2019 IEEE Symposium Series on Computational Intelligence (SSCI)
December 6-9 2019, Xiamen, China

Efficient Practice for Deep Reinforcement Learning

1st Venkata Sai Santosh Ravi Teja Kancharla
Seagate Technology
Longmont, CO 80503
ravi.kancharla@seagate.com

2nd Minwoo Lee
Department of Computer Science
University of North Carolina at Charlotte
Charlotte, NC 28223
minwoo.lee@uncc.edu

***Abstract*—Deep reinforcement learning has demonstrated its ability to solve diverse hard problems, which were not able to be solved previously. However, one of the main drawbacks is the required long training time for a task. Furthermore, trial-and-error in reinforcement learning is often inappropriate in most real-world tasks. There were some discussions on transfer learning, specifically sim-to-real transfer, but the design of new environments or defining an environment with similar or relevant goals require tedious human efforts. This research aims to leverage *practice* approaches, which do not require a new environmental design for transfer learning, to shorten the time for training a deep reinforcement learning agent and to achieve an optimal policy for the maximum expected rewards. We verify the efficacy of existing practice approach with Deep Q networks (DQN) in Atari games and also introduce a novel practice approach involving iterations of short periods of practice and reinforcement learning, to further improve the performance of an agent.**

***Keywords*—Deep Reinforcement Learning; Practice; Transfer Learning; Deep Q-Networks**

I. Introduction

Reinforcement learning (RL) has been making a big step forward as the development of deep neural networks progresses. Combining deep learning with RL, recent advances in deep reinforcement learning (DRL) has lead popularity in various application areas like natural language processing [1], resource management [2], and robotics [3]. One of the state-of-the-art models, Deep Q Network (DQN) [4] [5] has suggested an end-to-end model that enables an RL agent to learn directly from raw sensory data without tedious handcrafted feature engineering. It effectively approximates Q values to achieve or exceed human-level of playing video games and other environments.

Although 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 [6]. 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.

Transfer learning [7] is one good solution to address these problems. Transfer between different Atari games was examined to improve the learning speed [6]. However, transfer learning requires human efforts to define the source and target tasks for effective transfer of knowledge between them.

The efficiency of learning can be achieved through imitation learning (or apprenticeship learning) [8], 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 [9] extends the imitation learning to work with deep neural networks. However, it requires the expert's presence to provide proper feedback, and it is known that the transferred policy cannot exceed the demonstrated one. Deep Q-learning from Demonstrations (DQfD) [10] suggests a pre-training approach to overcome the limitation of learning-from-demonstration in Deeply AggreVaTed. The pre-training stage uses temporal difference loss along with a supervised loss to learn a policy and to imitate the demonstrator. This method combines imitation learning with RL which enables the agent to develop a superior policy than the expert. It showed an acceleration in learning but still rely on human experts to gather demonstration data.

Practice [11] suggests a new paradigm of transfer learning that does not need experts' help. The knowledge gained from a *non-RL source* task is applied to an *RL target* task. For instance, from a simple regression task of predicting state transitions, neural network weights are pre-trained and transferred to a Q network function approximator for target RL training [12]. Or, the practice can be defined as training a classifier from non-expert human demonstration data and successive transfer of weights to a target RL task [13].

The 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 the similar source and target tasks for transfer learning. To verify the approach, Sparse Bayesian Reinforcement Learning (SBRL) [11] explained how the knowledge obtained from practice helps to learn a target task by showing bases construction process in practice, their transfer, and use of the transferred bases in target learning.

Although the efficacy of practice is examined, practice with an end-to-end model is not well examined for learning from large practice space with raw sensor inputs.

In this paper, we extend the practice to be applicable to end-to-end models with deep reinforcement learning algorithms. To further improve the efficiency of practice and learning, we propose a novel strategy, *iterative practice*. Imitating human practice and learning model (we never stop practice and learning), iterative practice repeats practice and short-term learning until it converges.

By preventing any possible overfitting of function approximator, the iteration shortens the learning time and improves performance in complex tasks . We present empirical results

for iterative practice with DQN in Visual Maze and Atari games.

Our main contributions in this work are 1) suggesting an efficient way of training an end-to-end model by leveraging practice, 2) examining the efficacy of a novel iterative strategy for practice and learn, and 3) observing and visualizing the abstract knowledge representation in deep neural networks during practice and training.

