TASK GENERALIZED MDPS FOR MULTI-TASK REINFORCEMENT LEARNING

A Thesis
Presented to
The Academic Faculty

by

Himanshu Sahni

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Computer Science

School of Interactive Computing
Georgia Institute of Technology
May 2022
TASK GENERALIZED MDPS FOR MULTI-TASK REINFORCEMENT LEARNING

Approved by:

Professor Charles L. Isbell, Adviser
School of Interactive Computing
Georgia Institute of Technology

Professor Mark Reidl
School of Interactive Computing
Georgia Institute of Technology

Professor Judy Hoffman
School of Interactive Computing
Georgia Institute of Technology

Professor Dhruv Batra
School of Interactive Computing
Georgia Institute of Technology

Dr. Volodymyr Mnih
DeepMind Inc.

Date Approved: January 13, 2022
For my mother Manju Sahni and my father Ram Parshad Sahni.
Acknowledgements

My journey, in life and PhD, started with my mother Manju Sahni. She dreamt big for me and inspired me to do the same. It was her dream for me to become a scientist and I am proud to achieve it. My father, Ram Parshad Sahni, taught me, by his example, the value of hard work and the virtue of humility. Without the life they gave me and the sacrifices they made, I would not be here today.

Thank you to my brother, Vineet Sahni, my friend and companion from the day I was born. You taught me all the things about life that our parents couldn’t. I am happy we can share in life’s adventures and its burdens.

I would like to thank my advisor, Dr. Charles Isbell. Charles always made me feel like my ideas were better than I thought they were. He made me feel safe and allowed me to pick the research topics that interested me. As he says, he didn’t tell me what research to do, but instead taught me good taste. Thank you also for the countless discussions on pop culture, philosophy and life when we were supposed to be discussing research. I have learned about a lot more than just artificial intelligence from you.

I would also like to thank my thesis committee, Dr. Judy Hoffman, Dr. Mark Riedl, Dr. Dhruv Batra, and Dr. Volodymyr Mnih. Thank you for the great discussions and feedback that have made this thesis so much better.

Thank you Alicia Richhart for miraculously squeezing me into Charles’s calendar every time I needed a meeting. Thank you Dr. Thad Starner for getting me started in research and teaching me the basics of Machine Learning.

I want to thank my mentors during my various internships while in grad school. Dr. Katja Hofmann at Microsoft Research, Dr. David Nister at Tesla, Manohar Paluri at Facebook and Dr. Ilya Kuzovkin at OffWorld.

There are countless friends that have shared parts of this journey with me. Small or large, you all have had a role to play in my life and I thank you for befriending me.
Firstly, I want to thank my friend Ashley Edwards for sharing a large part of the PhD journey with me. Thank you for supporting me through the many lows, highs, times of stress, self doubt and uncertainty, and helping me see the best in myself and in life.

I want to thank Pushkar Kolhe and Shray Bansal for being the last ones stuck with me. Pushkar, thanks for breaking my wrist and putting me on a journey of self-discovery through adventure and outdoors. Thanks also for the countless pots of pav bhaji and chai with late night chats to warm my soul. Shray, thanks for always being there to patiently listen to all my thoughts and still be up for being my friend. We have been friends since the beginning of my PhD and you have seen me through pretty much everything.

Thank you Kalesha Bullard for helping me refine my research direction and not feel lost during job search. Your help came at just the right time and it means a lot to me. Thank you also for all the fun gossip and musings about life during our video chat sessions!

Thank you to my lifelong friends from IIT, Bismark Singh, Tarun Sengar, Ankur Taneja and Rehan Khan. Thank you Ashwath Kumar for being my eternal roommate, Yannick Schroecker for all the silent lunches as interns at Facebook (may there be more at DeepMind!), Steven Hickson for teaching me climbing and whiskey, Rahul Sawhney for leaving me his car, Jonathan Streater, Sasha Lambert, Amir Shaban, Safoora Yousefi, Sorrel Sanders, Qi Zhang, Daniel Kohlsdorf, Adam McLaughlin, Jon Scholz, Mr. Martin, Arya Irani, Kaushik Subramanian, Saurabh Kumar, Vitaly Marin, and many more that I am forgetting.

It has not been just a PhD, it has truly been a life that I have spent at Georgia Tech. Thank you to everyone for making it so sweet.
### TABLE OF CONTENTS

**DEDICATION** ................................................................. iii

**ACKNOWLEDGEMENTS** .................................................. iv

**LIST OF TABLES** ............................................................ ix

**LIST OF FIGURES** .......................................................... x

**SUMMARY** ........................................................................ xiii

**I  INTRODUCTION** ......................................................... 1

**II  BACKGROUND** ............................................................ 6

2.1 Reinforcement Learning and the Markov Decision Process ............... 6

2.1.1 Q learning .............................................................. 9

2.1.2 Actor Critic Methods ............................................... 10

2.1.3 Deep RL ............................................................... 11

2.1.4 Exploration in RL .................................................. 11

2.1.5 Online vs. Offline .................................................. 12

2.1.6 On-Policy vs. Off-Policy ........................................ 12

2.1.7 Model Based Learning ............................................ 13

2.2 Short Term Memory ...................................................... 14

2.3 Generative Adversarial Networks ....................................... 15

2.3.1 Conditional GANs and Auxiliary Conditional GANs ............... 15

2.3.2 Wasserstein GANs ................................................ 16

**III  HARD ATTENTION CONTROL BY MUTUAL INFORMATION MAXIMIZATION** ............................................. 18

3.1 Related works ............................................................. 20

3.2 Preliminaries .............................................................. 22

3.2.1 Internal representation ........................................... 22

3.2.2 The two agents .................................................... 23

3.3 Dynamic Memory Map .................................................. 23

3.3.1 Memory Modules ................................................... 24

3.3.2 Training loss ........................................................ 25
5.3 Curiosity Based Methods ..................................................... 67
5.4 DIAYN ................................................................. 69
5.5 Task Generalized Skills .................................................... 71
5.6 DIAYN as a Special Case of Task Generalized Skills ................. 72
5.7 Non-uniform Goals .......................................................... 73
5.8 Estimating $I(G; Z)$ from Samples ......................................... 74
5.9 Learning Task Generalized Skills ......................................... 75
5.10 Combining Diversity and Task Generalization ......................... 76
5.11 Boxworld ................................................................. 78
5.12 Results ................................................................. 78

VI CONCLUSIONS ............................................................... 87

REFERENCES ................................................................. 91
LIST OF TABLES

1  Per-pixel L2 loss between agent’s reconstruction $\tau$ and the ground truth full state $s$. All methods only see partial observations, but must attempt to reconstruct the full state from internal representation. In both environments, our method has the lowest reconstruction error. .................................................. 31

2  Goals sampled along the line $y = 0.8$ for each case shown in figure 23. .............. 82
## LIST OF FIGURES

1. Various successes of RL in recent years. ......................................................... 1
2. Agent-environment interaction. ................................................................. 6
3. Illustration of the PhysEnv domain (Du and Narasimhan, 2019). Here, the agent (dark blue) needs to navigate to the goal (red) while avoiding enemies (light blue). Typically the full environment states (top) are provided to the RL algorithm to learn. We consider the case where only observations from under a hard attention window (bottom) are available. The attention’s location (yellow square) at each time step is controllable by the agent. Using only partial observations, the agent must learn to represent its surroundings and complete its task. .......................................................... 18
4. The agent starts with a representation ($\mu_t$) from which it attempts a reconstruction ($\tau_t$) of the likely current full (unobserved) state $s_t$. Then, it picks an attention location ($l_t$) and receives an observation of the state from under the attention ($o_t$). We assume that the size of the attention window is known to the agent. The observation is written into $\mu_t$ to form $\mu_t^w$. The entire map is then stepped through an internal world model to update the dynamical objects, forming $\mu_{t+1}$, the representation for the next step. ................................................................. 22
5. Mean pixel prediction error for each model. Each model was trained on sequences of fixed length 25 but we predict states for a range of sequences from short (5) to long (100). Reconstruction error is measured over the full state at each time step and averaged over the sequence. ......................................................... 30
6. (left) Object-wise breakdown of reconstruction error in gridworld. (right) % of frames in which the object occurs, i.e. the frequency with which the attention is focused on a particular object. ........................................................................... 32
7. Average testing reward for goal seeking task in PhysEnv environment. Comparison between agent with full observations (upper limit) and partial observations trained with surprise (ours), environment rewards and simply following the agent’s location. 33
8. Visualizations of glimpse behavior in gridworld environment. Each sub-figure is the same testing episode at different stages of training, with the rows corresponding to the time within the episode. ........................................................................... 37
9. Visualizations of glimpse behavior in PhysEnv environment. ........................ 38
10. Glimpse behavior for fully trained models of our method and the baselines in gridworld. Each subfigure is a different baseline, with columns within the subfigure being the full unobserved state, heatmap of attention policy, and reconstruction of full state respectively. .............................. 40
11. Glimpse behavior for fully trained models of our method and the baselines in PhysEnv. 41
(left) PhysEnv environment. (right) Gridworld environment. Reconstruction error (L2) from the full state during training of the DMM + glimpse agent. All approaches have converged by 200k iters, with our approach having the lowest error, i.e. the most faithful reconstruction of the full unobserved state.

(left) PhysEnv environment. (right) Gridworld environment. Episodic reward for glimpse agent during training (higher is better).

State reconstruction and observation reconstruction errors in gridworld environment. Observation reconstruction error falls to almost zero for both methods while full state reconstruction remains high for variational approach.

HALGAN hallucinates the presence of goals in unsuccessful trajectories, ending in a perceived success. In this environment, the agent’s task is to search for a pebble randomly placed in its surroundings and collect it by approaching and centering it in its view. The top row shows a failed trajectory during exploration. The bottom row replays the same trajectory with a hallucination inserted by HALGAN at each step such that a pebble appears to be collected in the final state.

A schematic of the HALGAN training process. $c(s_t; g)$ informs the generator, $H$, on the desired location of the goal. The generated image is added to a random state from the replay, renormalized and passed to the discriminator. $D$ rates its authenticity and predicts goal location.

HALGAN output for different relative positions of the goal. The original failed state is on the top left. Demonstrates the fidelity to real states and ability to accurately control goal placement.

(a,b) Near goal states in experimental domains. (c,d) Episodic reward vs. environment steps for MiniWorld tasks averaged over 5 random seeds. (e) Results for Turtlebot task are averaged for 3 random seeds. (f) Minor variance in agent performance in Turtlebot task with decreasing size of HALGAN training dataset. 90% confidence intervals are shown for each plot.

(Left Column) Visualizations of skills learned by DIAYN on the boxworld over 3 random seeds. (Right Column) Visualization of skills learned by our method with 4 point goals on the straight line $y = 0.8$ on the same seeds.

Discriminator training loss for the first of the three random seeds visualized in 19. The negative of this is given as training reward to the agent.

The evolution of three quantities of interest over the training phase. On the top left is the entropy over the goals. High $H(G)$ means that all the goals are being attended to by some skills. On the top right is $H(G|Z)$, or entropy over the goals conditioned on a skill. It is colored according to the skill which it is measured for. Finally on the bottom is the total metric that we set out to maximize, $I(G; Z)$, the information between the skills and the goals.

On the left is the skill diversity metric as defined in DIAYN. This forms a part of our reward function and can be seen to be maximized during training. On the right is the overall episodic intrinsic reward attained by the agent during training which is a combination of the DIAYN diversity reward and our task generalization reward.
Skills learned for different number of sampled goals along the line $y = 0.8$. From left to right, the number of goals sampled are 2, 4, and 8, shown by the Xs on the figures. 83

Skills learned for different number of goals (4, 8 and 16) sampled from the Gaussian centered at (0.7, 0.7) with a standard deviation of 0.2. The darker circle shows one standard deviation and the light circle shows two standard deviations of the Gaussian. 83

Visualizations of learned skills for different Gaussians and mixtures as goal distributions. ................................. 84

Visualizations for different number of skills learned for the same goal distribution. 84

Average episodic reward achieved over testing runs on unseen goals for our method compared against DIAYN. The goals are drawn from a Gaussian task distribution. The results are averaged of 3 random seeds and the standard deviation across them is shown. This measures zero-shot generalization capability of both methods. . . . . 85

Average episodic reward achieved over testing runs on unseen goals for our method compared against DIAYN. The goals are drawn from a straight line task distribution. The results are averaged of 3 random seeds and the standard deviation across them is shown. This measures zero-shot generalization capability of both methods. . . . . 85

Average episodic reward achieved over testing runs on unseen goals for our method compared against DIAYN. The x-axis shows the number of goals sampled for training task-generalized skills. The goals are drawn from a Gaussian task distribution. . . . 86

Average episodic reward achieved over testing runs on unseen goals for our method compared against DIAYN. The x-axis shows the number of goals sampled for training task-generalized skills. The goals are drawn from a straight line task distribution. . 86
Reinforcement learning (RL) has seen widespread success in creating intelligent agents in several challenging domains. Yet, training RL agents remains prohibitively expensive in terms of the number of environment interactions required. One of the reasons for this inefficiency is that every new task is usually learned from scratch, instead of leveraging information from similar tasks.

The goal of this thesis is to build task-generalized Markov Decision Processes (MDPs), describing a distribution of tasks, defined as a collection of MDPs that are distinguished only by their reward functions. We focus on goal-oriented tasks, i.e. tasks that terminate with high reward in goal states. This thesis demonstrates that task-generalized MDPs can provide significant speedups for reinforcement learning in goal-oriented multi-task settings. Specifically, I claim that by first building a task-generalized MDP from a set of goal-oriented training tasks, one can achieve significant speedups on later tasks drawn from the set. Task-generalized MDPs can be optimized using standard reinforcement learning algorithms.

This thesis details work to support the thesis statement: By building a task-generalized MDP from a distribution of MDPs with different terminating goal states, one can enable or significantly speed up reinforcement learning.

The layout for this thesis is:

- Chapter 1 provides an introduction and motivation to the thesis. Here, I introduce task distributions, task-generalized MDPs and how they can be constructed using samples of goal states.

- Chapter 3 introduces the idea of combining attention, short term memory and unsupervised rewards to build a state representation in a limited field of view environment. By altering the underlying MDP’s state space, we can enable reinforcement learning of tasks within it. Attention and memory are trained adversarially to optimize the same objective derived from the mutual information between the state and attention location.
• Chapter 4 extends the idea of Hindsight Experience Replay to visual environments by inserting goal images into failed trajectories retroactively. This speeds up reinforcement learning by narrowing the exploration of the agent to near-goal states and quickly associating rewards with goal images. A key component is HALGAN, a generative model for inserting realistic goals into desired locations along the agent’s trajectory while respecting the environment dynamics.

• Chapter 5 describes a framework for task distribution biased unsupervised reinforcement learning. This framework allows for learning skills that are biased towards a task distribution and simultaneously distinct from one another. Skills learnt in this manner generalize better to downstream tasks compared against skill learning methods that do not incorporate this bias. Tasks distributions need not be known in closed form and samples of goal states can be used for learning. This framework demonstrates the idea of building task-generalized MDPs from distributions of tasks to speed up reinforcement learning.

In addition, Chapter 2 overviews some background knowledge relevant to the works in this thesis and Chapter 6 provides discussion.
I INTRODUCTION

Reinforcement learning has experienced tremendous success on the path to building generally intelligent agents (Mnih et al., 2015; Silver et al., 2017; Hessel et al., 2018; Espeholt et al., 2018; Haarnoja et al., 2018; Vinyals et al., 2019; Akkaya et al., 2019). Figure 1 shows a depiction of various challenging environments reinforcement learning has been successfully applied towards. Yet, training RL agents remains expensive in terms of number of environment interactions required to obtain good policies. Typically, an agent must experience a transition in the environment multiple times before it learns the optimal behavior. This may be unachievable in realistic environments where the state and action spaces of agents are usually large and can even be infinite in size. The agent may never experience the exact same state or action and hence has to learn to generalize from its experience.

Along with generalization, transfer poses a significant problem. Typically, RL benchmarks measure speed of learning from scratch without any prior knowledge on the tasks involved. From a real world learning perspective, learning each task from scratch is incredible inefficient as the experience of learning one task can potentially generalize to solving new tasks. For the purposes of this thesis, a set of tasks in an environment is a family of MDPs that differ only in their reward functions. Furthermore, we only consider tasks that terminate at a goal state. In general, a task defined to be successful if an agent achieves multiple goal states can be converted into one with a
The overall aim of this thesis is to significantly reduce sample complexity required for training reinforcement learning (RL) agents, making it easier to deploy them in the real world and quickly learn from experiences. Sample complexity in the context of RL usually refers to the number of agent-environment interactions but can also refer to the number of queries of a reward function. The strategy I propose to make progress towards reducing sample complexity focuses on using a distribution over tasks to construct a task-generalized MDP that is specialized towards solving tasks from that distribution. Given a new task, it is more sample efficient to learn using the task-generalized MDP than from scratch. In other words, the cost of building the task-generalized MDP is amortized over new tasks drawn from the distribution at test time.

I will demonstrate that by constructing a task-generalized MDP for a distribution of task MDPs, we can achieve significant speedups in learning new tasks drawn from the same distribution. Before proceeding further, I will describe what a distribution of tasks is and how we can use them to build task-generalized MDPs.

Reinforcement learning seeks to optimize agent behavior to maximize some notion of long term expected return (Sutton and Barto, 2018). Typically, return is defined so as to elicit a desired behavior within a domain. Examples of desired behavior could be breaking tiles in the Atari game of Breakout (Bellemare et al., 2013), manipulating a rubik’s cube to a specific position (Akkaya et al., 2019), rearranging objects in a 3D embodied environment (Szot et al., 2021), etc. Hence, reward is a proxy for desired behavior that completes a particular task. Typically, the behavior learned by reinforcement learning agents is brittle, i.e. it is only optimized for the task (or reward function) it is trained on. Agents deployed to the real world, though, will have to contend with a family of tasks, rather than a singular task with a specific reward function.

In fact, any task with the variability of a real world environment will be unique and novel to an artificial agent. Just as state and action spaces can be infinitely large, the task space of an agent can be infinite as well. Consider the task of pouring water into a glass. Depending on the type of glass, whether it is transparent or not, the amount of water to be poured, the time of day, whether it is being poured at a bar or a fancy restaurant, etc., i.e. depending on the context, the task can be slightly different. But in all cases, the tasks share a common structure. This suggests that tasks
are sampled from an underlying distribution that captures this structure but also the variability introduced by context. Artificial agents should not be solely concerned with solving individual tasks, but a family of tasks drawn from underlying distributions that define how likely each task is.

Further, there is a distribution that defines which tasks are meaningful in the context of the physical and social world we inhabit. Our earlier example of pouring water into a glass is considered a meaningful task, but flailing your arms around without effect is not considered meaningful in most contexts. As humans, we are able to intuit which tasks belong to this distribution of meaningful tasks and which do not. To a reinforcement learning agent that only knows to maximize rewards, this notion of meaningful vs. frivolous tasks is unknown. Hence, learning without imbibing a notion of what tasks are meaningful is bound to be inefficient in the real world.

A task-generalized MDP attempts to build a state, action, and/or reward space with access to this underlying task distribution in some form, rendering the solutions of future tasks simpler. Each of the components of the MDP can be biased by the task distribution to be more efficient at solving similar new tasks.

An important question is how can knowledge of this underlying task distribution be provided to the agent? A closed form description of task distributions in the real world is unlikely to be found. In this work, I will show how samples of goals from the underlying task distribution can be used to build task-generalized MDPs. Each sample of a goal can be considered an individual MDP, sharing the state and action spaces, differing only in reward. Through the works described in chapters 4 and 5, I will show examples of how goals sampled as states can be used to construct reward functions for the task-generalized MDP. The task distribution is reflected in the frequency of the sampled goals. In expectation, the task-generalized MDP reflects the true distribution of tasks.

