opened up many new challenges. In this section, I will discuss some of these challenges that I encountered during my research and why it is important to address these challenges.

8.2.1 Scaling due to quadratically growing interactions

Firstly and most importantly, all multi-agent learning setups deal with interactions between multiple agents. In a setup with N agents, if one considers all possible pairwise interactions, this results in $O(N^2)$ interactions. This is independent of which multi-agent system one considers, be it trajectory prediction, multi-resource spatial coverage or security games. Having such quadratic growth in the number of interactions can often be the key bottleneck in scaling multi-agent solution approaches since N^2 grows super-linearly with the number of agents/entities N. Hence it is important to address this issue and research solutions which allow us to reduce the number of interactions considered given a system with N agents.

8.2.2 Pitfalls of learning with game models

In chapter 7, we showed that using our differentiable approximation for spatial coverage domains results in much better resource allocations as compared to when one uses neural network based learnt approximations. This key observation reveals the bias neural network based learnt approximations can suffer when employed for single-agent or multi-agent reinforcement learning. In general, it is true that using any learnt differentiable approximations to a reward function and directly backpropagating through it can lead to poor performance due to the learnt model hallucinating artifacts which do not actually exist in the real system [41]. Hence, learning with fictitious play based methods like DeepFP can benefit substantially from: (a) either better ways of learning reward models or, (b) better ways of using potentially inaccurate learnt reward models. Both these challenges are in general still open research directions.

8.2.3 Addressing solutions for large spatial coverage domains

While we have primarily focused on scaling of our proposed methods with number of agents in this thesis, in certain multi-agent settings there are other factors to consider towards scalability

of a proposed approach. For instance, in the spatial coverage problem the size of the target domain being covered can often be a key bottleneck towards scalability. Consider as an example application, the placement of medical testing sites during a pandemic like the recent Covid-19 disease caused by the SARS-CoV-2 virus. In such a case, the testing sites can be considered as resources to be placed while the population density infected with the disease at a given location can be considered as target density. However, since a testing site covers a small geographical location in practice, e.g., a 10 mile radius, allocating such testing sites for a large city or a single state of the United States can lead to very large discretized tensors in our approach, especially if a fine-grained discretization is required. Addressing scalability in such cases can be an interesting future challenge and new frameworks to approximate the *Coverage Gradient Theorem* might be required in this case.

8.2.4 Games where agents do not know each others' objectives

Lastly, while researching multi-agent control, this thesis primarily deals with adversarial learning in security games between two agents. These assume a zero-sum objective for the two agents, i.e., each agent also knows the other agent's goal. Hence fictitious play and its extensions (OptGradFP and DeepFP) are viable algorithms here. However, many practical games in real life can be cooperative (non zero-sum) in nature. Further, the two agents may not even know each others' goals. An example of such a domain can be the design of a virtual reality (VR) assistive agent residing in a pair of VR glasses and potentially attached to a human. The goal of the VR agent is to assist the human in his/her day-to-day life. In such a case, the human is the first agent and he/she knows his/her goals, while the VR glass is the second agent and is not aware of the human's true goals at any instant. This implies that the VR glass agent now requires observation and inference capabilities built into its learning algorithm. It needs to be able to observe the human agent and infer what the human is trying to achieve at a given time before being able to offer assistance. Hence, extending learning algorithms to such cases where the agents may not

fully know each others' objectives is an important new challenge and an exciting future research direction.

8.3 Potential solutions and future research directions

The previous section introduced many upcoming new challenges in multi-agent learning systems that I observed during my research. This section briefly proposes future directions to explore in order to address some of the above mentioned challenges.

8.3.1 Scaling quadratically growing interactions with differentiable clustering

To address the challenge of quadratic (super-linear) scaling with the number of agents N, it is important to note that all pairwise interactions in a multi-agent system are not necessarily useful. We often see instances of this in our day-to-day lives, e.g., a pedestrian walking in a crowd only looks at close-by neighbors to make decisions about their path and not necessarily at everyone around. Incorporating such heuristics to reduce the size of the computation graph generally requires some domain knowledge and we have explored this approach in chapter 4 using our distance-based cutoff heuristic.