II. Background

A. Deep Q-Network

Deep Q Network (DQN) [4] is a deep reinforcement learning algorithm that uses deep neural networks as a function approximator. DQN architecture is comprised of convolutional layers, which learn encoded feature representations from raw input images, followed by fully connected (FC) dense layers and an output layer, which produces a single Q value output for each valid action. Here Q value, represented as $Q(s, a)$, is the overall expected reward assuming the agent is in state s and performs an action a.

Similar to classic RL algorithms, DQN interacts with an environment and collects samples of state, action, next state, and reward $(s_t, a_t, s_{t+1}, r_{t+1})$. These samples are stored in an experience replay memory. Mini-batches of random samples are used for training to reduce the sampling bias and improves the performance of DQN. For stable learning, two identically structured networks, namely online network Q^{online} and target network Q^{target}, are used for Q value approximation. The mini-batch gradient update for online network minimizes the mean squared loss function,

$$\mathcal{L} = \left(y_t - Q^{online}(s_t, a_t)\right)^2 \tag{1}$$

where the target y_t is defined by using the approximation from the target network:

$$y_t = r_t + \gamma \max_a Q^{target}(s_{t+1}, a). \tag{2}$$

The target network is periodically updated with the weights from the online network.

B. Practice

Practice 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. One of the previous works [12] showed that learning the dynamics of an environment accelerates the learning of deep RL agents. They explained how a neural network implements practice to obtain suitable knowledge (the network weights $\mathbf{w}$) and initialize the Q value network with it for learning in a target environment. A single neural network is used for both practice and training, whose architecture comprises of multiple densely connected layers. There are two different output layers, one gives the state difference during practice and the other gives the Q values for the actions during target training.

To find shareable knowledge, practice sets up a regression problem that predicts 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, $f_{\mathbf{w}}(s_t, a_t) \approx \Delta s_t = s_{t+1} - s_t$. 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:

$$\mathcal{L} = \left(\Delta s_t - f_{\mathbf{w}}(s_t, a_t)\right)^2. \tag{3}$$

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

The learned weights from practice are used as initial values for the RL training network which solves the actual RL task. 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 [14] update:

$$\mathcal{L} = \left(r_{t+1} + \gamma Q_{\mathbf{w}}(s_{t+1}, a_{t+1}) - Q_{\mathbf{w}}(s_t, a_t)\right)^2. \tag{4}$$

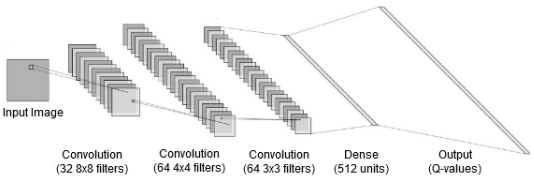
III. Methods

A. Practice for DQN

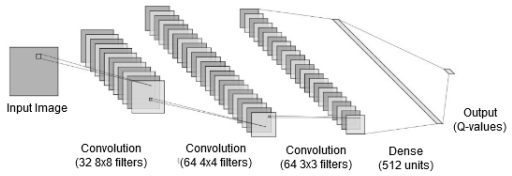
We leverage the practice approach and speed up the learning in DQN. Figure 1 shows our DQN network architecture for practice and RL training. The architectures of DQN and practice networks have an identical structure with three convolutional layers (32 8x8 filters, 64 4x4 filters, 64 3x3 filters) followed by a fully connected layer (512 hidden units). In our model, we train the practice network to solve a non-RL task of predicting the state change Δs_t given the current state s_t, i.e., the difference between the current state and next possible state [12]. The actions are chosen randomly instead of following any policy. Unlike the existing practice approach (Section II-B), here we do not feed action a_t as an input to the practice network but instead, we use it in the loss function. For RL training, we follow the traditional DQN approach with two identical networks, online and target. We train the networks on the RL-task and it predicts the Q-values for each possible action.

Since the model learns directly from raw images, we consider the pixel value from the snapshot of the game or task at time t as the state s_t. The first step is training the practice network followed by RL training. For practice, we collect n_{pr} number of samples of (s_t, a_t, s_{t+1}) with random actions and store in a practice memory. We use mini-batches of these samples to train the practice network to estimate the state difference $\Delta s_t = s_{t+1} - s_t$ for all possible states based on available actions. From the predicted state differences, we only consider the state difference based on the current action a_t. The target state difference here is the difference in pixel values of s_{t+1} and s_t. We use the mean squared loss (Eq. 3) to measure the inconsistency between predicted and target state difference. To minimize the loss, we train the practice network with Adam optimizer for a duration of $d_{pr}^{(0)}$.

