'PRACTICE' FOR ENHANCING THE PERFORMANCE OF A DEEP REINFORCEMENT LEARNING AGENT

by

Venkata Sai Santosh Ravi Teja Kancharla

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Computer Science

Charlotte

2019

Approved by:

_____
Dr. Minwoo Lee

_____
Dr. Srinivas Akella

_____
Dr. Min C. Shin

ABSTRACT

VENKATA SAI SANTOSH RAVI TEJA KANCHARLA. 'Practice' for Enhancing the Performance of a Deep Reinforcement Learning Agent. (Under the direction of DR. MINWOO LEE)

Deep reinforcement learning has demonstrated its capability to solve a diverse array of challenging problems, which were not able to solve previously. It has been able to achieve human-level performance in Atari 2600 games and it has shown great potential to self-driving cars, robotics, and natural language processing. However, it requires long training time to learn feature representations from raw sensory data and makes it difficult to use deep reinforcement learning in real-world applications. Transfer learning, specifically sim-to-real transfer, has been suggested to learn features but designing of new environments or defining an environment with similar or relevant goals requires careful human efforts. This thesis aims to leverage practice approaches, which do not require a new environmental design for transfer learning, to reduce the time for training a deep reinforcement learning agent and enhance its performance.

First, this thesis introduces the use of practice approach applicable to end-to-end models with a deep reinforcement learning algorithm. It shows that the approach improves the performance of an agent and presents experimental results of this approach in complex environments. Second, this thesis presents a novel strategy, iterative practice, which repeats short period of practice and short period of learning until desired results are achieved. It presents experimental results and verifies that iterative practice improves learning. Last, it introduces a new strategy, shared experience for iterative practice, which reduces the interactions with an environment. It presents observations on this approach applied in complex environments and examines the effect of reduced interactions on learning.

DEDICATION

To my late father, Jagadiswara Rao Kancharla

ACKNOWLEDGEMENTS

Firstly, a special thank to my advisor, Dr. Minwoo Lee, not only for believing in my potential but also for his support and motivation. His suggestions and guidance cleared many roadblocks in my path to complete this thesis.

Secondly, I would like to thank Dr. Srinivas Akella and Dr. Min C. Shin for their valuable feedback at every milestone. Their enthusiasm in my work motivated me to put my best efforts to accomplish this. My sincere appreciation to the University of North Carolina at Charlotte for equipping me with necessary infrastructure and aiding me in the entire process.

This would not be possible without the quintessential support from my mother Saroja Kancharla, and my brother Prasanth Kancharla. I honestly thank them. A special credit to all my friends for keeping me motivated all the time.

TABLE OF CONTENTS

LIST OF TABLES ix

LIST OF FIGURES x

LIST OF ABBREVIATIONS xii

CHAPTER 1: INTRODUCTION 1

1.1. Problem Statement 2

1.2. Contributions 2

CHAPTER 2: BACKGROUND 4

2.1. Reinforcement Learning 4

2.1.1. Markov Decision Process 4

2.1.2. Policy, Value and Q-value 5

2.1.3. Model-free vs Model-based 6

2.1.4. On-Policy vs Off-Policy 6

2.1.5. Action Policies 8

2.2. Deep Learning 9

2.2.1. Artificial neural Networks 9

2.2.2. Deep Neural Networks 10

2.2.3. Convolution Neural Networks 10

2.2.4. Deep Reinforcement Learning 13

2.3. Transfer Learning 16

2.4. OpenAI Gym 18

CHAPTER 3: RELATED WORK 19

3.1. Transfer Learning in Reinforcement Learning 19

3.2. Practice 20

CHAPTER 4: PROPOSED METHODS 23

4.1. Practice for DQN 23

4.2. Iterative Practice 25

4.3. Shared Experience for Iterative Practice 28

CHAPTER 5: EXPERIMENTS AND RESULTS 30

5.1. Test Environments 30

5.2. Pre-processing for Atari Game Environments 32

5.3. Practice for DQN [1] 33

5.3.1. Visual Maze 33

5.3.2. Atari Game Environments 34

5.4. Iterative Practice 38

5.4.1. Visual Maze 38

5.4.2. Atari Game Environments 39

5.5. Shared Experience for Iterative Practice 43

CHAPTER 6: DISCUSSIONS 45

6.1. Observations of Shareable Representations 45

6.2. Generalization through Iterative Practice 48

6.3. Model Learning from Practice? 49

CHAPTER 7: CONCLUSIONS 50

7.1. Future Work 50

REFERENCES 52

LIST OF TABLES

TABLE 5.1: List of other hyperparameters and their values for Visual Maze 34

TABLE 5.2: List of other hyperparameters and their values for Atari Environments 36

LIST OF FIGURES

FIGURE 2.1: Reinforcement learning setup 4

FIGURE 2.2: Q-learning algorithm 7

FIGURE 2.3: SARSA algorithm 8

FIGURE 2.4: Artificial Neural Network 10

FIGURE 2.5: Deep Neural Network 11

FIGURE 2.6: CNN example 12

FIGURE 2.7: Convolution layer extraction 12

FIGURE 2.8: Pooling layer 13

FIGURE 2.9: DQN architecture 14

FIGURE 2.10: Transfer learning 16

FIGURE 2.11: Performance with transfer learning 17

FIGURE 2.12: OpenAI Gym environments 18

FIGURE 3.1: Practice network architecture 21

FIGURE 4.1: DQN network architecture for practice and target training 24

FIGURE 4.2: One step practice vs iterative practice 26

FIGURE 4.3: Shared experience 28

FIGURE 5.1: Visual Maze 30

FIGURE 5.2: Pong 31

FIGURE 5.3: Breakout 31

FIGURE 5.4: Freeway 32

FIGURE 5.5: Reward curve for practice DQN on Visual Maze 34

FIGURE 5.6: Epsilon for practice DQN 36

FIGURE 5.7: Reward curve for practice on Breakout 37

FIGURE 5.8: Reward curve for practice on Pong 37

FIGURE 5.9: Reward curve for practice on Freeway 38

FIGURE 5.10: Reward curve for iterative practice on Visual Maze 40

FIGURE 5.11: Epsilon for iterative practice 41

FIGURE 5.12: Reward curve for iterative practice on Breakout and Freeway 42

FIGURE 5.13: Reward curve for shared experience on Breakout and Freeway 44

FIGURE 6.1: Visualizing activations for Visual Maze 46

FIGURE 6.2: Visualizing activations for Breakout 47

LIST OF ABBREVIATIONS

ANN	An acronym for Artificial Neural Networks
CNN	An acronym for Convolutional Neural Networks
DQN	An acronym for Deep Q-Networks
FC	An acronym for Fully Connected
MDP	An acronym for Markov Decision Process
RARL	An acronym for Robust Adversarial Reinforcement Learning
RGB	An acronym for Red, Green, Blue
RL	An acronym for Reinforcement Learning
SARSA	An acronym for State Action Reward State Action
SBRL	An acronym for Sparse Bayesian Reinforcement Learning

CHAPTER 1: INTRODUCTION

Reinforcement Learning (RL) [1] is a computational approach to learn from interactions with an environment and attain a complex task. Some RL approaches showed their effectiveness by theoretically proving the convergence to an optimal solution for a task [1, 2, 3]. RL is successfully applied to diverse domains such as robotics [4], traffic network management [5], energy [6], and finance [7].

Deep reinforcement learning, which combines deep learning with reinforcement learning, has demonstrated its ability to solve diverse hard problems, which were not able to be solved previously. It has been able to beat human best players in Go (AlphaGo) [8], StarCraft (AlphaStar) [9], and poker games [10]. Also, it has shown great potential in application areas like natural language processing [11], resource management [12], and robotics [13]. These successful applications were achieved by addressing the long-lasted challenges in the curse of dimensionality.

One of the state-of-the-art models, Deep Q networks (DQN) [14, 15] have suggested an end-to-end model that enables an RL agent to learn directly from raw sensory data without tedious handcrafted feature engineering. Although a DQN is successful in solving complex problems, it takes an excessively long time to learn, as learning directly from raw sensory inputs that require large search space [16, 17]. Therefore, it needs a large number of steps of trial-and-errors until converge. These drawbacks are consequential when applying RL solutions to real-world applications.

Several approaches were proposed to address these problems. Transfer learning between two similar RL tasks was examined to improve the performance [16], however, it requires human effort for similar task selection. Imitation learning [18], though promises enhanced performance of agent, has its limitations like limited performance

improvement and also relying on human experts for demonstrations. Practice [19, 20] is a new paradigm of transfer learning, which overcomes the limitations of previous methods by discarding the need for human efforts. It showed improved learning efficiency with the knowledge gained from a non-RL source task is applied to an RL target task.

1.1 Problem Statement

Hypothesis: *The performance of the Deep Reinforcement Learning agents in solving reinforcement learning tasks can be improved with efficient practice (or pre-training) strategies.*

Although the efficacy of practice approach has been examined, applying it to end-to-end models with deep reinforcement algorithms, which learn from raw sensory inputs, is not well examined. Thus, the primary objective of this thesis is to develop *efficient practice strategies* applicable to deep reinforcement learning algorithms. The research is split into the following goals:

- Goal 1: Examining the efficacy of practice with a Deep Reinforcement Learning algorithm, DQN,
- Goal 2: Developing new practice strategies and examining their effectiveness on improving RL agent performance,
- Goal 3: Understanding practice via observation of shareable representations between practice and a target task.

1.2 Contributions

Practice for DQN: In this thesis, I extend practice to be applicable to end-to-end models with deep reinforcement algorithms. Practice is applied to the DQN algorithm and the performance is evaluated by testing the model with Atari 2600 games.

Iterative Practice: To further improve the efficiency of practice and learning, I present a novel strategy, iterative practice. Imitating human practice and learning model (we continuously repeat practice and learning), iterative practice repeats practice and short-term learning until it converges. The model is applied to DQN algorithm and empirical results are presented for this model by testing it with Atari 2600 games.

Shared Experience for Iterative Practice: Adding to the above models, I also present a strategy, shared experience for iterative practice, which effectively reduces the interactions between the deep reinforcement learning agent and the environment. This model is also applied to DQN algorithm and tested upon Atari 2600 games.

CHAPTER 2: BACKGROUND

2.1 Reinforcement Learning

Reinforcement learning is a machine learning paradigm which deals with the problem of making an agent perform actions in order to achieve a goal in a given environment. It is an approach drawn from the fields of statistics, psychology, and computer science, where an artificial agent is trained to resemble human behavior in action selection.

RL algorithms help an agent to learn how to behave in an environment. The agent learns to map states to actions so as to maximize the numerical reward signal by interacting with the environment and receiving rewards for performing actions. It is quite different from supervised learning because the agent doesn't have labeled examples available from which the correct action can be identified. Figure 2.1 shows how the agent interacts with the environment.

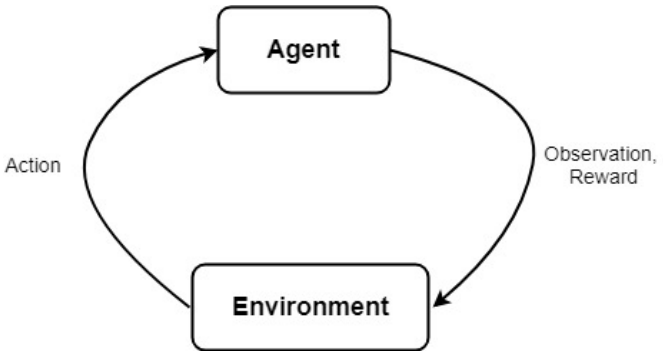


Figure 2.1: Reinforcement learning setup

2.1.1 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical framework for formalizing decision-making in an environment. An MDP is defined as a tuple of $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$

where

- $\mathcal{S}$: A set of possible states in the environment
- $\mathcal{A}$: A set of finite actions
- $\mathcal{P}$: A state transition probability, describing the probability of transition from state s to state s'. It is given by $\mathcal{P}^a_{ss'} = \mathcal{P}[\mathcal{S}_{t+1} = s'|\mathcal{S}_t = s, \mathcal{A}_t = a]$
- $\mathcal{R}$: A reward function which returns reward for performing action a in state s to move to state s' defined as $\mathcal{R}(s, s')^a$
- γ: a discount factor, which is a multiplicative factor for future rewards to denote their significance.

At each time step, the process is at some state s, and when action a is chosen, the process moves to new state s' at the next time step and gives a reward R_{t+1} defined by reward function $\mathcal{R}(s, s')^a$. The probability of moving to a new state is influenced by $\mathcal{P}^a_{ss'}$. Thus, the next state s' depends on current state s and action a and independent of all previous states and actions. This shows that MDP satisfies Markov property, which states that future is independent of past given present.

The main goal of an MDP is to find a policy (Section 2.1.2) for action selection that can maximize the reward obtained in the task.

2.1.2 Policy, Value and Q-value

A policy is a strategy that determines the possible action from a state. A policy can be any function, $\pi(a|s) = \mathcal{P}[\mathcal{A}_t = a|\mathcal{S}_t = s]$, mapping from states to actions. When an agent visits the states in the sequence $s_0, s_1...$ with actions $a_0, a_1...$ based on a given policy, the total discounted return or long-term cumulative reward can be defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

where R_{t+1} denotes the reward obtained at time $t+1$. An optimal policy is the policy with which the agent can get maximum cumulative reward in an environment. A value function V is the function which determines the long-term return with discount. For the given policy, the value function can be defined as:

$$V(s) = E[G_t|\mathcal{S}_t = s].$$

A Q-value function is similar to the Value function, only difference being that Q-value considers an extra parameter, the current action a. It can be defined as:

$$q_\pi(s, a) = E_\pi[G_t|\mathcal{S}_t = s, \mathcal{A}_t = a].$$

2.1.3 Model-free vs Model-based

A model is a simulation of the dynamics of the environment. In model-based algorithms, the agent learns the transition probability $\mathcal{P}^a_{ss'}$ which makes the agent know how likely that agent enters a specific state from the current state and action. Once the agent learns, it can use planning algorithms with the learned model to find optimal policy [1]. However, model-based algorithms become impractical as the state space and action space increases.

