Titre: Title:	The Challenges of Learning Representations for Reinforcement Learning Without Experience Replay
Auteur: Author:	Antoine Clavaud
Date:	2025
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Clavaud, A. (2025). The Challenges of Learning Representations for Reinforcement Learning Without Experience Replay [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/67120/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/67120/
Directeurs de recherche: Advisors:	Sarath Chandar Anbil Parthipan
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

The Challenges of Learning Representations for Reinforcement Learning Without Experience Replay

ANTOINE CLAVAUD
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Juillet 2025

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

The Challenges of Learning Representations for Reinforcement Learning Without Experience Replay

présenté par **Antoine CLAVAUD**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Christopher J. PAL, président
Sarath Chandar ANBIL PARTHIPAN, membre et directeur de recherche
François RIVEST, membre et codirecteur de recherche
Amir-Massoud FARAHMAND, membre

ACKNOWLEDGEMENTS

I would first like to thank the members of my jury, Christopher Pal and Amir-Massoud Farahmand, for accepting to evaluate my thesis and oral defense. I would like to thank my supervisor, Sarath Chandar who accepted to let me join his lab, guided me through my two years of master and gave me insights and feedback so that I could become a better researcher. I would also like to thank François Rivest, my co-supervisor, for the time and effort he gave me so that my work could be at its finest. Mathieu Reymond also played a major role in my master, always being available to help and guide my projects, sharing his experience and cluster priority with me. I would also like to thank Pranshu Malviya for his insights that helped me design better experiments for the paper. Finally I would like to thank all my friends at the Chandar Research Lab who made my time there so enjoyable!

Je souhaite aussi remercier les équipes du département de Génie Informatique et Génie Logiciel de Polytechnique Montréal pour leur accompagnement et aide précieuse pour naviguer les aspects administratifs de ma maîtrise.

RÉSUMÉ

Ce mémoire aborde la question de l'apprentissage de représentations appliqué à l'apprentissage par renforcement, dans le cas particulier où l'on ne s'autorise pas à stocker les expériences rencontrées par l'agent. L'apprentissage par renforcement, aussi appelé Reinforcement Learning (RL), a pour but de résoudre optimalement un problème de décision séquentiel. Ce dernier prend place dans un environnement dans lequel évolue un agent qui peut interagir avec l'environnement au travers d'actions. L'apprentissage par renforcement consiste à trouver une politique d'actions à prendre compte tenu des observations de l'environnement faites par l'agent et qui maximise les récompenses obtenues par celui-ci. Nous nous intéressons ici plus particulièrement au cas dans lequel le stockage d'expériences pour réutilisation future n'est pas autorisé. Une telle contrainte est dénommée *Apprentissage par Renforcement sur un Flux d'Expériences*, ou *Streaming Deep Reinforcement Learning* dans la littérature. Bien qu'à ses origines l'apprentissage par renforcement (alors complètement dépourvu d'apprentissage profond et de réseaux neuronaux) a été étudié dans le contexte de flux d'expériences, ce n'est que très récemment que ce contexte a été ré-introduit dans le paradigme de l'approximation de fonctions par réseaux de neurones. Ce nouveau paradigme permet de se débarrasser de la nécessité d'avoir accès à des ressources informatiques importantes, car sans stockage d'expériences l'entraînement n'a plus besoin d'avoir lieu sur une carte graphique, ou Graphical Processing Unit (GPU). Cependant, le stockage et l'agrégation d'expériences constituent l'une des techniques principales pour réduire l'instabilité rencontrée lors de l'entraînement d'agents via des réseaux de neurones. Aussi, en interdisant le stockage d'expériences, l'instabilité due à l'inhérente non-stationnarité de l'apprentissage par renforcement est fortement exacerbée, ce qui limite les performances des agents.

L'apprentissage de représentations est l'un des domaines majeurs de la recherche en intelligence artificielle. Le but de telles méthodes est d'obtenir des plongements (représentation d'un objet comme une image, un graphe, du texte, etc. sous forme d'un vecteur) de bonne qualité, au travers d'algorithmes et architectures spécifiques, afin de faciliter l'entraînement de réseaux de neurones et d'en améliorer les performances. Ces tâches nécessitent bien souvent d'avoir à disposition des quantités considérables de données, pas forcément étiquetées, et d'avoir accès à d'importantes ressources informatique pour les traiter. Dans le cadre de l'apprentissage par renforcement, de telles techniques ont été utilisées et de nombreux travaux montrent qu'elles augmentent les performances des agents qui les incorporent. L'apprentissage par renforcement se prête particulièrement bien notamment aux méthodes d'apprentissage non-supervisé basées sur la prédiction de la dynamique de l'environnement, c'est à dire basées

sur la prédiction des états futurs de l'environnement étant donnée une séquence d'actions.

Ces méthodes sont bien documentées dans la littérature, mais pas dans le cadre de l'apprentissage par renforcement sur un flux d'expériences. Or, ce cadre étant encore plus instable que l'apprentissage par renforcement profond classique, ajouter de telles méthodes d'apprentissage de représentations semble essentiel à stabiliser davantage l'apprentissage et améliorer les performances des agents. Peu de travaux étudient l'apprentissage de représentations sur un flux de données, et aucun n'est appliqué à l'apprentissage par renforcement. Devant cette situation, nous choisissons d'investiguer dans quelle mesure l'apprentissage de représentations peut être appliqué à l'apprentissage par renforcement sur un flux d'expériences dans le but d'améliorer les performances des agents entraînés dans un tel contexte.

Afin de traiter ce sujet, nous avons décidé de partir d'un agent d'apprentissage par renforcement augmenté par un objectif d'apprentissage de représentations déjà existant et performant, puis de l'adapter au contexte de l'apprentissage par renforcement sur un flux d'expériences. Cette approche consiste à combiner l'architecture appelée *Self-Predictive Representations* (SPR) à un agent entraîné sur un flux d'expériences. Étant donné que SPR n'a pas été conçu pour ce contexte d'apprentissage par renforcement, nous nous attentdions à ce que la forte non-stationnarité du problème soit un obstacle important au succès de notre méthode. Pour cette raison, nous avons décidé de rendre le problème de plus en plus stable au travers d'expériences successives, afin de déterminer quelles seraient les conditions nécessaires de stabilité de cette approche. Surprenamment, nous avons constaté qu'en imposant que la politique soit apprise uniquement sur un flux d'observations, peu importe le niveau de stabilité de la tâche d'apprentisage de représentations les performances des agents sont toutes moins bonnes que notre base de comparaison (agent entraîné sur un flux de données, sans tâche auxiliaire). Nous avons également confirmé que sans cette contrainte d'apprentissage sur un flux d'expériences, la même tâche secondaire mène à l'apprentissage de représentations riches et utiles aux agents. Aussi, nous concluons que bien que dans le cadre classique de l'apprentissage par renforcement ajouter une tâche secondaire d'apprentissage de représentations est bénéfique pour les performances, ce n'est pas le cas lorsque l'on se restreint à entraîner des agents sur des flux d'expériences. En effet, nos résultats suggèrent que l'optimisation jointe des deux tâches interfère négativement avec les performances de l'agent. Ce mémoire présente également les différentes stratégies d'optimisation ayant été considérées, sans pour autant qu'aucune n'amène de meilleures performances.

ABSTRACT

This thesis tackles the problem of learning representations for Reinforcement Learning (RL), in the specific case where we constrain ourselves to not store experiences for later re-use. The goal of RL is to find the optimal solution to a sequential decision problem. Sequential decision problems involve an environment in which an agent can evolve and interact through actions. RL then consists in finding a policy that gives the best action to take given any observation of the environment's state, so that the reward received by the agent is maximized when it follows the policy. Here, we focus more specifically on the case where storing experiences is prohibited, thus making experience replay forbidden. Such a constrained setting is called *Streaming Deep Reinforcement Learning*. Even though in its beginnings RL was studied in this setting as deep neural networks were not part of the field yet, it was only very recently that the streaming setting was reconsidered for modern deep reinforcement learning. This new paradigm makes it possible to train deep RL agent without relying on costly Graphical Processing Units (GPUs) capable of handling large batch sizes and storing large amounts of agent experiences. However, storing and replaying experiences is one of the main methods used to mitigate RL's inherent non-stationary training. Therefore, prohibiting the use of experience replay will lead to less stable training and thus poorer performances.

Representation learning is one of the core aspects in Artificial Intelligence (AI) research. The goal of such methods is to derive good quality embeddings (vectors in a high dimensional space representing objects like images, graphs or words) through algorithms and specific network architectures so that the downstream task learned by a neural network is easier and faster to learn. Representation learning tasks often require large quantities of data to be available, possibly not labeled, as well as large amounts of compute. Many papers have shown that including representation learning tasks to RL objectives yields better performing agents. Reinforcement learning is especially well suited for dynamics prediction-based unsupervised learning tasks. These tasks consists in predicting the next states of the environment given a current state and a sequence of next actions.

These methods are well documented in the literature, but not for *streaming* deep reinforcement learning. However, the streaming context being even less stable than its regular counterpart, adding unsupervised representation learning objectives to it seems like a necessary measure to mitigate instability during training and further improve the field of streaming deep RL. Few works study how to learn representations from a stream of data, and none are specifically focused on RL. Given this rather empty current state of the literature on the

matter, we decided to investigate how we can add unsupervised representation learning techniques from the standard RL literature to the streaming deep RL setting, so as to improve downstream performances.

In order to tackle this question, we decided to use an existing well-performing representation learning augmented RL agent and use it in the streaming setting of reinforcement learning. This approach consists in combining the *Self-Predictive Representations* (SPR) architecture with a streaming agent. Because SPR was not originally designed to be used in the streaming setting of RL, we expected non-stationarity to be a major obstacle to this method's success. Thus, we designed experiments with increasing levels of stability in order to determine what would be the minimal conditions required for a stable representation learning task to be beneficial for the agents. Surprisingly, we found that imposing that the policy be learned in the streaming setting, no amount of stability is ever enough for the representation learning task to help the agent reach better performances than our streaming agent baseline (without any auxiliary task). We also confirmed that without this streaming constraint, having the same representation learning objective helped agents learn better representations, leading them to reach better performances. As such, we concluded that although in standard reinforcement learning having a secondary representation learning objective improves performances, it is not the case for streaming agents. Indeed, our results suggest that jointly optimizing both tasks gives rise to interference that hinders the performance of the agents. We also present the many considerations we had regarding the optimization part of the problem, although none of the ones we tried helped getting better performance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

RL	Reinforcement Learning
GPU	Graphical Processing Unit
SPR	Self-Predictive Representations
AI	Artificial Intelligence
MDP	Markov Decision Problem
BYOL	Bootstrap Your Own Latents
MSE	Mean Squared Error
SimCLR	Simple Contrastive Learning of Representations
i.i.d.	Independent and Identically Distributed
ObGD	Overshooting-bounded Gradient Descent
SAC	Soft Actor-Critic
DDPG	Deep Deterministic Policy Gradient
TD3	Twin Delayed Deep Deterministic Policy Gradient
CURL	Contrastive Unsupervised representations for Reinforcement Learning
A-GEM	Averaged Gradient Episodic Memory
DQN	Deep Q-Networks
IQM	Inter-Quantile Mean
FS-SPR	Fully-Streaming Self-Predictive Representations
EMA	Exponential Moving Average
SAM	Sharpness Aware Minimization

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

The last decade has seen deep learning revolutionize many fields, from natural language processing [1] to image processing [2] and new crystalline material generation [3]. Reinforcement Learning (RL) has also majorly benefited from the deep learning wave, for instance in its ability to solve games and tasks with an ever-increasing complexity [4, 5]. This leap in abilities was made possible by the advent of deep reinforcement learning which leverages deep neural networks to learn intricate policies. Unfortunately, using deep neural networks with reinforcement learning introduces instability and requires dedicated methods such as the use of replay buffers and target networks to deal with them. Along with the use of deep neural networks, these methods have a heavy computational footprint as they require dedicated hardware (Graphical Processing Units or GPUs). This is however a strong requirement that is not met for all tasks RL can be applied to, such as on-device training for robotics where the robots themselves have to run RL algorithms on their limited hardware, making learning very slow.

This is the motivation that recently led researchers to study *streaming* deep reinforcement learning [6, 7], which does not require any GPU to run. Indeed, *streaming* RL adds the requirement that RL algorithms cannot store experiences to reuse them later, getting rid of large memory requirements and large batch sizes for gradient updates. However, the speed benefits come at the cost of a drastically increased instability of the training process. A common method used in standard reinforcement learning to alleviate part of the instability is to add a secondary unsupervised learning task for learning better representations. This has however never been applied in the case of streaming deep RL, where it could be greatly beneficial as it could improve stability and sample efficiency (agents learn faster). Adding a secondary, unsupervised, learning task to the context of streaming RL is not an easy feat as very few methods are designed for learning from a stream of inputs, and none are applied to streaming RL. Through this thesis, we aim to fill this gap and further our knowledge of streaming representation learning applied to (streaming) reinforcement learning.

1.1 Definitions and main concepts

Let us first introduce in more details the specific setting of reinforcement learning in which we base all of our work. This section contains high-level overviews of the main concepts required to understand our problem and contributions. It covers the topics of RL and representation learning and how the later can be used in the context of the former. A deeper dive into the

mathematical formulations of both RL and representation learning will be provided in the literature review (Chapter 2) as well as in the background section of the paper (section 4.4).

Reinforcement Learning Reinforcement learning, or RL, is one of the many fields of AI research. Reinforcement learning tackles the problem of solving sequential decision problems optimally. The usual mathematical framework used to describe such problems is that of the Markov Decision Problem (MDP) [8]. Sequential decision problems involve an environment, in which an agent evolves. The agent can interact with the environment through actions. Once an action has been taken by the agent, the state of the environment is updated to reflect the result of that action. For instance, in a grid-like world where the agent can be in any cell of the grid, after taking the action to "move to the right", the environment state is updated so that the agent now occupies the cell on the right of its previous location. Each transition of the environment also comes with a reward signal. This reward is a relative feedback on the new state reached by the agent, but the agent has no knowledge of whether it used the best action (the one leading to the maximum reward) nor even of which previously taken actions contributed to the observed reward and to what extent. This makes the problem much harder to solve. In this context, the agent is trained to learn an action policy which, when followed by the agent, maximizes the cumulative (possibly discounted) reward the agent gets along its trajectory.

Thus, contrary to supervised learning, reinforcement learning consists in solving a task without having access to the "correct" answers (best actions, equivalent to the label of a data point in supervised learning). This task is also a non-stationary one, partly because the experiences collected by the agent depend of the actions it takes. Since the policy dictating the actions to take is learned as the experiences are collected, the distribution induced by this collection of training data varies through time. Moreover, the regression objective used by most so-called value-based methods (methods that predict the discounted sum of future rewards from a given state) is recursively defined as a function of the prediction of the value of the next states. Therefore, the training objective depends on the current agent parameters, which are updated after each training step, thus making the regression objective non-stationary.

Representation Learning Representation learning is a very broad term that can technically be used to describe any deep learning method. Indeed, representation learning refers to any process that produces a mapping from a "real-world object" like an image, a graph or text to a high-dimensional vector space. However, in recent years, more effort has been put towards learning such embeddings so that they also preserve some form of semantic structure. This can be achieved through both the loss function and the network architecture. For ins-

tance, the famous Bootstrap Your Own Latents (BYOL) [9] method uses a clever architecture to impose that similar inputs will yield similar representations, namely this method trains a network f_θ such that if two inputs x_1 and x_2 are semantically similar (they describe very similar objects, or the same object in slightly different ways), their representations should be close too according to a similarity metric of our choice. Such a metric can for instance be the Mean Squared Error (MSE) or the cosine similarity, which consists in computing the angle between two vectors regardless of their magnitude. In the case of BYOL, the authors choose to maximize the cosine similarity between two slightly modified (augmented) versions of the same input. In the case of images, an augmentation can be a small random translation or rotation, adding gaussian blur, slightly transforming the colors of the image, or any other transformation that modifies the pixel content of the image without altering its semantic content. Another famous technique, called Simple Contrastive Learning of Representations (SimCLR) [10], makes use of a different loss function to achieve similar results. In SimCLR's case, the authors use a contrastive loss, meaning they want to maximize the similarity of the representations of a pair of similar examples, just like BYOL does, as well as to minimize the similarity between examples of this pair and negative examples. Negative examples are usually a set of inputs corresponding to objects unrelated to the target positive example. In both cases, the representations learned have the added benefit that inputs that are close in the input space should have representations that are relatively close in the learned latent space. Both methods require very large batch sizes, especially in the case of SimCLR, which also requires a large memory buffer of negative examples.

Representation Learning for Reinforcement Learning In the case of reinforcement learning, both representation learning approaches described in the previous paragraph have been applied in different papers [11, 12], presented in more details in the literature review. However, the BYOL approach is more widely used as it does not rely on providing negative examples, which makes the algorithm more adaptable. In both cases, an interesting idea to exploit is the temporal consistency of representations along a trajectory. Indeed, given that for successive actions the state of the environment should not drastically change, we can exploit this property in the observation space and enforce it in the latent space of the representations learned. This implies that one must first learn a model to predict the next states the agent might encounter after taking given actions. This is also known as learning a model of the dynamics of the environment and is an important part of model-based RL [4, 13]. We can then use this dynamics model to predict the future states the agent will encounter and train our unsupervised representation learning objective to push the representation of each state to be close to that of the next predicted states. This is the main idea behind the Self-Predictive

Representations (SPR) paper [11], which is the representation learning method that we use in the experiments performed in the paper.

1.2 Problem statement

As explained in the previous section, reinforcement learning is a hard optimization problem to solve in of itself as the distribution of inputs used for training is not Independent and Identically Distributed (i.i.d.), and the regression objective is non-stationary.

Dealing with the Instability These two assumptions about the i.i.d. nature of the inputs and the stationarity of the regression objective are critical for supervised learning to perform well. To get around them, the RL community came up with target networks and replay buffers. Target networks deal with the non-stationarity of the regression target by keeping a copy of the network weights that is updated much less frequently, or by a much smaller amount through Polyak averaging. This target network is used exclusively to compute the regression target during each training step of the agent. Since the target network's weights evolve at a much slower pace than that of the agent's network, the regression target also changes at a slower rate, effectively making the regression task appear more stable, at the price of an agent reacting slower to changes. Replay buffers tackle the non-i.i.d. nature of the training input distribution by storing a large number of previous transitions. Agents are then trained from transitions sampled from the replay buffer rather than the highly correlated last transitions it encountered. Since the buffer can be quite large (usually containing between 10^5 and 10^6 elements), states encountered from a larger number of different trajectories will be sampled, making the training input distribution more i.i.d. Moreover, using a batch rather than a single transition to perform learning updates improves the stability of the estimated gradient.

Streaming Reinforcement Learning Unfortunately, the streaming setting of reinforcement learning, such as described by [6] prohibits the use of both replay buffers and target networks. This forces us to get rid of all the aforementioned instability mitigation strategies. Although the authors introduce new stabilization mechanisms through the use of LayerNorm [14], eligibility traces [8], observation and reward normalization, as well as through the use of their custom optimizer, Overshooting-bounded Gradient Descent (ObGD), the problem setting is still unstable. ObGD and the algorithmic details of the streaming agent we based our work on are described in greater details in the literature review (section 2) and technical appendix of the paper (sections 4.9.1 and 4.9.2). Figure A.2 presents the detailed

architecture of the network used, divided into the SPR part and the Q-learning part. Moreover, getting rid of the replay buffer hinders sample efficiency as each newly experienced transition is only used once for training before being discarded.

Streaming Representation Learning Many works have shown that adding an unsupervised representation learning task helps standard RL agents achieve better performance [11, 12, 15–19]. This has been especially tested on the Atari benchmark that we are using. As such, it is our intuition is that the same should hold true in the streaming setting. Additionally, better representations should help make the problem more stable as semantically similar states would share similar representations, therefore reducing the impact of the non-i.i.d. characteristic of the training input data, at least in the latent space. However, as explained in the previous section, unsupervised learning tasks usually require very large batch sizes to average gradients over a large number of input augmentations, so as to reduce noise in the updates. Not only is this not possible in the streaming setting, the gradients used at each update are also highly correlated since they come from temporally close transitions. Translating standard representation learning tasks to the realm of streaming RL is thus not a straightforward process and may not even be suited for this harder setting.

Optimization Problem Finally, a very important part of this process is the consideration given to the optimization problem. Indeed, in our architecture (see Figure A.1 in Appendix A), the parameters from the Q-learning part of the network are updated using eligibility traces, which can be interpreted as a first order momentum of the previous gradient updates, whereas the parameters from the representation learning part of the network are updated using Adam-like [20] updates. Since our architecture uses shared parameters between the two parts of the network, the question of how to combine these two different sources of update arises. Multiple aspects must be considered : the relative norm of each updates, the potential interference caused if the gradients are not orthogonal and finally the optimizers to use for each part of the network.