However, when such domain knowledge is not available humans are still able to reduce the complexity of their decision making by often grouping similar agents/entities together. For instance, when one wanders through a crowded corridor in a school after the bell rings, one often views the group of students coming out a classroom as a single super-entity rather than viewing them all as separate entities! This concept of dynamically grouping agents/entities with similar behavior is key to the human decision making process and is another potential inductive bias that can be incorporated in any multi-agent learning architecture. While an exact implementation of such dynamic clustering is currently an open research problem, a version of this problem also often arises in: (a) graph clustering where one needs to merge graph nodes into super-nodes to create a clustered

graph and (b) image segmentation where parts of an features have to be recursively clustered to identify objects. Recent works in these domains have explored the concept of differentiable clustering [144, 127, 71] towards accomplishing dynamic clustering in computation graphs and this could be an interesting first step to explore for reducing the complexity of multi-agent interaction learning architectures.

8.3.2 Robust model-based learning

As discussed above, using learnt reward models directly for backpropagation can sometimes create hallucinatory effects which are not present in the true system. While it is unclear if this is always necessary, there are potential alternatives like I2A [103] which combine model-based and model-free learning in the case of single-agent reinforcement learning algorithms. These can act as potential starting points for exploration into a combination of model-free and model-based methods for multi-agent reinforcement learning. This can potentially lead to augmented variants or combinations of OptGradFP and DeepFP for learning in complex security games settings where a learnt reward model by itself may be inaccurate for direct backpropagation and a more accurate reward model like that presented in chapter 7 may not be available.

8.3.3 Adaptive sampling for large spatial coverage domains

While we presented a spatial discretization based architecture to implement the integrals involved in the *Coverage Gradient Theorem*, this could be a potential limitation of the framework. The size of the spatial maps and binary tensors involved depends on the number of bins chosen per dimension of the target domain. This number can be large if a fine-grained discretization is being used or the target space is huge and can require multiple GPUs in parallel to store the full forward and backward models. An example application could be a large scale placement of Covid-19 test centers in one of the US states.

In such cases a potential direction of research could be to use adaptive sampling to approximate the involved integrals. However, while it is much more manageable to store discretized shape tensors in GPU memory, working with samples from geometric shapes and defining operators on them is generally harder. To obtain the best trade-off between memory and parallelization on GPUs, working on scalable adaptive sampling-based frameworks is a promising next step for future research.

8.3.4 Cooperative Inverse Reinforcement Learning

When two agents need to cooperate with each other but one or more of them may not be aware of the other's goals, the learning problem becomes much more complex. In such *value alignment* problems, often simple inverse reinforcement learning cannot be directly applied. Rather, one needs to redefine the game taking into account the fact that a true solution may require inference and goal estimation as sub-steps. As a first step to solving such games, e.g., for real life artificial assistive agents, researching Cooperative Inverse reinforcement learning [43] is an exciting new future direction and can lead to substantial improvement in these domains.

Reference List

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 961–971, 2016.

[2] Stefano V Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.

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

[4] Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. *arXiv preprint arXiv:1705.08439*, 2017.

[5] Karl Johan Åström. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965.

[6] David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. In *International Conference on Machine Learning*, pages 354–363. PMLR, 2018.

[7] Nicola Basilico, Andrea Celli, Giuseppe De Nittis, and Nicola Gatti. Coordinating multiple defensive resources in patrolling games with alarm systems. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 678–686, 2017.

[8] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[9] Stefan Becker, Ronny Hug, Wolfgang Hübner, and Michael Arens. An evaluation of trajectory prediction approaches and notes on the trajnet benchmark. *arXiv preprint arXiv:1805.07663*, 2018.

[10] Soheil Behnezhad, Mahsa Derakhshan, Mohammadtaghi Hajiaghayi, and Saeed Seddighin. Spatio-temporal games beyond one dimension. In *Proceedings of the 2018 ACM Conference on Economics and Computation*, pages 411–428, 2018.

[11] Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, and Aleksandrs Slivkins. A polynomial time algorithm for spatio-temporal security games. In *Proceedings of the 2017 ACM Conference on Economics and Computation*, pages 697–714, 2017.

[12] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, 2017.

[13] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, 6(5):679–684, 1957.

[14] Daan Bloembergen, Karl Tuyls, Daniel Hennes, and Michael Kaisers. Evolutionary dynamics of multi-agent learning: A survey. *J. Artif. Intell. Res.(JAIR)*, 53:659–697, 2015.