Another perspective on task-generalized MDPs is through the lens of exploration. During online reinforcement learning (see Chapter 2), exploration plays a major role in how well the agent can learn in the environment. Wide exploration can present the agent with a variety of training data that it can use to learn the best action in many states to maximize its return. But exploring every state quickly becomes infeasible in environments with very large state spaces or in continuous state
domains. Further, simply exploring a large number of states may not guarantee a wide range of experience for the agent to learn from.

For example, consider a binary state space, \(\{0, 1\}^{100}\), where only the state that contains all zeros has a reward of 1 while every other state rewards 0. The agent can explore \(2^{100} - 1\) states without ever encountering a reward and thereby not learning very much. It is only when the agent randomly encounters the correct goal state that learning begins (see Chapter 4 for more details). We would like to guide or alter experience collection towards meaningful tasks such that the training signal for the agent is maximized.

While the field of optimizing exploration is widely studied through the years (Thrun, 1992; Strehl and Littman, 2008; Bellemare et al., 2016; Pathak et al., 2017; Fortunato et al., 2017; Burda et al., 2018b; Ciosek et al., 2019), the proposed work focuses on learning how to alter experience the agent collects through task distributions in order to benefit learning.

**Contributions**

The thesis statement is: **By building a task-generalized MDP from a distribution of MDPs with different terminating goal states, one can enable or significantly speed up reinforcement learning.**

I will begin by describing how a state representation can be learned by a hard attention trained through unsupervised rewards for a single task. Then I will show how a task-generalized MDP can be constructed in the case of visual goals with sparse reward. Finally, I will demonstrate a task-generalized MDP over a distribution of tasks in the case of rewardless learning of skills. The thesis statement will be supported with results in simulation from each of the three methods.

The first contribution (Chapter 3) is a method to learn a representation of the agent’s state in limited field of view environments. The state space of the task MDP is comprised of partially observed *glimpses* of the full state through a hard attention window. Attention control is adversarially trained against the agent’s model of the transition function using intrinsic rewards. The learned representation can be used to enable reinforcement learning in this partially observable environment. This work demonstrates that by modifying the state space, and hence the underlying MDP, on a single task using unsupervised rewards, one can enable reinforcement learning where otherwise no learning occurs.
Part of this first contribution is a short term memory architecture for limited field of view environments, the Dynamic Memory Map (DMM). The DMM uses write constraints and bundles (described in detail in Section 3.3) to efficiently backpropagate gradients over long sequences of observations. It also learns internal dynamics models for objects in the environment and the agent. The DMM combined with adversarial attention keeps the representation of the environment current even with the constraint of partial observation and allows the agent to learn to complete tasks.

The second contribution (Chapter 4) is a method to alter visual trajectories in hindsight using learned hallucinations of goal images. Combined with an existing technique known as Hindsight Experience Replay (HER) (Andrychowicz et al., 2017a), this significantly speeds up reinforcement learning as shown in two navigation based domains. Here, the state space of the agent is being modified by a learned generative model of what a goal state looks like. The reward function is also modified in hindsight by delivering rewards when hallucinated goal states are achieved. This contribution introduces a key component of task-generalized MDPs which is a generative model over goal states. It also demonstrates the modification of agent trajectories by altering a series of states and their associated rewards. This method applies to more realistic domains where non-trivial alteration of past experiences is required in order to take advantage of the hindsight replay principle.

The third contribution (Chapter 5) demonstrates fully the principle of learning task-generalized MDPs from a distribution of tasks. Sample tasks are used to bias skills that achieve tasks as well as are distinct from each other. These skills are learned without an extrinsic reward function. Instead, they maximize an intrinsic reward consisting of multiple terms of mutual information between state, goals, and skills. The skills are then used to measure zero-shot generalization to new tasks. In this work, it is the action space and reward function components of the MDP that are modified to enable faster learning on the task distribution.

Combined, these three contributions demonstrate that by modifying the original MDP to take into account task distributions, i.e. by constructing task-generalized MDPs, one can enable or significantly speed up reinforcement learning for new tasks.
II BACKGROUND

In this section, I will provide some background topics relevant to this proposal.

2.1 Reinforcement Learning and the Markov Decision Process

Reinforcement learning is a field of study concerning the design of autonomous decision making agents that seek to maximize some notion of long term utility in an environment. To make the distinction between agents and environments clear, a boundary is typically drawn depicting the interaction between them (Fig. 2). For simplicity, let us assume time is discretized and agent-environment interaction happens in steps. Then, at every time step, the environment provides the agent with a description of the world (the state) and the agent selects and transfers an action to be executed in the environment. In turn, the environment provides the next state of the world after the action is attempted and additionally a reward.

\[ a \]

\[ \begin{array}{c}
\text{Agent} \\
\text{Environment} \\
\end{array} \]

\[ s_{t+1}, r \]

Figure 2: Agent-environment interaction.

A favorite mathematical formalization of the concepts and terminology above is through the Markov Decision Process (MDP). An MDP is defined by a tuple of four quantities:

- \( S \) - The state space, or set of states the environment, and agent within it, are allowed to occupy.

- \( A(s) \) - A set of actions available to the agent, as a function of its state, that, when one is selected, force a transition in the environment.

- \( R(s, a, s') \) - A scalar reward received by the agent as a consequence of a transition.
- \( P : S \times R \times S \times A \rightarrow [0,1] \) - The dynamics model, sometimes referred to as just the model, defines the probability of observing a particular reward and next state, conditioned upon the current state and selected action, or \( P(s_{t+1}, r_{t+1} | s_t, a_t) \). Often times, as is true for this proposal, the conditional \( P(r_{t+1} | s_t, a_t, s_{t+1}) \) is assumed to be deterministic, and hence the reward function can be written as a function of the state, action and next state. So the model is simply \( P(s_{t+1} | s_t, a_t) \).

Together, they form the \( \langle S, A, R, T \rangle \) tuple.

Note the interesting property that each quantity in an MDP can be calculated based on the current state, or the state and action immediately preceding it. The history of states and actions that brought the agent there are irrelevant. This is said to be the Markovian property and is a standard assumption in the RL literature (Sutton and Barto, 2018). This assumption does not limit the applicability of reinforcement learning as much as it may seem at first as states can be constructed to ensure the Markovian property. This becomes especially important in partially observable MDPs, where the agent may not have access to the full state, \( s_t \), at a given time. Chapter 3 discusses how memory and attention can be used to approximate this property in a particular type of partially observable MDP.

In the mathematical structure of the MDP, the goal of an agent is to maximize the expected value of the cumulative sum of the rewards

\[
G_t = \mathbb{E}_P[r_{t+1} + r_{t+2} + \cdots + r_T | s_t].
\]

This formulation works well for episodic tasks, or tasks that have clearly defined end states. But for continual tasks that do not have a clear termination, this expectation may not be defined. In this case, a discount factor is added to the MDP formulation, a fifth member of the tuple \( \langle S, A, R, T, \gamma \rangle \).

\( 0 \leq \gamma < 1 \) keeps the sum

\[
G_t = \mathbb{E}_P[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \cdots | s_t] = \mathbb{E}_P[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} | s_t]
\]

finite as long as the rewards are bounded.

Through its actions, the agent can manipulate \( G_t \) by selecting the transitions it takes in the environment. Maximizing this expected sum is typically formulated as the goal of reinforcement
learning. The class of solutions is often expressed as a policy, \( \pi : S \rightarrow A \), a function that maps each state to an action. The optimal policy, \( \pi^* \) is the one that maximizes \( G_t \) for every state in \( S \).

The policy can be stochastic, leading to a probability distribution over actions given each state, \( \pi(a|s) \).

Along with the policy, a convenient mathematical tool is the value function, \( V : S \rightarrow \mathbb{R} \). The value of a state is the expected cumulative sum of rewards conditioned upon a specific policy selecting actions starting from that state.

\[
V^\pi(s_t) = \mathbb{E}_{P,\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}|s_t].
\]

Similar to \( \pi^* \), there is an optimal value function, \( V^* \) that expresses the optimal value from each state.

Schaul et al. (2015a) introduced the concept of Universal Value Function Approximators (UVFAs). UVFAs approximate the value function conditioned upon a goal in addition to the state, \( V : S \times G \rightarrow \mathbb{R} \). The optimal policy, \( \pi^*(a|s; g) \), in this case, maximizes the probability of achieving a particular goal, \( g \), from any state.

The value of a state is linked to the one succeeding it by a relationship known as the Bellman’s equation,

\[
V^\pi(s_t) = \mathbb{E}_{P,\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k}|s_t] = \mathbb{E}_{P,\pi}[r_{t+1} + \gamma \sum_{k=1}^{\infty} \gamma^k r_{t+1+k}|s_t] = \mathbb{E}_{P,\pi}[r_{t+1} + \gamma V(s_{t+1})|s_t] = \sum_a \pi(a|s_t) \sum_{s_{t+1}} P(s_{t+1}|s_t, a)[R(s_t, a, s_{t+1}) + \gamma V^\pi(s_{t+1})].
\]

With a known dynamics model, \( P \), the optimal policy and value functions can be found using the dynamic programming algorithms of policy and value iteration (Sutton and Barto, 2018). Reinforcement learning, though, is the study of learning the optimal policy and value functions when the dynamics of the environment are unknown.
2.1.1 Q learning

Q learning is an algorithm to estimate the optimal policy and value function in the case of unknown and uncertain dynamics. It introduces the *action-value* function,

\[
Q^\pi(s, a) = \mathbb{E}_{P, \pi} \left[ \sum_{k=0}^{\infty} \gamma^k r_{t+1+k} | s_t, a_t = a \right].
\]

The \( Q \) values estimate the utility of taking an action in a state and then acting optimally afterwards. Bellman’s equation can be extended for \( Q \) values as

\[
Q^\pi(s, a) = \sum_{s_{t+1}} P(s_{t+1} | s_t, a) [R(s_t, a, s_{t+1}) + \gamma \sum_{a_{t+1}} \pi(a_{t+1} | s_{t+1}) Q^\pi(s_{t+1}, a_{t+1})]
\]

\[
= \mathbb{E}_{s_{t+1} \sim P(s_{t+1} | s_t, a)} \left[ R(s_t, a, s_{t+1}) + \gamma \sum_{a_{t+1}} \pi(a_{t+1} | s_{t+1}) Q^\pi(s_{t+1}, a_{t+1}) \right].
\]

The Bellman’s optimality equation gives a formula for calculating the optimal \( Q \) values.

\[
Q^*(s, a) = \mathbb{E}_{s_{t+1} \sim P(s_{t+1} | s_t, a)} \left[ R(s_t, a, s_{t+1}) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \right].
\]

The optimal value and policy are related to the \( Q \) function as,

\[
V^*(s) = \max_a Q^*(s, a)
\]

\[
\pi^*(s) = \arg \max_a Q^*(s, a).
\]

Without knowing the model \( P \), this value can be estimated using samples. \( Q \) values for each state can be incrementally updated from suitable initial values as

\[
Q(s, a) \leftarrow (1 - \alpha) Q(s, a) + \alpha [R(s_t, a, s_{t+1}) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})]
\]

\[
= Q(s, a) + \alpha [R(s_t, a, s_{t+1}) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})] - Q(s, a)
\]

(1)

This is the main result of Q-learning. The value within the brackets is known as the Bellman error and \( \alpha \) is the learning rate. With infinite exploration, i.e. visiting each state an infinite number of times and selecting each action an infinite number of times within them, this update is guaranteed to converge to the optimal \( Q^* \) values (Sutton and Barto, 2018).
2.1.2 Actor Critic Methods

As an alternative to purely value-based methods, there are policy-based methods for Reinforcement Learning. Policy-based methods are sometimes preferable because most value-based methods including Q-learning implement deterministic policies. It may be desirable for the agent’s policy to be stochastic, which is more naturally handled using policy-based methods. Continuous action spaces are also more amenable to policy-based methods. Policy-based methods aim to directly optimize the return $G_t$ by manipulating the agent’s policy.

For stochastic policies the undiscounted

$$G_t = \mathbb{E}_{P,\pi} \left[ \sum_{k=0}^{\infty} r_{t+1+k} | s_t \right]$$

$$= \mathbb{E}_{P} \left[ \sum_{k=0}^{\infty} \pi_\theta(a|s_{t+1+k}) \cdot r_{t+1+k} | s_t \right]$$

assuming that the policy is parameterized by $\theta$.

To maximize $G_t$, one can take the gradient of it with respect to the policy parameters and use gradient ascent. By the policy gradient theorem,

$$\nabla G_t = \mathbb{E}_{P,\pi} \left[ \sum_{k=0}^{\infty} \nabla_\theta \log \pi_\theta(a|s_{t+1+k}) \cdot G_{t+1+k} | s_t \right]$$

This is known as the policy gradient (Sutton and Barto, 2018) and because of the expectation, it is computable using samples from the current policy in the environment. The REINFORCE algorithm (Williams, 1992) uses Monte Carlo rollouts to estimate this gradient. But Monte Carlo estimates can lead to high variance in the data samples. Intuitively, due to uncertainties in the environment dynamics, the same policy can lead to widely different outcomes from the same starting position. This can manifest in returns that are very different, leading to high variance gradients. One way to reduce the variance is by subtracting a baseline from the return within the gradient,

$$\nabla G_t = \mathbb{E}_{P,\pi} \left[ \sum_{k=0}^{\infty} \nabla_\theta \log \pi_\theta(a|s_{t+1+k}) \cdot (G_{t+1+k} - b) | s_t \right]$$

Note that the baseline $b$ should be independent of the policy parameters. The baseline could simply be the average reward seen by the agent so far, or a running average. The most natural baseline to use, though, is the value function.
\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\nMethods that estimate simultaneously the value and the policy gradient of a state are known as Actor Critic methods. The actor refers to the agent’s policy, which is acting in the environment, and the critic is the value function estimator that judges how promising a state is. Variants of the basic Actor Critic framework are used throughout this proposal, such as PPO (Schulman et al., 2017), DDPG (Lillicrap et al., 2015), A3C (Mnih et al., 2016), etc.

2.1.3 Deep RL

The functions \( \pi, V, Q, \) etc. can be learned exactly for each state or state-action pair. This type of learning is referred to as tabular learning. But for large state and action spaces, this becomes quickly infeasible. A better way would be to generalize these quantities across similar states. Deep reinforcement learning makes use of deep neural networks to approximate such functions.

Mnih et al. (2015) first showed that the \( Q \) function can be stably approximated for the suite of Atari games (Bellemare et al., 2013) by a deep neural network. Since then, deep learning has been successfully applied to many RL methods (Mnih et al., 2016; Lillicrap et al., 2015; Finn et al., 2016; Schulman et al., 2017; Schaul et al., 2015b). In this proposal, we make use of several deep RL algorithms to learn from hallucinated data.

2.1.4 Exploration in RL

There exists a fundamental tension hidden within Reinforcement Learning that is not solved by good generalization. An agent must explore the outcomes (transitions and rewards) of taking different actions in various states before it can be expected to have a good estimate of the optimal policy, especially in stochastic environments. At the same time, the agent must also exploit what it already knows to go further in the environment, and perhaps explore there. This is known as the exploration-exploitation trade-off and every RL agent must be endowed with some method of solving this fundamental problem.

A common solution is adding noise while selecting actions. In the discrete case, the most common way to do so is called epsilon greedy exploration. With a small \( \epsilon \) probability, the agent
selects a random action, rather than the one recommended by the current policy. For continuous actions, a small amount of Gaussian noise $\mathcal{N}(0, \epsilon)$ can be added to the recommended action.

Randomly altering actions can force the agent to see new states but oftentimes leads to diffuse Brownian motion like behavior. The agent can spend a long time near the initial states, rarely making progress towards completion of the task. If rewards in the environment are sparse, this can lead to slow learning. There is a large body of work dealing with how to take advantage of general principles such as curiosity (Schmidhuber, 1991; Storck et al., 1995; Pathak et al., 2017; Burda et al., 2018b), visitation frequency (Machado et al., 2018; Bellemare et al., 2016), exploration from reachable states (Ecoffet et al., 2019), etc. to help aid exploration.

This proposal introduces the idea that directly altering trajectories the agent has experienced using learned models can also significantly speed up or enable reinforcement learning. This can be seen as hallucinating experiences that would have otherwise required exploration.

### 2.1.5 Online vs. Offline

Training can be done online or offline. In the offline setting, a fixed, pre-collected dataset is available to the agent which must be used to learn how to act in the environment. No further data can be collected. In contrast, during online training the agent learns while interacting with the environment. This is the case we focus on throughout this proposal.

### 2.1.6 On-Policy vs. Off-Policy

A common distinction in RL algorithms is on-policy learning vs. off-policy learning. On-policy refers to learning directly from the behavior policy, or the policy currently being followed by the agent. Off-policy is when the learning happens about one policy, often the greedy optimal policy, but from data that is generated following some other behavior policy. Q-learning is an example of off-policy learning. It considers only the $\langle s_t, a_t, r_t, s_{t+1} \rangle$ tuple for learning, i.e. the immediate state, action, next state, and reward. It assumes that following this transition, the agent will act optimally. Hence, it can learn about the values of the current state as if the agent if following the optimal policy everywhere else. An on-policy variant is known as SARSA (Rummery and Niranjan, 1994). It uses the full tuple $\langle s_t, a_t, r_t, s_{t+1}, a_{t+1} \rangle$, i.e. the actual action taken by the agent in the next step instead of the greedy action as in equation (1).
Both on and off-policy methods are used in this proposal. In Chapter 3, we employ the on-policy algorithms PPO (Schulman et al., 2017) and A2C (Mnih et al., 2016; Wu et al., 2017b) specifically because the underlying state representation changes as the agent learns. On-policy methods can handle this sort of non-stationarity better as they are learning only from the transitions under the current policy. Whereas in Chapter 4, I employ off-policy algorithms, Double Deep Q-Networks (DDQN) (Van Hasselt et al., 2016) and Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015), specifically because they sample data generated from a behavior policy in the past. This allows for alteration of sampled past trajectories or learning about hallucinated future trajectories.

2.1.7 Model Based Learning

Another way to go about solving the Reinforcement Learning problem is by estimating the dynamics model, $P$, from environment samples. Methods such as $Q$ learning are referred to as model-free reinforcement learning because they do not directly try to learn the model. Quantities such as value functions can be seen as filtering the reward function through the long term dynamics leading out from a state. As such, model-free methods can be very efficient, only learning about the dynamics of the environment when necessary.

The general idea for model-based learning is to learn the model, i.e. the effect of taking an action in a state, and use it to plan, selecting the optimal predicted outcome at each step. For example, a deep neural net can be trained to predict the most likely $s_{t+1}$ given $s_t$ and $a_t$. Now, at each step, one can iterate through the available actions in each state and select the one that produces the most promising next state according to our model. There are cases where learning the model explicitly can be a lot more efficient than model-free algorithms. One example is transfer learning. Once the dynamics of an environment are learned, they can be used to plan over multiple tasks with differing reward functions. Another case is detailed in Chapter 3 of this proposal. Here, learning the model allows the agent to hallucinate the missing pieces of its state effectively at each step. This provides a bias when learning the policy using model-free methods with the hallucinated data. The agent is able to learn faster with model-based hallucinations than with standard methods.

Closely connected is the idea of the inverse dynamics model. This function tries to predict the
action that was taken, given the initial and final states of transition, i.e. $a_t = I(s_t, s_{t+1})$. This model can also be trained in the environment using supervised learning over the trajectory data.