1.3 Objectives of the research

Objectives The main goal of the research presented in this thesis is to fill the current gap in the literature on using representation learning in the context of streaming deep reinforcement learning. To do so, our first objective is to benchmark the performance of a straightforward translation of the SPR paper to the streaming RL setting. This benchmark should come with a detailed analysis of the successes or failures of this first attempt at combining representation

learning and streaming deep RL. Another important focus of this research was to determine the most adapted way to define the optimization problem, and study the impact of the different strategies used on performance.

Contributions Our contributions are two-fold. First we propose the first empirical study of the question of learning streaming representations for reinforcement learning. To that end, we performed a series of experiments in which we gradually removed some of the instability of the problem to determine the minimally stable setting in which we can still learn representations. After realizing that whenever a streaming RL agent was used, adding a secondary representation learning objective always led to poorer performances, we also decided to determine the reasons why learning streaming representations was so detrimental to streaming RL. Second, we will release the code we wrote for the benchmark so that the community can build upon this first work and develop better streaming representation learning algorithms for RL.

Most of the contributions in this thesis stem from our paper, *The Challenges of Learning Streaming Representations for Reinforcement Learning* :

- Authors : **Antoine Clavaud**, Mathieu Reymond, François Rivest, Sarath Chandar.
- Submitted at The Thirty-Ninth Annual Conference on Neural Information Processing Systems, **NeurIPS 2025**.
- Contributions : I led the project from the literature review to the writing of the code for experiments as well as the writing of the paper. Mathieu closely followed the project through our biweekly meetings, helping me to come up with interesting experiments ideas, run experiments on his cluster allocation and improve my writing for the paper through multiple proof-reading sessions. François and Sarath provided valuable guidance and higher-level planning for the project during our weekly meetings, as well as a rich feedback to improve the writing of the paper and the overall submission process.

1.4 Plan of the thesis

To answer these questions, we propose to start by presenting the relevant background and existing papers to better understand the theoretical roots of our work. This literature review is meant as both a background as well as an extension of the literature review present in the paper, as it goes into more details for each presented paper. Then, we present our methodology and results through our paper named *The Challenges of Learning Streaming*

Representations for Reinforcement Learning. This paper focuses on showing that learning streaming representations does not help streaming RL agents reach better performances, and understanding why that is. Finally, we go over the limitations of our approach and discuss potential directions that would be interesting to explore to deepen our understanding of learning streaming representations for RL. The appendix only contains visuals of our network architecture for easier understanding.

CHAPTER 2 LITERATURE REVIEW

This section will first cover RL related background needed to understand Chapter 4. The RL part of this background is very similar to the one in section 4.4. Second, this literature review will expand on the one present in our paper. It will delve into the most important methods to provide a clear understanding of either the core concepts and scientific background our work is based upon or the future directions our work proposes.

Moreover, this section assumes that the reader already has an understanding of neural networks and how they are trained. However, no preliminary knowledge about reinforcement learning or representation learning is needed.

Background As mentioned in the introduction, Reinforcement Learning tackles solving sequential decision problems, usually framed as Markov Decision Problems (MDPs) [8]. A MDP is described as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. $\mathcal{S}$ represents the set of all environment states and $\mathcal{A}$ the set of all actions that are available to the agent. $\mathcal{P} : \mathcal{S}, \mathcal{A} \to \mathcal{S}$ encompasses the (stochastic) transition dynamics of the environment, $\mathcal{R} : \mathcal{S}, \mathcal{A} \to \mathbb{R}$ is the reward function and γ is the discount factor. The agent can interact with the environment through a policy $\pi : \mathcal{S}, \mathcal{A} \to [0, 1]$ which gives the probability of taking each available action in any given state. At each time step t, the agent receives the current state as input $s_t \in \mathcal{S}$ and takes the action $a_t \sim \pi(\cdot|s_t)$. The environment state is updated following the transition function $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$ and gives a feedback to the agent in the form of a reward $r_t = \mathcal{R}(s_t, a_t)$. We can define the episodic return G_t as the summation of the discounted rewards obtained by an agent along a trajectory following a policy π and starting from timestep t. $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$, where T denotes the time step at which the episode terminates. We can further define the value function $V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s]$ which evaluate the value of being in a specific state based on the expected return an agent should get when following its policy π starting from state s. Thus, the goal of RL to find the optimal policy can be defined as finding $\pi^* = \text{argmax}_\pi\{V_\pi(s_0)\}$, where s_0 follows the initial state distribution of the environment, $\mathcal{D}_0$. Usually, we define $J(\theta) = \mathbb{E}_{s_0 \sim \mathcal{D}_0}[V_\pi(s_0)]$ as the performance of a policy π.

Deep Q-Learning uses Deep Q-networks (DQN) [21] to solve this optimization problem in the following way. First let us define the state-action value function, or Q-value $Q_\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a]$. The Q-value obeys the Bellman optimality equation [8] :

$$Q_{\pi^*}(s_t, a_t) = \mathbb{E}_{s \sim \mathcal{P}(.|s_t, a_t)} \left[r_t + \gamma \max_a \{Q_{\pi^*}(s, a)\} \right] \tag{2.1}$$

Deep Q-Learning exploits this fixed-point equation to learn an approximation of the Q-function, $Q_\theta \simeq Q_\pi$ parameterized by a neural network with weights θ. DQN then simply consists in updating the parameters θ in the direction of the semi-gradient of the following loss function, where $\texttt{SG}$ denotes the *Stop Gradient* operation :

$$\mathcal{L}(\theta) = \Big(Q_\theta(s_t, a_t) - r_t - \gamma \cdot \texttt{SG}(\max_a\{Q_\theta(s_{t+1}, a)\})\Big)^2 \tag{2.2}$$

This training process can be very unstable because of the non-stationarity of RL, as explained in the introduction. To stabilize training, it is very common to use experience replay through the use of a replay buffer and target networks. A target network is a slowly evolving copy $\theta^-_{t+1} = \tau\theta_t + (1-\tau)\theta^-_t$ of the main network's weights θ_t that is used to compute the bootstrapped Q-Learning objective. Its slower dynamics help stabilize the training process as the regression objective doesn't change so drastically through time. τ is a hyperparameter defining the update rate of the target network. DQN also uses a replay buffer that stores every transition encountered by the agent (up to a size of 10^5 to 10^6 before overwriting the first transitions stored). Gradients are then computed on batches of transitions, sampled from the replay buffer, that are much less correlated than if they were sequentially generated. This helps reduce the non-stationarity of the input distribution of the deep Q-network. The loss resulting from these two optimizations is the following :

$$\mathcal{L}(\theta_t) = \frac{1}{N}\sum_{k=1}^{N}\Big(Q_{\theta_t}(s_t^{(k)}, a_t^{(k)}) - r_t^{(k)} - \gamma \cdot \texttt{SG}(\max_a\{Q_{\theta_t^-}(s_{t+1}^{(k)}, a)\})\Big)^2 \tag{2.3}$$

Streaming Deep RL This section is very similar to its counterpart in the technical appendix of our paper (section 4.9.2). Before introducing the streaming RL agent that we have been using throughout this thesis, it is worth mentioning that at its origin, RL was framed in a purely streaming fashion. The first RL algorithms [8], SARSA, Temporal Difference Learning (TD) and Q-Learning were indeed designed without replay or target. However, these algorithms were not designed with deep neural networks in mind. As such, let us introduce the streaming deep RL agent that we used throughout this thesis : the Stream-Q agent from *Elsayed et al.* [6]. We also used their definition of the streaming setting of RL. As explained previously, the streaming setting of reinforcement learning definition we used mandates that any transition be discarded before the next one is experienced by the agent. This prohibits the use of replay-buffers and the authors further choose to not use target networks either. This setting of RL is much harder as it a lot less stable. The streaming agent *Stream-Q* proposed by *Elsayed et al.* [6], which we used as our streaming baseline, combines several architectural

and algorithmic choices carefully selected to improve sampling efficiency as well as training stability (through both normalization techniques and better optimization). To that end, the Stream-Q agent uses eligibility traces (see background in section 4.4), which can be seen as a first order momentum of gradients. Eligibility traces allow to perform credit assignment faster through a clever alternative type of returns called the λ-return. Stream-Q also uses a sparse initialization as it has been shown to reduce forgetting and to be beneficial for RL [22]. This initialization means that 90% of the agent's weights are initialized to zero, which can help the agent learn different behaviors for different situations with minimal interference from updating any of them. Stream-Q computes the running averages and standard deviations of both observations and rewards so as to normalize (and center) them before feeding them to the agent's network. The architecture of Stream-Q's network also makes extensive use of parameter-free LayerNorm [14] (see equation 4.2) as all pre-avtications are first passed through a parameter-free LayerNorm. Finally, one of the most significant contributors to Stream-Q's success is the custom optimizer introduced by its authors : Overshooting-bounded Gradient Descent (ObGD). This optimizers dynamically lowers the learning rate of an update if it is too large from the standpoint of a stability criterion. In the case of ObGD, the stability criterion used is the effective step-size defined by [23] as :

$$\xi = \frac{\delta(s_t) - \delta_+(s_t)}{\delta(s_t)} \tag{2.4}$$

where $\delta(s_t)$ is the TD-error and $\delta_+(s_t)$ is the TD-error on the same state after having updated the network parameters according to $\delta(s_t)$. The effective step size measures the amount of error that has been corrected by an update. As such, having $\xi > 1$ means that the agent over-corrected itself, which could lead to unstable behavior if this happened often. Therefore, an update is considered unstable if $\xi > 1$ as it is preferable to avoid this situation. *Elsayed et al.* [6] then come up with the following upper bound for the stability criterion : $\xi \leq \alpha\kappa\bar{\delta}_t||\mathbf{z}_t||_1$, where α is the step size, $\kappa > 1$ is an hyperparameter of ObGD, acting as a security coefficient. $\bar{\delta}_t = \max(1, |\delta(s_t)|)$ and $\mathbf{z}_t$ is the eligibility trace used in the update. From this, they derive the following maximum learning rate value that satisfies the stability condition : $\alpha_{\text{MAX}} = (\kappa\bar{\delta}_t||\mathbf{z}_t||_1)^{-1}$. For a target step-size α^*, ObGD computes the highest stable learning rate using the update rule in equation (2.5). In the best case scenario, the target learning rate α^* is used, otherwise the learning rate is down-scaled to ensure the update is stable.

$$\theta_t \leftarrow \theta_t + \min\left(\alpha^*, \frac{1}{\kappa\bar{\delta}_t||\mathbf{z}_t||_1}\right) \cdot \delta(s_t)\mathbf{z}_t \tag{2.5}$$

The adaptive step-size mechanism provided by ObGD is especially useful in the streaming setting since gradients are much noisier because of the absence of batches, making large updates in the wrong direction all the more likely.

However, Stream-Q is not the only deep streaming RL method. Indeed, *Vasan et al.* [7] propose a policy-based deep streaming agent that can be seen as a more stable version of a modified streaming Soft Actor-Critic (SAC) agent [24]. The Actor-Critic family of policy gradient methods is characterized by the fact that instead of training one agent, we train one actor (the policy network) and one critic (regular value- or Q-network) together. The actor takes actions based on the state and the critic is used to evaluate the choice of the actor and sway it towards taking better valued actions during the parameter update phase. The approach from *Vasan et al.* [7] is based on the reparameterization gradient theorem that gives an expression of the gradient the policy must follow to guarantee improvements in performances. Unlike the policy gradient theorem, stated as $\nabla_\theta J(\theta) \propto \mathbb{E}_{S\sim d_\pi, A\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(A|S) q_{\pi_\theta}(S,A)\right]$ where d_π represents the stationary state distribution of the policy π_θ, the reparameterization gradient theorem provides another unbiased estimate of $\nabla_\theta J(\theta)$ through a different sampling procedure. Indeed, the reparameterization gradient theorem consists in defining the following action sampling procedure for training the actor network : $A = f_\theta(g(\xi))$ where $\xi \sim \mathcal{N}(0, \mathbf{I})$ is independent of θ and g maps ξ to a more complex probability distribution. This streaming policy-gradient method also includes an entropy regularization term to its gradient in order to improve exploration and smoothen the loss landscape in some cases [25], giving the following update rule : $\nabla_\theta J(\theta) \propto \mathbb{E}_{S\sim d_\pi, A\sim\pi_\theta}\left[\nabla_\theta f_\theta(\xi; S) \nabla_A (q_{\pi_\theta}(S,A) - \eta \log \pi_\theta(A|S))\right]$. Similarly to the Stream-Q agent, this paper makes use of observation normalization, but normalizes the TD-errors rather than the rewards and uses penultimate normalization [26] instead of LayerNorm [14]. Penultimate normalization simply consists in normalizing the features produced by the penultimate layer of a network as such : $\hat{\psi}_\theta(x) = \psi_\theta(x)/||\psi_\theta(x)||$

Representation Learning for RL The representation learning technique we used throughout this thesis is called Self-Predictive Representations (SPR) [11]. SPR is based on the Bootstrap Your Own Latents (BYOL) architecture from unsupervised learning. It also involves learning a model of the dynamics of the environment to be able to iteratively predict the next $K = 5$ states of a trajectory given a current state and the next K actions taken. It is important to note that this prediction is performed in the latent space, meaning the dynamics model predicts a latent representation of the next states, not the states directly. This is important since forcing the model to learn to predict the entirety of the next states makes it learn how to predict features of the observations that are irrelevant to the agent (noise, aesthetic user interface elements, etc.). Thus, the dynamics model learns to predict

$\hat{\mathbf{z}}_{t+k+1} = d_\phi(\hat{\mathbf{z}}_{t+k}, a_{t+k})$ where $\hat{\mathbf{z}}_t = f_\theta(s_t)$ (for $k = 0$) is the actual latent representation of the state s_t. Since SPR is based on BYOL, every observations (states consisting of stacked images of game frames) are augmented with simple transformations such as small translations and light intensity rescaling. Then, following the BYOL architecture, two linear layers p and q (respectively called prediction and projection layers) are applied to these latent states, such that we have $\tilde{\mathbf{z}}_{t+k} = q(p(\hat{\mathbf{z}}_{t+k}))$. Finally, through a cosine similarity loss, SPR attempts to bring closer together the representations of the predicted next latent states and their associated ground truth. In other words, SPR tries to minimize the following loss where $\texttt{SG}$ denotes the *Stop Gradient* operation :

$$\mathcal{L}_{\text{SPR}} = -\sum_{k=1}^{K} \text{sim}(\tilde{\mathbf{z}}_{t+k}, \texttt{SG}(p(\hat{\mathbf{z}}_{t+k}))) \quad \text{where} \quad \text{sim}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\top \mathbf{v}}{||\mathbf{u}|| \cdot ||\mathbf{v}||} \quad \text{is the cosine similarity} \tag{2.6}$$

It is important to note that the computation of the $p(\hat{\mathbf{z}}_{t+k})$ term are performed without propagating gradients. In the SPR paper, the authors also propose a version of SPR without image augmentations, but a target encoder network instead, updated via an exponential moving average operation. Since the streaming setting of RL does not allow target networks, we stuck to the SPR version with image augmentations and no target networks.

SPR is not the only representation learning method applied to RL using a dynamics model. Indeed, several other papers [15, 17, 19] build and improve on the SPR architecture. Each one of these paper uses a very similar representation learning objective based on maximizing the cosine similarity of the representations of temporally close states. *Zhao et al.* [15] focus on applying the SPR's representation learning objective to a Deep Deterministic Policy Gradient (DDPG) [27] agent for state-based environments from the DeepMind Control suite [28]. In addition to predicting the next latent representations, they also predict the next rewards associated to these next transitions. *Scannell et al.* [17] tackle a similar problem but choose to use a quantized latent space to prevent representation collapse, as well as the Twin Delayed Deep Deterministic Policy Gradient (TD3) [29] algorithm. *Fujimoto et al.* [19] introduce Mr.Q and use a similar approach to learn both state and state-action representations with the difference that they use an MSE loss rather than a cosine similarity and that they also predict the next rewards and terminations. For stability purposes, the authors also normalize the rewards in their update equations and use a categorical loss for the reward prediction task. If all these methods are so similar in their representation learning objective, it is because the main requirement for representation learning tasks applied to RL is that the learned encoder should be self predictive [18], meaning that it should be able to predict its next

latent states. [18] show that even the simplest possible self-predictive representation learning framework delivers great performances for RL.

However, BYOL-based self predictive representations is not the only method used as a secondary objective in RL tasks. Indeed, some methods are based on the SimCLR method, which uses the following contrastive loss :

$$\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [l(2k-1, 2k) + l(2k, 2k-1)] \tag{2.7}$$

$$\text{where} \quad l(i,j) = -\log \left(\frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}{\sum_{k=1, k\neq i}^{2N} \exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_k)/\tau)} \right), \quad \text{sim}(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\top \mathbf{v}}{||\mathbf{u}|| \cdot ||\mathbf{v}||}, \tag{2.8}$$

and $\mathbf{z}_i$ is the representation of the i-th state in the batch. Here, the similarity function is also the cosine similarity, just as in BYOL. The "positive" example pair consists of two different augmentations of the same image, and the negative examples are the other images in the batch. In this case, we consider that the positive examples from a pair are placed next to one another in the batch, leading to their indexes being $2k$ and $2k-1$ in the general case. Contrastive Unsupervised representations for Reinforcement Learning (CURL) [12] adapts this loss function to the realm of reinforcement learning, and the images being augmented are the states of the environment sampled from a replay buffer. Since the states are sampled from a large enough replay buffer, every state should have low similarity with each other, guaranteeing the required property of negative examples. However, CURL does not harness the temporal consistency property of states and their representations along agent trajectories as it does not learn a model of the dynamics of the environment.

Another interesting way to learn representations for RL is through learning successor representations [30–33]. Successor representations are based on the assumption that the rewards can be decomposed as a linear function of (non-linear) state representations, or features, $\phi_\xi(s_t)$ and a task-specific vector w, such that $R_t = \phi_\xi(s_t)^\top w$. This can be used to define the Q-value function as $Q(s_t, a_t) = \psi(s_t, a_t)^\top w$ where $\psi(s_t, a_t)$ can be interpreted as the discounted occupancy measure of all future state features : $\psi(s,a) = \mathbb{E}_\pi \left[\sum_{k=t}^{\infty} \gamma^{k-t} \phi_\xi(s_k) | s_t = s, a_t = a \right]$. In practice, we learn a parameterized approximation $\hat{\psi}_\theta(s,a)$ of $\psi(s,a)$ in addition to the state features $\phi_\xi(s)$. Fortunately, ψ and ϕ verify a Bellman equation, which makes learning their approximations through deep neural networks possible, but not stable. Indeed, this type of technique is often subject to representation collapse, as $\phi(.) = (1-\gamma)\psi(.) =$ constant is a valid solution. Thus, as is the case with BYOL, some tricks must be used to prevent collapse.

In the case of successor features, this can be done by adding a secondary objective based on the prediction of the reward.

Learning Streaming Representations Until now, we have presented representation learning techniques that were designed to learn from large batches of data sampled from an even bigger replay buffer. Since this is not allowed in the streaming context, representation learning techniques must be designed consequently. This can range form the design of the architecture to the optimization method used during training. Not many works study this very challenging problem.

We can first look at Continual Learning methods designed to learn better representations in more challenging non-stationary prediction tasks. Continual learning consists in training a network without interruption on different changing tasks, meaning that the agent must be able to learn new tasks as they come and remember its training as these tasks may not come back before a long time. The two main challenges of continual learning are as follows : the agent must be able to learn new tasks no matter how many tasks it has previously learned. Moreover, the agent must remain good at solving every task it has seen so far. Learning a new task should not make it lose performance on previous tasks. These two challenges are respectively known as loss of plasticity [34] and catastrophic forgetting [35]. Although the paradigm of continual learning is closer to the streaming setting of reinforcement learning, it is not constrained to discarding replay buffers. Indeed, one of the main methods used to solve continual learning tasks is the use of replay buffers [36]. Recently, generative replay has been used to tackle the streaming setting of continual learning [37] and of reinforcement learning [38]. This type of replay uses a generative network to produce realistic examples following the distribution of the last encountered data.

