Enhancing Reinforcement through Ensemble Learning

A Project Report

submitted by

RAKESH R MENON

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2017

THESIS CERTIFICATE

This is to certify that the thesis titled **Enhancing Reinforcement through Ensemble Learning**, submitted by **Rakesh R Menon (EE13B056)**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelors of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

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

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

Place: Chennai

Date: May 9, 2017

ACKNOWLEDGEMENTS

I would like to thank **Prof. Balaraman Ravindran** for his constant guidance and support during the course of the projects. The discussions with Prof. Ravindran and understanding his approach to different problems motivated me to perform research in this field. The obsession that resulted gave me a proper sense of direction and has motivated me to pursue masters in this field.

I would also like to thank **Prof. Kaushik Mitra** for his discussions and encouragement.

I would also like to thank **Sarath Chandar** for his valuable feedbacks regarding the project during the DHRL meetings.

A special thanks to **Manu Srinath Halvagal** for being a great project partner for the project on shared learning. I would also like to thank **Ghulam Ahmed Ansari**, **Sagar JP** and **Deepak Mittal** for their constant support through the year during our mini group discussions. I am also grateful to my friends in my wing for their support and the group adventures.

I would also like to thank the reviewers for **RLDM**, The Multi- disciplinary Conference on Reinforcement Learning and Decision Making, and the **Adaptive Learning Agents Workshop**(AAMAS 2017) reviewers for their suggestions and comments about the paper on **Shared Learning in Ensemble Deep Q-Networks**.

Finally, words cannot describe the amount of gratitude I have towards my parents and my brother who have been through my every whim and win. Thank you for being such role models for me to look up to.

ABSTRACT

KEYWORDS: Overestimation Bias, Online Transfer, Unsupervised Auxiliary Tasks, Reinforcement Learning

The field of deep reinforcement learning has been able to solve a large number of complex environments successfully. Much of these successes are because of the development of good learning algorithms like Deep Q-Networks and Trust Region Policy Optimization. In this work, we would like to analyze the effect of ensemble learning in deep reinforcement learning. To this end, we propose two approaches, (i) learn a set of value function estimates on the same task to improve exploration and induce faster learning, and (ii) learn a set of auxiliary tasks that learns to control and predict changes in the environment.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

MDP	Markov Decision Process(es)
RL	Reinforcement Learning
TD	Temporal-Difference
DP	Dynamic Programming
DQN	Deep Q-Network(s)
CNN	Convolutional Neural Network(s)
LSTM	Long Short-Term Memory
i.i.d.	independent and identically distributed
DDQN	Double Deep Q-Network(s)
BDQN	Bootstrapped Deep Q-Network(s)
A3C	Asynchronous Advantage Actor Critic
UVFA	Universal Value Function Approximator
UNREAL	Unsupervised Reinforcement and Auxiliary Learning
VR	Value Replay
RP	Reward Prediction
PC	Pixel Control
PoTs	Policy over Tasks

NOTATION

θ	Model Parameters
s	State
a	Action
r	Reward
γ	Discount Factor
G	Return function
Q	State-action-value function
V	State-value function
A	Advantage Function
π	Policy
B	Replay Buffer

CHAPTER 1

INTRODUCTION

In this section, we briefly introduce the idea of reinforcement learning and try to introduce the problems that we have worked on in this thesis. The background and related work required for understanding some of the concepts have been summarised in the following chapters.

1.1 Reinforcement Learning

Reinforcement learning is the problem of learning how to act in an environment in order to maximise cumulative rewards. The basic reinforcement learning agent (Figure 1.1) consists of an agent and an environment. The agent receives state s_t from the environment. The agent then takes an action a_t, which is executed in the environment in order to get the next state s_{t+1} and a reward r_t. The reward r_t tells the agent how good it is to perform an action taken from a given state.

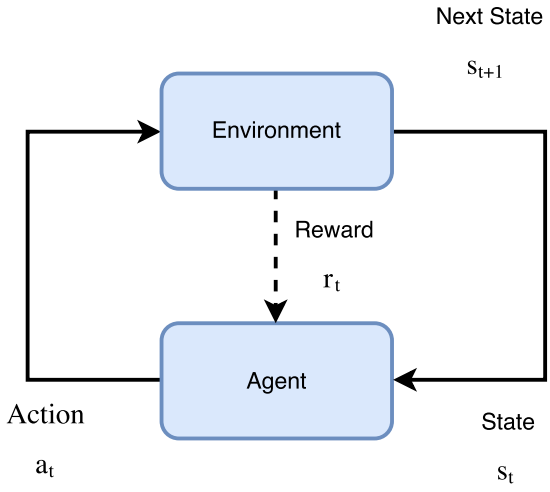


Figure 1.1: The reinforcement learning setup

The whole system can be formulated in terms of a Markov Decision Process which

has been described in Section 2.1.

In our definition of reinforcement learning, we have mentioned that the agent needs some notion of a cumulative reward that it needs to maximise. The agent gets this notion of cumulative reward through a function called as the ***return*** which basically tells the expected reward that an agent would get from a particular state. The agent has to learn the return function through a number of trials in the environment in order to understand how good different actions are from different states. The sequence of actions that results in the maximum cumulative reward is called as the *optimal policy*. The definitions and mathematical formulations for returns and policies have been mentioned in Section 2.2. The task of finding the optimal policy is usually met by the hurdle of the exploration-exploitation dilemma.

The exploration-exploitation dilemma is a fundamental trade-off in reinforcement learning which results from the dilemma of whether to take exploratory actions that could result in visiting more states in the environment (possibly rewarding) or to exploit the current knowledge that the agent and perform actions greedily to execute a certain policy. As an example, let us consider a real life decision of restaurant selection. Here, we could either go to our favourite restaurant or we could try out new restaurants (which may or may not be as good as our favourite restaurant). There has been a plethora of work on exploration-exploitation problems. Some of these works have been mentioned in Section 3.1.

Another recent development within reinforcement learning is the use of neural networks to give rise to a new sub-field called deep reinforcement learning. The idea of deep reinforcement learning was first introduced by Mnih *et al.* (2015). Further developments were made through the works of Van Hasselt *et al.* (2016), Schaul *et al.* (2015*b*) and Wang *et al.* (2015) among many others. The main reason for the success for deep reinforcement learning has been its ability to learn from only screen pixels and still solve problems to a great extent. More details about deep reinforcement learning have been described in Section 2.5.

In this work, we propose the use of ensemble networks in order to aid the model-free

algorithms in learning environments with large state spaces. We consider two-variants of ensemble learning:

- **Ensembles of Value Function estimates** : Here we study existing algorithms like Bootstrapped DQN (Osband *et al.* (2016*a*)) that have performed deep exploration in environments with large state spaces and produced amazing results on the Atari 2600 games. We also see that the Q-learning update rule of Bootstrapped DQN suffers from slight overestimation bias since there is a coupling between the online network and the target network of the ensemble network. Although this problem existed in Double Deep Q-Network (Van Hasselt *et al.* (2016)), there wasn't a clear solution to the problem. In this work, we present shared learning, an algorithm that reduces decoupling effects by leveraging ensemble Q-value estimates while at the same time sharing better knowledge in the process. We have called this work, **Shared Learning in Ensemble Deep Q-Networks**.
- **Ensemble of Auxiliary Tasks** : While we receive a lot of training signals from the environment, current deep reinforcement learning algorithms were only capable of leveraging a single signal, that is the pixels from the environment. Reinforcement Learning with Unsupervised Auxiliary Tasks introduced a novel method to take advantage of the different training signals available in the environment through unsupervised auxiliary learning. In this work, we wish to extend the work presented in Jaderberg *et al.* (2016) and introduce next frame prediction as an auxiliary task. Further, we try to learn a meta-policy that decides which auxiliary task needs to executed at what time. We have called this work, **Advancing Unsupervised Auxiliary Tasks for Reinforcement Learning**.

1.2 Shared Learning

While there have been many recent developments in exploration strategies in reinforcement learning, it has been realised that these strategies aren't performing deep exploration to search in large state spaces. Bootstrapped DQN Osband *et al.* (2016*a*), however, proposed a novel exploration strategy that performs deep exploration and showed a distinct improvement over existing methods at that time.

However, Bootstrapped DQN was suffering from a slight overestimation bias as the Double Q-learning update was performed as in Van Hasselt *et al.* (2016). While Double Q-learning (Hasselt (2010)) was able to completely remove the overestimation bias by maintaining different estimates of the action-value function. However, in Van Hasselt *et al.* (2016), the author(s) maintain a previous iteration of the action-value function and then perform the update. This causes some coupling between the estimates as mentioned in the same paper.

In this thesis, we propose the ***shared learning*** algorithm that tries to decouple estimates in Bootstrapped DQN by taking advantage of the different estimates present in the ensemble deep Q-network. Further, we go on to show how the choice of sharing the learned progress among the different action-value function estimates can lead to a better choice for performing perfect Double Q-learning updates and hence better learning overall. For our experiments, we present results on a toy Markov Decision Process chain and the ALE environment Bellemare *et al.* (2013).

1.3 Auxiliary Tasks for Reinforcement Learning

We as humans, live in an environment that provides us with a large number of sensory signals. We try to infer all of these signals in order to pursue our final goal of survival. For example, imagine a cricket ball coming directly towards you. Our learnt knowledge tells us that the ball needs to be avoided. But how do we know how to move away from the ball or where the ball currently is. To understand this, we would have to know what a ball is and what it looks like and the environment around it. Through Reinforcement Learning with Unsupervised Auxiliary Tasks (Jaderberg *et al.* (2016)), the author(s) have introduced unsupervised auxiliary tasks that learn through reinforcement learning to predict and control the environment. Some of the tasks they introduced include pixel control-a task that learns to control the pixels in the environment and reward prediction and value prediction tasks that together acts as a form of value iteration. They were able to produce amazing results on Atari 2600 games as well as the Labyrinth environment.

To this end, we would like to propose another auxiliary task based on next frame prediction that can help the agent predict the next frame given the previous few frames and the last action. This makes sense because in order to predict what kind of an action an agent must make, it must have some ability to predict the future frame that could possibly be more rewarding for the agent. This essentially amounts to having a dynamics model of the environment. We produce results for the prediction model on the Atari 2600 game of Space Invaders.

Additionally, we also try to learn a policy over all the auxiliary tasks presented in the

paper (Jaderberg *et al.* (2016)) as well as the next frame prediction model. The new Policy over Tasks model is evaluated on the Atari 2600 game of Space Invaders.

1.4 Summary

The rest of the thesis is organised as follows. Chapter 2 discusses some pre-requisites required for understanding the problem we are trying to solve. Chapter 3 discusses some literature related to the projects we are pursuing. 4 presents our work on shared learning with experiments on Markov Decision Process chains and ALE environment (Bellemare *et al.* (2013)). 5 presents our work on advancing unsupervised auxiliary tasks for reinforcement learning through next frame prediction models and Policy over Tasks(PoTs) model. In Chapter 6 we discuss the future directions for research for these projects.

CHAPTER 2

PRELIMINARIES

2.1 Markov Decision Processes

Markov Decision Processes (MDP) are frameworks that are used for representing a model of the environment in a reinforcement learning problem that satisfies the **Markov property**. A Markov Decision Process in the reinforcement learning setting is usually described by a five-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where :

- $\mathcal{S}$ represents the set of states that are present in the environment.
- $\mathcal{A}$ represents the set of actions that are available to the agent in the environment.
- $\mathcal{P}$ represents the dynamics model of the environment. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}_+$ is a probability distribution over the set of next states $s' \epsilon \mathcal{S}$ given that we are in a state $s \epsilon \mathcal{S}$. and have taken an action $a \epsilon \mathcal{A}$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function giving the expected reward associated with transitioning to state $s' \epsilon \mathcal{S}$ having taken an action $a \epsilon \mathcal{A}$ in state $s \epsilon \mathcal{S}$.
- $\gamma \epsilon (0, 1)$ is the discount factor which regulates how much we weight future rewards with respect to the more immediate rewards.