After the practice network finishes its training, we initialize the weights of the RL training network $\mathbf{\Theta}_{tr}^{(0)}$ with the practice



(a) Practice Network Architecture



(b) RL Training Architecture

Fig. 1: DQN network architecture for practice and RL training

network weights $\mathbf{\Theta}_{pr}^{(0)}$. During the RL training, we collect samples of s_t, a_t, a_{t+1}, reward r_{t+1} in a different replay memory. Unlike practice, we collect reward values as well for RL training and therefore a separate replay memory is required. We randomly sample experiences from this memory to train the RL training network. We use Huber loss for training which is described as:

$$L_\delta(y_t, Q_t) = \begin{cases} \frac{1}{2}(y_t - Q_t)^2 & \text{for } |y_t - Qt| \leq \delta \\ \delta|y_t - Qt| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \quad (5)$$

where y_t is the target output defined in Eq. 2 and $Q_t = Q^{online}(s_t, a_t)$. δ is the control parameter which can be tuned. We train the RL training network to minimize the loss using RMSprop optimizer [15] for a duration of $d_{tr}^{(0)}$ or until the model converges, which ever occur first. This model is also called as one-step practice as we perform practice once before RL training.

B. Iterative Practice

Humans generally perform short cycles of practice followed by an actual task which can be attributed to the limited attention span we have. Moving back and forth from actual task increases productivity compared to prolonged execution of a single task [16]. Motivated by this human practice and learning model, we propose a practice framework that iterates over a short period of practice and a short period of RL training until the model converges. Also, iterative practice fully leverages the benefits of one-step practice and helps the agent to learn quickly during the initial phase of RL training.

Figure 2 illustrates the stages in iterative practice comparing to the one-step practice. The iterative model has two stages: initial practice and iterative training. The initial practice is similar to the practice stage in one-step practice and the iterative training has cycles of short RL training and short practice. In this framework, we use the same practice network for initial practice and short cycles of practice. There aren't any modifications in the network architectures for practice and RL training from Section III-A and are shown in Figure 1.

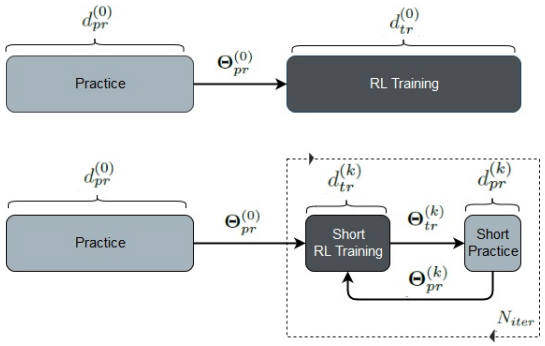


Fig. 2: Traditional Practice (top) vs. Iterative Practice (bottom)

For initial practice, we collect n_{pr} samples with random actions, store it a replay memory, and use mini-batches of these samples to train the practice network which estimates the state difference Δs_t. We use the mean squared loss (Eq. 3) to measure the inconsistency between predicted and target state difference. We train the practice network with Adam optimizer for a duration of $d_{pr}^{(0)}$.

After initial practice, we transfer the weights of layers $\mathbf{\Theta}_{pr}^{(0)}$ from practice network to the RL training network. This initiates the iterative training stage. During the short RL training we collect new experiences s_t, a_t, a_{t+1}, reward r_{t+1} and store them in a different replay memory. The following short practice does not collect new data but uses the same data collected from the replay memory used during initial practice.

We set the number of iteration be N_{iter}, for which short RL training and short practice are run. We train the RL training network for a short duration of $d_{tr}^{(k)}, k \in [1, N_{iter}]$ where $d_{tr}^{(k)} \ll d_{tr}^{(0)}$. We use Huber loss (Eq. 5) for training, which is minimized using RMSprop optimizer [15]. We transfer the trained RL training network weights $\mathbf{\Theta}_{tr}^{(k)}$ 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 we transfer the weights from the practice network back to the training network. This alternative transfer continues iteratively as follows:

$$\mathbf{\Theta}_{tr}^{(k)} \leftarrow \mathbf{\Theta}_{pr}^{(k)}, \quad (6)$$

$$\mathbf{\Theta}_{pr}^{(k+1)} \leftarrow \mathbf{\Theta}_{tr}^{(k)}. \quad (7)$$

This process continues for N_{iter} iterations or until the model converges. 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.