It is not necessary to learn a model of the environment to find an optimal policy [2]. The agent can instead learn the policy directly without estimating the dynamics of the environment, using some model-free algorithms. These algorithms rely on trail-and-error approach to update the knowledge. The agent learns from experiences, i.e. the agent interacts with the environment by performing an action and observes the reward and new state. As a result, it does not require to store all the combinations of states and actions.

2.1.4 On-Policy vs Off-Policy

An on-policy algorithm estimates the value of a policy while using the policy for control. In other words, an on-policy agent tries to learn the value based on the action derived from the current policy. The policy is usually 'soft' which means that

the policy has an element of exploration. The policy is not strict to choose the action which gives best reward. We will discuss more about these action policies below.

Off-policy algorithms use actions derived from a different policy which is unrelated to the policy that is estimated and improved. The policy used to derive actions is called behavioral policy, whereas the policy being evaluated is called estimation policy. Here, the behavior policy is 'soft', which enables the off-policy algorithms to separate exploration from control which on-policy algorithms cannot.

2.1.4.1 Q-Learning

Q-learning [1] is an off-policy, model-free algorithm based on the Q-value function approximation. The goal of the Q-learning algorithm is to maximize the Q-value. It iteratively updates the Q-value function using the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)].$$

Figure 2.2 outlines the Q-learning algorithm.

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε-greedy)
    Take action a, observe r, s'
    $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
    $s \leftarrow s'$;
  until s is terminal

Figure 2.2: Q-learning algorithm [1]

In the algorithm, α represents the learning rate which determines the extent to accept the new value vs the old value. When α is zero, we are leaving the old Q-value without updating it.

2.1.4.2 SARSA

State Action Reward State Action (SARSA) algorithm [1] is similar to Q-learning algorithm, but SARSA is an on-policy algorithm. In Q-learning, the future reward is the value of the highest possible action, but in SARSA, the future reward is the value of the actual action that was taken. This means that SARSA uses the control policy with which the agent is moving, to update the action values. Figure 2.3 outlines SARSA algorithm.

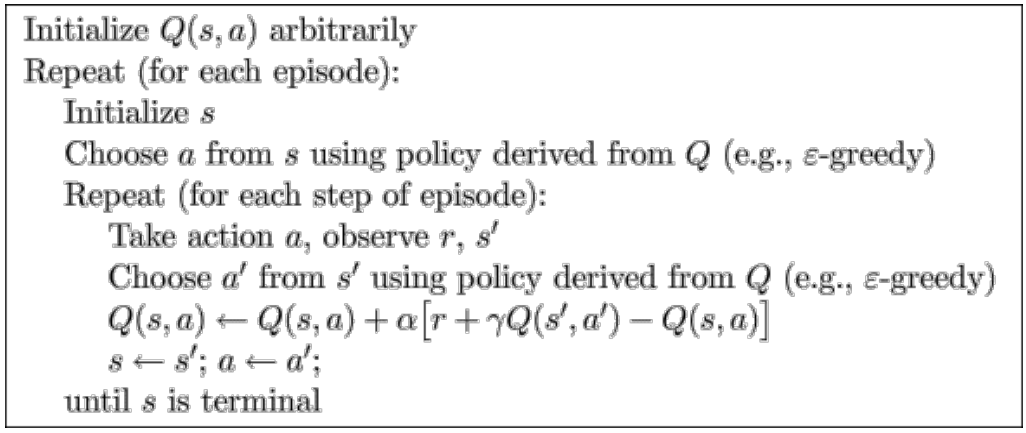


Figure 2.3: SARSA algorithm [1]

2.1.5 Action Policies

One of the challenges that arises in reinforcement learning is the trade-off between exploration and exploitation. The agent must prefer actions which are tried in the past and found to be effective (exploit) and at the same time it need to look for actions it has not selected before (explore). A right balance of both exploration and exploitation is required for the RL agent to learn successfully. One way to achieve a good balance is to try a variety of actions while progressively favoring the actions which can produce better rewards.

The three common policies used for action selection, which help in balancing the trade-off between exploration and exploitation:

- ϵ-Soft: In this, every action available for a state has at least $\frac{\epsilon}{|\mathcal{A}|}$ chance of being

selected for some $\epsilon > 0$.

- ϵ-Greedy: This is similar to ϵ-soft but with a greedy action selection. A random action is selected with a small probability of ϵ. Every other time a greedy action is chosen, i.e. the action which gives maximum reward. The random action is selected uniformly, independent of the action-value estimates.

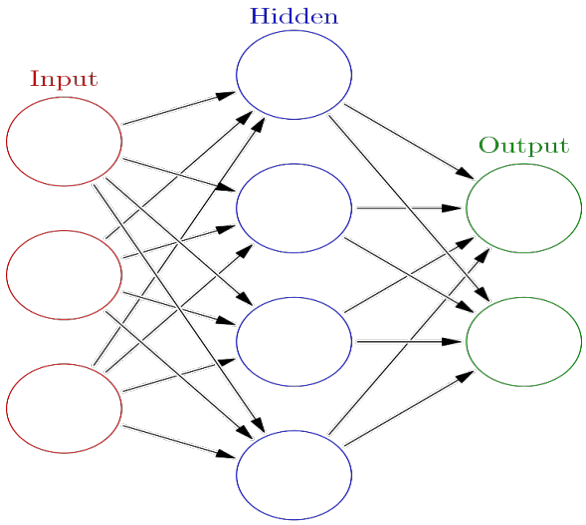
- Softmax: This method does not select random actions uniformly, but ranks each of the actions according to their action-value estimate and a random action is selected with regards to these ranks. This makes least ranked actions unlikely to be chosen.

2.2 Deep Learning

2.2.1 Artificial neural Networks

Artificial Neural Networks (ANN) [21] are a set of algorithms, inspired by the human brain, which are intended to replicate the way human brain learns. Artificial neurons are the elementary units of an ANN, which are mathematical functions conceived as a model of biological neuron. A neuron consists of variables called weights and biases. It takes an input, translates it into an output after multiplying with its weights and adding the bias.

A neural network consists of an input layer, an output layer and a set of hidden layers in between, which are all made up of multiple neurons (Figure 2.4). The neurons from each layer are connected with neurons of other layers to transfer outputs. The overall idea is that the input layer takes some inputs, which is carried through the hidden layers and finally an output is given by the output layer. The neural network creates a linear mapping from input to output. Certain activation functions like Sigmoid, TanH are used to infuse non-linearity in the neural networks, which makes the networks more flexible to learn mappings. Neural networks are used to recognize



Source: http://en.wikipedia.org/wiki/Artificial_neural_network

Figure 2.4: A representation of an artificial neural network

patterns in the data and thus can classify information, predict future outcomes or cluster data.

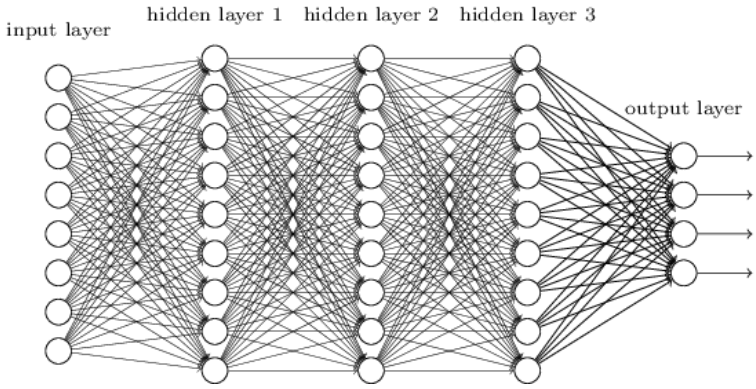
2.2.2 Deep Neural Networks

Deep neural networks [22] are artificial neural networks with more than one hidden layer. They have more depth, i.e. the number of layers the data has to go through before generating output (Figure 2.5). Earlier versions of neural networks are shallow, consisting of one input layer, one output layer and at the most one hidden layer in between.

The basic idea behind deep neural networks is that each layer learns distinct set of features from the data based on the previous layer's output. So, as the depth of the network increases, more complex features can be learned by the final layers as they aggregate features from the initial layers.

2.2.3 Convolution Neural Networks

A Convolution neural Network [23] is a Deep Learning model, which can read an input image, learn features of the image and can differentiate one image from another. These models have the ability to learn knowledge from raw form of data



Source: http://neuralnetworksanddeeplearning.com

Figure 2.5: A representation of a deep neural network

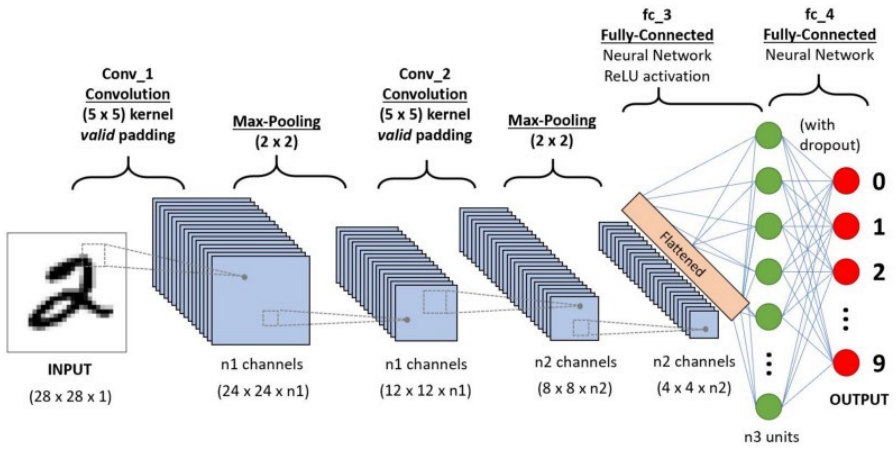
(images) without any pre-processing.

The architecture of CNN is similar to the connections in the human brain with the visual cortex. Individual neurons read only a restricted region in the image which is similar to Visual cortex's receptive field. Readings from each neuron overlap collectively to cover entire image.

CNN is able to capture the spatial and temporal dependencies in a given image which makes it preferable over normal feed forward network for images. The general architecture of CNN consists of series of convolution layer and pooling layer together followed by one or more fully connected layer(s) for classification. Figure 2.6 shows a CNN architecture which classifies hand written numbers from MNIST dataset [24].

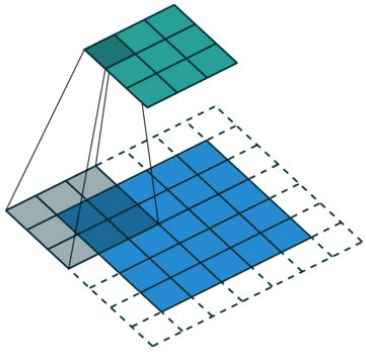
The convolution layer extracts the features of the image such as edges, color gradient. The initial convolution layers capture the low-level features of the image, while the final layers capture high-level features in the image. This layer uses filters, vector representation of weights, to convolve the input image. Figure 2.7 shows how a filter is applied on an image.

The pooling layer followed by a convolution layer is used to reduce the spatial size of the output from the convolution layer. This is performed to decrease the computations required to process the data. It extracts the dominant features in the data. Max pooling returns the maximum value from the portion of data defined by



Source: https://towardsdatascience.com/@_sumitsaha_

Figure 2.6: Convolution neural network for classifying hand written numbers (MNIST Dataset)

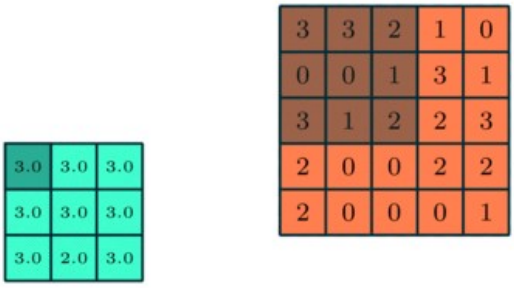


Source: https://towardsdatascience.com/@_sumitsaha_

Figure 2.7: Convolution layer extraction

pooling window size. Average pooling returns the average value of all the values in the data covered by the pooling window. Figure 2.8 depicts the working of max pooling layer with an example.

The fully connected layers at the end is used to generate the final output, i.e. classes for classifying the image. The output from the last pooling layer is flattened and sent as input to fully connected layer which outputs the predicted class.



Source: https://towardsdatascience.com/@_sumitsaha_

Figure 2.8: Functioning of pooling layer

2.2.4 Deep Reinforcement Learning

Applying RL to real-world problems, where the agents need to learn from a high-dimensional representation of data like images, is a challenging task. As deep learning has made it possible to extract high-level features from raw data like images, music, and videos, it is logical to think that applying deep learning techniques to RL can be beneficial while dealing with high-dimensional data. But this approach has some challenges.

The primary challenge is the availability of data for the deep learning models. Deep learning models required large amount of data to learn meaningful features from it. But, RL algorithms do not work that way. They depend on the scalar reward returned by the environment after performing an action. Another challenge is that deep learning models, generally, assume that data sample are independent and have fixed distribution, but in reinforcement learning the states of the agent are correlated. Also, the data distribution changes as the agent is learning better policies.

These challenges are addressed by the DQN algorithm [14, 15], which uses convolutional neural network to learn successful policies from raw images.

