SNAPSHOT-DRIVEN DEEP REINFORCEMENT LEARNING

by

Giang Dao

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

Charlotte

2022

ABSTRACT

GIANG DAO. Snapshot-driven deep reinforcement learning. (Under the direction of DR. MINWOO LEE)

Deep reinforcement learning (DRL) has suggested many effective solutions to complex problems. Despite the impressive achievements of DRL, insights into why DRL is effective are still limited. DRL is also known as a black-box model with high complexity which makes DRL becomes difficult to be interpreted in a human-understandable way to discover the rationale behind a DRL agent's behavior. Many prior research has proposed different techniques intended to improve interpretability and measured the effect of different interpretability methods on user trust, the ability to simulate models, and the ability to detect mistakes. Understanding DRL through interpretation methods can unlock knowledge to improve the quality of prediction, understand internal errors to be able to fix, answer why a model does not work, and improve the overall learning process.

Most existing interpretation approaches only assume a converged model, which cannot interpret the learning process of DRL and important concepts shaped. Therefore, the previous approaches are slow to produce a robust interpretation instantly. The sparsity of interpretation is another problem in the previous approaches because most of the known interpretation requires enormous human effort. Also, the frameworks for interpretation have not been used for supporting the learning process to have better performance. To address these challenges in DRL, snapshot-driven approaches, extracting a small number of examples (known as snapshots) for interpretation and using them for the improvement of learning, is proposed. Utilizing the snapshots, the change in the learning process's behavior can be interpreted. The snapshots can also provide an instant feedback to new samples for faster understanding of the decisions of a DRL agent. The understanding of DRL can improve the learning methods in diverse environmental settings. In order to fully validate the snapshot-driven learning, the following methods are proposed in this dissertation: 1) to extract snapshots from a DRL agent and understand the behavior, 2) to improve the learning process of the DRL agent and to evaluate the effectiveness of the snapshots on it, 3) to reduce the number of snapshots for easier interpretation, and 4) to apply snapshots usage in continual DRL settings for latent stabilization to prevent catastrophic forgetting problem.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

With the success of deep learning (DL) or neural networks, there are a lot of DL applications that solve complex tasks from unprocessed, high dimensional input data [2]. A wide range of applications has been built based on deep learning algorithms due to not only the effectiveness but the simplicity of the algorithms. Deep learning has suggested many effective solutions to complex problems such as image recognition [3], language understanding [4], robotics [5], etc. The exponentially increased numbers of research in deep learning[1] and by searching and observing keyword "deep learning" on Google Scholar in the past 9 years (2012 - 2021) shows the much growing attention towards research utilizing deep learning.

Although the popularity of applying neural networks has been increased, concerns in real world applications also have grown as deep neural networks are black box models, which do not allow interpreting what they have learned to understand the model's decision process [6]. To solve complex problems in real applications, deep learning models have become increasingly complex with different architectural structures and the huge number of parameters needed to be learned. The complexity of the model makes it extremely difficult to interpret when deep learning does not work or learn properly. Thus, when an erroneous event happens, finding the causes and fixing them can be a tedious process that requires time and effort. For example, it is very hard for neural networks to explain why self-driving car crashes (i.e., Tesla and Uber self-driving car accidents) or what causes the wrong decision due to lack of interpretation. Therefore, engineers spend a lot of time investigating the case to fix the error.

With the increasing complexity of deep neural networks, the need for understanding

[1]https://www.youtube.com/watch?v=ZHoNF28Nj98

neural networks thus enhance the trustworthiness of the deployed system led to an effort to establish regulations to require interpretability of AI systems. The European Union has adopted the General Data Protection Regulation (GDPR)[2] which became law in May 2018. The GDPR stipulated "a right of interpretability" in the clauses on automated decision-making. The inequality or bias, the safety of human users, industrial liability, and ethics concerns from the GDPR are endangered without establishing trustworthiness based on interpretation (thus understanding) of the systems. Therefore, the demand for interpretability has created a new line of research to understand how a neural network makes a decision not only for the curiosity of researchers but also becomes a requirement for corporations.

1.1 Challenges of Deep Reinforcement Learning

Understanding what has been learned from neural networks has become a major problem in interpretability research. There are three major categories of interpretable deep learning [7]: self-interpretable system, representation analysis, and re-approximation. The self-interpretable system has three noticeable methods: attention mechanism, disentanglement learning, and adversarial examples. Attention mechanism attempts to understand the relationship between information [8]. Disentanglement learning is a method to understand high level concepts from low level information [9]. Adversarial examples are used for interpreting the vulnerability of the learning system [10]. There are three different methods to get representation for analysis: layer and individual neurons, vector grouping, and saliency map. Visualization of the layer and individual neurons are helpful to understand which features have been learned [11]. Vector analysis is used for reducing high dimensional space into 2D or 3D which is easier for a computer to visualize [12, 13]. Saliency map reveals significant information that affects the model decision [14]. Reapproximation approaches apply inherently interpretable models to approximate deep neural networks. This category include linear approximation [15] utilizes a linear model, decision tree method [16], and rule extraction [17].

[2]https://gdpr-info.eu/

The majority of the aforementioned approaches, however, only focuses on explaining the neural networks after the learning model is converged. This makes it difficult to understand how a DRL agent shapes the features from inputs, and why the features are important for DRL agent to produce a decision or a prediction. Understanding the learning process allows us to fix possible errors, improve learning efficiency, and provide insight of a decision process. The limitation of previous approaches are: slow to produce a robust interpretation in timely manner, a fast way to understand the interpretation with instant feedback, sparsity of analysis, and using the interpretation for improving performance of DRL agent.

1.2 Problem Statement

With the increasing complexity of deep neural networks and the legal requirements, understanding the model becomes harder but necessary. Also, the previous works are limited to interpreting the deep networks only when the model finishes training. Moreover, the previous works are slow to produce a robust interpretation in quantity and quality. Therefore, my work aims to develop a method to help human interprets DRL model to understand not only the trained agent but also the whole training process to resolve common challenges of existing approaches. My work also provides methods of utilizing the developing snapshots to further enhance performance of DRL agent in different settings. The problems that this dissertation discusses are summarized as follows:

- A robust method to help understanding of DRL agent's decision process.
- Usage of the snapshots for enhancing the efficiency of DRL agent's training.
- A sparse concerns to help human interpretation and analysis.
- How the snapshots can help in continual DRL problems.

1.3 Proposed Works

Fig. 1.1 show the overall process of my dissertation from retrieving the important samples as snapshots, enforcing the number of snapshots to be small for easier interpretation,

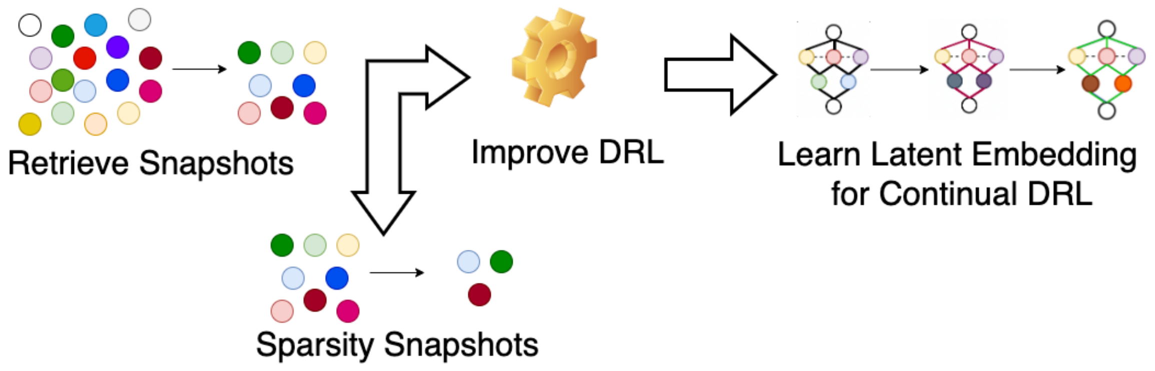


Figure 1.1: Dissertation framework to retrieve, enforce the snapshots and using snapshots for improving DRL agent and finding latent embedding for continual DRL.

improving the performance of a DRL agent, and finding good latent embedding to support continual DRL process. My dissertation aims to tackle the two major challenges of interpretation: robustness and sparsity of the DRL agent's interpretation. My work is also provide methods to use the proposed interpretation to improve the training of DRL agent in different settings. For this, a sparse analysis method is proposed that captures a small number of important samples, named as *snapshots*. The snapshots become "evidence" to explain deep learning instead of analyzing the whole dataset. The snapshots can also be used for enhancing the performance of DRL agent and provide good representations of inputs to prevent forgetting problem in continual DRL. Approaches are discussed to 1) achieve the snapshots from deep reinforcement learning process, 2) utilize the achieved snapshots to improve learning, 3) reduce the number of snapshots for faster and easier understanding of the agent, and 4) apply the snapshots in continual deep reinforcement learning problems to help prevent reduced performance of old tasks while training new tasks.

First, a re-approximation approach is proposed to neural networks by utilizing a sparse Bayesian learning framework for extraction of meaningful snapshots. The proposing method is applied to deep reinforcement learning to collect snapshots to better understand how the behavior changes during the learning process of a learning agent, provide instant feedback to explain why a agent takes an action and analyze the snapshots for explanation. Deep reinforcement learning is chosen as our initial domain of experiments because it is easier to understand the changes in the learning behavior of an agent. Next, these effect of snapshots to the learning process of a deep reinforcement learning agent is examined by using the snapshots as a part of memory replay. Then the number of snapshots is reduced that can be taken during the process. Last, the snapshots monitor process are leveraged to help improving and stabilizing training and adaptation processes in continual deep reinforcement learning.

The proposal is organized as follows. The dissertation first describes the related research and the challenges in Chapter 2. In Chapter 3, core concepts used for the proposal are introduced: deep learning, reinforcement learning, and sparse Bayesian learning. How to obtain snapshots to explain deep reinforcement learning agent is described in Chapter 4 with details in method development, experiments, and discussion. The application of the snapshots to show the enhancement in the deep reinforcement learning process is provided in Chapter 5. How to reduce number of snapshots for easier understanding is described in Chapter. 6. A proposed approach to generalize the snapshots for continual deep reinforcement learning is introduced in Chapter 7. The conclusion for my dissertation and future works are provided in Chapter 8.

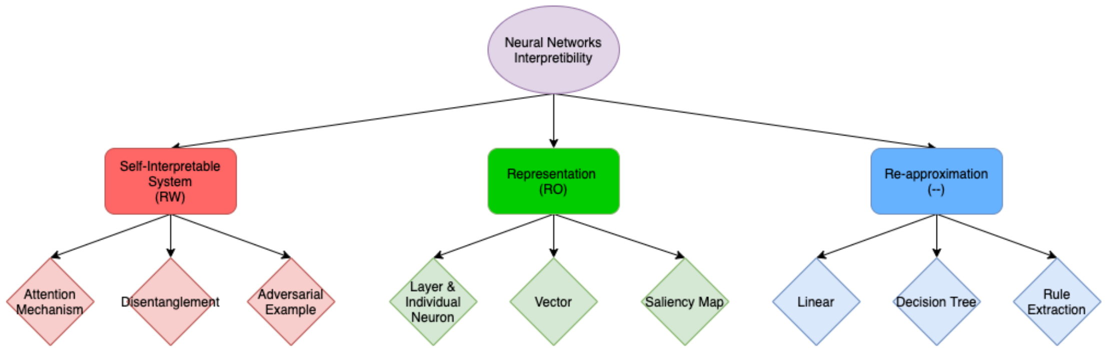


Figure 2.1: Splitting neural networks interpretability approaches into sub-categories and its methods of interpretation. The required the accessibility to the model for interpretation are denoted as: RW means read/write, RO means read-only, and – means no access requirement.

Fig. 2.1 depicts a high-level view of interpretability research in neural networks. There exists three main approaches to interpret neural networks. These three main branches are categorized by how much accessibility and permission a method needs to have to interpret a neural network model: requiring full access and modification (*Self-interpretable System*), requiring full access without modification (*Representation Analysis*), or requiring no access or modification privilege (*Re-approximation*) as follows:

1. *Self-Interpretable System* is a method that designs a neural network in a way that it can somewhat explain its decision. This approach requires to fully access the model to be able to modify and architect the neural network.

2. *Representation Analysis* is an approach to understand individual sub-system inside the neural network by simply observing the weights and gradient updates. As it is not necessary to modify the neural network model, only full read access is enough for methods in this category.

3. *Re-approximation* uses genuinely interpretable models to understand the neural networks. This approach does not read or modify the model to understand it. It simply monitors input and output of the model and re-approximates the neural networks for interpretation.

The interpretibility system is split into three main branches based on the user accessibility to the neural networks. For example, a neural network's creator can use all of the three branches to explain their model which they can modify the model to have better understanding. Users, who download models online for their application, cannot modify the model but can access the internal to understand the model's weights. Application programing interference (API) users, who call a neural networks API to get a result, can only understand the model by approximating it.

2.1 Self-Interpretable System

2.1.1 Attention Mechanism

Attention mechanism attempts to understand the relationship between information. Attention in deep learning is a vector of importance weights which shows how an input element correlates to target output. Attention weights can be formulated as a probability distribution of correlation between a target with other sources. A higher probability results from a higher correlation between a target and a source. There are two types of attention mechanisms: hard-attention and soft-attention. Hard-attention strictly enforce attention weights to either 0 for non-correlated or 1 for correlated (Bernoulli distributions). Soft-attention represents attention weights with more flexible probability distributions. With the flexibility, soft-attention recently dominates over hard-attention in most of the applications. An example of computing soft-attention weights is using softmax function to compute the correlation between a target with other sources:

$$\alpha_{ts} = \frac{exp(score(h_t, \bar{h}_s))}{\sum_{s'=1}^{S} exp(score(h_t, \bar{h}_s))}.$$

Attention mechanism has achieved remarkable success in natural language translation with different score functions as well as other optimization tricks [18, 19, 20, 21]. Not only showing the capability of self-interpretability in natural language processing tasks, attention mechanisms can also be designed to interpret neural network decision by looking at the attention pixels in different tasks: image classification [22, 23], image segmentation [24], and image captioning [25, 26, 27, 28]. Even though attention units reveal interpretable information, they are hardly evaluated because of the robustness in the comparison process. Therefore, Das et al. [29] has created human attention datasets to compare the attention between neural networks and humans to see if they look at the same regions when making a decision. To enforce the neural networks to look at the same region as human and to have similar human behavior, a method to train attention mechanisms explicitly through supervised learning with the attention datasets by constraining the machine attention to be similar to human attention in the loss function was proposed [30].

subsubsectionDisentanglement Learning Disentanglement learning is a method to understand a high level concepts from low level information. Disentanglement learning is a learning process that learns disentangled representations in lower dimensional latent vector space where each latent unit represents a meaningful and independent factor of variation. For example, an image contains a black hair man will have representation of gender: male, and hair color: black encoded in the latent vector space. A disentangled representation can be learned explicitly from training a deep neural network. There are two different ways that can be considered to learn disentangled representation. The disentangled representation can be learned through generative adversarial networks (GAN) [31] and variational autoencoder (VAE) [9].

GAN contains 2 main parts (generator and discriminator) which learns to map a vector representation into higher dimensional data. The generator takes a vector representation to generate a data point. The vector representation usually has lower dimension than the generated data point. The discriminator takes a data point and outputs true if the data is real and false if the data is generated. After the learning process, the vector representation usually provides high level information of the data. InfoGAN [32] is a scalable unsupervised approach to increase the disentanglement by maximizing the mutual information between subsets of latent variables and observations within the generative adversarial network. Auxiliary classifier GAN [33] extends InfoGAN by controlling a latent unit with actual categorical classes. This is simply adding a controllable disentangled unit with a known independent factor.

Instead of learning to map a vector representation into a data point, VAE learns to map a data point to a lower vector representation. VAE minimizes a loss function:

$$\mathcal{L}(\theta, \phi, x) = \frac{1}{L}\sum_{l=1}^{L}(logp_\theta(x|z^l)) - D_{KL}(q_\phi(z|x)||p_\theta(z)),$$

has been shown as a promising direction to explicitly learn disentanglement latent units with β-VAE [34]. β-VAE magnifies the KL divergence term with a factor $\beta > 1$:

$$\mathcal{L}(\theta, \phi, x) = \frac{1}{L}\sum_{l=1}^{L}(logp_\theta(x|z^l)) - \beta D_{KL}(q_\phi(z|x)||p_\theta(z)),$$

Further experiment [35] showed the disentangled and proposed modification of KL divergence term in the loss function to get improvement in reconstruction:

$$\mathcal{L}(\theta, \phi, x) = \frac{1}{L}\sum_{l=1}^{L}(logp_\theta(x|z^l)) - \beta|D_{KL}(q_\phi(z|x)||p_\theta(z)) - C|,$$

with C is a gradually increasing number to a large enough value to produce good reconstructions. The first term, $\frac{1}{L}\sum_{l=1}^{L}(logp_\theta(x|z^l))$, is an expected negative reconstruction error, while the second term, Kullback-Leibler divergence of approximate posterior from the prior $D_{KL}(q_\phi(z|x)||p_\theta(z))$, acts as a regularizer. The β magnifies the KL divergence term to have better constrain on the prior and the posterior. Since KL divergence term can grow to infinity, the gradually increasing number C makes the term stay numerically computable.

Both GAN and VAE methods can be trained in such a way that each individual latent unit is corresponding to a specific feature. [36] observed the disentangle learning leads to a better abstract reasoning. Graph construction ([37]) and decision trees (see more in Chapter 2.3.2) are additional methods using disentangle latent dimensions. High-level concepts can also be represented by organizing the disentanglement with capsule networks by [38]. Disentanglement learning is not only designed for interpretability, it recently shows huge improvement in unsuppervised learning tasks via encoding information ([39, 40]).

2.1.2 Adversarial Examples

Adversarial examples can be used for interpretation of neural networks bu revealing the vulnerability of the neural networks. An adversarial attack is a method to deceive a neural network model. The main idea is to slightly perturb the input data to get a false prediction from the neural networks model, although the perturbed sample makes no different to human perception. Early work has been proposed [10] to find the perturbation noise by minimizing a loss function:

$$\mathcal{L} = loss(\hat{f}(x+\eta), l) + c \cdot |\eta|,$$

where η is the perturbed noise, l is the desired deceived target label to deceive the neural networks, and c is a constant to balance the original image and the perturbed image. Goodfellow et al. [41] proposed a fast gradient method to find η by the gradient of the loss w.r.t to the input data: $\eta = \epsilon \cdot sign(\nabla_x \mathcal{L}(x, l))$. However, the two methods require a lot of pixels to be changed. Yousefzadeh and O'Leary [42] reduced the number of pixels using flip points. It is al possible to deceive a neural network classifier with only one pixel change [43].

However, it is hard to intentionally modify a digital image when the image is captured by a camera without hacking into a system. A method to print stickers that can fool a neural networks classifier [44] was designed. Similarly, the usage of 3D printer to print a turtle but is classified as a rifle [45] has also implemented.

2.2 Representation Analysis

2.2.1 Layers & Individual Neurons Analysis

Visualization of layer and individual neurons are helpful to understand which features have been learned. The information flows in neural networks can be subdivided into layers and individual neurons. A single individual neuron can be understood by visualizing the input patterns that maximize the neuron's response. With the neuron's responses visualization, researchers inpteret which information has been learned and passed through different layers and individual neurons of the neural networks.

Each individual neurons to observe the weights can directly be visualized. By visualizing and observing each layers of a small neural network, the neural network is shown to learn from simple concepts to high level concepts through each layer [11]. A neural network model first learns to detect edges, angles, contours, and corners in a different direction at the first layer, object parts at the second layer, and finally object category in the last layer. This sequence consistently happens during training different neural networks on different tasks.

Instead of visualizing neurons directly, researchers found out that the neurons' gradient can also be observed to reveal where important information parts come from. Gradient-based methods, which propagates through different layers and units [46, 47], were proposed. The gradient of the layers and units highlights areas in an image which discriminate a given class. An input can also be simplified which only reveals important information [48].

A method to synthesize an input that highly maximizes a desired output neuron using activation maximization [49] by utilizing gradients. For example, the method can synthesize an image of lighter that the neural network classifier would maximize the probability of the lighter. Mordvintsev et al. [50] has successfully improved style transfer, which modifies a content image with a style of different image, by maximizing the activation difference of different layers. There is a survey of different methods for visualization of layer representations and diagnosed the representations [51].

Another way to understand a single individual neuron and layers is to qualitatively validate its transferability to different tasks. A framework for quantifying the capacity of neural network transferability was introduced by comparing the generality versus the specificity of neurons in each layer [52]. Network dissection method [53] measures the ability of individual neurons by evaluating the alignment between individual neurons and a set of semantic concepts. By locating individual neurons to object, part, texture, and color concepts, network dissection can characterize the represented information from the neuron.

2.2.2 Vectors Analysis

Vector representations are taken before applying a linear transformation to the output from a neural network model. However, the vector representation most likely to have more than three dimensions which are hard to be visualized by computer. Vector visualization methods aim to reduce the dimension of the vector to two or three dimensions to be able to visualize by computer. Reducing the vector to two or three dimensions to visualize is an interesting research area. PCA [12] designs an orthogonal transformation method to convert a set of correlated variables into another set of linearly uncorrelated variables (called principal components). The higher impact principal component has a larger variance. T-distribution stochastic neighbor embedding (t-SNE by Maaten and Hinton [13]) performs a non-linear dimension reduction for visualization in a low dimensional space of two or three dimensions. t-SNE constructs low dimensional space probability distribution over pairs of high dimensional objects and minimize KL divergence with respect to the locations of the points on the map.

Vector representation visualization methods are well known for helping humans understand high dimensional data. For example, if a neural network performs well in a classification task, the vector representations need to be clustered together if they have a similar label. In order to ensure the vector representations are clustered, human needs to visualize the vector and validates the assumption, especially in unsupervised learning where no label is given. Both of the methods reduce high dimensional space to lower dimensions (usually two or three) for an easy visualization that helps human understand and validate the neural networks. PCA and t-SNE are widely used by researchers to visualize high dimension information.

2.2.3 Saliency Map

Saliency map reveals significant information that affects the model decision. Zeiler and Furgus exemplified the saliency map by creating a map shows the influence of the input to the neural network output [14]. There are different techniques built upon the saliency map which showing highly activated areas or highly sensitive areas. The saliency method requires the direct computation of gradient from the output of the neural network with respect to the input. However, such derivatives are not generalized and can miss important information flowing through the networks.

Researchers have been working on the solution to smoothly derive the required gradient for the saliency map. Layer-wise relevance propagation [54] is a method to identify contributions of a single pixel by utilizing a bag-of-words features from neural network layers. By simply modifying the global average pooling layer combined with class activation mapping (CAM), a good saliency map is shown [55]. DeepLIFT [56] compares the activation of each neuron with reference activations and assigns contribution scores based on the difference. A weighted method is used for CAM to smooth the gradient [57]. An integrated gradient method is used to satisfy the sensitivity and implementation variance of the gradient [58]. De-noising the gradient by adding noise to perturb original input then average the saliency maps collected [59] also shows a better saliency map. An application of using saliency map to interpret why a deep reinforcement learning agent behaves [60].

2.3 Re-approximation with Interpretable Models

2.3.1 Linear Approximation

A linear model can be he most simplified model that can provide interpretation of the observable outcomes. Linear model uses a set of weights w and bias b to make prediction: $\hat{y} = wx + b$. The linearity of the relationship between features, weights, and targets makes the interpretation easy. The weights of the linear model to understand how an individual input feature impacts the decision can be analyzed.

Local Interpretable Model Agnostic (LIME) [15] exemplified the linear approximation approach to classification problems. LIME first perturbs input data to probe the behavior of the neural networks. A local linear model is trained through the perturbed input and neural network output on the neighborhood information of the input.

2.3.2 Decision Tree

Linear approximation assumes input features to be independent. Therefore, linear approximation fails when features interact with each other to form a non-linear relationship. Decision trees split the data multiple times according to certain cutoff values in the data features. The approach results in an algorithm similar to nested if-then-else statements to compare (smaller/bigger) input features with corresponding threshold numbers. The interpretation is fairly simple by following the instruction from the tree root node to the leaf node. All the edges are connected by ‘AND’ operation.

Artifitial Neural Networks - Decision Tree (ANN-DT) [16] is an early work that converts a neural network into a decision tree. ANN-DT applied sampling methods to expand the training data using nearest neighbors to create the decision tree. Sato and Tsukimoto designed Continuous Rule Extractor via Decision tree (CRED) to interpret shallow networks [61]. By applying RxREN [62] to prune unnecessary input features and C4.5 algorithm [63] to create a parsimonious decision tree, an extension of CRED into DeepRED [64] is introduced to be able apply to deep neural networks. The decision tree method is also applied to interpret a reinforcement learning agent's decision making [65].

2.3.3 Rule Extraction

Similar to decision trees, rule extraction methods use nested if-then-else statements to approximate neural networks. While decision trees tell a user where to follow (left or right) in each node, the rule-based structures are sequences of logical predicates that are executed in order and apply if-else-then statements to make decisions. A decision tree can be transformed to a rule-based structure and vice versa. Rule extraction is a well-studied approach in decision summarization from neural networks [66]. There are two main approaches to extract rules from neural networks: decompositional and pedagogical approaches.

Decompositional approaches mimics every individual unit behavior from neural networks by extracted rules. Knowledgetron (KT) method [17] sweeps through every neural unit to find different thresholds and apply if-then-else rules. The rules are generated based on input rather than the output of the preceding layer in a merging step. However, the KT method has an exponential time complexity and is not applicable to deep networks. The KT method was improved to achieve the polynomial time complexity [67]. Fuzzy rules was also created from neural network using the decompositional approach [68]. Towell et al. [69] proposed M-of-N rules which explain a single neural unit by clustering and ignoring insignificant units. Fast Extraction of Rules from Neural Networks (FERNN) [70] tries to identify meaningful neural units and inputs. Unlike other reapproximation methods, the aforementioned decompositional approaches require a full access to the information of neural network models.