[15] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*, 2018.

[16] Jiří Cermák, Branislav Bošanský, Karel Durkota, Viliam Lisý, and Christopher Kiekintveld. Using correlated strategies for computing stackelberg equilibria in extensive-form games. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 439–445, 2016.

[17] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. In *International Conference on Learning Representations*, 2017.

[18] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In *Conference on Robot Learning*, pages 617–629. PMLR, 2018.

[19] Benjamin Coifman and Lizhe Li. A critical evaluation of the next generation simulation (ngsim) vehicle trajectory dataset. *Transportation Research Part B: Methodological*, 105(C):362–377, 2017.

[20] Vincent Conitzer. Approximation guarantees for fictitious play. In *47th Annual Allerton Conference on Communication, Control, and Computing*, pages 636–643. IEEE, 2009.

[21] Vincent Conitzer and Tuomas Sandholm. Computing the Optimal Strategy to Commit to. In *Proc. of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 82–90, 2006.

[22] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Michael Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. *arXiv preprint arXiv:1810.11187*, 2018.

[23] Nachiket Deo and Mohan M Trivedi. Convolutional social pooling for vehicle trajectory prediction. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1468–1476, 2018.

[24] Franck Dernoncourt and Elisabeth Métais. Fuzzy logic: introducing human reasoning within decision support systems?, 2011.

[25] Alireza Dirafzoon, Mohammad Bagher Menhaj, and Ahmad Afshar. Decentralized coverage control for multi-agent systems with nonlinear dynamics. *IEICE TRANSACTIONS on Information and Systems*, 94(1):3–10, 2011.

[26] 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.

[27] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

[28] Thomas S. Ferguson. *Game Theory*, volume 2. Online, 2014.

[29] Tharindu Fernando, Simon Denman, Sridha Sridharan, and Clinton Fookes. Soft + hardwired attention: An lstm framework for human trajectory prediction and abnormal event detection. *Neural networks*, 108:466–478, 2018.

[30] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017.

[31] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *International conference on machine learning*, pages 1146–1155. PMLR, 2017.

[32] Jakob Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 1942–1951. PMLR, 2019.

[33] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.

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

[35] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

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

[37] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[38] Audrunas Gruslys, Will Dabney, Mohammad Gheshlaghi Azar, Bilal Piot, Marc Bellemare, and Remi Munos. The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning. *arXiv preprint arXiv:1704.04651*, 2017.

[39] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, Bernhard Schölkopf, and Sergey Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. *arXiv preprint arXiv:1706.00387*, 2017.

[40] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2255–2264, 2018.

[41] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, 2018.

[42] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[43] Dylan Hadfield-Menell, Anca Dragan, Pieter Abbeel, and Stuart Russell. Cooperative inverse reinforcement learning. *arXiv preprint arXiv:1606.03137*, 2016.

[44] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[45] K. Hara, D. Saito, and H. Shouno. Analysis of function of rectified linear unit used in deep learning. In *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015.

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

[47] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

[48] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[49] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *International Conference on Machine Learning*, pages 805–813, 2015.

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

[51] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[52] Mikael Henaff, Alfredo Canziani, and Yann LeCun. Model-predictive policy learning with uncertainty regularization for driving in dense traffic. In *International Conference on Learning Representations*, 2019.

[53] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

[54] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[55] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[56] Josef Hofbauer and William H Sandholm. On the global convergence of stochastic fictitious play. *Econometrica*, 70(6):2265–2294, 2002.

[57] Yedid Hoshen. Vain: Attentional multi-agent predictive modeling. In *Advances in Neural Information Processing Systems*, pages 2701–2711, 2017.

[58] Andrew Howard, Maja J Matarić, and Gaurav S Sukhatme. Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In *Distributed Autonomous Robotic Systems 5*, pages 299–308. Springer, 2002.

[59] Taoan Huang, Weiran Shen, David Zeng, Tianyu Gu, Rohit Singh, and Fei Fang. Green security game with community engagement. *arXiv preprint arXiv:2002.09126*, 2020.

[60] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 2961–2970. PMLR, 2019.

[61] Ashesh Jain, Amir R Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 5308–5317, 2016.

[62] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. *arXiv preprint arXiv:1805.07733*, 2018.

