Hashing over Predicted Future Frames for Informed Exploration of Deep Reinforcement Learning

Haiyan Yin, Jianda Chen, Sinno Jialin Pan
Nanyang Technological University, Singapore
{haiyanyin, jianda001, sinnopan}@ntu.edu.sg

Abstract

In deep reinforcement learning (RL) tasks, an efficient exploration mechanism should be able to encourage an agent to take actions that lead to less frequent states which may yield higher accumulative future return. However, both knowing about the future and evaluating the frequency of states are non-trivial tasks, especially for deep RL domains, where a state is represented by high-dimensional image frames. In this paper, we propose a novel informed exploration framework for deep RL, where we build the capability for an RL agent to predict over the future transitions and evaluate the frequency for the predicted future frames in a meaningful manner. To this end, we train a deep prediction model to predict future frames given a state-action pair, and a convolutional autoencoder model to hash over the seen frames. In addition, to utilize the counts derived from the seen frames to evaluate the frequency for the predicted frames, we tackle the challenge of matching the predicted future frames and their corresponding seen frames at the latent feature level. In this way, we derive a reliable metric for evaluating the novelty of the future direction pointed by each action, and hence inform the agent to explore the least frequent one.

1 Introduction

Reinforcement learning (RL) involves an agent progressively interacting with an initially unknown environment, in order to learn an optimal policy with the objective of maximizing the cumulative rewards collected from the environment [Sutton and Barto, 1998]. Throughout the learning process, the RL agent alternates between two primal behaviors: exploration - to try out novel states that could potentially lead to high future rewards; and exploitation - to perform greedily according to the learned knowledge. In the past, exploitation of learned knowledge has been well studied, while how to efficiently explore through the state space is still remained as a critical challenge, especially for deep RL domains.

In deep RL domains, a state can be represented by low-level sensory inputs, such as image pixels [Mnih et al., 2015], which is often high-dimensional or and continuous. Thus, the state space for deep RL is huge and often intractable for searching. When performing exploration through such a huge state space, most existing single-task (e.g., [Wang et al., 2016; Schaul et al., 2016; Bellemare et al., 2016]) and multi-task (e.g., [Parisotto et al., 2016; Rusu et al., 2016; Yin and Pan, 2017]) deep RL approaches adopt a simple exploration heuristic, ε-greedy strategy, where the RL agent takes a random action with a probability of ε, e.g., via uniform sampling for discrete-action domains [Sutton and Barto, 1998; Mnih et al., 2015] or corrupting action with i.i.d. Gaussian noise for continuous-action domains [Lillicrap et al., 2016]. In such a way, the agent explores the state space undirected [Thrun, 1992], i.e., without incorporating any meaningful knowledge about the environment. Such exploration heuristic turns out to work well in simple problem domains but fails to handle more challenging domains, such as those with extremely sparse rewards that result in exponentially increasing state space.

Unlike the undirected exploratory behavior for agents using ε-greedy strategy, when human beings are intending to explore an unfamiliar task domain, one often actively applies domain knowledge for the task, accounts for the state space that has been less frequently visited, and intentionally tries out actions that lead to novel states. In this work, we aim to mimic such exploratory behaviors to improve upon the ε-greedy strategy with random action selection, and come up with a more efficient informed exploration framework for deep RL agents. On the one hand, we develop agent’s knowledge on the environment and make it able to predict the future trajectories. On the other hand, we integrate the developed knowledge with hashing techniques over the high-dimensional state space in order to make the agent be able to realistically evaluate the novelty for the predicted future trajectories.

Specifically, in our proposed informed exploration framework, first, we train an action-conditional prediction model to predict future frames given a state-action pair. Second, to perform hashing over the high-dimensional state space seen by the agent, we train a deep convolutional autoencoder to generate high-level features for the state and apply locality-sensitive hashing (LSH) [Charikar, 2002] on the high-level state features to generate binary codes to represent each state. However, the learned hashing function is counting over the actually seen states, while we need to query the counts for the predicted future frames to compute their novelty. Hence, we
introduce an additional training phase for the autoencoder to match the hash codes for the predicted frames with that of their corresponding ground-truth frames (i.e., the actually seen frames). In this way, we are able to utilize the environment knowledge and hashing techniques over the high-dimensional states to generate a reliable novelty evaluation metric for the future direction pointed by each action given a state.