IV. Experiments

The efficiency and efficacy of extended practice for DQN and iterative practice methods are tested on Visual Maze and Atari game environments, shown in Figure 3. The performances are compared to a base DQN model (without any practice) which is designed to have a similar architecture and parameters to the DQN model described in [4].

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)
  Transfer weights from practice to target (Eq. 6)
  Train target training network for $d_{tr}^{(k)}$ (Eq. 5)
  Transfer weights from target to practice (Eq. 7)
end for

A. Experimental Environments

a) Visual Maze: It is a navigation (8x8/16x16 blocks) task where an agent starts from the start position (pink block) and moves towards the goal position (green block) dodging the walls (black blocks). Its has 4 actions, left, right, up and down. A reward of -1 for each movement, -5 for hitting a block or moving out of the maze, or $+30$ for reaching the goal is given to the agent.

b) Pong: In this Atari game, the agent controls a green paddle to bounce a ball back to the left towards the opponent. The game has 6 possible actions. 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.

c) Breakout: The objective of the 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. The game has 4 actions. The reward is defined as $+1/+4/+7$ if the ball hits a brick (based on the color of brick). The game terminates when paddle misses the ball and it goes out of the screen.

d) Freeway: The objective of the game is to make the chicken cross the ten-lane highway from one side to the other side. The chicken should avoid hitting the vehicles on the highway. The player gets a point every time a chicken gets across to the 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 3 actions.

e) Boxing: The objective of the game is to score more punches than the opponent/bot in a boxing match. The agent controls a boxer who tries to land punches on the opponent and at the same time escaping from his punches. Each punch gives a reward point to the respective boxer. The game terminates when either the agent or the bot get 100 reward points or when time runs out. The game has 18 possible actions.

f) Tennis: The objective of the game is to win a six-game set against a bot in a tennis match. The game terminates when either the agent or the bot wins the six-game set. If the set ends in a draw, the set restarts from 0-0. The game has 18 possible actions for the agent.

B. Experimental Setup

1) Practice for DQN: For the Visual Maze task, we train the practice network for $d_{pr}^{(0)} = 10^4$ steps using mini-batches of 50 samples from the practice replay memory. We consider the weights of the three convolutional layers as $\mathbf{\Theta}_{pr}^{(0)}$ for this task. After practice, we transfer the weights $\mathbf{\Theta}_{pr}^{(0)}$ to the RL training network, and randomly initialize the weights of the dense layers in RL training network (see discussions in Section V).

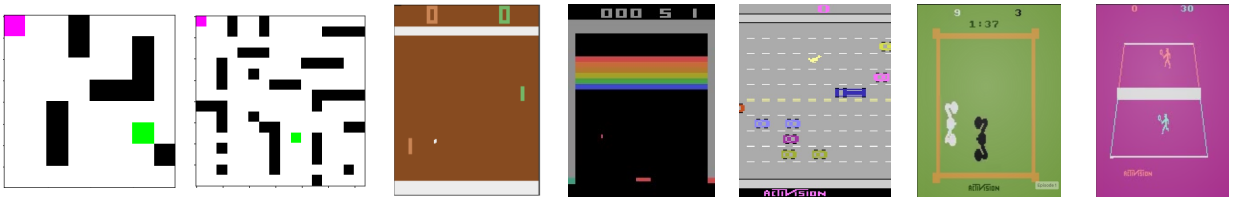
We feed mini-batches of size 50 to the training network which runs for $d_{tr}^{(0)} = 300$ episodes, with each episode allowing a maximum of 500 steps for the agent to reach the goal.

For Atari game environments, we train the practice network for $d_{pr}^{(0)} = 10^6$ steps using mini-batches of 64 samples. Similar to the Visual Maze task, we consider the weights of the three convolution layers as $\mathbf{\Theta}_{pr}^{(0)}$ and transfer the weights to the RL training network. We use mini-batches of size 64 for the training network which runs for a maximum of $d_{tr}^{(0)} = 2 \times 10^7$ steps, but we perform early stopping if the training loss converges approximately to zero. During the training, once the agent reaches the end of a game, we record it as an episode and record the accumulated reward for that episode as well. The game is then reset to the start position for the next iteration. There are no adjustments in the network architecture or hyper-parameters for each Atari environment. While the exploration rate (ϵ) for the agent in the base DQN model decays from 1.0 to 0.1 over the first million steps and 0.1 to 0.01 over the remaining steps, in practice for DQN model, it decays from 1.0 to 0.1 over the first 6×10^5 steps and 0.1 to 0.01 over the remaining steps. As the practice for DQN model already learns features from practice and requires less exploration, we bring the exploration rate down from 1.0 to 0.1 after fewer number of steps compared to the base DQN model.