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

[64] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

[65] Nitin Kamra, Fei Fang, Debarun Kar, Yan Liu, and Milind Tambe. Handling continuous space security games with neural networks. In *IWAISe: First International Workshop on Artificial Intelligence in Security*, 2017.

[66] Nitin Kamra, Umang Gupta, Fei Fang, Yan Liu, and Milind Tambe. Policy learning for continuous space security games using neural networks. In *AAAI*, 2018.

[67] Nitin Kamra, Umang Gupta, Kai Wang, Fei Fang, Yan Liu, and Milind Tambe. Deepfp for finding nash equilibrium in continuous action spaces. In *Decision and Game Theory for Security (GameSec)*, pages 238–258. Springer International Publishing, 2019.

[68] Nitin Kamra, Hao Zhu, Dweep Trivedi, Ming Zhang, and Yan Liu. Multi-agent trajectory prediction with fuzzy query attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[69] 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.

[70] 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.

[71] Wonjik Kim, Asako Kanezaki, and Masayuki Tanaka. Unsupervised learning of image segmentation based on differentiable feature clustering. *IEEE Transactions on Image Processing*, 29:8055–8068, 2020.

[72] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[73] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pages 2693–2702, 2018.

[74] Chan Sze Kong, New Ai Peng, and Ioannis Rekleitis. Distributed coverage with multi-robot system. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2423–2429. IEEE, 2006.

[75] 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.

[76] Vijay Krishna and Tomas Sjöström. On the convergence of fictitious play. *Mathematics of Operations Research*, 23(2):479–511, 1998.

[77] Prashanth Krishnamurthy and Farshad Khorrami. Optimal sensor placement for monitoring of spatial networks. *IEEE Transactions on Automation Science and Engineering*, 15(1):33–44, 2016.

[78] 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.

[79] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[80] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017.

[81] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 336–345, 2017.

[82] David S Leslie and Edmund J Collins. Generalised weakened fictitious play. *Games and Economic Behavior*, 56(2):285–298, 2006.

[83] Alistair Letcher, Jakob Foerster, David Balduzzi, Tim Rocktäschel, and Shimon Whiteson. Stable opponent shaping in differentiable games. *arXiv preprint arXiv:1811.08469*, 2018.

[84] Yaguang Li, Chuizheng Meng, Cyrus Shahabi, and Yan Liu. Structure-informed graph auto-encoder for relational inference and simulation. In *ICML Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019.

[85] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[86] Edward Lockhart, Marc Lanctot, Julien Pérolat, Jean-Baptiste Lespiau, Dustin Morrill, Finbarr Timbers, and Karl Tuyls. Computing approximate equilibria in sequential adversarial games by exploitability descent. *arXiv preprint arXiv:1903.05614*, 2019.

[87] Qian Long, Zihan Zhou, Abhibav Gupta, Fei Fang, Yi Wu, and Xiaolong Wang. Evolutionary population curriculum for scaling multi-agent reinforcement learning. *arXiv preprint arXiv:2003.10423*, 2020.

[88] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

[89] Yuexin Ma, Xinge Zhu, Sibo Zhang, Ruigang Yang, Wenping Wang, and Dinesh Manocha. Trafficpredict: Trajectory prediction for heterogeneous traffic-agents. *arXiv preprint arXiv:1811.02146*, 2018.

[90] Christoforos I Mavrogiannis and Ross A Knepper. Multi-agent trajectory prediction and generation with topological invariants enforced by hamiltonian dynamics. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, 2018.

[91] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[92] 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.

[93] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[94] Ali Nasri Nazif, Alireza Davoodi, and Philippe Pasquier. Multi-agent area coverage using a single query roadmap: A swarm intelligence approach. In *Advances in practical multi-agent systems*, pages 95–112. Springer, 2010.

[95] Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Combining policy gradient and q-learning. *arXiv preprint arXiv:1611.01626*, 2016.

[96] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *International Conference on Machine Learning*, pages 2681–2690. PMLR, 2017.

[97] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[98] Stefano Pellegrini, Andreas Ess, Konrad Schindler, and Luc Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *IEEE 12th International Conference on Computer Vision*, pages 261–268. IEEE, 2009.

[99] S. Perkins and D.S. Leslie. Stochastic fictitious play with continuous action sets. *Journal of Economic Theory*, 152:179 – 213, 2014.

