Deep Reinforcement Learning :
Reliability and Multi-Agent Environments

A Dual Degree Project Report

submitted by

ABHISHEK NAIK

under the guidance of

PROF. BALARAMAN RAVINDRAN

Department of Computer Science and Engineering
Indian Institute of Technology Madras

May 2018

THESIS CERTIFICATE

This is to certify that the thesis titled **Deep Reinforcement Learning : Reliability and Multi-Agent Environments**, submitted by **Abhishek Naik**, to the Indian Institute of Technology Madras, for the award of the degree of **Dual Degree (B.Tech + M.Tech)**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Balaraman Ravindran
Research Guide
Professor
Dept. of Computer Science & Engineering
IIT-Madras, 600036

Place: Chennai
Date:

ACKNOWLEDGEMENTS

First of all, I would like to thank Professor Balaraman Ravindran, who has been a guiding light for almost three years now. It was his unique teaching style and unparalleled enthusiasm that got me hooked to Machine Learning and subsequently Reinforcement Learning, which has now evolved into a career, starting with a PhD. He inspired me to broaden my horizon and study topics which went beyond the syllabus [1], and his valuable feedback and ideas have improved the quality of this thesis considerably.

Shout-out to my support cast - wingmates, close friends, family. Words aren't enough to express the gratitude I have towards them, for keeping me sane with all the insane antics, stunts, conversations, in this particularly difficult year. In alphabetical order - Monu, Motu, Serdaar, Stack, Ujjawal. If I could, I would relive all these five years with you guys again. And Fahima, who made this past year all the more colorful and stimulating, and gave me a reason to diligently finish my work early every day.

Literally none of this would have been possible without my pillars of support - Akka, Amma, and Pappa. I shall forever be thankful for their unwavering faith and unquestioning enthusiasm for every crazy endeavor I have ever pursued.

And last but not the least, I would like to thank the amazing set of collaborators that I've had the good fortune to work with. In particular, Anirban, whose infectious enthusiasm was refreshing amidst the year of toil it has been for us. Along with him, sincere thanks to my peers in the TRL course, weekly RL meet-ups - who all enabled invaluable breadths and depths of learning and thinking with the insightful discussions and debates over screens, paper presentations, and *biryanis*. Thanks to all the fellow lab-mates and TAs I've had, for making this year a memorable experience beyond the walls of the lab.

[1] 'ancient' papers from before the turn of the century

ABSTRACT

KEYWORDS: Reinforcement learning; Deep learning; Risk-averse learning; Multi-agent learning; Curriculum learning

Reinforcement learning is a subset of machine learning algorithms concerned with learning to perform sequential decision-making tasks in unknown environments in order to maximize some cumulative reward. As the field of artificial intelligence evolves to solve real-world problems of genuine interest to humans, specifically that of autonomous driving, issues of safety and reliability of the people involved comes to the fore. Furthermore, these environments are too complex for classical (and) hand-designed solutions. We hypothesize that the renaissance of reinforcement learning techniques with the recent advancements in deep learning hold the key towards making such applications of Artificial General Intelligence a reality.

Towards this end, we adapt a three-pronged strategy - firstly, we illustrate the unreliability of the state-of-the-art imitation learning algorithm and propose a new risk-averse imitation learning framework, which empirically appears as a potent alternative for improved reliability in risk-sensitive applications. Secondly, to address the deficiencies of existing driving simulators in terms of independent control of multiple agents as well as the extent of customizability of traffic, we develop the first open-source, multi-agent driving simulator to serve as a quick-start prototyping platform for the reseach community. Finally, we observe the challenges of non-stationarity and hand-engineered reward functions in the multi-agent domain of RoboSoccer, and move towards a curriculum learning based approach of breaking the complex target task into simpler skills that the agent leverages in order to master the final sparse-reward goal, for which we successfully demonstrate a proof-of-concept methodology. We aim to consolidate these individual modules together as viable steps towards achieving the ultimate goal of revolutionizing the transportation industry by safely and reliably deploying a homogeneous set of connected self-driving vehicles on our roads.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

One of the primary driving forces of artificial intelligence is creating what is called as Artificial General Intelligence (AGI), which entails ideal learning agents that are capable of learning in a diversity of situations and environments throughout their life, without requiring extensive redesign for every new problem that they encounters. Such AI systems encompass a broad range of physical and software-based systems - ranging from robots that sense, interact, and react to the world around them (autonomous vehicles, for instance), to behind-the-scenes algorithms which interact with users to showing the right advertisements and product recommendations.

One such principled mathematical framework for trial-and-error, experience-driven learning is that of Reinforcement Learning (RL), which is a term borrowed from animal learning literature by Minsky (1954). RL intuitively involves an agent interacting with the environment, learning an optimal policy by trail and error for sequential decision making problems in a wide range of fields - in both natural and social sciences (Sutton and Barto, 1998). After the works of Barto *et al.* (1983) and Watkins (1989), the field really took off.

Though RL has had plenty of success stories in the past - from the champion backgammon player (Tesauro, 1995) to autonomous acrobatic helicopter-flying (Ng *et al.*, 2006), most of the classical approaches lacked scalablity and were inherently limited to fairly low-dimensional problems. The integration of reinforcement learning and neural networks has a long history, and the recent rise of Deep Learning (DL) (LeCun *et al.*, 2015), thanks to powerful representation learning and function approximation properties of deep neural networks - coupled with big data, powerful computational resources, new algorithmic techniques, mature software packages and architectures - has been critical in the renaissance of RL.

In the past few years, deep neural networks have shown strong performance on a variety of supervised learning tasks, and are now considered state-of-the-art general-purpose function approximators for the tasks of image recognition (Simonyan and Zis-

serman, 2015), text generation (Graves, 2013), speech recognition (Hinton *et al.*, 2012), etc. Such recent advances in DL like that of Convolutional Neural Networks (CNNs) (Krizhevsky *et al.*, 2012) have enabled RL agents see, and Long Short-Term Memory Units (LSTMs) (Hochreiter and Schmidhuber, 1997) have enabled RL agents to remember the important events in the past by aggregating their observations over time.

The field of Deep Reinforcement Learning (DRL) exploded with the Mnih *et al.* (2015)'s demonstration of super-human performance across a variety of different Atari 2600 video games, and a breakaway world champion of Go by Silver *et al.* (2016), a game which has long been considered a holy grail for artificial intelligence scientists. This marriage of deep learning and reinforcement learning techniques have made it possible for researchers to purdue increasingly complex, sophisticated, and ambitious goals, avenues which were prohibitive not a very long time ago - in terms of memory, computational, and sample complexity.

1.1 Contributions

One such ambitious goal is that of autonomous driving. It is innately exciting to envision a future with self-driving cars whipping through the streets, running on renewable energy sources, ferrying commuters from place-to-place safely and reliably - all with a tap of an app. Despite the driving acumen demonstrated by pioneers like Waymo and Tesla, plenty of challenges of intricate complexity remain - from reliably detecting obstacles and predicting their trajectories to understanding surrounding cars' and pedestrians' intents for planning moves into the future - all with a near-perfect accuracy, in real-time. For such applications of Artificial General Intelligence, we strongly believe RL is a key ingredient.

Towards making this exciting dream a reality, we break down this daunting task into practical sub-problems.[1] We make the following contributions within this three-pronged strategy :

1. **Risk-Averse Imitation Learning** - Safety is the first and foremost aspect of autonomous driving cars. Since even a single mistake in a risk-sensitive application

[1] An astute reader will realize the pun shortly

like autonomous driving puts lives at stake, we propose a risk-averse imitation learning framework (christened RAIL), which improves the reliability of imitation learning algorithms for deployment in risk-sensitive applications like autonomous driving or robotic surgery. Complete details of RAIL are presented in Chapter 3.

2. **Multi-Agent Reinforcement Learning** - Negotiating in traffic to move from one place to another safely and reliably involves interactions with a plethora of entities. Towards this end, we perform a series of experiments in the inherently multi-agent setting of RoboSoccer, specifically demonstrating how parameter and memory-sharing architectures improve learning. And since no multi-agent driving simulator is available freely for research purposes, we develop our own, codenamed MADRaS. Chapter 4 provides details on both these aspects.

3. **Curriculum Learning in RoboSoccer** - Finally, inspired from the education system in place worldwide, we experiment with breaking down the sparse-reward task of RoboSoccer into a curriculum of subtasks, learning which would 'help' with the harder target task. Tasks are presented to the agent using specialized embeddings. We believe that such an approach is important for any such humonguously complicated and intricate tasks. More details are presented in Chapter 5.

1.2 Overview

The remainder of this thesis is organized as follows :

- Chapter 2 presents the background necessary for understanding the research contributions of this thesis. In specific, a terse introduction to deep reinforcement learning and risk-sensitivity is presented, followed by specific algorithms like parameterized DDPG and GAIL, which form the basis of the learning agents in later chapters. It also introduces HFO, the simulator used for experiments in Chapters 4 and 5.[2]

[2]The choice of HFO RoboSoccer as a simulator was mainly governed by the fact that it is one of the most popular and well-maintained multi-agent environments openly available. Also, watching and playing football (yes, 'football') happen to be two of my favourite hobbies.

- Chapter 3 presents the first technical contribution, which is a framework for making imitation learning algorithms more reliable for deployment in risk-sensitive applications. It outlines the reliability issues with the existing state-of-the-art, and demonstrates how those are addressed.

- Chapter 4 presents the motivation and need for using multi-agent RL, and presents experimental results of a set of agents sharing knowledge while trying to score goals against a goalkeeper. It also details the development and features of MADRaS, which is the first open-source fully-controllable multi-agent driving simulator, to the best of our knowledge.

- Chapter 5 demonstrates the efficacy of a curriculum learning approach for decomposing complex tasks with sparse rewards into sequences of easier tasks. It further underlines the importance of each component - the type of embedding, its size, as well as the ordering in which the sub-tasks are presented to the learning agents.

CHAPTER 2

Background

Let us consider a Markov Decision Process (MDP), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. A reinforcement learning (RL) agent interacts with an environment over time. At each time step t, the agent receives a state s_t in a state space $\mathcal{S}$ and selects an action a_t from an action space $\mathcal{A}$, following a policy $\pi(a_t|s_t)$, which is the agent's behavior, i.e., a mapping from state s_t to actions a_t, receives a scalar reward r_t, and transitions to the next state s_{t+1}, according to the environment dynamics, or model, for reward function $\mathcal{R}(s, a)$ and state transition probability $\mathcal{P}(s_{t+1}|s_t, a_t)$ respectively. In an episodic problem, this process continues until the agent reaches a terminal state and then it restarts. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the discounted, accumulated reward with the discount factor $\gamma \in (0, 1]$. The objective of the agent is to maximize the expectation of the return from each state.

Reinforcement learning finds a policy which decides which is the best action to take in every possible state of the environment, with the objective of maximizing the agent's accumulated long term reward. The agent learns *tabula rasa*, which means it has no background about the dynamics of the environment or the structure of the reward, which makes the problem even more challenging. Q-learning is one of the the most popular learning algorithms applied in this context (Watkins, 1989).

2.1 Deep Reinforcement Learning

Reinforcement Learning becomes Deep Reinforcement Learning (DRL) when deep neural networks are used to approximate any of the following components of RL : the value function $v(s|\theta)$ or $q(s, a|\theta)$, the policy $\pi(a|s; \theta)$, or the model (state transition function and reward function), where θ represents the parameterization using deep neural networks. If tile coding, decision trees, and so on are used as the function approximators, what we have is “shallow" RL. So the choice of function approximator

is the most prominent difference between “shallow" and “deep" RL. When DRL ingredients like stochastic gradient descent (SGD) and off-policy function approximation are put together, that is a recipe for instability and divergence (Tsitsiklis and Van Roy, 1997). It was only recently that Google DeepMind came up techniques to get DRL to work using Deep Q-Networks (Mnih *et al.*, 2015) to not only stabilize the learning, but also achieve outstanding results!

2.2 Deep Q-Networks

Mnih *et al.* (2015) kickstarted the Deep Reinforcement Learning gold-rush by introducing Deep Q-Networks to approximate these Q-value functions. Leveraging the powerful feature representation of convolutional neural networks (CNNs), the Deep Q-Networks obtained state-of-the-art results in the complex environments of ATARI games. What caught the world's fancy was that this agents achieved superhuman performance on a wide range of the ATARI games using only raw screen-pixels and the reward score.

2.3 DDPG

Several variants of DQN have been proposed - from Double Q-learning to get over the over-estimation bias of DQNs () to Deep Recurrent Q-Network (DRQNs) to address issues of limited memory and partial observability (). These networks work well in continuous state spaces but do not function in continuous action spaces because the output nodes of the network, while continuous, are trained to output Q-Value estimates rather than continuous actions.

Enter the Actor-Critic framework, which decouples the action selection from the value estimation (). Represented using two deep neural networks, the actor network outputs continuous actions while the critic estimates the values of these actions. The actor network μ, parameterized by θ^{μ}, takes as input a state s and outputs a continuous action a. The critic network Q, parameterized by θ^{Q}, takes as input a state s and action a and outputs a scalar Q-Value $Q(s, a)$. The actor is trained to maximize the critic's estimated Q-values by back-propagating through both networks.

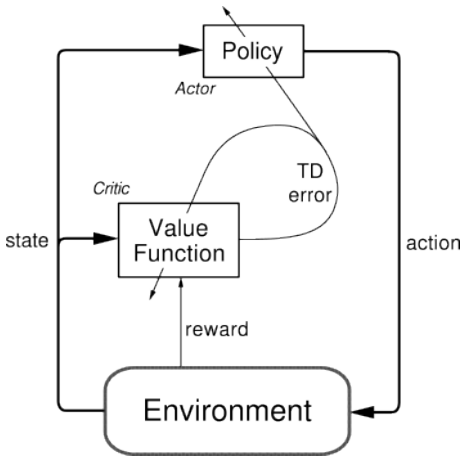


Figure 2.1: The actor-critic architecture (Sutton and Barto, 1998). In case of action-value functions, the activations to from the actor to the critic and gradients flow backwards from the critic to the actor. The gradients coming from the critic indicate directions of improvement in the continuous action space and are used to train the actor network without explicit targets.

The Actor-Critic framework actually builds on the REINFORCE algorithm (Williams, 1992) with baseline, which has yields the following update for a generic stochastic gradient ascent algorithm :

$$\theta_{t+1} = \theta_t + \alpha\big(G_t - b(S_t)\big)\frac{\nabla\pi(A_t|S_t,\theta_t)}{\pi(A_t|S_t,\theta_t)} \tag{2.1}$$

where G_t is the return, $b(S_t)$ is any arbitrary baseline, and θ is the parameterization of the policy.

Using the state-value function for bootstrapping (updating the value estimate for a state from the estimated values of subsequent states) and not just as a baseline makes this an actor-critic method. Updates to the critic network are computed by minimizing a loss function that comes from the standard temporal difference update, and that to actor network computed using the REINFORCE with baseline update. W.r.t. the loss function L for the critic and performance function J for the actor, the gradients are given by :

$$\nabla_{\theta^Q} L(\theta^Q) = \big(r + \gamma \max_{a'} \hat{Q}(s', a'|\theta^Q) - \hat{Q}(s, a|\theta^Q)\big)\nabla\hat{Q}(s, a|\theta^Q) \tag{2.2}$$

$$\nabla_{\theta^\mu} J(\theta^\mu) = \big(r + \gamma \max_{a'} \hat{Q}(s', a'|\theta^Q) - \hat{Q}(s, a|\theta^Q)\big)\nabla \ln \pi(a|s, \theta^\mu) \tag{2.3}$$

Now, the next challenge faced by the RL community was that of figuring out how to deal with continuous action spaces, since most interesting problems in the real world, in robotic control, etc., fall into this category. So when extending actor-critic to continuous action spaces, since maximizing over next-state actions a' in continuous action spaces is intractable, we output the next-state action $a' = \mu(s'|\theta^\mu)$ from the actor network. Silver *et al.* (2014) came up with the Deterministic Policy Gradient theorem, according to which the gradient of the performance objective J of the deterministic policy μ in the continuous action space w.r.t. policy parameters θ_μ is given by :

$$\nabla_{\theta^\mu} J(\theta^\mu) = \mathbb{E}_\mu[\nabla_a Q(s, a|\theta^Q)|_{a=\mu(s|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)] \tag{2.4}$$

which simply comes from a chain-rule expansion of $\nabla_{\theta^\mu} Q(s, \mu(s|\theta^\mu)|\theta^Q)$.

