# LIFELONG REINFORCEMENT LEARNING ON MOBILE ROBOTS

David Isele

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2019

Supervisor of Dissertation

Eric Eaton
Senior Lecturer in Computer and Information Science

Co-Supervisor of Dissertation

C.J. Taylor
Professor and Director of MSE in Robotics, Computer and Information Science

Graduate Group Chairperson

Rajeev Alur

Dissertation Committee

Jean Gallier
Professor, Computer and Information Science

Kostas Daniilidis
Professor, Computer and Information Science

David W. Aha
Naval Research Laboratory

David Brainard
RRL Professor, Psychology

# Acknowledgment

Creating a dissertation is not a small task. I am tremendously grateful to all the inspiring conversations, guidance, and kindness I have received along the way. I would especially like to thank my friends, editors, and all-around wonderful human beings: Sonia Roberts, Christine Allen-Blanchette, and Yulan Lin for all their unwavering help in preparing my dissertation. I was fortunate enough to have two thoughtful and patient advisors, Eric Eaton and C.J. Taylor, guiding me along the way, and without whom this work would not have been possible. I also want to acknowledge my committee members, each of whom has additionally served as mentors and targets of my various errant philosophical queries into mathematics (Jean Gallier), cognition (David W. Aha), psychology (David Brainard), and engineering (Kostas Daniilidis). Also, I want to recognize the guidance of Alan Wolf, Dan Koditschek, and Akansel Cosgun for their mentoring and advice. Finally, I want to thank my parents, who have been so reliable and supportive that my gratitude feels as unnecessary as thanking the air for being there.

# ABSTRACT


**LIFELONG REINFORCEMENT LEARNING ON MOBILE ROBOTS**

David Isele
Eric Eaton
C.J. Taylor

Machine learning has shown tremendous growth in the past decades, unlocking new capabilities in a variety of fields including computer vision, natural language processing, and robotic control. While the sophistication of individual problems a learning system can handle has greatly advanced, the ability of a system to extend beyond an individual problem to adapt and solve new problems has progressed more slowly. This thesis explores the problem of progressive learning. The goal is to develop methodologies that accumulate, transfer, and adapt knowledge in applied settings where the system is faced with the ambiguity and resource limitations of operating in the physical world.

There are undoubtedly many challenges to designing such a system, my thesis looks at the component of this problem related to how knowledge from previous tasks can be a benefit in the domain of reinforcement learning where the agent receives rewards for positive actions. Reinforcement learning is particularly difficult when training on physical systems, like mobile robots, where repeated trials can damage the system and unrestricted exploration is often associated with safety risks. I investigate how knowledge can be efficiently accumulated and applied to future reinforcement learning problems on mobile robots in order to reduce sample complexity and enable systems to

adapt to novel settings. Doing this involves mathematical models which can combine knowledge from multiple tasks, methods for restructuring optimizations and data collection to handle sequential updates, and data selection strategies that can be used to address resource limitations.

# Contents

# List of Tables

# List of Figures

# List of Symbols

Table 1: Table of Symbols

| Symbol | Meaning | First Occurance |
|---|---|---|
| $a$ | action | 2.1.1 |
| $\alpha$ | point estimate of single task model parameters | 2.2.4 |
| $\mathcal{A}$ | set of all actions | 2.1.1 |
| $\mathscr{A}$ | set of Action sets | 2.1.4 |
| $\aleph$ | non-zero subset of L in convergence proof | 4.4.2 |
| æ | extended action | 2.1.5 |
| Æ | set of extended actions | 2.1.5 |
| Å | set of set of extended actions | 7.4.1 |
| $\boldsymbol{A}$ | ELLA decomposition | 2.2.4 |
| $\boldsymbol{b}$ | ELLA decomposition | 2.2.4 |
| $B$ | episodic memory buffer | 6.2.3 |
| $\beta$ | TaDeLL combined point estimate | 4.2 |
| $\mathcal{B}$ | benefit | 4.5.4 |
| $c$ | constant | 7.5 |
| $\mathscr{C}$ | constraints | 2.2.3 |
| $d$ | dimension | 2.1.1 |
| $\boldsymbol{D}$ | TaDeLL description dictionary | 4.2 |
| $\mathfrak{D}_{\mathrm{KL}}$ | KL divergence | 2.1.2 |
| $\mathscr{D}(\cdot)$ | Distribution | 1.3 |
| $\delta$ | probability margin | 7.2 |
| $\Delta$ | an infinitesimal change | 3.1.1 |
| $e_h$ | experience at time step $h$ | 2.1.1 |
| $E$ | set of experiences | 2.2.3 |
| $\mathbb{E}$ | expectation | 2.1.1 |
| $\epsilon$ | noise | 4.2 |
| $f_{\boldsymbol{\theta}}$ | learned function for single task supervised learning | 2.2 |

| | | |
|---|---|---|
| $\mathsf{F}$ | Frobinius norm | 2.2.4 |
| $g$ | sparse code approx of $L$ in convergence proof | 4.4.2 |
| $G$ | fisher information matrix | 4.2 |
| $\gamma$ | discount factor | 2.1.1 |
| $\Gamma$ | Hessian | 2.2.4 |
| $h$ | time step | 2.1.1 |
| $H$ | final time step | 2.1.1 |
| $i$ | general purpose index | 2.1.4 |
| $I$ | Identity Matrix | 2.2.4 |
| $j$ | training iteration | 3.1.1 |
| $J$ | final training iteration | 3.1.1 |
| $\mathcal{J}$ | log of expected return | 2.1.2 |
| $k$ | size of the basis for ELLA, number of nearest neighbors | 2.2.4 |
| $K$ | TaDeLL combined dictionary | 4.2 |
| $\mathcal{K}$ | Ranking function | 6.2.3 |
| $\kappa$ | number of agents | 2.1.4 |
| $\lambda$ | regularization param | 2.2.4 |
| $l$ | vector of L | 4.4.1 |
| $L$ | shared knowledge basis | 2.2.4 |
| $\mathcal{L}$ | loss function | 2.1.1 |
| $\mathscr{L}$ | Learning Problem | 2.2.3 |
| $\boldsymbol{m}$ | task descriptor | 4.1 |
| $M$ | mutual coherence | 4.4.1 |
| $\mathscr{M}$ | Performance metric | 2.2.3 |
| $\mu$ | mean | 6.1.1 |
| $\beth$ | sparsity regularization parameter on $s$ | 2.2.4 |
| $n$ | number of sample observations/rollouts | 3.1.1 |
| $N$ | number of neighbors in maximum coverage selection strategy | 6.2.3 |
| $\mathcal{N}$ | Normal distribution | 4.2 |
| $\eta$ | distance for maximum coverage selection strategy | 6.2.3 |
| $\nabla_i$ | Gradient with respect to $i$ | 2.2.2 |
| $O$ | big O notation for computational complexity | 6.2.3 |
| $\mathscr{O}$ | constraint function | 2.2.3 |
| $\pi$ | policy | 2.1.1 |
| $\Pi$ | set of safe policies | 7.2 |
| $p$ | probability | 2.1.1 |
| $P$ | transition probability | 2.1.1 |
| $\phi(\cdot)$ | feature extraction and transformation | 4.1 |
| $\psi$ | target heading angle for turtlebot | 3.3.1 |

| | | |
|---|---|---|
| $q$ | probability, used to differentiate with $p$ when there are multiple distributions | 2.1.2 |
| $Q$ | Q function for Q learning | 2.1.3 |
| $\rho$ | turtlebot's distance from goal | 3.3.1 |
| $\varrho$ | fractional scalar constants for decomposing $\sigma$ | 7.5 |
| $\vartheta$ | scalar multiplier for $\sigma$ | 7.5 |
| $r$ | reward | 2.1.1 |
| $R$ | average reward (for a trajectory) | 2.1.2 |
| $\mathfrak{R}$ | expected return | 2.1.1 |
| $\mathbb{R}$ | Reals | 2.1.1 |
| $\mathcal{R}$ | reward function | 2.1.1 |
| $\mathscr{R}$ | set of reward function | 2.1.1 |
| $s$ | sparse code | 2.2.4 |
| $\sigma$ | standard deviation | 3.3.1 |
| $\boldsymbol{\theta}$ | learnable parameters | 2.1.1 |
| $\boldsymbol{\tau}$ | trajectory | 2.1.1 |
| $\mathbb{T}$ | set of all trajectories | 2.2.5 |
| $t$ | task index, time as it relates to the task sequence | 2.2 |
| $T$ | the number of unique tasks encountered so far. | 2.2 |
| $\mathscr{T}$ | set of all tasks | 2.2.2 |
| $u$ | linear velocity | 3.3.1 |
| $U$ | Estimator | 6.1.1 |
| $\upsilon$ | learning rate | 2.2.2 |
| $\Upsilon$ | termination function | 2.2.1 |
| $v$ | for proof of TaDeLL | 4.4.1 |
| $\boldsymbol{V}$ | TaDeLL combined Hessian | 4.2 |
| $\nu$ | vector describing a constraint | 2.2.3 |
| $w$ | angular velocity | 3.3.1 |
| $\boldsymbol{w}$ | Dictionary element | 4.4.1 |
| $\boldsymbol{W}$ | Dictionary | 4.4.1 |
| $\boldsymbol{\omega}$ | disturbance | 3.1.1 |
| $\mathbf{x}$ | datapoint(supervised learning), state(RL) | 2.1.1 |
| $\boldsymbol{X}$ | matrix of samples | 2.2 |
| $\mathcal{X}$ | set of all states | 2.1.1 |
| $\chi$ | world state | 2.1.4 |
| $\mathscr{X}$ | set of world states | 2.1.4 |
| $\xi$ | heading angle offset for turtlebot | 3.3.1 |
| $\Xi$ | function used for computational complexity | 4.4.3 |
| $\boldsymbol{y}$ | labels for regression/classification | 2.2 |
| $\mathcal{Y}$ | collection of labels | 2.2.2 |

| | | |
|---|---|---|
| $\mathscr{Z}^{(t)}$ | task $t$ | 2.2 |
| $\mathscr{Z}$ | terminal reward | 2.2.1 |
| $\zeta$ | balances fit in TaDeLL | 4.2 |

# List of Abbreviations

Table 2: Table of Abbreviations

| Abbreviation | Meaning | First Occurance |
|---|---|---|
| BK | Bicycle | 4.5 |
| CP | Cart Pole | 4.5 |
| DQN | deep Q-network | 6.1.1 |
| ELLA | Efficient Lifelong Learning Algorithm | 2.2.4 |
| eNAC | episodic Natural Actor Critic | 4.2 |
| FIFO | first-in-first-out | 6 |
| FD | Finite Differences | 3.1 |
| iid | independent and identically distributed | 4.4.2 |
| IDM | Intelligent Driver Model | 7.7.2 |
| LWTA | local winner-take-all | 6.1 |
| MTL | multi-task learning | 2.2.4 |
| MDP | Markov Decision Process | 2.1 |
| NELL | Never-Ending Language Learner | 2.2.3 |
| PG | Policy Gradient | 2.1 |
| PG-ELLA | Efficient Lifelong Learning Algorithm | 2.2.5 |
| RL | reinforcement learning | 1.1 |
| ROS | Robot Operating System | 3.3 |
| SMDP | Semi-Markov Decision Process | 2.1.5 |
| SM | Spring Mass Damper | 4.5 |
| TaDeLL | Task Descriptors for Lifelong Learning | 4 |
| TaDeMTL | Task Descriptors for Multi-Task Learning | 4.5.5 |
| TD | temporal difference | 2.1.3 |

# Chapter 1

# Introduction

## 1.1 Problem Statement and Motivation

Any robotic system deployed outside a carefully engineered environment will encounter a large variety of operating conditions. As the complexity of the robotic system increases, it will become more difficult for engineers to anticipate and design for every possible circumstance. This motivates the design of robots that can learn from their environment and leverage that prior learning to modify their behavior on new challenges. To further motivate the importance of robots that are capable of lifelong learning, consider the following motivating examples:

**Disaster Relief** As a motivating example, consider the problem of disaster relief. In 2011, a magnitude 8.9 earthquake and tsunami hit Fukushima, Japan (BBC, 2011). One of the many facilities damaged by the earthquake was the Daiichi nuclear power

plant. For safety reasons, robots could have been invaluable in inspecting the area and addressing radiation leaks and chemical spills, however this technology was beyond the state-of-the-art. Broken doors and toppled obstacles would not only likely require a robot to re-route several times, but would also make re-routing a difficult problem solving task: which obstacles are movable, and how are they best manipulated? A tele-operated robot could outsource difficult aspects to the human operator[1], however, in disasters, it is frequently the case that communications lines go down and any remaining channels are often congested due to increased use, placing the problem largely in the domain of fully autonomous and adaptive agents.

**Planetary Exploration** Many challenges in disaster relief relate to difficulty in handling the unknown. However, operating in unknown environments is one of the most compelling use-cases for robotics. Unknown environments can be dangerous, difficult, or time consuming to explore and robots can be designed to handle the harsh and tedious conditions without the risk to human life. For example, a planetary exploration robot on a data collection mission may encounter new objects that are of scientific interest, but that its human programmers did not anticipate it would find. With a limited carrying capacity, the robot cannot collect every unidentifiable object that it encounters, so it will need to learn in-situ which objects are potentially of interest and which are not. Moreover, an ability to make unanticipated decisions might not only improve the effectiveness of a robot, but may be necessary to a robot's continued operation. In 2009, the NASA's Spirit rover became stuck in soft soil and when the programmers were not able to design successful maneuvers to escape the soil, the rover was lost (Jaggard, 2010). Had Spirit been able to learn how its behaviors

[1] A few years after the Fukushima earthquake, the DARPA robotics challenge (Pratt and Manzo, 2013) looked at addressing various disaster relief scenarios through tele-operation.

affected the mechanics of operating on different types of soil it might have been able to learn how to either avoid or escape the soil that trapped it.

**Repair** As evidenced in the case of the Mars rover, deployed robots are going to break down and the ways in which they break will often be difficult to predict. Another case where a robot capable of learning and adapting would be useful is in self-diagnosing failure and managing recovery. If a robot's leg or wheel stops working, the dynamics of motion will change. By identifying that the dynamics have changed and learning an altered gate or other corrective method for navigating in an impaired state, the robot could navigate to a facility for repairs.

**Bootstrapping and Curriculum** The previous motivating scenarios are blue sky ideas which will likely not be achievable for some time. While there are currently some examples of robots that can be deployed for extended periods of time (Biswas and Veloso, 2013), generally speaking, robots still need supervision. However, there are also more immediate gains available for a robotic system that can learn over time and leverage that accumulated knowledge. A progressively learning robot might be able to learn tasks that are too difficult or take too long to learn otherwise. A learning problem may be difficult for any number of reasons: a highly complex objective function with many poor local optima, a large cost to collect data, the presence of only sparse rewards with very long time delays for guiding learning, or a problem that is very difficult to specify formally may all be contributing factors that make a task difficult. One possible strategy for approaching these difficult problems is to use a lifelong learning agent. Existing knowledge can be bootstrapped to acquire new information that might be helpful in acquiring yet more information that ultimately

enables the system to solve the problem. An example of this is the use of a curriculum (Bengio et al., 2009; Narvekar et al., 2016): the robot starts by learning simple skills and then uses those skills to learn more advanced skills, eventually building up to complex behaviors.

**Sample Complexity** A large cost to collecting data was one reason mentioned for why a task might be difficult to learn. There may literally be a large cost associated with collecting data. This is the case when an expensive medical test needs to be conducted to predict the outcome of a treatment. On robotic systems, large costs might take the form of time spent training or wear on the physical components. By leveraging previously learned knowledge, a robot may be able to reduce the amount of training required to learn a task. If the savings are substantial enough, it can be the difference between whether or not the task is feasible for a robot to learn.

To move away from discussing knowledge and learning as abstract terms, I will focus on reinforcement learning (RL) specifically. Originally a component of operant conditioning in the field of psychology (Ferster and Skinner, 1957), RL is a learning process in which rewards are used to shape behavior. In computer science, it was shown that algorithms could be designed that couple exploratory behavior with environmental rewards (Sutton and Barto, 1998). These algorithms incrementally learn policies for a robot or artificial agent that maximize reward in that environment. In an RL context, *learning* is the process of finding a policy, and *knowledge* is the information (such as learned parameters) relating to the policy that generates behaviors.

The problem statement of this thesis is to identify how knowledge can be efficiently accumulated and shared to facilitate improved learning on future RL problems. This

is done specifically in the context of mobile robots. RL is appealing as a general learning framework because many tasks can be phrased as RL problems. However, RL is particularly difficult when training on physical systems, like mobile robots, because repeated trials can damage the system, and unrestricted exploration is often associated with safety risks. For this reason, I specifically focus on how knowledge from previous tasks can be used in order to reduce sample complexity and enable the safe learning of a robotic system in novel settings. In the next section, I elaborate on the problem statement of my thesis by identifying the fundamental questions that my work addresses.

## 1.2 Fundamental Questions

There are several fundamental questions related to the process of a robotic system learning over the course of its lifetime. Identifying these questions is a first step in developing systems capable of lifelong machine learning. By the end of the thesis, I will address the following fundamental questions:

- **How can a system continually accumulate prior knowledge and incorporate it into the learning process and behavior of future tasks?**

  A deployed robot can easily acquire terabytes of data in a single day. This presents challenges both in storing and accessing the data as needed. The raw data is in many cases highly compressible and perfect recall is likely not needed for the system to perform optimally. If the raw data is compressed, or only the model learned on the data is stored, it is likely that some of the original data will be lost. This raises the question of whether some experiences more important

than others, and if so, how can the important experiences be identified?

Assuming some experiences are more important than other experiences, it is *possible* that importance is an invariant of the experience, but it is more likely that the importance of an experience is relative to the system. Since we are designing the system to change over time, does the importance of an experience change over time, and if so, how can the stored data change accordingly?

Additionally, there is only partial knowledge as the system makes choices concerning the data. Since data arrives as a stream, any techniques used must function online with no knowledge of future data. What methods enable data to be accumulated in a fully online manner?

- **How effectively does the incorporation of knowledge from past reinforcement learning tasks reduce sample complexity of new tasks?**

  The rationale for incorporating prior knowledge is to improve performance. This improvement may come in the form of reducing the time to learn, or by increasing the system's ability to generalize and therefore perform better at test time. This raises a question of impact: how many experiences are needed to learn a new task?

  The fewer experiences a system needs to learn, the less resources need to be spent on the learning process. In the ideal case, the system could figure out how to successfully perform tasks it has never encountered. If this is possible, how can knowledge be transferred without any training data?

  There are also questions that can be asked concerning *what* should be transfered and how is that transfer carried out. Perhaps there are reasons why transferring one form of knowledge would be preferable. What changes if you transfer general vs. specific information for a task?

- **What novel situations can a robotic system handle as a result of its previously learned experience?**

  When a lifelong learning system is transferring knowledge to new situations, judging the effectiveness of the algorithm requires understanding how much the new situation differs from previous tasks. In the simplest case, transfer might improve a system's ability to generalize. A similar task is performed, and an improved model is learned by not over-fitting the noise. Better lifelong learners will be able to transfer their knowledge to a more diverse set of tasks. Targeting the greater capability of the latter, one could ask, "How can source robots with different dynamics be used to correct a disturbance on a target robot" This would involve learning some foundational relationship between the underlying dynamic systems.

  Transfer may take place on a single robot that moves between tasks, or the transfer might take place across different robots. For example, transfer can occur between simulated and real robots. What differences exist between transferring between tasks on a single robot platform and transferring knowledge between platforms?

The next section describes my technical approaches to answering these questions.

## 1.3 Technical Approach and Contributions

One of the contributing factors to the success of recent machine learning techniques has been the increasing ability to process large amounts of data. This helps break incidental correlations of the sampling process, and encourages general and robust

models. When large amounts of data are not available, it has been shown that using related data (Raina et al., 2007; Ham et al., 2005), or learning related tasks simultaneously (Caruana, 1997) can provide similar generalization.

The principal claim of my thesis is that sharing knowledge across tasks increases the capabilities and efficiency of training mobile robots. For my dissertation, I investigate various methods for accumulating knowledge, including both learned linear and non-linear latent spaces, and memory storage techniques. The accumulated knowledge is then shared with new tasks using techniques including sparse coding, task descriptors, and mappings to shared latent spaces. I present lifelong learning systems that enable learning in the presence of disturbances, systems without training data, systems with changing state and action distributions, and systems that need to adhere to safety constraints.

I approach lifelong RL by examining the way one task relates to another. In RL, which I describe in greater detail in Chapter 2.1, a learning problem takes the form of learning a policy that maps states to actions $\pi_{\boldsymbol{\theta}} : \mathbf{x} \rightarrow a$. For a single problem, we assume that each state is drawn from a distribution $\mathbf{x} \sim \mathscr{D}_t(\mathcal{X})$. Consider a robot that first encounters a problem with states drawn from $\mathscr{D}_t(\mathcal{X})$ and then encounters a second learning problem with states drawn from $\mathscr{D}_{t+1}(\mathcal{X})$, where both $\mathscr{D}_t(\mathcal{X})$ and $\mathscr{D}_{t+1}(\mathcal{X})$ are distributions that sample from a global distribution $\mathscr{D}_{global}(\mathcal{X})$. A first observation is that knowledge of $\mathscr{D}_t(\mathcal{X})$ and $\mathscr{D}_{t+1}(\mathcal{X})$ can be used to approximate $\mathscr{D}_{global}(\mathcal{X})$. The greater number of tasks observed, the more accurate that prediction will be. A second observation is that the optimal policy for a robot that first encounters task $\mathcal{Z}^{(t)}$ (corresponding to samples drawn from $\mathscr{D}_t(\mathcal{X})$) and then encounters task $\mathcal{Z}^{(t+1)}$ is not necessarily optimizing for $\mathscr{D}_t(\mathcal{X})$, $\mathscr{D}_{t+1}(\mathcal{X})$, or $\mathscr{D}_{global}(\mathcal{X})$. Rather, the robot needs to adjust so that it performs optimally on $\mathscr{D}_t(\mathcal{X})$ when the robot is faced with $\mathcal{Z}^{(t)}$,

and then performs optimally on $\mathscr{D}_{t+1}(\mathcal{X})$ when facing with $\mathcal{Z}^{(t+1)}$. This is a difficult problem, as it requires performing optimally on tasks that have not been trained on.

Given a robot in this setting, I investigate how $\mathscr{D}_{global}(\mathcal{X})$ can be approximated. I then show how knowledge of $\mathscr{D}_{global}(\mathcal{X})$, and various subsets of $\mathscr{D}_{global}(\mathcal{X})$, can be used to efficiently and safely switch to a new task on a robot system. While the target distribution for any given task may not be $\mathscr{D}_{global}(\mathcal{X})$, knowing $\mathscr{D}_{global}(\mathcal{X})$ is not useless. Just like large datasets prevent over-fitting noise, $\mathscr{D}_{global}(\mathcal{X})$ is a powerful prior that will likely be a *good* policy. Here *good* means both that the policy serves as a starting point that can learn an optimal policy quickly, and that the policy will perform reasonably well as the robot acquires data about its new task. Having experienced many prior tasks, the knowledge of those prior tasks can be *shared* with a new task to assist the learning process.

Often, in transfer, it is assumed that tasks are related, but when dealing with multiple tasks, it becomes important to have an approach that is more expressive at describing the nuanced relationships between different tasks. In the general case, tasks can either be related, unrelated, or conflict. Knowledge sharing for the general case must be done more selectively to prevent the negative transfer that occurs when tasks conflict. To show how knowledge can selectively be shared between tasks, I adopt the method of PG-ELLA(Bou Ammar et al., 2015), described in detail in Chapter 2.2.5, and apply it to robots. This method uses sparse coding to decompose policies, which allows the learning process to identify and share knowledge between similar tasks, while isolating that knowledge from being corrupted by dissimilar tasks.

My first contribution, is to show that the process can be applied to robotic systems. I demonstrate that PG-ELLA reduces the sample complexity and saves experimenters'

time. I also identify disadvantages specific to using PG-ELLA on robotic systems. While it is important to reduce the sample complexity of training on a robotic system, some trials are more dangerous than others. PG-ELLA generally focuses training the system on these more dangerous trials because transfer is done as a refinement to a partially learned policy, and each partially learned policy starts from random initializations which can produce unpredictable behavior.

Having identified this shortcoming, I present a new algorithm, TaDeLL, to address *how* knowledge is being transfered. TaDeLL uses sparse coding techniques which elegantly couple similar policies (and similar components of different policies) like PG-ELLA, while additionally predicting policies for new tasks. This allows general models to be learned that fit similar subregions of $\mathscr{D}_{global}(\mathcal{X})$, while *also* allowing those general models to be used as initializations. This provides a robotic system with a starting policy which is already near optimal and therefore safer and more predictable. Having such a policy allows a robot to quickly adapt and perform optimally on a new task when switching from task $\mathcal{Z}^{(t)}$ to task $\mathcal{Z}^{(t+1)}$. This is a desired property for fast switching between tasks alluded to earlier.

TaDeLL is a powerful model for transferring knowledge, but it is of limited use on many more complex robotics problems. If we actually hope to handle many tasks, we must have a base learner that is very general and flexible. RL meets this need. It is a general framework that accommodates a large family of problems, but the linear function approximation used for PG-ELLA and TaDeLL greatly limits the classes of problem that can be handled. Robotic learning problems will often have high-dimensional input spaces as a result of being outfitted with numerous high-dimensional sensors. High dimensional features are often less strongly correlated with the robot's state, which in practice makes policies both harder to learn, and less likely

to be represented well by a linear policy. This brings me to my next contribution — modifying a deep network to handle multiple tasks.

Deep networks are capable of representing arbitrary functions (Hornik et al., 1989) and it has been shown that an effective way to transfer knowledge in deep networks is fine-tuning (Razavian et al., 2014; Yosinski et al., 2014). Fine-tuning involves pre-training a network on one task and then retraining the network or *fine-tuning* it for a different task. It provides a good initialization for learning a new task, while preserving a deep network's ability to represent highly non-linear functions. I present several experiments that elucidate the behavior of fine-tuning deep networks, including studies of retention and the effects of training time on transferability. Unfortunately, one of the properties I demonstrate, is that deep networks *forget* previous tasks and are not immediately applicable to learning multiple tasks. To address this shortcoming, I contribute a limited memory buffer that can be used to preserve performance on multiple tasks.

By learning near optimal initializations, we can drastically minimize the number of dangerous trials. Machine learning techniques can rarely achieve 100% accuracy, especially without first failing and subsequently learning from those failures. However, on many robotic systems, certain states should *never* be visited. This is the case with autonomous driving, where collision cases should neither be observed nor required to learn.

The final contribution of my thesis, which allows RL policies to be safely run on an autonomous vehicle, shows how to incorporate prediction as a safety constraint into the learning process. Since the safety constraints are designed to be provided by an expert, or are initially trained as an offline task, the information is incorporated

without having to perform exploratory trials that will put the system at risk.

This section outlined my contribution of several frameworks for implementing learning algorithms. In the next section, I describe how these contributions are organized in my thesis.

## 1.4 Thesis Organization

Several aspects of this thesis have been previously published (Isele et al., 2016a; Isele and Cosgun, 2017b, 2018; Isele et al., 2016b, 2017, 2018a), with most of the work being expanded to include more in-depth experiments and analysis. The rest of thesis is organized as follows:

**Chapter 2 – Background** formally introduces reinforcement learning, Markov decision processes, and several popular reinforcement learning algorithms used in this thesis. Additionally, lifelong machine learning is discussed and several state-of-the-art methods are presented.

**Chapter 3 – Lifelong Reinforcement Learning on Mobile Robots** applies the PG-ELLA algorithm (Ruvolo and Eaton, 2013b; Bou Ammar et al., 2014) to the task of disturbance rejection on mobile robots. I show how knowledge can be shared between a collection of mobile robots, and identify pros and cons of using this approach on physical systems.

**Chapter 4 – Predicting Policies** presents my contribution TaDeLL, which couples task knowledge with task descriptions to predict models for novel tasks. By using a coupled dictionary that works with a sparse coding framework, I show that a new policy can be predicted using only a description of the task.

**Chapter 5 – Fine-Tuning as Transfer in Deep Learning** explores properties of fine-tuning deep networks, specifically probing aspects related to the online learning of multiple tasks.

**Chapter 6 – Preserving Experiences** describes a series of experiments that probe the selection processes related to storing experiences when learning multiple tasks. I identify a strategy that preserves memories over the lifetime of the system and can be used to prevent forgetting previously learned abilities while acquiring new skills.

**Chapter 7 – Safe Training** presents methods to enable the deep learning techniques to be applied on real robotic systems. I investigate how predictions can be used to provide safety constraints during the learning process.

**Chapter 8 – Conclusions and Contributions** revisits my research questions, answering them in the context of my thesis work. I then discuss future directions for this research before finally concluding.

# Chapter 2

# Background

A mobile robot that learns from past experiences, bootstrapping its knowledge to new situations, is a challenge under a variety of different machine learning paradigms. To make the investigation more tractable, I focus specifically on sharing knowledge when reinforcement learning is used as a base learner. This chapter presents background information on reinforcement learning, going into detail on policy gradient methods and Q-learning — two types of reinforcement learning algorithms used on robotic systems. Then a discussion of existing knowledge sharing and lifelong learning techniques is presented. This overview aims at giving a basic background while emphasizing the particular techniques which later chapters build upon.

## 2.1 Reinforcement Learning

*Reinforcement learning* (RL) (Sutton and Barto, 1998; Kober et al., 2013) is a popular method for autonomous learning, where a system seeks to select appropriate actions in order to optimize rewards. One of the main difficulties of performing this optimization is that actions should not only be chosen to optimize an immediate reward, but should be selected to position the system to receive future rewards that may only result after long sequences of actions. Since the reward of taking a good action might not manifest for many time steps, there is an inherent difficulty in assigning credit to good actions taken in the past. Additionally, as the path to high rewarding states is often not known in advance, trying to optimize in the absence of information can cause conflicting objectives. Does the system try to improve based on its current knowledge or does it try to gather more knowledge? The challenge of addressing the exploration versus exploitation trade-off has led to a variety of techniques in RL. I will first review the typical formalization of RL problems as a Markov Decision Processes (MDP) and then describe two popular RL techniques: policy gradients (PG) and Q-learning.

### 2.1.1 Markov Decision Processes

In the reinforcement learning framework, at time $h$ the system in state $\mathbf{x}_h \in \mathbb{R}^{d_x}$ takes an action $a_h$ according to the policy $\pi_\theta$ parameterized by $\boldsymbol{\theta}$. The agent transitions to the state $\mathbf{x}_{h+1}$, and receives a reward $r_h$. The sequence of states, actions, and rewards is given as a trajectory $\boldsymbol{\tau} = \{(\mathbf{x}_0, a_0, r_0), \ldots, (\mathbf{x}_H, a_H, r_H)\}$ over a horizon $H$. And the set associated with a single transition is an experience $e_h = (\mathbf{x}_h, a_h, r_h, \mathbf{x}_{h+1})$.

A reinforcement learning task is typically formulated as a Markov Decision Process (MDP) $\langle \mathcal{X}, \mathcal{A}, P, \mathcal{R}, \gamma \rangle$, where $\mathcal{X}$ is the set of states, and $\mathcal{A}$ is the set of actions that the agent may execute. The state transition function $P : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$ describes the systems dynamics, the reward function $\mathcal{R} : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathbb{R}$ gives the real-valued reward for a given time step, and $\gamma \in (0, 1]$ is a discount factor that adds preference for earlier rewards and provides stability in the case of infinite time horizons. MDPs follow the Markov assumption that the probability of transitioning to a new state given the current state and action is independent of all previous states and actions $p(\mathbf{x}_{h+1}|\mathbf{x}_h, a_h, \ldots, \mathbf{x}_0, a_0) = p(\mathbf{x}_{h+1}|\mathbf{x}_h, a_h)$.

The goal of RL is to find the optimal policy $\pi^*$ with parameters $\boldsymbol{\theta}^*$ that maximizes the expected return $\mathfrak{R} = \mathbb{E}\left[\sum_{h=0}^{H} \gamma^h r_h\right]$ over a sequence of actions. In some sections the term $-\mathfrak{R}$ is replaced with $\mathcal{L}$ denoting a loss function. While the focus is on reinforcement learning, many of the techniques developed are designed to be general enough to handle classification and regression tasks. The use of a loss function $\mathcal{L}$ accommodates this generalization. The next section describes several specific approaches to solving RL problems.

### 2.1.2 Policy Gradients

Learning on robots often requires working in high-dimensional state spaces, where the curse of dimensionality (Bellman, 1961) can make many optimization approaches intractable. One set of optimization strategies that lends itself to high-dimensional state spaces is policy gradient methods. Policy gradient (PG) methods (Sutton et al., 1999a) are a class of RL algorithms that are effective for solving high dimensional problems with continuous state and action spaces, such as robotic control (Peters and

Schaal, 2008). The goal of PG is to optimize the expected return $\mathfrak{R}(\boldsymbol{\theta})$:

$$\begin{aligned}
\mathfrak{R}(\boldsymbol{\theta}) &= \mathbb{E}\left[\sum_{h=0}^{H} \gamma^h r_h\right] = \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau} \ , \text{ where} \\
R(\boldsymbol{\tau}) &= \sum_{h=0}^{H} \gamma^h r_h \ , \text{ and} \\
p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) &= P_0(\mathbf{x}_0) \prod_{h=0}^{H} p(\mathbf{x}_{h+1} \mid \mathbf{x}_h, \boldsymbol{a}_h) \ \pi_\theta(\boldsymbol{a}_h \mid \mathbf{x}_h) \ .
\end{aligned}$$

Here $\mathbb{T}$ is the set of all possible trajectories, the reward on trajectory $\boldsymbol{\tau}$ is given by $R(\boldsymbol{\tau})$, and $p_{\boldsymbol{\theta}}(\boldsymbol{\tau})$ is the probability of $\boldsymbol{\tau}$ under initial state distribution $P_0 : \mathcal{X} \mapsto [0, 1]$. Most PG methods (for example episodic REINFORCE (Williams, 1992), PoWER (Kober and Peters, 2009), and Natural Actor Critic (Peters and Schaal, 2008)) optimize the policy by employing supervised function approximators to maximize a lower bound on the expected return. This optimization is carried out by generating trajectories using the current policy $\pi_{\boldsymbol{\theta}}$, and then comparing the result with a new policy $\pi_{\tilde{\boldsymbol{\theta}}}$. Jensen's inequality can then be used to lower bound the expected return (Kober and Peters, 2009):

$$\begin{aligned}
\log \mathfrak{R}\left(\tilde{\boldsymbol{\theta}}\right) &= \log \int_{\mathbb{T}} p_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \ R(\boldsymbol{\tau}) \, \mathrm{d}\boldsymbol{\tau} \\
&= \log \int_{\mathbb{T}} \frac{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})}{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \ p_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{\tau}) \ R(\boldsymbol{\tau}) \, \mathrm{d}\boldsymbol{\tau} \\
&\geq \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \ R(\boldsymbol{\tau}) \log \frac{p_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{\tau})}{p_{\boldsymbol{\theta}}(\boldsymbol{\tau})} \, \mathrm{d}\boldsymbol{\tau} + \text{constant} \\
&\propto -\mathfrak{D}_{\mathrm{KL}}(p_{\boldsymbol{\theta}}(\boldsymbol{\tau}) \ R(\boldsymbol{\tau}) \ || \ p_{\tilde{\boldsymbol{\theta}}}(\boldsymbol{\tau})) = \mathcal{J}\left(\tilde{\boldsymbol{\theta}}\right) \ ,
\end{aligned}$$

where

$$\mathfrak{D}_{\mathrm{KL}}(p(\boldsymbol{\tau}) \ || \ q(\boldsymbol{\tau})) = \int_{\mathbb{T}} p(\boldsymbol{\tau}) \log \frac{p(\boldsymbol{\tau})}{q(\boldsymbol{\tau})} \, \mathrm{d}\boldsymbol{\tau} \ .$$

This is equivalent to minimizing the KL divergence between the reward-weighted trajectory distribution of $\pi_{\boldsymbol{\theta}}$ and the trajectory distribution $p_{\tilde{\boldsymbol{\theta}}}$ of the new policy $\pi_{\tilde{\boldsymbol{\theta}}}$.