[100] Huy Xuan Pham, Hung Manh La, David Feil-Seifer, and Aria Nefian. Cooperative and distributed reinforcement learning of drones for field coverage. *arXiv preprint arXiv:1803.07250*, 2018.

[101] S. Poduri and G. S. Sukhatme. Constrained coverage for mobile sensor networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[102] Neil Rabinowitz, Frank Perbet, Francis Song, Chiyuan Zhang, SM Ali Eslami, and Matthew Botvinick. Machine theory of mind. In *International Conference on Machine Learning*, pages 4215–4224, 2018.

[103] Sébastien Racanière, Théophane Weber, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5690–5701, 2017.

[104] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304. PMLR, 2018.

[105] Alessandro Renzaglia, Lefteris Doitsidis, Agostino Martinelli, and Elias B Kosmatopoulos. Multi-robot three-dimensional coverage of unknown areas. *The International Journal of Robotics Research*, 31(6):738–752, 2012.

[106] Ariel Rosenfeld and Sarit Kraus. 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*, pages 3814–3822, 2017.

[107] Christoph Rösmann, Malte Oeljeklaus, Frank Hoffmann, and Torsten Bertram. Online trajectory prediction and planning for social robot navigation. In *2017 IEEE International Conference on Advanced Intelligent Mechatronics*, pages 1255–1260. IEEE, 2017.

[108] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

[109] Martin Saska, Jan Chudoba, Libor Přeučil, Justin Thomas, Giuseppe Loianno, Adam Třešňák, Vojtěch Vonásek, and Vijay Kumar. Autonomous deployment of swarms of micro-aerial vehicles in cooperative surveillance. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 584–595. IEEE, 2014.

[110] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[111] John Schulman, Xi Chen, and Pieter Abbeel. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*, 2017.

[112] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[114] Jeff S Shamma and Gürdal Arslan. Unified convergence proofs of continuous-time fictitious play. *IEEE Transactions on Automatic Control*, 49(7):1137–1141, 2004.

[115] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[116] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5887–5896. PMLR, 2019.

[117] Sriram Srinivasan, Marc Lanctot, Vinicius Zambaldi, Julien Pérolat, Karl Tuyls, Rémi Munos, and Michael Bowling. Actor-critic policy optimization in partially observable multiagent environments. *arXiv preprint arXiv:1810.09026*, 2018.

[118] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. *arXiv preprint arXiv:1605.07736*, 2016.

[119] Chen Sun, Per Karlsson, Jiajun Wu, Joshua B Tenenbaum, and Kevin Murphy. Stochastic prediction of multi-agent interactions from partial observations. In *International Conference on Learning Representations*, 2019.

[120] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[121] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[122] 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.

[123] Andrea Tacchetti, H Francis Song, Pedro AM Mediano, Vinicius Zambaldi, Neil C Rabinowitz, Thore Graepel, Matthew Botvinick, and Peter W Battaglia. Relational forward models for multi-agent learning. In *International Conference on Learning Representations*, 2019.

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

[125] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[126] Daoqin Tong, Alan Murray, and Ningchuan Xiao. Heuristics in spatial analysis: a genetic algorithm for coverage maximization. *Annals of the Association of American Geographers*, 99(4):698–711, 2009.

[127] Anton Tsitsulin, John Palowitch, Bryan Perozzi, and Emmanuel Müller. Graph clustering with graph neural networks. *arXiv preprint arXiv:2006.16904*, 2020.

[128] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[129] Daksh Varshneya and G Srinivasaraghavan. Human trajectory prediction using spatially aware deep attention models. *arXiv preprint arXiv:1705.09436*, 2017.

[130] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[131] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[132] Anirudh Vemula, Katharina Muelling, and Jean Oh. Social attention: Modeling attention in human crowds. In *IEEE International Conference on Robotics and Automation*, pages 1–7, 2018.

[133] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[134] Binru Wang, Yuan Zhang, and Sheng Zhong. 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, pages 1751–1753, 2017.

[135] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[136] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[137] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[138] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *arXiv preprint arXiv:1708.05144*, 2017.

[139] Haifeng Xu, Fei Fang, Albert Xin Jiang, Vincent Conitzer, Shaddin Dughmi, and Milind Tambe. Solving zero-sum security games in discretized spatio-temporal domains. In *AAAI*, pages 1500–1506, 2014.