Pedagogical approaches are more straightforward than decompositional approaches by extracting rules directly from input and output space without sweeping through every layers and units. Validity interval analysis [71] identifies stable intervals that have the most correlation between input and output to mimic behavior of the neural networks. the pedagogical approach can also use sampling methods [72, 73, 74] to extract the rules.

2.4 Challenges

The trade-off of interpreting neural network exists between the accuracy and robustness of a neural network and the meaningful or simpleness of interpretation. The most accurate and robust model does not guarantee an interpretation of the network in an easy way. The simple and meaningful interpretation might not be easy to learn from a robust method. It is thus challenging when access to neural networks model is not provided to neither re-design nor extracting meaningful information from the model. Reviewing the interpretation methods, two challenges for interpreting neural networks are identified: robust interpretation and sparsity of analysis. **These challenges can be solved by snapshot-driven method which is a quick process to extract a small number of important samples to understand the decision making process.**

2.4.1 Robust Interpretation

Current approaches are slow to produce robust interpretation in a timely manner. Self-interpretable systems, even though the interpretation is fast on inference, still need to be trained for a long time. The representation systems need heavy computation in order to achieve visualization results. Re-approximation methods take a long time for both training to approximate neural networks as well as produce interpretation.

Noisy interpretation can severely harm trust of the model. A neural network is trained from the data, possibly training data often cause erroneous interpretation because of errors in labeling process. This phenomenon happens mostly with self-interpretable systems since the objective function designed to optimize the data-only, not the knowledge. The objective function might not be well-covered to interpret the problem that makes the interpretation harder. The representation methods can provide a lot of misleading information from layers and individual neurons, which are not related to human perceptions. Re-approximation methods have limited performance compared to the original neural networks model, so misleading towards the poor interpretation.

2.4.2 Sparsity of Analysis

For each method, interpretations are made from individual samples or a lot of different visualizations. If a problem is scaled up with a large number of samples, a tremendous amount of observations and human effort are required. The problem becomes worse if samples not from the dataset need to be interpreted. For example, in order to interpret the reasoning behind a neural network classifier, human needs to analyze different saliency maps from different input samples to validate the reasoning. With that being said, researchers should concern about sparsity of analysis by reducing the number of visualizations that human needs to analyze. The sparsity is one of the main challenge that need to be addressed to lessen human arduous effort in interpreting neural networks due to the large amount of data as well as computation units. A method to recognize a meaningful smaller subset of the whole dataset to interpret needs to be proposed. From the meaningful subset, there is also an interpretation between the relationship from different samples with different subsets that can be done.

This chapter is reused from Demysifying Deep Neural Networks Through Interpretation: A Survey [7] with permission from the authors.

CHAPTER 3: BACKGROUND

3.1 Deep Learning

A deep neural network (DNN) is a mathematical model that has been motivated by the functioning of the brain. Researchers use the DNN to analyze data instead of providing a biologically realistic model. The basic idea of a deep learning model is mapping an input $\mathbf{x} \in \mathbb{R}^n$ (n is the number of input features) to a prediction $\hat{y}$ through a series of linear and nonlinear transformations. Each linear transformation carries a set of weights $\mathbf{w} \in \mathbb{R}^{n \times k}$ (k is the number of output features) and bias $b \in \mathbb{R}^k$ and performs a linear mapping such that: $\mathbf{z} = \mathbf{w}^\top \mathbf{x} + b$. After each linear transformation, there is a nonlinear function, denoted by σ, to convert the linear transformation into nonlinear to better fit the data. The series of functions/transformations are represented as:

$$\hat{y} = \sigma_n(\mathbf{z}_n) \text{ where } \mathbf{z}_n = \sigma_{n-1}(\mathbf{w}_n^\top \mathbf{z}_{n-1} + b_n) \text{ with } \mathbf{z}_0 = \mathbf{x} \text{ and } n \in \mathbb{N}.$$

Nonlinear functions, also called activation functions, come in many different forms that map the linear relationships into nonlinear relationships. The most frequently used nonlinear functions are sigmoid, hyperbolic tangent (tanh), softmax [75], and Rectified Linear Unit (ReLU) [76]. Sigmoid reduces the limit of the value from $(-\infty, \infty)$ to $(0, 1)$ using mathematical function, $f(z) = \frac{1}{1+\exp(-z)}$. Tanh provides the limit $(-1, 1)$ using mathematical function, $f(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$. Softmax function is mostly used for categorical distributions where each category is represented by a probability number that sums up to one such that $f_{z_i} = \frac{\exp(z_i)}{\sum_{j=1}^{N} \exp(z_j)}$. ReLU is the most used nonlinear function because it provides a fast and direct gradient of 0 or 1. ReLU only keeps positive values and zeros out negative values such that $f(z) = \max\{0, z\}$.

Each nonlinear and linear mapping sequence is called a hidden layer. In the deep learning context, since the transformations are functions, the prediction is denoted as a function as: $\hat{y} = f(\mathbf{x}|\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is a set of parameters $(\mathbf{w}_1, b_1, \mathbf{w}_2, b_2, \ldots, \mathbf{w}_n, b_n)$. The final layer of the neural network is called the output layer. During the neural network training, $f(\mathbf{x}|\boldsymbol{\theta})$ is derived to match $f^*(\mathbf{x})$ where $f^*(\mathbf{x})$ is the true/optimal function which is unknown. Each sample $\mathbf{x}$ is accompanied by a label $y = f^*(\mathbf{x})$. The training sample specifies directly what output layer must produce at each point $\mathbf{x}$. It must produce a value $\hat{y}$ that is close to y.

A neural network model is trained by solving optimization problem that minimizes an objective function. The objective function is depended on the problem. The most frequently used objective functions in deep learning are cross-entropy for classification problems and mean squared error for regression problems. The optimization problem is typically solved in a gradient-based method such as stochastic gradient descent [77], adaptive moment estimation (Adam) [78], rectified Adam [79] where the gradient $\frac{\partial J}{\partial \boldsymbol{\theta}}$ is computed with the chain rule.

3.2 Reinforcement Learning

Reinforcement learning (RL) is a process of learning a policy from an agent's interaction with an environment. From the collected samples via the interactions, the agent evaluates its value on each move to achieve a policy to reach a better goal. The agent picks an action and then observe responses from the changes of environment. The responses include a new situation and a feedback/reward for the previous situation and chosen action. The accumulation of reward is an objective that the agent wants to maximize.

The RL problem is formulated as a Markov Decision Process (MDP). An MDP is defined as a tuple $(S, A, P^a_{ss'}, R, \gamma)$ for each time step $t \in \mathbb{N}$. Given a state $s_t \in S$ and an action $a_t \in A$ $P^a_{ss'}$, there is a transition probability $P^a_{ss'}$ that the environment turn into the next state $s' = s_{t+1} \in S$ and reward $r_{t+1} \in \mathbb{R}$ is observed from the transition. In an environment specified by the MDP, a reinforcement learning agent aims to maximize the total long-term reward. In general, the long-term reward is represented by the expected sum of discounted

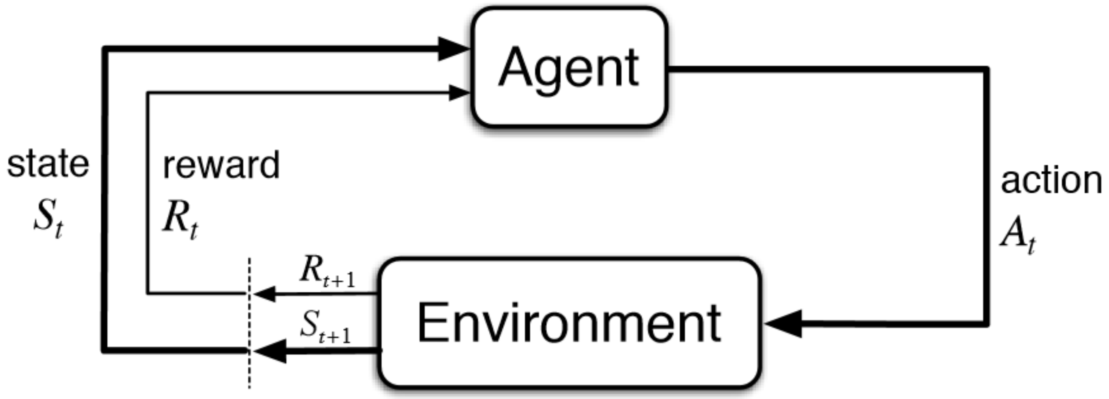


Figure 3.1: Reinforcement Learning Feedback Loop.

rewards as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + r_T = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1},$$

where $\gamma \in (0, 1]$ is a discount factor, and T is the time step when the environment terminates. The discount factor prevents infinity summation and tells the agent tries to get the reward as early as possible since the same reward is discounted in longer time horizon.

To achieve the goal, reinforcement learning algorithms estimate a value for each state the agent is at (state value function $V(s)$) or a value for each pair of state and action the agent performs (state-action value function $Q(s, a)$). The state value function $V(s) \in \mathbb{R}$ represents how good a state is in numerical value. The state-action value function $Q(s, a) \in \mathbb{R}$ also outputs a numerical value representing how good the taken action given a state. Both V and Q can be used for learning a better policy $\pi(s)$. Policy π can be represented as a function mapping from state space to action space such that $\pi : S \to A$.

The state value of a policy V^π can be estimated as the *expected* return as

$$V^\pi(s) = \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t r_{t+1} | s = s_t, \pi\Big].$$

The state value function satisfies Bellman equation such that

$$V^{\pi}(s) = \mathbb{E}[r_{t+1} + \gamma V^{\pi}(s_{t+1})|s = s_t, \pi].$$

This Bellman equation shows a relationship between the state value of current state and the next state. State-action value function is also estimated as the expected return as

$$Q^{\pi}(s,a) = \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t r_{t+1}|s = s_t, a = a_t, \pi\Big].$$

To see the relationship with the next state and next action, state-action value function can also be expressed with Bellman equation:

$$Q^{\pi}(s,a) = \mathbb{E}[r_{t+1} + \gamma \max_{a_{t+1}} Q^{\pi}(s_{t+1}, a_{t+1})|s = s_t, a = a_t, \pi].$$

Reinforcement learning agent looks for an optimal policy that maximizes either V^{π} or Q^{π}, which can be denoted by V^* and Q^* respectively such that:

$$\begin{aligned} V^*(s) &= \max_{\pi} V^{\pi}(s) \\ V^*(s) &= \max_{a} Q^*(s,a) = \max_{a}\max_{\pi} Q^{\pi}(s,a) \\ \text{therefore } Q^*(s,a) &= \mathbb{E}[r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}|s = s_t, a = a_t)] \\ &= \mathbb{E}[r_{t+1} + V^*(s_{t+1}|s = s_t)]. \end{aligned}$$

3.3 Sparse Bayesian Learning

Sparse Bayesian learning [80, 81] transforms i-th input data $\mathbf{x}_i$ into features such that $\boldsymbol{\phi}_i = k(\mathbf{x}_i, \mathbf{X})$ where k is the kernel function measures the similarity between $\mathbf{x}_i$ and $\mathbf{X}$, and $\mathbf{X}$ are the snapshots in the model. $\boldsymbol{\Phi}$ is defined as a matrix composed of feature vectors transformed by the kernel functions $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1, \boldsymbol{\phi}_2, \ldots, \boldsymbol{\phi}_n]$ with n is the number of samples. Let m is the number of snapshots at any moment: $\boldsymbol{\Phi} \in \mathbb{R}^{n \times m}$. Sparse Bayesian learning starts the learning process with a single random snapshot.

Sparse Bayesian learning assumes that the target t is approximated by a weighted sum of the feature vectors $\hat{t}$ with some noise such that:

$$t = \hat{t} + \epsilon = \mathbf{\Phi}\mathbf{w} + \epsilon$$

where $\mathbf{w} \in \mathbb{R}^m$ is the feature weight and $\boldsymbol{\epsilon}$ is a zero-mean Gaussian noise with variance $\sigma^2 = 0.1 \times var(t)$. $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_m)^\top$ is defined as a set of parameters controlling the strength of the prior over the corresponding weights to be infinity except for the starting snapshot:

$$\alpha_i = \frac{||\boldsymbol{\phi}_i||^2}{||\boldsymbol{\phi}_i^\top t||^2/||\boldsymbol{\phi}_i||^2 - \sigma^2}.$$

Having defined the prior $p(\mathbf{w}, \boldsymbol{\alpha}, \sigma^2)$, Bayesian inference proceeds by computing, from Bayes' rule, the posterior distribution of the target over all unknowns given the data

$$p(\mathbf{w}, \boldsymbol{\alpha}, \sigma^2|t) = \frac{p(t|\mathbf{w}, \boldsymbol{\alpha}, \sigma^2)p(\mathbf{w}, \boldsymbol{\alpha}, \sigma^2)}{p(t)}. \tag{3.1}$$

The posterior can be decomposed as

$$p(\mathbf{w}, \boldsymbol{\alpha}, \sigma^2|t) = p(\mathbf{w}|t, \boldsymbol{\alpha}, \sigma^2)p(\boldsymbol{\alpha}, \sigma^2|t). \tag{3.2}$$

The posterior distribution over the weights $p(\mathbf{w}|t, \boldsymbol{\alpha}, \sigma^2)$ is given by

$$p(\mathbf{w}|t, \boldsymbol{\alpha}, \sigma^2) = (2\pi)^{-(n+1)/2}|\mathbf{\Sigma}|^{-1/2} \exp\Big\{ -\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1}(\mathbf{w} - \boldsymbol{\mu})\Big\} \tag{3.3}$$

where the posterior mean and covariance are respectively

$$\boldsymbol{\mu} = \sigma^{-2}\mathbf{\Sigma}\mathbf{\Phi}^\top t, \tag{3.4}$$

$$\mathbf{\Sigma} = (\sigma^{-2}\mathbf{\Phi}^\top\mathbf{\Phi} + \mathbf{A})^{-1}, \tag{3.5}$$

with $\mathbf{A} = \text{diag}(\alpha_1, \alpha_2, \ldots, \alpha_n)$, a square zero matrix with the diagonal has the α values. Essentially, I have $\mathbf{w} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$.

Sparse Bayesian learning becomes the search for the parameters posterior mode to maximize $p(\boldsymbol{\alpha}, \sigma^2|t) \propto p(t|\boldsymbol{\alpha}, \sigma^2)p(\boldsymbol{\alpha})p(\sigma^2)$ with respect to $\boldsymbol{\alpha}$. The term $p(t|\boldsymbol{\alpha}, \sigma^2)$ needs to be maximized, which can be computed as:

$$p(t|\boldsymbol{\alpha}, \sigma^2) = (2\pi)^{-n/2}|\sigma^2\mathbf{I} + \mathbf{\Phi}\mathbf{A}^{-1}\mathbf{\Phi}^\top|^{-1/2} \exp\Big\{ -\frac{1}{2}t^\top(\sigma^2\mathbf{I} + \mathbf{\Phi}\mathbf{A}^{-1}\mathbf{\Phi}^\top)^{-1}t\Big\}. \tag{3.6}$$

I can replace $\mathbf{C} = \sigma^2\mathbf{I} + \mathbf{\Phi}\mathbf{A}^{-1}\mathbf{\Phi}^\top$ and $\mathbf{m} = t = \mathbf{\Phi}\boldsymbol{\mu}$ as the covariance and mean respectively. The covariance of the marginal likelihood $\mathbf{C}$ can be decomposed as: $\mathbf{C} = \mathbf{C}_{-i} + \alpha_i^{-1}\phi_i\phi_i^\top$ where $\mathbf{C}_{-i} = \sigma^{-2}\mathbf{I} + \sum_{j\neq i}\alpha_j\phi_j\phi_j^\top$ is $\mathbf{C}$ with the contribution of basis vector i removed. The *sparsity factor*

$$s_i = \boldsymbol{\phi}_i^\top\mathbf{C}_{-i}^{-1}\boldsymbol{\phi}_i, \tag{3.7}$$

measures the extent of overlaps of $\boldsymbol{\phi}_i$ with the existing other bases. The *quality factor*

$$q_i = \boldsymbol{\phi}_i^\top\mathbf{C}_{-i}^{-1}t, \tag{3.8}$$

measures the alignment error of $\boldsymbol{\phi}_i$ when the $i-$th output is excluded.

Three cases need to be considered:

- If $q_i^2 > s_i$ and $\alpha_i < \infty$, re-estimate α_i.
- If $q_i^2 > s_i$ and $\alpha_i = \infty$, add ϕ_i to the model and re-estimate α_i.
- If $q_i^2 \leq s_i$ and $\alpha_i < \infty$, delete ϕ_i from the model and set $\alpha_i = \infty$.

$\boldsymbol{\alpha}_i$ (when $q_i^2 > s_i$) and the noise level σ will be updated as following:

$$\alpha_i = \frac{s_i^2}{q_i^2 - s_i} \quad \text{and} \quad \sigma^2 = \frac{||t - \hat{t}||^2}{n - m + \sum_m \alpha_m \boldsymbol{\Sigma}_{mm}}.$$

The algorithm re-computes $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ and follows the process until a convergent condition is met. The relevant state and action pairs can be traced back by the $\boldsymbol{\phi}$ left in the model. The target value for new data $\mathbf{x}$ is predicted as

$$\hat{t} = k(\mathbf{x}, \mathbf{X})\mathbf{w}, \tag{3.9}$$

where $\mathbf{X}$ are the snapthots, and $\mathbf{w}$ are the weights along with the snapshots.

In related Bayesian models, this quantity is known as the marginal likelihood. Its maximization is known as the type-II maximum likelihood method [82]. The marginal likelihood is also referred as the **evidence for the parameters** [83], and its maximization as the **evidence procedure**. Lee [84] combined sparse Bayesian learning into reinforcement learning problem, called sparse Bayesian reinforcement leanring (SBRL).

CHAPTER 4: OBTAINING SNAPSHOTS IN DEEP REINFORCEMENT LEARNING

This chapter is reused from Deep Reinforcement Learning Monitor for Snapshot Recording [85] with permission from IEEE.

In this chapter, a novel method to obtain snapshots is proposed to explain a deep reinforcement learning agent throughout its learning process. To successfully obtain the snapshots, sparse Bayesian learning method [81] is utilized to re-approximate the Q-value function estimation of the reinforcement learning agent. Fig. 6.1 shows the overall structure of my method, the deep reinforcement learning monitor (DRL-Monitor), which consists of two parts. The first part is the deep reinforcement learning for learning and controlling the agent. The second part is the novel *monitor* for recording the most significant moments/snapshots. DRL-Monitor is applied to double deep Q-network (DQN) [1] in a visualized maze problem (Fig. 4.2) and two other Atari games (Fig. 4.3 and Fig. 4.4) to show the efficacy of my method.

4.1 Motivation

Understanding and interpreting neural networks are active and important research area to handle various applications that require confidence or trust of a model. There has been a number of techniques proposed to understand what neural networks have learned. Interpretable learning of the complex model affects to selection of an efficient and effective network architectures, identification and mitigation of bias, accounting for the context of problems, and improvement of generalization and performance [86]. Especially in decision making problems, understanding what action is taken and why the action is taken at a certain time is beneficial to interpret knowledge for transfer learning, to improve learning performance, and to fix potential errors in the critical domain as in autonomous car and

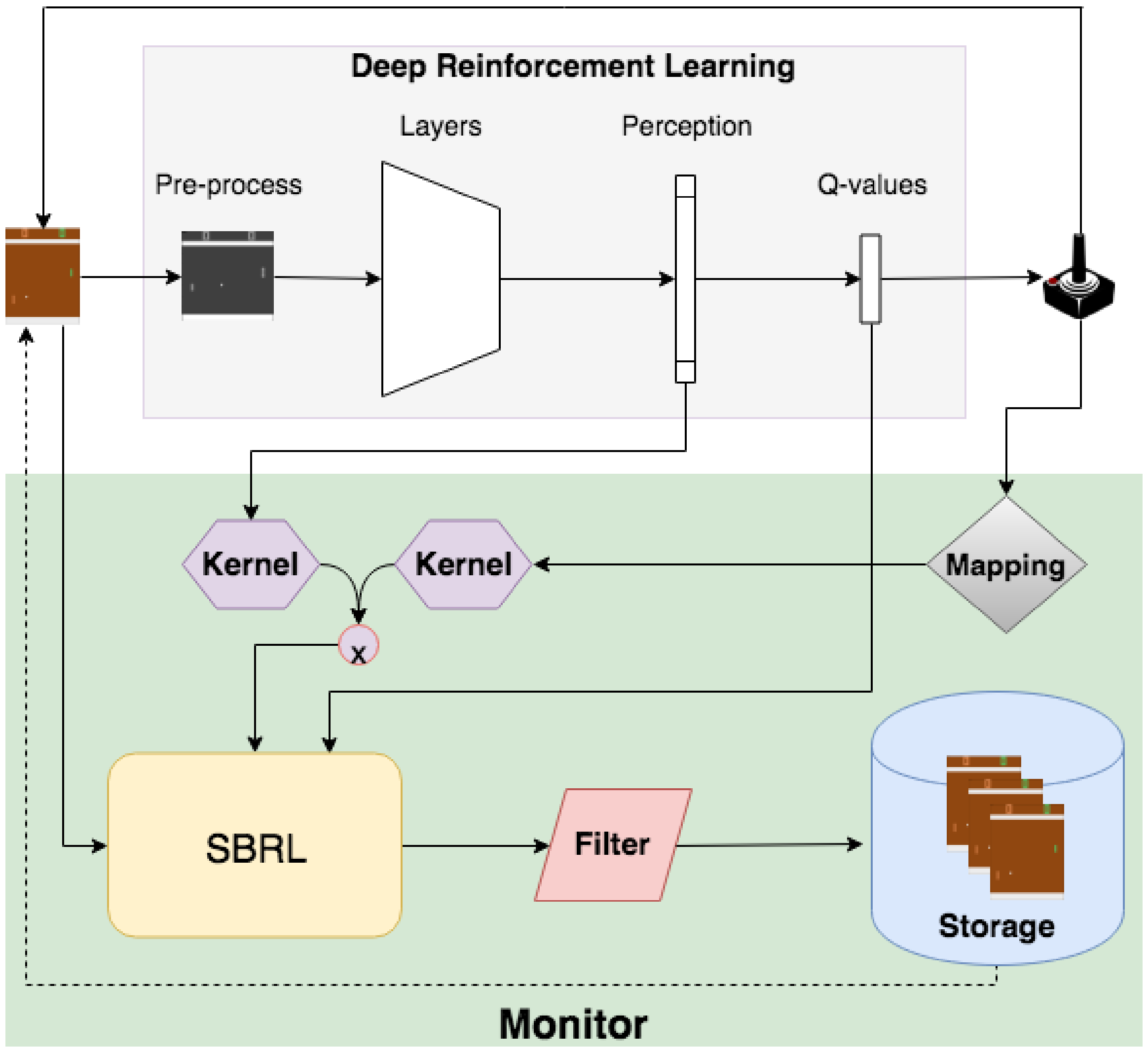


Figure 4.1: The diagram for the DRL-Monitor framework. Here, the Double DQN [1] is illustrated for the deep reinforcement learning module. The monitor can possibly be connected to many different DRL algorithms.

medical applications.

Previous publications [87, 88, 89, 90, 86] have attempted to visualize the neural networks to understand the learned model for supervised learning problems. Not many have attempted to visualize reinforcement learning problems yet but recent work by Zahavy et al. [91] and Greydanus et al. [60]. Zahany et al. apply Semi Aggregate Markov Decision Process and t-SNE to visualize strategy of a DQN agent with an Atari game. Greydanus et al. visualize the A3C policies through a salient map for Atari game agents. By observing the changes in the policy learned by the agent in input images, the method make it possible to observe the salient pixels that is highly relevant to the learned policy.

However, these are not sufficient enough to understand what and how deep reinforcement learning agent has learned. In real world, I (as human) recalls relevant past experiences when encounter a new problem. To mimic this psychological information processing process, the work focuses on memorizing significant moments through training the monitor to has an automated retention of snapshots with Bayesian model that accounts for what an agent is doing.

The benefits of the method are that a sparse set of snapshots is presented that takes less memory and is easy to interpret. The snapshots interpretation discovers unconsidered factors or features for learning process. The DRL-Monitor is complimentary to be combine with Graydanus, Zahany, or other visualization.

In this work, a new, novel method is applied to monitor deep reinforcement learning by memorizing important moments during training. The stored images help interpret what and how the agent builds up its knowledge and solves a reinforcement learning task. The method can be applied with many different state-of-the-art DRL algorithms to understand and evaluate their learning. By adopting a Bayesian model, the interpretation was able to bring additional insights.

4.2 Deep Reinforcement Learning Monitor

Since the input of the visualized maze and Atari games are images, DQN uses the neural network to transform state images into vector space (*perceptions*). The perception layer is a good feature vector representation of its complex input state (image). For the cases that actions are in a discrete space, independent action inputs are mapped into a real value vector so that distances between actions can be measured precisely. For example, left: [-1, 0], right: [1, 0], up: [0, 1], and down: [0, -1]. $\mathbf{s}$ is denoted as the perception of a state, a as the mapped chosen action given the state, and Q-value as the output of the neural network. sparse Bayesian learning method is applied to re-approximate Q-value with inputs $\mathbf{x} = (\mathbf{s}, a)$.

In order to record significant snapshots, the radial basis kernel function computes the similarity of samples' state-action pair using the transformed perceptions and the mapped actions. The monitor trains the sparse Bayesian reinforcement learning (SBRL) [84] module with the kernel features to predict Q-values of the selected DRL. For the training step, it augments input data with the snapshots stored in the snapshot storage. The kernel to train SBRL can be calculated as

$$\boldsymbol{\phi}_i = k_s(s_i, \mathbf{S}) \times k_a(a_i, A)$$

where k_s and k_a are two kernel functions, $\mathbf{S}$ and A are the snapshots.

The snapshots, which are retained from SBRL training, are passed through a filter. In this filter, it examines whether the SBRL training was successful. When SBRL training is evaluated as success, the collected snapshots become candidates for permanent recording in the snapshot storage. One of the possible heuristic rules can be defined as

$$\sqrt{\sum (\hat{Q}_i - Q_i)^2} < \tau \times \sqrt{\sum (\bar{Q} - Q_i)^2}$$

where Q is a target value, $\hat{Q}$ is a predicted value, $\bar{Q}$ is the average of the target Q, and $\tau \in [0, 1]$ is a control parameter. SBRL prediction needs to be better than a naive prediction of average value by a certain ratio of τ.