2.2 Short Term Memory

Here I will introduce key ways short term memory has been employed in RL literature. The most common reason for employing a memory architecture in RL has been to construct a state that is close to Markovian from non-Markovian observations. This is desirable because RL algorithms assume the inputs to them are Markovian but most real-world problems do not readily fit into this assumption. Therefore, the study of how to construct a state such that the most important features for decision making are contained in it at every step is important RL research.

The most basic way of organizing short term memory employed in deep RL was by Mnih et al. (2015) in their deep Q networks. Atari observations are non-Markovian due to some artifacts such as flickering frames or lack of velocity related information in a single frame. The core technique employed was very simple, the last $k$ frames of observations were simply stacked together to form a state, with $k = 4$ being the most common. This technique has no learnable parameters and hence requires no data to train but was effective in solving that task.

Next, Mnih et al. (2016) employed an LSTM (Hochreiter and Schmidhuber, 1997) to be able to condense long term information into a state. LSTM is the most commonly employed form of short term memory not only in Reinforcement Learning (Wierstra et al., 2010; Narasimhan et al., 2015; Hausknecht and Stone, 2015; Bakker, 2002), but also in deep learning in general (Hochreiter and Schmidhuber, 1997; Sak et al., 2014; Yu et al., 2019). In short, it uses a combination of input, output and forget gates, each with their own trainable parameters, to propagate information forward and gradients backward over a relatively large number of steps.

Recently, there have been more general memory architectures such as the Differentiable Neural Computers (Graves et al., 2016) which uses a structured external memory and trainable neural networks as separate read and write heads to solve a variety of memory-related supervised and reinforcement learning tasks. Here the memory itself is more structured that the LSTM, where traditionally a single vector, or cell, stores the information at each step. Another structured memory architecture that is relevant to this proposal is the Neural Map (Parisotto and Salakhutdinov, 2017).
It enforces a 2D structure to the memory, which is beneficial for image observations as it preserves the structural bias of the data. It employs a learnable write head that can choose to overwrite existing memory or preserve it for a long duration. The work proposed in Chapter 3 employs these techniques as well as others to reconstruct trajectories using partial observations and hallucinations.

### 2.3 Generative Adversarial Networks

The field of using neural networks as generative functions has exploded since the advent of the Generative Adversarial Networks, or GANs (Goodfellow et al., 2014). Two networks, the Generator $G$ and the Discriminator $D$, compete in a zero-sum game. The generator attempts to create a sample from a target distribution, such as a dataset of images. The discriminator has access to the distribution and the output of the generator and learns to tell them apart. The loss of the discriminator on the synthetic data is passed to the generator, which tries to maximize it. Hence, the generator is trying to “fool” the discriminator all the while the discriminator is getting better at telling apart the fakes from the real. This general technique of training two networks against one another is referred to as adversarial training.

In the original formulation, $G$ is trying to maximize the quantity

$$\mathbb{E}_{z \sim p(z)}[\log D(G(z))],$$

where $z$ is a random vector that the generator is conditioned on, sampled from $p(z)$ which can be any distribution such as the normal or the uniform. Hence, the generator is trying to maximize the log-likelihood of the discriminator classifying the generated samples as real.

$D$ is simultaneously trying to maximize

$$\mathbb{E}_{x \sim R}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))],$$

where $R$ is the distribution of the real data.

#### 2.3.1 Conditional GANs and Auxiliary Conditional GANs

Mirza and Osindero (2014) proposed a variant of GANs to control, to a certain extent, the data generated by the generator network. The discriminator and generator were allowed to be conditioned on desired properties of the generated data, such as object class.
Hence the utility to be maximized is,

$$\max_G \mathbb{E}_{z \sim p(z)}[\log D(G(z))] + \max_D \mathbb{E}_{x \sim R}[\log D(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))].$$

Conditional GANs, or C-GANs made generating specific kinds of images possible, such as digits from the MNIST dataset (LeCun et al., 1998) of a desired number.

Auxiliary Conditional GANs, or AC-GANs, refine this idea by treating the additional information as an auxiliary prediction task (Odena et al., 2017). The generator is conditioned as normal on the auxiliary variable, but the discriminator now has to predict its value rather than being provided it. The discriminator also outputs the real vs. fake label as in vanilla GANs.

The discriminator’s utility then becomes

$$\max_D \mathbb{E}_{x \sim R}[\log D(x) + \log P(c|x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z))) + \log P(c|G(z))],$$

where $c$ is the conditioning variable.

By separating the task of discrimination and classification, the generator and discriminator training is improved, as feedback on the class and quality of the generated sample can be provided independently. In other words, the discriminator now contains two functions, the real/fake detector, and the classifier, that share weights and benefit from the training of one another. AC-GANs proved to be more stable during training and allowed generation of higher quality images over a wide variety of datasets.

Radford et al. (2015) introduced DCGANs that improve the quality of image generation by using all convolutional operations in the networks.

### 2.3.2 Wasserstein GANs

Marked improvement in stability of training GANs arrived with the Wasserstein variant (Arjovsky et al., 2017). WGAN, as it is known, relies on the Earth mover distance between the generated and real data distributions to penalize the generator.

Gulrajani et al. (2017b) further improved the stability and resistance to mode collapse of WGANs by converting the constraint to a gradient penalty on the discriminator. This is the final version of GANs I will employ in Chapter 4 of this proposal. Combining all the benefits of
the above variants, the improved Wasserstein auxiliary DCGAN (iWADCGAN) proved to produce stable and realistic hallucinations in trajectories of RL agents.
Reinforcement learning (RL) algorithms have successfully employed neural networks over the past few years, surpassing human level performance in many tasks (Mnih et al., 2015; Silver et al., 2017; Berner et al., 2019; Schulman et al., 2017). But a key difference in the way tasks are performed by humans versus RL algorithms is that humans have the ability to focus on parts of the state at a time, using attention to limit the amount of information gathered at every step. We actively control our attention to build an internal representation of our surroundings over multiple fixations (Fourtassi et al., 2017; Barrouillet et al., 2004; Yarbus, 2013; Itti, 2005). We also use memory and internal world models to predict motions of dynamic objects in the scene when they are not under direct observation (Bosco et al., 2012). By limiting the amount of input information in these two ways, i.e. directing attention only where needed and internally modeling the rest of the environment, we are able to be more efficient in terms of data that needs to be collected from the environment and processed at each time step.

By contrast, modern reinforcement learning methods often operate on the entire state. Observing the entire state simultaneously may be difficult in realistic environments. Consider an embodied agent that must actuate its camera to gather visual information about its surroundings learning...
how to cross a busy street. At every moment, there are different objects in the environment competing for its attention. The agent needs to learn to look left and right to store locations, heading, and speed of nearby vehicles, and perhaps other pedestrians. It must learn to create a live map of its surroundings, frequently checking back on moving objects to update their dynamics. In other words, the agent must learn to selectively move its attention so as to maximize the amount of information it collects from the environment at a time, while internally modeling motions of the other, more predictable parts of the state. Its internal representation, built using successive glimpses, must be sufficient to learn how to complete tasks in this partially observable environment.

We consider the problem of acting in an environment that is only partially observable through a controllable, fixed-size, hard attention window (see figure 3). Only the part of the state that is under the attention window is available to the agent as its observation. The rest must be inferred from previous observations and experience. We assume that the location of the attention window at every time step is under control of the agent and its size compared to the full environment state is known to the agent. We distinguish this task from that of learning soft attention (Vaswani et al., 2017), where the full state is attended to, weighted by a vector, and then fed into subsequent layers. Our system must learn to 1) decide where to place the attention in order to gather more information about its surroundings, 2) record the observation made into an internal memory and model the motion within parts of the state that were unobserved, and 3) use this internal representation to learn how to solve its task within the environment.

Our approach for controlling attention uses RL to maximize an information theoretic objective closely related to the notion of surprise or novelty (Schmidhuber, 1991). It is unsupervised in terms of environment rewards, i.e. it can be trained on offline data (states and actions) without knowing the task or related rewards. We discuss this in more detail in section 3.4.

Memory also plays a crucial role in allowing agents to solve tasks in partially observable environments. We pair our attention control mechanism with a memory architecture inspired largely by Du and Narasimhan (2019)’s SpatialNet, but modified to work in partially observable domains. This short term memory architecture, called the Dynamic Memory Map (DMM), is able to predict parts of the world that are not immediately observed by utilizing previous observations and learned dynamics models of object behavior. The world model aims to keep the memory updated so that
it can make accurate predictions of the entire (unobserved) world state for the next step, while
the attention is guided to where the world is not easily predictable. This could be either because
observations have not been collected from there for some time or because it contains objects with
difficult to predict movements. The attention keeps difficult to model objects fresh in memory and
by tracking them through time, helps in learning of their dynamics. This is described in more detail
in section 3.3.

Empirically, we show in section 3.6.1 that our system is able to reconstruct the full state image
including dynamic objects at all time steps given only partial observations. Further, we show in
section 3.6.2 that the internal representation built by our attention control mechanism and memory
architecture is sufficient for the agent to learn to solve tasks in this challenging partially observable
environment.

3.1 Related works

Using hard attention for image classification or object recognition is well studied in computer
vision (Alexe et al., 2012; Butko and Movellan, 2009; Larochelle and Hinton, 2010; Paletta et al.,
2005; Zoran et al., 2020; Welleck et al., 2017). Attention allows for processing only the salient or
interesting parts of the image (Itti et al., 1998). Similarly, attention control has been applied to
tracking objects within a video (Denil et al., 2012; Kamkar et al., 2020; Yu et al., 2020). Surprisingly,
not a lot of recent work exists on the topic of hard attention control in reinforcement learning
domains, where a sequential decision making task has to be solved by using partial observations
from under the attention.

Inspired by human perception, computer vision techniques for saliency detection prioritize pro-
cessing only the “interesting” image regions (Itti et al., 1998). However, these techniques generally
focus on local low-level image features and are task-agnostic. Although, object detection methods
include selecting task-relevant regions likely to contain objects (Van de Sande et al., 2011) they
were developed for static environments (images) and the attention mechanism is not learned. Our
attention mechanism operates in a dynamic environment and we introduce a co-training method
that learns where to glimpse in conjunction with performing the task. Other models in deep learn-
ing have shown the utility of co-training attention with the task objective. For instance, image
captioning (Xu et al., 2015) and image generation (Gregor et al., 2015). However, all of these techniques do have access to the whole image and their focus is to reduce computation or distraction instead of dealing with partial observability.

Mnih et al. (2014) proposed a framework for hard attention control in the classification setting and a simple reinforcement learning task. Their approach consists of using environment rewards to train an attention control RL agent. Our approach differs mainly in that we train the attention control using our novel information theoretic objective as reward. Mnih et al. (2014)’s approach leads to a task specific policy for attention control, whereas our approach is unsupervised in terms of the task and can be applied generally to downstream tasks in the environment. Our approach also differs in that we use a memory architecture that is more suited to partially observable tasks with 2D images as input, compared to a RNN used by Mnih et al. (2014).

Mapping has been studied in computer vision and robotics with a range of sensing modalities using techniques like simultaneous localization and mapping (Fuentes-Pacheco et al., 2015; Snavely et al., 2008). However, these approaches take a geometric approach while our approach relies on learning.

There has been much prior work on memory and world models for reinforcement learning (Ha and Schmidhuber, 2018; Graves et al., 2016; Hausknecht and Stone, 2015; Khan et al., 2017). The work closest to our own is Du and Narasimhan (2019)’s SpatialNet, which attempts to learn a task-agnostic world model for multi-task settings. Our memory architecture is largely inspired by SpatialNet and adapted to work in the partially observable setting. We also use their PhysEnv environment to evaluate our approach. Closely related work is Neural Map (Parisotto and Salakhutdinov, 2017), which uses a structured 2D memory map to store information about the environment. Their approach also applies to partially observable RL tasks, but the attention is fixed to the agent. In contrast, we consider the problem of learning to control an attention window that can move independently of the agent location. Recently, Freeman et al. (2019) showed that world models can be learnt by simply limiting the agent’s ability to observe the environment. They apply observational dropout, where the output of the agent’s world model, rather than the environment state, is occasionally provided to the policy. We consider the related scenario where only a part of the true environment state is provided to the agent at each time step and the rest must be modeled.
using previous observations.

Finally, mutual information has been used to train self-supervised RL agents. This line of work originates in curiosity driven and intrinsically motivated RL (Schmidhuber, 1991; Pathak et al., 2017; Bellemare et al., 2016). Typically, some notion of predictive error or novelty about an aspect of the environment is optimized in lieu of environment rewards. Multiple papers have successfully used different formulations of mutual information to learn how to efficiently explore the environment without extrinsic rewards (Mohamed and Rezende, 2015; Houthooft et al., 2016; Achiam et al., 2018; Gregor et al., 2016; Eysenbach et al., 2018a; Sharma et al., 2019b). We apply the idea of using mutual information to the problem of curiosity driven attention control.

3.2 Preliminaries

In this section, we will attempt to formalize the components of the partially observable reinforcement learning problem under study.

3.2.1 Internal representation

![Diagram of internal representation](image)

Figure 4: The agent starts with a representation ($\mu_t$) from which it attempts a reconstruction ($\tau_t$) of the likely current full (unobserved) state $s_t$. Then, it picks an attention location ($l_t$) and receives an observation of the state from under the attention ($o_t$). We assume that the size of the attention window is known to the agent. The observation is written into $\mu_t$ to form $\mu^w_t$. The entire map is then stepped through an internal world model to update the dynamical objects, forming $\mu_{t+1}$, the representation for the next step.

First, we give a brief description of how the agent stores observations in an internal representation of its surroundings (figure 4). A map ($\mu$) tracks the full environment state ($s$), which is never
directly observed. The map is empty at the start of the episode and gets sequentially written into
as observations are made. The agent also creates a reconstruction of the full state (\(\tau\)) at every time
step based on its map. This will become useful later for training. For now, it suffices that \(\mu_t\) is the
map before an observation is made and \(\mu^w_t\) is the one after the observation is made. \(\mu_{t+1}\) is the
map after the dynamics of the system for the next time step are taken into account.

3.2.2 The two agents

We formulate the solutions to controlling the attention at every time step and completing the
environment task as two separate reinforcement learning agents. The attention location is controlled
by the *glimpse agent* (the eye in figure 4), and actions within the environment are taken by the
regular agent. These two agents have their own separate MDPs with their \(\langle S, A, R \rangle\) tuples defined
below.

The glimpse agent’s state at every timestep is \(\mu_t\). Its set of actions is all possible attention
locations within the full state (Width \(\times\) Height actions). Its reward is based on the information
theoretic objective discussed in section 3.4. Thus, the glimpse agent is provided the map before
an observation is made and it must decide *where* the observation should be made from in order to
optimize its reward.

The environment agent acting in the environment receives as input \(\mu^w_t\), i.e. the internal represen-
tation after the observation has been recorded. Its actions are the normal set of actions in the
environment and its reward is the normal environment reward. We emphasize that neither agent
has access to the full state at any time. They must both act based on the internal representation
alone. They also cannot make multiple observations from the same environment state. Once an
observation is made, an action must be selected that will change the environment state. In the
next section, we will describe in detail how \(\mu^w_t, \mu_{t+1}\) and \(\tau_t\) are formed and how the internal map
is trained through a sequence of partial observations.

3.3 Dynamic Memory Map

Memory plays a critical role in enabling agents to act in partially observable environments. It
allows the agent to remember parts of the state that are not under direct observation. Along with
memory, the agent must be able to model the dynamics of objects in its environment, as it may not
receive observation of them for long periods. Finally, the effect of its own actions must be reflected in its belief about the environment. For this purpose, we design a special recurrent memory unit for visual environments called the Dynamic Memory Map (DMM). DMM is inspired largely by Du and Narasimhan (2019)’s work on SpatialNet, but modified to handle partial observations and work in tandem with the glimpse agent.

3.3.1 Memory Modules

The DMM consists of three major modules: write, step and reconstruct.

Write

This module encodes an incoming observation, \( o_t \), into the current memory representation, \( \mu_t \). The observation is first passed through a series of convolutional operations, \( W \), possibly downsampling it for a more efficient representation. Then it is blended with \( \mu \) using a series of convolutions, \( B \). Finally it is written into the memory but only in the locations where the observation was made, leaving the rest of the memory intact. The write operation can be written as:

\[
\mu_t^w = C_t \ast B(W(o_t), \mu_t) + (1 - C_t) \ast \mu_t
\]

where \( C_t \) is a mask which is 1 under the attention window and 0 otherwise.

Step

This module is responsible for modeling the dynamics of the environment and updating the memory to track the full state. Objects in the environment can be static or dynamic and may be affected by the agent’s actions. To efficiently model the changes to memory precipitated by different types of objects, we split the memory into bundles. Each bundle is assigned a fixed number of channels within the memory representation. We assign each bundle a third of the memory channels.

The first static bundle, \( \mu_s \), remains unchanged by the step module. This allows the write head to efficiently encode static objects within this bundle. This bundle acts as a skip connection between two glimpses at the same location at different times, allowing gradients to pass through long stretches of memory updates. Next, the dynamic bundle, \( \mu_d \), consists of a series of residual convolutions, similar to SpatialNet (Du and Narasimhan, 2019). This bundle learns about the dynamics of objects in the scene, independent of what the agent is doing. The entire map within
this bundle is updated at every step. Finally, the ego bundle, $\mu^e$, updates the map conditioned on the action selected by the agent, $a_t$. This attempts to predict the effect of the agent’s actions on the environment at every step.

The updates carried out by this module can be represented as:

$$
\begin{align*}
\mu^{s}_{t+1} &= \mu^{s}_t \\
\mu^{d}_{t+1} &= D(\mu^{d}_t) \\
\mu^{e}_{t+1} &= E(\mu^{e}_t, a_t) \\
\mu_{t+1} &= \mu_t + \langle \mu^{s}_{t+1}, \mu^{d}_{t+1}, \mu^{e}_{t+1} \rangle
\end{align*}
$$

**Reconstruct**

This module converts the memory representation to a reconstruction of the full state using a series of deconvolutions ($R$).

$$
\tau_t = R(\mu_t)
$$

A reconstruction can also be made immediately after the observation, i.e. $\tau^w_t = R(\mu^w_t)$. Reconstruction is essential for training the other two modules of DMM as we explain in section 3.3.2.

The reconstruction error, or the discrepancy between the observation and the reconstruction under the attention window can be back-propagated through the write and step layers.

### 3.3.2 Training loss

The write loss, $L_w$ is incurred under the current attention window immediately after the observation is made into $\mu^w_t$. This ensures that the immediate observation is correctly encoded into the memory and trains the write and reconstruction modules.

$$
L_w = C_t \cdot ||\tau^w_t - o_t||_2
$$

The step loss, $L_s$ is incurred after stepping the map and under the attention window in the next step.

$$
L_s = C_{t+1} \cdot ||\tau_{t+1} - o_{t+1}||_2
$$
This trains the step module, $S$, to accurately model the motion of objects in the entire state, so that a faithful representation of the state can be guessed before the next observation is taken. Once the next observation is made, we can calculate the difference between what the agent’s model expected to be at the glimpse location and what actually was there. In figure 4 this is the difference between the images under the yellow rectangle in $\tau_t$ and $a_t$. This quantity is the amount of surprise in the dynamics model and we will use this again when training the glimpse agent.

In addition to the reconstruction loss, the absolute value of the output by the write and step modules is also penalized in order to regularize the contents of the DMM. The total loss for a single step within the rollout is (where $\alpha = \beta = 0.01$ here):

$$L_t = L_w + L_s + \alpha * C_t \cdot |\mu^w_t| + \beta * |S(\mu^w_t, a_t)|.$$  

We sum this loss over the entire rollout and jointly minimize it over all steps. The reconstruction error at every location can be back-propagated to the step an observation was recorded there.