2 Related Work

Recently, works on enhancing the exploration behavior of deep RL agents have demonstrated great potential in improving the performance of various challenging RL task domains. In [Mnih et al., 2016], asynchronous training techniques are adopted and multiple agents are created to perform gradient-based learning to update the model parameters separately in the Atari 2600 domain. In [Osband et al., 2016], an ensemble of Q-functions is trained to reduce the bias for the values being approximated and increases the exploration depth in the Atari 2600 domain. In [Houthooft et al., 2016], an exploration strategy based on maximizing the information gain of the agent’s belief about the environment dynamics is adopted for tasks with continuous states and actions. In [Achiam and Sastry, 2017], the KL-divergence between the probability obtained from a learned dynamics model before and after updated by the experience is used to measure the surprise over the experience. In [Bellemare et al., 2016; Tang et al., 2017; Martin et al., 2017], the novelty of a state is measured based on count-based mechanisms, and reward shaping is performed by adding a reward bonus term inferred from the value of counts. For all aforementioned approaches, the exploration strategy is incorporated either in the function approximation or during the optimization process, and the agent still needs to randomly choose action to explore without relying on any knowledge about the model. In this work, we aim to conduct informed exploration by utilizing the model-based knowledge to derive a deterministic choice of action for the agent to explore.

3 Methodology

3.1 Notations

In this paper, we consider a discounted finite-horizon Markov Decision Process (MDP) with discrete actions. Formally, it is defined by a tuple \((S, A, P, R, \gamma)\), where \(S\) is a set of states which could be high-dimensional or continuous, \(A\) is a set of actions, \(P\) is a state transition probability distribution with \(P(\cdot|s,a)\) specifying the probability for transitioning to state \(s'\) after issuing action \(a\) at state \(s\), \(R\) is a reward function mapping each state-action pair to a reward in \(\mathbb{R}\), and \(\gamma \in [0, 1)\) is a discount factor. The goal of the RL agent is to learn a policy \(\pi\) that maximizes the expected cumulative future rewards given the policy: \(E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]\). In the context of deep RL, at each step \(t\), an RL agent receives a state observation \(s_t \in \mathbb{R}^r \times m \times n\), where \(r\) is the number of consequent frames to represent a state, and \(m \times n\) is the dimension for each frame. The agent selects an action \(a_t \in A\) among all the \(l\) possible choices, and receives a reward \(r_t \in \mathbb{R}\).

3.2 Informed Exploration Framework

We propose an informed exploration framework to mimic the exploratory behavior of human beings under an unfamiliar task domain. The overall exploration decision making process is illustrated in Figure 1.

Generally, the RL agent no longer randomly selects an action to explore without incorporating any domain knowledge. Instead, we aim to let the agent intentionally select the action that leads to the least frequent future states and thus explore
Figure 1: An illustration over the decision making for the informed exploration framework. At state $S_t$, the agent needs to choose from $a_t^{(1)}$ to $a_t^{(4)}$, an action to explore. Specifically, the states inside the dashed rectangle indicates predicted future states. The color of circles after $S_t$ indicates frequency/novelty of states, where lighter ones indicate novel states and darker ones indicate frequent states. The agent first predicts future roll-outs, and then evaluate the novelty using hashing. In this example, $a_t^{(2)}$ is chosen because the followed roll-out is the most novel.

the state space in an informed and deterministic manner. To this end, we build the capability of the RL agent on performing the following two tasks: 1) predicting over future transitions, and 2) evaluating the visiting frequency for those predicted future frames.