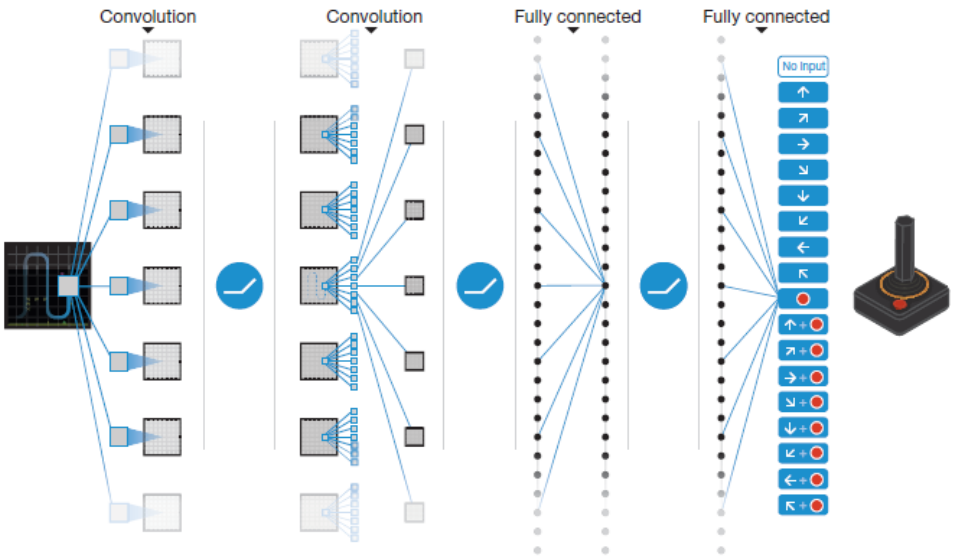


Figure 2.9: Deep Q-Network architecture [15]

2.2.4.1 Deep Q-Network

Deep Q-network (DQN) [14, 15] is a deep reinforcement learning algorithm which combines reinforcement learning with deep neural networks to learn directly from raw sensory data. The architecture of DQN comprises of convolutional neural network [25], which can learn encoded feature representations from raw input images.

To overcome the challenges we discussed above, the algorithm uses two key concepts:

- Experience replay: An agent's experiences are stored in a memory called experience replay buffer and random samples of data are sent as batches to the network during training. This removes the correlations between data and gives greater data efficiency. It reduces sampling bias in data towards some particular action by averaging the behavior distribution over many of its previous states.

- Target network: The algorithm uses two deep networks, one to retrieve Q values (online network) and the other includes all updates in the training (target network). After certain number of iterations, the weights from online network are

updated to target network. It allows the Q-value targets temporarily so that it does not have a moving target to chase, which makes the learning stable.

The algorithm makes Q-learning look like supervised learning. It uses a deep convolution neural network to approximate the optimal action-value function or Q-value function:

$$Q^*(s_t, a_t) = E_{s_{t+1}}[r(s_t, a_t) + \gamma Q^*(s_{t+1}, a_{t+1})],$$

where Q^* denotes optimal Q-value function.

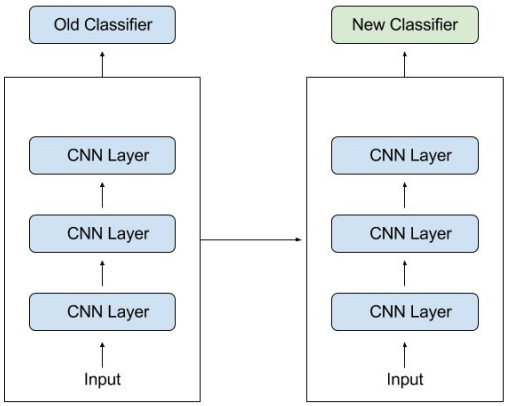
Similar to classic RL algorithms, DQN interacts with the environment and collects samples of state, action, reward and next state $(s_t, a_t, s_{t+1}, r_{t+1})$. These samples are stored into an experience replay buffer. Mini-batches of random samples from the buffer are used for training the online network Q^{online}. The mini-batch gradient update for online network minimizes the mean squared loss value defined by the function:

$$\mathcal{L} = \left(y_t - Q^{online}(s_t, a_t)\right)^2 \tag{2.1}$$

where the target y_t is defined by using the approximation from the target network Q^{target}:

$$y_t = r_t + \gamma \max_a Q^{target}(s_{t+1}, a). \tag{2.2}$$

As mentioned before, the target network is periodically updated with the weights of the online network. The model considers the state s_t as image pixels from the snapshot of the environment state at time t. The output layer is designed to output a single Q value for each valid action from the current state. The algorithm is applied to a wide range of Atari 2600 games with no adjustments in the architecture and it achieved human-level performance.



Source: https://indico.io/blog/exploring-computer-vision-transfer-learning

Figure 2.10: Transfer learning

2.3 Transfer Learning

Transfer learning [26] is a method where the model developed and trained on problem is reused as the starting point for a model for a different but related task. For example, a model's knowledge, which is trained to detect a car in an image, can be used by another model which needs to detect a red car in an image. This is very popular approach [27, 28, 29] in deep learning, especially in computer vision algorithms which take high amount of time and computational resources.

The general idea is to utilize the knowledge already learned in one task to improve generalization in other task. One of such approaches is simply transferring the weights of the neural network learned by the task to other neural network of the other task. Figure 2.10 shows how knowledge from one classifier is transferred to a new classifier through weight transfer.

The transfer of knowledge can be complete or partial, i.e. we can transfer only weights of some layers or weights of all the layers from one model. After the transfer, the layers transferred can be used without fine-tuning if the source task and target task are very close, or the layers can be fine-tuned to learn specific features of the target task. Here are some approaches used or transfer learning:

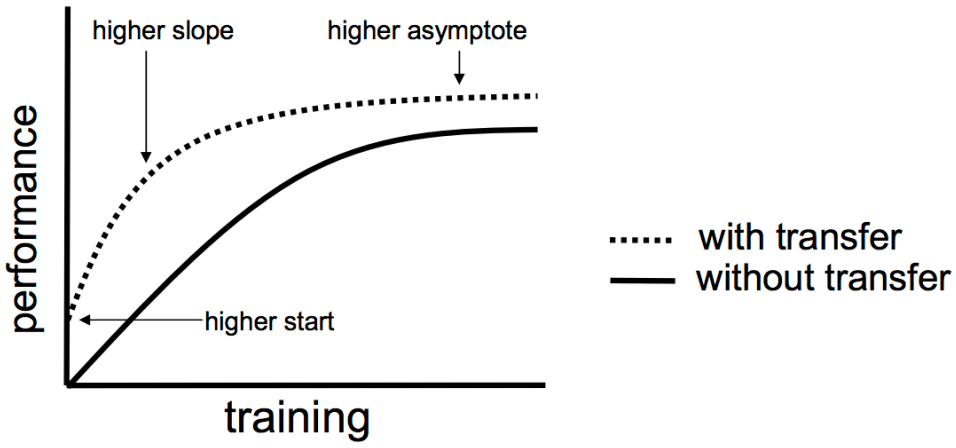
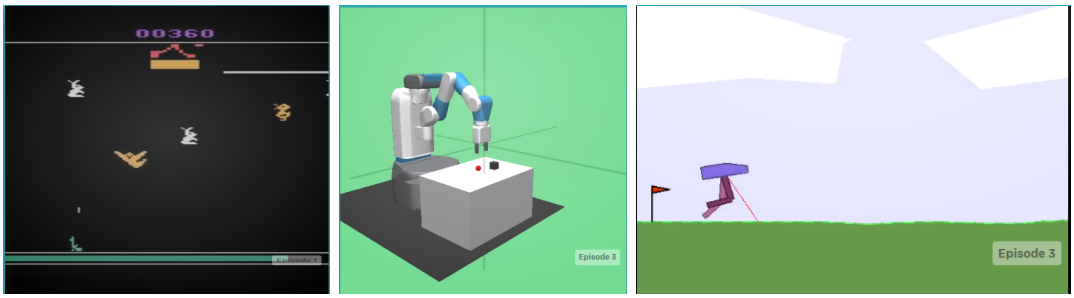


Figure 2.11: Performance comparison for transfer learning vs without transfer learning [26]

- Training a model to reuse it : In this approach, we will select a source task, develop a model and train it. We will then use this model to train for the target task. This approach is generally used when there is no sufficient data available for target task and source task has abundance of data available.

- Using a pre-trained model: In this approach, we will use an already pre-trained model available to our target task. Many pre-trained models are available online which can be used directly upon different target tasks.

Transfer learning helps in giving higher start, faster learning, and high asymptotes (Figure 2.11). But one of the major challenges in this approach is the selection of source and target task. When the source task is not similar to the target task, transfer learning will give negative effects as the model needs to unlearn its knowledge of source task and again start learning for target task. Measuring the similarity of task is still an open problem and requires domain expertise or intuition developed via experience.



(a) Atari Carnival (b) Fetch Pick and Place: Robotics (c) Bipedal Walker: Box2D

Source: https://gym.openai.com/

Figure 2.12: Example OpenAI Gym Environments

2.4 OpenAI Gym

A survey in 2016 [30], indicated that around 70 percent of researchers in reinforcement learning tried and failed to reproduce another researcher's experiments, and more than 50 percent have failed to reproduce their own environments. To overcome this problem of lack of standardization in test environments, and to create a better benchmarks by giving more varied environments, OpenAI [31] created Gym, a set of environments. It aims to increase reproducibility in this field and provide tools which can be easily set up.

Gym is a toolkit developed by OpenAI, which is used for developing and evaluating reinforcement learning algorithms. It is a collection of test problems or test environments that can be used for training the reinforcement learning algorithms. These environments have shared interface which allows us to use with general algorithms. It has variety of environments like Algorithmic, Atari games, Box 2D, and Robotics. Figure 2.12 shows some example OpenAI Gym environments.

CHAPTER 3: RELATED WORK

DQN algorithm is verified to be successful in solving complex problems like Atari games [15], computer vision problems [32]. One limitation for this method is that the training time to train a task is prohibitively long, as learning directly from raw images requires large search space. It thus requires a large number of trail-and error steps until it converges. These drawbacks are consequential when RL models are applied to real world applications.

3.1 Transfer Learning in Reinforcement Learning

Transfer learning is one good solution to speed up the learning. The core idea of transfer learning is that knowledge gained in one task can help in improving the performance in a related, but different task. In reinforcement learning, transfer learning can be performed between two RL agents, one trained on a source task and other on the target task [33]. The knowledge shared can be a policy, Q function, actions or model. In case of deep reinforcement learning, features learned in the neural network can be transferred. Based on this idea, transfer between models trained on different Atari games was examined and results showed improved learning speed [16]. However, transfer learning requires human efforts to define source and target tasks for effective transfer of knowledge between them.

The efficiency of learning can also be improved through imitation learning or apprenticeship learning [18], which is a kind of transfer learning that learns from an expert's demonstration. It trains an agent to match the performance of a human expert demonstrator in a given RL task. Recently, Deeply AggreVaTed [34] extended imitation learning to work with deep neural networks. However, it requires the ex-

pert's presence to provide proper feedback. Also, it is known that the transferred policy cannot exceed the demonstrated policy. Deep Q-learning from demonstrations (DQfD) [35] suggests a pre-training approach to overcome the limitation of learning from demonstrations in Deeply AggreVaTed. The pre-training stage uses temporal difference loss along with supervised loss to learn a policy and to imitate the demonstrator. This method effectively combines imitation learning with RL which enables an agent to develop a superior policy than the expert. It showed acceleration in learning, but still rely on human experts to gather demonstration data.

3.2 Practice

Practice [19, 20] suggests a new paradigm of transfer learning that does not need an expert's help. It is an approach that discovers shareable knowledge representations between a source non-RL task and a target RL task to solve the complex target task efficiently. Practice showed that learning the dynamics of an environment accelerates the learning of deep RL agents [19]. Figure 3.1 explains the neural network architecture that implements practice to obtain suitable knowledge (network weights w) and use it to initialize the Q function network to learn in a target environment. A single neural network is used for both practice and target training, whose architectures comprise of multiple densely connected layers. Two different output layers are used, one gives the state difference during practice and other gives the Q-values for the actions during target training.

To find shareable knowledge, practice sets up a regression problem that predicts the state changes given a state. For each time step t, it samples the state transition (s_t, a_t, s_{t+1}), and learns a function, $f(s_t, a_t) \approx \Delta s_t = s_{t+1} - s_t$. The inconsistency between predicted and the target state difference is defined as the loss function. The network is trained to minimize the mean squared loss:

$$\mathcal{L} = (\Delta s_t - f_(s_t, a_t))^2. \tag{3.1}$$

Practice does not involve any RL-associated variables like rewards and goals.

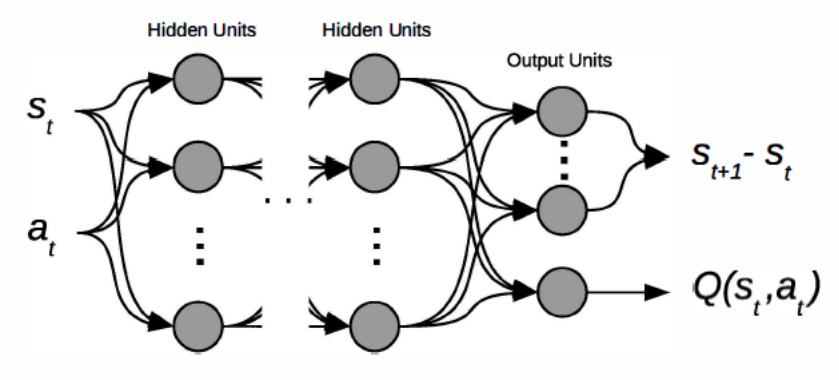


Figure 3.1: Practice model for pre-training neural networks from state change prediction [19]

The learned weights from practice are used as initial values for the target network. Interacting with a target environment, the RL agent collects the state transition samples with feedback $(s_t, a_t, r_{t+1},\ s_{t+1}, a_{t+1})$. Mini-batches of the samples are used for RL training with temporal difference update. RL training fine-tunes the network weights to correctly estimate the Q values. The mean squared loss is defined from SARSA update:

$$\mathcal{L} = (r_{t+1} + \gamma Q_(s_{t+1}, a_{t+1}) - Q_(s_t, a_t))^2.$$

Practice has shown improved learning efficiency by reducing the time-to-convergence and by achieving higher asymptote in diverse problems. Moreover, it does not require human efforts for defining similar source and target tasks for transfer learning.

To verify the approach, Sparse Bayesian Reinforcement Learning (SBRL) [20] explained how the knowledge obtained from practice helps to learn a target task by showing bases construction process in practice and their transfer and use of the transferred bases in target learning.