Another interesting continual learning technique is to project the gradient from the current training step ($\mathbf{g}_t$) orthogonally onto the gradient computed on an assortment of tasks sampled from a replay buffer ($\bar{\mathbf{g}}$). This method is called Averaged Gradient Episodic Memory (A-GEM) [39] and is defined as such :

$$\tilde{\mathbf{g}}_t = g_t - \frac{\mathbf{g}_t^\top \bar{\mathbf{g}}}{||\bar{\mathbf{g}}||^2}\,\bar{\mathbf{g}} \quad \text{if} \quad \mathbf{g}_t^\top \bar{\mathbf{g}} \leq 0 \tag{2.9}$$

The intuition behind this method is that this projection step should prevent the current gradient updates to make the model become worse at the first tasks it encountered. This idea of orthogonal projection of gradients has been re-used by [40] to learn video representations from a stream of data. Given that in this context, two consecutive frames from a video should

be very similar, the gradients induced by these images will also be very correlated. These gradients are then used to update weights of the network and their high correlations can lead to overfitting and poor performances more generally. The authors argue that since in the i.i.d. setting, gradients are usually not very correlated, using equation 2.9 with $\bar{\mathbf{g}} = \mathbf{g}_{t-1}$ and disregarding the condition on the sign of $\mathbf{g}_t^\top \bar{\mathbf{g}}$ should bring the learning dynamics of the streaming representation learning task closer to the dynamics in the i.i.d. case. In practice, if successive gradients are very aligned, their projections will get gradually smaller, until they lead to negligible updates, thus avoiding to overfit to a single over-represented example. Conversely, in the i.i.d. setting, the projection correction does not change the gradients by a lot and the learning dynamics are preserved.

Learning good representations from a stream of data is not an easy task, but through careful optimization choices, it can be made more achievable. However, our current streaming representation learning techniques are not good according to [41]. Indeed, this paper shows that random, fixed, representations match the performance of state of the art continual representation learning methods, even outperforming them in some cases. Even though their "learned" representations are random and not trained, the choice of architecture has been carefully studied to provide a pipeline that allows to linearly separate most classes efficiently, explaining the success of their method. This final paper of the literature review illustrates very well the current state of the research on streaming representation learning : we are still at the very beginning and there are no "magic" methods that work well in most cases as is the case for supervised learning. Many factors such as network architecture or optimization technique to use have to be considered to devise a new streaming representation learning method.

Conclusion As such, the state of learning streaming representations specifically tailored for reinforcement learning is very dire and under-explored. To our knowledge we are the first ones to study the adaptation of representation learning techniques to the realm of streaming deep reinforcement learning. It is however essential for streaming deep reinforcement learning applications as it could provide a way to improve training stability and sampling efficiency as has been the case for standard deep reinforcement learning.

CHAPTER 3 GLOBAL APPROACH OF THE RESEARCH AND GENERAL ORGANIZATION OF THE DOCUMENT

As mentioned in the introduction, the main contributions of this thesis are the ones contained in our paper, *The Challenges of Learning Streaming Representations for Reinforcement Learning*. This thesis extends on the literature review from said paper since the conference page limit didn't let us delve in enough details in the papers most relevant to our research. Moreover, Chapter 5 is meant as an extension of the discussion part of our paper.

It is important to mention that we decided to include the full version of the paper, along with its technical appendix. The paper and technical appendix both contain some background elements that are redundant with the introduction and literature review sections of this thesis. Namely, some of the paragraphs in the background (section 4.4) as well as the Stream-Q overview in the appendix (section 4.9.2) can be skipped for a smoother reading experience.

CHAPTER 4 ARTICLE 1 : THE CHALLENGES OF LEARNING STREAMING REPRESENTATIONS FOR REINFORCEMENT LEARNING

This paper was submitted to the NeurIPS 2025 conference on the 10^{th} of May 2025.

Authors : Antoine Clavaud, Mathieu Reymond, François Rivest, Sarath Chandar.

4.1 Abstract

An interesting feature of adaptive systems is their ability to learn in real time from a continuous stream of experiences and update their behavior as they interact with their environment. Deep Reinforcement Learning (RL) aims to mimic this ability using deep neural networks and representation learning techniques, but is inherently unstable and does not perform well in streaming settings. Recently, new works have shown that it is possible to adapt deep RL to the streaming context borrowing plasticity tricks from the continual learning literature. Their focus is mostly on the optimization part of the problem and leave leveraging representation learning or unsupervised signals mostly unexplored. In this paper, we investigate how representation learning can be integrated in a streaming RL pipeline in order to increase sample efficiency. We find that naive approaches do not work and perform thorough experiments isolating non-stationarity from other sources of instability. We show that the increased non-stationarity induced by the streaming setting is not the sole factor preventing good streaming representations to be learned. Rather, we show that self-predictive representations are fundamentally incompatible with the streaming context.

4.2 Introduction

At its beginning, Reinforcement Learning (RL) was theorized in a streaming and tabular setting [8]. An agent would store its value estimates in a table the size of the state-action space and would be able to infer the optimal policy given enough samples. This paradigm works relatively well for small problems like grid worlds, but completely breaks for more complex environments, where storing values for the whole state-action space is impossible. To tackle more realistic and challenging environments with exponentially large state and action spaces, ranging from controlling actuators in a simulated robot to playing Atari games from pixel input, the next innovation was to harness the generalization capabilities of deep neural networks [42] and use them to approximate Q-functions, value-functions and policies.

However, with great representational power comes great instability. Training deep Q-networks (DQN) [21] for instance can be very unstable and requires a number of tricks (double Q-learning [43], target networks, replay buffers, batch updates) to make the process smoother and less hyperparameter-dependent. Other methods like Rainbow [44] or Soft Actor-Critic (SAC) [24] greatly improve on vanilla DQN at the cost of making the problem further away from the streaming setting as they rely on big replay buffers and large batch sizes. Throughout this paper, we consider as *streaming* RL the setting used by [6], in which no transition can be stored to be exploited later (meaning no replay buffer or batches of size greater than one), and no target networks are used. Classical RL methods can also be more sensitive to hyperparameter tuning (PPO famously requires 37 implementation tricks to work well across multiple benchmarks [45]) and sadly tend to perform rather poorly when trained in a streaming setting. This is problematic because it makes current RL algorithm dependent on powerful and costly hardware to be trained successfully, which is not compatible with on-board learning or systems with resource constraints.

Recently, deep streaming RL has been proven to be a possible alternative [6, 7] to some traditional algorithms, like DQN or PPO, while not relying on batch updates or replay buffers. To achieve a similar range of performances without the usual mitigations for non-stationarity, the *Stream-X* family of algorithms relies on careful architectural and optimization considerations designed to improve training stability. They also use eligibility traces to improve sampling efficiency. Nevertheless, this new deep streaming RL paradigm does not take advantage of the recent deep, non-streaming, RL methods, based on learning better representations for RL.

The core idea of representation learning techniques for deep RL [18] inspired by model-based RL is to draw a maximum of information from each transition to build more expressive representations of the states, hopefully leading to an easier learning process for the downstream policy. Weight sharing and learning auxiliary, potentially unsupervised, tasks [46] is a good way to drastically increase sampling efficiency, which is one the the core weaknesses of streaming RL since each transition must be discarded immediately after having been processed by the learning algorithm. Yet, learning such representations often involves learning a model of the environment's dynamics, which even in the standard RL setting is a challenging task, and self-supervised objectives usually rely on very large batch sizes. Learning rich and useful representations in the streaming context can therefore be a very difficult task to achieve.

In this paper, we investigate what makes learning representations in a streaming fashion so unstable. We use the framework of Self-Predictive Representations [11] for the representation learning aspect of our study and first show that naively plugging it in the streaming context

results in policies that do not improve beyond random behavior. We proceed to analyze the role played by non-stationarity in SPR's failure in the streaming setting. Our results suggest that although increased non-stationarity can explain part of the reasons why SPR does not translate well to the streaming setting, SPR may be more fundamentally incompatible with the streaming context. We hope that the insights we provide will help future work design new representation learning methods tailored for the streaming setting.

4.3 Related work

Representation Learning for RL Model-free RL is inherently sample inefficient. It often requires each transition to be stored in a replay buffer so that it can be used to be trained on multiple times. One way to improve sample efficiency is to learn auxiliary tasks [46]. A common set of auxiliary tasks is to predict some part of the dynamics of the environment [11, 15, 17, 19, 47]. Usually, these methods add a secondary objective, such as predicting the next observation(s), next latent state(s), next reward(s), action(s) used (inverse dynamics) or a combination of these, and use the weight sharing in the encoder to boost the policy's performances. [18] give a good overview of such methods, showing that what matters is to have some form of self predictive representations. Tasks relying on self-supervised contrastive objectives have also been tried [12, 16].

Another way to learn representations for RL is to use successor features [31–33]. These emerge from the linear decomposition of the Q-value as the product of a (non-linear) representation of the states and a (learned) task vector. The key difference between successor features and self predictive representations is that only the former verify a Bellman optimality equation, meaning they induce a fixed point which can be easier to optimize towards. Nevertheless, a unifying characteristic of most representation learning techniques in RL is the use of large replay buffers to try and overcome non-stationarity. Not many works study representation learning in a streaming fashion [40], and none do for streaming RL. It is an open challenge.

Continual Learning Continual learning usually consists in training an agent on multiple tasks sequentially in such a way that the agent is still good at solving the first tasks it was trained on, while still presenting a good ability to learn new ones. These challenges are respectively known as catastrophic forgetting [35] and loss of plasticity [34]. Common practice to alleviate these is to use replay buffers [36], sparse representations [48] or layer normalization [14] among others. Continual learning deals more broadly with learning representations from a stream of data, which is generally seen as a very hard task [41]. Most of these representation learning methods are inspired by the supervised learning (i.i.d.) literature and do not translate

well to streaming, non-i.i.d. settings, or RL [49].

Deep Streaming RL Even though the first reinforcement learning algorithms to have been created, TD(λ), Q(λ) and AC(λ) [8] to name a few, were designed for the streaming tabular case, they have since widely been adapted to use non-linear value function approximation while replacing the streaming setting with a replay-buffer-based setting, closer to the i.i.d. setting. Taking Deep RL algorithms back to the streaming context is a much more recent achievement [6, 7], and is still an under-studied area of RL, apart from some previous works [50]. Deep streaming RL suffers from very high non-stationarity which makes the optimization problem much harder to solve. To our knowledge, no other papers have tried to combine representation learning and deep streaming reinforcement learning.

4.4 Background

Reinforcement Learning Reinforcement Learning aims to solve a sequential decision problem, which can be modeled as a Markov Decision Problem (MDP) [51] $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. $\mathcal{S}$ represents the set of all environment states and $\mathcal{A}$ the set of all actions that are available to the agent. $\mathcal{P} : \mathcal{S}, \mathcal{A} \to \mathcal{S}$ encompasses the (stochastic) transition dynamics of the environment, $\mathcal{R} : \mathcal{S}, \mathcal{A} \to \mathbb{R}$ is the reward function and γ is the discount factor. The agent can interact with the environment through a policy $\pi : \mathcal{S}, \mathcal{A} \to [0, 1]$ which maps a state to a probability distribution over the action space conditioned on the state. At each time step t, the agent receives the current state as input $s_t \in \mathcal{S}$ and takes the action $a_t \sim \pi(\cdot|s_t)$. The environment state is updated following the transition function $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$ and gives a feedback to the agent in the form of a reward $r_t = \mathcal{R}(s_t, a_t)$.

We define the episodic return G_t as the summation of the discounted rewards obtained by an agent along a trajectory following a policy π and starting from timestep t. $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$, where T denotes the time step at which the episode terminates. We further define the value function $V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s]$ which evaluate the expected episodic return of an agent following policy π and starting at a specific state s_t. The goal of RL is to find the optimal policy $\pi^* = \text{argmax}_\pi\{V_\pi(s_0)\}$ where s_0 follows the initial state distribution of the environment.

To do so, we define the state-action value function, or Q-value $Q_\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a]$ and notice that it obeys the Bellman optimality equation [8] $Q_{\pi^*}(s_t, a_t) = r_t + \gamma \max_a\{Q_{\pi^*}(s_{t+1}, a)\}$. Deep Q-Learning exploits this fixed-point equation to learn an approximation of the Q-function, $Q_\theta \simeq Q_\pi$ parameterized by a neural network with weights θ. The DQN algorithm then simply consists in updating the parameters θ in the direction of the

semi-gradient of the following loss function, where SG denotes the *Stop Gradient* operation :

$$\mathcal{L}(\theta) = \Big(Q_\theta(s_t, a_t) - r_t - \gamma \cdot \texttt{SG}(\max_a\{Q_\theta(s_{t+1}, a)\})\Big)^2$$

Streaming RL The problem statement of RL does not change in the streaming context. However, the set of tools used to solve said problem is restricted : in this paper, we use [6]'s definition of the streaming setting of Reinforcement Learning. It mandates that any element of the transition (s_t, a_t, r_t, s_{t+1}) must be discarded right before the next transition. Any learning can only happen on the immediately available data (from the current time step). By definition, this prohibits the use of replay buffers. Another restriction is that no target networks are allowed. This means that we cannot keep a fixed or slowly evolving version of the Q-network for update stability purposes, which makes approaches like double Q-Learning prohibited. We can however aggregate any running statistic of the past transitions that we want. For instance, observation and reward normalization are allowed because these operations can be performed online, without storing any transition. This is one of the methods used by [6] in their *Stream-Q* algorithm, of which we give a more detailed explanation in Appendix 4.9.2. They also use layer normalization [14] before each activation function across the entire network, eligibility traces and a custom optimizer, Overshooting-bounded Gradient Descent (ObGD). ObGD performs a one-step approximation of a backtracking line-search algorithm [52] which finds the highest learning rate that still leads to stable gradient update, according to a stability criterion [23] based on the TD-error before and after updating the network's parameters.

Eligibility traces To improve sampling efficiency in a Q-network we can use n-step returns. Instead of just considering the next step in the computation of the return, we consider a higher bootstrapping depth, thereby defining $Q_{\pi^*}(s_t, a_t) = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^N \max_a\{Q_{\pi^*}(s_{t+n}, a)\}$ as the Bellman optimality equation. This however is not allowed in the streaming setting, due to the need to store a small buffer of the n previous transitions. Instead, we can use λ-returns, which combine n-step returns for all possible values of n, weighted by λ. λ-returns provide a way to use a "backward-view" algorithm which only needs a running statistic of the previously seen states to perform a learning update in the current one. This running statistic is called the eligibility trace and is defined as such :

$$\begin{cases} \mathbf{z}_0 = \mathbf{0} \\ \mathbf{z}_{t+1} = \lambda\gamma\mathbf{z}_t + \nabla_\theta Q_\theta(s_t, a_t) \end{cases}$$

Eligibility traces are reset between episodes, and in the case of ϵ-greedy exploration whenever

the action taken is not the greedy one. This leads to the parameter update rule $\theta \leftarrow \theta + \alpha\delta_t \mathbf{z}_t$, where δ_t is the TD-error (difference between the current estimate of the Q-value and its bootstrapped estimate).

Self-Predictive Representations (SPR) In a broader setting, self predictive representations are a representation learning technique inspired from model-based RL. They are designed to learn rich state representations $z_t = f_\xi(s_t)$ through a secondary objective, relying on learning a model of the dynamics of the environment, usually such that $d_\phi(z_t, a_t) = z_{t+1}$. This prediction can also happen in the observation space rather than the latent space, but this has been show to be less effective as the learned representation must also be able to be used to reconstruct irrelevant details of the image [11, 53]. The encoder is shared between the representation learning and the Q-value parts of the overall network, effectively sharing weights to improve the Q-network's representations. A loss function (e.g. Mean Squared Error (MSE) or cosine similarity) is then applied between the predicted next latent state and the actual next latent state $\mathcal{L}_{SPR} = \text{loss}(d_\phi(f_\xi(s_t)), f_\xi(s_{t+1}))$ so that by learning to predict the dynamics of the environment, the shared encoder learns a good representation of the states. [11] build upon this idea and sequentially predict the next K latent states and use a cosine similarity loss rather than a reconstruction-oriented MSE loss. We use this approach with $K = 5$ as our representation learning baseline in this paper and for the rest of the paper, this is the method we will refer to as SPR. While we realize that predicting the next $K > 1$ steps violates the streaming RL setting, we find it offers improved learning capabilities in harder environments (see Figure 4.6 in Appendix 4.9.3) and consider that having a replay buffer of size 5 is still very close to a fully streaming setting. Ideally future works would get rid of this requirement.

4.5 Method

The goal of this paper is to study the dynamics of representation learning in the streaming RL setting, and provide an understanding of what makes it such a difficult task to learn. To address this question, we conduct two sets of experiments. In the first one, we gradually decrease the level of non-stationarity of our experimental setup, starting from a naive integration of SPR in the streaming setting. This allows us to isolate the impact of non-stationarity on SPR's performances. In the second set of experiments, we analyze the quality of the representations learned by a non-streaming SPR agent and their effect on Stream-Q agents. Finally, we discuss potential research directions to hopefully improve learning streaming representations for RL.

We conduct our analysis on 10 Atari environments, using ten seeds per environment unless stated otherwise. We train our agents for one million frames (250k steps). This shorter setting is much harder than what Stream-Q was originally designed for. However, it is very close to the setting used in SPR, where agents are trained for 400k frames but updated twice per environment step. Since our goal is to analyze SPR-based agents, we choose this 1M frame setting as it is close to the Atari100k setting SPR uses, while not being as short on the training time (250k steps instead of 100k).

We use the Stream-Q agent from [6] as our streaming baseline and a slightly modified version of [11]'s SPR agent, where the main difference is that we use a DQN agent with SPR instead of a Rainbow one. We keep image augmentations and $K = 5$ as the next latent state prediction window in SPR. As is done in Stream-Q's original setting we used parameter-free (no scaling or bias parameter is learned) layer normalization on all the pre-activations and used normalized observations and rewards. We give a more complete overview of our experimental setup, hyperparameter values and implementation choices in Appendix 4.9.1.

A core aspect of the streaming setting is the optimization. The ObGD optimizer was one of the key elements in the success of Stream-Q. In this paper, we also faced critical optimizer design choices as Stream-Q uses eligibility trace-based parameter updates by default and SPR uses Adam-based updates [20]. Combining these two separate sources of update for the shared parameters is not obvious and we investigated two distinct directions. On a one hand, we tested different strategies to merge these updates (more details in Appendix 4.9.4) which led to choosing the following update mechanism for shared parameters :

$$w_{\text{shared}} \leftarrow w_{\text{shared}} + \mu_{\text{shared}} \alpha_{\text{ObGD}} \delta_t \mathbf{z}_t - (1 - \mu_{\text{shared}}) \alpha_{\text{SPR}} \lambda_{\text{SPR}} \textsc{Adamify}(\nabla \mathcal{L}_{\text{SPR}}(w_{\text{shared}})) \quad (4.1)$$

where μ_{shared} is a mixing coefficient that was set to 0.5 by default and λ_{SPR} is the SPR loss coefficient. Here, Adamify means transforming the gradients following the Adam update, using the corrected first and second momentums of the gradient). Unless specified otherwise, every experiment involving SPR uses $\lambda_{SPR} = 2$. On the other hand, we assessed different optimization methods such as shrink and perturb [54], orthogonal gradient projection [39] and two-step optimization. We also decided to use different optimizer combinations for the Q-learning and SPR parts of the network (see Table 4.4 in Appendix 4.9.1) to see if having updates of similar type (Adam momentums) in both parts of the network improved performance. One noteworthy choice for AdamW [55] specifically is that we excluded bias terms from being decayed.

We use noisy networks [56] for exploration for all agents. However, with streaming agents using ObGD and SGD as their optimizers for the Q-network, we found that the standard

deviation of weights in the noisy layers barely decreased, effectively never exiting the initial random exploration phase. For that reason, agents using ObGD or SGD use a very short ϵ-greedy exploration schedule instead, going from $\epsilon = 1$ to $\epsilon = 0.05$ in 100k frames.

Finally, we evaluate the quality of the representations learned through the Inter-Quantile Mean (IQM) with a confidence interval of 95% (corresponding to the shaded area in the IQM plots) and use the Rliable [57] library. We also rely on the effective rank [58] of the output of the encoder on large batches which gives a clear indication of whether the representations have collapsed as well as an intuition about their quality. Although a high effective rank does not necessarily mean that the representations are good (random matrices have very high ranks), our experiments on SPR (see Figure 4.5b) show that better algorithms lead to increasing high effective ranks. As such, we consider it is a good proxy to infer representation quality in our setting. The shaded area around the effective rank plots represents their standard deviation.

4.6 Experiments

4.6.1 Impact of the non-stationarity on SPR

In the first three experiments, we aim to isolate the role played by non-stationarity in the ability of streaming SPR agents to learn representations. Each experiment following the first will remove some level of non-stationarity. The goal is to see if and when the problem setup becomes stationary enough for streaming agents augmented by a SPR objective to perform better than the Stream-Q baseline.