### 2.1.3 Q-learning

Policy gradient methods directly work to optimize the policy that governs behavior. An alternate approach to RL is Q-learning (Watkins and Dayan, 1992) which learns the value of taking a particular action at a given state. When the value of each state-action pair is known, an optimal policy falls out trivially. Q-learning defines an optimal action-value function $Q^*(\mathbf{x}, a)$ as the maximum expected return that is achievable following any policy given a state $\mathbf{x}_h$ and action $a_h$,

$$Q^*(\mathbf{x}, a) = \max_{\pi} \mathbb{E}\left[\sum_{h=\hat{h}}^{H} \gamma^{h-\hat{h}} r_h \mid \mathbf{x}_{\hat{h}} = \mathbf{x}, a_{\hat{h}} = a, \pi\right] . \tag{2.1}$$

This follows the dynamic programming properties of the Bellman equation, which state that if the values $Q^*(\mathbf{x}_{h+1}, a_{h+1})$ are known for all $a_{h+1}$ then the Q-value of an experience $e_h$ can be calculated by selecting the action $a_{h+1}$ that maximizes the expected value of $r_h + \gamma Q^*(\mathbf{x}_{h+1}, a_{h+1})$:

$$Q^*(\mathbf{x}, a) = \mathbb{E}[r_h + \gamma \max_{a_{h+1}} Q^*(\mathbf{x}_{h+1}, a_{h+1}) \mid \mathbf{x}_h = \mathbf{x}, a_h = a] . \tag{2.2}$$

In Deep Q-learning (Mnih et al., 2013), the optimal value function is approximated with a neural network $Q^*(\mathbf{x}, a) \approx Q(\mathbf{x}, a; \boldsymbol{\theta})$. The parameters $\boldsymbol{\theta}$ are learned by using the Bellman equation as an iterative update

$$Q_{i+1}(\mathbf{x}_h, a_h) = \mathbb{E}[r_h + \gamma \max_{a_{h+1}} Q_i(\mathbf{x}_{h+1}, a_{h+1}) \mid \mathbf{x}_h, a_h] , \tag{2.3}$$

and minimizing the temporal difference (TD) error between the expected return and the state-action value predicted by the network. This gives the loss for an individual experience in a deep Q-network (DQN)

$$\mathcal{L}(e_h, \boldsymbol{\theta}) = \left( r_h + \gamma \max_{a_{h+1}} Q(\mathbf{x}_{h+1}, a_{h+1}; \boldsymbol{\theta}) - Q(\mathbf{x}_h, a_h; \boldsymbol{\theta}) \right)^2 . \tag{2.4}$$

Given a learned Q-function, at every time step the optimal policy executes the action that has the maximum value given the current state. During learning, an $\epsilon$-greedy policy is often followed by selecting a random action with probability $\epsilon$ to promote exploration and otherwise greedily selecting the best action $max_a Q(\mathbf{x}, a; \boldsymbol{\theta})$ according to the current Q-function.

The next sections describe extensions to the MDP framework that accommodate domains with multiple agents and handle learnable skills that allow for hierarchical actions.

### 2.1.4 Stochastic Games

Stochastic games generalize MDPs to handle multiple agents. Since other agents can change their policies over time, the distribution of states is non-stationary, which forces the agent not just to learn but also to adapt. In a stochastic game, at time $h$ each agent $i$ in the world state $\chi_h$ takes an action $a_h^i$ according to the policy $\pi^i$. All the agents then transition to the state $\chi_{h+1}$ and receive a reward $r_h^i$. Stochastic games can be described by the tuple $\langle \mathscr{X}, \mathscr{A}, P, \mathscr{R} \rangle$, where $\mathscr{X}$ is the set of world states, and $\mathscr{A} = \{\mathcal{A}^1, \ldots, \mathcal{A}^\kappa\}$ is the joint action space consisting of the set of each agent's actions, where $\kappa$ is the number of agents. The reward functions $\mathscr{R} = \{\mathcal{R}^1, \ldots, \mathcal{R}^\kappa\}$

describe the reward for each agent $\mathscr{X} \times \mathscr{A} \times \mathscr{X} \rightarrow \mathbb{R}^{\kappa}$. The transition function $P : \mathscr{X} \times \mathscr{A} \rightarrow \mathscr{X}$ describes how the state evolves in response to all the agents' collective actions. Stochastic games are an extension to MDPs which generalize to multiple agents, each of which has its own policy and reward function.

### 2.1.5 Options

In order to scale RL to handle complex tasks over long time horizons, it is helpful to group sets of actions that are often repeated in sequence into a single high-order action or *skill*. Skills can also be grouped, allowing for hierarchical action learning. Learning with skills (also referred to as *options* in the literature) is formalized as a Semi-Markov Decision Process (SMDP) (Sutton et al., 1999b; Precup, 2000). The MDP, as defined earlier, forms the basis of the SMDP, which formalizes temporally extended actions as a Markovian option æ$\in$ Æ, where Æ is the triple $\langle \mathscr{I}_{æ}, \pi_{æ}, \beta_{æ} \rangle$, where $\mathscr{I}_{æ} \subseteq \mathcal{X}$ is the initiation set, $\pi_{æ}$ is the intra-option policy, and $\beta_{æ} : \mathcal{X} \rightarrow [0, 1]$ is the termination function.

The next sections describe how an RL problem can be incorporated into a lifelong learning setting, where an agent learns many tasks over the course of its lifetime.

## 2.2 Lifelong Machine Learning

In RL, a system explores and subsequently learns from its own experience. This draws parallels to the lifelong learning problem — past experiences are used to shape behavior and improve learning. A system that continues to accumulate knowledge

over the course of its lifetime has been called lifelong machine learning (Thrun, 1996; Ruvolo and Eaton, 2013b), continual learning (Ring, 1998; Rafols et al., 2005), and never-ending learning (Mitchell et al., 2015) by different groups of researchers. In this thesis, I will use lifelong machine learning or lifelong learning for short when it is clear from context that the learning relates to machines.

In a lifelong machine learning setting, a learner faces multiple, consecutive tasks and must rapidly learn each new task by building upon its previous experience. The learner may encounter a previous task at any time, and so must optimize performance across all tasks seen so far. A priori, the agent does not know the total number of tasks $T_{\max}$, the task distribution, or the task order.

At time $t$, the lifelong learner encounters task $\mathcal{Z}^{(t)}$. A task can take many forms; for example, a task may be a regression problem $\mathcal{Z}^{(t)} = \langle \boldsymbol{X}^{(t)}, \boldsymbol{y}^{(t)} \rangle$, or a classification problem $\mathcal{Z}^{(t)} = \langle \boldsymbol{X}^{(t)}, \boldsymbol{y}^{(t)} \rangle$. In this thesis, I focus on RL tasks where $\mathcal{Z}^{(t)}$ is specified by an MDP $\langle \mathcal{X}, \mathcal{A}, P, \mathscr{R}, \gamma \rangle$. I do not consider mixed learning paradigms—if a lifelong learning agent learns reinforcement learning tasks it will only face reinforcement learning problems during its lifetime. The agent will learn each task consecutively, acquiring training data (i.e., trajectories or samples) in each task before advancing to the next. The agent's goal is to learn the optimal models $\{f^*_{\boldsymbol{\theta}^{(1)}}, \ldots, f^*_{\boldsymbol{\theta}^{(T)}}\}$ or policies $\{\pi^*_{\boldsymbol{\theta}^{(1)}}, \ldots, \pi^*_{\boldsymbol{\theta}^{(T)}}\}$ with corresponding parameters $\{\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(T)}\}$, where $T$ is the number of unique tasks seen so far ($1 \leq T \leq T_{\max}$). Ideally, knowledge learned from previous tasks $\{\mathcal{Z}^{(1)}, \ldots, \mathcal{Z}^{(T-1)}\}$ should accelerate training and improve performance on each new task $\mathcal{Z}^{(T)}$. Also, the lifelong learner should scale effectively to large numbers of tasks, learning each new task rapidly from minimal data.

### 2.2.1 HORDE

The Horde architecture (Sutton et al., 2011) generalizes value function approximation techniques from reinforcement learning to span across multiple learners (*demons* in the authors' verbage) with differing objectives which collectively amass the knowledge of a shared experience stream. Each target policy is learned off-policy from the experiences generated by the behavior policy. The behavior policy may be a policy unrelated to the different demons being learned, or it might alternate between different behavior policies of demons in the horde.

For example, the behavior policy may be a policy designed to explore the environment, one demon might be using the experience to learn motion primitives, a second demon might be learning to navigate to a specific goal, a third demon might be learning to predict collisions, and a final demon might be learning to predict changes in the lighting conditions of the environment. The idea behind Horde is that by using off-policy learning, many different value functions (policies) can be learned from the same behavior.

Concretely, each demon corresponds to a generalized value function $Q : X \times A \to \mathbb{R}$. While it's common to denote $Q$ as $Q^\pi$ since the value is conditioned on the policy, the generalized value function would be more accurately written $Q^{\pi,\Upsilon,\mathscr{R},\mathscr{Z}}$ since it is conditioned on four auxiliary *question functions*: a policy function $\pi : X \times A \to [0, 1]$, a termination function $\Upsilon : X \to [0, 1]$, a per step reward function $\mathscr{R} : X \to \mathbb{R}$, and terminal reward $\mathscr{Z} : X \to \mathbb{R}$.

A value function can be used to describe a policy (repeatedly choosing the actions that maximizes a value produces a behavior) so a value function is often thought

of as being synonymous with a policy. However the Horde framework points out that a value function is more general. Value functions *may* be related to behavior functions that guide an agent for a particular task, but they can also be used to make predictions about the environment. For example, the value can learn an answer to the question, "If I'm in this state and take this action, how many steps should I expect to take before I crash?" With each different value function learning a different aspect of the environment, the collective Horde has a rich understanding of the world informed by the perspectives of each individual demon.

### 2.2.2 Meta-Learning

The framework of learning to learn (or meta learning) considers the problem of learning how to model an agent that is capable learning. In the work of Finn et al. (2017), a learner is exposed to many tasks, during which time the learner learns a model that maximizes the performance on each task after a single gradient update. Formally, on task $t$ the model parameters are

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta} - \upsilon \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathcal{Z}^{(t)}}(\boldsymbol{\theta}) \ , \tag{2.5}$$

where $\boldsymbol{\theta}$ is the global parameters being optimized. The meta-objective is then defined

$$\min_{\boldsymbol{\theta}} \sum_{\mathcal{Z}^{(t)} \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{Z}^{(t)}}(\boldsymbol{\theta}^{(t)}) \ . \tag{2.6}$$

This process learns a model that can be quickly adjusted, enabling one-shot learning.

### 2.2.3 Never-Ending Language Learner

Never Ending Learning (Mitchell et al., 2018) provides a very general framework for an agent that learns over time. In the never ending learning framework, a never-ending learning problem $\mathscr{L}$ consist of set of tasks $\mathscr{T}$ and constraints $\mathscr{C}$,

$$\mathscr{L} = (\mathscr{T}, \mathscr{C}) \ . \tag{2.7}$$

Each task $\mathscr{Z}^{(t)} \in \mathscr{T}$ consists of a pair describing the input and output space $\langle \mathcal{X}_t, \mathcal{Y}_t \rangle$, a performance metric $\mathscr{M}_t$, and a collection of experiences $E_t$,

$$\mathscr{T} = \{\langle \mathscr{Z}^{(t)}, \mathscr{M}_t, E_t \rangle\} \ . \tag{2.8}$$

The performance metric $\mathscr{M}_t : f \to \mathbb{R}$ defines the optimal learned function $f_t^* : \mathcal{X}_t, \to \mathcal{Y}_t$ for the $t^{th}$ learning task:

$$f_t^* \equiv \underset{f}{\operatorname{argmax}} \mathscr{M}_t(f) \ . \tag{2.9}$$

The coupling constraints $\mathscr{C}$ consist of a coupling function $\mathscr{O}_k$, which specifies how well the constraints are satisfied between the solutions of multiple learning tasks, and a vector of indices $\nu_k$ specifying the inputs to $\mathscr{O}_k$:

$$\mathscr{C} = \{\langle \mathscr{O}_k, \nu_k \rangle\} \ . \tag{2.10}$$

The example the authors provide is the Never Ending Language Learner (NELL)

(Mitchell et al., 2018) which applies the Never Ending Learning Framework to the problem of language understanding. The system is provided a set of categories (e.g., Sport, Athlete), a set of binary relations (e.g., AthletePlaysSport(x,y)), a small number of labeled training examples for each category and relation (e.g. Sport includes the noun phrases baseball and soccer), the internet (represented as an initial 500 million web pages from the ClueWeb 2009 collection (Callan et al., 2009), and access to 100,000 Google API search queries each day), and occasional interaction with humans (through NELL's public website http://rtw.ml.cmu.edu).

The goal is to endlessly learn new data from the internet. This takes the form of classifying new members of categories, and identifying new examples of binary relations. Additionally, the learner identifies new categories and binary relations, increasing confidences of previous beliefs, and removing incorrect beliefs.

### 2.2.4 Efficient Lifelong Learning Algorithm

One particular lifelong learning approach which has been shown to be quite versatile and which my thesis work builds upon is the Efficient Lifelong Learning Algorithm (ELLA) proposed by Ruvolo and Eaton (2013b).

ELLA assumes the parameters for each task model can be factorized using a shared knowledge base $\boldsymbol{L}$, facilitating transfer between tasks. Specifically, the model parameters for task $\mathcal{Z}^{(t)}$ are given by $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$, where $\boldsymbol{L} \in \mathbb{R}^{d \times k}$ is a shared basis over the model space, and $\boldsymbol{s}^{(t)} \in \mathbb{R}^k$ are the sparse coefficients over the basis. This factorization, depicted in Figure 2.1, has been effective for transfer in both lifelong and multi-task learning (Kumar and Daumé, 2012; Maurer et al., 2013).

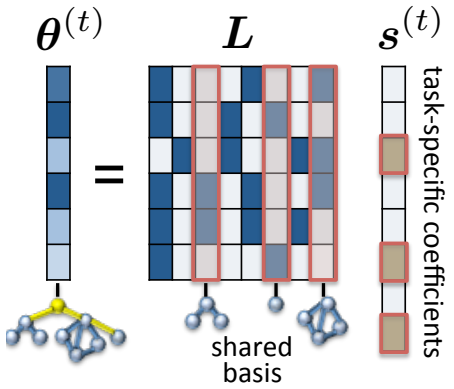



Figure 2.1: The task-specific model (or policy) parameters $\boldsymbol{\theta}^{(t)}$ are factored into a shared knowledge repository $\boldsymbol{L}$ and a sparse code $\boldsymbol{s}^{(t)}$. The repository $\boldsymbol{L}$ stores chunks of knowledge that are useful for multiple tasks, and the sparse code $\boldsymbol{s}^{(t)}$ extracts the relevant pieces of knowledge for a particular task's model (or policy).

Under this assumption, the multi-task learning (MTL) objective is:

$$\min_{\boldsymbol{L},\boldsymbol{S}} \frac{1}{T} \sum_{t=1}^{T} \left[ \mathcal{L}(\boldsymbol{\theta}^{(t)}) + \mu \|\boldsymbol{s}^{(t)}\|_1 \right] + \lambda \|\boldsymbol{L}\|_{\mathsf{F}}^2 \ , \tag{2.11}$$

where $\boldsymbol{S} = [\boldsymbol{s}^{(1)} \cdots \ \boldsymbol{s}^{(T)}]$ is the matrix of sparse vectors, $\mathcal{L}$ is the task-specific loss for task $\mathcal{Z}^{(t)}$, and $\|\cdot\|_{\mathsf{F}}$ is the Frobenius norm. The $L_1$ norm is used to approximate the true vector sparsity of $\boldsymbol{s}^{(t)}$, and $\mu$ and $\lambda$ are regularization parameters. Note that for a convex loss function $\mathcal{L}(\cdot)$, this problem is convex in each of the variables $\boldsymbol{L}$ and $\boldsymbol{S}$. Thus, one can use an alternating optimization approach to solve it in a batch learning setting. To solve this objective in a lifelong learning setting, Ruvolo and Eaton (2013b) take a second-order Taylor expansion to approximate the objective around an estimate $\boldsymbol{\alpha}^{(t)} \in \mathbb{R}^d$ of the single-task model parameters for each task $\mathcal{Z}^{(t)}$, and update only the coefficients $\boldsymbol{s}^{(t)}$ for the current task at each time step. This process reduces the MTL objective to the problem of sparse coding the single-task policies in the shared basis $\boldsymbol{L}$, and enables $\boldsymbol{S}$ and $\boldsymbol{L}$ to be solved efficiently by the following alternating online update rules that constitute ELLA (Ruvolo and Eaton,

2013b):

$$\boldsymbol{s}^{(t)} \leftarrow \arg\min_{\boldsymbol{s}} \|\boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s}\|^2_{\boldsymbol{\Gamma}^{(t)}} + \mu\|\boldsymbol{s}\|_1 \tag{2.12}$$

$$\boldsymbol{A} \leftarrow \boldsymbol{A} + (\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\top}) \otimes \boldsymbol{\Gamma}^{(t)} \tag{2.13}$$

$$\boldsymbol{b} \leftarrow \boldsymbol{b} + \text{vec}\left(\boldsymbol{s}^{(t)\top} \otimes \left(\boldsymbol{\alpha}^{(t)\top}\boldsymbol{\Gamma}^{(t)}\right)\right) \tag{2.14}$$

$$\boldsymbol{L} \leftarrow \text{mat}\left(\left(\frac{1}{T}\boldsymbol{A} + \lambda\boldsymbol{I}_{kd}\right)^{-1}\frac{1}{T}\boldsymbol{b}\right) \ , \tag{2.15}$$

where $\|\boldsymbol{v}\|^2_{\boldsymbol{A}} = \boldsymbol{v}^\top\boldsymbol{A}\boldsymbol{v}$, the symbol $\otimes$ denotes the Kronecker product, $\boldsymbol{\Gamma}^{(t)}$ is the Hessian of the loss $\mathcal{L}(\boldsymbol{\alpha}^{(t)})$, $\boldsymbol{I}_m$ is the $m \times m$ identity matrix, $\boldsymbol{A}$ is initialized to a $kd \times kd$ zero matrix, and $\boldsymbol{b} \in \mathbb{R}^{kd}$ is initialized to zeros. Here $\boldsymbol{A}^{-1}\boldsymbol{b}$ forms a column-wise vectorization of $\boldsymbol{L}$ in order to allow $\boldsymbol{L}$ to be constructed incrementally, reducing the computational complexity of the update.

ELLA was designed for classification and regression tasks. The next section describes how ELLA was extended to RL by Bou Ammar et al. (2014).

### 2.2.5 Policy Gradient Efficient Lifelong Learning Algorithm

Bou Ammar et al. (2014) showed that ELLA can be extended to handle policy gradient reinforcement learning. The resulting algorithm is named the Policy Gradient Efficient Lifelong Learning Algorithm (PG-ELLA). By letting $\boldsymbol{\theta}^{(t)}$ describe the parameters of a Gaussian policy and assuming the same factorization as ELLA: $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$,

the multi-task objective for RL tasks becomes:

$$\underset{\boldsymbol{L},\boldsymbol{S}}{\operatorname{argmin}} \frac{1}{T} \sum_{t} \left[ -\mathcal{J}\left(\boldsymbol{\theta}^{(t)}\right) + \text{מ} \left\| \boldsymbol{s}^{(t)} \right\|_1 \right] + \lambda \|\boldsymbol{L}\|_{\mathsf{F}}^2 \ , \tag{2.16}$$

where $\boldsymbol{S} = \left[\boldsymbol{s}^{(1)} \cdots \boldsymbol{s}^{(T)}\right]$ is the matrix of all coefficients, the L1 norm $\|\cdot\|_1$ enforces sparsity of the coefficients, and the Frobenious norm $\|\cdot\|_{\mathsf{F}}$ regularizes the complexity of $\mathbf{L}$ with regularization parameters $\text{מ}, \lambda \in \mathbb{R}$. PG-ELLA solves (3.4) by first approximating the RL multi-task objective by substituting in the lower-bound of the PG objective $\mathcal{J}(\boldsymbol{\alpha}^{(t)})$ in order to transform the expected return into a convex term. The lower bound is then approximated with a second-order Taylor expansion around an estimate $\boldsymbol{\alpha}^{(t)}$ of the single-task policy parameters for task $\mathcal{Z}^{(t)}$. Finally, to prevent the computation cost of the update from increasing linearly with the number of tasks, PG-ELLA optimizes $\boldsymbol{s}^{(t)}$ only when training on task $\mathcal{Z}^{(t)}$. These steps reduce the learning problem to a series of on-line update equations that constitute PG-ELLA (Bou Ammar et al., 2014) given as Algorithm 1. Note that $\mathbb{T}$ is the set of all trajectories.

The next section shows how PG-ELLA and other lifelong learning algorithms compare to each other. In this comparison, I show where the contribution of my thesis fit in with the current state of the art.

### 2.2.6 Comparisons

There are advantages and disadvantages to the existing state-of-the-art lifelong reinforcement learning approaches. In this thesis I build upon the existing approaches, presenting new algorithms with increased capabilities. Table 2.1 provides a compari-

**Algorithm 1** PG-ELLA $(k, \lambda, \text{מ})$ (Bou Ammar et al. 2014)

$T \leftarrow 0$
$\mathbf{A} \leftarrow \mathbf{zeros}_{k\times d,k\times d}, \quad \mathbf{b} \leftarrow \mathbf{zeros}_{k\times d,1}$
$\boldsymbol{L} \leftarrow \text{RandomMatrix}_{d,k}$
**while** some task $\mathcal{Z}^{(t)}$ is available **do**
  **if** isNewTask($\mathcal{Z}^{(t)}$) **then**
    $T \leftarrow T + 1$
    $\left(\mathbb{T}^{(t)}, R^{(t)}\right) \leftarrow \text{getRandomTrajectories}()$
  **else**
    $\left(\mathbb{T}^{(t)}, R^{(t)}\right) \leftarrow \text{getTrajectories}\left(\boldsymbol{\alpha}^{(t)}\right)$
    $\mathbf{A} \leftarrow \mathbf{A} - \left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\mathsf{T}}\right) \otimes \boldsymbol{\Gamma}^{(t)}$
    $\mathbf{b} \leftarrow \mathbf{b} - \text{vec}\left(\boldsymbol{s}^{(t)\mathsf{T}} \otimes \left(\boldsymbol{\alpha}^{(t)\mathsf{T}}\boldsymbol{\Gamma}^{(t)}\right)\right)$
  **end if**
  Compute $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\Gamma}^{(t)}$ from $\mathbb{T}^{(t)}$ using PG
  $\boldsymbol{s}^{(t)} \leftarrow \arg\min_{\boldsymbol{s}} \left\|\boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s}^{(t)}\right\|^2_{\boldsymbol{\Gamma}^{(t)}} + \text{מ}\,\|\boldsymbol{s}\|_1$
  $\mathbf{A} \leftarrow \mathbf{A} + \left(\boldsymbol{s}^{(t)}\boldsymbol{s}^{(t)\mathsf{T}}\right) \otimes \boldsymbol{\Gamma}^{(t)}$
  $\mathbf{b} \leftarrow \mathbf{b} + \text{vec}\left(\boldsymbol{s}^{(t)\mathsf{T}} \otimes \left(\boldsymbol{\alpha}^{(t)\mathsf{T}}\boldsymbol{\Gamma}^{(t)}\right)\right)$
  $\mathbf{L} \leftarrow \text{mat}\left(\left(\frac{1}{T}\mathbf{A} + \lambda\mathbf{I}_{k\times d,k\times d}\right)^{-1}\frac{1}{T}\mathbf{b}\right)$
  **for** $t \in \{1, \ldots, T\}$ **do**: $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{L}\boldsymbol{s}^{(t)}$
**end while**

son of the different approaches. My contributions are highlighted in bold.

Table 2.1: Comparison of Lifelong Learning Algorithms

| Method | Diversity | Efficiency | Safety | Preserving | Non-linear | On Robots |
|---|---|---|---|---|---|---|
| HORDE (Sutton et al., 2011) | Handles a large variety of learning objectives | Demons learn in parallel, but each demon learns from scratch and no knowledge sharing takes place. | No | No | No | Yes |
| PG-ELLA (Bou Ammar et al., 2014) | Demonstrated on different agents with the same objective. | Efficiently stores large numbers of policies. Policies are (partially) learned from scratch and then improved with shared knowledge. | No | Yes, each policy is stored and improved | No | **Yes (Isele et al., 2016a). Presented in Chapter 3** |
| **TaDeLL (Isele et al., 2016b) Presented in Chapter 4** | Demonstrated on different agents with the same objective. | Efficiently stores large numbers of policies. New policies can be predicted from a description. | Zero-shot learning skips over the earliest (most dangerous) part of learning | Yes, each policy is stored and improved | No | In simulation |
| Meta Learning (Finn et al., 2017) | Demonstrated on multiple tasks for a single agent. | A trained system (requiring many tasks) can perform one-shot learning. | One-shot learning helps skip over earliest (most dangerous) part of learning | The updates to each task could be stored, but would be costly. | Yes | In simulation |
| **Selective Experience Replay (Isele and Cosgun, 2018) Presented in Chapter 6** | Demonstrated on multiple tasks for a single agent. | Uses transfer to jump start performance on new tasks. | Can skip over the earliest (most dangerous) part of learning. **Can be combined with safe RL (Isele et al., 2018a) Presented in Chapter 7.** | Yes, preserves ability on previous tasks | Yes | In simulation |

# Chapter 3

# Lifelong Reinforcement Learning on Mobile Robots

The lifelong learning method ELLA (Ruvolo and Eaton, 2013b), described in section 2.2.4, sequentially receives and incorporates task information into a knowledge repository, sharing knowledge between all tasks. As described in section 2.2.5, this method has been extended to RL (Bou Ammar et al., 2015) to accelerate learning of the control policies for a variety of dynamical systems using PG methods (Sutton et al., 1999a; Williams, 1992). To understand how a state-of-the art lifelong learning algorithm can be applied to mobile robots, this chapter applies PG-ELLA to robotic control problems in simulation and on real systems.

PG-ELLA has only previously been applied to benchmark problems with known dynamics to demonstrate knowledge sharing, and not yet to more complex robotic control problems. My work significantly scales up the complexity of experiments by applying lifelong learning techniques to a set of Turtlebot 2 (Turtlebot 2, 2016) and

AR.Drone (Parrot AR.Drone 2, 2016) robots, each with their own control disturbances, in both the high-fidelity Gazebo simulator and on real robotic platforms. As such, this work represents a milestone in validating a prominent lifelong learning approach on physical robotic platforms, while also exposing new challenges to using lifelong learning techniques on physical systems.

In this work, different tasks correspond to control problems on different robots. No two robots are exactly the same—even for a given model of robot, different units will require slightly different controllers. This work leverages lifelong learning in order to learn controllers for different robots. In particular, it is shown that by learning a set of control policies over robots with different (unknown) motion models, a controller for a new robot with a unique set of disturbances can be learned more quickly. The approach is completely model-free, allowing us to apply this method to robots that have not, or cannot, be fully modeled.

RL is often used to learn controllers in a model-free setting. Among RL algorithms, policy gradient methods are popular in robotic applications (Kober and Peters, 2009; Peters and Schaal, 2008) since they accommodate continuous state/action spaces and can scale well to high-dimensional problems. The goal of lifelong learning is to learn a set of policies from consecutive tasks. By leveraging similarities between the tasks, it should be possible to learn the set of tasks much faster than if each task was learned independently. Previous work showed that lifelong learning could successfully leverage policy gradient methods (Bou Ammar et al., 2015), but had been applied only to simple dynamical systems and not more complex robotic control problems. There have been some successful examples of lifelong learning on robots, but and with few exceptions (White et al., 2012), they have mostly focused on skill refinement for a single robot (Kleiner et al., 2002; Thrun and Mitchell, 1995) and have not looked

at sharing information across multiple tasks.

Each different robot in our problem has a different unknown disturbance. In *disturbance rejection* problems, a controller is designed to complete a task while compensating for a disturbance that modifies its nominal dynamics. As long as there is an accurate model of the robot, current methods can handle constant, time-varying, and even stochastic disturbances (Khalil, 1996; Lewis and Syrmos, 2012; Dorato et al., 1994). However, such methods are generally inapplicable when the robot model is unknown, even if the disturbances are relatively simple. This work is motivated by control theory approaches, but focuses on leveraging model-free RL techniques.

## 3.1 Background

This approach uses PG RL as described in Section 2.1.2. PG is selected for this problem, because PG methods are well-suited for solving high dimensional problems with continuous state and action spaces, including robotic control (Peters and Schaal, 2008). Recall that the goal of PG is to use gradient steps to optimize the expected average return, given by

$$\mathfrak{R}(\boldsymbol{\theta}) = \int_{\mathbb{T}} p_{\boldsymbol{\theta}}(\tau) R(\tau) \mathrm{d}\tau \ .$$

Most PG methods (*e.g.,* episodic REINFORCE (Williams, 1992), Natural Actor Critic (Peters and Schaal, 2008), and PoWER (Kober and Peters, 2009)) optimize the policy

by maximizing a lower bound on the return, comparing trajectories generated by different candidate policies $\pi$. In this particular application, the PG method used in our experiments is finite differences (FD) (Kober et al., 2013) which optimizes the return directly.

### 3.1.1 Finite Differences for Policy Search

The FD method (Kober et al., 2013), which has shown past success in robotic control, optimizes the policy $\pi_{\boldsymbol{\theta}}$ directly by computing small changes $\boldsymbol{\Delta\theta}$ in the policy parameters that will increase the expected reward. This process estimates the expected return for each policy parameter variation $(\boldsymbol{\theta}_j + \boldsymbol{\Delta\theta}_i)$ given the sampled trajectories via

$$\Delta\hat{\mathfrak{R}}_{\mathbf{i}} = R(\boldsymbol{\theta}_j + \boldsymbol{\Delta\theta}_i) - \mathfrak{R}_{ref} \ , \tag{3.1}$$

where the estimate is taken over $n$ small perturbations in the policy parameters $\{\boldsymbol{\Delta\theta}_i\}_{i=1}^{n}$. The policy parameters at training iteration $j$ are given by $\boldsymbol{\theta}_j$, and $\mathfrak{R}_{ref}$ is a reference return, which is usually taken as the return of unperturbed parameters $\mathfrak{R}(\boldsymbol{\theta})$. The FD gradient method then updates the policy parameters, following the gradient of the expected return $\mathfrak{R}$ with a learning rate $\upsilon$, as given by

$$\boldsymbol{\theta}_{j+1} = \boldsymbol{\theta}_j + \upsilon\nabla_{\boldsymbol{\theta}}\mathfrak{R} \ . \tag{3.2}$$

For efficiency, the gradient $\nabla_{\boldsymbol{\theta}}\mathfrak{R}$ is estimated using the pseudoinverse

$$\nabla_{\boldsymbol{\theta}}\mathfrak{R} \approx \left(\boldsymbol{\Delta\Theta}^{\mathbf{T}}\boldsymbol{\Delta\Theta}\right)^{-1}\boldsymbol{\Delta\Theta}^{\mathbf{T}}\Delta\hat{\mathfrak{R}} \ , \tag{3.3}$$

where $\Delta\hat{\mathfrak{R}}$ contains all the stacked samples of $\Delta\hat{\mathfrak{R}}_{\mathbf{i}}$ and $\boldsymbol{\Delta\Theta}$ contains the stacked perturbations $\boldsymbol{\Delta\theta}_i$. This approach is sensitive to the type and magnitude of the perturbations, as well as to the learning rate $\upsilon$. Since the number of perturbations needs to be as large as the number of parameters, this method can be noisy and inefficient for problems with large sets of parameters (Kober et al., 2013), although it was found to work well and reliably in our setting.

The process is capable of optimizing a policy for a single RL task via repeatedly sampled trajectories, *i.e.,* $n$ trajectories for each $j \in \{1, \ldots, J\}$ iteration. In order to share information between different policies that are learned consecutively, the PG learning process using FD is incorporated into a lifelong learning setting, as described next.

## 3.2 Lifelong Learning with Policy Gradients for Disturbance Rejection

Section 2.2.5 described an efficient algorithm for lifelong learning with policy gradients known as PG-ELLA (Bou Ammar et al., 2014). I apply PG-ELLA to the multi-robot setting using FD methods as the base learner. Recall that the lifelong learner's goal is optimizing the policies for all known tasks and can be formulated as the following multi-task objective:

$$\underset{\boldsymbol{L},\boldsymbol{S}}{\operatorname{argmin}} \frac{1}{T} \sum_{t} \left[ -\mathfrak{R}\left(\boldsymbol{\theta}^{(t)}\right) + \mu \left\| \boldsymbol{s}^{(t)} \right\|_1 \right] + \lambda \|\boldsymbol{L}\|_{\mathsf{F}}^2 \ , \tag{3.4}$$

where $\boldsymbol{S} = \left[\boldsymbol{s}^{(1)} \cdots \boldsymbol{s}^{(T)}\right]$ is the matrix of all coefficients, the L1 norm $\|\cdot\|_1$ enforces sparsity of the coefficients, and the Frobenious norm $\|\cdot\|_{\mathsf{F}}$ regularizes the complexity of $\mathbf{L}$ with regularization parameters $\mu, \lambda \in \mathbb{R}$. To solve (3.4) efficiently, PG-ELLA

1. replaces $\mathfrak{R}(\cdot)$ with an upper bound as done in typical PG optimization,
2. approximates the first term with a second-order Taylor expansion around an estimate $\boldsymbol{\alpha}^{(t)}$ of the single-task policy parameters for task $\mathcal{Z}^{(t)}$, and
3. optimizes $\boldsymbol{s}^{(t)}$ only when training on task $\mathcal{Z}^{(t)}$.

These steps reduce the learning problem to a series of on-line update equations discussed in Section 2.2.5. These techniques are applied to disturbance rejection on both simulated and real Turtlebots, as well as simulated AR.Drones.

In disturbance rejection, it is assumed that the nominal dynamics of the plant are additively disturbed by a signal $\boldsymbol{\omega}$. The system dynamics are given by $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\omega}$, where $\boldsymbol{f} : \mathbb{R}^{d_x} \times \mathbb{R} \to \mathbb{R}^{d_x}$, and $\boldsymbol{\omega} \in \mathbb{R}^{d_x}$. The goal is to determine the control input that minimizes the effect of the disturbance in the return function, so that the plant can execute this task.

There are well known optimal control (Lewis and Syrmos, 2012; Dorato et al., 1994) techniques to solve this problem, if there is an available mathematical model. However, if such a model is not available, or there is only partial knowledge of the model, formal solutions are not effective. RL offers one alternative solution to this problem, but only in a single-task setting, and it requires numerous interactions with the environment to learn an effective control policy to compensate for the disturbance. However, in a lifelong learning setting, the learner could build upon its existing knowl-

edge in controlling other systems, each with their own disturbances, to rapidly learn a control for a system with a novel disturbance. It is assumed that the learner attempts to optimize control policies for a set of robots, all of which have the same nominal dynamics, given by $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$. Each robot is affected by a different disturbance function $\boldsymbol{\omega}^{(t)}$. All $\boldsymbol{\omega}^{(t)}$ share the same structure but different parameters, *e.g.,* all the disturbances are constant but different, all are sinusoidal with different phases or amplitudes, etc.

Lifelong machine learning takes advantage of the potential for knowledge transfer among the different tasks, learning how to compensate for the disturbance without requiring a mathematical model describing the system dynamics. A reward function is designed so that the lifelong learner penalizes the effect of the disturbance over either a realistic simulated robot or an actual one. In the next section, our application of lifelong learning to the problem of robotic control under disturbances is presented.

## 3.3 Experimental Design

In our disturbance rejection scenario, the focus was on learning control policies for navigating vehicles to a goal location as the vehicle experienced disturbances in its actuators. Two case studies are covered using two different commercial robotic platforms: the Turtlebot 2 (Turtlebot 2, 2016) and the AR.Drone quadrotor (Parrot AR.Drone 2, 2016). A simulated turtlebot is shown in 3.1(a) and a simulated AR.Drone is pictured in 3.4(a). Disturbances are emulated in the actuators, namely, wheel servos and propellers, respectively. Each case study involves multiple robots from a single platform, and the disturbances are assumed to be different for each

robot. The main goal in both case studies is to build a knowledge base from previously learned policies that allows each robot to compensate for the disturbance and fulfill its goal, while transferring this knowledge to new robots with different disturbances. This section describes the validation procedure of our lifelong machine learning approach. In order to obtain the simulation and experimental results, the Hydro version of the Robot Operating System (ROS) (System, 2016) is used.

### 3.3.1 Disturbance Rejection on Turtlebots

In this scenario, each Turtlebot faces a disturbance consisting of a bias on its angular and linear velocity, forcing the robots to compensate for the induced failure to navigate successfully. Note that this type of disturbance in actuation is common in physical robots and autonomous ground vehicles, stemming from a variety of sources, such as calibration issues, wear in the drive train, or interference from debris. To simulate these disturbances, a random and constant disturbance is introduced to the control signal. The disturbance is drawn uniformly from $[-0.1, 0.1] \subset \mathbb{R}$ for each robot and measured in $m/s$. These limits were selected to provide a large noise that was within the bounds of the Turtlebot control system.

To simulate lifelong learning on numerous Turtlebots, each with their own disturbances, simulations were conducted using the high-fidelity Gazebo simulator (Gazebo, 2016; System, 2016). To show that lifelong learning is similarly effective on real robots, our approach was also evaluated on five real Turtlebots.