Another Practice approach designed for end-to-end Deep RL models learns some of the crucial features by pre-training deep RL network's hidden layers using supervised learning on a small set of human demonstrations [17], showed significant improvement in the agent's performance. The approach is successfully applied to DQN and Asyn-

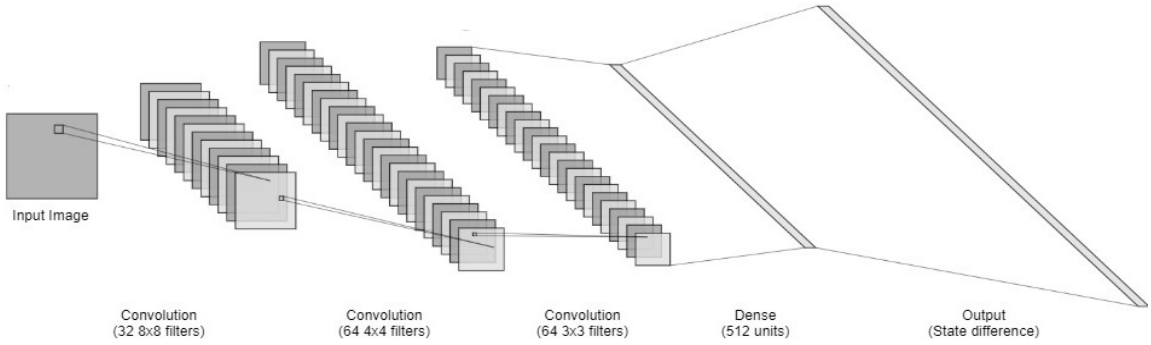
chronous advantage actor-critic (A3C) algorithms on Atari 2600 games. Although this model integrated practice approach with end-to-end models, it still requires human efforts for generating demonstration data.

CHAPTER 4: PROPOSED METHODS

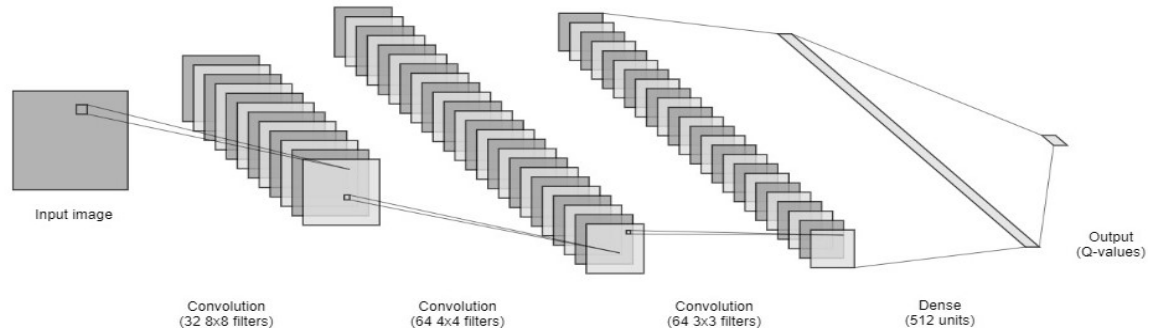
Practice has shown that, without an expert's help, an agent can learn faster in a target task from the knowledge gained from a non-RL task. Practice is also examined to be successful with end-to-end models, which learn from raw sensor inputs, with an approach that uses human demonstrations. In this thesis, I suggest a new approach for practice with end-to-end models which does not require human effort. For this, I extend the practice approach, which predicts state differences, to be applicable to models that use DQN algorithm. I also propose a novel strategy to further improve the learning of an agent. It is called *iterative practice*, which repeats practice and short-term training until it converges. Adding to these, I present a strategy, called *shared experience for iterative practice*, which effectively reduces the interactions between agent and environment. These models are applied to the DQN algorithm.

4.1 Practice for DQN

We leverage the existing practice approach and speed up learning in DQN. Figure 4.1 shows the DQN architecture for both practice and target RL training. The architectures of DQN and practice networks have an identical setup of 3 convolution layers (32 8x8 filters, 64 4x4 filters, 64 3x3 filters) followed by a fully connected layer (512 hidden units). It is understood that practice is trained to solve a non-RL task. Therefore, in my model, practice sets to solve a regression problem to predict the state change given the current state. So, the output for practice is not the next possible state, but the state difference between the current state and next possible state. Also, as there is no policy to follow for choosing actions based on states, all actions are taken randomly in the practice stage. Unlike the original practice approach [19],



(a) Practice Network Architecture



(b) RL Training Architecture

Figure 4.1: DQN network architecture for practice and target training

here action a_t is not fed into the network along with current state s_t, but it is used during the calculation of loss. The output for the target training is the same as the traditional DQN method, which is the Q-values for each possible action.

Since the model learns directly from raw images, image pixels are considered as a state. The pixel value from the snapshot of the game or task at time t is considered as the state s_t. The model first performs practice followed by RL training. For practice, n_{pr} number of samples of (s_t, a_t, s_{t+1}) are collected and stored in a practice memory. Mini-batches of these samples are used to train the practice network. The network is trained to estimate the state differences $\Delta s_t = s_{t+1} - s_t$ for all possible next states with each action. From this, only the state difference with action a_t is considered,

which will be the predicted state difference. The target state difference will be the difference in the pixel values of s_{t+1} with s_t. We use the mean squared loss (Eq. 3.1) to measure the inconsistency between predicted and target state difference. The network is trained with Adam optimizer to minimize this loss. The practice network is trained for a duration of $d_{pr}^{(0)}$.

After the practice network finishes its training, the weights of the RL training network $\mathbf{\Theta}_{tr}^{(0)}$ are initialized with the weights of the practice network weights $\mathbf{\Theta}_{pr}^{(0)}$. During the RL training, samples of s_t, a_t, a_{t+1}, reward r_{t+1} are collected in a different experience replay memory. Unlike practice, RL training collects reward values. Therefore, a separate replay memory is used for RL-training. Experiences from this buffer are randomly sampled and fed to the RL training network. We use Huber loss for training which can be described as:

$$L_\delta(y_t, Q_t) = \begin{cases} \frac{1}{2}(y_t - Q_t)^2 & \text{for } |y_t - Qt| \leq \delta \\ \delta|y_t - Qt| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{4.1}$$

where y_t is the target output defined in Eq. 2.2 and $Q_t = Q^{online}(s_t, a_t)$. δ is the control parameter which can be tuned. The loss is minimized using RMSprop optimization [36]. The network is trained to reduce the reduce the loss for $d_{tr}^{(0)}$. The training can be stopped when the model converges. This model is also called one step practice, as we only perform practice once before beginning of RL training.

4.2 Iterative Practice

We, humans, generally perform short cycles of practice and actual task, as we have limited attention span. Moving back and forth from actual task increases productivity compared to prolonged execution of actual task [37]. Motivated by this human practice and learning model, I developed a practice framework that iterates over short practice and short RL training until the model converges. Iterative practice

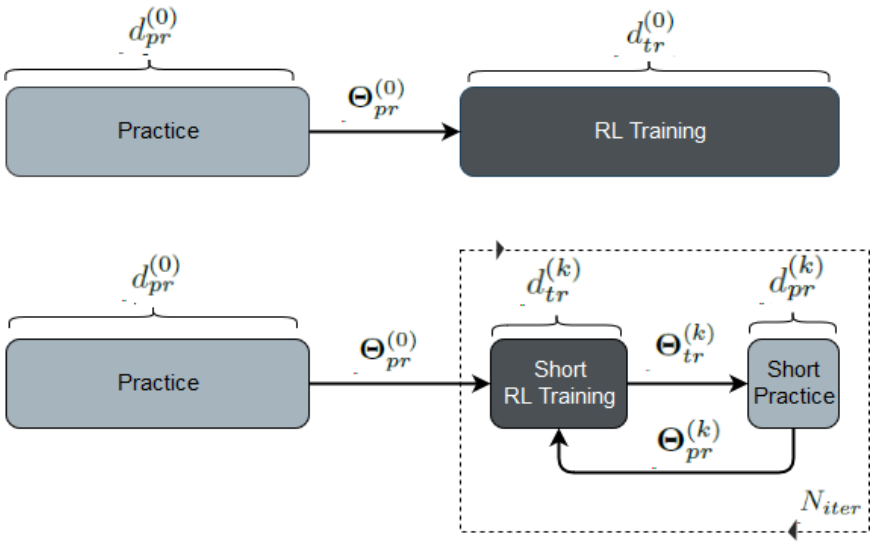


Figure 4.2: One Step Practice(top) vs Iterative Practice(bottom)

fully leverages the benefits of one step practice (Practice for DQN model) and helps learning quickly during initial phase of training.

Comparing with one step practice, Figure 4.2 illustrates the stages in iterative practice. The iterative model has two stages, initial practice and iterative training. Initial practice is similar to the practice stage in one step practice and network architectures will not have any modifications. Here, the same practice network is used for both initial practice and short practice. The network architectures for practice and target training are implemented the same as shown in Figure 4.1.

In this method, I used the same regression problem of predicting the state change given the current state. Both initial practice and short practice are trained to solve the same regression problem. The first stage in this model is initial practice, for which n_{pr} number of samples are collected by taking random actions and stored in the practice memory. Mini-batches of these samples are used to train the practice network, which predicts the state difference Δs_t. We again use the mean squared loss (Eq. 3.1) to measure the inconsistency between predicted and target state difference. The network is trained with Adam optimizer to minimize this loss for a duration of $d_{pr}^{(0)}$.

Algorithm 1 Iterative Practice

Input the duration of initial practice $d_{pr}^{(0)}$, short practice $d_{pr}^{(k)}$, and short target training $d_{tr}^{(k)}$
Initialize learning rate α, discounting factor γ
Collect practice n_{pr} samples with random actions.
for $k \leftarrow 0, N_{iter}$ **do**
  Train practice network with n_{pr} samples for $d_{pr}^{(k)}$ (Eq. 3.1)
  Transfer weights from practice to target (Eq. 4.2)
  Train target training network for $d_{tr}^{(k)}$ (Eq. 4.1)
  Transfer weights from target to practice (Eq. 4.3)
end for

After initial practice, weights of layers $\mathbf{\Theta}_{pr}^{(0)}$ from practice network are transferred to the target training network. Now, the iterations of short RL training and short practice begin. Short RL training collect new experiences s_t, a_t, a_{t+1}, reward r_{t+1} and stores in a different replay buffer. Following short practice does not collect new data but use the same data from the replay buffer used in initial practice.

Let the number of iterations for which short RL training and short practice are run, be N_{iter}. The target training network trains for duration of $d_{tr}^{(k)}, k \in [1, N_{iter}]$ where $d_{tr}^{(k)} \ll d_{tr}^{(0)}$. We use Huber loss for training which is described in eq 4.1, which is minimized using RMSprop optimization [36]. Trained target network weights $\mathbf{\Theta}_{tr}^{(k)}$ are transferred to the practice network weights $\mathbf{\Theta}_{pr}^{(k+1)}$ for short practice. The short practice lasts for $d_{pr}^{(k)}$ where $d_{pr}^{(k)} \ll d_{pr}^{(0)}$. Then the weights from practice network are transferred back to the target network. This alternative transfer continues incrementally as follows:

$$\mathbf{\Theta}_{tr}^{(k)} \leftarrow \mathbf{\Theta}_{pr}^{(k)}, \tag{4.2}$$

$$\mathbf{\Theta}_{pr}^{(k+1)} \leftarrow \mathbf{\Theta}_{tr}^{(k)}. \tag{4.3}$$

This process continues for N_{iter} iterations or until the model converges, which ever occur first. Here, we use the same short practice duration ($d_{pr}^{(i)} = d_{pr}^{(j)}, \forall i, j \in [1, N_{iter}]$) and the same short training duration ($d_{tr}^{(i)} = d_{tr}^{(j)}, \forall i, j \in [1, N_{iter}]$). Algorithm 1 summarizes the short iteration of practice and target train.

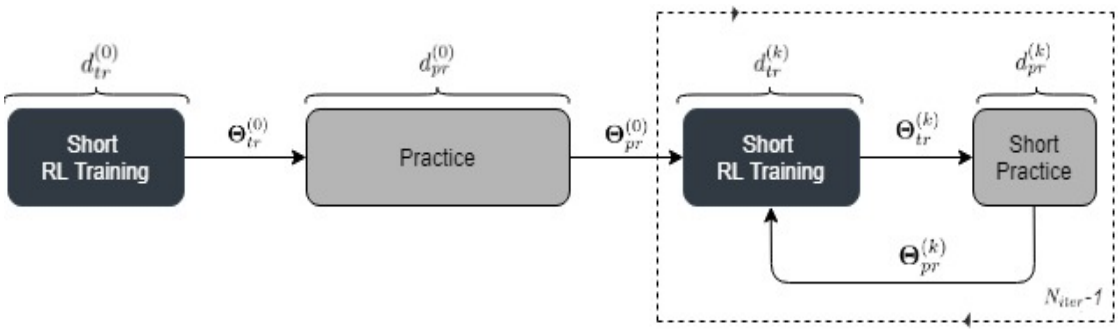


Figure 4.3: Delayed Practice for Shared Experience in Iterative Practice model

4.3 Shared Experience for Iterative Practice

Practice is a supervised learning model and does not involve any RL-associated variables like rewards and goals. Therefore, in practice for DQN and iterative practice, two memories are used for storing experiences, one to store and retrieve experiences (s_t, a_t, s_{t+1}) during practice, and the other for the experiences $(s_t, a_t, r_{t+1}, s_{t+1})$ for the target training. Thus, collecting additional experiences during practice and iterative practice, increases the interactions of the agent with the environment.

During practice, the agent takes random actions without depending on any policy. In environments like Breakout game, where the game almost immediately ends if the agent performs random actions, the practice memory is filled with experiences covering a small set of experiences near starting states of the game.

To overcome these drawbacks, I developed a strategy called *shared experience*, which increases the data distribution for practice without additional interactions with the environment. Here, I use a common experience replay memory for both practice and target training. As practice doesn't store experiences with reward value, those experiences cannot be used for target training. The experiences from the target training, however, can be used for practice by masking the reward details. To make this strategy feasible, I introduced a method called *delayed practice*. Figure 4.3 illustrates the stages in delayed practice for shared experience in iterative practice model. In this method, the agent starts with target training instead of practice, which is called

initial training. After training for a duration of $d_{itr}^{(0)}$, weights from target training $\mathbf{\Theta}_{tr}^{(0)}$ are used to initialize practice network weights $\mathbf{\Theta}_{pr}^{(0)}$ and practice network start training with mini-batches of (s_t, a_t, s_{t+1}) from the same replay buffer.

