Enhancing Deep Reinforcement Learning with Compressed Sensing-based State Estimation

Shaswot Shresthamali
Keio University
shaswot@acsl.ics.keio.ac.jp

Masaaki Kondo
Keio University
kondo@acsl.ics.keio.ac.jp

***Abstract*—In various real-world applications, sensor data collected for adaptive control using Reinforcement Learning (RL) often suffer from missing information due to sensor failures, data transmission errors, or other sources of noise. Such missing data can significantly hinder the agent's ability to make informed decisions and degrade performance. In this paper, we propose a novel approach to address this challenge by leveraging Compressed Sensing (CS) techniques to recover missing information from the sensor data and reconstruct the state observation. The reconstructed state is then fed to the RL agents. As a result, they exhibit enhanced robustness and intelligence, surpassing the performance achievable when solely presented with noisy data as state input.**

I. Introduction

The rapid advancement of Reinforcement Learning (RL) has sparked a lot of attention in their application in autonomous systems and decision-making applications (e.g., power management within processor-memory systems and in IoT networks [1], [2]). However in real world applications, the observed data may often suffer from missing information. This impedes the RL agent's ability to execute optimal policies and make reliable decisions. Missing data can arise from a variety of sources, including sensor failures, unreliable wireless communication channels, or the inherent limitations of sensors in capturing certain events.

To address this challenge, we propose an innovative approach that integrates compressive sensing techniques with RL. Compressive Sensing (CS) provides a powerful framework for recovering missing information from incomplete or undersampled data. By exploiting the sparsity of signals, compressive sensing enables the reconstruction of missing measurements with a high level of accuracy, even when significant portions of the data are missing.

Missing observation data degrades the performance of RL agents significantly during training and inference. However, by integrating CS with the RL process, we can recover the "lost" data and improve the quality of the observation data that is made available to the RL agent as shown in Figure 1. This mitigates the detrimental effects of missing data and enables the agent to make reliable and correct decisions. Consequently, the RL agent exhibits improved performance compared to agents trained solely on the noisy data.

In this study, we assume a pre-trained RL agent that needs to make decisions despite missing observation information. We take two representative (simulation) environments based on the Farama Gymnasium API -`BipedalWalker-v3` and `VizDoom` - and evaluate the effects of noisy observation and performance recovery using CS for each of the environments. The `BipedalWalker-v3` requires the agent to control four joints of a walking robot using 24-dimensional state observation. `VizDoom` requires the agent to learn to play a 3D first-person shooter game called Doom [3] using rendered game pixels as the observations. Naturally, in both environments, incomplete observation data leads to performance degradation. However, by utilizing CS-based state recovery, the agent can fill in the missing information in its observations, thereby avoiding significant compromise to the quality of its policy. While alternative signal processing filters can be used to interpolate the incomplete data, we demonstrate that such techniques require manual domain-specific tuning and may not necessarily improve the performance of RL agents. Although CS has little overhead for `BipedalWalker-v3`, it incurs significant computation time for `VizDoom` due to its large observation dimension space. This latency is a bottleneck for other fast-response environments. To overcome this, we develop a GPU-accelerated solution to reduce the CS computation time using the PyTorch framework. As mentioned earlier, we solely concentrate on the inference side of RL in this work. Nevertheless, similar performance enhancements can be expected during the training phase. Specifically, this work makes the following key contributions:

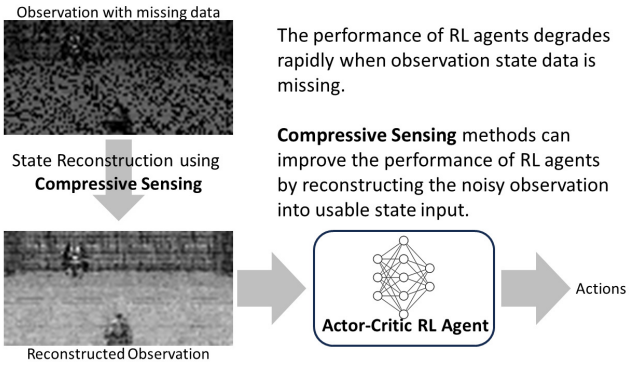

Fig. 1. Integrating Compressive Sensing for state reconstruction can drastically improve the performance of RL agents in environments where observation data may be missing.

- We analyze the effects of incomplete observation infor-

mation during training and evaluation (inference) of RL agents, demonstrating that incomplete state information degrades the performance of RL during both training and inference phases.
- We highlight that training RL agents on noisy data does not necessarily result in superior performance and robustness.
- We propose an RL framework that incorporates CS-based techniques to recover missing state information.
- We provide empirical evidence through experimental evaluations in different noisy scenarios, demonstrating that our proposed method significantly mitigates performance degradation.
- We perform a computational cost analysis of using CS-based techniques to illustrate the tradeoffs involved in using CS-based state recovery and show that GPU acceleration can significantly reduce the computation overhead of CS.

The rest of this paper is organized as follows: Section Section II provides an overview of related work in the field of RL and CS. Section Section III presents the fundamental concepts of CS and RL and Section IV details the proposed methodology for incorporating compressive sensing into RL training. Section V presents experimental methodology. Results and performance evaluations are reported in Section VI. Finally, Section VII concludes the paper.