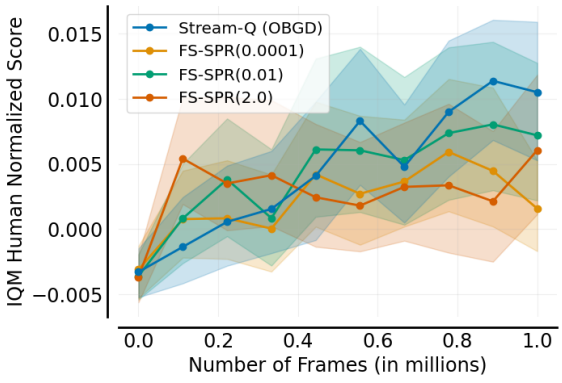
From Stream-Q to Fully-Streaming SPR This first experiment acts as a control. It is the most non-stationary setting as every part of the network is trained in the streaming context. By Fully-Streaming SPR (also denoted FS-SPR in the legend of the figures), we mean that both the Q-learning objective and the SPR objective are trained without replay. The goal of this experiment is to diagnose the potential shortcomings of Fully-Streaming SPR, as well as get a base effective rank level for both Stream-Q and Fully-Streaming SPR. This experiment was the result of a hyperparameter search, the details of which are further discussed in Appendix 4.9.3.

In our short 1M frames training setting, experiments with ObGD (Figure 4.1) show that training SPR in a purely streaming fashion does not improve on the baseline as Stream-Q mostly has better performance compared to Fully-Streaming SPR, both IQM- and effective rank-wise. We also tried different optimizers like SGD, as it is known to be better suited when dealing with non-stationary problems [59] and AdamW, as it has better generalization

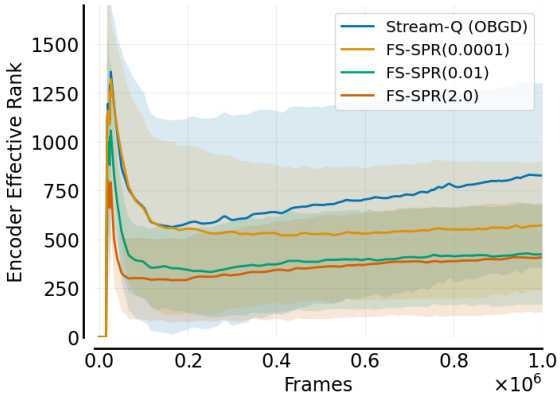
capabilities than Adam [55]. Although Figures 4.16 and 4.17 in Appendix 4.9.5 show that Fully-Streaming SPR agents using Adam, AdamW and SGD match their streaming equivalent without SPR both IQM- and effective rank-wise, they are at most on par with Stream-Q with ObGD.

The aforementioned longer setting, uses a 10M frame long ϵ-decay period, identical to what was originally used in the Stream-Q paper. In this setting, Stream-Q with ObGD performs at its best and we observe an clearer trend in Figure 4.2. There is a direct negative correlation between the SPR loss coefficient and the performances : lowering SPR's effect through smaller loss coefficient consistently improves performances. This suggests that Fully-Streaming SPR is harmful for Stream-Q and confirms that Fully-Streaming SPR is not failing solely because of our more challenging shorter training duration.

Due to the fact that learning representations for reinforcement learning is a famously difficult task, even when not being restricted to the streaming setting, we hypothesize that the poor performances mostly come from the exacerbated non-stationarity of the problem. To confirm this hypothesis, the next two experiments were designed to alleviate and even get rid of the non-stationarity.

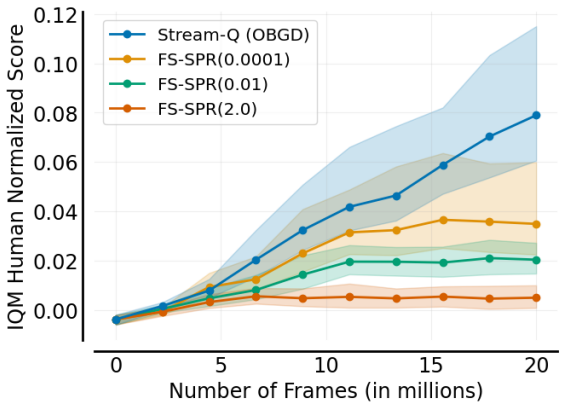


(a) Comparison of the IQM. No clear trend between loss coefficient values but Stream-Q performs slightly better without *any* auxiliary SPR objective.

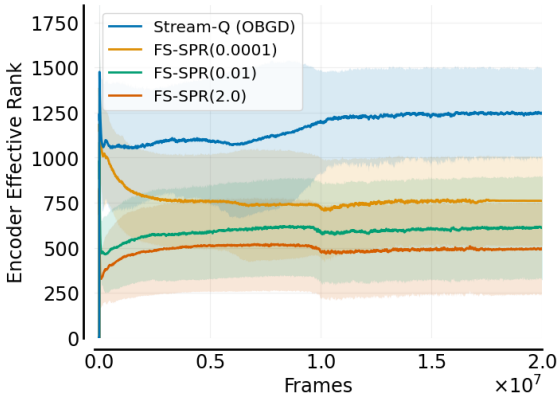


(b) Comparison of the effective rank. No clear trend between loss coefficient values but smaller values of λ_{SPR} seem to provide better representations.

FIGURE 4.1 Comparison of *a)* the IQM and *b)* the effective rank of the representations produced by the encoder between the Stream-Q baseline and Fully-Streaming SPR, FS-SPR(λ_{SPR}), for different values of the SPR loss coefficient in our default 1M frame setting. All agents use the ObGD optimizer for the Q-learning part of the network and Adam for the SPR-part.



(a) Comparison of the IQM. The lower the value of λ_{SPR} the better the performance of the agent : Fully-Streaming SPR hurts performances IQM-wise.



(b) Comparison of the effective rank. The lower the value of λ_{SPR} the better the quality of the learned representations : Fully-Streaming SPR hurts performances effective rank-wise.

FIGURE 4.2 Comparison of *a)* the IQM and *b)* the effective rank of the representations produced by the encoder between the Stream-Q baseline and Fully-Streaming SPR for 20M frames. All agents use the ObGD optimizer for the Q-learning part of the network and Adam for the SPR-part.

From Fully-Streaming SPR to Stream-Q + SPR This second experiment addresses the non-stationarity in Fully-Streaming SPR by adding a replay buffer to the SPR part of the network. To do so, the SPR loss is computed as the average SPR loss on a batch of size 32 sampled from a replay buffer of size 10^5. This is a big step toward a more stable problem setting, nevertheless the Q-network is still learning from a stream of observations and is ultimately the determining factor when it comes to performance. Since we are improving training stability compared to the previous experiment, we expect Stream-Q + SPR to perform at least as well as the fully streaming version.

This is however not what we observe empirically. Indeed, Figure 4.16 in Appendix 4.9.5 shows that uniformly across all optimizers except ObGD, even with a replay buffer Stream-Q + SPR does not perform better than Stream-Q or Fully-Streaming SPR. The effective ranks of the learned representations (Figure 4.17 in Appendix 4.9.5) are however higher on average when using SPR with replay, albeit with a slightly higher variance. This suggests that although removing the non-stationarity helped the SPR task (we get higher effective representation ranks), the downstream performances of the agent are worse. This seems to hint at the fact that Stream-Q may try to learn different representations from the ones learned by SPR. It is possible that even though individually Stream-Q and SPR learn good representations, when learned conjointly their updates interfere in a counter-productive way. This could explain

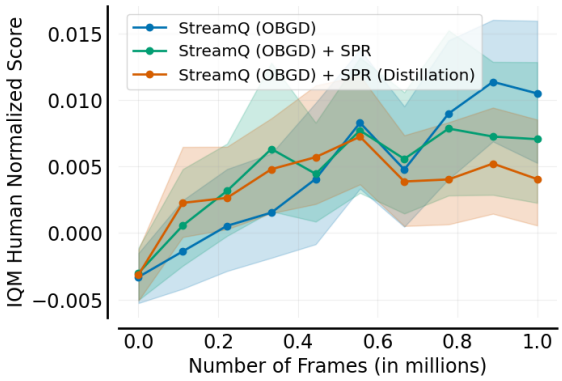
the drop in performance, even though the encoder seems to still be learning representations without collapse.

Distillation objective In this third experiment we go even further in removing the non-stationarity of the SPR task by training a Stream-Q + SPR agent with a distillation loss from a teacher network (pre-trained non-streaming SPR agent). The Q-learning part of the agent is trained in the Stream-Q fashion while the SPR objective is trained with a replay buffer, according to the mean squared error between the teacher SPR network's and the agent's SPR network prediction. This removes the non-stationarity in the target of the representation learning task as the new SPR target does not change through the training, but heavily relies on the teacher being good enough. We delve deeper into this last point in the second set of experiments. Another way to get rid of the non-stationarity is to use a completely random fixed policy. However, we tried this approach as discussed in Appendix 4.9.5 and it doesn't result in better results.

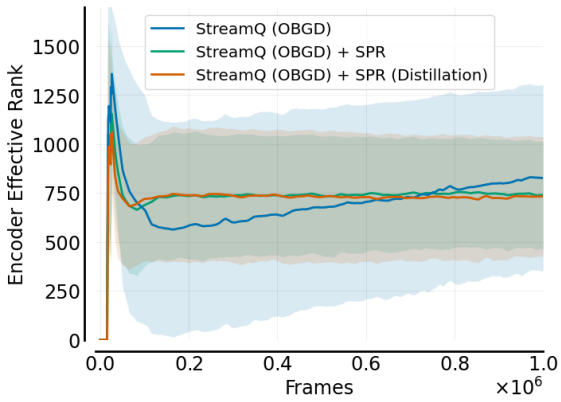
Our results (Figures 4.3a and 4.18 in Appendix 4.9.6) show that even though we removed all the non-stationarity beyond sample distribution in the training objective of the SPR part of the network, the performance of the agent have not changed from the Stream-Q baseline, if not worsened. However, we observe once more that the average effective rank of the encoder representations is equal or higher (Figures 4.3b and 4.19 in Appendix 4.9.6) when adding the distillation objective to the SPR part of the agent. This consistent trend further raises the question of the compatibility between the Q-updates and the SPR-updates. Indeed, the more stationary we have made the SPR learning objective, the better the rank of the encoder representations has been. This seems to indicate that SPR is indeed learning better representations the more stationary the setting is, these representations just happen to not help, and actually hinder the task of learning the Q-values.

4.6.2 Investigating SPR representations

Now that we have a clearer understanding of the role of non-stationarity in the failure of Fully-Streaming SPR and Stream-Q + SPR to learn, another question emerges : is the cause of the deteriorated performances of streaming-SPR due to the representations learned by SPR not being good representations for Stream-Q or to the joint learning of both tasks ? This second set of experiment attempts to address this question by investigating deeper the representations learned by SPR agents.



(a) Comparison of the IQM. No version of Stream-Q + SPR beats the Stream-Q baseline : non-stationarity is not the only factor causing Stream-Q + SPR to fail.



(b) Comparison of the effective rank. As opposed to the IQM, increasing the stability of the SPR task leads to equal (or higher with other optimizers) effective ranks : non-stationarity is detrimental to the SPR objective in the quality of the representations learned.

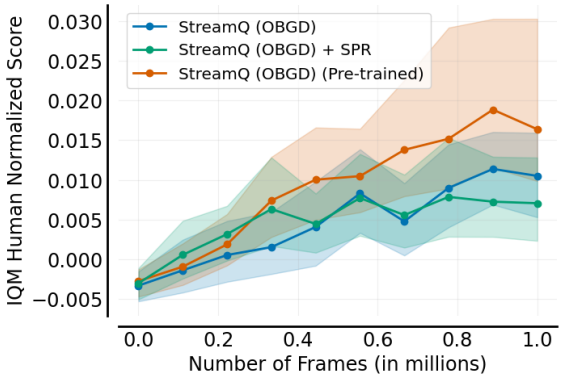
FIGURE 4.3 Comparison of *a)* the IQM and *b)* the effective rank of the representations produced by the encoder between Stream-Q and Stream-Q + SPR with a distillation loss for the SPR objective. All agents use the ObGD optimizer for the Q-learning part of the network and Adam for the SPR-part.

Pretrained encoders This first experiment studies the impact of non-streaming SPR's learned representations on a Stream-Q agent. We use the same pre-trained SPR agent as in last experiment, but rather than using it as a teacher model, we initialize the encoder weights of a Stream-Q agent with those of the pre-trained agent. Given that the goal of SPR is to learn good representations through the encoder, such an initialization of a Stream-Q agent should result in better performances. We run two versions of this experiment : one where the initialized encoder weights of the streaming agent are frozen and a second one in which the agent is free to update the encoder weights.

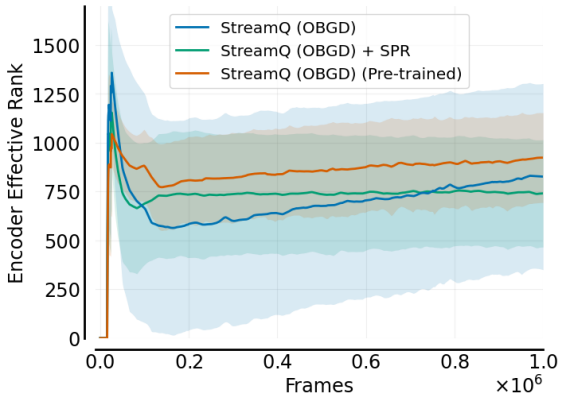
Figure 4.4 confirms our hypothesis that Stream-Q with pre-trained weights performs (slightly) better than the Stream-Q baseline, although only when the weights of the encoder are not frozen (see Figure 4.12 in Appendix 4.9.5). We also observe that the average effective rank of the encoder representations is higher when the encoder has been initialized with the weights of the pre-trained SPR agent, and stays relatively high throughout the training when they are not frozen. These trends are verified for all optimizers as shown by Figures 4.20 and 4.21 in Appendix 4.9.6.This indicates that the representations learned by the SPR agent are good enough to make Stream-Q learn faster. We further ran a control experiment in Appendix

4.9.5 where we replaced the Stream-Q agent by a DQN agent (Figure 4.11) and observe the same trends when the encoder weights are not frozen. This confirms that the representations learned by our expert model are good enough. Therefore, this experiment shows that the representations learned by SPR are valuable and useful to Stream-Q.

We also conducted the same pre-training experiment on Stream-Q + SPR in Appendix 4.9.5 and interestingly, the results we observed are opposite to that of Stream-Q alone. Adding the SPR task in addition, even with good starting pre-trained representations, completely prompted Stream-Q (ObGD) + SPR to fail, whereas Stream-Q (Adam) and Stream-Q (AdamW) saw their performance increase by a lot. This further hints at the unaligned nature of Q-learning and SPR based updates and highlights the importance of the design of the optimization problem.



(a) Comparison of the IQM. A Stream-Q agent initialized with pretrained encoder weights visibly outperforms the Stream-Q baseline : the representations learned by SPR are valuable to Stream-Q.

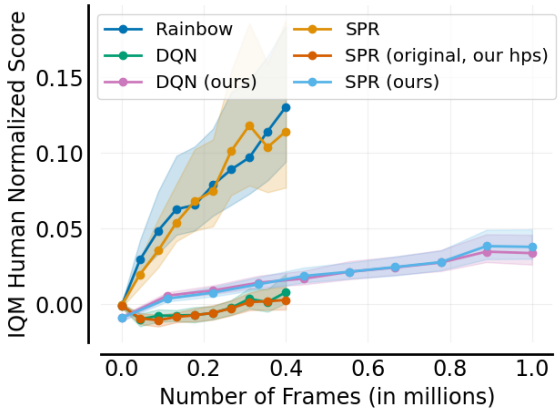


(b) Comparison of the effective rank. The Stream-Q agent initialized with pretrained encoder weights maintains the effective rank of its encoder's representations high : the representations learned by SPR are valuable to Stream-Q.

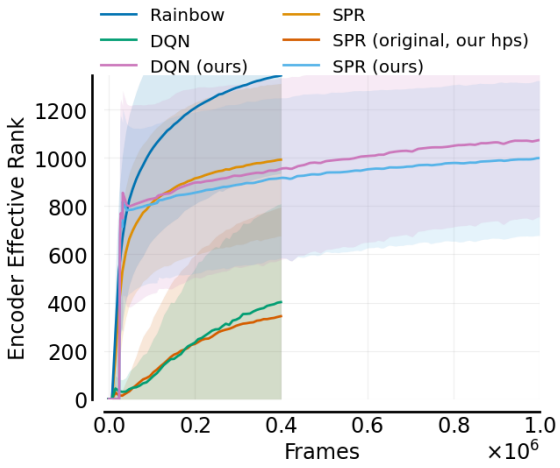
FIGURE 4.4 Comparison of *a)* the IQM and *b)* the effective rank of encoder representations between Stream-Q, Stream-Q + SPR and a Stream-Q agent whose encoder weights have been initialized from a pretrained agent. All agents use the ObGD optimizer for the Q-learning part of the network and Adam for the SPR-part.

Original SPR setting As mentioned earlier, the quality of the teacher in the distillation experiment is crucial for the student network to perform well. Therefore, as our last experiment, we analyze the impact of our simplified choice of hyperparameters for SPR (discarding the Rainbow improvements over DQN), on the learned representations of our teacher network. We compare the performances of DQN and SPR in our setting to their counterparts

in the original SPR setting, both with and without the Rainbow improvements. Figure 4.5b shows that, in the original SPR setting, the representations learned by Rainbow have a higher, but noisier, average effective rank than SPR, beating it by a non-negligible margin. As for their streaming-friendlier versions, DQN and our hyperparameter choice for SPR both have very similar performances rank-wise and IQM-wise, well below that of SPR although they perform better in our setting. We can draw two conclusions from these results. First, even though SPR performs worse IQM-wise (Figure 4.5a) when we remove the Rainbow improvements (in both settings), the difference in effective rank (Figure 4.5b) is much lower with our setting, thereby further confirming the quality of the representations learned by our teacher network. Second, these results suggest that part of SPR's success lies in the improvements provided by Rainbow over DQN. It opens an interesting followup question of whether an equivalent of Streaming-Rainbow would perform better with SPR. Figure 4.15 in Appendix 4.9.5 also shows the same experiment results extended for 15 environments to better capture the extent of SPR's capabilities.



(a) Comparison of the IQM. While SPR and Rainbow perform much better than DQN and SPR without any Rainbow improvements in our setting, our versions of DQN and SPR perform better than their equivalent in the original setting. This confirms the quality of our teacher networks.



(b) Comparison of the effective rank. Rainbow learns the representations with the highest rank, closely followed by both the original and our SPR settings, as well as our DQN setting. Dead last are DQN and SPR equivalents to ours in the original setting. This confirms the quality of the representations learned by our teacher networks.

FIGURE 4.5 Comparison of *a)* the IQM and *b)* the effective rank of encoder representations between our and SPR's original setting on our 10 chosen environments. "*ours*" refers to agents ran in our setting, while "*original, our hps*" refers to agents trained in the original SPR setting, with a replay ratio of 2 and no Rainbow improvement.

4.7 Discussion

Limitations In this paper, we took $K = 5$ for the prediction depth of SPR because it gives better learning dynamics, instead of $K = 1$ to respect the streaming setting to the letter. Another choice we made was to not consider the use of target networks, as they only affect the Q-network, not the SPR-part of the network. Such a choice however excludes trying the Exponential Moving Average (EMA) setup of SPR without image augmentation, which may provide more stable training dynamics. We also made two simplifying choices in our hyperparameter selection for SPR for ease of implementation purposes. Namely, we did not use Rainbow's categorical loss nor its dueling architecture. These could improve the stability of the training as categorical loss for RL has been shown to be better for scaling [60] and learning representations [61]. We leave investigating if these benefits are transferrable to the streaming setting for future works.

Future directions Our results suggest that the main issue of learning streaming representations stems from the difficulty of the optimization problem. The ObGD optimizer was also one of the key elements of the success of Stream-Q. For this reason, we think that investigating adaptive learning rates based on the TD-error of the Q-network for the SPR part of the network, in a way reminiscing of ObGD, would be an interesting direction to follow. In the same spirit, Sharpness Aware Minimization [62] could be another interesting alternative optimization strategy to use, as its goal is to keep the parameters in a flatter region of the loss landscape. This could help reconcile the parallel optimization of the Stream-Q and SPR objectives as small changes in the parameter vector should not lead to drastically different losses. Another interesting avenue is using orthogonal gradients based optimization [40]. This optimization strategy consists in orthogonally projecting the gradients at step t on the gradients at step $t-1$ and has been specifically designed for learning representations in a streaming setting (albeit not RL related). The idea is that for highly time-correlated gradients, this method will de-emphasize successive updates that are heavily correlated but leave unchanged new gradient directions. We also have not analyzed the role of dormancy [63] in the results we observe and believe this could give valuable insights as to why representations are not learned properly.

4.8 Conclusion

In this paper, we investigated the effects of adding a secondary representation learning task to a deep streaming RL pipeline to boost its sampling efficiency. Not only did we not

find that adding this secondary task helped, in part because of the increased non-stationarity of the problem, we also found that the nature of the task may itself be a problem. Through an ablation study in which we progressively removed the non-stationarity of the problem, we found that even though when the non-stationarity is lower SPR yields representations with higher effective ranks, the agent's downstream performances remain below that of the streaming baseline. We then confirmed that the representations ultimately learned by SPR are not harmful to streaming agents. Thus, we concluded that in addition to the increased non-stationarity of the problem, learning the SPR objective conjointly with the Q-learning bootstrapping objective causes the streaming SPR agents to fail.