The states and actions can take up both discrete and continuous values. However, in this thesis, we consider cases where the state space is continous values but the action space is discrete.

2.2 Return, Policy and Value Functions

We have discussed how the agent's ultimate goal is to maximize the cumulative reward it gets in the long run. Formally, this can be defined using a notion of *return* which evaluates the actions taken by an agent based on the rewards that are received.

Definition 1: *The return received by an agent at time t is given as,*

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Definition 2: *A policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}_+$ is a probability distribution over the set of actions $\mathcal{A}$ that can be taken from a state $s \in \mathcal{S}$.*

In other words, the policy describes the sequence of actions that an agent takes in the environment.

Definition 3: *The value function $V_\pi(s)$ is the expected return when starting from a state $s \in \mathcal{S}$ and following a policy π. Mathematically, it can be written as:*

$$V_\pi(s) = \mathbb{E}_\pi[G_t | s_t = s]$$

The value function is also called as the **state-value function** as it tells us how good it is to be in a state s while executing a policy π. We would also like to know how good an action is from a given state s while following policy π and so define another measure called the **state-action value function**. The state-action value function is given as,

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$

A policy π^* is said to be *optimal* if,

$$V_{\pi^*}(s) \geq V_{\pi'}(s)$$

for all $s \in \mathcal{S}$. Here, π' represents any other policy other than π^*. There maybe multiple optimal policies based on the definition given above and we denote all those policies by π^*. The shared *optimal value function*, $V_{\pi*}(s)$ for these policies is given by:

$$V_{\pi^*}(s) = \max_\pi V_\pi(s)$$

2.3 Temporal-Difference Learning

Traditionally, there are two methods of solving RL problems:

- Dynamic Programming (DP) methods : These methods are useful for solving RL problems in situations where we have a perfect model of the environment.
- Monte Carlo methods : These methods involve learning directly from the environment through experience (actual or simulated).

Temporal-difference (TD) learning is a combination of these two standard approaches. Sutton and Barto (1998) regard the idea of temporal difference learning to be the key and central idea in RL.

The advantage of TD learning is that it can learn directly by interacting with the environment like Monte Carlo methods, and at the same time it can also bootstrap estimates of the value function, like DP methods, and hence can be used to learn in an on-line fashion.

In the next couple of sections we briefly look into algorithms that use TD learning for finding optimal policies.

2.3.1 SARSA

SARSA (Rummery and Niranjan (1994)) is an on-policy method, where we first estimate $Q_\pi(s, a)$ for the current behaviour policy for all states $s \epsilon \mathcal{S}$ and all actions $a \epsilon \mathcal{A}$. SARSA enjoys some good convergence properties for ϵ-greedy and ϵ-soft policies. The update rule for the state-action value function using the SARSA algorithm is given as,

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha[R_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]$$

where $Q_t(s, a)$ and $Q_{t+1}(s, a)$ are the value function estimates at times t and $t + 1$ respectively, r_t is the reward obtained at time t for choosing action a_t in state s_t, α is the learning rate.

2.3.2 Q-learning

Q-learning (Watkins and Dayan (1992)) is one method to learn the optimal value function for an agent using an off-policy strategy. The optimal policy can be achieved by behaving greedily with respect to the learned state-action value function in each state. The update rule for Q-learning is given as :

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha(r_t + \gamma \text{max}_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t))$$

Double Q-learning

The max operator in the Q-learning update has been shown to produce overestimation of the value function in Hasselt (2010). This comes about because a single estimator is used both for choosing the next action and for giving an estimate for the value of that action. Double Q-learning (Hasselt (2010)) reduces the overestimations by decoupling the action selection and value estimation by training two estimators $Q^A(s, a)$ and $Q^B(s, a)$. The update rule for double Q-learning is given as:

$$Q^A_{t+1}(s_t, a_t) = Q^A_t(s_t, a_t) + \alpha(r_t + \gamma Q^B_t(s_{t+1}, \text{argmax}_a Q^A_t(s_{t+1}, a)) - Q^A_t(s_t, a_t))$$

2.4 Parametrised Policy/Value Function

In environments with large state spaces, it is not possible to learn values for every possible state-action pair. The need for generalizing from experience of a small subset of the state space to give useful approximations of the value function becomes a key issue (Sutton and Barto (1998)). The function approximators used for representing the value function can be linear or non-linear functions. Methods like tile coding and coarse coding are known to provide great generalizations in the tabular RL setting.

Additionally, we can also parametrize the policy of an agent using function approximators. However, the weights for such a policy function would have to be learnt through the evaluation available from a value function. The most common way to learn the weights is through gradients of some performance metric for the policy. This leads to a

general set of methods called *policy gradients*. Methods where both the value function and the policy are learnt through function approximations are called as **Actor-Crtic Methods**. Actor-Critic methods have the advantage of being able to solve continuous control problems as well.

In the next section, we talk about a new area of research called Deep Reinforcement Learning which uses neural networks as function approximators and the expansion of the range of problems that has opened up in RL.

2.5 Deep Reinforcement Learning

2.5.1 Deep Q-Networks

Neural networks, while attractive as potential value function approximators, were known to be unstable or even to diverge on reinforcement learning problems until recently. Mnih *et al.* (2015) successfully overcomes these problems with two crucial ideas:

- ***Replay Memory*** : The replay memory allows the network to replay random samples from the most recent transitions. This breaks the correlation between the subsequent samples obtained from the environment and makes the whole training process independent and identically distributed (i.i.d.).
- ***Target Networks*** : During the calculation of the loss function using the TD error as in Equation 2.1, if the on-line network weights are used for target calculation, the Q-values can go out of control and the whole process becomes unstable. Hence, we maintain a set of network weights constant for a period of time and use these for calculation of the target for our network. This way the training process becomes more stable.

The result, an end-to-end deep reinforcement learning agent called as a deep Q-network (DQN) that learns from the raw pixels of the game screen. The parametrized value function is trained using the expected squared TD-error as the loss signal given to the neural network.

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[((r + \gamma max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \tag{2.1}$$

Here, $L_i(\theta_i)$ is the loss function, θ_i are the online parameters of the network and θ_i^- are the parameters of the target network. The target network is needed in order to provide

stationary targets so as to ensure stable training. The purpose of the replay memory is to reduce the correlation of the samples provided to the network during training.

2.5.2 Double Deep Q-Networks

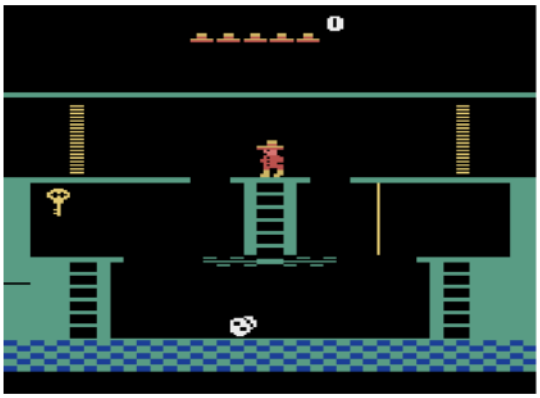
Van Hasselt *et al.* (2016) extends the idea of Double Q-learning to DQN in a computationally efficient manner by taking advantage of the target network. The greedy policy is evaluated using the online network and the value is given by the target network for this update. Thus the loss signal for the double DQN is given as:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta_i); \theta_i^-) - Q(s, a; \theta_i))^2] \tag{2.2}$$

2.5.3 Environments and Applications

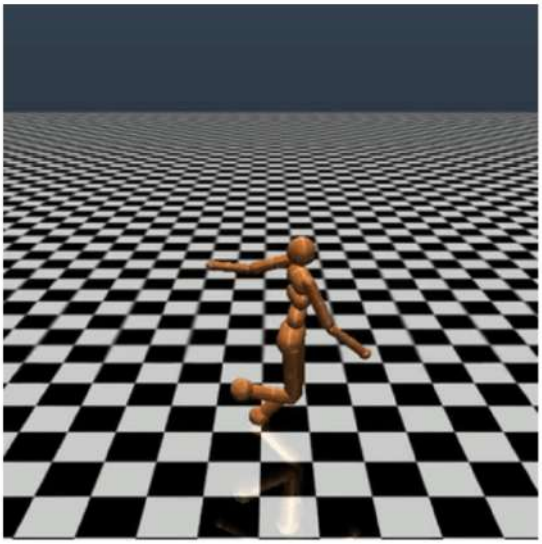
There emergence of the deep reinforcement learning meant that we could look into many new problems and solve even more complex environments. Some of the environments that have been recently developed to study deep reinforcement learning algorithms include Arcade Learning Environment (ALE) (Figure 2.1a) , VizDoom (Figure 2.1b), MuJuCo (Figure 2.1c) and Minecraft (Figure 2.1d) among many others.

Deep reinforcement learning has also advanced the field of robotics and spoken dialogue systems as well. Algorithms like Trust Region Policy Optimization (TRPO, Schulman *et al.* (2015)) (in simulated robotics tasks) and Guided Policy Search (GPS, Levine *et al.* (2015)) (for handling physical robots) provide new promise for robotics research in the future. Li *et al.* (2016) has designed a simulator for movie ticket booking and seeking. These are just a few fields where deep reinforcement learning has been of great advantage. For an extended overview the interested reader is requested to go through Li (2017).



(a) Montezuma's Revenge (ALE). Figure from Ostrovski *et al.* (2017)

(b) VizDoom. Figure from Kulkarni *et al.* (2016)



(c) Humanoid (MuJoCo). Figure from Ho and Ermon (2016)

(d) Minecraft. Figure from Tessler *et al.* (2016)

Figure 2.1: Deep Reinforcement Learning Testbeds

CHAPTER 3

RELATED WORK

3.1 The Exploration-Exploitation Dilemma

Prior work on exploration strategies in reinforcement learning have produced algorithms like R_{max} (Brafman and Tennenholtz (2002)) and E^3 (Kearns and Koller (1999)), which have near-optimal results and theoretical guarantees on MDP problems. However, such algorithms are intractable when it comes to exploration in domains with large state spaces. With the introduction of Deep Q-Networks (DQN) (Mnih *et al.* (2015)) for such domains, there was a need for better exploration strategies other than ϵ-greedy in order to perform human level control in these complex environments.

Some recent works have studied the use of **count-based methods** that incorporate exploration bonuses based on a count of the number of times a state-action pair is visited. These bonuses are meant to promote the visitation of state-action pairs that have been sparsely visited. Tang *et al.* (2016) uses the technique of hashing, wherein a part of the continuous state space is hashed into a discrete state space and exploratory bonuses are given based on the visitation of the discretized space. Bellemare *et al.* (2016) has tried to predict pseudo counts for state-action pairs using information theoretic approaches and conditional pixel probability models. This method has provided state-of-the-art performance on Montezuma's Revenge. Ostrovski *et al.* (2017) extends this work and uses a PixelCNN van den Oord *et al.* (2016) for the conditional pixel probability model.

Another way of calculating exploratory bonuses involves the use of **intrinsic motivation** (Singh *et al.* (2004), Barto (2013)) and its idea of state saliency. This state saliency idea has been exploited well in Stadie *et al.* (2015) wherein a model prediction error is used for calculating the exploratory bonus in Atari Games. Variational Information Maximizing Exploration (VIME, Houthooft *et al.* (2016)) extends the idea of intrinsic motivation to environments with continuous state action spaces and encourages exploration by getting information about the environment dynamics.

