Learning Mental Models of Experts in a Simulated Search and Rescue Scenario

Rohit Jena

CMU-RI-TR-21-34

July 29, 2021

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Katia Sycara, *chair*
Changliu Liu
Wenhao Luo

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Copyright © 2021 Rohit Jena. All rights reserved.

To mom and dad.

Abstract

Search and Rescue is a task where the rescuers need to be cognitively agile, strategically consistent and efficient to save as many trapped victims as possible. However, in such a dynamic scenario, the rescuers' mental models may be outdated, and it may be difficult to coordinate with other rescuers under a time constraint and cognitive overload. In this thesis, we propose to develop agents based on Machine Theory of Mind (MToM) to infer the beliefs, intentions, and desires of the rescuers from their observations and actions. By generating a mental model, an agent can intervene when it detects a rescuer might act based on a false belief. We also study approaches on using data from a different map to learn robust neural agents. Finally, we study the paradigm of imitation learning to learn policies from expert trajectories, and propose a sample efficient method over existing baselines.

Acknowledgments

I would start by sincerely thanking my advisor Prof. Katia Sycara for taking me in as an advisee. I wanted to try a relatively new field considering my existing background in computer vision and medical image analysis. Diving into a new research topic can be fun and exciting, but can be hard to navigate too. I thank Prof. Katia for helping me to navigate the deep waters of research in theory of mind and imitation learning. I would like to thank Prof. Changliu for her insightful discussions and support. Also a huge thanks to my labmates Ankur, Tejus, Vidhi, Siddharth, Akshay, Keitaro, Yisha, Swami for their immense support and long hours of thoughtful discussions. I would also like to thank Prof. Kayhan and team members at Batmanlab - Sumedha, Li, Ke, Brian, and others, with whom I had the chance to explore some medical research.

Finally, I would like to thank my mom and dad, for their immense emotional and mental support. I would also thank my brother Rohan for his endearing presence. In reality, no amount of thanks is enough for their belief in me and my abilities. Without their love and support, I would not have been fortunate enough to conduct such exciting research in a top-notch university, far, far away from home.

Funding

This work was supported by DARPA Award HR001120C0036. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA).

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

Consider a scenario where an office building gets damaged during an earthquake and has victims who need to be rescued. Victims would have varying degrees of injury, and therefore human rescuers have to develop appropriate search, navigation and victim triaging strategies so as to save the largest number of victims in limited time. The earthquake may have caused serious structural perturbations, such as blockages that limit the ability of the rescuers to reach certain areas.

In this thesis, we provide initial results of a computational agent that observes the environment and the behavior of a human rescuer (in a simulation environment) and predicts the beliefs, intents, and actions of the rescuer. This is the first step towards developing an agent that can provide assistance to the human based on their mental model and state. The assistance will consist of sparse interventions when the player deviates from the optimal actions for the task. Anticipatory assistance is important in dynamic and dangerous situations. By predicting the rescuer's mental model, the agent can warn the human to avoid recently collapsed and unstable areas, or give advice to mitigate human cognitive limitations, such as limited and short-term memory over current localization. This helps the human avoid duplicate effort by revisiting areas that have already been searched. Moreover, neural computational agents require huge amounts of data to learn such patterns, which is difficult and expensive to obtain. Therefore, we need approaches that use a small amount of human data as a ”bootstrap” for a generative model that generates more data over the distribution. This generated data can be used to train our agents as well.

In this work, we look at a baseline approach to train such computational models, build a transfer learning framework to use data from other maps, and finally build imitation learning agents that can generate more data from the given ”expert data”. The thesis is organized as follows: first, we explore the background required to formulate the tasks mentioned above. Then, we discuss some recent and older related work in the literature. Next, we propose each of the three methods in the following chapters. Finally, we conclude the document.

Chapter 2

Background

2.1 Theory of Mind

Humans are social creatures, who participate in day-to-day interactions with other humans. This requires understanding of the other person's behavior and their intentions and desires, which are usually inferred from their statements and expressions, as they are the only things that are directly observed. Each human is aware of their own mental model and how their mind works, via directly introspection of their mind. The human doesn't, however, have direct access to the mental models of other humans, and therefore must infer their latent mental models from their visible actions.

Theory of Mind (ToM) [49] refers to the innate ability of humans to infer other's minds by the assumption that other minds are analogous to their own. ToM allows a human to attribute beliefs, desires and intentions of others via past actions and behavior, to predict future actions and behavior. This helps the human to engage in social interactions and account other humans to modify their own mental model of how to perform certain actions. Humans develop a complete theory of mind with social or other experiences over many years. A person who does not have a completely developed theory of mind may indicate a sign of congitive or developmental impairment.

Machine theory of Mind (MToM) [49] aims to use this theory to train a machine learning algorithm to build a MToM agent that observes a human/machine's observations and actions in an environment, and makes inferences about its theory of mind, including prediction of belief states, intentions, and desires. For brevity in this subsection, we refer to the MToM agent as simply the *agent* and the observed entity as the *player*. To make an inference about intentions and detection of false beliefs, MToM agents generally possess more information than the player that is being observed by the agent. ToM models do not reference the player's underlying structure (their actual mental models) but make predictions about their future actions. MToM uses this salient feature of ToM models to build its prediction models without actually modelling the players it predicts.

The agent needs a strong prior about different types of players, over which it can then *finetune* after observing the actions of the player. This is similar to how we have a certain set of priors of how other humans behave (altruistic, greedy, calculative, random, etc.) and then based on the actions of a certain person, we update our theory of mind of that person. To learn a generalizable prior (encoded in its parameters), MToM uses a *meta-learning* [16] framework to build models of the players it encounters. *Meta-learning* has found huge successes in learning and adapting to new datasets with one to few examples (commonly known as *few-shot learning*) [68]. During *meta-training* the agent learns to adapt its parameters to input traces of behavior from different players and output predictions of future behavior. This *learning to adapt parameters* for different players is what constitutes meta-learning (learning to learn). During *meta-testing*, the agent takes behavior traces from a novel player as input, and adapts its parameters to predict the future behavior of the player.

In [49], the architecture of the model, which is called the *ToMnet* is described.

The ToMnet consists of three modules:

1. **Character Net**: The character network is designed to characterize the given player, by parsing its past episode trajectories. Given a set of trajectories $\{\tau_{ij}\}_{j=1}^{N_i^{past}}$ for player i, the character net f_θ, where θ denotes learnable parameters, outputs a player-trajectory embedding $e_{ij}^{char} = f_\theta(\tau_{ij})$. Then, the final character embedding of the player is given by $e_i^{char} = \sum_j e_{ij}^{char}$.
2. **Mental state net**: The mental network takes the current trajectory of the player τ_{ik} and outputs an estimate of the mental state of the player. The mental state net is given by g_ϕ, where ϕ denotes learnable parameters. Let the current

trajectory of the player contain observations and actions for t timesteps. The mental state embedding for the player for the given time is therefore given by $e_i^{mental} = g_\phi([\tau_{ik}]_{0:t-1}, e_i^{char})$. Note that the mental state embedding also depends on the character emebedding, which can be useful in determining mental states of the player like cognitive workload (some players may find it easier to do the same tasks than others resulting in a lower cognitive overload for them).

3. **Prediction net**: Finally, the goal of the prediction network is to leverage the character and current mental state of the player to predict the future actions of the player. Given the prediction net π, the output action is given by a deterministic or probabilistic value $\hat{\pi}(.|x_t, e^{char}, e^{mental})$. In addition to the action, other values such as successor representations, or usage of certain environment entities can be predicted by the prediction net. In practice, the prediction net uses a shared torso, and different heads for different predictions.

The full training of the MToM using meta-learning has been summarized in Algorithm 1.

Algorithm 1 Machine Theory of Mind

```
**procedure** MToM($\{\tau_{ij}\}_{i=1:N,j=1:N_i^{past}}, \{\tau_{ik}\}_{i=1:t}, T, \alpha$)
    $\theta, \phi, \psi$ = RANDOM_INIT()
    **for** p = 0 to $T-1$ **do**
        Sample player $i \in$ RANDINT$(1, N)$
        $e_i^{char} \leftarrow \sum_{j=1}^{N_i^{past}} f_\theta(\{\tau_{ij}\})$
        $e_i^{mental} \leftarrow g_\phi(\{\tau_{ik}\}, e_i^{char})$
        $\hat{a}_{ik} \leftarrow \pi_\psi(.|x_t, e_i^{char}, e_i^{mental})$
        Let $\Theta = (\theta, \psi, \phi)$.
        Update parameters.
            $\Theta \leftarrow \Theta - \alpha \nabla_\Theta l(\hat{a}_{ik}, a_{ik})$
    **end for**
**end procedure**
```

2.2 Search and Rescue Task

The USAR mission requires a human player/rescuer to work in an indoor environment in a limited time where victims were scattered. The rescuer is equipped with a medical kit which enabled them to triage victims in a few seconds. A device that beeps to report if one or more live victims were inside a room was utilized, however the human participant may or may not have understood the significance of the beeping. Knowledge of the beeping was a condition that was varied in different trials, i.e. in some conditions the participant was told the meaning of the beep at the beginning of the experiment, whereas in other conditions, the participant may not have been given this information. High rewards were given when seriously injured victims were saved, and low ones for slightly injured victims. The Rescuer State and Field of View Contents update at a rate of 10Hz, all other observations are asynchronous events. The sensor of the agent collects various events of rescuers in the input stream, including but not limited to participant location, yaw, pitch, information of the victims interacted, contents of the Field of View (FoV), events indicating opening / closing doors, entering / exiting rooms, start / end triage, beep message for detecting victim in the nearby room. The location and FoV messages appear at a rate of 10Hz. When the rescuers are in front of the door of a room, the detecting device sends out a beep message to give them a hint on if there are victims inside this room.

To train agents and evaluate the ability of agents to evaluate players, multiple USAR task domains were developed using two distinct building layouts: a smaller map with fewer rooms (referred to as *Sparky*), and a larger, more complex map (referred to as *Falcon*). For each domain, victims were placed in specific locations of the environment, and the environment was perturbed with blockage and holes on the walls, which are not initially known to rescuers, but are encountered as rescuers navigate the environment. Blockages in the hallways and rooms might make certain paths in the map impassable, while holes on the walls opened other possibilities for navigation.

For prediction of rescuer navigation, a total of four task domains were created by varying the location of victims and perturbations: a single variant of the Sparky map, and three variants of the Falcon map of increasing difficulty (*Falcon-easy*, *Falcon-medium*, *Falcon-hard*). For prediction of rescuer triage strategy, we created two domains using the *Falcon* environment. The first domain, *Falcon-2victim*, contains victims of two severity levels, which are common to all domains: *regular* victims require 7 seconds to triage, and *critical* victims require 15 seconds to triage, and will expire 5 minutes after the start of the mission. The second domain, *Falcon-3victim*, introduces an additional victim severity level, *medium* victims, which require 12 seconds to triage and will expire 7 minutes into the mission.

2.2.1 Agent Observations

The agent observes a stream of information containing the following values from the rescuer's trajectory:

- **Rescuer State:** The position and orientation of the rescuer in the environment
- **Field of View Contents:** Blocks of interest (e.g., victims) that are present in the rescuer's field of view
- **Victim Interaction Events:** Including starting, completing, and abandoning triage attempts
- **Environment Interaction Events:** Including opening or closing doors, and entering or exiting rooms
- **Victim Location Device:** Emitted beep when the rescuer is near a room containing an untriaged victim.

Given the player's behaviors and the agent's observations of these variables, the goal is to predict the navigation and triage strategy of the player as part of the theory of mind intent and action predictions.

2.3 Imitation Learning

Given an environment, humans may or may not act suboptimally. In the case that they act optimally (or near-optimally) for a task, we would like to build artificial agents (policies) that imitate their behaviors and actions. One way of doing this is to formulate the task as a reinforcement learning problem, and design a reward function for the policy to learn from. However, designing such a reward function may be challenging, or even impossible due to the intractability of the state-space (designing reward functions for a self-driving car, for example). However, for such tasks, we have a set of expert trajectories who operate with respect to an underlying reward function which is hard or impossible to quantify. The problem of learning to perform these tasks from expert demonstrations is the problem of *Imitation Learning*. Imitation Learning has several popular approaches, which are discussed in this section.

1. **Behavior cloning**: A simple way to learn a policy from expert trajectories is to simply "“copy” the expert action at any given state. This is commonly known as behavior cloning. Behavior cloning is a simple technique that works in the asymptotic case when we have huge amounts of expert data, enough to cover huge portions of the state-space. With fewer and fewer data, there is the problem of *compounding errors*, where the errors in predicted actions drift the policy progressively further from states with high *occupancy* as per the expert. In the case of less data, the actions at states further from the expert states are states where the policy is not trained, and can therefore act randomly. Therefore, the method is limited in its usage.

2. **Inverse Reinforcement Learning**: In the beginning of the section, we discussed the intractability of designing the reward function by hand. However, given the expert trajectories, one can *learn* a reward (or cost) function under which the expert performs optimally. The reward function will also discourage the policy from encountering state-action pairs which have low reward as per the expert trajectories.

   The major problem with such an approach is that there are an arbitrarily high number of reward functions that can satisfy this requirement. Therefore, choice of a reward function among these possible candidates becomes a problem. Some of the natural criteria to select a reward function which is more ”meaningful” than other reward functions are proposed in the original IRL paper [46]. One of the criteria is to make any single step deviation from the policy π to be as costly as possible. This is equivalent to increasing the difference of the Q functions for the expert action from all non-expert actions. Mathematically, we find a reward function that maximizes:

$$\sum_{s \in S} \left(Q^{\pi}(s, a_1) - \max_{a \in A \setminus a_1} Q^{\pi}(s, a) \right)$$

Moreover, following Occam's Razor, we may want to favor 'simpler' reward functions and therefore have a weight decay-like penality of the form $-\lambda\|R\|_1$ where λ is a coefficient. Minimizing the L1-loss will lead to a sparse reward function, which is in line with the 'simple reward function' criteria. For large state spaces where enumeration is not possible, one can also use state features $\phi_1 \ldots \phi_m$ and formulate a reward function that is linear in these features, i.e.

$$R(s) = \sum_j \alpha_j \phi_j(s)$$

This makes optimization easy to handle constraints on the Q funcions since they can be calculated easily using linearity of expectations. Multiple works have built upon this idea in various ways, which we discuss in the related section.