2) Iterative Practice: For the Visual Maze task, during the initial practice stage, we train the network for $d_{pr}^{(0)} = 10^4$ steps with mini-batches of 50 samples from the practice replay memory. For this task, we again consider the weights of the three convolution layers as $\mathbf{\Theta}_{pr}^{(0)}$. After practice, we transfer the weights $\mathbf{\Theta}_{pr}^{(0)}$ to the target training network, but unlike practice for the DQN model, here we lock the weights $\mathbf{\Theta}_{tr}^{(0)}$ after transferring from $\mathbf{\Theta}_{pr}^{(0)}$, i.e. we do not perform any fine-tuning of the weights during subsequent practice and training ($k \geq 1$). This is based on our observations from the practice for DQN model which we present in our discussions (Section V). For the iterative training, we only fine-tune dense layers and transfer them between practice and training networks.

During the iterative training, we set the number of maximum iterations to $N_{iter} = 20$, short RL training duration to $d_{tr}^{(k)} = 20$ episodes, where each episode allows a maximum of 500 steps for the agent to reach the goal, and short practice duration to $d_{pr}^{(k)} = 10^3$ steps. The size of mini-batches is set to 50 for short RL training.

For the Atari environments, during the initial practice stage, we train the network for $d_{pr}^{(0)} = 10^6$ steps with mini-batches of



(a) Maze8 (b) Maze16 (c) Pong (d) Breakout (e) Freeway (f) Boxing (g) Tennis

Fig. 3: Test Environments

64 samples from. We consider the weights of the three convolution layers as $\mathbf{\Theta}_{pr}^{(0)}$. After practice, we transfer the weights $\mathbf{\Theta}_{pr}^{(0)}$ to the target training network. For these environments, we do not lock the weights $\mathbf{\Theta}_{tr}^{(0)}$, i.e. we fine-tune the weights during subsequent training and practice.

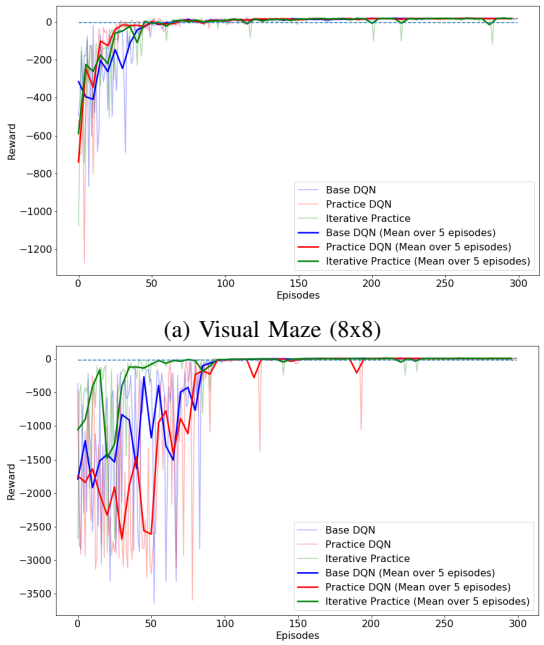
For the iterative training stage, we fine-tune the weights of both convolutional and dense layers, but we transfer only the dense layers between the practice and training networks. We ran some initial experiments to select optimum values for the parameters N_{iter}, $d_{tr}^{(k)}$ and $d_{pr}^{(k)}$. We set the maximum iterations to $N_{iter} = 70$, but perform early stopping when convergence is achieved. We perform short RL training for $d_{tr}^{(k)} = 3 \times 10^5$ steps followed by short practice for $d_{pr}^{(k)} = 10^2$ steps both with mini-batches of size 64. During the short RL training, once the agent reaches the end of a game, we record it as an episode and also record the accumulated reward for that episode. The game is reset to start position for the next iteration. Again, there are no adjustments in the network architecture or hyper-parameters for each Atari environment. In this model, we decay the ϵ from 1.0 to 0.1 over the first 5×10^5 steps and 0.1 to 0.01 over the remaining steps.

C. Results

Figure 4a depicts the accumulated rewards earned from training base DQN model, practice for DQN model and iterative practice model on a simple 8x8 Visual Maze environment. It shows that practice and iterative practice helps the agent to learn the optimal policy quicker than the base DQN model.