Putting this together, the gradients for the critic and the actor in the Deep Deterministic Policy Gradient (DDPG) setting (Lillicrap *et al.*, 2016) are given by :

$$\nabla_{\theta^Q} L(\theta^Q) = \big(r + \gamma \hat{Q}(s', \mu(s'|\theta^\mu)|\theta^Q) - \hat{Q}(s, a|\theta^Q)\big) \nabla \hat{Q}(s, a|\theta^Q) \tag{2.5}$$

$$\nabla_{\theta^\mu} J(\theta^\mu) = \nabla_a \hat{Q}(s, a|\theta^Q)|_{a=\mu(s|\theta^\mu)} \nabla_{\theta^\mu} \mu(s|\theta^\mu) \tag{2.6}$$

2.4 GAIL

Let $\xi = (s_0, a_0, s_1, \ldots, s_{L_\xi})$ denote a trajectory of length L_ξ, obtained by following a policy π. The expectation of a function $f(\cdot, \cdot)$ defined on $\mathcal{S} \times \mathcal{A}$ with respect to a policy π is defined as follows:

$$\mathbb{E}_\pi[f(s, a)] \triangleq \mathbb{E}_{\xi \sim \pi} \left[\sum_{t=0}^{L_\xi - 1} \gamma^t f(s_t, a_t) \right] \tag{2.7}$$

Apprenticeship learning algorithms (Abbeel and Ng, 2004) first estimate the experts' reward function using IRL and then find the optimal policy for the recovered reward function using RL. Mathematically, this problem can be described as:

$$RL \circ IRL(\pi_E) = \underset{\pi \in \Pi}{argmin} \max_{c \in \mathcal{C}} \mathbb{E}_\pi[c(s, a)] - \mathbb{E}_{\pi_E}[c(s, a)] - H(\pi) \tag{2.8}$$

where, π_E denotes the expert-policy. $c(\cdot, \cdot)$ denotes the cost function. Π and $\mathcal{C}$ denote the hypothesis classes for policy and cost functions. $H(\pi)$ denotes entropy of policy π. The term $-H(\pi)$ provides causal-entropy regularization (Ziebart *et al.*, 2008) which helps in making the policy optimization algorithm unbiased to factors other than the expected reward.

Ho and Ermon (2016) proposed Generative Adversarial Imitation Learning (GAIL) which packs the two step process of $RL \circ IRL_{\psi}(\pi_E)$ into a single optimization problem with special considerations for scalability in large environments. The name is due to the fact that this objective function can be optimized using the Generative Adversarial Network (GAN) framework (Goodfellow *et al.*, 2014). The objective function of GAIL is as follows :

$$\underset{\pi \in \Pi}{argmin} \max_{\mathcal{D} \in (0,1)^{\mathcal{S} \times \mathcal{A}}} \mathbb{E}_{\pi}[log(\mathcal{D}(s,a))] + \mathbb{E}_{\pi_E}[log(1 - \mathcal{D}(s,a))] - H(\pi) \tag{2.9}$$

Here, the agent's policy, π, acts as a *generator* of state-action pairs. $\mathcal{D}$ is a discriminative binary classifier of the form $\mathcal{D} : \mathcal{S} \times \mathcal{A} \rightarrow (0, 1)$, known as *discriminator*, which given a state-action pair (s, a), predicts the likelihood of it being generated by the generator. A two-player adversarial game is started, wherein the generator tries to generate (s, a) pairs that closely match the expert, while the discriminator tries to correctly classify the (s, a) pairs of the expert and the agent. At convergence, the agent's actions resemble those of the expert in any given state.

The generator and the discriminator are assigned parameterized models π_θ and $\mathcal{D}_w$ respectively. The training algorithm alternates between a gradient ascent step with respect to the discriminator parameters, w, and a policy-gradient descent step with respect to the generator parameters, θ. Both the generator and the discriminator are modeled with multi-layer perceptrons (neural networks with fully-connected layers).

2.5 Risk-sensitivity

Conditional-Value-at-Risk ($CVaR$) (Rockafellar and Uryasev, 2000) is the most conservative measure of tail risk (Dalleh, 2011) and unlike other measures like Variance and Value at Risk (VaR), it can be applied when the distribution of returns is not normal. Let Z be a random variable. Let $\alpha \in [0, 1]$ denote a probability value.

Definition 1. The *Value-at-Risk* of Z with respect to confidence level α, denoted by $VaR_\alpha(Z)$, is defined as the minimum value $z \in \mathbb{R}$ such that with probability α, Z will not exceed z.

$$VaR_\alpha(Z) = \min(z \mid P(Z \leq z) \geq \alpha) \tag{2.10}$$

Definition 2. The *Conditional-Value-at-Risk* of Z with respect to confidence level α, denoted by $CVaR_\alpha(Z)$, is defined as the conditional expectation of losses above $VaR_\alpha(Z)$:

$$CVaR_\alpha(Z) = \mathbb{E}\left[Z \mid Z \geq VaR_\alpha(Z)\right] = \min_{\nu \in \mathbb{R}} H_\alpha(Z, \nu) \tag{2.11}$$

where $H_\alpha(Z, \nu)$ is given by:

$$H_\alpha(Z, \nu) \triangleq \{\nu + \frac{1}{1-\alpha}\mathbb{E}\left[(Z-\nu)^+\right]\};\ (x)^+ = \max(x, 0) \tag{2.12}$$

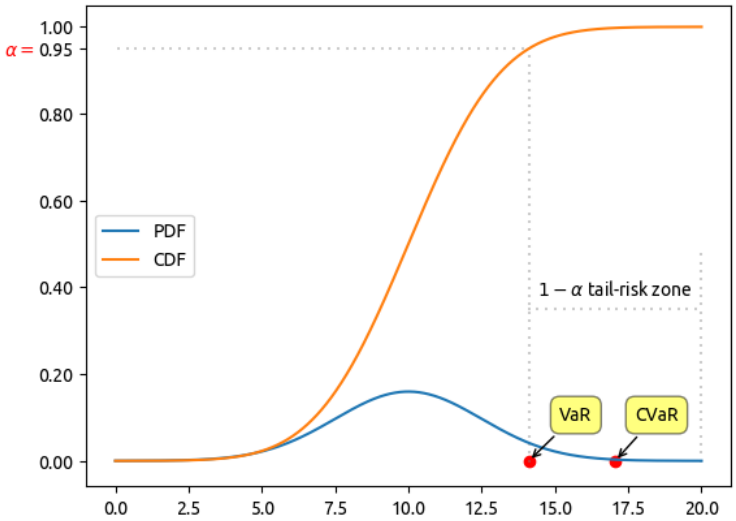


Figure 2.2: $VaR_{0.95}$ and $CVaR_{0.95}$ for a normal distribution.

2.6 Half Field Offense Simulator

The Half Field Offense (HFO) domain is subset of the full-fledged international robot soccer competition RoboCup, which works with an abstraction of soccer wherein the players, the ball, and the field are all 2-dimensional objects. HFO abstracts away the difficulties of full RoboCup and exposes the experimenter only to core decision-making logic, and to focus on the most challenging part of a RoboCup 2D game, which is scoring and defending goals (Hausknecht *et al.*, 2016).

2.6.1 Environment

HFO is naturally characterized as an episodic multi-agent POMDP because a) it has well-defined episodes which culminate in either a goal being scored or the ball leaving the play area, b) each agent receives its own state sensations and must independently select its own actions in order to achieve the cooperative and competing objective of scoring and defending goals, and c) and every agent receives only ego-centric features and not the complete game state as input. At the beginning of each episode, the agent and ball are positioned randomly on the offensive half of the field. The episode ends when a goal is scored, the ball leaves the field, or 500 timesteps pass.

2.6.2 State and Action spaces

The state space for each agent has 58 ego-centric features, Some of the most relevant ones are : the agent's position and velocity; distances to the ball, goal, other players; etc. A complete list of state features with additional details can be found in the manual at `https://github.com/mhauskn/HFO/blob/master/doc/manual.pdf`

HFO features a high-level parameterized action space :

1. `Dash`(power, direction): Moves in the indicated direction with the given scalar power. Movement is faster forward than sideways or backwards.

2. `Kick`(power, direction): Kicks the ball in the indicated direction with the given scalar power.

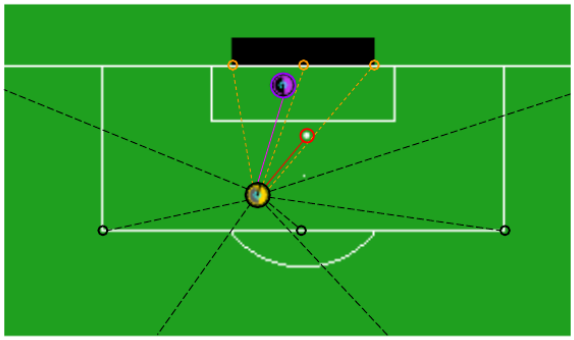


Figure 2.3: HFO's ego-centric state representation includes distances to various objects and points of interest on the field.

3. `Turn`(direction): Turns to indicated direction.

4. `Tackle`(direction): Contests the ball by moving in the indicated direction. This action is only useful when playing against an opponent.

wherein the range of the scalar power and direction is $[0, 100]$ and $[-180, 180]$ degrees respectively.

2.7 Parameterized DDPG

Hausknecht and Stone (2016) extend the the DPPG algorithm to parameterized, continuous, bounded action spaces by bounding the action space gradients. In specific, applying the notation in Masson *et al.* (2016) to parameterized action space as defined in the section above (2.6.2), we have $A = (\text{Dash}, p_1^{\text{dash}}, p_2^{\text{dash}}) \cup (\text{Turn}, p_3^{\text{turn}}) \cup (\text{Tackle}, p_4^{\text{tackle}}) \cup (\text{Kick}, p_5^{\text{kick}}, p_6^{\text{kick}})$. The actor network factors the action space into one output layer for discrete actions $(\text{Dash}, \text{Turn}, \text{Tackle}, \text{Kick})$ and another for all six continuous parameters $(p_1^{\text{dash}}, p_2^{\text{dash}}, p_3^{\text{turn}}, p_4^{\text{tackle}}, p_5^{\text{kick}}, p_6^{\text{kick}})$.

The deterministic action selection in this parameterized action space proceeds as follows: At each timestep, the actor network outputs values for all the ten parameters - the four discrete actions and the associated six continuous parameters. The discrete action is chosen to be the maximally valued output $a = \max(\text{Dash}, \text{Turn}, \text{Tackle}, \text{Kick})$ and paired with the associated parameters from the parameter output layer $(a, p_1^a, \ldots, p_{m_a}^a)$. In this way, the actor network chooses the action to be executed along with its associated parameters.

While training, without an indication as to which discrete action and the associated parameters were chosen, the critic network receives as input all the ten outputs of the actor network - all four discrete actions and all six action parameters. Similarly, the critic provides gradients for all the ten parameters to the actor for updating.

Finally, the exploration is performed using the standard ϵ-greedy method of picking between the four discrete actions. The associated parameters are sampled from uniform distributions of their ranges. Notably, the acclaimed Ornstein-Uhlenbeck exploration process (Lillicrap *et al.*, 2016) is not used.

With the above background, we are ready to present the our contributions in this thesis.

CHAPTER 3

Risk-Averse Imitation Learning

3.1 Motivation

The recently proposed Generative Adversarial Imitation Learning (GAIL) algorithm (Ho and Ermon (2016)) presents a novel mathematical framework in which the agent learns to act by directly extracting a policy from expert-demonstrated trajectories, as if it were obtained by RL following IRL. The authors show that unlike Behavioral Cloning, this method is not prone to the issue of compounding error and it is also scalable to large environments. Currently, GAIL provides state-of-the-art performance at several benchmark control tasks, including those in Table 3.1.

In order to evaluate the worst-case risk of deploying GAIL-learned policies, we studied the distributions (see Figure 3.1) of trajectory-costs (according to the expert's cost function) for the GAIL agents and experts at different control tasks (see Table 3.1).

We observed that the distributions for GAIL are more heavy-tailed than the expert, where the tail corresponds to occurrences of high trajectory-costs (refer to Figure 3.1). In order to quantify tail risk, we use Conditional-Value-at-Risk ($CVaR$) (Rockafellar and Uryasev, 2000) (refer to Section 2.5). $CVaR$ is defined as the expected cost above a given level of confidence and is a popular and coherent tail risk measure. The heavier the tail, the higher the value of $CVaR$. We observe that the value of $CVaR$ is much higher for GAIL than the experts at most of the tasks (see Table 3.1) which again suggests that the GAIL agents encounter high-cost trajectories more often than the experts. Since high trajectory-costs may correspond to events of catastrophic failure, GAIL agents are not reliable in risk-sensitive applications.

In this work, we aim to explicitly minimize expected worst-case risk for a given confidence bound (quantified by $CVaR$) along with the GAIL objective, such that the learned policies are more reliable than GAIL, when deployed, while still preserving the average performance of GAIL. Chow and Ghavamzadeh (2014) developed policy

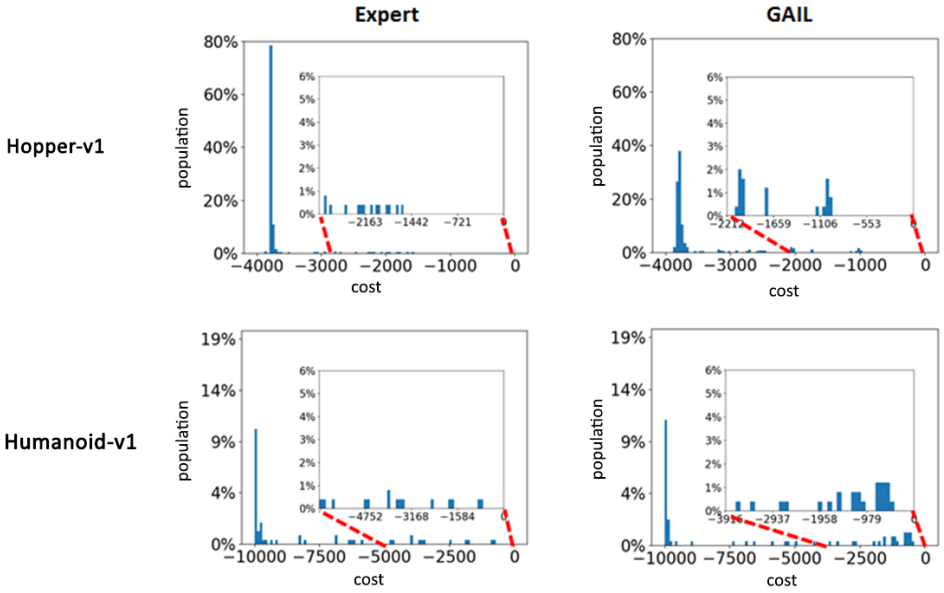


Figure 3.1: Histograms of the costs of 250 trajectories generated by the expert and GAIL agents at high-dimensional continuous control tasks, Hopper and Humanoid, from OpenAI Gym. The inset diagrams show zoomed-in views of the tails of these distributions (the region beyond 2σ of the mean). We observe that the GAIL agents produce tails heavier than the expert, indicating that GAIL is more prone to generating high-cost trajectories.

gradient and actor-critic algorithms for mean-$CVaR$ optimization for learning policies in the classic RL setting. However these algorithms are not directly applicable in our setting of learning a policy from a set of expert-demonstrated trajectories. We take inspiration from this work and make the following contributions:

1. We formulate the Risk-Averse Imitation Learning (RAIL) algorithm which optimizes $CVaR$ in addition to the original GAIL objective.

2. We evaluate RAIL at a number of benchmark control tasks and demonstrate that it obtains policies with lesser tail risk at test time than GAIL.

3.2 Related work

This section encapsulates the representative work in this space of imitation learning and risk-sensitivity.

3.2.1 Imitation Learning

Imitation learning or Learning from Demonstration (LfD) provides methods of learning policies through imitation of an expert's behavior without the need of a handcrafted cost function (Schaal, 1997). These algorithms fall into two broad categories:

1. Behavioral Cloning
2. Apprenticeship Learning
3. Model-free Imitation Learning

The first category, Behavioral Cloning (Pomerleau, 1989; Bojarski *et al.*, 2016), uses supervised learning to fit a policy function to the state-action pairs from expert-demonstrated trajectories. Despite its simplicity, Behavioral Cloning fails to work well when only a limited amount of data is available. These algorithms assume that observations are i.i.d. and learn to fit single time-step decisions. Whereas, in sequential decision making problems - where predicted actions affect the future observations (e.g. driving) - the i.i.d. assumption is violated. As a result, these algorithms suffer from the problem of compounding error due to covariate shift (Ross and Bagnell, 2010).

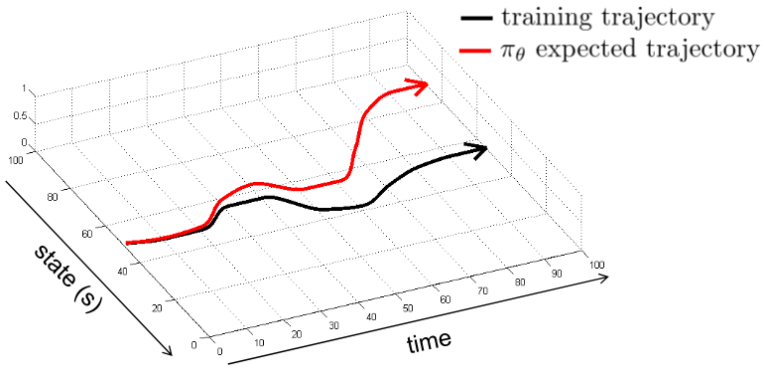


Figure 3.2: An illustration of the compounding error due to covariate shift (adapted from Sergey Levine's IL slides). As the learning agent deviates from the expected trajectory due to small errors, since it hasn't seen expert data in these erroneous zones, it makes even more errors, thereby compounding the deviation in trajectory.