Additionally, the representation $\mu$ can be compressed for efficiency. In our experiments, we reduced the image size by a factor of 4 by downsampling through the write network. At the beginning of each episode, the DMM is initialized with all zeros.

### 3.4 Maximizing mutual information to control attention

The intuition behind our approach is that attention should be used to gather information from the environment when and where it is required. The DMM is constantly modeling its environment and attention should be directed to parts of the world where the model is uncertain. This directly informs the agent about difficult to model parts of the environment, thus reducing uncertainty in the state as a whole. Secondly, by focusing on areas with hard to predict dynamics, more data can be collected from those areas, updating the model for future predictions.

#### 3.4.1 Mutual information objective

The idea of reducing uncertainty about the environment can be captured in the language of information theory by using mutual information. Specifically, we propose selecting the location of the attention such that its mutual information with the state at the following step is maximized,

$$\max_{l_t} I(s_{t+1}; l_t).$$  (2)
Where $l_t$ is the location of the attention window at time $t$. It is selected according to a stochastic policy $l_t \sim \pi^\theta_{\text{glimpse}}(\cdot | \mu_t)$ parameterized by $\theta$. Eq. 2 can be expanded as

$$\max_\theta \mathcal{H}(s_{t+1}) - \mathcal{H}(s_{t+1} | l_t), \quad (3)$$

where $\mathcal{H}$ denotes the entropy. This expansion brings out a very intuitive explanation of the objective. We would like to pick an attention location that maximizes the reduction in entropy (uncertainty) of the environment state after an observation is made.

### 3.4.2 Curiosity driven attention

In this section, we show how maximizing the amount of surprise can lead to maximizing the mutual information. Let us begin by further expanding eq. 3.

$$I(s_{t+1}; l_t) = \mathcal{H}(s_{t+1}) - \mathcal{H}(s_{t+1} | l_t)$$

$$= - \sum_{s_{t+1}} p(s_{t+1}) \log p(s_{t+1}) + \sum_{l_t} p(l_t) \sum_{s_{t+1}} p(s_{t+1} | l_t) \log p(s_{t+1} | l_t)$$

$$= - \mathbb{E}_{s_{t+1} \sim p(s_{t+1})} [\log p(s_{t+1})] + \sum_{l_t} \sum_{s_{t+1}} p(s_{t+1}, l_t) \log p(s_{t+1} | l_t)$$

$$= - \mathbb{E}_{s_{t+1} \sim p(s_{t+1})} [\log p(s_{t+1})] + \mathbb{E}_{s_{t+1}, l_t \sim p(s_{t+1}, l_t)} [\log p(s_{t+1} | l_t)]$$

So, maximizing mutual information between the attention location and the environment state is equivalent to minimizing $\log p(s_{t+1})$ and maximizing $\log p(s_{t+1} | l_t)$ under expectation. The first term is the log-likelihood of the next state prior to selecting $l_t$. Computing it requires marginalization over all possible $l_t$, which is prohibited by our environment as only a single partial observation from the full state can be provided to the agent. The second term, $\log p(s_{t+1} | l_t)$, is computable at a single attention location from a single state and hence we will focus on maximizing this term.

Now, assume the agent’s belief over the true environment state at the next step, $s_{t+1}$, is represented by a Gaussian with unit variance around the reconstruction, $\tau_{t+1}$. So, $s_{t+1} \sim \mathcal{N}(\tau_{t+1}, I)$ and

$$\log p(s_{t+1} | \tau_{t+1}) \propto \log \exp[-(s_{t+1} - \tau_{t+1})^T(s_{t+1} - \tau_{t+1})/2]$$

$$= - \sum_i (s^i_{t+1} - \tau^i_{t+1})^2 / 2 \quad (4)$$

27
So, minimizing the squared difference between $s_{t+1}$ and $\tau_{t+1}$ leads to maximizing $\log p(s_{t+1}|\tau_{t+1})$. Maximizing $\log p(s_{t+1}|\tau_{t+1})$ in turn means maximizing $\log p(s_{t+1}|l_t)$, since $\tau_{t+1}$ is constructed using the deterministic functions $W$, $S$, and $R$ once $l_t$ has been picked (see section 3.3).

In order to minimize the squared difference between $s_{t+1}$ and $\tau_{t+1}$, the agent needs to pick a location $l_t$ that observes the maximum error between $s_t$ and its reconstruction $\tau_t$. This is because at step $t$, if the agent observes the location with the highest reconstruction error, i.e. the least likelihood $p(s_t|\tau_t)$, it will ensure that at the next time step $t + 1$ the agent has the most recent information on that region and can make a good reconstruction of it. Hence, maximizing $\log p(s_{t+1}|l_t)$ is equivalent to picking an $l_t$ that will lead to the highest reconstruction error at step $t$.

$$\log p(s_{t+1}|l_t) = \sum_{i \in I_t} \frac{(s_i^t - \tau_i^t)^2}{2}$$

where $I_t$ forms the set of indices under the window at location $l_t$. Directly optimizing this quantity with respect to $l_t$ is not possible as it can only be computed once an observation has been made and only at a single location. Hence, the glimpse agent must learn to predict the location that is most likely to result in high reconstruction error before an observation is made. Our approach is to formulate the glimpse agent as a reinforcement learner with the reconstruction error as its reward.

An interpretation of this objective is that the glimpse agent is surprise seeking, or curiosity driven. It is attending to parts of the state that are novel in the sense that they are difficult for the agent’s current model to predict. Another interpretation is that the glimpse agent is acting adversarially to the DMM’s model of the environment. At each step the glimpse agent tries to focus on parts of the state that are the most difficult for DMM to reconstruct. By doing so, it is indirectly creating a curriculum of increasingly difficult to model aspects of the environment.

### 3.4.3 Full training objective

Let us look at a different expansion of the objective in eq. 2,

$$I(s_{t+1}; l_t) = \max_{\theta} \mathcal{H}(l_t) - \mathcal{H}(l_t|s_{t+1})$$

The first term is $\mathcal{H}(l_t)$, the entropy over the attention location, which is controlled by the policy of the glimpse agent. This term can easily be maximized by standard RL algorithms, such as A2C (Mnih et al., 2016; Wu et al., 2017a), using weighted entropy maximization.
Combining the first term from equation 6 and second term from equation 3, the final objective that we use to train the glimpse agent is

$$\max_{\theta} H(l_t) - H(s_{t+1}|l_t) \equiv \alpha H(\pi_{\text{glimpse}}^\theta) + \max_{\theta} \sum_{i \in I_t} (s_i^t - \tau_i^t)^2 / 2,$$

where $\alpha$ is the entropy weighting set to 0.001 in our experiments.

Note that this is the mutual information objective from equation 2 with an entropy regularization term on the glimpse policy. The first term in equation 2, $H(s_{t+1})$, does not depend on $\theta$ and hence is excluded from the final objective.

It is possible to design a variational algorithm that maximizes the approximate mutual information based on the expansion in eq. 6 alone (instead of mixing terms from both the expansions). Empirically, however, we found that our objective performs better than the variational approximation to mutual information. See section 3.9 for more detail.

### 3.5 Details of data collection

In PhysEnv we used a previously trained agent to generate 500,000 steps of demonstrations. We experienced that using a random policy in PhysEnv led to quick deaths of the agent with insufficient episode lengths for learning. The data are sequences of full states and agent actions from multiple episodes. In PhysEnv, the map size was $21 \times 21$ compared to the full state size of $84 \times 84$. The size of the attention window was fixed to be $21 \times 21$ in PhysEnv (6.25% of the state visible).

### 3.6 Results

We evaluate our approach in two environments: a gridworld environment (figure 8) and PhysEnv (figure 3). In both environments, the task is to navigate to a goal while avoiding obstacles and enemies. The agent’s actions are movement in the four discrete cardinal directions. In the gridworld, the state is a one-hot 2D encoding of the objects in the environment (agent, walls, enemies and goal). The agent’s and enemies’ movements are restricted to the grid squares and the dynamics of the enemies are simple: moving in straight lines until an obstacle is hit in which case it reflects. In PhysEnv, observations are provided as RGB images and the motions of the objects are more varied.
The glimpse agent + DMM are first trained on offline data of state and action trajectories (no reward) collected from each environment. More details on data collection are provided in section 3.5. We evaluate this part of the training by how accurately the full state is reconstructed by DMM at each step. Once the glimpse agent + DMM have converged, i.e. the reconstruction loss has plateaued, we freeze their parameters and initialize an agent within the environment. The glimpse agent + DMM serve as fixed components within the agent while it learns to solve a task. The agent is trained using an off-the-shelf RL algorithm PPO (Schulman et al., 2017) and is evaluated by the total environment reward it collects during testing episodes.

We compare our method against baselines inspired from related work. In the follow baseline the attention is always focused on the agent itself. This is a formulation of the partial observability as seen in Parisotto and Salakhutdinov (2017). Next, the environment baseline mimics the approach by Mnih et al. (2014), where the environment rewards are used to train the glimpse agent. Finally, the random baseline moves the attention randomly at each time step.

3.6.1 State reconstruction

![Mean Pixel Prediction Error](image)

**Figure 5:** Mean pixel prediction error for each model. Each model was trained on sequences of fixed length 25 but we predict states for a range of sequences from short (5) to long (100). Reconstruction error is measured over the full state at each time step and averaged over the sequence.

First, we evaluate how well the DMM reconstructs the full dynamic state from a series of glimpses. We collect a dataset of 500k images from the *PhysGoal* environment (Du and Narasimhan,
2019) by using rollouts from a trained agent for this task. The agent’s actions while performing the

task are also recorded as our method and some of the baselines make use of it. The size of the state

is 84 × 84 pixels and we assume an attention head of 21 × 21 pixels is available for all the methods.

We train the DMM along with several ablations of our approach and baselines from prior work.

- **LSTM**: Underlying memory representation used is a flat LSTM. The write/step/reconstruct

operations for this memory are as usual without a masked write head and bundles, but the

step module is conditioned on agent action.

- **SpatialNet**: The closest memory architecture to ours in the literature, does not employ

masked write heads, bundles or action conditioning.

- **SpatialNet + AC**: SpatialNet with additional action conditioning during step.

- **DMM + R-A**: DMM trained with glimpses from random locations in the state. An ablation

highlighting the importance of co-training attention with memory.

- **DMM + Co-A**: Our full method trained with co-attention

Figure 5 shows the results for testing sequences, excluded from the training set, of varying

lengths. At just 5 glimpses, all methods have equal difficulty in reconstructing the full state (it

would take at least 16 glimpses to cover the entire state area once). From 10 glimpses onward, our

method consistently beats all baselines, even past the sequence length of the training set (25).

<table>
<thead>
<tr>
<th></th>
<th>l2 reward (ours)</th>
<th>random</th>
<th>follow</th>
<th>environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gridworld</td>
<td>0.00555</td>
<td>0.007666</td>
<td>0.01941</td>
<td>0.00827</td>
</tr>
<tr>
<td>PhysEnv</td>
<td>0.0521</td>
<td>0.0614</td>
<td>0.1186</td>
<td>0.0826</td>
</tr>
</tbody>
</table>

Table 1: Per-pixel L2 loss between agent’s reconstruction τ and the ground truth full state s. All

methods only see partial observations, but must attempt to reconstruct the full state from internal

representation. In both environments, our method has the lowest reconstruction error.

We measure the performance of the glimpse agent + DMM by the average reconstruction error

to the true full (unobserved) state in unseen testing episodes. 25 consecutive observations from 6

unseen test episodes are passed through each model and error is averaged over reconstructions at

each time step. This gauges the ability of our attention mechanism and memory to reconstruct

the full state using only partial observations. Table 1 shows that using surprise driven reward
(our approach) for learning attention control leads to a lower reconstruction error than the other baselines. See figures 8 through 12 for visualizations of the reconstructions and training error. In particular, figure 10 and 11 show qualitative comparisons between our method with the baselines. Broadly speaking, our method learns to focus on dynamic objects within the environment and preserve them over many steps in the memory, whereas the baselines lose track of or blur objects as seen in their state reconstructions.

![Figure 6](image)

Figure 6: (left) Object-wise breakdown of reconstruction error in gridworld. (right) % of frames in which the object occurs, i.e. the frequency with which the attention is focused on a particular object.

Figure 6 shows an analysis of what objects are under focus during training in the gridworld environment and their corresponding reconstruction error. The attention forms an interesting curriculum for training the DMM, where it learns to focus on different objects in the environment over time. It starts with walls, then moves to the agent itself, then goal and finally the enemies. Thus, it moves gradually from static to dynamic, harder to model parts of the state, allowing the DMM to learn how to accurately model each object. This very interesting curriculum-like behavior emerges naturally and is not pre-programmed into DMM or the attention agent. It is a consequence of maximizing surprise at every time step. See section 3.7 for a more detailed discussion.

### 3.6.2 Reinforcement Learning

For the RL task, we compare against the baseline, *full*, where the full state is provided to the agent to solve the task. This is an upper limit for how well any method that only receives partial
observations can perform. Figure 7 shows the average episodic reward over five testing episodes during the training of the agent. We only show results for the PhysEnv environment because all baselines perform similar to our approach for the RL task on the gridworld, despite having poor state reconstructions as shown above. This is likely because the size of the environment is small and the dynamics of the enemies are simple, therefore an accurate representation of the state is not required for task completion.

In PhysEnv, our method performs the best, although worse than the upper bound of the full baseline. The environment baseline performs poorly in PhysEnv. This may be because the attention policy of the environment baseline has high entropy (figure 11c heatmap) and ends up exploring large parts of the state within an episode. But it does not learn to focus on the most unpredictable parts of the state as our method does. The follow baseline fails to learn at all, likely because it only focuses on the agent and does not explore the rest of the state and hence is unaware of the location of the enemies and the goals. This result shows that our method is able to construct a representation of the full state that can be used to achieve goals within the environment better than the representation of the baselines, although not as well as if the full state was known.

We also show the performance of our agent (l2 online) in the condition where the DMM and
glimpse agent is trained online along with the policy, all components starting from random initialization. This agent did not have a pretraining step and was trained entirely on random exploratory data, therefore obviating the need of a pretraining dataset. The performance is slightly worse than the pretrained agent. This is an encouraging result in a particularly challenging setting of a non-stationary state space as the memory representation and glimpse policy are changing while the agent learns.

3.7 Visualizations of Glimpses During Training

We will discuss where the attention focuses and how that effects the reconstruction (and underlying belief) as the training progresses for the gridworld environment. Please refer to 6 for this discussion.

Initially, the attention learns to focus on the wall objects and the corresponding reconstruction error for the walls decreases rapidly (6.b). This is because the walls are long objects in the environment, present at multiple pixels and side-by-side, providing the largest and most reliable source of error under the attention window initially. The error is provided as reward to the glimpse agent, hence it seeks more walls in the state.

Soon, the error it receives from walls decreases as the agent’s DMM learns to persistently record the location of stationary walls over multiple time steps. Then around 10000 iters, the attention switches to focusing on the agent’s location itself. This is because the agent’s apparent erratic movements provide a larger reward due to their unpredictability than the stationary walls can anymore. This has dual consequences. First, since now the DMM is seeing a lot of examples of the agent’s movement, it learns to accurately model the agent based on an initial state and its actions and the reconstruction error corresponding to the agent falls. At its peak, over 70% of glimpses contain the agent within them (figure 6.b). Interestingly, the error corresponding to the wall and goal objects rises as some forgetting of the representation and dynamics of these objects occurs. A little before 20000 iters though, the agent has learnt to represent simultaneously all three objects and it switches attention to the enemies, the only remaining un-modeled objects in the environment. The enemies move predictably, bouncing off walls and other objects, but otherwise in straight lines. The agent does not have direct access to their actions and must infer their motion over multiple time steps using only glimpses.
Figures 8 and 9 visualize the behavior of the glimpse agent during the initial stages of training. In each subfigure, the first column shows the full (unobserved) state. The second column is a heatmap showing the glimpse policy, i.e. the likelihood the glimpse agent will select a location for placing attention. The location that was sampled is indicated in the first column by the yellow square. The final, third, column is the reconstruction $\tau$ based on the agent’s current belief about the environment. Initially, (100 iters), the attention mostly moves around randomly as can be seen by the heatmap being active throughout the state. The reconstruction does not correspond to anything in the real state since the agent has only just begun learning. The glimpse agent here is receiving high reward for pretty much every location in the state. By 2000 iters, the glimpse agent has learnt to mainly focus on the wall once it locates it. This can be seen in the high probabilities of the glimpse agent policy near the wall locations and the attention repeatedly fixating on the wall. The agent is simultaneously striving to learn to represent the walls as they are showing up a lot in its training data, and steadily reducing the reconstruction error (the glimpse agent’s reward) coming from walls.

By iter 11000, the focus has shifted to tracking the agent as can be seen in the heatmap. The heatmap shows that the glimpse agent is singularly focused on the agent’s movements as the error reward it is getting from the walls has dropped off and the agent’s movements are not fully predictable yet. In the figure for iter 19000, it can be seen that the agent can now track its own location very reliably with an initial position and the actions it is taking, without having to focus the attention on itself. In fact, after step 8 when the agent is discovered, the attention almost never moved back to it and yet the location in the reconstruction perfectly matches that of the unobserved full state. Hence the agent’s location based reward source has largely dried up. The heatmap of attention policy shows that the glimpse agent has become interested in the goals and walls again briefly as it learns to represent all three objects together. It can also be seen from the reconstructions here that the agent is unable to accurately track and model the enemies locations at this point.

By iter 21000, the attention has shifted focus to the remaining object in the state that is producing a high reconstruction error, i.e. the enemy. Once the enemy is discovered on step 14, the glimpse agent’s policy puts all probability near its location so it can keep observing and model
its behavior.

The glimpse agent creates a natural curriculum for training the models within DMM, starting at large stationary objects that do not move, then moving to the agent’s followed by the objects with the most complex dynamics. We conjecture this automatically discovered curriculum is the reason our system is able to learn better reconstructions than the baselines.
Figure 8: Visualizations of glimpse behavior in gridworld environment. Each sub-figure is the same testing episode at different stages of training, with the rows corresponding to the time within the episode.
Figure 9: Visualizations of glimpse behavior in PhysEnv environment.
Next we show qualitative comparisons of the glimpse policy and state reconstruction between our method and the baselines. Each subfigure in figures 10 and 11 shows the same episode of length 20 played out with different glimpse agents. The first column is our method, the second shows a uniform random policy for the glimpse agent, the third shows a glimpse agent trained on environment (task) rewards, and lastly the fourth shows the glimpse following the agent. In both environments, it can be seen that our method learns to focus on dynamic objects within the environment and the resulting state reconstruction shows that these objects are preserved in the memory.

Note that the glimpse policy learned by the environment rewards is significantly more entropic than the one learned by our method, despite having the same entropy weighting hyperparameter. We conjecture that this is because the task reward does not send a clear and consistent learning signal to control the attention window to create better reconstructions, or as seen in our RL results, to facilitate downstream tasks. As such, the reconstructions appear qualitatively similar to the random baseline.

The random baseline quantitatively performs the closest to our approach, but it fails to focus on dynamic objects consistently like over time, and hence fails to track these objects over long periods in the episode. The reconstructions therefore have streaks and blurs where the objects were last spotted and the physics model has not learnt how to update their positions properly.
Figure 10: Glimpse behavior for fully trained models of our method and the baselines in gridworld. Each subfigure is a different baseline, with columns within the subfigure being the full unobserved state, heatmap of attention policy, and reconstruction of full state respectively.
Figure 11: Glimpse behavior for fully trained models of our method and the baselines in PhysEnv.
3.8 Error curves for glimpse agent + DMM

Figure 12: (left) PhysEnv environment. (right) Gridworld environment. Reconstruction error (L2) from the full state during training of the DMM + glimpse agent. All approaches have converged by 200k iters, with our approach having the lowest error, i.e. the most faithful reconstruction of the full unobserved state.