The practice network is trained for a duration of $d_{pr}^{(0)}$. After the practice network finishes training, the weights of the RL training network $\mathbf{\Theta}_{tr}^{(0)}$ are initialized with the weights of the practice network weights $\mathbf{\Theta}_{pr}^{(0)}$ and the RL training resumes again. When applied with the iterative practice model, it starts with short RL training instead of initial practice. The first short practice is replaced with initial practice, and the following short RL training and short practice will as usual.

CHAPTER 5: EXPERIMENTS AND RESULTS

5.1 Test Environments

The efficiency and efficacy of extended practice for DQN and iterative practice are tested Visual Maze and Atari game environments. Details about each of the environment used in this thesis are presented below.

Visual Maze : It is a navigation (8x8/16x16 blocks) task having RGB image representations as states (80x80x3/160x160x3). An agent starts from the start position (pink block) and moves towards the goal position (green block). The maze has obstacles in between, which are represented by black blocks. The environments have 4 discrete actions, left, right, up and down. The reward -1 for each movement, -5 for hitting a block or moving out of boundary, or $+30$ for reaching the goal is given to the agent. The task terminates when the agent reaches the goal.

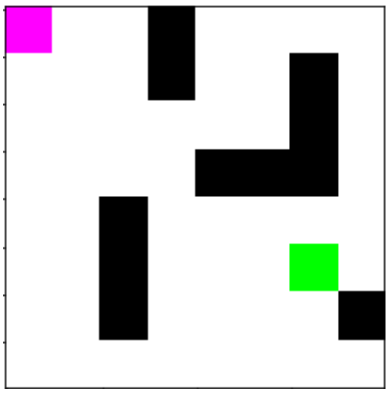


Figure 5.1: Visual Maze(8x8)

Pong : In this environment, the agent controls the green paddle to bounce the white ball back to the left while not passing it to right. The game has 5 possible actions, stay still, up, down, move up faster, and move down faster. The reward is defined

as +1 if the ball goes past the brown paddle (bot controlled) to the left and −1 if the ball goes past green paddle to the right, which means +1 for the bot. The game terminates when either the bot or agent gets 21 reward points.

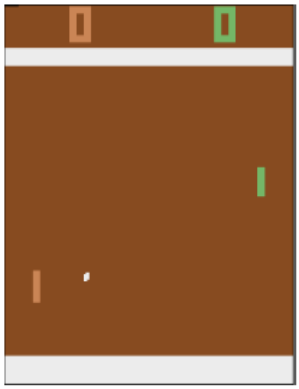


Figure 5.2: Atari 2600 Pong

Breakout : The objective of this game is to clear the bricks at the top by hitting each brick with the ball. Here the agent controls the paddle at the bottom to keep the ball in play. The game has three actions, stay still, left, and right. The reward is defined as $+1/+4/+7$ if the ball hits a brick (based on color of brick). The game terminates when paddle misses the ball and it goes down and out of the screen. The total reward will be the score before the game ends.

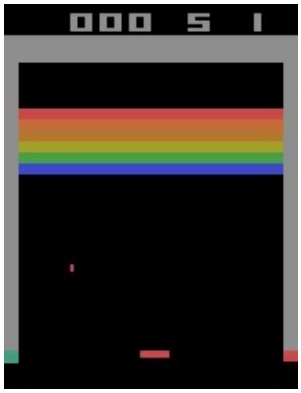


Figure 5.3: Atari 2600 Breakout

Freeway : The objective of the game is to make the chicken cross the ten lane highway from one side to other side. The highway is filled with traffic which the

chicken should avoid. The player gets a point every time a chicken gets across to other side. If hit by a car, the chicken is forced back slightly or entirely down. The player can score as much as he can in 2 minutes 16 seconds. The game has 2 actions either to jump ahead or stay still. The game terminates after the time runs out.

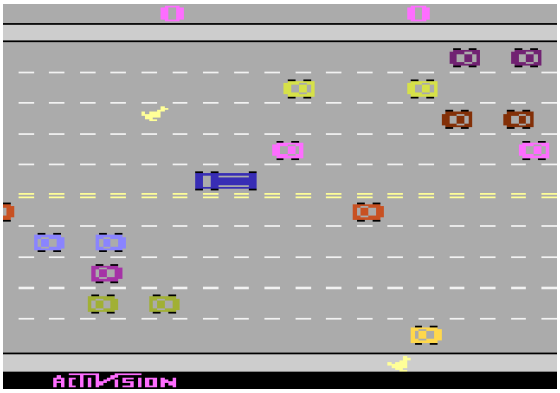


Figure 5.4: Atari 2600 Freeway

5.2 Pre-processing for Atari Game Environments

Atari game simulations are implemented using the OpenAI Gym library. It provides raw frames, which are 210 x 160 pixel images with a 128-color palette. This can be demanding in terms of computation and memory requirements. To reduce the input dimensionality, I used Atari wrappers provided by OpenAI Baselines [38], to re-scale the images to 84x84 and also make them gray-scale.

Following the previous approaches to playing Atari 2600 games, I also use a frame-skipping technique based on the frame-skipping parameter, which is set to 4. This means that only every 4^{th} frame is considered and remaining are skipped. Atari games are rendered at a rate of 60 frames per second and it is not needed to calculate Q values for every frame. This reduces the computational cost and gathers more frames.

Also, the the most recent four frames (after frame skipping) are pre-processed and stacked together to be used as an input for the training network. This will allow the agent to choose action depending on the prior sequence of game frames. For example, in Breakout, knowing the direction of the ball gives better idea regarding next action

than just knowing the ball position. The same strategy is applied while training the practice network.

5.3 Practice for DQN [1]

In this section, the experimental setup and results for the Practice for DQN model are presented. For comparison, the base DQN with no practice model is also implemented and tested on the same environments.

5.3.1 Visual Maze

Practice for DQN is first tested on the Visual Maze task. It is a simple environment compared to the other environments used in this thesis. For the practice stage in this task, the practice network is trained for $d_{pr}^{(0)} = 10^4$ steps. Mini-batches of 50 samples are used to train the practice network. For this task, I assigned the weights of the three convolution layers to the $\mathbf{\Theta}_{pr}^{(0)}$, which means that the weights of the 3 convolution layers are transferred to the RL training network. After practice, weights $\mathbf{\Theta}_{pr}^{(0)}$ are transferred to the target training network. As there is no transfer of the weights of the dense layer in training network, they are initialized randomly. Mini-batches of size 50 are fed to the training network which is run for $d_{tr}^{(0)} = 300$ episodes, with each episode allowing a maximum of 500 steps for the agent to reach the goal. The accumulated reward after each episode is collected. The list of other hyperparameters used for this task are presented in Table 5.1

Figure 5.5 depicts the accumulated rewards earned from training base DQN with no practice and DQN with practice (Practice DQN) in simple 8x8 Visual Maze environment. Assuming the threshold performance value as 0, we can observe that practice with DQN reaches threshold (dotted line) earlier than base DQN. The asymptotic performance, which is the overall performance of a model after training, cannot be compared in this case, as the maximum possible reward is limited and both the mod-

[1] Source code for the proposed models is available at https://github.com/kvssraviteja/Practice-for-Deep-Reinforcement-Learning.git

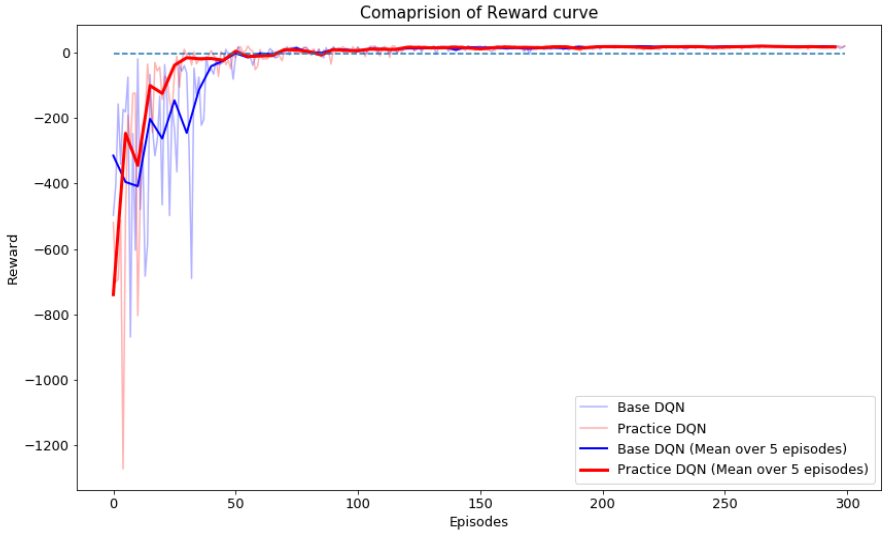


Figure 5.5: Reward curve for Practice with DQN and Base DQN (without practice) in Visual Maze (8x8).

els achieved that value after training. So, it can be seen that practice helps to learn the optimal solution faster than the base DQN.

Table 5.1: List of other hyperparameters and their values for Visual Maze

Hyperparameter	**Value**
Mini-batch size	50
Replay buffer size	1000
Target network update frequency	1
Discount factor	0.9
Learning rate (practice)	0.0001
Learning rate (RL training)	0.001
Epsilon	0.99 times episode number

5.3.2 Atari Game Environments

To test the performance of the model on complex problems, I moved on to the Atari 2600 games: Breakout, Pong and Freeway. For the practice stage in these tasks, the

practice network is trained for $d_{pr}^{(0)} = 10^6$ steps. Mini-batches of 64 samples are used to train the practice network. Similar to the Visual Maze task, I assigned the weights of the three convolution layers to the $\mathbf{\Theta}_{pr}^{(0)}$. After practice, weights $\mathbf{\Theta}_{pr}^{(0)}$ are transferred to the target training network and the weights of fully connected layer are initialized randomly. Mini-batches of size 64 are fed to the training network which is run for $d_{tr}^{(0)} = 2 \times 10^6$ steps, but early stopping is performed when convergence is achieved. During the training, once the agent reaches the end of a game, it is recorded as an episode and the accumulated reward for that episode is also recorded. The game is then reset to start position for the next iteration. There are no adjustments in the network architecture or hyper-parameters for each Atari environment. The epsilon ϵ (exploration rate) used in this model compared to base DQN model is presented in Figure 5.6. The exploration rate is brought down from 1.0 to 0.1 after less number of steps compared to the base DQN model, as this model already learned features from practice and requires less exploration. The list of other hyperparameters, used in this model, and their values are given in Table 5.2.

Here, the model is trained for very high number of episodes when compared to the previous task and the reward after each episode is highly fluctuating. Therefore, to better understand the performance of the agent, I considered the mean reward for the last 100 episodes as an evaluation metric. Figure 5.7, 5.8 and 5.9 depicts the accumulated mean rewards from training base DQN with no practice and DQN with practice (Practice DQN) in Breakout, Pong and Freeway.

For Pong and Freeway, in Figure 5.8 and 5.9, it can be observed that the practice for DQN model reached the threshold value earlier than the base DQN model. As the maximum reward that can be obtained in these environments is limited, asymptotic performance cannot be compared. In Breakout, Figure 5.7, in addition to achieving threshold (lower dotted line) earlier its asympotic performance is very high compared to the baseline. The lower dotted line represents the convergence level of base DQN

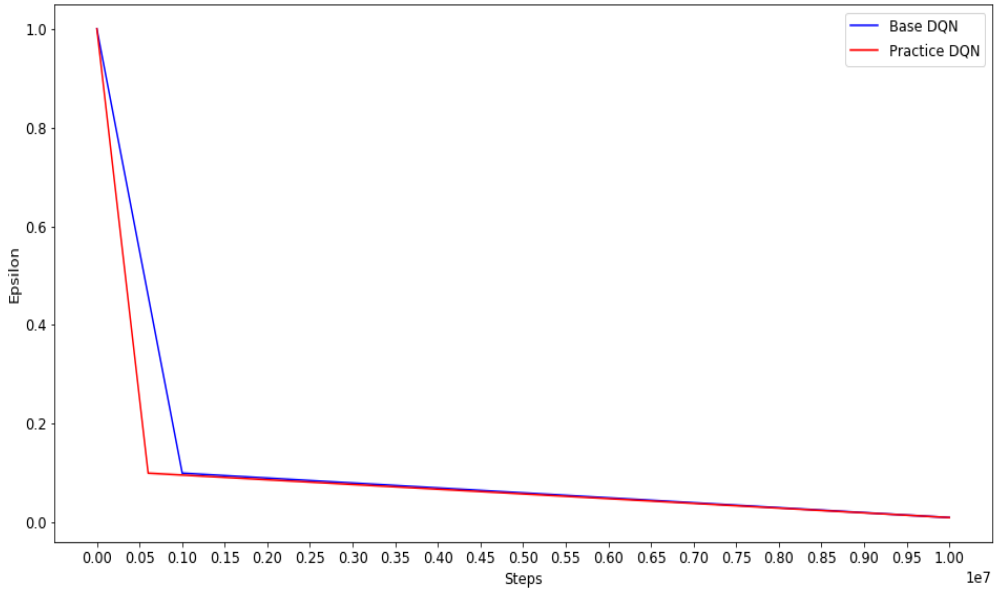


Figure 5.6: Epsilon used for practice DQN and base DQN

model, and the upper dotted line represents convergence level of practice for DQN model. It shows that even in complex problems, practice helps in faster learning and higher reward (Breakout).