We assume little knowledge of the Turtlebot's dynamics is assumed. In our application, each robot's state is defined as $\boldsymbol{x} = (\rho, \xi, \psi)^{\mathsf{T}}$, with $\rho, \xi$ and $\psi$ as illustrated in

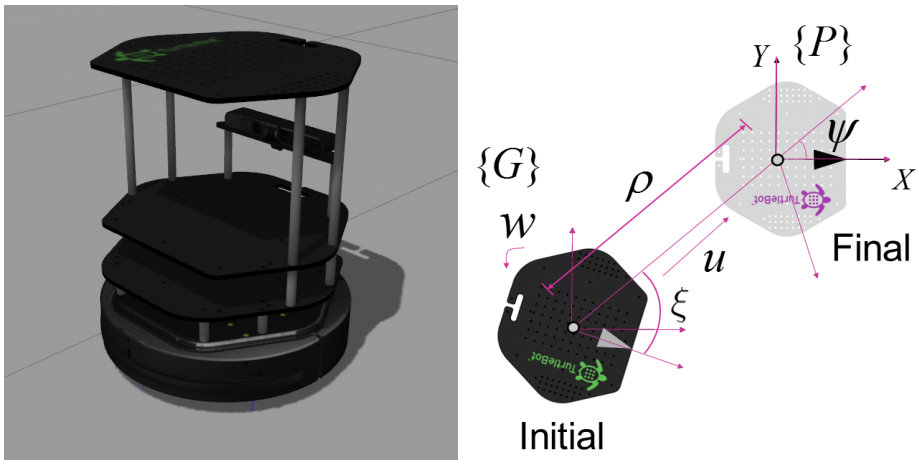
Figure 3.1(b). To extract state features for learning, the following nonlinear transformation of the position and heading angle was used:

$$\boldsymbol{\phi}(\boldsymbol{x}) = \begin{pmatrix} \rho\cos(\xi) \\ \xi \\ \frac{\cos(\xi)\sin(\xi)}{\xi}(\xi+\psi) \\ 1 \end{pmatrix} . \tag{3.5}$$

Given the stochastic policy $\pi_{\boldsymbol{\theta}^{(t)}} \sim \mathcal{N}(\boldsymbol{a}^{(t)}, \boldsymbol{\Sigma})$ for the $t$-th Turtlebot, the control action is then specified by $\boldsymbol{a}^{(t)} = {\boldsymbol{\theta}^{(t)}}^{\mathsf{T}}\boldsymbol{\phi}(\boldsymbol{x}) = (u, w)^{\mathsf{T}}$ where $u$ and $w$ are the linear and angular velocities of the robots. This particular choice of nonlinear transformation is inspired by a simplified kinematic model for unicycle-like vehicles in polar coordinates (Aicardi et al., 1994). In this model, the state space is given by $\mathcal{X} \subset \mathbb{R}^3$ and the action space is described by $\mathcal{A} \subset \mathbb{R}^2$. This simplified kinematics model ignores contributions to the dynamics of the system from the robot's mass, damping and friction coefficients, as well as inputs such as forces and torques.

**Simulation Methodology**

In these experiments, FD (Kober et al., 2013) was used as the base learner in PG-ELLA for its simplicity and good performance in simulation, despite its known stability issues (which were not not experienced). Twenty simulated Turtlebots were generated, each with a different constant disturbance and a unique goal, both selected uniformly. This number of robots provided a large task diversity, while still being small enough to simulate practically.




(a) Turtlebot (b) State variables

Figure 3.1: (a) The Turtlebot 2 model in Gazebo, and (b) its state variables in the simplified go-to-goal problem.

To evaluate the system performance, FD was used as our PG method to train 19 robots' initial policies for $J = 100$ iterations with $n = 21$ roll-outs per iteration and $H = 70$ time steps per roll-out. If the robot reached the goal in less than 70 time steps, the experiment continues to run to completion ensuring that a good policy reaches the goal and stops. The 20th robot is initialized with the mean policy of the observed tasks and trained for only $J = 10$ iterations. Note that all systems in our experiment require more than 10 iterations to converge to a good controller, so subsequent policy improvement is essential for decent performance. These policies are then used as the $\boldsymbol{\alpha}^{(t)}$ estimates for PG-ELLA. The number of roll-outs and time steps were selected to allow for successful learning while minimizing the runtime.

PG-ELLA trains the shared knowledge repository $\boldsymbol{L}$ and sparse policy representations $\boldsymbol{s}^{(t)}$ using the update equations given by (2.12)–(2.15). For our experiments, the Hessian is approximated with the identity matrix because it reduced the number of roll-outs with little effect on performance. For PG-ELLA's parameters, $k = 4$ columns were used in the shared basis, sparsity coefficient $\beth = 1 \times 10^{-5}$, and regularization

coefficient $\lambda = 1 \times 10^{-3}$. The learning rate was set to $\upsilon = 1 \times 10^{-6}$ and the policy's standard deviation was $\sigma = 0.001$.

### Application to Physical Robots

To demonstrate our method on a real Turtlebot, four tasks were learned using conventional PG and then transfer was evaluated on the fifth task. The number of roll-outs is reduced to $n = 11$ and the number of learning iterations to $J = 30$ to minimize the experimentation time. In contrast to simulation, physical robots have battery limitations, and the continuous generation of random trajectories require a human presence to guarantee correct execution of the experiments. All other parameters, including the added noise, are kept the same as the simulation. To get position information, a beacon which can easily be segmented from camera data is used to mark the goal. The Turtlebot is equipped with a Kinect whose depth information provides the angle and distance. If the robot loses sight of the goal, it enters a recovery mode which rotates it toward the last known goal location, and a penalty is applied for each time step spent in recovery mode. With the fewer number of iterations, only one of the four Turtlebots used as source tasks reached the goal. However, all four systems did exhibit learning improvement.

### Turtlebot Results

The left plot in Figure 3.2 compares the reward for policies learned by PG-ELLA on the 20th task against PG, averaged over 20 trials. Performance for PG-ELLA is shown beginning at 10 iterations, since the initial seed policies for PG-ELLA were learned using those first 10 iterations. The learning curves are plotted as the polices

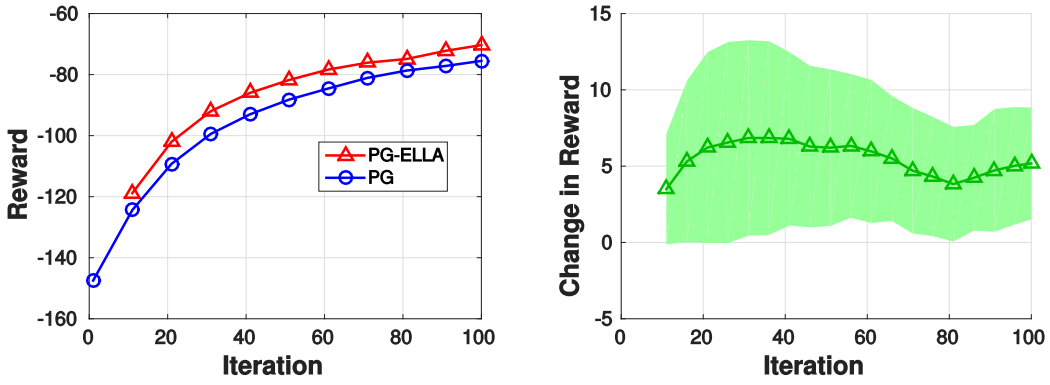



Figure 3.2: (Left) Learning curves for PG and PG-ELLA in the Turtlebot simulation, averaged over 20 trials. (Right) Change in reward as a measure of positive transfer achieved by lifelong learning on Turtlebots.

are improved by PG for an additional 90 learning iterations.

These results show that PG-ELLA is able to both successfully reconstruct and improve the control policies through positive transfer with respect to PG. This is clearly shown by both the initial *jump-start* improvement at the 10th iteration, and the consistently better learning performance over the first 100 iterations. The right plot in Figure 3.2 depicts the change in reward over the learning curve, showing positive transfer between tasks. After only 10 iterations to initialize $\boldsymbol{\alpha}^{(t)}$, an improved initial performance and better reward are obtained using PG-ELLA via transfer from the shared knowledge base.

Figure 3.3 shows the results of learning on real Turtlebots with the effect of disturbances, confirming the improvement from PG-ELLA on physical systems. These results compare the performance of PG-ELLA (red line) on a new Turtlebot to the performance of PG on four systems (light blue lines) and their mean (dark blue line). The PG policy exhibiting the best performance is the robot that with the smallest disturbance, yet this system is still out-performed by PG-ELLA. These results also show

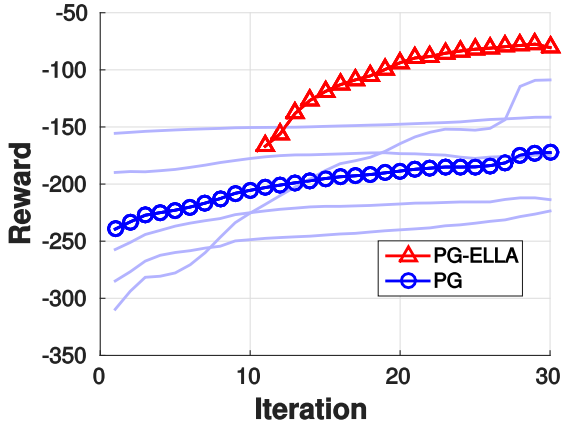


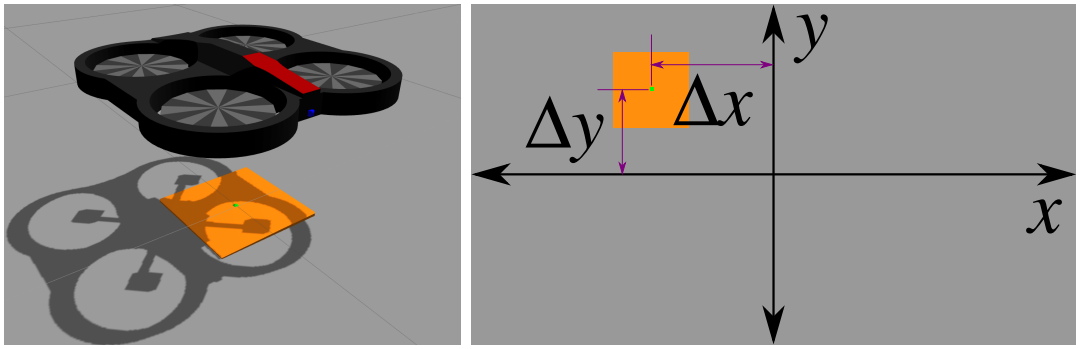

Figure 3.3: Learning curves on the real Turtlebots. PG-ELLA (red line) on a new Turtlebot performs better than PG on five other systems (light blue line) and their mean (dark blue line).

that in many cases, PG is negatively affected by the disturbances while PG-ELLA exhibits better and more consistent learning.

One possible explanation for this is that PG's learning is hindered on the real Turtlebots due to the noisy environment of the physical robots, the added complexity of the empirical reward function, and the presence of local optima in the objective function. By sharing knowledge from other tasks, PG-ELLA is able to compensate for these aspects and exhibit more rapid learning. This hypothesis agrees with our results on the larger set of simulated Turtlebots.

### 3.3.2 Disturbance Rejection on AR.Drones

Aerial vehicles are a compelling domain that allows us to demonstrate our approach generalizes to vastly different platforms. The AR.Drone quadrotor is evaluated using the TUM AR.Drone Simulator (Huang and Sturm, 2016) and Gazebo in order to simulate lifelong learning over numerous quadrotors, each with their own distur-




(a) $Z$ state (b) $X$,$Y$ state

Figure 3.4: (a) The AR.Drone 2 model in Gazebo, and (b) downward-facing camera feed used for localization.

bance. For each quadrotor, our approach learns a policy to land the AR.Drone on a marker, while the quadrotor is affected by a disturbance in its propellers that bias its motion. The disturbance is emulated by a bias on the linear velocities in the $x$ and $y$ coordinates, $u_x$ and $u_y$, as may commonly be caused by wind or wear on the rotors. To simulate these disturbances, a random and constant disturbance drawn uniformly from $[-0.15, 0.15] \subset \Re$ was added to each robot, measured in $m/s$.

The quadrotors are localized using a downward-facing camera with the assumption that the landing pad marker is visible. Each robot's state is defined as $\boldsymbol{x} = (x, y, z)^\mathsf{T}$, where $x$ and $y$ are the Cartesian coordinates of the goal relative to the center of the camera as the origin. The difference between the quadrotor's altitude and the landing altitude is given as $z$, measured using the robot's on-board sonar height sensor. The state features for learning are the normalized values of the robot's state. The normal range of values for $x$ and $y$ are $\pm 2$ and $\pm 1.5$, respectively. If the landing pad marker is not visible, the $(x, y)$ state is represented as $x_h = 4$ and $y_h = 3$, which is double the maximum possible state. During the learning phase, at time-step $h$, when the quadrotor is at $z_h = 0$ ($0.4\ m$ in altitude) and positioned over the landing pad marker

at $x_h = [-0.7, 0.7]$ and $y_h = [-0.8, 0.6]$, the landing action is triggered. There is an offset in $y$ since the downward-facing camera underneath the quadrotor is not centered. The landing altitude was selected to accommodate the specifications for the sonar which have 0.4 $m$ as the lowest altitude that the sensor can accurately measure.

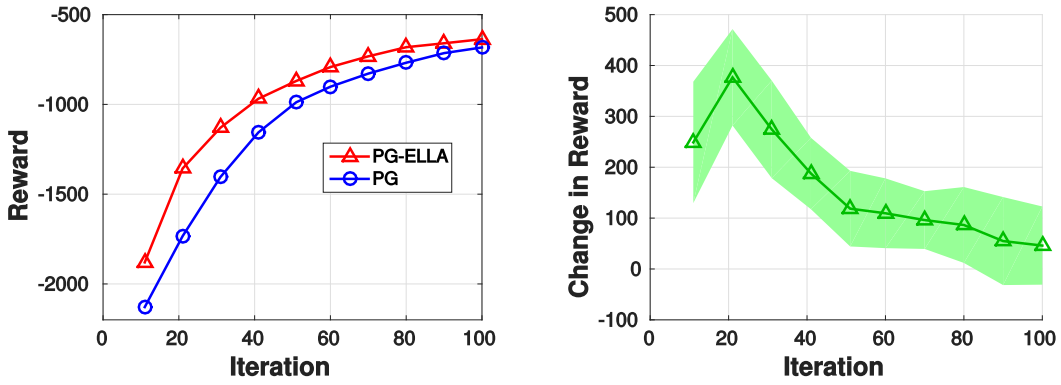



Figure 3.5: (Left) Learning curves for PG and PG-ELLA in the AR.Drone simulation, averaged over 20 trials. (Right) Change in reward as a measure of positive transfer achieved by lifelong learning on AR.Drones.

Landing occurs regardless of the quadrotor's orientation. The system sends signals in the format of a ROS Twist message, which consists of the instructions to control the linear velocity of the quadrotor in the $x, y$, and $z$ directions. Valid actions are in the range of $\pm 1 m/s$ for each axis of control.

### Simulation Methodology

20 simulated quadrotors were generated, each with a different constant disturbance selected uniformly. Using FD as our PG learner, 19 robot's initial policies were trained for $J = 100$ iterations with $n = 15$ roll-outs per iteration and $H = 150$ time steps per roll-out. The 20th robot is initialized with the mean policy over the 19 observed

policies and is trained for only $J = 10$ iterations. However, out of the 19 robots with 100 learning iterations, 3 systems did not converge to a good controller[1].

The same methodology used to train PG-ELLA on Turtlebots is followed for the quadrotors. The hyper-parameters are $k = 4$, $\mu = 1 \times 10^{-5}$, and $\lambda = 1 \times 10^{-6}$. The learning rate was $\upsilon = 1 \times 10^{-6}$ and the policy's standard deviation was set to $\sigma = 0$ to make it deterministic.

**AR.Drone Results**

Since the 20th task is trained for 10 iterations and then given a policy reconstructed from PG-ELLA, comparing PG-ELLA with PG starts after those 10 iterations. Learning is then continued for 90 more iterations using PG. The left plot in Figure 3.5 shows the learning curve for policies trained by PG-ELLA (red line) against those learned by PG (blue line), averaged over 20 tasks. The results show that PG-ELLA is also effective in the quadrotor domain for both reconstructing and improving the learned policies. Similar to the Turtlebots, improved initial *jump-start* performance and more rapid learning than PG are observed. The right plot in Figure 3.5 depicts the gain in reward.

## 3.4 Discussion

This work demonstrates the effectiveness of lifelong reinforcement learning at reducing training time in both simulated 3D environments and on real robots. The approach

[1] AR.Drone experiments were conducted by my co-authors at WSU (Isele et al., 2016a).

is not platform specific — the same algorithm is used on two very different platforms: Turtlebots and AR.Drones. My investigation showcased a moderate amount of diversity, sharing knowledge between robots with different goals and different disturbances. Additionally, the boost in performance is sustained after additional training, allowing policies to be further improved. While the gains were modest, PG-ELLA giving comparable performance with 10 to 20 fewer iterations equates to saving hundreds of trials.

While the number of trials was reduced using this method, the number of trials required for training a physical robot is still prohibitively expensive. Currently this approach is limited to linear function approximation. This limits the types of problems that the algorithm can handle, and suggests that a move to more expressive models will have an even greater issue concerning sample complexity. This makes reducing the number of trials absolutely crucial for physical systems.

An online sparse coding framework was used to learn task relationships. This decomposition of knowledge into a shared repository and a task specific code suggests a modular approach to thinking about knowledge: there is general knowledge that can be acquired from similar tasks, and there is task specific knowledge that needs to be learned separately. This is a simplification: different tasks will likely have different aspects in common. However, this simplification provides some insight. Assuming that general information is learned during the early stages of training, the general knowledge accumulated by the system is not being utilized when a new system is trained from scratch in order to identify its relation to other tasks. This suggests that our gains were far less than they could be, and suggests that a better approach might be to predict an initialization from our prior knowledge and subsequently refine that knowledge with task specific training. In the next chapter, I not only show that

such a strategy is possible, but also confirm that it produces a much greater transfer of knowledge.

# Chapter 4

# Predicting Policies

Selecting the appropriate knowledge to transfer requires an accurate estimate of the inter-task relationships to identify relevant knowledge. Inter-task relationships are typically estimated based on training data for each task (Baxter, 2000; Ando and Zhang, 2005; Bickel et al., 2009a; Maurer et al., 2013). For example, in Chapter 3, a turtlebot with an unknown disturbance was trained on a go-to-goal target task. The learned policy was then compared against other learned policies, and an improved policy was constructed. *Before* transfer could take place, the system needed to gather experience on the new task.

This means that in a lifelong learning setting, before a new task can be addressed, the system must stop, collect data, and train on the new task to identify appropriate information to transfer. To reduce this burden, this chapter develops a lifelong learning method that utilizes high-level task descriptions to model the inter-task relationships. I show that using task descriptors improves the performance of the learned task policies, which is justified theoretically and demonstrated empirically. More im-

portantly, given only the description for a new task, the lifelong learner is able to use the coupled dictionary and task description to accurately predict a model for the new task, eliminating the need to gather training data in order to identify the inter-task relationships.

To make our approach more clear, consider the human ability to rapidly bootstrap a model for a new task, given *only* a *high-level task description*—before obtaining experience on the actual task. For example, viewing only the image on the box of a new IKEA chair, previously related assembly tasks can immediately be identified and a plan to begin assembling the chair can be formulated. In the same manner, an experienced inverted-pole-balancing agent may be able to predict the controller for a new pole given its mass and length, prior to interacting with the physical system. These examples suggest that an agent could similarly use high-level task information to bootstrap a model for a new task more efficiently.

I investigate how a high-level task description can be incorporated into a lifelong learning process. The resulting algorithm, Task Descriptors for Lifelong Learning (TaDeLL), encodes task descriptions as feature vectors that identify each task, treating these descriptors as side information in addition to training data for individual tasks (Isele et al., 2016b, 2017). Several other works have explored the use of high-level task descriptors to model the inter-task relationships in MTL and transfer learning settings. Task descriptors have been used in combination with neural networks (Bakker and Heskes, 2003) to define a task-specific prior and to control the gating network between individual task clusters. Bonilla et al. (2007) explore similar techniques for multi-task kernel machines, using task features in combination with the data for a gating network over individual task experts to augment the original task training data. These papers focus on multi-task classification and regression in batch

settings where the system has access to the data and features for all tasks. Work by Sinapov et al. (2015) uses task descriptors to estimate the transferability between each pair of tasks for transfer learning. Given the descriptor for a new task, they identify the source task with the highest predicted transferability, and use that source task for a warm start in reinforcement learning. Though effective, their approach is computationally expensive, since they estimate the transferability for every task pair through repeated simulation. Their evaluation is also limited to a transfer learning setting, and they do not consider the effects of transfer over consecutive tasks or updates to the transferability model, which is done in the lifelong setting. In comparison, our approach operates online over consecutive tasks and is much more computationally efficient.

To model the inter-task relationships between the task descriptions and the individual task policies in lifelong learning, *coupled dictionary learning* is used. Coupled dictionary learning has been used in image processing (Yang et al., 2010) and was recently explored in the machine learning literature (Xu et al., 2016). The core idea is that two feature spaces can be linked through two dictionaries that are coupled by a joint sparse representation. The coupled dictionary enforces the notion that tasks with similar descriptions should have similar policies, but still allows dictionary elements the freedom to accurately represent the different types of information.

In addition to improving the task models, by modeling the relationship between task descriptions and task policies, accurate predictions of new policies for unseen tasks can be created given only their description. This process of learning without data is known as *zero-shot learning*. Zero-shot learning seeks to successfully label out-of-distribution examples, often by learning an underlying representation and using outside information to learn appropriate mappings to the latent space (Palatucci

et al., 2009; Socher et al., 2013). This capability is particularly important in an online setting where it enables the system to accurately predict policies for new tasks, without requiring the system to pause to gather training data on the new task. This chapter presents and extends work that I have previously published (Isele et al., 2016b, 2017).

## 4.1 Task Descriptors

While most MTL and lifelong learning methods use task training data to model inter-task relationships, high-level descriptions can describe task differences. For example, in multi-task medical domains, patients are often grouped into tasks by demographic data and disease presentation (Oyen and Lane, 2012). In control problems, the system parameters (e.g., the spring, mass, and damper constants in a spring-mass-damper system) can be used to describe the task (Isele et al., 2016b). When a large set of tasks are being generated for an agent, the generator variables can also serve as task features (Sinapov et al., 2015). Task descriptors can also be used to provide the designer of the system a way to influence or control the agent. Goals or instructions can be encoded with task features (Schaul et al., 2015a). For example, variables represent the action to be performed, the object that receives the action, and the number of times the action is meant to be repeated (Oh et al., 2017). Descriptors can also be derived from external sources, such as natural language descriptions (Pennington et al., 2014; Huang et al., 2012) or Wikipedia text associated with the task (Socher et al., 2013).

While there are many methods for selecting task features, selecting the *best* and

most relevant features is not straightforward. Feature selection is a problem faced by the broader machine learning community (Guyon and Elisseeff, 2003), with notable recent approaches that learn the features representations (Lee et al., 2009). Very recently, Higgins et al. (2017) has shown that learning approaches can also be used to discover task features. While the selection and learning of task features is an important concern relating to the adoption of task features for lifelong learning, it is an open problem. I assume that task features are provided as part of the learning problem so that the focus can be on policy generation.

To incorporate task descriptors into the learning procedure, I assume that each task $\mathcal{Z}^{(t)}$ has an associated descriptor $\boldsymbol{m}^{(t)}$ that is given to the learner upon first presentation of the task. The learner has no knowledge of future tasks, or the distribution of task descriptors. The descriptor is represented by a feature vector $\phi\left(\boldsymbol{m}^{(t)}\right) \in \mathbb{R}^{d_m}$, where $\phi(\cdot)$ performs feature extraction and (possibly) a non-linear basis transformation on the features. I make no assumptions on the uniqueness of $\phi\left(\boldsymbol{m}^{(t)}\right)$, although in general, tasks will have different descriptors.[1] Additionally, each task also has associated training data $\boldsymbol{X}^{(t)}$ to learn the model. In the case of RL tasks, the data consists of trajectories that are dynamically acquired by the agent through experience in the environment.

## 4.2 Coupled Dictionary Optimization

As described previously in Section 2.2.4 and Section 3, many multi-task and lifelong learning approaches have found success with factorizing the policy parameters $\boldsymbol{\theta}^{(t)}$ for

[1] This raises the question of what descriptive features to use, and how task performance will change if some descriptive features are unknown. I explore these issues in Section 4.6.1.

each task as a sparse linear combination over a shared basis: $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$. In effect, each column of the shared basis $\boldsymbol{L}$ serves as a reusable model or policy component representing a cohesive chunk of knowledge. In a lifelong learning setting, the basis $\boldsymbol{L}$ is refined over time as the system learns more tasks. The coefficient vectors for each task $\boldsymbol{S} = [\boldsymbol{s}^{(1)} \dots \boldsymbol{s}^{(T)}]$ encode the task policies in this shared basis, providing an embedding of the tasks based on how their policies share knowledge. This assumes each policy is a linear combination of basis vectors. This assumption creates a clearly defined method for relating one task to another, but in cases where the policies are not related linearly, many basis vectors may need to be learned in order to approximate the non-linear relations.

I make a similar assumption about the task descriptors: that the descriptor features $\phi\left(\boldsymbol{m}^{(t)}\right)$ can be linearly factored[2] using a latent basis $\boldsymbol{D} \in \mathbb{R}^{d_m \times k}$ over the descriptor space. This basis captures relationships among the descriptors, with coefficients that similarly embed tasks based on commonalities in their descriptions. From a co-view perspective (Yu et al., 2014), both the policies and descriptors provide information about the task, and so each can augment the learning of the other. Each underlying task is common to both views, and task embeddings are sought that are consistent for *both* the policies and their corresponding task descriptors. As depicted in Figure 4.1, coupling can be enforced between the two bases $\boldsymbol{L}$ and $\boldsymbol{D}$, by sharing the same coefficient vectors $\boldsymbol{S}$ to reconstruct both the policies and descriptors. Therefore, for task $\mathcal{Z}^{(t)}$,

$$\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)} \qquad\qquad \phi\left(\boldsymbol{m}^{(t)}\right) = \boldsymbol{D}\boldsymbol{s}^{(t)} \ . \tag{4.1}$$

[2]This is potentially non-linear w.r.t $\boldsymbol{m}^{(t)}$, since $\phi$ can be non-linear.

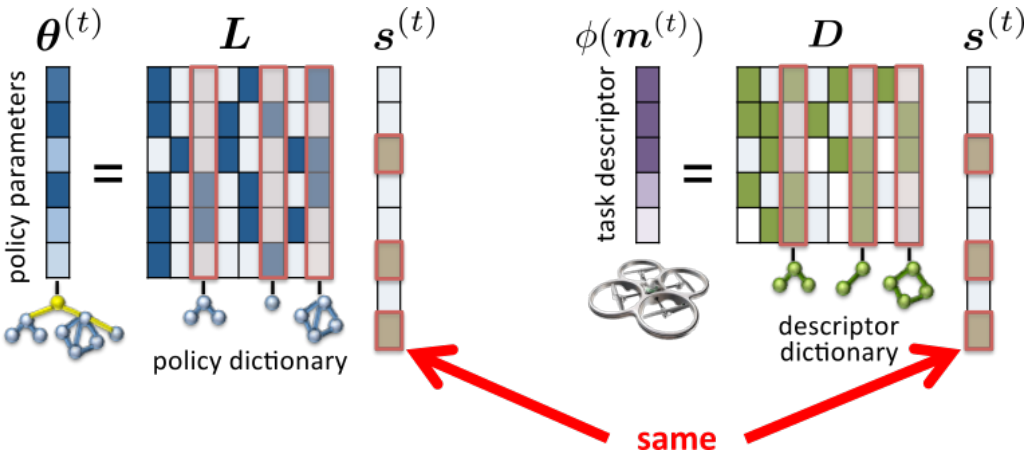



Figure 4.1: The coupled dictionaries of TaDeLL, illustrated on an RL task. Policy parameters $\boldsymbol{\theta}^{(t)}$ are factored into $\boldsymbol{L}$ and $\boldsymbol{s}^{(t)}$ while the task description $\phi\left(\boldsymbol{m}^{(t)}\right)$ is factored into $\boldsymbol{D}$ and $\boldsymbol{s}^{(t)}$. Because both dictionaries are forced to use the same sparse code $\boldsymbol{s}^{(t)}$, the relevant pieces of information for a task become coupled with the description of the task.

To optimize the coupled bases $\boldsymbol{L}$ and $\boldsymbol{D}$ during the lifelong learning process, coupled dictionary optimization methods are employed from the sparse coding literature (Yang et al., 2010). These techniques optimizes the dictionaries for multiple feature spaces that share a joint sparse representation. The notion of coupled dictionary learning has led to high-performance algorithms for image super-resolution (Yang et al., 2010), and allowing the reconstruction of high-resolution images from low-resolution samples for both multi-modal retrieval (Zhuang et al., 2013) and cross-domain retrieval (Yu et al., 2014). The core idea is that features in two independent subspaces can have the same representation in a third subspace.

Given the factorization in Eq. 4.1, the multi-task objective (Eq. 2.11) can be re-formulated for the coupled dictionaries as

$$\min_{\boldsymbol{L},\boldsymbol{D},\boldsymbol{S}} \frac{1}{T}\sum_{t}\left[\mathcal{L}\left(\boldsymbol{\theta}^{(t)}\right)+\zeta\left\|\phi\left(\boldsymbol{m}^{(t)}\right)-\boldsymbol{D}\boldsymbol{s}^{(t)}\right\|_2^2+\text{מ}\left\|\boldsymbol{s}^{(t)}\right\|_1\right]+\lambda(\|\boldsymbol{L}\|_{\mathsf{F}}^2+\|\boldsymbol{D}\|_{\mathsf{F}}^2)\ , \quad (4.2)$$

where $\zeta$ balances the model's or policy's fit to the task descriptor's fit.

To solve Eq. 4.2 online, $\mathcal{L}(\cdot)$ is approximated by a second-order Taylor expansion around $\boldsymbol{\alpha}^{(t)}$, the minimizer for the single-task learner. In reinforcement learning, $\pi_{\boldsymbol{\alpha}^{(t)}}$ is the single-task policy for $\mathcal{Z}^{(t)}$ based on the observed trajectories.

Approximating Eq. 4.2 leads to

$$\min_{\boldsymbol{L},\boldsymbol{D},\boldsymbol{S}} \frac{1}{T}\sum_{t}\left[\left\|\boldsymbol{\alpha}^{(t)}-\boldsymbol{L}\boldsymbol{s}^{(t)}\right\|^2_{\boldsymbol{\Gamma}^{(t)}} + \zeta\left\|\phi\left(\boldsymbol{m}^{(t)}\right)-\boldsymbol{D}\boldsymbol{s}^{(t)}\right\|_2^2 + \text{מ}\left\|\boldsymbol{s}^{(t)}\right\|_1\right] + \lambda(\|\boldsymbol{L}\|_{\mathsf{F}}^2+\|\boldsymbol{D}\|_{\mathsf{F}}^2) \ . \tag{4.3}$$

I can merge pairs of terms in Eq. 4.3 by choosing:

$$\boldsymbol{\beta}^{(t)} = \begin{bmatrix} \boldsymbol{\alpha}^{(t)} \\ \phi\left(\boldsymbol{m}^{(t)}\right) \end{bmatrix} \qquad \boldsymbol{K} = \begin{bmatrix} \boldsymbol{L} \\ \boldsymbol{D} \end{bmatrix} \qquad \boldsymbol{V}^{(t)} = \begin{bmatrix} \boldsymbol{\Gamma}^{(t)} & \boldsymbol{0} \\ \boldsymbol{0} & \zeta\boldsymbol{I}_{d_m} \end{bmatrix} \ ,$$

where $\boldsymbol{0}$ is the zero matrix, letting us rewrite Eq. 4.3 concisely as

$$\min_{\boldsymbol{K},\boldsymbol{S}} \frac{1}{T}\sum_{t}\left[\left\|\boldsymbol{\beta}^{(t)}-\boldsymbol{K}\boldsymbol{s}^{(t)}\right\|^2_{\boldsymbol{V}^{(t)}}+ \text{מ}\left\|\boldsymbol{s}^{(t)}\right\|_1\right]+\lambda\|\boldsymbol{K}\|_{\mathsf{F}}^2 \ . \tag{4.4}$$

This objective can now be solved efficiently online, as a series of per-task update rules given in Algorithm 2, which is called TaDeLL (Task Descriptors for Lifelong Learning). $\boldsymbol{L}$ and $\boldsymbol{D}$ are updated independently using Equations 2.13–2.15, following a recursive construction based on an eigenvalue decomposition.

In an RL setting, at each timestep TaDeLL receives a new RL task and samples trajectories for the new task. The single-task policy is computed using a twice-differentiable policy gradient method as $\boldsymbol{\alpha}^{(t)}$. The Hessian $\Gamma^{(t)}$, calculated around the

**Algorithm 2** TaDeLL (k, $\lambda$, מ)

$\boldsymbol{L} \leftarrow \text{RandomMatrix}_{d,k}$, $\boldsymbol{D} \leftarrow \text{RandomMatrix}_{m,k}$
**while** some task $\left(\mathcal{Z}^{(t)}, \phi\left(\boldsymbol{m}^{(t)}\right)\right)$ is available **do**
  $\mathbb{T}^{(t)} \leftarrow \text{collectData}(\mathcal{Z}^{(t)})$
  Compute $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\Gamma}^{(t)}$ from $\mathbb{T}^{(t)}$
  $\boldsymbol{s}^{(t)} \leftarrow \arg\min_{\boldsymbol{s}} \left\|\boldsymbol{\beta}^{(t)} - \boldsymbol{K}\boldsymbol{s}\right\|^2_{\boldsymbol{V}^{(t)}} + \text{מ}\|\boldsymbol{s}\|_1$
  $\boldsymbol{L} \leftarrow \text{update}L(\boldsymbol{L}, \boldsymbol{s}^{(t)}, \boldsymbol{\alpha}^{(t)}, \boldsymbol{\Gamma}^{(t)}, \lambda)$ Eq. 2.13–2.15
  $\boldsymbol{D} \leftarrow \text{update}D(\boldsymbol{D}, \boldsymbol{s}^{(t)}, \phi\left(\boldsymbol{m}^{(t)}\right), \zeta\boldsymbol{I}_{d_m}, \lambda)$ Eq. 2.13–2.15
  **for** $t \in \{1, \ldots, T\}$ **do**: $\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{L}\boldsymbol{s}^{(t)}$
**end while**

point $\boldsymbol{\alpha}^{(t)}$, is derived according to the particular policy gradient method being used. Bou Ammar et al. derive it for the cases of Episodic REINFORCE (Williams, 1992) and Natural Actor Critic (Peters and Schaal, 2008). The reconstructed $\boldsymbol{\theta}^{(t)}$ is then used as the policy for the task $\mathcal{Z}^{(t)}$. For completeness, the Hessian calculations follow.

**Episodic REINFORCE Hessian** (Bou Ammar et al., 2014) In episodic REINFORCE (Williams, 1992), the stochastic policy for task $t$ is chosen according $\boldsymbol{a}^{(t)} = \boldsymbol{\theta}^{(t)\mathsf{T}}\boldsymbol{x}_h^{(t)} + \boldsymbol{\epsilon}_h$, with $\boldsymbol{\epsilon}_h \sim \mathcal{N}(0, \sigma^2)$, and so $\pi\left(\boldsymbol{a}_h^{(t)} \mid \boldsymbol{x}_h^{(t)}\right) \sim \mathcal{N}\left(\boldsymbol{\theta}^{(t)\mathsf{T}}\boldsymbol{x}_h^{(t)}, \sigma^2\right)$. Recall from 2.1.2 that $\mathcal{J}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right)$ is the KL divergence used to bound the log of the expected return, and its gradient

$$\nabla_{\tilde{\boldsymbol{\theta}}^{(t)}} \mathcal{J}\left(\tilde{\boldsymbol{\theta}}^{(t)}\right) = -\mathbb{E}\left[\mathfrak{R}^{(t)}(\boldsymbol{\tau}) \sum_{h=1}^{H^{(t)}} \sigma^{-2}\left(\boldsymbol{a}^{(t)} - \boldsymbol{\theta}^{\mathsf{T}}\boldsymbol{x}_h^{(t)}\right)\right] ,$$

is used to minimize the KL-divergence, equivalently maximizing the total discounted pay-off. The second derivative for episodic REINFORCE is given by

$$\boldsymbol{\Gamma}^{(t)} = \mathbb{E}\left[\sum_{h=1}^{H^{(t)}} \sigma^{-2}\boldsymbol{x}_h^{(t)}\boldsymbol{x}_h^{(t)\mathsf{T}}\right] .$$

**Natural Actor Critic Hessian** (Bou Ammar et al., 2014) In episodic Natural Actor Critic (eNAC), the stochastic policy for task $t$ is chosen in a similar fashion to that of REINFORCE: $\pi\left(\boldsymbol{a}_h^{(t)} \mid \boldsymbol{x}_h^{(t)}\right) \sim \mathcal{N}\left(\boldsymbol{\theta}^{(t)\mathsf{T}}\boldsymbol{x}_h^{(t)}, \sigma^2\right)$. The change in the probability distribution is measured by a KL-divergence that is approximated using a second-order expansion to incorporate the Fisher information matrix. Accordingly, the gradient follows: $\tilde{\nabla}_{\boldsymbol{\theta}}\mathcal{J} = \boldsymbol{G}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta})$, where $\boldsymbol{G}$ denotes the Fisher information matrix. The Hessian can be computed in a similar manner to the REINFORCE Hessian.

## 4.3 Zero-Shot Transfer Learning

In a lifelong setting, when faced with a new task, the agent's goal is to learn an effective policy for that task as quickly as possible. At this stage, previous multi-task and lifelong learners incurred a delay before they could produce a decent policy, since they needed to acquire data from the new task in order to identify related knowledge and train the new policy via transfer.

Incorporating task descriptors enables our approach to predict a policy for the new task immediately, given *only* the descriptor. This ability to perform zero-shot transfer is enabled by the use of coupled dictionary learning, which allows us to observe a data instance in one feature space (i.e., the task descriptor), and then recover its underlying latent signal in the other feature space (i.e., the policy parameters) using the dictionaries and sparse coding.