These training curves correspond to the results in table 1. The reconstruction error for each method has stabilized. Our method has the lowest reconstruction error for both environments.

Figure 13: (left) PhysEnv environment. (right) Gridworld environment. Episodic reward for glimpse agent during training (higher is better).

Figure 13 shows the episodic reward for the glimpse agent on six testing episodes as the training progresses. This corresponds to the mutual information objective in equation 5, or the reconstruction error under the attention window in the next step. The modules within DMM are constantly improving to reduce prediction error and hence the reward goes down as the training progresses.
But the glimpse agent is simultaneously trying to maximize the objective by searching for areas within the state that still provide high error. Our method maximizes this objective in both environments. This leads to lower overall reconstruction error as the glimpse agent is learning to predict where the highest source of error is going to be at the next step. The environment baseline performs close to random and the follow baseline gathers the least reward in both environments.

### 3.9 Variational Approximation to Mutual Information Maximization

The mutual information between attention and environment state as expanded in eq. 6 is

\[
I(s_t; l_t) = H(l_t) - H(l_t|s_t).
\] (7)

In our algorithm, we combine the first term in this expansion with another term in eq. 3. This works well empirically. Theoretically, however, it is computationally possible to maximize both terms in eq. 7. However, as we shall show here, this does not work well practically.

The first term can be maximized by using maximum entropy RL algorithms for the attention policy as discussed in the main text. Let us consider the second term.

\[-H(l_t|s_t) = - \sum_{s_t} p(s_t) H(l_t|s_t) \]

\[= \sum_{s_t} p(s_t) \sum_{l_t} p(l_t|s_t) \log p(l_t|s_t) \]

\[= \mathbb{E}_{s_t, l_t \sim p(s_t, l_t)} [\log p(l_t|s_t)] \]

While \(\log p(l_t|s_t)\) is not directly known, we can construct a variational function, \(q_\phi(l_t|s_t)\), to approximate it.

Since \(KL(p(l_t|s_t)||q_\phi(l_t|s_t)) \geq 0\), we have,

\[\sum_{l_t} p(l_t|s_t) \log p(l_t|s_t) \geq \sum_{l_t} p(l_t|s_t) \log q_\phi(l_t|s_t) \]

\[\mathbb{E}_{s_t, l_t \sim p(s_t, l_t)} [\log p(l_t|s_t)] \geq \mathbb{E}_{s_t, l_t \sim p(s_t, l_t)} [\log q_\phi(l_t|s_t)] \]
Plugging this into eq. 7,

\[ I(s_t; l_t) \geq \mathcal{H}(l_t) + \mathbb{E}_{s_t, l_t \sim p(s_t, l_t)} [\log q_\phi(l_t | s_t)] \]

Therefore, we can maximize the mutual information by maximizing this lower bound w.r.t. the policy parameters of the glimpse agent. This can be achieved by providing \( \log q_\phi(l_t | s_t) \) as a reward to the glimpse agent and using RL algorithms to maximize its long term value along with the policy entropy. As such, maximizing the mutual information.

We can additionally condition \( q_\phi \) on the internal state and action of the agent at previous time step. Since we do not have access to the full environment state, \( s_t \), we substitute it with the agent’s internal state. The final variational function looks like \( q_\phi(l_t | \mu_t, \mu_{t-1}, a_{t-1}) \). \( q_\phi \) must be trained to match the true posterior \( p(l_t | s_t) \) to provide accurate rewards to the glimpse agent.

The question now is how to learn \( q_\phi \)? In this work, it is a neural network trained on samples. This is done by minimizing the KL divergence,

\[
\min_{\phi} \mathbb{E}_{s_t, l_t \sim p(s_t, l_t)} [KL(p(l_t | s_t) || q_\phi(l_t | \mu_t, \mu_{t-1}, a_{t-1}))].
\]

We do not know \( p(l_t | s_t) \) directly, but we have samples of it from the agent’s glimpse policy. Given the current and previous internal representations, along with the agent action, \( q_\phi \) tries to guess where the glimpse location must have been.

Intuitively, \( q_\phi \) will be high at locations of attention that are easy to identify by looking at the previous state, the resulting state, and the agent action. Otherwise, its output will be flat and hard to predict. Since \( q_\phi \) is provided to the glimpse agent as reward, it will be rewarded for selecting easily identifiable locations and prefer such a policy. These can be locations that produce a high amount of localized surprising information in \( \mu_t \) as compared to \( \mu_{t-1} \), as unexpected appearance of goal or enemies somewhere will signal that the attention was just moved there, hence producing a high likelihood in \( q_\phi \). Conversely, low reward locations will be ones that are hard to identify by \( q_\phi \) as they do not impact any specific part of \( \mu_t \), leaving it largely the same as \( \mu_{t-1} \), or observe a location that was already well-modeled by the agent and hence do not leave an identifiable \textit{mark} on \( \mu_t \) that is a tell-tale sign for \( q \).
Figure 14: State reconstruction and observation reconstruction errors in gridworld environment. Observation reconstruction error falls to almost zero for both methods while full state reconstruction remains high for variational approach.

The above method does not work well in practice. Figure 14a shows the reconstruction error of the unobserved state with our proposed method and the variational approach. Figure 14b shows the observation reconstruction error, which is the error in reproducing just the observation under the attention immediately after it is made. It can be seen that the latter quantity goes to near zero for both methods, meaning that immediate observations are being faithfully recorded into memory and reconstructed. Yet, the overall state reconstruction error remains high for the variational approach, which means that the agent is not able to retain observations over time steps as the attention moves around the state and the environment model is not learning to predict the evolution of the state over time without direct observation.

A reason why the variational approach fails in practice could be that we do not have direct access to $p(l_t|s_t)$ in order to train $q$. Instead we must rely on the agent’s internal estimate of the environment state, $\mu_t$, which itself is being trained alongside the glimpse agent. This results in noisy rewards to the glimpse agent as the $q$ function is not able to accurately predict where the attention is. As we have seen in section 3.7, attention control creates a curriculum for training the agent’s state estimate. Hence this forms a loop, where attention control is dependent on agent’s state estimate which is in return dependent on the curriculum generated by the attention.

Extensive hyperparameter tuning may also be required for this approach to work as the glimpse
agent can easily default to choosing the same attention location at each time step as it ensures the highest $q(l_t|s_t)$.

3.10 Conclusions

We have presented an approach for using mutual information to control a hard attention window in environments with dynamic objects. From partial observations under the attention window, our system is able to reconstruct the full state at every time step with the least error. The representation learned using our method enables RL agents to solve tasks within the environment, while the baseline methods are unable to learn useful representations on the same memory architecture. This demonstrates that attention control plays a large role in enabling task completion.

Note that our attention control objective is independent of the task. It is solely concerned with gathering information from the environment where it is most unpredictable. This is similar to curiosity driven RL that seeks novel states or surprising transitions. Future work may look into combining it with task-specific attention control by fine-tuning on a task reward. A better foveal glimpse model may also be used instead of the fixed size hard attention used here. We made an assumption that only one glimpse is allowed per environment state. But the rate of collecting glimpses may be variable with respect to environment speed.

Curiosity based approaches tend to suffer from the “noisy TV” problem (Savinov et al., 2018) where the agent tends to get distracted by and focus on random, unpredictable events in the environment. Orthogonal approaches have been developed to counter this common weakness by, for example, taking into account the agent’s own actions (Pathak et al., 2017). These approaches can be combined with our method as well. We leave the incorporation of this in attention control for future work.
IV VISUAL HINDSIGHT EXPERIENCE REPLAY

Deep RL algorithms are highly sample inefficient for complex tasks and learning from sparse rewards can be challenging. In these settings, millions of steps are wasted exploring trajectories that yield no learning signal. On the other hand, shaping the rewards in an attempt to make learning easier is non-trivial and can often lead to unexpected ‘hacking’ behavior (Ng et al., 1999; Randløv and Alstrøm, 1998). Therefore, an important vector for RL research is towards more sample efficient methods that minimize the number of environment interactions, yet can be trained using only sparse rewards. In this chapter of the proposal, I will investigate whether the use of hallucinations to alter the unsuccessful past trajectories to appear successful can lead to faster learning.

To this end, Andrychowicz et al. (2017b) introduced Hindsight Experience Replay (HER), which can rapidly train goal-conditioned policies by retroactively imagining failed trajectories as successful ones. HER was able to learn a range of robotics tasks that traditional RL approaches are unable to solve. But it was only shown to work in non-visual environments, where the state input is composed of object locations and proprioceptive features and it is straightforward to convert any state into a goal. Therefore, it was trivial to substitute a new goal in hindsight and hallucinations were not required. During HER, the precise goal configuration is provided to the agent’s policy throughout training through a universal value function approximator (UVFA) Schaul et al. (2015a). UVFAs provide a simple mechanism for reimagining goals by allowing direct substitution of a new goal in off-policy settings. In many visual environments, though, goal states appear different from other states. Moreover, if the agent’s policy is conditioned solely on its state, goals states have to be sought out in the state image using their distinct visual cues and, in order to reimagine goals, the agent’s observations themselves must be altered retroactively. HER is not directly applicable to such tasks as it provides no such mechanism for altering agent observations.

Yet, we desire for RL agents to quickly learn to operate in the complex visual environments that humans inhabit. For challenging visual domains resembling real-world applications, HER cannot be directly applied as the goal has to be inserted within the visual trajectory itself. To make progress towards this, this proposal outlines a new technique for visual hindsight experience replay that addresses the high sample complexity of RL in such visual environments by combining a
Figure 15: HALGAN hallucinates the presence of goals in unsuccessful trajectories, ending in a perceived success. In this environment, the agent’s task is to search for a pebble randomly placed in its surroundings and collect it by approaching and centering it in its view. The top row shows a failed trajectory during exploration. The bottom row replays the same trajectory with a hallucination inserted by HALGAN at each step such that a pebble appears to be collected in the final state.

hallucinatory generative model, HALGAN, with HER to rapidly solve tasks using only state images as input to the agent policy. To retroactively hallucinate success in a visual environment, the failed trajectory of state observations must be altered to appear as if the goal was present in its new location throughout. HALGAN minimally alters images in snippets of failed trajectories to appear as if the desired goal is achieved by the end (see figure 15). HALGAN is trained using relatively few snapshots of near goal images, where the relative location of the agent to the goal is annotated beforehand. It is then combined with HER during reinforcement learning, where the goal location is unknown but agent location can be estimated, to hallucinate goals in desired locations along unsuccessful trajectories in hindsight. We primarily focus on tasks where the completion of a goal can be visually identified within the agent state.

The key contributions of this work are to expand the applicability of HER to visual domains by providing a way to retroactively transform failed visual trajectories into successful ones and hence allow the agent to rapidly generalize across multiple goals using only the state as input to its policy. We aim to do so in conjunction with minimizing the amount of direct goal configuration information required to train HALGAN. We believe that the sample complexity reduction HALGAN provides is an important step towards being able to train RL policies directly in the real world.

4.1 Related Work

Below I review recent works most relevant to this proposal.
4.1.1 Hindsight Experience Replay

The essential idea is to store each trajectory, \( Traj_i = s^i_0, s^i_1, ..., s^i_T \), with a number of additional goals, typically future agent states, along with the originally specified goals. An off-policy algorithm employing an experience replay is used in conjunction with a UVFA that allows for direct substitution of new goals in hindsight. The reward is also modified retroactively to reflect the new goal being replayed. In particular, HER assumes that every goal, \( g \in G \), can be expressed as a predicate \( f_g : S \rightarrow \{0, 1\} \). That is to say, all states can be judged as to whether or not a goal \( g \) has been achieved in them. Thus, while replaying a trajectory with a surrogate goal \( \overline{g} \), one can easily reassign rewards along the entire trajectory as

\[
    r_{\overline{g}}(s^i_t) = \begin{cases} 
        1 & \text{if } f_{\overline{g}}(s^i_t) = 1 \\
        0 & \text{otherwise.}
    \end{cases}
\]

Andrychowicz et al. (2017b) report that selecting \( \overline{g} \) to be a future state from within the same (failed) episode leads to the best results. This training approach forms a sort of implicit curriculum for the agent. In the beginning, it encourages the agent to explore further outwards along trajectories it has visited before. The agent soon learns to associate the hindsight rewards with the surrogate goals, \( \overline{g} \). Over time, the agent is able to generalize to achieve any goal in \( G \).

4.1.2 Wasserstein GANs

We employ a Wasserstein ACGAN Gulrajani et al. (2017a); Odena et al. (2017) as our generative model because of its stability, realistic outputs, and ability to condition on the desired class. A typical W-ACGAN has a generator, \( H \), that takes as input a class variable and a latent vector of random noise. It generates an image that is fed into the discriminator, \( D \) which rates the image on fidelity to the training data. As an auxiliary task, \( D \) also predicts class membership. The Wasserstein distance between the distributions of real, \( p_R \), and generated, \( p_H \), images is used as a loss to train the combined model. W-ACGANs produce realistic hallucinations that will allow the agent to easily generalize from imagined goal states to real ones. Realistic insertion of goals was not an issue in HER because a new goal could directly be substituted in a replayed transition without any modification to the observations.
4.1.3 Generative Models in RL

In recent years, generative models have demonstrated significant improvements in the areas of image generation, data compression, denoising, and latent-space representations, among others Goodfellow et al. (2014); Brock et al. (2018); Karras et al. (2017); Chen et al. (2016); Vincent et al. (2008). Reinforcement learning has also benefited from incorporating generative models in the training process. Ha and Schmidhuber (2018) unify many approaches in the area by proposing a Recurrent Neural Network (RNN) based generative dynamics model Schmidhuber (1990) of popular OpenAI gym Brockman et al. (2016) and VizDoom Kempka et al. (2016) environments. They employ a fairly common procedure of encoding high dimensional visual inputs from the environment into lower dimension embedding vectors by using a Variational Auto Encoder (VAE) Kingma and Welling (2013) before passing it on to the RNN model. Another approach, GoalGAN Held et al. (2017), uses a GAN to generate goals of difficulty that match an agent’s skill on a task. But it assumes that goals can easily be set in the environment by the agent and does not make efficient use of trajectories that failed to achieve these objectives. Generative models have also been used in the closely related field of imitation learning to learn from human demonstrations or observation sequences Ho and Ermon (2016); Edwards et al. (2018b); Schroecker et al. (2019). Our approach does not require demonstrations of the task, or even a sequence of observations, only relatively few random snapshots of the goal with a known configuration which we use to speed up reinforcement learning.

4.1.4 Goal Based RL

Some recent work has focused on leveraging information on the goal or surrounding states to speed up reinforcement learning. Edwards et al. (2018a) and Goyal et al. (2018) learn a reverse dynamics model to generate states backwards from the goal which are then added to the agent’s replay buffer. The former work assumes that the goal configuration is known and backtracks from there, whereas in the latter, high-value states are picked from the replay buffer or a GoalGAN is used to generate goals. The latter work also learns an inverse policy, \[ \pi(a_t|s_{t+1}) \] to generate plausible actions leading back from goal states. In contrast, we focus on minimally altering states in existing failed trajectories already in the replay buffer to appear as if a goal has been completed in them.
This avoids having to generate entirely new trajectories and allows us to make full use of the environment dynamics already present in previous state transitions.

Others have focused on learning goal-conditioned policies in visual domains by using a single or few images of the goal Xie et al. (2018); Zhu et al. (2017). Nair et al. (2018) train a $\beta$-VAE Burgess et al. (2018) on state images for a threefold purpose: (1) to sample new goals during training, (2) to use the Euclidean distance between feature encodings of current and goal images as a dense reward, and (3) to retroactively substituted goals with images generated by the VAE and reassign rewards appropriately. Here also, the set of goals $G$ is assumed to be the same as the set of states $S$, i.e. goal states appear similar to regular states and hence they are easy to swap back and forth. This works well for domains where the goal is separately provided to the policy along with the agent state, and where states do not have to be modified for changing goals. In this work, we attempt learning in domains where the goal may or may not be present in a particular agent state and hence has to be added in during hindsight and the goal image is not separately provided to the agent’s policy.

## 4.2 The missing component in HER

First, we will formally discuss what is missing from the original HER formulation that does not allow it to readily extend to visual domains. In the next section, we describe in detail how the use of hallucinatory generative models can help bridge the gap.

Andrychowicz et al. (2017b) make the assumption that “given a state $s$ we can easily find a goal $g$ which is satisfied in this state”. This requires a mapping, $m : S \rightarrow G$ from every state $s \in S$ to a goal $g \in G$ that is achieved by the agent being present in $s$. While this mapping may be relatively straightforward to hand-design for real-valued state spaces, its analog for visual states cannot be constructed easily. For example, if $S$ is the plane of real values in $\mathbb{R}^2$, the goal may be to achieve a particular $x$-coordinate. So in the state $(x = 0.5, y = 1.0)$, a goal that is satisfied is simply $\bar{g} : x = 0.5$. But in visual environments, goal states may have visual features distinct from regular states. Imagine if the agent must instead navigate to a beacon on a 2D plane using camera images as input. In order to convert a state into one in which a goal is satisfied, the beacon must be visually inserted into the state image itself. In this case, a function capable of mapping states
to goals is difficult to hand design.

To fully utilize the power of HER, not only should the agent be able to hallucinate goals in arbitrary states, but also consistently in the same absolute position throughout the failed trajectory. Note that with each step along the trajectory, the position of the goal (the beacon) changes relative to the agent’s and thus the agent’s observation must be correctly updated to reflect this change. The goal must appear to be solved in a future state along every step of the trajectory in a way that is consistent with the environment dynamics. Only then can we make use of the existing transitions along the trajectory for replay with hallucinated as well as original goals. Thus, visual settings require the mapping $m$ to be extended along the entire trajectory, $m_V : S^T_{Traj} \rightarrow G$, where $Traj$ is the space of failed trajectories and $T$ is the maximum length of a trajectory snippet. Every state along the trajectory, $s_0, s_1, \ldots, s_T \in S_{Traj}$, must be modified by the mapping into a near goal state, $\tilde{s}_0, \tilde{s}_1, \ldots, \tilde{s}_T$, that is consistent with the final hallucinated goal state, $\tilde{s}_T = \tilde{g}$ (see figure 15). This work’s main contribution lies in showing that such a mapping can be learned by a generative model using some knowledge of the goal in the form of goal snapshots with known relative location.

Lastly, the use of UVFAs does not extend to visual settings where the agent’s policy is not conditioned on a specific goal location, but where a desired goal must be searched for within the environment using visual cues, such as navigating to a beacon. We show how the learned model mapping unsuccessful trajectories to successful ones can be applied to rapidly train RL agents with policies solely conditioned on their state image.

### 4.3 Approach

To address the shortcomings of HER in visual domains, we adopt a two-part approach. First, a generative model, HALGAN, is trained to modify any existing state from a failed trajectory into a goal or near goal state. Then, during reinforcement learning, HALGAN generates goal hallucinations conditioned on the configuration of the agent in the current state relative to its own configuration in a future state from the same episode. Details on each component of HALGAN and how it all fits together to generate consistent hallucinations of the goal are discussed next.
4.3.1 Minimal Hallucinations of Visual Goals

Our aim is to minimally alter a failed trajectory in order to turn its states into goal or near-goal states. This makes full use of existing trajectories and does not require HALGAN to re-imagine the environment dynamics or unnecessary details about the goal state such as the background.