Table 5.2: List of other hyperparameters and their values for Atari Environments

Hyperparameter	**Value**
Mini-batch size	64
Replay buffer size	1000000
Frame stack size	4
Target network update frequency	10000
Discount factor	0.99
Learning rate (practice)	0.001
Learning rate (RL training)	0.0001

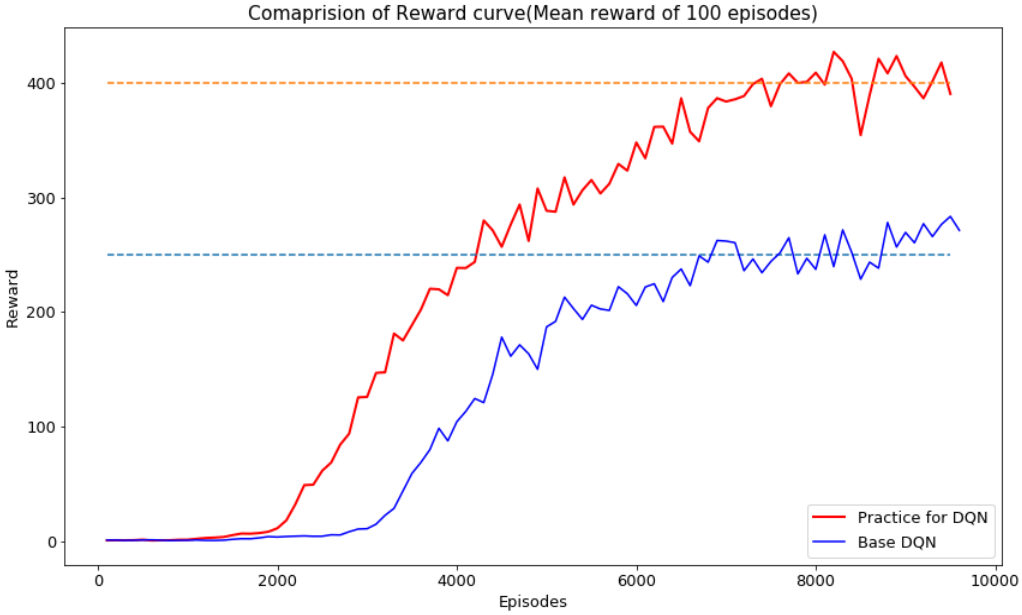


Figure 5.7: Reward curve for practice with DQN and base DQN (without practice) on Breakout.

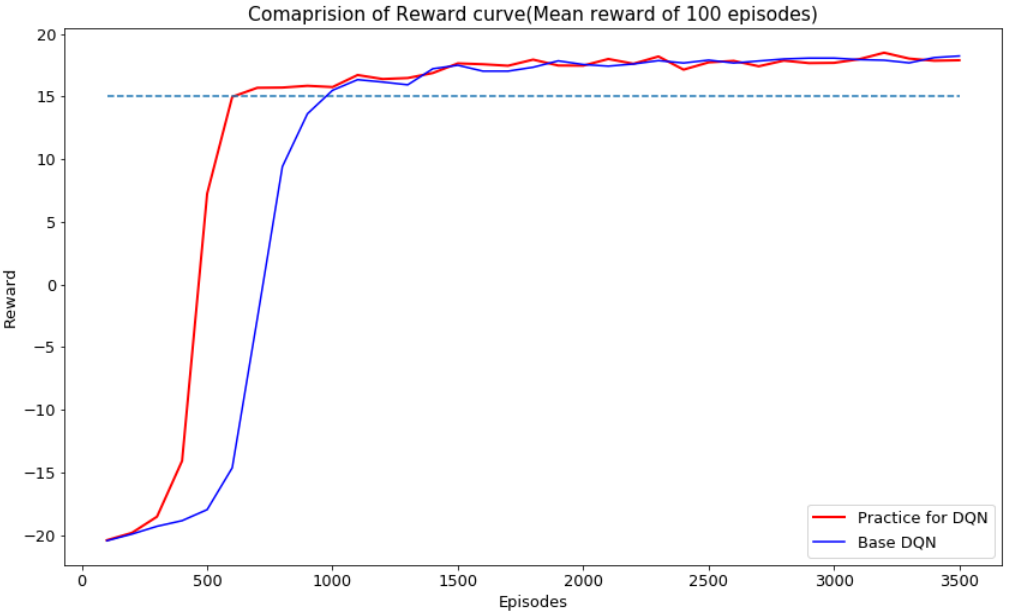


Figure 5.8: Reward curve for practice with DQN and base DQN (without practice) on Pong.

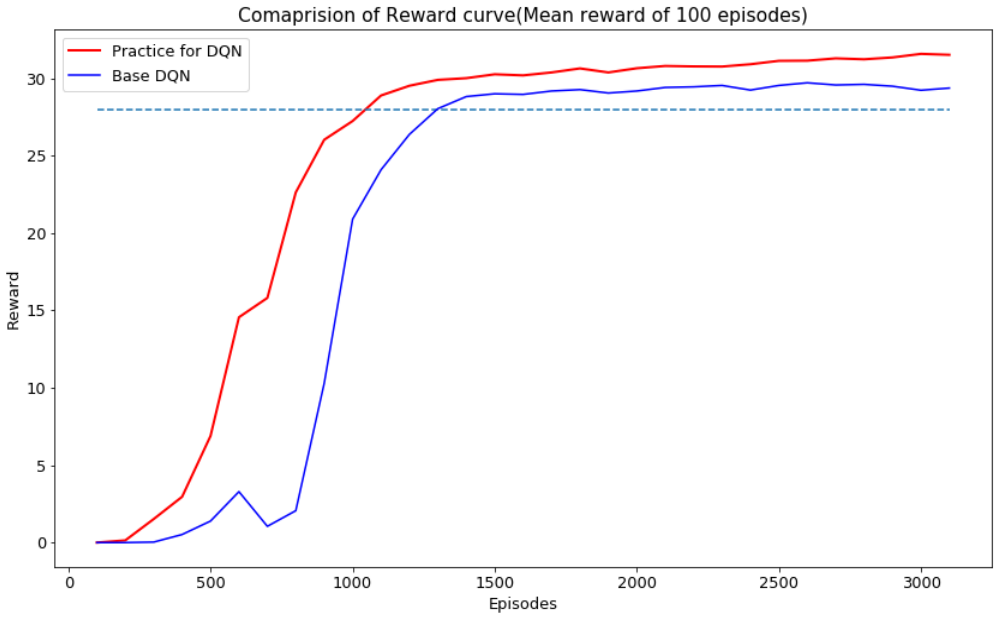


Figure 5.9: Reward curve for practice with DQN and base DQN (without practice) on Freeway.

5.4 Iterative Practice

In this section, the experimental setup and results for the iterative practice for DQN model are presented. For comparison, the results from practice for DQN model and base DQN model with no practice are used.

5.4.1 Visual Maze

The iterative model is tested initially on the both the 8x8 and 16x16 visual maze environments. For the initial practice stage, the practice network is trained for $d_{pr}^{(0)} = 10^4$ with mini-batches of 50 samples from the practice experience replay. For this task, I assigned the weights of the three convolution layers to the $\mathbf{\Theta}_{pr}^{(0)}$. After practice, weights $\mathbf{\Theta}_{pr}^{(0)}$ are transferred to the target training network, but unlike practice for DQN model, the weights $\mathbf{\Theta}_{tr}^{(0)}$ are locked after transferred from $\mathbf{\Theta}_{pr}^{(0)}$, i.e. the weights are not fine tuned during followup practice and training ($k \geq 1$). This is based on the observations from our first model which we present in the following chapter. For the

successive iterative training, we only fine-tune dense layers and transfer them between practice and training networks.

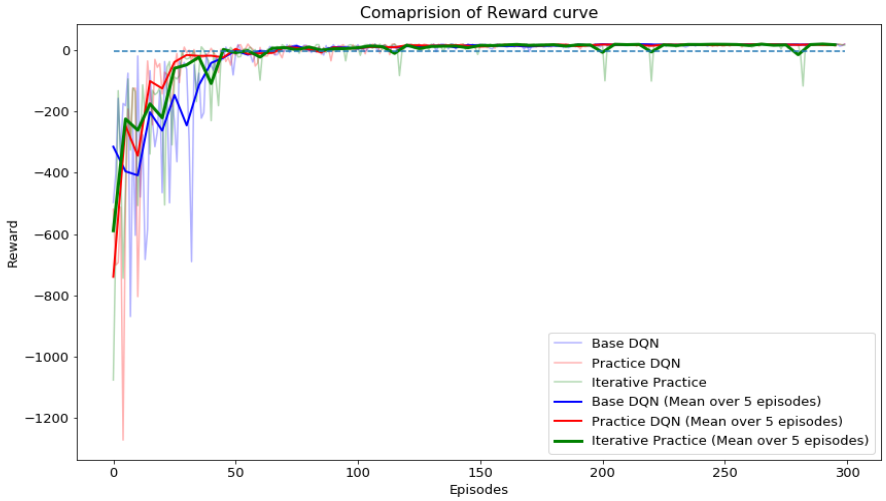
For the iterative training, the maximum iteration is set $N_iter = 20$. Short RL training duration $d_{tr}^{(k)} = 20$ episodes, with each episode allowing a maximum of 500 steps for the agent to reach the goal, and short practice duration $d_{pr}^{(k)} = 10^3$ steps. Mini-batches of size 50 are fed to the training network during short RL training. The accumulated is collected for each episode.

The results are presented in Figure 5.10. Although there is no significant improvement in simple 8x8 Maze (comparable to one-step practice) (Figure 6.2a), it can be observed that the learning speed of iterative practice surpass that of one-step practice when the search space of environment is increased by changing the maze to 16x16 (Figure 6.2b). Iterative practice reaches threshold of 0 after 50 episodes, which is much earlier than the other two methods. It indicates that iterative practice improves performance over one step practice.

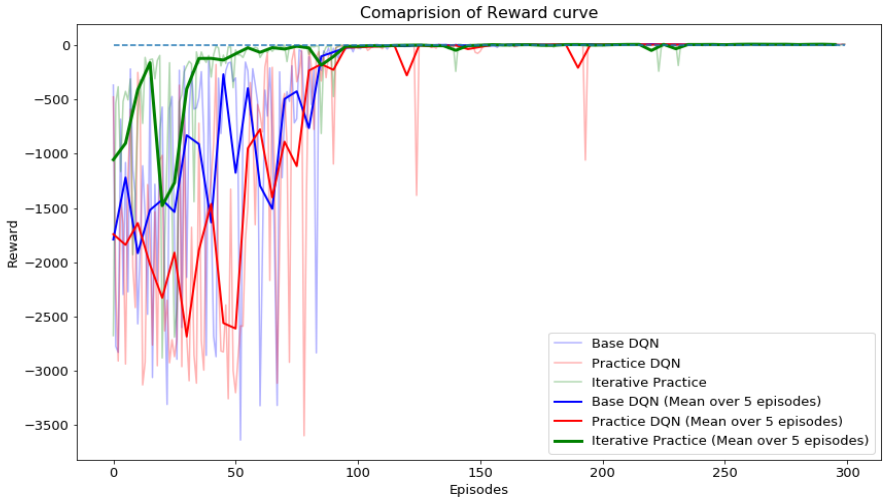
5.4.2 Atari Game Environments

To test the performance of the model on complex problems, I moved on to the Atari 2600 games: Breakout, and Freeway. For the initial practice stage in these tasks, the practice network is trained for $d_{pr}^{(0)} = 10^6$ steps. Mini-batches of 64 samples are used to train the practice network. I assigned the weights of the three convolution layers to the $\mathbf{\Theta}_{pr}^{(0)}$. After practice, weights $\mathbf{\Theta}_{pr}^{(0)}$ are transferred to the target training network. For these environments, the weights $\mathbf{\Theta}_{tr}^{(0)}$ are not locked, i.e. the weights are fine-tuned during followup training and practice.

For the successive iterative training, we fine-tune both, convolution layers and dense layers, but we transfer only the dense layers them between practice and training networks. I ran some initial experiments to select optimum values for the parameters N_iter, $d_{tr}^{(k)}$ and $d_{pr}^{(k)}$. For the iterative training, the maximum iteration is set to $N_iter = 40$, but early stopping is performed when convergence is achieved. Mini-



(a) Visual maze(8x8)



(b) Visual maze(16x16)

Figure 5.10: Reward curve for iterative practice method vs practice with DQN method vs base DQN method (without practice) on visual maze

batches of size 64 are fed to the training network during Short RL training for a duration $d_{tr}^{(k)} = 5 \times 10^5$ steps followed by short practice for a duration $d_{pr}^{(k)} = 10^2$ steps. During the training, once the agent reaches the end of a game, it is recorded as an episode and the accumulated reward for that episode is also recorded. The game is then reset to start position for the next iteration. Here also, there are no adjustments