3. **Adversarial Imitation Learning**: For environments with large state and action spaces, behavior cloning is a suboptimal algorithm, and IRL is an expensive one. The limitation of most IRL algorithms stem from their inefficiencies in solving the RL problem (finding the optimal policy for a given intermediate reward function) in an inner loop (the outer loop being finding the optimal reward function itself). Although IRL returns a reward function from which a policy may be learnt, the optimal reward function is not necessary. AIL methods use this as motivation to directly return a policy which imitates the expert, bypassing the intermediate IRL step completely. The Generative Adversarial Imitation Learning (GAIL) framework first shows that policies and state-action occupancy measures are uniquely related to each other. Next, the IRL is the dual of the occupancy matching problem. Using these ideas, finding a policy that imitates the expert is the problem of matching the state-action occupancy measures of the policy and the expert. A GAN framework is used where the discriminator is used to measure the discrepancy between the expert and policy state-action occupancies. The GAIL algorithm is given in Algorithm 2.

Algorithm 2 Generative Adversarial Imitation Learning

```
1: procedure GAIL(Expert trajectories $\tau_E = \{\tau_1 \dots \tau_n\}$, $T$, $\alpha$)
2:     Init policy $\theta$, Discriminator $\phi \sim$ RANDOM_INIT()
3:     for i = 0 to $T - 1$ do
4:         Sample trajectories $\tau_i \sim \pi_\theta$
5:         $l_D \leftarrow \mathbb{E}_{\tau_i}\left[\log\left(D_\phi(s,a)\right)\right] + \mathbb{E}_{\tau_E}\left[\log\left(1 - D_\phi(s,a)\right)\right]$
6:         $\phi \leftarrow \phi - \alpha \nabla_\phi l_D$
7:         $l_P \leftarrow \mathbb{E}_{\tau_i}\left[\log\left(\pi_\theta(a|s)\right) Q(s,a)\right] - \lambda H(\pi_\theta)$
8:             where $Q(s,a) = \mathbb{E}_{\tau_i}\left[\log\left(D_\phi(\bar{s},\bar{a})\right) | s_0 = s, a_0 = a\right]$
9:         $\theta \leftarrow \theta + \alpha \nabla_\theta l_P$
10:    end for
11: end procedure
```

Chapter 3

Related Work

Disaster scenario in indoor environments has been considered and looked into by [61], [14] where the crowd flow has been analyzed with simulated agents. Though these techniques assess simple fork in the corridor structures for navigation with simulated agents only, we study human participants and their biases in a larger simulated setting as an office building with many such structures. Neural sequence models have shown promising results in several domains like trajectory prediction [20] [44]. In particular, the transformer [67] is a popular attention mechanism for sequence modeling tasks and achieves state-of-the-art results on various benchmarks. But these models require a large amount of training data. As it is particularly challenging to obtain large human data in our setting, we discuss approaches with limited human data and our design of faux-human agents. LSTMs do not deal with irregularly sampled observations by default. To alleviate this problem, a model-agnostic representation of time [31] can be learnt and fed as an additional input. An alternative approach is to update the hidden state of the predictor between observations, such as through the use of an learned ordinary differential equation [57]. Methods based on Theory of Mind (ToM) framework reason in joint belief-intent space to reason about the demonstrator's behavior. Previous work [4] has shown that inferences using ToM models closely match predictions of human observers. However, these results were demonstrated in smaller settings, and it is challenging to scale Bayesian inference using ToM models to large environments such as ours. Incorporating ToM with neural networks is shown to be successfully to reason about machine agents, where ground truth about the internals of the decision making are available. But these approaches are yet to be applied to reason about human mental state. Unlike settings with a single goal in 11x11 sized grid-worlds without any structural priors [49], we consider a more realistic multi-goal task setup where the agent can be attributed with different types of strategies. The challenge of understanding humans' beliefs from behavior has also been studied in [52] where humans are assumed to act optimally but have incorrect knowledge of environment dynamics. We observe a similar case in our task setup where differing internal perspective on task complexity leads to variations across human behaviors.

For the past decade, transfer learning has been studied extensively [48, 63, 65, 75, 76] . It has been recently used in reinforcement learning, where multiple tasks are learnt instead of a single one. Knowledge gained in some Markov Decision Processes can be leveraged to speed up the solutions of others [6, 7, 8], In computer vision, Joint Distribution Adaptation is proposed for robust transfer learning [38], which jointly adapts the marginal and conditional distribution. Domain invariant features of the source and target are extracted for visual object recognition [5].

Transfer learning applied to graphs has recently gained attention. [12] proposes a network transfer learning framework using the adversarial domain and graph convolution. Although training and test data still require having the same feature space and distribution, new tasks that share similar representations can be resolved easier in [35], and this work transfers the geometric information from source to target. Graph-based domain mapping is used to identify previously encountered games, and this provides a good starting place for learning [34]. With graph based skill acquisition, [35] and [60] capture community detection from a connectivity graph, and speed up learning using the transferred knowledge. The theoretical grounded framework for the transfer learning of GNNs can be found in [74].

There are works on training an agent to navigate while adapting to new environments, including [73] where knowledge is transferred from previous navigation tasks using a successor-feature-based RL algorithm. In [42], the autonomous agent is trained to navigate in diverse city environments, while performing transferred tasks of navigating in target new locations. We build upon our previous work on the observing and predicting agent [28] by incorporating a graphical representation of the environment, amenable to inference using graph embedded Recurrent Neural

Network models and transfer to new domains (i.e., environment layouts). We refer readers to [27] [28] for the data collection process where rescuers are equipped with Minecraft skills.

Our approach to navigation prediction builds upon the Transfer Learning Diffusion Convolutional Recurrent Neural Network (TL-DCRNN) architecture presented in [41] to infer participant navigation, partitioning each domain's map utilizing the concept of a "clique" in a graph (a set of connected nodes) to form our clique group assignment for graph division. The original work predicts on traffic flow based on the Diffusion Convolutional Recurrent Neural Network (DCRNN) model proposed in [36]. It was able to predict the traffic flow of one city using the data collected from another. In our work, instead of using the partition for cities [40], we partition the graph based on the spatial recognition of the rescuers when performing the search task.

To mitigate the compounding errors in the naive supervised approach, [55] train an iterative algorithm where at each time step t, the policy π_t learns the expert behavior on the trajectories induced by $\pi_1 \dots \pi_{t-1}$. [55] also introduce a stochastic mixing algorithm based on [13]. The initial policy starts off as the expert policy, and at each iteration, a new policy is obtained by training on the trajectory induced by the previous policy π_{t-1}. The policy at timestep t is obtained by a geometric stochastic mixing of the expert and the previous policies. [56] train a policy using the expert demonstrations, generate new trajectories and use the expert to correct the behavior in these new trajectories iteratively. Although this method performs much better in a variety of scenarios, it requires access to the expert, which might be very expensive.

Another approach to tackling the problem is to use Inverse Reinforcement Learning. Inverse reinforcement learning attempts to find a reward function which best explains the behavior by an expert. Inverse reinforcement learning has shown successes in variety of tasks, including gridworld environments, car driving simulations, route inference based on partial trajectories, and path planning. The reward function is modeled as a linear function of the state features, and the weights are learned to match the feature expectations of the expert and the learnt policy [1], [51]. [77] use the principle of maximum entropy to disambiguate the underdefined problem of multiple possible rewards. Other methods like [50] use priors and evidence from expert's actions to derive probabilistic distributions over rewards.

Recently, adversarial imitation learning methods have shown successes in a variety of imitation tasks, from low dimensional continous control to high dimensional tasks like autonomous driving from raw pixels as input. [26] propose a framework for directly extracting a policy from trajectories without performing reinforcement learning inside a loop. This approach utilizes a discriminator to distinguish between the state-action pairs induced by the expert and the policy, and the policy uses the output of the discriminator as the reward. Different approaches build on top of this method, with [37] proposing an algorithm that can infer the latent structure of the expert trajectories without explicit supervision. This approach maximizes a mutual information term between the trajectory and the latent space to capture the variations in the trajectories. GAIL was further extended by [17] to produce a scalable inverse reinforcement learning algorithm based on adversarial reward learning. This approach gives a policy as well as a reward function. These approaches have led to faster imitation learning in both low and high dimensional tasks.

Chapter 4

Predicting Human Strategies in SAR

In a search and rescue scenario, rescuers may have different knowledge of the environment and strategies for exploration. Understanding what is inside a rescuer's mind will enable an agent to proactively assist them with critical information that can help them perform their task efficiently. The first part of the work is to build models of the rescuers based on their trajectory observations to predict their strategies. In our efforts to model the rescuer's mind, we use human trajectories in the Minecraft environments mentioned in Section 2.2. More specifically, we focus on the Falcon maps with different perturbations because of the decent amount of complexity in triage and navigation strategies that can occur. This becomes the first step towards building a MToM model that can predict future behaviors of the player based on their past trajectories. For simplicity, instead of encoding a prior in the form of a character embedding, we restrict the set of possible behaviors in triage and navigation strategies by binning them into certain distinct groups. This is based on the assumption that individual deviations from these triage strategies and navigation patterns is minimal, which we found to be a valid assumption given the data. We use these biases to train neural sequence models which predict the triage strategies.

4.1 Approach

To train the neural sequence models, we require a large amount of data. Since it is expensive to collect such a large dataset of human behavior in this setting, we augment our dataset of trajectories with a collection of faux-human agents. The faux-human agents are AI agents designed to quickly search the map and triage as many victims as possible. Since our navigation and triage strategies fall into different categories, the faux-humans we design are based on rules that implement these strategies. These agents are then run on maps with different perturbations in victim locations and rubble, to obtain trajectories that capture the variability in the actions taken for a particular high-level strategy. To avoid the complexity of performing actions in a big 3D space in the Minecraft environment, we reduce the state-space by discretizing the environment at the block level, and reducing it to a 2D minimap instead. For this, we use the Minigrid environment as the environment of choice, to which we transfer the Falcon maps for faux-human data collection and easy visualization. These faux-humans act 'optimally' under the given strategies and do not show spikes of random/suboptimal behavior like humans. Humans do not always act rationally, and therefore, these naive faux-human agents' behavior may be quite different from human behavior. We ameliorate this issue by collecting a small set of pilot human data and incorporating the rescuers' observed biases into the faux-human agents.

Some of the observed biases in the rescuer's decision-making were: (1) choosing subgoals in a soft-optimal manner (modeled using the Boltzmann distribution), (2) planning over room sequences instead of low-level actions, and (3) using greedy frontier-based search for short-horizon combined with long term planning aligned with the cliques in the graph representation of areas of a map. We thus obtain a rich and diverse set of faux-human agents that incorporate human biases while optimizing for the task objective.

We model the rescuer's intent across the victim saving (triaging) strategies, and navigation behavior in terms of next area to visit. First, we consider two victim saving strategies - saving the critically injured first (selective) or saving whoever comes first (opportunistic) - formulating it as a binary classification problem. These preferences tend to change in humans with time and proximity to the victims, making it a challenging sequential binary prediction task. For example, a rescuer may start the mission with 'opportunistic' strategy but may switch to be 'selective' on seeing the critically injured victims die. Another case where the rescuer may switch saving strategy is when a less critically injured (green) victim is close by in an area that is hard to reach later, we can expect a change in preference from being selective only to attend to the victim nearby. Second, given the area segments of the rooms and corridors from the original floorplan, we formulate the next location prediction as a multi-class classification problem. Both approaches are sequential classification tasks, where we learn to predict from the observations of a given trajectory.

We evaluate neural sequence models like RNNs and multi-head attention based transformer model [67] on our dataset. We compare this approach with a rule-based evidence accumulation method for prediction. This rule-based system is effective when we know the relevant evidence to track based on the rescuer's decision-making model and its knowledge state. In contrast, the neural sequence model learns what evidence to use and how to use it in an end-to-end manner using data.

4.1.1 Evidence accumulation approach

In this approach, we are explicitly incorporating full knowledge of SAR task and admissible strategies. We maintain a belief over the likelihood over each of the classes for the strategy prediction per human trajectory. At the initial state with no evidence, let the belief vector have a uniform distribution for each strategy as a prior. We assume access to a library (or a look-up table) for evidences e_i and their corresponding operation f_i to update the belief. Throughout the rescuer's trajectory, this approach provides a way to update the likelihood sequentially depending on the evidence for each strategy/intent and predict the one which is the most likely, as shown in Algorithm 3.

Algorithm 3 Evidence accumulation algorithm to predict most likely strategy/intent

```
1: procedure EVIDENCE ACCUMULATION(Rescuer trajectory $\tau_0^n$, evidence library
   $E = \{e_1 : f_1(\mathbf{b}), \dots e_m : f_m((b))\}$, Condition set of strategy/intent $\mathcal{C}$ of size $d$)
2:     Belief vector $\mathbf{b_t} = \left[\frac{1}{d}, \dots \frac{1}{d}\right]$ at $t = 0$;;
3:     for $t \in \{0 \dots n-1\}$ do
4:         if evidence $e_t$ found in $\tau_o^t$ then
5:             Obtain a function from the evidence library $f_t = E(e_t)$;
6:             Update belief vector $\mathbf{b_{t+1}} = f_t(\mathbf{b_t})$;
7:         end if
8:         $i = \arg\max \mathbf{b_{t+1}}$ ;
9:         most likely condition at time $t = \mathcal{C}[i]$ ;
10:    end for
11: end procedure
```

Though the evidence accumulation algorithm depends on knowing every e_i and its appropriate operation f_i to update the belief, it provides an upper bound on the prediction, limited with only individual differences and momentary variations. For specific evidences used for triage and next location prediction, refer to the Appendix A.1.