To this end, we train an additive model such that the generator, $H$, has to produce only differences to the state image that add in the goal. To obtain a hallucinated image $\tilde{s}$ with the goal at the final state of the trajectory, $s_T$, we compute,

$$
\tilde{s}_t = \text{Tanh} \left( s_t + H \left( c(s_t; s_T), l \right) \right),
$$

where $c(s_t; g)$ is the relative configuration of the robot to a desired goal state $g$ and $l$ is a random latent conditioning vector. $\text{Tanh}$ re-normalizes the hallucinated state image to $[-1, 1]$. Note that hallucinations are generated independent of other states in the trajectory. Temporal consistency between hallucinations on two consecutive states of a failed trajectory is only enforced through the relative configuration to a common final desired goal state.

The hallucinated state, $\tilde{s}_t$, along with a state $s_r$ sampled from dataset $R$, is then fed to the discriminator $D$ to compute the discriminative loss,

$$
L_D = \mathbb{E}_{\tilde{s} \sim p_H} [\log D(\tilde{s})] - \mathbb{E}_{s_r \sim p_R} [\log (D(s_r))].
$$

As a result of generating only image differences, the trained hallucinatory model is invariant to some kinds of visual variations, such as background, presence of other objects, etc.

Additionally, a gradient penalty is typically employed in the training of Wasserstein GANs Gulrajani et al. (2017a).

$$
L_\nabla = \mathbb{E}_{\hat{s} \sim P_2} (\| \nabla D(\hat{s}) \|_2 - 1)^2
$$

To further encourage the model to generate minimal modifications to the original failed image, we also add a $L_2$ norm loss on the output of $H$. In our experiments, this helped remove unnecessary elements in the hallucinations such as multiple goals or background elements such as walls.

$$
L_H = \| H (c(s_t; s_T), l) \|_2
$$
4.3.2 Regression Auxiliary Task

Typical ACGANs are conditioned on a discrete set of classes, such as flower, dog, etc Odena et al. (2017). In our approach, the generator is conditioned on the relative configuration of the agent from the desired goal state, which is a real-valued vector $c(s_t; g) \in \mathbb{R}^n$. The auxiliary task for the discriminator is to regress to the real valued relative location of the goal seen in a training image. To train this regression based auxiliary task, we use a mean squared error loss,

$$L_A = \|\overline{c(\overline{s})} - c(s_t; g)\|_2$$  \hspace{1cm} (12)

where $\overline{c(\overline{s})}$ is the relative configuration predicted by $D$. We found it helpful to add a small amount of Gaussian noise to the auxiliary inputs for robust training, especially on smaller datasets.

4.3.3 HALGAN

Our final loss to the combined HALGAN is,

$$L = L_D + \alpha L_N + \beta L_H + \lambda L_A$$ \hspace{1cm} (13)

where, $\alpha$, $\beta$, and $\lambda$ are weighting hyperparameters, which we fix to 10, 1, and 10 respectively.

To summarize, the training procedure is as follows. $H$, conditioned on a randomly drawn desired relative goal location produces a hallucination which is then added to a randomly selected image from a failed trajectory. The discriminator is provided these hallucinated images, as well as ground truth images from $R$ and has to score them on their authenticity and also predict the relative goal location. See figure 16 for a representation of the HALGAN training process and the appendix for more details on the network architectures and training hyperparameters. Figure 17 shows examples of the output from our model for a range of goal configurations.

Data Collection. HALGAN is trained on a dataset, $R$, of observations of the goal where the relative configuration to the agent is known. These snapshots of the goal can be collected and annotated before RL and are only used once to train HALGAN. During RL, hallucinations are created using only the agent’s own configuration, which can be obtained in realistic applications using SLAM or other state tracking techniques Montemerlo et al. (2002). For our experiments, we collect the training data in $R$ by using the last 16 or 32 states of a successful rollout. At the
end of a successful rollout, assuming that the agent’s configuration corresponds to the goal location relative poses can be calculated automatically using only agent configuration. We did not manually inspect all images in $R$ to ensure that the goal is visible, but there was enough relevant data for HALGAN to infer the object of interest. Note that the exact data required are randomly selected snapshots from near the goal, in any order. Only observations of the goal along with the annotated relative configurations are used, no actions have to be provided or demonstrated, which allows the generative model to be independent of the agent and demonstrator action spaces. Thus, the burden of collecting goal information for HER is not entirely eliminated but can be significantly reduced to a few thousand states. We also collect a dataset of failed trajectories. Most off-policy RL methods that employ an experience replay have a \textit{replay warmup} period where actions are taken randomly to fill the replay to a minimum before training begins. This dataset of failed trajectories can be the same as the replay warmup.
4.3.4 Visual HER Using HALGAN

During reinforcement learning, the agent explores the environment using its behavior policy. Snippets of past trajectories are sampled from the experience replay at every step and a few of the failed ones are augmented with goal hallucinations to appear successful. Again, this is in contrast to the regular HER approach or the approach by Nair et al. (2018), where end states were directly designated as goals using a hand-designed mapping and the observations in the failed trajectory did not have to be modified. The detailed process is explained in algorithm 1. The result is that the agent encounters hallucinated near goal states with a much higher frequency than if it were randomly exploring. This, in turn, encourages the agent to explore close to real near goal states.

An important consideration is the retroactive reassignment of rewards. HER uses a manually defined function $f_g(s)$, which decides if the goal $g$ is satisfied in state $s$, to designate rewards during
Algorithm 1 HALGAN+HER

1: **Given:** Trained hallucinatory model $H$, Reward reassignment strategy $r_g(s)$.
2: Initialize off-policy Algorithm $A$. ▷ eg. DDQN, DDPG
3: Initialize Experience Replay $E$ by random exploration.
4: **for** step = 1, $N$ **do**
5: Sample an action according to behavior policy $a_t \leftarrow \pi(s_t)$ in current state.
6: Execute $a_t$ in the environment and observe state $s_{t+1}$, reward $r_t$.
7: Store tuple $(s_t, a_t, r_t, s_{t+1})$ in $E$.
8: Sample minibatch $B$ from $E$ for training.
9: **for** $e = (s_i, a_i, r_i, s_{i+1})$ in $B$ **do**
10:  **if** $c \sim \text{Bern}(p)$ **then** ▷ $p =$ hallucination prob.
11:    Sample $d \sim \text{Unif}(\{0, 1, \ldots, D\})$ ▷ distance to goal state
12:    Compute relative configurations $c(s_i; s_{i+d})$ and $c(s_{i+1}; s_i+d)$.
13:    $s_i \leftarrow s_i + H(c(s_i; s_{i+d}), l)$
14:    $s_{i+1} \leftarrow s_{i+1} + H(c(s_{i+1}; s_i+d), l)$
15:    $r_i \leftarrow r_{s_{i+d}}(s_{i+1})$
16: **end if**
17: **end for**
18: Perform one step of optimization using $A$ on the modified minibatch $B$.
19: **end for**

hindsight. This sort of reward function is hard to hand design in visual environments. Comparing
state and goal images pixel by pixel is typically ineffective. For the purpose of hindsight replay
where a future state is set as the goal, one needs only to compare two states to reassign rewards,
$f_s : S \times S \rightarrow \{0, 1\}$. As mentioned in section 4.1, Nair et al. (2018) use a trained $\beta$-VAE as $f_s$ to
reassign rewards in a dense manner. Here, we make use of access to the agent’s own configuration.
We assume that any two states with similar configuration must satisfy the same goal. During
retroactive reward reassignment, we compare the configuration of the agent in the sampled state
to that at the end of the trajectory. A sparse reward of $+1$ is awarded if they are the same up to
a threshold value.

4.4 Experiments

We test our method on two first-person visual environments. In a modified version of MiniWorld
Chevalier-Boisvert (2018), we design two tasks. The first one is to navigate to a red box located
in an enclosed room (figure 18a top). The second task is to successively navigate, first to the red
box, picking it up by visually centering it, and then carrying it to a green box somewhere else in
the room (see figure 18a bottom). The second environment is a more visually realistic simulated
robotics domain, where a TurtleBot2 Wise and Foote (2011) equipped with an RGB camera is simulated within Gazebo Koenig and Howard (2004). We use gym-gazebo Zamora et al. (2016) to interface with Gazebo. Here, the agent must collect a pebble scattered randomly on a road by approaching and centering it in its visual field (figure 18b). The environment only provides a sparse reward of 1 for achieving the goal. We demonstrate that our method applies easily to both discrete (TurtleBot) and continuous control (MiniWorld) environments.

The size of the near goal dataset, $R$, for the Turtlebot, navigation and successive navigation tasks is 6840, 2000, and 6419 images with relative goal configurations respectively but we also show results on smaller datasets for the Turtlebot environment (figure 18f). In the Turtlebot and MiniWorld navigation tasks, the configuration of the agent is simply it’s $(x, y, \text{yaw})$. In successive navigation, an additional binary field indicates whether the red box is held by the agent. The agent’s relative configuration is calculated with respect to the red box before it is picked up, and the green box afterward. Hallucinations are generated for the agent approaching both boxes. We found it helpful to anneal the number of hallucinations in a batch over time as the agent fills its replay with real goal images. Details of all experimental hyperparameters are provided in the appendix.

**Comparisons.** There is no prior work that attempts HER in visual domains without explicit goal conditioning. Hence, we compare our approach to multiple extensions of existing approaches and standard RL baselines. All baselines used the same hyperparameters as our approach. First, a naive extension of HER into the visual domain, $her$, simply rewards the agent for states at the end of failed trajectories during replay without hallucinating. A second baseline is derived from RIG Nair et al. (2018) which trains goal-conditioned policies with a dense reward based on the distance between the embedding of the sampled state and that of a goal image. RIG’s reimagining of goals relies on the use of UVFAs, which is not possible for our domains where the goal image is unknown. Therefore, we design two variants of this baseline in an attempt to find a suitable comparison. For both, we first train a VAE on near goal images in $R$ and failed state images. Then, during RL, $vae-her$ simply sets the final image in a failed trajectory as the goal and uses the trained VAE to reassign reward for a transition along that trajectory. $rig$ follows a similar dense reward shaping strategy but computes the distance of a state to a randomly sampled goal image in $R$. Hence,
rig-rewards the agent for being in states that look similar to goal states in $R$. The distance-based rewards provided by the VAE in rig-had to be re-scaled by a constant factor of 0.02 to be the same order of magnitude as the environment rewards.

### 4.5 Results

In all of our experiments, HALGAN trained agent begins learning immediately (figure 18). This is due to the realistic hallucinated goals being quickly identified as desirable states. This incentivizes the agent to explore more around goal states. This is in contrast to standard RL which rarely encounters reward and must explore at length to begin the learning process, if at all.

![Figure 18](attachment:image.png)  
Figure 18: (a,b) Near goal states in experimental domains. (c,d) Episodic reward vs. environment steps for MiniWorld tasks averaged over 5 random seeds. (e) Results for Turtlebot task are averaged for 3 random seeds. (f) Minor variance in agent performance in Turtlebot task with decreasing size of HALGAN training dataset. 90% confidence intervals are shown for each plot.

In the discrete TurtleBot pebble collection domain (figure 18e), the naive HER strategy provides a good enough exploration bonus for the agent to explore further and quicker than standard DDQN. It begins learning by 100K steps. The rig-baseline performs only slightly better. HALGAN agent,
by contrast, starts learning to navigate to real goals immediately.

For the continuous control experiments in MiniWorld (figure 18c, 18d), only HALGAN agent is able to learn to complete the task. Note that only positive rewards indicate achievement of the goal. DDPG never encounters any reward during exploration and hence learns to minimize its actions to avoid movement penalty. Naive her initially encourages exploration and hence incurs a heavy penalty, but does not learn to associate the hallucinated rewards with the presence of a goal. vae-her, the augmentation of her with dense rewards from a trained VAE, also proves unsuccessful for either task, demonstrating that dense rewards without hallucinated or real goals in failed trajectories are ineffective for learning in these domains. Only the rig- strategy of providing dense rewards relative to random goal images eventually learns to complete the navigation task for some of the seeds. For the successive navigation task, rig- only learns a working policy on a single seed and the other baselines perform similarly or worse. Interestingly, rig’s dense reward reassignment can be readily combined with our approach of modifying observations, providing directions for future work.

Finally in figure 18f, we show the change in performance on the TurtleBot task due to using fewer training samples in $R$. The effect is only slightly slower learning even for the largely reduced dataset of only 1000 images. The minimalistic hallucinations created by HALGAN require a relatively small amount of training data to provide a significant boost in reinforcement learning.

4.6 Discussion

A major impediment to training RL agents in the real world is the amount of data an agent must collect before it encounters rewards and associates them with goals. High sample complexity makes problems such as the fragility of physical systems, energy consumption, speed of robots and sensor errors manifest themselves acutely. In this work, we have shown that Hindsight Experience Replay can be extended to visual scenarios by retroactively hallucinating goals into agent observations. We empirically demonstrate that by utilizing failed trajectories in such a way, the agent can begin learning to solve tasks immediately. HALGAN+HER trained agent converges faster than standard RL techniques and derived baselines on navigation tasks in a 3D environment and on a simulated robot.
The principle of visually hallucinating goals could potentially be applied to speed up training for many other tasks, for example, avoiding collisions (negative penalty on hallucinated collisions), following a human or object (positive reward for hallucinations within a range of distance), or placing objects in a visually identified zone (hallucinating a visual safety marker). HALGAN is currently conditioned solely on relative agent configuration. Complex visual environments may include cues in the background that influence goal appearance, such as occlusions or lighting. Conditioning HALGAN on features of the current or intended goal state could extend this approach to such environments. Conditioning on a history of states in the trajectory could also enforce further temporal consistency between hallucinations. Other future directions of work include collecting training data for HALGAN online as the agent explores and automatically annotating goal configuration.
V TASK GENERALIZED UNSUPERVISED REINFORCEMENT LEARNING

There has been a flurry of development on the frontier of intrinsic motivation and unsupervised behavior learning in reinforcement learning. Both terms generally refer to learning policies without access to an extrinsic reward function. Intrinsic motivation (Barto, 2013; Oudeyer and Kaplan, 2009) techniques generally focus on improving exploration by using some notion of novelty or surprise as a reward (Schmidhuber, 1991, 2010; Baranes and Oudeyer, 2009; Pathak et al., 2019; Burda et al., 2018b; Houthooft et al., 2016; Şimşek and Barto, 2006; Groth et al., 2021). One way this can be done is to train a predictive model of the environment and use the model’s error as a reward signal. In this way, the agent seeks difficult to predict parts of the state space. This was used in Chapter 3 to create a curriculum for attention control and representation learning. Another way is to use count-based or density estimation methods for calculating state visitation and encouraging the agent to explore less visited parts of the state space (Bellemare et al., 2016). If solving tasks in an environment is highly correlated with exploration, these methods can generalize without seeing any rewards (Burda et al., 2018a).

Alternatively, some approaches discover structure in the environment in the form of skills that accomplish various objectives not related to any single task (Mohamed and Rezende, 2015; Gregor et al., 2016; Eysenbach et al., 2018a; Warde-Farley et al., 2018; Hansen et al., 2019; Sharma et al., 2019a; Pong et al., 2019). Most of these approaches maximize a mutual information objective between a latent vector, corresponding to an individual skill, and some observable from the environment other than the reward, typically some portion of the trajectory. These approaches are very successful in inducing diverse behavior across skills that tend to maximize coverage over the state space.

Despite the significant advances in learning unsupervised behavior, when the task is introduced, generalization is generally poor. One of the reasons could be that none of the approaches consider the prior distribution over useful tasks in an environment. In other words, tasks in the real world may be samples from an underlying distribution assigning non-uniform probability to all possible tasks that may exist. By biasing an unsupervised learner towards this task distribution, one can
make learning of downstream tasks easier.

So far in this thesis, we have discussed settings with a single task and techniques to speed up learning in them. In more realistic scenarios, and certainly in continual or lifelong learning settings, there will be multiple tasks to be solved. Policies learned for certain tasks might help in solving others. In fact, as discussed in Chapter 1, every task in an environment with the complexity of the real world will be like none the agent has encountered before. The agent will have to learn to generalize across similar tasks without needing significant retraining.

In this chapter, I consider the scenario where each new task is drawn from a distribution over tasks. This distribution specifies which tasks are more likely than others. This distribution could be discrete or continuous leading to a finite set of tasks with varying frequency or unique tasks whose probability of being seeing again is exactly zero. The latter is the more realistic scenario as tasks in the real world are rarely exactly the same.

Before proceeding further, I will define some concepts such as tasks and skills that will be helpful in clarifying the contribution.

5.1 Definitions

I have discussed loosely the terms tasks and goals. First, I will provide a more precise definition of them.

Consider the following components of an MDP,

\[ <S, A, T> \]

This is an incomplete decision problem, but it describes fully an environment within which such a problem could exist. It contains the states the agent can exist in, the actions it is allowed to take, and the transitions that follow. I will call this an environment description.

There are multiple instantiations of reward functions that are compatible with this environment description.

\[ R : S \times A \rightarrow \mathcal{R} \]

The domain of these reward functions remains fixed, the state and action spaces of the environment. Different mappings of them to scalar values leads to different reward functions within the same
environment description. Different reward functions will lead to different behavior when a policy
is learnt to optimize them.

A task, \( \tau \), is an MDP formed with a specific instantiation of the reward function within an
environment description. A task is described by the full MDP associated with it, but multiple
tasks can share the same environment description. Hence, there exists a family of tasks that
share environment descriptions but differ in their reward functions.

Next, I will define goals, which will help us narrow down the types of tasks this thesis is
applicable towards.

I define a goal as a state, or set of states, \( G \), which are terminating in nature and provide
high reward to the agent relative to the rest of the state space. The aim of an agent in such
environments is to learn a policy that enters goal states consistently.

Goals can be viewed as describing special kinds of tasks that terminate at a state with high
reward. An environment description can be combined with different goals to form a family of these
specific tasks.

In this work, we will only consider tasks that can be described using goals. Distributions of
such tasks, therefore, can be defined as distributions over goal states.

\[
\tau_i \sim P(G),
\]

where \( G \in S \) is a set of goal states within the state space and \( \tau_i \) is a task.

The idea of skills derives from the field of hierarchical reinforcement learning. Hierarchical
learning holds the promise of speeding up reinforcement learning by reusing policies between dif-
ferent tasks. Instead of learning individual policies for each task from scratch, hierarchical learning
aims to find and remember options or skills that are applicable towards multiple tasks.

An option is a policy that can be activated in certain states and terminates typically when a
desired outcome has been achieved.

Options have a specific mathematical definition,

\[
O :< I, \pi^o, \beta >.
\]
Here, \( I \in S \) is the initiation set of states, \( \pi^o \) is the policy when this option is engaged and \( \beta \) is the termination set.

Multiple options can exist within an agent’s repertoire, but only one can be active at a time. While an option is active, the agent chooses actions according to only that option’s policy. The idea is to break down a complex task into components that can be handled by individual options. The agent then simply learns to initialize and terminate the appropriate series of options to solve a task.

Skills are a related concept to options, but they are defined more loosely.

A skill, \( z \), is a policy that solves a specific task or set of tasks consistently.

Useful skills can be combined with other skills to solve complicated tasks. Typically skills can be initiated in any state and run for a fixed number of steps or till the end of the episode. Usually, for efficiency, instead of learning separate policies for each skill, a combined policy is learned, conditioned on a vector that describes the skill, \( \pi(a|s,z) \).

