Structured Policy Learning: Hierarchies, Temporal Abstractions and Meta-Controllers for Deep Reinforcement Learning

A Project Report

submitted by

L ARAVIND SRINIVAS

in fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY AND MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2017

THESIS CERTIFICATE

This is to certify that the thesis titled **Structured Policy Learning: Hierarchies, Temporal Abstractions and Meta-Controllers for Deep Reinforcement Learning**, submitted by **L Aravind Srinivas**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology and Master of Technology**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Balaraman Ravindran
Research Guide
Associate Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Dr. Kaushik Mitra
Research Co-Guide
Assistant Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: May 1, 2017

ACKNOWLEDGEMENTS

I am lucky to have been mentored by Dr. Balaraman Ravindran. His enthusiasm in Reinforcement Learning and Cognitive Science is contagious and I am happy it rubbed off on me to the extent that I am motivated to pursue it further during my PhD. I also wish I can write and communicate as eloquently as him one day. His knack for finding obvious loopholes in research ideas made me change my outlook to research when I realized it is more about sticking to intuitive first principles than being lost in the math or existing work. I love his wit and geeky sense of humour and the fact that he is gracious and cool to take jokes on himself. By far, he is the coolest professor and adviser I have met so far, and probably ever will. I am forever grateful for the faith he had in me in the beginning when I was slow and not too successful in my first research project with him. His advising and teaching styles have inspired me a lot and I admire the way he excites the students and gives them a lot of freedom. He has been instrumental in me being able to build a research profile and supportive of my visiting studentship at the premier academic Deep Learning Lab in the world headed by Dr. Yoshua Bengio. My stint there has given me a lot of maturity as a researcher. I can't thank him enough for always being helpful and understanding whenever I had any problems. I will try my best to make him proud through my future research papers.

I would also like to thank my co-advisor Prof. Kaushik Mitra who has been very supportive and given me a lot of creative, administerial and technical freedom in carrying out my thesis. I am inspired by his patience and calmness all the time.

Next, I would like to thank my collaborators on the papers mentioned in this thesis. I am happy to have worked with Janarathan Rajendran on Transfer Learning. More than the publication it lead to at ICLR 2017, it is this project where I learned a lot about myself and research. It tested my abilities to design experiments, generate and test ideas scientifically, debug and refine my thoughts. He has been signficantly helpful by mentoring me through this phase, patient enough to reply to mails and have long Whatsapp calls despite timezone differences. I learned a lot from his different style of intuitive thinking. He is easily one of the very few researchers I have met who have novel ideas.

I also thank Mitesh Khapra, Prasanna Parthasarathi, Sahil Sharma and Ramnandan Krishnamurthy for their time and discussions. I learned different things from each one of them. Mitesh - the ability to decide among several idea-paths prioritizing what would lead to a publication, Sahil - knack for identifying logical loopholes, Ramnandan - systematic way of going about things, Prasanna - building intuitions from first principles.

Apart from my collaborators, I would really like to thank Sherjil Ozair and Yoshua Bengio from MILA. Sherjil is my closest research buddy till now. He has been instrumental in my transition from following papers to judging them. His passion and attention to detail were humbling and the amount of time we spent and continue to spend, discussing and critiquing ideas has made me a much better researcher than what I was before meeting him. Yoshua is an inspiring example for success with humility.

I would also like to thank my lab mates, Vinod, Varun, Priyesh, Vishvesh and Yash for making my stint at the RISE Lab an enjoyable and memorable one. Among them, I am very much indebted to Varun Gangal and Vinod Ganesan. Varun was the one who introduced me to Machine Learning towards the end of my second year when I absolutely had zero idea of what to do in life. He has always guided me on what to do, had several late-night dinners and teas and shared useful advice on many decisions I had to make. I was fortunate to have him as room neighbor in my second year and I am not even sure if I would have worked in Machine Learning had I not met him. Vinod has been one of my closest friends and I am fortunate to have met him while working in the lab several nights on my Transfer Learning project. He has been a big source of moral support and accompanied me for several dinners and lunches. His insights on intelligence from someone outside the community were thought-provoking.

Thanks to RISE Lab as a whole for making it possible to work on this thesis. Thanks to my other friends from IITM, Krishnan, Ranjan, Ajay, Santhosh, Mohana.

I am deeply thankful to my father, uncle and aunt for their help and support. This thesis is a dedication to my mother, Jayanthi. Words are not sufficient to summarize her sacrifices for me. I will continue to work hard and make her proud.

ABSTRACT

KEYWORDS: Reinforcement Learning, Deep Learning, Deep Reinforcement Learning, Meta-Learning, Hierarchical Reinforcement Learning, Spatio-Temporal Abstractions, Representation Learning

In this thesis, I discuss a combination of three different papers, which attempt to explore different aspects of a broad theme called *Structured Policy Learning for Deep Reinforcement Learning*. I present models and algorithms to learn efficiently from raw pixel inputs that handle temporal abstractions, hierarchies and meta-control respectively. For scalable agents that can learn autonomously, temporal structures in the policy space are inevitable. Thus, we need to induce different kinds of structural priors in the policy space to be able to realize such autonomous agents for complex tasks with large state spaces. I present three different ways to explore structured policy learning: i) Temporal abstractions in the policy space induced through the structure of repetitive macro-actions, ii) Segmenting the state-space of the task into weakly connected components and learning macro-actions that transition across discovered segments (or abstractions), and iii) Meta-Controller Transfer Learning framework to be able to selectively use previously learned skills while learning to solve a new task in a continual learning setup.

Contents

Chapter 1

Overview

1.1 Reinforcement Learning

Reinforcement Learning is a division of Machine Learning that deals with the study of sequential decision making problems. It considers an agent in an environment where at every time step, the agent decides and executes an action, and receives the next observation and reward for the executed action from the environment. The agent uses the relevant history of observations to decide on the next action. The decision is taken such that the agent tries to maximize the cumulative reward through a trial-and-error learning process. Chapter 2 provides a more detailed description with a mathematical formulation of Reinforcement Learning.

1.2 Deep Learning

Deep Learning is a paradigm of Machine Learning that fundamentally thrives on a simple recipe: choose a loss function, build an expressive function approximator (multiple layers of a neural network) and learn the parameters of the function approximator with gradient descent methods. The success of this approach can be seen from the paradigm shift in computer vision and speech recognition. In computer vision, convolutional neural networks are typically used for several problems like scene classification, object detection, semantic segmentation, image captioning and region proposals. This has been possible because of the rich hierarchy of features discovered for optimizing the loss function. The recipe in Deep Learning involves a reduction from a learning problem to an optimization problem: in Supervised Learning, we are reducing the problem from *learning* a function that makes good predictions on unseen data to that of *optimizing* the prediction error plus regularization on the training data using expressive non-linear functions.

However, the reduction from learning to optimization is less straightforward in Reinforcement Learning than it is in Supervised Learning. There are multiple reasons for this. Firstly, in Reinforcement Learning, it is hard to have an analytic estimate for the objective function to optimize for, without knowledge of the underlying dynamics model and the reward function. Secondly, in the case of estimating an approximate version of the same, the input data is dependent on the agent's current behavioral policy, thereby introducing the problem of non-stationarity that is not prevalent in Supervised Learning.

1.3 Deep Reinforcement Learning

The intersection of Deep Learning and Reinforcement Learning is a promising direction to explore for research directed at creating agents that can learn goal-directed behavior in complex scenarios with minimal human intervention. Reinforcement Learning (RL) is a natural framework for learning autonomously by exploring the environment, while Deep Learning (DL) provides an apt function approximation paradigm for discovering the necessary abstractions to learn from raw sensory inputs without human effort for feature engineering. Combining RL and DL allows an agent to build abstractions of the world through the the reward signals acquired while trying to solve the task at hand. This allows the agent to learn features that naturally combine perception and control from several experiential trials. A policy is the strategy that tells an RL agent which action to pick at each state in the world. The agent tries to optimize its policy at each state so as to maximize the cumulative sum of rewards obtained as a consequence of its actions. On the other hand, a value function is an evaluation of the policy that the agent adopts in terms of how good the policy is at optimizing the long-term rewards. A Deep Reinforcement Learning (DRL) agent attempts to learn the optimal policy and (or) value function for the specific task using multi-layered neural networks as function approximators for the parametrized policy and (or) value function. This is essential to scale the learning over large state-spaces where tabular approximate dynamic programming methods are infeasible for practical purposes. Thus, a DRL agent represents the value function / policy using highly non-linear transformations of the original state-space.

1.4 Contributions of this thesis

Function approximation is a solution for generalizing to similar but unseen states over large state-spaces. However, that *alone* does not solve a more important problem - How do we induce appropriate structure and learn hierarchies? There are different scales of planning actions to solve a task. For instance, a self-driving car agent aiming to reach a destination would have low-level actions concerned with acceleration, brakes, etc while the high level actions would be to deal with the traffic control, and even higher, to plan the route involved in reaching the destination. This leads us to the question of how to induce appropriate structures in the policy space that generalize across states so that the agent can execute scalable control policies at multiple hierarchies. For definition, let's call a macro-action as a block of low-level granular actions. One could extend this hierarchy by defining subsequent levels of macro-actions as being composed of the previous level of macro-actions. How do we learn the appropriate macro-actions at each level and learn control policies at the level of macro-actions to maximize the objective for a specific task? It's also necessary that the learning methodology we adopt must be compatible with function-approximation so as to scale across large state-spaces. In this thesis, I take an attempt to look at the above problem by exploring three different kinds of structures which I describe in brief detail below.

The first, is to explore the structure of repetitive macro-actions. Here, I constrain the structure of temporal abstraction to be composed of the same action but such that the time-scale (length (or) frequency) of the action could vary across the macro-actions. The agent learns a control policy composed of these macro-actions, whereby, learning to pick a macro-action becomes equivalent to choosing an action and an appropriate time scale for executing the action. This could lead to executing a control policy of the type AABBBBCCDDDDBAAA, where A,B,C,D are different *granular* actions available to the agent. AA, BBBB, CC, DDDD, B, AAA are different *macro-actions* that the agent picks. This structure, though restrictive, augments the agent with the ability to look ahead as long as repeating the same action still leads it to a favorable future state. Such an ability can help speed up the learning process by bypassing the need to *decide* on those intermediate states where the agent would be better-off repeating the on-going action. At present, Scalable Reinforcement Learning and Temporal Abstractions are still not well understood. Therefore, it is important to explore even simplistic struc-

tures as above and adopt them if they lead to more stable learning and better sample efficiency. I demonstrate the improved sample complexity and better stability in training through experiments where an agent learns from raw pixels in high dimensional state-spaces on the Atari 2600 domain. This idea lead to two conference papers - 1. *Dynamic Action Repetition for Deep Reinforcement Learning*, Association for the Advancement of Artificial Intelligence (AAAI), 2017, and 2. *Learning to Repeat: Fine Grained Action Repetition for Deep Reinforcement Learning*, International Conference on Learning Representations (ICLR) 2017. Sahil Sharma was my collaborator in both of these works. I introduced the idea of dynamic action repetitions and repetitive macro-actions in the AAAI paper with simple experimental demonstrations on expanding the action space for each repetitive extent considered, and introduced the framework for extending it to large number of time scales for a more practical algorithm. The second paper follows up on this with an implementation of the more practical version and was lead by Sahil Sharma. I will primarily be talking about the first paper in this thesis, with a reference to the second for extra reading.

Secondly, I will talk about another approach for inducing spatio-temporal structure in Reinforcement Learning through segmenting the state-space of the underlying Markov Decision Process (MDP) into weakly connected components. Each state now has an associated membership function to its macro-state (component), and a macro-action here is the sequence of actions that would lead to a change in macro-state. Such macro-actions can be acquired without *learning*, by ascending (hill-climbing) on the membership function. We adopt Perron Cluster Analysis (PCCA+), a spectral clustering algorithm grounded in Molecular Dynamics, for acquiring the clustering and membership information. Hierarchical Reinforcement Learning algorithms are then used to learn a meta-controlling policy over the available options and base actions. However, this pipeline of option discovery and control relying on MDP segmentation is infeasible for larger practical problems. This is because the option discovery part of the pipeline requires the transition matrix (weighted graph of the MDP) to be fed in as an input. It is infeasible to compute and store a huge sparse transition matrix for larger problems with exponential state spaces. In this thesis, I will discuss an attempt to scale the above pipeline using recent advances in Deep Learning. The approach adopted here is to learn a non-linear projection of the original state space to lower dimensional subspace through unsupervised learning methods using deep neural networks. The state space is

then, *aggregated* through simple clustering techniques on this lower-dimensional manifold. A transition matrix is then estimated from sampled trajectories on the *aggregated* state space. The dimensionality of the *aggregated* state space is chosen such that a transition matrix on such a state space is practical to work with. This matrix is then fed to the PCCA+ clustering algorithm to discover weakly connected components and macro-actions on the aggregated state space. This way, we acquire macro-actions that apply to the original state space and can adopt Hierarchical Value / Policy based Deep Reinforcement Learning algorithms to learn an optimal control policy over the actions and options. This modified pipeline is again not a perfect solution to the problem of generalizable hierarchical Reinforcement Learning, but was one of the few approaches that attempted to answer the question at the time of its introduction in 2016. This was joint work done with Ramnandan Krishnamurthy and Peeyush Kumar and was accepted as a workshop contribution at the Abstractions in Reinforcement Learning Workshop held at the International Conference in Machine Learning, 2016, conducted by Google DeepMind. The original PCCA+ approach on small state spaces was proposed by Peeyush Kumar. I contributed by proposing to use an Action Conditional Video Prediction Model for learning unsupervised representations from sampled trajectories, on which the state-space is clustered and aggregated for PCCA+ to work on the aggregated state space. This representation proved to be better for clustering as compared to using Deep Q Learning based representations for the pipeline (which was explored by Ramnandan). This is reasonable given that explicit spatio-temporal structure is encoded in video prediction models and hence aggregation on manifolds encoded with rich spatio-temporal information is expected to be more aligned for spectral clustering algorithms tied to the MDP topological structure like PCCA+.

Thirdly, I will discuss recent work in the space of Transfer and Continual Learning which lead to the conference paper, *Attend, Adapt and Transfer: An Attentive Deep Architecture for Adaptive Transfer from Multiple Source Tasks in the Same Domain* at the International Conference on Learning Representations (ICLR), 2017. This work attempts to explore meta-controller structure in Reinforcement Learning through the lens of transfer learning. This work attempts to exploit previously learned skills while acquiring a new skill. Using previously learned skills selectively can help speed up the learning of a new skill. At the same time, one must ensure that previously learned skills which are adversarial and unrelated to the current task are ignored. This is im-

portant because acquiring a new skill must not become harder just because of unrelated previous experience. Thus, we need a robust framework that learns how to selectively use the prior skills when acquiring a new skill (so that only the useful ones for a given situation are invoked to speed up the process of learning the new skill) and harmful or useless prior skills (from the context of the specific skill being acquired) are ignored to ensure the agent is at least as good as learning the new skill from scratch without prior experience. To this end, I discuss an attentive meta-controller framework that can help achieve this, whereby, a soft-attention based meta-controller combines the solutions of the previously learned tasks and that of the ongoing task, to execute the resultant convex combination of these solutions on the ongoing task. Based on how useful each of the solutions are, specific to the state at which the agent is, the meta-controller learns to assign the weights to those solutions accordingly. The solution being learned on the new task bootstraps knowledge from the samples collected by this aggregate solution being executed on the task. At the end of the training, the solution for the new task is sandboxed with the list of prior skills thereby making this framework suitable for a continual learning setup. The solution for a task could be the optimal policy (or) value function. My contribution to the paper (on top of Janarthanan Rajendran's work on policy transfer with simple toy tasks) was to explore transfer of value functions for more complex high dimensional tasks. Further, since the network that learns from scratch on the new task undergoes *off-policy* learning, value based methods are theoretically sound and avoid the need for importance sampling. This issue was not addressed in off-policy learning in Policy Transfer. I present the experiments I designed to show how well this framework works for Value Transfer with inputs in the space of raw pixels on a couple of Atari 2600 games, Pong and Breakout, and also discuss possibilities for future work.

Chapter 2

Reinforcement Learning Preliminaries

2.1 Markov Decision Processes

Markov Decision Processes (MDPs) are a common paradigm used to study Reinforcement Learning Problems (Sutton and Barto, 1998*a*). An MDP is a mathematical object describing an agent's interactions with the environment and is defined by:

- S - **state space** - set of states in the environment
- A - **action space** - set of actions available to the agent at every time step
- $P(s'|s,a)$ - **transition dynamics** - Given state s and action a, provides the probability distribution over possible next states s'.
- $R(s,a,s')$ - **reward function** associated with executing an action a at a state s and moving to next state s'.
- $\mu_0(s)$ - **initial state distribution**

The initial state is drawn from a stationary distribution $\mu_0(s)$ characteristic to the MDP. The goal in Reinforcement Learning is to find an optimal policy π which is a mapping from states to actions. In value based approximate dynamic programming methods, one typically uses deterministic policies $a = \pi(s)$ that is greedy with respect to the current estimate of the optimal value function, while in policy optimization methods, one searches for the ideal stochastic policy $\pi(a|s)$. The graphical model of the MDP framework is shown in Figure 2.1 with the same notations as described above.

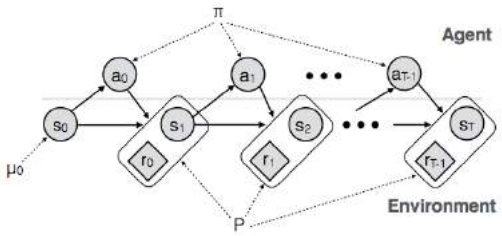


Figure 2.1: Markov Decision Process

2.2 Reinforcement Learning Agent

A Reinforcement Learning agent tries to maximize the cumulative sum of rewards received as a consequence of its actions in the environment. At every time step, the agent interacts with the environment by executing an action, receiving in return (from the environment) a reward for the executed action and an observation for the next time step. The agent executes the action based on the policy it uses to decide on the actions.

A policy $\pi : S \times A \rightarrow [0, 1]$ defines the way an RL agent behaves. The objective of an RL agent is to find a policy which maximizes its cumulative reward. The cumulative reward could have a discount factor $\gamma \in [0, 1]$ which balances the tradeoff between immediate reward and future rewards. A myopic agent would have γ close to 0 not giving too much importance for future rewards obtained as a consequence of future actions.

The policy of the agent at any state is driven by the need to maximize the sum of the reward plus the discounted value of the next state. Value based RL algorithms maintain an estimate for how valuable it is to be a state in the MDP under the given policy. The theory defines two coupled value functions for this: $V^{\pi}(s) : S \rightarrow \mathbb{R}$ which is the value function of a given state s when the agent follows the policy π in the MDP, $Q^{\pi}(s, a) : S \times A \rightarrow \mathbb{R}$ which is the action-value function for a given state s and action a when the agent follows policy π. The action value function $Q^{\pi}(s, a)$ is by definition the sum of the expected immediate reward and the value function $V^{\pi}(s')$ of the next state. To put these in mathematical form, consider the notation t for time index and state s_t represents the state in the MDP at time step t. Also assume that $r(s, a)$ is the reward for picking action a at state s, while $r(s_t)$ is the reward for reaching state s_t. The definition for $V^{\pi}(s)$ assumes that the time counting starts at s with 0.

$$V^{\pi}(s) = \mathbb{E}_{\pi}[\sum_{t=1}^{\infty} \gamma^t r(s_t)]$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[r(s, a) + \gamma V^{\pi}(s')]$$

For an RL agent, the objective is to follow a policy that is $\text{argmax}_{\pi} V^{\pi}(s)$ in the MDP. Let us define the optimal value functions below:

$$V^*(s) = \max_\pi V^\pi(s)$$

$$Q^*(s,a) = \max_\pi Q^\pi(s,a)$$

Value based RL methods try to converge to V^* or Q^* from random initial estimates through approximate value iteration methods in expectation. The agent behaves greedily with respect to the current value function to improve the current policy, and the new behavior is evaluated to modify the value function. Repeated cycles of the policy improvement and evaluation leads to convergence (with guarantees for certain cases) (Sutton and Barto, 1998*b*). A policy that is greedy with respect to current value function can be defined comfortably using the Q function as: $\pi^{i+1}(s) = \text{a}Q^i(s,a)$ where i denotes the iteration index. Once the value based methods converge to the optimal value function Q^*, a policy that is greedy with respect to Q^* is by definition the optimal value function.

To drive the learning from initial random estimates of the optimal value function to correct ones, value based methods exploit the Bellman Equation (Sutton and Barto, 1998*b*). The state value function V^π and the action value function Q^π obey:

$$V^\pi(s) = \mathbb{E}_{s',r\sim P(s',r|s,a)\, a\sim\pi}[r + \gamma V^\pi(s')]$$

$$Q^\pi(s,a) = \mathbb{E}_{s',r\sim P(s',r|s,a)}[r + \gamma \mathbb{E}_{a'\sim\pi}[Q^\pi(s',a')]]$$

When π is a greedy policy π^* that is optimal, the inner expectation in the Bellman equation for Q^π goes away and is replaced by $\max_{a'} Q^{\pi^*}(s',a')$. Since π^* is greedy and has no explicit policy representation and is meant to be the optimal policy, we shall just denote the action value function as Q^*. This gives us the Bellman Optimality Equation:

$$Q^*(s,a) = \mathbb{E}_{s',r\sim P(s',r|s,a)}[r + \gamma \text{max}_a Q^*(s',a')]$$

Repeatedly enforcing reduction of the Bellman Error between $Q^*(s,a)$ and $r + \gamma Q^*(s', \text{argmax}_{a'})$ converges to Q^* in expectation. The s,a,r,s' for learning Q^* in this way can be simulated from any policy π since the outer expectation is over the MDP stochasticity and agnostic to the policy π that simulated $s,a,r,s')$. This type of learning

is called *off-policy* learning in RL and applying it specifically in the above manner for learning the optimal action value function Q^* gives the algorithm called Q-Learning.

Reduction of the Bellman Error to get a better estimate of the value function is the evaluation step. The policy improvement step comes from being greedy with respect to the current estimate of the value function. Repeated iterations of better estimates and greedy policies with respect to the current estimates drive Q-Learning. One caveat in this is the exploration problem. When the current estimates are bad, being greedy with respect to the current estimates will simply lead to repeated execution of bad behavior leading the agent to unfavorable parts of the state space. Typically, instead of greedy, $\epsilon-$greedy methods are adopted so that the agent executes random actions with ϵ probability and behaves greedily with $1 - \epsilon$ probability. ϵ is typically annealed from a high value to a low value like 0.05 over the course of the agent executing Q-Learning induced control policies.

My contributions for this thesis involved value learning methods majorly. However, I will introduce policy gradients and Actor-Critic briefly in the respective chapters and cite the relevant papers for further details. This chapter is not a comprehensive introduction to Reinforcement Learning and one should refer to (Sutton and Barto, 1998*b*) for an excellent and concise introduction to Reinforcement Learning.

As for deep neural networks (Deep Learning) in RL, it is infeasible to cover background material without delving into Deep Learning itself. (Goodfellow *et al.*, 2016) provide an excellent and exhaustive introduction to Deep Learning while for Deep RL, a brief survey of existing literature is provided by (Li, 2017).

Chapter 3

Dynamic Action Repetition for Deep Reinforcement Learning

3.1 Abstract

One of the long standing goals of Artificial Intelligence (AI) is to build cognitive agents which can perform complex tasks from raw sensory inputs without explicit supervision (Lake *et al.*, 2016). Recent progress in combining Reinforcement Learning objective functions and Deep Learning architectures has achieved promising results for such tasks. An important aspect of such sequential decision making problems, which has largely been neglected, is for the agent to decide on the duration of time for which to commit to actions. Such *action repetition* is important for computational efficiency, which is necessary for the agent to respond in real-time to events (in applications such as self-driving cars). Action Repetition arises naturally in real life as well as simulated environments. The time scale of executing an action enables an agent (both humans and AI) to decide the granularity of control during task execution. Current state of the art Deep Reinforcement Learning models, whether they are off-policy (Mnih *et al.*, 2015*a*; Wang *et al.*, 2015) or on-policy (Mnih *et al.*, 2016*b*), consist of a framework with a static action repetition paradigm, wherein the action decided by the agent is repeated for a fixed number of time steps regardless of the contextual state while executing the task. In this chapter, we propose a new framework - Dynamic Action Repetition which changes Action Repetition Rate (the time scale of repeating an action) from a hyper-parameter of an algorithm to a dynamically *learnable* quantity. At every decision-making step, our models allow the agent to commit to an action and the time scale of executing the action. We show empirically that such a dynamic time scale mechanism improves the performance on relatively harder games in the Atari 2600 domain, independent of the underlying Deep Reinforcement Learning algorithm used.