The asymptotic performance, which is the overall performance of a model after training, cannot be compared in this environment, as the maximum possible reward is limited and both the models achieved that value after training. Although there is no significant improvement for iterative practice in simple 8x8 Maze when compared to one-step practice, the learning speed of iterative practice surpassed that of one-step practice when the search space increased with changing the maze to 16x16 (Figure 4b). Iterative practice converged at zero after 50 episodes, which is much earlier than the other two methods. It indicates that iterative practice improves the performance of the agent over the one-step practice.

For Atari environments, we train the models for a large number of episodes when compared to the Visual Maze task and the reward after each episode is highly fluctuating. Therefore, to better evaluate the performance of the agent, we considered the mean reward for every 100 episodes as an evaluation metric. Figure 5 depicts the accumulated mean



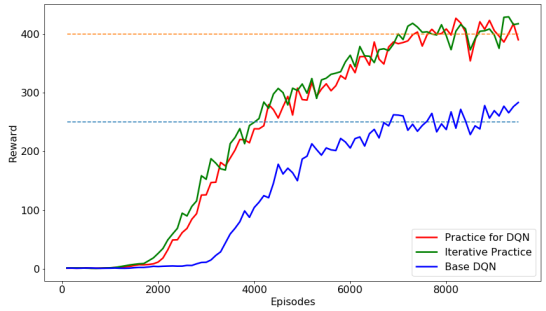
(a) Visual Maze (8x8)

(b) Visual Maze (16x16)

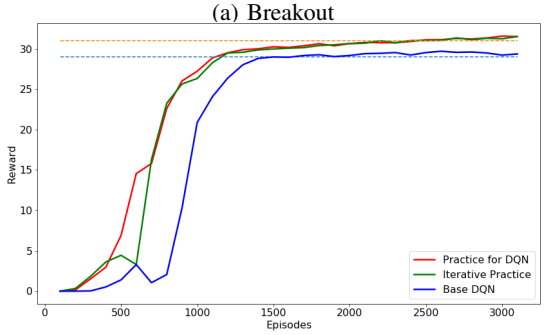
Fig. 4: Reward curve for iterative practice, practice for DQN and base DQN method implemented on Visual Maze

rewards for base DQN, practice DQN and iterative practice in Breakout, Pong, Freeway, Boxing and Tennis.

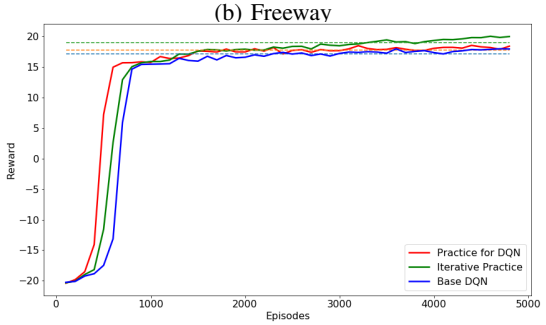
In Breakout, Freeway and Pong, practice for DQN and iterative practice models reached the convergence value (blue dotted) of base DQN model much earlier and both of them converged at a higher value (orange dotted line) than that of the base DQN model. This suggests that, in addition to faster learning, practice also helps in achieving a better asymptotic performance in these environments. Noticeably, iterative practice model learned faster than the one-step practice model in Breakout. In Freeway, although there is an initial dip in the performance of the iterative practice agent, it recovered well enough to match the performance of the one-step practice model. Whereas in Pong, iterative practice is initially slower compared to one-step practice, but it eventually achieved better asymptotic performance (green dotted line). We attribute the initial lag in the performance of the iterative practice model in Pong and Freeway, to the low complexity of these environments, which confines the benefits of the iterative practice model. By low complexity, we mean that it takes fewer numbers of steps for any agent to learn an optimal