5.2 Skill Learning

The aim of the this chapter will be to demonstrate the construction of a task-generalized MDP from a distribution of tasks that differ in their goal states. The task-generalized MDP will share the same state space as the underlying MDP of each task, but will replace actions with learned skills.

\[
MDP : < S, Z, R, T >
\]

From a top-down perspective of hierarchical learning, a long and complex task can be broken down into a series of smaller tasks, which in themselves can be broken down further till the level of atomic actions. These sub-tasks and sub-sub-tasks etc. can be solved in series or concurrently to form a solution to the original task. Solutions to the sub-tasks, or sub-policies, are assumed to be simpler to find than the original task as the horizon is shorter. Another assumption is that sub-policies can be reused and recombined at different stages of the original task, resulting in higher sample efficiency.
In the bottom-up view of hierarchical learning, the agent endeavors to discover skills that are usually independent of a particular task. It then combines skills together for completing a combinatorial number of tasks.

It is easier to understand this work with the latter view. Here, we will focus on skills learned independent of a specific task, but that can be used to learn to solve many tasks downstream.

Two common degenerate solutions to skill learning are: 1) specializing to a single task, and 2) trying to solve every task using a single skill. Neither case is desirable as the skills cannot be used for building up to other tasks. Skills are useful if they can be combined to achieve as many interesting goals as possible, but do not attempt to specialize to a single goal.

Recently, there has been progress in learning skills without a specific reward function. In most cases, the agent still learns to optimize for a reward function using reinforcement learning, but the reward calculation is intrinsic to the agent and does not depend on extrinsic environment rewards. In Chapter 3, I introduce a similar idea for intrinsic motivation for learning to control a hard attention window in a partially observable environment.

These approaches learn multiple skill policies conditioned on a skill vector $z$. This vector can be discrete or continuous, implying a finite or infinite number of skills. We will focus on the finite, discrete set of skills case in this thesis.

Formally, a skill policy can be defined as a mapping from pairs of states and skills to actions,

$$\pi : S \times Z \rightarrow A.$$ 

By conditioning the policy on a skill, one can produce distinct behavior across skills, allowing them to accomplish different tasks.

Vanilla reinforcement learning agents learn to map states to actions via the policy to maximize a reward. The reward function is designed as a proxy for a task to be completed by the agent or a goal state to be achieved. In lieu of rewards or a concrete goal, most unsupervised reinforcement learning approaches attempt to maximize some intrinsic notion of information between the environment variables and the skills.
5.3 Curiosity Based Methods

Before describing our approach in detail, I will overview some curiosity-based methods that predate the more structured skill-based learning ideas in this thesis. The core idea behind most curiosity-based methods is to reward the agent for visiting states that are unexpected in some sense. The agent then learns to seek out this novelty. How to measure novelty varies, but, for example, could be how often a state has been visited in the past or how predictable the future states are from the current one.

Curiosity-based learning is a kind of unsupervised reinforcement learning. It is done in lieu of an extrinsic reward function and can help the agent explore its environment before a task is presented. Generalization to downstream tasks is better if they are highly correlated with wide exploration (Burda et al., 2018a). What distinguishes curiosity based methods from skill learning is that the former is used to learn a flat policy that efficiently explores the state space while the latter builds a set of policies that can be used hierarchically to accomplish tasks.

In Schmidhuber (1991), two fully recurrent self-supervised neural networks are used, one for learning the environment model and one for the reactive policy. The intrinsic reward is aligned to increase the model’s knowledge about the world. The work introduces the idea that curiosity is correlated with how much one knows about their environment. But curiosity simply does not direct one towards all unexplored states. Highly harmful states should still be avoided. Hence, Schmidhuber (1991) combine their intrinsic reward with certain permanent goals, such as pain-avoidance.

In Schmidhuber (2010), a formal theory for intrinsic motivation is presented which leads to curiosity and creativity. Intrinsic reward is maximized by seeking novel or surprising patterns, which leads to better future prediction and data compression. Pathak et al. (2017) extend this idea to high dimensional image state spaces. Additionally, they only try to predict, and hence reward the agent for, parts of the state that are directly under the control of the agent. This avoids the 'white-noise' problem in curiosity based methods, in which the agent gets drawn to inconsequential randomness in the environment for the sake of novelty. They do this by constructing an inverse dynamics model for the agent’s actions, and use the features constructed by this network to do forward prediction. The prediction error of this forward dynamics model is then given to the agent.
as reward.

Most curiosity-based reinforcement learning methods follow a similar principle of error-based exploration. The error could be in model prediction (Singh et al., 2005; Stadie et al., 2015) or value (Şimşek and Barto, 2006). In either case, it is provided to the agent as reward which then learns to seek this error. As it encounters the same states or transitions, this error decreases as the agent learns to accurately predict its world. It has to continue to seek out novel situations to receive high rewards.

Bellemare et al. (2016) develop an alternative approach to intrinsic motivation based on the idea of count-based exploration. In tabular settings, explicit state-visitation counts can be maintained and the agent rewarded for entering less visited states. Hence, the agent seeks novelty in terms of unseen states. In large state spaces though, direct count is not very helpful as very few states are ever visited twice. Instead, some form of generalization is needed. Bellemare et al. (2016) develop a sequential density model to estimate pseudo-counts. Pseudo-counts can be seen as an analogy of the count based approaches in a large state space where data distribution is estimated empirically. Such curiosity-based approaches are successful in allowing the agent to discover distinct parts of the state space even in high-dimensional environments. Bellemare et al. (2016) show that their agent is able to quickly explore a significant portion of the first level of the infamously difficult Atari game of Montezuma’s Revenge.

Whether it is error-based or count-based exploration, curiosity based methods simply reward the agent for seeking out novelty. They do not associate states or trajectories with underlying latent skills. As such, simple curiosity based methods do not provide a structured way to explore the task space using latent skills as mutual information based methods do. Mutual information methods, hence, tend to perform better in discovering diverse skills that are useful for downstream tasks (Houthooft et al., 2016; Warde-Farley et al., 2018).

In the next section, I discuss a prior work in learning skills using mutual information maximization between latent skill vector and trajectories. Our method, described in detail in section 5.5, generalizes past approaches by tying the latent skills to goals sampled from the downstream task distribution.
5.4 DIAYN

Diversity is all you need (DIAYN) (Eysenbach et al., 2018b), introduces an algorithm to learn skills by maximizing the mutual information between skills and states. Simultaneously, the information between the skills and the actions is minimized and the entropy of the policy is maximized. The reasoning is that skills should be determined by the states that they visit and not by the actions taken as multiple skills can share the same actions. The entropy of the policy is maximized to encourage exploration where the returns are not high. The overall objective to be maximized can be expressed as,

\[
J(\theta) = I(S; Z) + H(A|S) - I(A; Z|S)
\]

\[
= (H(Z) - H(Z|S)) + H(A|S) - (H(A|S) - H(A|S, Z))
\]

\[
= H(Z) - H(Z|S) + H(A|S, Z). \tag{14}
\]

Where \( I \) is the mutual information between two quantities and \( H \) is the entropy of a distribution. \( S \) is a random state from a skill trajectory and \( Z \) is an abstract skill vector.

Equation 14 shows the final objective used in DIAYN to learn skill policies. The first term encourages the uncertainty of the prior distribution over skills to be high. During training, the skills are sampled uniformly at random, hence this quantity is fixed and already maximized,

\[
H(Z) = -\sum_{z \in Z} p(z) \log p(z)
\]

\[
= -\sum_{z \in Z} \frac{1}{|Z|} \log(1/|Z|)
\]

\[
= \log |Z|.
\]

The second term encourages a low entropy over the skill distribution conditioned on the state. This means that the skills should be easily distinguishable from the states that they visit. Computing this quantity would require integrating, or sampling, \( p(z|s) \) over the entire skill-state space. Since this is intractable, Eysenbach et al. (2018b) learn an approximate discriminator \( q_\phi(z|s) \) using environment rollouts of agent behavior.

\[
-H(Z|S) = E_{z \sim p(z), s \sim \pi(z)} [\log p(z|s)]
\]

\[
\geq E_{z \sim p(z), s \sim \pi(z)} [\log q_\phi(z|s)] \tag{15}
\]
Where (15) comes from Jensen’s inequality (see (Agakov, 2004) for details).

The final term encourages a high entropy of the policy conditioned on the state and skill. Most modern model-free reinforcement learning algorithms, such as PPO (Schulman et al., 2017) or DDPG (Lillicrap et al., 2015), already do this by penalizing the negative entropy of the policy while calculating the policy gradient. Eysenbach et al. (2018b) achieve this by using the off-policy algorithm SAC (Haarnoja et al., 2018) which incorporates the entropy term directly in its value function.

The final objective maximized by DIAYN is,

$$\max_{\theta} J(\theta) \geq H(A|S,Z) + \mathbb{E}_{z \sim p(z), s \sim \pi(z)} [\log q_\phi(z|s) + \log|Z|]$$

The expectation term cannot be directly optimized as a function of policy parameters $\theta$ as the state distribution depends upon the environment dynamics. But, the expectation can be computed for samples which are obtained as the agent explores the environment. This is provided as a reward to the agent at each step.

$$r_z(s,a) = \log q_\phi(z|s) + \log|Z|.$$  

As the agent explores its environment, it learns to associate states with certain skills and only visit them when the skill is active. This behavior results in higher accuracy of the discriminator $q_\phi$, i.e. greater rewards.

Even though $\log |Z|$ is a fixed quantity, Eysenbach et al. (2018b) claim that its inclusion helps in learning as it provides a positive bias encouraging the agent to explore in the initial phases of training.

The training procedure is as follows. A skill is sampled at random from $z_i \sim p(z)$ and the policy $\pi(a|s,z_i)$ is followed till the termination of the episode, typically after a fixed number of steps, $T$. The agent is rewarded for visiting states that lead to easy classification of $z_i$ according to $q_\phi(z|s)$. Meanwhile, data for training $q_\phi$ is collected as state-skill tuples $(s_t, z_i)$.

$q_\phi$ is trained to minimize,

$$\mathcal{L} = \frac{1}{T} \sum_t -\log q_\phi(z_i|s_t),$$
which is the cross-entropy loss between $p(z|s)$ and $q_\phi(z|s)$.

DIAYN learns a diverse set of skills that tend to maximize coverage of the state space. Eysenbach et al. (2018b) show that in a gridworld, DIAYN evenly partitions the state space between the skills. Each skill has a uniform stationary state visitation distribution over its partition.

5.5 Task Generalized Skills

The utility of learning skills is in faster training on new tasks. The purpose of allowing the agent to explore its environment without a specific task or extrinsic reward to optimize is to enable it to learn how to manipulate its environment such that future tasks are easier to solve. I claim that for this purpose, it is not necessary to maximally explore the state space. In fact, learning skills that spread out over the entire state space may be inefficient from the lens of learning policies for future tasks.

Meaningful tasks in an environment often seem to involve pockets of states within the state space. Consider the example of manipulating objects using a robotic arm. There are a certain set of states that can be associated with tasks within this environment. These could include grasping, pushing/pulling or otherwise maneuvering the objects. It could also include the arm positioning itself near an object or a desired object configuration. But the state space includes many states are not involved in the solution of the task at all and moreover can be dangerous or harmful.

Examples of these undesirable parts of the state space include moving the arm about in the air without purpose, dropping objects on the floor, getting too close to humans in the vicinity of the arm, etc. A purely unsupervised learning algorithm may discover each of these parts of the state space and learn skills leading towards them. This could happen if the agent is solely optimizing an objective tied to state reachability, discriminability, affordance etc. Many random trajectories of the arm in mid-air can be easy to distinguish from each other by a trained neural network and hence DIAYN will discover these trajectories as skills. I claim that skills are useful if they help solve multiple tasks downstream in an efficient manner.

Consider a distribution of tasks that are deemed useful in an environment, $p(\tau)$. For the purposes of this thesis, I assume that this distribution exists for every environment and that individual tasks can be sampled from it, even if its closed form solution may be unknown. This distribution can be
multi-modal, but is typically non-uniform over the state space.

The overall contribution of this section of the thesis is to create a framework for learning skills that are optimal for solving downstream tasks sampled from $p(\tau)$. Optimality here refers to requiring the least amount of environment samples (including rewards) to learn how to solve a task $\tau \sim p(\tau)$ once all the skills are trained. The skills in $Z$ are trained using intrinsic rewards only.

We consider tasks that terminate in a goal state. That is, the form of $p(\tau)$ is $p(g)$ where $g \in S$ is a goal state.

To learn skills that maximize the efficiency of learning new tasks, we maximize the mutual information between the distributions of the goal states and the skills,

$$I(G; Z) = H(G) - H(G|Z).$$  \hspace{1cm} (16)

Here $G$ is a sampled goal state and $Z$ is the abstract skill vector. The first term corresponds to maximizing the entropy over goals. Intuitively, this means that we want to learn skills that maximally cover the task space. The second term corresponds to minimizing the goal entropy conditioned on the skill. This means the skills should lead to distinguishable goals. Combined, optimizing these two terms ensures that the skills do not all lead to the same goal and that a particular skill does not try to achieve all goals. These were the two desirable qualities of skills listed in section 5.2

5.6 DIAYN as a Special Case of Task Generalized Skills

It can be shown that skills learned by DIAYN form a special case of the task generalized skills framework.

We can rewrite this metric in equation 16 in another way,

$$I(G; Z) = H(Z) - H(Z|G)$$

$$= H(Z) + \mathbb{E}_{z \sim p(z), g \sim p(g)} \left[ \log p(z|g) \right]$$

The first term here is the same as in the DIAYN objective in equation 14. The second term involves an expectation over the goal distribution $p(g)$ and the discriminator accuracy term $\log p(z|g)$ similar to DIAYN.
In this equation, if we assume that goals are sampled according to the stationary state distribution induced by the skill policy, i.e. \( p(g) \equiv \rho_{\pi(z)} \), we recover the DIAYN objective in expectation.

\[
I(G; Z) = H(Z) + \mathbb{E}_{z \sim p(z), g \sim \pi(z)}[\log p(z|g)] = I(G; S)
\]

Therefore, the task generalized skills framework and DIAYN have the same objective if the goals are sampled uniformly from the stationary state distribution of the skill policy at every step. In Appendix B of their work, Eysenbach et al. (2018b) show that skills learned in a gridworld by DIAYN evenly partition the state space amongst themselves, with each skill forming a uniform stationary distribution over its partition.

Since skills are sampled uniformly and the stationary state distributions of skills partition the state space uniformly, sampling goals from the stationary state distribution of each skill is equivalent to sampling goals uniformly over the state space in a gridworld.

Hence, DIAYN converges to the same skill policies as task generalized skill learning if goals are sampled uniformly across the state space.

5.7 Non-uniform Goals

I will now show how interesting behaviors can be recovered if we assume non-uniform distributions of goals over the state space.

We focus on maximizing \( I(G; Z) = H(G) - H(G|Z) \). To do this, we need a way to maximize the entropy over reaching all goals and minimize the entropy of goals reached by a specific skill.

\[
I(G, Z) = H(G) - H(G|Z)
\]

\[
= - \sum_g p(g) \log p(g) + \mathbb{E}_{z \sim p(z)}[\sum_g p(g|z) \log p(g|z)]
\]

(17)

Note that we are overloading terminology here slightly. \( p(g) \) in this context corresponds to the probability a goal \( g \) is achieved by the skill currently active. \( p(g) \) can be written as,

\[
p(g) = \sum_z p(g, z) = \sum_z p(z)p(g|z).
\]
\( H(g) \) then becomes,

\[
H(G) = - \sum_g \sum_z p(z)p(g|z) \log \sum_z p(z)p(g|z)
\]

\[
= - \sum_g p(z) \sum_z p(g|z)(\log p(z) \sum_z p(g|z))
\quad \text{since } p(z) \text{ is constant}
\]

\[
= - \sum_g p(z) \sum_z p(g|z)(\log p(z) + \log \sum_z p(g|z))
\]

\[
= - \sum_g p(z) \sum_z p(g|z)(k + \log \sum_z p(g|z))
\quad \text{assuming } \log p(z) = k
\]

\[
= - \sum_z p(g|z) (k + \log \sum_z p(g|z))
\]

\[
= - \mathbb{E}_{z \sim p(z)} \left[ \sum_g p(g|z)(\log \sum_z p(g|z) + k) \right]
\]

Where \( k \) is equal to \( \log p(z) \) which is constant since our skills are sampled uniformly at the start of each episode.

Plugging this back into equation 17 we get,

\[
I(G, Z) = \mathbb{E}_{z \sim p(z)} \left[ \sum_g p(g|z) \log p(g|z) - \sum_g p(g|z)(\log \sum_z p(g|z) + k) \right]
\]

\[
= \mathbb{E}_{z \sim p(z), g \sim p(g|z)} \left[ \log p(g|z) - \log \sum_z p(g|z) - k \right]
\]

\[
= \mathbb{E}_{z \sim p(z), g \sim p(g|z)} \left[ \log \frac{p(g|z)}{\sum_z p(g|z)} - k \right] \quad (18)
\]

To calculate this quantity, we need to estimate \( p(g|z) \) for all \( z \) and \( g \). If \( p(g|z) \) is known in its closed form, this can be done easily. But typically it will be unknown and complex due to unpredictable environment dynamics.

In general, any generative model that is capable of learning this distribution from samples collected from the environment can be applied with the task generalized skills objective. In the next section, I will outline a practical algorithm that does so. I will show how we can build a generative model to estimate \( p(g|z) \) using samples of goals from the underlying task distribution.

### 5.8 Estimating \( I(G; Z) \) from Samples

Now, I will describe a practical algorithm to estimate \( I(G; Z) \) and outline a procedure to maximize it. In the absence of closed form solutions to \( p(g|z) \), we estimate it using samples collected from
the environment. One approach would be to build a generative model over goals for each skill. A challenge with this approach is that it would require a lot of training data in terms of goals corresponding to each skill.

We can get around this problem by assuming that we have the ability to draw samples of goals from \( g_i \sim p(g) \), the prior distribution over goals in the environment. We then cluster the states visited by each skill. Any probabilistic clustering algorithm can be applied here. We use Multivariate Gaussians for each skill, creating a mixture of Gaussians distribution over the state space. Each Gaussian describes the state distribution of the skill.

\[
\rho_{\pi(z)} \sim \mathcal{N}(\mu_z, \Sigma_z)
\]

Since goals are drawn from the state space, we can now estimate \( p(g_i|z) \) for each goal \( g_i \) from the Gaussian for that skill, \( \mathcal{N}(\mu_z, \Sigma_z) \). This gives the probability that the goal is within the distribution over states that the skill visits. We have now all the necessary components to calculate \( I(G; Z) \) using equation 18.

### 5.9 Learning Task Generalized Skills

As the skills evolve, the corresponding state distributions keep changing and hence so will the Gaussian clusters and \( p(g|z) \). We would like to incentivize the agent to achieve stationary state distributions that maximize \( I(G; Z) \).

Following is an algorithm to do so. At the start of each episode, a skill \( z_t \) is sampled uniformly at random and a goal \( g_t \) is sampled from \( p_t(g|z_t) \). The agent completes a rollout in the environment for a fixed number of steps according to \( \pi(z) \), receiving a stream of states associated with that skill. Once the episode is complete, the cluster for that particular skill is updated to \( p_{t+1}(g|z_t) \) by recalculating the cluster means and covariance on the new trajectory. The remaining clusters remain unchanged.

\[
r_t = \log \left( \frac{p_{t+1}(g_t|z_t)}{\sum_z p_{t+1}(g_t|z)} \right) - k
\]

is given as reward to the agent in retrospect for each step in the episode. Intuitively, if the most recent trajectory moved the state distribution of the skill closer to the goal \( g_t \) as compared to the rest of the goals, it is highly rewarded. If incorporating the trajectory into the state distribution of the skill reduced the probability of achieving the goal \( g_t \), the skill gets lower reward. Please refer to algorithm 2 for further details.
Where do the goals that form the distribution \( p(g|z) \) come from? They are not sampled from the skill’s state distribution \( \rho(\pi(z)) \) as that would lead to skills that approximate those generated by DIAYN. Additionally, the skill will attain states that may or may not be meaningful goals in the environment as discussed earlier.