Approaches to ameliorate the issue of compounding error like SMILe (Ross and Bagnell, 2010), SEARN (Daumé *et al.*, 2009) suffer from instability in practical applications (Ross *et al.*, 2011) while DAgger (Ross *et al.*, 2011) and AggreVaTe (Ross

and Bagnell, 2014) require the agent to query the expert during training which is not allowed in our setting of learning from a fixed set of expert demonstrations.

Another drawback of Behavioral Cloning is that it does not allow the agent to explore alternate policies for achieving the same objective that might be efficient in some sense other than what the expert cared for. An exemplar of this behaviorwherein an agent learns some unexpected policies are the 'alien-esque' gameplay shown by the AlphaZero agent (Silver *et al.*, 2017), whose novel moves (ones that humans have never played) have taken the chess community by awe and surprise (Knight, 2017).

The second category of algorithms is known as Apprenticeship Learning (Abbeel and Ng, 2004), which involves Inverse Reinforcement Learning followed by Reinforcement Learning. Inverse Reinforcement Learning (IRL) (Ng *et al.*, 2000; Abbeel and Ng, 2011) attempts to uncover the underlying reward function that the expert is trying to maximize from a set of expert-demonstrated trajectories. This reward function succinctly encodes the expert's behavior and are then used by an agent to learn a policy through an RL algorithm. IRL algorithms find reward functions that prioritize entire trajectories over others. Unlike behavioral cloning, they do not fit single time-step decisions, and hence they do not suffer from the issue of compounding error. However, Apprenticeship Learning algorithms are indirect because they first learn a reward function that explains expert behavior but do not tell the learner how to act directly. The job of learning an actionable policy is left to RL algorithms. Moreover, IRL algorithms are computationally expensive and have scalability issues in large environments (Levine and Koltun, 2012).

The recently proposed Generative Adversarial Imitation Learning (GAIL) algorithm (Ho and Ermon, 2016) presents a novel mathematical framework in which the agent learns to act by directly extracting a policy from expert-demonstrated trajectories, as if it were obtained by RL following IRL. Hence, this is a model-free imitation learning algorithm which harnesses generative adversarial training to fit distributions of states and actions defining expert behavior. The authors show that unlike Behavioral Cloning, this method is not prone to the issue of compounding error and it is also scalable to large environments. Currently, GAIL provides state-of-the-art performance at several benchmark control tasks, including complex, high-dimensional physics-based control tasks over various amounts of expert data.

3.2.2 Risk-Sensitivity

Risk sensitivity is integral to human learning (Nagengast *et al.*, 2010), and risk-sensitive decision-making problems, in the context of MDPs, have been investigated in various fields, e.g., in machine learning (Heger, 1994) and robotics (Shalev-Shwartz *et al.*, 2016; Rajeswaran *et al.*, 2016). Garcıa and Fernández (2015) give a comprehensive overview of different risk-sensitive RL algorithms. They fall in two broad categories. The first category includes methods that constrain the agent to safe states during exploration while the second modifies the optimality criterion of the agent to embed a term for minimizing risk. Studies on risk-minimization are rather scarce in the imitation learning literature. Much of the literature on imitation learning has been developed with average-case performance at the center, overlooking tail-end events.

In the portfolio-risk optimization literature, tail risk is a form of portfolio risk that arises when the possibility that an investment moving more than three standard deviations away from the mean is greater than what is shown by a normal distribution (Investopedia (2017)). Tail risk corresponds to events that have a small probability of occurring. When the distribution of market returns is heavy-tailed, tail risk is high because there is a probability, which may be small, that an investment will move beyond three standard deviations.

Conditional-Value-at-Risk ($CVaR$) (Rockafellar and Uryasev (2000)) is the most conservative measure of tail risk (Dalleh (2011)) and unlike other measures like Variance and Value at Risk (VaR), it can be applied when the distribution of returns is not normal.

3.3 Methodology

We use $CVaR$ to quantify the tail risk of the trajectory-cost variable $\mathcal{R}^\pi(\xi|c(\mathcal{D}))$, defined in the context of GAIL as:

$$\mathcal{R}^\pi(\xi|c(\mathcal{D})) = \sum_{t=0}^{L_\xi - 1} \gamma^t c(\mathcal{D}(s_t, a_t)) \tag{3.1}$$

where $c(\cdot)$ is order-preserving.

Next, we formulate the optimization problem to optimize $CVaR$ of $\mathcal{R}^\pi(\xi|c(\mathcal{D}))$ as:

$$\min_{\pi} \max_{c} \; CVaR_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D}))) = \min_{\pi,\nu} \max_{c} \; H_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu) \tag{3.2}$$

Integrating this with the GAIL objective of equation 2.9, we have the following:

$$\begin{aligned} \min_{\pi,\nu} \max_{\mathcal{D}\in(0,1)^{\mathcal{S}\times\mathcal{A}}} \mathcal{J} = \min_{\pi,\nu} \max_{\mathcal{D}\in(0,1)^{\mathcal{S}\times\mathcal{A}}} \Big\{ & - H(\pi) + \mathbb{E}_\pi[\log(\mathcal{D}(s,a))] \\ & + \mathbb{E}_{\pi_E}[\log(1-\mathcal{D}(s,a))] \\ & + \lambda_{CVaR}\, H_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu) \Big\} \end{aligned} \tag{3.3}$$

Note that as $c(\cdot)$ is order-preserving, the maximization with respect to c in equation 3.2 is equivalent to maximization with respect to $\mathcal{D}$ in equation 3.3. λ_{CVaR} is a constant that controls the amount of weightage given to $CVaR$ optimization relative to the original GAIL objective. Equation 3.3 comprises the objective function of the proposed Risk-Averse Imitation Learning (RAIL) algorithm. Algorithm 1 gives the pseudo-code. Please refer to Section 3.3.1 for the expressions of gradients of the $CVaR$ term, $H_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu)$ with respect to π, $\mathcal{D}$ and ν and their derivations. When $\alpha \to 0$, namely the risk-neutral case, $CVaR_0$, by definition, is equal to the mean of all trajectory costs and hence, RAIL $\to$ GAIL. We use Adam algorithm (Kingma and Ba (2015)) for gradient ascent in the discriminator and Trust Region Policy Optimization (TRPO) (Schulman *et al.* (2015)) for policy gradient descent in the generator. The $CVaR$ term α is trained by batch gradient descent (Haykin (1998)).

3.3.1 Derivation of gradients

In this section we derive expressions of gradients of the $CVaR$ term in equation 3.3 w.r.t. $\mathcal{D}$, π, and ν, inspired by those shown by Chow and Ghavamzadeh (2014). Let us denote $H_\alpha(D^\pi(\xi|c(\mathcal{D})), \nu)$ by $\mathcal{L}_{CVaR}$.

Algorithm 1 Risk-Averse Imitation learning (RAIL)

Input: Expert trajectories $\xi_E \sim \pi_E$, hyper-parameters α, β, λ_{CVaR}
Output: Optimized learner's policy π

1: Initialization: $\theta \leftarrow \theta_0$, $w \leftarrow w_0$, $\nu \leftarrow \nu_0$, $\lambda \leftarrow \lambda_{CVaR}$
2: **repeat**
3: Sample trajectories $\xi_i \sim \pi_{\theta_i}$
4: Estimate $\hat{H}_\alpha(D^\pi(\xi|c(\mathcal{D})), \nu)$
$= \nu + \frac{1}{1-\alpha}\mathbb{E}_{\xi_i}[(D^\pi(\xi|c(\mathcal{D})) - \nu)^+]$
5: Gradient ascent on discriminator parameters using:
$\nabla_{w_i}\mathcal{J} = \hat{\mathbb{E}}_{\xi_i}[\nabla_{w_i}\log(\mathcal{D}(s,a))]$
$+ \hat{\mathbb{E}}_{\xi_E}[\nabla_{w_i}\log(1-\mathcal{D}(s,a))]$
$+ \lambda_{CVaR}\nabla_{w_i}H_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu)$
6: KL-constrained natural gradient descent step (TRPO) on policy parameters using:
$\nabla_{\theta_i}\mathcal{J} = \mathbb{E}_{(s,a)\sim\xi_i}[\nabla_{\theta_i} log(\pi_\theta(a|s)Q(s,a)]$
$- \nabla_{\theta_i}H(\pi_\theta)$
$+\lambda_{CVaR}\nabla_{\theta_i}H_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu)$
where $Q(\bar{s},\bar{a}) =$
$\mathbb{E}_{(s,a)\sim\xi_i}[log(\mathcal{D}_{w_{i+1}}(s,a))|s_0 = \bar{s}, a_0 = \bar{a}]$
7: Gradient descent on CVaR parameters:
$\nabla_{\nu_i}\mathcal{J} = \nabla_{\nu_i}H_\alpha(\mathcal{R}^\pi(\xi|c(\mathcal{D})), \nu)$
8: **until** $i ==$ max_iter

3.3.1.1 Gradient of $\mathcal{L}_{CVaR}$ w.r.t. $\mathcal{D}$:

$$\begin{aligned}\nabla_{\mathcal{D}}\,\mathcal{L}_{CVaR} &= \nabla_{\mathcal{D}}\left[\nu + \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[(D^\pi(\xi|c(\mathcal{D})) - \nu)^+\right]\right] \\ &= \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[\nabla_{\mathcal{D}}\,D^\pi(\xi|c(\mathcal{D}))\mathbf{1}(D^\pi(\xi|c(\mathcal{D})) \geq \nu)\right]\end{aligned} \tag{3.4}$$

where $\mathbf{1}(\cdot)$ denotes the *indicator function*.

Now,

$$\nabla_{\mathcal{D}}\,D^\pi(\xi|c(\mathcal{D})) = \nabla_c\,D^\pi(\xi|c(\mathcal{D}))\,\nabla_{\mathcal{D}}\,c(\mathcal{D}) \tag{3.5}$$

$$\begin{aligned}\nabla_c\, D^{\pi}(\xi|c(\mathcal{D})) &= \nabla_c \sum_{t=0}^{L_\xi - 1} \gamma^t c(s_t, a_t) \\ &= \sum_{t=0}^{L_\xi - 1} \gamma^t \\ &= \frac{1-\gamma^{L_\xi}}{1-\gamma}\end{aligned} \tag{3.6}$$

Substituting equation 3.6 in 3.5 and then 3.5 in 3.4, we have the following:

$$\nabla_{\mathcal{D}} \mathcal{L}_{CVaR} = \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[\frac{1-\gamma^{L_\xi}}{1-\gamma}\,\mathbf{1}(D^{\pi}(\xi|c(\mathcal{D})) \geq \nu)\nabla_{\mathcal{D}}\, c(\mathcal{D})\right] \tag{3.7}$$

3.3.1.2 Gradient of $\mathcal{L}_{CVaR}$ w.r.t. π

$$\begin{aligned}\nabla_\pi\, \mathcal{L}_{CVaR} &= \nabla_\pi\, H_\alpha(D^{\pi}(\xi|c(\mathcal{D})), \nu) \\ &= \nabla_\pi \left[\nu + \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[(D^{\pi}(\xi|c(\mathcal{D})) - \nu)^+\right]\right] \\ &= \frac{1}{1-\alpha}\nabla_\pi\, \mathbb{E}_{\xi\sim\pi}\left[(D^{\pi}(\xi|c(\mathcal{D})) - \nu)^+\right] \\ &= \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[(\nabla_\pi\, \log P(\xi|\pi))(D^{\pi}(\xi|c(\mathcal{D})) - \nu)^+\right]\end{aligned} \tag{3.8}$$

3.3.1.3 Gradient of $\mathcal{L}_{CVaR}$ w.r.t. ν

$$\begin{aligned}\nabla_\nu\, \mathcal{L}_{CVaR} &= \nabla_\nu \left[\nu + \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[(D^{\pi}(\xi|c(\mathcal{D})) - \nu)^+\right]\right] \\ &= 1 + \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[\nabla_\nu\, (D^{\pi}(\xi|c(\mathcal{D})) - \nu)^+\right] \\ &= 1 - \frac{1}{1-\alpha}\mathbb{E}_{\xi\sim\pi}\left[\mathbf{1}(D^{\pi}(\xi|c(\mathcal{D})) \geq \nu)\right]\end{aligned} \tag{3.9}$$

3.4 Experiments

In this section, we provide details on the experiments performed - namely the environments tested on, the model architecture, the hyperparameters used.

We compare the tail risk of policies learned by GAIL and RAIL for five continuous control tasks listed in Table 3.1. All these environments were simulated using MuJoCo Physics Simulator (Todorov *et al.*, 2012). Each of these environments come packed with a "true" reward function in OpenAI Gym (Brockman *et al.*, 2016). Ho and Ermon (2016) trained neural network policies using Trust Region Policy Optimization (TRPO) (Schulman *et al.*, 2015) on these reward functions to achieve state-of-the-art performance and have made the pre-trained models publicly available for all these environments as a part of their repository[1]. They used these policies to generate the expert trajectories in their work on GAIL. For a fair comparison, we use the same policies to generate expert trajectories in our experiments. Table 3.1 gives the number of expert trajectories sampled for each environment. These numbers correspond to the best results reported in Ho and Ermon (2016).

Table 3.1: Hyperparameters for the RAIL experiments on various continuous control tasks from OpenAI Gym.

Task	**#training iterations**	**#expert trajectories**	λ_{CVaR}
Reacher	200	18	0.25
HalfCheetah	500	25	0.5
Hopper	500	25	0.5
Walker	500	25	0.25
Humanoid	1500	240	0.75

Again, following Ho and Ermon (2016), we model the generator (policy), discriminator and value function (used for advantage estimation (Sutton and Barto, 1998) for the generator) with multi-layer perceptrons of the following architecture:

```
obsDim - fc_100 - tanh - fc_100 - tanh - outDim
```

where `fc_100` means fully connected layer with 100 nodes, `tanh` represents the hyperbolic-tangent activation function of the hidden layers, `obsDim` stands for the dimensionality of the observed feature space, `outDim` is equal to 1 for the discriminator

[1] https://github.com/openai/imitation

and value function networks and equal to the twice of the dimensionality of the action space (for mean and standard deviation of the Gaussian from which the action should be sampled) for the policy network. For example, in case of Humanoid, $\texttt{obsDim} = 376$ and $\texttt{outDim} = 34$ in the policy network. The value of the $CVaR$ coefficient λ_{CVaR} is set as given by Table 3.1 after a coarse hyperparameter search. All other hyperparameters corresponding to the GAIL component of the algorithm are set identical to those used by Ho and Ermon (2016) in their repository for all the experiments. The value of α in the $CVaR$ term is set to 0.9 and its lone parameter, ν, is trained by batch gradient descent with learning rate 0.01. We set $\alpha = 0.9$ for VaR_α and $CVaR_\alpha$ and estimate all metrics with $N = 50$ sampled trajectories (as followed by Ho and Ermon (2016)).

3.5 Results

Table 3.2: Comparison of expert, GAIL, and RAIL in terms of the tail risk metrics - $VaR_{0.9}$ and $CVaR_{0.9}$. All the scores are calculated on samples of 50 trajectories. With smaller values of VaR and $CVaR$, RAIL outperforms GAIL in all the 5 continuous control tasks and also outperforms the expert in some cases.

Environment	**Dimensionality**		**VaR**			**CVaR**		
	Obs	**Action**	**Expert**	**GAIL**	**RAIL**	**Expert**	**GAIL**	**RAIL**
Reacher	11	2	5.88	9.55	7.28	6.34	13.25	9.41
Hopper	11	3	-3754	-1758	-3745	-2674	-1347	-3727
HalfCheetah	17	6	-3431	-2688	-3150	-3356	-2220	-2945
Walker	17	6	-5402	-5314	-5404	-2310	-3359	-3939
Humanoid	376	17	-9839	-2641	-9252	-4591	-1298	-4640

Now we define the metrics we use to evaluate the efficacy of RAIL at reducing the tail risk of GAIL learned policies. Given an agent A's policy π_A we roll out N trajectories $T = \{\xi_i\}_{i=1}^N$ from it and estimate VaR_α and $CVaR_\alpha$ as defined in Section 2.5. VaR_α denotes the value under which the trajectory-cost remains with probability α and $CVaR_\alpha$ gives the expected value of cost above VaR_α. Intuitively, $CVaR_\alpha$ gives the average value of cost of the worst cases that have a total probability no more than $(1 - \alpha)$. The lower the value of both these metrics, the lower is the tail risk.