4.1.2 Neural sequence models

In settings where we have sufficient data in terms of the rescuer's trajectory but with no knowledge of specific evidences for prediction, we need to learn to infer them implicitly from their trajectory data. In this case, we train neural sequence models on faux human trajectories to evaluate their performance on the human data. For victim triaging strategy, we train the three variants of recurrent neural networks, namely LSTM + RNNDecay [57], LSTM + RNN-ODE [57] and LSTM + Time2Vec [31]. We observed the best performance of LSTM + Time2Vec [31] and use this model for comparison in table 4.1. For detailed analysis of all three methods, refer table A.1 in Appendix A.1. The input to each of the models is a sequence of observations, each of which consists of vector representation of the time, the 2D coordinates, and the condition of the victim seen (critically injured or not). For the

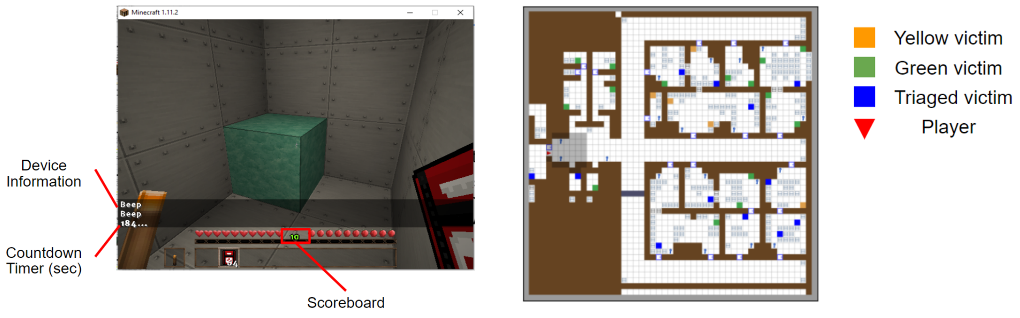

Figure 4.1: Human participants as rescuer see the view in (a). Human observers view the replay of the trajectory as in (a) and the bird's eye view of the environment as in (b).

next location prediction, we formulate the input as a sequence of areas visited. When the rescuer enters a new area segment, we take as input all the previously visited areas to predict the next area that they will transition to. We use a 2-layered 2-head Transformer model [67], which is neural network of encoder – decoder structure with partial masking of input for sequential prediction. Unlike the evidence accumulation approach, we do not provide the map connectivity. Rather, it is inferred from the input sequences by the network. We provide further details for both the models in Appendix A.1.

4.2 Experiment setup

4.2.1 Task scenario

We build a damaged office building in Minecraft environment as the task scenario for rescuers. The game screen and a 2D map layout of this environment are shown in Fig 4.1.

The scenario in the Minecraft environment represents a structurally damaged office building after an unspecified incident. Initially, it contains 26 area segments consisting of corridors, rooms, and elevators. The current building layout and segment connectivity were changed by perturbations such as collapses, wall openings, and sporadic fires. There are 20 injured victims inside the building who need to be rescued. Out of these, five victims are severely injured (yellow) and might die if not treated in time. Other victims are denoted in green. in Fig 4.1. Both victims depend on the first responders' help to stabilize and evacuate out of the building.

4.2.2 Human trajectories collection

Eight participants were recruited to collect real human trajectories in the search and rescue task. Participants were given detailed instructions on the search task and the virtual environment they would interact with. Each participant completed the same searching task 3 times on maps with 3 different perturbations. In each of the 15-min trial, human participants were asked to control a rescuer avatar and search for victims in the building. Player's position, interaction history and field of view were recorded as the game log in a sampling rate of 10Hz. In total 24 trajectories were collected from human players, 6 out of which were excluded from following analysis because of incomplete data record.

4.2.3 Human observer experiment

To provide a sound baseline of our agent prediction, we provide the same Theory of Mind inference task to human observers. The gameplay screen recordings and minimap videos were segmented by 'decision points' at which behaviors occur such as spotting a victim or leaving a room. Segments were viewed in the order recorded so that prior segments can inform judgments. Human observers were asked to supply a prediction of the expected action and choose among alternative beliefs and intentions from menus. The action taken was then presented at the start of the following sequence. Fifty workers on Amazon MTurk participated in the observation experiment. They were asked to predict five different types of rescuers' action in corresponding decision points. Refer to Appendix A.1 for a detailed definition of decision points and online survey design.

4.3 Results

The results shown in Table 4.1 highlight the prediction performance of our computational agent as compared to the human observers. To evaluate our approach,

Table 4.1: Comparison of strategy prediction accuracy.

Prediction Method	Triage strategy	Next location
Human observers	65.50%	58.20%
Evidence accumulation	**98.80%**	68.53%
Neural sequence	**70.74%**	66.98%

we compare it to an evidence accumulation method that explicitly incorporates all available background knowledge and thus provides an upper bound for the expected performance. The human observers were evaluated on a selected set of 10 decision timesteps for 18 human trajectories, while the two approaches were tested at every decision timestep relevant for triaging and next location prediction. The evidence accumulation approach is our intended upper bound for prediction accuracy. Within this setting, we observe that both computational methods outperform humans at the prediction tasks.

For triage strategy prediction, we obtain a high performance of 98.80% with evidence accumulation approach. This is a sequential binary classification task for which the selected evidence and belief update functions incorporate most of the domain knowledge. The neural sequence modeling approach implicitly infers from the trajectory observations and performs with an accuracy of 70.74%. The next location prediction is a 26-way multi-class classification, and a harder task than triage prediction. In this case, we found that devising exact evidences is challenging as human rescuers tend to be stochastic in the next location selection and therefore, we observe about 68.53% average prediction accuracy. The neural sequence models achieves a close performance with accuracy of 66.98%.

Overall, we observe that the computational methods outperform humans at both the prediction tasks. The results highlight evidence accumulation is a good enough approach to incorporate domain knowledge in limited data setting and can serve as an upper bound to the neural sequence model's performance. Further, neural sequence models can perform comparable to evidence accumulation approaches when the tasks get more complex, which makes it challenging to encode all the required evidences and their corresponding belief update functions for strategy prediction.

4.4 Discussion

This work aims to develop artificial agents that provide anticipatory assistance to human rescuers in disaster relief. Anticipatory assistance is important especially for dangerous and dynamic environments, such as disaster relief. To provide such assistance, the agent would observe the human and the environment and try to predict human's actions from observed behavior, so as to be able to e.g. warn the human of dangerous dynamic changes or mitigate shortcoming resulting from human cognitive limitations, such as limitations of memory or confusion over localization (e.g. the human spends redundant effort in revisiting places they have already searched). We have obtained initial encouraging results compared with humans in the role of the assisting agent.

One of the bottlenecks that we observed was the difficulty of obtaining human data for bigger maps. Sparky being a map with more manageable size was tractable to work with, however, the algorithms do not tend to scale much. For instance, the Falcon map is at least 4 times the size of Sparky, and the Saturn map is $\sim$4 times the size of Falcon map. Collecting data on Sparky and transferring learnt representations from the smaller map to the larger maps may be a viable strategy rather than training from scratch which will require more trajectories. In the next chapter, we try to tackle this problem with transfer learning, TL-DCRNN, and attention based models.

Chapter 5

Transfer Learning for navigation and triage prediction

In the previous chapter, we worked on both evidence based and neural prediction models for predicting human navigation and triage strategies. The entire suite of experiments was performed on the Sparky map, which is a smaller variant of the Search and Rescue maps. For bigger maps, the neural models need to adapt to a quadratic increase in the state space (we assume that the height of the map does not scale along with the length and width of the map) . However, collecting the same amount of data would not be sufficient due to curse of dimensionality and overparameterization of neural networks. Therefore, transfer learning and curriculum approaches are needed where the MToM agents are progressively trained on maps of increasing size.

To build an agent providing assistance to human rescuers in an urban search and rescue task, it is crucial to understand not only human actions but also human beliefs that may influence the decision to take these actions. Developing data-driven models to predict a rescuer's strategies for navigating the environment and triaging victims requires costly data collection and training for each new environment of interest. Transfer learning approaches can be used to mitigate this challenge, allowing a model trained on a *source environment/task* to generalize to a previously unseen *target environment/task* with few training examples.

In an urban search and rescue (USAR) task, human rescuers may navigate better and rescue more victims with the help of an artificial agent that observes and predicts their navigation and rescue activities, and opportunistically intervenes to give them assistance. Simple assistance and guidance include reminding rescuers not to revisit an area already visited, how to efficiently go to desired places, and whether they are likely to find victims that they would consider high priority in saving. The utility of an agent's advice to rescuers is dependent on the accuracy of agent's predictions of the rescuer's intents; interventions based on incorrect predictions may be misleading. Behavior prediction in an USAR task is challenging due to various factors, including but not limited to (a) incomplete information about where victims are, (b) changes in environmental and victim conditions , and (c) difficulty in obtaining data on rescue missions performed by humans. In a natural disaster scenario, such as after an earthquake, traversability of a building may change due to holes opening in walls or debris blocking passages, requiring rescuers to form ad-hoc navigation strategies during exploration. Rescuers also may be faced with decisions regarding *triage* (providing essential medical care to victims) priority. For example, a rescuer may decide to temporarily disregard lightly injured victims in order to search for and triage critical victims first, triaging lightly injured victims later.

In USAR tasks, real data is expensive to obtain, and is usually associated with unique configurations of the environment. This raises the question: how can an agent learn navigation and victim triage prediction in USAR task that can efficiently generalize and transfer to more complex environments and tasks? Representation and abstraction of spatial recognition in the humans has been studied widely. The knowledge human navigation relies on can be primarily characterized by a labeled graph [11]. Instead of perceiving environments into a global coordinate system, this cognitive map built from local information cannot guarantee geometric consistency [72]. There is a general agreement that people have hierarchical representations of space [10, 21, 25, 64], and this typically leads to the wrong answer to trick questions such as "What direction is Reno from San Diego?" (Answer: Northwest). Many people know San Diego is in California and Reno is in Nevada, and the spatial perception on state locations misleads that on cities. Similar effects are found on a local scale where people cluster buildings and other landmarks together into regions. Distances between locations within a cluster are judged to be shorter than they actually are, while the distance between locations in different clusters are judged to be longer. Simulations of learning and question answering based on this hypothesized hierarchical organization have been developed and analyzed in various works [54], [69], [39]. The goal of the work in this chapter is to develop transfer learning methods that provide zero- or few-shot transfer for human navigation and triage strategies from a *source* domain to a more complex *target* environment and task. In particular, we developed effective transfer learning techniques for navigation prediction from a smaller to a larger indoor environment and for a larger number of victim injury criticality classes. In this work, we used the 3D Minecraft platform [15, 30] as a testbed. Specifically, we used the Sparky and Falcon maps from our USAR environments as source and target environments respectively. Both source and target environments represented building interior spaces. The two environments had different room layouts, and we collected human rescuer navigation trajectories while they performed the simulated USAR tasks. In this paper, we demonstrate several benefits of our approach: (1) the agent is able to make good predictions on the navigation and triage strategy of rescuers, (2) its prediction accuracy grows fast even with a small input number of trajectories, (3) the convergence time of training process is shortened significantly in the target domain while not affecting the accuracy, and (4) the experiment demonstrates the potential of transfer learning for USAR missions.

5.1 Approach

The transfer-learning agent we developed is able to predict the navigation and triage activities of rescuers in the source task, using training data from human Minecraft players. The agent divides the space using graph methods and utilizes the TL-DCRNN to predict navigation when the rescuers search for victims; it also uses LSTM to predict triage strategies when the rescuers found victims and made decisions. In this work, we transfer the prediction model which is trained with the old scenarios, to predict the navigation and rescue activities of human rescuers in an unobserved situation. The implementation code has been released.[1]

[1]https://github.com/sophieyueguo/tl_navi

5.1.1 USAR Domains

To train agents and evaluate the ability of agents to transfer to new domains, multiple USAR task domains were developed using two distinct building layouts (shown in Figure 5.1): a smaller map with fewer rooms (*Sparky*), and a larger, more complex map (*Falcon*). For each domain, victims were placed in specific locations of the environment, and the environment was perturbed with blockage and holes on the walls, which are not initially known to rescuers, but are encountered as rescuers navigate the environment. Blockages in the hallways and rooms might make certain paths in the map impassable, while holes on the walls opened other possibilities for navigation. For prediction of rescuer navigation, a total of four task domains were created by varying the location of victims and perturbations: a single variant of the Sparky map, and three variants of the Falcon map of increasing difficulty (*Falcon-easy*, *Falcon-medium*, *Falcon-hard*), shown in Figure 5.2.

For prediction of rescuer triage strategy, we created two domains using the *Falcon* environment. The first domain, *Falcon-2victim*, contains victims of two severity levels, which are common to all domains: *regular* victims require 7 seconds to triage, and *critical* victims require 15 seconds to triage, and will expire 5 minutes after the start of the mission. The second domain, *Falcon-3victim*, introduces an additional victim severity level, *medium* victims, which require 12 seconds to triage and will expire 7 minutes into the mission.

The USAR mission required a human player/rescuer to work in an indoor environment in a limited time where victims were scattered. The rescuer is equipped with a medical kit which enabled them to triage victims in a few seconds. A device that beeps to report if one or more live victims were inside a room was utilized, however the human participant may or may not have understood the significance of the beeping. Knowledge of the beeping was a condition that was varied in different trials, i.e. in some conditions the participant was told the meaning of the beep at the beginning of the experiment, whereas in other conditions, the participant may not have been given this information.