Given only the descriptor $\boldsymbol{m}^{(t_{new})}$ for a new task $\mathcal{Z}^{(t_{new})}$, the embedding of the task in the latent descriptor space can be estimated via LASSO on the learned dictionary

**Algorithm 3** Zero-Shot Transfer to a New Task $\mathcal{Z}^{(t_{new})}$

1: **Inputs:** task descriptor $\boldsymbol{m}^{(t_{new})}$, learned bases $\boldsymbol{L}$ and $\boldsymbol{D}$
2: $\tilde{\boldsymbol{s}}^{(t_{new})} \leftarrow \arg\min_{\boldsymbol{s}} \left\| \phi\left(\boldsymbol{m}^{(t_{new})}\right) - \boldsymbol{D}\boldsymbol{s} \right\|_2^2 + \mu \left\| \boldsymbol{s} \right\|_1$
3: $\tilde{\boldsymbol{\theta}}^{(t_{new})} \leftarrow \boldsymbol{L}\tilde{\boldsymbol{s}}^{(t_{new})}$
4: **Return:** $\pi_{\tilde{\boldsymbol{\theta}}^{(t_{new})}}$

$\boldsymbol{D}$:

$$\tilde{\boldsymbol{s}}^{(t_{new})} \leftarrow \arg\min_{\boldsymbol{s}} \left\| \phi\left(\boldsymbol{m}^{(t)}\right) - \boldsymbol{D}\boldsymbol{s} \right\|_2^2 + \mu \left\| \boldsymbol{s} \right\|_1 \quad . \tag{4.5}$$

Since the estimate given by $\tilde{\boldsymbol{s}}^{(t_{new})}$ also serves as the coefficients over the latent policy space $\boldsymbol{L}$, a policy for the new task can immediately be predicted as: $\tilde{\boldsymbol{\theta}}^{(t_{new})} = \boldsymbol{L}\tilde{\boldsymbol{s}}^{(t_{new})}$. This zero-shot transfer learning procedure is given as Algorithm 3.

Like other zero-shot learning methods, our approach works by finding a common space which can be used to relate a novel task to known models. Where other groups have used semantic space (Palatucci et al., 2009) or word vectors (Socher et al., 2013) as the common space, this work uses task descriptors to relate different tasks. Since my work was published, other groups have adopted the idea of using task descriptions to share knowledge between tasks (Oh et al., 2017).

## 4.4 Theoretical Analysis

This section examines theoretical issues related to incorporating task descriptors into lifelong learning via coupled dictionaries. The theoretical analysis of TaDeLL was led by my co-author Mohammad Rostami (Isele et al., 2016b). This section begins by outlining the theory that supports why the inclusion of task features can improve

performance of the learned policies and enable zero-shot transfer to new tasks safely. Then I present the convergence guarantees of TaDeLL.

### 4.4.1 Connections to Mutual Coherence in Sparse Coding

To analyze the policy improvement, since the policy parameters are factored as $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$, the next step shows that incorporating the descriptors through coupled dictionaries can improve both $\boldsymbol{L}$ and $\boldsymbol{S}$. In this analysis, the concept of *mutual coherence* is employed, which has been studied extensively in the sparse recovery literature (Donoho et al., 2006). Mutual coherence measures the similarity of a dictionary's elements as

$$M(\boldsymbol{W}) = \max_{1 \leq i \neq j \leq k} \left( \frac{|\boldsymbol{w}_i^\top \boldsymbol{w}_j|}{\|\boldsymbol{w}_i\|_2 \|\boldsymbol{w}_j\|_2} \right) \in [0, 1] \ ,$$

where $\boldsymbol{w}_i$ is the $i^{th}$ column of a dictionary $\boldsymbol{W} \in \mathbb{R}^{d \times k}$. If $M(\boldsymbol{W}) = 0$, then $\boldsymbol{W}$ is an invertible orthogonal matrix and so sparse recovery can be solved directly by inversion; $M(\boldsymbol{W}) = 1$ implies that $\boldsymbol{W}$ is not full rank and a poor dictionary. Intuitively, low mutual coherence indicates that the dictionary's columns are considerably different, and thus a "good" dictionary that can represent a wider range of tasks, potentially yielding more knowledge transfer. This intuition is shown formally:

**Theorem 4.4.1.** *(Donoho et al., 2006) Suppose there exist noisy observations $\hat{\boldsymbol{\theta}}$ of the linear system $\boldsymbol{\theta} = \boldsymbol{W}\boldsymbol{s}$, such that $\|\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}\|_2 \leq \epsilon$. Let $\boldsymbol{s}^*$ be a solution to the system, and let $v = \|\boldsymbol{s}\|_0$. If $v < 0.5(1 + M(\boldsymbol{W})^{-1})$, then $\boldsymbol{s}^*$ is the unique sparsest solution of the system. Moreover, if $\boldsymbol{s}^+$ is the LASSO solution for the system from the noisy observations, then: $\|\boldsymbol{s}^* - \boldsymbol{s}^+\|_2^2 \leq \frac{4\epsilon^2}{1 - M(\boldsymbol{W})(4v-1)}$.*

Therefore, an $\boldsymbol{L}$ with low mutual coherence would lead to more stable solutions of the $\boldsymbol{\theta}^{(t)}$'s against inaccurate single-task estimates of the policies (the $\boldsymbol{\alpha}^{(t)}$'s). Next it is shown that TaDeLL either lowers or does not affect the mutual coherence of $\boldsymbol{L}$.

TaDeLL alters the problem from training $\boldsymbol{L}$ to training the coupled dictionaries $\boldsymbol{L}$ and $\boldsymbol{D}$ (contained in $\boldsymbol{K}$). Let $\boldsymbol{s}^{*(t)}$ be the solution to Eq. 2.11 for task $\mathcal{Z}^{(t)}$, which is unique under sparse recovery theory, so $\left\|\boldsymbol{s}^{*(t)}\right\|_0$ remains unchanged for all tasks. Theorem 4.4.1 implies that, if $M(\boldsymbol{K}) < M(\boldsymbol{L})$, coupled dictionary learning can help with a more accurate recovery of the $\boldsymbol{s}^{(t)}$'s. To show this, note that Eq. 4.2 can also be derived as a result of an MAP estimate from a Bayesian perspective, enforcing a Laplacian distribution on the $\boldsymbol{s}^{(t)}$'s and assuming $\boldsymbol{L}$ to be a Gaussian matrix with elements drawn i.i.d: $l_{ij} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2)$. When considering such a random matrix $\boldsymbol{W} \in \mathbb{R}^{d \times k}$, Donoho & Huo (2001) proved that asymptotically $M(\boldsymbol{W}) \propto \sqrt{\frac{\log(dk)}{d}}$ as $d \to \infty$. Using this as an estimate for $M(\boldsymbol{L})$ and $M(\boldsymbol{K})$, since incorporating task descriptors increases $d$, asymptotically $M(\boldsymbol{K}) < M(\boldsymbol{L})$, implying that TaDeLL learns a superior dictionary. Moreover, if $M(\boldsymbol{D}) \leq M(\boldsymbol{L})$, the theorem implies $\boldsymbol{D}$ alone can be used to recover the task policies through zero-shot transfer.

To show that task features can also improve the sparse recovery, the following theorem about LASSO is used:

**Theorem 4.4.2.** *(Negahban et al., 2009) Let $\boldsymbol{s}^*$ be a unique solution to the system $\boldsymbol{\theta} = \boldsymbol{W}\boldsymbol{s}$ with $\|\boldsymbol{s}\|_0 = K$ and $\boldsymbol{W} \in \mathbb{R}^{d \times k}$. If $\boldsymbol{s}^+$ is the LASSO solution for the system from noisy observations, then with high probability: $\|\boldsymbol{s}^* - \boldsymbol{s}^+\|_2 \leq c' \sqrt{K \frac{\log k}{d}}$ , where the constant $c' \in \mathbb{R}^+$ depends on properties of the linear system and observations.*

This theorem shows that the error reconstruction for LASSO is proportional to $\frac{1}{\sqrt{d}}$. Incorporating the descriptor through $\boldsymbol{\beta}^{(t)}$, the right-hand side denominator increases

from $d$ to $(d + d_m)$ while $K$ and $k$ remain constant, yielding a tighter fit. Therefore, task descriptors can improve learned dictionary quality and sparse recovery accuracy. To ensure an equivalently tight fit for $\boldsymbol{s}^{(t)}$ using either policies or descriptors, Theorem 4.4.2 suggests it should be that $d_m \geq d$ to ensure that zero-shot learning yields similarly tight estimates of $\boldsymbol{s}^{(t)}$.

### 4.4.2 Theoretical Convergence of TaDeLL

In this section, the convergence of TaDeLL is proven, showing that the learned dictionaries become increasingly stable as it learns more tasks. The proof builds upon the theoretical results from Bou Ammar et al. (2014), Ruvolo & Eaton (2013b), and Mairal et al. (2009), demonstrating that these results apply to coupled dictionary learning with task descriptors, and use them to prove convergence.

Let $\hat{g}_T(\mathrm{L})$ represent the sparse coded approximation to the MTL objective, which can be defined as:

$$\hat{g}_T(\boldsymbol{L}) = \frac{1}{T}\sum_{t=1}^{T} \|\boldsymbol{\alpha}^{(t)} - \boldsymbol{L}\boldsymbol{s}^{(t)}\|^2_{\boldsymbol{\Gamma}^{(t)}} + \mu\|\boldsymbol{s}^{(t)}\|_1 + \lambda\|\boldsymbol{L}\|^2_{\mathsf{F}} \ .$$

This equation can be viewed as the cost for $\boldsymbol{L}$ when the sparse coefficients are kept constant. Let $\boldsymbol{L}_T$ be the version of the dictionary $\boldsymbol{L}$ obtained after observing $T$ tasks. Intuitively the convergence proof states that as more tasks are encountered, each new task is less likely to drastically change the model. This is shown formally in the following theorem:

**Theorem 4.4.3.** *(Ruvolo and Eaton, 2013b)*

1. *The trained dictionary* $\boldsymbol{L}$ *is stabilized over learning with rate:* $\boldsymbol{L}_T - \boldsymbol{L}_{T-1} = O(\frac{1}{T})$
2. $\hat{g}_T(\boldsymbol{L}_T)$ *converges almost surely.*
3. $\hat{g}_T(\boldsymbol{L}_T) - \hat{g}_T(\boldsymbol{L}_{T-1})$ *converges almost surely to zero.*

This theorem requires two conditions:

1. The tuples $\boldsymbol{\Gamma}^{(t)}$, $\boldsymbol{\alpha}^{(t)}$ are drawn i.i.d. from a distribution with compact support to bound the norms of $\boldsymbol{L}$ and $\boldsymbol{s}^{(t)}$.
2. For all $t$, let $\boldsymbol{L}_{\aleph}$ be the subset of the dictionary $\boldsymbol{L}_t$, where only columns corresponding to non-zero elements of $\boldsymbol{s}^{(t)}$ are included. Then, all eigenvalues of the matrix $\boldsymbol{L}_{\aleph}^{\mathsf{T}}\boldsymbol{\Gamma}^{(t)}\boldsymbol{L}_{\aleph}$ need to be strictly positive.

Bou Ammar et al. (2014) show that both of these conditions are met for the lifelong learning framework given in Eqs. 2.12–2.15. When incorporating the task descriptors into this framework, $\boldsymbol{\alpha}^{(t)} \rightarrow \boldsymbol{\beta}^{(t)}$, $\boldsymbol{L} \rightarrow \boldsymbol{K}$, and $\boldsymbol{\Gamma}^{(t)} \rightarrow \boldsymbol{V}^{(t)}$ are altered. Note both $\boldsymbol{\beta}^{(t)}$ and $\boldsymbol{V}^{(t)}$ are formed by adding deterministic entries and thus can be considered to be drawn i.i.d. (because $\boldsymbol{\Gamma}^{(t)}$ and $\boldsymbol{\alpha}^{(t)}$ are assumed to be drawn i.i.d.). Therefore, incorporating task descriptors does not violate Condition 1.

To show that Condition 2 holds, analogously if $\boldsymbol{K}_{\aleph}$ is formed, then the eigenvalues of $\boldsymbol{K}_{\aleph}$ are strictly positive because they are either eigenvalues of $\boldsymbol{L}$ (which are strictly positive according to Bou Ammar et al. (2014)) or the regularizing parameter $\zeta$ by

definition. Thus, both conditions are met and convergence follows directly from Theorem 4.4.3.

### 4.4.3 Computational Complexity

In this section, the computational complexity of TaDeLL is analyzed. Each update begins with one PG step to update $\boldsymbol{\alpha}^{(t)}$ and $\boldsymbol{\Gamma}^{(t)}$ at a cost of $O(\Xi(d, n_t))$, where $\Xi()$ depends on the base PG learner and $n_t$ is the number of trajectories obtained for task $\mathcal{Z}^{(t)}$. The cost of updating $\boldsymbol{L}$ and $\boldsymbol{s}^{(t)}$ alone is $O(k^2d^3)$ (Ruvolo and Eaton, 2013b), and so the cost of updating $\boldsymbol{K}$ through coupled dictionary learning is $O(k^2(d+d_m)^3)$. This yields an overall per-update cost of $O(k^2(d + d_m)^3 + \Xi(d, n_t))$, which is independent of $T$.

Next, the benefits of TaDeLL are empirically demonstrated on a variety of different learning problems.

## 4.5 Evaluation on Reinforcement Learning Domains

TaDeLL is applied to a series of RL problems, considering the problem of learning a collection of different, related systems. For these systems, three benchmark control problems and an application to quadrotor stabilization are used.

### 4.5.1 Benchmark Dynamical Systems

**Spring Mass Damper (SM)** The SM system is described by three parameters: the spring constant, mass, and damping constant. The system's state is given by the position and velocity of the center of mass. The controller applies a force to the mass, attempting to stabilize it to a given position. The SM system is a canonical benchmark because it is a second-order system that models a large number of natural phenomena and is frequently exploited in the design of many man-made objects like RLC circuits.

**Cart Pole (CP)** The CP system involves balancing an inverted pendulum by applying a force to the cart. The system is characterized by the cart and pole masses, pole length, and a damping parameter. The states are the position and velocity of the cart and the angle and rotational velocity of the pole.

**Bicycle (BK)** This system focuses on keeping a bicycle balanced upright as it rolls along a horizontal plane at constant velocity. The system is characterized by the bicycle mass, $x$- and $z$-coordinates of the center of mass, and parameters relating to the shape of the bike (the wheelbase, trail, and head angle). The state is the bike's tilt and its derivative; the actions are the torque applied to the handlebar and its derivative.

### 4.5.2 Methodology

In each domain 40 tasks were generated, each with different dynamics, by varying the system parameters. The reward for each task was taken to be the distance between the

current state and the goal. For lifelong learning, tasks were encountered consecutively with repetition, and learning proceeded until each task had been seen at least once. The same random task order between methods was used to ensure fair comparison. The learners sampled trajectories of 100 steps, and the learning session during each task presentation was limited to 30 iterations. For MTL, all tasks were presented simultaneously. Natural Actor Critic (Peters and Schaal, 2008) was used as the base learner for the benchmark systems and episodic REINFORCE (Williams, 1992) for quadrotor control. The regularization parameters $k$ and were chosen independently for each domain to optimize the combined performance of all methods on 20 held-out tasks, and $\zeta = mean(diag(\zeta^{(t)}))$ was set to balance the fit to the descriptors and the policies. Learning curves were averaged for each of the 40 tasks, and the experiment was repeated for six trials with different random orderings. The system parameters for each task were used as the task descriptor features $\phi(\boldsymbol{m})$; several non-linear transformations were also tried as $\phi(\cdot)$, but it was found that the linear features worked well.

### 4.5.3 Diversity in Tasks

When considering transfer, an important question is "how related are the tasks?" This is a deceptively nuanced question as similar looking tasks may require very different policies. Likewise, tasks that appear very different might have underlying principles that make them very similar. While the notion of task similarity is an open problem (Kifer et al., 2004; Lopez-Paz et al., 2017), an empirical measure of the diversity of tasks is to look at how well an arbitrary policy does on a different task.

Twenty-five systems are examined, where the policy of the single-task learner is com-

pared against the policy of a single-task learner trained on a different task. The number of tasks was reduced from the 40 tasks of earlier experiments due to increased computation required to test each policy on every other task. Results are averaged across 10 trials with different random seeds for generating the tasks. Table 4.1 shows the difference in performance of a learned single-task policy on the task it was trained on, and the performance when copied to other related tasks.

Table 4.1: Variety in Tasks

| Method | SM | CP | BK | Quadrotor |
|---|---|---|---|---|
| Learned Policy | $-2.367 \pm 0.001$ | $-13.69 \pm 0.02$ | $-8.1 \pm 0.1$ | $-2.31 \pm 0.01$ |
| Copied Policy | $-2.4504 \pm 0.0002$ | $-15.908 \pm 0.004$ | $-9.91 \pm 0.05$ | $-2.73 \pm 0.03$ |

The drop in performance using the copied policy demonstrates the diversity of tasks; *i.e.*, if they were identical tasks, performance would not change. Copying cart-pole policies transfers little (basically nothing) and is roughly equal to a random initialization. Copied bike policies drops roughly a third of the performance increase seen from single-task learning. The spring mass systems are the most similar, but there is still a statistically significant drop in performance from using a copied policy.

Quadrotors behave similarly to cart-pole: a copied policy performs similarly to a random initialization. Note that these experiments were conducted with different random seeds generating a different set of tasks, so the results differ slightly for the single task learning results presented in other sections.

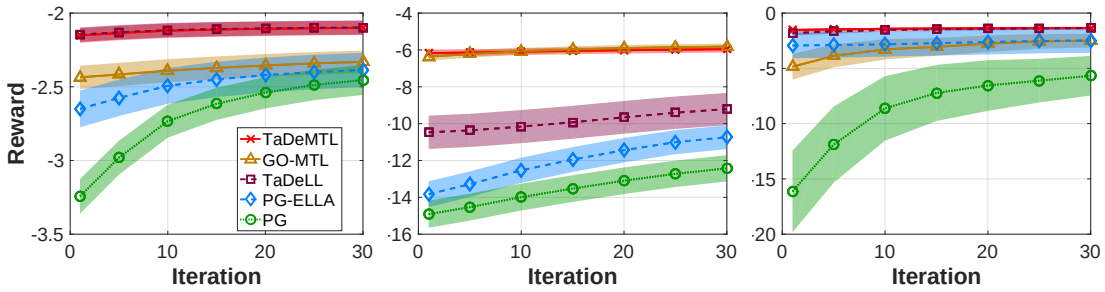
### 4.5.4 Comparison Method

In our experiments TaDeLL is compared against the related approach of Sinapov et al. (2015). Sinapov et al. uses task descriptors to estimate the transferability between

each pair of tasks for transfer learning. Each task policy is used as a initialization for every other task. Let $r_k^{baseline}$ be the reward after training on task $j$ for $k$ steps, and let $r_k^{transfer}$ be the reward after transferring task $i$ and performing $k$ steps of additional training on task $j$. The jumpstart benefit $\mathcal{B}_m(i,j)$ observed after $m$ steps of training when transferring task $i$ to task $j$ is measured:

$$\mathcal{B}_m(i,j) = \frac{\sum_{k=1}^{m}(r_k^{transfer} - r_k^{baseline})}{m} \quad .$$

After computing the benefit of transferring each task to every other task, Sinapov et al. learn a linear regression model that uses the normalized change in task features between task $i$ and task $j$ to predict the benefit $\mathcal{B}_m(i,j)$. Given the descriptor for a new task, they identify the source task with the highest predicted transferability, and use that source task for a warm start in reinforcement learning. Though effective, their approach is computationally expensive, since they estimate the transferability for every task pair through repeated simulation. Their evaluation is also limited to a transfer learning setting, and they do not consider the effects of transfer over consecutive tasks or updates to the transferability model, as is done in the lifelong setting.



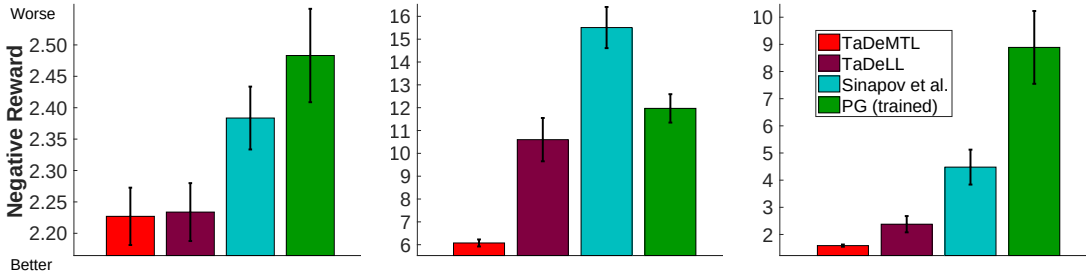

(a) Simple Mass (b) Cart Pole (c) Bicycle

Figure 4.2: Performance of multi-task (solid lines), lifelong (dashed), and single-task learning (dotted) on benchmark dynamical systems. (Best viewed in color.)

### 4.5.5 Results on Benchmark Systems

Figure 4.2 compares our TaDeLL approach for lifelong learning with task descriptors to a) PG-ELLA (Bou Ammar et al., 2014), which does not use task features, b) GO-MTL (Kumar and Daumé, 2012), the MTL optimization of Eq. 2.11, and c) single-task learning using PG. For comparison, an offline MTL optimization of Eq. 4.2 via alternating optimization was also performed, this algorithm is called Task Descriptors for Multi-Task Learning (TaDeMTL). The shaded regions on the plots denote standard error bars.

It is observed that task descriptors improve lifelong learning on every system, even driving performance to a level that is unachievable from training the policies from experience alone via GO-MTL in the SM and BK domains. The difference between TaDeLL and TaDeMTL is also negligible for all domains except CP, demonstrating the effectiveness of our online optimization.




(a) Simple Mass (b) Cart Pole (c) Bicycle

Figure 4.3: Zero-shot transfer to new tasks. The figure shows the initial *jumpstart* improvement on each task domain. (Best viewed in color.)

To measure zero-shot performance, an additional 40 tasks were generated for each domain, averaging results over these new tasks. Figure 4.3 shows that task descriptors are effective for zero-shot transfer to new tasks. It is observed that TaDeLL improves

the initial performance (i.e., the "jumpstart" (Taylor and Stone, 2009)) on new tasks, outperforming Sinapov et al.'s method and single-task PG, which was allowed to train on the task. The especially poor performance of Sinapov et al. on CP is attributed to the fact that the CP policies differ substantially. In domains where the source policies are vastly different from the target policies, Sinapov et al.'s algorithm does not have an appropriate source to transfer. Their approach is also much more computationally expensive (quadratic in the number of tasks) than our approach (linear in the number of tasks), as shown in Figure 4.8. Details of the runtime experiments are included in Section 4.6.2. Figure 4.4 shows that the zero-shot policies can be used effectively as a warm start initialization for a PG learner, which is then allowed to improve the policy. Even if the zero-shot policy is better than a learned single-task policy, the zero-shot policy can sometimes be further improved. This is the case in the cart-pole experiments where additional training improves the policy. In the bicycle experiments the zero-shot policy is close to optimal, and little improvement is seen with additional training.

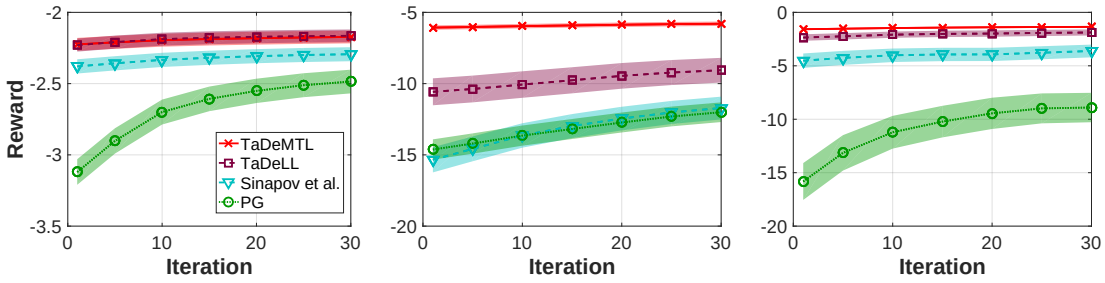



Figure 4.4: Learning performance of using the zero-shot policies as warm-start initializations for PG. The performance of the single-task PG learner is included for comparison. (Best viewed in color.)

### 4.5.6 Application to Quadrotor Control

TaDeLL was also applied to the more challenging domain of quadrotor control, focusing on zero-shot transfer to new stability tasks. To ensure realistic dynamics, the model of Bouabdallah and Siegwart (2005) was used. This model has been verified on physical systems. The quadrotors are characterized by three inertial constants and the arm length, with their state consisting of roll, pitch, yaw, and their derivatives.

Figure 4.5 shows the results of our application, demonstrating that TaDeLL can predict a controller for new quadrotors through zero-shot learning that has equivalent accuracy to PG, which had to train on the system. As with the benchmarks, TaDeLL is effective for warm-start learning with PG.

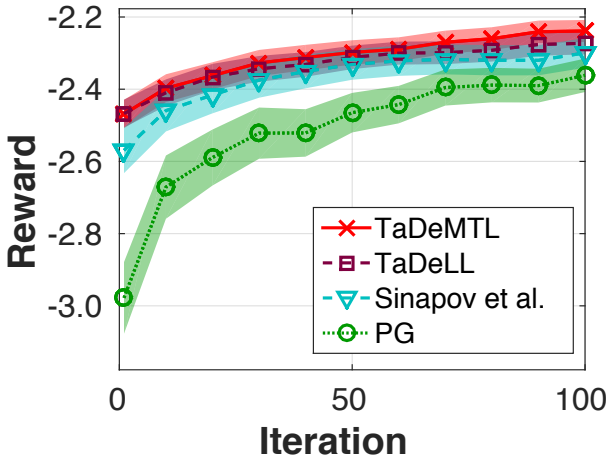



Figure 4.5: Warm-start learning on quadrotor control. (Best viewed in color.)

### 4.5.7 Experiments on Synthetic Domains

To better understand the connections between TaDeLL's performance and the structure of the tasks, TaDeLL is evaluated on two synthetic classification domains. The

| Algorithm | Lifelong Learning | Zero-Shot Prediction |
|---|---|---|
| TaDeLL | $0.926 \pm 0.004$ | $0.930 \pm 0.002$ |
| ELLA | $0.814 \pm 0.008$ | N/A |
| STL | $0.755 \pm 0.009$ | $0.762 \pm 0.008$ |

Table 4.2: Classification accuracy on Synthetic Domain 1

use of synthetic domains allows us to tightly control the task generation process and the relationship between the target model and the descriptor.

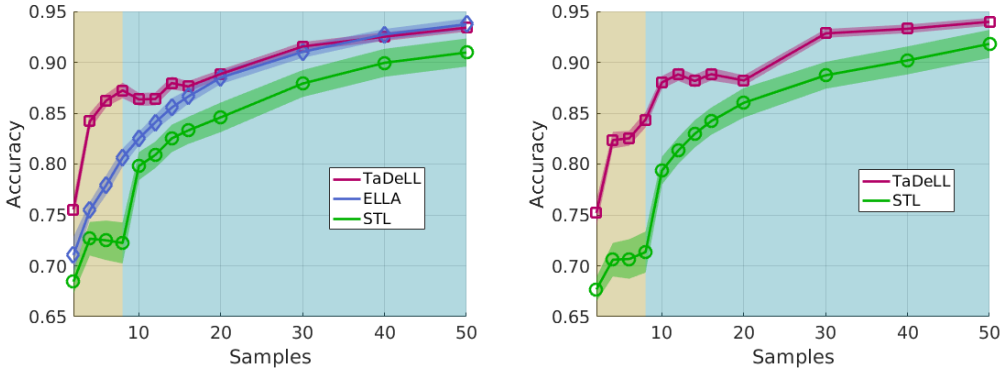
The first synthetic domain consists of binary-labeled instances drawn from $\mathbb{R}^8$, and each sample $\boldsymbol{x}$ belongs to the positive class iff $\boldsymbol{x}^\top \boldsymbol{m} > 0$. Each task has a different parameter vector $\boldsymbol{m}$ drawn from the uniform distribution $\boldsymbol{m} \in [-0.5, 0.5]$; these vectors $\boldsymbol{m}$ are also used as the task descriptors. This first experiment looks at the case where the task description is directly related to the policy being learned. Note that by sampling $\boldsymbol{m}$ from the uniform distribution, this domain violates the assumptions of ELLA that the samples are drawn from a common set of latent features. Each task's data consists of 10 training samples, and 100 tasks were generated to evaluate lifelong learning.

Table 4.2 shows the performance on this Synthetic Domain 1. The inclusion of meaningful task descriptors enables TaDeLL to learn a better dictionary than ELLA in a lifelong learning setting. An additional 100 unseen tasks were also generated to evaluate zero-shot prediction, which is similarly successful.

For the second synthetic domain, $\boldsymbol{L}$ and $\boldsymbol{D}$ matrices were generated, and then generated a random sparse vector $\boldsymbol{s}^{(t)}$ for each task. The true task model is then given by a logistic regression classifier with $\boldsymbol{\theta}^{(t)} = \boldsymbol{L}\boldsymbol{s}^{(t)}$. This generation process directly follows the assumptions of ELLA and TaDeLL, where $\boldsymbol{D}$ is generated independently. One hundred tasks were similarly generated for lifelong learning and another 100 un-

| Algorithm | Lifelong Learning | Zero-Shot Prediction |
|---|---|---|
| TaDeLL | $0.889 \pm 0.006$ | $0.87 \pm 0.01$ |
| ELLA | $0.821 \pm 0.007$ | N/A |
| STL | $0.752 \pm 0.009$ | $0.751 \pm 0.009$ |

Table 4.3: Classification accuracy on Synthetic Domain 2




(a) Lifelong Learning (b) Zero-Shot Prediction

Figure 4.6: Performance versus sample complexity on Synthetic Domain 2. The yellow (blue) background indicates that the number of samples is fewer (more) than the number of features.

seen tasks for zero-shot prediction, and use the true task models to label 10 training points per task. This experiment empirically demonstrated that TaDeLL works in the case of this assumption (Table 4.3) in both lifelong learning and zero-shot prediction settings.

This domain is also used to investigate performance versus sample complexity, as varying amounts of training data were generated per task. In Figure 4.6(a), it is observed that TaDeLL is able to greatly improve performance given only a small number of samples, and as expected, its benefit becomes less dramatic as the single-task learner receives sufficient samples. Figure 4.6(b) shows similar behavior in the zero-shot case.

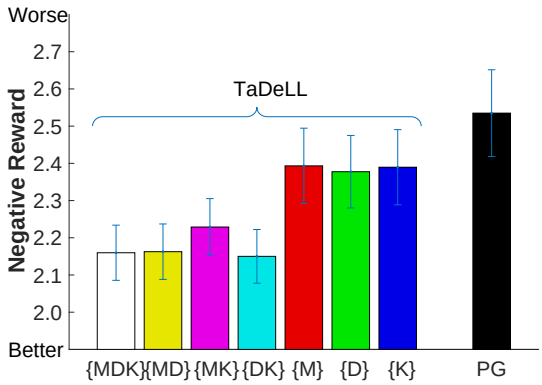



Figure 4.7: Performance using various subsets of the SM system parameters (mass $M$, damping constant $D$, and spring constant $K$) as the task descriptors.

## 4.6 Additional Experiments

Having shown how TaDeLL can improve learning in a variety of settings, attention is now turned to understanding other aspects of the algorithm. Specifically, I investigate the issue of task descriptor selection and partial information, runtime comparisons, and the effect of varying the number of tasks used to train the dictionaries.

### 4.6.1 Choice of Task Descriptor Features

For reinforcement learning problems, the system parameters were used as the task description. While, in the situations studied, this made the choice of task descriptors straightforward, this might not always be the case. It is unclear exactly how the choice of task descriptor features might affect the resulting performance. To examine this, experiments are conducted in situation where there was only partial knowledge of the system parameters.

The question of partial knowledge was investigated by conducting additional exper-

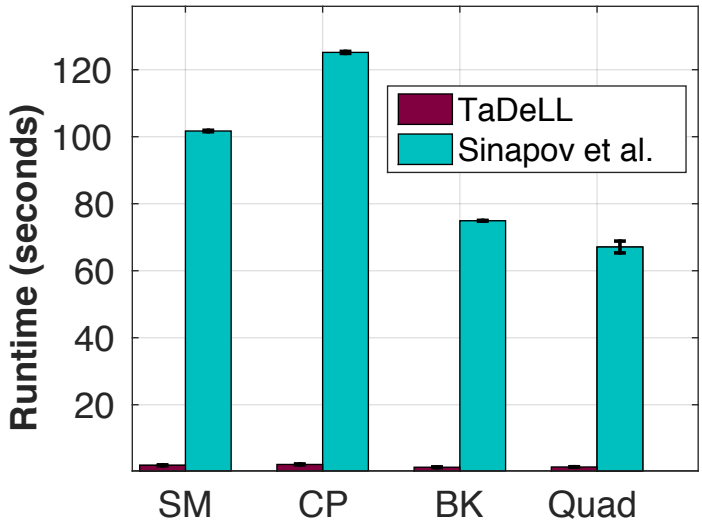



Figure 4.8: Runtime comparison.

iments on the SM system, using all the subsets of task descriptor features when learning the coupled dictionaries. Figure 4.7 shows how the number and selection of parameters affected performance on the SM domain. Jumpstart performance is evaluated when using all possible subsets of the system parameters as the task descriptor features. The subsets of the SM system parameters (mass $M$, damping constant $D$, and spring constant $K$) are shown along the horizontal axis for the task descriptors. Overall, the results show that the learner performs better when using larger subsets of the system parameters as the task descriptors. This is not surprising. As long as all the features are informative, the addition of more features allows the system to model a more accurate policy.

### 4.6.2 Computational Efficiency

The average per-task runtime of TaDeLL is compared to that of Sinapov et al. (2015), the most closely related method to our approach. Since Sinapov et al.'s method

requires training transferability predictors between all pairs of tasks, its total runtime grows quadratically with the number of tasks. In comparison, our online algorithm is substantially more efficient. As shown in Section 4.4.3, the per-update cost of TaDeLL is $O\left(k^2(d+m)^3 + \xi(d, n_t)\right)$. Note that this per-update cost is independent of the number of tasks $T$, giving TaDeLL a total runtime that scales linearly in the number of tasks.

Figure 4.8 shows the per-task runtime for each algorithm based on a set of 40 tasks, as evaluated on an Intel Core I7-4700HQ CPU. TaDeLL samples tasks randomly with replacement and terminates once every task has been seen. For Sinapov et al., 10 PG iterations were used for calculating the warm start, ensuring fair comparison between the methods. These results show a substantial reduction in computational time for TaDeLL: two orders of magnitude over the 40 tasks. These results are expected given the increased computational complexity associated with Sinapov's algorithm.

### 4.6.3 Performance for Various Numbers of Tasks

Although it was shown in Section 4.4.2 that the learned dictionaries become more stable as the system learns more tasks, it is not currently guaranteed that this will improve the performance of zero-shot transfer. An additional set of experiments was conducted on the Simple-Mass domain to evaluate the effect of the number of tasks on zero-shot performance. Our results, shown in Figure 4.9, reveal that zero-shot performance does indeed improve as the dictionaries are trained over more tasks. This improvement is most stable and rapid in an MTL setting, since the optimization over all dictionaries and task policies is run to convergence, but TaDeLL also shows clear improvement in zero-shot performance as $T_{max}$ increases. Since zero-shot transfer

involves only the learned coupled dictionaries, it can be concluded that the quality of these dictionaries for zero-shot transfer improves as the system learns more tasks.

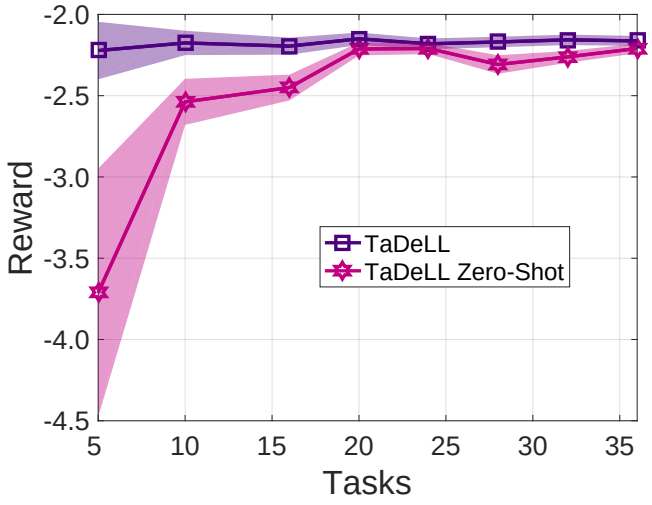



Figure 4.9: Zero-shot performance as a function of the number of spring mass tasks used to train the dictionary. As more tasks are used, the performance of zero-shot transfer improves.

## 4.7 Discussion

This chapter demonstrated how incorporating high-level task descriptors into lifelong learning both improves learning performance and also enables zero-shot transfer to new tasks. The mechanism of using a coupled dictionary to connect the task descriptors with the learned models is relatively straightforward, yet highly effective in practice. Critically, it provides a fast and simple mechanism to predict the model or policy for a new task via zero-shot learning, given only its high-level task descriptor. While the focus was on RL tasks here, this approach is general and can handle multiple learning paradigms, including classification and regression. Experiments demonstrate that our approach outperforms the state-of-the-art and requires

substantially less computational time than competing methods.