4.9 Technical Appendix

4.9.1 Experimental details

This section goes over the hyperparameter choices, architectural choices and implementation details in more depth to allow for better understanding of our setup and easier reproduction of our results.

Hyperparameters We provide the details of all our hyperparameter choices in Tables 4.1, 4.2 and 4.3.

TABLE 4.1 SPR specific hyperparameters

Parameter	**SPR (original setting)**	**SPR (our setting)**
Training steps	100k	250k
Replay factor	2	1
Dueling architecture	True	False
Categorical loss	True	False
n-step returns	10	1
Replay	PER	Standard
Replay warm-up	2000	2048
Replay capacity	N/A	10^5
Update mixing coefficient	N/A	$\mu_{shared} = 0.5$
Batch size	32	32
Q-target network	False	False
Augmentations	Intensity ($\sigma = 0.05$) Random shifts (± 4 pixels)	Intensity ($\sigma = 0.05$) Random shifts (± 4 pixels)
EMA parameter	$\tau = 0$	$\tau = 0$
Noisy nets parameter	0.5	0.5
Dynamics prediction depth	$K = 5$	$K = 5$
SPR loss coefficient	$\lambda_{SPR} = 2$	$\lambda_{SPR} = 2$
Learning rate	10^{-4}	See Table 4.2

Atari environments All agents evaluated with our SPR settings are trained on the following 10 environment from the Arcade Learning Environment (ALE) [64] : Alien, Asterix, BankHeist, Breakout, Enduro, Freeway, Frostbite, MsPacman, Pong and Qbert. This list was designed to contains a mix of environments where both Stream-Q and SPR reportedly perform well (e.g. Frostbite) or rather poorly (e.g. Qbert), as well as environments where only one of the two algorithms performs very well (e.g. Breakout, Asterix).

TABLE 4.2 Optimizer specific hyperparameters

Parameter	**ObGD**	**SGD**	**Adam**	**AdamW**
Learning rate	1	10^{-4}	10^{-4}	10^{-4}
Update type	Traces	Gradients	Gradients	Gradients
κ (ObGD)	2	N/A	N/A	N/A
β_1 (Adam)	N/A	N/A	0.9	0.9
β_2 (Adam)	N/A	N/A	0.999	0.999
ϵ (Adam)	N/A	N/A	0.999	0.999
Weight decay (AdamW)	N/A	N/A	N/A	0.01

TABLE 4.3 Other hyperparameters used

Parameter	**All experiments**
Discount factor	$\delta = 0.99$
Eligibility trace decay	$\lambda = 0.8$

In order to get closer results to the ones from the original SPR paper, we assessed SPR's original performances with its original settings (Figure **??**) on the following 15 environments : Alien, Asterix, BankHeist, BattleZone, Breakout, DemonAttack, Enduro, Freeway, Frostbite, Kangaroo, MsPacman, Pong, Qbert, RoadRunner and UpNDown.

We use the same setup as [6] regarding the Atari environments. Each environment frame is down-sampled to a shape of 84×84 and converted to grayscale. Frames are then stacked by 4 (to deal with partial observability) and this constitutes the (unnormalized) state returned by the environment. At the beginning of each episode, the agent is forced to take a random number of no-op actions (up to 30). For environments where the game starts after the firing action has been taken, the agent is forced to take a random action at the start. Episodes are terminated when the agent loses all of its lives and each action taken by the agent repeated 4 times. Finally, we normalize the observations and rewards before giving them as input to the agent, according to [6].

LayerNorm It is important to note that throughout this paper, when we mention using layer normalization we refer to the parameter-free layer normalization used by [6]. Meaning the LayerNorm we use does not have the usual two learnable parameters of re-scaling and bias. For a given input $\mathbf{x} \in \mathbb{R}^n$, it corresponds to the following formula :

$$\forall i \in [\![1, n]\!],\ \textsc{LayerNorm}(\mathbf{x})_i = \frac{\mathbf{x}_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \qquad \text{where } \mu = \frac{1}{n}\sum_{k=1}^{n} \mathbf{x}_k \text{ and } \sigma^2 = \frac{1}{n}\sum_{k=1}^{n}(\mathbf{x}_k - \mu)^2 \tag{4.2}$$

Epsilon is a small quantity added for numerical stability and in our case we use the default pytorch value of $\epsilon = 10^{-5}$.

Architecture design (Q-network) We use the default encoder architecture for Atari consisting of three convolutional layers. This architecture differs from that of [6] as they use a smaller 256-dimensional bottleneck for the output of their encoder. However, in order to have meaningful comparisons of effective rank with the original SPR setting, we decided to extend the output of our encoder to have the default $64 \times 7 \times 7$ shape (dimension of 3136). Thus, the first convolutional layer has a kernel of size 8×8, a stride of 4 and outputs 32 channels. The second one has a kernel of size 4×4, a stride of 2 and outputs 64 channels. The last one has a kernel of size 3×3, a stride of 1 and outputs 64 channels. The output of each convolution layer is passed through a LayerNorm and we use LeakyReLU(0.01) as the activation function. We then have two fully-connected layers computing the Q-values per action from the state representation. We use a hidden dimension of 256, so the layers have respective shapes of 3136×256 and $256 \times |\mathcal{A}|$. The output of the first fully-connected layer is passed through a LayerNorm and a LeakyReLU(0.01) activation. The last layer has no bias term and directly outputs the Q-values. These fully connected layers are replaced with equivalent noisy layers when using noisy networks [56] for exploration. We also apply the same sparse initialization scheme as [6] to all the layers used in the Q-network (encoder and fully-connected layers). This effectively sets all biases to zero and initializes 90% of each layer's weights to zero while the 10% left follows the LeCun initialization scheme [65].

Architecture design (SPR-network) The SPR part of the network shares the encoder with the Q-learning part of the network. A difference between our architecture and the one from [11] is that we use layer normalization before each activation (as opposed to batch normalization after some of the layers). For this reason, we drop the renormalization of the encoder and dynamics network outputs, as we already use LayerNorm everywhere. The dynamics network consists of two convolutional layers, both using kernel sizes of 3×3, padding of type `"same"` and padding mode of type `"reflect"`. The first convolutional layer also takes a one-hot encoding of the action as input, concatenated in the channel dimension. Again, all pre-activations are passed through a LayerNorm and a LeakyReLU(0.01) activation. Then follows the prediction layer, a potentially noisy fully-connected layer. By default, this layer

is shared with the Q-network. As such, it is exactly the same as the first fully-connected layer of the Q-network (also sharing the same noisiness). Finally the projection layer is a fully-connected layer of shape 256×256. None of these two last layers use LayerNorm or activation functions. The predicted next latents are iteratively computed via the dynamics model : $\hat{z}_{t+j+1} = d_\phi(\hat{z}_{t+j}, a_{t+j})$ for $0 \leq j < K$ and $\hat{z}_t = z_t$. These prediction are then passed through the prediction and projection layers. Their targets are computed from the encoder directly applied to the next states in the trajectory, followed by the prediction layer and no gradients are propagated through these operations.

Optimizer As mentioned in the method section, because Stream-Q with the ObGD optimizer uses eligibility traces-based updates and SPR uses Adam-based updates, weight sharing could not be implemented as easily as back-propagating the sum of the two losses directly. The update process of non-shared parameters is unchanged and is performed using the gradients or eligibility traces deriving from the relevant loss. For shared parameters, we perform two separate phases of back-propagation (one per loss). We compute the updates induced by each one of them, either with eligibility traces or gradient momentums, and apply the final update to each shared parameter as the weighted sum of the updates induced by the SPR objective and the Q-learning objective. This approach entails the two following observations : First, the optimal learning rate computation in the ObGD optimizer only considers the L_1 norm of the update coming from the Q-learning objective and disregards any update coming from the SPR objective. This overestimation of the maximum stable learning rate should however be accounted for by the κ coefficient, taken to be equal to 2 as in [6], as well as our mixing coefficient $\mu_{shared} = 0.5$. Second, when using the Adam optimizer for both the Q-learning and SPR part of the network, the shared parameters will be updated according to $\textsc{Adam}(\nabla\mathcal{L}_Q) + \textsc{Adam}(\nabla\mathcal{L}_{SPR})$ which is different from $\textsc{Adam}(\nabla(\mathcal{L}_Q + \mathcal{L}_{SPR}))$. In practice, our experiments comparing the results of SPR between our setting and [11]'s setting show that both lead to SPR agents learning meaningful representations, even though our setting uses the aforementioned Adam computation. Additionally, for the case of AdamW, we excluded bias terms from the decayed parameters. Finally, we used the optimizer pairings described in Table 4.4 for the Q-learning and SPR parts of the networks.

Compute resources To run these experiments, we mostly used Nvidia L40S, RTX8000 and V100 GPUs, with 8GB RAM requirements for the experiments without a replay buffer and 64GB otherwise. Our Stream-Q agents complete a 1M frames training in roughly 3 to 4 hours. Our Fully-Streaming SPR agents and other agents using a replay buffer run in around 1 day on the same task. In the original SPR codebase, 400k frames experiments take from 2

TABLE 4.4 Optimizer pairings

Q-learning objective		**SPR objective**
ObGD	*with*	Adam
SGD	*with*	SGD
Adam	*with*	Adam
AdamW	*with*	AdamW

to 6 hours to complete depending on the algorithm.

4.9.2 The Stream-Q agent

This section goes over a detailed explanation of the Stream-Q agent from [6]. In the streaming setting, the non-stationarity of the RL framework is exacerbated due to the fact that updates are performed on highly correlated gradients. The idea behind the design of the Stream-Q agent is to address the instability factors that are the most affected by the streaming setting, namely the decreased sampling efficiency, the learning instability due to highly correlated gradients and the poor learning dynamics induced by improper scaling of data.

The authors address sampling efficiency in two ways. First, by using eligibility traces rather than regular gradients for their updates as they provide better and faster credit assignment. Second, they use a sparse initialization scheme in order to promote sparse representations, as they have been shown to reduce forgetting and be beneficial for RL [22].

Because gradients between successive updates are highly correlated, the resulting parameter updates may vary a lot in magnitude. The authors propose a new optimizer, named Overshooting-bounded Gradient Descent (ObGD). As mentioned in the background section, the ObGD optimizer performs a one-step approximation of a backtracking line-search algorithm [52]. In other words, ObGD gives a good candidate value for the highest achievable learning rate such that a stability criterion is verified. In the case of SPR, the stability criterion considered is the effective step size defined by [23] as :

$$\xi = \frac{\delta(s_t) - \delta_+(s_t)}{\delta(s_t)} \tag{4.3}$$

where $\delta(s_t)$ is the TD-error and $\delta_+(s_t)$ is the TD-error on the same state after having updated the network parameters according to $\delta(s_t)$. An update is considered unstable if $\xi > 1$. Usually, a backtracking line-search algorithm would iteratively reduce the learning rate until it finds a value that verifies the stability criterion. ObGD approximates this process by performing

only one iteration, and without computing $\delta_+(s_t)$ as this can be a costly operation. [6] come up with the following upper bound for a the stability criterion : $\xi \leq \alpha\kappa\bar{\delta}_t||\mathbf{z}_t||_1$, where α is the step size, $\kappa > 1$ is an hyperparameter of ObGD, acting as a security coefficient. $\bar{\delta}_t = \max(1, |\delta(s_t)|)$ and $\mathbf{z}_t$ is the eligibility trace used in the update. This combined with the stability criterion gives the following condition on the stability of the learning rate : $\alpha \leq (\kappa\bar{\delta}_t||\mathbf{z}_t||_1)^{-1}$. For a desired maximum step-size α^*, ObGD is thereby defined by the following update rule :

$$\mathbf{w}_t \leftarrow \mathbf{w}_t + \min\left(\alpha^*, \frac{1}{\kappa\bar{\delta}_t||\mathbf{z}_t||_1}\right) \cdot \delta(s_t)\mathbf{z}_t \tag{4.4}$$

The adaptive step-size offered by ObGD allows to efficiently deal with highly correlated updates by ensuring that the behavior of the network after update will be close to its behavior before the update. In order to provide more stability to the network, the authors also use layer normalization before every activation function. They use parameter-free LayerNorm as explained in the previous section of the Appendix.

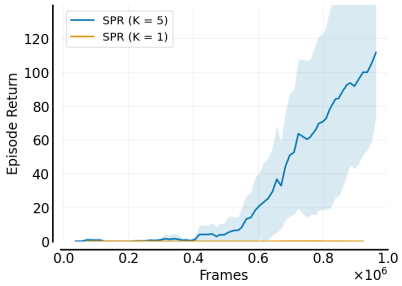
Finally, the authors normalize the rewards and observations of the agent, keeping a running average and standard deviation of all states and rewards encountered so far. This is a known method to improve training stability [66].

4.9.3 SPR Hyperparameter search

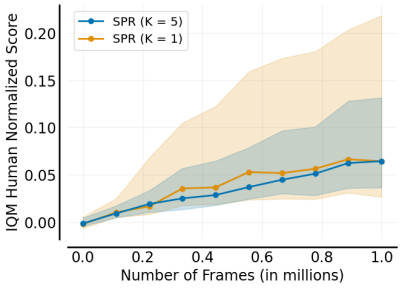
First search Given the exponentially large number of possible hyperparameter combination, we performed the first part of this hyperparameter search over a smaller set of environments, namely Asterix, Breakout, Frostbite and Qbert. We exclusively ran experiments in a longer setting, where agents are trained for 50M frames give or take. In this setting, we use the same ϵ-greedy exploration scheme as [6], which involves a linear decay of ϵ from 1 to 0.01 in 10M frames. Most of these experiments were ran using the Stream-Q's encoder architecture in which the encoder output has a dimension of 256 instead of 3136. We also used $K = 1$ in these experiments. These experiments were performed on Fully-Streaming SPR agents. Unfortunately, this first search was rather inconclusive as we did not find any hyperparameter combination that gave better or even similar performances to Stream-Q. Indeed, as we have shown in the paper, Fully-Streaming SPR agents never achieve better performances than the Stream-Q baseline, at least in terms of IQM. This makes it very tricky to determine which values of hyperparameters work better and is one of the main reason we decided to measure the effective rank of the learned representations in the following searches and later experiments, in order to have a better understanding of the performance of our agents.

Role of λ_{SPR} We studied the impact of the SPR loss coefficient, as it plays a very important role in scaling the magnitude of the updates coming from the SPR part of the network in the shared parameters. We tried different orders of magnitude for λ_{SPR} to cover a large range of SPR contribution scales. Our results have already been presented in our first experiment (Figures 4.1 and 4.2). They show that no value of λ_{SPR} is fundamentally better than the other as even very small values reach lower performances than the baseline and decreasing λ_{SPR} as close to 0 as possible seems to be the only way to not lose performance. We concluded that for Fully-Streaming SPR, the added self-predictive objective hurts the performance.

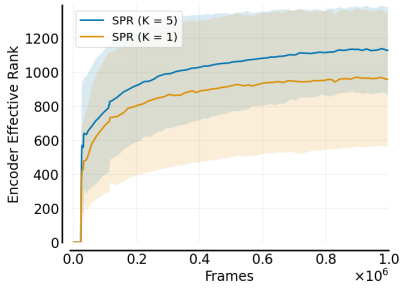
Choice of K = 5 In section 4.4 we mentioned that we ultimately chose $K = 5$ because it offered improved learning capabilities in harder environments. We indeed noticed that using $K = 5$ instead of $K = 1$ led to SPR agents being able to learn in the Enduro environment as shown in Figure 4.6a. Overall, the two variants of SPR have very close IQMs (Figure 4.6b), with $K = 1$ having a higher variance because of a smaller number of random seeds being ran. However, using $K = 5$ consistently yields representations with higher effective rank than when using $K = 1$ (Figure 4.6c).



(a) Episode return on *Enduro*. $K = 1$ does not lead to any learning while $K = 5$ does. Shaded area corresponds to the standard deviation.



(b) Comparison of the IQM. Both choices of K lead to very similar performance.



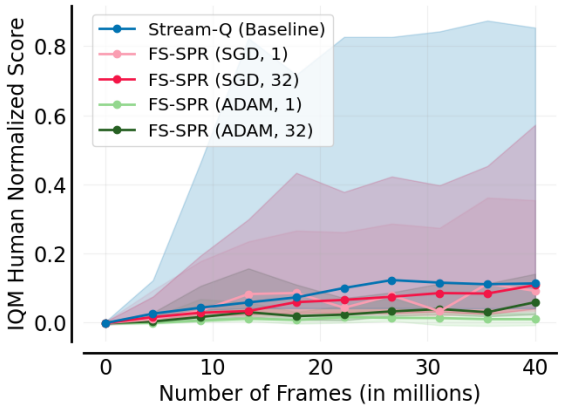
(c) Comparison of the effective rank. $K = 5$ leads to higher effective ranks than $K = 1$.

FIGURE 4.6 Comparison of *a)* the episode return on the Enduro environment, *b)* the IQM and *c)* the effective rank of encoder representations between two versions of SPR, one using $K = 1$ (2 seeds) and the other $K = 5$ (5 seeds).

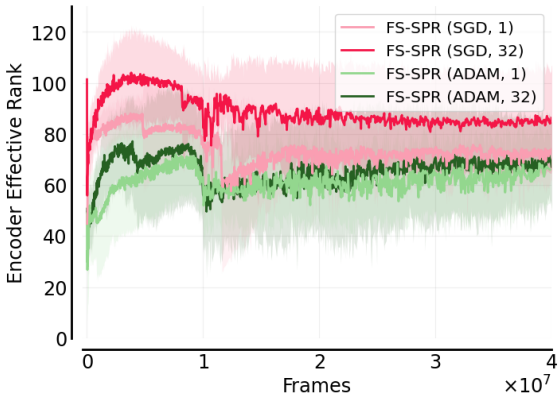
4.9.4 Optimization experiments

Gradient accumulation Since adding SPR updates to Stream-Q seems to hinder performance and that the SPR task is sensitive to non-stationarity, we studied the impact of

accumulating SPR gradients, only updating the SPR part of the network every other step. To do so, we kept a running average of the SPR updates that should have been applied to the network and only apply the average of the last 32 updates once every 32 steps. The updates from the Q-network are still applied every step, even to the shared parameters. It is important to note that this experiment was conducted in the setting described in the first paragraph of section 4.9.3 and as such, the effective ranks have completely different values. Figure 4.7a shows that although the Stream-Q baseline is still better than any of the Fully-Streaming SPR agent configurations we tried, using a gradient accumulation of 32 steps seemed to help agents achieve better performance IQM-wise, as well as better effective ranks (Figure 4.7b). This further builds our argument that SPR updates in the streaming setting seem harmful to the Q-learning objective. An interesting followup would be to investigate if these improvements are due to less correlated updates of the SPR part of the network or to the lower frequency of these updates. We did not end up using gradient accumulation in our 1M frames setting because of the decreased sampling efficiency. Indeed updating the SPR network only once every 32 steps seemed too little to learn good representations early enough in the training. After all, we want to add an SPR objective specifically to help the Stream-Q agent learn better representations faster.



(a) Comparison of the IQM. Although None of the Fully-Streaming SPR optimizer configuration beat the Stream-Q baseline, we observe that gradient accumulation of 32 steps lead to slightly better performances than without gradient accumulation.



(b) Comparison of the effective rank. Agents using gradient accumulation of 32 steps have representations with higher effective ranks.

FIGURE 4.7 Comparison of *a)* the IQM and *b)* the effective rank of encoder representations between different Fully-Streaming SPR agents using different gradient accumulation values and the Stream-Q baseline (2 seeds per experiment).

Gradient projection One of our concerns when coming up with a merging strategy for the updates of the shared parameters was that the two tasks may give updates that would be too dissimilar and would end-up interfering destructively. Following the ideas from the A-GEM [39] method for continual learning, we decided to project the updates from the SPR objective orthogonally onto the updates coming from the Q-learning objective. This should reduce the interferences caused by learning a task on learning the other. In practice, we did not observe much difference between this method and the simple averaging of the updates. We realized that the updates coming from the Q-learning objective and the SPR objective were already relatively orthogonal without any intervention (with a cosine similarity hovering between -0.02 and 0.05) as shown on Figure 4.8. This can explain why we did not see a difference when using gradient projection instead of averaging the updates. Interestingly, the results reveal that the updates are slightly more correlated when the Q-learning objective and the SPR objective are both trained from the same distribution (replay or stream).