Some of these methods seem to face an issue when it comes to dealing with sparse rewards. Ensemble learning algorithms are known to perform well in any task due to their ability to incorporate knowledge from independent estimates and provide results that perform better than each of the individual estimates. The same idea has been leveraged for exploration in RL tasks by RLSVI Osband *et al.* (2016*b*) and PSRL Osband *et al.* (2013). Bootstrapped DQN Osband *et al.* (2016*a*) was one of the first algorithms to introduce and show the effectiveness of ensemble learning of Q-functions towards deep exploration in Atari games. We talk more about this in the next section.

3.2 Bootstrapped DQN

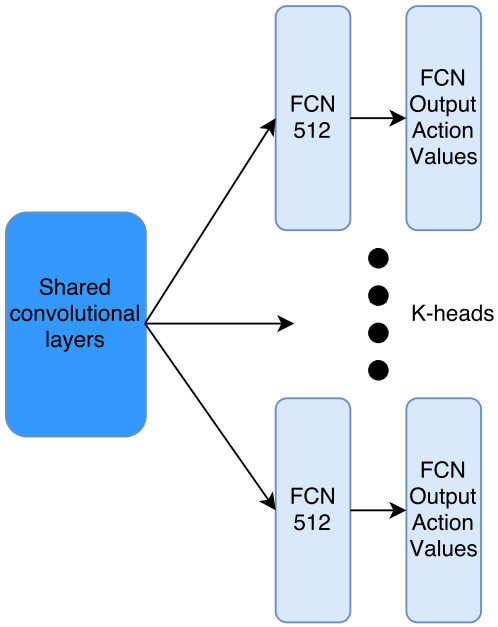


Figure 3.1: Bootstrapped DQN architecture

Bootstrapped DQN (Osband *et al.* (2016*a*)) introduces a novel exploration strategy that is capable of performing deep exploration in large state spaces. The key idea behind Bootstrapped DQN is the use of randomized value function estimates to approximate a distribution over possible action-values. At the start of every episode, Bootstrapped DQN samples a single value function estimate at random according to a uniform distribution. The agent then follows the greedy policy with respect to the selected estimate until the end of the episode. The authors propose that this is an adaptation of the Thompson sampling heuristic to RL that allows for temporally extended (or deep) exploration.

Bootstrapped DQN is implemented by adding multiple head networks which branch out from the output of the CNN as shown in Figure 3.1. Suppose there are K heads in this network. The outputs from each head represent different independent estimates of the action-value function. Let $Q_k(s, a; \theta_i)$ be the value estimate and $Q_k(s, a; \theta_i^-)$ be the target value estimate of the kth head. The loss signal for the kth head is exactly that given in equation 2.2. Each of the K heads is updated this way and the gradients are aggregated and normalized at the CNN layers.

3.3 Asynchronous Advantage Actor-Critic

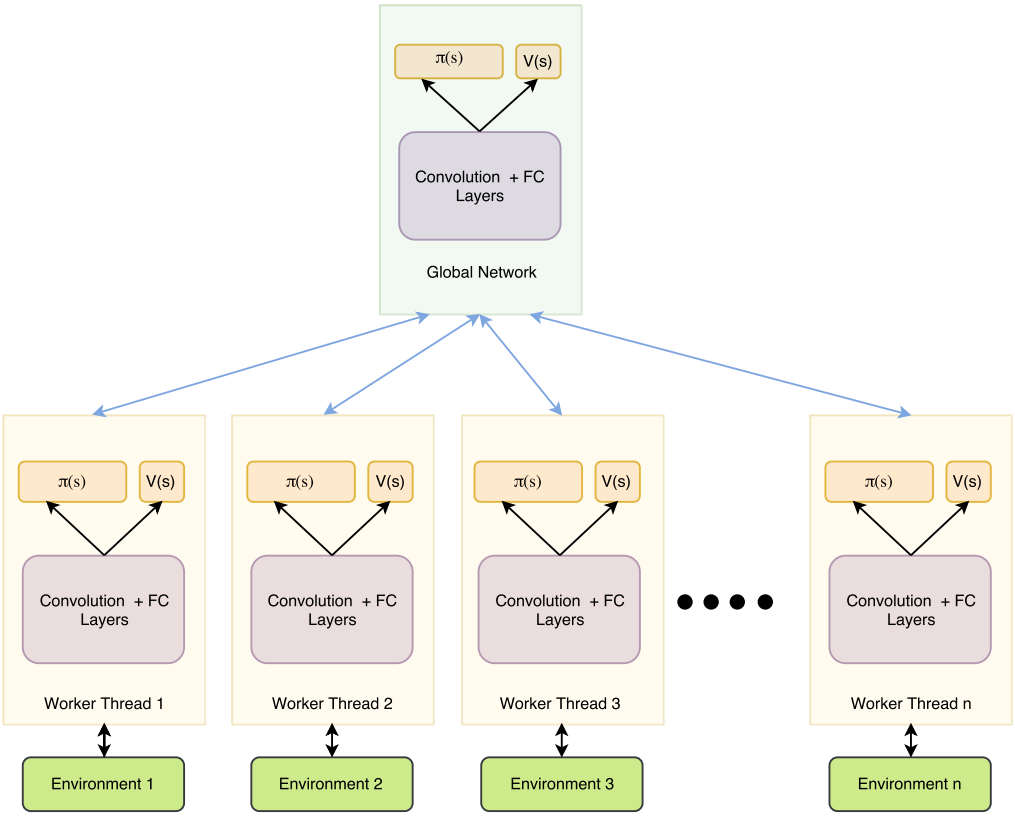


Figure 3.2: A schematic diagram of the A3C architecture

While DQNs were effective in solving a variety of challenging domains, the memory and computation required took a lot of space and time respectively. Mnih *et al.* (2016) introduces an asynchronous method to train multiple agents in parallel on multiple instances of the environment. The assumption is that the different agents executed asynchronously will explore differently and hence observe more unique parts of the

state-space. The best result of the paper was the asynchronous advantage actor critic (A3C), which provides great performance on Atari 2600 games as well as continuous control tasks.

A3C provides a parametrized representation for the policy $\pi(s)$ and value function $V(s)$. The algorithm is trained through n-step returns to update both the policy and value function. The update for the policy of the algorithm is as follows:

$$L(\theta_\pi) = \log \pi_{\theta_\pi}(s_t) A(s_t, a_t; \theta_v, \theta_\pi) \tag{3.1}$$

Usually, the advantage function $A(s_t, a_t)$ is defined as,

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

However, in this case we approximate the Q-value using the n-step return,

$$G(s_t) = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

here, T is the horizon which can be finite or infinite. Subtracting the value function from the n-step return helps in reducing the variance of the estimate of the advantage function. However, the estimate remains biased since the n-step return is a biased estimate of the Q-value.
The value function is updated using the n-step TD error given by,

$$L(\theta_v) = (V'(s_t) - V_{\theta_v}(s_t))^2$$

here, $V'(s_t)$ is an estimate of the n-step return from the current state.
Practically, the A3C algorithm also gives an advantage that the experiments can be run on a multi-core CPU instead of a GPU.

3.4 Maximizing Pseudo-reward Functions

Recently, there have been a lot of studies on how developing policies for the different sensory signals available in the environment can aid a reinforcement learning agent to

maximize the overall objective. In the following subsections, we have an overview at some of the architectures that aim at maximizing pseudo-reward functions.

3.4.1 The Horde architecture

The Horde architecture (Sutton *et al.* (2011)) consists of a multiple reinforcement learning agents, which they call as demons, that each focus on a specific task aimed towards maximizing a pseudo-reward function. The architecture when deployed on the Critterbot was also able to learn off-policy in real-time.

3.4.2 Universal Value Function Approximators

The Universal Value Function Approximators (UVFA, Schaul *et al.* (2015*a*)), much like the Horde architecture, creates a factorization of the value function into a set of embeddings for states and goals. Initial work in this direction mainly focused on the architectural design. The UVFA was however able to generalize to goals that were previously unseen.

3.4.3 Successor Representation

Similar to UVFAs, the successor representation (Dayan (1993), Kulkarni *et al.* (2016)) works by factorizing the value function into expected pseudo-reward function and the expected future state occupancy. The successor representation has the added advantage of being sensitive to distal rewards in sparse reward games because of its ability to factorize the reward and the world dynamics.

3.4.4 Auxiliary Tasks

Learning better representations through auxiliary tasks has been studied in the reinforcement learning setup in Lample and Chaplot (2016) and Mirowski *et al.* (2016). Most recently, Jaderberg *et al.* (2016) introduced a set of unsupervised auxiliary tasks which significantly outperformed other state-of-the-art algorithms on Atari games. More about this algorithm has been mentioned in the section below.

3.5 Reinforcement Learning with Unsupervised Auxiliary Tasks

As mentioned in the previous section, there are a multitude of sensory signals present in the environment of which we haven't taken full leverage. Jaderberg *et al.* (2016) introduces a novel approach to incorporate these signals by learning policies for these tasks using unsupervised learning while sharing some parameters with the base agent network. The auxiliary tasks are said to be used only for getting better representations and not to affect the main policy control in any manner. The set of auxiliary tasks proposed in the paper are as follows:

- ***Pixel change*** : In this task, a policy is learnt for changing the pixels maximally over a non-overlapping $n \times n$ grid places over the input.
- ***Network Features*** : A policy is learnt to maximally activate the units in the hidden layer of the agent's neural network.
- ***Reward Prediction*** : Here, the agent is made to predict the reward given a sequence of previous frames. Additionally, the reward predictor samples history in a skewed manner in order to sample non-zero reward and zero reward frames equally.
- ***Value Function Replay*** : Once the agent is capable of predicting the rewards, we can replay the value function from a *replay buffer* and perform an extra value function regression over the samples. This amounts to value iteration and helps in speeding up the whole learning process.

The first two tasks described above are trained using the n-step Q-learning update as in Mnih *et al.* (2016). The replay buffer used for the above auxiliary tasks use a memory size of about 2000 samples and the auxiliary tasks are trained every 20 steps.

Combining all the tasks mentioned above, the author(s) further came up with an UNsupervised Reinforcement through Auxiliary Learning (UNREAL, 3.3) agent that solves an objective function given by,

$$L_{UNREAL}(\theta) = L_{A3C} + \lambda_{VR} L_{VR} + \lambda_{RP} L_{RP} + \lambda_{PC} \sum_{c} L_{Q}^{(c)} \tag{3.2}$$

Here, $L_{UNREAL}(\theta)$ is the loss of the UNREAL agent, L_{A3C} is the loss of the base A3C agent, L_{VR} is the loss of the value replay, L_{RP} is the loss of the reward predictor and finally $L_{Q}^{(c)}$ is the loss of the pixel change auxiliary task.