The remaining snapshots are moved to the storage. For additional analysis and explanation of learning, the weight distributions (means and variance) are stored for the corresponding snapshots. This information provides the measurements of how strong a snapshot affects the environment or policy development.

If a new snapshot does not exist in the storage, the monitor uses the newly achieved weights, and add the snapshot to storage. If a snapshot was already in the storage, the weight is updated using the following convex update rule,

$$\mathbf{w}_i^{ss} = (1-c) \times \mathbf{w}_i^{ss} + c \times \mathbf{w}_i^{new}$$

where $\mathbf{w}_i^{new}$ is the weight of the filtered candidate snapshot and $\mathbf{w}_i^{ss}$ is the weight of the same snapshot found in the storage. $c \in [0, 1]$ is a convex update parameter.

For the sparsity of the storage, how different stored snapshots and new candidates are measured by computing their similarity using the kernel function. When new candidates have a similar snapshot in the storage, the similarity is measured higher than a preset threshold value. Then, instead of adding new snapshots (indexed by j), the weight of the most similar snapshot (indexed by i) is updated with the similarity ratio:

$$\mathbf{w}_i^{ss} = (1-c) \times \mathbf{w}_i^{ss} + c \times (\sum_j k(\mathbf{x}_j, \mathbf{x}_i) \times \mathbf{w}_j^{new}).$$

4.3 Experiments

4.3.1 Experimentals Setup

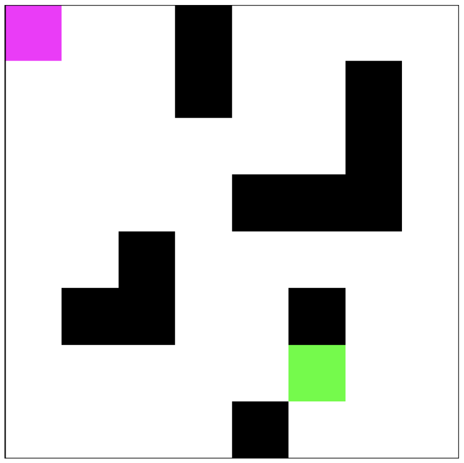


Figure 4.2: Maze Environment

Visual Maze is a navigation (8×8 blocks) task with RGB image representation as state ($80 \times 80 \times 3$). An agent starts at the pink block and moves toward to the goal location in the green block. The black blocks represent obstacles. The four discrete actions are defined as left, right, up, and down. The agent receives -1 reward for the cost of each movement. If the agent moves out of the boundary or hit the obstacles, it stays at the current location, and it receives -5 as a penalty. If the agent reaches the goal, the game terminates, and it receives $+30$ reward.

Pong (Fig. 4.3) is one of Atari game environments that Double DQN has played very well. An agent controls the green bar. There are six possible actions: stay still, start the game (stay still if game already started), up, down, fire up (move up faster), fire down (move down faster). The agent gets $+1$ reward if the ball (white dot) passes brown bar to the left, and -1 reward if the ball passes green bar to the right. The game terminates when either the bot or the agent reaches 21 rewards (or points).

In MsPacman (Fig. 4.4), an agent controls yellow Pacman. The agent will have three lives. There are 9 different actions: stay still, left, right, up, down, up right, up left, down

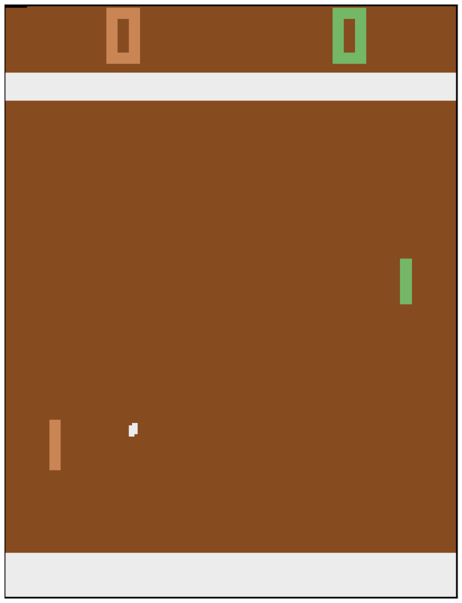


Figure 4.3: OpenAI Gym Pong Environment

right, and down left. If the agent eats one dot, the agent gets $+10$ reward. The agent loses one life if ghost catches it, and it is reseted to the starting position. The game terminates when the agent runs out of its lives.

4.3.2 Visual Maze

Fig. 4.5 illustrates the gradual snapshot acquisition as the agent learns. The bottom figure shows a total reward of each episode and above three figures show the activated snapshots, whose weights are not close to 0's, in the snapshot storage. The collected active snapshots are relevant to the perceptions and the policy at that time. There are negative snapshots at first and in the middle of training process. When the training converges, I observe that there are only positive active snapshots left because the majority of samples in the experience replay buffer contains positive ones as it exploits the learned policy more to reduce the steps to reach the goal.

Fig. 4.6 shows the maximum Q values in the maze and some of the active snapshot samples. When they lie on the path to the goal, and as getting closer to the goal, they end up with getting high Q values. In this path, the weights of those snapshots become high. I also observe that the weights are also affected by the exploration. Since the training always

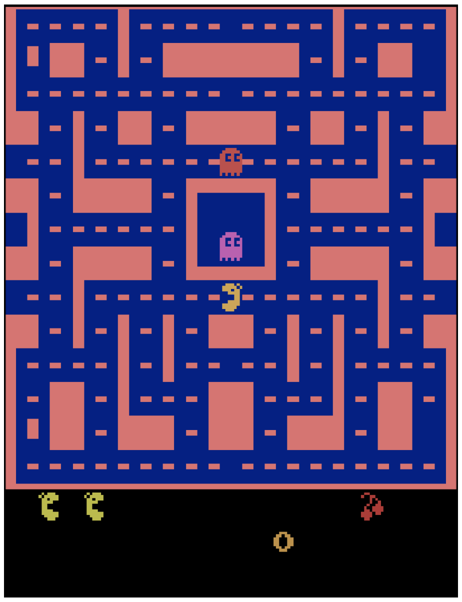


Figure 4.4: OpenAI Gym MsPacman Environment

starts at the top left of the maze, there is less exploration made starting from the right side of the maze. This results in low weight values to those snapshots.

Once Double DQN achieves the knowledge from training, it exploits the learned policy as shown in Fig. 4.7. With two different starting positions, one with the same start Fig. 4.7a and the other with a different start location Fig. 4.7b from training, the three most relevant snapshots are presented, which is computed by the kernel function. In Fig. 4.7a, the agent has developed an optimal policy that makes a right decision at each moment (at the junction) and the snapshots has captured them. On the other hand, when starting from the start location with lack of experience, it came up with a sub-optimal path. Snapshots on the right show when and what were the wrong decisions.

The snapshots explain the learned rule of what action an agent takes in a given state. The snapshots also provide information of which neural networks was able to learn. For example, there might be a dangerous zone that agent wants to take a sub-optimal solution. When comparing a sample with the snapshots, the similarity between the sample and the snapshots reveal a relationship between their Q-values. If the sample is closed to a snapshot which resulted a low Q-value, the sample is in a dangerous state and needs to take a sub-

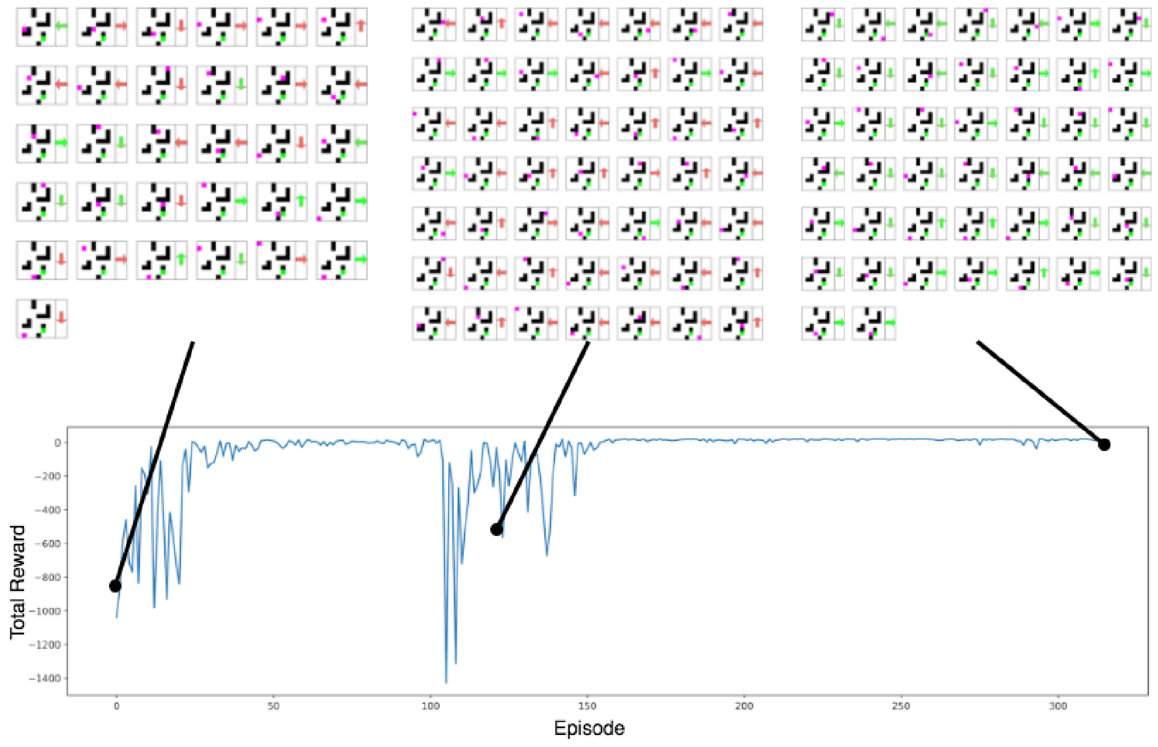


Figure 4.5: DRL-Monitor gradually adds snapshots to a snapshot storage during learning. The upper images show the content of snapshot storage at each moment. Red actions (arrows) mean negative weights of the snapshots, and green actions mean positive weights.

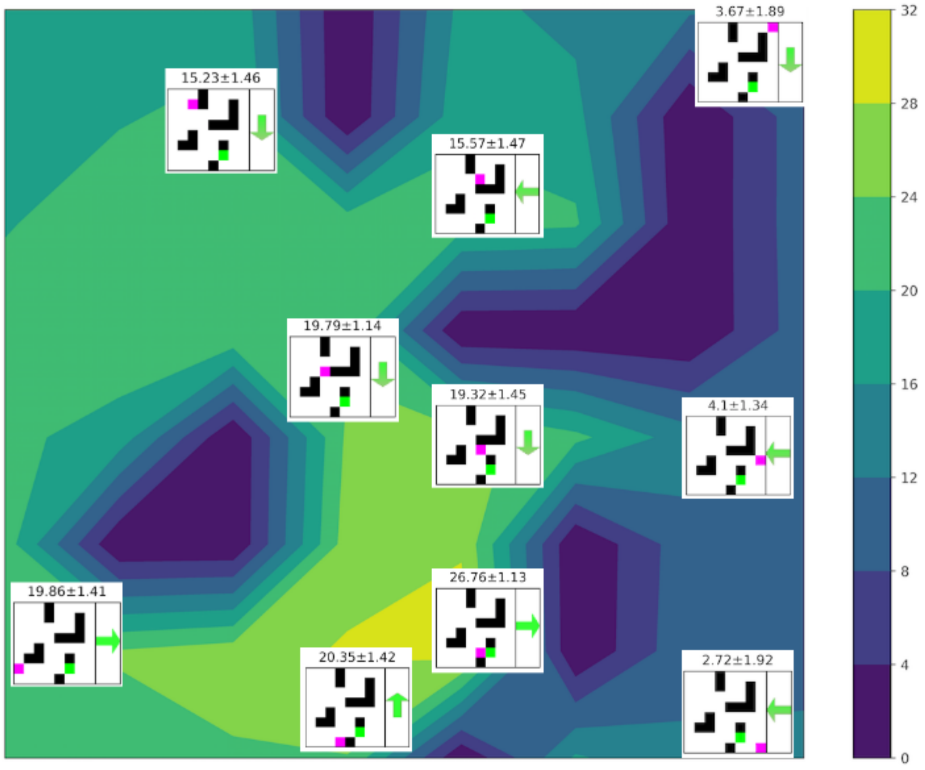
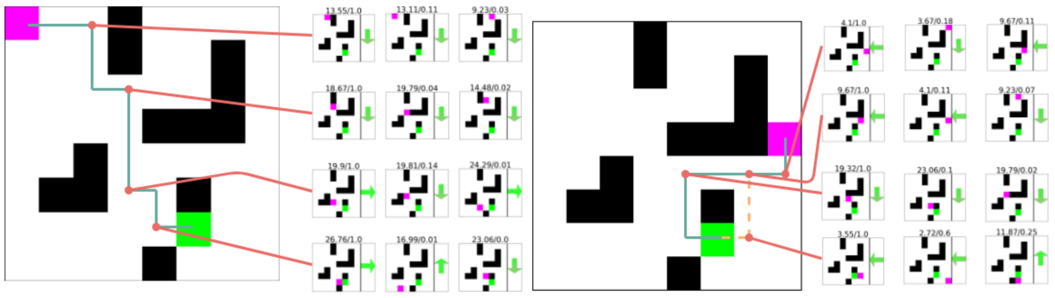


Figure 4.6: Q contour plot after fully trained with snapshots. The numbers present for weight distribution.



(a) Optimal path selection (b) Sub-optimal path selection

Figure 4.7: Exploitation of learned policy with different starting position: small images on the right are the snapshots for each circled moment. The snapshots are ordered from left to right showing its effects, and left has the highest effect. The numbers above each snapshot are weights and similarity.

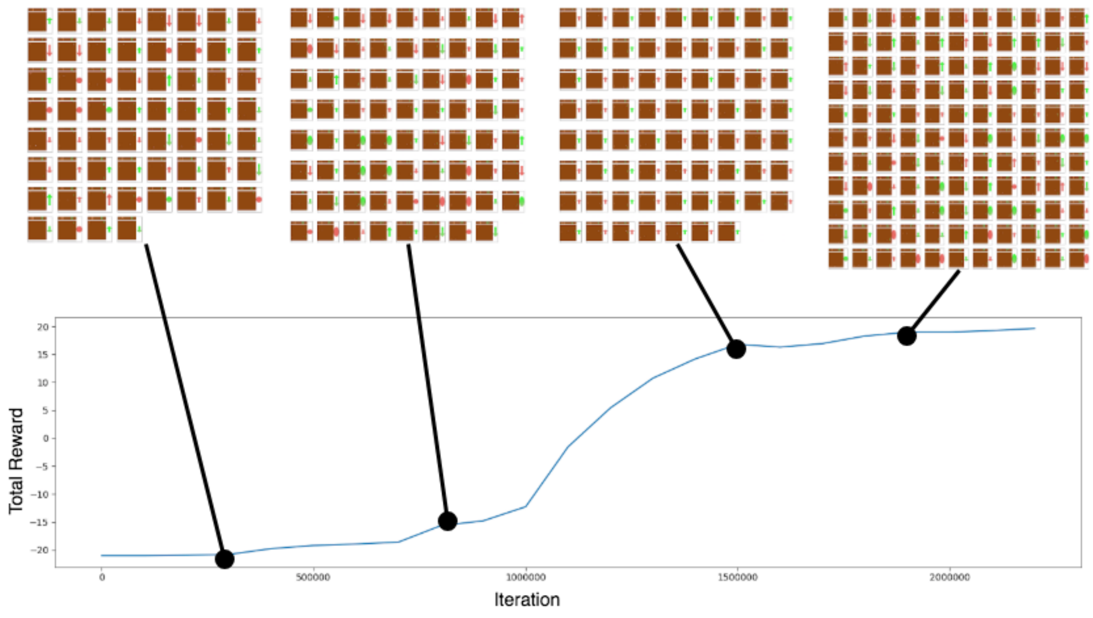


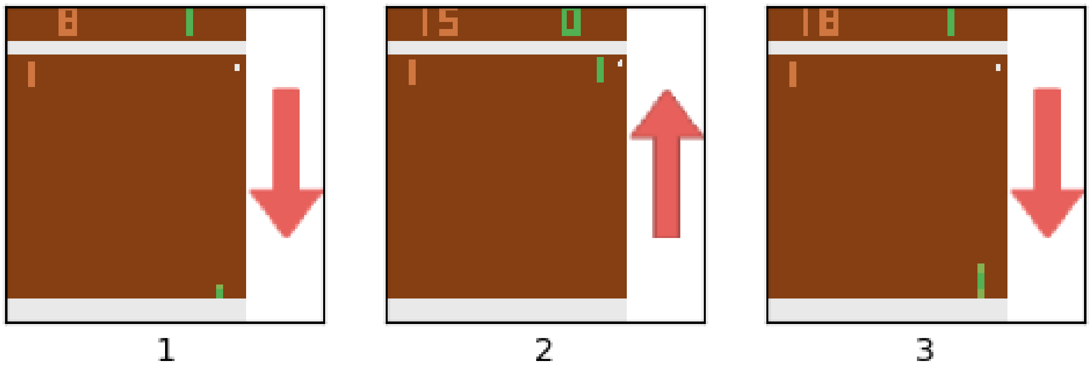
Figure 4.8: DRL-Monitor on Pong with reward curve and active snapshots. Red actions mean negative weights of the snapshots, and green actions mean positive weights of the snapshots. Longer actions indicate a faster action toward a direction.

optimal solution instead.

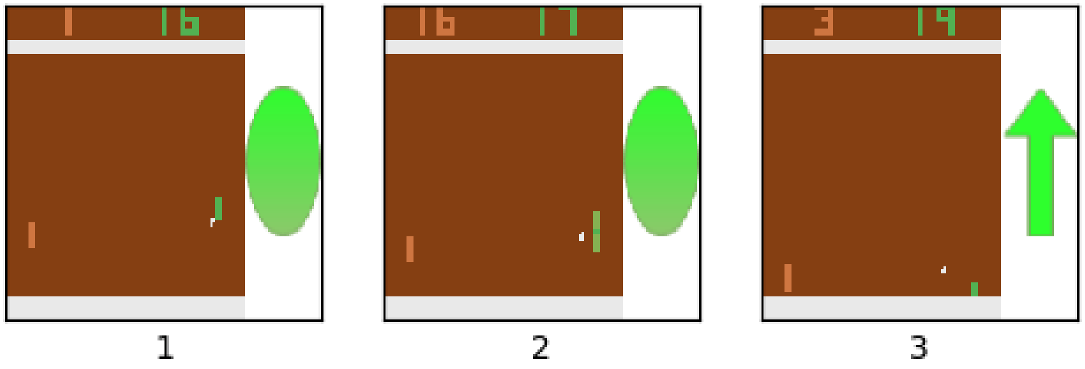
4.3.3 Pong

Fig. 4.8 shows active snapshots at 4 different training iteration (300K, 800K, 1.5M, 1.9M iterations). The bottom plot represents total rewards evaluated every 100K iterations. I observe that the slightly growing number of active snapshots thanks to the sparse control parameters in kernel, a filter heuristic, and snapshot storage. In our experiment, the total number of snapshots at the end of the training is 764 snapshots, and 221 of them are active.

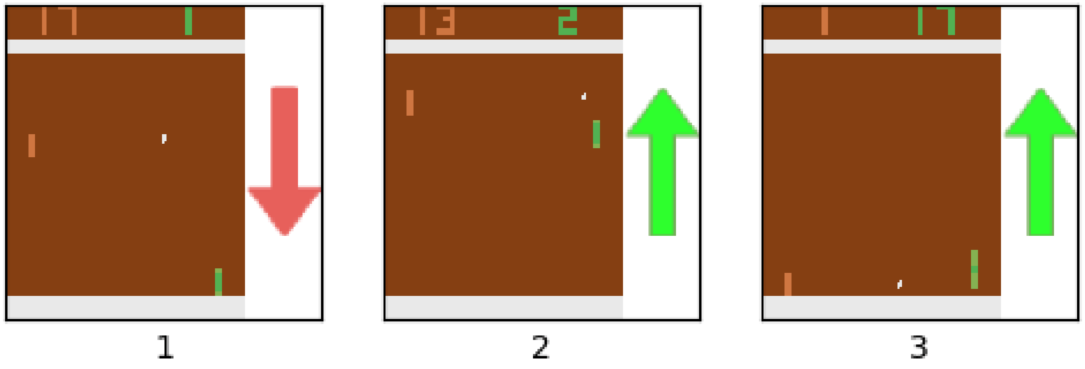
Fig. 4.9 presents the three most effective snapshots that has the high absolute weight values at 300K, 1.5M, and 1.9M iteration. At 300K iteration, agent remembers key snapshots that lead directly to a negative reward signal because Double DQN has not been successfully train the agent. I can also see lack of training by looking at the reward curve in Fig. 4.8. As the training progresses, the agent learns how to catch the ball to avoid negative reward shown in Fig. 4.9b. However, the agent only knows how to correctly behave when the ball is close. When the agent starts to converge at 1.9M iteration, agent reuses more



(a) Pong most effective snapshots at 300K iteration



(b) Pong most effective snapshots at 1.5M iteration



(c) Pong most effective snapshots at 1.9M iteration

Figure 4.9: The three most effective snapshots each timestamps by the weights.

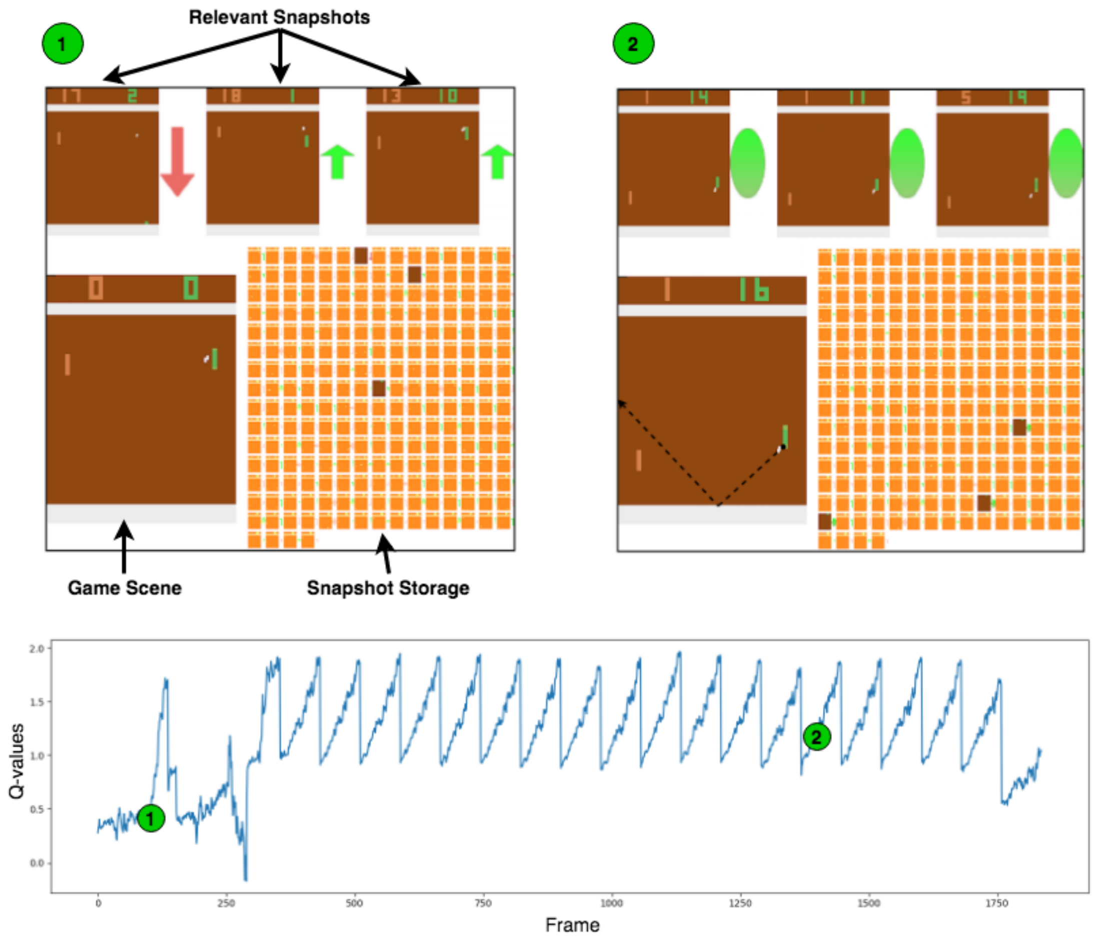


Figure 4.10: Visualization of learned Q-values on Pong. The video is available at: `https://youtu.be/Si4SvglUjzk`.

past experiences stored in snapshot storage to mark down a harmful state-action pair as well as to master how to move when the ball is far away.

In Fig. 4.10, exploitation of the learned policy in a Pong game is visualized. Here, the Q-values at each frame of the video game screen are shown. On the top, it shows how the agent exploits the winning policy with a game scene, active snapshots in the storage, and three most relevant snapshots.