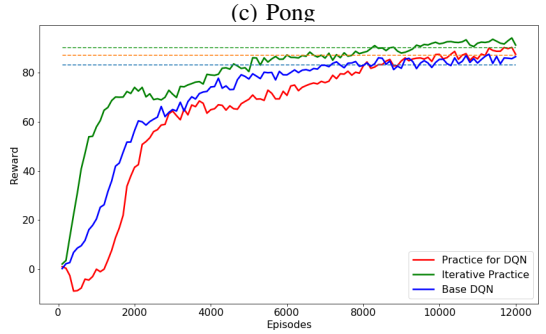
(a) Breakout



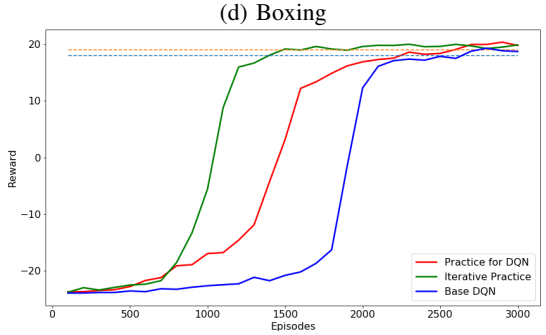
(b) Freeway



(c) Pong



(d) Boxing



(e) Tennis

Fig. 5: Reward curve (Mean reward of 100 episodes) for iterative practice method vs practice with DQN vs base DQN method on Atari Environments.

policy in these environments compared to Breakout, Boxing or Tennis. We also made a similar observation in the case of 8x8 Visual Maze.

In Boxing (Figure 5d), the performance of the practice model initially trailed behind the base DQN model, but it eventually achieved better asymptotic performance at the end. In Tennis (Figure 5e), however, practice for DQN model reached the convergence value of the base DQN model (blue dotted line) much earlier. In these environments, iterative practice model clearly exceeded the performance of the other models. In addition to faster learning, iterative practice also helped in converging at a higher value in Boxing (green dotted line) and Tennis (orange dotted line). We attribute this performance of the iterative practice model to the relatively high complexity of Boxing and Tennis environments (though Tennis has less number of episodes, each episode take a very high number of steps to complete, compared to Pong or Freeway), which we also observed in 16x16 Visual Maze. It is evident that iterative practice is promising and it has the potential to achieve better results with further hyperparameter tuning.

V. Discussions

A. Training Time

One of the goals of practice and iterative practice is to lessen the training time required for the agent to learn a complex task. Training time in transfer learning models is often evaluated either as *total time*, which includes the time taken to train the source task and time taken to train the target task, or *target task time*, which only considers the time taken to train the target task. For autonomous models like practice and iterative practice, where human efforts are redundant, it is appropriate to use *target task time* as the metric [7].

Besides, the practice network is trained for much less number of iterations than the RL training network (about 20 times less number of iterations), so the impact of practice on the overall training time is negligible. Thus, we consider the time taken for the agents to learn a task using the knowledge from the practice without being charged for the time spent during practice.

From the experimental results, we observe that iterative practice and practice DQN models achieve convergence after lesser episodes than base DQN model. As an episode represents a game played from its beginning to end or until it is lost, the number of episodes can be directly proportional to the time taken for training. Thus, it indicates that the training time required for iterative practice and practice DQN models is lesser than the base DQN model. Furthermore, iterative practice model need far less training time than the practice DQN model in complex environments like Boxing and Tennis.

B. 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 the effect of practice on target RL training, it is important to observe the shareable representations. For this, we visualized

the outputs of the convolution layers and fully connected dense (FC) layer, of both practice network and training network, using activation heat maps and observed the similarity in the activated areas of the images during practice and RL training.

Figure 6 shows the visualizations of filters of the third convolutional layer, after practice and after RL training, for Visual Maze task. We observed that the activations are mostly identical, except for a few filters (marked in red). It indicates that practice provides good perception knowledge for target learning. So, the weights of convolutional layers of the RL training network, when initialized with the practice network weights, are less likely fine-tuned. However, in another observation we made, there is no significant similarity between FC layer outputs after practice and after training. This motivated us to propose an iterative model that helps training the FC layer for further improvement as we achieved.

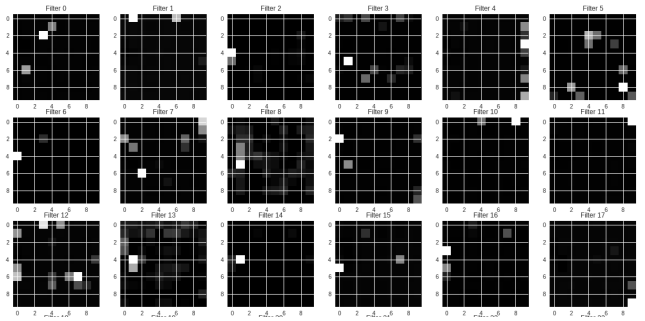
In case of the Visual Maze, we locked the convolutional layers from the fine-tuning based on our observations of shareable representation. But in case of Atari games, locking the convolutional layers from fine-tuning did not produce desirable results. This observation indicated us that, for Atari games, where the state space is very large, fine-tuning of convolutional layers is necessary.