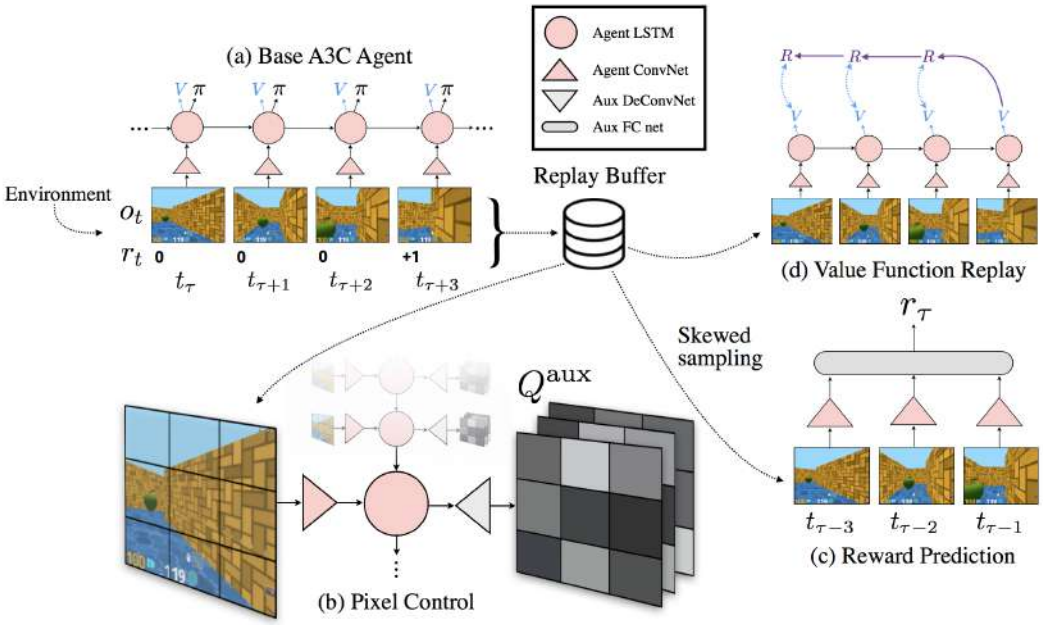


Figure 3.3: An overview of the UNREAL agent. Figure from Jaderberg *et al.* (2016)

3.6 Next Frame Prediction

The task of next frame prediction is a well studied topic in computer vision (Srivastava *et al.* (2015), Lotter *et al.* (2016)). In reinforcement learning problems, next frame prediction is not a new problem(Levine and Finn (2017), Oh *et al.* (2015), Finn *et al.* (2016)). Oh *et al.* (2015) has been able to proposed an LSTM-based future frame predictor that can predict upto 100 frames in the future. They further went on to show in the paper how the future predictor model can further be used towards informed exploration strategies. The paper gave a good insight into how we can develop a deep Dyna-Q model Sutton (1991) that can integrate simulated trajectories along with real trajectories to train a deep reinforcement learning agent. Similarly, Finn *et al.* (2016)) proposed another model that predicts a distribution over pixel motion from previous frames. This paper differs from the action-conditional video prediction paper in Oh *et al.* (2015) in that the main focus of the model is the motion of the pixels rather than some internal state of the model.

CHAPTER 4

SHARED LEARNING

4.1 Shared Learning for Bootstrapped DQN

4.1.1 Coupled Estimates in Double DQN

As discussed in section 3.2, the update for the action-value estimate of each head in the bootstrapped DQN architecture is given by the double Q-learning update rule as in equation 2.2. However the target network used here is an earlier iteration of the online network and hence there exists some coupling between the two networks. This poses a problem as the double Q-learning update is no longer perfect since the idea was hinged on decoupled estimates (Van Hasselt *et al.* (2016)).

4.1.2 Proposed Algorithm

The ensemble architecture put forward in Osband *et al.* (2016*a*) suggests a computationally efficient way to circumvent the problem described above. To this end, we propose an algorithm that generates decoupled target value estimates by taking advantage of the fact that the each head in Bootstrapped DQN already maintains a value estimate that is completely independent of the other heads.

Consider the K-headed Bootstrapped Deep Q-Network described in Section 3.2. In order to decouple the estimates, we use the values from a head other than the one being updated, to pick the greedy action for the target value in the double Q-learning update. The new loss signal can be formulated as follows:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma Q_k(s', \text{argmax}_{a'} Q_m(s', a'; \theta_i); \theta_i^-) - Q_k(s, a; \theta_i))^2] \quad (4.1)$$

where $m \in \{1, 2, \ldots, K\} \setminus \{k\}$.

The target value given here is expected to be a more robust estimate when compared to

Double DQN, since it is derived from a completely independent estimate of the value function.

4.1.3 Online Transfer of Learning Progress

The head selected for picking the greedy action for the target value in equation 4.1 provides a degree of guidance to the head being updated, allowing transfer of some amount of learned knowledge. Hence, basing the selection of this head on a measure of local learning progress rather than choosing an arbitrary head seems a better choice. This would be in some sense an online transfer of learned knowledge.

More concretely, at any point of time, we would expect at least one head to have learned more about the current region of the state space than the other heads. Online transfer of experience from this head to the rest of the ensemble network could aid learning in the other heads and result in more directed exploration. With this in mind, we propose the *shared learning* algorithm which modifies the loss signal from equation 4.1 in the following way:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma Q_k(s', \text{argmax}_{a'} Q_{best}(s', a'; \theta_i); \theta_i^-) - Q_k(s, a; \theta_i))^2] \quad (4.2)$$

Here $Q_{best}(s', a'; \theta_i)$ represents the action-values of the 'best' head (selected at regular intervals) that has progressed furthest in learning about the current state. In our implementation, we quantify the learning progress using the action-values of the online network. This measure was chosen because the head which outputs the highest action-value is expected to have learned off of more rewarding subsequent states. The complete algorithm has been summarized in Algorithm 1.

4.2 Discussion

An alternative way to get more robust target estimates in Bootstrapped DQN would be to use other heads to provide the actual target Q-value instead of just choosing the

Algorithm 1 Shared Learning Algorithm

Input: Action-value function networks Q with K outputs $\{Q_k\}_{k=1}^{K}$

Let B be a replay buffer storing experience for training and select_best_int be the interval during which the best head is selected.
numSteps $\leftarrow 0$
best_head $\sim$ Uniform$\{1, 2, \ldots, K\}$
for each episode **do**
    Obtain initial state from environment s_0
    Pick value function to act using $k \sim$Uniform$\{1, 2, \ldots, K\}$
    for step $t = 1, \ldots$ until end of episode **do**
        Pick an action according to $a_t \in \text{argmax}_a Q_k(s_t, a)$
        Take action a_t and receive state s_{t+1} and reward r_t
        Sample minibatch from B
        Train network with loss function given in equation 4.2
        numSteps $\leftarrow$ numSteps + 1
        if numSteps mod select_best_int $== 0$ **then**
            best_head = $\text{argmax}_k Q_k(s_t, a_t)$
        end if
    end for
end for

greedy action. The loss signal for this approach would then be:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}[(r + \gamma Q_m(s', \text{argmax}_{a'} Q_k(s', a'; \theta_i); \theta_i^-) - Q_k(s, a; \theta_i))^2] \quad (4.3)$$

where $m \in \{1, 2, \ldots, K\} \setminus \{k\}$.

In fact, this update rule would be closer to the true double Q-learning update than our approach. In this case, the actual values end up getting shared between the heads which then no longer remain independent estimates. However, variability between the action-value estimates of the heads is the basis of deep exploration in Bootstrapped DQN. Hence, direct value transfer is not desirable within the Bootstrap framework. This is why *shared learning* is centered around partial policy transfer rather than value transfer.

Additionally, we would also like to point out that the added performance due to our algorithm over Bootstrapped DQN increases progressively with increasing number of heads. This is because of the fact that with fewer heads (2 in the extreme case), it is more likely that a given head is updated with its own target network. In other words, the best head is the same as the given head for a larger fraction of heads when there are fewer heads (one out of K heads). Even in this case, we would have a worst-case performance equal to that of Bootstrapped DQN.

4.3 Experiments

We first illustrate how online transfer of learning progress can speed up learning on an MDP chain environment. In this tabular environment, the issue of coupling between target estimates and online estimates is nonexistent, and any speedup due to our algorithm would primarily be because of online transfer. Finally, we demonstrate the effectiveness of our algorithm on a subset of Atari Games using the ALE environment where we observe the advantage of both the decoupled estimates as well as the shared learning progress.

4.3.1 MDP experiments

Experimental Setup

We present a toy chain MDP experiment to demonstrate the faster learning of the new approach. The MDP consists of n states, arranged sequentially as shown in Figure 4.1. The agent always starts from state s_2. From each state *four* actions are allowed namely, go left, go right, do nothing or jump to state s_1 incurring a reward of -10 and terminating the episode. The episode ends when the agent has reached the nth state upon which it receives a reward of +10.

We use normal Q-learning (with ϵ-greedy) and double Q-learning (with ϵ-greedy) as baselines to compare the performance of a bootstrap agent (5 heads) with and without shared learning. Figures 4.2 and 4.3 show the result for a 35 and 50 state MDP respectively.

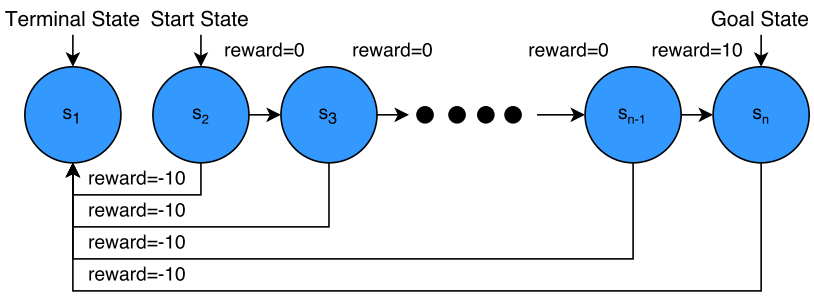


Figure 4.1: MDP Chain Structure with n-states

Results

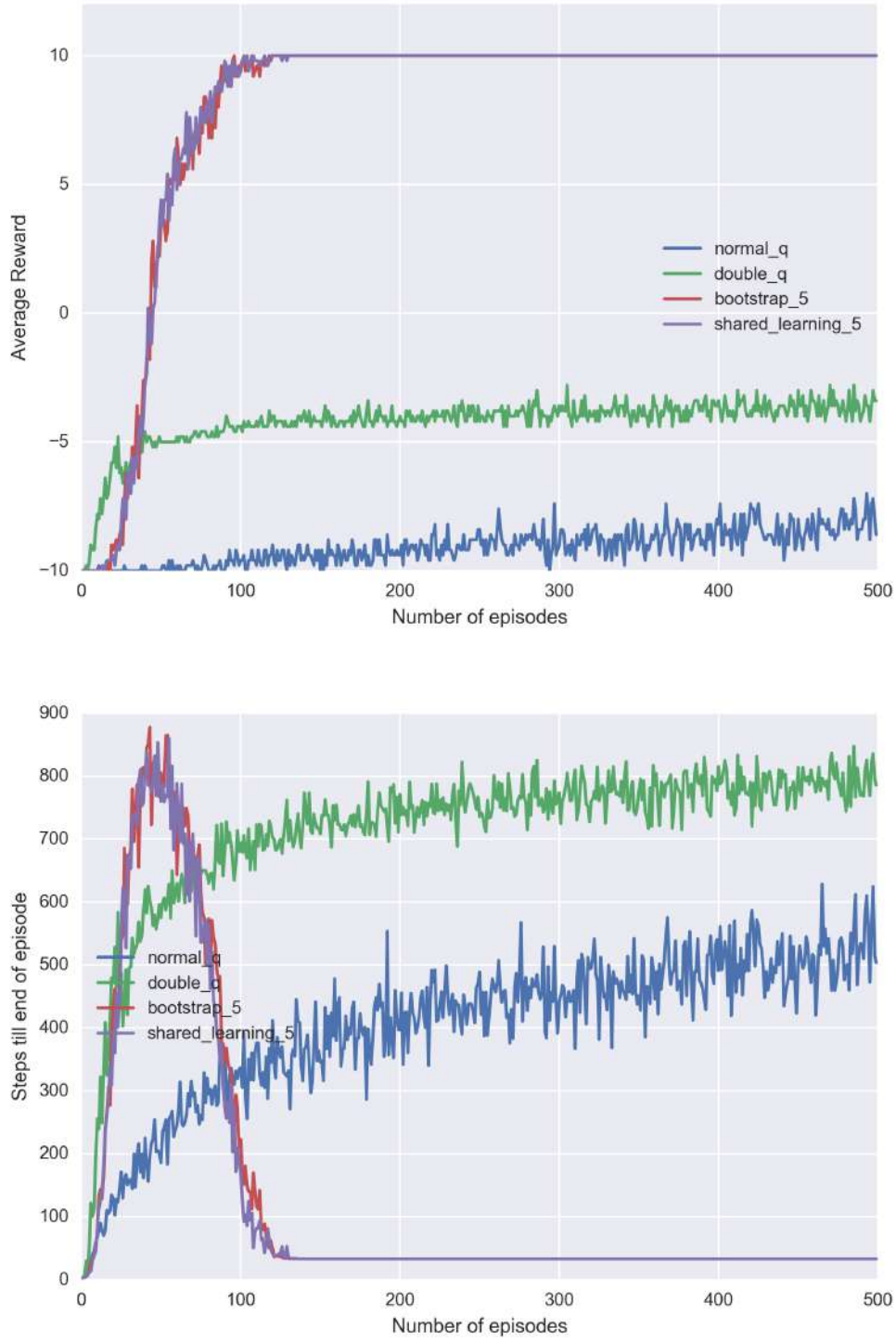


Figure 4.2: MDP results showing average reward obtained per episode and number of steps taken until completion for 35-state chain MDP

Solving the above environment requires deep exploration, especially with a larger number of states. This is illustrated by the fact that Q-learning and double Q-learning, with ϵ-greedy exploration, are unable to solve large MDP chains (beyond 20 states).