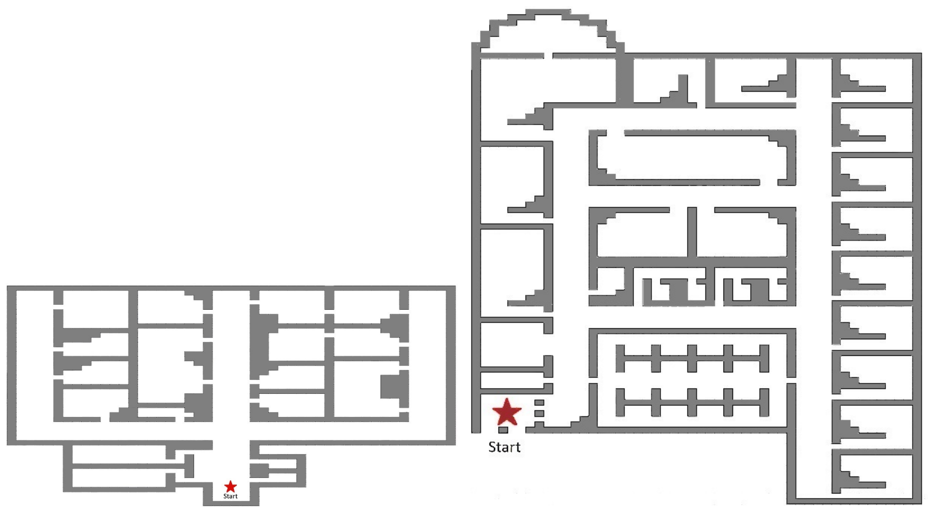

Figure 5.1: The Original Maps of Sparky (left) and Falcon (right). Red stars indicate start locations.

5.1.2 Agent Observations

The agent observes a stream of information containing the following values from the rescuer's trajectory:

- **Rescuer State:** The position and orientation of the rescuer in the environment,
- **Field of View Contents:** Blocks of interest (e.g., victims) that are present in the rescuer's field of view,
- **Victim Interaction Events:** Including starting, completing, and abandoning triage attempts,
- **Environment Interaction Events:** Including opening or closing doors, and entering or exiting rooms,
- **Victim Location Device:** Emitted beep when the rescuer is near a room containing an untriaged victim.

The Rescuer State and Field of View Contents update at a rate of 10Hz, all other observations are asynchronous events.

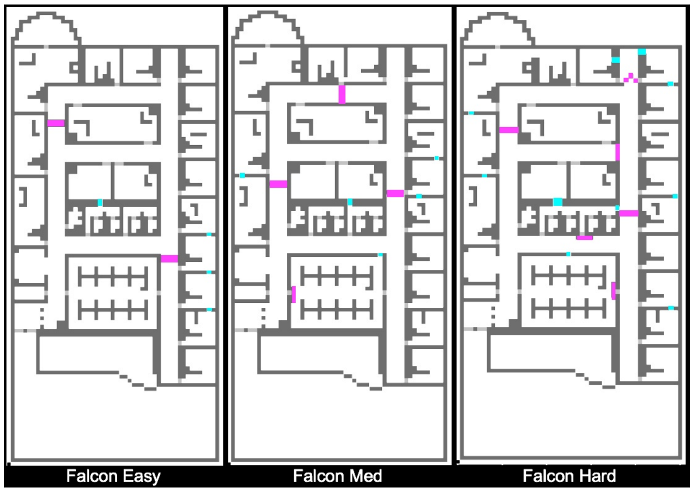

Figure 5.2: Map of the three versions of Falcon: easy, medium, and hard collected in [27]. Grey indicates walls, Magenta indicates blockages, and Cyan is for openings.

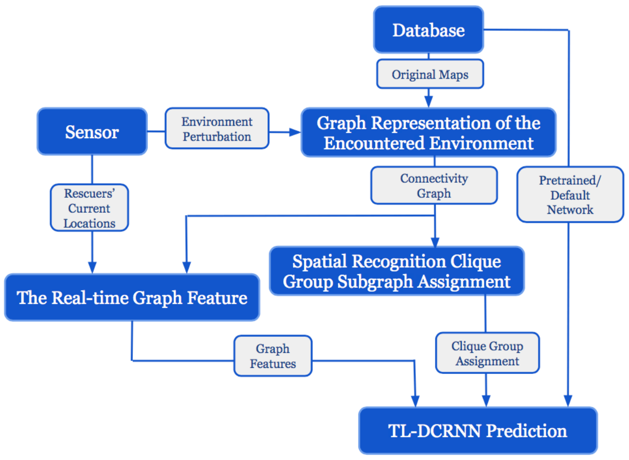
Database
Original Maps
Sensor
Environment Perturbation
Graph Representation of the Encountered Environment
Connectivity Graph
Pretrained/ Default Network
Rescuers' Current Locations
Spatial Recognition Clique Group Subgraph Assignment
The Real-time Graph Feature
Graph Features
Clique Group Assignment
TL-DCRNN Prediction

Figure 5.3: The Architecture of the Navigation Prediction Process

5.1.3 Navigation Prediction

The navigation prediction task involves the agent predicting the next room a test (unknown) human will enter next, predicted at particular time intervals. Figure 5.3 illustrates the main components / processes of our architecture (blue blocks) and data flow between the components (grey blocks). In addition to the agent's observation stream listed above, the agent is provided with a map of the environment and victim and environmental perturbations (e.g. , debris blockages and openings in walls), from which it generates a graph-based representation of the environment. The agent updates the representation based on victims and environment perturbations observed by the rescuer in the agent's data stream. Features are extracted from the graph-based representation, and a TL-DCRNN model is used to forecast future features, from which room visitation predictions can be made.

Note: The navigation strategy prediction was done by one of my co-authors Yue Guo. Please refer to our paper for more information on the graph architecture proposed for navigation strategy prediction.

5.1.4 Triage Strategy Prediction

In the triage task, the agent predicts over time the next victim type the human rescuer will triage. We hypothesize that a rescuer's triage strategy is agnostic to the location of the victim, and only depends on the reward that a rescuer gets in triaging a victim of a particular severity class of injuries.

Triage Strategy

We formulate the triage strategy prediction as a classification problem, using a sequence of Field of View observations (specifically, the location and severity of victims within the rescuer's Field of View) and observation timestamp as input.

Based on the set of victim severity levels, we identify four categories of strategies a rescuer can employ for triaging victims:

1. **Strict:** Rescuer exclusively triages *critical* victims during the first 5 minutes, *medium* victims from 5–7 minutes and *regular* victims from 7–10 minutes. In

the *Falcon-2victim* domain, the rescuer following this strategy will triage *regular* victims from 5–10 minutes.

2. **Slack:** Rescuer triages *critical* and *medium* victims as they are discovered during the first 7 minutes, then triage *regular* from 7–10 minutes.
3. **Preemptive:** Rescuer triages victims as discovered regardless of victim severity.
4. **Probabilistic:** Rescuer triages a discovered victim only if the expected number of victims that can be triaged in the remaining time is less than the number of victims remaining in the environment. The rescuer is made aware of the total number of victims from each severity class, and can therefore calculate both the expected and remaining number of victims.

Note that in the *Falcon-2victim* domain, the **Strict** and **Slack** strategies are equivalent, and is therefore considered a single category for this domain.

Attention based LSTM

We use a sinusoidal embedding for the position of the victims and the timestamp, following the works of [66]. At every timestep, we collect the list of victims, and use a common feedforward network E_ψ to convert the information into embeddings. To consider the case of no victims, we also include a "dummy" victim embedding at each timestep. Let there be N_t victims at time t. Each victim is depicted by a tuple of severity level, r, and (x, y) locations i.e. $v_i^t = (r_i^t, x_i^t, y_i^t), \quad i \in \{1 \dots N_t\}$. This representation is fed into the feedforward network to get the victim embedding $e_i = E_\psi(v_i)$. The embeddings are given by $\{e_i\}\Big|_{i=1}^{N_t} \cup \{e_0\}$ where e_0 is the learned dummy victim embedding. This set of embeddings is passed into a self-attention network A_θ where θ are the learnable parameters, giving us modified embeddings $\{f_i\}_{i=0}^{N_t} = A_\theta\Big(\{e_i\}_{i=0}^{N_t}\Big)$. Next, we take the average of these embeddings as the "summary" vector that goes into the LSTM, $s_t = \frac{\sum_{i=0}^{N_t} f_i}{N_t+1}$. This architecture allows us to account for variable number of victims at each timestep without making the architecture task specific. This vector is used as input to the LSTM. The final triage strategy prediction y_t is given by the LSTM equation $y_t = \mathrm{softmax}(g_\phi(h_t)), \quad h_t, c_t = \mathrm{LSTM}(s_t, h_{t-1}, c_{t-1})$, where h_t, c_t are the hidden and context state of the LSTM, and g_ϕ is a feedforward network. The recurrent architecture allows the network to predict

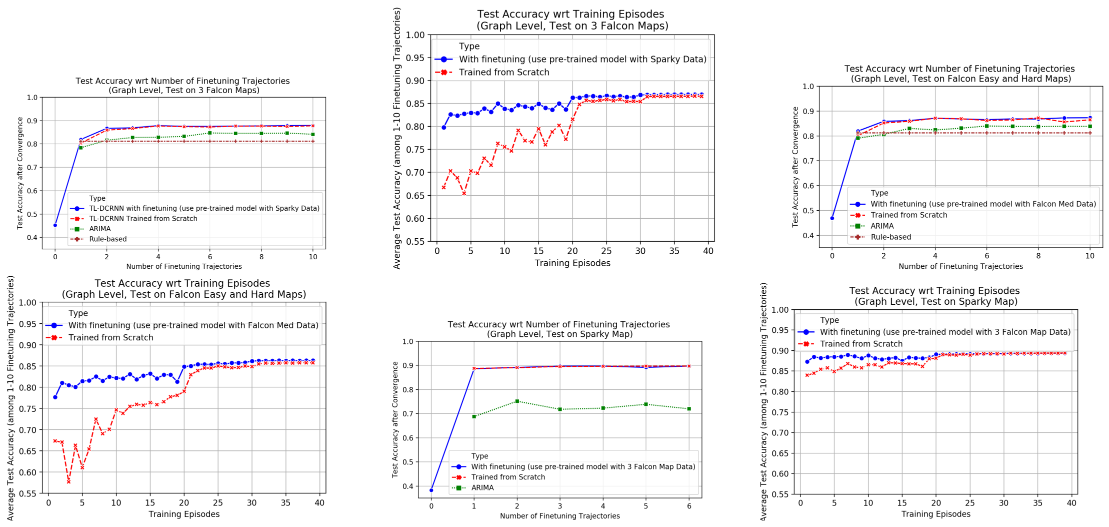

Figure 5.4: Graph Level Transfer Learning on Navigation Prediction among Different Maps and Perturbations

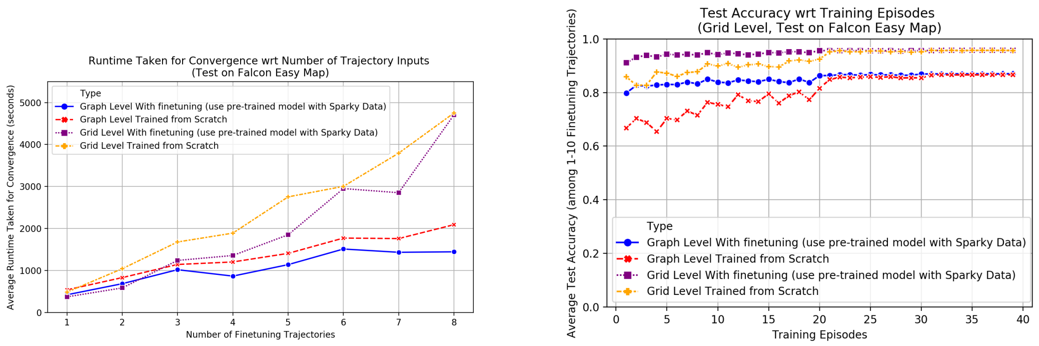

Figure 5.5: Grid Level Transfer Learning on Navigation Prediction from Map Sparky to Map Falcon Easy

the triage strategy by taking into account the historical behavior of the player in terms of which victims they triaged (can be inferred from a change of victim state to "saved") and which victims are neglected.

5.2 Results

We performed a series of experiments to evaluate the agent's ability transfer prediction models trained on a source domain to a target domain in the USAR domains described in Section 5.1.1. To train and evaluate the networks, we used a previously collected set of trajectories generated by human participants on the Falcon map variants [27], and collected trajectories generated by human participants on the Sparky map. In each experimental run, a single human participant navigated a given map and triaged victims, accumulating score points for each triaged victim. Score point were allocated in ascending order on seriousness of injury. Each participant was assigned to only one map or map variant, in the experiment. In other words no participant repeated the experiment in two or more maps.

For navigation strategy prediction, there were 8 trajectories of map Sparky for training, 138 trajectories of map Falcon for training, and 33 trajectories of map Falcon for test. Trajectories of different map Falcon perturbations were considered as different trials. For triage strategy prediction, we used a rule-based agent conditioned on rescue strategy to generate trajectories in the Falcon map. For domains for triage prediction (*Falcon-2victim*, *Falcon-3vicim*), we generated 100 trajectories for each triage strategy, resulting in a total of 300 trajectories for the *Falcon-2victim* domain and 400 trajectories for the *Falcon-3victim* domain. To introduce variation in the trajectories, we introduced perturbations in victim locations by making the victims do a "random walk" around their starting locations. Victim severity levels were also randomized in each training run. From each rescue strategy, we used up to 70 trajectories for training, 10 for validation, and 20 for testing. Evaluating the transferability of the prediction models to new domains involves pre-training a model on trajectories from a *source* domain (e.g., *Falcon-easy*), and evaluating the performance of the model on predicting trajectories from a *target* domain (e.g., *Falcon-hard*) after performing additional training on trajectories from the target domain.