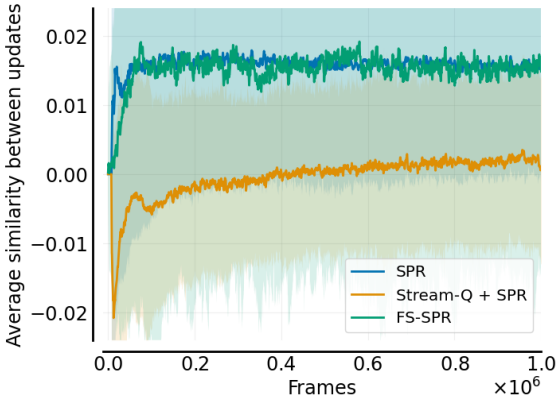


FIGURE 4.8 Average similarity between the updates coming from the SPR-objective and the Q-learning objective across all Fully-Streaming SPR, Stream-Q + SPR and SPR agents trained on the 10 environment we considered for the 4 optimizer combinations we considered. The similarity is very low (which could explain why projecting these gradients didn't have noticeable effect), even more so when both objectives are not trained from the same distribution (stream vs. replay).

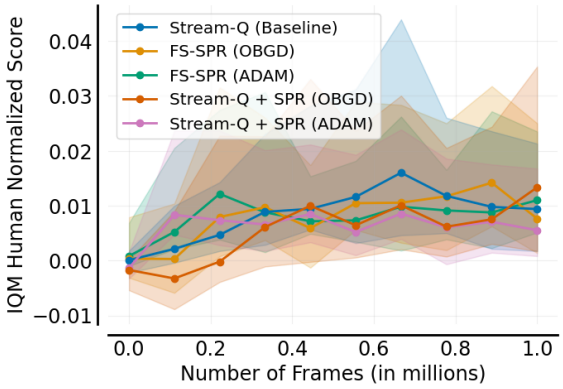
Two step optimization We also considered applying updates to the shared parameters from only one source of updates (Q-learning of SPR) and change which source every few steps. This is also known as two step optimization and it allows each task to be learned without interference while still being trained together. However, since this experiment was run in the setting used to perform the first hyperparameter search (defined in section 4.9.3), we did not get meaningful results.

Shrink and perturb We noticed that the parameter norm in the SPR parts of the network grew to very large values before stabilizing, which can decrease the effective learning rate by a lot, effectively slowing training in the SPR part of the network. This is a consequence of using layer normalization [67] before every activation function, even in the SPR part of the network. Indeed, LayerNorm introduces parameter-scale invariance which causes its gradients to be inversely proportional to the parameter norm. This means that in order to get more meaningful parameter changes, we should enforce lower parameter norms. Thus, in addition to trying AdamW, we investigated if shrink and perturb [54] could help improve performances. We chose to only apply shrink and perturb in the encoder and dynamics model of the SPR agent and excluded biases, as shrinking the parts of the Q-network responsible for predicting the Q-values affected the performances of our agents in a bad way in our experiments, preventing them to learn Q-values high enough to beat random policies. The shrink and perturb update consists in scaling down each parameter by a factor $\lambda \in]0, 1[$ and adding a gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$, giving the following update : $\mathrm{w} \leftarrow \lambda(\mathrm{w} + \alpha\Delta\mathrm{w}) + \epsilon$. For this experiment, we used a value of 0.99 for λ and 10^{-3} for σ^2.

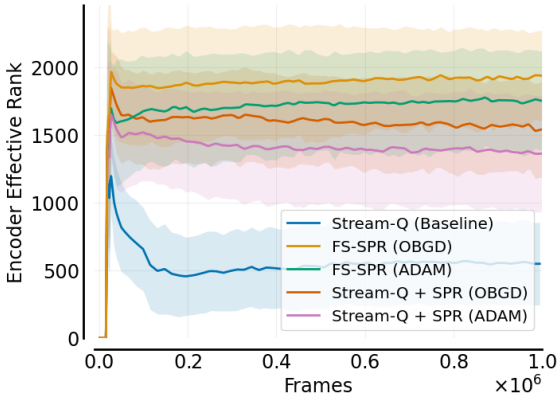
Figure 4.9a shows that no agent subjected to a shrink and perturb update manages to beat the Stream-Q + OBGD baseline. They all seem to have rather similar performances IQM-wise. Unfortunately adding shrink and perturb does not seem to have helped the optimization problem. Effective rank-wise, Figure 4.9b shows what appears to be the rank of an almost random encoder. Indeed, even though the perturbation part of the process has the benefit of moving the parameter vectors around slightly, thus allowing them to stumble on better regions of the weight space that they would otherwise not have reached, if this perturbation is too important the network may not be able to learn at all and be mostly random. Particularly, random matrices are known to have very high ranks as their rows/columns are not correlated. The high effective ranks we observe seem too high to be coming from a good encoder. We would however need to use a smaller value of σ^2 to confirm that shrink and perturb does not help our streaming agents get better performances.

4.9.5 Additional results

Getting rid of the non-stationarity In addition to experimenting with the effect of replacing SPR's objective by a distillation loss, we ran an alternative experiment to get rid of the non-stationarity in a different way. Indeed, in this experiment, a Fully-Streaming SPR agent learns representations from a completely random stationary policy. This corresponds to setting α_{ObGD} to zero in equation 4.1 and using an ϵ-greedy exploration scheme with $\epsilon = 1$ for the entirety of the training. With such a setting, not only do we get rid of the non-stationarity



(a) Comparison of the IQM. Every agent seems to behave very similarly IQM-wise, regardless of the optimizer used, with a tendency to under-perform compared to the Stream-Q baseline.



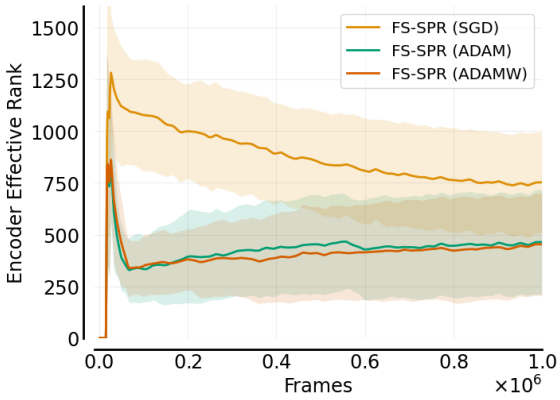
(b) Comparison of the effective rank. All agents have much higher effective ranks than the baseline. However, this seems to indicate that the perturbation is too large, thereby inducing more random representations which will have higher rank.

FIGURE 4.9 Comparison of *a)* the IQM and *b)* the effective rank of encoder representations between different agents subjected to a shrink and perturb update and the Stream-Q baseline (no shrinking or perturbation of the weights) on 2 random seeds.

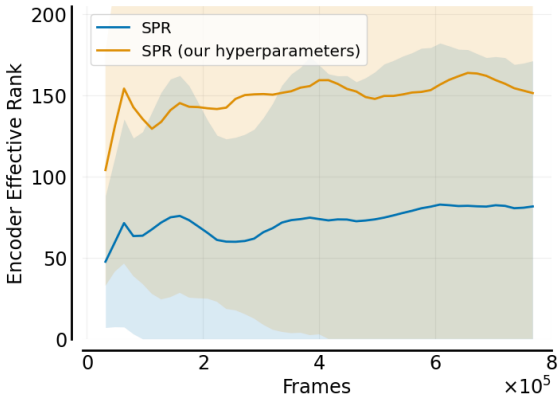
of the input observations given to SPR as the policy is not updated over time, but we also get rid of any possible interference between the Q-updates and the SPR-updates. However, since the policy is random, it is likely to span only a restricted subset of the state-space, and as such it is more likely that training a SPR agent would enter an overfitting regime. Moreover, [11] state that the SPR objective is prone to representation collapse but that the Q-learning objective should act as a regularizer preventing said collapse. Therefore, although this experiment does get rid of most of the non-stationarity, the learning dynamics it induces are potentially not suitable for training. To assess the quality of the representations learned, we compare the effective rank of the Fully-Streaming SPR agent to the effective rank obtained from the original SPR setting in a similar setting (we use a regular replay buffer instead of PER because the policy is not updated, and thus the TD-errors never change).

The results of this experiment are shown on Figure 4.10. We can see that the effective rank of the learned representations collapses slowly in the case of SGD, but both Adam and AdamW seem to perform similarly, with a mediocre but increasing effective rank. However, in its original setting, SPR does not seem to learn any meaningful representations as the effective rank of the encoder's outputs stays constant at a relatively low value. This indicates that even though Fully-Streaming SPR with stationary inputs and without interferences from the

Q-network seems to be stable enough and does not collapse, our concerns about the poor learning dynamics are confirmed. This experimental setup is not the most meaningful way to get rid of the non-stationarity. For this reason we ran our distillation experiment to get a clearer picture.



(a) Results using our SPR setting. Although every optimizer seems to undergo some sort of representation collapse at the very beginning, the rank stabilizes and slowly increases afterwards (except for SGD), showing that streaming SPR can learn representations without collapse, but not very good ones.

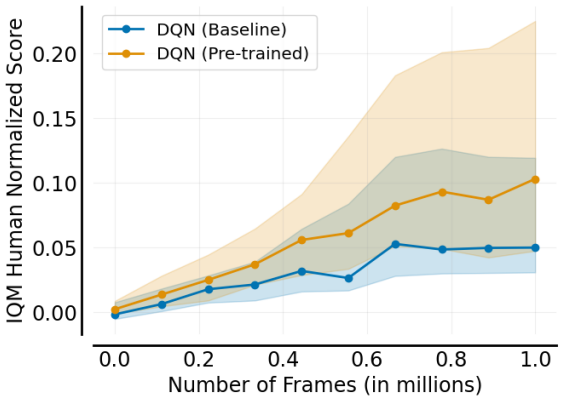


(b) Results using SPR's original setting. Here, no matter the setting, SPR didn't learn any meaningful representations. This highlights the poor learning dynamics induced by this experiment.

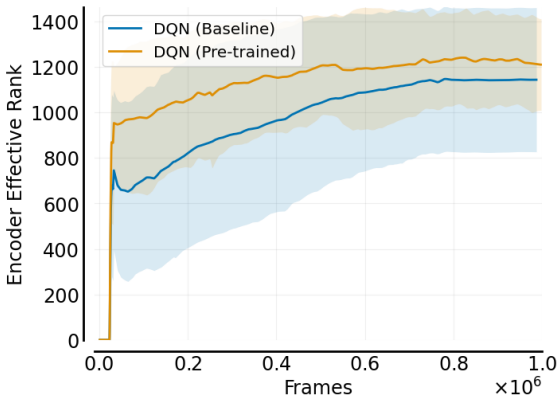
FIGURE 4.10 Comparison of the effective ranks of the representations produced by the encoder of a SPR agent using both *a)* our setting and *b)* the original SPR setting. Here, only the SPR-part of the agent's network is updated and the policy is random ($\epsilon = 1$) (2 seeds per experiment).

Pretrained encoders for DQN As mentioned in section 4.6.2, we further verify that the representations learned by our teacher network (non-streaming SPR agent using our setting) are of good quality. We run the same experimental setup as in our **Pretrained encoders** experiment, except that we train a DQN agent instead of a Stream-Q + SPR agent. Both the baseline and the agent initialized with pretrained weights are trained with replay and a target network. Figure 4.11 confirms the quality of the representations learned by our teacher network as the DQN agent initialized with pretrained weights performs better than its vanilla version.

Additional pre-training experiments First, Figure 4.12 presents the results of the same pre-trained encoder experiment as described in section 4.6.2 (except for only 2 random seeds),



(a) Comparison of the IQM. The DQN agent initialized with (unfrozen) pretrained encoder weights learns faster than its trained from scratch version, confirming the quality of the representations learned by our teacher network.



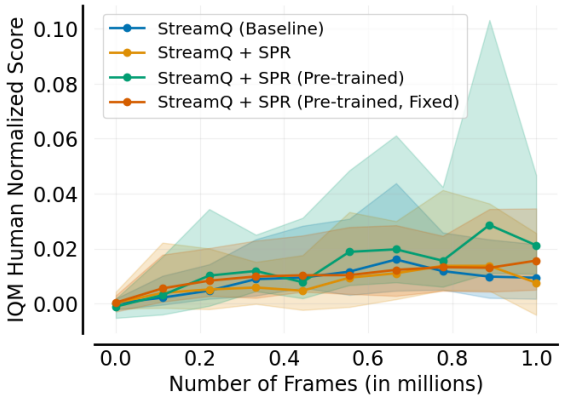
(b) Comparison of the effective rank. Similarly to what we observe for the IQM, the effective rank of the model using pretrained weights is higher, further confirming the quality of our teacher network.

FIGURE 4.11 Comparison of *a)* the IQM and *b)* the effective rank of encoder representations between regular DQN and a DQN agent whose encoder weights have been initialized from the same pretrained agent as in our distillation experiment. Both algorithms were ran with replay and a target network (hard updates every 4000 steps) on 2 random seeds.

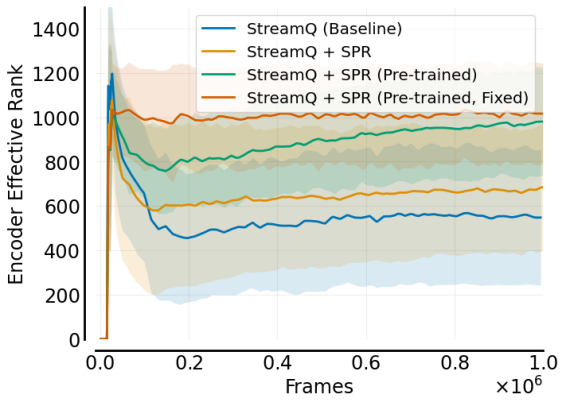
but where we also run an experiment where the encoder weights are frozen after being initialized to that of a pre-trained SPR agent. We observe that representations keep a rather high rank (Figure 4.12b), as one could expect since the encoder is frozen, but the performance IQM-wise are pretty bad (Figure 4.12a). This shows that although representations learned by SPR are useful to a streaming agent, the learning dynamics to get them is just as important, if not more.

Second, as mentioned in section 4.6.2, in addition to trying to run a Stream-Q agent with pre-trained encoder weights, we also ran this experiment using pre-trained encoder weights from SPR agents on Stream-Q + SPR agents, effectively adding a representation learning task to the pre-trained Stream-Q experiment. Figure 4.13 shows that adding this SPR task completely nullified the performance gained by using pre-trained representations for ObGD and SGD (Figure 4.20). Adam and AdamW on the other hand greatly benefit from adding the SPR objective on top of pre-trained representations. Rank-wise, we can see on Figure 4.21 that while for ObGD and SGD, the effective rank of the agents using pre-trained representations are overall higher than (or equal to) both other baselines, this rank stays constant and does not increase like it did as shown on Figure 4.21. Again, the opposite situation occurs for

Adam and AdamW for which the effective rank benefits from the added SPR objective. All in all, these trends further show that Q-Learning and SPR are incompatible in the streaming setting under some conditions, seemingly dictated by the way the optimization problem is framed.



(a) Comparison of the IQM. Although a Stream-Q + SPR agent initialized with *fixed* pretrained encoder weights does not improve on the Stream-Q baseline, the same agent with unfrozen weights outperforms the baseline : the representations learned by SPR are valuable to Stream-Q but Stream-Q needs to be able to alter the initial representations in order to work properly.



(b) Comparison of the effective rank. Stream-Q + SPR agents initialized with pretrained encoder weights maintain the effective rank of their encoder's representations high : the representations learned by SPR are valuable to Stream-Q.

FIGURE 4.12 Comparison of *a)* the IQM and *b)* the effective rank of encoder representations between Stream-Q and a Stream-Q agent whose encoder weights have been initialized from a pretrained agent. Specifically, we are interested in the case where these weights are then frozen. All agents use the ObGD optimizer for the Q-learning part of the network and Adam for the SPR-part. 2 random seeds were used.

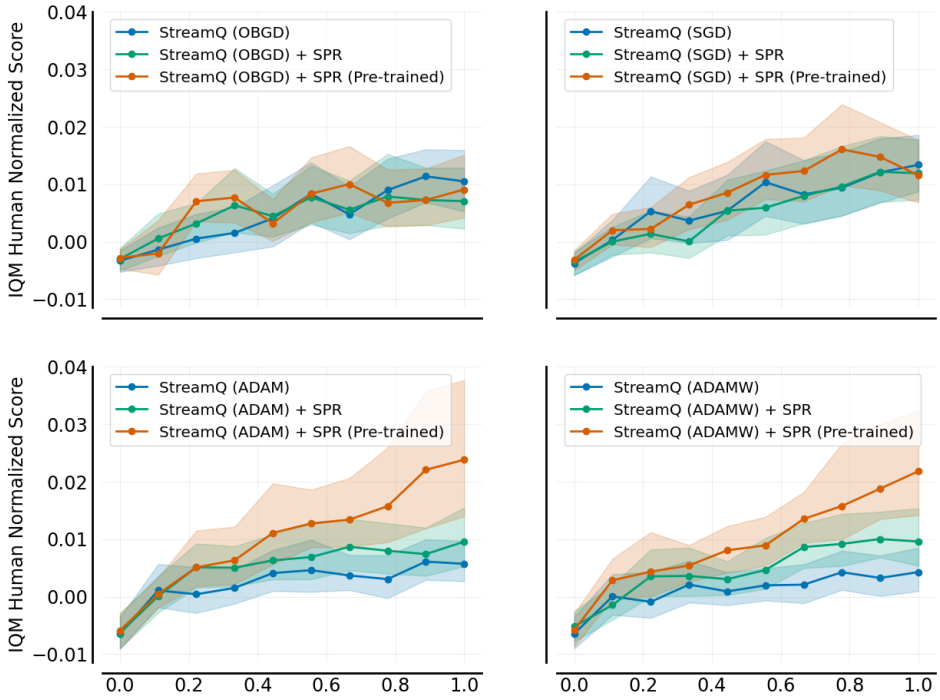


FIGURE 4.13 Comparison of the IQM across multiple optimizer configurations between the Stream-Q baseline, Stream-Q + SPR and a Stream-Q + SPR agent whose encoder weights have been initialized from a pretrained agent. In this setting, adding the SPR objective to a Stream-Q agent initialized with pre-trained encoder weights kills all performance gains from the pre-trained representations for both ObGD and SGD. For Adam and AdamW this greatly benefits performance, showcasing the importance of the role played by the choice of optimizers and how more "aligned" setups can benefit streaming SPR.

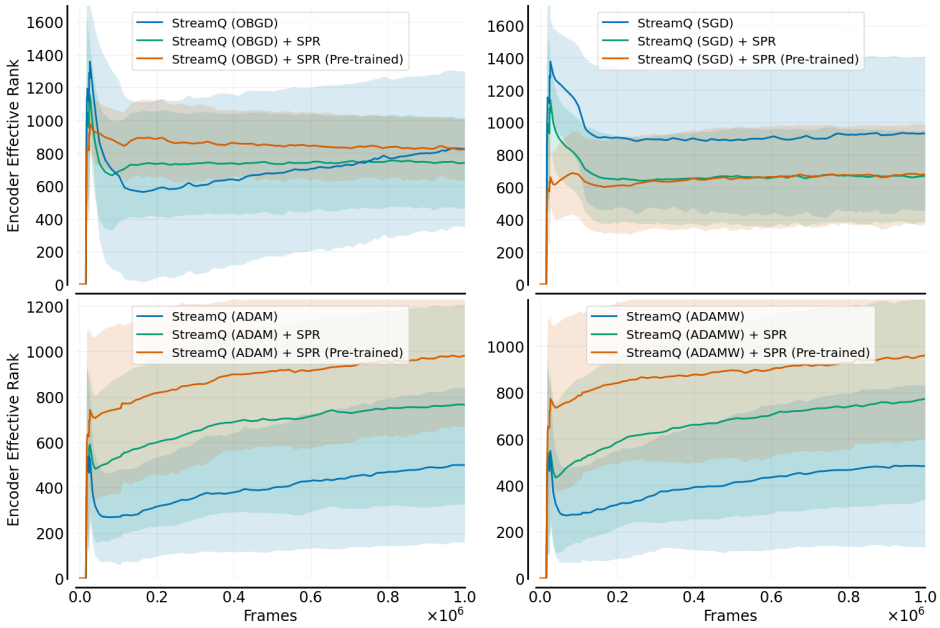
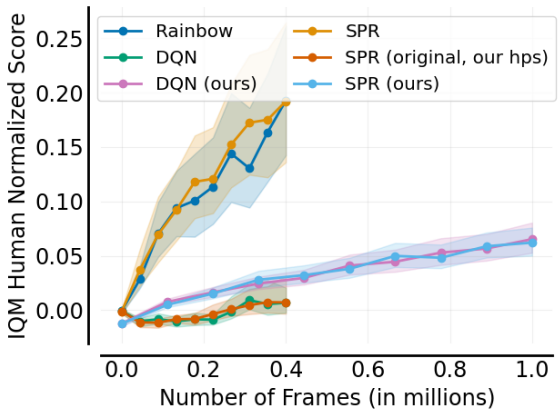
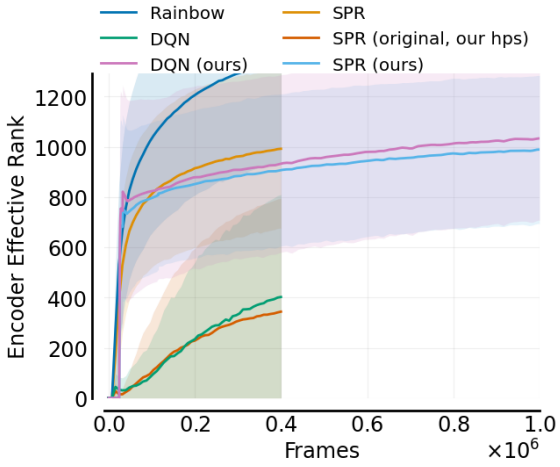


FIGURE 4.14 Comparison of the effective rank of the representations produced by the encoder across multiple optimizers between the Stream-Q baseline, Stream-Q + SPR and a Stream-Q + SPR agent whose encoder weights have been initialized from a pretrained agent. Most optimizers show a clear trend in which Stream-Q + SPR agents with pretrained weights learn representations with higher ranks than those without pre-trained weights, regardless of the optimizer (except for SGD).