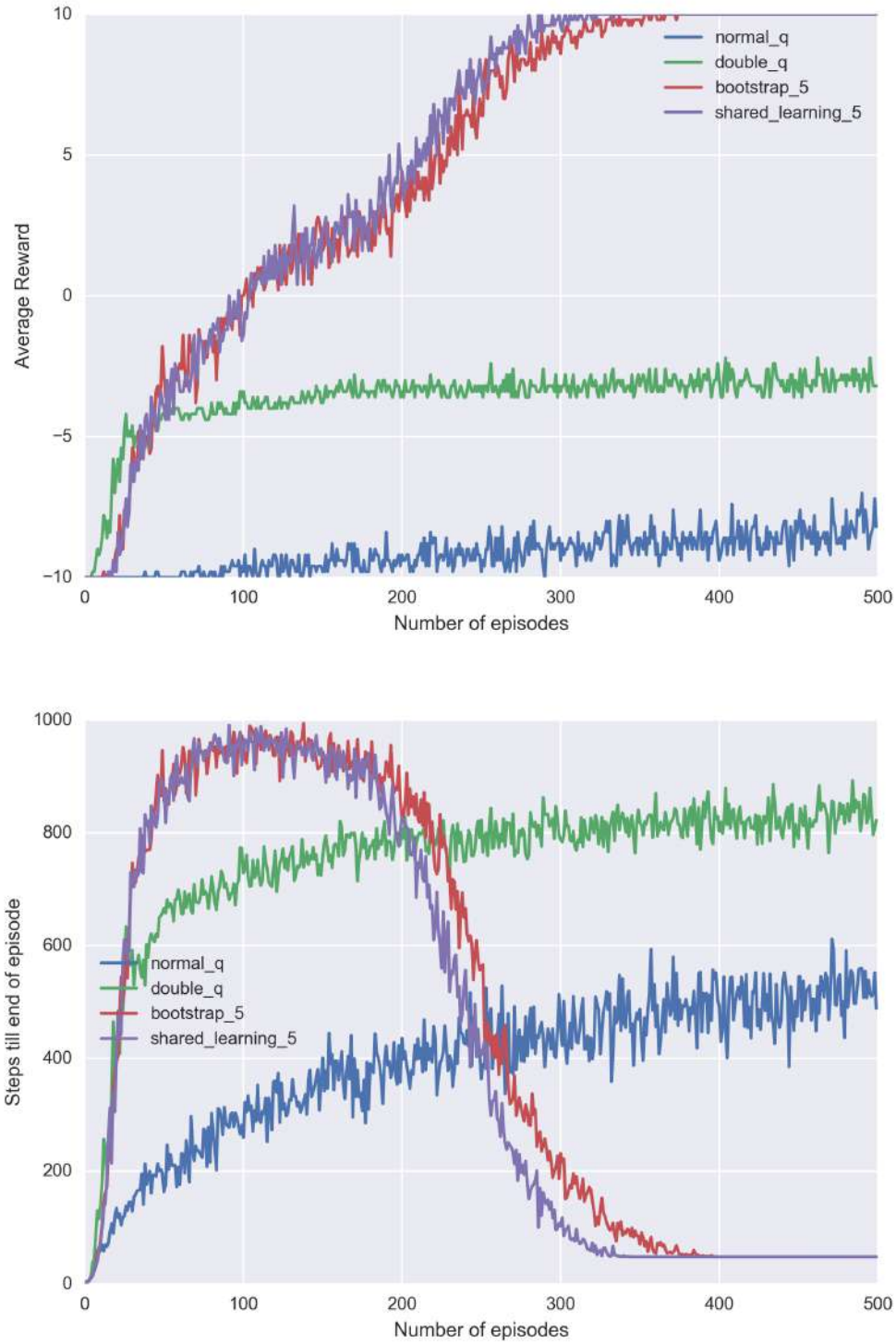


Figure 4.3: MDP results showing average reward obtained per episode and number of steps taken until completion for 50-state chain MDP

The results verify that Bootstrapped DQN (adapted for tabular settings) can indeed perform deep exploration. The fact that our algorithm is still capable of performing deep exploration shows that sharing does not take away from the diversity among the estimates, which is what drives the exploration in Bootstrapped DQN.

We observe that the speedup due to shared learning becomes more evident with increasing chain length (Figure 4.2, Figure 4.3). The performance of both Bootstrapped DQN and *shared learning* would probably be identical on this problem until the first time the goal state is reached. It is at this point that sharing learned experience becomes vital so that knowledge of the goal state propagates out to every head. This is the reason why *shared learning* outperforms the bootstrap algorithm on larger MDP chains.

Discussion

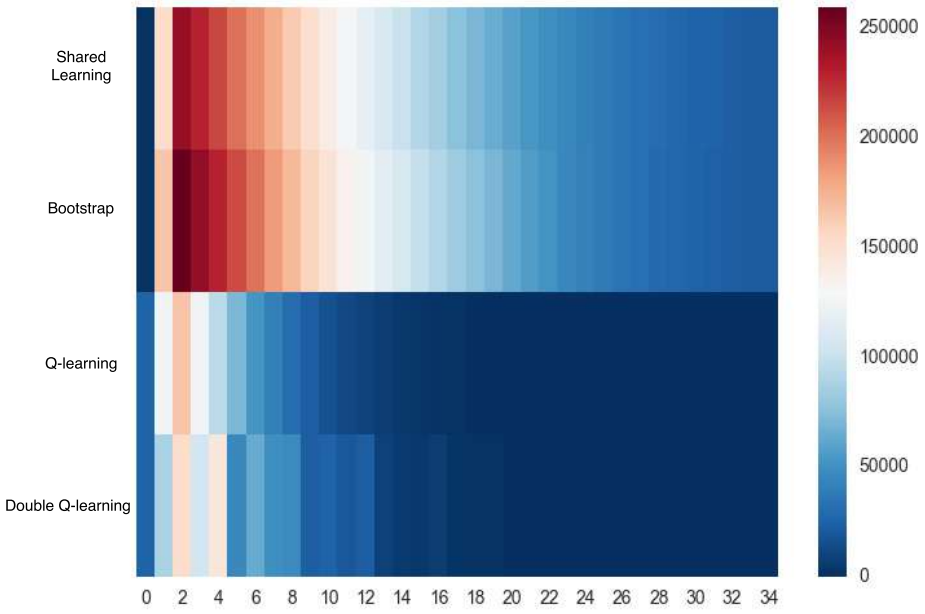


Figure 4.4: Heatmap of states visited by different algorithm in 35-state MDP

To analyze if our approach based on shared learning is able to perform exploration nearly as fast as the bootstrapped DQN version, we measured the number of state visitations made by the agent over 50 runs for 1000 episodes (max steps per episode 1000). During the experiment, the observation was that the while the bootstrap algorithm was able to reach the goal faster it was the *shared learning* approach that was able to reach the goal more consistently. This has been made evident for the 35-state chain MDP and the 50-state chain MDP in Figure 4.4 and Figure 4.5 respectively. This provides more concrete evidence to our hypothesis about shared learning that while it may affect exploration marginally, once the solution has been reached, the sharing of the knowledge

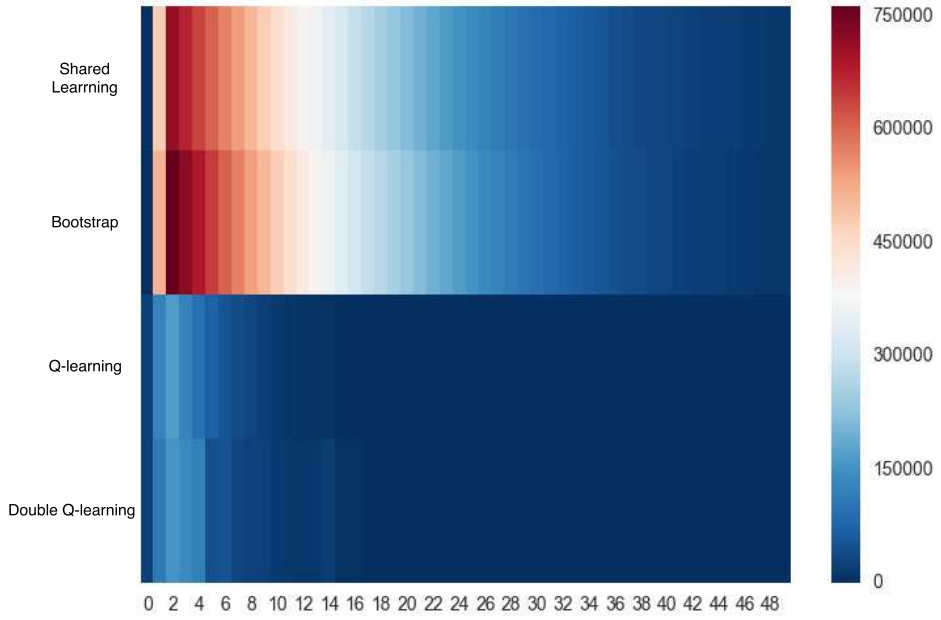


Figure 4.5: Heatmap of states visited by different algorithm in 50-state MDP

of the solution to the other estimates are fast enough by our metric and so we can reach the goals more consistently.

4.3.2 Atari 2600 experiments

We evaluate our algorithm on 6 Atari games on the Arcade Learning Environment Bellemare *et al.* (2013). The games chosen for evaluation were Seaquest, Frostbite, Hero, Breakout, Qbert and Pong. Most of these games have been chosen in order to compare with the results shown in Osband *et al.* (2016*a*). The network architecture used for shared learning is the same as that of Bootstrapped DQN and consists of a shared convolutional layer followed by 10 bootstrap heads to provide value function estimates. Gradient normalization of $1/K$ (K is the number of bootstrap heads) was also applied to the network with no *masking* (see Osband *et al.* (2016*a*)) of heads. The learning progress for the Atari games have been measured using the Q-values at intervals of 100 steps. The head corresponding to the maximum Q-value is chosen to be the best head that is suitable for information transfer. In Figures 4.6, 4.7 and 4.8, we show the results on Seaquest, Frostbite and Hero for Shared Learning against Bootstrapped DQN over 50 epochs and for Breakout in Figure 4.9 over 35 epochs.