[140] Mohamed Amine Yakoubi and Mohamed Tayeb Laskri. The path planning of cleaner robot for coverage region using genetic algorithms. *Journal of innovation in digital ecosystems*, 3(1):37–43, 2016.

[141] Kota Yamaguchi, Alexander C Berg, Luis E Ortiz, and Tamara L Berg. Who are you with and where are you going? In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1345–1352. IEEE, 2011.

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

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

[144] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*, 2018.

[145] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pages 3394–3404, 2017.

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

[147] Eric Zhan, Stephan Zheng, Yisong Yue, Long Sha, and Patrick Lucey. Generating multi-agent trajectories using programmatic weak supervision. In *International Conference on Learning Representations*, 2019.

[148] Stephan Zheng, Yisong Yue, and Jennifer Hobbs. Generating long-term trajectories using deep hierarchical networks. In *Advances in Neural Information Processing Systems*, pages 1543–1551, 2016.

Appendix A

DeepFP for Finding Nash Equilibrium in Continuous Action Spaces

A.1 Hyperparameters and model architectures

All our models were trained using TensorFlow v1.5 on a Ubuntu 16.04 machine with 32 CPU cores and a Nvidia Tesla K40c GPU.

- **Cournot game and Concave-convex game**: Best response networks for the Cournot game and the Concave-convex game consist of single fully connected layer with a sigmoid activation, directly mapping the 2-D input noise $z \sim \mathcal{N}([0,0], I_2)$ to a scalar output q_p for player p. Best response networks are trained with Adam optimizer [72] and learning rate of 0.05. To estimate payoffs, we use exact reward models for the game model networks. Maximum games were limited to 30,000 for Cournot game and 50,000 for Concave-convex game.

- **Forest protection game**: The action u_p of player p contains the cylindrical coordinates (radii and angles) for all resources of that player. So, the best response network for the Forest protection game maps $Z_A \in \mathbb{R}^{64}$ to the adversary action $u_A \in \mathbb{R}^{n \times 2}$. It has 3 fully connected hidden layers with $\{128, 64, 64\}$ units and ReLU activations. The final output

comes from two parallel fully connected layers with n (number of lumberjacks) units each: (a) first with sigmoid activations outputting n radii $\in [0, 1]$, and (b) second with linear activations outputting n angles $\in [-\infty, \infty]$, which are modulo-ed to be in $[0, 2\pi]$ everywhere. All layers are L2-regularized with coefficient 10^{-2}:

$$x_A = relu(FC_{64}(relu(FC_{64}(relu(FC_{128}(Z_A))))))$$

$$u_{A,rad} = \sigma(FC_n(x_A)); \quad u_{A,ang} = FC_n(x_A)$$

The game model takes all players' actions as inputs (i.e. matrices u_D, u_A of shapes $(m, 2)$ and $(n, 2)$) respectively) and produces two scalar rewards r_D and r_A. It internally converts the angles in the second columns of these inputs to the range $[0, 2\pi]$. Since the rewards should be invariant to the permutations of the defender's and adversary's resources (guards and lumberjacks resp.), we first pass the input matrices through non-linear embeddings to interpret their rows as sets rather than ordered vectors (see Deep Sets [145] for details). These non-linear embeddings are shared between the rows of the input matrix and are themselves deep neural networks with three fully connected hidden layers containing $\{60, 60, 120\}$ units and ReLU activations. They map each row of the matrices into a 120-dimensional vector and then add all these vectors. This effectively projects the action of each player into a 120-dimensional action embedding representation invariant to the ordering of the resources. The players' embedding networks are trained jointly as a part of the game model network. The players' action embeddings are further passed through 3 hidden fully connected layers with $\{1024, 512, 128\}$ units and ReLU activations. The final output rewards are produced

by a last fully connected layer with 2 hidden units and linear activation. All layers are L2-regularized with coefficient 3×10^{-4}:

$$emb_p = \sum_{dim=row} (DeepSet_{60,60,120}(u_p)) \qquad \forall p \in \{D, A\}$$

$$\hat{r}_D, \hat{r}_A = FC_2(relu(FC_{128}(relu(FC_{512}(relu(FC_{1024}(emb_D, emb_A))))))$$

The models are trained with Adam optimizer [72]. Note that the permutation invariant embeddings are not central to the game model network and only help to incorporate an inductive bias for this game. We also tested the game model network without the embedding networks and achieved similar performance with about 2x increase in the number of iterations since the game model would need to infer permutation invariance from data.