The ability to rapidly bootstrap policies for new tasks is critical to the development of lifelong learning systems that will be deployed for extended periods in real environments and tasked with handling a variety of learning problems. High-level descriptions provide an effective way to relate different models. The description could be provided by another agent, or as in the case when a person reads an instruction manual, the description can come from information in the environment. Enabling lifelong learning systems to similarly take advantage of these high-level descriptions provides an effective step toward their practical effectiveness.

The two major limitations of TaDeLL are the restriction to linear policies and the additional need for task descriptions. In certain instances, a mechanism like a description is *necessary* for handling multiple tasks. Consider a situation when state and action spaces are identical between tasks but the goal is different. The agent must rely on a description, known objective, or reward function to disambiguate tasks. However, in domains with high-dimensional state representations, the uniqueness of the state space can often act as a description to orient the agent. The next chapter considers a learner in such a high-dimensional state space. To handle high-dimensional state spaces, and additionally to allow for more expressive models, I shift the focus to deep learning and investigate transfer and lifelong learning issues related to neural networks. This serves to address the shortcoming of the linear policies, while the need for task descriptions is left as an open problem.

# Chapter 5

# Fine-Tuning as Transfer in Deep Learning

One of the biggest limitations of TaDeLL is the reliance on linear policies. For many problems, the relationship between the feature space and the desired output is highly non-linear. Chapter 6 presents an approach for learning multiple tasks in an online fashion using deep learning. To better frame that work, I use this chapter to present an overview of transfer learning in deep neural networks. Transfer is a main component of lifelong machine learning, and the issues related to transfer also appear in lifelong machine learning. By understanding the former, we can better understand the latter. In this chapter, I focus on properties of deep networks that are specific to transfer learning, and present issues that arise when conducting transfer with deep models.

An easy and effective way to transfer knowledge in deep networks is fine-tuning. Fine-tuning involves pre-training a network on one task and then retraining the network or *fine-tuning* it for a different task. Fine-tuning was one of the early successes of

deep learning – it was shown high-level features learned from training a network on ImageNet (Deng et al., 2009) allowed for breakthrough results in other domains where only smaller image datasets were available (Girshick et al., 2014; Hoffman et al., 2014).

One hypothesis for the success of fine-tuning is that the increased training on related tasks produces internal representations in the network that generalize to a broader set of problems and are therefore able to reduce over-fitting (Razavian et al., 2014; Caruana, 1997). Considering this phenomenon more closely, one might ask, "Can the initial training process be over-specialized and actually hurt transfer performance?" This was shown to be the case. Yosinski et al. (2014) investigated fixing layers and showed that earlier layers appear to be more general, while later layers are more specific and require retraining. In this chapter, I will explore other properties of fine-tuning. Specifically, I show that a directly copied network that performs poorly on a new task can improve performance on that new task with fine-tuning. I demonstrate that the amount of training a network receives affects how well a network performs as an initialization, and I investigate retention or how well performance on the initial task is preserved.

The problem I will use to investigate issues related to transfer in deep learning is the problem of intersection handling by autonomous vehicles. I show that a network trained on one type of intersection generally is not able to generalize to other intersections. However, a network that is pre-trained on one intersection and fine-tuned on another intersection performs better on the new task compared to training in isolation. This network also retains knowledge of the prior task, even though some forgetting occurs. Finally, I show that the benefits of fine-tuning hold when transferring *simulated* intersection handling knowledge to a *real* autonomous vehicle.

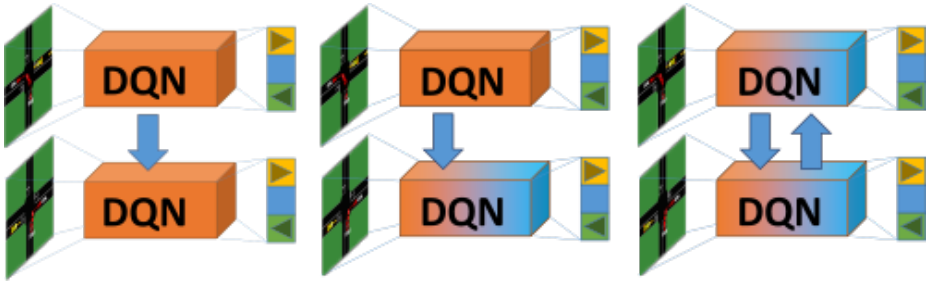



Figure 5.1: We analyze knowledge transfer between different types of intersections. The knowledge to handle an intersection is represented as a Deep Q-Network (DQN). We investigate a) directly copying a network to a new intersection b) fine-tuning a previously trained network on a new intersection, c) whether fine tuning destroys old intersection knowledge in reverse transfer.

Autonomous Driving (AD) has the potential to reduce accidents caused by driver fatigue and distraction and will enable more active lifestyles for the elderly and disabled (Blanco et al., 2016). While AD technology has made important strides over the last couple of years, current technology is still not ready for large scale roll-out (Aoude et al., 2011). Urban environments are particularly difficult for AD, due to the unpredictable nature of pedestrians and vehicles in city traffic.

Rule-based methods provide a predictable method to handle intersections. However, rule-based intersection handling approaches do not scale well due to the difficulty of designing hand-crafted rules that remain valid as the diversity and complexity of possible scenes increase. Recently it has been shown that deep reinforcement learning can improve over rule-based techniques (Isele and Cosgun, 2017a; Isele et al., 2018b), but before such techniques can be utilized, it is important to understand how well these techniques are able to generalize to different scenarios and real systems.

We explore the ability of a reinforcement learning agent to generalize, focusing specifically on how the knowledge for one type of intersection, represented as a Deep Q-Network (DQN), translates to other types of intersections. We treat both different

geometries and goal behaviors of intersections as different tasks for a lifelong learning agent. First I look at **direct copy**: how well a network trained for Task A performs on Task B. Second, I analyze how a network initialized on Task A and **fine-tuned** on Task B compares to a randomly initialized network exclusively trained on Task B. Third, I investigate **reverse transfer**: if a network pre-trained for Task A and fine-tuned to Task B, preserves knowledge for Task A. Finally, I present early results of using a network trained in simulation to initialize learning on real data.

## 5.1 Background on Autonomous Driving and Transfer in Deep Learning

Intersections require attending to many different agents in multiple different directions, with highly varied and often unpredictable behaviors. Making appropriate decisions can be challenging even for human drivers: 20% of all accidents occur at intersections (National Highway Traffic Safety Administration, 2014). Robots have the potential to be safer, with some research suggesting they might already be safer (Blanco et al., 2016; Weiland and Crow, 2018). However, in practice, intersections are considered one of the most difficult problems for autonomous vehicles (Fuerstenberg, 2005; Aoude et al., 2011). According to a presentation by Pravin Varaiya at the 2018 Berkeley Deep Drive consortium (Pravin, 2018), 23 out of 26 reported autonomous driving accidents in California occurred at intersections.

Researchers have recently been investigating using machine learning techniques to control autonomous vehicles (Cosgun et al., 2017). Imitation learning strategies have investigated learning from a human driver (Bojarski et al., 2016). Markov Deci-

sion Processes (MDP) have been used offline to address the problem of intersection handling (Brechtel et al., 2014; Song et al., 2016). Online planners based on partially observable Monte Carlo Planning (POMCP) have been applied to intersection problems when an accurate generative model is available (Bouton et al., 2017). Additionally, machine learning techniques have been used to optimize comfort in a space where solutions are constrained to safe trajectories (Shalev-Shwartz et al., 2016).

Large amounts of data often improve the performance of machine learning techniques. In the absence of huge datasets, training on multiple related tasks gives similar performance gains (Caruana, 1997). A large breadth of research has investigated transferring knowledge from one system to another in machine learning in general (Pan and Yang, 2010), and in reinforcement learning specifically (Taylor and Stone, 2009).

Large training times and high sample complexity make transfer methods particularly appealing in deep networks (Razavian et al., 2014; Yosinski et al., 2014). Recent work in deep reinforcement learning has looked at combining networks from different tasks to share information (Rusu et al., 2016; Yin and Pan, 2017). Researchers have looked at using options (Sutton and Barto, 1998) in Deep RL to expand an agent's capabilities (Jaderberg et al., 2016; Tessler et al., 2016; Kulkarni et al., 2016), and efforts have been made to enable a unified framework for learning multiple tasks through changes in architecture design (Srivastava et al., 2013) and modified objective functions (Kirkpatrick et al., 2016) to address the problem of catastrophic forgetting[1] (Goodfellow et al., 2013).

In our scenario, there is the added difficulty of transferring to the real vehicle. It is a well-known problem that policies trained in simulation rarely work on real robots

[1]Catastrophic forgetting is discussed in Section 5.3

(Barrett et al., 2010). Recent work has investigated grounding imperfect simulators to a robot's behavior (Hanna and Stone, 2017) and there is evidence that transferring from simulation to real robots can be addressed by targeting the system's ability to generalize (Tobin et al., 2017). Given the variety of intersections, the importance of learning a general model, and the difficulty of training on the real vehicle, the prospects of multi-task learning are examined for the problem of intersection handling.

## 5.2 Intersection Handling using Deep Networks

Each intersection handling task is viewed as a reinforcement learning problem, and a Deep Q-Network (DQN) is used to learn the Q-function which maps each state-action pair to its corresponding value. The details of deep Q-learning are described in detail in Section 2.1.3. We assume the vehicle is at the intersection, the path is known, and at every time step, the network is tasked with choosing between two actions: wait or go. Once the agent decides to go, it continues until it either collides or successfully navigates the intersection. My previous work has shown that deciding the wait time generally outperforms approaches that learn an entire acceleration profile (Isele and Cosgun, 2017a; Isele et al., 2018b).

## 5.3 Knowledge Transfer

We are interested in how a policy can be re-used for sharing knowledge between different driving tasks. By sharing knowledge from different tasks, it is possible to reduce learning time and create more general and capable systems. Ideally, knowledge

sharing can be extended to involve a system that continues to learn after it has been deployed (Thrun, 1996) and can enable a system to accurately predict appropriate behavior in novel situations (Isele et al., 2016b). We examine the behavior of various knowledge sharing strategies in the autonomous driving domain.

**Direct Copy** Directly copying a policy indicates the differences between tasks. To demonstrate how well a network trained on one task fits another, a network is trained on a single source task for 25,000 iterations. The *unmodified* network is then evaluated on every other task. We repeat this process, using each different task as a source task.

**Fine-Tuning** Fine-tuning allows a network to adapt from the source to the target task. Starting with a network trained for 10,000 iterations on a *source* task, a network is then fine-tune for an additional 25,000 iterations on a second *target* task. We use 10,000 iterations because it demonstrates substantial learning, but is suboptimal in order to emphasize the possible benefits gained from transfer. Fine-tuning demonstrates the *jumpstart* and *asymptotic performance* benefits described by Taylor and Stone (2009). For didactic purposes, a single task is also trained for 200,000 iterations, while intermittently testing the performance on other tasks. This is done to show how the specificity of a network over-fitting to a single task can adversely affect performance on other tasks.

**Reverse Transfer** After a network has been fine-tuned on the target task, the performance of that network is evaluated on the source task. If training on a later task improves the performance of an earlier task this is known as reverse transfer. It is known that neural networks often *forget* earlier tasks in what has been termed

catastrophic forgetting (McCloskey and Cohen, 1989; Ratcliff, 1990; Goodfellow et al., 2013). It is desirable to have a system that can learning a large variety of intersections, so investigations are run to understand how much knowledge of a previous task is preserved.

## 5.4 Deep Neural Network Setup

Our DQN uses a convolutional neural network with two convolution layers, and one fully connected layer. The first convolutional layer has 32 $6 \times 6$ filters with stride two, the second convolution layer has 64 $3 \times 3$ filters with stride two. The fully connected layer has 100 nodes. All layers use leaky ReLU activation functions (Maas et al., 2013). The final linear output layer has five outputs: a single *go* action, and a *wait* action at four time scales (1, 2, 4, and 8 time steps). The use of variable length time scales or *dynamic frame skipping* helps to speed up the learning process (Srinivas et al., 2017). The network is optimized using the RMSProp algorithm (Tieleman and Hinton, 2012). A visualization of the network is shown in Figure 5.2.

At each learning iteration a batch of 60 experiences is sampled. Since the use of an experience replay buffer imposes a delay between an experience occurring and being trained on, it is possible to calculate the return for each state-action pair in the trajectory prior to adding each step into the replay buffer. This allows us to train directly on the n-step return (Peng and Williams, 1996).

The state space of the DQN is represented as a $18 \times 26$ grid in local coordinates that denote the speed and direction of cars within the grid. Because the length and width of the input correspond to spatial dimensions, the input looks like a pixel

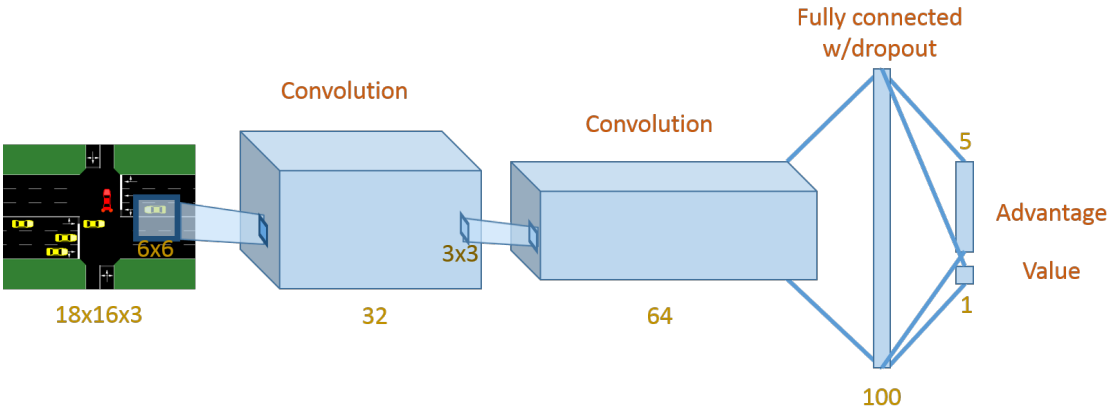



Figure 5.2: Visualization of the network used for the intersection handling domain.

representation of a bird's eye view of the intersection. A visualization of the state space is shown in Figure 5.3. The red channel corresponds to heading angle, the green channel corresponds to velocity, and the blue channel is an indicator that a car is present in the corresponding region of space. This means that cars traveling right will appear pink to white (white being the faster cars), and cars traveling left will be appear blue to teal (teal being the faster cars).

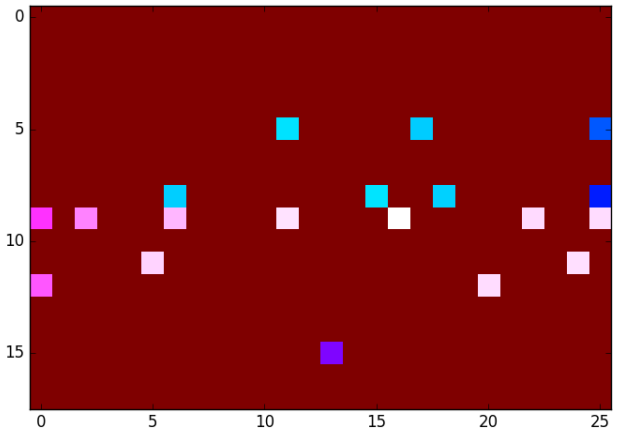



Figure 5.3: Example input representation to the network. The $18 \times 26 \times 3$ tensor uses the height and width represent spatial position and uses the three depth channels to describe heading angle and velocity with the third channel serving as an indicator.

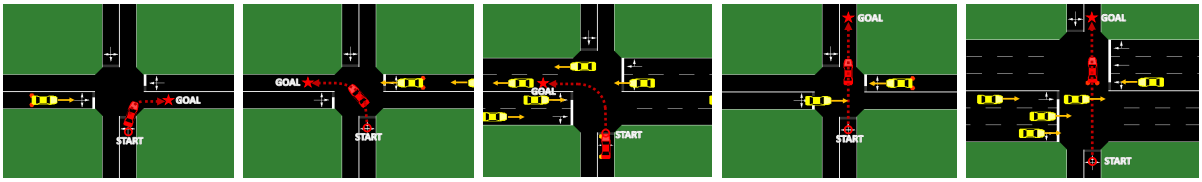
The epsilon governing random exploration is 0.05. The reward is +1 for successfully navigating the intersection, −1 for a collision, and −0.01 step cost.

## 5.5 Experimental Setup

Experiments were run using the Sumo simulator (Krajzewicz et al., 2012), which is an open-source traffic simulation package. This package allows users to model road networks, road signs, traffic lights, a variety of vehicles (including public transportation), and pedestrians. For the purpose of testing and evaluation of autonomous vehicle systems, Sumo provides tools that facilitate online interaction and vehicle control. For any traffic task, users can have control over a vehicle's position, velocity, acceleration, steering direction and can simulate motion using basic kinematic models. Traffic tasks like multi-lane intersections can be setup by defining the road network (lanes and intersections) along with specifications that control traffic conditions. To simulate traffic, users have control over the types of vehicles, road paths, vehicle density, and departure times. Traffic cars follow the intelligent driver model to control their motion. In Sumo, randomness is simulated by varying the speed distribution of the vehicles and by using parameters that control driver imperfection, based on the Krauss stochastic driving model (Krauss, 1998). The simulator runs based on a predefined time interval which controls the length of every step.

The Sumo simulator allows for the creation of a variety of different driving tasks with varying levels of difficulty, many of which benefit from transfer while being sufficiently different that adaptation is necessary for good performance (Isele and Cosgun, 2017b). Autonomous driving has multiple objectives which often conflict, such as maximizing

safety, minimizing the time to reach the goal, and minimizing the disruption of traffic. The autonomous agent chooses between two types of actions: to go or to wait for some number of iterations. I showed that this action space is preferable to learning with accelerations in the absence of occlusions (Isele and Cosgun, 2017a; Isele et al., 2018b). The path to take is assumed to be provided to the agent by a high-level controller.




(a) Right (b) Left (c) Left2 (d) Forward (e) Challenge

Figure 5.4: Visualizations of intersection tasks used for our experiments.

The experiments focus on the number of successfully completed trials. Successful trials consist of the car navigating safely through the intersection, considering both collisions and timeouts to be failures. The Sumo traffic simulator is configured so that each lane has a 45 miles per hour (20 m/s) max speed. The car begins from a stopped position. Each time step is equal to 0.2 seconds. The max number of steps per trial is capped at 100 steps (20 seconds). The traffic density is set by the probability that a vehicle will be emitted randomly per second. We use a departure probability of 0.2 for *each lane* for all tasks.

While navigating intersections involves multiple conflicting metrics (including time to cross, number of collisions, and disruption to traffic), this investigation focuses on the percentage of trials the vehicle successfully navigates the intersection. All simulated state representations ignore occlusion, assuming all cars are always visible.

We ran experiments using five different intersection tasks; each of these tasks is depicted in Figure 5.4:

- *Right* involves making a right turn
- *Left* involves making a left turn
- *Left2* involves making a left turn across two lanes
- *Forward* involves crossing the intersection
- *Challenge* involves crossing a six-lane intersection.

### 5.5.1 Real Data

We conducted a preliminary study in order to evaluate whether knowledge obtained by simulated intersections could be useful for real ones. We collected data from an autonomous vehicle in Mountain View, California, at an unsigned T-junction, similar to the *Left* scenario. An image of the intersection is shown in Figure 5.5. A point cloud, obtained by a combination of six IBEO LIDAR sensors, is first pre-processed to remove points that reside outside the road boundaries. A clustering method with hand-tuned geometric thresholds is used for vehicle detection. Each vehicle is tracked

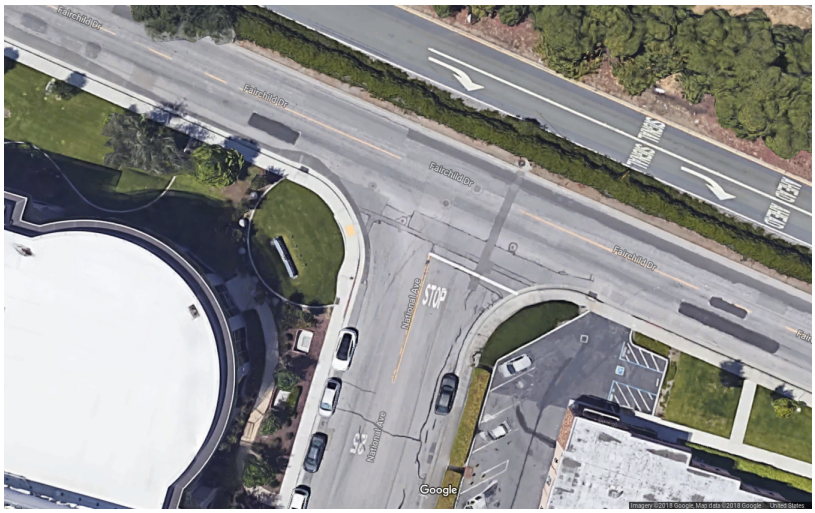



Figure 5.5: Image of the intersection where data was recorded.

by a separate particle filter. During data collection, a human observer in the vehicle labeled whether making a left turn would be safe or not. Given a random starting point in the recording, the system is able to select *wait* actions that move it ahead in the recording, and a go action which results in either a collision or a success based on the human provided labels. Because data is collected at a higher sampling rate than the simulation step frequency (and therefore the behavior frequency), states are sampled from within the simulation step window, allowing a recording to be inflated into a large number of experiments. Note that this process only gives real sensor readings, and that the system is not able to observe how its behavior affects other drivers. Because the same few recorded scenarios are replayed, it is expected that training on recorded data in this way will overfit to the recording.

We train a network on approximately one minute of recorded data, during which time approximately 20 cars drive past. We then test the network on a separate recording made at the same intersection. We compare the results against a network that has been pre-trained on simulation data and then fine-tuned on the real data.

## 5.6 Results

**Direct Copy:** Figure 5.6 shows the average performance of training on one task and applying it to another in light gray. While only the average performance is plotted, the quality of transfer is dependent on the particular source and target task. In no instance does a network trained on a different task surpass the performance of a network trained on the matching task, but several tasks achieve similar performance with transfer. Particularly it is seen that each network trained on a single-lane task

(right, left, and forward) is consistently a top performer on other single-lane tasks. Additionally the more challenging multi-lane settings (left2 and challenge) appear related. The *Left2* network does substantially better than any of the single lane tasks on the *Challenge* task. The representation aligns the single lanes, and it is evident that this is important for transfer. Even though the multi-lane *Left2* and *Challenge* tasks have a different number of lanes, the increased overlap provides better transfer.

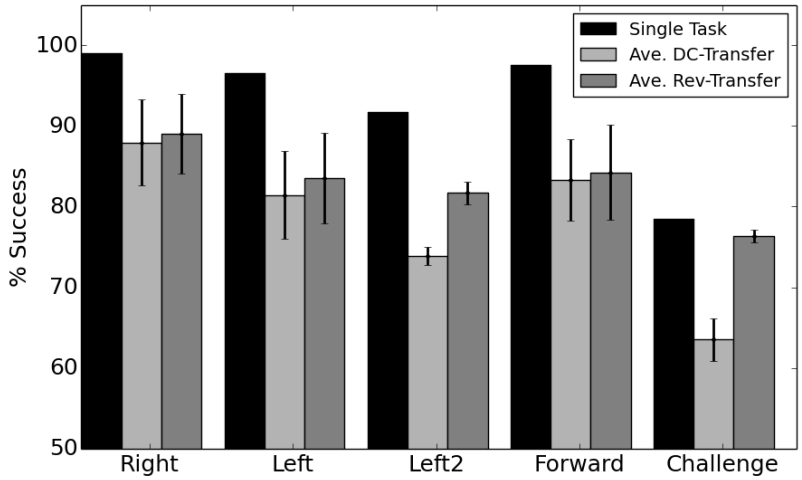
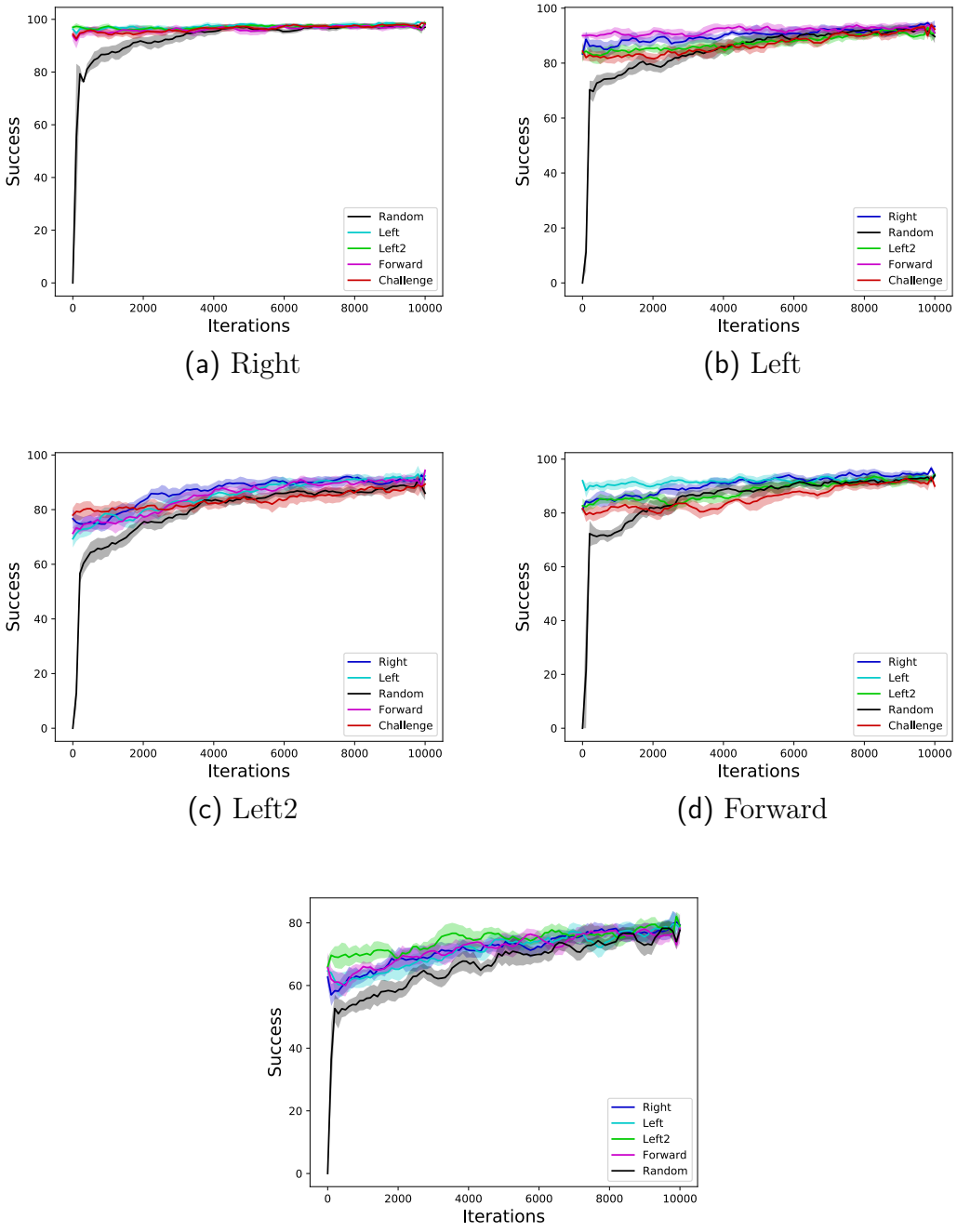



Figure 5.6: Direct Copy and Reverse Transfer. The horizontal axis denotes the test condition. Black bars show the performance of single task learning. Light gray bars show the average performance of a network trained on one task and tested on another. The drop in performance demonstrates the difference between tasks. The dark gray indicates the average performance of reverse transfer: a network is trained on the first task (listed on the horizontal axis), fine-tuned on second task, and then evaluated on the first task. All possible choices of the second task are tested and the average is shown. The drop in performance indicates catastrophic forgetting, but networks exhibit some retention of the initial task.

**Fine-Tuning:** Figure 5.7 shows the fine-tuning results. We see that in nearly all cases, pre-training with a different network gives a significant *jumpstart* advantage (Taylor and Stone, 2009) and in several cases there is an asymptotic benefit as well.



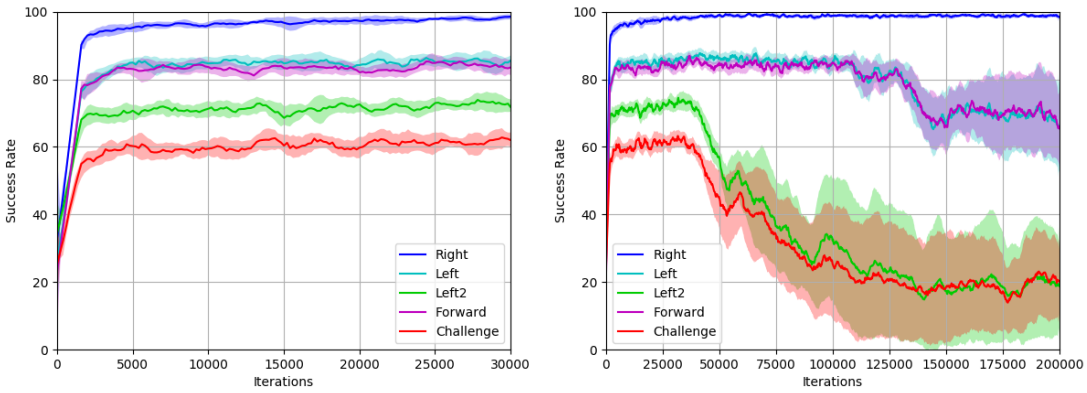

(a) Right (b) Left

(c) Left2 (d) Forward

(e) Challenge

Figure 5.7: Fine-tuning comparison. A network for one task is initialized with the network of a different task. The colored lines indicate the initialization network. The black line indicates the performance of a network with a random initialization. Results were averaged over three trials. The shaded region depicts the standard error.

The results of Figure 5.7 used a fixed-length initialization of 10,000 iterations. The amount of training on the source task does matter. 10,000 iterations was chosen to provide a good initialization but not fully learn the task. Fully training on the source task tends to hurts generalization and over-fit to the source task.




(a) Detail (b) Full training session

Figure 5.8: The effect of fine-tuning duration on transfer. The network is trained on the *Right* task (shown in blue) for 200,000 total iterations. a) A detail of the first 30,000 iterations. We observe that training on the right task improves performance of all the other tasks, appearing to asymptote at different performances for different tasks. b) As the network is continually refined on the *Right* task, performance degrades on the other tasks. Results were averaged over three trials and the shaded region depicts the standard error.

The result of over training on the source task is shown in Figure 5.8. The *Right* task is trained for 200,000 iterations while the network's performance on other tasks is intermittently tested during the training process. As the network converges to peak performance on the *Right* task, the performance on the other tasks suffers.

**Reverse Transfer:** Figure 5.6 shows the reverse transfer performance in dark gray. The performance on the source task dropped after fine-tuning on the target, but performance improved compared to direct copy. This indicates that some information was retained by the network. Note that the *Left2* and *Challenge* tasks have less

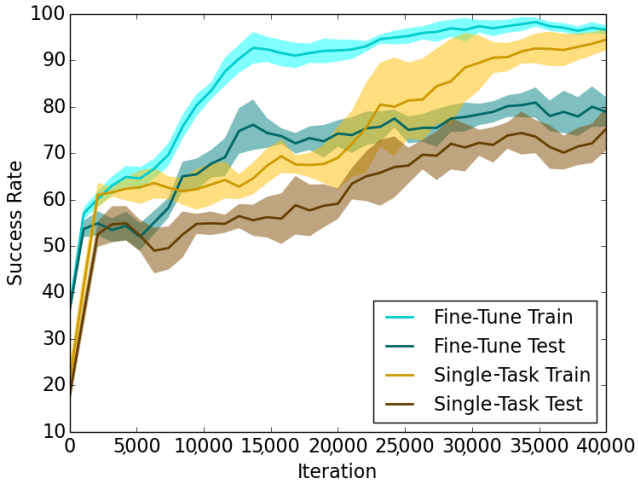



Figure 5.9: Transfer from simulation to real environment. The yellow lines indicate the training and test performance of a network trained on real data collected from an autonomous vehicle. The blue lines indicate the performance of a network that is first trained on simulated data and then fine-tuned on a real vehicle. We see that fine-tuning accelerates training time and improves generalization.

overlap with other tasks in the state space. It is possible that non-overlapping regions can remain unchanged by fine-tuning. This might explain why it is observed that the most retention occurs on the least-related tasks.

**Real Data:** Figure 5.9 shows the performance of training on real data. The blue lines indicate a network that has been pre-trained on simulated data, the yellow lines indicate the performance of a network that has only been trained on the real data. Lighter lines indicate performance on the training data, and darker lines indicate performance on the test data. The network pre-trained in simulation rises above 90% success on the training data in approximately half the iterations required by the network trained only on real data to reach the same performance. The test results follow the learning curve of the training data, but the performance asymptotes at 80% success. These results show that fine-tuning can reduce the training time of the network, however in both networks the lower performance on the test data suggests

over-fitting. While fine-tuning improves training time, it worth mentioning that the network needed to make adjustments to observe those benefits, as directly copying produced very poor performance.

It is interesting to note that training on the real task required more iterations. This may be due to imperfections in the labeling process or greater noise and variation in the state space. For completeness, there is an investigation into transfer from real data to simulation. We observed that using a network initialized on a real left turn accelerates training a left turn in simulation. Directly applying the network fine-tuned on real data to a left turn in simulation resulted in a model that consistently timed out. We suspect this is due to the network over-fitting the small amount of data.

## 5.7 Discussion

In this chapter I investigated different properties of transfer between intersections, when using Deep Q-Networks. Our results identify autonomous intersection handling as a domain that benefits from transfer. First, it was shown that the success rates were consistently low when a network is trained on Task A but directly tested on Task B. Second, a network that is initialized with the network of Task A and then fine-tuned on Task B generally performed better than a randomly initialized network. Third, when a network that is initialized with Task A, fine-tuned on Task B, and tested on Task A, it performed better than a network directly copied from Task B to Task A, but worse than a network trained recently on Task A. Fourth, it was shown that a real intersection can be treated as a separate task and transfer from simulation can be used to improve learning.

Unlike the methods presented in Chapters 3 and 4, a limitation of fine-tuning is that it transfers from one source task to one target task. The next chapter focuses on how a single network can be trained to learn multiple tasks, but the ability to selectively transfer from different source tasks and different combinations of source tasks remains an open problem. Targeting a single agent that can handle many tasks, the investigation is directed towards lifelong reinforcement learning, and more specifically to the case where tasks are allowed to change dynamically. We focus on addressing the issue of catastrophic forgetting so that the single agent preserves its performance on prior tasks. I will show how memory is important to preserve prior experiences and allow an agent to continue to perform as tasks change over time.

# Chapter 6

# Preserving Experiences

Advances in deep learning have begun to motivate research into networks that can learn multiple tasks. Some approaches include training separate networks for each task (Rusu et al., 2016; Yin and Pan, 2017) and discovering action hierarchies (Tessler et al., 2016), but these methods become inefficient as the number of tasks increases. When trying to efficiently train a single network that can handle multiple tasks, a problem emerges concerning how to retain knowledge for every task over the course of a system's lifetime.

When learning multiple tasks in sequence, each new task changes the distribution of experiences. A large set of states are no longer visited, and the changing set of transitions direct the learning process towards a moving optimum. As a result of the non-stationary training distribution, the network loses its ability to perform well in previous tasks in what has been termed *catastrophic forgetting* (McCloskey and Cohen, 1989; Ratcliff, 1990; Goodfellow et al., 2013).

To mitigate forgetting, the network or learning process could be modified, or memory could be incorporated into the system. I explore the latter. It has previously been shown that past experiences can preserved and replayed to improve learning (Lin, 1992). In deep reinforcement learning, this is typically done with a first-in-first-out (FIFO) replay buffer (Mnih et al., 2015). A network may experience forgetting on a *single task* if rarely occurring events are inadvertently pushed out of the buffer (Lipton et al., 2016) or if the network converges to behavior that does not consistently visit the state space (de Bruin et al., 2015). This can be mitigated by having a replay buffer with limitless capacity. Alternatively, if the task boundaries are known a priori, a separate buffer can be used for each task. However, as the number of tasks grows large, storing all the experiences of all tasks becomes prohibitively expensive, and ultimately infeasible for lifelong learning.

In this chapter, I develop a process for accumulating experiences online that enables their long-term retention given limited memory. Preserving prior ability is treated as an active process where the replay of prior tasks is incorporated into the current learning process. In order to efficiently store prior experiences, I propose a rank-based method for the online collection and preservation of a limited set of training examples to reduce the effects of forgetting. I then explore four different selection strategies to identify a good ranking function for selecting the experiences. To validate the selection process, I apply each strategy to an autonomous driving domain where different tasks correspond to navigating different unsigned intersections. This domain has recently been analyzed as a multi-task setting that both benefits from transfer and allows for the specification of a large variety of tasks (Isele and Cosgun, 2017b). It is additionally shown that the results generalize with further testing in a grid world navigation task. In this chapter, I propose a selective experience replay process

to augment the FIFO replay buffer to prevent catastrophic forgetting. I present and compare four selection strategies for preserving experiences over the course of a system's lifetime, and I analyze the trade-offs between the two most successful selection strategies: distribution matching and coverage maximization.