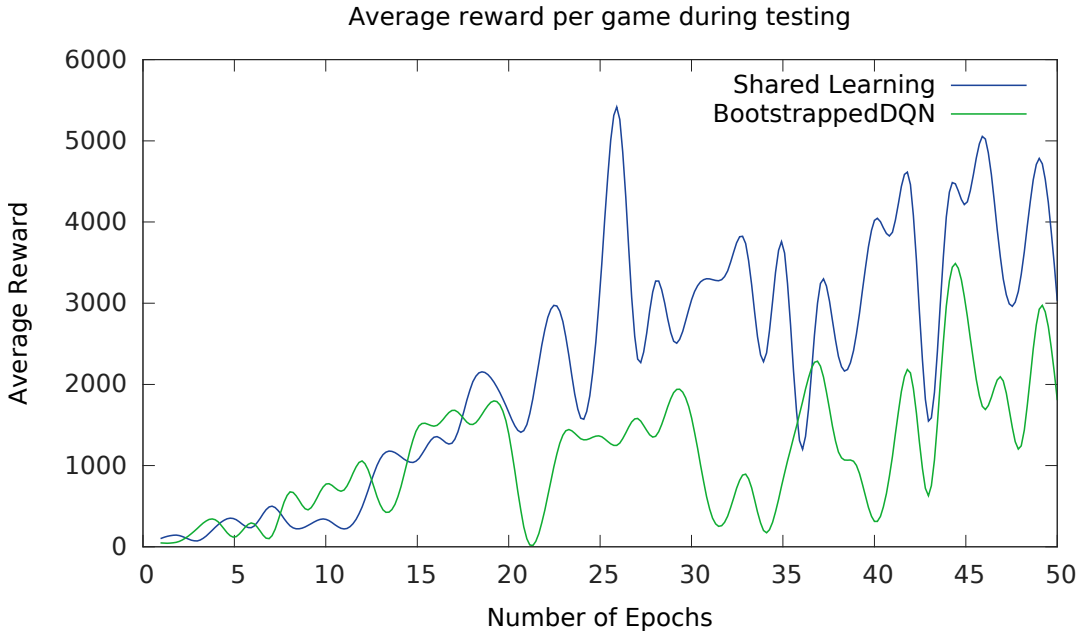


Figure 4.6: Average Reward on Seaquest after 50 epochs of training

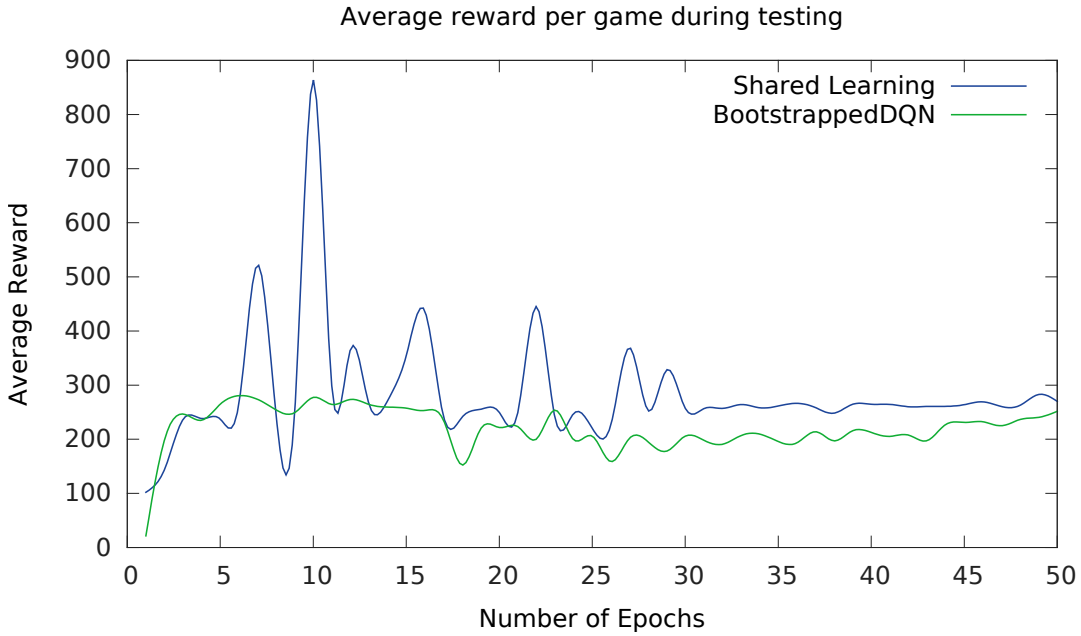


Figure 4.7: Average Reward on Frostbite after 50 epochs of training

Table 4.1 shows the cumulative rewards obtained on the Atari games Qbert and Pong over 20 epochs.

We have not presented the performance of DQN Mnih *et al.* (2015) and double DQN Van Hasselt *et al.* (2016), since it was shown in Osband *et al.* (2016*a*) that Bootstrapped DQN clearly outperforms them. Our results show an increased cumulative reward dur-

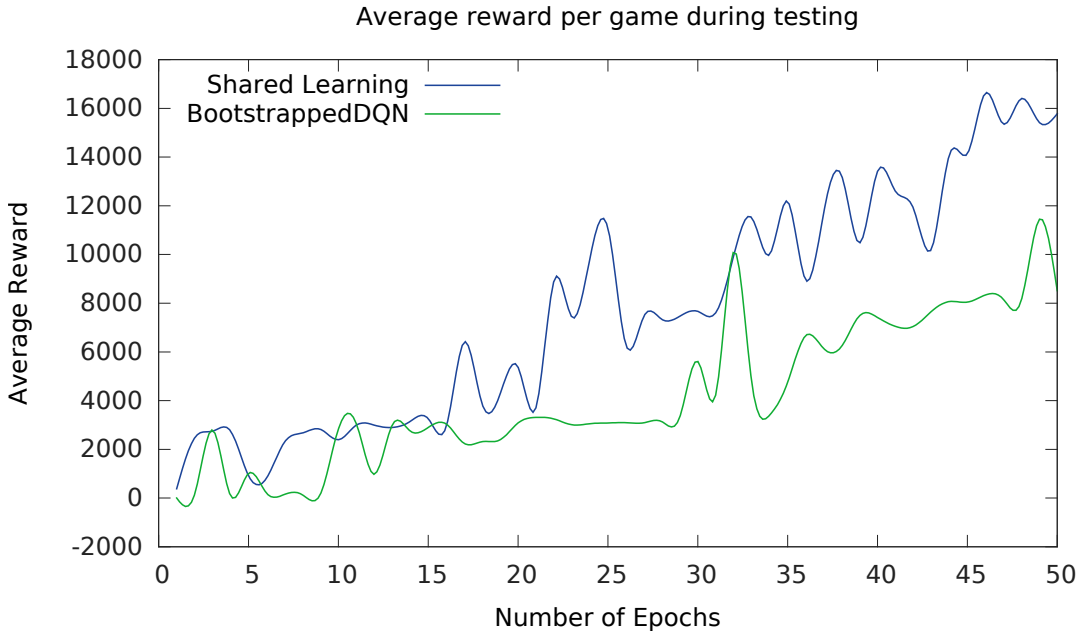


Figure 4.8: Average Reward on Hero after 50 epochs of training

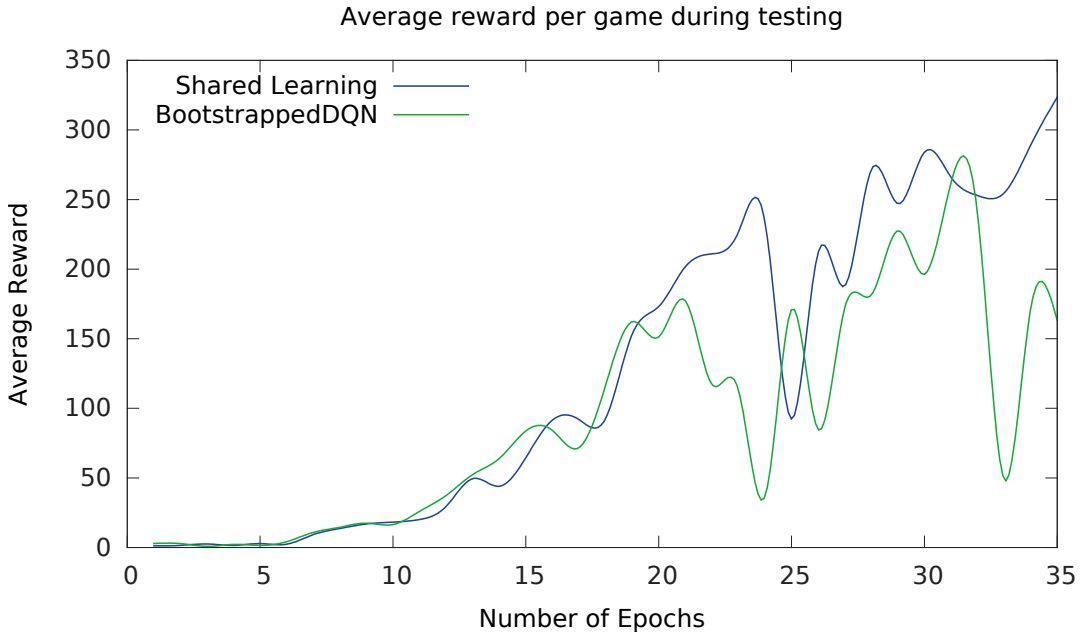


Figure 4.9: Average Reward on Breakout after 35 epochs of training

Cumulative Reward on Atari Games during testing		
Game	Shared Learning Score	Bootstrapped DQN Score
Qbert	**39001.6661**	15286.0259
Pong	**-44.9735**	-80.4237

Table 4.1: 20 epoch cumulative scores

ing learning, while Bootstrapped DQN appears to dither to a larger extent.

The significant boost in performance on the Atari games over Bootstrapped DQN confirms that the double Q-learning updates in DDQN and Bootstrapped DQN were in fact imperfect.

CHAPTER 5

ADVANCING UNSUPERVISED AUXILIARY TASKS FOR REINFORCEMENT LEARNING

5.1 Next Frame Prediction as an Auxiliary Task

5.1.1 Motivation

In section 3.5, we saw how unsupervised auxiliary tasks which learn various control policies through pixel control, reward prediction and value replay can lead to better representations for the extrinsic goal while trying to solve internal goals themselves.

Better representation learning has been a leading question in the field of artificial intelligence for a long time. In the case of reinforcement learning, a better representation can be obtained by developing tasks that help in getting a better model of the environment. One such unsupervised task which helps is next frame prediction (NFP). By using imagined and predicted frames of the environment we can model the physical interactions in the world which can be generalized to unseen settings as well.

Why unsupervised next frame prediction should learn good features?

As mentioned in Srivastava *et al.* (2015), in order to perform next frame prediction correctly, it is important for the model we create to understand the kind of objects that are present in the screen and how they move in the environment. This kind of abstract information is learnt by the encoding architecture of a next frame predictor model. Mirza *et al.* (2016) further goes on to show that such predictor models can also be made competent to generalize in unseen settings.

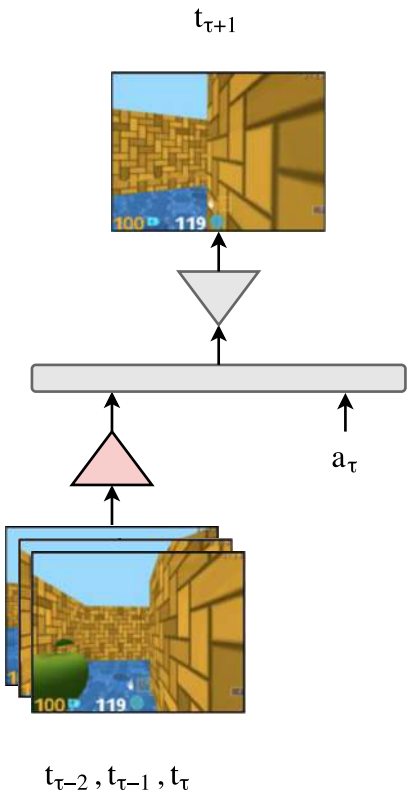


Figure 5.1: Next Frame Predictor Model architecture. The notations for the networks are consistent with Figure 3.3.