![Deep neural network architectures adopted for informed exploration. Up: action-conditional prediction model for predicting over future transition frames; down: autoencoder model for conducting hashing over the state space.](image)

**Learning Transition Model with Prediction Network**

The architecture for action-conditional prediction is shown in Figure 2 (up). To be specific, we train a deep prediction model $f : (S_t, a_t) \rightarrow S_{t+1}$ to make the agent able to predict over the future transitions given a state-action pair. The state input $S_t$ is a set of $r$ recent image frames, and the action input $a_t$ is represented by $a_t \in \mathbb{R}^l$, which is a one-hot vector, where $l$ is the number of actions for the task domain. To predict a new state, the model predicts one single frame at a time, denoted as $\hat{s} \in \mathbb{R}^{m \times n}$. The new state $S_{t+1}$ is formed by concatenating the predicted new frame with the most recent $r-1$ frames. We adopt the action-conditional transformation as proposed in [Oh et al., 2015] to form a joint feature for the state input and the action input. Specifically, the state input is first passed through three stacked convolutional layers to form a feature vector $h_t^s \in \mathbb{R}^k$. Then the state feature $h_t^s$ and the one-hot action feature $a_t$ perform a linear transformation by multiplying with their corresponding transformation matrix $W_2^s \in \mathbb{R}^{k \times h}$ and $W_2^a \in \mathbb{R}^{k \times 1}$. After the linear transformation, both features are shaped with the same dimensionality. Then the features for the state and action after the linear transformation performs a multiplicative interaction to form a joint feature as follows,

$$h_t = W_2^s h_t^s \odot W_2^a h_t^a.$$  

Afterwards, the joint feature $h_t$ is passed through stacked deconvolutional layers and a sigmoid layer to form the final prediction output. To predict over multiple future steps, the prediction model progressively composes the new state using its prediction result to predict the next-step transition. Note that our prediction on frame-to-frame level is more precise than those on feature level because the prediction target of frame-to-frame is exact image from game environment while feature prediction target is encoded feature extracted from a trained neural network.

**Hashing over the State Space with Autoencoder and LSH**

To evaluate the novelty of a state, we adopt a hashing model to count over the state space. We first train an autoencoder model on frames, $g : s \in \mathbb{R}^{m \times n} \rightarrow \hat{s} \in \mathbb{R}^{m \times n}$, in an unsupervised manner (to classify the pixels), with the reconstruction loss defined as follows [Kingma and Welling, 2014],

$$\mathcal{L}_{\text{Rec}}(s_t) = -\frac{1}{mn} \sum_{j=1}^{m} \sum_{i=1}^{n} (\log p(\hat{s}_{ij})),$$  

where $\hat{s}_{ij}$ is the reconstructed pixel at the $i$-th row and the $j$-th column. The architecture for the autoencoder model is shown in Figure 2 (down). To be specific, each convolutional layer is followed by a Rectifier Linear Unit (ReLU) layer and a max pooling layer with a kernel size $2 \times 2$. To discretize the
state space, we hash over the last frame \( s_t \) of each state. We adopt the output of the last ReLU layer from the encoder as the high-level state features, and denote by \( \phi(\cdot) \) the corresponding feature map that generates the high-level feature vector \( z_t \in \mathbb{R}^d \) of a state, i.e., \( \phi(s_t) = z_t \). To further discretize the state feature, locality-sensitive hashing (LSH) [Charikar, 2002] is adopted upon \( z_t \). To this end, a projection matrix \( A \in \mathbb{R}^{p \times d} \) is randomly initialized with i.i.d. entries drawn from a standard Gaussian \( \mathcal{N}(0,1) \). Then by projecting feature \( z \) through \( A \), the sign of the outputs forms a binary code, \( c \in \mathbb{R}^p \). With the introduced discretization scheme, we are able to count over the state space for the problem domain. During the RL process, a hash table \( H \) is created. The count for a state \( S_t \), denoted by \( \psi_t \), can be stored, queried and updated from the hash table. Overall, the process for counting over a state \( S_t \) is expressed in the following way:

\[
    z_t = \phi(s_t), \quad c_t = \text{sgn}(Az_t), \quad \text{and} \quad \psi_t = H(c_t). \tag{2}
\]

### Matching the Prediction with Reality

To derive the novelty for the predicted frames while updating the hash table by counting over the seen frames, we need to match the predictions with realities, i.e., make the hash codes for the predicted frames to be the same as their corresponding ground-truth seen frames in training. To this end, we introduce an additional training phase for the autoencoder model \( g(\cdot) \). To make the hash codes to be the same, the derived feature vectors of the predicted frames and the ground-truth seen frames through \( \phi(\cdot) \) need to be close to each other. We introduce an additional loss function of a pair of ground-truth seen frame and predicted frame \( (s_t, \hat{s}_t) \) as follows,

\[
    L_{\text{mat}}(s_t, \hat{s}_t) = \| \phi(s_t) - \phi(\hat{s}_t) \|_2 \tag{3}
\]