3.2 Introduction

There has been a growing interest both in creating AI agents which can play computer games well (Mnih *et al.*, 2015*a*; Hausknecht and Stone, 2016; Mnih *et al.*, 2016*b*), as well as creating Human-like AI agents against which human opponents can enjoy playing (Hingston, 2010; Ortega *et al.*, 2013; Van Hoorn *et al.*, 2009). Indeed, both the problems are related. A solution to the former is necessary, but not sufficient for a solution to the later. This is because agents which can play the game well automatically lead to AI opponents against which humans might enjoy playing computer games. Hence, any advancement in the former domain leads to an advancement in the latter. We present one such advancement in Reinforcement Learning (RL)-based game playing.

Video game domains such as Mario (Togelius *et al.*, 2010), Atari 2600 (Bellemare *et al.*, 2013) and Half Field Offensive (Hausknecht *et al.*, 2016) have served as a test bed to measure performance of learning algorithms in AI-based game playing. Such games present unique challenges to an AI agent since performing well in them requires being able to understand the current state of the game (involves pattern recognition) as well as being able to choose actions after considering long term implications of choosing those actions (involves planning). Recent applications of Deep Reinforcement Learning (DRL) to these domains has resulted in state-of-the-art performance, when the policies have to be learnt directly from pixels in a model-free RL setting (Mnih *et al.*, 2015*a*, 2016*b*; Hausknecht and Stone, 2016).

Game playing AI should be real-time for it to appear plausible to humans and for it to have real world applications. This is one of the motivations behind the evolution of Deep Learning (DL)-based AI agents as a replacement for simulation based AI. However, DRL agents involve significant computation in the form of convolutional neural networks and recurrent neural networks. One way to reduce such computation is to introduce a hyper-parameter in DRL algorithms called the *action repetition rate* (ARR) which denotes the number of times an action selected by the agent is to be repeated. If the ARR is low, the decision making frequency of the agent is high. This leads to policies which evolve rapidly in space and time. On the other hand, a high ARR causes infrequent decisions, which reduces the time to train at the cost of losing fine-grained control. A higher ARR also leads to the agent learning human-like policies for game play. This is because the policies learnt with a higher ARR have fewer action sequences

which exhibit super-human reflexes as compared to an agent which plays with a low ARR ((Mnih *et al.*, 2015*a*, 2016*b*) use an ARR of 4). A reduction in decision making frequency gives the agent the advantage of not having to learn the control policy in the intermediate states given that a persistent action from the current state can lead to an advantageous next state. Such skills would be useful for the agent in games where good policies require some level of temporal abstraction. An example of this situation in Seaquest (an Atari 2600 game) is when the agent has to continually shoot at multiple enemies arriving together and at the same depth, one behind another. In the case of Space Invaders (Atari 2600 game), the lasers are not visible in every fourth frame, as noted by (Mnih *et al.*, 2015*a*). Hence, a higher action repetition could help the agent avoid the confusion of having to decide an action in such peculiar intermediate states, where the lasers are not visible.

Having said all of that, there are also situations where agents have to take actions which require quick reflexes to perform well with an example from Seaquest being states in which multiple enemies attack simultaneously and are placed at varying depths. The agent is then expected to manoeuvre skillfully to avoid or kill the enemies. Similarly, in Space Invaders the agent needs to avoid multiple lasers being shot simultaneously by enemies that are progressively moving closer to the agent. Such situations merit a finer granularity of control. Therefore, a high but static ARR is probably not a solution for better game playing strategies in spite of the temporal abstractions it provides to the agent. As a small step in the direction of temporal abstractions in the policy space (in the form of temporally elongated actions), we propose a dynamic action repetition paradigm of game-playing. The paradigm is generic enough that it can be combined with any off-the-shelf discrete action space DRL algorithm. The proposed paradigm presents a more structured policy framework, wherein the policy consists of the action probabilities (or state-action values) along with the number of times the corresponding action is to be repeated, each time a decision is to be made. This is on the lines of (Hausknecht and Stone, 2016), in which an Actor Critic setup is used in the Half Field Offense domain (Hausknecht *et al.*, 2016), to learn policies that contain both the probabilities of selecting actions as well as their associated parameters. Policies learnt under this paradigm guide the agent in deciding whether it is beneficial to exert super-human-reflexes, or not. In the case of longer temporal persistence of the chosen action, the optimal level of persistence would help the agent to get to most advantageous (in

terms of expected cumulative discounted future reward) temporally distant state.

One of the major drawbacks of the current DRL game playing algorithms (Mnih *et al.*, 2015*a*, 2016*b*; Wang *et al.*, 2015) is that the action repetition rate is a static hyper-parameter which has to be tuned using usual hyper-parameter tuning techniques. We demonstrate that this inability to decide the granularity of control deeply inhibits the kind of control policies that are learnt by the algorithms. We show the efficacy of the policies learnt using our paradigm over those learnt using the existing algorithms by empirically establishing the fact that dynamic action repetition is an important way in which an AI agent's capabilities can be extended. This is done by combining existing DRL algorithms with our paradigm. To demonstrate that this paradigm (Dynamic Action Repetition) can be combined with any Deep Reinforcement Learning algorithm, we perform experiments in off-policy (Mnih *et al.*, 2015*a*) as well as on-policy (Mnih *et al.*, 2016*b*) setting. Thereby we make the claim that there is a need to explore and experiment with structured parametrized policies for finding temporal abstractions using an Actor Critic setup in the Deep Reinforcement Learning based Game Playing domain.

3.3 Related work

One of the first efforts which pushed the state of the art significantly in the domain of game playing based on raw sensory inputs was (Mnih *et al.*, 2015*a*). Their architecture, DQN, motivated the use of convolutional neural networks for playing games in the Atari 2600 domain.

(Braylan *et al.*, 2015) focus on the power of the frame skip rate hyper-parameter (it is the hyper-parameter in ALE which directly controls the action repetition rate) with experiments in the Atari 2600 domain. Their learning framework is a variant of Enforced Sub-Populations (ESP) (Gomez and Miikkulainen, 1997), a neuroevolution approach which has been successfully trained for complex control tasks such as controlling robots and playing games. They show empirically that the frame skip parameter is an important factor deciding the performance of their agent in the Atari domain. They demonstrate for example that Seaquest achieves best performance when an ARR of 180 frames is used. This is equivalent to the agent pausing (continuing the same action) for three seconds between decisions since the ALE emulator runs at 60 frames per second.

They argue that a higher value of action repetition allows the agent to not learn action selection for the states that are skipped and hence develop associations between states that are temporally distant. However, they experiment only with a static action repetition setup. The key way in which our work differs from both the ESP and the DQN approaches is that we make action repetition a dynamic learnable parameter.

The idea of dynamic-length temporal abstractions in the policy space on Atari Domain has been explored by (Vafadost, 2013). They use a Monte Carlo Tree Search (MCTS) planner with macro-actions that are composed of the same action repeated k times, for different k. The way in which our approach differs from (Vafadost, 2013) is in terms of using a Deep Neural Network to build non-linear Q-function approximators instead of making use of search techniques that cannot generalize across unseen states. We also learn to pick the suitable repetition extent for a state through deep networks instead of searching through rollouts for the optimal repetition extent k.

The utility of repetitive *macro-actions* has been pointed out by (Ortega *et al.*, 2013) in designing high level actions to imitate human-like play in Mario such as *Walk, Run, Right Small Jump, Right High Jump* and *Avoid enemy*, which are composed of varying length repetitive key presses. For instance, *Right High Jump* involves pressing the *Jump and Right* keys together for 10 frames, while *Right Small Jump* requires the same for 2 frames.

3.4 Background

The following four sub-sections introduce standard Reinforcement learning algorithms and their deep counterparts.

3.4.1 Q-Learning algorithm

One of the approaches for solving the sequential decision making problem is to estimate the optimal value of every state-action pair - $Q(s, a)$. The optimal value of an action a in a state s is defined as the expected cumulative sum of future rewards on taking a in s and following the optimal policy thereafter. $Q(s, a)$ is a measure of the long-term reward obtained by taking action a in state s. Computing approximations for the Q-function

enables the agent to select the optimal action a in a state s. Hence, one way for the agent to learn optimal policies is to estimate $Q(s, a)$ for the given task. Q-learning (Watkins and Dayan, 1992) is an off-policy Temporal Difference (TD) learning algorithm which does exactly that. The Q-values are updated iteratively using the Bellman optimality equation (Sutton and Barto, 1998*b*) with the rewards obtained from the game as below:

$$Q_{t+1}(s, a) = \mathbb{E}[r + \gamma \text{max}_{a'} Q_t(s', a')|s, a]$$

Here t denotes the time-step during the episode. During training, the behavioral policy is ϵ- greedy with respect to $Q(s, a)$ to ensure sufficient exploration.

3.4.2 Deep Q Network (DQN)

In high dimensional state spaces, it is infeasible to compute optimal Q-values for all possible state-action pairs explicitly. One way to address this problem is to approximate $Q(s, a)$ using a parametrized function approximator $Q(s, a; \theta)$, thereby gaining the power to generalize over unseen states by operating on low level features (Sutton and Barto, 1998*b*). Recent advances in representation learning using deep neural networks (LeCun *et al.*, 2015) provide an efficient mechanism to learn hierarchical features across large state spaces and avoid feature engineering. DQN (Mnih *et al.*, 2015*a*) combines the representation learning power of deep neural networks with the Q-Learning objective to approximate the Q-value function for a given state and action. The loss function used for learning a Deep Q Network is :

$$L(\theta) = \mathbb{E}_{s,a,r,s'}[(y^{DQN} - Q(s, a; \theta))^2],$$

with

$$y^{DQN} = (r + \gamma \text{max}_{a'} Q(s', a', \theta^-))$$

Here, L represents the expected TD error corresponding to current parameter estimate θ. θ^- represents the parameters of a separate *target network*, while θ represents the parameters of the *online network*. The usage of a *target network* is to improve the stability of the learning updates. The gradient descent step is as follows:

$$\nabla_\theta L(\theta) = \mathbb{E}_{s,a,r,s'}[(y^{DQN} - Q(s, a; \theta))\nabla_\theta Q(s, a)]$$

The off-policy nature of the algorithm ensures that DQN is able to avoid correlated updates, which are likely, when learning on temporally correlated transitions that the *online network* simulates. This is facilitated through an experience replay memory (Lin, 1993) D (of fixed maximum capacity) where the state transitions and rewards experienced by the agent are pooled in a First-In-First-Out (FIFO) fashion.

3.4.3 Advantage Actor-Critic Algorithm

The Q-learning algorithm described in the previous subsections computes only the value function associated with state-action pairs. Explicit policies are not learnt by the agent during Q-Learning. Rather, the policy of the agent is induced implicitly (by picking the action with the maximum Q-value) along with low-probability exploration based on an ϵ-greedy approach (Sutton and Barto, 1998*b*). In contrast, Actor-Critic methods (Konda and Tsitsiklis, 2003) explicitly model the policy that is executed by the agent along with a value function component updated using Temporal Difference (TD) learning methods. Hence, they have two distinct components: an *actor* and a *critic*. We describe the parametric version of Actor Critic below.

The actor proposes a parametrized policy $\pi(a_t|s_t;\theta_a)$ while the critic provides the basis for improvement in the policy estimation by approximating the optimal value function $V^*(s)$ using a parametrized estimate; $V(s_t|\theta_c)$. We describe a parametric critic as a general setup applicable to high dimensional state spaces. In principle a non-parametric critic could also be learnt, using table look ups. The state value function computed by the critic is used for obtaining the Temporal Difference (TD)-error term $(r + \gamma V(s') - V(s))$ required for computing the updates to the actor parameters θ_a. The updates on the actor parameters θ_a are based on policy gradient methods with the critic term providing the sign and weight for the updates (Sutton *et al.*, 1999*a*). The ideal unbiased policy gradient for updating the parameters of the probability of execution of action a_t in state s_t is $\nabla_{\theta_a} \log \pi(a_t|s_t;\theta_a)\mathbb{E}[R_t]$. Here R_t is the return obtained from state s_t after executing action a_t. Naturally, based on how $Q(s_t, a_t)$ is defined, $\nabla_{\theta_a} \log \pi(a_t|s_t;\theta_a)Q(s_t, a_t)$ provides a biased estimate for the same but with lower variance when compared to using a point estimate of R_t as proxy for $\mathbb{E}[R_t]$. The variance of the former estimator can be improved even further without adding to the bias by using $\nabla_{\theta_a} \log \pi(a_t|s_t;\theta_a)(Q(s_t, a_t) - b(s_t))$ where $b(s_t)$ is a baseline term that's

independent of the entity on whose *decision* the gradient is computed (a_t here). The baseline could however be state-dependent. Often, the baseline used is the value function estimate of state s_t: $V(s_t|\theta_c)$.

The proxy function for the utility R_t is hence $Q(s_t, a_t) - V(s_t)$. This is defined as the Advantage Function $A(s_t, a_t)$. Since this instantiation of Actor Critic uses the Advantage Function as the proxy utility, this method is referred to as *Advantage Actor Critic*. The efficacy of this method relies on the critic approximating the optimal value function $V(s_t)$ since the Advantage Function $A(s_t, a_t)$ is estimated using the one step TD error, as described below. The target value for $Q(s_t, a_t)$ from the Bellman optimality criterion (Sutton and Barto, 1998*b*) is $r+\gamma \text{max}_{a'} Q(s_{t+1}, a') = r+\gamma V(s_{t+1})$. Therefore, $A(s_t, a_t)$ is estimated as $(r + \gamma V(s_{t+1})) - V(s_t)$, which is just the one-step TD error in estimating $V(s_t)$.

3.4.4 Asynchronous Advantage Actor Critic

On-policy Actor Critic algorithms are non-trivial to model with Deep Neural Networks. This is because the on-policy nature of the learning leads to correlated updates by violating an important assumption made by popular stochastic gradient descent-based parameter update algorithms that the input data points are independently and identically distributed. Asynchronous Advantage Actor Critic (A3C) (Mnih *et al.*, 2016*b*) overcomes this problem by running multiple actor threads which explore different parts of the state space in parallel. The updates from the multiple threads are synchronized with respect to the global policy network parameters θ_a. This ensures that the updates made to the global policy parameters are reasonably uncorrelated. Due to multiple actor learner threads operating at different parts of the state space and pooling in gradient updates in an Advantage Actor Critic paradigm, this method is referred to as *Asynchronous Advantage Actor Critic*.

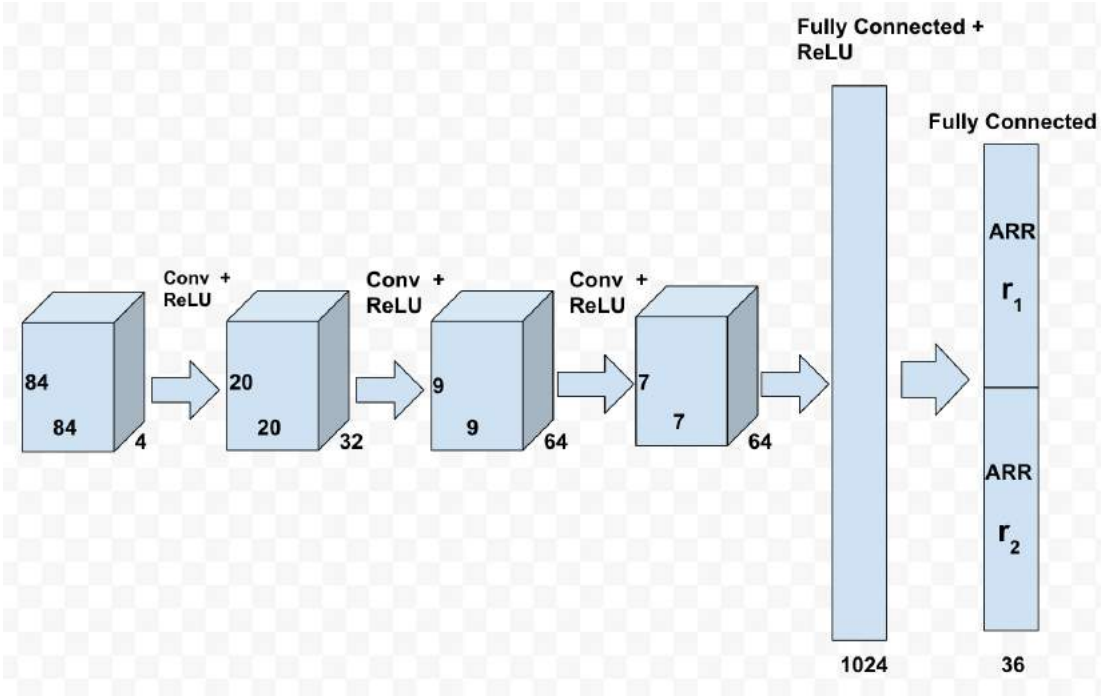


Figure 3.1: DQN's architecture under the proposed Paradigm. The upper part of the final layer corresponds to Q-values for actions that operate at action repetition rate r_1 and lower part corresponds to same actions which operate at action repetition rate of r_2.

3.5 Dynamic Action Repetition For Deep Reinforcement Learning

The key motivation behind the *Dynamic Action Repetition* paradigm is the observation that when humans execute tasks (such as playing games), the actions tend to be temporally correlated and elongated, almost always. Such behavior occurs because expectancy for upcoming action requirements is a fundamental prerequisite for human action control (Gilbert and Wilson, 2007; Thomaschke and Dreisbach, 2013). For certain states of the game, we play long sequences of the same action whereas for some states, we switch the actions performed quickly. We base these decisions on what we expect from the temporally distant future, as a result of these actions. For example, in Seaquest, if the agent is low on oxygen and deep inside the sea, we would want the agent to resurface and fill up oxygen using a long sequence of *up* actions.

Although the framework we propose is generic enough to be combined with any discrete-action space DRL algorithm, we take the example of DQN to explain the Dynamic Action repetition scheme. To demonstrate the generality of the paradigm, exper-

iments are performed with DQN as well as A3C. The paradigm, explained with the help of DQN is as follows:

Let $\mathcal{A} = \{a_1, \cdots, a_{|\mathcal{A}|}\}$ denote the set of all legal actions for a game (for Seaquest $\mathcal{A} = \{0, 1, \cdots, 17\}$). We introduce $|\mathcal{A}|$ new actions $\{a_{|\mathcal{A}|+1}, \cdots, a_{2|\mathcal{A}|}\}$. Figure 1 depicts this scheme using a diagram.

The semantics associated with the actions are as follows: action a_k results in the same basis action being played in the ALE as the action $a_{(k\%|\mathcal{A}|)}$. The difference is in terms of the number of times the basis action is repeated by the ALE emulator. DQN as well A3C operate with a static action repetition hyper-parameter which is equal to 4. Hence any selected action is repeated 4 times by ALE. In our model, action a_k is repeated r_1 number of times if $k < |\mathcal{A}|$ and r_2 number of times if $k \geq |\mathcal{A}|$. One can think about each a_k as representing a *temporally repeated* action or in RL terms as a macro-action. This scheme is implemented by doubling the number of neurons in the final layer of the DQN architecture. For a given state s, the augmented model (with double the number of output neurons as DQN) thus outputs the discrete set of value function approximations $\{Q(s, a_1), \cdots, Q(s, a_{2|\mathcal{A}|})\}$. $Q(s, a_k)$ representing an approximation to the return the agent would get on taking action a_k in state s and following the optimal policy thereafter. In the case of the augmented A3C model, the output would be a probability distribution over all the actions $\{a_1, \cdots, a_{2|\mathcal{A}|}\}$. These new macro-actions can lead to more-frequent and larger rewards, and thus significantly impact the value functions learnt by the augmented model. Note that r_1 and r_2 are hyper-parameters in this model and must be chosen before the learning begins. This paradigm presents the agent with 2 discrete levels of control, as far as action repetition is concerned, regardless of the underlying Reinforcement Learning algorithm. It is ideal to learn policies in a parametrized action space where the agent outputs an action a_k and a parameter r where r denotes the number of times that the agent wants to repeat action a_k consecutively. Such a framework would have the representative power to select the optimal number of times an action is to be executed. But, it would also be proportionately complex and as a result difficult to train. Our paradigm seeks to provide an approximation to this richer model and favors simplicity over optimality of game play, which would come at the cost of a more complicated learning procedure. We leave it to future work, to explore the learning of policies paramaterized by the ARR in the Game Playing domain, learnt with the help of an actor-critic algorithm.

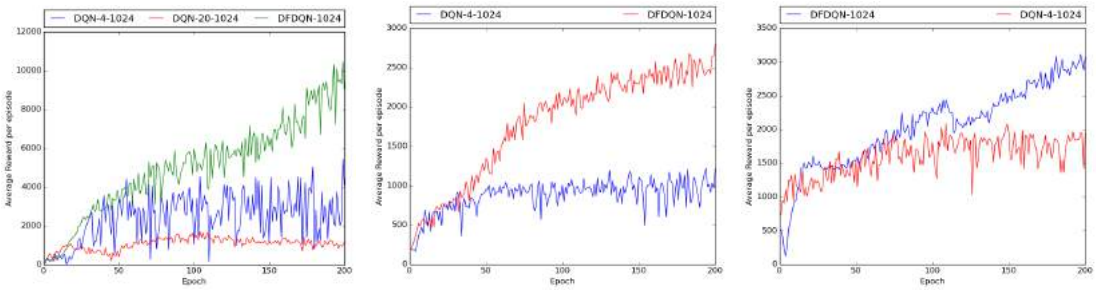
3.6 Experimental Setup and Results

To demonstrate the generality of our framework, experiments were conducted with 2 DRL algorithms: DQN and A3C. The following sub-sections document the respective experimental setup and the results.

3.6.1 Augmented DQN

General game-playing evaluation was performed on 4 Atari 2600 domain games: **Seaquest, Space Invaders, Alien** and **Enduro**. A single algorithm and architecture, with a fixed set of hyper-parameters was used to play all 4 games. The combination of DQN with our paradigm results in a model (Augmented DQN) which has the same low-level convolutional structure and input preprocessing of DQN (Mnih *et al.*, 2015*a*). Due to the partially observable nature of the Atari 2600 domain games, the last 4 frames are combined and given as a $84 \times 84 \times 4$ multi-channel input to the model. This is followed by 3 convolutional layers, which are in turn followed by 2 fully-connected layers.

Since the augmented model has double the number of output neurons as a static action-repetition model, we wanted to give more representational power to it, for being able to decide from a larger set of actions. Therefore, the augmented model has 1024 units in the pre-output hidden layer as compared to 512 units used in the DQN architecture (Mnih *et al.*, 2015*a*).



(a) Seaquest Training Comparison (b) Space Invaders Training Comparison (c) Alien Training Comparison

Figure 3.2: DQN vs DFDQN comparison for Seaquest, Space Invaders and Alien

S.No.	Game	HLS	ARR	Arch	AS
1	Seaquest	1024	4	DQN	5450
2	Seaquest	1024	20	DQN	1707
3	Seaquest	1024	D	DFDQN	**10458**
4	Seaquest	512	4	DQN	5860
5	SI	1024	4	DQN	1189
6	SI	1024	D	DFDQN	**2796**
7	SI	512	4	DQN	1692
8	Alien	1024	4	DQN	2085
9	Alien	1024	D	DFDQN	**3114**
10	Alien	512	4	DQN	1620
11	Enduro	1024	4	DQN	707
12	Enduro	512	4	DQN	729
13	Enduro	1024	D	DFDQN	**1002**
14	Enduro	1024	20	DQN	124

Table 3.1: Experimental results for DQN architectures. AS denotes the average score in the best testing epoch. SI is the game of Space Invaders. HLS denotes the pre-final layer hidden layer size.