5.1.2 Next Frame Predictor Model

In order to perform next frame prediction, we design a model as in Figure 5.1. To the model, we encode the stacked last three frames (from the *replay buffer*) using a convolutional network and the last action taken by the agent and pass the encodings to a fully connected layer. The embedding from the fully connected layer is further passed through a deconvolutional network in order to construct the next frame.

Loss Function

The loss functions used for frame predictions has been a long standing debate. Ranzato *et al.* (2014) discusses how the use of squared error as the loss function is not a good enough metric for frame prediction. They quantized the images into patches and try to predict the identity of the target patches as a way to learn the frame predictions. Mirza *et al.* (2016) , however, use the predictive mean squared error with the Adam optimizer and early stopping for training. In our experiments we use the predictive mean squared

error given by,

$$L_{NFP} = \frac{1}{M}\frac{1}{N}\frac{1}{C}\sum_{m=1}^{M}\sum_{n=1}^{N}\sum_{c=1}^{C}[(\hat{f}(m,n,c) - f(m,n,c))^2] \tag{5.1}$$

Here, $\hat{f}$ is the predicted frame representation, f is the actual frame representation. M, N and C represent the number of pixels along x-axis, y-axis and the number of channels respectively.

5.1.3 Experiments

Initial results

In this section, we show the results of our experiments on the Space Invaders game using the ALE environment (Bellemare *et al.* (2013)).

As a baseline for our model, we use the original set of tasks from the UNREAL paper (Jaderberg *et al.* (2016)). To test our model, we try swapping the pixel control task from the original paper with the our next frame predictor model and check our results. The algorithm we use has been summarized in Algorithm 2.

The result on Space Invaders is shown below in Figure 5.2.

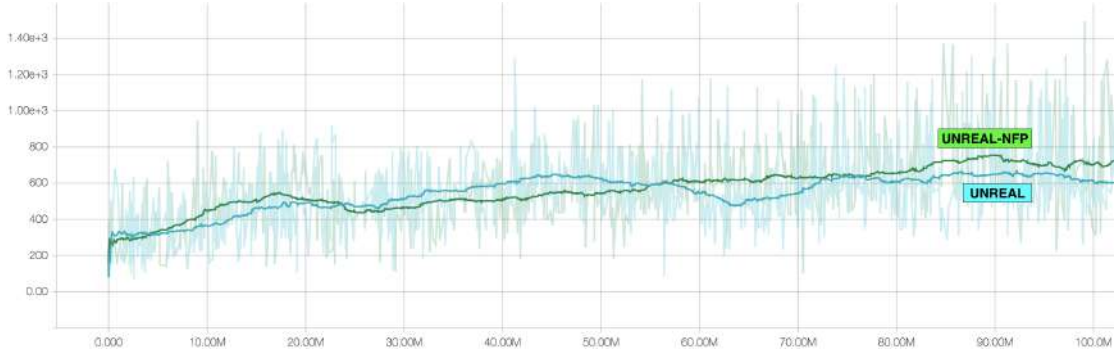


Figure 5.2: UNREAL agent vs UNREAL-NFP(as in Algorithm 2) on Space Invaders

Algorithm 2 Reinforcement Learning with Value Replay, Reward Prediction and Next Frame Prediction Auxiliary Tasks - for each thread

Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
Assume thread-specific parameter vectors θ' and θ'_v
Let B be a replay buffer storing experience for training. Fill buffer with transitions using a random policy.
Assume θ^c_{shared} be the shared parameters for the auxiliary task c with the base agent.
Initialize thread time counter $\tau \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = \tau$
    Get state t_τ
    repeat
        Perform a_τ according to policy $\pi(a_\tau|t_\tau, \theta')$
        Receive reward r_τ and new state $t_{\tau+1}$
        $\tau \leftarrow \tau + 1$
        $T \leftarrow T + 1$
    until terminal state s_τ or $\tau - t_{start} == t_{max}$

$$R = \begin{cases} 0 & \text{for terminal state} \\ V(t_\tau; \theta'_v) & \text{for non-terminal state } t_\tau \end{cases}$$

    for $i \in \{\tau - 1, ..., t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt θ' : $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i; t_{i;\theta')(R - V(t_i;\theta'_v))}$
        Accumulate gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v + (R - V(t_i; \theta'_v))^2 / d\theta'_v$
    end for
    Perform skewed replay sampling from B to pass through Reward Predictor model
    Compute L_{RP}
    Perform skewed replay sampling from B to pass through NFP model
    Compute L_{NFP}
    Perform uniform random sampling from B to pass through Value Replay model
    Compute L_{VR}
    $L_{aux} = \lambda_{VR} L_{VR} + \lambda_{RP} L_{RP} + L_{NFP}$
    Accumulate auxiliary task gradients wrt θ' : $d\theta \leftarrow d\theta +_{aux} / d\theta'$
    Accumulate auxiliary task gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v +_{aux} / d\theta'_v$
    Perform asynchronous update of θ using $d\theta$ and θ_v using $d\theta_v$
until $T > T_{max}$

Discussion

From the Figure 5.2, we see that the next frame prediction performs well initially and the scores have saturated to a low score very early. This could be because the model overfit and started focusing on detail which wasn't very important. For example, in a game like Seaquest the model may have focused too much on reconstructing the sea

and sky properly instead of the game score and the opponents. So, we try to provide some regularization through the use of ***dropouts*** (Srivastava *et al.* (2014)).

Results with Dropout

For the experiments with dropout, we use a keep probability for the neurons as 0.75 and 0.5. From the results in figure 5.3, we see that the regularized model does much better than the original UNREAL agent proposed in Jaderberg *et al.* (2016). This shows how important NFP is as an unsupervised auxiliary task in guiding the agent towards the overall extrinsic goal. UNREAL-NFP(from Algorithm 2) performs best with dropout having keep probability 0.75 in this game.

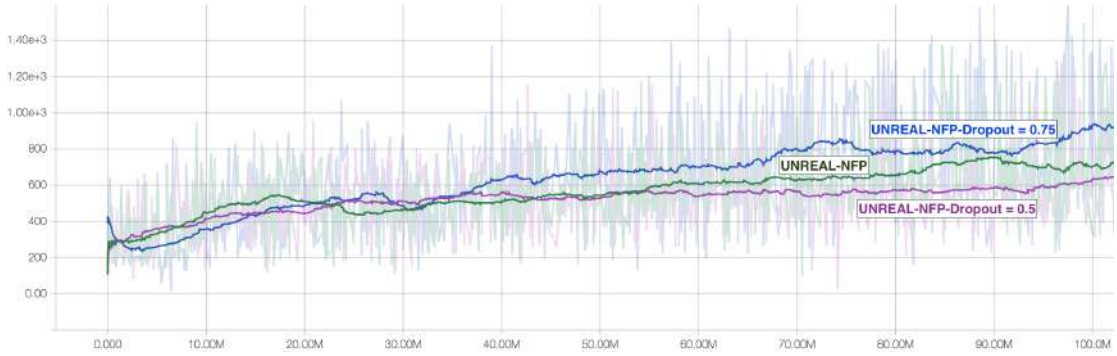


Figure 5.3: UNREAL-NFP vs UNREAL-NFP (Dropout=0.75) vs UNREAL-NFP (Dropout=0.5) on Space Invaders

5.2 Sequencing Auxiliary Tasks for Reinforcement Learning

5.2.1 Motivation

In the previous section, we always considered a scenario when multiple auxiliary tasks are running at the same time. However, it is questionable, how effective running all the control policies at once would be. For example, consider a game of sparse rewards and suppose that we have reached a state that is providing us a high score. Now it is important for the agent to understand the rewarding features of this state first before trying to learn a value function over the state. In such a case, we would want the

reward replay to be performed more initially, followed by the value replay auxiliary task to promote the policy that lead to the rewarding state. So, we need to sequence the auxiliary tasks. Hence, we design a new Policy over Tasks (PoTs) model that can predict which one auxiliary task that needs to be executed.

5.2.2 Policy over Tasks Model

Model setup

The policy over tasks (PoTs) model develops a policy over the set of available auxiliary tasks using the state encoding of the previous t_{max} states, where t_{max} is a hyper-parameter or the number of time steps before a terminal state has occurred. In order to do this an extra policy layer is built over the base A3C agent. A softmax output of the policy layer gives the unsupervised auxiliary tasks to execute. The policy is learnt using the policy gradient update rule as given by the equation below,

$$L(\theta_{\pi_{PoTs}}) = \log \pi_{\theta_{\pi_{PoTs}}}(s_t) A(s_t, a_t; \theta_v, \theta_{\pi_{PoTs}}) \tag{5.2}$$

Note, that the value function used for learning is the same as the value function used by the base A3C agent and so we are essentially solving a multi-objective reinforcement learning problem.

5.2.3 Experiments

To check the efficacy of the model, we test our model summarised in Algorithm 3 against the base UNREAL model from Jaderberg *et al.* (2016) on the Space Invaders game using the ALE environment Bellemare *et al.* (2013). The result has been produced in Figure 5.4.

From the above graph we can see that the UNREAL-PoTs model does better than the UNREAL+NFP model after about 50 million steps. This probably shows that learning a policy over the set of auxiliary tasks was in fact necessary to perform better data-efficient learning.

Figure 5.4: UNREAL vs UNREAL-PoTs(as in Algorithm 3) on Space Invaders

Algorithm 3 Policy over Tasks model - for each thread

Assume global shared parameter vectors θ, θ_{PoTs} and θ_v and global shared counter $T = 0$
Assume thread-specific parameter vectors θ', θ'_{PoTs} and θ'_v
Let B be a replay buffer storing experience for training. Fill buffer with transitions using a random policy.
Assume θ^c_{shared} be the shared parameters for the auxiliary task c with the base agent.
Initialize thread time counter $\tau \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$, $d\theta_{PoTs} \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread parameters $\theta' = \theta$, $\theta'_{PoTs} = \theta_{PoTs}$ and $\theta'_v = \theta_v$
    $t_{start} = \tau$
    Get state t_τ
    repeat
        Perform a_τ according to policy $\pi(a_\tau|t_\tau, \theta')$
        Execute policy $\pi(a_\tau|t_\tau, \theta'_{PoTs})$
        Receive reward r_τ and new state $t_{\tau+1}$
        $\tau \leftarrow \tau + 1$
        $T \leftarrow T + 1$
    until terminal state s_τ or $\tau - t_{start} == t_{max}$
    Based on $\pi(a_\tau|t_\tau, \theta'_{PoTs})$ choose auxiliary task c

$$R = \begin{cases} 0 & \text{for terminal state} \\ V(t_\tau; \theta'_v) & \text{for non-terminal state } t_\tau \end{cases}$$

    for $i \in \{\tau - 1, ..., t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt θ' : $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i; t_{i;\theta')(R - V(t_i;\theta'_v))}$
        Accumulate gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v + (R - V(t_i; \theta'_v))^2 / d\theta'_v$
    end for
    Perform auxiliary task c.
    Compute $L_{aux} = L_c$
    Accumulate auxiliary task gradients wrt θ' : $d\theta \leftarrow d\theta +_{aux} / d\theta'$
    Accumulate auxiliary task gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v +_{aux} / d\theta'_v$
    Perform asynchronous update of θ using $d\theta$ and θ_v using $d\theta_v$
until $T > T_{max}$

CHAPTER 6

FUTURE WORK

6.1 Shared Learning in Ensemble Deep Q-Networks

In this thesis, we have shown how *shared learning* is advantageous using the Bootstrapped DQN architecture. In the future, we would like to show our experimental results over more epochs and for more number of games. The deep reinforcement learning algorithm that is of current interest is the A3C algorithm (Mnih *et al.* (2016)). While it is not completely straightforward as to how to implement shared learning in this algorithm, we hypothesize that it will be very useful.