(a) Comparison of the IQM. While SPR and Rainbow perform much better than DQN and SPR without any Rainbow improvements in our setting, our versions of DQN and SPR perform better than their equivalent in the original setting. This confirms the quality of our teacher networks.



(b) Comparison of the effective rank. Rainbow learns the representations with the highest rank, closely followed by both the original and our SPR settings, as well as our DQN setting. Dead last are DQN and SPR equivalents to ours in the original setting. This confirms the quality of the representations learned by our teacher networks.

FIGURE 4.15 Comparison of *a)* the IQM and *b)* the effective rank of encoder representations between our and SPR's original setting on 15 chosen environments. "*ours*" refers to agents ran in our setting, while "*original, our hps*" refers to agents trained in the original SPR setting, with a replay ratio of 2 and no Rainbow improvement.

4.9.6 Plots on all optimizers

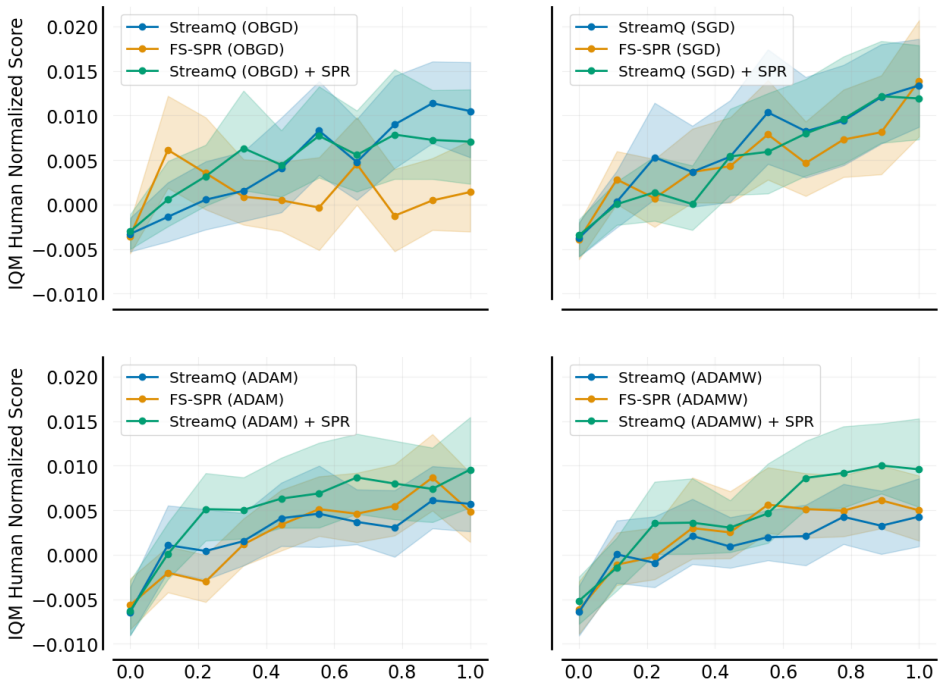


FIGURE 4.16 Comparison of the IQM across multiple optimizer configurations between the Stream-Q baseline, Fully-Streaming SPR and Stream-Q + SPR, with a loss coefficient $\lambda_{SPR} = 2$. Although Adam and AdamW seem to lead to improvements when adding a streaming SPR objective, none of them beat Stream-Q + ObGD. Stream-Q + SPR seem to provide very marginal, if any, improvements over FS-SPR performances.

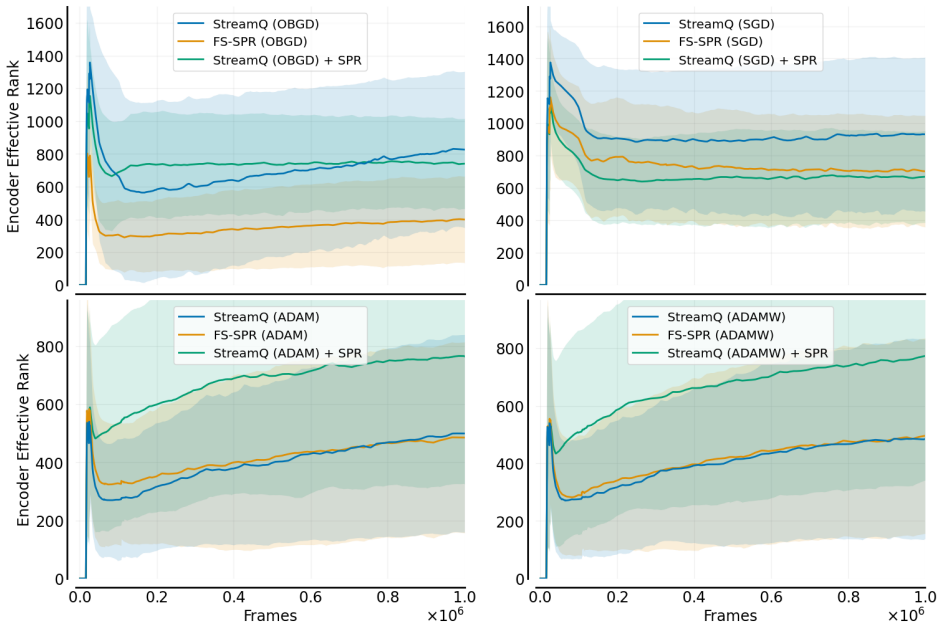


FIGURE 4.17 Comparison of the effective rank of the representations produced by the encoder across multiple optimizers between the Stream-Q baseline, Fully-Streaming SPR and Stream-Q + SPR, with a loss coefficient $\lambda_{SPR} = 2$. Most optimizers show a clear trend in which more stable SPR objectives lead to representations with higher ranks. Exceptions are noted for SGD, in which Stream-Q + SPR does not improve representations, and ObGD in which FS-SPR gives worse representations than Stream-Q alone and Stream-Q + SPR is only better at the beginning of the training.

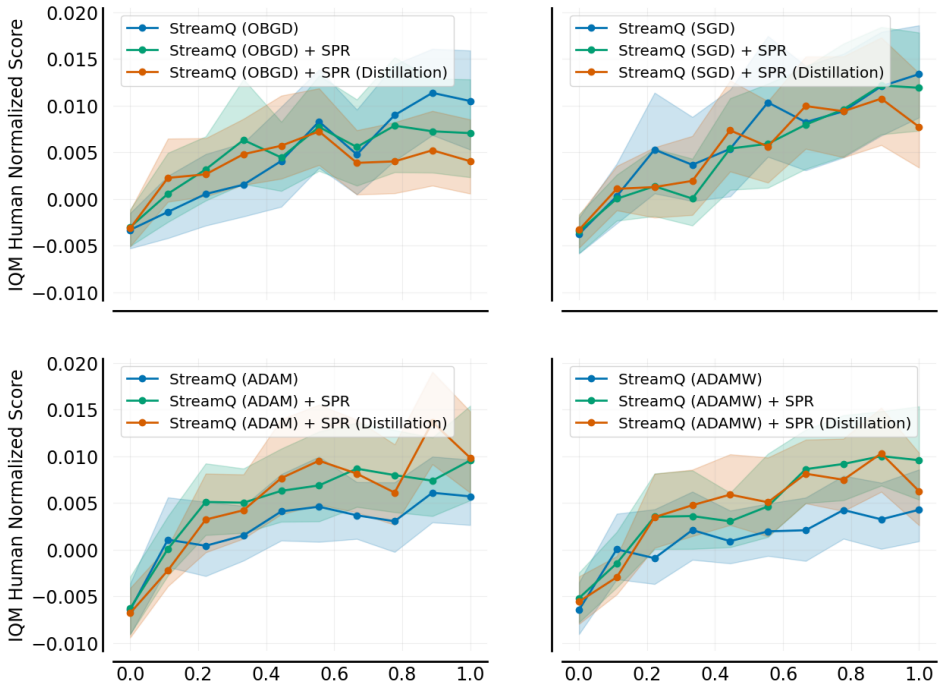


FIGURE 4.18 Comparison of the IQM across multiple optimizer configurations between the Stream-Q baseline, Stream-Q + SPR and Stream-Q + SPR with a distillation loss for the SPR objective. Although Adam and AdamW seem to lead to improvements when adding a streaming SPR objective (regardless of the distillation objective), none of them beat Stream-Q + ObGD.

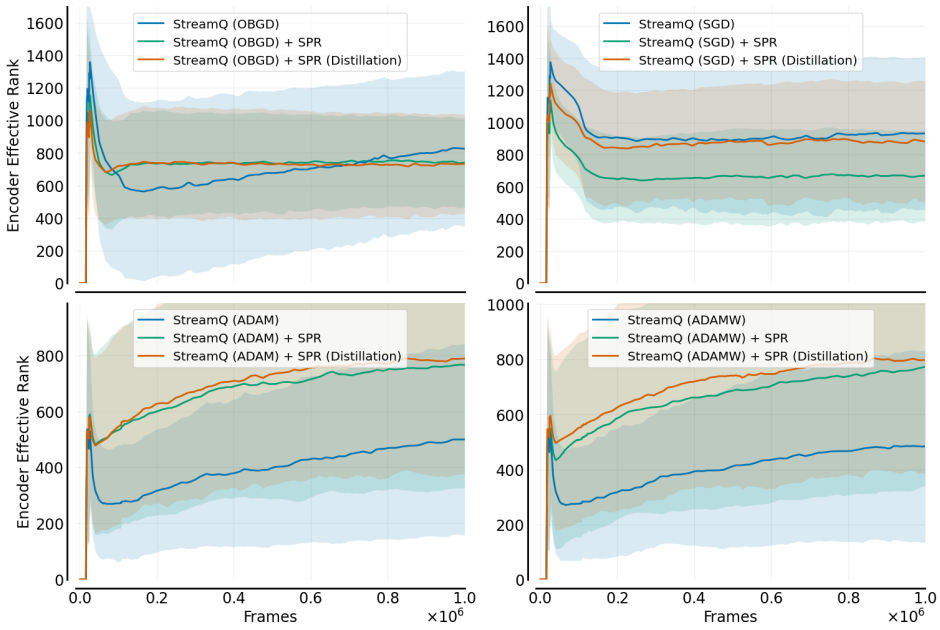


FIGURE 4.19 Comparison of the effective rank of the representations produced by the encoder across multiple optimizers between the Stream-Q baseline, Stream-Q + SPR and Stream-Q + SPR with a distillation loss for the SPR objective. Most optimizers show a shy improvement the rank of representations when the SPR objective becomes more stable. Stream-Q (ObGD) + SPR does not benefit from the increased stability of the distillation objective.

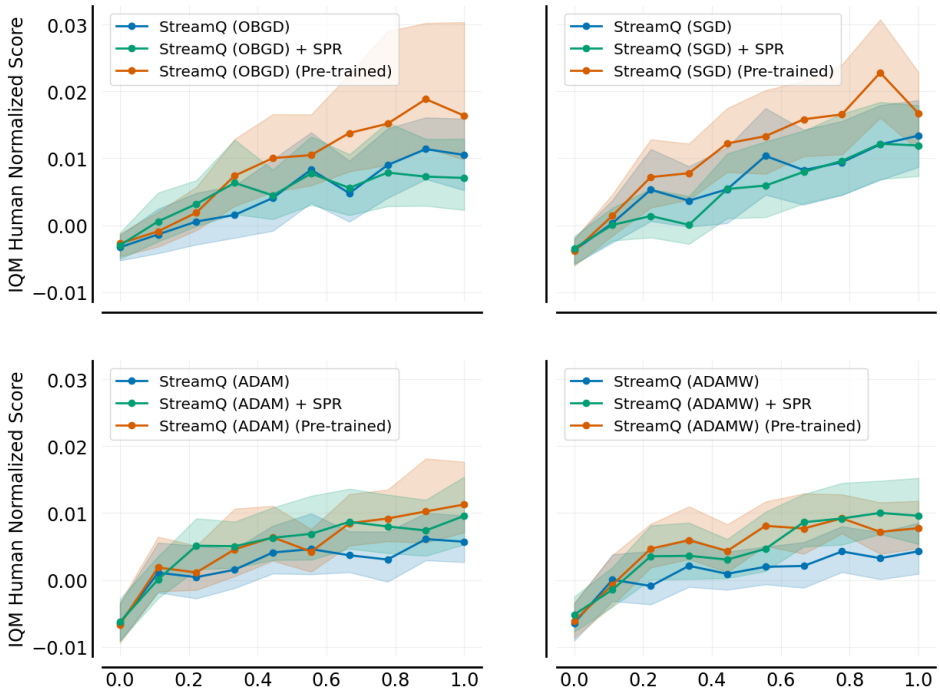


FIGURE 4.20 Comparison of the IQM across multiple optimizer configurations between the Stream-Q baseline, Stream-Q + SPR and a Stream-Q + SPR agent whose encoder weights have been initialized from a pretrained agent. A Stream-Q agent initialized with pretrained encoder weights visibly outperforms the Stream-Q baseline : the representations learned by SPR are valuable to Stream-Q. More importantly, for ObGD and SGD, Stream-Q with pre-trained encoder outperforms Stream-Q + SPR, highlighting the detrimental nature of SPR for Stream-Q with certain optimizers.

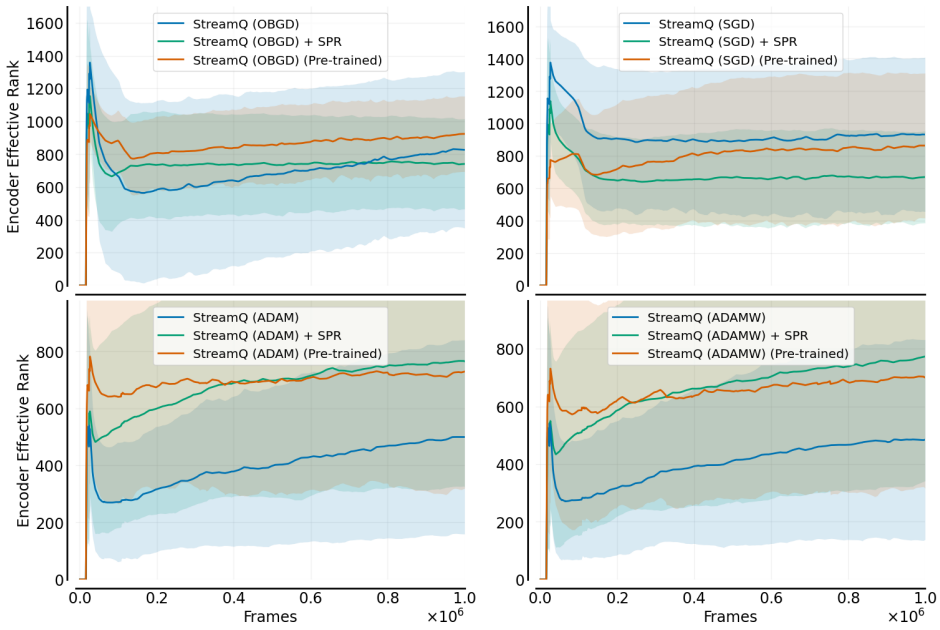


FIGURE 4.21 Comparison of the effective rank of the representations produced by the encoder across multiple optimizers between the Stream-Q baseline, Stream-Q + SPR and a Stream-Q + SPR agent whose encoder weights have been initialized from a pretrained agent. Stream-Q agents initialized with pretrained encoder weights maintain the effective rank of their encoder's representations high : the representations learned by SPR are valuable to Stream-Q.

CHAPTER 5 GENERAL DISCUSSION

The research presented in this paper is an empirical study of the learning dynamics of unsupervised learning objectives in the streaming RL setting. We used a notorious representation learning method for reinforcement learning, SPR [11], that we combined with the recent streaming RL agent from [6]. To our knowledge, no other papers focused on adapting representation learning methods to the realm of deep streaming RL. We used optimization ideas from the continual learning literature [39, 54] and presented several others that were not tested in the paper [40, 62, 63].

As for our contribution, through a series of carefully crafted experiments, we attempted to determine what were the root causes of our failed attempts at learning streaming representations for reinforcement learning. The non-stationarity of the problem, although it was the first culprit we had in mind, appeared to not be the only cause preventing the agent to learn anything beyond random exploration. Instead, the very essence of the representation learning task, learned jointly with the streaming Q-learning objective seems to be the issue. Unfortunately, our analysis cannot provide an explanation of what causes this divergent learning dynamics, although it highlights the issue at hand where future works will need to investigate further. It is also important to note that most works tackling representation learning for RL are empirical papers, and very few [68, 69] analyze the theoretical reasons that make adding an unsupervised learning task to RL such a beneficial endeavor. We did not tackle the optimization problem from a theoretical perspective in this research, but our results strongly suggest that such an analysis is needed to progress further in the domain. We believe that such a contribution would be very helpful not only to the field of deep streaming reinforcement learning but of continual learning as well. As such, we recommend looking into optimization methods designed to improve plasticity in the continual learning literature [70].

CHAPTER 6 CONCLUSION

6.1 Summary of the work done

In this thesis, we presented the work we did on investigating the reasons why learning streaming representations for streaming deep reinforcement learning does not work when directly translating existing representation learning methods to the streaming setting of RL. This work led to the submission of our article *The Challenges of Learning Streaming Representations for Reinforcement Learning*. As such, this article constitutes the core of the thesis. We also expanded our literature review to cover the most relevant papers in more details than in the paper.

Our main contribution is a detailed empirical analysis of the reasons why directly translating the Self-Predictive Representation (SPR) method from [11] to the streaming context fails. We started by investigating the role of non-stationarity in this failure, as streaming RL prevents the use of standard mitigation measures, such as replay buffers and target networks. Through a series of experiments, gradually removing some level of non-stationarity, we showed that although non-stationarity plays a role in the bad performance we observed, it is not the sole factor. Indeed, our results suggest that this failure to learn stems from the optimization problem itself : learning the unsupervised task along with the Q-learning task in of itself seems to be a cause of the poor performance of our agents. We confirmed this through two sets of experiments, one showing that an agent learning its representation learning objective through a distillation loss from a pre-trained agent does not improve its performance. Second, we showed that when removing the representation learning objective, a streaming agent using the pre-trained encoder weights from the same teacher network as the one used in the distillation experiment performs better than its baseline. Thus, our results suggest that even though we removed an important amount of non-stationarity and verified that the representation learning task learns representations that are good for streaming agents too, the two objectives interfere in a detrimental way for the agent when trained jointly. In other words, learning self-predictive representations from a stream of transitions requires more sophisticated optimizations techniques and requires a better understanding of the how the learning dynamics of each task interacts with the other's.

6.2 Limitations

Several limitations in our work were described in the paper as well as the general discussion. We summarize them in this section. First, we chose to take a next-latent state prediction depth of $K = 5$ for the SPR task, which requires storing the last 5 transitions and is therefore not in complete agreement with the strict streaming RL requirements of discarding every transition right after they have been used. The reasons behind this choice have been addressed in the background section of the paper (section 4.4) and figure 4.6 in Appendix 4.9.3. We also did not study the impact of using target networks as a relaxation of the streaming setting and leave this for future works. Finally, we chose to use the standard DQN architecture as our base for SPR, thus not considering the use of a dueling architecture or of a categorical loss. Finally, as mentioned in the general discussion, our approach is entirely empirical, and a followup, theoretical, analysis of the learning dynamics of representations in the streaming setting would be greatly beneficial.