The values of r_1, r_2 (defined in the previous section) are kept the same for all three games and are equal to 4 and 20 respectively. To ensure sufficient exploration (given that the augmented model has double the number of actions), the exploration probability ϵ is annealed from 1 to 0.1 over 2 million steps as against 1 million steps used in DQN.

We claim that the improvement in performance is not just due to the increase in the

representational power by having double the number of pre-final hidden layer neurons. to validate this hypothesis, we run DQN baselines with 1024 units in the pre-final layer for all three games. We also report the scores obtained from original DQN architecture with 512 pre-final hidden layer neurons from a recent usage of DQN in (Wang *et al.*, 2015), where DQN is reported as baseline for the Duelling DQN model. A training epoch consists of 250000 steps (action selections). This is followed by a testing epoch which consists of 125000 steps. The score of a single episode is defined to be the sum of the rewards received from the ALE (Bellemare *et al.*, 2013) emulator in that episode. The score of a testing epoch is defined to be the average of the scores obtained in all of the complete episodes played during that testing epoch. The scores reported in this section are for the testing epoch with the highest average episode score (known as the *best score* for a game). Table 1 presents the experimental results. Since the hyper-parameter for action repetition is known as Frame Skip in the ALE, the augmented model is referred to as DFDQN (Dynamic Frameskip DQN). HLS denotes the pre-final hidden layer size. SI stands for Space Invaders. ARR stands for the action-repetition rate used. A value of D for ARR represents that the action repetition rate is dynamic. Arch. denotes the architecture used. AS denotes the best average testing epoch score as defined above.

In each of plots in Fig 3.2, one epoch consists of 125000 steps (decisions). DQN-a-b refers to DQN architecture that has b units in the pre-output layer and operates with an ARR of a. DFDQN-b is the Dynamic Action Repetition variant of Deep Q-Network architecture with b units in pre-final layer.

An interesting characteristic of all the graphs is that Augmented Model's scores-graph has a large slope even at the end of 200 epochs. This means that there is still scope for the performance to improve on further training beyond 200 epochs. To verify our claim that temporally extended actions can lead to better policies, we did an analysis of the percentage of times the longer sequence of basis actions was selected. Using the network which yielded the *best score* (notion of best score defined in the previous section), we ran a testing epoch of 10000 decision-making steps, which resulted in 17 episodes for Seaquest, 18 episodes for Space Invaders, 18 episodes for Alien and 17 episodes for Enduro. We recorded the actions executed at each decision-step.

The table above shows that the augmented model is able to make good use of the

S.No.	Game	Percentage
1	Seaquest	69.99
2	Space Invaders	78.87
3	Alien	71.31
4	Enduro	67.84

Table 3.2: Longer action selection percentages for DFDQN

longer actions most of the times, but does not ignore the temporally shorter actions either. The agent has learnt through repeated exploration-feedback cycles to prefer the extended actions but still exercises the fast reflexes when the situation needs it to do so. We can safely claim that the addition of these higher action repetition options has been an important contributing factor to the improvement in performance. To strengthen our claim that there is a need for *dynamic* action repetition, we show results on two of the games (Seaquest and Enduro) in which the DQN operates with just the higher but *static* ARR of 20. Not surprisingly, it scores poorly during gameplay. Videos of some of the learned policies for Space Invaders, Seaquest and Alien are available at `https://goo.gl/VTsen0`, `https://goo.gl/D7twZc` and `https://goo.gl/aCcLb7`) respectively.

3.6.2 Augmented A3C

To demonstrate that the proposed paradigm is not specific to the DQN algorithm but rather works across different types of DRL Algorithms for game playing, we conducted similar game-playing experiments using the Asynchronous Advantage Actor Critic (Mnih *et al.*, 2016*b*) model as well. Unlike in the case of Augmented DQN, **all** the hyper-parameters were kept **exactly** the same as that in the case of A3C. This represents an adversarial setting for the augmented A3C case since deciding the optimal ARR could possibly benefit from having a larger state-representation as well as changes in other hyper-parameters. We used the LSTM-controller and the best-performing open source implementation of A3C algorithm. The baseline A3C models as well as the augmented A3C models were trained for 100 million decision steps.

Evaluation was performed by sampling from the probability distribution representing the final policy and averaging the scores over 100 episodes. The results are presented in table 3. A3C denotes the implementation of A3C used by us. DFA3C denotes

the A3C model Augmented with Dynamic Action Repetition. A3CP denotes the scores of the published A3C models (Mnih *et al.*, 2016*b*), which have been included for the sake of completeness. A note of caution is to not compare the scores obtained in this

S.No.	Game	Arch	AS
1	Seaquest	A3C	1769.12
2	Seaquest	DFA3C	**2047.74**
3	Seaquest	A3CP	1326.1
4	SI	A3C	603.04
5	SI	DFA3C	**2220.18**
6	SI	A3CP	23846.0
7	Alien	A3C	1440.82
8	Alien	DFA3C	**2722.19**
9	Alien	A3CP	945.3
10	Enduro	A3C	0.77
11	Enduro	DFA3C	**497.29**
12	Enduro	A3CP	-82.5
13	Q*bert	A3C	21184.75
14	Q*bert	DFA3C	**21405.35**
15	Q*bert	A3CP	21307.5

Table 3.3: Experimental results for A3C architectures. AS denotes the average score in the final testing epoch. SI is the game of Space Invaders.

work with the published scores in (Mnih *et al.*, 2016*b*) since they use a very different evaluation strategy called *human starts*, which is impossible to replicate exactly in the absence of human testers. We clearly see that when compared to A3C, DFA3C leads to improved metrics on all 5 tasks with the improvement on Enduro being as high as 645 times better.

Chapter 4

Option Discovery using Spatio-Temporal Clustering

4.1 Abstract

This chapter introduces an automated skill acquisition framework in Reinforcement Learning which involves identifying a hierarchical description of the given task in terms of abstract states and extended actions between abstract states. Identifying such structures present in the task provides ways to simplify and speed up Reinforcement Learning algorithms. These structures also help to generalize such algorithms over multiple tasks without relearning policies from scratch. We use ideas from dynamical systems to find metastable regions in the state space and associate them with abstract states. The spectral clustering algorithm PCCA+ is used to identify suitable abstractions aligned to the underlying structure. Skills are defined in terms of the sequence of actions that lead to transitions between such abstract states. The connectivity information from PCCA+ is used to generate these skills or options. These skills are independent of the learning task and can be efficiently reused across a variety of tasks defined over the same model. This approach works well even without the exact model of the environment by using sample trajectories to construct an approximate estimate. We present our approach to scaling the skill acquisition framework to complex tasks with large state spaces for which we perform state aggregation using the representation learned from an action conditional video prediction network and use the skill acquisition framework on the aggregated state space.

4.2 Motivation and Introduction

The core idea of hierarchical Reinforcement Learning is to break down the Reinforcement Learning problem into subtasks through a hierarchy of abstractions. In terms of Markov Decision Processes (MDP), a well studied framework in Reinforcement Learning literature (Sutton and Barto, 1998*a*), one way of looking at the full Reinforcement

Learning problem is to assume that the agent is in one state of the MDP at each time step. The agent then performs one of several possible primitive actions which along with the current state decides the next state. However, for large problems, this can lead to too much granularity: when the agent has to decide on each and every primitive action at every granular state, it can often lose sight of the *bigger* picture (Finney *et al.*, 2002). If a series of actions can be abstracted out as a single *macro* action, the agent can just remember the series of actions that was useful in getting it to a temporally distant useful state from the initial state. This is typically referred to as a skill and more specifically, as an option in (Sutton *et al.*, 1999*b*). A good analogy is a human planning his movement from current location A to destination B. We identify intermediate destinations C_i to lead us from A to B when planning from A rather than worrying about the exact mechanisms of immediate movement at A which are *abstracted over*. Options are a convenient way to formalise this abstraction. In keeping with the general philosophy of Reinforcement Learning, we want to build agents that can automatically discover options without prior knowledge, purely by exploring the environment. Thus, our approach falls into the broad category of *automated discovery of skills*. Such skills which are learnt in one task could be easily reused in a different task if necessary. Options also make exploration more efficient by providing the decision maker with a high-level behavior to look ahead to the completion of the corresponding subroutine. This helps to come up with scalable approaches in Reinforcement Learning.

Our focus in this chapter is to present our framework on automated discovery of skills. Automated discovery of skills or options has been an active area of research and several approaches have been proposed for the same. The current methods could be broadly classified into sample trajectory based and partition based methods. Some of them are:

- Identifying bottlenecks in the state space, where the state space is partitioned into sets. The transitions between two sets of states that are rare introduce *bottleneck* states at the respective points of such transitions. Policies to reach such bottleneck states are cached as options - for example, (McGovern and Barto, 2001).
- Using the structure present in a factored state representation to identify sequences of actions that cause what are otherwise infrequent changes in the state variables: these sequences are cached away as options (Hengst, 2004).
- Obtaining a graphical representation of an agent's interaction with its environment and using betweenness centrality measures to identify subtasks (Şimşek and Barreto, 2009).

- Using clustering methods (spectral or otherwise) to separate out different strongly connected components of the MDP and identifying *bottlenecks* that connect different clusters (Menache *et al.*, 2002).

There are certain deficiencies with the above methods, even though they have had varying degrees of success. Bottleneck based approaches do not have a natural way of identifying the part of the state space where options are applicable without external knowledge about the problem domain. Spectral methods need some form of regularization in order to prevent unequal splits that might lead to arbitrary splitting of the state space. There have also been attempts to learn skill acquisition in robotics. (Konidaris *et al.*, 2011) attempt to discover skills for a robot given abstractions of the surroundings it has to operate with. (Ranchod *et al.*, 2015) try to use inverse Reinforcement Learning for skill discovery with reward segmentation. However, the above two works assume the knowledge of the abstractions of the world and demonstrations of expert trajectories respectively while our goal is to acquire skills and discover the abstractions without any prior knowledge.

We adopt a framework that detects well-connected or meta-stable regions of the state space from an MDP model estimated from trajectories. We use PCCA+, a spectral clustering algorithm from conformal dynamics (Weber *et al.*) that not only partitions the MDP into different regions but also returns the connectivity information between the regions, unlike other clustering approaches used earlier for option discovery. This helps us to build an abstraction of the MDP, where we call the regions identified by PCCA+ as *abstract states*. Then we define options that take an agent from one abstract state to another connected abstract state. Since PCCA+ also returns the membership function between states and abstract states, we propose a very efficient way of constructing option policies directly from the membership function, which is to perform *hill climbing* at the granular states on the membership function of the *destination* abstract state, to yield the option policy without further learning. Since the abstract states are aligned with the underlying structure of the MDP the same option policies could be efficiently reused across multiple tasks in the same MDP. Once we have these options, we could use standard Reinforcement Learning algorithms to learn a policy over these subtasks to solve a given task. Specifically, we use SMDP Q-Learning (Sutton *et al.*, 1999*b*) and Intra Option Q-Learning (Sutton *et al.*) for our experiments. Note that our approach works well even without the exact model of the full MDP, by being able to work on

approximate estimate of the model using sample trajectories, as demonstrated by the experiments.

Finally, we present our attempt to extend our pipeline to large state spaces where spectral methods on the original state space are infeasible. We therefore propose to operate the PCCA+ pipeline on an *aggregated state space*. The original state space is *aggregated* through clustering on the representation of the state space learned using deep neural networks. Specifically, we try to learn a representation that captures spatio-temporal structure of the state-space, for which we borrow the framework of (Oh *et al.*, 2015*a*) which uses a Convolutional-LSTM Action Conditional Video Prediction Network to predict the next frames of the game conditioned on the trajectory so far. We use the Arcade Learning Environment (ALE) platform (Bellemare *et al.*, 2012) which provides a simulator for Atari 2600 games. We present our results on the Atari 2600 game Seaquest which is a relatively complex game and requires abstract moves like *filling up oxygen, evading bullets, shooting enemies*.

We summarize the components of this chapter below:

- A novel automated skill acquisition pipeline to operate without prior knowledge or abstractions of the world. The pipeline uses a spectral clustering algorithm from conformal dynamics - PCCA+, which builds an abstraction of the MDP by segmenting the MDP into different regions called *abstract states* and also provides the membership function between the MDP states and *abstract states*.
- An elegant way of composing options (sequence of actions to move from one abstract state to another) from the membership function so obtained by doing hill climbing on the membership function of the *target abstract state* at the granular states belonging to the *current abstract state*.
- Extension of the pipeline to complex tasks with large state spaces, for which we propose to run the pipeline on an *aggregated state space* due to infeasibility to operate on the original state space of the MDP. To aggregate the state space, we learn a representation capturing spatio-temporal aspects of the state space using a deep action-conditional video prediction network.

Our approach can be considered as an attempt to address automated option discovery in RL using spatio-temporal clustering: Spatial because the abstract states are segmenting the state space into abstractions (clusters); and Temporal since the transition structure of the MDP is used to discover these abstractions.

4.3 Preliminaries

4.3.1 Markov Decision Process

Markov Decision Process (MDP) is a widely used framework on which Reinforcement Learning algorithms are used to learn control policies. A finite discrete MDP is formalised as $M = (S, A, P, R)$ with S with a finite discrete state space S, finite action space A, a distribution over the next states $P(s, a, s')$ when an action a is performed at state s, and a corresponding reward $R(s, a, s')$ specifying a scalar cost or reward associated with that transition. A policy is a sequence of actions involved in performing a task on the MDP, while value functions are a measure of the expected long term return in following a policy in the MDP.

4.3.2 Options

The option framework is one of the formalisations used to represent hierarchies in RL (Sutton *et al.*, 1999*b*). Formally, an option is a tuple $O = (I, \mu, \beta)$ here:

- $I \subseteq S$ is the initiation set: a set of states from which the action can be activated
- $\mu : S \times A \rightarrow (0, 1)$ is a policy function where $\mu(s, a)$ represents the preference value given to action a in state s following option O.
- $\beta : S \rightarrow (0, 1)$ is the termination function: When an agent enters a state s while following option O, the option could be terminated with a probability $\beta(s)$.

When the agent is in state s where it can start an option, it can choose from all the options O for which $s \in I(O)$ and all primitive actions that can be taken in s. This choice is dictated by the policy guiding the agent.

4.3.3 SMDP Value Learning

An MDP appended with a set of options is a Semi-Markov Decision Process (SMDP). In an SMDP, at each time step, the agent selects an option which can be initiated at that state and follows the corresponding option policy until termination. In SMDP theory, the effects of an option are modelled using r_s^o and $p_{ss'}^o$ which represent the total return

and the probability of terminating at state $s^{'}$ for an option o initiated at state s.

$$r_s^o = E\{r_t + \cdots + \gamma^{k-1} r_{t+k-1} | s_t = s, o_t = o\}$$

$$p_{ss'}^o = \sum_{j=0}^{\infty} \gamma^j Pr\{s_{t+j} = s^{'} | s_t = s, o_t = o\}$$

Each option is viewed as an opaque, indivisible unit and its structure not utilized. The update rule of SMDP Q-learning (Sutton *et al.*, 1999*b*) is given by

$$Q(s,o) \leftarrow Q(s,o) + \alpha[r + \gamma^k max_{a \in O} Q(s,a) - Q(s,o)]$$

where Q is the action/option value function.

4.3.4 Intra-Option Value Learning

A major drawback of SMDP learning methods is that we need to execute an option completely upto termination before we begin to learn about its outcome. At any point of time, we would potentially be learning about the value function of only either an option standalone or primitive action standalone. This slows down the learning significantly because an option is treated as a black-box and structure within the option is not exploited while learning. The motivation for intra-option value learning methods comes from being able to bootstrap the structural commonalities among options and primitive actions for more efficient Reinforcement Learning (Sutton *et al.*, 1998).

In intra-option methods, we allow all options that are consistent with an observed trajectory to have their values updated, effectively making these methods *off-policy* since they learn about one policy (or sub-policy) while actually executing a different policy (or sub-policy). It can therefore help the agent to learn the values of certain options without even executing those options.

True to the motivation, Intra-option value learning is much more efficient than SMDP methods. Let us consider the following example. Let o be an option chosen at state s_t at time t which terminates at time $t + \tau$. In SMDP methods, we obtain a single training example to update $Q(s_t, o)$ for the τ time steps. However if option o is Markov, then each of the transitions in the τ time steps are also valid training examples

since o was initiated at each of the time steps. These can be used to learn $Q(s_i, o)$ for $i \in [t, t+\tau]$. Further, if the action taken at some of these time steps is consistent with any other option $o^{'}$, we can update Q-value of these actions as well, based on this transition. Thus, intra-option value learning methods are able to use behavioral transitions collected from different policies much more effectively than SMDP methods.

Suppose action a_t is executed at a state s_t to produce next state s_{t+1} and reward r_{t+1}. Let us assume that a_t was selected in a way consistent with the Markov policy μ of an option $o =< \mu, \beta, I >$. The following intra-option Q-learning update rule applies to every option o consistent with action a_t:

$$Q(s_t, o) \leftarrow Q(s_t, o) + \alpha[(r_{t+1} + \gamma Q(s_{t+1}, o)) - Q(s_t, o)]$$

4.4 Spectral Graph Theory

We wish to identify spatial structures in the environment in order to generate temporal abstractions which are aligned to the structure of the underlying MDP. It has been observed that the spectra of a graph, constituted by the eigensystem, provides an embedding of the underlying graph. The eigenvalues are closely related to most major invariants of the graph and link one extremal property to another.

The Laplacian ($\mathcal{L}$) is an algebraic representation of a graph which allows a natural link between discrete representations such as graphs and continuous representations such as manifolds and vector spaces. One of the most important applications of the Laplacian is spectral clustering used in the graph partitioning problem (Luxburg, 2007). Thus, it provides an elegant way to detect spatial abstractions in the environment.

The automated option discovery framework proposed in (Kumar *et al.*, 2012) uses the normalized graph Laplacian $\mathcal{L} = \mathcal{D}^{-1}(\mathcal{D} - \mathcal{W})$ where $\mathcal{W}$ is the adjacency matrix representation of the underlying MDP graph and $\mathcal{D}$ is the valency matrix, i.e., a diagonal matrix with entries corresponding to the degree of each vertex.

There have been many successful applications of spectral clustering methods for spatial abstraction in a wide range of domains (Shi and Malik, 2000*a*; White and Smyth, 2005; Simsek *et al.*, 2005; Xu *et al.*, 2003; Cai *et al.*, 2005). As mentioned above, there

are strong connections between the topological properties of a graph and the graph Laplacian matrix, which spectral clustering methods exploit for graph partitioning.

The framework described in (Kumar *et al.*, 2012) uses a spectral clustering approach that attempts to exploit the structural properties in the configuration space of the objects as well as the spectral subspace. This approach draws inspiration from the conformal dynamics literature (Weber *et al.*, 2004) where a similar analysis is done to detect conformal states of a dynamical system and they proposed a spectral clustering algorithm **PCCA+**.

The advantages of PCCA+ include:

- *Global as well as local information to define spatial abstractions*. It exploits the local structural properties of the underlying data space by using pairwise similarity functions, while using spectral methods to encode the global structural properties.
- *A formal notion of macro states as vertices of a simplex in the eigen-subspace of the Laplacian*. The clustering is performed by minimizing deviations from a simplex structure and hence does not require any arbitrary regularization term. The clustering procedure does not assume anything about the underlying structure and the mapping to a simplex is inherently built in the properties of the Laplacian.
- *Characteristic functions that describe the degree of membership of each state to a given abstract state*. We can interpret the membership functions as the likelihood of a state belonging to a particular abstract state. The algorithm could also generate crisp partitioning of the states into abstract states, as and when required.
- *Connectivity information between the abstract states* which is seldom required in dynamical systems. For example one might be interested to know the connectivity information between abstract states to learn decision policies across such states.

4.4.1 Perron Cluster Analysis (PCCA+)

Given an algebraic representation of the graph representing an MDP, we want to find suitable abstractions aligned to the underlying structure. A spectral clustering algorithm can be used to do this. Central to the idea of spectral clustering is the graph Laplacian which is obtained from the similarity graph. In this approach, the spectra of the Laplacian $\mathcal{L}$ (derived from the adjacency matrix $\mathcal{S}$) is constructed and the best transformation of the spectra is found such that the transformed basis aligns itself with the clusters of data points in the eigenspace. A projection method described in (Weber *et al.*) is used to find the membership of each of the states to a set of k special points lying on the

transformed basis, which are identified as vertices of a simplex in the $\mathbb{R}^k$ subspace (the Spectral Gap method is used to estimate the number of clusters k). For the first order perturbation, the simplex is just a linear transformation around the origin and to find the simplex vertices, one needs to find the k points which form a convex hull such that the deviation of all the points from this hull is minimized. This is achieved by finding the data point which is located farthest from the origin and iteratively identify data points which are located farthest from the hyperplane fit to the current set of vertices. Refer to Algorithm 1 for details.

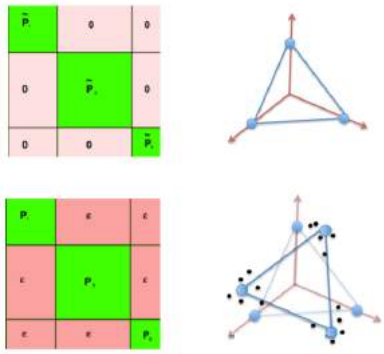
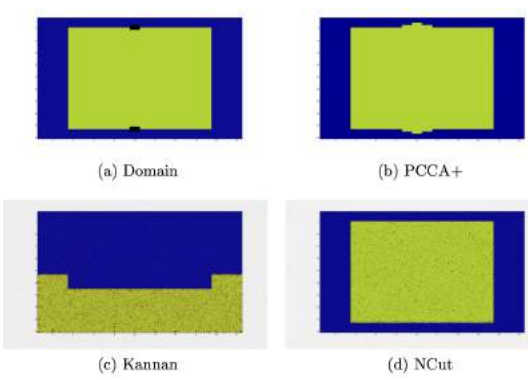
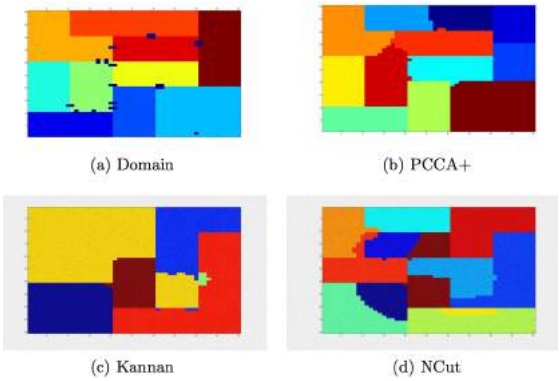


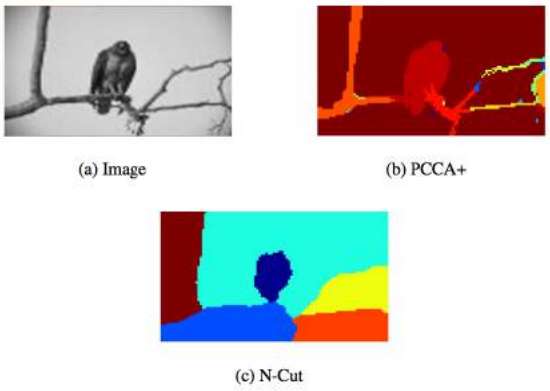
Figure 4.1: Simplex First order and Higher order Perturbation: Shows the visualization of the first order perturbation assumption to identify the simplex vertices on data points which have higher order perturbations. The first case shows data points which satisfy the first order assumption and hence the simplex fits perfectly without any noise. In the second case, the simplex vertices obtained through the linear transformation are only able to capture the structure with some noise and these deviations could be understood as the effect of the higher order perturbations present.