## 6.1 Catastrophic Forgetting in Neural Networks

Previous lifelong learning efforts using a single neural network have focused on preserving prior knowledge within the network. Numerous works in shallow networks investigate catastrophic forgetting for classification tasks (French, 1999). The first paper to address the topic in deep nets proposed a local winner-take-all (LWTA) activation to allow unique network configurations to be learned for different tasks (Srivastava et al., 2013). The LWTA strategy of using different subsets of the network draws parallels to dropout, which was shown to be superior to LWTA at preventing catastrophic forgetting in some domains (Goodfellow et al., 2013). Most recently, the Fisher information matrix of previous tasks has been incorporated into the objective function to selectively slow down learning on important weights (Kirkpatrick et al., 2016).

I instead consider a complementary method to these techniques by selecting and preserving experiences for future training. The idea of retraining on old experiences has been briefly suggested for a similar purpose in training recurrent neural networks (Schmidhuber, 2015). It has been shown that experience selection can improve training in the context of selecting samples for training (Schaul et al., 2015b), but that did not look at the more permanent decision of selecting which samples to keep and which samples to discard. De Bruin et al. (2016) showed some evidence to sug-

gest that retraining on old experiences can benefit transfer, but did not investigate the effects of training over repeated transfers. I believe replaying experiences is a necessary component to consolidating knowledge from multiple tasks. By preserving experiences from prior tasks and incorporating them into the learning process, the bias that exists when training on tasks in sequence can be overcome. Additionally, this approach is agnostic to task boundaries, putting less burden on the system to identify the changes of a non-stationary distribution.

### 6.1.1 Bias in Lifelong Learning

Recall from Section 2.1.3 that the loss for an individual experience in a deep Q-network (DQN) is

$$\mathcal{L}(e_i, \theta) = \left( r_i + \gamma \max_{a'_i} Q(x'_i, a'_i; \theta) - Q(x_i, a_i; \theta) \right)^2 \ . \tag{6.1}$$

In an MTL setting, this loss yields the multi-task objective

$$\frac{1}{T} \sum_{t=1}^{T} \frac{1}{n_t} \sum_{i=1}^{n_t} \mathcal{L}(e_i, \theta) \ , \tag{6.2}$$

where $n_t$ is the number experiences for task $t$, $T$ is the total number of tasks, and $\theta$ is common across for all tasks. In multi-task learning, it is typically assumed that each task is drawn from a different distribution. In many circumstances this causes experiences from a single task to be correlated with respect to the global distribution. Consider trying to model the global mean given only data from a single task for an estimator $U$,

$$\text{bias} = \mathbb{E}(U) - \mu_{global} \ . \tag{6.3}$$

As more tasks are incorporated into the estimator, the bias will reduce. But if the estimator always forgets the previous tasks, the estimator will track the current task and never converge to the true mean.

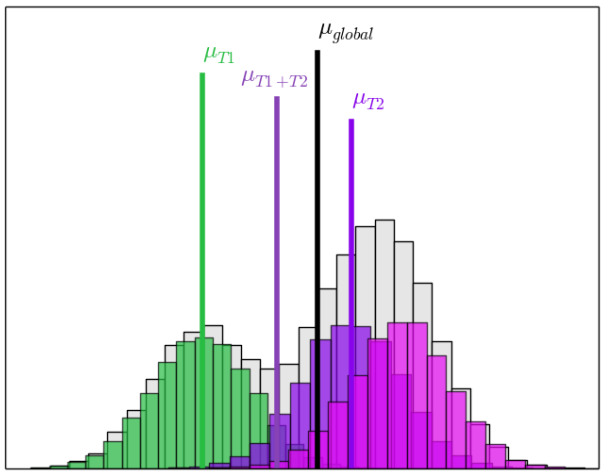



Figure 6.1: Cartoon of how training on a single task introduces bias. An estimator that only uses data from task $T_1$ produces a biased estimate of the global mean. If the estimator is allowed to accumulate data from more tasks it will approach the true mean, but if it forgets previous tasks the estimate will endlessly track the current task.

Experience replay in reinforcement learning has been noted to remove temporal correlations in the data (Mnih et al., 2015) and speed up the assignment of credit (Lin, 1992). However, when a system is learning a collection of tasks, the training time on a single task can force experiences from previous tasks out of the buffer, and as a result, the FIFO implementation of experience replay does not address task-related correlations.

When training a single network on a sequence of tasks, experiences can be preserved from previous tasks by retaining a buffer for each task, or by preserving a single buffer with unlimited capacity. Experiences can then be incorporated into the learning process by sampling from previous tasks. By preserving experiences for every task, the

network can optimize all tasks in parallel. However, preserving experiences for all tasks will quickly run into memory limitations. Assuming limited capacity, the standard FIFO implementation of an experience replay buffer will lose earlier experiences and the training distribution will drift over time.

In order to address this problem, I propose a selective process for preserving experiences. Through the use of an experience replay storage mechanism that selectively retains experiences in long-term memory, the non-stationary effects caused by using a FIFO replay buffer can be eliminated.

## 6.2 Selective Experience Replay

In biological systems, experience replay is an important aspect of learning behaviors. Research shows that blocking replay disrupts learning (Girardeau et al., 2009; Ego-Stengel and Wilson, 2010) and some scientists suggest that replay is a necessary component of memory consolidation (Carr et al., 2011).

The idea that experience replay could be beneficial in machine learning was first explored by Lin (1992) who used replay in order to speed up training by repeating rare experiences. More recently, experience replay has seen widespread adoption, since it was shown to be instrumental in the breakthrough success of deep RL (Mnih et al., 2013). But in its current form, experience replay buffers are typically implemented as randomly sampled first-in-first-out (FIFO) buffers (Mnih et al., 2015; Van Hasselt et al., 2016; Lillicrap et al., 2016).

### 6.2.1 Experience Selection

In order to efficiently store a sample population that preserves performance across tasks, I introduce a long-term storage, *episodic memory*, which accumulates experiences online, storing a subset of the system's history under the restriction of resource constraints. While episodic memory could involve generative processes (Goodfellow et al., 2014) or compression (Argyriou et al., 2008), in this work I focus specifically on the case of selecting complete experiences for storage.

Episodic memory accumulates knowledge over time which helps to preserve past performance and constrains the network to find an optimum that respects both the past and present tasks. To accumulate a collection of experiences from all tasks, experiences are ranked according to importance for preserving prior knowledge. As long as the ranking function is independent of time, the stored experiences will be as well. In practice, episodic memory is implemented with a priority queue, ensuring that the least preferred experiences are the first to be removed. Episodic memory serves as long-term storage, and is not intended to replace the short-term FIFO system.

### 6.2.2 The Importance of the FIFO Buffer

In addition to the episodic memory, a small FIFO buffer is used to ensure that the network has the opportunity to train on all incoming experiences. A system with limited storage that preserves memories over an entire lifetime will have very little capacity for new experiences, and using long-term only storage will impair the learning of new tasks[1]. Additionally, increasing the variety of experiences to which the

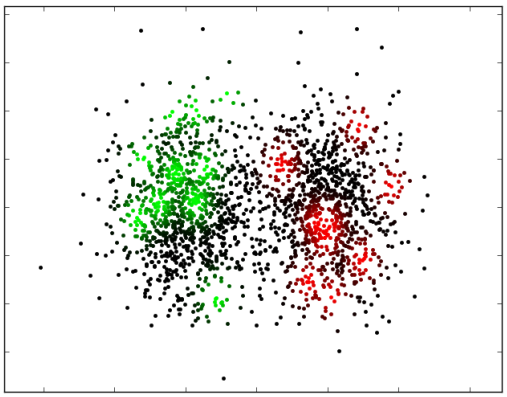
[1]We experimented with training the network using long-term storage only and confirmed this hypothesis. The network performs acceptably, but does not match the performance of the combined approach.

network is exposed prevents the network from overfitting. Note that only the long-term memory will hold experiences for all tasks. By training the network on both the long-term and short-term memories, the gradients are biased towards prior tasks. I now direct attention to the problem of identifying an appropriate ranking function for selecting which experiences to preserve in long-term memory.

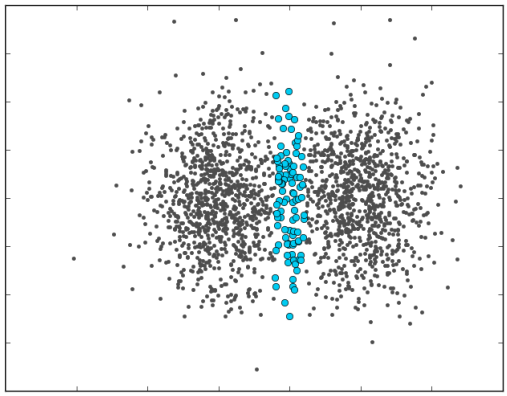
### 6.2.3 Selection Strategies

There are several reasonable solutions when faced with the design choice of which experiences to store in the episodic memory. Neuroscience literature suggests that replay occurs most often when events are infrequent, and the breadth of replayed experiences are not limited to the most recent events experienced nor the current task being undertaken (Gupta et al., 2010). Notably replayed events show preference for surprising (Cheng and Frank, 2008) and rewarding situations (Atherton et al., 2015). Alternatively, statistical strategies can be used. Experiences can be selected to match the distribution of all tasks (Bickel et al., 2009b), or, given that the future test distribution may differ from the training distribution, experiences can be selected to optimize coverage of the state space (de Bruin et al., 2016). For the rest of this section, I examine selection strategies based on surprise, reward, distribution matching, and coverage, comparing them to the traditional FIFO memory buffer. Each strategy gives a different ranking function $\mathcal{K}(\cdot)$ to order experiences.

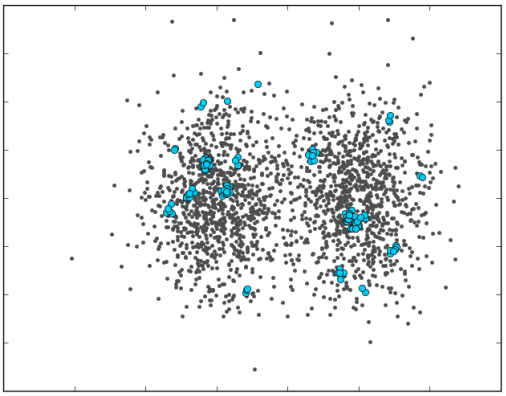
**Surprise** Unexpected events have been shown to be connected to replay in rodents (McNamara et al., 2014; Cheng and Frank, 2008) and *surprise* has a convenient counterpart in incremental reinforcement learning methods like Q-learning. The temporal difference (TD) error describes the prediction error made by the network and can be



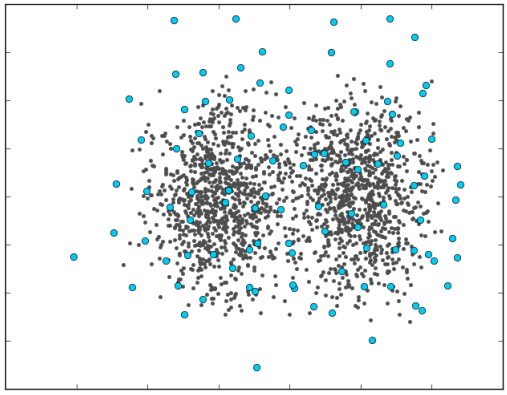
(a) Cartoon of experience space



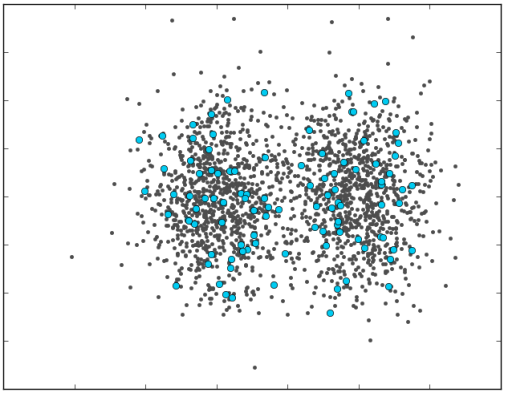
(b) Surprise



(c) Reward



(d) Coverage maximization



(e) Distribution matching

Figure 6.2: In the cartoon of experience space, green indicates positive value and red indicates negative value. The different diagrams illustrate how the different selection strategies would select points in this domain.

interpreted as the surprise of a particular transition:

$$\mathcal{K}(e_i) = |r_i + \gamma \max_{a'} Q(s'_i, a') - Q(s_i, a_i)| \ . \tag{6.4}$$

This idea is similar to the idea of uncertainty sampling in active learning (Settles, 2010), identifying unexpected inputs in history compression (Schmidhuber, 1993), and it has been effectively used as the foundation for a prioritized sampling strategy (Schaul et al., 2015b).

**Reward** Neuroscience research suggests replay occurs more often for rewarding events (Atherton et al., 2015; Ólafsdóttir et al., 2015). Prioritized sampling that favors rewarding transitions has also been shown to be helpful in RL (Jaderberg et al., 2016). This is implemented by using the absolute value of the future discounted return of a given experience

$$\mathcal{K}(e_i) = |R_i(e_i)| \ . \tag{6.5}$$

**Global Distribution Matching** It can be expected that the best test performance occurs if the training distribution matches the test distribution. This motivates a strategy that tries to capture the combined distribution of *all* tasks. To turn this into a ranking function, random selection is used to down sample the set of experiences to a smaller population that approximately matches the global distribution. In order to maintain a random sample over the global distribution even though our experiences arrive sequentially and the global distribution is not known in advance, a reservoir sampling strategy that assigns a random value to each experience is used. This random value is used as the key in a priority queue where the experiences with the

highest key values are preserved

$$\mathcal{K}(e_i) \sim \mathcal{N}(0,1) \ . \tag{6.6}$$

At time $j$ the probability of any experience being the max experience is $1/j$ regardless of when the sample was added, guaranteeing that at any given time the sampled experiences will approximately match the distribution of all samples seen so far

$$\forall e_i \ p(e_i \in B) = min\left(1, \frac{|B|}{j}\right) \ . \tag{6.7}$$

Here $|B|$ is the number of experiences stored in episodic memory $B$.

**Coverage Maximization** When limiting the number of preserved experiences, it might also be advantageous to maximize coverage of the state space. Research shows that limited FIFO buffers can converge to homogeneous transitions when the system starts consistently succeeding, and that this lack of error samples can cause divergent behavior (de Bruin et al., 2015). While it is often not possible to know the complete state space, de Bruin et al. (2016) shows that approximating a uniform distribution over the visited states can help learning when buffer size is limited.

In order to maximally cover the space, a uniform distribution is approximated by ranking experiences according to the number of neighbors in episodic memory within a fixed distance $\eta$ and replacing entries with the most neighbors. This approach is similar to the kernel method used by de Bruin et al. (2016) and makes the time complexity of adding entries $O(|E|)$ where $|E|$ is the number of stored experiences in episodic memory. This time complexity can be reduced to $O(log(|E|))$ by using a KD-tree (Bentley, 1975) and only comparing against the $k$ nearest neighbors or to

$O(k)$ by sampling the buffer. Formally, the set of neighbors $N_i$ is used to determine the ranking function

$$N_i = \{e_j \ s.t. \ dist(e_i - e_j) < \eta\} \tag{6.8}$$

$$\mathcal{K}(e_i) = -|N_i| \ . \tag{6.9}$$

## 6.3 Experiments

Selective experience replay is evaluated on the problem of autonomously handling unsigned intersections using the Sumo simulator (Krajzewicz et al., 2012) and confirm the generality of this results with a follow-up study in a grid world domain.

### 6.3.1 Simulation Environment: SUMO

Experiments were conducted in the open source Sumo traffic simulator (Krajzewicz et al., 2012) described in detail in Chapter 5.5. The traffic simulator is configured so that each lane has a 45 miles per hour (20 m/s) max speed. The car begins from a stopped position. Each time step is equal to 0.2 seconds. The max number of steps per trial is capped at 100 steps, which is equivalent to 20 seconds. The traffic density is set by the probability that a vehicle will be emitted randomly per second. A departure probability of 0.2 for *each lane* is used for all tasks.

Experiments were run using five different intersection tasks: *Right*, *Left*, *Left2*, *Forward* and a *Challenge*. Each of these tasks is depicted in Figure 5.4. The *Right* task

involves making a right turn, the *Forward* task involves crossing the intersection, the *Left* task involves making a left turn, the *Left2* task involves making a left turn across two lanes, and the *Challenge* task involves crossing a six lane intersection.

The order in which tasks are encountered does impact learning, and several groups have investigated the effects of ordering (Bengio et al., 2009; Ruvolo and Eaton, 2013a; Narvekar et al., 2016). For our experiments, a random task ordering that demonstrates forgetting is selected and held fixed for all experiments. All of our experiments use the dropout training method (Hinton et al., 2012) which was shown to help prevent forgetting (Goodfellow et al., 2013).

In order to evaluate the effectiveness of the proposed approach, I ran three sets of experiments. In the first experiment, the upper bound is evaluated using a network with an unlimited capacity FIFO buffer, and the lower bound on performance is evaluated used a limited memory FIFO buffer. A network was trained sequentially on five intersection tasks with a fixed order and $10,000$ training experiences on each task. The success rate is evaluated and plotted at periodic intervals during training. In the second experiment, the performance of selective experience replay with a limited-size experience replay buffer is investigated. A different network is trained for each of the four selection strategies and the performance is evaluated to check whether the network undergoes catastrophic forgetting during training. In the third experiment, the two most promising strategies are compared when the number of training samples are not evenly distributed across tasks. For each strategy, a network is trained on the *right* task for $2,000$ experiences followed by $25,000$ experiences on the *challenge* task. These tasks were selected as being the most different from each other based on previous experiments. To evaluate how well our results generalize, a second environment is also explored. The next section presents the second domain.

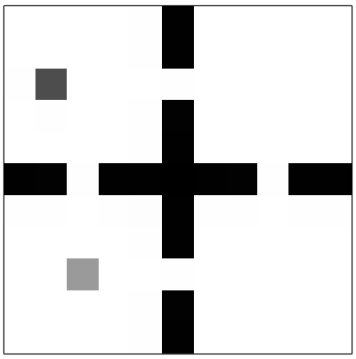


Figure 6.3: Grid world domain. Black indicates a wall, dark gray is the agent, and light gray is the goal.

### 6.3.2 Simulation Environment: Grid World

To confirm that our results generalize to other domains, I additionally conducted experiments on a grid world navigation task. In the grid world domain, the world is divided into four rooms. Each tasks corresponds to the room which contains the goal. The agent starts in the top left of a four room world sized $11 \times 11$. The agent's objective is to navigate to a goal in one of the other rooms. Figure 6.3 shows an example state in grid world. Each room corresponds to different task. The agent's action space consists of moving $\{up, down, left, right\}$. The agent is allowed up to 100 steps per trial, and the agent trains for $10,000$ iterations for each task for a total of $30,000$ iterations.

### 6.3.3 Experimental Setup

The state space of the DQN is represented as a $18 \times 26$ grid in global coordinates from the intersection domain ($11 \times 11$ for grid world). The DQN network uses a convolutional neural network with two convolution layers, and one fully connected

layer. The first convolutional layer has 32 $6 \times 6$ filters with stride two, the second convolution layer has 64 $3 \times 3$ filters with stride two. The fully connected layer has 100 nodes. All layers use leaky ReLU activation functions (Maas et al., 2013). The final linear output layer has outputs corresponding to the domain specific action space. The network is optimized using the RMSProp algorithm (Tieleman and Hinton, 2012).

Our experience replay buffers have an allotment of $1,000$ experiences. The FIFO only trials have a single replay buffer with $1,000$ experiences. Networks using a selective replay buffer store 900 experiences in the selective buffer and 100 in the smaller FIFO buffer. At each learning iteration, a total batch size of 60 experiences are sampled. This is split into 30 experiences from each buffer when using selective replay. This means mini-batches are an even mix of data from both buffers.

Since the experience replay buffer imposes off-policy learning, the return for each state-action pair in the trajectory can be calculated prior to adding each step into the replay buffer. This allows us to train on the n-step return (Peng and Williams, 1996).

Because surprise changes over the course of training, one can imagine implementing the surprise method to either retain the examples that were initially surprising, or to update samples with their new TD values as they are trained upon. In the later case, the TD errors would shrink as the network learns the hard examples and they would eventually be removed from the buffer. In practice this is what is observed: updating the TD errors eventually removes samples from older tasks until the buffer only has samples from the current task. This causes the network to forget earlier tasks, and in every domain tried, better performance was obtained by retaining the initial surprising examples. For this reason the results from the method that retains

the initially surprising examples are shown.

The number of neighbors in coverage maximization is dependent upon the samples currently in the buffer. As the buffer changes, the number of a neighbors a sample has may also change. To accommodate the change in buffer distribution, an experience's score is updated when it is sampled for training.

For calculating the distances between experiences in the coverage maximization approach, all components of the experience are concatenated (state, action, next state and reward) and the extended Jaccard metric is used for the intersection domain because the state representation is sparse (cosine similarity worked equally well). For the states in grid world, the L1 distance between the different agents and the different goals was used.

The epsilon governing random exploration was 0.05. For the reward $+1$ was used for successfully navigating the intersection, $-1$ for a collision, and $-0.01$ step cost. For grid world, the agent receives a reward of 10 for reaching the goal and a suffers a step cost of $-0.01$. Testing is run as separate procedure that does not have an impact on the replay buffer or the learning process of the network.

## 6.4 Results

### 6.4.1 Baselines

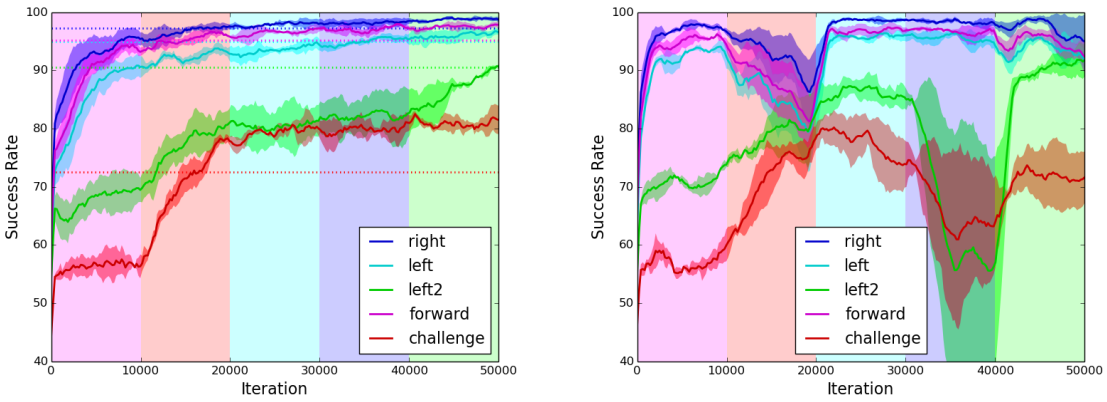
Figure 6.4 shows the performance of a network training with an unlimited capacity replay buffer in the intersection handling domain. There are five tasks in this example, horizontal dotted lines indicate the final performance of a single task network

trained for an equivalent number of training steps. The learning curves indicate the performance of a single lifelong learning network training on all tasks. Each color indicates the performance on a different task. The background color indicates the task the network is currently training on. It is observed that when the experience buffer has unlimited capacity, the network improves over the performance of the single task networks given an equivalent amount of training. Training on a new task receives the *jumpstart* (Taylor and Stone, 2009) benefit of prior knowledge and the network is resistant to forgetting. However, training with unlimited capacity is not practical as the number of tasks increases. This experiment demonstrates the upper bound performance our method hopes to match when the experience replay buffer capacity is limited.

When a FIFO replay buffer has limited capacity, previous tasks are forgotten (Figure 6.4). It is observed that training on the *forward* task initially boosts every tasks' performance. However, when the network begins training on the challenge task, performance drops on all single lane tasks (*right*, *left*, and *forward*). Similarly, training on the *right* task hurts the performance of other tasks. Recall from Chapter 5.6 that the more a network is refined on a single task, the more it hurts different tasks.

### 6.4.2 Selective Replay Results

The results of the networks using selective replay are shown in Figure 6.5. Our results show that both the surprise and reward strategies behaved poorly as experience selection strategies as they exhibit catastrophic forgetting. It is believed that surprise performed poorly as a selection method, because surprising experiences are exactly the samples the system is most uncertain about. It makes sense to focus on these



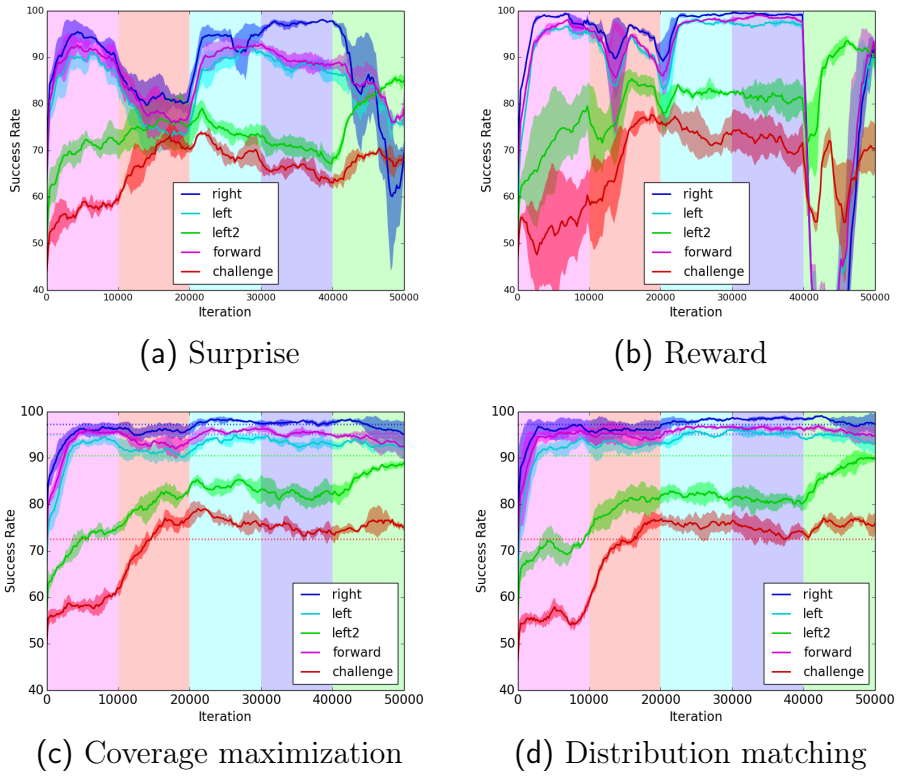

(a) Unlimited Capacity (b) Limited capacity (FIFO buffer)

Figure 6.4: (a) Lifelong learning of multiple tasks when the full history of experiences is available for each task. The dotted lines indicate the final performance of the single task network with an equivalent amount of training. (b) Training with a FIFO experience replay buffer on multiple tasks with limited capacity. The traditional method of storing memories for experience replay demonstrates catastrophic forgetting.

examples to improve learning, however when it comes to retaining knowledge, surprise represents the information the system has *not* learned. When the goal is to preserve memory, experiences where the network makes errors might be very poor experiences to store[2]. In the reward selection method, only rewarding experiences are stored, and training on only rewarding experiences ignores the credit assignment problem. Added noise was kept small, and only intended to break ties. Increasing the noise will select states around the rewarding experiences as well and will become a combination of the reward and distribution matching selection strategies. Increasing the amount of noise, the behavior converges to the performance of distribution matching.

Both the distribution matching and coverage maximization strategies prevented catastrophic forgetting, although they displayed somewhat different behaviors. By maximizing coverage, frequently visited regions in the state space can be covered with

[2]We also ran exploratory tests where we preserved experiences with the minimal TD error, but the results were not favorable




(a) Surprise (b) Reward

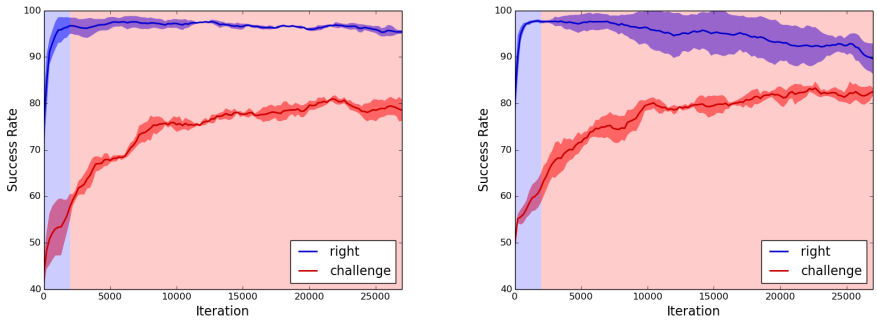
(c) Coverage maximization (d) Distribution matching

Figure 6.5: Selective experience replay results. While surprise and reward selection strategies exhibit catastrophic forgetting on previously learned tasks, coverage maximization and distribution matching strategies do not.

only a few samples, so maximizing coverage will be under-represented compared to distribution matching in places where the system spends most of its time. Similarly, strategies that seek to maximize coverage will favor outliers and rare events. This can be helpful if these error cases are catastrophic failure states that would be otherwise missed (Lipton et al., 2016). On physical systems, behavior that exhibits a large amount of exploration can quickly wear and damage components, so preserving these experiences for replay can also increase the safety of learning (de Bruin et al., 2015). In our domain, distribution matching exhibited slightly more stable behavior and had a slight advantage in performance. However, because the two approaches are so different, it is likely that the specifics of the domain could change which strategy

performs better.

### 6.4.3 Unbalanced Training




(a) Coverage maximization (b) Distribution matching

Figure 6.6: Unbalanced training. The network is trained for 2000 iterations on *right* and then trained for an additional 25000 iterations on *challenge*. The background color indicates the training task. If performance on the task with less training time is important, selecting to preserve samples based on coverage can give better results.

While the distribution matching strategy slightly outperformed coverage maximization in the previous experiment, that setup assumes that each task receives equal training time. However, it is reasonable to hypothesize that this strategy will under perform if the assumption that the importance of the task is proportional to the time spent training on it is broken.

In this experiment, both strategies are evaluated when the training time of the tasks are unbalanced. For each selection strategy, a single network is trained like previous experiments, but with only two sequential tasks in order to test the hypothesis. Both networks were trained on *right* task first, then on *challenge* task for 12.5 times longer.

The results of the unbalanced training is shown in Figure 6.6. On the *challenge* task, both strategies performed about the same. On the *right* task, coverage maximization

retained better performance than distribution matching. The reason was that the coverage maximization strategy kept more experiences from the *right* task in episodic memory. It could be the case that certain experiences, while only encountered briefly, are valued more. In this case, it is expected that the strategy that covers the state space more will perform better. It was observed that the distribution matching strategy had higher variance on the *right* task and which could be because of the random nature of the selection process.

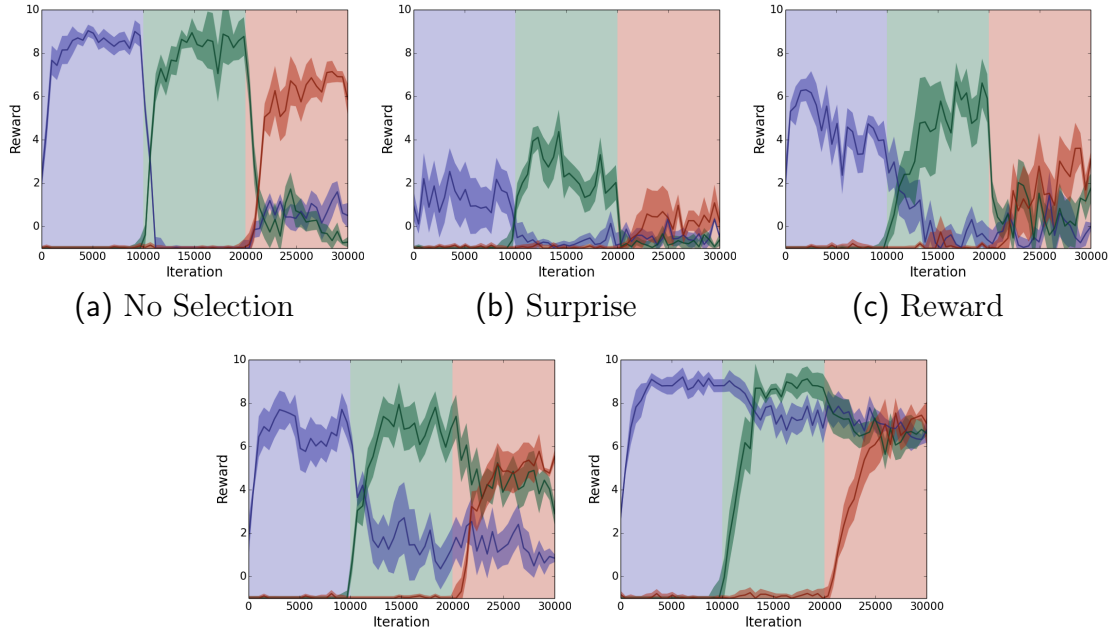
### 6.4.4 Grid World

Grid World shows much more dramatic catastrophic forgetting, where the agent almost immediately forgets performance of previous tasks without selection. Results are shown in Figure 6.7. Since the final task passes through the two earlier rooms some recovery was observed, but this is probably due to the configuration of the rooms. Surprise and Reward again both perform poorly. Coverage maximization preserves some performance on earlier tasks, but distribution matching both reaches higher performance on current tasks and preserves more performance on earlier tasks.

Additional studies were conducted to probe various aspects of the selection strategies. These evaluations confirm that our selection strategies worked as planned.

### 6.4.5 Evaluating Strategies

To confirm that Coverage Maximization does indeed give us good coverage, the state space of our intersection domain was projected down to two dimensions using the non-




(a) No Selection (b) Surprise (c) Reward

(d) Coverage Max. (e) Distribution Matching

Figure 6.7: Selective experience replay results on grid world domain. Without selection, performance on previous tasks almost immediately drops to zero. While surprise and coverage maximization help somewhat, distribution matching is the best at preserving experience.

linear randomized projection t-SNE (Maaten and Hinton, 2008) as seen in Figure 6.8. This projection also visualizes how different tasks come from different distributions in the state space. Since the selection process is done in the high dimensional space, not the mapped space, it is not expected that the selected points will look perfectly uniform after projection. However, the projection does show that the algorithm does indeed cover the space.

To store surprising samples, a priority queue ordered by the absolute value of the TD error was used. While large errors are not necessarily distributed uniformly throughout the learning process, in practice, our experiments show large errors occur throughout training - see Figure 6.8.

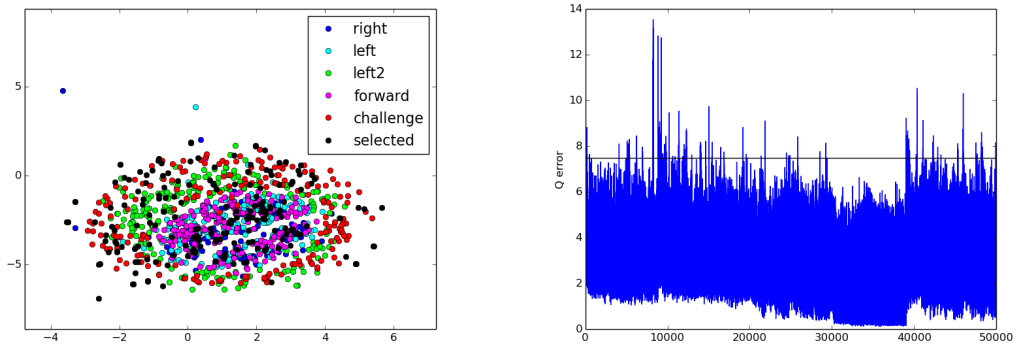



Figure 6.8: Additional Experiments]
(Left) Projection of state space into two dimensions using t-SNE (Maaten and Hinton, 2008). Black points indicate the points selected by the coverage maximization selection process. While it is not expected that the coverage is perfectly uniform after the projection into two dimensions, it is observed that the selected points cover the state space and include samples from every task. (Right) Magnitude of TD error over a typical run of five tasks. While the TD error is not consistent throughout learning, large TD errors are observed occurring throughout the learning process.

### 6.4.6 Distributions of Samples Stored in Buffer

Additional investigation showed how many samples from each task were in the buffer at the end of training under the different approaches. Results are shown in Table 6.1. Mistakes made by the network early in training were often smaller errors and had a longer time to get pushed out of the buffer. As a result, the first task had fewer samples. The harder tasks (Left2 and Challenge) also appear to correlate with larger errors. Reward has balanced examples from each task, since each task has a fixed reward, although since long trials that result in a collision will have more negative reward than short failures, it is observed that Left2 and Challenge have slightly more samples. Coverage Maximization has a large variance on the number of samples for Right, Left2, and Forward. This is because they occupy a similar region in space, see Figure 6.8. As a result, the buffer may have many samples of Right, and hold on

to fewer for Left and Forward. However, which task it held more samples for varied across trials, resulting in the large variance. Distribution Matching is balanced with more samples for Left2 and Challenge. This slight imbalance is due to the fact that the harder tasks tended to have longer trials, therefore producing more samples.