The three most relevant snapshots are selected by kernel similarity between the game scene and active snapshot image. Since the agent uses the same strategy repeatedly to defeat its opponent, the point 2 in the figure has three most significant (similar) moments to send the ball through the winning trajectory shown in the dashed arrow line. There are

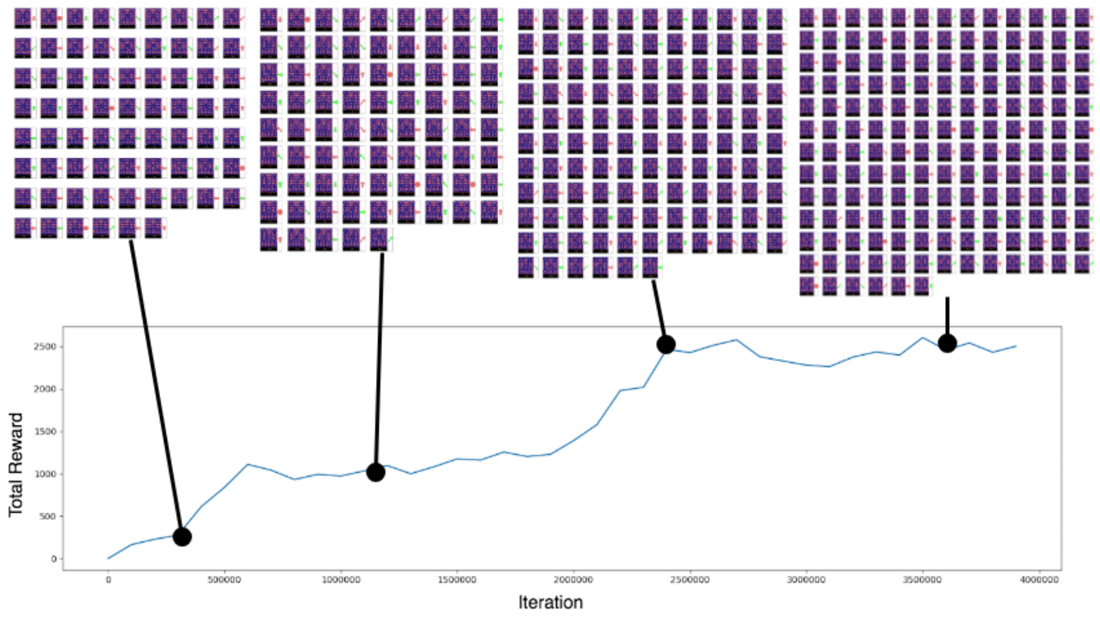


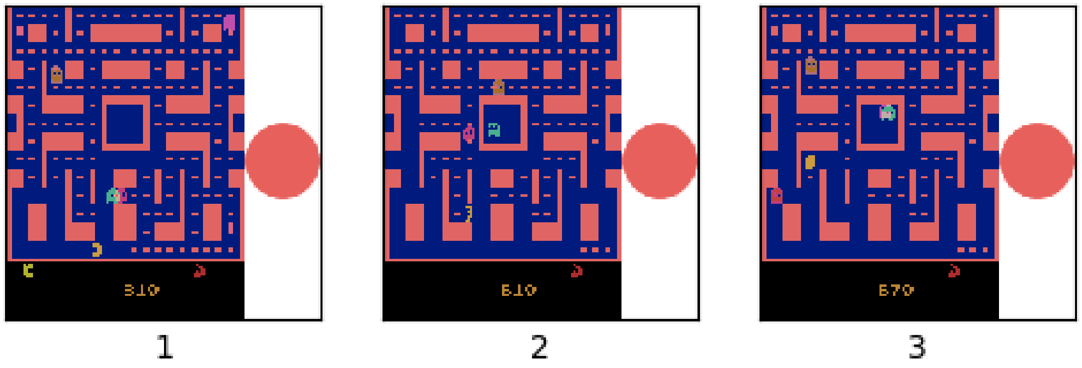
Figure 4.11: DRL-Monitor on MsPacman with reward curve and active snapshots of 4 different timestamps. Red actions mean negative weights of the snapshots, and green actions mean positive weights of the snapshots

three similar relevant snapshots at point 2 indicating a strong action selected such that the ball hit the edge of the agent to go back down and defeat its opponent.

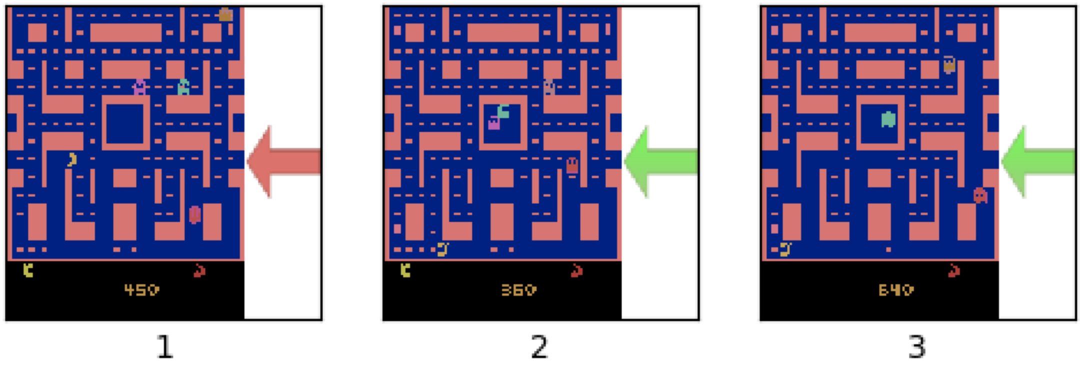
4.3.4 MsPacman

The gradually snapshot collection is shown as the agent learns with active snapshots at 4 different iterations (300K, 1.2M, 2.4M, and 3.6M) in Fig. 4.11 along with the reward curve. Comparing to Pong, there are more active snapshots are gathered at each time. I observe the total number of snapshots is 993, and the active snapshots at the end is 236.

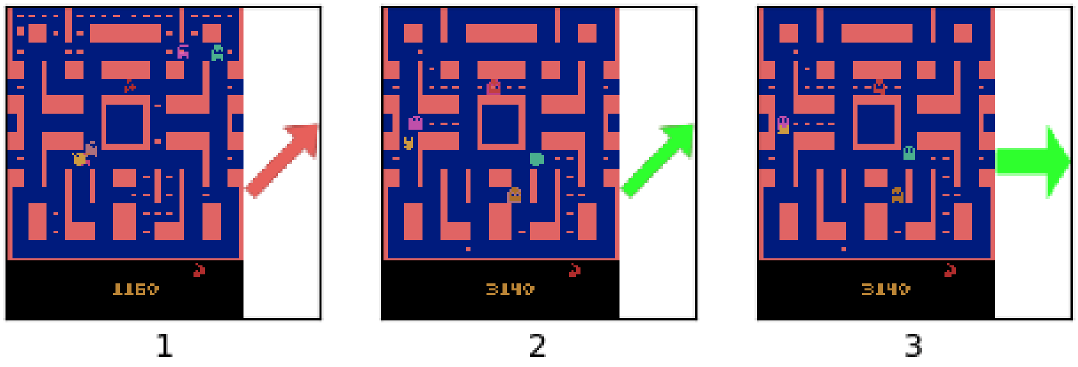
As in Pong, the three most effective snapshots at 300K, 1.2M, and 2.4M iterations are shown by comparing the absolute values of the weights (Fig. 4.12). At the beginning of the training, the snapshots are favor of staying still action. The agent understands that staying still is not a good action. In fact, the agent should always move in MsPacman game to be successfully survive. In the middle of the training when the agent is developing its policy, the agent learns the importance of the food which give positive immediate rewards. Moreover, the agent develops better perception of the foods as well as its own location than



(a) MsPacman most effective snapshots at 300K iteration



(b) MsPacman most effective snapshots at 1.2M iteration



(c) MsPacman most effective snapshots at 2.4M iteration

Figure 4.12: MsPacman most effective snapshots each timestamps by the weights. Red actions mean negative weights of the snapshots, and green actions mean positive weights of the snapshots.

previous iterations. When the Double DQN starts to converge, I observe that the agent avoids the ghost shown in Fig. 4.12c.1 by a negative weight of the snapshot.

However, Fig. 4.12c.2 and Fig. 4.12c.3 show that agent is heading towards to the ghost with positive weight snapshots. From the snapshot images, I observe that the agent is heading to the foods although the ghost is nearby. The agent overly weighted on eating foods to be safe keeping its lives. This is the key reason that Double DQN cannot reach human-level in MsPacman game. This might happen because of a sample bias or imbalance between positive and negative reward samples in the replay buffer so that the agent is not able to develop an optimal policy.

CHAPTER 5: ENHANCING DEEP REINFORCEMENT LEARNING WITH SNAPSHOTS

Previous chapter proposed to develop a method to extract snapshots and maintain the sparsity. The usage of the snapshots is only focused on interpretation. This chapter explores a different benefit of the snapshots to enhance deep reinforcement learning process by a simple tagging process. This chapter is reused from Relevant Experiences in Replay Buffer [92] with permission from IEEE.

5.1 Motivation

Experience replay [93, 94] stores past experiences in memory and replays some of them by randomly sampling from a uniform distribution. This breaks the direct correlation in training data and simulates independent distribution for gradient descent updates. Simply storing all experiences also prevents possible forgetting problems. Experience replay also improves the data efficiency [95], which suits many different RL algorithms. Deep Q-Network (DQN) [96] has shown the stability of the training process by using experience replay.

After the success of DQN, a fairly large amount of memory seems to be necessary for the experience replay (replay buffer) [96, 97, 98]. Without any justification of the size of the replay buffer, a lot of followup research adopts the heuristically determined size for the memory. However, storing all the experiences and then sampling some from uniform distributions is not the most effective way to use a replay buffer. Thus, it requires investigation on the design of a replay memory: *storing* problem of which experiences to store and *replaying* problem of which one to replay [99].

Recent research focuses on the *replaying* problem to improve the efficiency of experience replay: most notably, Prioritized Experience Replay (PER) [99], Hindsight Experience Replay (HER) [98], and Distributed Prioritized Experience Replay (DPER) [100]. PER modifies the traditional experience replay whereby instead of uniformly choosing experiences from replay buffer, the agent is more likely to sample experiences with higher temporal difference error. HER realizes sample augmentation in order to overcome the rare relevant (reaching goal trajectories) experiences problem with virtual goals and shows much improved performance on learning. For an environment that multiple agents learn asynchronously, DPER adopts PER in distributed learning framework for efficient learning. These approaches make advances on efficient *replaying*, eventual effective learning, but the *storing* problem is not so far addressed.

In this chapter, a simple, but novel method is proposed, *Relevant Experience Replay* (RER), as an initial step to solving the *storing* problem. RER adopts DRL-Monitor [85] to identify important experiences to tag them as "relevant." DRL-Monitor, attached to a DRL agent, learns a policy developed by the agent by observing the learning process and collecting important moments to understand the rationale of the agent's decision making. When DRL-Monitor tags some samples as relevant or important, a replay buffer contains both the relevant samples and irrelevant samples. DRL samples from this mixture with a certain ratio and examines the quality of relevance tagging for learning. This requires a slight modification on the sampling module of the DRL. The experiments show the efficacy and efficiency of the use of relevant samples identified by DRL-Monitor in 11 different Atari game environments.

5.2 Relevant Experience Replay

My previous works on obtaining snapshots leverages DRL-Monitor to systematically select important experiences to tag as relevant experiences for replay. Samples from the mixture of the tagged "relevant" experiences are replayed with other irrelevant experiences for efficient training as shown in Fig. 5.1.

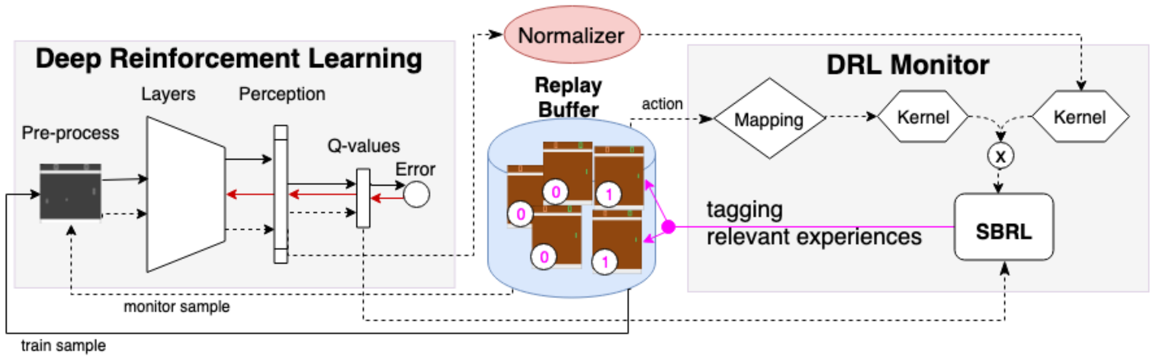


Figure 5.1: Relevant experience replay framework with DRL-Monitor. DRL-Monitor identifies and tags relevant experiences in a replay buffer. With a sampling ratio paramemter, a certain ratio of relevant experiences are guaranteed to be replayed for DRL training.

5.2.1 Training DRL-Monitor

DRL-Monitor answers to the question of what to store during replaying experiences. DRL-Monitor observes a DRL agent's learning progress in a training environment and its application of a learned policy in test environments. From training, the agent tunes the neural network weights to produce Q values for appropriate actions to take. Encoded state representation is developed through multiple neural network layers (i.e. convolutional and fully connected layers). I refer this encoded state vector from the fully connected layer before the output layer as *perception*. Observation of DRL enables DRL-Monitor to recognize important samples from remembered frame images, actions, and rewards.

When the monitor considers all data samples as relevant experiences, the model will be identical to traditional experience replay. This can be avoided by discarding similar samples. To prevent memorizing any possible similar experience, DRL-Monitor measures the similarity by using kernel function. For simplicity, our initial model defines a product kernel of Radial Basis Function (RBF) kernels to measure the similarity of both perceptions and mapped actions as:

$$\phi_i = k_s(s_i, \mathbf{S}) \times k_a(a_i, A).$$

The product kernel is also used for training DRL-Monitor to construct sparse bases to approximate the DRL $\mathbf{Q}$ function space [85]. The bases built in the monitor are the key data samples capable of estimating any representation that a DRL agent can make. Thus, the kernel bases are treated as the relevant experience to replay for efficient learning.

5.2.2 Tagging Relevant Experiences

The data in the replay buffer is assumed to be sequential presentation similar to taking state after state in the simulation. A single experience sample e_t in a replay memory can be represented as $e_t = (s_t, a_t, r_t, d_t, f_t)$ where the boolean terminal state indicator d_t tells whether the state s_t is terminal or not, and a boolean tagging flag f_t tells the relevance. Then, the experience replay buffer $M = [e_1, \dots, e_{N_m}]$ where N_m is the number of stored experiences.

f_t is initialized as zero for every new experience comes into the replay buffer. After finishing DRL-Monitor training, $\boldsymbol{\alpha}$, the set of hyper-parameters controlling the strength of the prior over the corresponding weights, tells the significance samples among the constructed bases. Corresponding f_t's are marked as 1:

$$f_t = \begin{cases} 1 & \text{if } \alpha_t < \infty \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

With this implementation, all relevant experiences are tagged with value $f_t = 1$, and other irrelevant experiences are tagged with value $f_t = 0$ in the replay buffer.

5.2.3 Replaying Experiences

For informed random sampling with relevance tagging, a training batch is sampled which concatenates a portion from a relevant experience set with another portion from the other irrelevant experience set. A batch size b and a ratio $c \in [0, 1]$ are pre-defined for a mixture of the two types of experiences. The ratio $c = 1$ replays the relevant experiences only and $c = 0$ replays the irrelevant ones only.

The relevant experience set $R = \{e_t | e_t \in M, f_t = 1\}$ is retrieved from the memory using the tagging f_t, and the irrelevant set $I = \{e_t | e_t \in M, f_t = 0\}$. Concatenating the two sets, a batch B draws a total of b samples from the memory to train the DRL agent:

$$B = [e_t^{(r)}, e_t^{(i)}]_{e_t^{(r)} \sim R, e_t^{(i)} \sim I}$$

where $|e_t^{(r)}| = c * b$ and $|e_t^{(i)}| = (1 - c) * b$.

With this strategy, a certain proportion of relevant experiences are enforced to be re-played in every training step. The process of monitoring and training DRL agent with RER is summarized in Algorithm 1.

5.2.4 Normalize Perception

The perception is output of an activation function, Parametric Rectifier Linear Units (PReLU) [101] in this case. Therefore, the distribution of the perception on each dimension is not consistent for kernel similarity measurement. Therefore, a standard normalization on each dimension of the perception is incrementally applied in order to stabilize the similarity computation.

The perception is normalized with information from all previously seen perceptions but without storing actual perceptions. The normalization method accumulates the total number (N_p), the sum ($\boldsymbol{p_{sum}}$), and the sum of squared ($\boldsymbol{p^2_{sum}}$) of all the perceptions observed at time t and a set of current N_t perceptions p_t as:

$$\begin{cases} N_p = N_p + N_t \\ \mathbf{p}_{sum} = \mathbf{p}_{sum} + \sum_{p \in \mathbf{p}_t} p \\ \mathbf{p}^2_{sum} = \mathbf{p}^2_{sum} + \sum_{p \in \mathbf{p}_t} p^2. \end{cases}$$

The mean and standard deviation are computed as:

$$\boldsymbol{\mu} = \frac{\mathbf{p}_{\boldsymbol{sum}}}{N_p} \quad \text{and} \quad \boldsymbol{\sigma} = \sqrt{\frac{\mathbf{p}^{\mathbf{2}}_{\boldsymbol{sum}} - 2 * \mathbf{p}_{\boldsymbol{sum}} * \boldsymbol{\mu}}{N_p} + \boldsymbol{\mu}^2}.$$

With the accumulated normalization, the perception $\mathbf{p}$ is standardized to $\tilde{\mathbf{p}}$ as:

$$\tilde{\mathbf{p}} = \frac{\mathbf{p} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}. \tag{5.2}$$

Algorithm 1 Relevant Experience Replay (RER)

```
1:  **Input**: batch size $b$, training period $K_{train}$, monitoring period $K_{monitor}$, total step $T$,
    relevant replay ratio $c \in [0, 1]$.
2:  Initialize a replay buffer memory $M$
3:  Observe an initial state $s_0$
4:  **for** $t = 0$ **to** $T - 1$ **do**
5:      Get action $a_t$ from $s_t$ through policy $\pi(s_t)$
6:      Execute $a_t$ and collect reward $r_t$
7:      Initialize $f_t = 0$ and check terminal $d_t$
8:      Store experience $e_t = (s_t, a_t, r_t, d_t, f_t)$ in $M$
9:      **if** $t \mod K_{monitor} = 0$ **then**
10:         Retrieve $K_{monitor}$ previous experiences from $M$
11:         Extract $\mathbf{p}$, $a$, and $Q$-values
12:         Normalize $\mathbf{p}$ using Eq.(5.2) and map $a$
13:         Monitor the DRL agent
14:         Modify tagging flag $f$ with Eq.(5.1)
15:     **end if**
16:     **if** $t \mod K_{train} = 0$ **then**
17:         Empty the batch memory $B$
18:         $R \leftarrow$ relevant experiences with $f_t = 1$
19:         $I \leftarrow$ irrelevant experiences with $f_t = 0$
20:         Sample $c \times b$ of $e_t^{(r)}$ from $R$ and add to $B$
21:         Sample $(1 - c) \times b$ of $e_t^{(i)}$ from $I$ and add to $B$
22:         Perform one step gradient update with $B$
23:     **end if**
24: **end for**
```

5.3 Experiments

5.3.1 Experiment Setup

The quality of the relevant experience replay with one of the state-of-the-art DRL algorithms, Double DQN [1], is examined. All tests were run with the same training configuration for all the Atari game environments. The performance of RER with Double DQN is compared against two baselines, vanilla Double DQN that uses traditional experience replay and the one that uses prioritized experience replay.

The architecture of Double DQN is slightly modified. The activation function to PReLU for both baselines and RER experiments is replaced instead of ReLU. For computational efficiency, 256 hidden units in the last hidden layer are used.

The discount factor γ is set to 0.99, and the learning rate is set to 10^{-4}. The number of steps between target network updates is $10,000$. The loss for DRL training is optimized with Adam optimizer [78]. All gradients are clipped between a range of $[-10, 10]$ to prevent gradient explosion problem when one gradient is too high or low.

The agent evaluates policy every 100K simulation steps and reports the average of the total reward from 30 episodes with random no-op actions from $1-30$ at the beginning of the game. By given random starting states, the agent has more robustness and generalization as the agent cannot rely on repeating a single memorized trajectory. The same random no-op action strategy is applied to the training phase as well. The mini-batch of $b = 64$ training samples are collected every $K_{train} = 4$ steps. The agent runs $T = 12$ million simulation steps.

The monitor is trained every $K_{monitor} = 256$ steps to ensure the monitor is up-to-date with the current policy. The gamma of perception and action for RBF kernel are set to $\gamma_p = 0.35$ and $\gamma_a = 1.0$ correspondingly.

To ensure rich exploration, ϵ-greedy is used with linearly decreasing ϵ from 1 to 0.1 for 1M simulation steps, from 0.1 to 0.01 for the next 5M simulation steps, and constantly 0.01 after that.

Randomly selected 11 different Atari game environments are tested: Boxing, Breakout, Double Dunk, Enduro, Fishing Derby, Freeway, Ice Hockey, MsPacman, Pong, Robotank, and Tennis. Fig. 5.2 shows the 11 games in the order from left to right and top to bottom.

The environments are in NoFrameskip-v4 version from OpenAI Gym [102], which is similar environment done in [96]. A similar environmental setup to [96] is used except that the terminal state gets the reward of -1 instead of 0 if the reward was initially 0, which

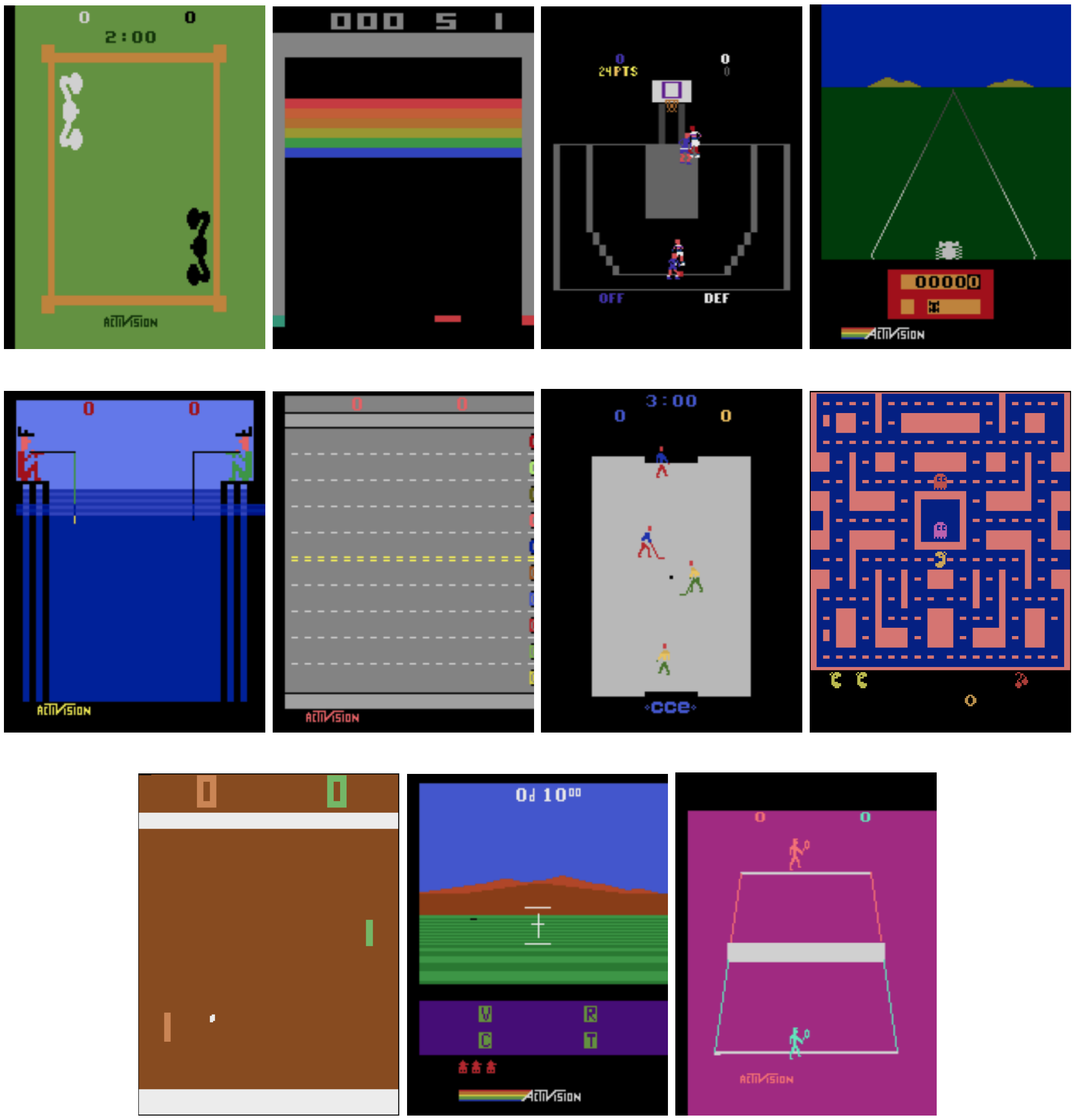


Figure 5.2: 11 games visualization.

indicates the agent lost of life.

Due to high memory usage and computation time for experience replay, a buffer with the size $N_m = 10^4$ is used for traditional experience replay, PER and RER to perform comparison. This will simulate the cases of how experience replay affects learning performance in complex problems that require large replay memory. The replay ratio c is set at 0.5 for balancing relevant and irrelevant experiences.

5.3.2 Empirical Results

Fig. 5.3 shows the raw average total rewards over 12 million steps of training time. In the presented nine Atari game experiments, relevant replay buffer shows improved performance, comparing to the traditional replay buffer and PER. Replaying with 50% ($c = 0.5$) of relevant experiences achieves faster learning speed (shorter time to reach the optimum) and higher asymptote (better solution with higher scores).

The RER does not immediately show an improvement in the early stage of the training because the agent needs to build up correct perception mapping through the convolutional layers. When the perception has been well established, the DRL-Monitor is capable of tagging the relevant experiences correctly to improve the agent's learning.

The efficiency of RER is established after the perception is well-formed. The process of formation typically takes 400K steps. The result in Fig. 5.3 shows a better improvement after 400K steps of RER compared to traditional experience replay and PER. The speed of learning curve is also improved with RER to reach a higher score.

Boxing, Breakout, Enduro, Ice Hockey, and Tennis show the improvements in term of learning speed after a well-established perception in Fig. 5.3. In Freeway and Pong, the problems are too simple to achieve further improvement in learning speed, but the RER does not slow down learning and achieves the equivalent learning speed and performance. In other environments, RER performs at least equivalent to or slightly better than traditional experience replay and PER due to the hardness of the problem with 10^4 buffer size.