For learning task generalized skills, we assume that a number of goals can be sampled from \( p(g) \), the underlying distribution of meaningful tasks in the environment. This distribution can be specified to be a part of the environment, from which a goal can be sampled for a fixed duration, for example, at the start of each episode. Or there can be a fixed number of meaningful goal states manually specified by the environment designer or by an expert. In other words, there needs to be some way to specify which tasks are meaningful to the agent so it biases the learned skills to those parts of the state space. The agent learns to explore the environment with the tasks in mind, discovering skills that lead to these meaningful goal states.

### 5.10 Combining Diversity and Task Generalization

Along with maximizing the mutual information between goals and skills, we would like the policy to remain as random as possible to encourage exploration. It is also helpful to include the DIAYN objective during training with non-uniformly distributed goals. This ensures that the skills resist collapsing into each other in areas of low task distribution support. In other words, skills would be encouraged to find unique ways of exploring the task space. Hence, the final objective to optimize is,

\[
J(\theta) = F(I(G; Z), I(S; Z)) + H(A|S, Z),
\]

where \( F \) is some function that combines the two information metrics together.

The specific form of \( F \) that was empirically found to work the best for our domains is multiplication. Therefore, the diversity and task generalized rewards are multiplied at each time step and then provided to the agent. The final reward the agent then receives is,

\[
r_t = \left( \log \frac{p_{t+1}(g_t|z_t)}{\sum_z p_{t+1}(g_t|z)} - k \right) \left( \log q_\phi(z|s) - k \right)
\]
Algorithm 2 Learning task generalized skills

1: **Given:** A set of goal samples \( g_t \sim p(g) \).
2: **Initialize:** A generative model over goal samples \( p(g|z) \) \( \triangleright \) eg. Mixture of Gaussians.
3: while \( t < t_{\text{max}} \) do
4: Sample a skill \( z_t \sim p(z) \). \( \triangleright \) Uniform
5: Sample a goal \( g_t \sim p_t(g|z_t) \).
6: while not done do
7: \( a \sim \pi_{z_t}(a|s) \).
8: Execute \( a \) in the environment and observe state \( s \) and done.
9: Compute DIAYN intrinsic reward \( r \) as \( \log q_{\phi}(z|s) - k \).
10: Store \( \langle s, r \rangle \) in trajectory \( Traj_t \).
11: end while
12: Recompute \( p_{t+1}(g_t|z_t) \) using \( Traj_t \).
13: Modify reward in \( Traj_t \) as \( r \times \frac{p_{t+1}(g_t|z_t)}{\sum_z p_{t+1}(g_t|z_t)} \).
14: Use REINFORCE to update \( \pi_{z_t} \). \( \triangleright \) or any online RL algorithm
15: end while

A central idea of this work is that the main benefit of learning skills is to speed up learning for new tasks. By biasing skills towards goals in the ways described above, we improve efficiency of learning on new tasks sampled from the same underlying task distribution. An assumption of this thesis is that samples of meaningful tasks can be drawn from this underlying distribution, such as through a human designer or an expert.

This is not such an unnatural assumption. Consider again the case of a robot arm learning to manipulate objects on a table in front of it. It is reasonable to assume that the agent can obtain a few goal snapshots, or states in which a desirable goal has been attained. For example, it could be a state with a successful grasping of a hammer, or successful pouring of water from one cup to another. These are illustrative examples of what meaningful goal states would look like in this environment. They are contrasted against frivolous states that nevertheless form unique trajectories. This could include knocking off the cup full of water from the table or even randomly moving the arm about away from any relevant objects.

As humans, we intuitively know which tasks are meaningful vs. not. For unsupervised RL agents, this knowledge has to be injected in some form. Traditionally in reinforcement learning, it has been injected in the form of predefined reward functions that optimize for a single or handful of tasks at a time. These policies are specialized to singular tasks and can often be brittle to small variations in the tasks or their contexts. This thesis advocates for reward-free exploration of the
environment in the presence of examples of goal states to create diverse skills that can be useful for an infinite number of downstream tasks drawn from the same distribution.

To empirically demonstrate this idea, I will contrast the types of skills learned by an agent in the presence of sparse goals against those learned independent of them. Following that, I will show results on generalization to unseen tasks drawn from the same distribution.

5.11 Boxworld

The environment used to evaluate the effectiveness of task-generalized skills is a 2D continuous environment called Boxworld. The states are the $x$ and $y$ locations of the agent, restricted to $[-1,1]$ in both axes. The actions are in $\mathbb{R}^2$, restricted to $[-0.1,0.1]$. The effect of an action is to deterministically displace the agent by the specified amount. An episode ends after 25 steps.

5.12 Results

First, I will show visualizations of skill trajectories learned by DIAYN on boxworld for different random seeds and then compare them to trajectories learned by our method for different types of non-uniform goal distributions. For these first set of results, the number of skills is fixed to 6 for both methods. Results for different number of skills will be shown later.

As seen in left column of figure 19, DIAYN learns skills that maximally spread out over the state space. This is because these configurations tend to minimize the discrimination loss between the skills, creating maximally distinct trajectories.

In contrast, figure 19 (right column) shows the skills learned by our method on the same three random seed when trained with four point goals shown on the figure. All goals are on the line $y = 0.8$. All skills learned by our method are limited to the top half of the box. The learned skills are distinct and spread out over the four goals. Some of the skills even interpolate between the goals (such as skill 4 in the leftmost figure). But none of the skills are wasted in covering parts of the state space that do not contain any goals as is the case with DIAYN.

In figure 20, we can see the training loss for the discriminator in DIAYN steadily goes down with training examples. As training progresses, the skills spread themselves out over the state space following the gradient induced by the rewards from the discriminator.

These results empirically demonstrate that our method can learn skills that explore the state
Figure 19: (Left Column) Visualizations of skills learned by DIAYN on the boxworld over 3 random seeds. (Right Column) Visualization of skills learned by our method with 4 point goals on the straight line $y = 0.8$ on the same seeds.

space biased by specified goals specified. The skills learned are distinct and approach the goals in different ways, not collapsing into a single policy. But they also learn to achieve all the specified
goals, not simply evenly exploring the state space evenly.

In figure 21, I show some qualitative results on quantities that we set out to maximize including the entropy over the goals and the mutual information in equation 17.

Figure 21 shows that as training progresses, the overall $H(G)$ increases and $H(G|Z)$ decreases. $H(G|Z)$ decreases during training quite a lot more significantly so for some skills than others. These skills are more concentrated around one sampled goal and the others skills may be interpolating between the goals. The overall information $I(G;Z)$ increases as training progresses.

There is a sharp drop in $H(G)$ and consequently in $I(G;Z)$ between 100k-150k environment steps. This is because one of the skills (yellow) gets highly concentrated on one of the sampled goals. This leads to a sharp drop in $H(g)$ because the probability of achieving any other goal using yellow becomes highly unlikely. As can be seen in the plot of $H(G|Z)$, the agent soon corrects that by increasing the entropy of the yellow skill slightly until $H(g)$ recovers.

Along with $I(G;Z)$ the DIAYN reward, $I(S;Z)$ is also important in our framework to create diverse skills that achieve goals. In figure 22 I show the evolution of the DIAYN metric as training progresses and the total intrinsic reward generated by the agent.

The results above demonstrate that our method biases skill learning towards goals if they are specified as points in the state space. Next, I will show visualizations of the skills learned through our method for different distributions of tasks in boxworld, such as along a straight line and a set
Figure 21: The evolution of three quantities of interest over the training phase. On the top left is the entropy over the goals. High $H(G)$ means that all the goals are being attended to by some skills. On the top right is $H(G|Z)$, or entropy over the goals conditioned on a skill. It is colored according to the skill which it is measured for. Finally on the bottom is the total metric that we set out to maximize, $I(G;Z)$, the information between the skills and the goals.

of Gaussian distributions.

Figure 23 shows visualizations of the skills learned with goals sampled randomly at uniform from the straight line $y = 0.8$. In the three figures from left to right, 2, 4, and 8 goals are sampled respectively at the start of training and held fixed henceforth. The information metric $I(G;Z)$ is calculated based on these goals only and not with any other point on the line. In other words, the agent does not know the explicit form of the goal distribution, it just receives samples from it. The specific goals sampled for this experiment are in table 2

In figure 23, we see that as the number of goals sampled increases, the skills spread out over the line. This is expected as our method relies on the sampled goals to estimate $H(G)$. In the leftmost figure, only two goals are sampled and the skills are seen to cluster around them with little
Figure 22: On the left is the skill diversity metric as defined in DIAYN. This forms a part of our reward function and can be seen to be maximized during training. On the right is the overall episodic intrinsic reward attained by the agent during training which is a combination of the DIAYN diversity reward and our task generalization reward.

<table>
<thead>
<tr>
<th>Case</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(0.39, 0.8), (-0.43, 0.8)</td>
</tr>
<tr>
<td>4</td>
<td>(0.39, 0.8), (-0.43, 0.8), (-0.55, 0.8), (0.10, 0.8)</td>
</tr>
<tr>
<td>8</td>
<td>(0.39, 0.8), (-0.43, 0.8), (-0.55, 0.8), (0.10, 0.8)</td>
</tr>
<tr>
<td></td>
<td>(0.44, 0.8), (-0.15, 0.8), (0.96, 0.8), (0.37, 0.8)</td>
</tr>
</tbody>
</table>

Table 2: Goals sampled along the line $y = 0.8$ for each case shown in figure 23.

to no interpolation. In this case, we expect lower generalization to unseen tasks sampled from the line $y = 0.8$. As the number of goals sampled increases, the generalization is expected to improve. Qualitatively, this can be seen in the other two cases in figure 23 where the skills appear to spread out over the line more.

Next, we visualize skills learned for goals sampled from a Gaussian distribution centered on $(0.7, 0.7)$ with a standard deviation of 0.2. The results for that are shown in figure 24.

The skills are again biased towards the goal distribution. The skills become better at approximating the support of the Gaussian and become more distinct from each other as the number of sampled goals increases.

Figure 25 shows visualizations of learned skills for a fixed number of goal samples (16) drawn from variations of Gaussian distributions. The first one is a wider Gaussian with standard deviation of 0.4. The second one is a mixture of two smaller Gaussians. The skills spread out to cover both Gaussians and interpolate to some degree between them. The final one shows a mixture with
Figure 23: Skills learned for different number of sampled goals along the line $y = 0.8$. From left to right, the number of goals sampled are 2, 4, and 8, shown by the Xs on the figures.

Figure 24: Skills learned for different number of goals (4, 8 and 16) sampled from the Gaussian centered at $(0.7, 0.7)$ with a standard deviation of 0.2. The darker circle shows one standard deviation and the light circle shows two standard deviations of the Gaussian.

components on diagonally opposite ends of the state space. With six skills, our method struggles to cover both components, with all of the skills focusing on one of the modes. By increasing the number of skills to 12, we can see that both Gaussians are covered. The skills do not learn to be very distinct from each other.

To further study the effect of number of skills on learned skill policies, we show visualizations for learning 2, 8, and 16 skills using our method for a Gaussian goal distribution in figure 26.

The gains in efficiency are realized only when the skills learned are generalized to other tasks that are drawn from a task distribution. I will now show quantitative results on how well the skills generalize to unseen tasks within the same distribution without any further training.

Below we show results of zero-shot generalization to downstream tasks for different number of
skills. New tasks are drawn independently from the task distribution, and hence are unseen during training time. Generalization is tested on 16 new goals sampled in this manner over 3 different random seeds used for training. The metric used to measure generalization is average extrinsic reward achieved associated with the goal during the episode. The extrinsic reward, which is solely used for testing and wasn’t included in the training in any way is defined as follows:

\[ r_{ext} = \exp(-\|s - g\|) \]

Figure 27 shows the average episodic reward on testing goals for the task distribution of an axis-aligned Gaussian centered at \((0, 0.8)\) with a standard deviation of 0.2. Figure 28 shows the average episodic reward on testing goals for a task distribution along the straight line at \(y = 0.9\). In both cases, our method outperforms DIAYN at zero-shot generalization for all number of skills, with DIAYN approaching our method at higher skill numbers. At higher number of skills, evenly partitioning the state space amongst the skills results in coverage of the task space. But most of
Figure 27: Average episodic reward achieved over testing runs on unseen goals for our method compared against DIAYN. The goals are drawn from a Gaussian task distribution. The results are averaged of 3 random seeds and the standard deviation across them is shown. This measures zero-shot generalization capability of both methods.

Figure 28: Average episodic reward achieved over testing runs on unseen goals for our method compared against DIAYN. The goals are drawn from a straight line task distribution. The results are averaged of 3 random seeds and the standard deviation across them is shown. This measures zero-shot generalization capability of both methods. 

the skills are directed at parts of the state space without any goals. Our method is able to make efficient use of a small number of skills to attain a higher reward without any further training on new tasks. 

How many goals does it take for task-generalized goals to start learning better zero-shot policies than a goal agnostic method such as DIAYN? In figures 29 and 30, I show the results on varying
Figure 29: Average episodic reward achieved over testing runs on unseen goals for our method compared against DIAYN. The x-axis shows the number of goals sampled for training task-generalized skills. The goals are drawn from a Gaussian task distribution.

Figure 30: Average episodic reward achieved over testing runs on unseen goals for our method compared against DIAYN. The x-axis shows the number of goals sampled for training task-generalized skills. The goals are drawn from a straight line task distribution.

the number of sampled goals for training task generalized skills. DIAYN is shown as a constant line as its skills do not depend on the goals. Both figures show that task-generalized skills surpass the performance of DIAYN for a handful of sampled goals. On lower number of goal samples, the skills focus themselves around the goal and do not learn to spread out over the entire task distribution. Given enough support, this is not an issue.

The above results show that we are able to improve efficiency of reinforcement learning using task generalized skill learning.
VI CONCLUSIONS

The central contribution of this thesis is to introduce task-generalized MDPs as a more efficient way of learning new tasks drawn from a probability distribution. Tasks that are more meaningful are more likely under this distribution. A natural question is, what does it mean to for a task to be more meaningful?

As humans, we intuitively understand this distribution of meaningful tasks in many everyday situations. We can identify and imagine such tasks, i.e. evaluate and draw samples from this distribution. For example, consider the domain of self-driving cars. A list of tasks that are meaningful in this situation include lane following, lane changes, making safe left and right turns, stopping, observing the speed limit, etc. These are tasks that we would desire our self-driving agents to learn. Depending on the country, rider preference, car capabilities, etc., the exact task may vary slightly in their details. But they share an underlying structure. In other words, they are distributed according to some stochastic pattern in the task space. A task can then be sampled from this distribution and assigned to a particular self-driving agent to complete. In this thesis, I demonstrate how agents can quickly learn to solve goal-driven tasks by creating generalized MDPs from a few samples of goals.

Moving forward, I believe more attention should be paid to learning architectures with the capability of integrating numerous training tasks and generalizing to unseen tasks during test time. This is a central capability of intelligent agents as tasks in the real world are rarely the same each time they are presented. There are always slight variations in the same task and agents should be trained to be robust to them by being able to adapt quickly. Going one step further, agents should learn skills that they can combine to solve entirely new tasks that are beyond just slight variations of training tasks. They should be able to quickly interpolate between tasks and build on top of them.

In chapters 3 - 5, I presented three techniques to modify the original MDP, or collection of MDPs, to create a task-generalized MDP that results in more efficient learning.

Chapter 3 describes an attention mechanism and short term memory architecture learned using intrinsic rewards. This architecture creates a state representation for the agent that allows it to
solve tasks within a partially observable environment. Hence, by learning a new state space, $S$, and thereby modifying the MDP, we can enable reinforcement learning.

Chapter 4 introduces a technique to hallucinate goals within past failed trajectories. A generative model is trained to create goal images at specified locations with respect to the agent. Goals are then inserted into failed trajectories retrospectively to convert them into successful ones, all the while respecting the environment dynamics. The reward function is modified to reflect hallucinated success. Combined with the idea of Hindsight Experience Replay (Andrychowicz et al., 2017a), this framework significantly speeds up reinforcement learning.

Chapter 5 describes a task-generalized MDP, where a distribution of tasks is used to generalize to new tasks. In this instance, samples of tasks are used to learn skills that mimic the task distribution as well as are distinct from one another. Skills are learned without a reward function, instead using an intrinsic reward that maximizes the mutual information between skills and tasks. Skills learned in this manner enable better zero-shot generalization to new tasks. The modified task-generalized MDP here consists of skills as the action space instead of atomic actions.

This thesis thus introduces the idea of task-generalized MDPs and demonstrates their effectiveness. There are many ways this work can be extended. There are a few key ingredients to creating task generalized MDPs, such as learned state representations, deriving reward functions through intrinsic motivation, generative models for goal states, hierarchical learning, etc., but this list is in no way exhaustive. This thesis merely prescribes a framework for reducing sample complexity in reinforcement learning by using distributions over tasks.

An exciting direction for future work is to study more deeply the conjecture of distributions of meaningful tasks. How do humans define which tasks are worthwhile vs. not? This could be explored from the sociological as well as physics standpoint. Tasks that are worthwhile seem to have solutions that display lower entropy than those that are futile. An example is grasping an object using a robotic arm vs. waving the arm around randomly in mid-air. But solutions with very low entropy, such as ones repeating the same action in all states, also do no solve any meaningful tasks. Similarly, humans are interested in all the ways a liquid can be poured into containers, but not very interested in exploring the diversity of solutions of spilling liquid on the floor.
An orthogonal approach to discovering useful task distributions is through multi-agent interaction. Multiple agents existing simultaneously in a complex environment involved in competitive or cooperative play can discover various sub-tasks of interest. An example is Baker et al. (2019), where agents gradually discover the use of various tools in the environment for being successful at hide-and-seek. The game is simultaneously cooperative and competitive as teams of agents hide and other teams seek. The hiders must learn to work cooperatively to move and lock in place boxes and walls to build shelters for themselves. The seekers must learn to overcome the hiders’ strategies by using other objects such as ramps. These can be seen as rudimentary skills solving autonomously discovered tasks.

Such environments are rich in interaction capability, but the game’s rules are relatively simple. Through multi-agent interaction, complex strategies, or skills, are discovered that are not explicitly programmed for. They are implicitly discovered without being directly incentivized for them. Jaques et al. (2019) use social influence as an intrinsic reward signal for multi-agent reinforcement learning. Agents are rewarded for having causal influence over other agents’ actions. This leads to higher cumulative reward in environments with social dilemmas that require effective coordination and communication to solve. Whether it is through tool use or social cues, agents can pressure other agents in the same environment to discover skills that solve a collection of tasks not explicitly programmed for.

A parallel thread of future work is to determine how tasks can be effectively conveyed to agents. Silver et al. (2021) argue that all components of intelligent behavior, such as social intelligence, language, perception, knowledge representation, planning, imagination, memory, motor control, etc., arise from maximizing a single reward in a complex environment. But Abel et al. (2021) show that there exist instances of tasks that cannot be expressed as a Markov reward. The reward hypothesis defines Markov rewards loosely as a single scalar that encodes all desired behavior, provided immediately in a state. It is a desirable quality as it allows the environment designer to simply specify tasks. This apparent contradiction suggests that better representations of reward functions are needed that can express non-Markovian objectives as Markovian rewards. In Chapter 5, we specify task distributions using goal states instead of rewards. Not all tasks can be expressed in this manner.
In conclusion, exploring how to mathematically define distributions of meaningful tasks and convey them as rewards for reinforcement learning will spur further development of algorithms and frameworks that learn to achieve them, an important step on the path to artificial intelligence.
Bibliography