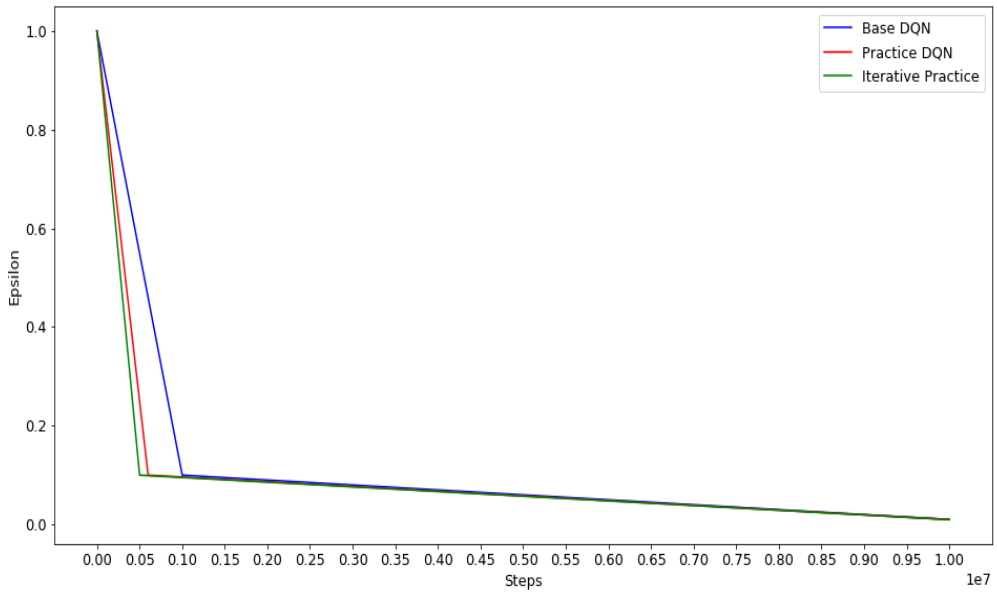
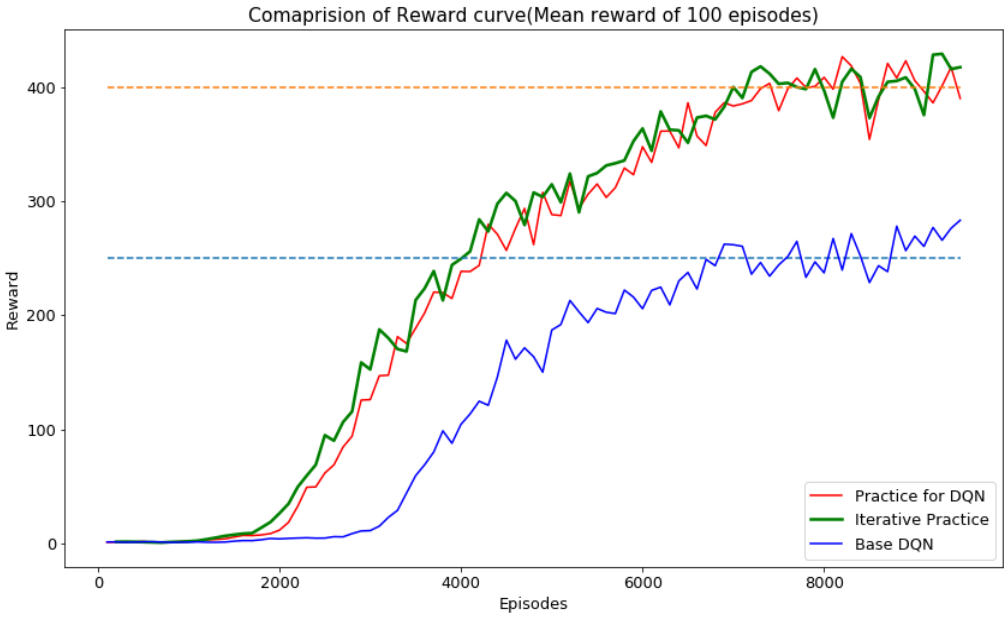


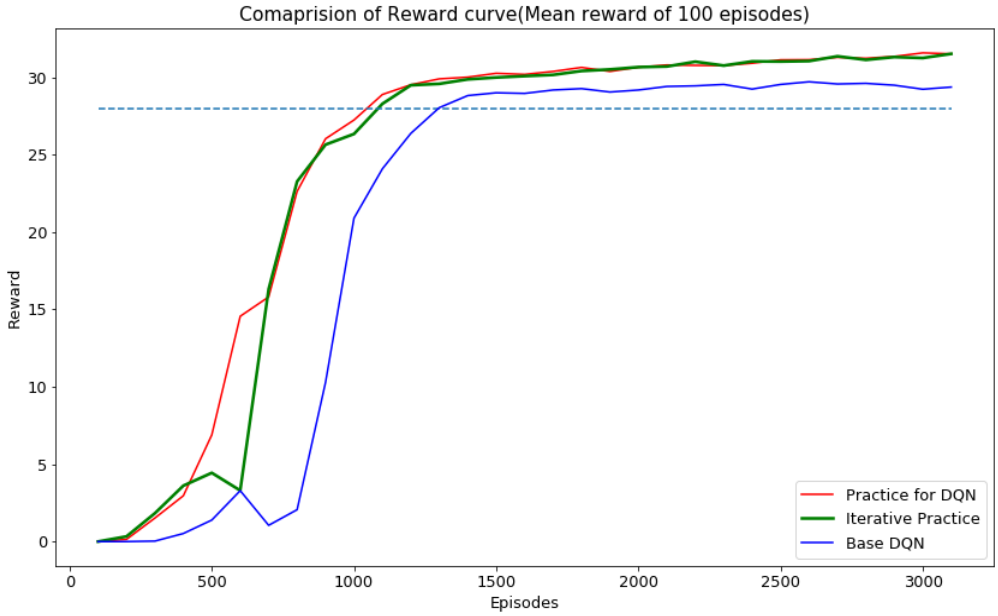
Figure 5.11: Epsilon used for iterative practice vs other models

in the network architecture or hyper-parameters for each Atari environment. The epsilon ϵ used for this model is presented in Figure 5.11. Here also, the exploration rate is reduced from 1.0 to 0.1 in less number of steps as in practice for DQN. The list of other hyperparameters, used in this model, and their values are similar to practice for DQN model and are given in Table 5.2.

From Fig 5.12, it can be observed that iterative practice shows slight improvement in performance of the agent compared to one step practice and base DQN, in complex environments as well. Results in Breakout environment show that, iterative practice also achieves high asymptotic performance compared to baseline. It can be noticed that iterative model reaches the threshold little earlier than the other two models. In Freeway, although there is an initial dip in the performance of the agent, it recovers well enough to match the performance of the one step practice model. It is clear that iterative practice is promising and it is possible to achieve better results with further hyperparameter tuning.



(a) Breakout



(b) Freeway

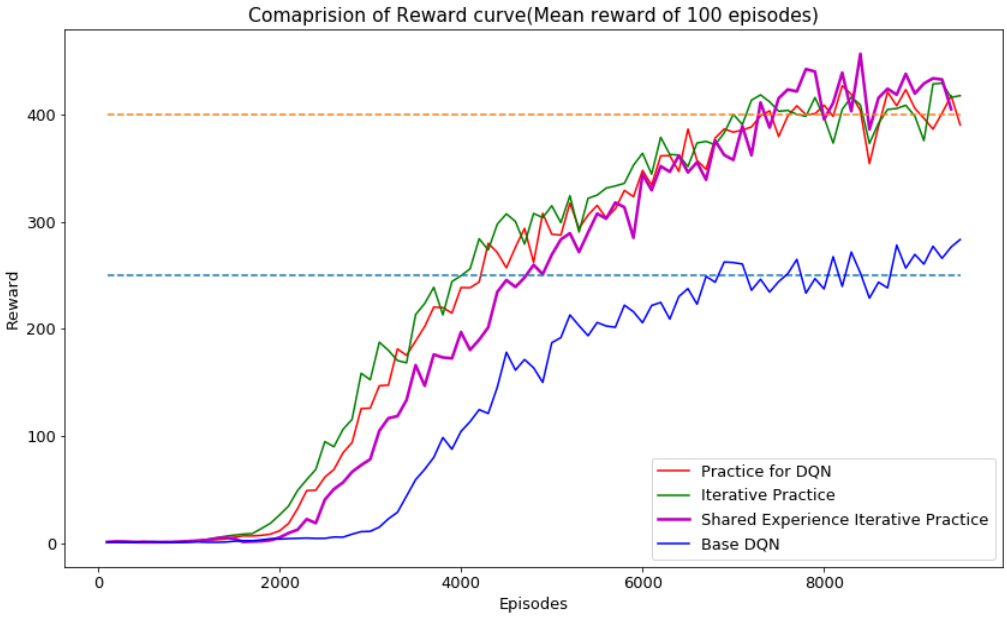
Figure 5.12: Reward curve for iterative practice method vs practice with DQN vs base DQN method (without practice) on Breakout and Freeway.

5.5 Shared Experience for Iterative Practice

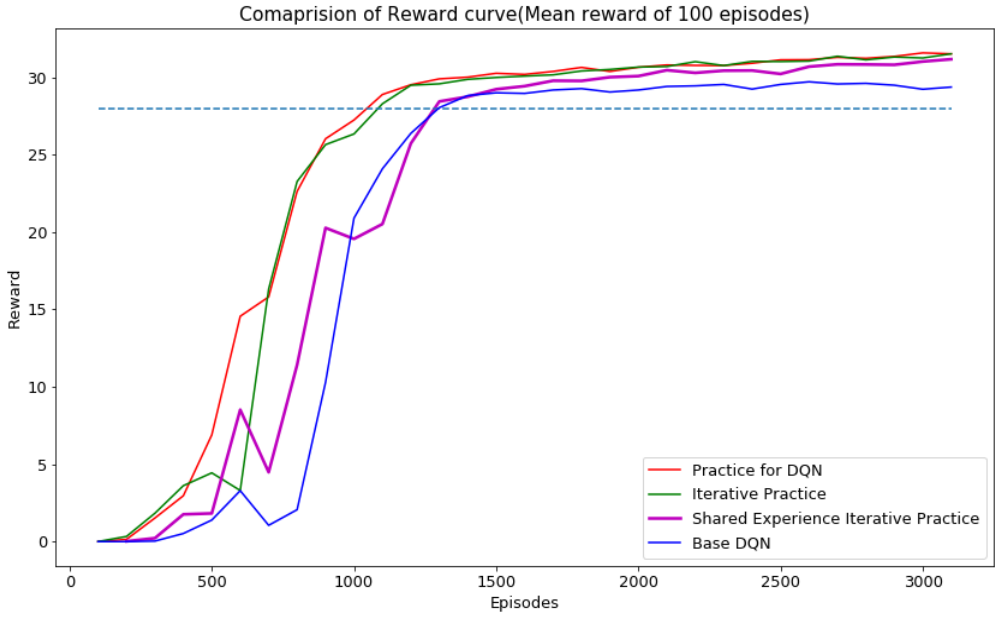
The idea behind shared experience model is to reduce the interactions between the agent and the environment, and at the same time it helps the practice network to train with data from large state space. As visual maze is not a complex task, this method is tested on the Atari environments, Breakout and Freeway, to examine the benefits of this model.

For the initial practice stage in these tasks, the practice network is trained for $d_{tr}^{(0)} = 5 \times 10^5$ steps which is one round of short training. After the initial training, the weights of the first three convolution layers $\mathbf{\Theta}_{pr}^{(0)}$ are transferred to the weights $\mathbf{\Theta}_{pr}^{(0)}$ in practice network. The practice network is trained for $d_{pr}^{(0)} = 25 \times 10^4$ steps. After the practice, weights $\mathbf{\Theta}_{pr}^{(0)}$ are transferred back to the target training network and iterative training follows. For the iterative training, the maximum iteration is set to $N_{iter} = 39$ (one step of Short RL training and practice is completed already), but early stopping is performed when convergence is achieved. Short RL training runs for a duration $d_{tr}^{(k)} = 5 \times 10^5$ steps followed by short practice for a duration $d_{pr}^{(k)} = 10^2$ steps. Here, there are no adjustments in the network architecture or hyper-parameters for each Atari environment. The list of other hyperparameters, used in this model, and their values can be referred from Table 5.2.

The results are presented in Figure 5.13. It can be observed that, shared experience for iterative practice approach suffers with slow learning initially when compared to iterative practice and one step practice. This can be attributed to the delayed initial practice. The agent starts with a short RL training session without initialization from practice, so the learning is hampered initially. It can be seen that, the performance of the agent matches with iterative practice belatedly which indicates that the overall performance is not affected due to shared experience.



(a) Breakout



(b) Freeway

Figure 5.13: Reward curve for shared experience for iterative practice compared to other models on Breakout and Freeway.

CHAPTER 6: DISCUSSIONS

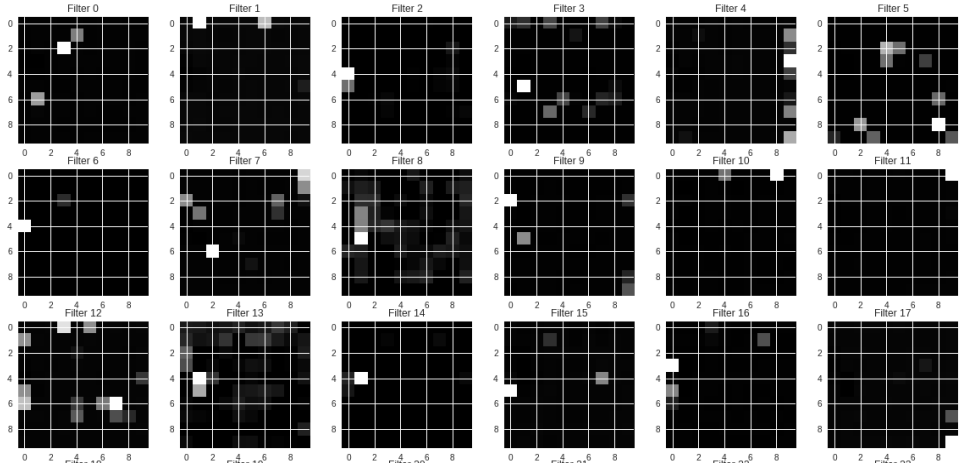
6.1 Observations of Shareable Representations

Improvement in the performance by practice with DQN model is attributed to the shared knowledge from the practice network transferred to the RL training network. To understand how practice is helping the target RL training, it is important to observe the shareable representations. For this, I visualized the outputs of the convolution layers and FC layer, whose weights are transferred, of both practice network and training network using activation heat maps. This gives a better understanding of which areas in the image are activated during practice and during RL training.

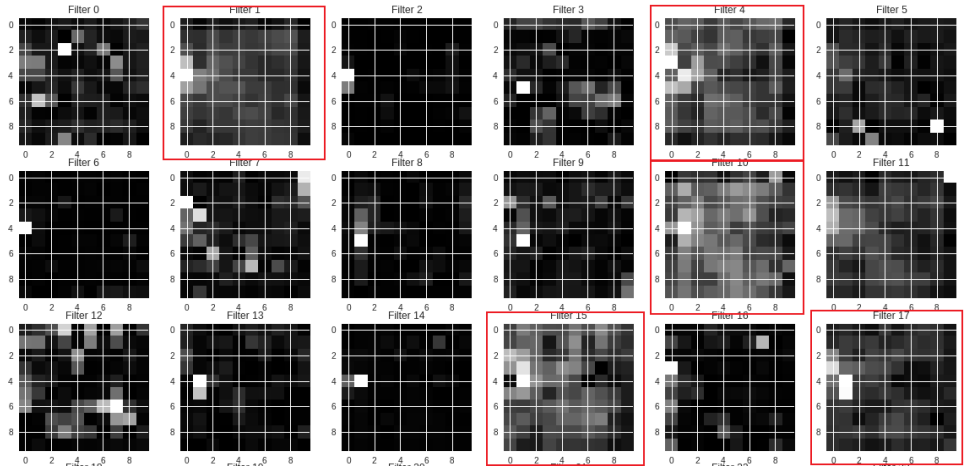
Figure 6.1 and 6.2 shows the visualizations of filters of third convolution layer, after practice and after RL training, for Visual Maze task and Breakout. I observed that the activations are identical mostly, except for a few filters (marked in red). This indicates that practice provides good perception knowledge for target learning. So, the convolution layers of the training network, when initialized with the practice network weights, are less likely fine-tuned. However, in other observations I made, there is no significant similarity between FC layer outputs after practice and after training. This motivated me to propose an iterative model that helps training the FC layer for further improvement as I achieved.