5.2.1 Evaluation Metrics

For evaluating navigation prediction, we used Mean Average Error (MAE), as used in [36] and [41],

$$MAE(\mathbf{f}, \hat{\mathbf{f}}) = \frac{1}{|\Omega|} \sum_{i \in \Omega} |f_i - \hat{f}_i|$$

where the ground truth is represented by $\mathbf{f} = f_1, f_2 ... f_T$, the predictions are represented by $\hat{\mathbf{f}} = \hat{f}_1, \hat{f}_2 ... \hat{f}_T$, and Ω referred to the observed samples.

For triage strategy prediction, we used the mean accuracy over time (MAT) as our final evaluation metric. In the initial stages, the prediction can be random due to lack of rescuer's behavior data around the victims. Therefore, the average is taken only after one minute. Given a sequence of observations $\mathbf{O}$ and the ground truth rescue strategy $\mathbf{c}$, the metric is given by:

$$\mathbf{MAT}(\mathbf{O}, \mathbf{c}) = \frac{1}{|\mathbf{O}|} \sum_t \mathbb{I}(\mathbf{y_t} = \mathbf{c})$$

where $\mathbf{y_t}$ is the predicted strategy at timestep t. $\mathbf{c}$ is one of the four classes mentioned in Section 5.1.4.

5.2.2 Triage Strategy Prediction with Transfer Learning

In this section, we compare the performance of the model trained on the harder *Falcon-3victim* domain from scratch, and transferring a model pre-trained on the easier *Falcon-2victim* source domain, and finetuning on few trajectories from the *Falcon-3victim* target domain, reducing the required training data and time. We therefore perform the following experiments:

1. Models trained on the *Falcon-3victim* domain from scratch, i.e. without finetuning, with a varying number of training trajectories.
2. Models pre-trained on the *Falcon-2victim* domain as a source domain with maximum number of training trajectories, and transferring the model to the *Falcon-3victim* as the target domain by finetuning with varying number of target-domain trajectories.

The results are shown in Figure 5.6. The dotted plot shows the test accuracy with 70 training trajectories in the *Falcon-2victim* domain. We train with 1, 2, 5, 10, 20 and 50 training trajectories from each strategy. Finetuning improves generalization performance even given a small number, such as 2, of finetuning trajectories from target *Falcon-3victim*. In our case, the source task does not contain information about the target task at all (for example, no *Falcon-2victim* trajectory contains information about *medium* severity victims), so the finetuning indeed improves generalization.

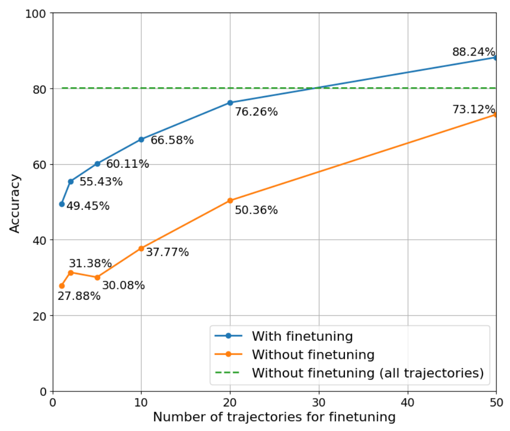

Figure 5.6: Transfer learning on triage strategy prediction

Moreover, using even as few as 30 trajectories for finetuning the *Falcon-2victim* source model reaches the performance of the *Falcon-3victim* model trained from scratch with *all* trajectories (green constant line in the figure) and for more than 30 finetuning trajectories, it outperforms the *Falcon-3victim* model.

5.3 Discussion

We built an agent that makes predictions on the navigation and rescue strategies of a human rescuers in a simulated urban search and rescue mission. We showed experimentally that: (a) using an abstract representation, i.e graphs enables efficient navigation strategy transfer from source to target domains from smaller to large maps with different victim and perturbation configurations; (b) for triage strategy, training a source model with smaller number of victim classes and adding a few finetuning trajectories form the target domain with larger number of triage victim classes, is not only high performing and efficient, but surprisingly outperforms a model trained from scratch in the target domain, while also converging faster. This is an interesting finding and we will explore it further in additional domains in future work. We also plan to study transfer with a team of rescuers instead of a single one. This is a very challenging task, not only because of the increase in the number of humans but also because of the inter-dependency of rescuer policies since they coordinate as a team.

These works, however, can still benefit if we can build faux-human players that can imitate these humans from nothing but their trajectories. This would allow us to generate more trajectories from these faux-humans that we can use to train these methods. The final part of the thesis aims to build an imitation learning framework that is significantly faster than existing IL methods, and also works with bigger state spaces.

Chapter 6

Augmenting GAIL with BC for sample efficient imitation learning

In this chapter, we attempt to solve the problem of imitation learning, where a task has to be performed by an agent using only expert demonstrations. The agent cannot query for more information from the expert in an iterative manner. One approach to solve the problem is to use behaviour cloning, where learning from demonstrations has been formulated as a supervised learning task. However, supervised learning assumes the data to be i.i.d. which is an incorrect assumption since the action taken at a state influences the future actions that the expert might take. Here, the i.i.d. assumption is with respect to the transition function. In behavior cloning, the assumption is that the action for a given state will not influence the distribution of states that the agent sees later on. Due to this incorrect assumption, behavior cloning cannot deal with covariate shift. Therefore, the 'best' action for a state is chosen from the expert and no other environment dynamics are considered. However, in RL, the log-probabilities are weighted by the Q-function, which encode the dynamics of the transition function and rewards from future states.

This i.i.d. assumption leads to compounding errors in behaviour cloning, and a large number of expert state action pairs must be provided to mitigate this error. GAIL, on the other hand, is very sample efficient in terms of the number of expert trajectories required but is very sample inefficient in terms of environment interactions. Environment interactions in many such scenarios require massive amounts of compute time and space. This is all the more problematic in real-world problems where additional concerns of safety, wear and tear and cost also kick in. This calls for the need of sample efficient algorithms which require minimal environment interactions.

Off-policy imitation learning may be a viable strategy, however, truly off-policy data is hard to learn from ([18]). Off-policy algorithms are also difficult to implement and often require delicate replay buffer manipulations. Off-policy may also not be an option when there are ethical or privacy related issues regarding persistent storage of data (EU regulations, for example). To that end, we propose a strategy to perform imitation learning in an on-policy manner to outperform GAIL in terms of sample efficiency. This method uses the fact that behavior cloning is a fast learning procedure but cannot be used as a pre-training step for GAIL as shown in the experiments section. This method is fast to implement into an existing GAIL implementation.

Scalability of the environments in Minecraft is an issue for reinforcement and imitation learning environments which are brittle in learning, with respect to training seeds, batch size, and hyperparameters. Our work in this chapter aims to mitigate the stability issue so that training on environments with large state spaces becomes feasible. To validate our work to the community, we also run experiments on commonly used continuous control environments (MuJoCo) and an image-based environment (Car Racing) as well.

6.1 Pretraining in Imitation Learning

GAIL and BC offer complementary benefits for imitation learning, namely, asymptotic optimality and fast convergence respectively. Therefore, a natural question to ask would be if there are any obvious ways combine the two while keeping their respective benefits. One approach that has found repeated mentions in the literature is pretraining with BC and then finetuning the policy with GAIL. Although this sounds like a reasonable strategy, our empirical results show that pretraining with behaviour cloning did not help and the agent learns a suboptimal policy as compared to GAIL trained from scratch. This observation is not uniquely found by us, as demonstrated by [58], where they show that GAIL pretrained with behaviour cloning failed to reach optimal performance as compared to GAIL trained from scratch. The following subsection discusses the effect of warm-started neural networks, and why that may hinder learning in GAIL after pretraining with behavior cloning.

6.1.1 Suboptimal performance of warm-started neural networks

Pre-training networks has shown a number of successes in deep learning, from image classification to natural language inference among others. The success of pretraining lies in the fact that it can be used on a base model that can be used to finetune later to domain specific tasks with little data. However, [23] show that random initialization is very robust and performs *no worse* than pretrained networks. The networks take longer to train than pretrained networks, but their generalization errors are almost always better than that of pretrained networks as shown in their work. This holds especially true when networks are trained with less data, which is surprising. [3] takes this a step further and shows that warm starting a network might lead to poorer generalization although the training losses may be the same. In the context of imitation learning, behaviour cloning does not train with all the expert trajectories because some validation data is required to prevent overfitting. GAIL, however, can work with all of the data, and training can stop when the discriminator loss becomes stable or after a fixed number of environment interactions. Since the policy is warm-started with a fraction of the expert data during behavior cloning, it may lead to an overall poor generalization error when trained on the entire set of trajectories during GAIL training.

6.2 Approach

The motivation for our method is inspired by the fact that optimizing the behavior cloning term alone leads to the agent learning a mapping from states to actions in a few iterations. However, the i.i.d. supervised training objective does not consider the sequential decision making aspect at all. Since there is no information about the transition dynamics or the value of following an action at a state (Q-function), behavior cloning would be suboptimal unless a lot of data is provided. However, even in a limited data setting, behavior cloning can still learn important features that map states to probable actions. GAIL, on the other hand, is simply reinforcement learning with a learnt reward function which is provided by the discriminator. However, the rewards provided by the discriminator are not informative in the beginning of the training procedure, and changes along with the policy that adapts to this reward function. The uninformative rewards do not provide any strong signal that the agent can use to map states to expert actions. Experiments in Section 6.3.2 shows that simply adding a temporal dependency to the behavior cloning term can improve convergence speed over GAIL without even training the discriminator.

Formally, consider the behavior cloning loss which is given by:

$$\mathcal{L_{BC}} = -E_{\tau_E}[\log(\pi(a|s))]$$

In adversarial imitation learning, we also train a discriminator D parameterized by ω. The discriminator is trained by minimizing the loss

$$\mathcal{L_D} = -E_{\tau_E}[\log(D_\omega(s,a))] - E_{\tau_\pi}[\log(1 - D_\omega(s,a))]$$

And the policy is trained using a policy gradient algorithm:

$$\mathcal{L_P} = -E_{\tau_\pi}[\log(\pi_\theta(a|s))A_{\omega,\psi}(s,a)]$$

where the advantage A is estimated using the value network V_ψ and the discriminator D_ω:

$$A_{\omega,\psi}(s,a) = -\log(1 - D_\omega(s,a)) + \gamma \mathbb{E}_{s' \sim T(s'|s,a)}\left[V_\psi(s')\right] - V_\psi(s)$$

Let the expert trajectories be denoted by a dataset $\mathcal{D}$, where $\mathcal{D} = \{(s_1,a_1),(s_2,a_2),\ldots(s_N,a_N)\}$, containing tuples of states s_i and actions taken by the expert a_i. Let the state-action visitation probability be denoted by $\rho(s,a)$. The behavior cloning term can also be written as:

$$\mathcal{L_{BC}} = -\sum_{s,a} \rho_E(s,a)\log(\pi(a|s)) = -\sum_{s,a} \rho_\pi(s,a)\left[\frac{\rho_E(s,a)}{\rho_\pi(s,a)}\log(\pi(a|s))\right] = -E_{\tau_\pi}\left[\frac{\rho_E(s,a)}{\rho_\pi(s,a)}\log(\pi(a|s))\right]$$

This is nothing but a simple manipulation based on importance sampling that allows us to directly add this term to the GAIL term, giving us the final loss function:

$$\mathcal{L}_{\mathcal{BC}} = -E_{\tau_\pi}\left[\left(\frac{\rho_E(s,a)}{\rho_\pi(s,a)} + A_{\omega,\phi}(s,a)\right)\log(\pi(a|s))\right]$$

The new advantage term inside the RL term intuitively adds an advantage for greedily following the expert action at a given state. Since the expert is only available indirectly in the form of samples and assuming a deterministic policy, the value $\rho_E(s,a)$ can be replaced with a Kronecker delta function

$$\rho_E(s,a) = \delta_{\mathcal{D}}(s,a) = \begin{cases} 1 & \text{if } (s,a) \in \mathcal{D} \\ 0 & \text{otherwise} \end{cases}$$

This leads to a very interesting interpretation of the first advantage term. If the expert did not perform some action then there is zero advantage in performing that action, but if the expert does perform the action a at state s, the advantage term is $\frac{1}{\rho_\pi(s,a)}$ which gives more advantage to the agent for following this behavior if the agent does not follow this behavior already. If the agent takes action a at state s and so does the expert, then the advantage term is close to 1, which is still positive, but the advantage decreases as the agent starts imitating the expert more precisely.
More generally, a weighted sum of the behavior cloning and GAIL term can be used with coefficients α and $1-\alpha$ with $\alpha \in [0,1]$ resulting in the following policy gradient term:

$$-E_{\tau_\pi}\left[\left(\alpha\frac{\rho_E(s,a)}{\rho_\pi(s,a)} + (1-\alpha)\,A_{\omega,\phi}(s,a)\right)\log(\pi(a|s))\right] = \alpha\mathcal{L}_{BC} + (1-\alpha)\mathcal{L}_{\mathcal{P}}$$

However in a practical scenario, with lack of data, the behavior cloning term may lead to overfitting and choice of α becomes crucial. Setting α too high can lead to the GAIL term not having enough impact, and setting it too low does not provide the apparent benefit of fast feature learning from the BC term. However, we observe that α only needs to be high in the initial stages, and can be ignored in the later stages of training since GAIL rewards would become informative then. Therefore, simulated annealing is a very elegant way to set the value of α.

Simulated annealing is used in optimization techniques for approximating the global optimum of a function. A common use of annealing is done in the learning rate in training of deep neural networks. [47] optimize a sequence of gradually improving mosaic functions that approximate the original non-convex objective using an annealing scheme. [19] use an exponentially moving average of the parameters of the target Q function at each time step. [29] use an annealing scheme to stabilize training in the context of semantic segmentation in medical images.