Table 6.1: Distribution of Stored Experiences

| Method | Right | Left | Left2 | Forward | Challenge |
|---|---|---|---|---|---|
| Surprise | $4 \pm 1$ | $70 \pm 20$ | $345 \pm 60$ | $97 \pm 20$ | $384 \pm 30$ |
| Reward | $117 \pm 5$ | $156 \pm 6$ | $210 \pm 16$ | $154 \pm 36$ | $265 \pm 10$ |
| Coverage | $221 \pm 110$ | $163 \pm 80$ | $109 \pm 107$ | $106 \pm 104$ | $300 \pm 22$ |
| Matching | $117 \pm 10$ | $156 \pm 16$ | $229 \pm 13$ | $150 \pm 11$ | $247 \pm 26$ |

### 6.4.7 Comparison with EWC

In order to compare selective experience replay with the state-of-the-art, two lifelong variants of MNIST (LeCun et al., 1998) were created. The first uses all ten classes of MNIST, and randomly permutes the pixels for each task. It is expected that there is no transfer benefit on this dataset, as the classes for each task are independent from the classes in the previous task. The second lifelong MNIST variant uses all ten classes but each task only includes data from two classes *i.e* task one includes '0's and '5's, task two includes '1's and '6's. Since there is a correlation between digits ('3's look like '8's) and the initialization will push any newly encountered '1' or '6' on the second task to be classified as either a '0' or '5', there is likely a greater effort to *forget* the previous learning that channeled all inputs into '0' or '5' outputs. The first variant is referred to as *permutations* and the second variant as *pairs*. A publicly available implementation of EWC (Seff, 2018) is used. A two layer fully connected neural network with 50 hidden nodes and ReLU activation functions is used as the architecture. The network is optimized using gradient descent. Each task is trained

for 1000 iterations with a batch size of 100 samples per iteration. The experiment was repeated six times with different random seeds to collect statistics.

On the permutation domain, selective experience replay and EWC have similar performance, see Figure 6.9 and Table 6.2. The shaded region shows the standard error. Subtle differences are observed in behavior. EWC preserves performance on the earlier tasks better, but learns and preserves performance on the later tasks slightly worse than selective experience replay. Unfortunately, combining the two methods does not appear to have a benefit over the individual approaches separately.

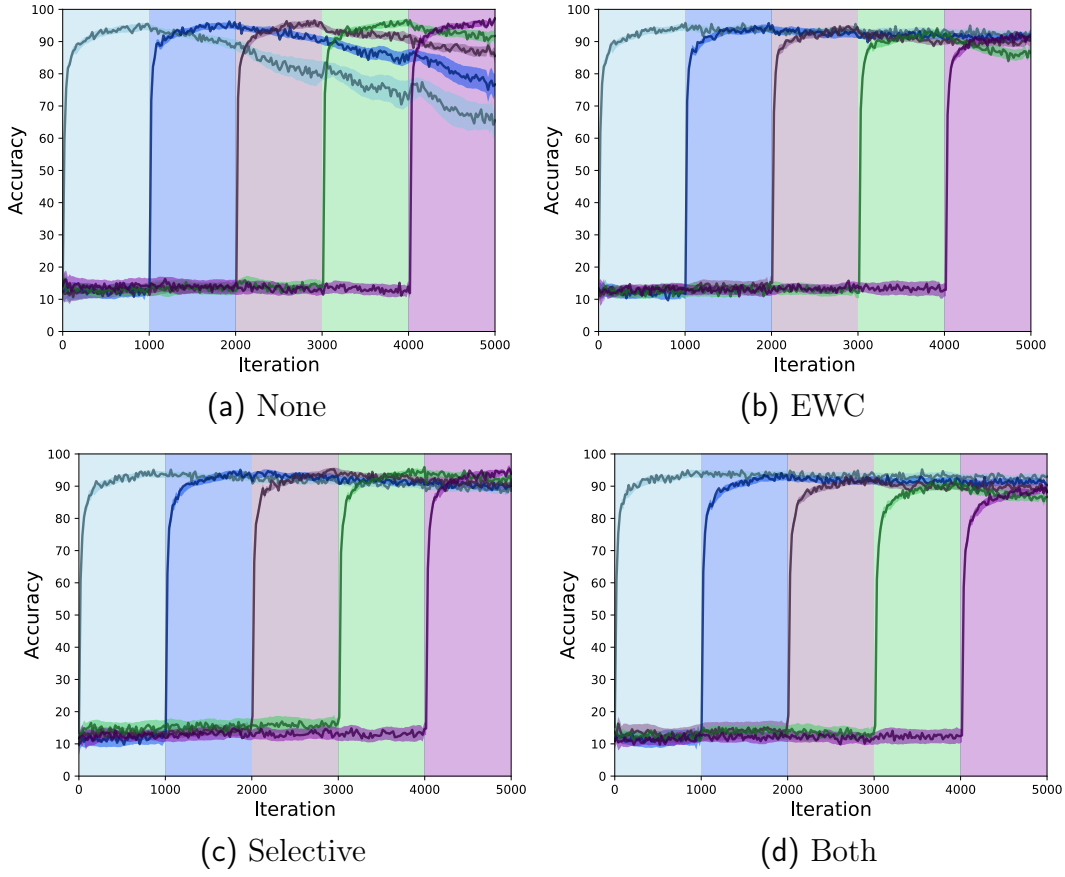



Figure 6.9: MNIST permutation domain. Distribution matching is used for the selective replay strategy. It is observed that both EWC and selective experience replay successfully prevent forgetting.

On the pairs domain, EWC under performs compared to selective experience replay,

Table 6.2: Lifelong MNIST permutations

| Method | Task 1 | Task 2 | Task 3 | Task 4 | Task 5 |
|---|---|---|---|---|---|
| None | $0.65 \pm 0.04$ | $0.76 \pm 0.03$ | $0.85 \pm 0.02$ | $0.91 \pm 0.02$ | $0.971 \pm 0.006$ |
| EWC | $0.91 \pm 0.01$ | $0.92 \pm 0.01$ | $0.90 \pm 0.01$ | $0.86 \pm 0.01$ | $0.91 \pm 0.01$ |
| Selective | $0.88 \pm 0.01$ | $0.900 \pm 0.007$ | $0.91 \pm 0.01$ | $0.91 \pm 0.01$ | $0.937 \pm 0.009$ |
| Both | $0.92 \pm 0.01$ | $0.903 \pm 0.007$ | $0.881 \pm 0.007$ | $0.86 \pm 0.01$ | $0.88 \pm 0.01$ |

see Figure 6.10 and Table 6.3. EWC does preserve performance longer than a traditional network, but by the end of training on all tasks, performance on earlier tasks has entirely eroded. Note that performance on early tasks does spike up during the early stages of training a new task. For example, consider the performance of task one at the beginning of training task three. We believe this is because the weights for the new task need to grow large to suppress the classification of earlier tasks. When those large weights are themselves reduced, a brief return of performance on the earlier tasks is observed. It is often the case that new data will be correlated with old data, and coarse decision boundaries that work on a small amount of data will need to be refined as new data is accumulated. We hypothesize that experience replay is essential for these refinements, and the results confirm that there is a substantial benefit to storing and replaying earlier experiences.

Table 6.3: Lifelong MNIST pairs

| Method | 0,5 | 1,6 | 2,7 | 3,8 | 4,9 |
|---|---|---|---|---|---|
| None | $0 \pm 0$ | $0 \pm 0$ | $0 \pm 0$ | $0 \pm 0$ | $0.946 \pm 0.006$ |
| EWC | $0 \pm 0$ | $0.005 \pm 0.005$ | $0.006 \pm 0.006$ | $0.12 \pm 0.02$ | $0.936 \pm 0.005$ |
| Selective | $0.82 \pm 0.01$ | $0.924 \pm 0.006$ | $0.69 \pm 0.04$ | $0.72 \pm 0.01$ | $0.914 \pm 0.003$ |
| Both | $0.73 \pm 0.02$ | $0.84 \pm 0.02$ | $0.47 \pm 0.02$ | $0.69 \pm 0.02$ | $0.876 \pm 0.004$ |

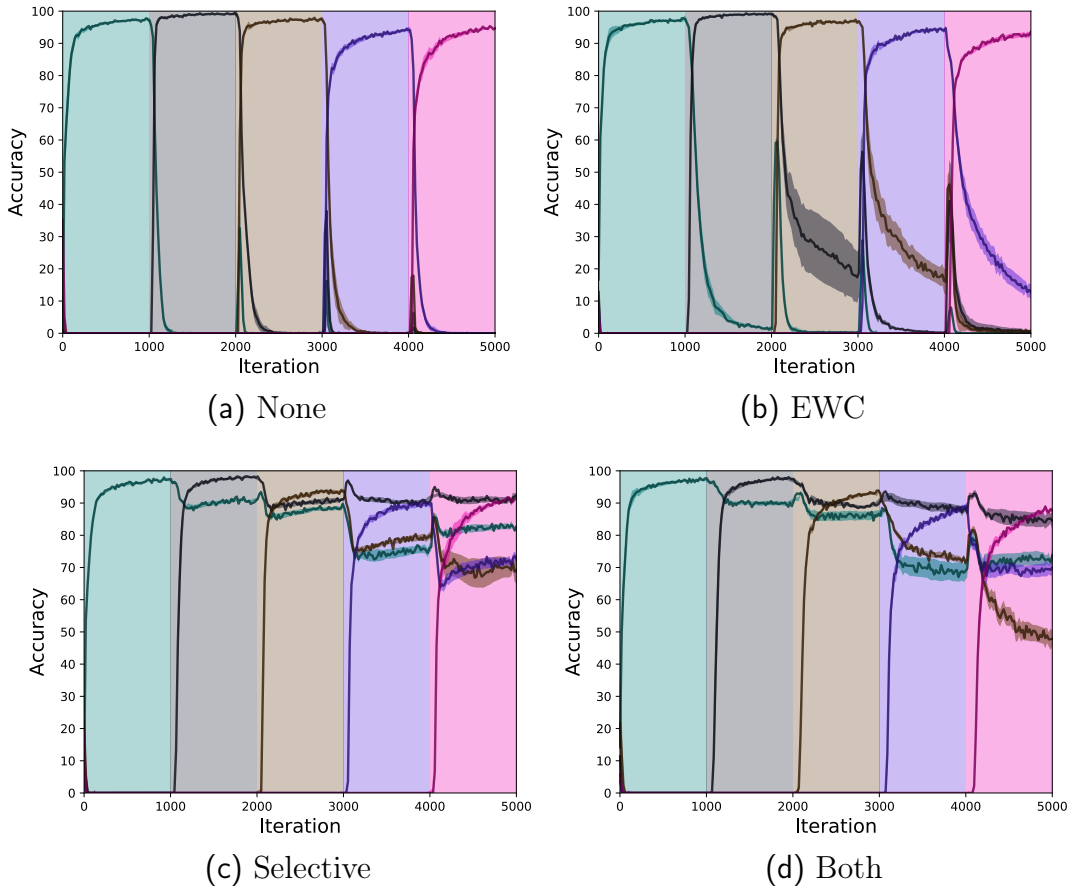



Figure 6.10: MNIST pairs domain. Distribution matching is used for the selective replay strategy. In this domain, selective experience replay is better at preventing forgetting.

## 6.5 Discussion

I proposed a method for selectively retaining experiences to preserve training on previous tasks in a lifelong learning setting. Forgetting previous task knowledge is a major hurdle when applying deep networks to lifelong learning scenarios. Unlimited storage of experiences is intractable for lifelong learning, and the use of FIFO experience buffers has been shown to lead to catastrophic forgetting. Our method involves keeping a long-term experience replay buffer in addition to the short-term FIFO buffer. Four strategies were compared for selecting which experiences will be

stored in the long-term buffer: a) Surprise, which favors experiences with large TD error, b) Reward, which favors experiences that lead to high absolute reward, c) Distribution Matching, which matches the global training state distribution, d) Coverage Maximization, which attempts to preserve a large coverage of the state space.

The four selection methods were evaluated in an autonomous driving domain where the tasks correspond to handling different types of intersections. In a setting where five intersection tasks were learned sequentially, Surprise and Reward selection methods still displayed catastrophic forgetting, whereas Distribution Matching and Coverage Maximization did not. The latter two strategies achieve nearly comparable performance to a network with unlimited experience replay capacity. While distribution matching has slightly better performance, one case was identified in which it is not an ideal strategy: the case where tasks which received limited training are more important. I additionally carried out experiments in a grid world domain, in which it was again seen that Distribution Matching performed best. Overall, our results show that selective experience replay, when suitable selection algorithms are employed, can prevent catastrophic forgetting in lifelong reinforcement learning.

Matching the distribution appears to be the best strategy for preserving experiences, it is likely surprise, reward, and coverage have their purposes in other apsects of the learning process. I have shown that coverage maximization can be preferable when training and test distributions differ. It has already been shown that surprise is helpful for learning. And I hypothesize that reward might have importance for partitioning a continuous timeline into a set of experiences. Note that this is built into our training process, each success or failure terminates an episode.

While the addition of episodic memory is intended to address forgetting in lifelong

learning, limited storage and the resulting divergent behavior are also concerns in single task networks and our contribution may find more general application outside of lifelong learning domains. While these experiments were carried out in an autonomous driving domain, sample complexity and safety concerns prevent us from directly applying standard Deep RL to real vehicles. The next chapter looks at how safety constraints can be incorporated into the learning environment.

# Chapter 7

# Safe Training

The previous chapter considered training a policy for an autonomous vehicle in simulation. While it is encouraging to see that policies can be learned that offer some benefits over rule-based approaches, there are several barriers preventing their application to real vehicles. The two primary difficulties are sample complexity and safety.

Even if transfer methods reduce the number of iterations, the unconstrained trial-and-error searches typical to reinforcement learning (RL) methods are not appropriate to physical systems which act in the real world, where failure cases result in real consequences. An autonomous vehicle cannot be allowed to crash in order to learn safe driving behavior. Recent work in safe RL uses idealized models to achieve their guarantees, but these models do not easily accommodate the stochasticity or high dimensionality of real-world systems. I investigate how prediction provides a general and intuitive framework to constrain exploration, and show how it can be used to safely learn intersection handling behaviors on an autonomous vehicle. The specific

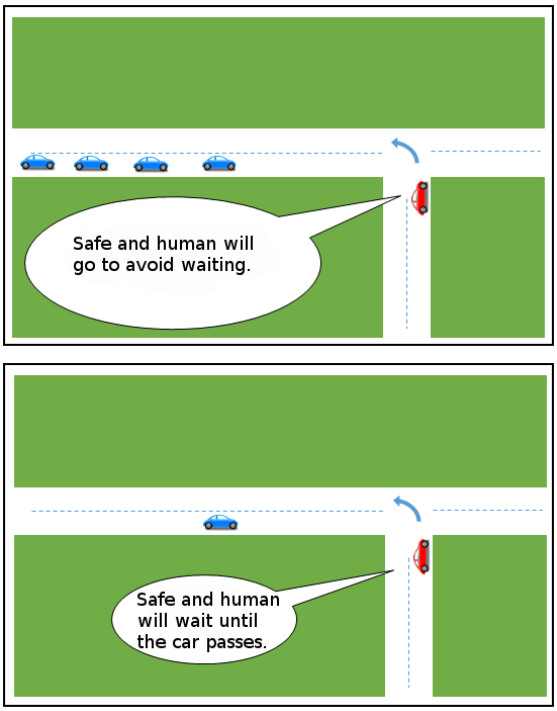



Figure 7.1: In dense traffic, a human driver (red vehicle) might take the opening to minimize the wait that would result from the approaching heavy traffic. However, in sparse traffic, it would be preferable to accept a small delay and let the car pass.

application I investigate is making a turn at an unsigned intersection. I explored this problem as a non-safety constrained RL domain (Isele et al., 2018b) as discussed in the previous chapter. In that chapter, I noted that the learned policy, which optimized efficiency, might be disruptive to traffic vehicles in practice. The primary concerns of these maneuvers are safety and efficiency, but balancing the two is a dynamic task. In dense traffic it may be desirable to seize an opportunity that leaves only several meters of a safety margin, as otherwise it would require waiting an unacceptable amount of time for the next opportunity. However in sparse traffic, adding an increased margin will only add a negligible delay and will likely be preferable to the passengers and other traffic vehicles. Figure 7.1 demonstrates the scenarios in dense and sparse traffic.

I demonstrate the use of prediction as a safety constraint by learning a policy that minimizes disruption to traffic (as measured by traffic braking) while avoiding collisions. Additionally, I learn a policy that maximizes distance to other vehicles, while still getting through the intersection in a fixed time window. It is shown that these two optimizations produce different behaviors and that both can be learned using RL with zero collisions.

## 7.1 Background

To mitigate the safety concerns associated with training an RL agent, there have been various efforts at designing learning processes with safe exploration. As noted by Garcia and Fernandez (2015), these approaches can be broadly classified into approaches that modify the objective function and approaches that constrain the search space.

Modifying the objective function mostly focuses on catastrophic rare events which do not necessarily have a large impact on the expected return over many trials. Proposed methods take into account the variance of return (Howard and Matheson, 1972), the worst outcome (Heger, 1994; Howard and Matheson, 1972; Lipton et al., 2016), and the probability of visiting error states (Geibel and Wysotzki, 2005). Modified objective functions may be useful on robotic systems where a small number of failures is acceptable. However on safety-critical systems, often a single failure is prohibited and learning must be confined to *always* satisfy the safety constraints.

Methods that constrain the search space are preferable when stricter safety requirements are necessary. Because these approaches can completely forbid undesirable states, they are usually accompanied by formal guarantees, however satisfying the necessary conditions on physical systems can be quite difficult in practice. For example, strategies have assumed a known safe policy which can take over and return to safe operating conditions(Hans et al., 2008), a learning model that is restricted to tabular RL methods (Wen et al., 2015; Wen and Topcu, 2016), and states that can be deterministically perceived and mapped to logical expressions (Alshiekh et al., 2018; Shalev-Shwartz et al., 2016). While there are some approaches that have been implemented on physical robots (such as the work of Gillula and Tomlin (2013) which uses reachability to enforce strict safety guarantees), these approaches tend to be computationally expensive, preventing their application to higher dimensional problems such as domains with multiple agents.

I investigate how prediction can be used to achieve a system that scales better to higher dimensions and is more suited to noisy measurements. Using prediction methods, it can be shown that it is possible to safely constrain learning to optimize intersection behaviors on an autonomous vehicle where it must consider the behaviors

Figure 7.2: An autonomous vehicle navigating an intersection. Prediction is used to shield the vehicle from making dangerous decisions, while allowing it to learn policies that are both efficient and not disruptive to other vehicles.

of multiple other agents. While I believe prediction is a very general framework that lends itself to implementations on a variety of stochastic physical systems, its safety constraints are weaker than other approaches in the literature: it is assumed that other agents (traffic vehicles) follow a distribution and are not adversarial.

## 7.2 Problem Statement

The problem of safely navigating an intersection is formulated as a stochastic game as described in Section 2.1.4. A safe set of policies $\mathbf{\Pi}^i$ is defined as the set of policies $\pi$ that generates a trajectory $\tau$ that with probability less than $\delta$ has agent $i$ entering a dangerous state at any step in its execution[1]. I use the subscript/superscript notation

[1]Interesting corner cases were proposed for many existing definitions of safety by Moldovan and Abbeel (2012). Their proposed definition of safety in terms of ergodicity does not easily extend to a multi-agent setting.

$variable_{time}^{agent,action}$ for consistency when possible.

Let $x_h^i$ be the local state of a single agent. The sequence of the local states, actions, and rewards for a single agent is referred to as a trajectory

$$\boldsymbol{\tau}^i = \{(x_1^i, a_1^i, r_1^i), \ldots, (x_H^i, a_H^i, r_H^i)\} \ ,$$

over a horizon $H$. The goal is to learn an optimal ego-agent policy $\pi^{*ego}$ where at every point in the learning process $\pi^{ego} \in \mathbf{\Pi}^{ego}$.

## 7.3 Proposed Pipeline

To achieve safe learning, I propose a pipeline that uses prediction of future states to mask unsafe actions. Figure 7.3 shows a visualization of the pipeline. In this pipeline, perception provides input to both an RL network and a prediction module. The prediction module masks undesired actions at each time step.

Given predictions, it is possible to mask actions that result in unsafe behaviors. Masking unsafe behaviors in RL has also very recently been proposed when states can be mapped to linear temporal logic (Alshiekh et al., 2018). This pipeline provides us with a mechanism to explore the safe subspace of an agent's possible behaviors during training and also guarantees that RL picks safe decisions during execution.

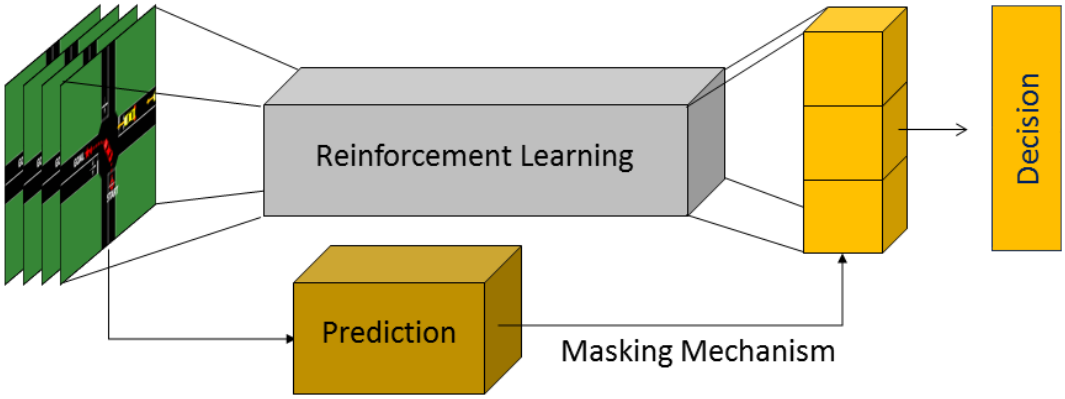



Figure 7.3: Proposed Pipeline.

## 7.4 Prediction

Probabilistic predictions serve as an approximation to the computationally expensive task of identifying safe trajectories. Using prediction models to mask unsafe actions from the agent, the agent is allowed to freely explore the remaining safe state space using traditional RL techniques.

RL algorithms have been proposed for stochastic games without any restrictions on safety (Littman, 1994; Hu and Wellman, 2003). However, in order to ensure that the agent never takes an unsafe action, it is necessary to check not only that a given action will not cause the agent to transition to an unsafe state in the next time step, but also that the action will not force the agent into an unsafe state at some point in the future. Note that this is closely related to the credit assignment problem, but the risk must be assigned prior to acting. One might imagine that ensuring the agent safely avoids all dangerous situations requires branching through all possible action combinations for a fixed time horizon $H$. Brute force implementations would result in an intractable runtime of $O\left(|\mathscr{A}|^H\right)$, where $|\mathscr{A}| = |\mathcal{A}^1| \times \ldots \times |\mathcal{A}^\kappa|$. Indeed it has been shown that even for the more restricted case of MDPs, identifying the set

of safe policies is NP-Hard (Moldovan and Abbeel, 2012). For this reason efficient approximations for restricting the exploration space are considered.

### 7.4.1 Simplifying Assumptions

To reduce the complexity it is assumed that the actions $a_h^i$ at each time step are components of a high-level action $æ^i$ (known as extended actions, skills, and options in the RL literature, and intentions in the autonomous driving literature). This has the effect of collapsing the branching factor of time associated with the exponential complexity. The cost of this approximation is that, for the fixed horizon, each agent is restricted in their ability to react and interact with other agents.

To accommodate the breadth of low-level action sequences that correspond to a single high-level action and also to allow for a bounded level of interaction, each high-level action is made a probability distribution over functions $f$. First the trajectory is described in terms of high-level actions $p(\boldsymbol{\tau}^i) \approx \prod_{j=1}^{|æ^i|} p(æ^i = j) p_{æ^{i,j}}(x_1^i, \ldots, x_H^i)$. Here $j$ indexes the high-level action æ, and the approximation takes into account the high-level action assumption. Then the functional local-state update is described as $x_{h+1} = f^{i,j}(x_h) + \epsilon_h$ where the noise is modeled as a Gaussian distribution $\epsilon_h = \mathcal{N}(0, \sigma_h)$. This means that the updated local state has a corresponding mean and standard deviation.

Within the fixed time horizon, each agent takes a single high-level action. The standard deviation is used to create a bound that encompasses the variety of low-level actions that produce similar high-level actions. These bounds also allow some slack for each agent to react to other agents while executing a high-level action without

violating our safety constraints. This can be thought of as selecting an open-loop high-level decision followed by subsequent bounded closed-loop low-level corrections. Note that by restricting an agent's ability to interact and limiting each agent to a restricted set of high-level actions, the existence of many pathological cases are ignored that may arise in an adversarial setting.

Given the assumption of high-level actions that follow a distribution, satisfying safety constraints can be computed in $O(|\mathring{\mathrm{A}}|H)$ where $|\text{Æ}^i|$ is the number of high-level actions available to agent $i$ and $|\mathring{\mathrm{A}}| = |\text{Æ}^1| \times \ldots \times |\text{Æ}^\kappa|$. This is still expensive for problems with a large number of actions or agents.

A further simplifying assumption arises when it is assumed that an agent's action space is unimodal. This is the case when it is assumed that the agent has a single action (e.g., a constant velocity assumption) or a hard prediction of the most probable action is made. This reduce the time complexity of a forward safety-checking prediction to a runtime of $O(\kappa H)$.

## 7.5 Safety Guarantees

One might suspect that these relaxations make it difficult to provide any safety guarantees. It does greatly limit the strength of the guarantees that can be made, however it is still possible to provide probabilistic guarantees on safety.

From Chebyshev's inequality, the likelihood of an agent $i$ taking action $j$ leaving its

safety margins $c\sigma^{i,j}$ is

$$p\left[|\boldsymbol{\tau}^{i,j} - \mathbb{E}(\boldsymbol{\tau}^{i,j})| \geq c\sigma^{i,j}\right] \leq \frac{1}{c^2} \ , \tag{7.1}$$

where $\sigma^{i,j} = \left[\sigma_1^{i,j}, \ldots, \sigma_H^{i,j}\right]$ and we define $|\boldsymbol{\tau}^{i,j} - \mathbb{E}(\boldsymbol{\tau}^{i,j})| \equiv \max_k |x_k^{i,j} - \mathbb{E}(x_k^{i,j})|$. Intuitively, this works to bound the largest variation. Note that we use the weaker Chebyshev inequality for our bounds since according to the Fisher−Tippett−Gnedenko theorem the max operation results in a distribution that is not Gaussian.

Since generally it is only important to consider one-sided error (e.g. if the traffic car is further away than predicted, there is no risk of a collision with the ego vehicle) it is possible to shrink the error by a factor of two.

$$p\left[\boldsymbol{\tau}^{i,j} - \mathbb{E}(\boldsymbol{\tau}^{i,j}) \geq c\sigma^{i,j}\right] \leq \frac{1}{2c^2} \ . \tag{7.2}$$

In our model, it is assumed that the safety margins create an envelope for an agent's expected trajectory. Collecting sufficient samples of independent trials and selecting appropriate safety margins, it can be assumed that the predicted trajectory roughly models the reachable space of the agent. Now, the independence of trials is probed. In expectation the agent follows the mean, but on each trial the deviations are likely not a purely random process, but are biased by a response to other agents.

In the autonomous driving literature, it is a common assumption that each agent behaves with self-preservation (Lefèvre et al., 2014). It is assumed that the measured distribution of the trajectory is the sum of two normally distributed random processes: the first associated with the agent's control and the second a random noise variable. The measured variance of a trajectory $\sigma_M^2$ is the sum of controlled $\sigma_c^2$ and noise $\sigma_n^2$

variances. This can be expressed relative to the measured standard deviation of the trajectory as $\varrho_c\sigma_M$ and $\varrho_n\sigma_M$ where $\varrho_c^2 + \varrho_n^2 = 1$. If it is assumed that an agent controls away from the mean by $\vartheta_c\varrho_c\sigma_M < c\sigma_M$, the probability that an agent leaves its safety margin can now be related to the variation being greater than both the measured bounds and the other agent's control

$$p[\boldsymbol{\tau}_j^i - \mathbb{E}(\boldsymbol{\tau}_j^i) \geq \vartheta_n\varrho_n\sigma_M] \leq \frac{1}{2\vartheta_n^2} \ , \tag{7.3}$$

where $\vartheta_n = \frac{c+\vartheta_c\varrho_c}{\varrho_n}$. To put this in concrete terms for an autonomous driving scenario, if it is assumed that there is a 5m measured standard deviation, 4m control standard deviation, 3m noise standard deviation, safety margin of $3\sigma_M$, and control action of $2\sigma_c$, the resulting safety margin is $7.6\sigma_n$. This analysis neglects any corrective controls of the ego agent. Applying the union bound and assuming a fixed $\vartheta_n$ for notational clarity, it is possible to achieve the desired confidence $\delta$ by satisfying

$$\frac{\kappa}{2\vartheta_n^2} < \delta \ . \tag{7.4}$$

## 7.6 Applications to Autonomous Driving

There are many works in the autonomous driving literature that look at risk-averse driving (Damerow and Eggert, 2015) and risk assessment (Lefèvre et al., 2015; Damerow and Eggert, 2015; Brechtel et al., 2011). Prediction is often used for safety in autonomous driving, and accurate prediction models are a current topic of research in the autonomous driving community (Lefèvre et al., 2014). Simpler models are built upon kinematic motion models (Schubert et al., 2008) with added uncertainty es-

timates to allow for errors in the measurements and assumptions (Carvalho et al., 2014). These methods are limited in that the Gaussian probability models and kinematic transitions assume cars roughly follow a known trajectory. More sophisticated models allow for multiple maneuvers (Aoude et al., 2011) which can be done by including road information (either heuristically or learned for particular intersections (Streubel and Hoffmann, 2014)) to allow for multiple possible maneuvers. More recent work in vehicle prediction considers the interactions between multiple vehicles (Agamennoni et al., 2012; Kuefler et al., 2017). However, these works are generally not concerned with learning behaviors. The research that does investigate learning policies (Song et al., 2016; Gindele et al., 2013; Isele and Cosgun, 2018; Bouton et al., 2017) are restricted to simulation and do not investigate the issue of preserving safety throughout the learning process under uncertainty.

Using prediction as a safety constraint does not necessarily require additional learning. Accurate prediction models can be sufficient to create behaviors that enforce safety constraints. A robust vehicle prediction module could itself be used to safely navigate intersections:

1. Predict the movement of the ego car entering the intersection in conjunction with forward predictions of all other vehicles.
2. If a collision is predicted, wait.
3. Otherwise, go.

However, this assumes a fixed behavior for the ego car — a single acceleration profile, a set time allowance to enter the intersection, and a set safety buffer to leave between cars.

Limiting the behavior of the autonomous vehicle to a one-size-fits-all motion is likely to lead to sub optimal behavior. Certain intersections may call for more aggressive accelerations to prevent excessive waiting, and the ability of other agents to interpret the system's actions can be just as important to safety as leaving sizable margins to allow for uncertainty. This work sets up a methodology by which it is possible to explore the more nuanced aspects of decision making. As an example, learning a model that minimizes traffic disruptions is explored.

## 7.7 Experiments

To demonstrate how prediction can be used as a safety constraint, deep Q-learning networks (DQNs) are used to learn policies that optimize aspects of intersection handling on autonomous vehicles. Two objectives are considered. The first objective is to learn an adaptive stand off which seeks to increase the safety margin without compromising the ability to make the turn given a fixed time window. The second model looks at minimizing the disruption to other vehicles while navigating the intersection in the given time.

### 7.7.1 Prediction

Traffic vehicles are modeled using a constant velocity assumption based on our Kalman filter estimates of the detected vehicle. Each vehicle is modeled with a fixed 2m uncertainty in detection. An additional uncertainty per time step is accumulated forward in time following a quadratic curve which was fit to data collected from errors in the forward velocity assumption, targeting a margin of six standard deviations. This

allows the model to make allowances for some accelerations and braking of the traffic vehicles. The ego car has similar forward predictions of its behavior based on the target trajectory and three potential acceleration profiles. The prediction errors are smaller for the ego car, since the intentions are known in advance. At each time step, going forward in time until the ego car has completed the intersection maneuver, the predicted position of the ego car is compared against the predicted position of all traffic cars. If an overlap of the regions is detected, the action is marked as *unsafe*. Actions that are marked as *safe* are passed on to the network as permissible actions. If there are no permissible actions, or the network chooses to wait, the system waits at the intersection. Otherwise the vehicle moves forward with the selected acceleration until it reaches its target speed.

### 7.7.2 Simulation

Experiments were run using the Sumo simulator (Krajzewicz et al., 2012), which is an open source traffic simulation package. To simulate traffic in Sumo, users have control over the types of vehicles, road paths, vehicle density, and departure times. Traffic cars follow the Intelligent Driver Model (IDM) (Treiber et al., 2000) to control their motion. In Sumo, randomness is simulated through driver imperfection models (based on the Krauss stochastic driving model (Krauss, 1998)). The simulator runs based on a predefined time interval which controls the length of every step. For our experiments a 0.2 second time step is used.

Each lane has a 30 mile per hour (13.4 m/s) maximum speed. The car begins from a stopped position. The maximum number of steps per trial is capped at 100 steps, which is equivalent to 20 seconds, starting from the first time the prediction says a

safe action is possible. This guarantees that a safe action is always possible in the allotted time. The traffic density is set by the probability that a vehicle will be emitted randomly per second. An emission probability of 0.1 is used for all experiments.

The DQN architecture is similar to the architecture used in the last chapter (Isele et al., 2018b). The simulator is designed to see cars 100m in either direction. This was selected to correspond to 25mph intersections. If it is assumed that it takes roughly 5 seconds to enter an intersection from a stopped position (measured from human demonstrations), it would be desireable to detect traffic at a minimum of 55m from the intersection. This minimum increases to 75m when allowing for traffic vehicles that travel slightly above the speed limit. The remaining 25m provide an added buffer to allow for accurate detection and tracking. The IBEO sensors used on the real vehicle are specified at 200m max range. The representation bins the traffic car positions into 26 bins per lane. Each lane is depicted as a separate row. Each spatial pixel, if occupied, contains the normalized real valued heading angles, velocity, and binary indicator.

The network has four outputs corresponding to *wait* and *go* commands where *go* can select from three different accelerations (0.5, 1.0, and 1.5 $m/s^2$). The network is optimized using the RMSProp algorithm (Tieleman and Hinton, 2012).

Each network was trained on $20,000$ simulations. When learning a network that seeks to minimize braking, the per trial reward is $+1$ for successfully navigating the intersection with a $-0.1$ penalty applied for every time step a traffic vehicle was braking. When learning a behavior that seeks to maximize the safety margin, the per trial reward is

$$r = \begin{cases} 0.1(d-10), & \text{if success} \\ z, & \text{if timeout} \end{cases},$$

where $d$ is the minimum distance the ego car gets to a traffic vehicle during the trial. The minimum observed distance during training is 4m, and $d$ can be a maximum of 50m. Experiments are conducted with different $z$ values, $z = \{-1, -5, -10\}$, to study the effect on timeouts.

To evaluate the learned models, two metrics are used: the average number of time steps per trial a traffic vehicle is braking, and the minimum distance between the ego car and the closest traffic car per trial. Statistics are collected over 1,000 trials. Since both objectives could be improved by increasing the safety margin, experiments are conducted where the safety margin of a rule-based only agent is increased to ensure the learned policy gives an improvement. A rule-based method that chooses the first safe time will follow close behind cars as they pass, so for a more fair comparison, the rule-based approach selects a random departure time for the set of safe departure times.

### 7.7.3 Real Vehicle

Training is conducted in simulation and the learned policy is verified on an autonomous vehicle. Data was collected from an autonomous vehicle in Mountain View, California, at an unsignaled T-junction, where the objective is to make a left turn. A point cloud, obtained from six IBEO Lidar sensors, is first pre-processed to remove points that reside outside the road boundaries. A clustering of the Lidar points

with hand-tuned geometric thresholds is combined with the output from three Delphi radars to create the estimates for vehicle detection. Each vehicle is tracked by a separate particle filter. Figure 7.2 depicts an intersection where our algorithm was evaluated. We used the network trained to maximize the safety margin.

## 7.8 Results

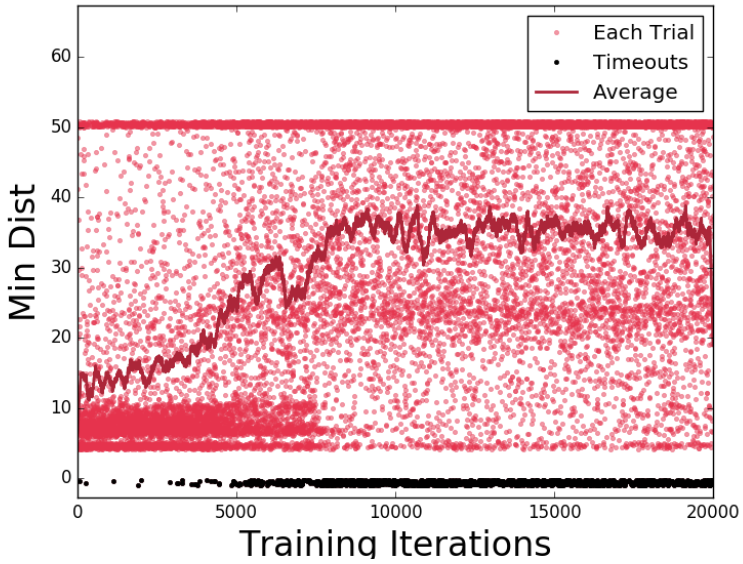



Figure 7.4: Minimum distance to traffic vehicles throughout the training process.