In order to compare tail risk of an agent with respect to the expert, E, we define percentage relative-VaR_α as follows:

$$VaR_\alpha(A|E) = 100 \times \frac{VaR_\alpha(E) - VaR_\alpha(A)}{|VaR_\alpha(E)|}\% \tag{3.10}$$

Similarly, we define percentage relative-$CVaR_\alpha$ as:

$$CVaR_\alpha(A|E) = 100 \times \frac{CVaR_\alpha(E) - CVaR_\alpha(A)}{|CVaR_\alpha(E)|}\% \tag{3.11}$$

The higher these numbers, the lesser is the tail risk of agent A. We define Gain in Reliability (GR) as the difference in percentage relative tail risk between RAIL and GAIL agents.

$$GR\text{-}VaR_\alpha = VaR_\alpha(RAIL|E) - VaR_\alpha(GAIL|E) \tag{3.12}$$

$$GR\text{-}CVaR_\alpha = CVaR_\alpha(RAIL|E) - CVaR_\alpha(GAIL|E) \tag{3.13}$$

Table 3.3: Values of percentage relative tail risk measures and gains in reliability on using RAIL over GAIL for the different continuous control tasks. RAIL shows a remarkable improvement over GAIL in both the metrics.

Environment	$VaR_{0.9}(A\|E)$ **(%)**		$GR\text{-}VaR_\alpha$ (%)	$CVaR_{0.9}(A\|E)$ **(%)**		$GR\text{-}CVaR_\alpha$ (%)
	GAIL	**RAIL**		**GAIL**	**RAIL**	
Reacher	-62.41	-23.81	**38.61**	-108.99	-48.42	**60.57**
Hopper	-53.17	-0.23	**52.94**	-49.62	39.38	**89.00**
HalfCheetah	-21.66	-8.20	**13.46**	-33.84	-12.24	**21.60**
Walker	-1.64	0.03	**1.66**	45.39	70.52	**25.13**
Humanoid	-73.16	-5.97	**67.19**	-71.71	1.07	**72.78**

3.6 Discussion

In this section, we discuss the results of comparison between GAIL and RAIL. The expert's performance is used as a benchmark. Tables 3.2 and 3.3 present the values of our evaluation metrics for different continuous-control tasks. The following are some interesting observations that we make:

- RAIL obtains superior performance than GAIL at both tail risk measures – $VaR_{0.9}$ and $CVaR_{0.9}$, across a wide range of continuous-control tasks, without increas-

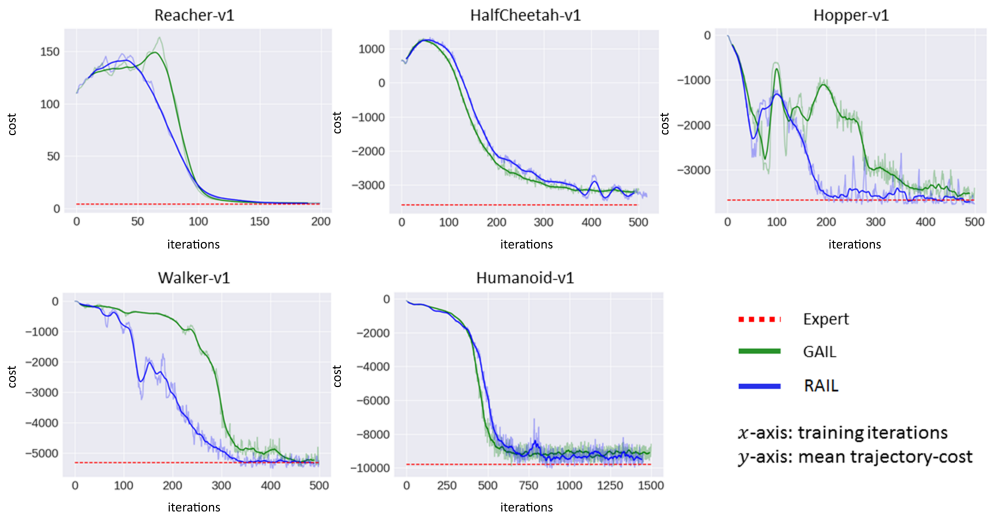


Figure 3.3: Convergence of mean trajectory-cost during training. The faded curves corresponds to the original value of mean trajectory-cost which varies highly between successive iterations. The data is smoothened with a moving average filter of window size 21 to demonstrate the prevalent behavior and plotted with solid curves. RAIL converges almost as fast as GAIL at all the five continuous-control tasks, and at times, even faster.

ing sample complexity. This shows that RAIL is a superior choice than GAIL for imitation learning in risk-sensitive applications.

- The applicability of RAIL is not limited to environments in which the distribution of trajectory-cost is heavy-tailed for GAIL. Rockafellar and Uryasev (2000) showed that if the distribution of the risk variable Z be normal, $CVaR_\alpha(Z) = \mu_Z + a(\alpha)\sigma_Z$, where $a(\alpha)$ is a constant for a given α, μ_Z and σ_Z are the mean and standard deviation of Z. Thus, in the absence of a heavy tail, minimization of $CVaR_\alpha$ of the trajectory cost aids in learning better policies by contributing to the minimization of the mean and standard deviation of trajectory cost. The results on Reacher-v1 corroborate our claims. Although the histogram does not show a heavy tail (Figure 3.4), the mean converges fine (Figure 3.3) and tail risk scores are improved (Table 3.2) which in this case indicates the distribution of trajectory-costs is more condensed around the mean than GAIL. Thus we can use RAIL instead of GAIL, no matter whether the distribution of trajectory costs is heavy-tailed for GAIL or not.

- Figure 3.3 shows the variation of mean trajectory cost over training iterations for GAIL and RAIL. We observe that RAIL converges almost as fast as GAIL at all the continuous-control tasks in discussion, and at times, even faster.

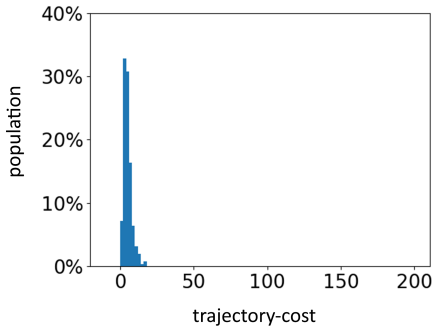


Figure 3.4: Histogram of costs of 250 trajectories generated by a GAIL-learned policy for Reacher-v1. The distribution shows no heavy tail. From Table 3.2 and Figure 3.3, we observe that RAIL performs as well as GAIL even in cases where the distribution of trajectory costs is not heavy-tailed.

- The success of RAIL in learning a viable policy for Humanoid-v1 suggests that RAIL is scalable to large environments. Scalability is one of the salient features of GAIL. RAIL preserves the scalability of GAIL while showing lower tail risk.

3.7 Conclusion and Future Work

RAIL agents show lesser tail risk than GAIL agents after training has been completed. However it still requires the agent to act in the real world and sample trajectories (line 3 in Algorithm 1) during training. One way to rule out environmental interaction during training is to make the agent act in a simulator while learning from the expert's real-world demonstrations. The setting changes to that of third person imitation learning (Stadie *et al.* (2017)). The RAIL framework can be ported to this formulation, which is left as future work. Depending on the environmental constraints, one could also adopt the active learning framework (Ross *et al.*, 2011). Furthermore, more work is required for theoretically justifying the reliability of the RAIL framework.

In conclusion, this work presents the RAIL algorithm, which incorporates $CVaR$ optimization within the original GAIL algorithm to minimize tail risk and thus improve reliability of learned policies. We report significant improvement over GAIL at a number of evaluation metrics on five continuous-control tasks. Thus the proposed algorithm is a viable step in the direction of learning low-risk policies by imitation learning in complex environments, especially in risk-sensitive applications like robotic surgery and autonomous driving. We plan to test RAIL on fielded robotic applications in the future.

CHAPTER 4

Multi-Agent Learning

4.1 Motivation

In the real world, learning often happens in groups rather individually, in silos. For instance, nearly all animals from the time of infancy learn from their parents or other members of the species. Classical learning methods, inherently single-agent, are hence handicapped since they act alone and can only use their own experiences with the environment as guidance. We believe that multiple learning agents working together can accomplish far more than a single agent ever could, that too more efficiently.

Multi-agent reinforcement learning (MARL) is concerned with a set of autonomous agents that share a common environment. Reasoning about other agents' intentions and being able to predict their behavior is important in multi-agent systems, in which the agents might have a diverse, and sometimes competing, set of goals. This remains a challenging problem due to the inherent non-stationarity of such domains. Over and above the problems faced in single-agent learning (e.g. exploration-exploitation dilemma, temporal credit-assignment problem, etc), Busoniu *et al.* (2008) outline the following critical issues faced in MARL :

1. Non-stationarity - MARL presents a moving-target problem since the best policy changes as the other agents' policies change.

2. Curse of dimensionality - The growth of state and action variables with the number of agents is exponential.

3. Specifying a good goal - Since the agents' returns are correlated and cannot be maximized independently, this is hard.

4. Exploration - Apart from having to explore the environment, the agents also have to obtain information about other agents.

5. Coordination - The effect of an agent's action on the environment also depends on the actions taken by other agents. So a) the first agent doesn't know for sure how much to credit its own action for the obtained reward, and b) there arises a need of mutually consistent actions in order to achieve the intended effect.

Most practical real world applications are inherently multi-agent in nature, and require strategic interactions among a large number of entities. Ranging from trading agents in stock markets and online advertising bidding to gaming bots in Massively Multiplayer Online Role-Playing Games (MMORPGs, like StarCraft and DoTA), and more recently, autonomous vehicles.

4.2 Related work

4.2.1 Classical approaches

The classical MARL survey paper (Busoniu *et al.*, 2008) offers an excellent birds-eye view of classical Multi-Agent RL, organizing it into an insightful taxonomy. As mentioned above, it also outlines the challenges in MARL as compared to the single-agent case. Learning in MARL is fundamentally difficult since agents not only interact with the environment but also with each other.

The simplest option is to let each agent a learn an individual action-value function Q_a independently, as in Independent Q-learning (Tan, 1993), which considers other agents as a part of the environment. This ostrich-esque approach of modeling everything as multiple single-agents often fails as the multi-agent environment breaks the theoretical convergence guarantees and makes the learning unstable: the changes in strategy of one agent would affect the strategies of other agents and vice-versa (Matignon *et al.*, 2012). In simple words, each agent's learning is confounded by the learning and exploration of others.

With multiple agents learning simultaneously in long, episodic tasks, it makes sense for them to be able to learn and bootstrap from each other's experience. Using classical RL techniques of SARSA with the CMAC-tiling for function approximation, (Kalyanakrishnan *et al.*, 2006) uses explicit communication via a central controller for

sharing their individual network's parameters. Even with tiny errors in communication, all the agents learn the same policy in expectation. Similarly, (Duan *et al.*, 2012) uses communication to create a joint-state, joint-action space and use Nash Q-learning for the agents.

4.2.2 Recent (and deep) approaches

Moving on to the deep RL setting, (Tampuu *et al.*, 2017) extend the classic DQN to a setting in which 2 agents can receive the game inputs and rewards simultaneously. By modifying the reward scheme for the classic Pong game in ATARI, they analyze the behaviour of the agents under a continuum of cooperative and competitive setting.

The novel paradigm of centralized training with decentralized execution (Kraemer and Banerjee, 2016) has recently attracted a lot of attention in the RL community (Lowe *et al.*, 2017; Foerster *et al.*, 2018). However, many challenges surrounding how to best exploit centralized training remain open. In Foerster *et al.* (2018), the authors proposed a counterfactual multi-agent policy gradient method that uses a centralized advantage to estimate whether the action of one agent would improve the global reward, and decentralized actors to optimize the agent policy.

Lowe *et al.* (2017) also utilize the framework of decentralized execution and centralized training to develop multi-agent multi-agent actor-critic algorithm that can coordinate agents in mixed cooperative-competitive environments. Thhey propose a multi-agent extension to the DDPG algorithm (Lillicrap *et al.*, 2016), christened MADDPG, using a set of actor and all-knowing critic models. The primary motivation behind MADDPG is that, if we know the actions taken by all agents, the environment is stationary even as the policies change since $P(s'|s, a_1, \ldots, a_N, \pi_1, \ldots, \pi_N) = P(s'|s, a_1, \ldots, a_N) = P(s'|s, a_1, \ldots, a_N, \pi'_1, \ldots, \pi'_N)$ for any $\pi_i \neq \pi'_i$ (Lowe *et al.*, 2017). This is not the case if we do not explicitly condition on the actions of other agents, as done for most traditional RL methods. More recent approaches have tried modeling the other agents' actions using the ego-centric agent's own policy and updating their belief in the others' hidden states, in both cooperative and competitive settings (Raileanu *et al.*, 2018).

Some common paradigms that emerge from the literature are enumerated as follows :

Reward Sharing

Now, one of the main problems of multi-agent RL that distinguishes it from the classical single-agent is that multi-agent domains are non-stationary from agents' local perspectives, due to teammates' interactions with the environment. Moreover, exploratory actions by other agents could lead to low communal returns even though the current agent might have taken an optimal action. (Omidshafiei *et al.*, 2017) handles this issue by using Hysteretic Q-learning, which does not ignore low returns by assuming that they are caused by teammates' exploratory actions, but instead supposes that these could be a result of the environment stochasticity. Using a smaller learning rate when the TD-error is negative, this paper combines Hysteretic Q-learning with Deep Recurrent Q-Networks for these partially observable environments.

In the setting of RoboSoccer, if one agent scores the goal, should his teammates get a reward for it as well? If so, to what extent? In (Hausknecht, 2016), the authors try to hand-craft role-based reward functions for the agents with limited success because of the difficulty in finding the right weightages of the sub-rewards. On the other hand, (Liu *et al.*, 2012) encourages the learning of individual reward functions. Assuming perfect information, each agent learns other agents' stochastic policies using the empirical probabilities observed in historical data in order to sample their actions. In toy cooperative domains like 'food-shelter', this method results indeed results in specialization of tasks with the individual reward functions learnt.

Parameter and Memory Sharing

Parameter sharing can be done in the following ways:

1. Sharing all the weights - This will yield similar policies among the agents, subject to the reliability in the mode of communication of these parameters, as in (Kalyanakrishnan *et al.*, 2006).

2. Sharing weights partially - Sharing weight only amongst the lowest network layers makes sense, since they are responsible for basic processing of the features (Hausknecht, 2016). Actors will have similar policies, and critics will give similar directions.

Sharing of the network weights helps in reducing the number of unique parameters. (Hausknecht, 2016) also introduces sharing of the experience replay buffer, which indirectly encourages similar policies. The expected resultant behaviour is that the agents learn individual skills well.

Though MADDPG (Lowe *et al.*, 2017) obtains state-of-the-art results for some multi-agent games, it does not scale well with growing amount of agents. Recently, (Chu and Ye, 2017) has proposed a parameter sharing deterministic policy gradient method with 3 variants - shared actor-critic networks in case of common rewards, shared actor only when rewards are different, and partially-shared critic with shared lower-layers and separate heads for the value function. Thsi method reportedly gets better results than MADDPG in multi-agent versions of WaterWorld, Ant, and Humanoid environments.

Communication Strategies

Explicitly sharing network parameters may not be feasible over limited badnwidth channels. Some principled methods have been proposed. (Foerster *et al.*, 2016) introduces Reinforced Inter-Agent Learning (RIAL), wherein each agent uses independent reinforcement learning to determine the communicated messages. (Hausknecht, 2016) uses this with the communication 'actions' as output from the actor, which are concatenated into the state space that is input to the critic. Another strategy is to allow agents to influence the communication strategy of their teammates through shared gradients applied to the teammates' communication actions (Hausknecht, 2016). The Differentiable Inter-Agent Learning (DIAL) framework (Foerster *et al.*, 2016) has a feedback signal (in the form of gradients) from the other agents for the communication actions. This makes it end-to-end differentiable as compared to Reinforced Inter-Agent Learning (RIAL), which is only end-to-end trainable within each agent.

Recently, Hausknecht and Stone (2016) came up with an extension to the DDPG algorithm to account for stable learning in continuous, bounded action spaces by notably bounding action space gradient. The approach is described in detail in Section 2.7. We use this algorithm for our subsequent experiments in the RoboSoccer domain.

4.3 Methodology

Following Hausknecht and Stone (2016) and the discussion in Section 2.7, we use the following actor-critic architecture proposed for parameterized action spaces :

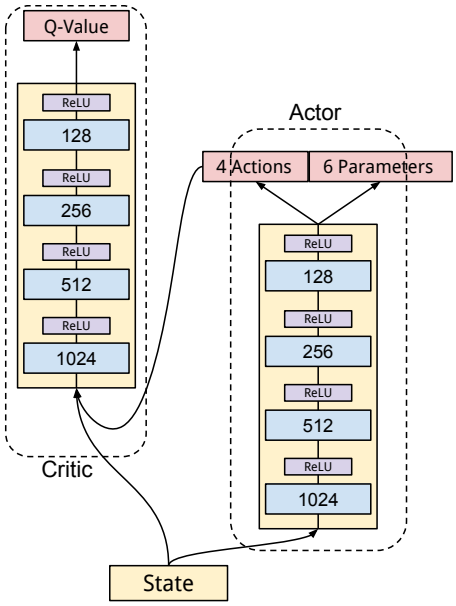


Figure 4.1: Model architecture (adapted from Hausknecht and Stone (2016)) : an actor-critic model wherein in every timestep, the actor takes the state features as input and outputs the four discrete actions and six associated parameters *in unison*, which the critic then takes as input along with the state features to compute Q-values of.