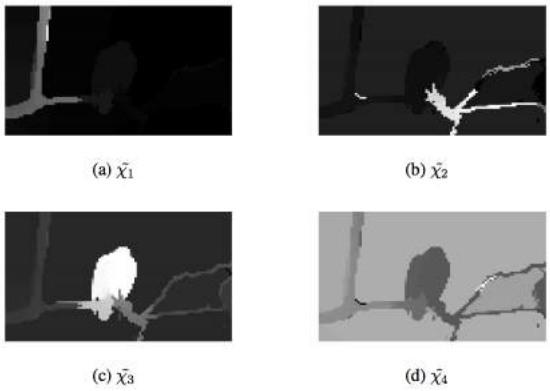
(a) PCCA+ vs rest - Room-in-a-room Domain



(b) PCCA+ vs rest - Maze Domain



(c) PCCA+ vs NCut - Real Image



(d) PCCA+ Membership functions visualization

Figure 4.2: PCCA+ visualizations

Algorithm 1 PCCA+

1: Construct Laplacian $\mathcal{L}$

2: Compute n (number of vertices) eigenvalues of $\mathcal{L}$ in descending order

3: Choose first k eigenvalues for which $\frac{e_k - e_{k+1}}{1 - e_{k+1}} > t_c$ (Spectral Gap Threshold).

4: Compute the eigenvectors for corresponding eigenvalues $(e_1, e_2, \cdots, e_k)$ and stack them as column vectors in eigenvector matrix $\mathcal{Y}$.

5: Let's denote the rows of $\mathcal{Y}$ as $\mathcal{Y}(1), \mathcal{Y}(2), \cdots, \mathcal{Y}(N) \in \mathbb{R}^k$.

6: Define $\pi(1)$ as that index, for which $||Y(\pi(1))||_2$ is maximal. Define $\gamma_1 = span\{Y(\pi(1))\}$.

7: **For** $i = 2, \cdots, k$: Define π_i as that index, for which the distance to the hyperplane γ_{i-1}, i.e., $||\mathcal{Y}(\pi_i) - \gamma_{i-1}||_2$ is maximal. Define $\gamma_i = span\{\mathcal{Y}(\pi_1), \cdots, \mathcal{Y}(\pi_i)\}$. To compute $||\mathcal{Y}(\pi_i) - \gamma_{i-1}||_2$, use $||\mathcal{Y}(\pi_i) - \gamma_{i-1}||_2 = ||\mathcal{Y}(\pi_i) - \gamma_{i-1}^T((\gamma_{i-1}\gamma_{i-1}^T)^{-1}\gamma_{i-1})\mathcal{Y}(\pi_i)^T)||$

Comparison to other clustering algorithms

We are in particular concerned with dynamical systems where there are no a priori assumptions on the size and the relative placements of the metastable states. So it becomes essential for the clustering algorithm to work in a generalized setting as far as possible. When different spectral clustering algorithms are tested on different classes of problems, PCCA+ was found to give better results (Fig 4.2) in capturing the structure (topology) (Kumar *et al.*, 2012). Normalized Cut (NCut) Algorithms by (Shi and Malik, 2000*b*),(Ng *et al.*, 2002) and PCCA+ by (Weber *et al.*) have deep similarities, but the main difference behind the usage of PCCA+ is the change of point of view for identification of metastable states from crisp clustering to that of relaxed almost invariant sets. Both the methods are similar till the identification of the eigenvector matrix as points in n dimensional subspace but after this, (Shi and Malik, 2000*b*),(Ng *et al.*, 2002) use standard clustering algorithms like K-Means while PCCA+ uses a mapping to the simplex structure. Since the bottlenecks occur rather rarely, it is shown by (Weber *et al.*) that in general for such cases, soft methods outperform crisp clustering. Although the other spectral algorithms result in good clusters for the simple room-in-a-room domain, they give poor results on more topologically complex spaces and need some form of regularization to work well in such settings (Fig 4.2). PCCA+ has an intrinsic regularization mechanism by which it is able to cluster complex spaces neatly.

Spatial Abstraction using PCCA+

Although the spectra of the Laplacian preserves the structural properties of the graph, clustering data in the eigenspace of the Laplacian does not guarantee this. For example, k-means clustering (Ng *et al.*, 2001) in the eigenspace of the Laplacian will only work if the clusters lie in disjoint convex sets of the underlying eigenspace. Partitioning the data into clusters by projecting onto the largest k-eigenvectors (Meila and Shi, 2001) does not preserve the topological properties of the data in the eigenspace of the Laplacian. For the task of spatial abstraction, the proposed framework requires a clustering approach that exploits the structural properties in the configurational space of objects as well as the spectral sub-space, quite unlike earlier methods. They take inspiration from the work of (Weber *et al.*, 2004) which proposes a spectral clustering algorithm PCCA+ based on the principles of Perron Cluster Analysis of the transition structure and ex-

tends their analysis to detect spatial abstractions in autonomous controlled dynamical systems. In this approach, the spectra of the Laplacian $\mathcal{L}$ (derived from the adjacency matrix $\mathcal{S}$) is constructed and the best transformation of the spectra is found such that the transformed basis aligns itself with the clusters of data points in the eigenspace. A projection method described in (Weber *et al.*, 2004) is used to find the membership of each of the states to a set of special points lying on the transformed basis, which are identified as vertices of a simplex in the $\mathbb{R}^k$ subspace (the Spectral Gap method is used to estimate the number of clusters k). For the first order perturbation, the simplex is just a linear transformation around the origin and to find the simplex vertices, one needs to find the k points which form a convex hull such that the deviation of all the points from this hull is minimized. This is achieved by finding the data point which is farthest located from the origin and iteratively identify data points which are located farthest from the hyperplane fit to the current set of vertices.

The PCCA+ algorithm returns a membership function, χ, defining the degree of membership of each state s to an abstract state S_j. The connectivity information between two abstract states (S_i, S_j) is given by $(i, j)^{th}$ entry of $\chi^T L \chi$ while the diagonal entries provide relative connectivity information within a cluster. The connectivity information is utilized to learn decision policies across abstract states. This framework also contains an intrinsic mechanism to return information about the goodness of clustering of states from the presence of sharp peaks (indicates good clustering) in the eigenvalue distribution.

The transition structure of the MDP could be arbitrarily complex and hence, to identify abstractions that are well aligned to the state space transition structure, we use PCCA+ to abstract the MDP in place of other clustering algorithms. This is inpired by (Weber *et al.*) who use PCCA+ to detect the conformal states of a dynamical system by operating on the transition structure. On providing the graph (transition structure) of the MDP to PCCA+, we derive the abstraction of the MDP where the simplex vertices identified by PCCA+ are the *abstract states*. We also get a membership function from PCCA+, χ, which defines the degree of membership of each state s to an abstract state S. We describe how this information can used to compose options in the next section.

4.5 Composing Options from PCCA+

Given the membership functions and the abstractions discovered by PCCA+, there is a very elegant way to compose option policies to move from one abstract state to another. The transition to another abstract state is done simply by *following the positive gradient (hill climbing) of the membership value to the destination abstract state* (Kumar *et al.*, 2012). This navigates the agent through states whose membership functions to the target abstract state progressively increases.

Specifically, consider that we have N states $s_1, s_2, s_3, \cdots, s_N$ and k abstract states $S_1, S_2, \cdots, S_k$. Typically, $N \gg k$. Let χ_{ij} denote the membership of state s_i to the abstract state S_j. s_i is said to *belong* to abstract state S_a, where $a = \text{argmax}_j \chi_{ij}$. Thus, any option between abstract states S_i and S_j will start at a state in S_i and end at a state in S_j. The connectivity information between two abstract states (S_i, S_j) is given by the $(i,j)^{\text{th}}$ entry of $\chi^T L \chi$ where L is the Laplacian corresponding to the MDP transition matrix. The diagonal entries provide the relative connectivity information within a cluster. An option is generated for every pair of connected abstract states. For an option from S_i to S_j:

- The initiation set I represents states that belong to S.
- The **option policy** $\mu(s,a)$ that takes the agent from abstract state S_i to S_j is a stochastic gradient function given by:

$$\mu(s,a) = \alpha(s) \left[\max\left((\chi_{S_j S_i}(s,a) - \chi_{S_j}(s)), 0\right)\right]$$

  where

$$\chi_{S_j S_i}(s,a) = \left[\left(\sum_{s'} P(s,a,s')\chi_{S_j}(s')\right) - \chi_{S_j}(s)\right] \;\forall\; s \in S_i$$

  $\alpha(s)$ is a normalisation constant to ensure μ represents a probability function in the range $[0,1]$.

- Finally, **termination condition** β is a function which assigns the probability of termination of the current option at a state s. It could also be viewed as the probability of a state s being decision epoch given the current option being executed. For an option taking the agent from abstract state S_i to S_j, we define β as follows:

$$\beta(s) = \min\left(\frac{\log(\chi_{S_i}(s))}{\log(\chi_{S_j}(s))}, 1\right) \;\forall\; s \in S_i$$

We offer an intuitive explanation for the choice of the option policy μ and option termination condition β. $\chi_{S_j S_i}(s,a)$ is the expected increase in the membership function

to abstract state S_j on taking action a at state s, while $\chi_{S_j}(s)$ is the membership function to S_j at current state s. The probability of an action a involved in taking the agent from state $s \in S_i$ to an abstract state S_j must proportional to the expected increase in the membership function to the abstract state S_j if positive and 0 if the expected increase is negative. The choice for the termination condition also has a simple heuristic: When transitioning from S_i to S_j, the agent would encounter bottleneck state(s) $s*$ which would approximately satisfy $\chi_{S_i}(s) = \chi_{S_j}(s)$, and hence be suitable for the termination of the option. Till then, the membership value of S_i would be higher than S_j (with the difference gradually reducing along the option trajectory), and hence the probability to terminate would be proportional to how much of a *bottleneck* the current state s is. Even though we do not look for *bottleneck states* directly in our approach unlike (McGovern and Barto, 2001), the termination condition proposed by (Kumar *et al.*, 2012) naturally captures transitioning through *bottleneck* states.

4.6 Option Discovery using Spatio-Temporal Clustering (ODSTC)

The previous section discussed how to discover options given that abstract states are obtained through the PCCA+ approach operating on the transition structure of the MDP. However, an *online* agent has no prior knowledge of the model of the MDP and has to learn these abstractions from scratch. In this section, I will discuss the ODSTC pipeline, first for small state spaces as envisioned by (Kumar *et al.*, 2012). We shall first see how the pipeline works in terms of a pseudocode of the pipeline. This is followed by some additional aspects that can be injected into this pipeline such as grounding the abstractions discovered with the reward function of the MDP and results on the 3-room domain task. Once we understand the efficacy of the pipeline for small problems, we shall delve into the problems in using it for larger state spaces, how to modify the pipeline to be usable for larger problems and results for the same.

4.6.1 ODSTC for Small State Spaces

Let us look at Algorithm 2 for an *online* agent to perform Option Discovery using Spatio-Temporal Clustering (ODSTC) in the case of small state spaces where it is feasible to compute transition matrices over the entire state space by bootstrapping from trajectories sampled from different policies. We call this as the ODSTC Online Agent Pipeline - Small.

Let O be the set of options available to the agent after every iteration. Initially, we assume that the agent starts without prior knowledge. Hence, to begin with, O is empty. We denote the set of new options discovered at the end of the iteration as o and augment the agent with these additional options.

Algorithm 2 ODSTC Online Agent Pipeline - Small

1: **while** Not Converged **do**
2: Sample trajectories $\tau_{i i=i}^{k}$ using current behavioral policy π.
3: Estimate transition model $T(s'|s,a)$ from sample trajectories $\tau_{i i=i}^{k}$.
4: Operate PCCA+ on the estimated model T to derive abstract states and memberships.
5: Discover options $o = o_{i i=1}^{N}$ from the abstract states and memberships through hill climbing.
6: Augment the agent with new options: $O = O \cup o$.
7: Update value functions Q and behavioral policy π using SMDP Q Learning
8: **end while**

4.6.2 Model estimation and incorporation of reward structure

For every sampled trajectory, we maintain transition counts $\phi_{ss'}^{a}$ of the number of times the transition $s \xrightarrow{a} s'$, starting from ϕ_0. These transition counts are used to populate the local adjacency matrix D and the transition count model as: $D_{\text{posterior}}(s,s') = D_{\text{prior}}(s,s') + \sum_a \phi_{ss'}^{a}$, $U_{\text{posterior}}(s,a,s') = U_{\text{prior}}(s,a,s') + \phi_{ss'}^{a}$, after every sampled trajectory. PCCA+ is used to find suitable spatial abstractions and subtask options where the transition probabilities in the hill-climbing step are calculated as $P(s,a,s') = \frac{U(s,a,s'}{\sum_{s'} U(s,a,s')}$. We then use an SMDP value learning algorithm to find the optimal policy over the constructed options.

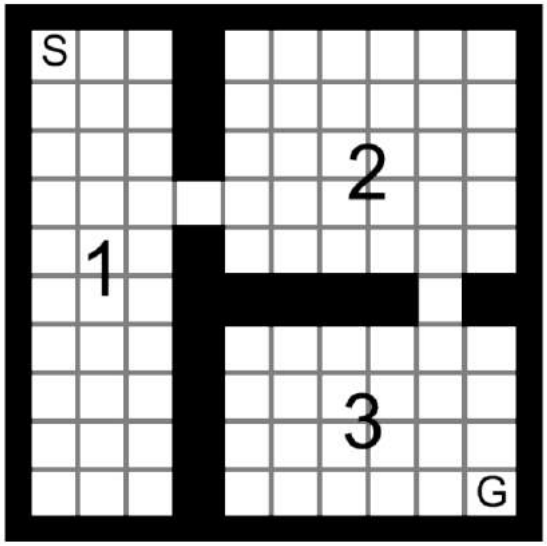
The underlying transition structure $\phi^a_{ss'}$ encodes only the topological properties of the state space. There are other kinds of structures in the environments which we would like an autonomous agent to discover. For example, a monkey climbing a tree to pick up a fruit abstracts the task as to reach the branch of the tree nearest to the fruit. These structures are functional in nature, depending on the functional properties of the task. We use reward counts for each transition to encode the functional properties in the transition structure. The idea is that the spatio-temporal abstractions should *degenerate* in places where there is a spike in the reward distribution, so that our agent interprets a state of high reward (as goal states typically are) as a different abstract state, and naturally composes options that would lead it to that state. Let $R^a_{ss'}$ be the reward in performing a transition $s \xrightarrow{a} s'$. We modify the local adjacency matrix D's posterior update as: $D_{\text{posterior}} = D_{\text{prior}}(s, s') + \sum_a \phi^a_{ss'} e^{-v|R^a_{ss'}|}$ where v is the regularization constant that balances the relative weight of the underlying reward and transition distribution. We use an exponential weighting for rewards to ensure that the adjacency function has a very low value at the spike points where we want to abstraction to degenerate. It also returns a value of 1 for zero rewards, preserving the original transition structure and allows easy tuning of relative weights (v).

4.6.3 Matching options

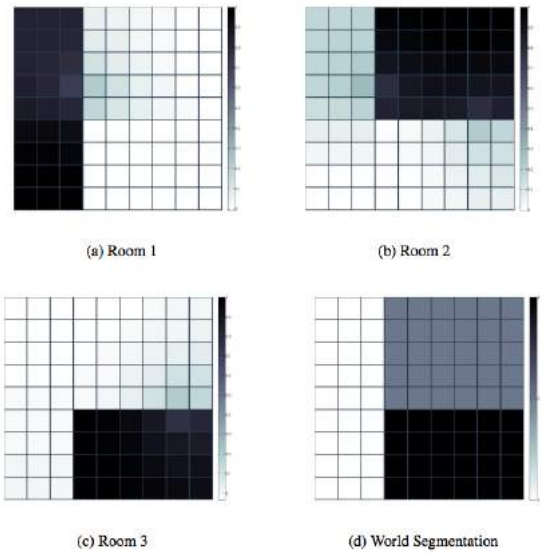
Constructing spatio-temporal abstractions as described above poses a matching problem for learning policies using SMDP learning techniques. With every sampled trajectory, the structure of the transition matrix changes which in turn changes the spatial abstractions identified. In order to define the SMDP update rule, we still need a mechanism to match the previous options to new ones. This can be done easily by mapping the vertices of the simplex returned by PCCA+. Hence, consider two iterations which returned the simplex vertices $\widetilde{Y_1}$ and $\widetilde{Y_2}$ where $\widetilde{Y_1}$ and $\widetilde{Y_2}$ are matrices whose rows denote the location of these vertices. The matching criteria $\kappa_{12} = \widetilde{Y_1}\widetilde{Y_2}^{-1}$ is used to assign vertex i of simplex 1 to vertex j of simplex 2 using the Munkres algorithm (a.k.a Hungarian Method) where the match weighting between i and j is $\kappa_{12}(i, j)$.

4.6.4 Experiments on 3 Room Domain

Let us consider a simple 3 room domain as shown, borrowed from (Kumar *et al.*, 2012). Below are a few experimental results from (Kumar *et al.*, 2012) that are illustrative to elucidate the efficacy of the PCCA+ pipeline for small state spaces. The world is composed of rooms separated by walls as shown in Figure 4.3a. For the task at hand, let us assume that the goal of the agent is to start from the tile marked S and reach the goal tile marked G. The agent can either move a step in the direction towards north, east, west or south. PCCA+ discovers three abstract states without incorporating the reward structure, each corresponding to one room in the 3-room world. On incorporation of the reward structure, there would be 4 abstract states, with the additional one corresponding to the goal state alone. The doorway could be seen as bottleneck states at which the membership functions lead to termination conditions of options exiting one abstract state (current room) and entering another abstract (destination room). The navigation task could simply be abstracted as exiting the current room by moving to the doorway and entering the correct room to reach the goal state in it. We visualize the membership function identified by PCCA+ on the 3-room world in Fig 4.3b without reward structure incorporation. It is clear that each abstract state can be identified with one of the three rooms in the world and the MDP transition structure has been neatly segmented by PCCA+. Fig 4.3c shows the MDP segmentation and the option policies discovered without and with the reward structure incorporated. For the first case (left), without the reward structure, there are only 3 abstract states corresponding to each room and hence the option policies discovered would be navigation from one room to another. The figure shows stochastic option policies to navigate to room 3 from the other two rooms. In the second case (right), with the reward (goal) considered in the transition structure, 4 abstract states are identified, with the new one lonely being the goal state in the bottom right corner as indicated. The figure shows the (stochastic) option policies in getting to room 3 from rooms 1 and 2, and in getting from the group of granular states other than the goal state in room 3 to goal state. We hence clearly see the advantage of discovering these *structural* (from transition structure) and *functional* (from reward structure) abstractions since the solving the task now is as simple as planning to move from one abstract state to another instead of trying to learn the optimal policy at every granular state in the MDP.

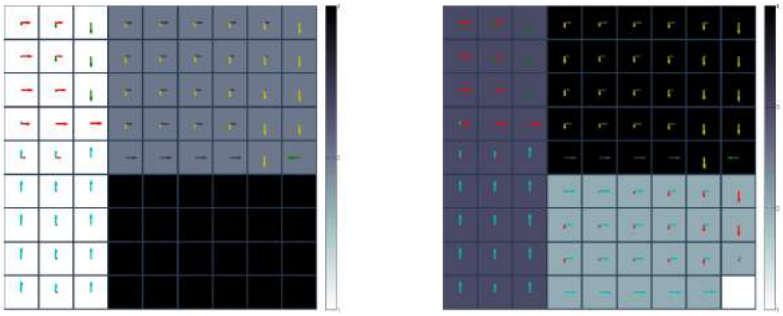


(a) 3-Room domain



(a) Room 1 (b) Room 2 (c) Room 3 (d) World Segmentation

(b) Membership Function



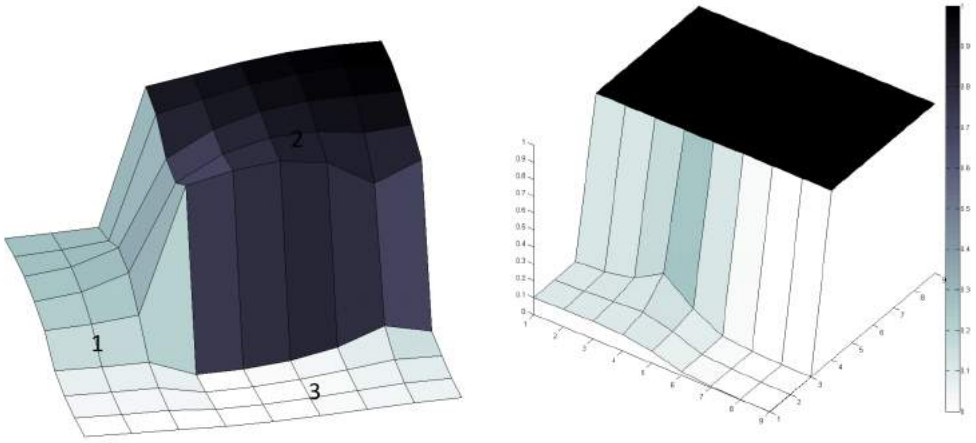
(a) Policy composed from transition structure only (b) Policy composed from transition and reward structure

(c) Effect of Reward Structure Incorporation

Figure 4.3: 3 Room Domain Visualization

We next look at plots in Figure 4.4 of the membership function and the termination criterion in 3D. The membership plot in 3D reveals a lot more in terms of understanding how naturally an option arises by ascending the membership function, or in other words, *hill climbing*. For instance, in Figure 4.4a, one can think of moving from abstract state 1 to abstract 2 by climbing on sloped surface of states that are part of the abstract state 1 until the flat surface of abstract state 2 is attained. The gradient between the membership functions of abstract states 1 and 2 provides a natural way to perform this hill climbing. The next plot in Figure 4.4 allows us to better understand the termination criterion pictorially. To recount what the termination criterion is, for an option taking the agent from abstract state S_i to S_j, we defined β as $\beta(s) = \min\left(\frac{\log(\chi_{S_i}(s))}{\log(\chi_{S_j}(s))}, 1\right) \ \forall\ s \in S_i$. We see from the 3D intensity plot that the probability of an option leading from abstract

state 1 to 2 gradually increases on hill climbing, getting closer to a probability of 1 as you approach the bottleneck states separating the two abstract states. This is distinctly visible by the darker regions marked on the top. The visualization is consistent with the math in the termination criterion where the membership functions are approximately the same and $\log 1 = 0$.



(a) Membership Function 3D (b) Termination Condition

Figure 4.4: 3 Room Domain 3D Visualization

4.6.5 Modifying the pipeline for larger state spaces

Seaquest

Consider the task of Seaquest, which is an Atari 2600 game in which the goal of the RL agent is to retrieve as many divers as possible while destroying enemy submarines and killer sharks before the oxygen runs out. The reward structure is as follows: At the beginning of the game, killing any of the enemies is worth 20 points and retrieving a diver is worth 50 points. Every time the agent reaches the surface with 6 divers, the reward for killing an enemy is increased by 10 and that of retrieving a diver is increased by 50. Also, the agent receives an extra life and receives a bonus based on how much oxygen it has left. On the other hand, if the agent resurfaces with less than 6 divers for oxygen, it doesn't receive any bonus. In the extreme scenario of it resurfacing with 0 divers, it loses a life. We use the Arcade Learning Environment simulator (Bellemare *et al.*, 2013) for running experiments with Seaquest.

We chose Seaquest for experimentation among the Atari 2600 games since it is one

of the games for which the Deep Q-network performs inferior to a human expert. The reason for this is that killing enemies is rewarded immediately while retrieving divers is rewarded only after 6 of them have been retrieved. This makes it a complex RL task which requires long-term planning.

Figure 4.5: A visual scene from Seaquest

Problems with the pipeline so far

Figure 4.6 shows the flowchart for the pipeline so far. It pictorially summarizes the pseudocode for ODSTC for small state spaces. The problems in using this pipeline for larger state spaces mainly comes from having to estimate transition dynamics of the MDP. For MDPs with exponential state spaces operating on pixel-level input such as Atari 2600 games like Seaquest, the transition matrix is over such large dimensions and incredibly sparse. It is practically infeasible to think of constructing such matrices owing to memory and sparsity issues. On the other hand, the PCCA+ pipeline is rigid that you need to feed in a weighted graphical transition structure for the abstractions to be discovered through spectral clustering.

Secondly, similar to the motivation for using Deep Q Networks, it is infeasible to perform tabular SMDP learning methods for large state space problems like Seaquest for the same reasons that transition matrices are infeasible. Storing explicit action and option values for every state is not a practical approach.