In Fig. 5.4, the trained agent is directly compared with relevant experience replay with

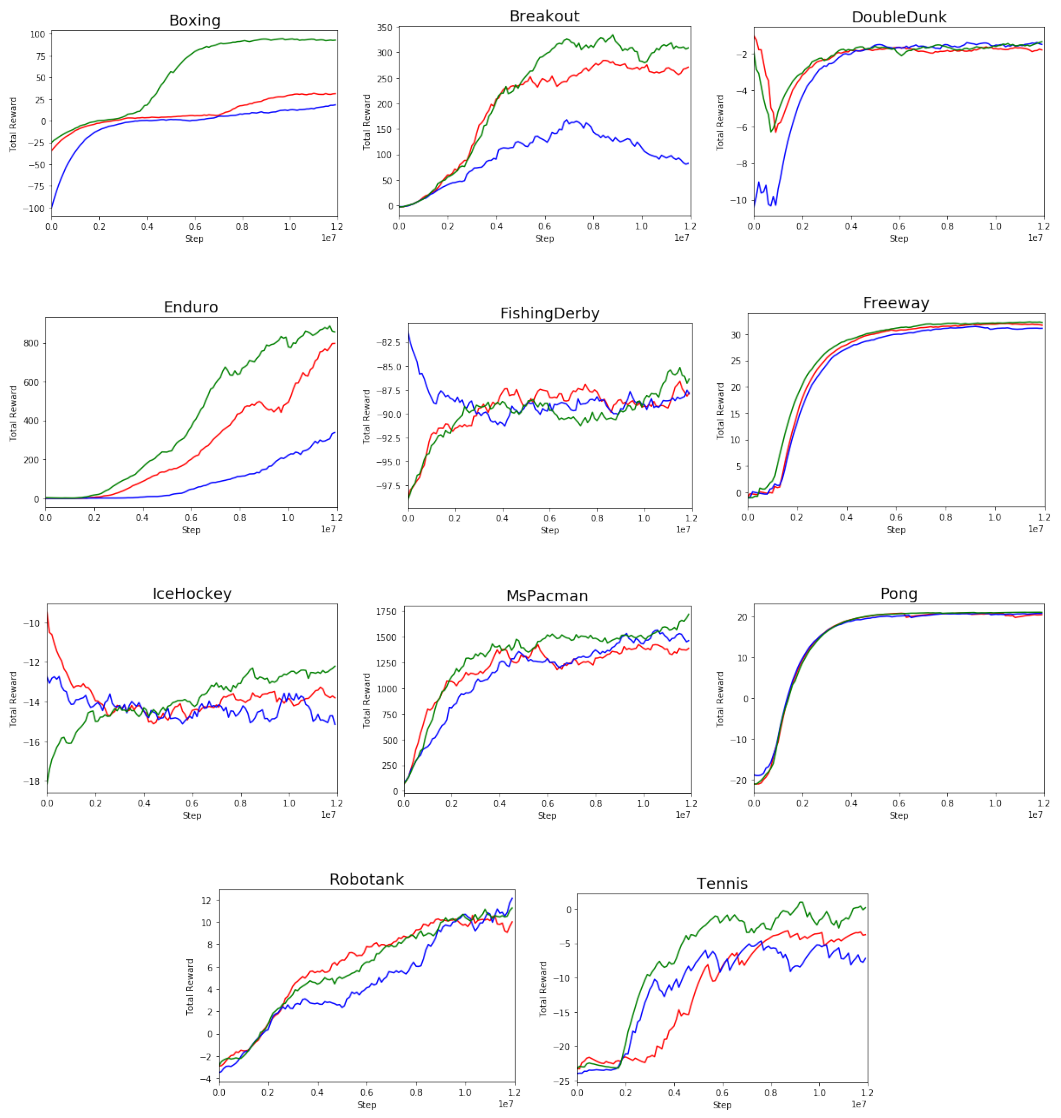


Figure 5.3: Learning curves (average total reward of 30 episodes) for Double DQN with traditional experience replay in red, prioritized experience replay in blue, and relevant experience replay in green. Each curve corresponds to a single training run 12 million simulation steps.

the two other baselines to see how much the proposed method improves quantitatively. The normalized improvement score of RER to traditional experience replay is computed comparing them with base human score as in [96]:

$$score_r = \frac{score_{relevant} - score_{human}}{score_{traditional} - score_{human}}.$$

This score tells how much better performance can be achieved by replacing traditional experience replay with relevant experience replay.

The traditional experience replay with PER is measured using a similar metric formula. However, if the PER score is less than a human score, the formula is reversed and apply absolute value in the case of different sign when the human score is in between traditional experience replay and PER:

$$score_p = \left| \frac{score_{traditional} - score_{human}}{|score_{prioritized} - score_{human}|} \right|.$$

I observed a significant percentage improvement by applying relevant experience replay with an average increased of 47% across 11 Atari games that have been evaluated. Notably, RER achieves higher improvements especially when the complexity of an environment grows. Fig. 5.4 depicts the comparison of the normalized scores in each Atari game with three comparison benchmarks after 12M simulation steps.

I observed a significant improvement in games where the immediate reward has both directions (negative and positive). They are shown with Boxing, Robotank, and Tennis environments. Pong and Freeway achieved maximum long term reward (thus the optimal policy), so they did not discover any better solution (or policy).

One directional immediate reward (only positive) also showed a noticeable improvement in Breakout, Enduro, and MsPacman. As I observe in Pong, exposure to too many irrelevant experiences results in instability of learning with traditional experience replay. The instability of Pong is shown by the red curve not always gets the maximum score 21.

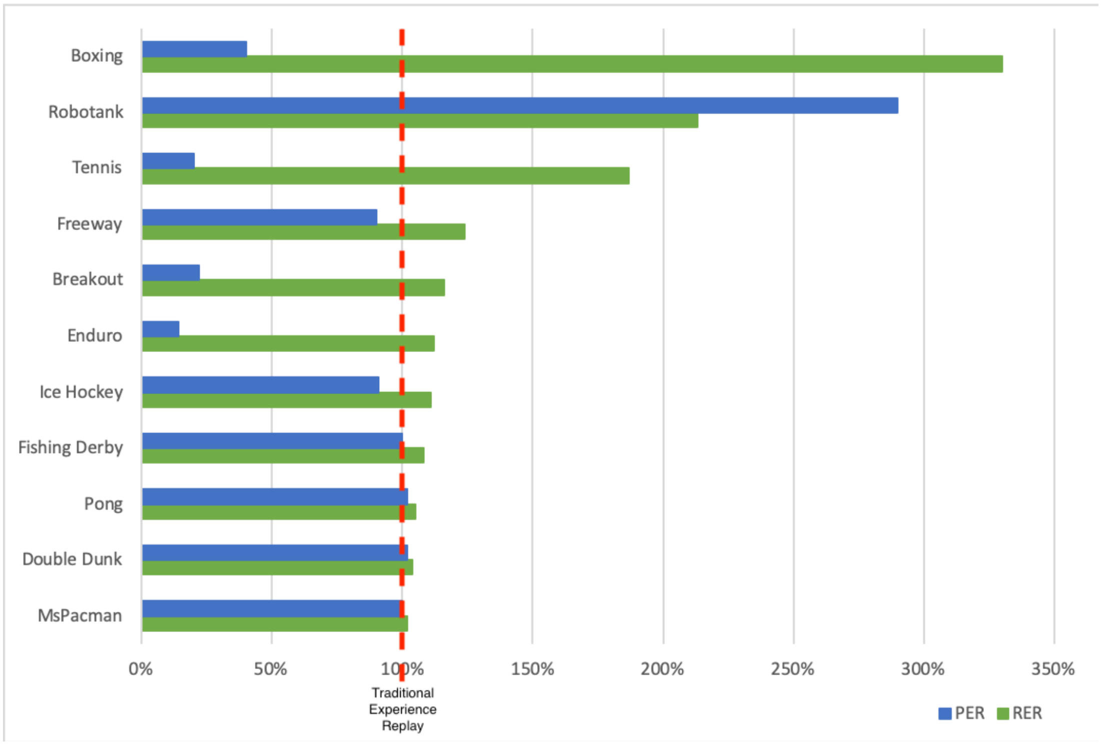


Figure 5.4: Normalized improvement score comparison between RER, PER, and traditional experience replay in 11 different Atari game environments.

Double Dunk and Fishing Derby are hard games for our agent with the environment setup of 10^4 buffer size. Therefore, RER was able to learn a slightly better solution than the one with traditional experience replay.

I also observed poor performance with the prioritized experience replay when the size of the replay buffer is limited. The performance decrements happen because of the gradient clipping. The probability of an experience to be drawn in PER depends on the temporal difference error. Therefore, when the gradient is clipped, the temporal difference error of a high error experience changes slowly. That makes the probability of the high error experience reduces slower. The buffer size of 10^4 is also not an ideal buffer size for PER. Since PER depends too much on the temporal difference error, small buffer size gets important experiences being pushed out faster. These two reasons make PER sometimes performs worse than traditional experience replay. Especially the latter reason reveals the limitation of PER when the replay memory size is limited or when the environmental changes or complexity requires larger replay memory.

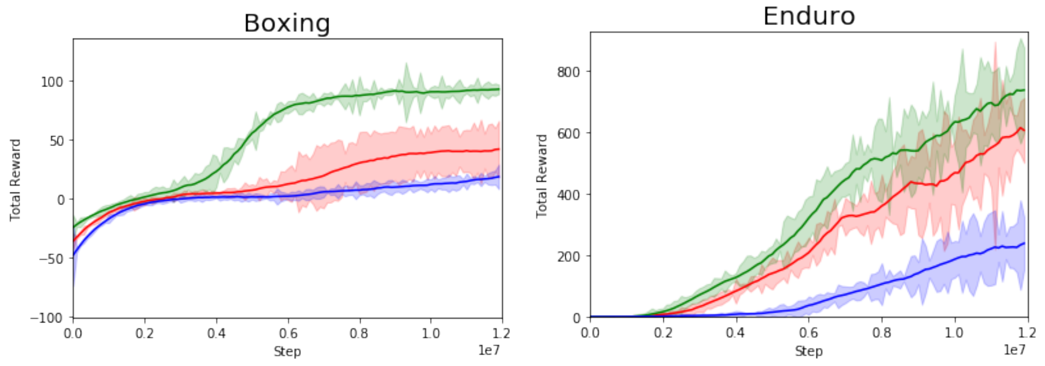


Figure 5.5: Average and variance of 5 experiments on Boxing and Enduro environments. The color scheme is same as in Fig. 5.3 (traditional experience replay in red, prioritized experience replay in blue, and relevant experience replay in green).

Fig. 5.5 show that mean and variance of learning performance in Boxing and Enduro environments for 5 runs. Because of limited computation resources, only these two environment are able to fully run. Small repetitions, however, show the similar trend to these results and previous results.

As discussed before, I were able to observe the significantly improved asymptotic performance $p << 0.05$ from one-way ANOVA (p are 8.4×10^{-5} and 0.0012 in Boxing and Enduro respectly) in the two environments. Also, when prioritizing the experience (in green and blue curves), I can observe improved stability of learning learning while prioritized experience can be misguided when enough memory is not provided.

5.4 Discussion

Experience replay has been one of the major components in reinforcement learning. There has been a number of techniques proposed to improve the data efficiency and stability of learning for the experience replay. The process of determining relevant experiences is very important for storing problem in experience replay to mimic the psychological information processing from a human that remembers only relevant experiences when tackling a problem. In this work, a simple, but novel method is introduced, called Relevant Experience Replay, that leverages DRL-Monitor to identify relevant experiences. The proposed approach examines the efficacy of DRL-Monitor as a tool for the decision-making problem of which experiences to store for experience replay. The empirical results verify the proposed approach. That is, DRL-Monitor is capable of identifying significant experiences to construct memory-efficient and effective experience replay model.

CHAPTER 6: ENFORCING SPARSITY OF SNAPSHOTS FOR EFFICIENCY AND INTERPRETATION

Chapter 4 suggested a novel method in retrieving snapshots for interpretation. However, the end storage contains a lot of samples needed to be looked at by human. In this chapter, an approach to significantly scale down the number of snapshots in the storage is proposed. This brings a faster interpreting and understanding of DRL agent's behavior and requires less human effort. This chapter is reused from Learning Sparse Evidence-Driven Interpretation to Understand Deep Reinforcement Learning Agents [103] with permission from IEEE.

6.1 Motivation

Understanding what has been learned from neural networks has become a major problem in machine learning research. Self-interpretable system, representation analysis, and re-approximation are three major approaches to interpret deep learning [7]. The self-interpretable system are built by using attention mechanism, disentanglement learning, and adversarial examples. Attention mechanism attempts to understand the relationship between input and output [8]. Disentanglement learning tries to understand high level concepts from low level information [9]. Adversarial examples are used for interpreting the vulnerability of the learning system [10]. Representations are analyzed in layer and individual neurons, vector grouping, and saliency map. Visualization of the layer and individual neurons are helpful to understand which features have been learned [11]. Vector analysis is used for reducing high dimensional space into 2D or 3D which is easier for a computer to visualize [12, 13]. Saliency map reveals significant information that affects the model decision [14]. Reapproximation applies inherently interpretable models to approximate deep neural networks to produce interpretations of neural networks. This category include linear approximation [15], decision trees [16], and rule extraction [17].

The diverse interpretations of the aforementioned approaches, however, are made from individual samples or a lot of different visualizations. The sparsity of interpretations is one of the main challenges that needs to be tackled to lessen human arduous effort in interpreting neural networks due to a large amount of data as well as computation units. Most interpretation of reinforcement learning are done with reapproximation with decision trees and visualization [104] and they are hard to avoid the inherent sparsity challenge. A decision tree can get very complicated to understand when the state space is large. Visualization with saliency map, region sensitive, and counterfactual states requires tremendous human effort to analyze the interpretation. Therefore, a method is needed that can reduce the human effort to efficiently understand reinforcement learning agent behavior.

Deep reinforcement learning monitor (DRL-Monitor) [85] utilizes sparse Bayesian reinforcement learning [84] to understand the behavior of a deep reinforcement learning agent. The method monitors behavior through collected important moments (snapshots). However, the method also suffers from a high number of snapshots that need to be examined at the end of the process for explanation (almost 1000 snapshots depending on the experiment). The high number of snapshots due to the basis is mapped by state and action. Slightly different state can have different action, this makes the basis of the two consecutive states become different.

In this chapter, a novel method is proposed to drastically reduce the number of snapshots while minimizing the loss of important information for easier explaining of the agent's behaviors. The suggested approach is consist of two main sparsity modules: value re-estimation and saliency module. The value re-estimation ignores action mapping so that the snapshot retrieval (or evidence collection) occurs only in state space and thus remove the common state samples with different actions taken. The saliency module produces a binary mask to sift highly attended area of snapshot images that are relevant to actions taken.

Using the action-driven salience map, the state (perception) space can be constructed for snapshot collection. Different images can be mapped to the same perception after masked by interested areas through the action-driven salience map when they share the same skill. Therefore, the suggested method can safely prevent the similar states to be present in snapshot storage, hence can reduce the number of snapshots. In general, the information flow for state and action is separated by two different modules instead of only one combined product kernel [85]. This allows effective reduction in the number of snapshots without losing too much important information. Our experimental results in Atari games clearly illustrate this claim.

6.2 Sparse Interpretable Reinforcement Learning

Fig. 6.1 illustrates the overall structure of the Sparse Interpretable Reinforcement Learning method that includes two major parts: policy gradient module for learning and control and the novel *sparse interpretable* module for a small number of significant moments. An agent starts at time $t = 0$ and perceives state $\boldsymbol{s}_t$ from an interactive environment. The agent decides action a_t from policy $\boldsymbol{\pi}_{a \in \mathcal{A}}(a|\boldsymbol{s})$. The environment returns next state $\boldsymbol{s}_{t+1}$ and a reward r_{t+1}. The agent continues the process until the environment terminate or a certain timestep T reached. The *sparse interpretable* evaluates trajectories in order to help explaining the behavior of the learned or learning agent.

6.2.1 Policy Gradient Method

Proximal Policy Optimization [105] is used as our main policy gradient method to optimize the black-box model. A neural network function $f_{\boldsymbol{\phi}}$ maps complex state input (sequence of images) $\boldsymbol{s} \in \mathbb{R}^d$ into a low level representation (latent space) $\boldsymbol{z} \in \mathbb{R}^k$ such that:

$$\boldsymbol{z} = f_{\boldsymbol{\theta}}(\boldsymbol{s}).$$

A linear transformation (more parameters add to $\boldsymbol{\theta}$) then takes the representation as input and outputs a policy (action probability distribution) $\boldsymbol{\pi}(a|\boldsymbol{s}, \boldsymbol{\theta})$ and state value $V(\boldsymbol{s}, \boldsymbol{\theta}) \in \mathbb{R}$.

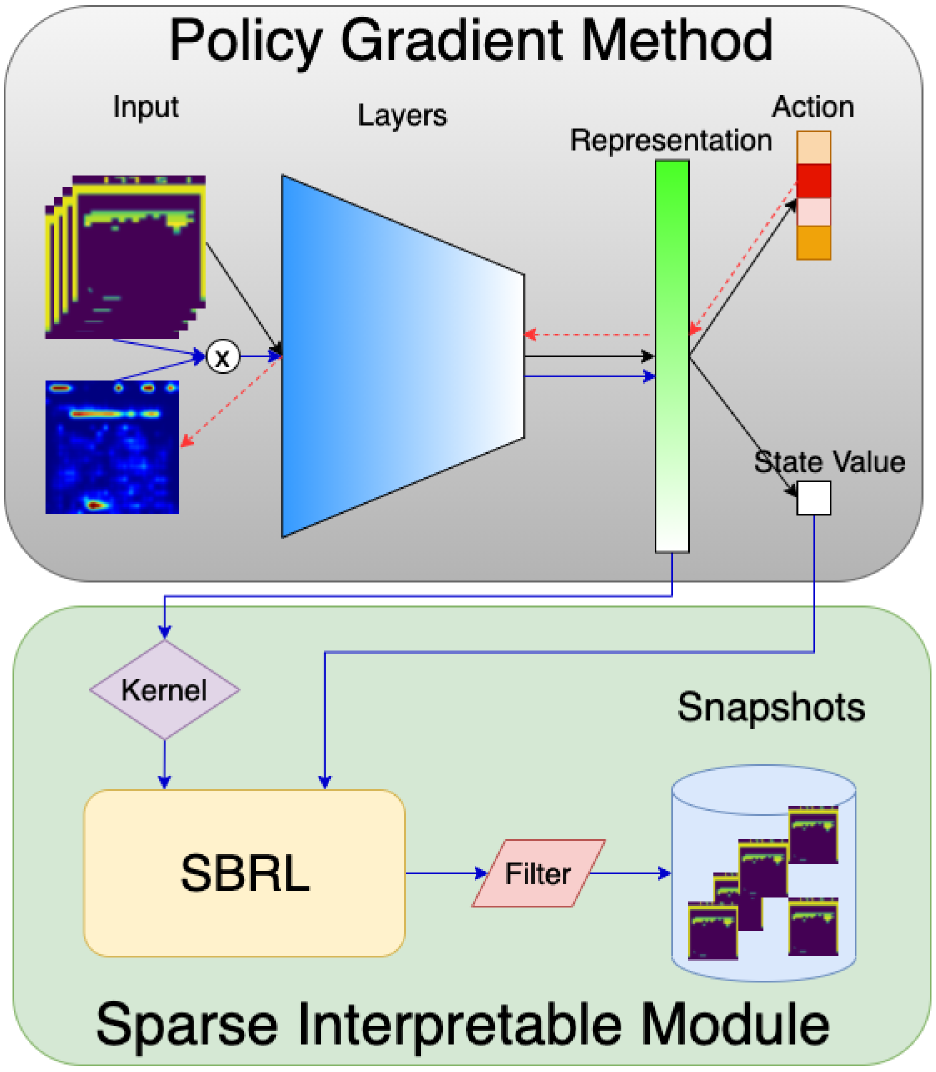


Figure 6.1: The diagram for Sparse Interpretable method to explain Reinforcement Learning agent. In this diagram, policy gradient method is illustrated for the Reinforcement Learning. This method can be replaced with any value approximation learning method for Reinforcement Learning.

The state value V estimates the total discounted reward from the start to the current time which: $V(\boldsymbol{s}_t) = r_t + \gamma V(\boldsymbol{s}_{t+1})$ where $\gamma \in [0, 1]$ is the discount factor. Advantage $\hat{A}_t$ is computed for every timestep from $[0, T]$ such that:

$$\hat{A}_t = -V(\boldsymbol{s}_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-1} V(\boldsymbol{s}_T).$$

An objective $J(\boldsymbol{\phi})$ is computed as:

$$J(\boldsymbol{\theta}) = \sum_{t=0}^{T} \Big(\alpha (r_t + \gamma V(\boldsymbol{s}_{t+1}) - V(\boldsymbol{s}_t))^2 - \beta \hat{A}_t \log \boldsymbol{\pi}(a_t | \boldsymbol{s}_t) \Big)$$

as a close value prediction and optimize the action in favor of higher advantage is needed. Parameters $\boldsymbol{\phi}$ is updated using gradient descent method.

Policy gradient method is used because it estimates state-value and has less input space to explore without actions. DRL-Monitor with the product kernel of state and action kernels map the bases to unnecessarily large search space to result in many similar snapshots. For this reason, DRL-Monitor [85] suffers from more or less 1000 snapshots for each experiment, which makes it hard to interpret.

6.2.2 Gradient of Chosen Action Policy Activation Mapping as Masking

The chosen action can be used for computing the gradient class activation map in order to see the areas in the state space that impacts the decision. Since the output of the first convolution layer retains relevant information from the state input as well as transformational information, the gradient of the first layer L^0 is taken. Noise is also added to the state input to smooth the gradient [59]. The output of the first layer is weighted by the gradient of the chosen action policy w.r.t L^0 such that:

$$\boldsymbol{r} = \frac{1}{N} \sum^{N} \frac{\partial \boldsymbol{\pi}(a_t | \boldsymbol{s}_t)}{\partial L^0(\boldsymbol{s}_t + \mathcal{N}(0, \lambda))}$$

The gradient output is the product of the weight and the output of the first layer: $\boldsymbol{g} = \boldsymbol{r} \times L^0(\boldsymbol{s}_t)$. The mask is computed by the product of the weight and the output of the first layer and binarized by a threshold to get a binary mask:

$$\boldsymbol{M} = \Big(\frac{\boldsymbol{g} - \min(\boldsymbol{g})}{\max(\boldsymbol{g}) - \min(\boldsymbol{g})} >= \tau\Big).$$

Fig. 6.2 shows an example of the smoothed gradient from a chosen action in Breakout game. The representation is computed with the masked state input such that

$$\boldsymbol{z} = f_{\boldsymbol{\theta}}(\boldsymbol{M} \times \boldsymbol{s}).$$

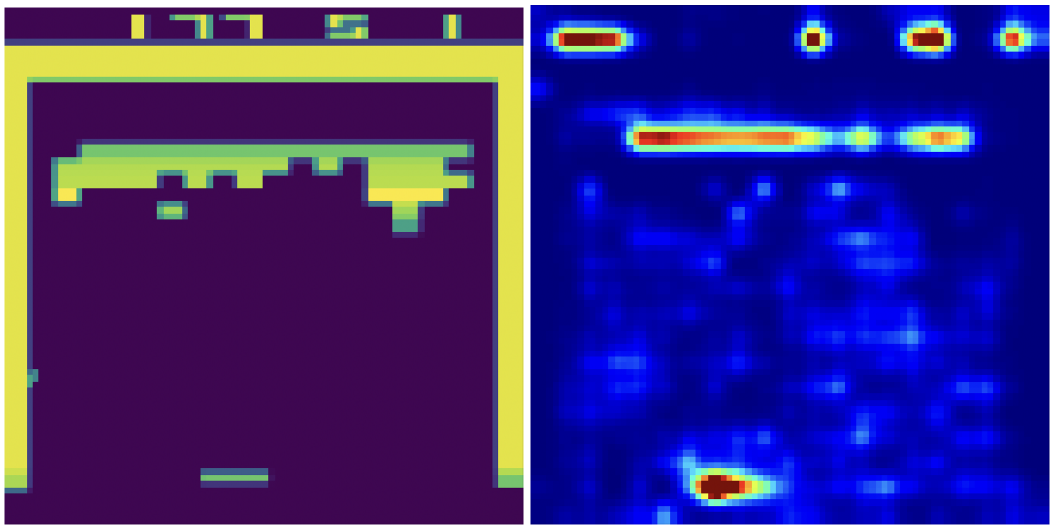


Figure 6.2: An example of smoothed gradient of a chosen action policy activation in Breakout game.

6.2.3 Sparse Interpretable Module

The relevance vector machine process [81] is followed that given state latent space and state-value pairs $(\boldsymbol{z}_i, V(\boldsymbol{z}_i))$. I assume the values are sampled from the model with additive noise such that

$$V(\boldsymbol{z}_i) = f_{\mathbf{w}}(\boldsymbol{z}_i) + \epsilon_i \quad \text{where } \epsilon \in \mathcal{N}(0, \sigma^2).$$

The initial variance is set as the variance of all values: $\sigma^2 = 0.1 \times var(\boldsymbol{V})$ where $\boldsymbol{V} = (V_1, V_2, \dots, V_N)$.

Basis function (kernel) is used for mapping the sample latent space $\boldsymbol{z}$ to different basis space, $\boldsymbol{\phi}_i(\boldsymbol{z}) = k(\boldsymbol{z}, \boldsymbol{z}_i)$. Hence, the value can be predicted as $V(\boldsymbol{z}_i) = \boldsymbol{\phi}_i(\boldsymbol{z})^\top \mathbf{w} + \epsilon_i$ where each $w \sim \mathcal{N}(\mu, \Sigma)$. Different type of kernels such aslinear kernel, radial basis function kernel, and polynomial kernel can be used. In general, a kernel matrix can be constructed as $\boldsymbol{\Phi} = [\boldsymbol{\phi}_1(\boldsymbol{z}), \boldsymbol{\phi}_2(\boldsymbol{z}), \dots, \boldsymbol{\phi}_n(\boldsymbol{z})]$, the $N \times (N+1)$ matrix representing inputs of the dataset.