Appendix B

Gradient-based Optimization for Multi-resource Spatial Coverage Problems

B.1 Hyperparameters and model architectures

B.1.1 Learning differentiable reward models

While learning differentiable reward models with neural networks, we trained all networks for $100,000$ iterations with the Adam optimizer having learning rate 0.01 and a batch size of 64. The network architectures used are shown in Table B.1.

B.1.2 DeepFP

For DeepFP, we run a total of 1000 outer fictitious play iterations and 100 inner optimization iterations to update best responses using the Adam optimizer with learning rate 0.001 and batch size 16. The network architecture for best response nets in *brnet* variant are shown in Table B.2.

B.2 Divide and conquer based shape discretizer

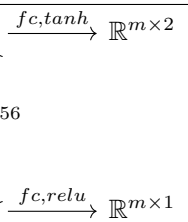
The python pseudo-code for the discretizer is shown below and makes use of a recursive geometric map-filling method which uses divide and conquer to efficiently compute the interior, exterior and

Table B.1: Network architectures for reward models

Game	Net type	Structure
Areal Surveillance	*nn*	$\mathbb{R}^{m\times 3} \xrightarrow{fc,relu} \mathbb{R}^{128} \xrightarrow{fc,relu} \mathbb{R}^{512} \xrightarrow{fc,relu} \mathbb{R}^{128} \xrightarrow{fc,relu} \mathbb{R}^{1}$
Areal Surveillance	*gnn*	$\mathbb{R}^{m\times 3}, \_, \_ \xrightarrow[3\to 32]{node\_enc} \mathbb{R}^{32}, \_, \_ \xrightarrow[64\to 16]{edge\_net} \mathbb{R}^{32}, \mathbb{R}^{16}, \_$ $\xrightarrow[48\to 32]{node\_net} \mathbb{R}^{32}, \mathbb{R}^{16}, \_ \xrightarrow[48\to 16]{glob\_net} \mathbb{R}^{32}, \mathbb{R}^{16}, \mathbb{R}^{16}$ $\xrightarrow[96\to 16]{edge\_net} \mathbb{R}^{32}, \mathbb{R}^{16}, \mathbb{R}^{16} \xrightarrow[64\to 32]{node\_net} \mathbb{R}^{32}, \mathbb{R}^{16}, \mathbb{R}^{16}$ $\xrightarrow[64\to 1]{glob\_net} \mathbb{R}^{1}$
Adversarial Coverage	*nn*	$\mathbb{R}^{m\times 3} \xrightarrow{fc,relu} \mathbb{R}^{128} \xrightarrow{fc,relu} \mathbb{R}^{128} \xrightarrow{fc} \mathbb{R}^{1}$ $\downarrow cat \qquad \uparrow$ $\mathbb{R}^{256} \xrightarrow{fc,relu} \mathbb{R}^{512}$ $\uparrow cat \qquad \downarrow$ $\mathbb{R}^{n\times 2} \xrightarrow{fc,relu} \mathbb{R}^{128} \xrightarrow{fc,relu} \mathbb{R}^{128} \xrightarrow{fc} \mathbb{R}^{1}$
Adversarial Coverage	*gnn*	$\mathbb{R}^{(m+n)\times 3}, \_, \_ \xrightarrow[3\to 64]{node\_enc} \mathbb{R}^{64}, \_, \_ \xrightarrow[128\to 32]{edge\_net} \mathbb{R}^{64}, \mathbb{R}^{32}, \_$ $\xrightarrow[96\to 64]{node\_net} \mathbb{R}^{64}, \mathbb{R}^{32}, \_ \xrightarrow[96\to 32]{glob\_net} \mathbb{R}^{64}, \mathbb{R}^{32}, \mathbb{R}^{32}$ $\xrightarrow[192\to 32]{edge\_net} \mathbb{R}^{64}, \mathbb{R}^{32}, \mathbb{R}^{32} \xrightarrow[128\to 64]{node\_net} \mathbb{R}^{64}, \mathbb{R}^{32}, \mathbb{R}^{32}$ $\xrightarrow[128\to 2]{glob\_net} \mathbb{R}^{2}$

boundary of any geometric shape stored in the *Shapely* geometric library format. Note that a minimal functional pseudo-code using *Numpy* has been presented here to facilitate understanding. Our actual code is more complex and allows working with PyTorch tensors on both CPU and GPU while also supporting batches of geometric objects. We also have other specialized versions (not shown here) which work faster for circular geometries.