Similar observations of shareable knowledge were previously made [17]. They showed that higher layers have general or less specific features, while lower layers learn more specific features of a task. In transfer learning, general features from a source task assist the learning of a target task rather than specific features. In another work [13], up to three convolutional layers were transferred between two different Atari games. The authors compared the performance of the agent in the target Atari, after transferring one, two and three convolutional layers from the source Atari game. Among all the three transfers, transfer of three layers had an edge over other transfers.

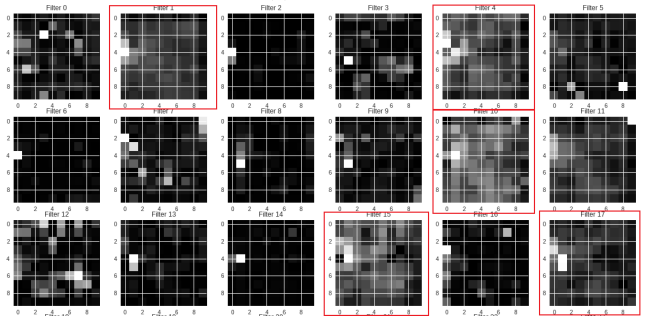
C. Model Learning from Practice?

We showed that 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. On similar lines, Dyna-Q [18] 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 [19] 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.

On similar lines, another approach uses auxiliary losses [20] to improve the agent performance in a task. This approach



(a) Practice



(b) RL training

Fig. 6: Visualizing activations for the last convolutional layer in practice and RL training networks for an intermediate state in the Visual Maze environment

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 the aforementioned approaches are similar to the practice in perspective that they prepare the learning and use it in improving RL training. Significant difference and contribution of our suggested practice approaches lie in the efficient feature learning in an RL task through the transfer of knowledge from a non-RL task.

D. Generalization through Iterative Practice

Human's ability to adapt to new, unseen situations and environments can be attributed to the generalized representations of the world that we have. On the other hand, RL algorithms are generally trained and tested in same or similar environments and thus they fail to learn representations that can generalize to unseen situations. This can have serious implications when RL is applied to real-world systems

Generalization is a familiar notion used in deep learning architectures. Dropout regularization [21], which is known as generalization technique in deep learning models, reduce 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. L1, L2 regularization [22] 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) [23] 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 can serve the purpose of providing a generalization to RL agents as well. As discussed above, prolonged RL training tunes the hidden weights to specific features of the task, which happens in base DQN and practice for DQN models. But, the iterative model prevents this by shifting from RL training to practice and vice versa periodically. We see this as an interesting aspect of iterative practice which needs to be investigated further with suitable experiments.

VI. Conclusions

In this work, we suggested an efficient way to train an end-to-end deep reinforcement learning model by leveraging practice to reduce the learning time and to find an optimal policy for the maximum expected rewards. Also, we proposed and examined the efficacy of a novel practice strategy called iterative practice. Our experiments showed that iterative practice is a promising approach for improving the performance of the agent over the one-step practice. We also discussed the possible reasons for the success of practice with DQN by presenting our observations of the abstract knowledge representations in the deep neural networks after practice and after RL training.

Examining the adaptability of practice and iterative practice framework in other deep reinforcement learning algorithms and experimenting on diverse environments is a natural next step. As discussed in Section V-D, 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.

VII. Acknowledgement

The Titan XP GPU used for this research was donated by the NVIDIA Corporation.

References

[1] 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.

[2] 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)*, May 2017, pp. 3357–3364.

[3] 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: ACM, 2016, pp. 50–56. [Online]. Available: http://doi.acm.org/10.1145/3005745.3005750

[4] 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.

[5] 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.

[6] 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.

[7] 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. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577069.1755839

[8] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 21:1–21:35, Apr. 2017. [Online]. Available: http://doi.acm.org/10.1145/3054912

[9] 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, 2017, pp. 3309–3318.

[10] 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.

[11] 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)*, May 2017, pp. 4006–4013.

[12] 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.

[13] 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.

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

[15] 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.

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

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

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

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

[20] 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.

[21] 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.

[22] 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. [Online]. Available: http://www.jstor.org/stable/2346178

[23] 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*. JMLR. org, 2017, pp. 2817–2826.