Using prediction as a constraint, a DQN is trained to minimize distance to other traffic vehicles. There were no recorded collisions during the entire training process. Figure 7.4 shows how the minimum distance to traffic vehicles changes throughout the learning process when $z = -1$. Timeouts are shown as having a distance of -1 and are colored black. The minimum distance of each individual trial is plotted as a point. The moving average using a sliding window of 200 trials is shown in dark red. It is observed that the concentration of points with a distance of less than 10m

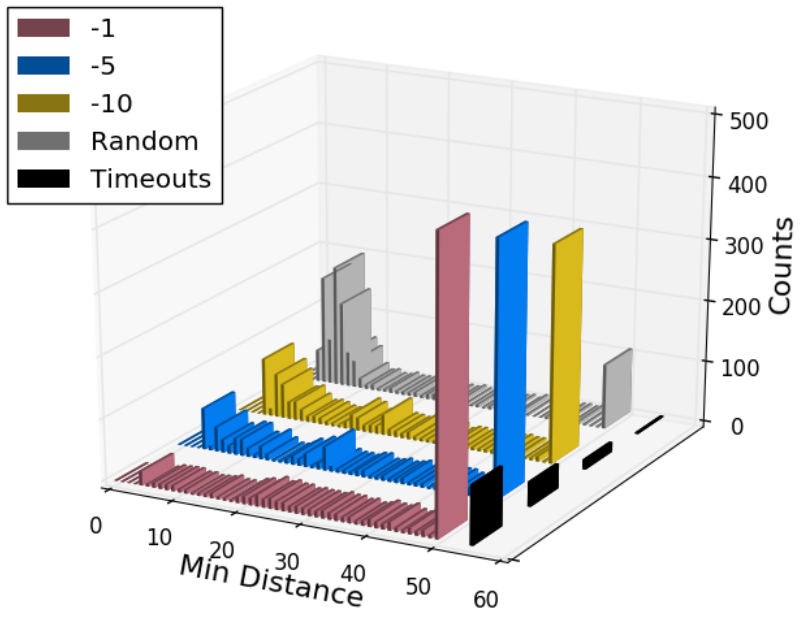



Figure 7.5: Comparison of the effect of the timeout penalty on the network trained to maximize the minimum distance. Timeouts are shown in black.

disappears at around 7000 training iterations, and there is an increased density in the region of 20m. Training does increase the number of timeouts. The extent of this can be changed by adjusting the penalty for timeouts, see Figure 7.5. Note that larger penalties produce larger gradients which can have an adverse affect on the learning process so there is a limit to how large the penalty can be set.

The number of trials that have a minimum distance of 50m or more also increases. This is more clear in Figure 7.5 where a histogram of the distances before and after training are shown. Figure 7.6 shows that naively increasing the safety margin with a rule-based strategy using prediction gives much worse performance compared to the learned networks.

It is expected that the network trained to minimize braking should leave a large margin when moving in front of a car, however it may come up very close behind a

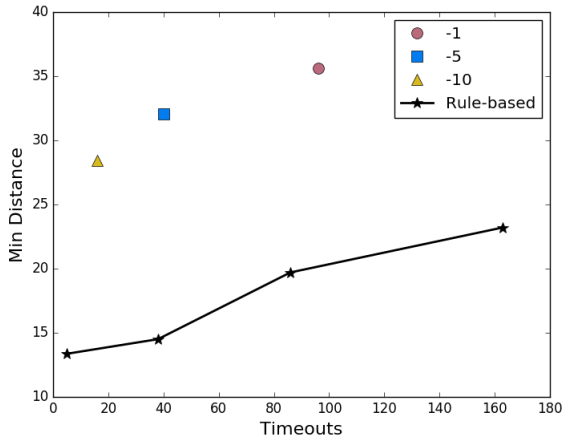



Figure 7.6: Comparison between the networks trained with different timeout penalties, and a rule-based method with a fixed safety margin. The safety margin for the rule-based method is varied across trials.

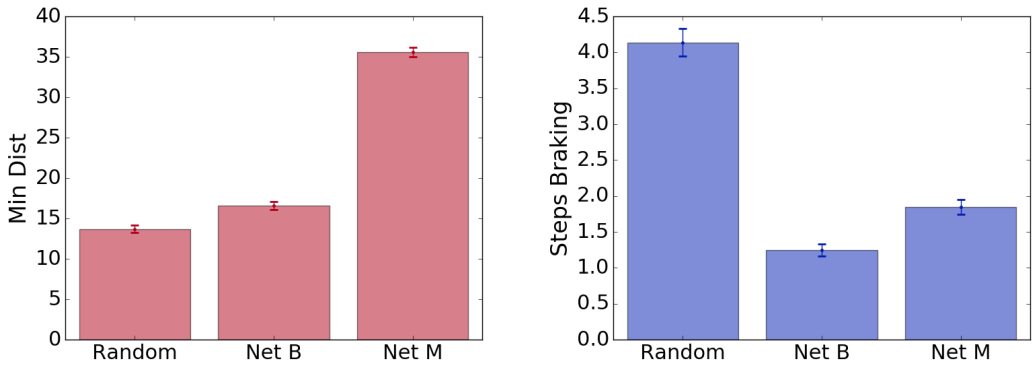



Figure 7.7: **Left:** Comparison of the minimum distance to traffic vehicles using different policies. Net B is trained to minimize braking, net M is trained to maximize the minimum distance. **Right:** The amount of time steps traffic cars spend braking. It is assumed that the ego vehicle is responsible for all traffic braking.

car. This is the case in the left plot of Figure 7.7. As expected, the network trained to maximize the minimum distance (Net M) greatly increases the average minimum distance. The network trained to minimize braking does achieve a larger average distance than a random policy acting in the safety constrained prediction framework, but the difference is much less pronounced. The network that was trained to maximize distance should also reduce braking. This result can be seen in right plot of Figure 7.7. Again it is observed the network specifically designed to minimize braking is better at satisfying the objective. The fact that these two objectives produce different behaviors makes sense. If viewed in terms of the gap between two cars, it would be optimal to be in the middle of the gap if maximizing distance and it would be optimal to be in the front of the gap if minimizing braking of other vehicles.

Qualitatively, it is observed that the network trained to maximize the safety margin will often add short delays after prediction determines the situation is safe if traffic is sparse. In cases where traffic is more dense, the network is more likely to move the moment an opportunity presents itself.

While the framework for safe RL is applicable to single task RL problems, it was motivated by the safety concerns of transferring a policy to new tasks in a lifelong learning setting. To bring the work full circle, and connect it back to the thesis topic, additional experiments are conducted. The different tasks consider changes in traffic density. A policy is trained in a low traffic density setting (0.1 depart probability per lane) and transferred to a high traffic density setting (0.15 depart probability per lane). A Safe RL policy is trained to increase the minimum distance between the ego car and other vehicles while still navigating through the intersection before timing out. The learning curves are shown in Figure 7.8. The network trained in the lower density traffic setting learns a policy that provides a substantial jump-start for

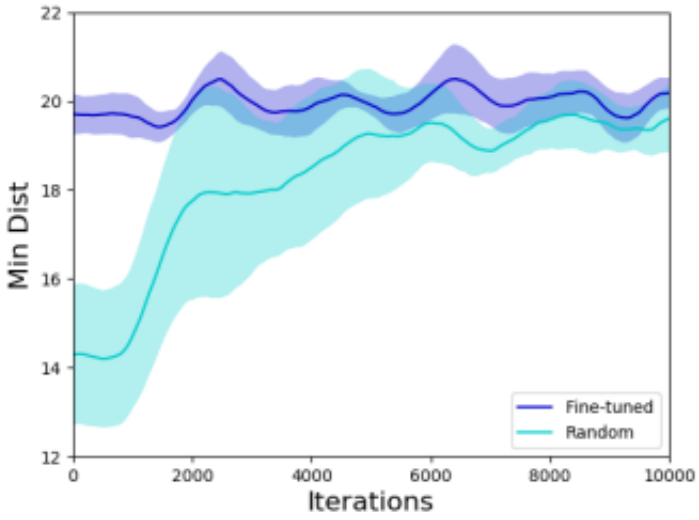



Figure 7.8: Learning curves for a network pre-trained on low density traffic and transferred to high-density traffic and a network trained directly on high-density traffic. Pre-training provides a substantial jump-start. Networks are trained using the safe RL framework, so even though the distribution changes during training, no collisions occur.

operating in a high-density traffic setting. Even though the distribution changes, the safe RL framework prevents collisions for the entire duration of training.

## 7.9 Discussion

In this work a framework is presented for safe RL using predictions to mask unsafe actions. This methodology is applied to an autonomous driving domain to learn policies that improve the performance of unsigned intersection handling. Specifically investigations look at 1) minimizing disruption to other vehicles and 2) maximizing safety margins while still navigating the intersection in a fixed time window.

While the safety guarantees made using prediction are not as strong as other ap-

proaches proposed in the literature, the framework is more general and likely more applicable to many real world applications. Since actions are masked, some of which are known to be safe, in order to provide safety margins when dealing with uncertainty, the final policies are possibly suboptimal. This suggests open problems both related to developing more sophisticated prediction modules and a more careful characterization of the regret associated with them.

# Chapter 8

# Conclusion and Contributions

My thesis has developed several strategies for running lifelong reinforcement learning on mobile robots. I identified disadvantages of existing state-of-the-art lifelong learning methods when used on robotic systems. Addressing these issues lead to the development of a method that uses task features to predict policies. By predicting policies, large amounts of information are able to be transfered to new tasks, greatly reducing the amount of training required.

Additionally, I presented methods for overcoming limitations when training multiple tasks with deep learning. These techniques, which selectively replay experiences, also appear to be connected to a system adapting when training on non-stationary tasks. Finally, I show how prediction can be incorporated into a reinforcement learning process. This enables the benefits of transfer while enforcing safety constraints throughout the entire learning process.

Now I return to the research questions presented in Chapter 1, answering the questions

with information gleaned from my experiences.

## 8.1 Research Questions

**How can a system continually accumulate prior knowledge and incorporate it into the learning process and behavior of future tasks?**

Knowledge can be continually accumulated through selection processes, compression schemes, and by incrementally learning models that represent the knowledge. Accumulated knowledge can then be used by biasing new data towards the accumulated knowledge or by revisiting and reinforcing previously learned concepts. Concerning the implementation details of accumulating and incorporating task knowledge: Chapter 3 and Chapter 4 dealt with accumulating and storing knowledge in a sparse dictionary and Chapter 6 investigated accumulating knowledge and storing information in a deep neural network.

- **Are some experiences more important than others, and if so, how can the important experiences be identified?** Chapter 6 looked at the importance of experiences. Our results showed that experiences should try to model the true distribution. This suggests that individual experiences are neither important nor unimportant, but the distribution of the population is what matters. As systems become increasing complex, it will be interesting to see if this result holds true, or if reasons emerge to place greater importance on select rare events.

- **Does the importance of an experience change over time, and if so,**

**how can a system change accordingly?** Chapter 6 considered a system with finite memory, where experiences approximate the true distribution. As the system moves on to new tasks, new experiences are added. So while earlier experiences are still important to perform earlier tasks, they take up less real estate in order to make room for new experiences.

- **What methods enable data to be accumulated in a fully online manner?** I have presented online algorithms for techniques that compress (Chapters 3 and 4) and select (Chapter 6) knowledge. The former was enabled by iterative updates to an approximation of the multi-task objective function using a Taylor series expansion. The latter was enabled by a priority queue with a reservoir sampling strategy. The core idea behind both is having a model that can be built incrementally.

**How effectively does the incorporation of knowledge from past reinforcement learning tasks reduce sample complexity of new tasks?**

We saw, in Chapter 3, an example that reduces training time by hundreds of trials. The method presented in Chapter 4 does even better - given enough previous experience, new tasks can be learned with little or no new training.

- **How many experiences are needed to learn a new task?** The good news, which we saw in Chapter 4, is that in some cases it is possible to learn a new task without any new experiences. However looking at the sample complexity in Figure 4.6(a) and Figure 4.6(b) and task complexity in Figure 4.9, more data is still better.

- **Is it possible to successfully perform a task without any training data,**

**and if so, how is that knowledge transferred?** Yes, it is possible! I show that knowledge can be transferred without any training data by finding an alternate method to model the task. One example of this uses task features as described in Chapter 4.

- **What changes if you transfer general vs. specific information?** We can separate general and specific knowledge in several ways. The first view we will consider is in the context of sparse coding, where the knowledge basis can be considered the general knowledge, and the sparse code is the task specific knowledge. The second view treats general knowledge as what related tasks can share about a new task, and specific knowledge as the refinements that get learned by additional fine tuning. In the first view, by predicting the sparse code, TaDeLL is transferring both specific and general knowledge to the new task. When PG-ELLA sparse codes the single task model, it is refining both the general and specific knowledge by relating it to prior tasks. However in the second view, where we consider specific knowledge that is acquired by fine-tuning and additional refinements, TaDeLL transfers general knowledge while PG-ELLA transfers the refinements or specific knowledge. In terms of getting a good model quickly, transferring general knowledge offers greater benefits. This is seen in the more dramatic improvements of TaDeLL compared to PG-ELLA. However the long tails characteristic of diminishing returns, suggest that techniques that can transfer refinements can offer huge time savings when mastery of skills is important.

**What novel situations is a robotic system able to handle as a result of its previously learned experience?**

This thesis investigated robot learning in the presence of unknown disturbances, systems without training data, systems with changing distributions, and learning in the presence of safety constraints. This is likely just a subset of the novel situations where transfer can be useful. I would expect almost all skills that can be learned could be benefited by transfer. But speaking to the point of which situations *require* transfer, I hypothesize that learning without training data and learning on systems with changing distributions both require transfer.

- **How can source robots with different dynamics be used to correct a disturbance on a target robot?** As shown in Chapter 3, modeling many systems with different dynamics allows the partially learned policy of a new system to be mapped to a common space of all known solutions.

- **What differences exist between transferring between tasks on a single robot platform and transferring knowledge between platforms?** Transferring between platforms and tasks can both be viewed as mapping problems where the goal is to learn a common space between learning problems. However, strategies might differ based on the number of platforms or tasks being considered. If learning for hundreds or thousands of platforms is desired, taking the time to train a dozen from scratch might be trivial. If there are a smaller number of systems (as in the case of transferring from simulation to a real robot), a strategy that looks at the efficiency of each individual system will be preferable. Learning many tasks on a single system also highlights concerns that might not be as immediate when looking at many systems. This includes issues like changing task distributions (which we looked at in Chapter 6) or progressions from simple tasks to more complicated tasks, which we did not consider here.

## 8.2 Future Work

Lifelong learning on mobile robots is still far from being a solved problem. While the transfer of prior knowledge and transfer from simulation to real systems can drastically reduce the number of samples required to train a robot, there is still a great interest in reducing the sample complexity further.

There are some promising strategies to achieve this — model-based approaches to RL (Deisenroth and Rasmussen, 2011) have very small sample complexity and the ideas have been extended to deep RL frameworks (Gal et al., 2016). However, computational costs can be prohibitive and there is much work to be done in understanding both the learned models and how they can be adapted and made robust for handling multiple tasks.

In order to scale the capabilities of robotics systems it will likely be necessary to not only learn tasks quickly, but also be able to compose those learned tasks to learn new and more complicated tasks. While many hierarchical strategies exist in RL (Sutton et al., 1999b; Bacon et al., 2016; Kulkarni et al., 2016; Tessler et al., 2017), there are far fewer examples that have been shown to have the robustness or data efficiency to be applied to robotic learning. Additionally, most of these approaches only currently allow for (or have been demonstrated on) a single level of abstraction.

As robots become increasingly capable of learning, it will become advantageous to have a system propose its own learning problems. This will allow a system to continue to learn without the need for intervention from an operator. I have recently published some initial work on this topic (Isele et al., 2018c), but numerous advances are likely needed before it can be applied to a robotic system.

This thesis focused on reinforcement learning, but there are many other learning paradigms. Robotic systems often have input from many sensors inputs and many tasks that use these sensor inputs in different ways. Some of these learning problems are better phrased as classification, regression, or clustering problems. However, it is likely that there is still information that can be shared between these tasks. There are currently very few examples of work looking at transferring across different domains (Bou Ammar et al., 2015). Once that technology is mature, there is still the open problem of understanding how to transfer across learning paradigms.

## 8.3 Conclusion

In my thesis, I look at how lifelong learning ideas can extend the abilities and improve the training of RL when it is applied to mobile robots. I have investigated transfer learning in the presence of disturbances, changing distributions, and limited data for the target task, and I have shown how learning can be done in a way that increases the safety of training on a physical system. I presented techniques for predicting policies for unseen tasks, I identified processes for selectively storing experiences and preserving knowledge throughout learning, and I have shown how safety constraints can be incorporated into the learning process. Collectively, the work of my thesis has shown how the controlled accumulation of knowledge can increase both the safety and efficiency of learning on a robotic system.

# Bibliography

Agamennoni, G., Nieto, J. I., and Nebot, E. M. (2012). Estimation of multivehicle dynamics by considering contextual information. *IEEE Transactions on Robotics*, 28(4):855–870.

Aicardi, M., Casalino, G., Balestrino, A., and Bicchi, A. (1994). Closed loop smooth steering of unicycle-like vehicles. In *Decision and Control, 1994., Proceedings of the 33rd IEEE Conference on*, volume 3, pages 2455–2458. IEEE.

Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., and Topcu, U. (2018). Safe reinforcement learning via shielding. *AAAI Conference on Artificial Intelligence (AAAI)*.

Ando, R. K. and Zhang, T. (2005). A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853.

Aoude, G., Joseph, J., Roy, N., and How, J. (2011). Mobile agent trajectory prediction using bayesian nonparametric reachability trees. *Proc. of AIAA Infotech@ Aerospace*, pages 1587–1593.

Argyriou, A., Evgeniou, T., and Pontil, M. (2008). Convex multi-task feature learning. *Machine Learning*, 73(3):243–272.

Atherton, L. A., Dupret, D., and Mellor, J. R. (2015). Memory trace replay: the shaping of memory consolidation by neuromodulation. *Trends in neurosciences*, 38(9):560–570.

Bacon, P.-L., Harb, J., and Precup, D. (2016). The option-critic architecture. *arXiv preprint arXiv:1609.05140*.

Bakker, B. and Heskes, T. (2003). Task Clustering and Gating for Bayesian Multitask Learning. *Journal of Machine Learning Research*, 4:83–99.

Barrett, S., Taylor, M. E., and Stone, P. (2010). Transfer learning for reinforcement learning on a physical robot. In *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*.

Baxter, J. (2000). A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198.

BBC (2011). Japan earthquake: Explosion at fukushima nuclear plant.

Bellman, R. (1961). Adaptive control processes. In *USA Today*. Princeton University Press.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. *International Conference on Machine Learning (ICML)*, pages 41–48.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517.

Bickel, S., Sawade, C., and Scheffer, T. (2009a). Transfer learning by distribution matching for targeted advertising. *Advances in Neural Information Processing Systems (NIPS)*, pages 145–152.

Bickel, S., Sawade, C., and Scheffer, T. (2009b). Transfer learning by distribution matching for targeted advertising. In *Advances in Neural Information Processing Systems (NIPS)*, pages 145–152.

Biswas, J. and Veloso, M. M. (2013). Localization and navigation of the cobots over long-term deployments. *The International Journal of Robotics Research*, 32(14):1679–1694.

Blanco, M., Atwood, J., Russell, S., Trimble, T., McClafferty, J., and Perez, M. (2016). Automated vehicle crash rate comparison using naturalistic data. Technical report, Virginia Tech Transportation Institute.

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

Bonilla, E. V., Agakov, F. V., and Williams, C. (2007). Kernel multi-task learning using task-specific features. *International Conference on Artificial Intelligence and Statistics*, pages 43–50.

Bou Ammar, H., Eaton, E., Luna, J. M., and Ruvolo, P. (2015). Autonomous cross-domain knowledge transfer in lifelong policy gradient reinforcement learning. *International Joint Conference on Artificial Intelligence (IJCAI)*.

Bou Ammar, H., Eaton, E., and Ruvolo, P. (2014). Online multi-task learning for policy gradient methods. *International Conference on Machine Learning (ICML)*.

Bouabdallah, S. and Siegwart, R. (2005). Backstepping and sliding-mode techniques applied to an indoor micro quadrotor. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2247–2252.

Bouton, M., Cosgun, A., and Kochenderfer, M. J. (2017). Belief state planning for navigating urban intersections. *IEEE Intelligent Vehicles Symposium (IV)*.

Brechtel, S., Gindele, T., and Dillmann, R. (2011). Probabilistic mdp-behavior planning for cars. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 1537–1542. IEEE.

Brechtel, S., Gindele, T., and Dillmann, R. (2014). Probabilistic decision-making under uncertainty for autonomous driving using continuous pomdps. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 392–399. IEEE.

Callan, J., Hoy, M., Yoo, C., and Zhao, L. (2009). Clueweb09 data set.

Carr, M. F., Jadhav, S. P., and Frank, L. M. (2011). Hippocampal replay in the awake state: a potential substrate for memory consolidation and retrieval. *Nature neuroscience*, 14(2):147–153.

Caruana, R. (1997). Multitask Learning. *Machine Learning*, 28:41–75.

Carvalho, A., Gao, Y., Lefevre, S., and Borrelli, F. (2014). Stochastic predictive control of autonomous vehicles in uncertain environments. In *12th International Symposium on Advanced Vehicle Control*.

Cheng, S. and Frank, L. M. (2008). New experiences enhance coordinated neural activity in the hippocampus. *Neuron*, 57(2):303–313.

Cosgun, A., Ma, L., Chiu, J., Huang, J., Demir, M., Anon, A. M., Lian, T., Tafish, H., and Al-Stouhi, S. (2017). Towards full automated drive in urban environments: A demonstration in gomentum station, california. *IEEE Intelligent Vehicles Symposium (IV)*.

Damerow, F. and Eggert, J. (2015). Risk-aversive behavior planning under multiple situations with uncertainty. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 656–663. IEEE.

de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. (2015). The importance of experience replay database composition in deep reinforcement learning. In *Deep Reinforcement Learning Workshop, Neural Information Processing Systems*.

de Bruin, T., Kober, J., Tuyls, K., and Babuška, R. (2016). Improved deep reinforcement learning for robotics through distribution-based experience retention. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3947–3952. IEEE.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, pages 465–472.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Donoho, D. L., Elad, M., and Temlyakov, V. N. (2006). Stable recovery of sparse overcomplete representations in the presence of noise. *IEEE Transactions on Information Theory*, 52(1):6–18.

Donoho, D. L. and Huo, X. (2001). Uncertainty principles and ideal atomic decomposition. *IEEE Transactions on Information Theory*, 47(7):2845–2862.

Dorato, P., Cerone, V., and Abdallah, C. (1994). *Linear-quadratic control: an introduction*. Simon & Schuster.

Ego-Stengel, V. and Wilson, M. A. (2010). Disruption of ripple-associated hippocampal activity during rest impairs spatial learning in the rat. *Hippocampus*, 20(1):1–10.

Ferster, C. B. and Skinner, B. F. (1957). Schedules of reinforcement.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*.

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

Fuerstenberg, K. C. (2005). A new european approach for intersection safety-the ec-project intersafe. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 432–436. IEEE.

Gal, Y., McAllister, R., and Rasmussen, C. E. (2016). Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*.

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

Gazebo (2016). http://gazebosim.org/.

Geibel, P. and Wysotzki, F. (2005). Risk-sensitive reinforcement learning applied to control under constraints. *Journal of Artificial Intelligence Research*, 24:81–108.

Gillula, J. H. and Tomlin, C. J. (2013). Reducing conservativeness in safety guarantees by learning disturbances online: iterated guaranteed safe online learning. *Robotics: Science and Systems VIII*, page 81.

Gindele, T., Brechtel, S., and Dillmann, R. (2013). Learning context sensitive behavior models from observations for predicting traffic situations. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 1764–1771. IEEE.

Girardeau, G., Benchenane, K., Wiener, S. I., Buzsáki, G., and Zugaro, M. B. (2009). Selective suppression of hippocampal ripples impairs spatial memory. *Nature neuroscience*, 12(10):1222–1223.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2672–2680.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. (2013). An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*.

Gupta, A. S., van der Meer, M. A., Touretzky, D. S., and Redish, A. D. (2010). Hippocampal replay is not a simple function of experience. *Neuron*, 65(5):695–705.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3(Mar):1157–1182.

Ham, J., Lee, D. D., and Saul, L. K. (2005). Semisupervised alignment of manifolds. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 120–127.

Hanna, J. P. and Stone, P. (2017). Grounded action transformation for robot learning in simulation. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Hans, A., Schneegaß, D., Schäfer, A. M., and Udluft, S. (2008). Safe exploration for reinforcement learning. In *European Symposium on Artificial Neural Networks (ESANN)*, pages 143–148.

Heger, M. (1994). Consideration of risk in reinforcement learning. In *Machine Learning*, pages 105–111. Elsevier.

Higgins, I., Pal, A., Rusu, A. A., Matthey, L., Burgess, C. P., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. (2017). Darla: Improving zero-shot transfer in reinforcement learning. *arXiv preprint arXiv:1707.08475*.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hoffman, J., Guadarrama, S., Tzeng, E., Donahue, J., Girshick, R., Darrell, T., and Saenko, K. (2014). LSDA: Large Scale Detection Through Adaptation. *Advances in Neural Information Processing Systems (NIPS)*, pages 3536–3544.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Howard, R. A. and Matheson, J. E. (1972). Risk-sensitive markov decision processes. *Management science*, 18(7):356–369.

Hu, J. and Wellman, M. P. (2003). Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4(Nov):1039–1069.

Huang, E. H., Socher, R., Manning, C. D., and Ng, A. (2012). Improving word representations via global context and multiple word prototypes. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882.

Huang, H. and Sturm, J. (2016). tum_simulator - ros wiki.

Isele, D. and Cosgun, A. (2017a). To go or not to go: A case for q-learning at unsignalized intersections. *International Conference on Machine Learning Workshop (ICML-WS)*.

Isele, D. and Cosgun, A. (2017b). Transferring autonomous driving knowledge on simulated and real intersections. *International Conference on Machine Learning Workshop (ICML-WS)*.

Isele, D. and Cosgun, A. (2018). Selective experience replay for lifelong learning. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Isele, D., Luna, J. M., Eaton, E., de la Cruz, G. V., Irwin, J., Kallaher, B., and Taylor, M. E. (2016a). Lifelong learning for disturbance rejection on mobile robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3993–3998. IEEE.

Isele, D., Nakhaei, A., and Fujimura, K. (2018a). Safe reinforcement learning on autonomous vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Isele, D., Rahimi, R., Cosgun, A., Subramanian, K., and Fujimura, K. (2018b). Navigating occluded intersections with autonomous vehicles using deep reinforcement learning. *IEEE International Conference on Robotics and Automation (ICRA)*.

Isele, D., Roberts, M., Eaton, E., and Aha, D. (2018c). Modeling consecutive task learning with task graph agendas. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Isele, D., Rostami, M., and Eaton, E. (2016b). Using task features for zero-shot knowledge transfer in lifelong learning. *International Joint Conference on Artificial Intelligence (IJCAI)*.

Isele, D., Rostami, M., and Eaton, E. (2017). Using task descriptions in lifelong machine learning for improved performance and zero-shot transfer. *arXiv preprint arXiv:1710.03850*.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

Jaggard, V. (2010). Mars rover to roam no more – it's official.

Khalil, H. K. (1996). *Noninear Systems*. Prentice-Hall, New Jersey.

Kifer, D., Ben-David, S., and Gehrke, J. (2004). Detecting change in data streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 180–191. VLDB Endowment.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2016). Overcoming catastrophic forgetting in neural networks. *arXiv preprint arXiv:1612.00796*.

Kleiner, A., Dietl, M., and Nebel, B. (2002). Towards a life-long learning soccer agent. In *Robot Soccer World Cup*, pages 126–134. Springer.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *Internatinal Journal of Robotics Research*, 32(11):1238–1274.

Kober, J. and Peters, J. R. (2009). Policy search for motor primitives in robotics. *Advances in Neural Information Processing Systems (NIPS)*, pages 849–856.

Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker, L. (2012). Recent development and applications of SUMO–simulation of urban mobility. *International Journal on Advances in Systems and Measurements (IARIA)*, 5(3–4).

Krauss, S. (1998). *Microscopic modeling of traffic flow: Investigation of collision free vehicle dynamics*. PhD thesis, Deutsches Zentrum fuer Luft-und Raumfahrt.

Kuefler, A., Morton, J., Wheeler, T., and Kochenderfer, M. (2017). Imitating driver behavior with generative adversarial networks. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 204–211. IEEE.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3675–3683.

Kumar, A. and Daumé, H. (2012). Learning task grouping and overlap in multi-task learning. *International Conference on Machine Learning (ICML)*, pages 1383–1390.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, H., Pham, P., Largman, Y., and Ng, A. Y. (2009). Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1096–1104.

Lefèvre, S., Vasquez, D., and Laugier, C. (2014). A survey on motion prediction and risk assessment for intelligent vehicles. *Robomech Journal*, 1(1):1.

Lefèvre, S., Vasquez, D., Laugier, C., and Ibañez-Guzmán, J. (2015). Intention-aware risk estimation: Field results. In *Advanced Robotics and its Social Impacts (ARSO), 2015 IEEE International Workshop on*, pages 1–8. IEEE.

Lewis, F. L. and Syrmos, V. L. (2012). *Optimal control*. John Wiley & Sons.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321.

Lipton, Z. C., Gao, J., Li, L., Chen, J., and Deng, L. (2016). Combating reinforcement learning's sisyphean curse with intrinsic fear. *arXiv preprint arXiv:1611.01211*.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning*, pages 157–163. Elsevier.

Lopez-Paz, D. et al. (2017). Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 6470–6479.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML)*, volume 30.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2009). Online dictionary learning for sparse coding. In *International Conference on Machine Learning (ICML)*, pages 689–696. ACM.

Maurer, A., Pontil, M., and Romera-Paredes, B. (2013). Sparse coding for multitask and transfer learning. *International Conference on Machine Learning (ICML)*, 28:343–351.

McCloskey, M. and Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 24:109–165.

McNamara, C. G., Tejero-Cantero, Á., Trouche, S., Campo-Urriza, N., and Dupret, D. (2014). Dopaminergic neurons promote hippocampal reactivation and spatial memory persistence. *Nature neuroscience*, 17(12):1658–1660.

Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Betteridge, J., Carlson, A., Mishra, B., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., Wang, R., Wijaya, D., Gupta, A., Chen, X., Saparov, A., Greaves, M., , and Welling, J. (2015). Never-ending learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, page 2302–2310.

Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., et al. (2018). Never-ending learning. *Communications of the ACM*, 61(5):103–115.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

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

Moldovan, T. M. and Abbeel, P. (2012). Safe exploration in markov decision processes. *arXiv preprint arXiv:1205.4810*.

Narvekar, S., Sinapov, J., Leonetti, M., and Stone, P. (2016). Source task creation for curriculum learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 566–574. International Foundation for Autonomous Agents and Multiagent Systems.

National Highway Traffic Safety Administration (2014). Traffic Safety Facts. Technical Report DOT HS 812 261.

Negahban, S., Yu, B., Wainwright, M., and Ravikumar, P. (2009). A unified framework for high-dimensional analysis of $m$-estimators with decomposable regularizers. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1348–1356.

Oh, J., Singh, S., Lee, H., and Kohli, P. (2017). Zero-shot task generalization with multi-task deep reinforcement learning. *International Conference on Machine Learning (ICML)*.

Ólafsdóttir, H. F., Barry, C., Saleem, A. B., Hassabis, D., and Spiers, H. J. (2015). Hippocampal place cells construct reward related sequences through unexplored space. *Elife*, 4:e06063.

Oyen, D. and Lane, T. (2012). Leveraging domain knowledge in multitask Bayesian network structure learning. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Palatucci, M., Hinton, G., Pomerleau, D., and Mitchell, T. M. (2009). Zero-shot learning with semantic output codes. *Advances in Neural Information Processing Systems (NIPS)*.

Pan, S. J. and Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10).

Parrot AR.Drone 2 (2016). ardrone2.parrot.com.

Peng, J. and Williams, R. J. (1996). Incremental multi-step q-learning. *Machine learning*, 22(1-3):283–290.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543.

Peters, J. and Schaal, S. (2008). Natural actor-critic. *Neurocomputing*, 71(7):1180–1190.

Pratt, G. and Manzo, J. (2013). The darpa robotics challenge [competitions]. *IEEE Robotics & Automation Magazine*, 20(2):10–12.

Pravin, V. (2018). Intelligent intersections. *https://youtu.be/6xmsjM-EOXQ?t=6414*.

Precup, D. (2000). Temporal abstraction in reinforcement learning.

Rafols, E. J. et al. (2005). Using predictive representations to improve generalization in reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 835–840. Citeseer.

Raina, R., Battle, A., Lee, H., Packer, B., and Ng, A. Y. (2007). Self-taught Learning : Transfer Learning from Unlabeled Data. *International Conference on Machine Learning (ICML)*, pages 759–766.

Ratcliff, R. (1990). Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285–308.

Razavian, A. S. et al. (2014). CNN Features off-the-shelf: an Astounding Baseline for Recognition. *Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 512–519.

Ring, M. B. (1998). Child: A first step towards continual learning. In *Learning to learn*, pages 261–292. Springer.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *arXiv preprint arXiv:1606.04671*.

Ruvolo, P. and Eaton, E. (2013a). Active task selection for lifelong machine learning. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Ruvolo, P. and Eaton, E. (2013b). ELLA: An efficient lifelong learning algorithm. *International Conference on Machine Learning (ICML)*, 28:507–515.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015a). Universal value function approximators. In *International Conference on Machine Learning (ICML)*, pages 1312–1320.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015b). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Schmidhuber, J. (1993). Learning unambiguous reduced sequence descriptions. *Advances in Neural Information Processing Systems (NIPS)*, pages 291–291.

Schmidhuber, J. (2015). On learning to think: Algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models. *arXiv preprint arXiv:1511.09249*.

Schubert, R., Richter, E., and Wanielik, G. (2008). Comparison and evaluation of advanced motion models for vehicle tracking. In *Information Fusion, 2008 11th International Conference on*, pages 1–6. IEEE.

Seff, A. (2018). Implementation of "overcoming catastrophic forgetting in neural networks" in tensorflow.

Settles, B. (2010). Active learning literature survey. *University of Wisconsin, Madison*, 52(55-66):11.

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

Sinapov, J., Narvekar, S., Leonetti, M., and Stone, P. (2015). Learning inter-task transferability in the absence of target task samples. *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Socher, R., Ganjoo, M., Manning, C. D., and Ng, A. Y. (2013). Zero-shot learning through cross-modal transfer. *Advances in Neural Information Processing Systems (NIPS)*, pages 935–943.

Song, W., Xiong, G., and Chen, H. (2016). Intention-aware autonomous driving decision-making in an uncontrolled intersection. *Mathematical Problems in Engineering*, 2016.

Srinivas, A., Sharma, S., and Ravindran, B. (2017). Dynamic action repetition for deep reinforcement learning. *AAAI Conference on Artificial Intelligence (AAAI)*.

Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F., and Schmidhuber, J. (2013). Compete to compute. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2310–2318.

Streubel, T. and Hoffmann, K. H. (2014). Prediction of driver intended path at intersections. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 134–139. IEEE.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999a). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems (NIPS)*, 99:1057–1063.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems.

Sutton, R. S., Precup, D., and Singh, S. (1999b). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211.

System, R. O. (2016). http://www.ros.org/.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. (2016).

A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. (2017). A deep hierarchical approach to lifelong learning in minecraft. In *AAAI Conference on Artificial Intelligence (AAAI)*, volume 3, page 6.

Thrun, S. (1996). Is learning the n-th thing any easier than learning the first? *Advances in Neural Information Processing Systems (NIPS)*, pages 640–646.

Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and autonomous systems*, 15(1-2):25–46.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop, coursera: Neural networks for machine learning. *University of Toronto, Tech. Rep*.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. *arXiv preprint arXiv:1703.06907*.

Treiber, M., Hennecke, A., and Helbing, D. (2000). Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62(2):1805.

Turtlebot 2 (2016). www.turtlebot.com.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 2094–2100.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Weiland, J. and Crow, A. (2018). How safe are autonomous cars? Technical report.

Wen, M., Ehlers, R., and Topcu, U. (2015). Correct-by-synthesis reinforcement learning with temporal logic constraints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4983–4990. IEEE.

Wen, M. and Topcu, U. (2016). Probably approximately correct learning in stochastic games with temporal logic specifications. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3630–3636.

White, A., Modayil, J., and Sutton, R. S. (2012). Scaling life-long off-policy learning. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, pages 1–6. IEEE.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Xu, X., Hospedales, T. M., and Gong, S. (2016). Multi-task zero-shot action recognition with prioritised data augmentation. In *European Conference on Computer Vision (ECCV)*, pages 343–359. Springer.

Yang, J., Wright, J., Huang, T. S., and Ma, Y. (2010). Image super-resolution via sparse representation. *IEEE Transactions on Image Processing*, 19(11):2861–2873.

Yin, H. and Pan, S. J. (2017). Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Yosinski, J. et al. (2014). How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NIPS)*, 27.

Yu, Z., Wu, F., Yang, Y., Tian, Q., Luo, J., and Zhuang, Y. (2014). Discriminative coupled dictionary hashing for fast cross-media retrieval. *Proceedings of the 37th*

*International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 395–404.

Zhuang, Y. T., Wang, Y. F., Wu, F., Zhang, Y., and Lu, W. M. (2013). Supervised coupled dictionary learning with group structures for multi-modal retrieval. *AAAI Conference on Artificial Intelligence (AAAI)*.