At the beginning of the training, a random important moment (snapshot) at index m is used. A vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_M)$ is initialized as a set of *hyper-parameter* to control the strength of the prior over the corresponding weights $\mathbf{w}$. The starting values of $\boldsymbol{\alpha}$ are infinities numbers except for one at index m as

$$\alpha_m = \frac{||\boldsymbol{\phi}_m||^2}{||\boldsymbol{\phi}_m^\top \boldsymbol{V}||^2 / ||\boldsymbol{\phi}_m||^2 - \sigma^2}.$$

Given $\boldsymbol{\alpha}$, $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ can be computed as:

$$\boldsymbol{\Sigma} = (\boldsymbol{A} + \sigma^{-2}\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \quad \text{and} \quad \boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top \boldsymbol{V}$$

where $\boldsymbol{A} = \boldsymbol{\alpha}\boldsymbol{I}$.

The marginal likelihood of the state-values over the weights is represented by a zero-mean Gaussian distribution such that $\boldsymbol{V} \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{C})$ where the mean $\boldsymbol{m}$ and variance $\boldsymbol{C}$ are

$$\boldsymbol{m} = \boldsymbol{\Phi}\mathbf{w} \quad \text{and} \quad \boldsymbol{C} = \sigma^2 \boldsymbol{I} + \boldsymbol{\Phi}\boldsymbol{A}^{-1}\boldsymbol{\Phi}^\top.$$

The variance $\boldsymbol{C}_{-i}$ is $\boldsymbol{C}$ with the contribution of basis vector index i removed can be decomposed as $\boldsymbol{C}_{-i} = \boldsymbol{C} - \alpha_i \boldsymbol{\phi}_i \boldsymbol{\phi}_i^\top$. The sparsity factor (measures the extent of a basis with other bases) and quality factor (measures the alignment error when a basis is removed) can be computed of for every basis vector i as:

$$s_i = \phi_i^\top \boldsymbol{C}_{-i}^{-1} \phi_i \quad \text{and} \quad q_i = \phi_i^\top \boldsymbol{C}_{-i}^{-1} \boldsymbol{V}.$$

Only important bases are added into the model when the squared quality factor greater than its sparsity factor ($q_i^2 > s_i$) and re-estimate $\alpha_i = \frac{s_i^2}{q_i^2 - s_i}$. The important basis can also be removed in the model when its squared quality factor less than or equal its sparsity factor ($q_i^2 \leq s_i$) and set $\alpha_i = \infty$. These important bases are also called *snapshot*.

The variance of the noise ϵ is also updated as following:

$$\sigma^2 = \frac{||\boldsymbol{V} - \boldsymbol{\Phi}\boldsymbol{\mu}||^2}{N - M + \sum \boldsymbol{\alpha}\boldsymbol{\Sigma}}$$

with M is the number of important basis in the model at that training step.

$\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ are computed given the new $\boldsymbol{\alpha}$ and σ^2 and continue the process again until a convergence condition is met or a certain number of steps reached. The root mean squared error of state-value approximated by this module is used for comparing with the state-value given by the neural network as convergence condition such that $\sqrt{\frac{1}{N}\sum(\boldsymbol{V} - \boldsymbol{\Phi}\boldsymbol{\mu})^2} < \delta$ where δ is small number to trigger the convergence.

6.3 Experiments

6.3.1 Environments

The proposed approach is tested in Breakout and Pong environments (Fig. 6.3). In Breakout, an agent controls the blue bar at the bottom with four possible actions: stay still, start the game (stay still if the game already started), left, and right. The agent gets $+1$ reward if the ball hits and destroys the brick layers at the top. The game terminates when the agent runs out of 5 lives. In Pong environment, an agent controls the blue bar on the right against the bot on the left. There are six possible actions: stay still, start the game (stay still if the game already started), up, down, fire up (move up faster), fire down (move down faster).

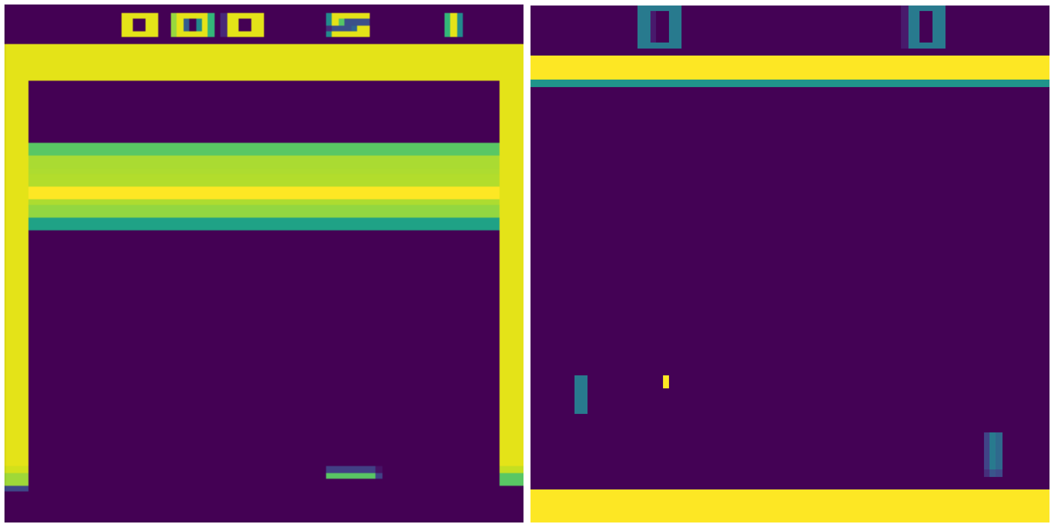


Figure 6.3: Test Breakout (left) and Pong (right) environments.

The agent gets $+1$ reward if the ball passes all the way to the left, and -1 reward if the ball passes the blue bar to the right. The game terminates when either the bot or the agent reaches 21 rewards (or points).

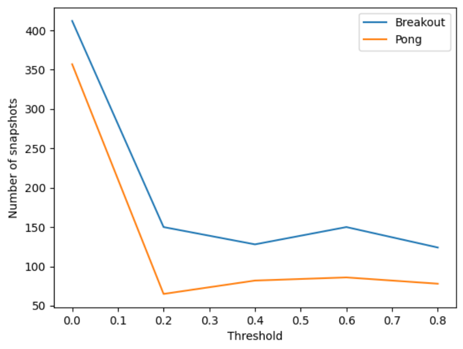
6.3.2 Experimental Setup

Across all of the run experiment, a single set of hyper-parameters as follows is used.

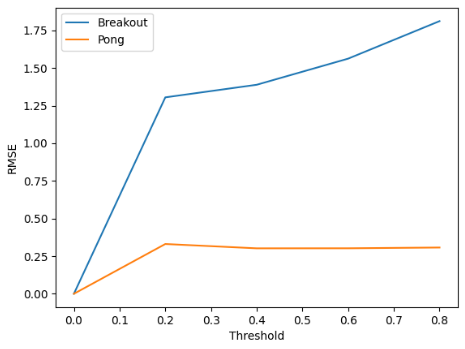
Deep reinforcement learning agent he following standard PPO settings are selected to train the agent. The discount factor γ is set at 0.99 to capture a long term accumulated reward. The neural network learning rate is set to be 0.0001 for a stable learning.

Saliency map The λ is set at 0.15 for a small noise to perturb the state to get smoothen gradient.

Sparse interpretable module In this work, linear kernel is use for mapping the sample latent space to basis space. δ is set at 0.1 for a good re-estimation of the state-value $\boldsymbol{V}$ for convergence condition. If the sparse interpretable module is not converged after 2000 steps, the algorithm stops and records the result.



(a) The number of snapshots



(b) RMSE

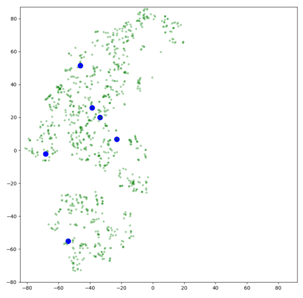
Figure 6.4: The sparsity and estimation accuracy for the varying threshold.

6.3.3 Experimental Results and Analysis

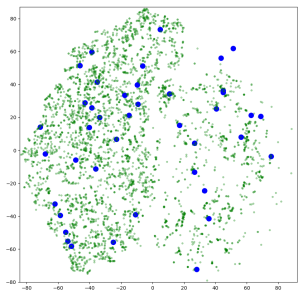
Fig. 6.4 shows the decreasing number of snapshots and increasing RMSEs when increasing the threshold τ. In Breakout, the suggested approach successfuly reduces the number of snapshots to [414, 152, 128, 150, 124] when the threshold is increased τ to [0.0, 0.2, 0.4, 0.6, 0.8] accordingly (Fig. 6.4a). The root mean squared error for the reapproximation of value function, however, increases [0.0, 1.30, 1.39, 1.56, 1.82] as shown in Fig. 6.4b due to the high similarity of state inputs after applying saliency map.

In Pong, 764 snapshots reported in [85] with the reapproximation of the state-action value (Q-value) are now drastically reduced. I also observe a further decreasing trend in term of number of snapshots [357, 65, 82, 86, 78] when increasing the threshold τ respectively [0.0, 0.2, 0.4, 0.6, 0.8]. The root mean squared error for Pong is lower than the Breakout game because the state-value in this game has a different distribution with smaller value scale. The changes also reflected in Fig. 6.4.

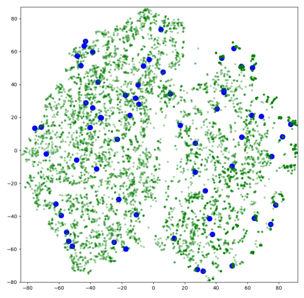
To better understand the sparsity learning process in Breakout, The states and snapshots are illustrated in 2D t-SNE space [13] in Fig. 6.5. The green dots represent the samples from different trajectories in the training process, and the blue dots are the snapshots collected. In the beginning, the agent's experience is limited to a small segment of the state space, thus the required snapshots to explain the experienced states are small. The number of



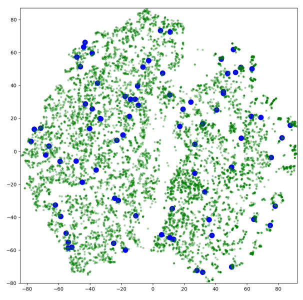
(a) Epoch 1000



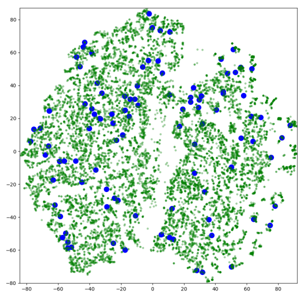
(b) Epoch 5000



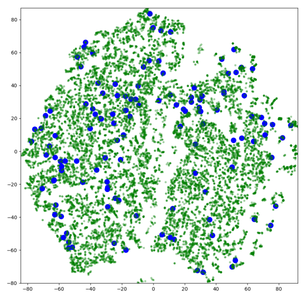
(c) Epoch 9000



(d) Epoch 12000

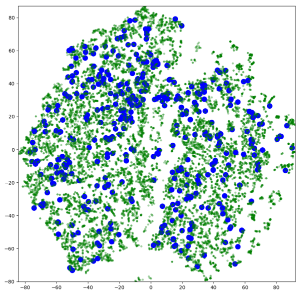


(e) Epoch 16000

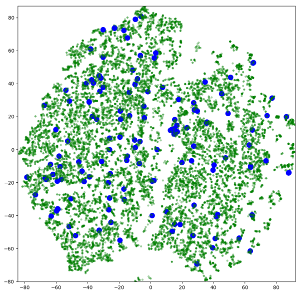


(f) Epoch 20000

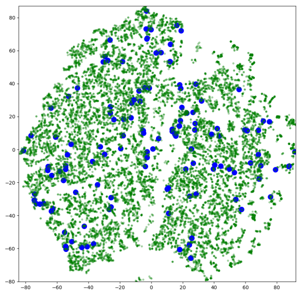
Figure 6.5: Evolution of the state samples in green and the snapshots in blue using $\tau = 0.8$ threshold to mask the gradient in the Breakout environment.



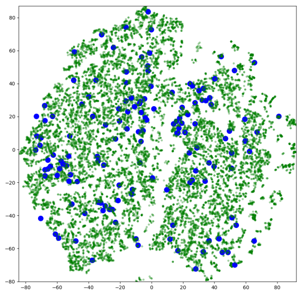
(a) $\tau = 0.0, M = 412$



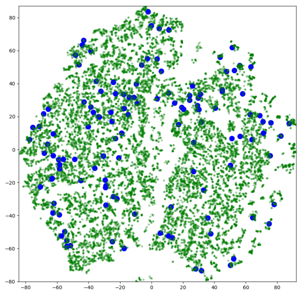
(b) $\tau = 0.2, M = 150$



(c) $\tau = 0.4, M = 128$



(d) $\tau = 0.6, M = 150$



(e) $\tau = 0.8, M = 124$

Figure 6.6: The snapshot placements in t-SNE space for different threshold τ with the number of snapshots M.

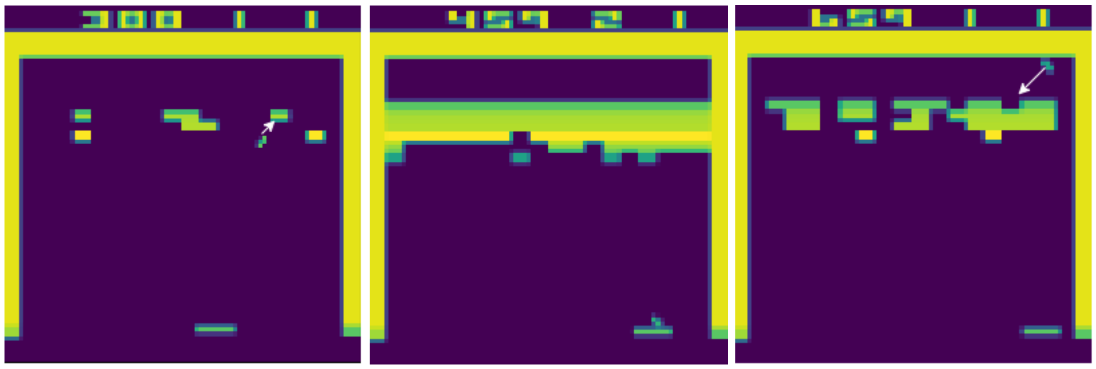

(a) (b) (c)

Figure 6.7: Three selected snapshots that are retained when increasing threshold from 0.6 to 0.8 in Breakout environment. White arrow shows the direction of the ball.

state space expands to different variations when the agent experiences diverse trajectories from the evolved policy and different action distributions. The number of snapshots slowly increases as the agent's experienced state space grows. I can observe that the snapshots well cover the whole state space and distributed somewhat evenly.

Fig. 6.6 depicts how the choice of threshold τ affects the selection of snapshots for interpretation. When no masking is applied to the state space ($\tau = 0$), the snapshots distribute densely although the use of state-value re-estimation reduces more than half of the snapshots compared to Q-value re-estimation. The saliency masking further reduces the snapshots as shown in the figure. Very closely located (unnecessarily similar) images to the most representative (important) snapshots seem to be discarded by unifying the input space after masking based on the saliency. This helps us to tell a story (or explain the agent's behavior) by focusing on the part of images where action-related event is occurring.

Fig. 6.7 shows three snapshots that are retained when increasing the threshold from $\tau = 0.6$ to $\tau = 0.8$. The image represents tricks of the agent need to memorize to record high score. Fig. 6.7a shows a small number of bricks at the top which is harder to get. However, the agent is able to capture the moment that it can control the bar in such a way that the ball can hit one of the bricks. Fig. 6.7b shows the ability of picking the ball so that the agent does not lose a life. The ability of getting huge sequence of reward through a trick shot is

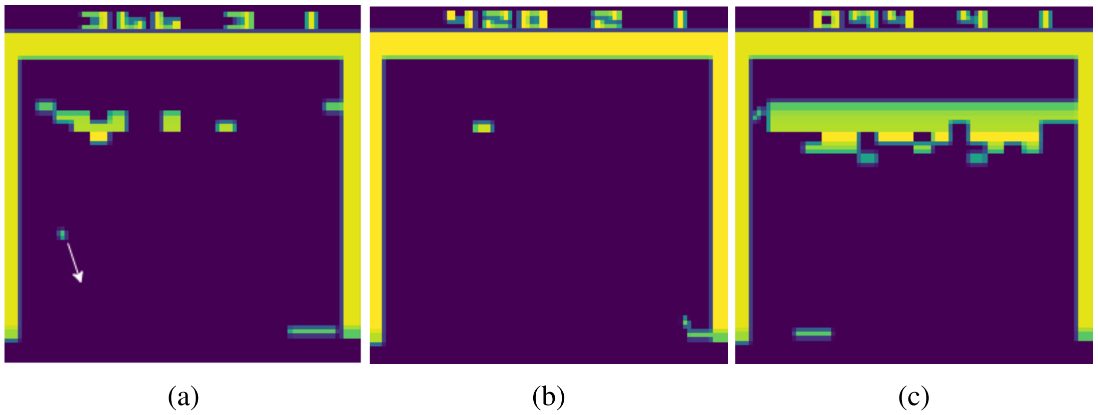


Figure 6.8: Three snapshots that newly appears when increasing threshold from 0.6 to 0.8 in Breakout environment. White arrow shows the direction of the ball.

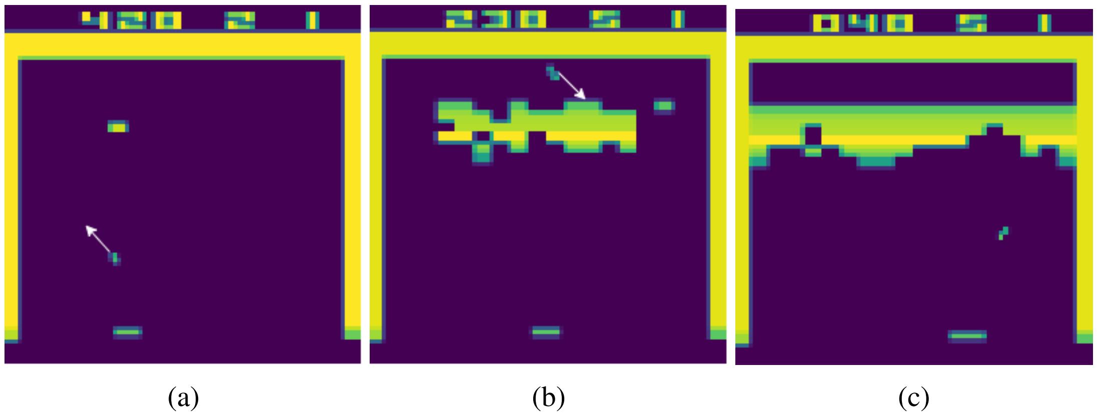


Figure 6.9: Three snapshots that are discarded when increasing threshold from 0.6 to 0.8 in Breakout environment. White arrow shows the direction of the ball.

also demonstrated in Fig. 6.7c. Even though the threshold is increased to get less number of snapshots, the core snapshots to maintain the idea of how to get more score and survive in Breakout still stay in the model.

Fig. 6.8 shows three new snapshots that appear even when increasing threshold from $\tau = 0.6$ to $\tau = 0.8$. The first snapshot shows a regret of not being able to catch the fall, which leads to a lost of life in Fig. 6.8a. Fig. 6.8b shows the ability of ball manipulation to get the final brick to reset the brick layers. Finally, a new trick shot also captured to remove almost all of the brick in Fig. 6.8c. Although the higher sparsity is imposed, important new experience or tricks will be kept to improve the quality of explanation.

Three example of discarded snapshots are presented when increasing the threshold from

$\tau = 0.6$ to $\tau = 0.8$ in Fig. 6.9. They are discarded because there are similar versions of them in the snapshots with higher threshold. I can see the agent tries to get the final brick in Fig. 6.9a which is similar idea with the new snapshots in Fig. 6.8b replaced), the high reward trick shot in Fig. 6.9b that overlaps with Fig. 6.7c and Fig. 6.8c, and a normal state catching in Fig. 6.9c that is similar to Fig. 6.7b.

6.4 Discussion

In this work, a novel method is developed to enhance the sparsity of evidence collection process to understand the behavior of deep reinforcement learning agent. Our method utilizes state-value approximation to construct sparse basis space. Saliency map is used for sifting the interested areas of state space to further reduce the number of snapshots by grouping the similar states with common attention. In this way, a lot less number of snapshots are remained which reduce human effort to efficiently analyze. The reduction of the number of snapshot images to examine for interpretation help us bring additional insights to the deep reinforcement learning agent's behaviors. This allowed us to discuss the observed snapshots, their meanings, and why some snapshots are removed or retained. The interpretation of sparse snapshots allows us to understand what skills are learned from the experience. Grouping of related snapshots will help further reduce the burden for interpretation, thus it could be natural next stop. An automatic grouping of snapshots would be helpful not only to understand the behavior of an agent but also to provide a radical reason of why some states or behaviors (skills) are similar.

CHAPTER 7: LEVERAGING SNAPSHOTS TO LEARN EFFECTIVE LATENT REPRESENTATION FOR CONTINUAL DEEP REINFORCEMENT LEARNING

Previous chapters have provide a core framework to retrieve, enforce, and improve DRL training process in one task. In this chapter, the knowledge of snapshots is leveraged to help continual DRL process. By helping the behavior interpretation of the agent, snapshots provide a great way to learn latent representation of old tasks. This prevents the catastrophic forgetting problem in continual DRL which the agent does not have access to samples from the old tasks while training new tasks. This chapter is reused from a submitting work (at the time of writing) with permission from the authors.

7.1 Motivation

In most reinforcement learning (RL) setups, an RL agent is trained in a stationary environment which assumes the environment does not change over time. This stationary assumption is, however, rarely true in real-world problem settings such as robotics [106, 107], finance [108, 109], recommendation system [110, 111], etc. Therefore, continual learning [112], thus life-long learning [113], has been suggested to address the challenges of adaptation in non-stationary environments. In a continual learning problem, the environment dynamics can be changed over time in a non-stationary environment or with changing tasks. For example, weather changes in outdoor robotic operation problems or gradually changing preferences of humans with co-creation robots are necessary to be considered [114]. RL is a natural fit in a continual learning setting as an agent-environment interaction paradigm amenable to studying the topic of learning in a continual fashion [115].

The discussion of this paper focuses on the continual learning challenges of deep reinforcement learning (DRL) as deep learning and neural network have become standard

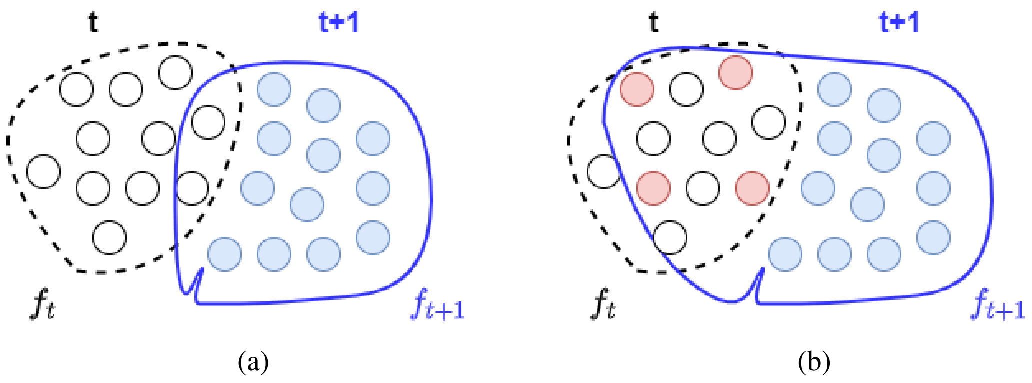


Figure 7.1: Leveraging snapshots for deep reinforcement learning agent's better coverage with using snapshots (b) and not using snapshots (a) to support the learning process with t and $t + 1$ are tasks changing sequentially; f_t and f_{t+1} are the function changed during the training of each task.

learning methods to learn complex real-world problems where continual learning demand is the largest. However, simple uses of neural networks in a continual RL environment often face two major issues: catastrophic forgetting [116] and stability-plasticity dilemma [117]. Catastrophic forgetting is a problem of preserving past knowledge and is caused by removing old knowledge when learning from new experiences retrieved from agent-environment interactions. The stability-plasticity dilemma is the tradeoff that is caused by rapid learning and adaptation to current experiences by sacrificing the stability or robustness of the RL agent, and vice versa.

In this work, the catastrophic forgetting and stability-plasticity dilemma are addressed for continual DRL problems in which the environment is split into a sequence of tasks where each task is different from the other tasks. For this, a framework is applied to identify important samples (known as *snapshots*) [92] and to generalize a representation learning function through those snapshots and new tasks' samples. Fig. 7.1 illustrates the motivation of our method to construct a better representation space encompassing the previous experience in order to prevent catastrophic forgetting and enhance the generalization. In Fig. 7.1a, when the traditional DRL agent switches from task t to task $t + 1$, the embedding function f_t evolves towards f_{t+1} to incorporate new experience with the cost of f_{t+1}'s losing a key representation for the past. The use of snapshots (red samples in Fig. 7.1b) allows f_{t+1} to develop robust representation that minimizes forgetting the past.

The general idea of our solution is to utilize the memorization of a small number of snapshots from old tasks along with an additional latent representation regularization method to enhance continual generalization for quickly adapting to new tasks and being resistant to catastrophic forgetting. Our approach (Fig. 7.1b) enables a DRL agent to leverage snapshots to have sufficient variance from representative data from the old tasks and new task data, which helps f_{t+1} develop robust embedding that minimizes forgetting through the proposed novel regularization. Furthermore, the snapshot-driven regularized representation learning consistently reinforce past knowledge by continuously relating new learning to the past snapshot knowledge.

Our main contributions are 1) designing an integrated framework for the retrieval and collection of snapshots for continual DRL, 2) proposing a novel relevant latent regularization leveraging the snapshots, and 3) validating our method on a set of different simulated non-stationary environments. In our experimental evaluation, I find that our method outperforms a strong baseline Meta-Experience Replay (MER) [118]. A visual explanation of the learned latent representation through the proposed regularization and its impact on continual DRL performance is provided.