Finally, by combing (1) and (3), we define the following overall loss function,

\[
    L(s_t, \hat{s}_t; \theta) = L_{\text{rec}}(s_t) + L_{\text{rec}}(\hat{s}_t) + \lambda L_{\text{mat}}(s_t, \hat{s}_t), \tag{4}
\]

where \( \theta \) is the parameter for the autoencoder. Note that even though the prediction model could generate almost identical frames, training the autoencoder with the only reconstruction loss may lead to distinct state codes in all the task domains (details will be shown in Section 4.2). Therefore, the effort for matching the codes is necessary.

However, matching the state code while guaranteeing a satisfying reconstruction behavior is extremely challenging. Fine-tuning an autoencoder fully trained with \( L_{\text{rec}} \) by involving the additional code matching loss \( L_{\text{mat}} \) would fast disrupt the reconstruction behavior before the code loss could decrease to the expected level. Training the autoencoder from scratch with both \( L_{\text{rec}} \) and \( L_{\text{mat}} \) is also difficult, as \( L_{\text{mat}} \) is initially very low and \( L_{\text{rec}} \) is very high. The network can hardly find a direction to consistently decrease \( L_{\text{rec}} \) with such an imbalance. Therefore, in this work, we propose to train the autoencoder for two phases, where the first phase uses \( L_{\text{rec}} \) to train on seen frames until convergence, and the second phase uses the composed loss function \( L \) as in (4) to address the requirement for matching the prediction with reality.

### Computing Novelty for States

Once the prediction model \( f(\cdot) \) and the autoencoder model \( g(\cdot) \) are both trained, the agent could perform informed exploration as illustrated in Figure 1. At each step, the agent performs exploration with a probability less than \( \epsilon \) and performs greedy action selection otherwise. When performing exploration, the agent strategically selects the most novel action direction to explore. Given the state \( S_t \), the agent first performs roll-out with length \( H \) for the future trajectories predicted by the prediction model, for all the possible actions \( a_j \in A \). Formally, the novelty score for an action \( a_j \) given state \( S_t \), denoted by \( \rho(a_j|S_t) \), is computed as,

\[
    \rho(a_j|S_t) = \sum_{i=1}^{H} \frac{\beta^{i-1}}{\psi_{t+i} + 0.01}, \tag{5}
\]

where \( \psi_{t+i} \) is the count for the future state \( S_{t+i} \) derived from (2) by hashing over the predicted frame \( \hat{s}_{t+i} \), \( H \) is a predefined roll-out length, and \( \beta \) is a real-valued discount rate. After evaluating the novelty for all the possible actions, the agent selects the one with the highest novelty score to explore. Overall, the policy for the RL agent with the proposed informed exploration strategy is defined as:

\[
    a_t = \begin{cases} 
    \arg \max_a [Q(S_t, a)] & p \geq \epsilon, \\
    \arg \max_a [\rho(a|S_t)] & p < \epsilon,
    \end{cases}
\]

where \( p \) is a random value sampled from Uniform \((0,1)\), and \( Q(S_t, a) \) is the output of the Q-value function.

### Experiments

In the empirical evaluation, we use the Arcade Learning Environment (ALE) [Bellemare et al., 2013], which consists of Atari 2600 video games as the testing domain. We choose 5 representative games that require significant exploration to learn the policy: Breakout, Freeway, Frostbite, Ms-Pacman and Q-bert. Four of the five games, Freeway, Frostbite, Ms-Pacman and Q-bert are classified as hard exploration games, based on the taxonomy of exploration proposed in [Bellemare et al., 2016], where Freeway has sparse reward and the others have dense reward. Breakout, though not classified as hard exploration game, demonstrates significant exploration bottleneck with standard exploration attempt, as the state space is changing rapidly as agent learn, and the performance with standard exploration falls far behind advanced exploration technique. Hence it is also included as a test domain.

For all the tasks, we use the state representation that concatenates 4 consequent image frames of size \( 84 \times 84 \).