The hand-coded reward function being used is a linear combination of :

1. Move To Ball Reward - proportional to the change in distance between the agent and the ball $d(a, b)$
2. First Touch Reward - $\mathbb{I}^{kick}$ - a reward given when the agent reaches the ball for the first time in an episode
3. Kick To Goal Reward - proportional to the change in distance between the ball and the center of the goal $d(b, g)$
4. Score Reward - $\mathbb{I}^{goal}$ - a reward for shooting the ball into the goal (the ultimate task).

Mathematically, the reward at time t,

$$r_t = (d_{t-1}(a, b) - d_t(a, b)) + \mathbb{I}^{kick} + 3(d_{t-1}(b, g) - d_t(b, g)) + 5\ \mathbb{I}^{goal} \tag{4.1}$$

4.4 Experiments

Using the code accompanying Hausknecht and Stone's work on the parameterized version of the DDPG algorithm (Hausknecht and Stone, 2016) as the baseline, the following experiments were performed on the open-source[1] HFO domain Hausknecht *et al.* (2016). The actor-critic architecture modified for the parameterized action space was used for the following combination of scenarios as baselines:

- one or more agents
- independent and shared network (lower) layers
- independent and shared replay buffers
- with and without a goalkeeper
- an expert or a naive goalkeeper

Both the actor and critic use the same architecture as shown in Figure 4.1 The 58 state input features are processed by four fully connected layers consisting of $1024 - 512 - 256 - 128$ units respectively. Each fully connected (fc) layer is followed by a rectified linear (ReLU) activation function with slope -10^{-2}. Weights of the fc-layers use Gaussian initialization with $\sigma = 10^{-2}$. The linear output layer outputs a vector of size 10 : four discrete actions and the six accompanying parameters. The critic also takes as input this output vector from the actor and outputs a single scalar Q-value. We use the ADAM solver (Kingma and Ba, 2015) for gradient descent, with learning rates of both the actor and critic networks set to 10^{-3}. Finally, both the target networks follow the actor and critic networks with a tracking parameter of $\tau = 10^{-4}$.

4.5 Results and discussion

Some interesting observations:

- 2v0 (indp) - The 2 independent agents don't score as often as the single agent, but score faster when they do. This is probably because in expectation, one of the agents is initialized closer to the ball as compared to the 1v0 case. It performs

[1] https://github.com/LARG/HFO/

Table 4.1: Some interesting results showing the percentage of trials ending in a goal being scored along with the average number of frames taken per goal. The scenarios are described in the accompanying text.

Scenario	Trials	Goals		Iterations	AvgFrame/Goal
		#	%		
1v0	275896	234031	**84.83**	250000	**126.4**
2v0 (indp)	247900	178995	**72.20**	250000	**116.9**
2v0 (memory)	307341	232201	**75.55**	∼300000	**116.3**
2v0 (layers)	241160	183751	**76.19**	250000	**120**
1v1 (expert)	646046	392	**0.06**	∼650000	**136.8**
2v1 (ind)	300127	197	**0.07**	300000	**135.7**
2v1 (memory)	250000	72	**0.029**	250000	-
2v1 (memory, pass)	198039	68	**0.03**	300000	**220**
1v1 (goalie)	236821	116909	**49.37**	250000	**130**
1v1 (goalie; noFreeze)	227804	119070	**52.27**	250000	**127.6**

worse probably because the agents compete and push the ball out of bounds, or just don't have enough samples to learn from.

- 2v0 (memory) - Sharing of the replay memory does only marginally better than having 2 independent experience replays.

- 2v1 - Like 1v1, both the independent agents and those with a shared experience replay memory performed pathetically with the (champion) heuristic goalie in place. (For reference, 1v0 and 2v0 goal percentage is $\sim 80\%$)

- On visualizing the learnt policies, it is observed that the agents do not learn passing at all, and learn identical behavior of blindly chasing the ball, taking it near the goal, and shooting it, which the champion goalie easily saves. The obvious expert behavior in this case would be to pass it to a 'free' teammate, who will be in a better position to score (since the goalie comes close to the first agent who has the ball) i.e. learn a passing skill.

- 2v1 (memory, pass) - Even with the passing reward turned on, the goal conversion rate is abysmal, and the agents do not learn 'passing'. They do receive passing rewards, but that is mostly by accident - one agent kicks the ball and the next time the other does, while both follow the ball in the same line.

  The weightage to an explicit penalty for the distance between the agents (to make sure there is sufficient distance between them to consider passing) requires a lot of parameter tuning, which highlights the problem of reward-shaping.

- 1v1 (goalie) - Another run featured a naive goalie, who would follow the y-coordinates of the ball and freeze when the agent comes closer than a specified threshold. The objective was to have a curriculum wherein this naive goalie kicks in after 100000 episodes, after the agent would have learnt a decent policy of dribbling and shooting into the goal.

- 1v1 (goalie; noFreeze) - To check to what extent the freezing contributed to the good results, another experiment involved the goalie tracking the ball position even when the agent is very close. Surprisingly, this resulted in a better performance on both metrics!

- The visualization of the policy learnt revealed that the agent learns to run/dribble right into the goal, and not explicitly kick into the goal from somewhere near it. Further inspection revealed that is there is indeed nothing that incentivizes the agent to learn to kick. The authors (Hausknecht and Stone, 2016) probably use a penalty for the time taken as well, because their agent kicks the ball a short distance every time it touches it.

Before outlining some limitations and directions of future work, we take a brief digression from the RoboSoccer domain and explore the driving environment, which has been the driving force behing this entire effort.

4.6 Development - MADRaS

In this section, we describe the development of "**M**ulti-**A**gent **DR**iving **S**imulator" (MADRaS). It is a multi-agent version of TORCS (Wymann *et al.*, 2000), a racing simulator popularly used for autonomous driving research by the reinforcement learning and imitation learning communities (Loiacono *et al.*, 2010; Cardamone *et al.*, 2009; Mnih *et al.*, 2016; Lillicrap *et al.*, 2016).

4.6.1 Motivation

MADRaS is a multi-agent extension of Gym-TORCS [2] and is open source, lightweight, easy to install, and has the OpenAI Gym API, which makes it ideal for beginners in autonomous driving research. It enables independent control of tens of agents within the same environment, opening up a prolific direction of research in multi-agent reinforcement learning and imitation learning research aimed at acquiring human-like negotiation skills in complicated traffic situations - a major challenge in autonomous driving that all major players are racing to solve.

The main issues with most open-source autonomous driving simulators like CARLA (Dosovitskiy *et al.*, 2017), AirSim (Shah *et al.*, 2017), and DeepDrive (DeepDrive.io, 2018) are :

1. **Lack of multi-agent control** - To the best of our knowledge, these innately support only egocentric control; that is, single agent behavior, and have pre-programmed behaviors for the other agents. The task of negotiation in traffic can be posed as that of finding the winning strategy in a multi-agent game, and hence learning policies to control in a multi-agent setting is critical.

2. **Lack of customizability of non-ego-control cars** - The difficulty in introducing agents with custom behaviors in these simulators restricts the diversity of real-world scenarios that can be simulated. Since acting and learning in the real world is dangerous and prohibitively expensive, we need the ability to be able to conjure up all kinds of scenarios in a simulator, the lack of which would raise questions about the capability of these self-driving cars in handling 'unseen' scenarios.

The difficulty in introducing agents with custom behaviors, some of which could be learned (in possible collaboration) in these simulators restricts the diversity of real-world scenarios that can be simulated. To address this issues, we developed MADRaS, wherein each car on the racing track can be independently controlled, and the control policies of these multiple agents can be learned simultaneously. Since it is built on top of TORCS, the customized control of each of the cars on the road enables the creation of rich, custom-made traffic scenarios.

[2] https://github.com/ugo-nama-kun/gym_torcs

Here are some links of the project[3] :

- Blog - https://software.intel.com/en-us/articles/madras-a-multi-agent-driving-simulator
- Github repository - https://github.com/abhisheknaik96/MultiAgentTORCS
- Video teaser - https://www.youtube.com/watch?v=ZKzExvth3UE

Figure 4.2: MADRaS in action. The figure shows two controllable agents (blue-green) in the foreground, along with other custom traffic cars. The two terminal windows are logging the progress of both the agents simultaneously. The tracks and the extent and kinds of sensory information logged are customizable.

4.7 Future work

With a lot of evidence pointing towards the drawbacks of using hand-engineered reward functions and the additional complications posed by the non-stationarity of the multi-agent environment from the experiments performed in this chapter, the Curriculum Learning approach is considered (Chapter 5) in order to make more headways into the problem.

Even in the current multi-agent setup, we identify the following limitations and suggest some directions of future work :

- Explicit communication - The sharing of experiences and parameters across layers is only an indirect method of communication. Humans and animals alike use a

[3]MADRaS also featured in FactorDaily, a popular-science media outlet : https://factordaily.com/multi-agent-driving-sim-madras

distinct communication channel in order to convey thoughts and achieve cooperation. Recent efforts have achieved success in limited conditions (Foerster *et al.*, 2016; Hausknecht, 2016; Sukhbaatar *et al.*, 2016).

- Scalability - One of the main issues of multi-agent learning is that of scalability - the extent of non-stationarity, the curse of dimensionality, all come into play. We need to keep this in mind as we expand our proof-of-concept to beyond a couple of agents trying to outwit a goalkeeper without requiring the need of unrealistic computation power.

Moreover, the aim of this entire effort has been towards getting autonomous driving vehicles on the road. Towards this end, we looked for existing open-source simulators out there for trying out some of the multi-agent ideas (not limited to the ones outlined in Section 4.2). We discovered that all of them innately support only egocentric control; that is, single agent behavior, and have pre-programmed behaviors for the other agents. The difficulty in introducing agents with custom behaviors in these simulators restricts the diversity of real-world scenarios that can be simulated. To address this issue, we developed MADRaS, wherein each car on the racing track can be independently controlled, enabling the creation of rich, custom-made traffic scenarios, and learning the policy of control of multiple agents simultaneously.

The following are some potential directions of future work that could be taken up in the multi-agent autonomous driving space, using MADRaS :

1. Creating custom driving traffic scenarios - with different number of vehicles following different kinds of driving behaviour of lane-following, overtaking, etc. The aim is to be able to create as many kinds of custom scenarios as possible using TORCS. The objective is two-fold : firstly, being able to introduce all such scenarios for the learning agent to encounter and learn from, and secondly, using the TORCS game engine to annotate each frame of the sequences of episodes of each of the scenarios in order to use them in imitation learning and active learning paradigms.

2. Benchmarking multi-agent RL algorithms - this task would entail comparing the performance of agents trained using multi-agent RL algorithms in MADRaS with those trained via single-agent algorithms to demonstrate the need and efficacy of

attempting to solve the autonomous driving problem via a multi-agent framework. For the purpose of benchmarking, we propose to use the following multi-agent RL algorithms/frameworks : MADDPG (Lowe *et al.*, 2017), PSMADDPG (Chu and Ye, 2017), SOM (Raileanu *et al.*, 2018), DIAL and RIAL (Foerster *et al.*, 2016).

3. Simulation of classical multi-agent scenarios :

   - **Platooning** - One of the earliest instances of multi-agent systems being deployed in vehicles was in the use of platooning (Varaiya, 1993), wherein vehicles travel at highway speeds with small inter-vehicle spacing to reduce congestion and still achieve high throughput without compromising safety.
   - **Pooling knowledge** - Apart from transferring information about pile-ups and possible diversions ahead to all the vehicles in the geographical vicinity, this power of reliable communication can be used to pool together the knowledge of multiple learning agents. An intuitive motivation could be to consider a large gridworld. With a single learning agent, one could solve the gridworld in n hours of training. With multiple learning agents pooling their experiences, we could cut down the training time significantly, possibly even linearly!
   - **Leveraging intent** - Drivers on the road constantly anticipate the potential actions of fellow drivers. As an example, for close maneuvering in car parks and intersections, eye contact is made to ensure a shared understanding.[4] As inter vehicle communication becomes ubiquitous and reliable (shout out to 5G!), autonomous vehicles will be able to transmit their intent to neighboring vehicles to implement the level of coordination beyond what human drivers currently achieve using eye contact.

We wish to point out that MADRaS is not a finished end product, and we have released it to the community for collaborative development. Among other things, our wishlist of things we would like to see as a part of MADRaS are:

[4]Defense Advanced Research Projects Agency (DARPA) stated that traffic vehicle drivers, unnerved by being unable to make eye contact with the robots, had resorted to watching the front wheels of the robots for an indication of their intent (Fletcher *et al.*, 2008).

1. **The release of a rich set of custom-made traffic scenarios** - This will help old and new members of the community in bootstrapping their research by training and testing on a bunch of such scenarios that are encountered in the real world.

2. **Inter-vehicular communication module** - With the advent of 5G, explicit communication between the various agents on the road could lead to a richer pool of knowledge that can be leveraged to make driving decisions. As of now, MADRaS does not have an explicit communication module.

3. **Integration with Intel RL Coach**[5] - With MADRaS being compatible with the OpenAI-gym interface, it would be great to see its integration with Intel RL Coach as well. This will enable the testing of a wide variety of the popular RL algorithms in the self-driving space without the enormous overhead of coding up all of those on one's own before testing it out, especially for new entrants in this field.

Interested researchers and developers in the community can report any incompatibility or bug by creating an issue in the GitHub repository. We hope MADRaS enables us to work together with new and veteran researchers in the industry and academia alike to make this FAD a reality!

[5]Intel RL Coach is an open source research framework for training and evaluating reinforcement learning (RL) agents. It contains multi-threaded implementations for some of the state-of-the-art RL algorithms, combined with various games and robotics environments. It enables efficient training of reinforcement learning agents on a desktop computer, without requiring any additional hardware. (Intel-AI, 2017)

CHAPTER 5

Curriculum Learning

5.1 Motivation

Instead of attacking a difficult learning problem in a monolithic fashion, we nearly always break the problem down to a sequence of manageable stages and sub-goals that are of progressively greater complexity. Humans make extensive use of this heuristic to learn many motor skills - crawling, walking, running, etc (Schmidt *et al.*, 2005). This sequenced acquisition of skills is a valuable heuristic in machine learning and robotics as well, e.g., (Konidaris and Barto, 2009). Such a defined order for learning skills is not unlike our education system, wherein simpler topics are taught first before introducing more intricate concepts.

Till now, we've been using hand-engineered reward functions that provide the agent with frequent, informative rewards. However, the actual reward obtained from the environment is only when a goal is scored, which is far too infrequent for DeepRL agents to bootstrap a policy form scratch. While domain knowledge might be enough to specify the 'correct' or optimal reward functions for small, low-dimensional tasks such as grid-worlds and ATARI games, hand-designing reward functions becomes more complicated in complex, high-dimensional, real-world scenarios like RoboSoccer or autonomous driving. The complexity intensifies in the multi-agent case with a multitude of factors requiring consideration, not limited to:

- How much reward should be given for a teammate scoring versus the agent?
- Should an agent be encouraged to move away from the ball if its teammate is already approaching it?
- Should an agent be rewarded if its teammate moves the ball towards the goal?
- How should agents be rewarded for passing?
- How much sub-optimality will be introduced by a mistake in weighting these different rewards?

In many senses, hand-designing a reward function for a complex task can be as difficult as hand-coding a policy to solve that task. Hence, we turn towards a curriculum learning-based framework.

5.2 Related work

5.2.1 Classical Usage

The idea of 'starting small' was introduced as early as in 1993 to learn a simple grammar (Elman, 1993). In 2009, Bengio et al. (Bengio *et al.*, 2009) introduced multi-stage curriculum learning strategies for vision and language tasks. In the experiments they conduct, they observe that training only on the 'clean' (not noisy) examples in the beginning makes the training faster, since it doesn't 'confuse' the learner. Additionally, introducing gradually more difficult examples (classified using domain knowledge) improves the training time as well.

5.2.2 Task Generation

A basic implementation of curriculum in the domain of robotics involves a 2-level hierarchy (Karpathy and Van De Panne, 2012), wherein the higher-level curriculum has a hand-designed ordering of tasks from easy to hard, and the lower-level curriculum learns these tasks in stages - achieve (stochastic greedy local search for the first successful experience), explore (try variants of a successful experience from a buffer), generalize (learning an inverse model that maps initial and target states to the set of actions to be taken). While such methods require a curriculum that needs to be manuall specified, there has been some work in the space of automatic curriculum generation as well. In (Narvekar *et al.*, 2016), tasks are created using both domain knowledge and by observing the agent's performance on a task. Given a target task and some trajectory tuples from it, they generate a set of source tasks. Work has also been done to automatically generate a curriculum of start states that adapts to the agent's performance (Florensa *et al.*, 2017).