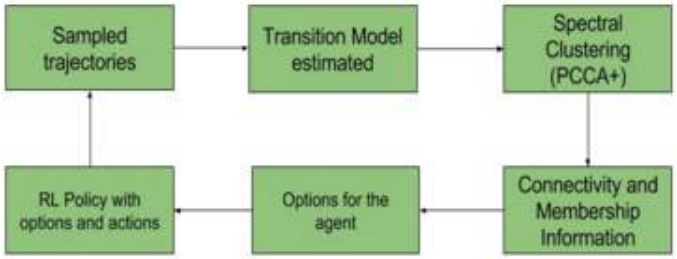


Figure 4.6: Option generation pipeline

Modified pipeline with function approximation

To address the problem of estimating the transition dynamics on the original state space, I propose a simple and reasonably effective solution. Instead of having to estimate the transition matrix on the original state space, we learn a lower dimensional representation of the original state space by mapping the original state space through a highly non linear mapping to a much lower dimensional manifold. The state space can then by binned by performing simple K-Means clustering on the lower dimensional representation. This leads us to a discrete state space of much lower dimensionality than the original discrete state space of exponential dimensions. The transition matrix can then be estimated on this lower dimensional *aggregated* state space for PCCA+ to operate on. PCCA+ would then discover macro-states corresponding to the *aggregated state space*. As long as the aggregation is tied to the underlying spatio-temporal structure of the MDP in the way the non-linear mapping is done, the discovered macro-states (from PCCA+ on the aggregated state sapce) can qualify to be the right abstractions for the original MDP. I will discuss the details in the next section on how the non-linear mapping is learned using deep neural networks and unsupervised learning. Figure 4.7 summarizes the modified pipeline with a flowchart where I retain the previous components in green and indicate the new components added in blue for easier understanding.

As for the second problem of having to perform tabular value learning methods on large problems, we borrow ideas from DQN (Mnih *et al.*, 2015*b*) again. We learn the action and option value functions with function approximation based on deep neural networks. Specifically, for Seaquest, we adopt a similar deep convolution architecture as adopted in (Mnih *et al.*, 2015*b*). The difference from (Mnih *et al.*, 2015*b*) is that we learn value functions for options as well. Once again, due to the better efficiency

of intra-option value learning methods over SMDP value learning methods, we adopt Intra-Option Q Learning (Sutton *et al.*) instead of SMDP Q Learning for driving the learning in the function approximation setting.

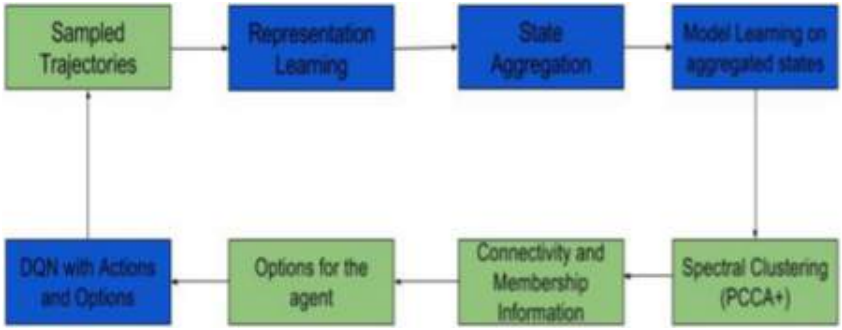


Figure 4.7: Modified Option Generation Pipeline

4.6.6 ODSTC for Large State Spaces

We present the modified pipeline in terms of pseudocode below. We call this modified pipeline as Option Discovery using Spatio-Temporal Clustering (ODSTC) for Large State Spaces. Once again, let the set of options the agent has be denoted by the set O while the new set of options discovered at every iteration o.

Algorithm 3 ODSTC Online Agent Pipeline - Large

1: **while** Not Converged **do**
2: Sample trajectories using current behavioral policy π.
3: Run the trajectory data points $\tau_{i_{i=i}}^{k}$ through an Action Conditional Video Prediction Network.
4: Use the learned representation to aggregate states into *microstates* using K-Means Clustering.
5: Estimate the model $T(s'|s,a)$ on the *microstate* space from sample trajectories $\tau_{i_{i=i}}^{k}$.
6: Operate PCCA+ on the estimated model to derive abstract *macrostates* and memberships.
7: Discover options from the abstract *macrostates* and memberships through hill climbing.
8: Augment agent with new options: $O = O \cup o$.
9: Update value functions Q and behavioral policy π using Intra-Option Q Learning with DQN
10: **end while**

Here are two caveats in the above pipeline though:

- It might be too expensive to perform PCCA+ for large scale problems after every episode simulated by the agent's current policy and hence we could update the skills or options from PCCA+ after a fixed number of episodes depending on the problem complexity.
- Though for small scale problems, the first initial exploration could come from random policies, this wouldn't work for large problems like Seaquest, where an informed exploration with a partially trained Deep Q Network (DQN) is necessary to ensure sufficient portion of the state space is seen for learning a good enough aggregate space for PCCA+ to operate on.

4.7 Playing video games with options

In this section, we delve into the difficulties faced in complex tasks such as learning to play video games due to high dimensionality of the state space and (or) complexity of the task. We ran experiments on the Atari 2600 game, Seaquest to test our proposed model. Atari games such as Seaquest provide a pixel-space representation to the agent

which makes object detection an added task for the agent. We present the model for this task since it is more general with respect to planning based on perception.

4.7.1 Sampling trajectories

Spatial abstraction using PCCA+ is essentially breaking down the MDP into *metastable* states which are densely connected within and have sparse connections to other metastable states. These densely connected regions have low mixing times and hence particles stay in same region of state space for long periods of time without external stimulus (an idea originating from conformal dynamics theory). The important assumption here is that particles are performing random walks. For hard tasks such as playing video games, we cannot sample trajectories where the agent is performing random walks since it would very rarely cross the initial region of the state space due to enemies and obstacles. Hence, we need guided exploration to help the agent cross enemies and obstacles and at the same time, not get restricted to visiting particular parts of the state space if the guiding policy is executed greedily.

We train a Deep Q-network (with primitive actions) as explained in the paper (Mnih *et al.*, 2015*a*). The Deep Q-network (DQN) combines the compositional representation capabilities of convolutional neural networks with the robustness of a Q learning setup, predicting the action-value $Q(s, a)$ for each primitive action a for a state s. We sample trajectories from a partially trained DQN (12.5 million frames) with exploration factor ϵ annealed from 0.5 to 0.3.

Once options are generated based on the initial set of sampled trajectories (from a primitive action based DQN), the agent begins executing options which would lead to unexplored (yet) regions of the state space. Hence, we need to sample trajectories periodically as the agent trains using options to capture the changes in the structure of the transition matrix, and hence, improve our option learning.

4.7.2 Representation Learning

Due to infeasibility in applying the PCCA+ framework directly on the exponential state space, we perform a *state space aggregation* by using K-Means clustering. However,

aggregating directly on the input pixel space ($210 \times 160 \times 3$) in Seaquest is infeasible and no latent concepts (spatial and temporal) are captured in such a space. We are therefroe interested in the transformation of high-dimensional visual input data which are RGB pixel images to a lower-dimensional space using a non-linear transformation. We wanted to capture spatio-temporal information in the latent space representation more effectively. There are multiple objects in a video game screen with objects being controlled directly by the agent's action as well as indirectly (such as enemy sharks entering or leaving the game screen in Seaquest). Thus, there is deep partial observability which needs to be addressed in the learnt representation. This prompted us to incorporate recurrent and convolutional neural networks in order to capture temporal and spatial information from the sequence of images observed into the state representation.

We adopt the work on action-conditional video modelling problem (Oh *et al.*, 2015*b*). In this work, the authors propose and evaluate architectures to generate future frames given the sequence of frames observed conditioned on the agent's action. Inspired by their work, we closely follow their deep neural network architecture, which we refer to as *TemporalRepLearner* that performs the task of an encoder. The architecture involves an LSTM (Hochreiter and Schmidhuber, 1997) layer after the convolutional layers to capture temporal information in the latent space representation. In order to capture the effect of the agent's action on the objects in the game screen, there are multiplicative (element-wise) interactions between the LSTM's hidden representation and the action variable to predict the next frame in the high-level latent representation. The decoder uses deconvolution layers to reconstruct the image for the next frame using the high-level encoder representation. Figure 4.8 presents the architecture adopted by (Oh *et al.*, 2015*b*).

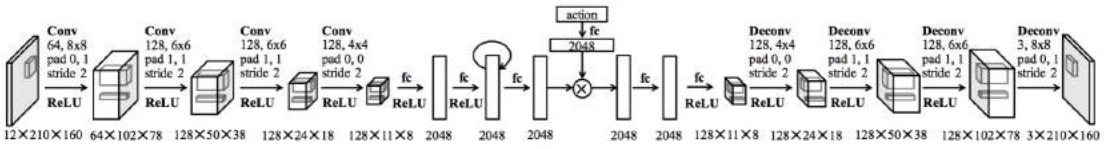


Figure 4.8: Architecture of *TemporalRepLearner*

Since the encoding layers involve both convolution layers as well as an RNN, we refer to it as a recurrent encoding. The input to *TemporalRepLearner* is a frame of the

game for each time-step. There are 4 convolutional layers with rectifier nonlinearity applied after each, followed by a dense layer consisting of 2048 hidden units. The LSTM layer with 2048 hidden units is unrolled through last 11 frames. Thus, there are 2048 factors in the action-conditional transformation. A fully connected layer is then used to construct a 3D feature map from which the 4 deconvolution layers in the decoder finally output a pixel space prediction for the next frame.

The recurrent network is trained using frames partitioned into training and test sets from the trajectories sampled using the trained Deep Q-network. We use the same splits as (Oh *et al.*, 2015*b*). The 2048-dimensional hidden state of the LSTM layer is used as the lower dimensional representation for the state space in Seaquest.

4.7.3 State Aggregation

We observe that in order to model the transition dynamics, we need to perform aggregation of the game states so as to capture the dynamics of an exponential state space in a matrix. This is equivalent to discretizing the state space.

K-Means algorithm, a well-known clustering algorithm has an average complexity of $O(n * k * d * i)$ where n is the number of data points, k is the number of cluster centroids, d is the dimensionality of the space and i is the number of iterations where we begin with a new random initialization of the cluster centroids. This presents a scalability problem for the video game setting - the learnt representation has a high dimensionality (2048) and the number of data points is large owing to the exponential state space. Also in high dimensional spaces, it is very common that for any two data points, there exist at least two dimensions along which the points are far apart from each other. Distance functions that use all the input dimensions are thus not effective and we need feature selection prior to clustering.

We utilize *mini-batch* based K-Means to handle the issue of scalability with respect to number of data points.The main idea proposed in (Sculley, 2010) is to use randomly sampled batches (of a fixed small batch-size) at each iteration. Each *mini-batch* updates the cluster centroids using a convex combination of existing centroids and the mean of the mini-batch data points. With respect to curse of dimensionality, global feature selection techniques are not apt for discovering clusters that exist in different subspaces.

Each dimension could possess information that is relevant to a particular cluster which makes global pruning of features globally inefficient . In order to capture the correlations among the features while clustering, feature selection must be performed locally. (Parsons *et al.*, 2004) propose a soft feature selection procedure in which features are assigned local weights based on correlation among data points in each dimension. This is used to assign to each cluster a weight vector for the dimensions based on the correlation of points in that cluster represented by them.

4.7.4 Model Learning

Post state-aggregation, sampled trajectories of game play (in terms of aggregated micro-states) are used to populate a count matrix which keeps track of the number of transitions seen for every (s, a, s') tuple. That is, if at a particular time step, an agent is at state s, takes action a and moves to state s', we increment the element corresponding to (s, a, s') in the matrix. Note that the states here are not the original states (at pixel level), but the aggregated micro-states found after Deep Representation Learning & K-Means clustering.

We also tried to incorporate the reward structure of the environment into the transition matrix: the idea is that our spatial abstractions should *degenerate* in places where there is a spike in the reward distribution so that our agent interprets a state of high reward (as goal states typically are) as a different abstract state, and naturally composes options that lead the agent to that state. The update equations when we see a transition from state s to state s' on taking action a and obtaining reward r are given by:

$$\phi(s, a, s') = \phi(s, a, s') + 1$$
$$V(s, a, s') = V(s, a, s') + \phi(s, a, s')e^{-\nu|r|}$$

where ν is a regularization parameter

Now, with the transition count matrix V, we estimate the following matrices:

$$D(s, s') = \sum_{a} V(s, a, s')$$
$$T(s, s') = \frac{D(s, s')}{\sum_{s'} D(s, s')}$$
$$P(s, a, s') = \frac{V(s, a, s')}{\sum_{s'} V(s, a, s')}$$

The state-to-state transition matrix T is supplied as input to the spectral clustering algorithm PCCA+ from which options are generated. The probability matrix P is utilized in defining the option policy.

4.7.5 Intra-option Value Learning

In section, we introduced *TemporalRepLearner*, which is a deep neural architecture involving convolutional and recurrent layers in order to encode spatio-temporal history into the learnt representation. We feed the learnt representation as input to a Multi Layer Perceptron (MLP) with one fully connected hidden layer and a sigmoid activation function. The output linear layer has as many units as the sum of number of primitive actions and options. The representation learner layers are frozen while the MLP is trained using stochastic gradient descent.

Training this architecture involves adjusting its parameters at each iteration to minimize a cost function. There hasn't been any work in the past which combines an options framework with a non-linear function approximator. It is to be noted that there are added features to the non-linear function approximator such as an experience replay memory and a separate target Q-network to break correlations among successive updates, thus reducing variance in the updates and avoiding unnecessary feedback loops.

The cost function of the above network is designed such that the parameters θ_i at iteration i are adjusted according to the Intra-option Q learning rule (Sutton *et al.*, 1998). For a single transition sampled from the experience replay, we may need to apply multiple updates to the network. In order to do this in minimal time, we designed a dictionary in order to map a (micro-state, primitive action) pair to the set of options which are consistent with executing the primitive action at that micro-state. This dictionary is

populated once before training begins.

When the agent is at play, primitive actions are used at the emulator level even when an option is being executed. Thus, when transitions are dumped into the experience replay, they consist of only primitive actions. When a transition $(s, a, s^{'}, r)$ is sampled from the experience replay, we compute the micro-state assignment of s - say c_s, lookup the dictionary using the key (c_s, a) to find the set of all options O which are consistent with this primitive action. For the nodes corresponding to each option $o \in O$ in the output layer along with the node for the primitive action a, we apply the Intra-option Q-value update.

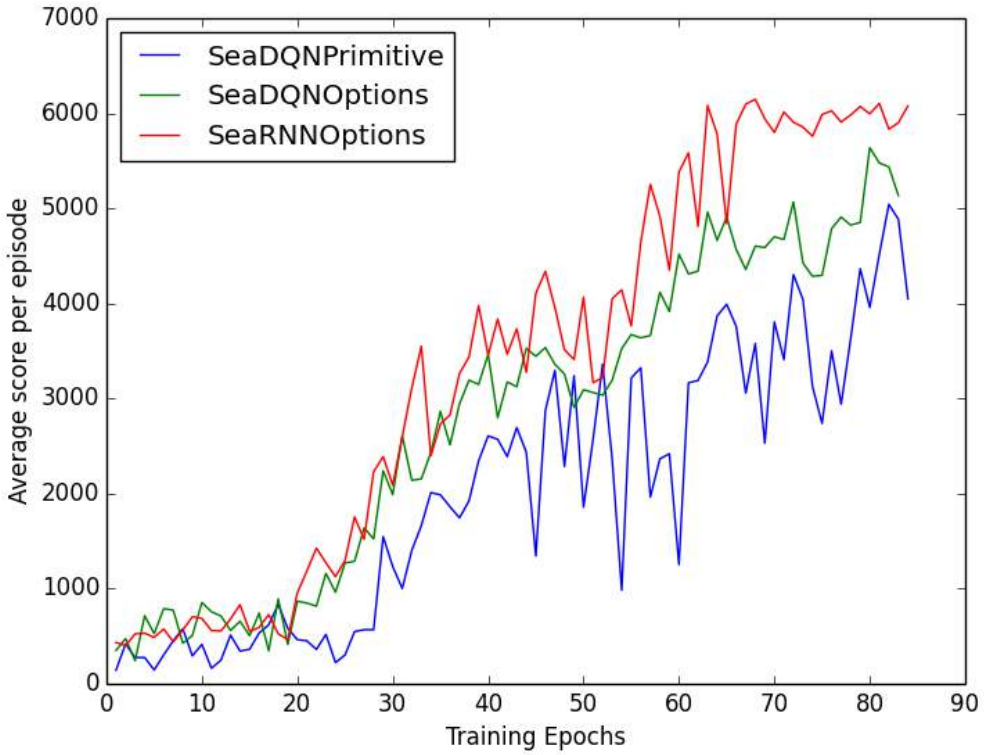


Figure 4.9: Comparing performance of SeaDQNPrimitive, SeaDQNOptions and SeaRNNOptions

Figure 4.9 plots the average score per episode against training epochs. *SeaDQN-Primitive* corresponds to a DQN trained on Seaquest with only primitive actions while *SeaDQNOptions* corresponds to a modified DQN which is trained with the output layer having both primitive actions and the options generated by our approach. *SeaRNNOp-tions* corresponds to the model described in section above where a latent representation

is learnt using *TemporalRepLearner*. Intra-option Q-learning is used in both *SeaDQNOptions* and *SeaRNNOptions*. Figure 4.10 helps to appreciate the semantics of an example option generated in Seaquest. We plot the agent's membership in the abstract state corresponding to the particular option for a random episode's trajectory. The skill learnt here corresponds to resurfacing to replenish oxygen in Seaquest.

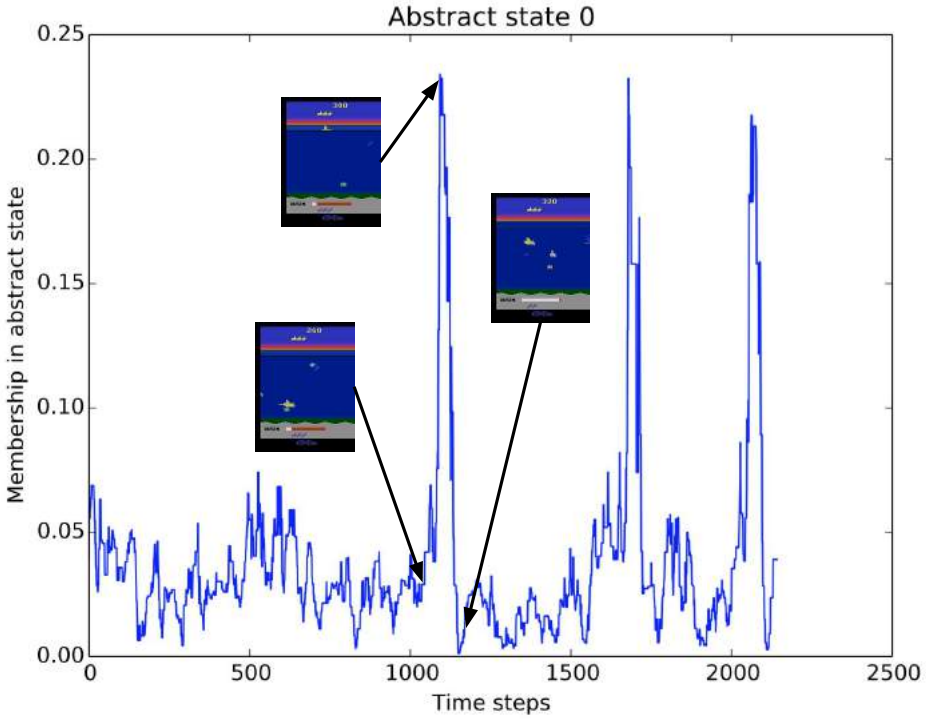


Figure 4.10: Visualization of an option in terms of membership value in corresponding abstract state in Seaquest which corresponds to resurfacing to replenish oxygen

Chapter 5

Attend, Adapt and Transfer: Attentive Deep Architecture for Adaptive Transfer from Multiple Source Tasks

5.1 Abstract

Transferring *knowledge* from prior source tasks in solving a new target task can be useful in several learning applications. The application of transfer poses two serious challenges which have not been adequately addressed. First, the agent should be able to avoid negative transfer, which happens when the transfer hampers or slows down the learning instead of helping it. Second, the agent should be able to selectively transfer, which is the ability to select and transfer from different and multiple source tasks for different parts of the state space of the target task. We propose A2T (Attend, Adapt and Transfer), an attentive deep architecture which adapts and transfers from these source tasks. Our model is generic enough to effect transfer of either policies or value functions. Empirical evaluations on different learning algorithms show that A2T is an effective architecture for transfer by being able to avoid negative transfer while transferring selectively from multiple source tasks in the same domain.

5.2 Introduction

One of the goals of Artificial Intelligence (AI) is to build autonomous agents that can learn and adapt to new environments. Reinforcement Learning (RL) is a key technique for achieving such adaptability. The goal of RL algorithms is to learn an optimal policy for choosing actions that maximize some notion of long term performance. Transferring knowledge gained from tasks solved earlier to solve a new target task can help, either in terms of speeding up the learning process or in terms of achieving a better solution, among other performance measures. When applied to RL, transfer could be

accomplished in many ways (see (Taylor and Stone, 2009, 2011) for a very good survey of the field). One could use the value function from the source task as an initial estimate in the target task to cut down exploration [(Sorg and Singh, 2009)]. Alternatively one could use policies from the source task(s) in the target task. This can take one of two forms - (i) the derived policies can be used as initial exploratory trajectories [(Atkeson and Schaal, 1997; Niekum *et al.*, 2013)] in the target task and (ii) the derived policy could be used to define *macro-actions* which may then be used by the agent in solving the target task [(Mannor *et al.*, 2004; Brunskill and Li, 2014)].

While transfer in RL has been much explored, there are two crucial issues that have not been adequately addressed in the literature. The first is *negative transfer*, which occurs when the transfer results in a performance that is worse when compared to learning from scratch in the target task. This severely limits the applicability of many transfer techniques only to cases for which some measure of relatedness between source and target tasks can be guaranteed beforehand. This brings us to the second problem with transfer, which is the issue of identifying an appropriate source task from which to transfer. In some scenarios, different source tasks might be relevant and useful for different parts of the state space of the target task. As a real world analogy, consider multiple players (experts) who are good at different aspects of a game (say, tennis). For example, Player 1 is good at playing backhand shots while Player 2 is good at playing forehand shots. Consider the case of a new player (agent) who wants to learn tennis by selectively learning from these two experts. We handle such a situation in our architecture by allowing the agent to learn how to pick and use solutions from multiple and different source tasks while solving a target task, selectively applicable for different parts of the state space. We call this *selective transfer*. Our agent can transfer knowledge from Player 1 when required to play backhand shots and Player 2 for playing forehand shots. Further, let us consider consider the situation that both Player 1 and Player 2 are bad at playing drop shots. Apart from the source tasks, we maintain a base network that learns from scratch on the target task. The agent can pick and use the solution of the base network when solving the target task at the parts of the state space where transferring from the source tasks is negative. Such a situation could arise when the source task solutions are irrelevant for solving the target task over a specific portion of the state space, or when the transferring from the source tasks is negative over a specific portion of the state space (for example, transferring the bad drop shot abilities

of Players 1 and 2). This situation also entails the first problem of avoiding negative transfer. Our framework allows an agent to avoid transferring from both Players 1 and 2 while learning to play drop shots, and rather acquire the drop shot skill by learning to use the base network. The architecture is trained such that the base network uses not just the experience obtained through the usage of its solutions in the target task, but the overall experience acquired using the combined knowledge of the source tasks and itself. This enables the base network solutions to get closer to the behavior of the overall architecture (which uses the source task solutions as well). This makes it easier for the base network to assist the architecture to fine tune the useful source task solutions to suit the target task perfectly over time.

The key contribution in the architecture is a *deep **attention** network*, that decides which solutions to attend to, for a given input state. The network learns solutions as a function of current state thereby aiding the agent in adopting different solutions for different parts of the state space in the target task.