#### 4.1 Evaluation on Prediction Model

The architecture of the prediction model is identical to the one shown in Figure 2 (up). To train the prediction model, we create a training dataset which consists of 500,000 transition records generated by a fully trained DQN agent performing under standard \( \epsilon \)-greedy, where \( \epsilon \) is set equal to 0.3 (same as [Oh et al., 2015]). During training, we adopt Adam [Kingma and Ba, 2014] with a learning rate of \( 10^{-4} \) and a mini-batch size of 100. Moreover, we discount the gradient scale by multiplying the gradient value by \( 10^{-2} \).

We show the pixel prediction loss in mean square error (MSE) for multi-step future prediction in Table 1. For all
Figure 3: The prediction and reconstruction result for each task domain. For each task, we present 1 set of frames, where the four frames are organized as follows: (1) the ground-truth frame seen by the agent; (2) the predicted frame by the prediction model; (3) the reconstruction of autoencoder trained only with reconstruction loss; (4) the reconstruction of autoencoder trained after the second phase (i.e., trained with both reconstruction loss and code matching loss). Overall, the prediction model could perfectly produce frame output, while the fully trained autoencoder generates slightly blurred frames.

the task domains, the prediction losses are within a small scale. For multi-step prediction, as expected, the prediction loss increases with the increase of the prediction length. We demonstrate that the trained prediction models are able to generate realistic future frames which are visualized to be very close to the ground-truth frames in Figure 3.

<table>
<thead>
<tr>
<th>Game</th>
<th>1-step</th>
<th>3-step</th>
<th>5-step</th>
<th>10-step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breakout</td>
<td>1.114e-05</td>
<td>3.611e-04</td>
<td>4.471e-04</td>
<td>5.296e-04</td>
</tr>
<tr>
<td>Freeway</td>
<td>2.856e-05</td>
<td>9.339e-05</td>
<td>1.424e-04</td>
<td>2.479e-04</td>
</tr>
<tr>
<td>Frostbite</td>
<td>7.230e-05</td>
<td>2.401e-04</td>
<td>5.142e-04</td>
<td>1.800e-03</td>
</tr>
<tr>
<td>Ms-Pacman</td>
<td>1.413e-04</td>
<td>4.353e-04</td>
<td>6.913e-04</td>
<td>1.226e-03</td>
</tr>
<tr>
<td>Q-bert</td>
<td>5.300e-05</td>
<td>1.570e-04</td>
<td>2.688e-04</td>
<td>4.552e-04</td>
</tr>
</tbody>
</table>

Table 1: The prediction loss in MSE for the trained prediction model.

4.2 Evaluation on Hashing with Autoencoder and LSH

The architecture for the autoencoder model is identical to that shown in Figure 2 (down). The autoencoder is trained on a dataset collected in an identical manner as that for the prediction model. It is trained under two phases. In the first phase, it is trained with only reconstruction loss. We use Adam as optimization algorithm, $10^{-3}$ as learning rate, a mini-batch size of 100 and discount the gradient by multiplying $10^{-2}$. In the second phase, we train the autoencoder based on the loss in (4). Specifically, we use Adam as optimization algorithm, $10^{-4}$ as learning rate, a mini-batch size of 100, and $\lambda$ value of 0.01. We discount the gradient by multiplying the gradient value by $5 \times 10^{-3}$.

Overall, it is extremely challenging to match the state codes for the predicted frames and their corresponding seen frames while maintaining a satisfying reconstruction performance. We demonstrate this in Figure 5 (a) by showing the code loss, which is measured in terms of the number of mismatch in binary codes between a pair of predicted frame and its corresponding ground-truth frame. The presented result is derived by averaging over 10,000 pairs of codes. First, the result shows that without the second phase, it is impossible to perform hashing with autoencoder trained only by the reconstruction loss, since the average code losses are above 1 in all the domains and with distinct hash codes, the count values returned from querying the hash table are meaningless. Second, the result shows that after the training of the second phase, the code loss is significantly reduced.
Figure 4: Left: the predicted future trajectories for each action in Breakout. In each row, the first frame is the ground-truth frame and the following five frames are the predicted future trajectories with length 5. In each row, the agent takes one of the following actions (continuously): (1) no-op; (2) fire; (3) right; (4) left. Right: the hash codes for the frames in the same row ordered in a top-down manner. To save the space, four binary codes are grouped into one hex code, i.e., in a range of 0 to 15, and the color map is normalized linearly by the hex value.