5.2.3 Task Sequencing

As an alternative of manually ordering the tasks, automatic sequencing of these tasks has also been attempted (Narvekar *et al.*, 2017). They propose a curriculum MDP framework (CMDP) wherein the state space is the policy the agent can have; the actions the different tasks the agent can train on (i.e. the goal state is the ultimate policy); the transition function takes the CMDP to a new policy; and the reward function is negative of the amount of time required to train on a task. The goal is to find a sequence of tasks such that the total cost is less the cost incurred by learning directly on a target task. Since the tasks are learnt sequentially, the method could potentially suffer from *catastrophic forgetting* (French, 1999), as in artificial neural networks, there is a tendency for knowledge of previously learnt task(s) (e.g. task A) to be abruptly lost as information relevant to the current task (e.g. task B) is incorporated. In other words, training a neural network with new data causes it to overwrite (and thereby forget) what it has learned on previous data. In such a scenario, ideas of elastic weight consolidation (Kirkpatrick *et al.*, 2017) or interleaving data from multiple tasks (Parisotto *et al.*, 2016) might be useful.

5.2.4 Incorporating Active Feedback

One of the main drawbacks of curriculum learning methods are the lack of usage of active feedback. Once the curriculum is designed, it remains fixed and does not adapt with training. In the paradigm of Self-Paced Learning (SPL) (Kumar *et al.*, 2010), the curriculum is dynamically generated by the learner itself, according to what the learner has already learned. It embeds curriculum design as a regularization term into the learning objective such that the points with a low error (and hence 'easy') are used for training first. Samples with larger losses will be gradually appended to train a more 'mature' model. Narvekar *et al.* (2017) and Florensa *et al.* (2017) also design frameworks which adapt to the agent's performance.

5.3 Methodology

5.3.1 Task Generation

As discussed in Section 5.2.2 and 5.2.3, there exist methods which shown results on small domains with automatic generation of the curriculum and the subsequent task sequencing, but I believe this is an overkill. Given that we have so much prior domain knowledge that we could use, it seems pointless to make an already difficult problem even harder by expecting it to discover it's own curriculum as well. For the application of Half Field Offense, with the objective of just scoring a goal against an opponent team (as compared to the full-fledged setting of RoboSoccer which requires defending skills as well), the following set of simple sub-tasks of increasing complexity seem sufficient for the ultimate 'goal' of dribbling the ball past a team of opponents in order to score a goal:

1. Go to ball - the basic skill of approaching the ball
2. Dribble to goal - requires knowledge of (1), otherwise doesn't know what to do beyond kicking the ball once.
3. Shoot - another basic skill for the agent to learn that kicking the ball into the goal is actually fruitful.
4. Shoot against keeper - once the agent knows how to shoot, it now has to learn to score against a keeper.
5. Pass - another basic skill which could seen as a special case of shooting.
6. Pass and shoot - a combination of (4) and (5).

5.3.2 Task Sequencing

Curriculum Learning posits that if the agent is presented with a sequence or curriculum of tasks in such as way that the knowledge learned in each task is utilized in the next task, the agent will be able to more quickly and effectively learn to perform the target task. Hence just designing these tasks isn't enough, they have to be ordered in a principled manner. Two prospective methodologies :

1. Random ordering - this approach should be used as a baseline, which naively selects a random task from the set of possible tasks at each episode. The main disadvantage is that the agent may be presented with an advanced task before learning a basic skill, rendering the entire point of a curriculum useless. Since given an infinite learning time, the agent will learn all the tasks as each task is visited an infinite number of times.

2. Sequential ordering - using the same ordering as presented in Section **??**. To choose a task, an evaluation phase occurs every 10000 episodes or an average performance of over 80% of maximum possible reward for that particular task (as determined by domain knowledge). If the condition is met, the next task in the ordering is chosen. The complete heuristic, inspired by Hausknecht (2016), is reported in Algorithm 2.

As literature suggests, the experimental settings are ripe for catastrophic forgetting to occur, wherein the network's 'knowledge' of the previous tasks is 'overwritten' by the learning on subsequent tasks (French, 1999). We investigate this phenomenon in the Section 5.4.4.2.

Possible fixes:

1. Keep cycling through the tasks at regular intervals.

2. Keep a shared experience replay so that the agent has access to tuples from the other tasks at in every iteration.

3. Use concepts like Elastic Weight Consolidation which exploits the Fischer Information Matrix in order to overcome the catastrophic forgetting (Kirkpatrick *et al.*, 2017).

We use a heuristic-based approach to tackle this problem.

Algorithm 2 Sequential Ordering

procedure LEARN
    current task index $i = EvaluateTasks()$
    while $iter < maxIter$ **do**
        $PlayEpisode(T_i)$ ▷ Play and learn on current task
        if $iter \ \% \ 10000 == 0$ **then**
            $i = EvaluateTasks()$ ▷ Update the task to be evaluated

function EVALUATETASKS
    for $i \in 1 \ldots |T|$ **do** ▷ Following the ordering of tasks
        average return $R_i^{avg} = Evaluate(T_i)$
        if $R_i^{avg} < 0.8 \times R_i^{max}$ **then**
            return i ▷ Task T_i needs more training
    return $|T|$

5.3.3 Task Encoding

The next challenge of learning from a curriculum of tasks is informing the agent which task is currently active. Since that the state and action space is same, the agent has to know which task it is currently optimizing for, otherwise it may inadvertently accrue negative rewards simply by performing the wrong actions for the current task.

Again, the following methods of encoding these tasks will be used:

1. Integer IDs : Simple, but isn't meaningful when used with neural networks.
2. One-hot representation : Possible, but does not encode any similarity between tasks.
3. Task embeddings : Is more descriptive - encodes the objectives of the tasks and hence any similarities between them.

With n tasks, the one-hot representation $i \in \mathbb{R}^n$ is projected into the task embedding $\mathcal{T} \in \mathbb{R}^d$ using an embedding matrix $W^{emb} \in \mathbb{R}^{d \times n}$ (the size and choice of which is discussed later in this section) :

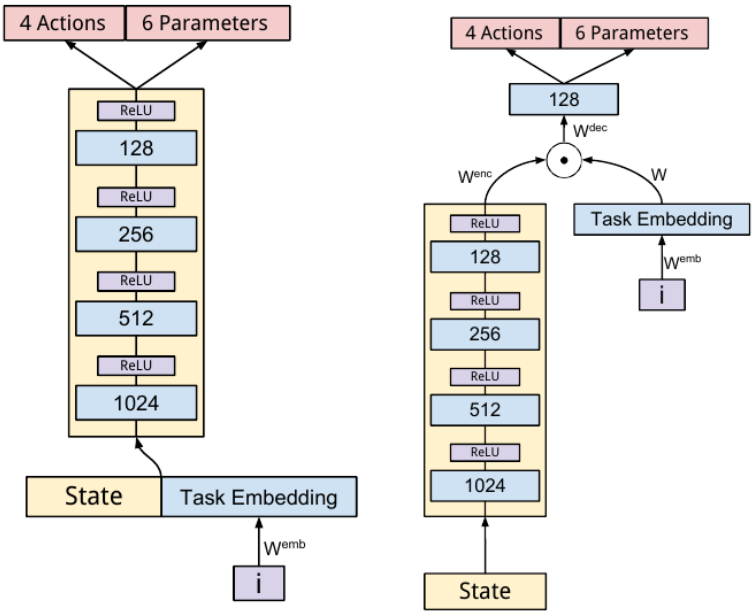
$$\mathcal{T} = W^{emb} i \tag{5.1}$$

In specific, task embeddings can be formed in the following two ways in order to encode the tasks for integration into the network representations (Hausknecht, 2016) :

5.3.3.1 State Embedding

Once the task embedding is formed, it is concatenated with the state representation vector of the agent (refer to Figure 5.1(a)). The advantage of this approach is its simplicity, but increases the size of the state space which could potentially make learning more difficult. A point to note is that since the embedding is now a part of the large state space, the agent could possibly learn to 'ignore' or pay less attention to it, rendering it useless.

The number of unique sub-tasks in the curriculum determine the size of the one-hot vector i. The size d of the task embedding vector is a hyperparameter. The experiments in the following sections use 8 and 128-dimensional embedding vectors.



(a) State Embedding (b) Weight Embedding

Figure 5.1: The task embedding architectures (courtesy Hausknecht (2016)). The State Embedding architecture simply has the task embedding concatenated into the feature state representation. On the other hand, the Weight Embedding architecture has the activations of the task embedding vector multiplicatively interact with the activations of the agent's second-to-last layer of the network.

5.3.3.2 Weight Embedding

The Weight Embedding architecture is motivated by Oh *et al.* (2015)'s action-conditional video prediction architecture for ATARI games. In specific, the activations of the task embedding vector multiplicatively interact with the activations of the agent's network:

$$o = W^{dec}(W\mathcal{T} \odot W^{enc}h) + b \tag{5.2}$$

where $\mathcal{T} \in \mathbb{R}^d$ is the task embedding vector, $h \in \mathbb{R}^c$ are the activations of a layer of the agent's network, $W \in \mathbb{R}^{p \times d}$ and $W^{enc} \in \mathbb{R}^{p \times c}$ are matrices which transform $\mathcal{T}$ and h into the same p-dimensional space. The Hadamard product of these encodings are taken and decoded into the output space $o \in \mathbb{R}^o$ using the decoder matrix $W^{dec} \in \mathbb{R}^{o \times f}$ and bias vector $b \in \mathbb{R}^o$.

The dimension p of the intermediate encoding is a hyperparameter which controls the trade-off between the extent of information encoded and the number of extra parameters. Following Hausknecht (2016), we used a fixed encoding of size $f = 128$.

5.3.3.3 Some design choices

Regarding the usage of task embeddings, the following questions had to be answered:

1. At which layer should the weight embedding be used?

   Intuitively, the lower level layers (defined as the ones towards the input features) are used for processing the state inputs and extracting features from them, and the higher layers are concerned with output selection (e.g. action selection or q-value estimation). Hence, it makes sense for the weight embedding to be incorporated higher up in the network such that it influences the action selection and q-value estimation. giving the network a chance to share the lower layers across tasks. This discussion leads us to a follow-up question.

2. Should the task embeddings be incorporated in both the actor and critic networks?

   Intuitively, yes. While the task embeddings are important for the actor network to choose actions based on the task-at-hand, it is equally important for the critic to be able to predict the q-value estimates based conditioned on the task. While we don't need the critic at 'test time', it is important to incorporate the embeddings in the critic network in order to ensure the propagation of meaningful gradients backward into the actor.

The final piece of the puzzle is W^{emb}. The current framework gives the flexibility of either freezing it after a manual initialization or allowing the network to learn it along with the other parameters. The former allow for the incorporation of the available domain knowledge. Since we know which components of reward function each of the sub-tasks defined in section 5.3.1 depend on, the rows of W^{emb} can be set accordingly (for instance, assigning a higher 'weightage' to some tasks over others).

5.4 Experiments

This section describes the experiments performed with the aforementioned methodology. The experiments were designed to answer the following questions:

1. Do the task embedding ideas actually help the agent discern between tasks to optimize for their reward functions individually?
2. If so, what size of embeddings is necessary to incorporate enough information for them to do so?
3. Additionally, how important is the order in which the sub-tasks are presented to the learner? Does catastrophic forgetting of older tasks occur?

5.4.1 Evaluation procedure

The performance graphs in the following pages are made in the following way :
After every 10000 iterations, the training is paused and the agent is evaluated 100 times on all the tasks present in the curriculum. An average performance over all the 100 runs is plotted as a data point for each of the tasks.

5.4.2 Sanity check

5.4.2.1 Experimental setup

In order to check the efficacy of the task embeddings, we perform a simple experiment[1] having two sub-tasks : *MoveToBall* and *MoveAwayFromBall*. As the names suggest, the goal of the above sub-tasks is to move towards and away from the ball respectively, and the agent gets a continuous positive reward for following the respective objectives, and negative otherwise.

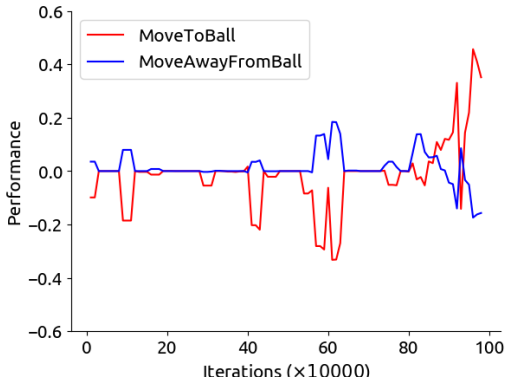
While these are seemingly straightforward tasks, they are chosen such that the agent *has to* somehow differentiate between the two in order to learn optimal policies for both of them simultaneously. If it doesn't use the task embeddings, because of the intentional high negative correlation within the tasks, the agent can optimize for atmost one of the two tasks.

5.4.2.2 Results and discussion

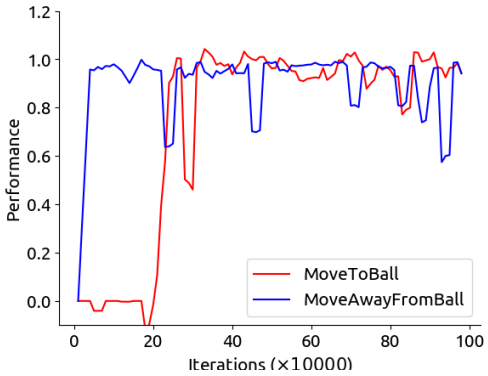
From Figure 5.2a, it is clear that without the use of task embeddings, the agent has no way to differentiate between the two tasks having the same feature representations. The agent takes a long to learn anything meaningful, and when it does, it is at the cost of the other. Remember, the tasks are complementary - naïvely optimizing for one will result in poor performance in the other, as is clearly visible in Figure 5.2a.

Figure 5.2b and 5.2c demonstrate that using either of the task embedding architecture, the agent successfully learns to differentiate between both the tasks and learns to master both of them quickly. This success of both the task embeddings validates our belief of the necessity of making the agent aware of the task-at-hand in the context of curriculum learning. In the next subsection, we present a harder set of sub-tasks as a curriculum for the task of scoring goals in RoboSoccer.

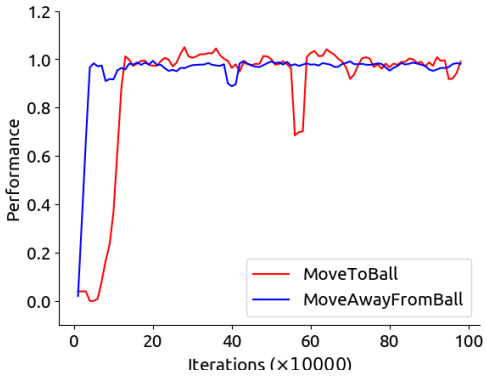
[1]The experiments in this particular section were performed as a part of a course project in 'CS7011 - *Topics in Reinforcement Learning*', offered in Jul-Nov 2017. Other experiments, including the ones in the subsequent section (5.4.3), were performed as a part of the DDP.



(a) No task embedding



(b) State embedding



(c) Weight embedding

Figure 5.2: Comparison of performance of the task embeddings on the two simple and complementary tasks. When no embeddings are used, the agent cannot discern between the two tasks and ends of oscillating between optimizing for one at the cost of the other. In contrast, both the tasks are quickly learned using the state and weight embeddings (of size 32).

5.4.3 Soccer task

5.4.3.1 Experimental setup

For a starter single agent soccer task, a curriculum of the following sub-tasks mentioned in Section 5.3.1 is used:

1. Reach ball task (*MoveToBall*)

   The agent and the ball are randomly initialized on the field. The objective is for the agent to reach the ball, and a reward to minimize the distance the distance between them is provided. Mathematically,

$$r_t = d_{t-1}(\text{agent, ball}) - d_t(\text{agent, ball}) \tag{5.3}$$

2. Dribble task (*KickToGoal*)

   The agent in possession of the ball is randomly initialized on the field. The objective is to minimize the distance between the ball and the goal. Mathematically, the reward looks like :

$$r_t = d_{t-1}(\text{goal, ball}) - d_t(\text{goal, ball}) \tag{5.4}$$

3. Score goal task (*Soccer*)

   This is the classic soccer task (Hausknecht, 2016). The agent and the ball are randomly initialized on the field. The objective is to score a goal by putting the ball through the goalposts. A reward of $+1$ is given when this is achieved, and 0 at all other times.