Table B.2: Network architectures for DeepFP *brnet* best responses

Net type	Structure
Defender's *brnet*	$\mathbb{R}^{32} \xrightarrow{fc,relu} \mathbb{R}^{256}$ 
Attacker's *brnet*	$\mathbb{R}^{32} \xrightarrow{fc,relu} \mathbb{R}^{256} \xrightarrow{fc,tanh} \mathbb{R}^{n\times 2}$

```
import numpy as np
from shapely.geometry import Polygon, Point


def get_g_map(geom, lims, deltas):
    ''' Computes the geometric maps from geometry.
    Args:
        geom: Shapely geometry object
        lims: Tuple (x_min, x_max, y_min, y_max) for generated
            geometric map
        deltas: Discretization bin size; tuple (delX, delY)

    Returns:
        g_map: numpy.ndarray of shape (nbinsX, nbinsY, 3)
        containing (interior, boundary, exterior) indicator of
        geometry in the third dimension.
    '''
    x_min, x_max, y_min, y_max = lims
    delX, delY = deltas
```

```
    nbinsX = round((x_max - x_min) / delX)
    nbinsY = round((y_max - y_min) / delY)

    g_map = np.zeros((nbinsX, nbinsY, 3)) # (int, bound, ext)
    fill(geom, g_map, 0, nbinsX, 0, nbinsY, lims, deltas)
    return g_map


def fill(geom, g_map, i1, i2, j1, j2, lims, deltas):
    ''' Fills g_map of shape (nbinsX, nbinsY, 3) with 1s at
        appropriate locations to indicate interior, exterior and
        boundary of the shape geom. This method makes recursive
        calls to itself and fills up the g_map tensor in-place.

    Args:
        geom: A shapely.geometry object, e.g. Polygon
        g_map: A numpy.ndarray of shape (nbinsX, nbinsY, 3)
        i1: left x-coord of recursive rectangle to check against
        i2: right x-coord of recursive rectangle to check against
        j1: bottom y-coord of recursive rectangle to check against
        j2: top y-coord of recursive rectangle to check against
        lims: Tuple (x_min, x_max, y_min, y_max) for generated
            geometric map
        deltas: Discretization bin size; tuple (delX, delY)
    '''
```

```
x_min, x_max, y_min, y_max = lims
delX, delY = deltas

box = Polygon([(x_min + i1*delX, y_min + j1*delY), \
               (x_min + i2*delX, y_min + j1*delY), \
               (x_min + i2*delX, y_min + j2*delY), \
               (x_min + i1*delX, y_min + j2*delY)])

if box.disjoint(geom):
    g_map[i1:i2, j1:j2, 2] = 1.0
elif box.within(geom):
    g_map[i1:i2, j1:j2, 0] = 1.0
else: # box.intersects(geom)
    if (i2 - i1 <= 1) and (j2 - j1 <= 1):
        g_map[i1:i2, j1:j2, 1] = 1
    elif (i2 - i1 <= 1) and (j2 - j1 > 1):
        j_mid = (j1 + j2) // 2
        fill(geom, g_map, i1, i2, j1, j_mid, lims, deltas)
        fill(geom, g_map, i1, i2, j_mid, j2, lims, deltas)
    elif (i2 - i1 > 1) and (j2 - j1 <= 1):
        i_mid = (i1 + i2) // 2
        fill(geom, g_map, i1, i_mid, j1, j2, lims, deltas)
        fill(geom, g_map, i_mid, i2, j1, j2, lims, deltas)
    else: # (i2 - i1 > 1) and (j2 - j1 > 1):
        i_mid = (i1 + i2) // 2
```

```
    j_mid = (j1 + j2) // 2
    fill(geom, g_map, i1, i_mid, j1, j_mid, lims, deltas)
    fill(geom, g_map, i_mid, i2, j1, j_mid, lims, deltas)
    fill(geom, g_map, i1, i_mid, j_mid, j2, lims, deltas)
    fill(geom, g_map, i_mid, i2, j_mid, j2, lims, deltas)
```