Moreover, we use Breakout as an illustrative example to demonstrate the hashing can generate meaningful hash codes for predicted future frames (see Figure 4). For a given ground-truth frame, we show the predicted frames with length 5 for taking each action. It can be found that different actions lead to different trajectories of board positions. For the hash codes, as three of the actions, no-op, fire and left, lead to little change in frames, most of the future frames are hashed into the same code. The action right leads to the most significant changes in board position, so the codes for future frames are much more distinct than the rest. Meanwhile, Figure 4 also shows that the multi-step prediction model generates realistic future frames.

<table>
<thead>
<tr>
<th>Model</th>
<th>Breakout</th>
<th>Freeway</th>
<th>Frostbite</th>
<th>Ms-Pacman</th>
<th>Q-bert</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN-Random</td>
<td>401.2</td>
<td>30.9</td>
<td>328.3</td>
<td>2281</td>
<td>3876</td>
</tr>
<tr>
<td>A3C</td>
<td>432.42</td>
<td>0</td>
<td>283.99</td>
<td>2327.8</td>
<td>19175.72</td>
</tr>
<tr>
<td>A3C-CTS</td>
<td>473.93</td>
<td>30.48</td>
<td>325.42</td>
<td>2401.04</td>
<td>19257.55</td>
</tr>
<tr>
<td>pixelCNN</td>
<td>448.2</td>
<td>31.7</td>
<td>1480</td>
<td>2489.3</td>
<td>5876</td>
</tr>
<tr>
<td>DQN-Informed</td>
<td>0.93</td>
<td>32.2</td>
<td>1287.01</td>
<td>2522</td>
<td>8238</td>
</tr>
<tr>
<td>DQN-Informed-Hash</td>
<td>451.93</td>
<td>33.92</td>
<td>1812.10</td>
<td>3526.60</td>
<td>8827.83</td>
</tr>
</tbody>
</table>

Table 2: Performance score for the proposed approach and baseline RL approaches.

Among all the test domains, DQN-Informed-Hash outperforms DQN-Informed, and there are significant performance gains observed in each domain. Note that in Breakout, the agent fails to progress with DQN-Informed and always scores almost 0. It may due to that the kernel-based pixel distance evaluation metric used in DQN-Informed encourages the agent to explore states that is dissimilar from the recent history, which is insufficient to let the agent explore. Note that DQN-Informed-Hash demonstrates the superior performance with a deterministic exploration mechanism. It indicates that counting over the predicted future frames could provide a meaningful direction for exploration.

4.3 Evaluation on Informed Exploration

We integrate the proposed informed exploration framework into DQN algorithm [Mnih et al., 2015] and compare with the following baselines: (1) DQN that performs e-greedy with uniform random action selection, denoted by DQN-Random; (2) A3C [Mnih et al., 2016], (3) A3C with density model on Atari features [Bellemare et al., 2016], (4) exploration with pixelCNN-based density model [Ostrovski et al., 2017], (5) the state-of-the-art informed exploration approach proposed in [Oh et al., 2015], denoted by DQN-Informed. Our proposed approach is denoted by DQN-Informed-Hash. We use $q = 3$ as the length for hashing over future frames. We report the result in Table 2.

5 Conclusion

In this paper, we propose an informed exploration framework for deep RL domains with discrete action space. By incorporating a deep action-conditional prediction model over future transitions and a hashing mechanism based on a deep autoencoder model and LSH, we enable the agent to predict over the future trajectories and intuitively evaluate the novelty for each future action direction based on the hashing result. Empirical results on Atari 2600 domain show that the proposed informed exploration framework could significantly improve the exploration efficiency in several challenging deep RL domains.
Acknowledgments

This work is funded by NTU Singapore Nanyang Assistant Professorship (NAP) grant M4081532.020, MOE AcRF Tier-1 grant 2016-T1-001-159, and Microsoft Research Asia.

References