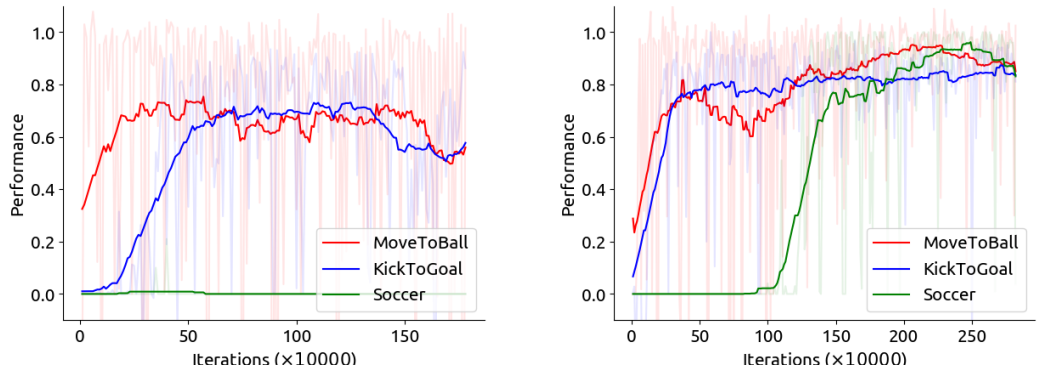
The *Soccer* task has a very sparse reward structure, and hence is extremely hard to learn independently. But if the agent knows how to approach the ball and dribble it towards the goal, this makes the original *Soccer* task a lot easier - which was the primary motivation of using the paradigm of curriculum learning.

Note : The common termination conditions for the above tasks are when the objective of the task is achieved, or if the ball is untouched for 100 timesteps, or if a maximum of 500 timesteps elapse.

5.4.3.2 Results and discussion

In Figure 5.3, we observe that only the weight embedding architecture when used with the sequential curriculum approaches stable learning in all of the three tasks involved. In particular, when the state embedding is used, the agent learns to perform decently on the *MoveToBall* and *DribbleToGoal* sub-tasks, but fail to learn the more difficult *Soccer* task altogether.

The failure of the state embedding architecture is interesting. Intuitively, since the architecture enables the agent to be aware of the different tasks at hand, it should be able to show stable learning of all the policies. But this is not the case. One possibility



(a) State embedding

(b) Weight embedding

Figure 5.3: The sequential curriculum ordering is used for generating both the performance curves. An embedding of size 128 is used for both state and weight embeddings. The weight embedding architecture seems to help the agent in learning a good control policy for all the three tasks, as opposed to the state embedding architecture, which fails to learn on the third and main *Soccer* task altogether.

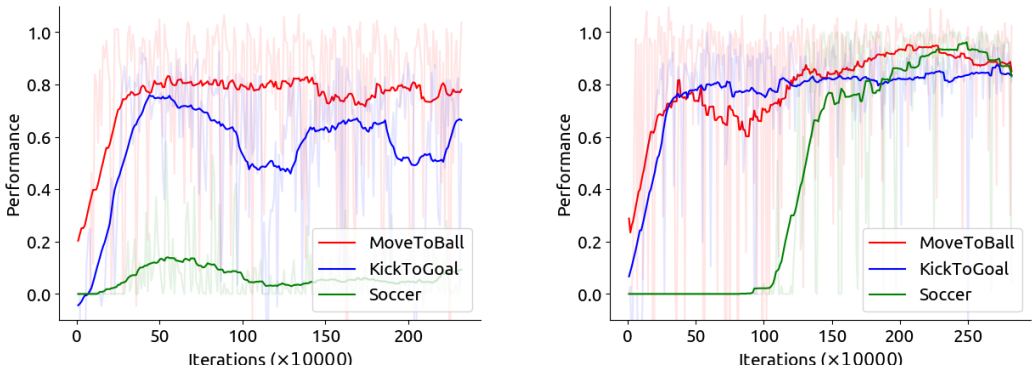
is that since the task embedding is concatenated with the state features themselves (i.e. at the lowest layer), the network could possibly learn to 'ignore' these set of augmented features. Experiments with concatenation of these embeddings with the higher levels will be taken up in future work.

5.4.4 Ablative analysis

5.4.4.1 Importance of task embedding

In this subsection, we check the importance of the usage of the task embeddings in this specific context of RoboSoccer experiments, wherein we learn to score a goal from scratch. Plots in Figure 5.4 show that when task embeddings are not supplied to the agent in any form, the agent fails to achieve stable learning across the three tasks. This is in contrast to the best-performing model with the weight embedding architecture of size 128, which converges to stable policies for all the three tasks simultaneously.

The failure when no embedding is used is easy to explain. In this case, since the agent cannot differentiate between the tasks (and hence their objectives), it fails to learn a meaningful policy for the sparse reward *Soccer* task. This underlines the original motivation of using the task embeddings in the first place, as mentioned in Section 5.3.3.



(a) No embedding

(b) Weight embedding

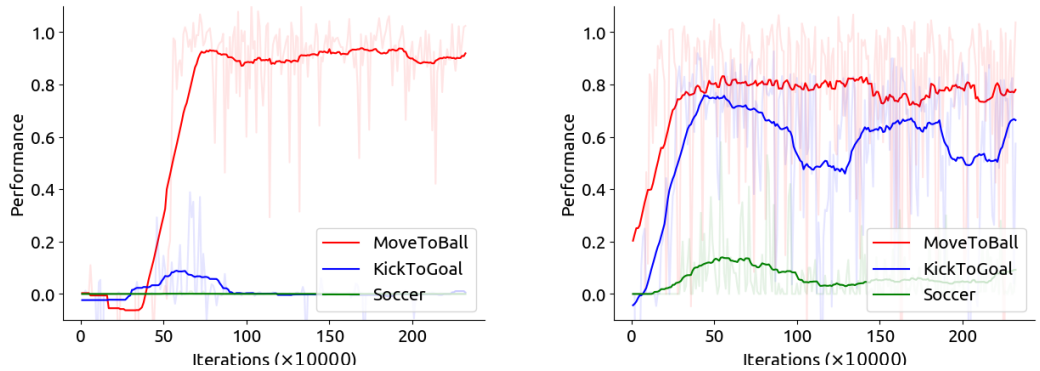
Figure 5.4: Comparison of performance of the agent trained naïvely with no embeddings versus the one trained with the weight embedding architecture (with the sequential ordering and embedding size 128). As expected, the agent fails to learn a stable control policy for all the three tasks when no embeddings are used.

5.4.4.2 Importance of task ordering

From Figure 5.5, we can see that the sequential ordering outlined in Algorithm 2 is necessary for the agent to achieve a decent performance in the curriculum of tasks. Specifically, in Figure 5.5a and 5.5b, we see that even the naïve model without any knowledge of which task it is optimizing for achieves a decent score on both the helper tasks with they are presented to it in a principled manner. We believe that further analysis of the exact tasks being presented to the agent will probably reveal that the agent is almost never presented the full *Soccer* task since it never achieves a score above the specified threshold on the helper tasks.

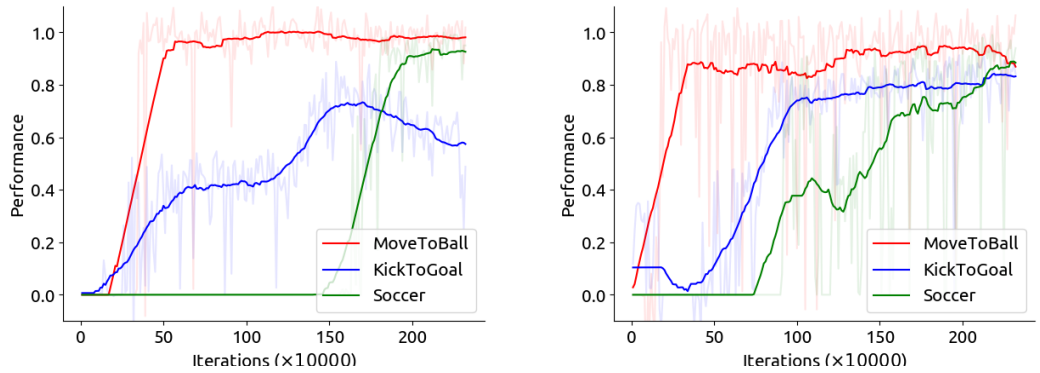
Closer inspection of Figure 5.5c and 5.5d reveal that only when the sequential ordering is used, the agent achieves stable performance across all the three tasks. When the tasks are presented in a random order, though the agent learns to optimize for them simultaneously with the help of the weight embedding architecture, there is catastrophic forgetting seen in the second *KickToGoal* task (blue) in Figure 5.5c.

This underlines the importance of the usage of a principled ordering in learning a curriculum of tasks, the lack of which might lead to catastrophic forgetting of the older tasks (French, 1999).



(a) No embedding - random ordering

(b) No embedding - sequential ordering



(c) Weight embedding - random ordering

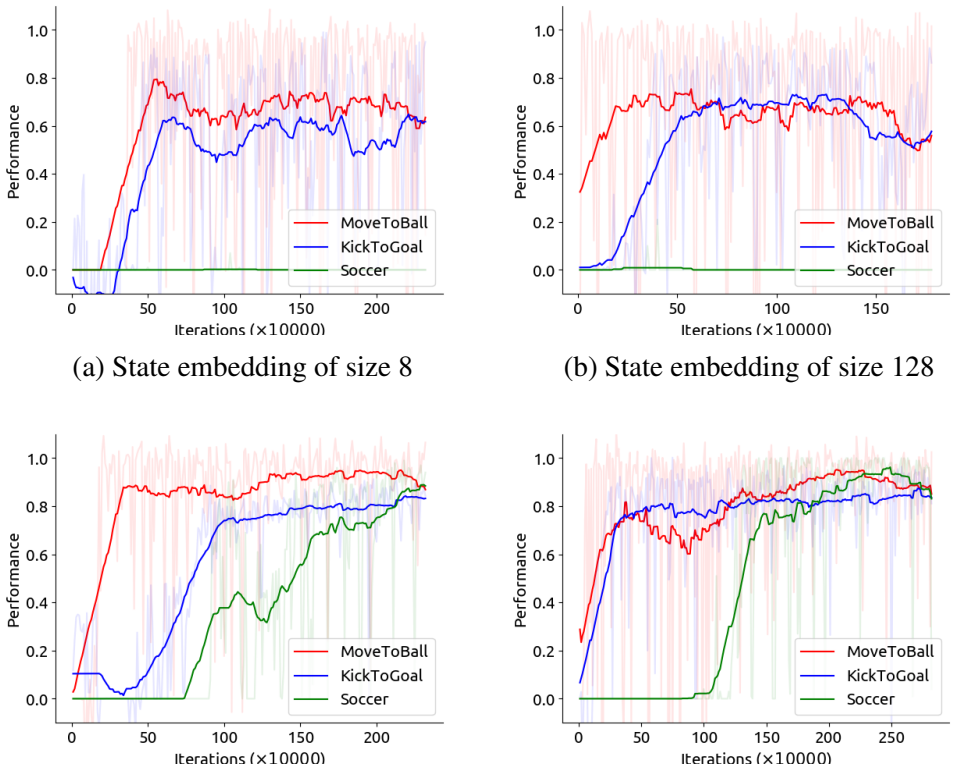
(d) Weight embedding - sequential ordering

Figure 5.5: Comparison of performance with the sequential ordering and the lack of it for the different types of embeddings. The agent fails to demonstrate stable learning on all the three tasks when they are presented in a random order, while catastrophically forgetting the older tasks. The weight embedding has size 8. Please refer to the main text for a detailed discussion.

Additional analysis of the size of the task embeddings

- From Figure 5.6a and 5.6b, it is clear that the state embedding architecture fails with both the small and large dimensions of the embeddings with the reason stated in Section 5.4.3.2.

- On the other hand, both the small and large dimensions of weight embeddings enable the agent to successfully learn the harder, sparse reward *Soccer* task as well. Since a larger amount of information can be encoded within a larger embedding, the model with the weight embedding of size 128 shows more stable learning as compared to the one with size 8 (refer to Figure 5.6c and 5.6d)



(a) State embedding of size 8

(b) State embedding of size 128

(c) Weight embedding of size 8

(d) Weight embedding of size 128

Figure 5.6: Comparison of performance with different sizes of embeddings. The weight embedding architecture is seen to show a decent performance across the three tasks for both sizes of embeddings, while the state embeddings fail completely on the third task.

5.5 Conclusions and Future Work

We've successfully answered the questions we set out to answer in Section 5.4, namely:

1. Yes, tasks embeddings indeed help in discerning between the different sub-tasks that have been designed to make the target task easier, which enables the agent to optimize for them individually (and simultaneously) despite no change in the state feature representations.
2. We've also seen that the weight embedding architecture is fairly robust to the size of the embeddings used, with larger sizes encoding more and sufficient information.
3. Finally, we've also seen that the order in which the sub-tasks are presented to the agent is critical in enabling stable learning across all the tasks at hand, while also preventing catastrophic forgetting of the previous tasks.

This is just the beginning, though. There is a long way to go in achieving the final objective of training an complete multi-agent RoboSoccer team in the 2D simulation league with reinforcement learning. The following list enumerates some of the challenges with this approach and further problems that need to be addressed in order to make this a reality :

- Extending to multi-agent scenarios - now that we have a proof-of-concept of the curriculum learning approach in a single-agent setting, we can extend this to the multi-agent setting, with the sub-tasks enumerated in Section 5.3.1. Approaches with both indirect and explicit communication can be explored, for instance MADDPG (Lowe *et al.*, 2017) and DIAL (Foerster *et al.*, 2016).

- Handling catastrophic forgetting - the current heuristic ordering detailed in Algorithm 2 is arguably an ad-hoc method to ameliorate the problem of catastrophic forgetting while learning to achieve a high performance in the end goal. In this scenario, more principled approaches using information theoretic concepts like by Kirkpatrick *et al.* (2017) to keep the network parameters of the new task close to the original ones could be worth exploring.

- Getting state embeddings to work - we concluded that the state embedding architecture failed because the network could have possible learned to ignore these embeddings which were concatenated at the lowest feature-level. As with the weight embedding architecture, the state embeddings can be concatenated at higher levels of the network, above the feature processing layers.

- More ablative experiments - apart from the experiments performed in Section 5.4.4, the importance of each component of the proposed approach can be understood by :

    1. Using task embeddings only in the actor which has to choose the actions to be performed to check how important it is for the critic to discern between the sub-tasks as well.
    2. Removing or modifying some of the 'helpful' sub-tasks to see if the agent can then learn the target-task with just the spare reward.
    3. Checking how the sequential curriculum is being used in terms of frequency and times of visits to the older tasks to prevent catastrophic forgetting.

Adequately answering the above questions by pursuing the aforementioned directions would amount to significant progress towards a team of deep reinforcement learning soccer players.

CHAPTER 6

Summary and Conclusions

This chapter tersely summarizes the contributions and takeaways from this thesis.

We first break down the ambitious project of getting autonomous vehicles on roads of India(!) into three smaller problems in Chapter 1 - that of learning safe and reliable policies from a set of expert demonstrations; learning a set of policies for multiple agents simultaneously while interacting in the same environment; and finally learning to perform hard and complex tasks by breaking them down into smaller and tractable sub-tasks.

In particular, Chapter 3 first identifies a drawback with the existing state-of-the-art algorithm of learning a behavioral policy from a fixed set of expert trajectories. The heavy-tail exhibited by the trained agents make them unreliable in risk-sensitive applications like autonomous driving. Taking inspiration from portfolio risk-minimization literature, we then propose a risk-averse imitation learning framework which explicitly minimizes the tail risk within the generative adversarial framework. Testing this framework on a set on benchmark physics-based control tasks, we demonstrate that the trajectories produced by this new learned agent exhibits lower tail-risk, making this effort a viable step in the direction of learning low-risk policies by imitation learning in complex risk-sensitive environments.

In Chapter 4, we first outline the need for multi-agent learning and the associated complexities as compared to classical single-agent learning. Motivated by sharing within agents, we evaluate several approaches like sharing parameters across agent, sharing the experience replay buffer, etc, while trying to score goals in a goal manned by a naïve or expert goalkeeper in the HFO RoboSoccer simulator. Analysis of the results points towards the drawbacks of using hand-engineered reward functions and motivates the usage of a curriculum learning based approach. Finally, faced with a lack of an equivalent multi-agent simulator for autonomous driving research, we develop our own on top of the existing TORCS. MADRaS offers the ability to create customized traffic scenarios and train various multi-agent algorithms in a plug-and-play fashion.

Finally, in Chapter 5, we build on the inferences and insights from Chapter 4 of the difficulty in designing appropriate reward functions for complex, real-world tasks. We apply curriculum learning in the context of DRL by breaking down the sparse reward goal-scoring task of RoboSoccer into smaller, individual sub-tasks of dribbling and kicking the ball. We condition the network weights on the task-at-hand, and achieve stable performance across the curriculum of tasks. Furthermore, we demonstrate the significance of each component - the type of embedding, its size, the heuristic ordering - via an elaborate ablative analysis.

While we aren't ready to deploy self-driving cars on Indian roads yet, each of the aforementioned efforts is a viable step towards achieving that dream in its own way.

6.1 The long-term goal

Now that we have a proof-of-concept in each of the above modules, we can start putting them together. After successful deployment on simulated physics-based control tasks, the risk-averse imitation learning framework can now be applied in the vehicular setting, starting with the TORCS driving environment. The next step would be to incorporate the ideas of multi-agent learning into it, since negotiating in traffic is a multi-player game. Various kinds of traffic scenarios can me modeled with the newly-developed MADRaS to subsequently test various multi-agent learning algorithms. We can then leverage the literature on third-person learning to explore the transfer of these learned policies to the real world. Finally, all of these aforementioned tasks require a flavor of curriculum learning in order to make them more feasible and tractable.