To this end, we propose A2T: Attend, Adapt and Transfer, an Attentive Deep Architecture for Adaptive Transfer, that avoids negative transfer while performing selective transfer from multiple source tasks in the same domain. In addition to the tennis example, A2T is a fairly generic framework that can be used to selectively transfer different skills available from different experts as appropriate to the situation. For instance, a household robot can appropriately use skills from different experts for different household chores. This would require the skill to transfer manipulation skills across objects, tasks and robotic actuators. With a well developed attention mechanism, the most appropriate and helpful combination of object-skill-controller can be identified for aiding the learning on a related new task. Further, A2T is generic enough to effect transfer of either action policies or action-value functions, as the case may be. We also adapt different algorithms in Reinforcement Learning as appropriate for the different settings and empirically demonstrate that the A2T is effective for transfer learning for each setting.

5.3 Related Work

As mentioned earlier, transfer learning approaches could deal with transferring policies or value functions. For example, (Banerjee and Stone, 2007) describe a method for

transferring value functions by constructing a *Game tree*. Similarly, (Sorg and Singh, 2009) use the value function from a source task as the initial estimate of the value function in the target task.

Another method to achieve transfer is to reuse policies derived in the source task(s) in the target task. Probabilistic Policy Reuse as discussed in (Fernández and Veloso, 2006) maintains a library of policies and selects a policy based on a similarity metric, or a random policy, or a max-policy from the knowledge obtained. This is different from the proposed approach in that the proposed approach can transfer policies at the granularity of individual states which is not possible in policy-reuse rendering it unable to learn customized policy at that granularity.(Atkeson and Schaal, 1997; Niekum *et al.*, 2013) evaluated the idea of having the transferred policy from the source tasks as explorative policies instead of having a random exploration policy. This provides better exploration behavior provided the tasks are similar. (Talvitie and Singh, 2007) try to find the promising policy from a set of candidate policies that are generated using different action mapping to a single solved task. In contrast, we make use of one or more source tasks to selectively transfer policies at the granularity of state. Apart from policy transfer and value transfer as discussed above, (Ferguson and Mahadevan, 2006) discuss representation transfer using Proto Value Functions.

The idea of negative and selective transfer have been discussed earlier in the literature. For example, (Lazaric and Restelli, 2011) address the issue of negative transfer in transferring samples for a related task in a multi-task setting. (Konidaris *et al.*, 2012) discuss the idea of exploiting shared common features across related tasks. They learn a *shaping function* that can be used in later tasks.

The two recent works that are very relevant to the proposed architecture are discussed in (Parisotto *et al.*, 2015) and (Rusu *et al.*, 2016). (Parisotto *et al.*, 2015) explore transfer learning in RL across Atari games by trying to learn a multi-task network over the source tasks available and directly fine-tune the learned multi-task network on the target task. However, fine-tuning as a transfer paradigm cannot address the issue of negative transfer which they do observe in many of their experiments. (Rusu *et al.*, 2016) try to address the negative transfer issue by proposing a sequential learning mechanism where the filters of the network being learned for an ongoing task are dependent through lateral connections on the lower level filters of the networks learned already for the pre-

vious tasks. The idea is to ensure that dependencies that characterize similarity across tasks could be learned through these lateral connections. Even though they do observe better transfer results than direct fine-tuning, they are still not able to *avoid* negative transfer in some of their experiments.

5.4 Proposed Architecture

Let there be N source tasks and let $K_1, K_2, \ldots K_N$ be the solutions of these source tasks $1, \ldots N$ respectively. Let K_T be the solution that we learn in the target task T. Source tasks refer to tasks that we have already learnt to perform and target task refers to the task that we are interested in learning now. These solutions could be for example policies or state-action values. Here the source tasks should be in the same domain as the target task, having the same state and action spaces. We propose a setting where K_T is learned as a function of $K_1, \ldots, K_N, K_B$, where K_B is the solution of a base network which starts learning from scratch while acting on the target task. In this work, we use a convex combination of the solutions to obtain K_T.

$$K_T(s) = w_{N+1,s} K_B(s) + \sum_{i=1}^{N} w_{i,s} K_i(s) \tag{5.1}$$

$$\sum_{i=1}^{N+1} w_{i,s} = 1, w_{i,s} \in [0, 1] \tag{5.2}$$

$w_{i,s}$ is the weight given to the ith solution at state s.

The agent uses K_T to act in the target task. Figure 5.1a shows the proposed architecture. While the source task solutions $K_1, \ldots, K_N$ remain fixed, the base network solutions are learnt and hence K_B can change over time. There is a central network which learns the weights ($w_{i,s}$, $i \in 1, 2, \ldots, N+1$), given the input state s. We refer to this network as the *attention network*. The $[0, 1]$ weights determine the attention each solution gets allowing the agent to selectively accept or reject the different solutions, depending on the input state. We adopt a *soft-attention* mechanism whereby more than one weight can be non-zero [(Bahdanau *et al.*, 2014)] as opposed to a *hard-attention* mechanism [(Mnih *et al.*, 2014)] where we are forced to have only one non-zero weight.

$$w_{i,s} = \frac{\exp\left(e_{i,s}\right)}{\sum_{j=1}^{N+1} \exp\left(e_{j,s}\right)}, i \in \{1, 2, \ldots, N+1\} \tag{5.3}$$

$$(e_{1,s}, e_{2,s}, \ldots, e_{N+1,s}) = f(s; \theta_a) \tag{5.4}$$

Here, $f(s; \theta_a)$ is a deep neural network (attention network), which could consist of convolution layers and fully connected layers depending on the representation of input. It is parametrised by θ_a and takes as input a state s and outputs a vector of length $N+1$, which gives the attention scores for the $N+1$ solutions at state s. Eq.(5.3) normalises this score to get the weights that follow Eq.(5.2).

If the ith source task solution is useful at state s, then $w_{i,s}$ is set to a high value by the attention network. Working at the granularity of states allows the attention network to attend to different source tasks, for different parts of the state space of the target task, thus giving it the ability to perform selective transfer. For parts of the state space in the target task, where the source task solutions cause negative transfer or where the source task solutions are not relevant, the attention network learns to give high weight to the base network solution (which can be learnt and improved), thus avoiding negative transfer.

Depending on the feedback obtained from the environment upon following K_T, the attention network's parameters θ_a are updated to improve performance.

As mentioned earlier, the source task solutions, $K_1, \ldots, K_N$ remain fixed. Updating these source task's parameters would cause a significant amount of unlearning in the source tasks solutions and result in a weaker transfer, which we observed empirically. This also enables the use of source task solutions, as long as we have the outputs alone, irrespective of how and where they come from.

Even though the agent follows K_T, we update the parameters of the base network that produces K_B, as if the action taken by the agent was based only on K_B. Due to this special way of updating K_B, apart from the experience got through the unique and individual contribution of K_B to K_T in parts of the state space where the source task solutions are not relevant, K_B also uses the valuable experience got by using K_T which uses the solutions of the source tasks as well.

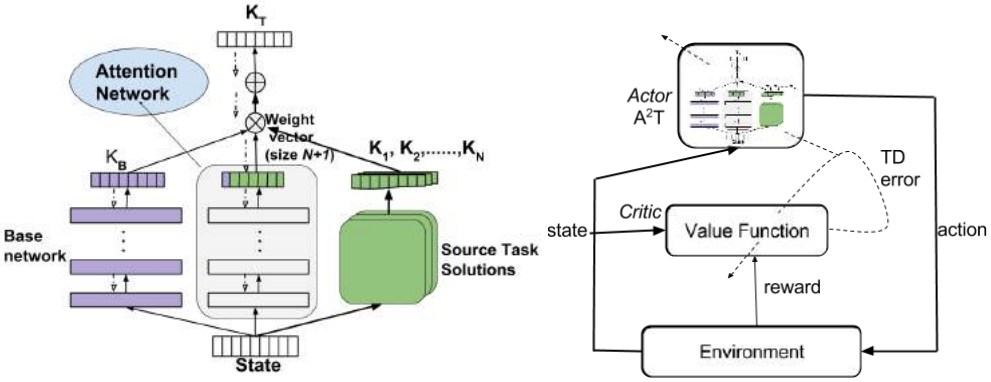


Figure 5.1: (a) A2T architecture. The doted arrows represent the path of back propagation. (b) Actor-Critic using A2T.

This also means that, if there is a source task whose solution K_j is useful for the target task in some parts of its state space, then K_B tries to replicate K_j in those parts of the state space. In practise, the source task solutions though useful, might need to be modified to suit perfectly for the target task. The base network takes care of these modifications required to make the useful source task solutions perfect for the target task. The special way of training the base network assists the architecture in achieving this faster. Note that the agent could follow/use K_j through K_T even when K_B does not attain its replication in the corresponding parts of the state space. This allows for a good performance of the agent in earlier stages training itself, when a useful source task is available and identified.

Since the attention is soft, our model has the flexibility to combine multiple solutions. The use of deep neural networks allow the model to work even for large, complex RL problems. The deep attention network, allows the agent to learn complex selection functions, without worrying about representation issues a priori. To summarise, for a given state, A2T learns to *attend* to specific solutions and *adapts* this attention over different states, hence attaining useful *transfer*. A2T is general and can be used for transfer of solutions such as policy and value.

5.4.1 Policy Transfer

The solutions that we transfer here are the source task policies, taking advantage of which, we learn a policy for the target task. Thus, we have $K_1, \ldots, K_N, K_B, K_T \leftarrow \pi_1, \ldots \pi_N, \pi_B, \pi_T$. Here π represents a stochastic policy, a probability distribution over

all the actions. The agent acts in the target task, by sampling actions from the probability distribution π_T. The target task policy π_T is got as described in Eq.(5.1) and Eq.(5.2). The attention network that produces the weights for the different solutions, is trained by the feedback got after taking action following π_T. The base network that produces π_B is trained as if the sampled action came from π_B (though it originally came from π_T), the implications of which were discussed in the previous section. When the attention network's weight for the policy π_B is high, the mixture policy π_T is dominated by π_B, and the base network learning is nearly on-policy. In the other cases, π_B undergoes off-policy learning. But if look closely, even in the latter case, since π_B moves towards π_T, it tries to be nearly on-policy all the time. Empirically, we observe that π_B converges. This architecture for policy transfer can be used alongside any algorithm that has an explicit representation of the policy. Here we describe two instantiations of A2T for policy transfer, one for direct policy search using REINFORCE algorithm and another in the Actor-Critic setup.

Policy Transfer in REINFORCE Algorithms using A2T:

REINFORCE algorithms [(Williams, 1992)] can be used for direct policy search by making weight adjustments in a direction that lies along the gradient of the expected reinforcement. The full architecture is same as the one shown in Fig.5.1a with $K \leftarrow \pi$. We do direct policy search, and the parameters are updated using REINFORCE. Let the attention network be parametrized by θ_a and the base network which outputs π_B be parametrized by θ_b. The updates are given by:

$$\theta_a \leftarrow \theta_a + \alpha_{\theta_a}(r-b)\frac{\partial \sum_{t=1}^{M} \log(\pi_T(s_t, a_t))}{\partial \theta_a} \tag{5.5}$$

$$\theta_b \leftarrow \theta_b + \alpha_{\theta_b}(r-b)\frac{\partial \sum_{t=1}^{M} \log(\pi_B(s_t, a_t))}{\partial \theta_b} \tag{5.6}$$

where $\alpha_{\theta_a}, \alpha_{\theta_b}$ are non-negative factors, r is the return obtained in the episode, b is some baseline and M is the length of the episode. a_t is the action sampled by the agent at state s_t following π_T. Note that while $\pi_T(s_t, a_t)$ is used in the update of the attention network, $\pi_B(s_t, a_t)$ is used in the update of the base network.

Policy Transfer in Actor-Critic using A2T:

Actor-Critic methods [(Konda and Tsitsiklis, 2000)] are Temporal Difference (TD) methods that have two separate components, *viz.*, an *actor* and a *critic*. The actor proposes a policy whereas the critic estimates the value function to critique the actor's policy. The updates to the actor happens through *TD-error* which is the one step estimation error that helps in reinforcing an agent's behaviour.

We use A2T for the actor part of the Actor-Critic. The architecture is shown in Fig.5.1b. The actor, A2T is aware of all the previous learnt tasks and tries to use those solution policies for its benefit. The critic evaluates the action selection from π_T on the basis of the performance on the target task. With the same notations as REINFORCE for $s_t, a_t, \theta_a, \theta_b, \alpha_{\theta_a}, \alpha_{\theta_b}, \pi_B, \pi_T$; let action a_t dictated by π_T lead the agent to next state s_{t+1} with a reward of r_{t+1} and let $V(s_t)$ represent the value of state s_t and γ the discount factor. Then, the update equations for the actor are as below:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{5.7}$$

$$\theta_a \leftarrow \theta_a + \alpha_{\theta_a} \delta_t \frac{\frac{\partial \log \pi_T(s_t,a_t)}{\partial \theta_a}}{\left| \frac{\partial \log \pi_T(s_t,a_t)}{\partial \theta_a} \right|} \tag{5.8}$$

$$\theta_b \leftarrow \theta_b + \alpha_{\theta_b} \delta_t \frac{\frac{\partial \log \pi_B(s_t,a_t)}{\partial \theta_b}}{\left| \frac{\partial \log \pi_B(s_t,a_t)}{\partial \theta_b} \right|} \tag{5.9}$$

Here, δ_t is TD error. The state-value function V of the critic is learnt using TD learning.

5.4.2 Value Transfer

In this case, the solutions being transferred are the source tasks' action-value functions, which we will call as Q functions. Thus, $K1, \ldots, K_N, K_B, K_T \leftarrow Q_1, \ldots, Q_N, Q_B, Q_T$. Let A represent the discrete action space for the tasks and $Q_i(s) = \{Q(s, a_j) \,\forall\, a_j \in A\}$. The agent acts by using Q_T in the target task, which is got as described in Eq.(5.1) and Eq.(5.2). The attention network and the base network of A2T are updated as described in the architecture.

Value Transfer in Q learning using A2T:

The state-action value Q function is used to guide the agent to selecting the optimal action a at a state s, where $Q(s,a)$ is a measure of the long-term return obtained by taking action a at state s. One way to learn optimal policies for an agent is to estimate the optimal $Q(s,a)$ for the task. Q-learning [(Watkins and Dayan, 1992)] is an off-policy Temporal Difference (TD) learning algorithm that does so. The Q-values are updated iteratively through the Bellman optimality equation [(Puterman, 1994)] with the rewards obtained from the task as below:

$$Q(s,a) \leftarrow \mathbb{E}[r(s,a,s') + \gamma \text{max}_{a'} Q(s',a')]$$

In high dimensional state spaces, it is infeasible to update Q-value for all possible state-action pairs. One way to address this issue is by approximating $Q(s,a)$ through a parametrized function approximator $Q(s,a;\theta)$,thereby generalizing over states and actions by operating on higher level features [(Sutton and Barto, 1998*a*)]. The DQN [(Mnih *et al.*, 2015*b*)] approximates the Q-value function with a deep neural network to be able to predict $Q(s,a)$ over all actions a, for all states s.

The loss function used for learning a Deep Q Network is as below:

$$L(\theta) = \mathbb{E}_{s,a,r,s'}[\big(y^{DQN} - Q(s,a;\theta)\big)^2],$$

with

$$y^{DQN} = \big(r + \gamma \text{max}_{a'} Q(s',a',\theta^-)\big)$$

Here, L represents the expected TD error corresponding to current parameter estimate θ. θ^- represents the parameters of a separate *target network*, while θ represents the parameters of the *online network*. The usage of a *target network* is to improve the stability of the learning updates. The gradient descent step is shown below:

$$\nabla_\theta L(\theta) = \mathbb{E}_{s,a,r,s'}[(y^{DQN} - Q(s,a;\theta))\nabla_\theta Q(s,a)]$$

To avoid correlated updates from learning on the same transitions that the current network simulates, an experience replay [(Lin, 1993)] D (of fixed maximum capacity) is

used, where the experiences are pooled in a FIFO fashion.

We use DQN to learn our experts $Q_i, i \in 1, 2 \ldots N$ on the source tasks. Q-learning is used to ensure $Q_T(s)$ is driven to a good estimate of Q functions for the target task. Taking advantage of the off-policy nature of Q-learning, both Q_B and Q_T can be learned from the experiences gathered by an ϵ-greedy behavioral policy based on Q_T. Let the attention network that outputs w be parametrised by θ_a and the base network outputting Q_B be parametrised by θ_b. Let ${\theta_a}^-$ and ${\theta_b}^-$ represent the parameters of the respective target networks. Note that the usage of *target* here is to signify the parameters (θ_a^-, θ_b^-) used to calculate the *target* value in the Q-learning update and is different from its usage in the context of the *target* task. The updates equations are:

$$y^{Q_T} = (r + \gamma \text{max}_{a'} Q_T(s', a'; {\theta_a}^-, {\theta_b}^-)) \tag{5.10}$$

$$L^{Q_T}(\theta_a, \theta_b) = \mathbb{E}_{s,a,r,s'}[(y^{Q_T} - Q_T(s, a; \theta_a, \theta_b))^2] \tag{5.11}$$

$$L^{Q_B}(\theta_b) = \mathbb{E}_{s,a,r,s'}[(y^{Q_T} - Q_B(s, a; \theta_b))^2] \tag{5.12}$$

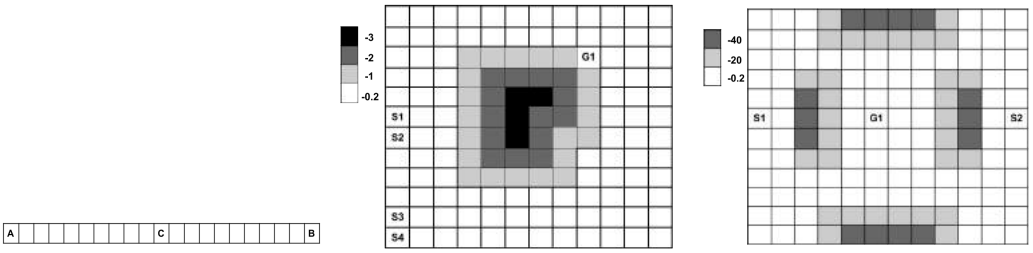
$$\nabla_{\theta_a} L^{Q_T} = \mathbb{E}[(y^{Q_T} - Q_T(s, a)) \nabla_{\theta_a} Q_T(s, a)] \tag{5.13}$$

$$\nabla_{\theta_b} L^{Q_B} = \mathbb{E}[(y^{Q_T} - Q_B(s, a)) \nabla_{\theta_b} Q_R(s, a)] \tag{5.14}$$

θ_a and θ_b are updated with the above gradients using RMSProp. Note that the Q-learning updates for both the attention network (Eq.(5.11)) and the base network (Eq.(5.12)) use the target value generated by Q_T. We use *target* networks for both Q_B and Q_T to stabilize the updates and reduce the non-stationarity as in DQN training. The parameters of the *target* networks are periodically updated to that of the *online* networks.

5.5 Experiments and Discussion

We evaluate the performance of our architecture A2T on policy transfer using two simulated worlds, *viz.*, chain world and puddle world as described below. The main goal of these experiments is to test the consistency of results with the algorithm motivation.
Chain world: Figure 5.2a shows the chain world where the goal of the agent is to go from one point in the chain (starting state) to another point (goal state) in the least number of steps. At each state the agent can choose to either move one position to the left



(a) Chain World (b) Puddle World 1 (c) Puddle World 2

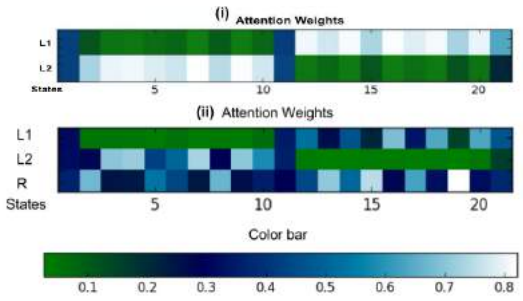
Figure 5.2: Different worlds for policy transfer experiments

or to the right. After reaching the goal state the agent gets a reward that is inversely proportional to the number of steps taken to reach the goal.

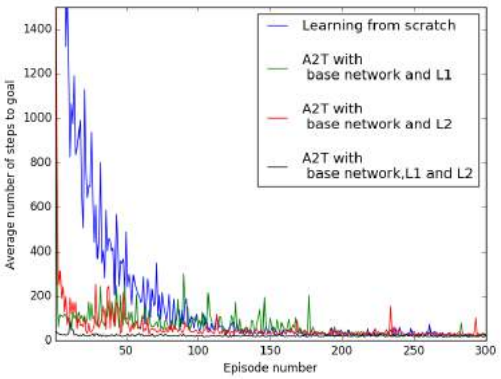
Puddle worlds: Figures 5.2b and 5.2c show the discrete version of the standard puddle world that is widely used in Reinforcement Learning literature. In this world, the goal of the agent is to go from a specified start position to the goal position, maximising its return. At each state the agent can choose one of these four actions: move one position to the north, south, east or west.With 0.9 probability the agent moves in the chosen direction and with 0.1 probability it moves in a random direction irrespective of its choice of action. On reaching the goal state, the agent gets a reward of $+10$. On reaching other parts of the grid the agent gets different penalties as mentioned in the legend of the figures. . We evaluate the performance of our architecture on value transfer using the Arcade Learning Environment (ALE) platform [(Bellemare *et al.*, 2012)]. **Atari 2600:** ALE provides a simulator for Atari 2600 games. This is one of the most commonly used benchmark tasks for Deep Reinforcement Learning algorithms [(Mnih *et al.*, 2015*b*), (Mnih *et al.*, 2016*a*), (Parisotto *et al.*, 2015), (Rusu *et al.*, 2016)]. We perform our adaptive transfer learning experiments on the Atari 2600 game Pong.

5.5.1 Ability to do Selective Transfer

In this section, we consider the case when multiple partially favorable source tasks are available such that each of them can assist the learning process for different parts of the state space of the target task. The objective here is to first show the effectiveness of the attention network in learning to *focus* only on the source task relevant to the state the agent encounters while trying to complete the target task and then evaluating the full architecture with an additional randomly initialised base network.



(a) The weights given by the attention network. Selective transfer in REINFORCE



(b) Selective transfer in Actor-Critic

Figure 5.3: Results of the selective policy transfer experiments

This is illustrated for the Policy Transfer setting using the chain world shown in (Fig. 5.2a). Consider that the target task LT is to start in A or B with uniform probability and reach C in the least number of steps. Now, consider that two learned source tasks, *viz.*, $L1$ and $L2$, are available. $L1$ is the source task where the agent has learned to reach the left end (A) starting from the right end (B). In contrast, $L2$ is the source task where the agent has learned to reach the right end (B) starting from the left end (A). Intuitively, it is clear that the target task should benefit from the policies learnt for tasks $L1$ and $L2$. We learn to solve the task LT using REINFORCE given the policies learned for $L1$ and $L2$. Figure 5.3a (i) shows the weights given by the attention network to the two source task policies for different parts of the state space at the end of learning. We observe that the attention network has learned to ignore $L1$, and $L2$ for the left, and right half of the state space of the target task, respectively. Next, we add base network and evaluate the full architecture on this task. Figure 5.3a (ii) shows the weights given by the attention network to the different source policies for different parts of the state space at the end of learning. We observe that the attention network has learned to ignore $L1$, and $L2$ for the left, and right half of the state space of the target task, respectively. As the base network replicates π_T over time, it has a high weight throughout the state space of the target task.