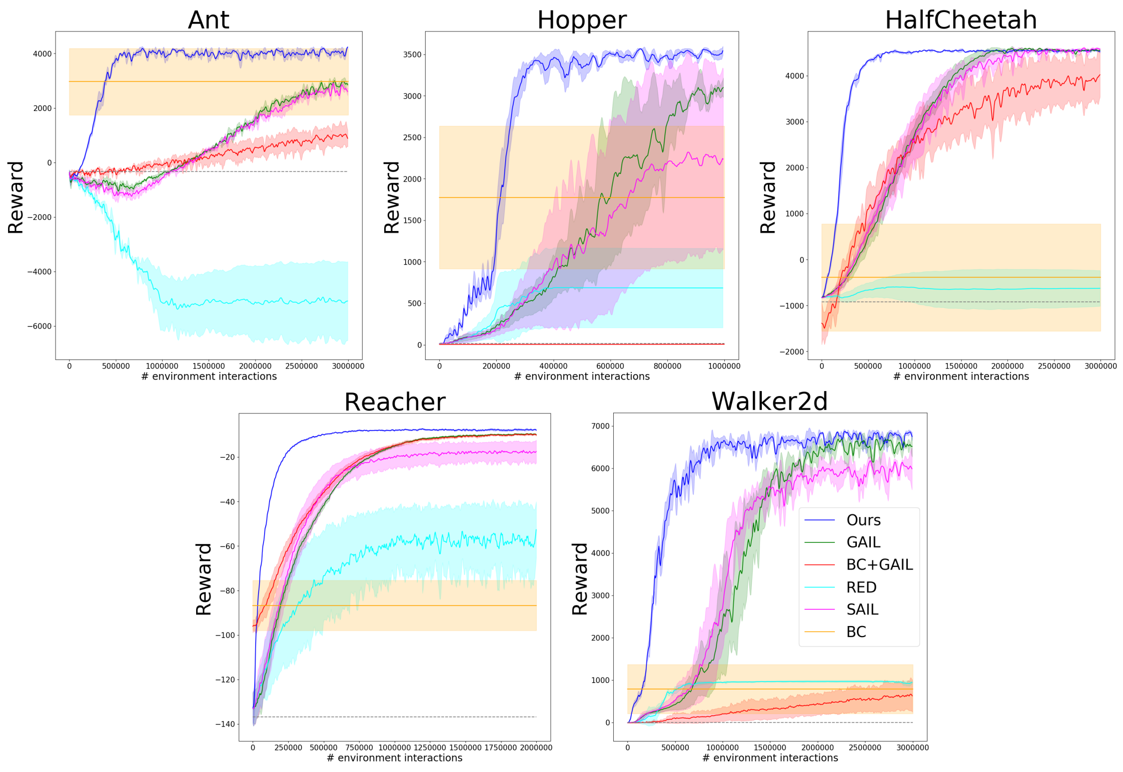

Figure 6.1: Performance of different imitation learning algorithms on MuJoCo tasks. All methods are tested with 3 random seeds.

Following these works, we use simulated annealing. The weighing parameter α is annealed out such that as the number of iterations $t \to \infty$, the optimization looks identical to GAIL, which provides better asymptotic performance. Specifically, at iteration t, we train the policy using the following loss:

$$L_{Total}^{(t)} = \alpha_t \mathcal{L}_{\mathcal{BC}} + (1 - \alpha_t)\mathcal{L}_{\mathcal{P}}$$

where $\alpha_t \in [0, 1]$. Note that $\alpha_t = 0$ corresponds to training with GAIL, and $\alpha_t = 1$

corresponds to behaviour cloning. In our case, we anneal α_t from 1 to 0, which transitions the gradients from a greedy action-matching behaviour to gradually accounting for more long term reward. The policy produces better-than random behaviour in the initial iterations which provides the discriminator with better trajectories from the policy. The tradeoff parameter is annealed using an exponential decay $\alpha_t = \alpha_0^t$, with $\alpha_0 \in (0, 1)$.

6.3 Experiments

	Ant	**HalfCheetah**	**Hopper**	**Reacher**	**Walker2d**
BC	2967.09 ± 1223.82	-389.14 ± 1166.19	1776.43 ± 858.93	-86.74 ± 11.25	788.82 ± 579.34
GAIL	2732.78 ± 1107.56	4546.93 ± 117.10	3035.83 ± 720.62	-9.78 ± 2.26	6718.34 ± 935.17
BC+GAIL	1237.88 ± 725.48	3808.17 ± 1298.91	14.13 ± 30.32	-9.77 ± 3.04	712.16 ± 542.22
RED	-4952.94 ± 1551.64	-626.47 ± 384.42	684.52 ± 478.09	-49.69 ± 42.90	940.27 ± 82.59
SAIL	2750.59 ± 938.21	$\mathbf{4584.18 \pm 86.88}$	2307.69 ± 1198.76	-14.31 ± 11.84	5993.21 ± 793.99
Ours	$\mathbf{3941.69 \pm 944.67}$	4558.09 ± 89.50	$\mathbf{3554.35 \pm 165.73}$	$\mathbf{-7.98 \pm 2.66}$	$\mathbf{6799.93 \pm 387.85}$
Random	-327.04 ± 790.06	-922.94 ± 97.30	15.17 ± 30.58	-136.72 ± 23.96	-3.03 ± 4.49
Expert	4066.96 ± 695.57	4501.09 ± 119.37	3593.06 ± 19.64	-3.92 ± 1.78	6512.85 ± 1116.62

Table 6.1: Performance of imitation learning algorithms. For each method, agents are trained with three random seeds. Final performance is measured by averaging across 60 rollouts for each method, 20 rollouts for each seed.

6.3.1 Low dimensional control tasks

We evaluate the proposed algorithm on a variety of continuous control tasks in MuJoCo. Specifically, we test our method on the **Ant, Hopper, Half Cheetah, Reacher,** and **Walker2d** environments. We compare our algorithm with the following baselines:

- **Behavior cloning:** Behavior cloning is a greedy approach to imitation learning. Although behaviour cloning is very fast since it does not require environment interactions, its asympotic performance is not optimal unless a lot of data is provided. Since our experiments do not use iterative data collection, we do not use the other behavior cloning baselines which use iterative feedback from experts [55], [56].
- **GAIL:** Adversarial imitation learning has been successful in a lot of environments. However, adversarial methods are shown to be unstable, and in the

presence of low amounts of data, can take a long time to converge.

- **BC+GAIL:** [26] mention that GAIL can be trained to converge faster by pretraining it with behavior cloning. However, they do not report the results for this baseline. [58], however, report that GAIL pretrained with behavior cloning does not work as effectively as GAIL. To make the baseline fairer, we also train the discriminator to differentiate between expert versus pretrained policy trajectories.
- **SAIL:** [71] is the only other method that claims to improve sample efficiency of GAIL without resorting to off-policy methods. Although, our experiments show that the claim is only partly true, since the asymptotic performance is not at par with GAIL (except for HalfCheetah where it performs only marginally better). The method also produces high variance policies across different random seeds which is not desirable.
- **Random Expert Distillation:** [70] is used in SAIL and it does not use adversarial training to learn a reward function. This method is also more sample efficient than GAIL, but that is only in the first few environment interactions and the peak performance is not as good as GAIL. Our results are consistent with the results reported in [71].

We use the code provided by [32] for implementing all baselines. For all experiments, we use a shared value and policy networks, which is an MLP with 2 hidden layers containing 64 hidden units with *tanh* nonlinearities, followed by their individual heads. For our algorithm, we choose α_0 according to the iterations taken for α_t to reach the value 0.5, which we denote as the 'half-life'. The half-life H is related to the value of α_0 as $\alpha_0^H = 0.5 \implies \alpha_0 = (0.5)^{\frac{1}{H}}$. We choose a half-life of 10 iterations across all experiments, which corresponds to $\alpha_0 \sim 0.933$. This value was found empirically by running the behavior cloning baseline and setting it equal to half the number of epochs it took for behavior cloning to converge. All algorithms (except the BC+GAIL baseline) are trained from scratch. Each algorithm is run across 3 random seeds, as done in [59]. The behavior cloning algorithm is trained only on 70% of the data, and 30% is used for validation. For all other experiments, all of the data is used. Note that although our loss contains a behavior cloning term, it does not require any validation data. The final performance of each method is evaluated by taking an average of 20 episodes for each seed. The final performance is shown in Table 6.1 and the reward curves are shown in Figure 6.1.

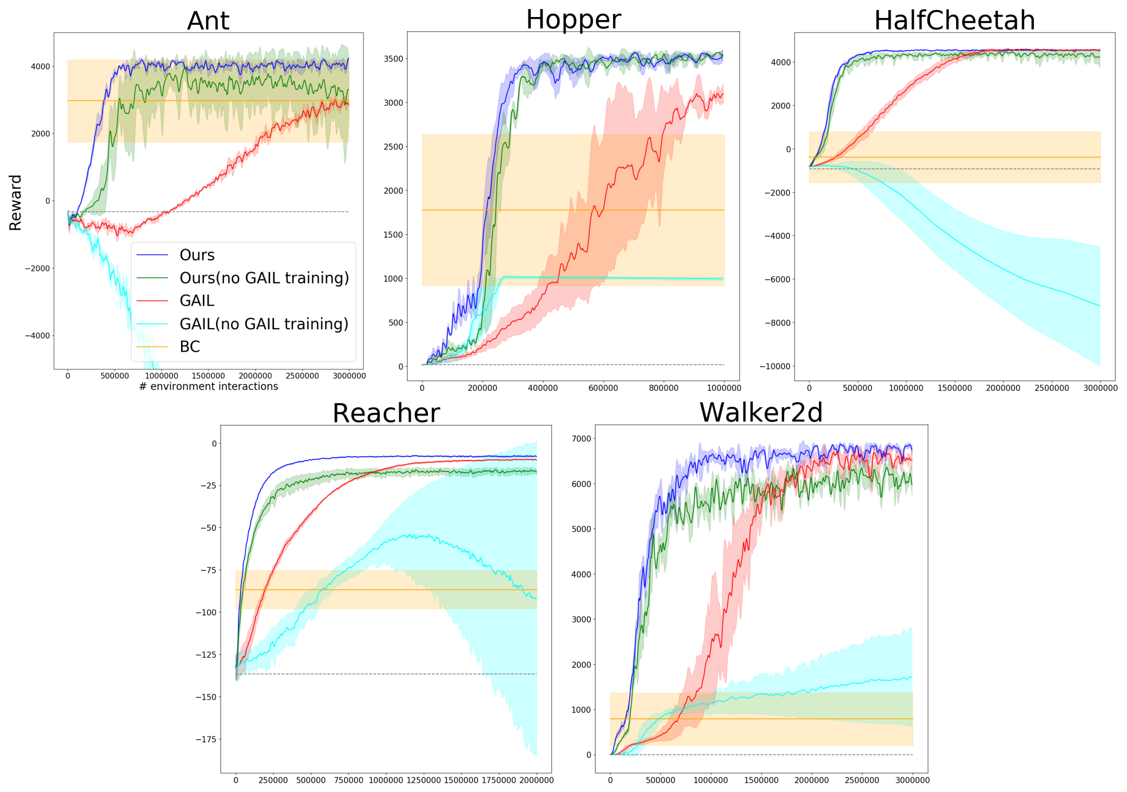

Figure 6.2: Performance of our method with and without discriminator training. Notice that our method outperforms BC even with random rewards from the discriminator, which shows that adding a temporal dependency in behavior cloning improves performance significantly.

Our method performs consistently across all the environments, whereas GAIL is very slow and behaviour cloning never reaches the best performance. SAIL seems to be outperforming GAIL initially, however, GAIL catches up and has better asymptotic performance than SAIL. SAIL also has a very high variance compared to other methods, potentially due to amplification of the variance of the two rewards used in their algorithm. The authors report the best agent performance across all seeds, which obscures the overall stability of the method. RED performs suboptimally because there is no feedback received from the agent to the reward function for adjusting its reward. Our method is very sample efficient as it learns much faster than GAIL, and in a lot of cases, converges to a slightly higher reward than GAIL.

6.3.2 Effect of temporal dependencies on Behavior Cloning

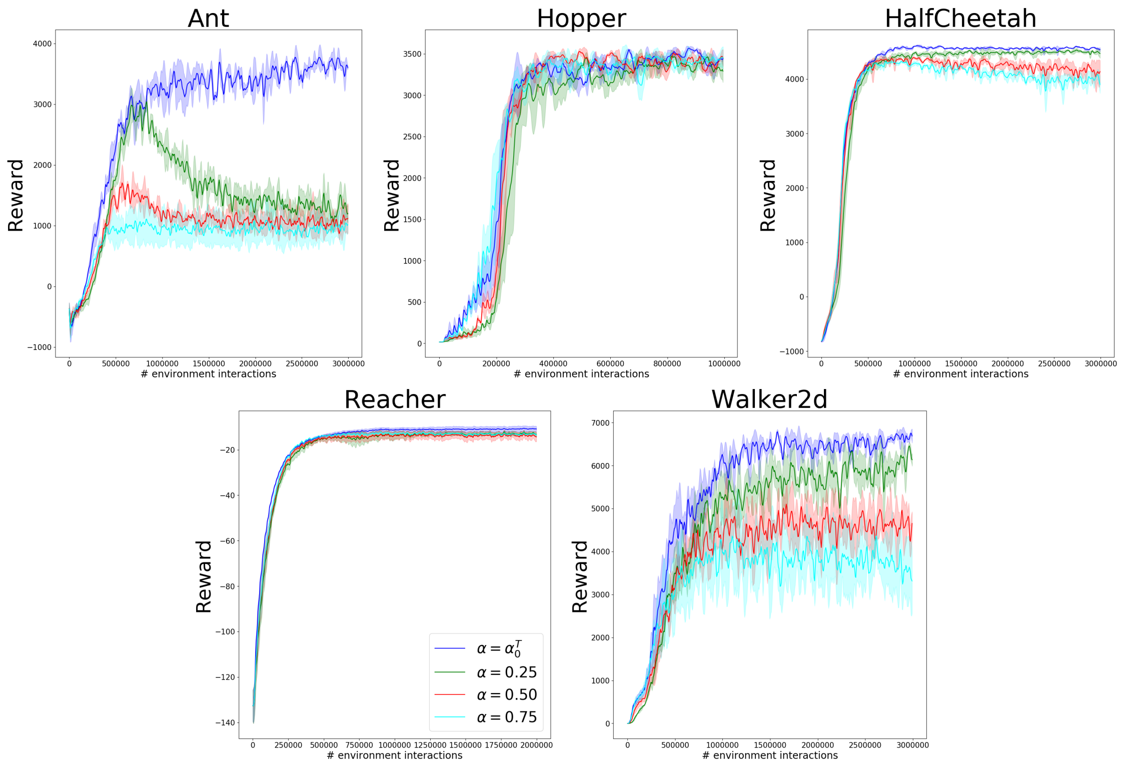

Figure 6.3: Performance of our method with and without annealing the tradeoff parameter. Notice that the final performance decreases with increasing value of α because the advantage term due to BC contributes in addition to the advantage of RL term, thus rendering the agent prone to overfitting. Our method reaches the best asymptotic performance and is more sample efficient than its constant α variations.