In the case of Visual Maze, the CNN layers are locked from the fine-tuning based on my observations of shareable representation. But in the case of Atari games, locking the CNN layer from fine-tuning didn't give desirable results. This observation indicated that, for Atari games, where the state space is very large, fine-tuning of CNN layers is necessary.

Similar observations of shareable knowledge are previously made [39]. They showed



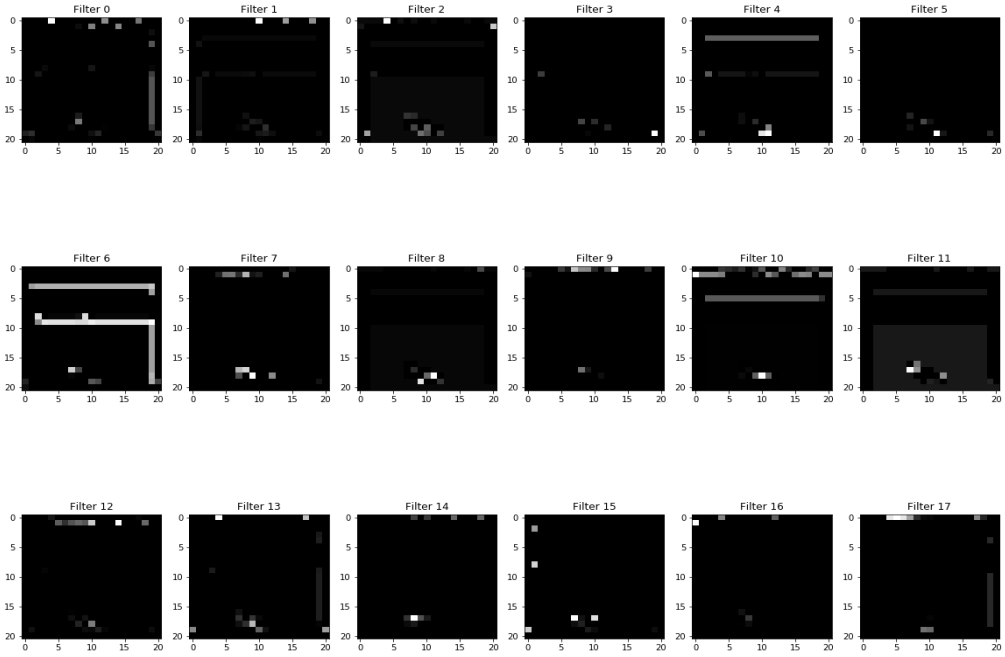
(a) Practice



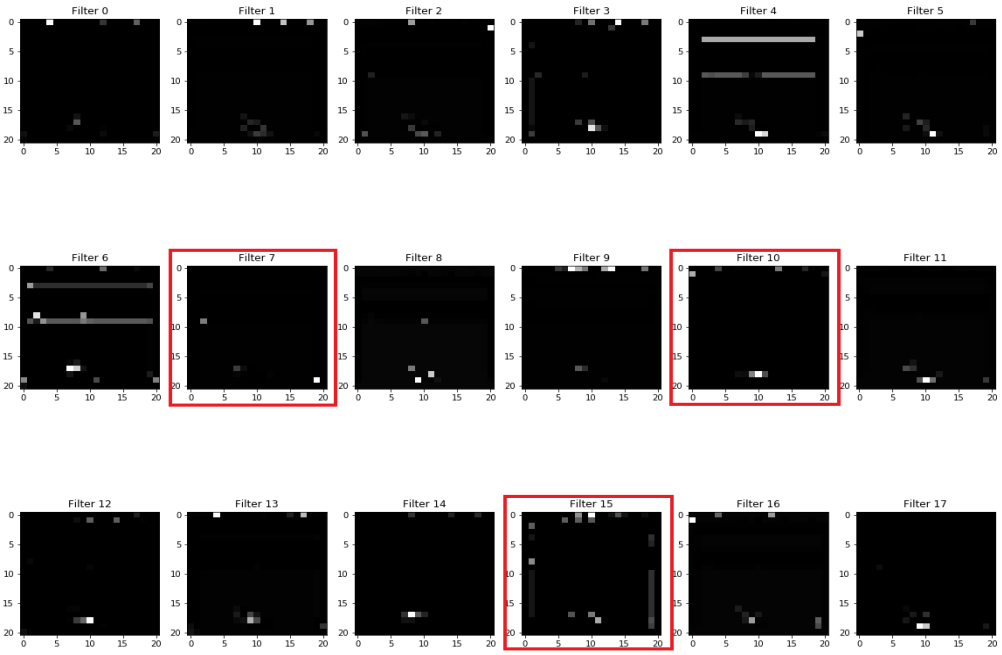
(b) Target RL training

Figure 6.1: Visualizing activations for some of the filters in the last convolution layer in the networks for Visual Maze(8x8)

that higher layers have general or less specific features, while lower layers learn more specific features of the task the network is trained upon. In transfer learning, general features from a source task help the learning of a target task rather than specific features. In another work [17], up to 3 convolution layers were transferred between two different Atari games. They compared the performance of the agent in the target Atari, after transferring 1, 2 and 3 convolution layers from the source Atari game. Among all the three transfers, transfer of 3 CNN layers had edge over other transfers.



(a) Practice



(b) Target RL training

Figure 6.2: Visualizing activations for some of the filters in third convolution layer in the networks for Breakout

6.2 Generalization through Iterative Practice

Humans have the ability to adapt to new, unseen situations and environments, like adjusting to driving a car in a new city with different roads and weather conditions. This can be attributed to the generalized representations of the world that humans have. On the other hand, RL algorithms are generally trained and tested in same or similar environments and thus they will not learn representations that can generalize to unseen situations. This can have serious implications when RL is applied to real-world systems, like self-driving cars, where situations are not always familiar.

Generalization is not new in deep learning architectures. Dropout regularization [40] in deep learning models reduces the co-complex adaptations or specialization of the weights of the network to specific features. This reduces the overfitting of the model to the training data and provides generalization. L1, L2 regularization [41] also helps in reducing the complexity in the model and solves the overfitting problem. Similar methodologies are required for RL to improve generalization. Robust adversarial reinforcement learning (RARL) [42] helps the agent to learn generalized policies as the method is robust to differences in training and test conditions. It uses an adversarial agent to impede disturbances to the RL agent, which makes the RL agent learn robust policies.

Although iterative practice method is quite different from RARL, it also serves the purpose of providing generalization to RL agents. Iterative practice also helps the agent to learn better-generalized policies than base DQN or one step practice. As discussed above, prolonged RL training tunes the hidden weights to specific features of the task. Iterative model prevents this by shifting from RL training to practice and vice versa periodically. This aspect of iterative training needs to be investigated further with suitable experiments and this plan is discussed in the future work.

6.3 Model Learning from Practice?

Combining practice with RL training helps the RL training network to learn features faster than otherwise. In a way, practice is preparing the learning and reusing it during training to find an optimal policy. In similar lines, Dyna-Q [43] is an algorithm which combines Q-learning with planning. It learns a model while learning the value/policy for a task. The model is improved during training and at the same time used for planning the next state. Thus, Dyna-Q prepares a model and uses it to improve RL training, unlike practice which does not learn a model but only state transitions. Also, preparation and improvement go parallel in Dyna-Q which is not the case with practice. Another algorithm called Value Iteration Networks [44], uses a differential planning module to learn policies, embedded to a feed-forward neural network which predicts the actions. The planning module uses value iteration to compute optimal policy using reward function and transition probability, which is used by the neural network to output the probabilities for possible actions.

In similar lines, another approach uses auxiliary losses [45] to improve the agent performance in a task. This approach considers learning the RL task by augmenting the loss of RL training with losses from auxiliary tasks. The auxiliary tasks are chosen in such a way that they support navigation or planning of agent in the environment and thus they help the agent get richer training signals for RL task. Unlike practice, the agent is jointly trained on goal-driven RL problem and auxiliary tasks.

All these approaches are similar in perspective that they prepare learning and use the learning in improving RL training. But, they are significantly different in the way they are implemented. Practice fundamentally focuses on transfer of knowledge from non-RL task to assist feature learning in RL task which is unlike the above discussed approaches.

CHAPTER 7: CONCLUSIONS

Practice for DQN, showed that practice approach can be applied to deep reinforcement learning algorithms. The method is tested on Visual Maze and Atari environments, and empirical results were provided to show that practice helps an agent learn faster and achieve higher asymptote. I also proposed a novel practice strategy called iterative practice, which further improves the performance of the agent. The efficacy of this approach is tested on Visual Maze and Atari environments. Although the improvement is not astounding, it enables further research into different configurations of the network and hyperparameters for better performance. Adding to these two methods, I also proposed a method called shared experience for iterative practice, which effectively reduces interactions between agent and environment, without affecting the performance of the agent. I examined this method on Breakout and Freeway games and showed that despite a slight drop of performance initially, the model achieves convergence as the iterative practice model. Additional experiments for this method can further strengthen that it accomplishes its objective.

I discussed possible reasons for the success of practice with DQN by presenting my observations of the abstract knowledge representations in the deep neural network layers after practice and target RL training.

7.1 Future Work

Examining the adaptability of practice and iterative practice framework in other deep reinforcement learning algorithms and experimenting on diverse environments can be a natural next step. As discussed in Section 6.2, this research can be extended to understand the generalization provided by the iterative practice. Also, leveraging

iterative practice in developing meta-learning models for reinforcement learning can be an interesting direction for future research.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, May 1992.

[3] D. Silver, *Reinforcement Learning and Simulation-based Search in Computer Go*. PhD thesis, Edmonton, Alta., Canada, 2009.

[4] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[5] X. Bu, J. Rao, and C. Xu, "A reinforcement learning approach to online web systems auto-configuration," in *2009 29th IEEE International Conference on Distributed Computing Systems*, pp. 2–11, June 2009.

[6] Z. Wen, D. OâNeill, and H. Maei, "Optimal demand response using device-based reinforcement learning," *IEEE Transactions on Smart Grid*, vol. 6, no. 5, pp. 2312–2324, 2015.

[7] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, pp. 875–889, July 2001.

[8] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 354–359, 10 2017.

[9] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, Y. Wu, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis, and D. Silver, "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II." Available: https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019. Accessed: 2018-09-20.

[10] J. Heinrich and D. Silver, "Deep reinforcement learning from self-play in imperfect-information games," *arXiv preprint arXiv:1603.01121*, 2016.

[11] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky, "Deep reinforcement learning for dialogue generation," *arXiv preprint arXiv:1606.01541*, 2016.

[12] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, (New York, NY, USA), pp. 50–56, ACM, 2016.

[13] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3357–3364, May 2017.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[16] G. de la Cruz, Y. Du, J. Irwin, and M. Taylor, "Initial progress in transfer for deep reinforcement learning algorithms," in *25th International Joint Conference on Artificial Intelligence (IJCAI)*, 07 2016.

[17] G. V. de la Cruz, Y. Du, and M. E. Taylor, "Pre-training neural networks with human demonstrations for deep reinforcement learning," *CoRR*, vol. abs/1709.04083, 2017.

[18] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, pp. 21:1–21:35, Apr. 2017.

[19] C. W. Anderson, M. Lee, and D. L. Elliott, "Faster reinforcement learning after pretraining deep networks to predict state dynamics," *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, 2015.

[20] M. Lee and C. W. Anderson, "Can a reinforcement learning agent practice before it starts learning?," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 4006–4013, May 2017.

[21] J. M. Zurada, *Introduction to artificial neural systems*, vol. 8. West publishing company St. Paul, 1992.

[22] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[23] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[24] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Taipei, Taiwan, 1998.

[25] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *ArXiv e-prints*, 11 2015.

[26] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pp. 242–264, IGI Global, 2010.

[27] S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. J. Mooney, and K. Saenko, "Translating videos to natural language using deep recurrent neural networks," *CoRR*, vol. abs/1412.4729, 2014.

[28] H. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE Transactions on Medical Imaging*, vol. 35, pp. 1285–1298, May 2016.

[29] M. M. Ghazi, B. Yanikoglu, and E. Aptoula, "Plant identification using deep neural networks via optimization of transfer learning parameters," *Neurocomputing*, vol. 235, pp. 228–235, 2017.

[30] M. Baker, "1,500 scientists lift the lid on reproducibility," *Nature*, vol. 533, p. 452â454, May 2016.

[31] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[32] F. Liu, S. Li, L. Zhang, C. Zhou, R. Ye, Y. Wang, and J. Lu, "3dcnn-dqn-rnn: A deep reinforcement learning framework for semantic parsing of large-scale 3d point clouds," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 5679–5688, Oct 2017.

[33] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *Journal of Machine Learning Research (JMLR)*, vol. 10, pp. 1633–1685, Dec. 2009.

[34] W. Sun, A. Venkatraman, G. J. Gordon, B. Boots, and J. A. Bagnell, "Deeply aggrevated: Differentiable imitation learning for sequential prediction," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 3309–3318, 2017.

[35] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, *et al.*, "Deep q-learning from demonstrations," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[36] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude." COURSERA: Neural Networks for Machine Learning, 2012.

[37] A. D. Baddeley, *Human memory: Theory and practice*. Psychology Press, 1997.

[38] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines." https://github.com/openai/baselines, 2017.

[39] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Advances in neural information processing systems*, pp. 3320–3328, 2014.

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

[41] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[42] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2817–2826, JMLR. org, 2017.

[43] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM SIGART Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[44] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, pp. 2154–2162, 2016.

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