We also evaluate our architecture in a relatively more complex puddle world shown in Figure 5.2c. In this case, $L1$ is the task of moving from $S1$ to $G1$, and $L2$ is the task of moving from $S2$ to $G1$. In the target task LT, the agent has to learn to move to $G1$ starting from either $S1$ or $S2$ chosen with uniform probability. We learn the

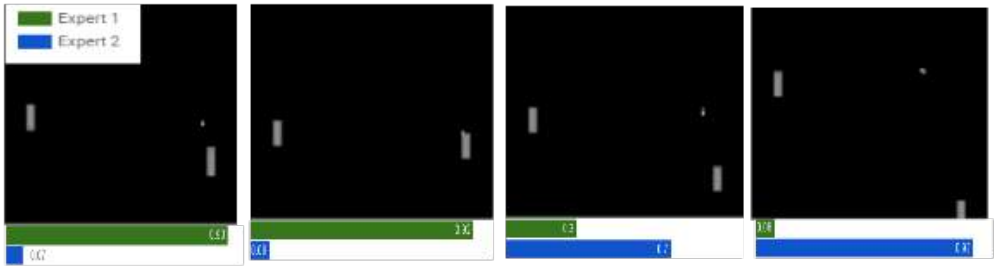


Figure 5.4: Visualisation of the attention weights in the Selective Transfer with Attention Network experiment: Green and Blue bars signify the attention probabilities for Expert-1 ($L1$) and Expert-2 ($L2$) respectively. We see that in the first two snapshots, the ball is in the lower quadrant and as expected, the attention is high on Expert-1, while in the third and fourth snapshots, as the ball bounces back into the upper quadrant, the attention increases on Expert-2.

task LT using Actor-Critic method, where the following are available (i) learned policy for $L1$ (ii) learned policy for $L2$ and (iii) a randomly initialized policy network (the base network). Figure 5.3b shows the performance results. We observe that actor-critic using A2T is able to use the policies learned for $L1$, and $L2$ and performs better than a network learning from scratch without any knowledge of source tasks.

We do a similar evaluation of the attention network, followed by our full architecture for value transfer as well. We create partially useful source tasks through a modification of the Atari 2600 game Pong. We take inspiration from a real world scenario in the sport Tennis, where one could imagine two different right-handed (or left) players with the first being an expert player on the forehand but weak on the backhand, while the second is an expert player on the backhand but weak on the forehand. For someone who is learning to play tennis with the same style (right/left) as the experts, it is easy to follow the forehand expert player whenever he receives a ball on the forehand and follow the backhand expert whenever he receives a ball on the backhand.

We try to simulate this scenario in Pong. The trick is to blur the part of the screen where we want to force the agent to be weak at returning the ball. The blurring we use is to just black out all pixels in the specific region required. To make sure the blurring doesn't contrast with the background, we modify Pong to be played with a black background (pixel value 0) instead of the existing gray (pixel value 87). We construct two partially helpful source task experts $L1$ and $L2$. $L1$ is constructed by training a DQN on Pong with the upper quadrant (the agent's side) blurred, while $L2$ is constructed by training a DQN with the lower quadrant (the agent's side) blurred. This

essentially results in the ball being invisible when it is in the upper quadrant for $L1$ and lower quadrant for $L2$. We therefore expect $L1$ to be useful in guiding to return balls on the lower quadrant, and $L2$ for the upper quadrant. The goal of the attention network is to learn suitable filters and parameters so that it will focus on the correct source task for a specific situation in the game. The source task experts $L1$ and $L2$ scored an average of **9.2** and **8** respectively on Pong game play with black background. With an attention network to suitably weigh the value functions of $L1$ and $L2$, an average performance of **17.2** was recorded just after a single epoch (250,000 frames) of training. (The score in Pong is in the range of $[-21, 21]$). This clearly shows that the attention mechanism has learned to take advantage of the experts adaptively. Fig. 5.4 shows a visualisation of the attention weights for the same.

We then evaluate our full architecture (A2T) in this setting, i.e with an addition of DQN learning from scratch (base network) to the above setting. The architecture can take advantage of the knowledge of the source task experts selectively early on during the training while using the expertise of the base network wherever required, to perform well on the target task. Figure 5.5 summarizes the results, where it is clear that learning with both the partially useful experts is better than learning with only one of them which in turn is better than learning from scratch without any additional knowledge.

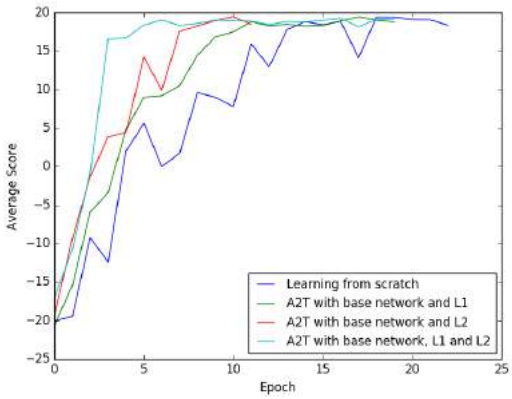


Figure 5.5: Selective Value Transfer.

5.5.2 Ability to Avoid Negative Transfer and Ability to Transfer from Favorable Task

We first consider the case when only one learned source task is available such that its solution K_1 (policy or value) can hamper the learning process of the new target task. We refer to such a source task as an unfavorable source task. In such a scenario, the attention network shown in Figure 5.1a should learn to assign a very low weight (ignore) to K_1 . We also consider a modification of this setting by adding another source task whose

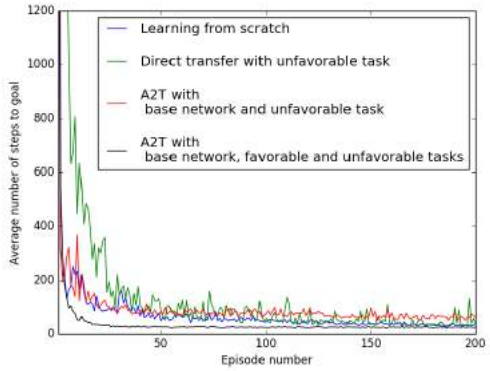
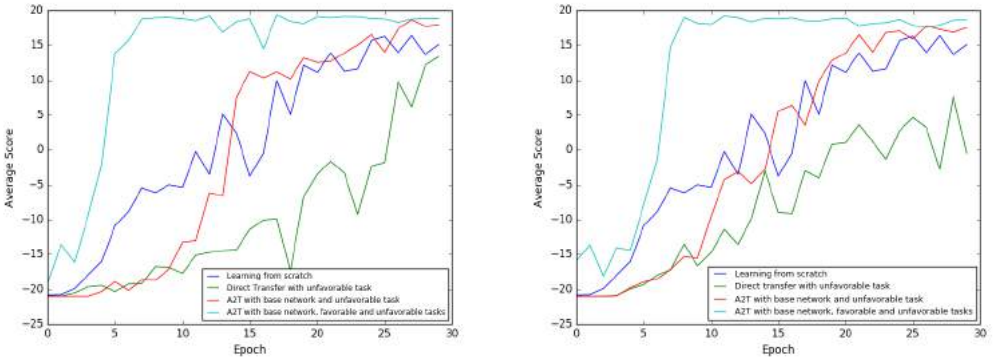


Figure 5.6: Avoiding negative transfer and transferring policy from a favorable task.

solution K_2 is favorable to the target task. In such a scenario, the attention network should learn to assign high weight (attend) to K_2 while ignoring K_1.

We now define an experiment using the puddle world from Figure 5.2b for policy transfer. The target task in our experiment is to maximize the return in reaching the goal state $G1$ starting from any one of the states $S1, S2, S3, S4$. We artificially construct an unfavorable source task by first learning to solve the above task and then negating the weights of the topmost layer of the actor network. We then add a favorable task to the above setting. We artificially construct a favorable source task simply by learning to solve the target task and using the learned actor network. Figure 5.6 shows the results. The target task for the value transfer experiment is to reach expert level performance on Pong. We construct two kinds of unfavorable source tasks for this experiment. **Inverse-Pong**: A DQN on Pong trained with negated reward functions, that is with $R'(s, a) = -R(s, a)$ where $R(s, a)$ is the reward provided by the ALE emulator for choosing action a at state s. **Freeway**: An expert DQN on another Atari 2600 game, Freeway, which has the same range of optimal value functions and same action space as Pong. We empirically verified that the Freeway expert DQN leads to negative transfer when directly initialized and fine-tuned on Pong which makes this a good proxy for a negative source task expert even though the target task Pong has a different state space. We artificially construct a favorable source task by learning a DQN to achieve expertise on the target task (Pong) and use the learned network. Figure 5.7a compares the performance of the various scenarios when the unfavorable source task is Inverse-Pong, while Figure 5.7b offers a similar comparison with the negative expert being Freeway.

From all the above results, we can clearly see that A2T does not get hampered by the unfavorable source task by learning to ignore the same and performs competitively



(a) Avoiding negative transfer(Pong) and transfer-ring from a favorable task

(b) Avoiding negative transfer(Freeway) and transferring from a favorable task

Figure 5.7: Avoiding negative transfer and transferring value from a favorable task. Specific training and architecture details are mentioned in APPENDIX.

with just a randomly initialized learning on the target task without any expert available. Secondly, in the presence of an additional source task that is favorable, A2T learns to transfer useful knowledge from the same while ignoring the unfavorable task, thereby reaching expertise on the target task much faster than the other scenarios.

5.6 Details of the Network Architecture in Value Transfer Experiments

For the source task expert DQNs, we use the same architecture as (Mnih *et al.*, 2015*b*) where the input is $84 \times 84 \times 4$ with 32 convolution filters, dimensions 8×8, stride 4×4 followed by 64 convolution filters with dimensions 4×4 and stride 2×2, again followed by 64 convolution filters of size 3×3 and stride 1×1. This is then followed by a fully connected layer of 512 units and finally by a fully connected output layer with as many units as the number of actions in Pong (Freeway) which is 3. We use ReLU nonlinearity in all the hidden layers.

With respect to the A2T framework architecture, we have experimented with two possible architectures:

- The base and attention networks following the NIPS architecture of (Mnih *et al.*, 2013) except that the output layer is softmax for the attention network.
- The base and attention networks following the Nature architecture of (Mnih *et al.*, 2015*b*) with a softmax output layer for the attention network.

Specifically, the NIPS architecture of (Mnih *et al.*, 2013) takes in a batch of $84 \times 84 \times 4$ inputs, followed by 16 convolution filters of dimensions 8×8 with stride 4×4, 32 convolution filters with dimensions 4×4 and stride 2×2, a fully connected hidden layer of 256 units, followed by the output layer. For the Selective Transfer with Blurring experiments described in Section 4.1, we use the second option above. For the other experiments in Section 4.2 and the additional experiments in Appendix, we use the first option. The attention network has $N+1$ outputs where N is the number of source tasks.

5.7 Training Details

5.7.1 Training Algorithm

For all our experiments in Value Transfer, we used RMSProp as in (Mnih *et al.*, 2015*b*) for updating gradient. For Policy Transfer, since the tasks were simple, stochastic gradient descent was sufficient to provide stable updates. We also use reward clipping, target networks and experience replay for our value transfer experiments in exactly the same way (all hyper parameters retained) as (Mnih *et al.*, 2015*b*). A training epoch is 250,000 frames and for each training epoch, we evaluate the networks with a testing epoch that lasts 125,000 frames. We report the average score over the completed episodes for each testing epoch. The average scores obtained this way are averaged over 2 runs with different random seeds. In the testing epochs, we use $\epsilon = 0.05$ in the ϵ-greedy policy.

5.7.2 Learning Rate

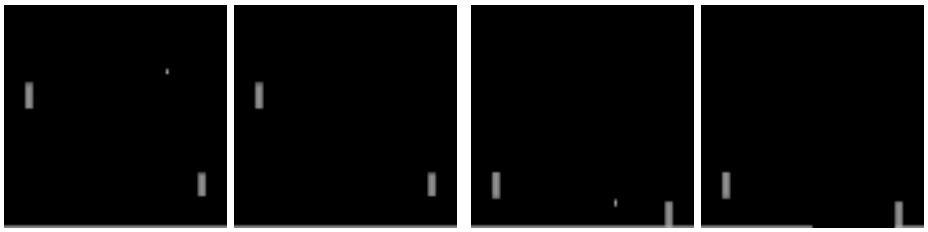
In all our experiments, we trained the architecture using the learning rates, 0.0025 and 0.0005. In general, the lower learning rate provided more stable (less variance) training curves. While comparing across algorithms, we picked the best performing learning rate out of the two (0.0025 and 0.0005) for each training curve.

5.8 Blurring Experiments on Pong

The experts are trained with blurring (hiding the ball) and black background as illustrated in Figure 5.8. Therefore, to compare the learning with that of a random network without any additional knowledge, we ran the baseline DQN on Pong with a black background too. Having a black background provides a rich contrast between the white ball and the black background, thereby making training easier and faster, which is why the performance curves in that setting are different to the other two settings reported for Inverse Pong and Freeway Negative transfer experiments where no blacking is done and Pong is played with a gray background. The blurring mechanism in Pong is illustrated in the next section.

5.9 Blurring Mechanism in Pong - Details

The figures in Figure 5.8 explain the blurring mechanism for selective transfer experiments on Pong. The background of the screen is made black. Let X (84×84) denote an array containing the pixels of the screen. The paddle controlled by the agent is the one on the right. We focus on the two quadrants $X1 = X[: 42, 42 :]$ and $X2 = X[42 :, 42 :]$ of the Pong screen relevant to the agent controlled paddle. To simulate an expert that is weak at returning balls in the upper quadrant, the portion of $X1$ till the horizontal location of agent-paddle, ie $X1[:, : 31]$ is blacked out, while similarly, for simulating weakness in the bottom quadrant, we blur the portion of $X2$ till the agent-paddle's horizontal location, ie $X2[:, : 31] = 0$. Figures 5.8a and 5.8b illustrate the scenarios of blurring the upper quadrant before and after blurring; and similarly do 5.8c and 5.8d for blurring the lower quadrant. Effectively, blurring this way with a black screen is equivalent to hiding the ball (white pixel) in the appropriate quadrant where weakness is to be simulated. Hence, Figures 5.8b and 5.8d are the mechanisms used while training a DQN on Pong to hide the ball at the respective quadrants, so to create the partially useful experts which are analogous to forehand-backhand experts in Tennis. $X[: a, : b]$ indicates the subarray of X with all rows upto row index a and all columns upto column index b.



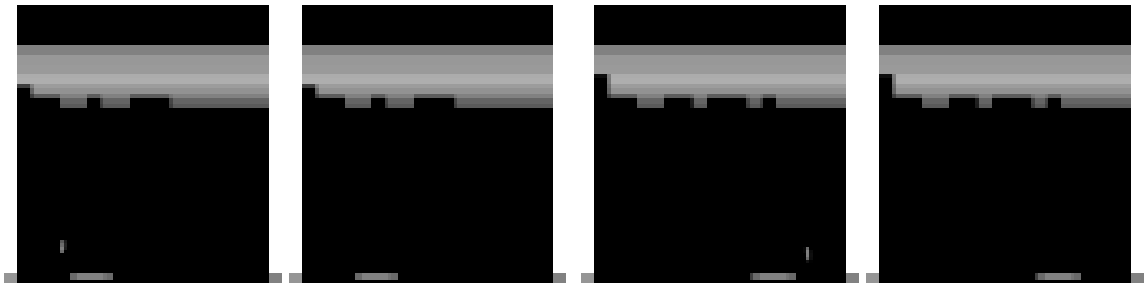
(a) Ball in upper quadrant (b) Blurring the upper quadrant (c) Ball in lower quadrant (d) Blurring the lower quadrant

Figure 5.8: Figures supplementing the explanation in the above section

5.10 Blurring experiments on Breakout

Similar to our Blurring experiment on Pong, we additionally ran another experiment on the Atari 2600 game, Breakout, to validate the efficiency of our attention mechanism. We consider a setup with two experts $L1$ and $L2$ along with our attention network. The experts $L1$ and $L2$ were trained by blurring the lower left and right quadrants of the breakout screen respectively. We don't have to make the background black like in the case of Pong because the background is already black in Breakout and direct blurring is sufficient to hiding the ball in the respective regions without any contrasts introduced. We blur only the lower part so as to make it easy for the agent to at least anticipate the ball based on the movement at the top. We empirically observed that blurring the top half (as well) makes it hard to learn any meaningful partially useful experts $L1$ and $L2$.

The goal of this experiment is to show that the attention network can learn suitable filters so as to dynamically adapt and learn to select the expert appropriate to the situation (game screen) in the task. The expert $L1$ which was blurred on the left bottom half is bound to weak at returning balls on that region while $L2$ is expected to be weak on the right. This is in the same vein as the forehand-backhand example in Tennis and its synthetic simulation for Pong by blurring the upper and lower quadrants. During game play, the attention mechanism is expected to ignore $L2$ when the ball is on the bottom right half (while focusing on $L1$) and similarly ignore $L2$ (while focusing on $L1$) when the ball is on the left bottom half. We learn experts $L1$ and $L2$ which score **42.2** and **39.8** respectively. Using the attention mechanism to select the correct expert, we were able to achieve a score of **94.5** after training for 5 epochs. Each training epoch corresponds to $250,000$ decision steps, while the scores are averaged over completed episodes run



(a) Ball in lower left quadrant (b) Blurring the lower left quadrant (c) Ball in lower right quadrant (d) Blurring the lower right quadrant

Figure 5.9

for $125,000$ decision steps. This shows that the attention mechanism learns to select the suitable expert. Though the performance is limited by the weaknesses of the respective experts, our goal is to show that the attention paradigm is able to take advantage of both experts appropriately. This is evident from the scores achieved by standalone experts and the attention mechanism. Additionally, we also present a visualization of the attention mechanism weights assigned to the experts $L1$ and $L2$ during game play in Section 5.12. The weights assigned are in agreement with what we expect in terms of selective attention. The blurring mechanism is visually illustrated in the next section.

5.11 Blurring Mechanism in Breakout - Details

The figures in Figure 5.9 explain the blurring mechanism used for selective transfer experiments on Breakout. The background the screen is already black. Let X (84×84) denote an array containing the pixels of the screen. The paddle controlled by the agent is the one on the right. We focus on the two quadrants $X1 = X[31:81, 4:42]$ and $X2 = X[31:81, 42:80]$. We perform blurring in each case by ensuring $X1 = 0$ and $X2 = 0$ for all pixels within them for training $L1$ and $L2$ respectively. Effectively, this is equivalent to hiding the ball in the appropriate quadrants. Blurring $X1$ simulates weakness in the lower left quadrant, while blurring $X2$ simulates weakness in the lower right quadrant. We don't blur all the way down upto the last row to ensure the paddle controlled by the agent is visible on the screen. We also don't black the rectangular border with a width of 4 pixels surrounding the screen. Figures 5.9a and 5.9b illustrate the scenarios of blurring the lower left quadrant before and after blurring; and similarly do 5.9c and 5.9d for blurring the lower right quadrant.

5.12 Blurring Attention Visualization on Breakout

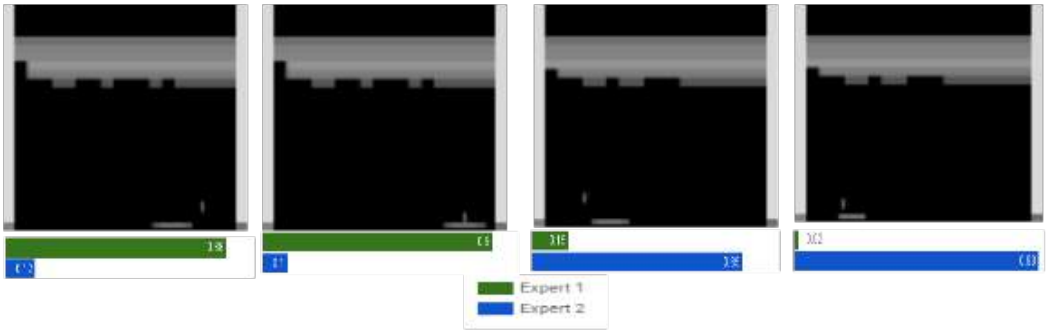


Figure 5.10: Visualisation of the attention weights in the Selective Transfer with Attention for Breakout: Green and Blue bars signify the attention probabilities for Expert-1 ($L1$) and Expert-2 ($L2$) respectively on a scale of $[0, 1]$. We see that in the first two snapshots, the ball is in the lower right quadrant and as expected, the attention is high on Expert-1, while in the third and fourth snapshots, the ball is in the lower right quadrant and hence the attention is high on Expert-2.

5.13 Evolution of Attention Weights Visualization

We present the evolution of attention weights for the experiment described earlier where we focus on the efficacy of the A2T framework in providing an agent the ability to *avoid negative transfer* and *transfer from a favorable source task*. Figure 5.11 depicts the evolution of the attention weights (normalised in the range of $[0, 1]$) during the training of the A2T framework. The corresponding experiment is the case where the target task is to solve Pong, while there are two source task experts, one being a perfect Pong playing trained DQN (to serve as positive expert), and the other being the *Inverse-Pong* DQN trained with negated reward functions (to serve as negative expert). Additionally, there's also the base network that learns from scratch using the experience gathered by the attentively combined behavioral policy from the expert networks, the base network and itself. We train the framework for 30 epochs, and the plot illustrates the attention weights every second epoch. We clearly see that there is no weird co-adaptation that happens in the training, and the attention on the negative expert is uniformly low throughout. Initially, the framework needs to collect some level of experience to figure out that the positive expert is optimal (or close to optimal). Till then, the attention is mostly on the base network, which is learning from scratch. The attention then shifts to the positive expert which in turn provides more rewarding episodes and transition

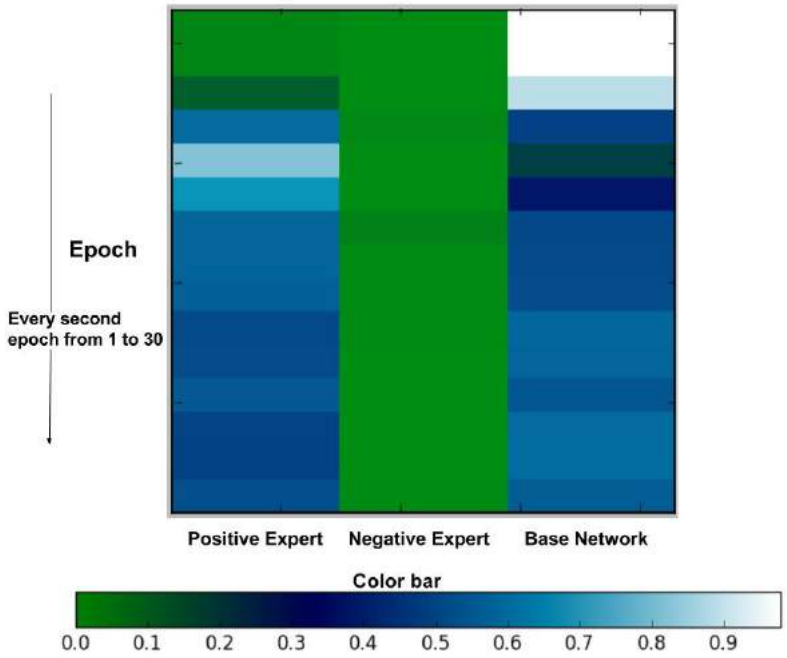


Figure 5.11: Visualization of the attention weights with one positive and one negative expert

tuples to learn from. Finally, the attention drifts slowly to the base network from the positive expert again, after which the attention is roughly random in choosing between the execution of positive expert and the base network. This is because the base network has acquired sufficient expertise as the positive expert which happens to be optimal for the target task. This visualization clearly shows that A2T is a powerful framework in ignoring a negative expert throughout and using a positive expert appropriately to learn quickly from the experience gathered and acquire sufficient expertise on the target task.