In Section 6.2 we hypothesized that behavior cloning fails most likely due to miscalibrated actions at out-of-distribution states. To analyse this effect we train agents with the our method, but we do not train the discriminator. The second term $A_{\omega,\phi}$ will not provide any useful signal since the discriminator is not trained. Therefore, the only useful signal can come from the behavior cloning term, and the GAIL term ensures that policy is trained with these uninformative advantage terms. Since the GAIL term is uninformative, we cannot anneal the value of α, otherwise the random rewards can interfere with the agent's learning. Therefore, we fix $\alpha = 0.5$ for this experiment. Figure 6.2 contains the performance of using our method with random rewards from the discriminator. The sample efficiency is better than GAIL, and the asymptotic performance is better than behavior cloning, which suggests that behavior cloning can be a powerful candidate for imitation learning. The untrained GAIL is also plotted to show the effect of potential reward bias that may occur. We observe that reward bias does not contribute to the task reward, with the exception of Hopper. This baseline is better than behavior cloning because the agent learns to output more random actions at the states which are outside the expert distribution because the uninformative advantage function does not prefer any action over the other in those states. The agent learns to perform the expert action at the states that are in the distribution of the expert. In contrast, behavior cloning never encounters out-of-distribution states during its training, and might output less random actions in those states due to network miscalibration [22]. The positive reward function offers bias only in encouraging survival and not necessarily in achieving a high task reward.

6.3.3 Effect of annealing

Next, we show the effect of annealing versus a fixed value of α on the final performance in all the MuJoCo tasks. To show that overfitting might be an issue, we limit the number of expert trajectories available to the imitation learning algorithms. Specifically, we use only 1 full expert trajectory for learning. The reward curves in Figure 6.3 demonstrate that as the value of α increases, the agent learns to imitate faster, but the asymptotic performance does not reach as far as the agents with a lower value of α. This is the speed versus performance tradeoff associated with α. To have the best of both worlds, α is annealed from a high value which promotes faster learning, and is annealed to 0 for better peak performance. The graphs show that the agent with annealing learns faster *and* achieves the best performance, especially in Ant and Walker environments.

6.3.4 Imitation learning with RL in Grid World environments

Imitation learning can also be used to provide a signal in addition to the environment rewards to enable faster learning ([9], [45], [24]), especially if the environment rewards are sparse. We use a gridworld environment as used in [62]. We evaluate on the “Key-Door” task, where the grid is divided into two rooms. The agent has to pick up a key, open the locked door and move to the goal location in the other room. The wall, key, door, goal location and agent are initialized at a different location every time, making the task harder. The agent recieves a reward of 1 for reaching the goal location, and 0 otherwise. This sparse reward does not provide information about the preconditions that need to be satisfied to reach the goal, i.e. picking the key and unlocking the door. The expert trajectories, however, contain this information and the agent is rewarded in the short-term horizon for imitating these behaviors. Therefore, imitation learning can be an extra learning signal for faster learning.

We use the code provided by [62] and extend it for training the imitation learning algorithms. The input is a $H \times W$ grid corresponding to the object present in each grid cell. To prevent the problems of reward bias in this setting [33], we follow the work of [37] and opt to use a Wasserstein GAN ([2]) framework which provides a reward that can be positive or negative. In addition, we use the REINFORCE algorithm as another baseline which uses the sparse reward. Experiments show that imitation learning can significantly boost learning compared to a sparse reward signal. The expert trajectories are collected from an A* agent. We test with grid sizes of 8, 10, and 12 to analyse the effect of progressively tougher environments. We use a total of 200, 350, and 500 expert trajectories for grid sizes 8, 10, 12 respectively. Since expert trajectories are small, there are a lot of unvisited states, and behavior cloning is expected to perform very suboptimally.

Figure 6.4 provides a comparison of all baselines. Behavior cloning performs suboptimally for all grid sizes, and its performance worsens with increasing grid size. Policy gradient learns slowly owing to a sparse reward function. In the case of grid size 12, REINFORCE only reaches about 70% of the performance of GAIL and our method after 30M steps. The performance of the BC+GAIL baseline drops to 0 after the behavior cloning stage, and never recovers, showing similar effects to that of the

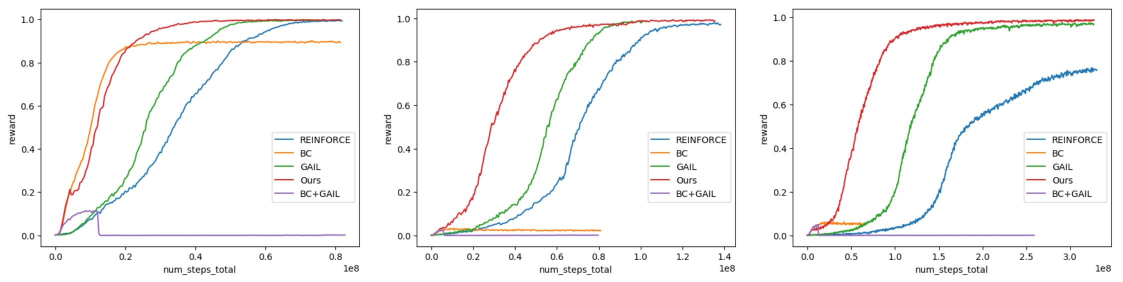

Figure 6.4: Performance in GridWorld environments. From left to right, the reward curves are for $8 \times 8, 10 \times 10, 12 \times 12$ grids respectively. Note that our method performs better than GAIL consistently across grid sizes. Behavior cloning is implemented by setting $\alpha = 1$ within our framework for ease of implementation (therefore the reward curve for behavior cloning).

MuJoCo experiments. This suggests that pretraining with behavior cloning is not a good option across environments and different RL implementations. However, our method reaches the same performance much faster than GAIL.

6.3.5 Imitation Learning in Image-based Environments

To demonstrate the effectiveness of incorporating the behavior cloning term into GAIL, we compare the variety of methods on Car Racing, a continous control task. In this environment,the agent must learn to keep a car on track from a top view of the car. We train an expert using PPO and collect 20 trajectories from randomly selected tracks. To train all agents, we concatenate the last four frames as the state as a single frame does not encode time dependent variables like velocity and acceleration. This setting is the same as [43] for training agents without learning recurrent networks.

	Score
Random	-75.01 ± 4.10
BC	695.36 ± 97.63
GAIL	419.82 ± 198.61
BC+GAIL	594.86 ± 263.12
Ours	$\mathbf{732.55 \pm 45.73}$
Expert	740.42 ± 86.36

Table 6.2: Performance on Car Racing environment

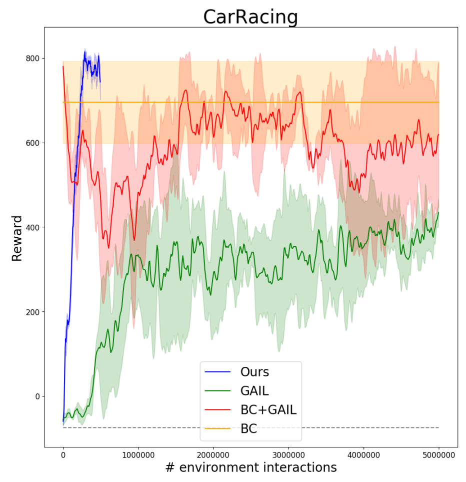

Figure 6.5: Reward curves on Car Racing environment

The results in Table 6.2 demonstrate that our method learns a policy that recovers a near-optimal policy from the expert trajectories. GAIL tends to fail because the agent has to learn image features from a noisy reward signal coming from the discriminator, which is significantly harder than MuJoCo due to curse of dimensionality. Since this environment can be solved greedily in most parts without solving the credit assignment problem, behavior cloning is already a very strong baseline [53]. Our method uses this aspect of behavior cloning and is able to recover a policy. The reward curve in Figure 6.5 shows that our method is at least 10x sample efficient than GAIL without resorting to any off-policy schemes. A model pretrained with behavior cloning starts off with a very good score, but the noisy GAIL reward interferes with its performance and this baseline also performs suboptimally. Our method is relatively stable and spares the usage of a lot of environment interactions.

6.4 Discussion

As demonstrated, we show that our method provides stability in adversarial imitation learning, especially in low dimensional tasks with less expert data and in high dimensional tasks where adversarial learning methods are unstable. In both low and high dimensional tasks, we observe that behavior cloning learns in a few iterations but performs suboptimally. GAIL learns much slower than behavior cloning but reaches optimal performance. Pre-training with BC collapses due to problems with initialization and warm starting, and the policy does not converge to peak performance during GAIL training. Our method combines the best of both worlds by maintaining optimal asymptotic performance, and learns in upto an order of magnitude faster than GAIL. Experiments on different types of environments demonstrate that our method improves substantially on GAIL by covering up for its one major weakness, its sample efficiency, without compromising on its stability or ease of implementation.

Chapter 7

Conclusions

Developing Machine Theory of Mind models for search and rescue scenarios is a challenging task. Search and Rescue tasks are based on navigation, memory of objects of interest that are seen, and good decision making. A theory of mind agent can observe a rescuer and make inferences about their belief states from observations and actions. Furthermore, the MToM agent can use these inferences to predict thei high level actions, such as the victims they will rescue, or the navigation strategy they follow. As a first step, we implemented rule-based and neural agents for triage strategy and navigation strategy prediction. To scale up the neural agents to maps of larger sizes and increasing complexity, we leverage a transfer learning scheme from trajectories in a smaller map to transfer knowledge into a larger map. This strategy can be used in a curriculum learning fashion with maps of increasing complexity, which can lead to MToM agents with better generalization capabilities. Finally, we explore the paradigm of imitation learning, which aims to learn a policy from expert trajectories alone. These policies can then be used to collect more “faux human” trajectories which will provide a richer dataset to train on. We deal the problem of sample efficiency and convergence speed of GAIL and augment it with a novel strategy to enable faster learning.

These steps are a foundational cornerstone of building scalable, functional and robust MToM models.

Appendix A

Appendix

A.1 Predicting Human Strategies in SAR

A.1.1 Evidence accumulation approach

For victim triage prediction, an event when the rescuer ignores less-critical victim is an evidence, say e_1. The corresponding update f_1 will increase the likelihood of the selective triage strategy in the belief vector $\mathbf{b_t}$ for the current time t. Similarly, if a less-critical victim is triaged upon sight, it is evidence e_2 for increasing the likelihood of opportunistic victim triage strategy.

For next location prediction, the map connectivity of the perturbed environment serves as a useful domain knowledge. We capture the layout as a graph where nodes are the area segments and the edges denote that areas are connected. At each step $e_i = (x, y)$ where x, y are position coordinates of the rescuer, which is in turn used to infer the area segment. The corresponding f_i increases the likelihood of each connected area depending on (1) the out degree of the node representing the area, and (2) the distance of the rescuer's position to that area. Another evidence is based on the fact that the rescuer is on an exploration task, which means they are unlikely to go to already visited areas and so, we decrease the likelihood of the area segments that have been visited.

A.1.2 Architecture details for neural-based sequence models

Victim triage strategy: We formulate the input to the neural network model as a sequence based on every event when a victim comes into the field of view of the rescuer, and when the rescuer finishes triaging a victim. We train LSTM + RNNDecay [57], LSTM + RNN-ODE [57] and LSTM + Time2Vec [31]. For the most part, we use the original architecture's hyperparameters except choosing batch size 1, and decay half-life as 60. The networks are trained to minimize the cross entropy loss with the groundtruth label with Adam optimizer with AMSGrad technique with learning rate 3e-4 and weight decay for regularization as 1e-5.

Table A.1: Comparison of baselines for triage strategy prediction

Prediction Method	Accuracy
Evidence accumulation	98.80%
LSTM + RNNDecay [57]	63.03%
LSTM + RNN-ODE [57]	67.44%
LSTM + Time2Vec [31]	70.74%

Next room prediction: We formulate the input as a sequence of rooms that the player moves to. For example such a sequence from a rescuer's trajectory is: 'Stairwell Starting Point', 'Center Hallway Lobby', "Women's Bathroom", 'Center Hallway Lobby', 'Elevator 1', 'Center Hallway Lobby', 'Center Hallway', 'Room 211', 'Room 213', 'Room 218', 'Right Hallway', 'Room 216', 'Room 209', 'Center Hallway', 'Room 208', 'Center Hallway', 'Room 210', 'Room 207', 'Room 210', 'Center Hallway', 'Room 215', 'Center Hallway', 'Room 208', 'Room 203', 'Room 201', 'Left Hallway', 'Center Hallway Lobby', "Men's Bathroom", 'Center Hallway Lobby', 'Left Hallway', 'Room 205', 'Left Hallway', 'Left Hallway', 'Center Hallway', 'Room 211', 'Room 213', 'Room 218', 'Right Hallway', 'Room 220' .