Additionally, in this paper, we have only discussed two ways of measuring learning progress, we would also like to look at how we can use better measures of learning to improve shared learning. For example, we believe that using a certain type of saliency-based measure could help in environments with sparse rewards since here the prime importance would be exploration rather than maximizing rewards.

We have also introduced the notion of online transfer in this thesis. We would like to study many more different applications of the online transfer in the future.

6.2 Advancing Unsupervised Auxiliary Tasks for Reinforcement Learning

While, we have run the models for only one game here. We will be testing out the models on many more Atari games.

Before proceeding to the future work on top of the work done in this thesis, we would also like to point out some other auxiliary tasks we would like to test out. The inverse dynamics model that can predict the the action taken given the two consecutive

frames is a great way for the agent to understand the kind of action it has taken. Hence, we believe that this task would be a great addition to the UNREAL agent. However, the ALE environment poses some problems for this task as the environment is slightly stochastic. An example of this is the stochastic appearance of predators in the Seaquest game. This will mislead the agent that is trying to understand an action. However, if we use our algorithm in the MuJoCo environment, we may be able to observe some significant improvement.

6.2.1 Next Frame Prediction as an Auxiliary Task

In this thesis, we have only considered a primitive version of next frame prediction where the model has as input the as stacked frames. In the future, we wish to consider models that perform better state encoding using an LSTM architecture. This also allows for more parameter sharing with the base A3C agent.

The next change we could bring about with this auxiliary task is to test models that predict more frames in the future. In the current setup, we use only a single frame predictor. Oh *et al.* (2015) has also done this task of many frame predictions. They were successful up-to predicting 100 frames. Hence, we have strong reason to believe that this model could help the base agent's representation. We could also try to use an LSTM encoding-decoding architecture as in the future predictor model in Srivastava *et al.* (2015).

Another task which maybe of particular advantage would be to try to predict not only the next frame, but the previous frame as well. This task is of particular use in tasks that have partially observable environments when the agent wants to understand where its current location is in an MDP. Prediction of the previous frame has also been shown to be helpful in Hausknecht and Stone (2015).

Generative Adversarial Networks (Goodfellow *et al.* (2014)) has been of great interest to researchers in artificial intelligence in the recent past. As a modelling change to our task, we would like to see how using an adversarial training as in Radford *et al.* (2015) can help in next frame prediction. Also, solving the task on a latent space (input to the discriminator of the generative adversarial model) will be of particular help

towards representation learning for the UNREAL agent.

Recently, in a deep learning workshop at NIPS, Prof. Honglak Lee, from University of Michigan, Ann Arbor discussed the importance of disentagled representations in getting better future state representations in reinforcement learning. This is also an exciting area of work linked with next frame prediction that could be done.

6.2.2 Sequencing Auxiliary Tasks for Reinforcement Learning

In this model, we have tried to predict the auxiliary task that needs to be executed at every possible step. Sharma *et al.* (2017) introduced a new method that selects how many time steps an action needs to be persisted by an agent. We could try to use the same strategy with our PoTs policy as well and test how auxiliary task persistence could help in extrinsic learning. This could particularly be computationally helpful and probably can help in further exploration with respect to the PoTs policy.

On the other end of the spectrum, we could also try to make sure that we never repeat an auxiliary task by making successive task representation vectors orthogonal. This has been tried out in the context of natural language processing Nema *et al.* (2017).

REFERENCES

1. **Barto, A. G.**, Intrinsic Motivation and Reinforcement Learning. *In Intrinsically motivated learning in natural and artificial systems*. Springer, 2013, 17–47.

2. **Bellemare, M.**, **S. Srinivasan**, **G. Ostrovski**, **T. Schaul**, **D. Saxton**, and **R. Munos**, Unifying Count-based Exploration and Intrinsic Motivation. *In Advances in Neural Information Processing Systems*. 2016.

3. **Bellemare, M. G.**, **Y. Naddaf**, **J. Veness**, and **M. Bowling** (2013). The Arcade Learning Environment: An evaluation platform for general agents. *J. Artif. Intell. Res.(JAIR)*, **47**, 253–279.

4. **Brafman, R. I.** and **M. Tennenholtz** (2002). R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, **3**(Oct), 213–231.

5. **Dayan, P.** (1993). Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, **5**(4), 613–624.

6. **Finn, C.**, **I. Goodfellow**, and **S. Levine**, Unsupervised learning for physical interaction through video prediction. *In Advances In Neural Information Processing Systems*. 2016.

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

8. **Hasselt, H. V.**, Double Q-learning. *In Advances in Neural Information Processing Systems*. 2010.

9. **Hausknecht, M.** and **P. Stone** (2015). Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*.

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

11. **Houthooft, R.**, **X. Chen**, **Y. Duan**, **J. Schulman**, **F. De Turck**, and **P. Abbeel**, Vime: Variational Information Maximizing Exploration. *In Advances in Neural Information Processing Systems*. 2016.

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

13. **Kearns, M.** and **D. Koller**, Efficient Reinforcement Learning in Factored MDPs. *In IJCAI*, volume 16. 1999.

14. **Kulkarni, T. D.**, **A. Saeedi**, **S. Gautam**, and **S. J. Gershman** (2016). Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*.

15. **Lample, G.** and **D. S. Chaplot** (2016). Playing fps games with deep reinforcement learning. *arXiv preprint arXiv:1609.05521*.

16. **Levine, S.** and **C. Finn** (2017). Deep visual foresight for planning robot motion. *ICRA*. URL `https://arxiv.org/abs/1610.00696`.

17. **Levine, S.**, **C. Finn**, **T. Darrell**, and **P. Abbeel** (2015). End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*.

18. **Li, X.**, **Z. C. Lipton**, **B. Dhingra**, **L. Li**, **J. Gao**, and **Y. Chen** (2016). A user simulator for task-completion dialogues. *CoRR*, **abs/1612.05688**. URL `http://arxiv.org/abs/1612.05688`.

19. **Li, Y.** (2017). Deep reinforcement learning: An overview. *CoRR*, **abs/1701.07274**. URL `http://arxiv.org/abs/1701.07274`.

20. **Lotter, W.**, **G. Kreiman**, and **D. Cox** (2016). Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*.

21. **Mirowski, P.**, **R. Pascanu**, **F. Viola**, **H. Soyer**, **A. Ballard**, **A. Banino**, **M. Denil**, **R. Goroshin**, **L. Sifre**, **K. Kavukcuoglu**, *et al.* (2016). Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*.

22. **Mirza, M.**, **A. Courville**, and **Y. Bengio** (2016). Generalizable features from unsupervised learning. *arXiv preprint arXiv:1612.03809*.

23. **Mnih, V.**, **A. P. Badia**, **M. Mirza**, **A. Graves**, **T. P. Lillicrap**, **T. Harley**, **D. Silver**, and **K. Kavukcuoglu**, Asynchronous Methods for Deep Reinforcement Learning. *In International Conference on Machine Learning*. 2016.

24. **Mnih, V.**, **K. Kavukcuoglu**, **D. Silver**, **A. A. Rusu**, **J. Veness**, **M. G. Bellemare**, **A. Graves**, **M. Riedmiller**, **A. K. Fidjeland**, **G. Ostrovski**, *et al.* (2015). Human-level Control through Deep Reinforcement Learning. *Nature*, **518**(7540), 529–533.

25. **Nema, P.**, **M. Khapra**, **A. Laha**, and **B. Ravindran** (2017). Diversity driven Attention Model for Query-based Abstractive Summarization. *ArXiv e-prints*.

26. **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.

27. **Osband, I.**, **C. Blundell**, **A. Pritzel**, and **B. Van Roy**, Deep Exploration via Bootstrapped DQN. *In Advances In Neural Information Processing Systems*. 2016*a*.

28. **Osband, I.**, **B. V. Roy**, and **Z. Wen**, Generalization and Exploration via Randomized Value Functions. *In Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. 2016*b*. URL `http://jmlr.org/proceedings/papers/v48/osband16.html`.

29. **Osband, I.**, **D. Russo**, and **B. Van Roy**, (more) Efficient Reinforcement Learning via Posterior Sampling. *In Advances in Neural Information Processing Systems*. 2013.

30. **Ostrovski, G.**, **M. G. Bellemare**, **A. v. d. Oord**, and **R. Munos** (2017). Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*.

31. **Radford, A.**, **L. Metz**, and **S. Chintala** (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

32. **Ranzato, M.**, **A. Szlam**, **J. Bruna**, **M. Mathieu**, **R. Collobert**, and **S. Chopra** (2014). Video (language) modeling: a baseline for generative models of natural videos. *CoRR*, **abs/1412.6604**. URL `http://arxiv.org/abs/1412.6604`.

33. **Rummery, G. A.** and **M. Niranjan**, *On-line Q-learning using connectionist systems*. 1994.

34. **Schaul, T.**, **D. Horgan**, **K. Gregor**, and **D. Silver**, Universal value function approximators. *In Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015*a*.

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

36. **Schulman, J.**, **S. Levine**, **P. Abbeel**, **M. I. Jordan**, and **P. Moritz**, Trust region policy optimization. *In ICML*. 2015.

37. **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*.

38. **Singh, S. P.**, **A. G. Barto**, and **N. Chentanez**, Intrinsically Motivated Reinforcement Learning. *In NIPS*, volume 17. 2004.

39. **Srivastava, N.**, **G. E. Hinton**, **A. Krizhevsky**, **I. Sutskever**, and **R. Salakhutdinov** (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, **15**(1), 1929–1958.

40. **Srivastava, N.**, **E. Mansimov**, and **R. Salakhutdinov**, Unsupervised learning of video representations using lstms. *In ICML*. 2015.

41. **Stadie, B. C.**, **S. Levine**, and **P. Abbeel** (2015). Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models. *CoRR*, **abs/1507.00814**. URL `http://arxiv.org/abs/1507.00814`.

42. **Sutton, R. S.** (1991). Integrated modeling and control based on reinforcement learning and dynamic programming. *Advances in neural information processing systems*, **3**, 471–478.

43. **Sutton, R. S.** and **A. G. Barto**, *Reinforcement Learning: An Introduction*, volume 1. MIT press Cambridge, 1998.

44. **Sutton, R. S.**, **J. Modayil**, **M. Delp**, **T. Degris**, **P. M. Pilarski**, **A. White**, and **D. Precup**, Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. *In The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

45. **Tang, H.**, **R. Houthooft**, **D. Foote**, **A. Stooke**, **X. Chen**, **Y. Duan**, **J. Schulman**, **F. D. Turck**, and **P. Abbeel** (2016). #Exploration: A Study of Count-based Exploration for Deep Reinforcement Learning. *CoRR*, **abs/1611.04717**. URL `http://arxiv.org/abs/1611.04717`.

46. **Tessler, C.**, **S. Givony**, **T. Zahavy**, **D. J. Mankowitz**, and **S. Mannor** (2016). A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*.

47. **van den Oord, A.**, **N. Kalchbrenner**, **L. Espeholt**, **O. Vinyals**, **A. Graves**, *et al.*, Conditional image generation with pixelcnn decoders. *In Advances in Neural Information Processing Systems*. 2016.

48. **Van Hasselt, H.**, **A. Guez**, and **D. Silver**, Deep Reinforcement Learning with Double Q-Learning. *In AAAI*. 2016.

49. **Wang, Z.**, **T. 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*.

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

LIST OF PAPERS BASED ON THESIS

1. **Rakesh R Menon***, Manu Srinath Halvagal*, Balaraman Ravindran, "Shared Learning in Ensemble Deep Q-Networks." *The 3rd Multidisciplinary Conference on Reinforcement Learning and Decision Making. (RLDM), 2017 (Accepted).*

2. **Rakesh R Menon***, Manu Srinath Halvagal*, Balaraman Ravindran, "Shared Learning in Ensemble Deep Q-Networks." *The 15th Adaptive Learning Agents (ALA) Workshop. AAMAS, 2017 (Accepted).*