7.2 Relevant Latent Regularization for Continual Deep Reinforcement Learning

Fig. 7.2 illustrates the overall framework of the proposed relevant latent regularization for continual deep reinforcement learning by leveraging DRL, DRL-Monitor, and snapshot storage [85]. For each task in the training process, the agent perceives state $\boldsymbol{s}_t$ for $t \in [0, T]$ from an interactive environment. An action a_t is drawn from a policy function $\boldsymbol{\pi}_{a \in \mathcal{A}}(a|\boldsymbol{s})$. The agent continues to perceive next state $\boldsymbol{s}'_t$ (equivalent with $\boldsymbol{s}_{t+1}$) with a reward r_t from the environment until the environment terminate by a condition or the agent has reached T steps. After training each task, important snapshots of the task are determined by the DRL-Monitor through the task's samples trajectory. The important snapshots are kept inside

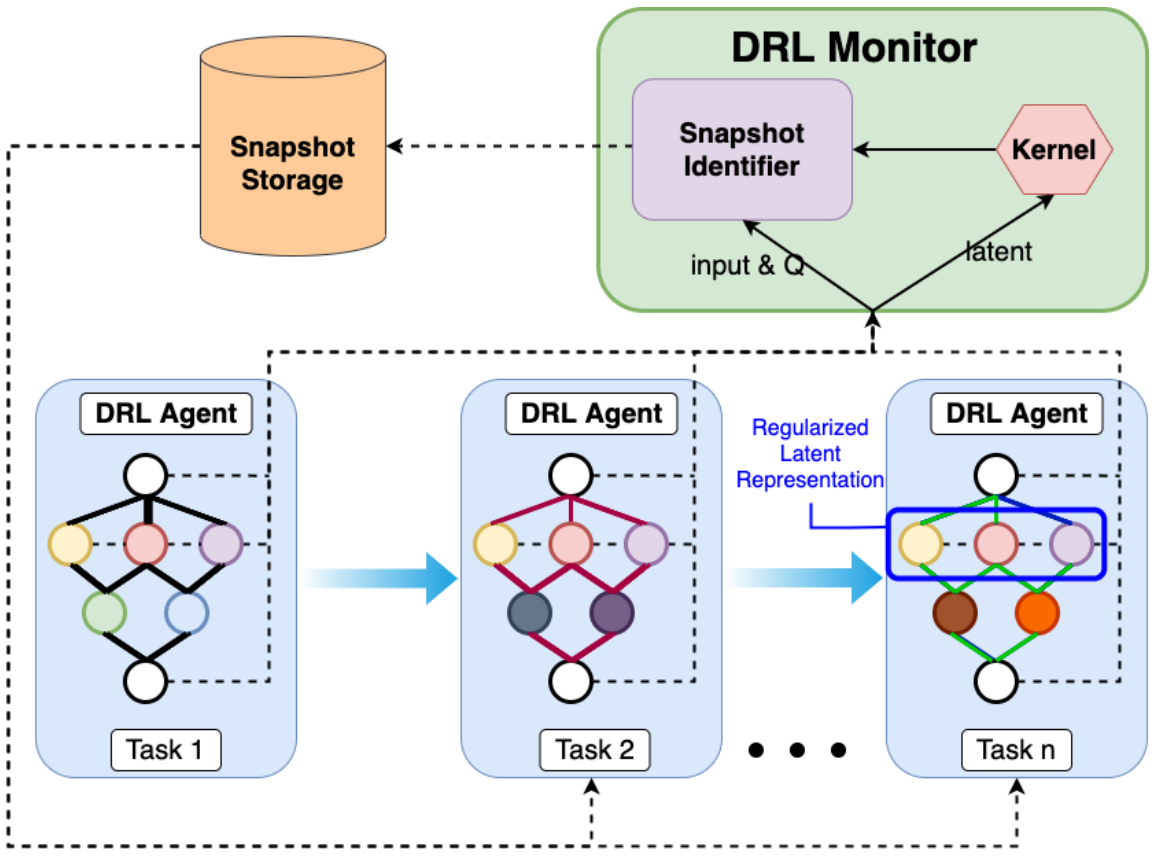


Figure 7.2: Evolution of a DRL agent with DRL-Monitor memorizing a small number of important snapshots to address the forgetting problem in continual DRL tasks $(1 \ldots n)$. The latent representations from the DRL agent are regularized to maximize the genralizability and stability.

snapshots storage and are used for preventing current and previous tasks from forgetting through the novel relevant latent regularization.

7.2.1 Relevance-Regularized Deep Reinforcement Learning Agent

In the DRL agent training process, tasks are considered in which the agent interacts with environments through a sequence of states, actions, and rewards. The agent's goal is to select actions from corresponding states that maximize the long-term cumulative rewards with a discount factor $\gamma \in [0, 1)$ which shows how much the agents care about a future reward.

At each timestep from the interaction, an experience tuple $(\boldsymbol{s}_t, a_t, r_t, \boldsymbol{s}'_t)$ is added to an experience replay buffer [93] $\mathcal{M}$. Training data is randomly sampled from $\mathcal{M}$ to provide an identical and independent distribution for the training process. The uniform distribution sampling method also helps avoid sampling bias to improve learning performance.

At the end of each task, a small set of important snapshots are identified from the task's samples in $\mathcal{M}$ and the information of snapshots information $(\boldsymbol{s}_{sn}, a_{sn}, r_{sn}, \boldsymbol{s}'_{sn}, \boldsymbol{l}_{sn})$ are stored in a snapshots storage $\mathcal{S}n$ (see Section. 7.2.2 for more detail). An uniform distribution sampling method is used for randomly sampling from $\mathcal{S}n$ to train the DRL agent.

The double Q-learning method [1] is used with two functions to predict Q-value and Q'-value. Q is the state-action value function to measure how good a state and action pair is $Q(\boldsymbol{s} \in \mathcal{S}, a \in \mathcal{A}) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Q function is design as a composition of 2 functions such that:

$$Q = h(g(\boldsymbol{s}; \boldsymbol{\theta}_g); \boldsymbol{\theta}_h) \qquad Q' = h(g(\boldsymbol{s}; \boldsymbol{\theta}'_g); \boldsymbol{\theta}'_h) \tag{7.1}$$

with g function outputs high-level representation (latent space) of the state $\boldsymbol{l} = g(\boldsymbol{s})$, h function outputs Q-values for all possible actions $a \in \mathcal{A}$, and $\boldsymbol{\theta}_g$ and $\boldsymbol{\theta}_h$ are parameters of g and h respectively. Huber loss [119] is used as the objective loss to optimize Q-value:

$$\mathcal{L}_Q^M = \begin{cases} \frac{1}{2}(y - Q(\boldsymbol{s}, a))^2 & \text{where } |y - Q(\boldsymbol{s}, a)| < \delta \\ \delta(|y - Q(\boldsymbol{s}, a)| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{7.2}$$

where $y = r_{t+1} + \gamma Q'(\boldsymbol{s}', \arg\max_a Q(\boldsymbol{s}', a))$ with $(\boldsymbol{s}, a, r, \boldsymbol{s}')$ are drawn from $\mathcal{M}$.

7.2.1.1 Relevant Latent Regularization

To keep previous tasks performing similar as it was before a new task is training, the latent space $\boldsymbol{l}$ of snapshots is regularized to be as they were. In order to do that, mean squared error is used for computing the loss for $\boldsymbol{l}$ from $\mathcal{S}n$ with current prediction of $\boldsymbol{l}$ from $g(\boldsymbol{s})$ to optimize g function such that:

$$\mathcal{L}_{\boldsymbol{l}}^{\mathcal{S}n} = \sum(\boldsymbol{l} - g(\boldsymbol{s}))^2. \tag{7.3}$$

h function is optimized by using Huber loss $\mathcal{L}_Q^{\mathcal{S}n}$. The samples information $(\boldsymbol{s}, a, r, \boldsymbol{s}', \boldsymbol{l})$ used for optimizing the losses $\mathcal{L}_{\boldsymbol{l}}^{\mathcal{S}n}$ and $\mathcal{L}_Q^{\mathcal{S}n}$ are drawn from $\mathcal{S}n$. The additional loss $\mathcal{L}_{\boldsymbol{l}}^{\mathcal{S}n}$ for snapshots helps the latent space of the task remains as before and helps the agent not forgetting of what the agent has learned from previous tasks. $\mathcal{L}_Q^{\mathcal{S}n}$ loss helps the agent be able to have better performance even when learning new tasks.

The gradients of $\boldsymbol{\theta}_g$ and $\boldsymbol{\theta}_h$ are computed as:

$$(\nabla_{\boldsymbol{\theta}_g}, \nabla_{\boldsymbol{\theta}_h}) = \left(\frac{\partial(\mathcal{L}_Q^M + \mathcal{L}_{\boldsymbol{l}}^{\mathcal{S}n})}{\partial \boldsymbol{\theta}_g}, \frac{\partial(\mathcal{L}_Q^M + \mathcal{L}_Q^{\mathcal{S}n})}{\partial \boldsymbol{\theta}_h} \right). \tag{7.4}$$

Adam optimizer [78] is used for optimizing the parameters $\boldsymbol{\theta}_h$ and $\boldsymbol{\theta}_g$ using the computed gradients in Eq. 7.4. After a certain iterations of gradient updates, $\boldsymbol{\theta}'_g = \boldsymbol{\theta}_g$ and $\boldsymbol{\theta}'_h = \boldsymbol{\theta}_h$ are set.

7.2.2 DRL-Monitor & Snapshots Storage

$\mathcal{M}$ has a cap of maximum number of samples can be stored. Therefore, as time goes on, previous tasks' samples will be wiped out from the experience replay buffer, and there is no access to previous tasks' samples. A mechanism to permanently store a small number of important snapshots is needed to gain access to previous tasks' samples and prevent the forgetting problem. Therefore, at the end of each task, N number of samples in $\mathcal{M}$ of the task are collected to identify M number of important snapshots of the task. In general, a small number of snapshots that are relevant to the task is stored but the prediction of other task's samples Q-value can still be interpolated.

7.2.2.1 Kernel

Positive kernel (basis function) is used for computing similarity between latent space $\boldsymbol{l}$ of one sample to another sample such that: $\phi_i(\boldsymbol{l}_j) = k(\boldsymbol{l}_j, \boldsymbol{l}_i) = k(\boldsymbol{l}_i, \boldsymbol{l}_j) \in \mathbb{R}$. $\boldsymbol{\Phi}$ is defined as the feature matrix which is a symmetric positive-definite matrix by using radial basis function (RBF) kernel such that:

$$\mathbf{\Phi} = \begin{bmatrix} 1, & \boldsymbol{\phi}_1(\boldsymbol{l}_1), & \boldsymbol{\phi}_1(\boldsymbol{l}_2), & \cdots, & \boldsymbol{\phi}_1(\boldsymbol{l}_N) \\ 1, & \boldsymbol{\phi}_2(\boldsymbol{l}_1), & \boldsymbol{\phi}_2(\boldsymbol{l}_2), & \cdots, & \boldsymbol{\phi}_2(\boldsymbol{l}_N) \\ & \vdots & & \ddots & \vdots \\ 1, & \boldsymbol{\phi}_N(\boldsymbol{l}_1), & \boldsymbol{\phi}_N(\boldsymbol{l}_2), & \cdots, & \boldsymbol{\phi}_N(\boldsymbol{l}_N) \end{bmatrix} \tag{7.5}$$

7.2.2.2 Snapshots Identifier

The target Q-value is a weighted sum of the feature matrix with some noise such that:

$$Q = \hat{Q} + \boldsymbol{\epsilon} = \mathbf{\Phi}\mathbf{w} + \boldsymbol{\epsilon} \tag{7.6}$$

where $\boldsymbol{\epsilon}$ is zero-mean Gaussian noise with variance σ^2. σ^2 is suggested to be initialized by a fraction (0.1) of variance of the target Q-values [80]:

$$\sigma^2 = 0.1\text{Var}(Q). \tag{7.7}$$

M is the number of important snapshots that changes during this identifying process. Thus, M is also the number of non-zero values in $\mathbf{w}$. A set of hyper-parameters is initialize to control the strength of the prior over corresponding samples that $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_N)$ to be infinity. The training process starts with $M = 1$, and the largest projection of samples on targets is picked and the first α is assigned by the following:

$$\boldsymbol{\alpha}_i = \frac{diag(\mathbf{\Phi}^\top \mathbf{\Phi})_i}{\left(\frac{(\mathbf{\Phi}^\top Q)^2}{diag(\mathbf{\Phi}^\top \mathbf{\Phi})}\right)_i - \text{Var}(Q)} \quad \text{where } i = \arg\max \frac{(\boldsymbol{\phi}^\top Q)^2}{diag(\boldsymbol{\phi}^\top \boldsymbol{\phi})}. \tag{7.8}$$

Given $\boldsymbol{\alpha}$, the posterior distribution is sampled from a normal distribution as $p(\mathbf{w}|Q, \boldsymbol{\alpha}, \sigma^2) \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$. Since $\mathbf{\Phi}$ and $\boldsymbol{\alpha}$ are symmetric positive-definite matrix and vector, a Hermitian matrix $\mathbf{H}$ over real positive number field can be computed as:

$$\mathbf{H} = \frac{1}{\sigma^2}\mathbf{\Phi}^\top \mathbf{\Phi} + \boldsymbol{\alpha I}. \tag{7.9}$$

$\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ are computed as:

$$\mathbf{\Sigma} = \mathbf{H}^{-1} \qquad \boldsymbol{\mu} = \frac{1}{\sigma^2}\mathbf{\Sigma}\mathbf{\Phi}^\top Q. \tag{7.10}$$

Cholesky decomposition is used for optimizing the computation of matrix inversion for faster run-time [120] of calculating $\mathbf{\Sigma}$.

Sparsity s and *quantity* $\mathbf{q}$ factors are used for measuring the extent of overlaps from one basis to other bases and measure the alignment error when the basis is removed such that:

$$\mathrm{s} = \frac{1}{\sigma^2}diag(\mathbf{\Phi}^\top \mathbf{\Phi}) - \frac{1}{\sigma^4}\sum_m (\mathbf{\Phi}^\top \mathbf{\Phi})_{:,m}\mathbf{\Sigma}_{m,m} \times (\mathbf{\Phi}^\top \mathbf{\Phi})_{:,m} \tag{7.11}$$

$$\mathbf{q} = \frac{1}{\sigma^2}\mathbf{\Phi}^\top Q - \frac{1}{\sigma^4}(\mathbf{\Phi}^\top \mathbf{\Phi})_{:,m}\mathbf{\Sigma}_{m,m}(\mathbf{\Phi}^\top Q)_{:,m} \tag{7.12}$$

where m indicates the important snapshots of all samples at that moment of training process. s_m and $\mathbf{q}_m$ are suggested to be updated [81] as:

$$\mathrm{s}_m = \frac{\boldsymbol{\alpha}_m \times \mathrm{s}_m}{\boldsymbol{\alpha}_m - \mathrm{s}_m} \qquad \mathbf{q}_m = \frac{\boldsymbol{\alpha}_m \times \mathbf{q}_m}{\boldsymbol{\alpha}_m - \mathrm{s}_m} \tag{7.13}$$

s and $\mathbf{q}^2$ are compared for all samples to determine whether a sample is an important snapshot:

- If $\mathrm{s}_i < \mathrm{q}_i^2$ and $\alpha_i < \infty$, α_i is re-estimated using Eq. 7.14 since sample i is already a snapshot.
- If $\mathrm{s}_i < \mathrm{q}_i^2$ and $\alpha_i = \infty$, sample i is considered as new snapshot and re-estimated α_i using Eq. 7.14.

- If $\mathrm{s}_i \geq \mathrm{q}_i^2$ and $\alpha_i < \infty$, sample i is considered not to be a snapshot and set $\alpha_i = \infty$.

Weight's strength α_i (when $\mathrm{s}_i < \mathrm{q}_i^2$) and noise level σ^2 are re-estimated as:

$$\alpha_i = \frac{\mathrm{s}_i^2}{\mathrm{q}_i^2 - \mathrm{s}} \qquad \sigma^2 = \frac{(Q - \hat{Q})^2}{N - M + \sum_m \alpha_m \boldsymbol{\Sigma}_{m,m}} \tag{7.14}$$

The algorithm re-computes $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ and the followed process until a convergent condition is met or a certain number of iterations reached.

7.2.2.3 Snapshots Storage

The snapshots are collected from the process to store them in the snapshots storage. The information of the snapshots includes state $\boldsymbol{s}$, action a, reward r, next state $\boldsymbol{s}'$, and the latent space $\boldsymbol{l}$. The snapshots will not be removed by the system and stay until the end of the program. It is not necessary to specify which sample is from which task.

7.3 Experiments

7.3.1 Environments

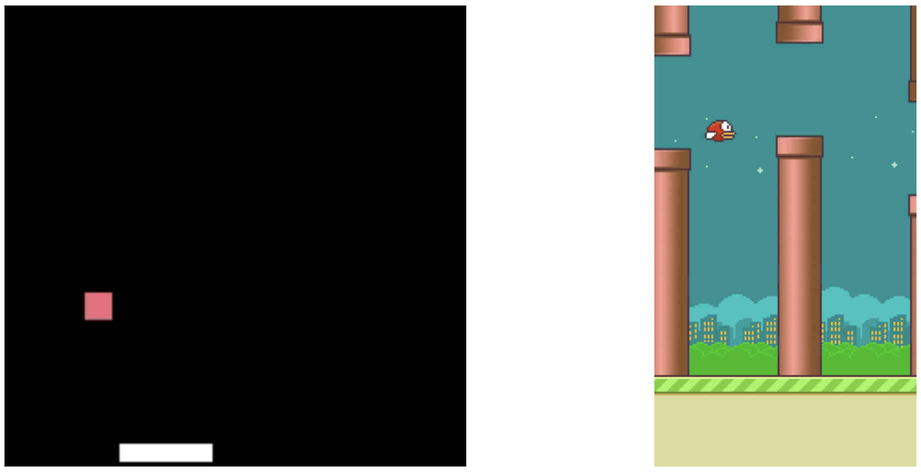


Figure 7.3: Catcher (left) and Flappy Bird (right) game environments.

The efficacy of the proposed Relevant Latent Regularization is evaluated in two different continual reinforcement learning environments by comparing it to the state-of-the-art, Meta-Experience Replay (MER) [118]. Environmental changeable Catcher and Flappy

Bird are used [121], which the environment setups of the MER is duplicated.

Fig. 7.3 shows screenshots of the two environments. 30 frames per second for the agent to play the game are used. The agent observes screen frames and puts four consecutive frames into a queue as a state and receives a reward after every action. The agent does not have access to the changeable parameters to prevent knowledge of the environmental changes. The environment changes every $25,000$ step to simulate the highly non-stationary settings.

7.3.1.1 Catcher

In this game, the red fruit starts randomly on top of the screen and falls down. There are three possible actions: move left, move right, and stay. The agent needs to catch the fruit in order to receive a $+1$ reward. If the agent cannot catch the fruit when it falls to the bottom, the agent will get a -1 reward and lose one life in the game. Zero rewards will be issued for the other situations. The agent has three lives to accumulate as many rewards as it can. To save time, the number of frames per game to $15,000$ steps maximum is limited. The speed of the fruit is increased by 0.03 (starting from 0.608) to create different environments.

7.3.1.2 Flappy Bird

In this game, the bird needs to pass through the pipes from left to right. The agent can choose to flap the wing to slightly move up or no action which will let the bird lower down. The agent gets a $+1$ reward when passing the pipes, and a -1 reward when the bird touches the pipe, ground, or goes over the top of the screen, and loses one life. The agent only has one life to accumulate as many rewards as it can. The agent gets zero rewards otherwise. When the environment is changed to create a new task, the gap between the two vertical pipes decreases by 5, starting at 100.

7.3.2 Neural Networks & Hyper-parameters Setup

The neural network architecture and optimization parameters is used similar to deep Q-network [122]. The neural network takes an input of stacked four consecutive frames with the size of 84×84 and goes through a series of convolution layers to get a latent space and use a fully connected layer to output Q-values of all possible actions. The latent embedding has a length of 256 units. The size of experience replay buffer $\mathcal{M}$ is limited by a maximum capacity of $50,000$ transitions which oldest transition is removed when a new transition comes in so that old tasks' samples are quickly removed from the replay buffer. Our network and environment setups are similar to MER which outperforms traditional deep Q-network in both Catcher and Flappy Bird games.

7.3.3 Training Evaluation

Fig. 7.4 shows the performance on each individual task throughout the $125,000$ training steps for both Catcher (left figures) and Flappy Bird (right figures). The yellow color is showing which task the agent is currently evaluating. Since the environments have a limited number of steps to take, the maximum rewards of each task in Catcher are also different due to the speed of the fruit as a faster fall will result in higher total rewards given the same number of maximum steps taken. Performance in Catcher reaches the maximum for every task after $15,000$ steps which indicates the agent did not spend all three lives.

The result shows a consistently better performance of RLR than the optimization approach using MER while both RLR and MER do not forget past experience. MER only be able to maintain the performance of the target task performance when switched to different tasks. However, by using snapshots and relevance-regularized latent representations, our RLR agent is able to continuously increase its performance while learning different tasks. This behavior is intuitive since the agent has access to a small number of old task samples and it continuously relates new experience or knowledge to the past. Learning with snapshots exhibits the kind of learning patterns from humans which remember a few key moments from the previous environment to be able to accelerate learning of new tasks as well as to reinforce knowledge of the past tasks, while MER fails to reestablish or strengthen past knowledge with new experience from different tasks.

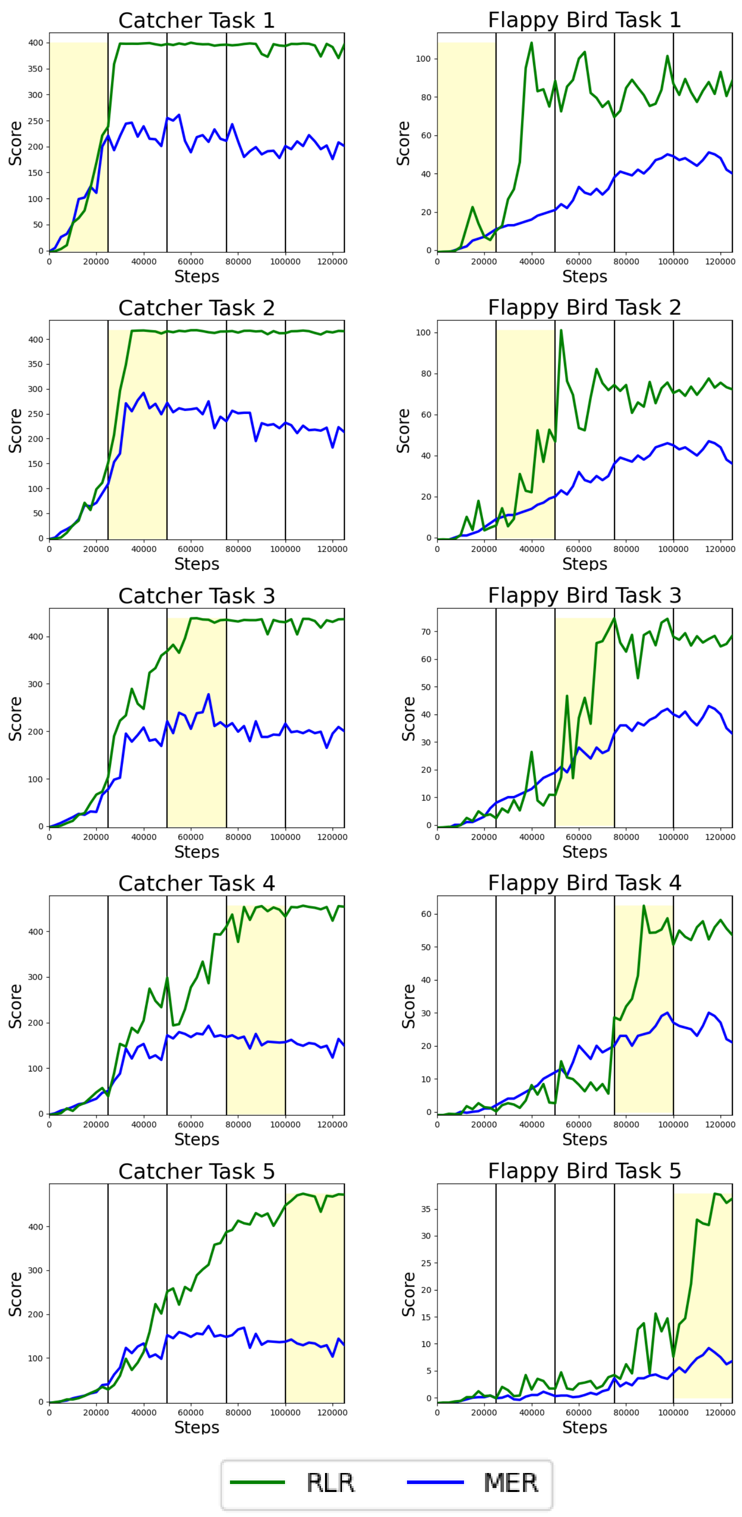


Figure 7.4: Catcher (left) & Flappy Bird (right) evaluation on five different tasks during the training process. The score values are the average of 10 evaluations in a single training from evaluation process for each task separately. The vertical grid lines indicate a task switch, moving from task 1 to task 5. The yellow area shows the currently evaluating task. The proposed Relevant Latent Regularization (RLR) are shown in green and MER in blue.