While many open challenges remain to be addressed, we believe that sustained effort in the direction of these seemingly small steps will lead to a giant leap for mankind in terms of completely revolutionizing the transportation industry as we know it.

REFERENCES

1. **Abbeel, P.** and **A. Y. Ng**, Apprenticeship learning via inverse reinforcement learning. *In Proceedings of the 21st International Conference on Machine Learning*. ACM, 2004.

2. **Abbeel, P.** and **A. Y. Ng**, Inverse reinforcement learning. *In Encyclopedia of machine learning*. Springer, 2011, 554–558.

3. **Barto, A. G.**, **R. S. Sutton**, and **C. W. Anderson** (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5), 834–846.

4. **Bengio, Y.**, **J. Louradour**, **R. Collobert**, and **J. Weston**, Curriculum learning. *In Proceedings of the 26th annual International Conference on Machine Learning*. ACM, 2009.

5. **Bojarski, M.**, **D. Del Testa**, **D. Dworakowski**, **B. Firner**, **B. Flepp**, **P. Goyal**, **L. D. Jackel**, **M. Monfort**, **U. Muller**, **J. Zhang**, *et al.* (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

6. **Brockman, G.**, **V. Cheung**, **L. Pettersson**, **J. Schneider**, **J. Schulman**, **J. Tang**, and **W. Zaremba** (2016). OpenAI gym. *arXiv preprint arXiv:1606.01540*.

7. **Busoniu, L.**, **R. Babuska**, and **B. De Schutter** (2008). A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008*.

8. **Cardamone, L.**, **D. Loiacono**, and **P. L. Lanzi**, Learning drivers for torcs through imitation using supervised methods. *In Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*. IEEE, 2009.

9. **Chow, Y.** and **M. Ghavamzadeh**, Algorithms for cvar optimization in mdps. *In Advances in neural information processing systems*. 2014.

10. **Chu, X.** and **H. Ye** (2017). Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1710.00336*.

11. **Dalleh, N.** (2011). *Why is CVaR superior to VaR?(c2009)*. Ph.D. thesis.

12. **Daumé, H.**, **J. Langford**, and **D. Marcu** (2009). Search-based structured prediction. *Machine learning*, **75**(3), 297–325.

13. **DeepDrive.io** (2018). Deepdrive. `https://github.com/deepdrive/deepdrive`.

14. **Dosovitskiy, A.**, **G. Ros**, **F. Codevilla**, **A. Lopez**, and **V. Koltun**, CARLA: An open urban driving simulator. *In Proceedings of the 1st Annual Conference on Robot Learning*. 2017.

15. **Duan, Y.**, **B. X. Cui**, and **X. H. Xu** (2012). A multi-agent reinforcement learning approach to robot soccer. *Artificial Intelligence Review*, 1–19.

16. **Elman, J. L.** (1993). Learning and development in neural networks: The importance of starting small. *Cognition*, **48**(1), 71–99.

17. **Fletcher, L.**, **S. Teller**, **E. Olson**, **D. Moore**, **Y. Kuwata**, **J. How**, **J. Leonard**, **I. Miller**, **M. Campbell**, **D. Huttenlocher**, *et al.* (2008). The mit–cornell collision and why it happened. *Journal of Field Robotics*, **25**(10), 775–807.

18. **Florensa, C.**, **D. Held**, **M. Wulfmeier**, and **P. Abbeel**, Reverse curriculum generation for reinforcement learning. *In Proceedings of the 1st Conference on Robotic Learning (CoRL)*. 2017.

19. **Foerster, J.**, **Y. Assael**, **N. de Freitas**, and **S. Whiteson**, Learning to communicate with deep multi-agent reinforcement learning. *In Advances in Neural Information Processing Systems*. 2016.

20. **Foerster, J.**, **G. Farquhar**, **T. Afouras**, **N. Nardelli**, and **S. Whiteson**, Counterfactual multi-agent policy gradients. *In Proceedings of the 2018 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

21. **French, R. M.** (1999). Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, **3**(4), 128–135.

22. **Garcıa, J.** and **F. Fernández** (2015). A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, **16**(1), 1437–1480.

23. **Goodfellow, I.**, **J. Pouget-Abadie**, **M. Mirza**, **B. Xu**, **D. Warde-Farley**, **S. Ozair**, **A. Courville**, and **Y. Bengio**, Generative adversarial nets. *In Advances in neural information processing systems*. 2014.

24. **Graves, A.** (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

25. **Hausknecht, M.**, **P. Mupparaju**, **S. Subramanian**, **S. Kalyanakrishnan**, and **P. Stone**, Half field offense: An environment for multiagent learning and ad hoc teamwork. *In AAMAS Adaptive Learning Agents (ALA) Workshop*. 2016.

26. **Hausknecht, M.** and **P. Stone**, Deep Reinforcement Learning in Parameterized Action Space. *In Proceedings of the 4th International Conference on Learning Representations (ICLR-16)*. 2016.

27. **Hausknecht, M. J.** (2016). *Cooperation and communication in multiagent deep reinforcement learning*. Ph.D. thesis.

28. **Haykin, S.**, *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1998, 2nd edition. ISBN 0132733501.

29. **Heger, M.**, Consideration of risk in reinforcement learning. *In Proceedings of the 11th International Conference on Machine Learning*. 1994.

30. **Hinton, G.**, **L. Deng**, **D. Yu**, **G. E. Dahl**, **A.-r. Mohamed**, **N. Jaitly**, **A. Senior**, **V. Vanhoucke**, **P. Nguyen**, **T. N. Sainath**, *et al.* (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, **29**(6), 82–97.

31. **Ho, J.** and **S. Ermon**, Generative adversarial imitation learning. *In Advances in Neural Information Processing Systems*. 2016.

32. **Hochreiter, S.** and **J. Schmidhuber** (1997). Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

33. **Intel-AI** (2017). Intel reinforcement learning coach. `https://github.com/NervanaSystems/coach`.

34. **Investopedia** (2017). Definition of tail risk. `http://www.investopedia.com/terms/t/tailrisk.asp`. Accessed: 2017-09-11.

35. **Kalyanakrishnan, S.**, **Y. Liu**, and **P. Stone**, Half field offense in robocup soccer: A multiagent reinforcement learning case study. *In Robot Soccer World Cup*. Springer, 2006.

36. **Karpathy, A.** and **M. Van De Panne** (2012). Curriculum learning for motor skills. *Advances in Artificial Intelligence*, 325–330.

37. **Kingma, D.** and **J. Ba** (2015). Adam: A method for stochastic optimization. *arXiv:1310.5107 [cs.CV]*.

38. **Kirkpatrick, J.**, **R. Pascanu**, **N. Rabinowitz**, **J. Veness**, **G. Desjardins**, **A. A. Rusu**, **K. Milan**, **J. Quan**, **T. Ramalho**, **A. Grabska-Barwinska**, *et al.* (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 201611835.

39. **Knight, W.** (2017). Alpha zero's "alien" chess shows the power, and the peculiarity, of ai. URL `https://www.technologyreview.com/s/609736/alpha-zeros-alien-chess-shows-the-power-and-the-peculiarity-of-ai/`.

40. **Konidaris, G.** and **A. G. Barto**, Skill discovery in continuous reinforcement learning domains using skill chaining. *In Advances in Neural Information Processing Systems*. 2009.

41. **Kraemer, L.** and **B. Banerjee** (2016). Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, **190**, 82–94.

42. **Krizhevsky, A.**, **I. Sutskever**, and **G. E. Hinton**, Imagenet classification with deep convolutional neural networks. *In Advances in neural information processing systems*. 2012.

43. **Kumar, M. P.**, **B. Packer**, and **D. Koller**, Self-paced learning for latent variable models. *In Advances in Neural Information Processing Systems*. 2010.

44. **LeCun, Y.**, **Y. Bengio**, and **G. Hinton** (2015). Deep learning. *nature*, **521**(7553), 436.

45. **Levine, S.** and **V. Koltun** (2012). Continuous inverse optimal control with locally optimal examples. *arXiv preprint arXiv:1206.4617*.

46. **Lillicrap, T. P.**, **J. J. Hunt**, **A. Pritzel**, **N. Heess**, **T. Erez**, **Y. Tassa**, **D. Silver**, and **D. Wierstra**, Continuous control with deep reinforcement learning. *In Proceedings of the 4th International Conference on Learning Representations (ICLR-16)*. 2016.

47. **Liu, B.**, **S. Singh**, **R. L. Lewis**, and **S. Qin**, Optimal rewards in multiagent teams. *In Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*. IEEE, 2012.

48. **Loiacono, D.**, **A. Prete**, **P. L. Lanzi**, and **L. Cardamone**, Learning to overtake in torcs using simple reinforcement learning. *In Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010.

49. **Lowe, R.**, **Y. Wu**, **A. Tamar**, **J. Harb**, **P. Abbeel**, and **I. Mordatch**, Multi-agent actor-critic for mixed cooperative-competitive environments. *In Advances in Neural Information Processing Systems*. 2017.

50. **Masson, W.**, **P. Ranchod**, and **G. Konidaris**, Reinforcement learning with parameterized actions. *In AAAI*. 2016.

51. **Matignon, L.**, **G. J. Laurent**, and **N. Le Fort-Piat** (2012). Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, **27**(1), 1–31.

52. **Minsky, M.** (1954). *Theory of neural-analog reinforcement systems and its application to the brain-model problem. Princeton University Ph. D.* Ph.D. thesis, Dissertation.

53. **Mnih, V.**, **A. P. Badia**, **M. Mirza**, **A. Graves**, **T. Lillicrap**, **T. Harley**, **D. Silver**, and **K. Kavukcuoglu**, Asynchronous methods for deep reinforcement learning. *In International Conference on Machine Learning*. 2016.

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

55. **Nagengast, A. J.**, **D. A. Braun**, and **D. M. Wolpert** (2010). Risk-sensitive optimal feedback control accounts for sensorimotor behavior under uncertainty. *PLoS computational biology*, **6**(7), e1000857.

56. **Narvekar, S.**, **J. Sinapov**, **M. Leonetti**, and **P. Stone**, Source task creation for curriculum learning. *In Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

57. **Narvekar, S.**, **J. Sinapov**, and **P. Stone**, Autonomous task sequencing for customized curriculum design in reinforcement learning. *In Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*. 2017.

58. **Ng, A. Y.**, **A. Coates**, **M. Diel**, **V. Ganapathi**, **J. Schulte**, **B. Tse**, **E. Berger**, and **E. Liang**, Autonomous inverted helicopter flight via reinforcement learning. *In Experimental Robotics IX*. Springer, 2006, 363–372.

59. **Ng, A. Y.**, **S. J. Russell**, *et al.*, Algorithms for inverse reinforcement learning. *In International Conference on Machine Learning*. 2000.

60. **Oh, J.**, **X. Guo**, **H. Lee**, **R. L. Lewis**, and **S. Singh**, Action-conditional video prediction using deep networks in atari games. *In Advances in Neural Information Processing Systems*. 2015.

61. **Omidshafiei, S.**, **J. Pazis**, **C. Amato**, **J. P. How**, and **J. Vian**, Deep decentralized multi-task multi-agent reinforcement learning under partial observability. *In International Conference on Machine Learning*. 2017.

62. **Parisotto, E.**, **J. L. Ba**, and **R. Salakhutdinov**, Actor-mimic: Deep multitask and transfer reinforcement learning. *In Proceedings of the 4th International Conference on Learning Representations (ICLR)*. 2016.

63. **Pomerleau, D. A.**, Alvinn: An autonomous land vehicle in a neural network. *In Advances in Neural Information Processing Systems*. 1989.

64. **Raileanu, R.**, **E. Denton**, **A. Szlam**, and **R. Fergus** (2018). Modeling others using oneself in multi-agent reinforcement learning. *arXiv preprint arXiv:1802.09640*.

65. **Rajeswaran, A.**, **S. Ghotra**, **S. Levine**, and **B. Ravindran** (2016). Epopt: Learning robust neural network policies using model ensembles. *5th International Conference on Learning Representations*.

66. **Rockafellar, R. T.** and **S. Uryasev** (2000). Optimization of conditional value-at-risk. *Journal of risk*, **2**, 21–42.

67. **Ross, S.** and **D. Bagnell**, Efficient reductions for imitation learning. *In Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*. 2010.

68. **Ross, S.** and **J. A. Bagnell** (2014). Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*.

69. **Ross, S.**, **G. J. Gordon**, and **D. Bagnell**, A reduction of imitation learning and structured prediction to no-regret online learning. *In International Conference on Artificial Intelligence and Statistics*. 2011.

70. **Schaal, S.**, Learning from demonstration. *In Advances in Neural Information Processing Systems*. 1997.

71. **Schmidt, R. A.**, **T. D. Lee**, *et al.*, *Motor control and learning: A behavioral emphasis*, volume 4. Human kinetics Champaign, IL, 2005.

72. **Schulman, J.**, **S. Levine**, **P. Abbeel**, **M. Jordan**, and **P. Moritz**, Trust region policy optimization. *In International Conference on Machine Learning*. 2015.

73. **Shah, S.**, **D. Dey**, **C. Lovett**, and **A. Kapoor**, Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *In Field and Service Robotics*. 2017. URL `https://arxiv.org/abs/1705.05065`.

74. **Shalev-Shwartz, S.**, **S. Shammah**, and **A. Shashua** (2016). Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*.

75. **Silver, D.**, **A. Huang**, **C. J. Maddison**, **A. Guez**, **L. Sifre**, **G. Van Den Driessche**, **J. Schrittwieser**, **I. Antonoglou**, **V. Panneershelvam**, **M. Lanctot**, *et al.* (2016). Mastering the game of go with deep neural networks and tree search. *nature*, **529**(7587), 484–489.

76. **Silver, D.**, **T. Hubert**, **J. Schrittwieser**, **I. Antonoglou**, **M. Lai**, **A. Guez**, **M. Lanctot**, **L. Sifre**, **D. Kumaran**, **T. Graepel**, *et al.* (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

77. **Silver, D.**, **G. Lever**, **N. Heess**, **T. Degris**, **D. Wierstra**, and **M. Riedmiller**, Deterministic policy gradient algorithms. *In ICML*. 2014.

78. **Simonyan, K.** and **A. Zisserman**, Very deep convolutional networks for large-scale image recognition. *In International Conference on Learning Representations*. 2015.

79. **Stadie, B. C.**, **P. Abbeel**, and **I. Sutskever** (2017). Third-person imitation learning. *6th International Conference on Learning Representations*.

80. **Sukhbaatar, S.**, **R. Fergus**, *et al.*, Learning multiagent communication with backpropagation. *In Advances in Neural Information Processing Systems*. 2016.

81. **Sutton, R.** and **A. Barto**, *Reinforcement Learning: An Introduction*. A Bradford book. Bradford Book, 1998. ISBN 9780262193986. URL `https://books.google.co.in/books?id=CAFR6IBF4xYC`.

82. **Tampuu, A.**, **T. Matiisen**, **D. Kodelja**, **I. Kuzovkin**, **K. Korjus**, **J. Aru**, **J. Aru**, and **R. Vicente** (2017). Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, **12**(4), e0172395.

83. **Tan, M.**, Multi-agent reinforcement learning: Independent vs. cooperative agents. *In Proceedings of the tenth international conference on machine learning*. 1993.

84. **Tesauro, G.**, Td-gammon: A self-teaching backgammon program. *In Applications of Neural Networks*. Springer, 1995, 267–285.

85. **Todorov, E.**, **T. Erez**, and **Y. Tassa**, Mujoco: A physics engine for model-based control. *In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012.

86. **Tsitsiklis, J. N.** and **B. Van Roy**, Analysis of temporal-difference learning with function approximation. *In Advances in neural information processing systems*. 1997.

87. **Varaiya, P.** (1993). Smart cars on smart roads: problems of control. *IEEE Transactions on automatic control*, **38**(2), 195–207.

88. **Watkins, C. J. C. H.** (1989). *Learning from delayed rewards*. Ph.D. thesis, King's College, Cambridge.

89. **Williams, R. J.**, Simple statistical gradient-following algorithms for connectionist reinforcement learning. *In Reinforcement Learning*. Springer, 1992, 5–32.

90. **Wymann, B.**, **E. Espié**, **C. Guionneau**, **C. Dimitrakakis**, **R. Coulom**, and **A. Sumner** (2000). Torcs, the open racing car simulator. *Software available at http://torcs.sourceforge. net*, **4**.

91. **Ziebart, B. D.**, **A. L. Maas**, **J. A. Bagnell**, and **A. K. Dey**, Maximum entropy inverse reinforcement learning. *In AAAI*, volume 8. Chicago, IL, USA, 2008.

LIST OF PAPERS BASED ON THESIS

1. Santara, A.*, **Naik, A.***, Ravindran, B., Das, D., Mudigere, D., Avancha, S., and Kaul, B. RAIL: Risk-averse Imitation Learning *To appear in the Proceedings of the Seventeenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (Extended Abstract), (2018).