5.14 Additional Experiment with Partial Positive Expert

In our experiments so far that dealt with the prevention of negative transfer and using a favorable source task, we consider the positive expert as a perfect (close to optimal) expert on the same task we treat as the target task. This raises the question of relying on the presence of a perfect expert as a positive expert. If we have such a situation, the obvious solution is to execute each of the experts on the target task and vote for them with probabilities proportional to the average performance of each. The A2T framework is however generic and not intended to just do *source task selection*. We illustrate this

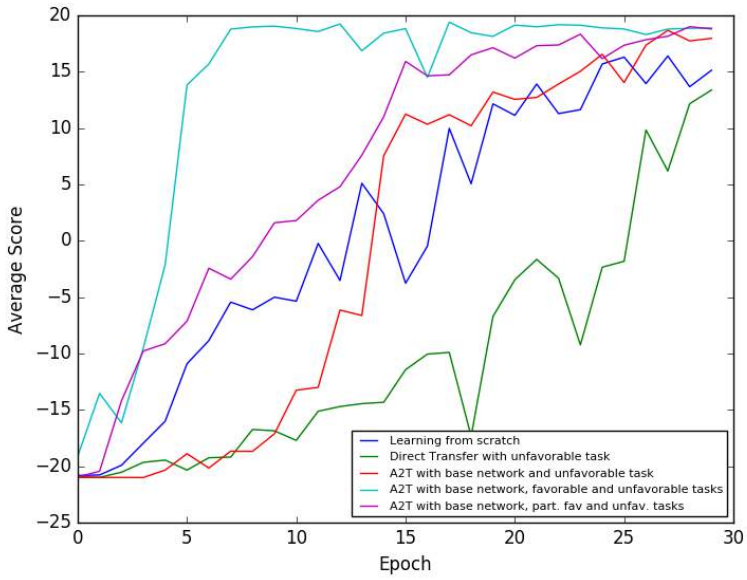


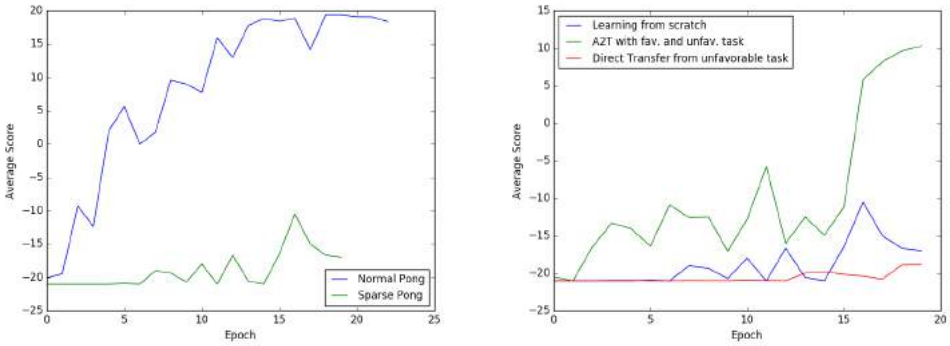
Figure 5.12: Learning curves with partial positive expert

with an additional baseline experiment, where the positive source task is an *imperfect expert on the target task*. In such a case, just having a weighted average voting among the available source task networks based on their individual average rewards is upper bounded by the performance of the best available positive expert, which happens to be an imperfect expert on the target task. Rather, the base network has to acquire new skills not present in the source task networks. We choose a partially trained network on Pong, that scores an average of 8 (max: 21). The graph in Figure 5.12 clearly shows that the A2T framework with a partial Pong expert and a negative expert performs better than i) learning from scratch, ii) A2T with only one negative expert, and performs worse than A2T with one *perfect* positive expert and one negative expert. This is expected because a partial expert cannot provide as much of expert knowledge as a perfect expert, but still provides some useful knowledge in speeding the process of solving the target task. An important conclusion from this experiment is that the A2T framework is capable of discovering new skills not available among any of the experts when such skills are required for optimally solving the target task. To maintain consistency, we perform the same number of runs for averaging scores and experimented with both learning rates and pick the better performing one (0.00025).

5.15 Case study of target task performance limited by data availability

This experiment is a case study on a target task where the performance is limited by data availability. So far, we focused on experiments where the target task is to solve Pong (normal or black background) for Value Transfer, and Puddle Worlds for Policy Transfer. In both these cases, a randomly initialized value (or policy) network learning without the aid of any expert network is able to solve the target task within a reasonable number of epochs (or iterations). We want to illustrate a case where solving the target task in reasonable time is hard and the presence of a favorable source task significantly impacts the speed of learning. To do so, we consider a variant of Pong as our target task. In this variant, only a small probability ρ of transition tuples (s, a, r, s') with *non-zero reward* r are added to the Replay Memory (and used for learning through random batch sampling). This way, the performance on the target task is limited by the availability of rewarding (positive or negative) transitions in the replay memory. This synthetically makes the target task of Pong a sparse reward problem because the replay memory is largely filled with transition tuples that have zero reward. We do not use any prioritized sampling so as to make sure the sparsity has a negative effect on learning to solve the target task. We use a version of Pong with black background (as used in Section 4.1 for the Blurring experiments) for faster experimentation. $\rho = 0.1$ was used for the plots illustrated above. Figure 5.13a clearly shows the difference between a normal Pong task without any synthetic sparsity and the new variant we introduce. The learning is much slower and is clearly limited by data availability even after 20 epochs (20 million frames) due to reward sparsity. Figure 5.13b describes a comparison between the A2T setting with one positive expert which expertly solves the target task and one negative expert, learning from scratch, and direct fine-tuning on a negative expert. We clearly see the effect of having the positive expert in one of the source tasks speeding up the learning process significantly when compared to learning from scratch, and also see that fine-tuning on top of a negative expert severely limits learning even after 20 epochs of training. We also see that the A2T framework is powerful to work in sparse reward settings and avoids negative transfer even in such cases, while also clearly learning to benefit from the presence of a target task expert among the source task networks. Importantly, this experiment demonstrates that transfer learning has a significant effect on

tasks which may be hard (infeasible to solve within a reasonable training time) without any expert available. Further, A2T is also beneficial for such (sparse reward) situations when accessing the weights of an expert network is not possible, and only outputs of the expert (policy or value-function) can be used. Such synthetic sparse variants of existing tasks is a good way to explore future directions in the intersection of Inverse Reinforcement Learning and Reward-Based Learning, with A2T providing a viable framework for off-policy and on-policy learning.



(a) Comparison of Sparse Pong to Normal Pong (b) A2T with a positive and negative expert

Figure 5.13

Chapter 6

Conclusions and Future Work

In this chapter, I summarize the contents of each chapter and touch upon possible avenues of future research in each of them.

6.1 Dynamic Action Repetition for Deep Reinforcement Learning

In this chapter, we introduced a novel paradigm to strengthen existing Deep Reinforcement Learning agents by providing them the ability to decide the time scale of executing an action in addition to deciding the specific action to execute. This scheme enables AI agents to dynamically decide how long a chosen action in the current state is to be repeated. The ability to decide dynamically, the extent of the repetition of an action can be seen as a skill in the direction of looking ahead (planning). Our scheme allows the agent to exert quick reflexes when required and to continue performing the same action as long as the chosen *macro-action* leads the agent to an advantageous (*valuable*) next state. Through this paradigm, we present an elegant way to introduce temporal structure in discrete-action space policies with the extent of repetition of the chosen action being decided based on the current state. We show empirically that this setup leads to significant improvement in performance, regardless of the underlying Deep Reinforcement Learning algorithm used, and regardless of the nature of the algorithm (off-policy or on-policy), with results on five relatively harder Atari 2600 domain games: Seaquest, Space Invaders, Alien, Enduro and Q*Bert. The improvement in performance has been achieved without much tuning of the DQN (Mnih *et al.*, 2015*a*) network parameters. In the case of Augmented A3C, *no* tuning of hyperparameters was performed. The dynamic time scale mechanism can be incorporated into any existing Deep Reinforcement Learning methods (both value and policy based) and an exhaustive analysis on its utility for different methods is an interesting direction for future work

Our work naturally leads to a parametrized policy setup, where each action has an associated parameter: its Action Repetition Rate (ARR). An Actor Critic setup similar to (Mnih *et al.*, 2016*b*) to learn such structured policies with multiple time scales for both discrete and continuous action spaces is a compelling future direction. The structure in the policy naturally introduces temporal abstractions through *macro-actions* composed of the same action being repeated, with different lengths. (Sharma *et al.*, 2017) provide a thorough empirical analysis of this setup on several Atari games and a few continuous control tasks. Another direction to deal with action space explosion when considering more time scales is to investigate the use of action embeddings for value-based methods. In our setup, we do not consider the ability to stop executing a macro-action that the agent has committed to. However, this is a necessary skill in the event of unexpected changes in the environment while executing a chosen macro-action. Thus, *stop* and *start* actions for exiting and committing to plans can be augmented with the dynamic time scale setup for more robust planning.

6.2 Option Discovery using Spatio-Temporal Clustering

In this chapter, we presented another paradigm to incorporate structure in policy learning for Deep Reinforcement Learning agents through abstractions derived from metastability. This was an extension of (Kumar *et al.*, 2012) to be scalable for larger state spaces. Scaling approaches based on spectral connectivity like PCCA+ or other graph based approaches that rely on knowing the structure of the graph in entirety to operate, is not an easy problem. Our approach is heuristic and exploits the expressiveness in the representations uncovered by deep neural networks driven by self-supervised learning like video prediction models. As future work, we wish to investigate the options this pipeline discovers on more tasks which require planning such as continuous control with sparse rewards and hierarchies and more visual planning tasks. One negative about the experiments is that we relied on demonstrations from a partially trained expert on the task to learn our abstractions and options. Thus, this is not a perfect autonomous way of doing hierarchical Reinforcement Learning. However, innovative forms of self-supervised learning to provide reasonable base policies or representations to start with

can help avoid the dependence on an expert. A more interesting and deeper question is to make this framework end-to-end and differentiable instead of having the representation learning and option discovery parts of the pipeline decoupled as is now. This is a hard problem since it is not very clear how to make clustering approaches end-to-end in machine learning. Exploring other ideas in hierarchical Reinforcement Learning with function approximation maybe more viable in that case.

6.3 Attend, Adapt and Transfer

In this chapter we presented a very general deep neural network architecture, A2T for transfer learning that avoids negative transfer while enabling selective transfer from multiple source tasks in the same domain. We show simple ways of using A2T for policy transfer and value transfer. We empirically evaluate its performance with different algorithms, using simulated worlds and games, and show that it indeed achieves its stated goals. Apart from transferring task solutions, A2T can also be used for transferring other useful knowledge such as the model.

While in this work we focused on transfer between tasks that share the same state and action spaces and are in the same domain, the use of deep networks opens up the possibility of going beyond this setting. For example, a deep neural network can be used to learn common representations (Parisotto *et al.*, 2015) for multiple tasks thereby enabling transfer between related tasks that could possibly have different state-action spaces. A hierarchical attention over the lower level filters across source task networks while learning the filters for the target task network is another natural extension to transfer across tasks with different state-action spaces. We would also like to explore this setting in avoiding negative transfer in continuous control tasks since negative transfer has practical importance in Robotics.

The nature of tasks considered in our experiments is very naturally connected to Hierarchical Reinforcement Learning and Continual Learning. For instance, the blurring experiments inspired from Tennis based on experts for specific skills like Forehand and Backhand could be considered as learning sub-goals (program modules) like Forehand and Backhand, and learning to solve a more complex and broader task like Tennis by invoking relevant sub-goals (program modules). This could be very useful to build a

household robot for general navigation and manipulation, by invoking specific skills such as manipulation of different objects, navigating across different source-destination points, etc. The A2T framework will be a powerful tool in using Continual Learning for worlds where a wide variety of sub-goals are present and re-used. Additionally, we believe that this framework is an important tool to build upon and apply for source-domain adaptation, where we could selectively decide to pick information of the source domain (model) in addition to using the solutions of the source tasks while learning to solve the target task. This way, model based approaches could take advantage of the model information from the source tasks in addition to the source task solutions (policies or value functions). Over all, A2T is a novel way to approach transfer learning that opens up many new avenues of research in this area.

Bibliography

1. **Atkeson, C. G.** and **S. Schaal**, Robot learning from demonstration. *In In Proceedings of International Conference on Machine Learning*, volume 97. 1997.

2. **Bahdanau, D.**, **K. Cho**, and **Y. Bengio** (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

3. **Banerjee, B.** and **P. Stone**, General game learning using knowledge transfer. *In In The 20th International Joint Conference on Artificial Intelligence*. 2007.

4. **Bellemare, M. G.**, **Y. Naddaf**, **J. Veness**, and **M. Bowling** (2012). The arcade learning environment: An evaluation platform for general agents. *arXiv preprint arXiv:1207.4708*.

5. **Bellemare, M. G.**, **Y. Naddaf**, **J. Veness**, and **M. Bowling** (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, **47**, 253–279.

6. **Braylan, A.**, **M. Hollenbeck**, **E. Meyerson**, and **R. Miikkulainen** (2015). Frame skip is a powerful parameter for learning to play atari. *AAAI workshop*.

7. **Brunskill, E.** and **L. Li**, Pac-inspired option discovery in lifelong reinforcement learning. *In Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. 2014.

8. **Cai, D.**, **X. He**, and **J. Han** (2005). Document clustering using locality preserving indexing. *IEEE Transactions on Knowledge and Data Engineering*, **17**(12), 1624–1637.

9. **Ferguson, K.** and **S. Mahadevan** (2006). Proto-transfer learning in markov decision processes using spectral methods. *Computer Science Department Faculty Publication Series*, 151.

10. **Fernández, F.** and **M. Veloso**, Probabilistic policy reuse in a reinforcement learning agent. *In Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM, 2006.

11. **Finney, S.**, **N. H. Gardiol**, **L. P. Kaelbling**, and **T. Oates**, The thing that we tried didn't work very well: deictic representation in reinforcement learning. *In Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 2002.

12. **Gilbert, D. T.** and **T. D. Wilson** (2007). Prospection: Experiencing the future. *Science*, **317**(5843), 1351–1354.

13. **Gomez, F.** and **R. Miikkulainen** (1997). Incremental evolution of complex general behavior. *Adaptive Behavior*, **5**(3-4), 317–342.

14. **Goodfellow, I.**, **Y. Bengio**, and **A. Courville** (2016). Deep learning.

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

16. **Hausknecht, M.** and **P. Stone** (2016). Deep reinforcement learning in parametrized action space. *4th International Conference on Learning Representations*.

17. **Hengst, B.**, Model approximation for hexq hierarchical reinforcement learning. *In European Conference on Machine Learning*. Springer, 2004.

18. **Hingston, P.** (2010). A new design for a turing test for bots. *IEEE Conference on Computational Intelligence and Games (CIG)*.

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

20. **Konda, V.** and **J. Tsitsiklis**, Actor-critic algorithms. *In SIAM Journal on Control and Optimization*. MIT Press, 2000.

21. **Konda, V. R.** and **J. N. Tsitsiklis** (2003). On actor-critic algorithms. *SIAM journal on Control and Optimization*, **42**(4), 1143–1166.

22. **Konidaris, G.**, **S. Kuindersma**, **R. A. Grupen**, and **A. G. Barto**, Autonomous skill acquisition on a mobile manipulator. 2011.

23. **Konidaris, G.**, **I. Scheidwasser**, and **A. G. Barto** (2012). Transfer in reinforcement learning via shared features. *The Journal of Machine Learning Research*, **13**(1), 1333–1371.

24. **Kumar, P.**, **V. Mathew**, and **B. Ravindran**, Abstraction in reinforcement learning in terms of metastability. *In European Workshop on Reinforcement Learning (EWRL)*. 2012.

25. **Lake, B. M.**, **T. D. Ullman**, **J. B. Tenenbaum**, and **S. J. Gershman** (2016). Building machines that learn and think like people. *arXiv preprint arXiv:1604.00289*.

26. **Lazaric, A.** and **M. Restelli**, Transfer from multiple mdps. *In Advances in Neural Information Processing Systems*. 2011.

27. **LeCun, Y.**, **Y. Bengio**, and **G. Hinton** (2015). Deep learning. *Nature*, **521**(7553), 436–444.

28. **Li, Y.** (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

29. **Lin, L.-J.** (1993). Reinforcement learning for robots using neural networks. Technical report, DTIC Document.

30. **Luxburg, U.** (2007). A tutorial on spectral clustering. *Statistics and Computing*, **17**(4), 395–416. ISSN 1573-1375. URL `http://dx.doi.org/10.1007/s11222-007-9033-z`.

31. **Mannor, S.**, **I. Menache**, **A. Hoze**, and **U. Klein**, Dynamic abstraction in reinforcement learning via clustering. *In Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004.

32. **McGovern, A.** and **A. G. Barto** (2001). Automatic discovery of subgoals in reinforcement learning using diverse density.

33. **Meila, M.** and **J. Shi**, A random walks view of spectral segmentation. 2001.

34. **Menache, I.**, **S. Mannor**, and **N. Shimkin**, Q-cut—dynamic discovery of sub-goals in reinforcement learning. *In European Conference on Machine Learning*. Springer, 2002.

35. **Mnih, V.**, **A. P. Badia**, **M. Mirza**, **A. Graves**, **T. P. Lillicrap**, **T. Harley**, **D. Silver**, and **K. Kavukcuoglu** (2016*a*). Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*.

36. **Mnih, V.**, **A. P. enech Badia**, **M. Mirza**, **A. Graves**, **T. Harley**, **T. P. Lillicrap**, **D. Silver**, and **K. Kavukcuoglu** (2016*b*). Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*.

37. **Mnih, V.**, **N. Heess**, **A. Graves**, *et al.*, Recurrent models of visual attention. *In Advances in Neural Information Processing Systems*. 2014.

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

39. **Mnih, V.**, **K. Kavukcuoglu**, **D. Silver**, **A. A. Rusu**, **J. Veness**, **M. G. Bellemare**, **A. Graves**, **M. Riedmiller**, **A. K. Fidjeland**, **G. Ostrovski**, **S. Petersen**, **C. Beattie**, **A. Sadik**, **I. Antonoglou**, **H. King**, **D. Kumaran**, **D. Wierstra**, **S. Legg**, and **D. Hassabis** (2015*a*). Human-level control through deep reinforcement learning. *Nature*, **518**(7540), 529–533. URL `http://dx.doi.org/10.1038/nature14236`.

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

41. **Ng, A. Y.**, **M. I. Jordan**, and **Y. Weiss**, On spectral clustering: Analysis and an algorithm. *In ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*. MIT Press, 2001.

42. **Ng, A. Y.** *et al.* (2002). On spectral clustering: Analysis and an algorithm.

43. **Niekum, S.**, **S. Chitta**, **A. G. Barto**, **B. Marthi**, and **S. Osentoski**, Incremental semantically grounded learning from demonstration. *In Robotics: Science and Systems*, volume 9. 2013.

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

45. **Oh, J.**, **X. Guo**, **H. Lee**, **R. L. Lewis**, and **S. Singh**, Action-Conditional Video Prediction using Deep Networks in Atari Games. *In* **C. Cortes**, **N. D. Lawrence**, **D. D. Lee**, **M. Sugiyama**, **R. Garnett**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015*b*, 2845–2853.

46. **Ortega, J.**, **N. Shaker**, **J. Togelius**, and **G. N. Yannakakis** (2013). Imitating human playing styles in Super Mario Bros. *Entertainment Computing, Elsevier*, **4**, 93–104.

47. **Parisotto, E.**, **J. Ba**, and **R. Salakhutdinov** (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. *CoRR*, **abs/1511.06342**.

48. **Parsons, L.**, **E. Haque**, and **H. Liu** (2004). Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, **6**(1), 90–105.

49. **Puterman, M. L.** (1994). Markov decision processes: Discrete stochastic dynamic programming.

50. **Ranchod, P.**, **B. Rosman**, and **G. Konidaris**, Nonparametric bayesian reward segmentation for skill discovery using inverse reinforcement learning. *In Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015.

51. **Rusu, A. A.**, **N. C. Rabinowitz**, **G. Desjardins**, **H. Soyer**, **J. Kirkpatrick**, **K. Kavukcuoglu**, **R. Pascanu**, and **R. Hadsell** (2016). Progressive neural networks. *CoRR*, **abs/1606.04671**.

52. **Sculley, D.**, Web-scale k-means clustering. *In Proceedings of the 19th International Conference on World Wide Web*, WWW '10. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-799-8. URL `http://doi.acm.org/10.1145/1772690.1772862`.

53. **Sharma, S.**, **A. S. Lakshminarayanan**, and **B. Ravindran** (2017). Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*.

54. **Shi, J.** and **J. Malik** (2000*a*). Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, **22**(8), 888–905. URL `http://dx.doi.org/10.1109/34.868688`.

55. **Shi, J.** and **J. Malik** (2000*b*). Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, **22**(8), 888–905.

56. **Şimşek, Ö.** and **A. S. Barreto**, Skill characterization based on betweenness. *In Advances in neural information processing systems*. 2009.

57. **Simsek, Ö.**, **A. P. Wolfe**, and **A. G. Barto**, Identifying useful subgoals in reinforcement learning by local graph partitioning. *In Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*. 2005. URL `http://doi.acm.org/10.1145/1102351.1102454`.

58. **Sorg, J.** and **S. Singh**, Transfer via soft homomorphisms. *In Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

59. **Sutton, R. S.** and **A. G. Barto**, *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1998*a*, 1st edition. ISBN 0262193981.

60. **Sutton, R. S.** and **A. G. Barto** (1998*b*). Introduction to reinforcement learning. *MIT Press*.

61. **Sutton, R. S.**, **D. A. McAllester**, **S. P. Singh**, **Y. Mansour**, *et al.*, Policy gradient methods for reinforcement learning with function approximation. *In Conference on Neural Information Processing Systems*, volume 99. 1999*a*.

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

63. **Sutton, R. S.**, **D. Precup**, and **S. P. Singh**, Intra-option learning about temporally abstract actions.

64. **Sutton, R. S.**, **D. Precup**, and **S. P. Singh**, Intra-option learning about temporally abstract actions. *In ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998. ISBN 1558605568. URL `http://portal.acm.org/citation.cfm?id=657453`.

65. **Talvitie, E.** and **S. Singh**, An experts algorithm for transfer learning. *In Proceedings of the 20th international joint conference on Artifical intelligence*. Morgan Kaufmann Publishers Inc., 2007.

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

67. **Taylor, M. E.** and **P. Stone** (2011). An introduction to intertask transfer for reinforcement learning. *AI Magazine*, **32**(1), 15.

68. **Thomaschke, R.** and **G. Dreisbach** (2013). Temporal predictability facilitates action, not perception. *Psychological science*, **24**(7), 1335–1340.

69. **Togelius, J.**, **S. Karakovskiy**, and **R. Baumgarten**, The 2009 mario ai competition. *In IEEE Congress on Evolutionary Computation*. IEEE, 2010.

70. **Vafadost, M.** (2013). Temporal abstraction in monte carlo tree search. *Master's thesis, Department of Computer Science, University of Alberta*.

71. **Van Hoorn, N.**, **J. Togelius**, **D. Wierstra**, and **J. Schmidhuber**, Robust player imitation using multiobjective evolution. *In 2009 IEEE Congress on Evolutionary Computation*. IEEE, 2009.

72. **Wang, Z.**, **Schaul**, **M. Hessel**, **H. van Hasselt**, **M. Lanctot**, and **N. de Freitas** (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

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

74. **Weber, M.**, **W. Rungsarityotin**, and **A. Schliep**, *Perron cluster analysis and its connection to graph partitioning for noisy data*.

75. **Weber, M.**, **W. Rungsarityotin**, and **A. Schliep** (2004). Perron cluster analysis and its connection to graph partitioning for noisy data. Technical Report 04-39, ZIB, Takustr.7, 14195 Berlin.

76. **White, S.** and **P. Smyth**, A spectral clustering approach to finding communities in graphs. *In In SIAM International Conference on Data Mining*. 2005.

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

78. **Xu, W.**, **X. Liu**, and **Y. Gong**, Document clustering based on non-negative matrix factorization. *In Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03. ACM, New York, NY, USA, 2003. ISBN 1-58113-646-3. URL `http://doi.acm.org/10.1145/860435.860485`.

LIST OF PAPERS BASED ON THESIS

1. Aravind S. Lakshminarayanan*, Sahil Sharma*, Balaraman Ravindran, "**Dynamic Action Repetition for Deep Reinforcement Learning**", Proceedings of 31st International Conference of AAAI, February 2017, https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14866.

2. Janarthanan Rajendran*, Aravind S. Lakshminarayanan*, Mitesh M. Khapra, Prasanna P, Balaraman Ravindran, "**Attend, Adapt and Transfer: Attentive Deep Architecture for Adaptive Transfer from multiple sources in the same domain**", Proceedings of the International Conference on Learning Representations, May 2017, https://openreview.net/pdf?id=Sy6iJDqlx.

3. Aravind S. Lakshminarayanan*, Ramnandan Krishnamurthy*, Peeyush Kumar*, Balaraman Ravindran, "**Hierarchical Reinforcement Learning using Spatio-Temporal Abstractions and Deep Neural Networks**", Proceedings of the Abstraction in Reinforcement Learning Workshop at International Conference in Machine Learning, June 2016, http://media.wix.com/ugd/3195dc_cad805b14f2e44babc108a147c5f785e.pdf.

* - Equal Contribution