6.3 Future directions

As mentioned through this thesis, optimization is at the core of the issue of learning streaming representations for RL. As such, we strongly recommend that this be the main focus of future works on the matter. Two main directions can be explored in parallel : expanding adaptive learning rates, in an ObGD fashion, to the representation learning part of the network, and investigating new optimizers, better suited for streaming updates. The first idea could improve the dynamics of the joint training of both the representation learning and Q-learning tasks as with such real-time control of the learning rate of the whole network, updates from the SPR-part of the network that would harm the Q-learning performances could be down-scaled, and conversely when the updates from the two parts of the network are more aligned. As for the study of other optimizers, we suggest looking at Sharpness Aware Minimization (SAM) [62] to promote flatter regions of the loss surface, and investigate if this leads to easier joint learning dynamics as there would be more room for interference without exiting the local minimum. Another promising optimizer idea is to use orthogonal gradients [40] to de-emphasize highly correlated updates being performed one after the other, as is often the case in RL, especially in the streaming setting. This very recent work has the added benefit of having been designed specifically for learning streaming representations. Finally, we have not studied the impact of neuron dormancy on our results. We believe that this added analysis layer could give valuable insights on the learning dynamics of the problem.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones *et al.*, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. [Online]. Available : https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[2] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss *et al.*, "Zero-shot text-to-image generation," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8821–8831. [Online]. Available : https://proceedings.mlr.press/v139/ramesh21a.html

[3] C. Zeni, R. Pinsler, D. Zügner, A. Fowler, M. Horton *et al.*, "A generative model for inorganic materials design," *Nature*, vol. 639, no. 8055, pp. 624–632, Mar. 2025. [Online]. Available : https://doi.org/10.1038/s41586-025-08628-5

[4] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre *et al.*, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, Dec. 2020. [Online]. Available : https://doi.org/10.1038/s41586-020-03051-4

[5] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung *et al.*, "Dota 2 with Large Scale Deep Reinforcement Learning," Dec. 2019. [Online]. Available : https://arxiv.org/abs/1912.06680v1

[6] M. Elsayed, G. Vasan, and A. R. Mahmood, "Streaming Deep Reinforcement Learning Finally Works," Dec. 2024. [Online]. Available : https://arxiv.org/abs/2410.14606

[7] G. Vasan, M. Elsayed, A. Azimi, J. He, F. Shariar *et al.*, "Deep Policy Gradient Methods Without Batch Updates, Target Networks, or Replay Buffers," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet *et al.*, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 845–891. [Online]. Available : https://proceedings.neurips.cc/paper_files/paper/2024/file/019ef89617d539b15ed610ce8d1b76e1-Paper-Conference.pdf

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning : An Introduction*, the mit press ed., 2018.

[9] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond *et al.*, "Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 21 271–21 284.

[10] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A Simple Framework for Contrastive Learning of Visual Representations," in *Proceedings of the 37th International Conference on Machine Learning*. PMLR, Nov. 2020, pp. 1597–1607, iSSN : 2640-3498. [Online]. Available : https://proceedings.mlr.press/v119/chen20j.html

[11] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. Courville *et al.*, "Data-efficient reinforcement learning with self-predictive representations," in *International Conference on Learning Representations*, 2021. [Online]. Available : https://openreview.net/forum?id=uCQfPZwRaUu

[12] M. Laskin, A. Srinivas, and P. Abbeel, "CURL : Contrastive Unsupervised Representations for Reinforcement Learning," in *Proceedings of the 37th International Conference on Machine Learning*. PMLR, Nov. 2020, pp. 5639–5650, iSSN : 2640-3498.

[13] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse control tasks through world models," *Nature*, vol. 640, no. 8059, pp. 647–653, Apr. 2025. [Online]. Available : https://doi.org/10.1038/s41586-025-08744-2

[14] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," Jul. 2016. [Online]. Available : https://arxiv.org/abs/1607.06450

[15] Y. Zhao, W. Zhao, R. Boney, J. Kannala, and J. Pajarinen, "Simplified temporal consistency reinforcement learning," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato *et al.*, Eds., vol. 202. PMLR, 23–29 Jul 2023, pp. 42 227–42 246. [Online]. Available : https://proceedings.mlr.press/v202/zhao23k.html

[16] R. Zheng, X. Wang, Y. Sun, S. Ma, J. Zhao *et al.*, "*TACO* : Temporal latent action-driven contrastive loss for visual reinforcement learning," in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Online]. Available : https://openreview.net/forum?id=ezCsMOy1w9

[17] A. Scannell, K. Kujanpää, Y. Zhao, M. Nakhaei, A. Solin *et al.*, "Quantized representations prevent dimensional collapse in self-predictive RL," in *ICML Workshop*

on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET), 2024. [Online]. Available : https://openreview.net/forum?id=XYayXLuTXe

[18] T. Ni, B. Eysenbach, E. SeyedSalehi, M. Ma, C. Gehring *et al.*, "Bridging state and history representations : Understanding self-predictive RL," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available : https://openreview.net/forum?id=ms0VgzSGF2

[19] S. Fujimoto, P. D'Oro, A. Zhang, Y. Tian, and M. Rabbat, "Towards general-purpose model-free reinforcement learning," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available : https://openreview.net/forum?id=R1hIXdST22

[20] D. P. Kingma and J. Ba, "Adam : A Method for Stochastic Optimization," *International Conference on Learning Representations*, 2015. [Online]. Available : https://arxiv.org/abs/1412.6980

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[22] Q. Lan and A. R. Mahmood, "Elephant Neural Networks : Born to Be a Continual Learner," Oct. 2023. [Online]. Available : http://arxiv.org/abs/2310.01365

[23] A. K. Kearney, "Letting the Agent Take the Wheel : Principles for Constructive and Predictive Knowledge," Ph.D. dissertation, University of Alberta, 2023. [Online]. Available : https://era.library.ualberta.ca/items/7eaa9f9a-4e72-40a1-8dd6-0939d6cd3112

[24] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic : Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *Proceedings of the 35th International Conference on Machine Learning*. PMLR, Jul. 2018, pp. 1861–1870, iSSN : 2640-3498.

[25] Z. Ahmed, N. L. Roux, M. Norouzi, and D. Schuurmans, "Understanding the Impact of Entropy on Policy Optimization," in *Proceedings of the 36th International Conference on Machine Learning*. PMLR, May 2019, pp. 151–160, iSSN : 2640-3498. [Online]. Available : https://proceedings.mlr.press/v97/ahmed19a.html

[26] J. Bjorck, C. P. Gomes, and K. Q. Weinberger, "Is high variance unavoidable in RL ? a case study in continuous control," in *International Conference on Learning Representations*, 2022. [Online]. Available : https://openreview.net/forum?id=9xhgmsNVHu

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez *et al.*, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available : http://arxiv.org/abs/1509.02971

[28] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez *et al.*, "dm_control : Software and tasks for continuous control," *Software Impacts*, vol. 6, p. 100022, 2020. [Online]. Available : https://www.sciencedirect.com/science/article/pii/S2665963820300099

[29] S. Fujimoto, H. Hoof, and D. Meger, "Addressing Function Approximation Error in Actor-Critic Methods," in *Proceedings of the 35th International Conference on Machine Learning*. PMLR, Jul. 2018, pp. 1587–1596, iSSN : 2640-3498. [Online]. Available : https://proceedings.mlr.press/v80/fujimoto18a.html

[30] P. Dayan, "Improving generalization for temporal difference learning : The successor representation," *Neural Computation*, vol. 5, no. 4, pp. 613–624, 1993.

[31] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul *et al.*, "Successor Features for Transfer in Reinforcement Learning," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus *et al.*, Eds., vol. 30. Curran Associates, Inc., 2017.

[32] C. Ma, D. R. Ashley, J. Wen, and Y. Bengio, "Universal Successor Features for Transfer Reinforcement Learning," Jan. 2020. [Online]. Available : https://arxiv.org/abs/2001.04025

[33] R. Chua, A. Ghosh, C. Kaplanis, B. A. Richards, and D. Precup, "Learning Successor Features the Simple Way," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet *et al.*, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 49 957–50 030. [Online]. Available : https://proceedings.neurips.cc/paper_files/paper/2024/file/597254dc45be8c166d3ccf0ba2d56325-Paper-Conference.pdf

[34] S. Dohare, J. F. Hernandez-Garcia, Q. Lan, P. Rahman, A. R. Mahmood *et al.*, "Loss of plasticity in deep continual learning," *Nature*, vol. 632, no. 8026, pp. 768–774, Aug. 2024, publisher : Nature Publishing Group.

[35] M. McCloskey and N. J. Cohen, "Catastrophic Interference in Connectionist Networks : The Sequential Learning Problem," in *Psychology of Learning and Motivation*, G. H. Bower, Ed. Academic Press, Jan. 1989, vol. 24, pp. 109–165.

[36] S. Purushwalkam, P. Morgado, and A. Gupta, "The challenges of continuous self-supervised learning," in *Computer Vision – ECCV 2022*, S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, and T. Hassner, Eds. Cham : Springer Nature Switzerland, 2022, pp. 702–721.

[37] A. Krawczyk, B. Bagus, Y. Denker, and A. Gepperth, "Continual Reinforcement Learning Without Replay Buffers," in *2024 IEEE 12th International Conference on Intelligent Systems (IS)*, Aug. 2024, pp. 1–9, iSSN : 2767-9802.

[38] D. Levenstein, A. Efremov, R. H. Eyono, A. Peyrache, and B. Richards, "Sequential predictive learning is a unifying theory for hippocampal representation and replay," *bioRxiv*, 2024. [Online]. Available : https://www.biorxiv.org/content/early/2024/06/04/2024.04.28.591528

[39] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with a-GEM," in *International Conference on Learning Representations*, 2019. [Online]. Available : https://openreview.net/forum?id=Hkf2_sC5FX

[40] T. Han, D. Gokay, J. Heyward, C. Zhang, D. Zoran *et al.*, "Learning from streaming video with orthogonal gradients," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, June 2025, pp. 13 651–13 660.

[41] A. Prabhu, S. Sinha, P. Kumaraguru, P. H. Torr, O. Sener *et al.*, "RanDumb : Random Representations Outperform Online Continually Learned Representations," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet *et al.*, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 37 988–38 006.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou *et al.*, "Playing Atari with Deep Reinforcement Learning," Dec. 2013, nIPS Deep Learning Workshop 2013. [Online]. Available : https://arxiv.org/abs/1312.5602

[43] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, Mar. 2016.

[44] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski *et al.*, "Rainbow : combining improvements in deep reinforcement learning," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances*

in Artificial Intelligence, ser. AAAI'18/IAAI'18/EAAI'18. New Orleans, Louisiana, USA : AAAI Press, 2018, pp. 3215–3222.

[45] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang, "The 37 Implementation Details of Proximal Policy Optimization," 2022. [Online]. Available : https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

[46] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo *et al.*, "Reinforcement learning with unsupervised auxiliary tasks," in *International Conference on Learning Representations*, 2017. [Online]. Available : https://openreview.net/forum?id=SJ6yPD5xg

[47] M. Tomar, U. A. Mishra, A. Zhang, and M. E. Taylor, "Learning Representations for Pixel-based Control : What Matters and Why ?" *Transactions on Machine Learning Research*, 2023. [Online]. Available : https://openreview.net/forum?id=wIXHG8LZ2w

[48] E. J. Meyer, A. White, and M. C. Machado, "Harnessing discrete representations for continual reinforcement learning," *Reinforcement Learning Journal*, vol. 2, pp. 606–628, 2024.

[49] B. Bagus and A. Gepperth, "A Study of Continual Learning Methods for Q-Learning," in *2022 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2022, pp. 1–9, arXiv :2206.03934 [cs].

[50] K. Javed, A. Sharifnassab, and R. S. Sutton, "SwiftTD : A Fast and Robust Algorithm for Temporal Difference Learning," *Reinforcement Learning Journal*, vol. 2, pp. 840–863, 2024.

[51] M. L. Puterman, *Markov Decision Processes : Discrete Stochastic Dynamic Programming*, 1st ed. USA : John Wiley & Sons, Inc., 1994.

[52] L. Armijo, "Minimization of functions having Lipschitz continuous first partial derivatives." *Pacific Journal of Mathematics*, vol. 16, no. 1, pp. 1–3, Jan. 1966, publisher : Pacific Journal of Mathematics, A Non-profit Corporation.

[53] A. Zhang, R. T. McAllister, R. Calandra, Y. Gal, and S. Levine, "Learning invariant representations for reinforcement learning without reconstruction," in *International Conference on Learning Representations*, 2021. [Online]. Available : https://openreview.net/forum?id=-2FCwDKRREu

[54] J. Ash and R. P. Adams, "On Warm-Starting Neural Network Training," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 3884–3894.

[55] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *International Conference on Learning Representations*, 2019. [Online]. Available : https://openreview.net/forum?id=Bkg6RiCqY7

[56] M. Fortunato, M. G. Azar, B. Piot, J. Menick, M. Hessel *et al.*, "Noisy networks for exploration," in *International Conference on Learning Representations*, 2018. [Online]. Available : https://openreview.net/forum?id=rywHCPkAW

[57] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, and M. G. Bellemare, "Deep Reinforcement Learning at the Edge of the Statistical Precipice," *Advances in Neural Information Processing Systems*, 2021, original-date : 2021-08-20T00 :41 :06Z. [Online]. Available : https://github.com/google-research/rliable

[58] O. Roy and M. Vetterli, "The Effective Rank : a Measure of Effective Dimensionality," in *15th European Signal Processing Conference*, 2007, pp. 606–610.

[59] P. Zhou, J. Feng, C. Ma, C. Xiong, S. C. H. Hoi *et al.*, "Towards Theoretically Understanding Why Sgd Generalizes Better Than Adam in Deep Learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 21 285–21 296.

[60] J. Farebrother, J. Orbay, Q. Vuong, A. A. Taïga, Y. Chebotar *et al.*, "Stop regressing : training value functions via classification for scalable deep rl," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24. JMLR.org, 2024.

[61] A. Kumar, R. Agarwal, D. Ghosh, and S. Levine, "Implicit under-parameterization inhibits data-efficient deep reinforcement learning," in *International Conference on Learning Representations*, 2021. [Online]. Available : https://openreview.net/forum?id=O9bnihsFfXU

[62] P. Foret, A. Kleiner, H. Mobahi, and B. Neyshabur, "Sharpness-aware minimization for efficiently improving generalization," in *International Conference on Learning Representations*, 2021. [Online]. Available : https://openreview.net/forum?id=6Tm1mposlrM

[63] G. Sokar, R. Agarwal, P. S. Castro, and U. Evci, "The Dormant Neuron Phenomenon in Deep Reinforcement Learning," in *Proceedings of the 40th International Conference on Machine Learning*. PMLR, Jul. 2023, pp. 32 145–32 168, iSSN : 2640-3498.

[64] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The Arcade Learning Environment : An Evaluation Platform for General Agents," vol. 47, 2013, pp. 253–279. [Online]. Available : https://ale.farama.org/

[65] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, pp. 9–48.

[66] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin *et al.*, "What matters for on-policy deep actor-critic methods ? a large-scale study," in *International Conference on Learning Representations*, 2021. [Online]. Available : https://openreview.net/forum?id=nIAxjsniDzg

[67] C. Lyle, Z. Zheng, K. Khetarpal, J. Martens, H. van Hasselt *et al.*, "Normalization and effective learning rates in reinforcement learning," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet *et al.*, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 106 440–106 473.

[68] Y. Tang, Z. D. Guo, P. H. Richemond, B. Avila Pires, Y. Chandak *et al.*, "Understanding self-predictive learning for reinforcement learning," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato *et al.*, Eds., vol. 202. PMLR, 23–29 Jul 2023, pp. 33 632–33 656. [Online]. Available : https://proceedings.mlr.press/v202/tang23d.html

[69] C. A. Voelcker, T. Kastner, I. Gilitschenski, and A.-m. Farahmand, "When does self-prediction help ? understanding auxiliary tasks in reinforcement learning," *Reinforcement Learning Journal*, vol. 4, pp. 1567–1597, 2025.

[70] H. Ghallab, M. Nasr, and H. Fahmy, "Mitigating catastrophic forgetting in continual learning using the gradient-based approach : A literature review," *International Journal of Advanced Computer Science and Applications*, vol. 16, no. 4, 2025. [Online]. Available : http://dx.doi.org/10.14569/IJACSA.2025.0160414

APPENDIX A NETWORK ARCHITECTURE

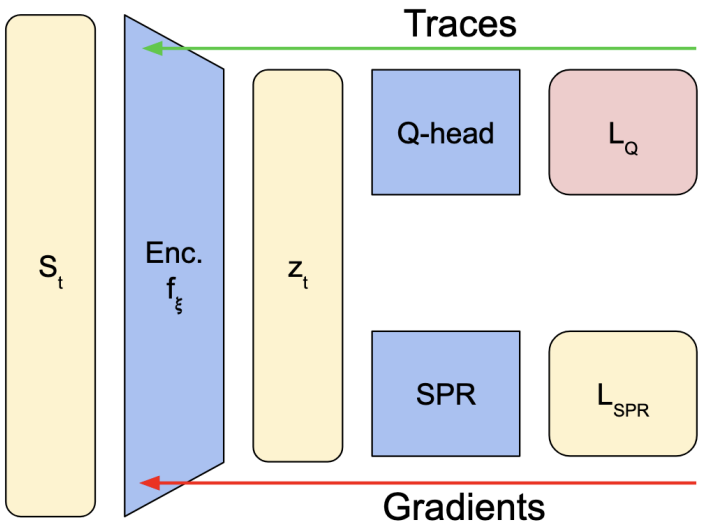
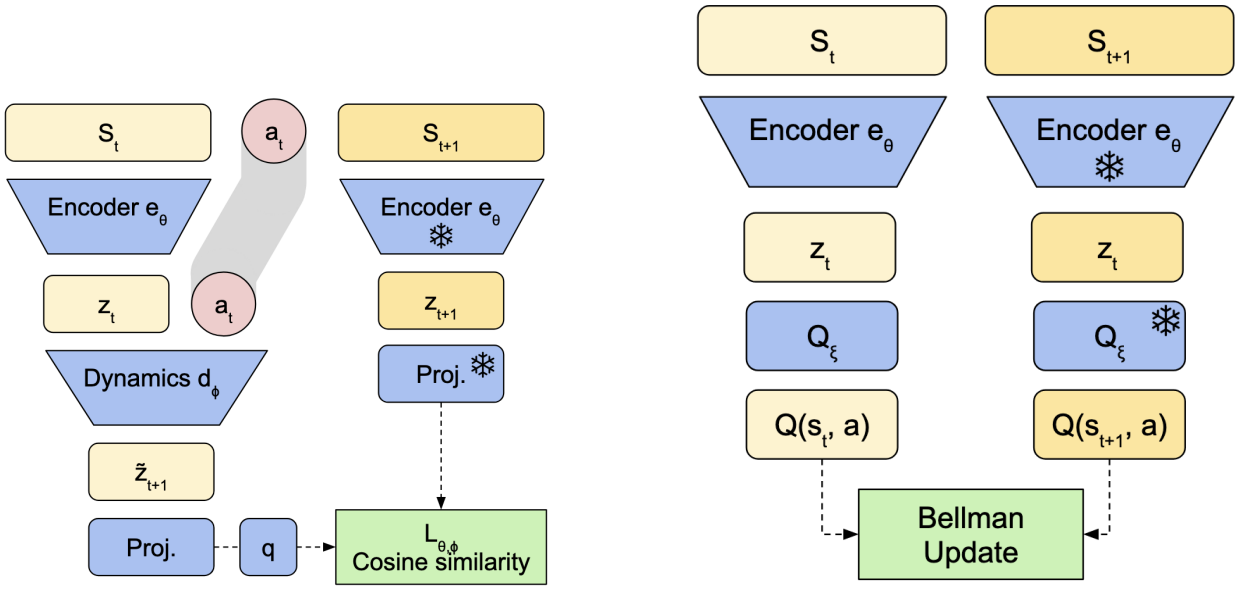


FIGURE A.1 High-level representation of the global network architecture, showcasing the two sources of parameter updates and the shared parameters (encoder). This figure shows what is referred to as the *Q-learning* part of the network (updated with eligibility traces) and the *SPR* part of the network (updated with regular gradient updates). The shared parameters which are affected by both sources of updates lie in the encoder. It is important to note that in our experiments, a projection layer from the Q-learning head is also shared with the SPR part of the network.



(a) Architecture of the SPR part of the network

(b) Architecture of the Q-learning part of the network

FIGURE A.2 Visual representation of the architecture of what we refer to as the *a)* SPR part of the network, updated with the BYOL-like cosine similarity loss and the *b)* Q-learning part of the network, updated through the (streaming) DQN loss. The snowflake icon indicates parameters that don't participate in the gradient backpropagation. This figure only describes what happens when $K = 1$ (SPR predicts only the immediate next state). In practice we use this architecture to iteratively predict the $K = 5$ next states as described in the literature review (Chapter 2) and sum the SPR-losses.