Since there are 26 area segments in the chosen map, we learn to predict the next room given a sequence of previously visited rooms. We use a neural network of 2-head, 2-layer transformer model, that is an encoder – decoder structure with multi-head attention over the masked sequence. We learn the first layer as a 26 dimensional embedding of each room area in the masked input sequence. This is processed by

Transformer encoder with hidden state dimension as 8. We use 5 steps for propagation through time. The rest of the hyperparameter are the same as existing pytorch code on transformer models [1]. Finally, the output is decoded by a linear weights such that each dim corresponds to the log likelihood of that being the next room.

A.1.3 Human observation experiments

Materials

In total 18 rescuer trajectories were used in the human observation experiment. Depending on the performance of original rescuer, the length of each trajectory range from 8 minutes to 15 minutes. Based on the collected human trajectories, we generated following materials: game screen recording videos, dynamic minimap videos and a static building layout image. Human observers can watch the first person screening recording of rescuers to understand what they were doing, and refer to the dynamic/static maps to help locate the rescuers' current location and navigation path. Video materials were were segmented by 'decision points' at which behaviors occur such as spotting a victim or leaving a room. The decision points are explained below.

- Triage decision points
  - Definition: When a victim block enters rescuer's FOV.
  - Prediction task: Will the rescuer triage the victim in front of him?
- Navigation decision points
  - Definition: When a room entrance (door/hole) enters rescuer's FOV.
  - Prediction task: Will the rescuer enter the room in front of him?
- General decision points
  - Definition: When the rescuer finishes triaging a victim / leaving a room.
  - Prediction task:
    What is the next location the rescuer will go?
    What is the rescuer's triage strategy?
    What is the rescuer's knowledge condition?

[1] https://github.com/pytorch/examples/tree/master/word_language_model

At each of the different decision points, human observers were given different prediction tasks including predicting next room and triage strategy of the securer etc. They were asked choose among alternative locations or strategies from menus. Video segments were viewed in the order recorded so that prior segments can inform judgments. The actual action taken by the rescuer in video was then presented at the start of the following sequence. The total number of decision points in one trajectory is around 300, which is too demanding for human observers to annotate. We sampled 10 decision points for each type and generated 30 video segments with corresponding prediction questions for each trajectory.

Procedure

50 human observers were recruited from Amazon Mechanical Turk. Participant accessed the online survey on their own computer. A detailed instruction was given to observers about the searching environment and the prediction task they need to complete. Then the observers need to pass a quiz about basic knowledge of our experiment in order to process to the formal task. Each observer was assigned one trajectory from human rescuer. In each of the 30 trials, human observers were presented a video clip and the prediction questions. The length of this human observation experiment is around 45 minutes.

Bibliography

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004. 3

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017. 6.3.4

[3] Jordan T Ash and Ryan P Adams. On the difficulty of warm-starting neural network training. *arXiv preprint arXiv:1910.08475*, 2019. 6.1.1

[4] Chris Baker, Rebecca Saxe, and Joshua Tenenbaum. Bayesian theory of mind: Modeling joint belief-desire attribution. In *Proceedings of the annual meeting of the cognitive science society*, volume 33, 2011. 3

[5] Mahsa Baktashmotlagh, Mehrtash T Harandi, Brian C Lovell, and Mathieu Salzmann. Unsupervised domain adaptation by domain invariant projection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 769–776, 2013. 3

[6] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado Van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1606.05312*, 2016. 3

[7] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *International Conference on Machine Learning*, pages 501–510. PMLR, 2018. 3

[8] André Barreto, Shaobo Hou, Diana Borsa, David Silver, and Doina Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117(48):30079–30087, 2020. 3

[9] Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E Taylor, and Ann Nowé. Reinforcement learning from demonstration through shaping. In *Twenty-fourth international joint conference on artificial intelligence*, 2015. 6.3.4

[10] Adrijana Car, George Taylor, and Chris Brunsdon. An analysis of the performance of a hierarchical wayfinding computational model using synthetic graphs. *Computers, environment and urban systems*, 25(1):69–88, 2001. 5

[11] Elizabeth R Chrastil and William H Warren. From cognitive maps to cognitive graphs. *PloS one*, 9(11):e112544, 2014. 5

[12] Quanyu Dai, Xiao Shen, Xiao-Ming Wu, and Dan Wang. Network transfer learning via adversarial domain adaptation with graph convolution. *arXiv preprint arXiv:1909.01541*, 2019. 3

[13] Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009. 3

[14] Rohit K. Dubey, Samuel S. Sohn, Christoph Hoelscher, and Mubbasir Kapadia. Cognitive agent based simulation model for improving disaster response procedures, 2019. URL https://arxiv.org/abs/1910.00767. 3

[15] Sean C Duncan. Minecraft, beyond construction and survival. 2011. 5

[16] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017. 2.1

[17] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017. 3

[18] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018. 6

[19] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018. 6.2

[20] Francesco Giuliari, Irtiza Hasan, Marco Cristani, and Fabio Galasso. Transformer networks for trajectory forecasting, 2020. URL https://arxiv.org/abs/2003.08111. 3

[21] Yongxi Gong, Yu Liu, Jian Yang, and Guicai Li. Structural hierarchy of spatial knowledge based on landmarks and its application in locality descriptions. In *2010 18th International Conference on Geoinformatics*, pages 1–5. IEEE, 2010. 5

[22] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks, 2017. 6.3.2

[23] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4918–4927, 2019. 6.1.1

[24] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal

Piot, Dan Horgan, John Quan, Andrew Sendonaris, Gabriel Dulac-Arnold, et al. Deep q-learning from demonstrations. *arXiv preprint arXiv:1704.03732*, 2017. 6.3.4

[25] Stephen C Hirtle and John Jonides. Evidence of hierarchies in cognitive maps. *Memory & cognition*, 13(3):208–217, 1985. 5

[26] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 4565–4573, 2016. 3, 6.3.1

[27] L Huang, J Freeman, N Cooke, M Cohen, X Yin, J Clark, M Wood, V Buchanan, C Carrol, F Scholcover, A Mudigonda, L Thomas, A Teo, M Freiman, J Colonna-Romano, L Lapujade, and K Tatapudi. Using humans' theory of mind to study artificial social intelligence in minecraft search and rescue. In *(to be submitted to the) Journal of Cognitive Science*, 2021. (document), 3, 5.2, 5.2

[28] Vidhi Jain, Rohit Jena, Huao Li, Tejus Gupta, Dana Hughes, Michael Lewis, and Katia Sycara. Predicting human strategies in simulated search and rescue task. In *Artificial Intelligence for Humanitarian Assistance and Disaster Response Workshop, NeurIPS*, 2020. 3

[29] Rohit Jena and Suyash P. Awate. A bayesian neural net to segment images with uncertainty estimates and good calibration. In *Information Processing in Medical Imaging*, pages 3–15, Cham, 2019. Springer International Publishing. ISBN 978-3-030-20351-1. 6.2

[30] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The malmo platform for artificial intelligence experimentation. In *IJCAI*, pages 4246–4247. Citeseer, 2016. 5

[31] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019. 3, 4.1.2, A.1.2, A.1

[32] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018. 6.3.1

[33] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. *arXiv preprint arXiv:1809.02925*, 2018. 6.3.4

[34] Gregory Kuhlmann and Peter Stone. Graph-based domain mapping for transfer learning in general games. In *European Conference on Machine Learning*, pages 188–200. Springer, 2007. 3

[35] Jaekoo Lee, Hyunjae Kim, Jongsun Lee, and Sungroh Yoon. Transfer learning for deep learning on graph-structured data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017. 3

[36] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017. 3, 5.2.1

[37] Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3812–3822. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/6971-infogail-interpretable-imitation-learning-from-visual-demonstrations.pdf`. 3, 6.3.4

[38] Mingsheng Long, Jianmin Wang, Guiguang Ding, Jiaguang Sun, and Philip S Yu. Transfer feature learning with joint distribution adaptation. In *Proceedings of the IEEE international conference on computer vision*, pages 2200–2207, 2013. 3

[39] Tamas Madl, Stan Franklin, Ke Chen, Robert Trappl, and Daniela Montaldi. Exploring the structure of spatial representations. *PloS one*, 11(6):e0157343, 2016. 5

[40] Tanwi Mallick, Prasanna Balaprakash, Eric Rask, and Jane Macfarlane. Graph-partitioning-based diffusion convolutional recurrent neural network for large-scale traffic forecasting. *Transportation Research Record*, 2674(9):473–488, 2020. 3

[41] Tanwi Mallick, Prasanna Balaprakash, Eric Rask, and Jane Macfarlane. Transfer learning with graph neural networks for short-term highway traffic forecasting. *arXiv preprint arXiv:2004.08038*, 2020. 3, 5.2.1

[42] Piotr Mirowski, Matthew Koichi Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Koray Kavukcuoglu, Andrew Zisserman, and Raia Hadsell. Learning to navigate in cities without a map. *arXiv preprint arXiv:1804.00168*, 2018. 3

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 6.3.5

[44] Alessio Monti, Alessia Bertugli, Simone Calderara, and Rita Cucchiara. Dag-net: Double attentive graph neural network for trajectory forecasting, 2020. URL `https://arxiv.org/abs/2005.12661`. 3

[45] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations.

In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018. 6.3.4

[46] Andrew Ng and Stuart Russell. Algorithms for inverse reinforcement learning. *ICML '00 Proceedings of the Seventeenth International Conference on Machine Learning*, 05 2000. 2

[47] Hengyue Pan, Xin Niu, RongChun Li, Yong Dou, and Hui Jiang. Annealed gradient descent for deep learning. *Neurocomputing*, 2019. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2019.11.021. URL http://www.sciencedirect.com/science/article/pii/S0925231219315802. 6.2

[48] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009. 3

[49] Neil C. Rabinowitz, Frank Perbet, H. Francis Song, Chiyuan Zhang, S. M. Ali Eslami, and Matthew Botvinick. Machine theory of mind. *CoRR*, abs/1802.07740, 2018. URL http://arxiv.org/abs/1802.07740. 2.1, 3

[50] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. In *IJCAI*, volume 7, pages 2586–2591, 2007. 3

[51] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM, 2006. 3

[52] Siddharth Reddy, Anca D. Dragan, and Sergey Levine. Where do you think you're going?: Inferring beliefs about dynamics from behavior. *CoRR*, abs/1805.08010, 2018. URL http://arxiv.org/abs/1805.08010. 3

[53] Siddharth Reddy, Anca D Dragan, and Sergey Levine. Sqil: imitation learning via regularized behavioral cloning. *arXiv preprint arXiv:1905.11108*, 2019. 6.3.5

[54] Emilio Remolina, Juan A Fernandez, Benjamin Kuipers, and Javier Gonzalez. Formalizing regions in the spatial semantic hierarchy: An ah-graphs implementation approach. In *International Conference on Spatial Information Theory*, pages 109–124. Springer, 1999. 5

[55] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668, 2010. 3, 6.3.1

[56] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011. 3, 6.3.1

[57] Yulia Rubanova, Ricky TQ Chen, and David Duvenaud. Latent odes for irregularly-sampled time series. *arXiv preprint arXiv:1907.03907*, 2019. 3,

4.1.2, A.1.2, A.1

[58] Fumihiro Sasaki, Tetsuya Yohira, and Atsuo Kawaguchi. Sample efficient imitation learning for continuous control. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BkN5UoAqF7. 6.1, 6.3.1

[59] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 6.3.1

[60] Farzaneh Shoeleh and Masoud Asadpour. Skill based transfer learning with domain adaptation for continuous reinforcement learning domains. *Applied Intelligence*, 50(2):502–518, 2020. 3

[61] Samuel S. Sohn, Seonghyeon Moon, Honglu Zhou, Sejong Yoon, Vladimir Pavlovic, and Mubbasir Kapadia. Deep crowd-flow prediction in built environments, 2019. URL https://arxiv.org/abs/1910.05810. 3

[62] Sainbayar Sukhbaatar, Emily Denton, Arthur Szlam, and Rob Fergus. Learning goal embeddings via self-play for hierarchical reinforcement learning. *arXiv preprint arXiv:1811.09083*, 2018. 6.3.4

[63] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018. 3

[64] Adriana Tapus, Shrihari Vasudevan, and Roland Siegwart. Towards a multilevel cognitive probabilistic representation of space. In *Human Vision and Electronic Imaging X*, volume 5666, pages 39–48. International Society for Optics and Photonics, 2005. 5

[65] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009. 3

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017. 5.1.4

[67] 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. 3, 4.1, 4.1.2

[68] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. *Advances in neural information processing systems*, 29:3630–3638, 2016. 2.1

[69] Horatiu Voicu. Hierarchical cognitive maps. *Neural Networks*, 16(5-6):569–576,

2003. 5

[70] Ruohan Wang, Carlo Ciliberto, Pierluigi Amadori, and Yiannis Demiris. Random expert distillation: Imitation learning via expert policy support estimation. *arXiv preprint arXiv:1905.06750*, 2019. 6.3.1

[71] Ruohan Wang, Carlo Ciliberto, Pierluigi Vito Amadori, and Yiannis Demiris. Random expert distillation: Imitation learning via expert policy support estimation. *CoRR*, abs/1905.06750, 2019. URL http://arxiv.org/abs/1905.06750. 6.3.1

[72] William H Warren. Non-euclidean navigation. *Journal of Experimental Biology*, 222(Suppl 1), 2019. 5

[73] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. Deep reinforcement learning with successor features for navigation across similar environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2371–2378. IEEE, 2017. 3

[74] Qi Zhu, Yidan Xu, Haonan Wang, Chao Zhang, Jiawei Han, and Carl Yang. Transfer learning of graph neural networks with ego-graph information maximization. *arXiv preprint arXiv:2009.05204*, 2020. 3

[75] Zhuangdi Zhu, Kaixiang Lin, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020. 3

[76] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020. 3

[77] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008. 3