7.3.4 Continual Training

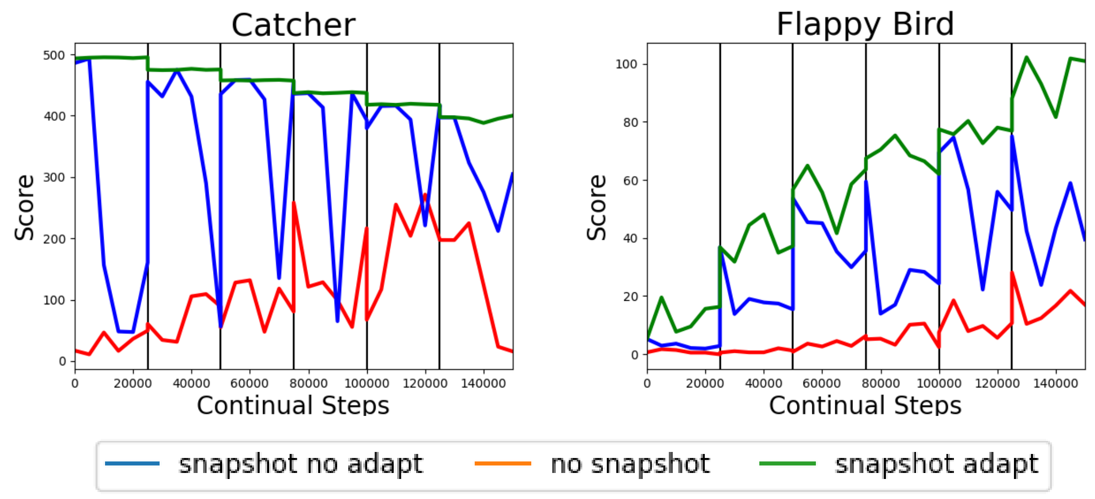


Figure 7.5: Continual training with a new task in the beginning for Catcher (left) and Flappy Bird (right) comparison of the average 10 run score between using snapshots adaptation (green), no snapshots adaptation (blue), and deep Q-network (red).

To better observe the impact of the use of snapshots and regularization (no adaptation) in a continual setting. Fig. 7.5 shows the performance when using different methods to continuously train task 6 (new task), task 5, task 4, task 3, task 2, and task 1 for $25,000$ steps each task. The green line shows the agent was trained using the joint losses ($\mathcal{L}_l^{Sn}$ and $\mathcal{L}_Q^{Sn}$) for RLR and using the same method for this continual training session. The blue line reflects the agent trained using RLR but not using the regularization method for this session. The red line is the traditional DQN continual training.

In Catcher, I observe that the performance when using snapshots with RLR method in green lines is better than without using the two losses in blue. Without using RLR, the agent has a big drop in the training task but recovers when switches back to task the agent is familiar with in blue. This shows the possible memorization of old tasks when the agent is trained with regularizing snapshots. A simple DQN model cannot achieve the same level of performance in red. The lower reward after each task is due to the maximum number of steps in a task with higher task index results a faster red fruit drop; hence higher reward achieves.

I observe a similar phenomenon in the Flappy Bird environment. The agent trained with RLR is able to achieve a similar or higher level of performance as the training evaluation session. A similar drop in performance in Catcher when not using the regularization method also happens in Flappy Bird. The traditional DQN method cannot reach a high reward in this setup.

The snapshots are analyzed for a possible reason why there is a drop in performance when not using RLR method in Section. 7.3.5.

7.3.5 Snapshots Analysis

At the end of the training process, the agent records (over $25,000$ samples of each task) 755 snapshots for task 1, 939 snapshots for task 2, $1,231$ snapshots for task 3, $1,269$ snapshots for task 4, and $1,363$ snapshots for task 5 in Catcher environment. The agent also records $1,590$ snapshots for task 1, $1,665$ snapshots for task 2, $1,637$ snapshots for task 3, $1,663$ snapshots for task 4, and $1,625$ snapshots for task 5 in Flappy Bird environment. I observe the growth of the number of snapshots for each task slows down and stays in a range after several tasks, given the similarity of experience. The rate of snapshots over the total number of samples for each task $(25,000)$ ranges from 3% to 6% which is significantly smaller than remembering all of the samples.

Fig. 7.6 shows t-SNE [13] projections of Catcher's snapshots using the snapshots identifier method. The left figure shows the projections when snapshots are not used for training (purely deep Q-network), and the right figure shows the projections when the snapshots are used and adding $\mathcal{L}_l^{Sn}$ and $\mathcal{L}_Q^{Sn}$ losses to the objective function. I observe that the snapshots are mixed without using the two additional losses. In contrast, the snapshots of each task are well-defined in different space, especially for task 1 and task 2 when the two additional losses are used to regularize the latent representation. This shows that the RLR method has an ability of separating task without explicitly telling which task it is. The well-defined space could be the reason for not having performance loss when adapting to new task with-

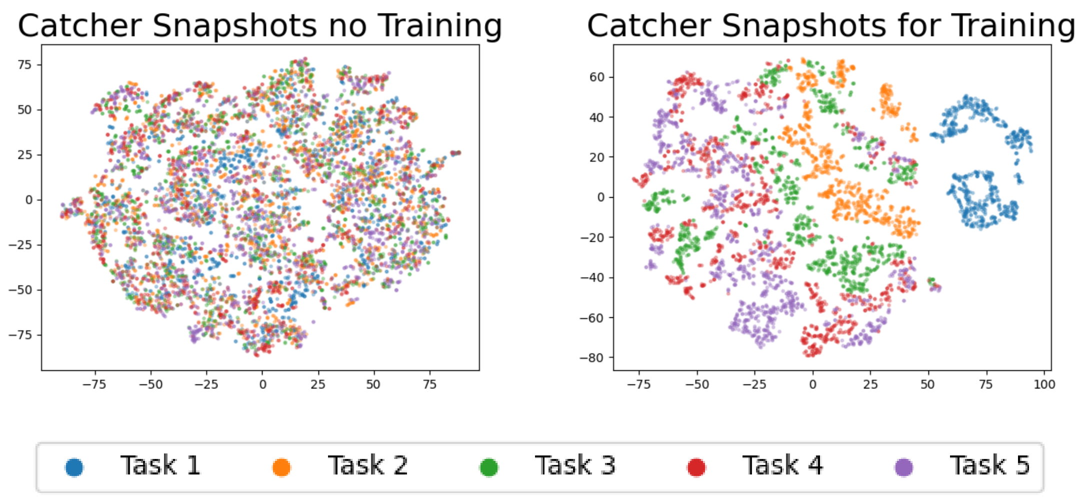


Figure 7.6: Catcher snapshots t-SNE projections when using our DRL monitor with (right) and without (left) using the snapshots during the training process. The tasks sequence is in order of blue, orange, green, red, and purple.

out the RLR in blue lines in Fig. 7.5. When not using RLR in continual training session, the training task samples could move to be too close to other tasks (previously separated) and become difficult to choose a correct action which previously learns for different task. Therefore, using RLR is important to maintain the separation of each task in order to have an unified decision process. Well-defined knowledge representation makes it easy to relate new experience to the old experience to learn better and strengthen the old tasks at the same time.

7.4 Discussion

In this work, a novel latent regularization is proposed for continual deep reinforcement learning by leveraging snapshots of past experience. I empirically presented that the additional losses regulate the latent representation of DRL agents in a way that it does not forget old tasks while training on a new task. Furthermore, I observe the regularized learning of a new task strengthen the past knowledge at the same time, which outperforms the state-of-the-art Meta-Experience Replay. This advance in continual reinforcement learning was achieved by form uncluttered latent space by continuously regularizing it with past snapshot memories.

Although memorizing snapshots and using it for regularizing DRL networks are effective, the memory requirements are expected to grow when the sequential tasks are distant from previous tasks. Exploring space-performance trade-off and examining generative models [123, 31] to address it can be an interesting future direction. Theoretical examination of relevance regularization and design of new robust objective function can be also explored.

CHAPTER 8: CONCLUSION & FUTURE WORKS

Memory is one of the most important components for successful learning and generalization of knowledge. Retrieving and reusing knowledge are two challenging problems when applying these steps in intelligent systems. In this work, a framework is proposed for systematic knowledge acquisition, retention, and transfer to support human in interpreting DRL agent's behavior and help improving the learning process. By remembering important snapshots, representative information can be easily stored. The snapshots can be considered as knowledge representation that can be generalized over different tasks. By monitoring the snapshots, I understand how DRL agent builds up its knowledge and how I can use the snapshots to improve the performance of the agent. The results show that the knowledge retention and transfer approach through the proposed framework can make learning robust and efficient to prevent catastrophic forgetting that causes instability of learning. The snapshots have shown its power of analysis, improve learning performance, and generalize to continual deep reinforcement learning problems.

Main contributions of this dissertation to reinforcement learning research can be summarized as follow:

- **Monitor deep reinforcement learning agent as a tool to understand behavior:**

  By adopting sparse Bayesian reinforcement learning, the agent's behavior are well monitored during every step of the learning process. Through the snapshots, I am able to understand how the agent builds its knowledge to perform well in different environments. I also show possible problems of the DRL methods in some environments to explain why an agent does not perform well. With the interpretation from the snapshots, I provide suggestion of how to fix the agent during the training.

- **Improved learning with snapshots tagging:**

  From the proposed DRL-Monitor for understanding, the agent can benefit from using the snapshots to improve the performance during the training. A simple, but novel relevant experience replay leverages the monitor to identify important snapshots and tag them. By mixing the samples between snapshots and regular samples in each training step, I show a better performance for the DRL agent compared to the existing methods.

- **Snapshots reduction for easier analysis and understanding:**

  Saliency maps sift the interested areas of state space to further reduce the number of snapshots by grouping the similar states with common attention. By applying saliency map, I am able to enhance the sparsity of the evidence collection process to understand the behavior of the deep reinforcement learning agent. This also helps reduce human effort to have an efficient analysis. Through the analysis of the groups, I am able to understand why some snapshots are removed and why additional snapshots are presented as new knowledge.

- **Prevent catastrophic forgetting in continual deep reinforcement learning:**

  With the memorization of old task snapshots, I am able to address a continual deep reinforcement learning problem. The snapshots not only help to retain significant past knowledge but also provide the improvement of old tasks while training new similar tasks. t-SNE analysis (Figure 7.6) explains that the snapshots of one task are grouped together, exclusively from the other tasks. This phenomenon helps the agent learns better in the new tasks while preserving the knowledge from old tasks.

- **Stabilize representation with snapshots' latent regularization:**

  I propose a simple, snapshot-regulating method through the latent representation in continual deep reinforcement learning setting. The snapshot-regulation enforce the representations of the old task snapshots to stay the same and be separated the representations of new task samples. By constraining the modification of the latent representation of old tasks' snapshots, the agent tries to create effective knowledge separation boundaries between tasks. These boundaries are shown to be helpful for the agent to retain old knowledge while learning new tasks.

Since this dissertation has shown a large potential of the proposed approaches, it has many more steps to further investigate. The following lists the interesting future directions:

- **Grouping snapshots to reduce human analysis effort:**

  Even though the number of snapshots can be reduced through saliency map method and simplified reapproximation, the key snapshots for human interpretation still needs to be a small number. We can device a human analysis model by leveraging a group of snapshots in a macro scale, instead of interpreting an agent's micro-level behavior given a snapshot.

- **Discovering new kernel method to reduce number of snapshots:**

  Kernel methods often provide great ways to describe the similarity between samples. A better-described similarity (potentially domain-specific or use-inspired) can produce high-quality snapshots, which are both sparse and effective performance and interpretation.

- **Combining snapshots with other sampling techniques:**

  The usage of snapshots has proven to significantly improve the performance of the agent. However, in this work, only uniform sampling technique is used. Focusing more on larger loss snapshots using prioritized sampling technique [99] can possibly further enhance the DRL agent's performance.

- **Discovering new latent regularization method with snapshots:**

Current latent regularization method used in this work is using a naive L2 loss. Further investigation to better landscape tasks and well separate different task knowledge to leverage past and current experience for effective continual reinforcement learning.

REFERENCES

[1] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[2] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. P. Reyes, M.-L. Shyu, S.-C. Chen, and S. Iyengar, “A survey on deep learning: Algorithms, techniques, and applications,” *ACM Computing Surveys*, vol. 51, no. 5, pp. 1–36, 2018.

[3] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, “A survey on deep learning for big data,” *Information Fusion*, vol. 42, pp. 146–157, 2018.

[4] D. W. Otter, J. R. Medina, and J. K. Kalita, “A survey of the usages of deep learning for natural language processing,” *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[5] A. I. Károly, P. Galambos, J. Kuti, and I. J. Rudas, “Deep learning in robotics: Survey on model structures and training strategies,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2020.

[6] J. Yosinski, J. Clune, A. M. Nguyen, T. J. Fuchs, and H. Lipson, “Understanding Neural Networks Through Deep Visualization.,” *CoRR*, vol. abs/1506.06579, 2015.

[7] G. Dao and M. Lee, “Demysifying deep neural networks through interpretation: A survey,” *arXiv preprint arXiv:2012.07119*, 2020.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[9] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.

[10] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *arXiv preprint arXiv:1312.6199*, 2013.

[11] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in *Proceedings of the 26th annual International Conference on Machine Learning*, pp. 609–616, 2009.

[12] D. Frey and R. Pimentel, “Principal component analysis and factor analysis,” 1978.

[13] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[14] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *European Conference on Computer Vision*, pp. 818–833, Springer, 2014.

[15] M. T. Ribeiro, S. Singh, and C. Guestrin, “" why should i trust you?" explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, 2016.

[16] G. P. Schmitz, C. Aldrich, and F. S. Gouws, “Ann-dt: an algorithm for extraction of decision trees from artificial neural networks,” *IEEE Transactions on Neural Networks*, vol. 10, no. 6, pp. 1392–1401, 1999.

[17] L. Fu, “Rule generation from neural networks,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 8, pp. 1114–1124, 1994.

[18] A. Graves, G. Wayne, and I. Danihelka, “Neural turing machines,” *arXiv preprint arXiv:1410.5401*, 2014.

[19] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014.

[20] M.-T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” *arXiv preprint arXiv:1508.04025*, 2015.

[21] A. Canziani, A. Paszke, and E. Culurciello, “An analysis of deep neural network models for practical applications,” *arXiv preprint arXiv:1605.07678*, 2016.

[22] T. Xiao, Y. Xu, K. Yang, J. Zhang, Y. Peng, and Z. Zhang, “The application of two-level attention models in deep convolutional neural network for fine-grained image classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 842–850, 2015.

[23] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, “Residual attention network for image classification,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3156–3164, 2017.

[24] L.-C. Chen, Y. Yang, J. Wang, W. Xu, and A. L. Yuille, “Attention to scale: Scale-aware semantic image segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3640–3649, 2016.

[25] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual attention,” in *International Conference on Machine Learning*, pp. 2048–2057, 2015.

[26] J. Lu, J. Yang, D. Batra, and D. Parikh, “Hierarchical question-image co-attention for visual question answering,” in *Advances in Neural Information Processing Systems*, pp. 289–297, 2016.

[27] J. Lu, C. Xiong, D. Parikh, and R. Socher, "Knowing when to look: Adaptive attention via a visual sentinel for image captioning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 375–383, 2017.

[28] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, "Bottom-up and top-down attention for image captioning and visual question answering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6077–6086, 2018.

[29] A. Das, H. Agrawal, L. Zitnick, D. Parikh, and D. Batra, "Human attention in visual question answering: Do humans and deep networks look at the same regions?," *Computer Vision and Image Understanding*, vol. 163, pp. 90–100, 2017.

[30] A. S. Ross, M. C. Hughes, and F. Doshi-Velez, "Right for the right reasons: Training differentiable models by constraining their explanations," *arXiv preprint arXiv:1703.03717*, 2017.

[31] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

[32] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in Neural Information Processing Systems*, pp. 2172–2180, 2016.

[33] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *Proceedings of the 34th International Conference on Machine Learning*, pp. 2642–2651, JMLR. org, 2017.

[34] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework.," 2016.

[35] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner, "Understanding disentangling in beta-vae," *arXiv preprint arXiv:1804.03599*, 2018.

[36] S. van Steenkiste, F. Locatello, J. Schmidhuber, and O. Bachem, "Are disentangled representations helpful for abstract visual reasoning?," in *Advances in Neural Information Processing Systems*, pp. 14222–14235, 2019.

[37] Q. Zhang, R. Cao, Y. N. Wu, and S.-C. Zhu, "Growing interpretable part graphs on convnets via multi-shot learning," in *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017.

[38] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in Neural Information Processing Systems*, pp. 3856–3866, 2017.

[39] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[40] S. Löwe, P. O'Connor, and B. Veeling, "Putting an end to end-to-end: Gradient-isolated learning of representations," in *Advances in Neural Information Processing Systems*, pp. 3033–3045, 2019.

[41] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[42] R. Yousefzadeh and D. P. O'Leary, "Interpreting neural networks using flip points," *arXiv preprint arXiv:1903.08789*, 2019.

[43] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 5, pp. 828–841, 2019.

[44] T. Brown, D. Mane, A. Roy, M. Abadi, and J. Gilmer, "Adversarial patch," 2017.

[45] A. Athalye, L. Engstrom, A. Ilyas, and K. Kwok, "Synthesizing robust adversarial examples," *arXiv preprint arXiv:1707.07397*, 2017.

[46] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[47] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The building blocks of interpretability," *Distill*, vol. 3, no. 3, p. e10, 2018.

[48] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Object detectors emerge in deep scene cnns," *arXiv preprint arXiv:1412.6856*, 2014.

[49] A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," in *Advances in Neural Information Processing Systems*, pp. 3387–3395, 2016.

[50] A. Mordvintsev, N. Pezzotti, L. Schubert, and C. Olah, "Differentiable image parameterizations," *Distill*, 2018. https://distill.pub/2018/differentiable-parameterizations.

[51] Q.-s. Zhang and S.-C. Zhu, "Visual interpretability for deep learning: a survey," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 1, pp. 27–39, 2018.

[52] 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.

[53] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network dissection: Quantifying interpretability of deep visual representations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6541–6549, 2017.

[54] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, 2015.

[55] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2921–2929, 2016.

[56] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," in *Proceedings of the 34th International Conference on Machine Learning*, pp. 3145–3153, JMLR. org, 2017.

[57] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, pp. 618–626, 2017.

[58] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proceedings of the 34th International Conference on Machine Learning*, pp. 3319–3328, JMLR. org, 2017.

[59] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, "Smoothgrad: removing noise by adding noise," *arXiv preprint arXiv:1706.03825*, 2017.

[60] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," *arXiv preprint arXiv:1711.00138*, 2017.

[61] M. Sato and H. Tsukimoto, "Rule extraction from neural networks via decision tree induction," in *Proceedings of International Joint Conference on Neural Networks*, vol. 3, pp. 1870–1875, IEEE, 2001.

[62] M. G. Augasta and T. Kathirvalavakumar, "Reverse engineering the neural networks for rule extraction in classification problems," *Neural Processing Letters*, vol. 35, no. 2, pp. 131–150, 2012.

[63] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[64] J. R. Zilke, E. L. Mencía, and F. Janssen, "Deepred–rule extraction from deep neural networks," in *International Conference on Discovery Science*, pp. 457–473, Springer, 2016.

[65] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Advances in Neural Information Processing Systems*, pp. 2494–2504, 2018.

[66] R. Andrews, J. Diederich, and A. B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge-based Systems*, vol. 8, no. 6, pp. 373–389, 1995.

[67] H. Tsukimoto, "Extracting rules from trained neural networks," *IEEE Transactions on Neural Networks*, vol. 11, no. 2, pp. 377–389, 2000.

[68] J. M. Benítez, J. L. Castro, and I. Requena, "Are artificial neural networks black boxes?," *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1156–1164, 1997.

[69] G. G. Towell and J. W. Shavlik, "Extracting refined rules from knowledge-based neural networks," *Machine Learning*, vol. 13, no. 1, pp. 71–101, 1993.

[70] R. Setiono and W. K. Leow, "Fernn: An algorithm for fast extraction of rules from neural networks," *Applied Intelligence*, vol. 12, no. 1-2, pp. 15–25, 2000.

[71] S. Thrun, "Extracting rules from artificial neural networks with distributed representations," in *Advances in Neural Information Processing Systems*, pp. 505–512, 1995.

[72] M. W. Craven, "Extracting comprehensible models from trained neural networks," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 1996.

[73] I. A. Taha and J. Ghosh, "Symbolic interpretation of artificial neural networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 3, pp. 448–463, 1999.

[74] U. Johansson, R. Konig, and L. Niklasson, "Automatically balancing accuracy and comprehensibility in predictive modeling," in *Proceedings in the 7th International Conference on Information Fusion*, vol. 2, pp. 7–pp, IEEE, 2005.

[75] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE Transactions on Neural Networks*, vol. 8, no. 1, pp. 98–113, 1997.

[76] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[77] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*, pp. 421–436, Springer, 2012.

[78] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[79] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," *arXiv preprint arXiv:1908.03265*, 2019.

[80] M. E. Tipping, "The relevance vector machine," in *Advances in Neural Information Processing Systems*, pp. 652–658, 2000.

[81] M. E. Tipping, "Sparse bayesian learning and the relevance vector machine," *Journal of Machine Learning Research*, vol. 1, no. Jun, pp. 211–244, 2001.

[82] J. O. Berger, *Statistical decision theory and Bayesian analysis*. Springer Science & Business Media, 2013.

[83] D. J. MacKay, "Bayesian interpolation," *Neural Computation*, vol. 4, no. 3, pp. 415–447, 1992.

[84] M. Lee, *Sparse Bayesian Reinforcement Learning*. PhD thesis, Colorado State University, 2017.

[85] G. Dao, I. Mishra, and M. Lee, "Deep reinforcement learning monitor for snapshot recording," in *2018 17th IEEE International Conference on Machine Learning and Applications*, pp. 591–598, IEEE, 2018.

[86] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization.," in *ICCV*, pp. 618–626, 2017.

[87] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The Building Blocks of Interpretability.," *Distill*, 2018. https://distill.pub/2018/building-blocks.

[88] H. Li, Z. Xu, G. Taylor, and T. Goldstein, "Visualizing the Loss Landscape of Neural Nets.," *arXiv preprint arXiv:1712.09913*, 2017.

[89] G. Montavon, W. Samek, and K.-R. Müller, "Methods for Interpreting and Understanding Deep Neural Networks.," *Digital Signal Processing*, 2017.

[90] M. Tulio Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?: Explaining the Predictions of Any Classifier.," *arXiv preprint arXiv:1602.04938*, 2016.

[91] T. Zahavy, N. Ben-Zrihem, and S. Mannor, "Graying the black box: Understanding dqns," in *International Conference on Machine Learning*, pp. 1899–1908, 2016.

[92] G. Dao and M. Lee, "Relevant experiences in replay buffer," in *2019 IEEE Symposium Series on Computational Intelligence*, pp. 94–101, IEEE, 2019.

[93] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3, pp. 293–321, 1992.

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

[95] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[96] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie,

A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[97] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Hess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2016.

[98] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, “Hindsight experience replay,” in *Advances in Neural Information Processing Systems*, pp. 5048–5058, Curran Associates, Inc., 2017.

[99] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized experience replay,” *arXiv preprint arXiv:1511.05952*, 2015.

[100] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, “Distributed prioritized experience replay,” *arXiv preprint arXiv:1803.00933*, 2018.

[101] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[102] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “OpenAI gym,” *arXiv preprint arXiv:1606.01540*, 2016.

[103] G. Dao, W. H. Huff, and M. Lee, “Learning sparse evidence-driven interpretation to understand deep reinforcement learning agents,” in *2021 IEEE Symposium Series on Computational Intelligence*, pp. 1–7, IEEE, 2021.

[104] A. Alharin, T.-N. Doan, and M. Sartipi, “Reinforcement learning interpretation methods: A survey,” *IEEE Access*, vol. 8, pp. 171058–171077, 2020.

[105] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

[106] T. Lesort, V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez, “Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges,” *Information Fusion*, vol. 58, pp. 52–68, 2020.

[107] N. Churamani, S. Kalkan, and H. Gunes, “Continual learning for affective robotics: Why, what and how?,” in *2020 29th IEEE International Conference on Robot and Human Interactive Communication*, pp. 425–431, IEEE, 2020.

[108] D. Philps, T. Weyde, A. d. Garcez, and R. Batchelor, “Continual learning augmented investment decisions,” *arXiv preprint arXiv:1812.02340*, 2018.

[109] D. Philps, *A temporal continual learning framework for investment decisions*. PhD thesis, City, University of London, 2020.

[110] M. Xie, K. Ren, Y. Lu, G. Yang, Q. Xu, B. Wu, J. Lin, H. Ao, W. Xu, and J. Shu, "Kraken: memory-efficient continual learning for large-scale real-time recommendations," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–17, IEEE, 2020.

[111] F. Mi, X. Lin, and B. Faltings, "Ader: Adaptively distilled exemplar replay towards continual learning for session-based recommendation," in *Proceedings in the 14th ACM Conference on Recommender Systems*, pp. 408–413, 2020.

[112] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[113] S. Thrun, "Lifelong learning algorithms," in *Learning to Learn*, pp. 181–209, Springer, 1998.

[114] N. Davis, C.-P. Hsiao, K. Yashraj Singh, L. Li, and B. Magerko, "Empirically studying participatory sense-making in abstract drawing with a co-creative cognitive agent," in *Proceedings of the 21st International Conference on Intelligent User Interfaces*, pp. 196–207, 2016.

[115] K. Khetarpal, M. Riemer, I. Rish, and D. Precup, "Towards continual reinforcement learning: A review and perspectives," *arXiv preprint arXiv:2012.13490*, 2020.

[116] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," in *Psychology of Learning and Motivation*, vol. 24, pp. 109–165, Elsevier, 1989.

[117] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer Vision, Graphics, and Image Processing*, vol. 37, no. 1, pp. 54–115, 1987.

[118] M. Riemer, I. Cases, R. Ajemian, M. Liu, I. Rish, Y. Tu, and G. Tesauro, "Learning to learn without forgetting by maximizing transfer and minimizing interference," *arXiv preprint arXiv:1810.11910*, 2018.

[119] P. J. Huber, "Robust estimation of a location parameter," in *Breakthroughs in Statistics*, pp. 492–518, Springer, 1992.

[120] A. Krishnamoorthy and D. Menon, "Matrix inversion using cholesky decomposition," in *2013 Signal Processing: Algorithms, Architectures, Arrangements, and Applications*, pp. 70–72, IEEE, 2013.

[121] N. Tasfi, "Pygame learning environment." `https://github.com/ntasfi/PyGame-Learning-Environment`, 2016.

[122] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[123] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," *Advances in Neural Information Processing Systems*, vol. 30, 2017.