II. Related Work

Compressive Sensing (CS) has garnered considerable attention in various fields, particularly in applications where measurements are noisy and challenging to obtain. While various studies have explored the combination of RL and CS, most of them concentrate on enhancing CS using RL techniques. [4] utilize RL to optimize the measurement matrix for time-varying signals, and [5] tackle a similar problem for human activity sensing systems. In contrast, this work focuses on how CS can enhance RL performance by recovering missing observation data. Although the vulnerability of RL systems to noise has been extensively studied, the integration of CS with RL agents in the presence of missing observation data remains unexplored. [6] introduce a framework for learning better rewards with faster convergence, even in biased noisy environments. [7] address the challenge of adversarial action noise and propose methods for training RL agents in such scenarios. In terms of stability and convergence, [8] highlight the detrimental effects of noise in the loss function during training, which arises from the combination of bootstrapping, function approximation, and off-policy training. CS-inspired deep learning-based approaches [9], [10] propose to leverage generative models to enhance signal recovery. The most closely related work is [11], where the authors propose a method that employs image reconstruction loss and variational autoencoders to learn a robust latent space representation resilient to observational noise. However, this usually incurs significant computational and training overhead.

III. Background

A. Compressed Sensing

With compressed sensing framework, it is possible to recover a high dimensional signal $\mathbf{x} \in \mathbb{R}^n$ from a linear measurement $\mathbf{y} \in \mathbb{R}^p$ s.t. $\mathbf{y} = \mathbf{C}\mathbf{x}$ where $\mathbf{C} \in \mathbb{R}^{p \times n}$ is the linear measurement matrix. $\mathbf{C}$ is usually a short-fat matrix i.e., $p << n$. This means that it is possible to reconstruct $\mathbf{x}$ with only p measurements instead of n measurements.

Solving for $\mathbf{x}$ is generally not possible because the **??** is underdetermined. However, if we transform $\mathbf{x}$ to a K-sparse signal $\mathbf{s} \in \mathbb{R}^n$ (i.e., it has only K non-zero elements) through a suitable transform basis, $\mathbf{\Psi} \in \mathbb{R}^{n \times n}$ (e.g., the Fourier basis) such that $\mathbf{y} = \mathbf{C}\mathbf{x} = \mathbf{C}\mathbf{\Psi}\mathbf{s} = \mathbf{\Theta}\mathbf{s}$, then it is possible to recover a nearly perfect reconstruction of $\mathbf{x}$ with high probability given a random $\mathbf{C}$ and a sparse $\mathbf{s}$. [12], [13]. It is necessary for $\mathbf{\Theta}$ to satisfy the Restricted Isometry Property (RIP) which requires $\mathbf{C}$ and $\mathbf{\Psi}$ to be incoherent (i.e., the rows of $\mathbf{C}$ are not correlated with the columns of $\mathbf{\Psi}$). Fortunately, the RIP holds with high probability for random $\mathbf{C}$ and sparse signals $\mathbf{s}$. Furthermore, it is also important that the number of measurements p be sufficiently large on the order of

$$p \approx \mathcal{O}(K \log(n/K)) \approx k_1 K \log(n/K) \tag{1}$$

where the constant multiplier k_1 depends on how incoherent $\mathbf{C}$ and $\mathbf{\Psi}$ are.

As a consequence of these properties, the distances between two signals $\mathbf{s_1}$ and $\mathbf{s_2}$ are preserved (within limits) after their projections by $\mathbf{\Theta}$ i.e., it acts as a unitary transformation on K-sparse vectors $\mathbf{s}$ and therefore enabling signal reconstruction with l_1 convex optimization with high probability.

B. Reinforcement Learning

RL is a machine learning paradigm where an agent learns to make sequences of decisions in an environment to maximize a cumulative reward. The agent interacts with the environment, takes actions, receives feedback in the form of rewards, and learns to choose actions that lead to higher rewards over time.

Actor-Critic algorithms combine the benefits of policy-based methods (which can handle high-dimensional action spaces and stochastic policies) with the advantages of value-based methods (which provide stability and better convergence properties). The actor learns a policy that maps states to actions, aiming to maximize the expected cumulative reward. The critic evaluates the actions taken by the actor and learns a value function that estimates the expected cumulative reward when following a particular policy. This provides feedback to the actor on the quality of its actions.

In this work we employ two popular actor-critic algorithms - the Proximal Policy Optimization (PPO) algorithm [14] and Soft Actor Critic (SAC) algorithm [15]. Although the details of the algorithm are outside the scope of this work, in essence PPO maintains stability in the learning process by ensuring that the probability ratios between old policies and new policies do not exceed a certain threshold while SAC trades off between the expected return and the entropy of the

policy and avoiding bad local optima. Both of these algorithms are suitable for continuous state spaces and discrete actions and show stable learning behavior.

IV. Compressed Sensing for RL

We explore a scenario where the RL agent observes a multi-dimensional state that contains random missing data. It is important to note that this setting differs from a Partially Observable Markov Decision Process (POMDP), where the agent has incomplete *information* about the state despite having access to complete observational state *data*. Our focus excludes POMDP settings. Instead, we assume that the state is fully Markovian i.e., the observation is complete but the incoming observation *data* is incomplete. To mitigate the impact of missing data on the RL agent's performance, we propose utilizing compressed sensing principles to recover the missing state information.

To illustrate the effectiveness of our approach, we consider two different scenarios where a trained RL agent has to make action decisions when the incoming observation is noisy due to incomplete observation data. In `BipedalWalker-v3` scenario, the state dimensions are sensor information from a walking robotic but some sensors fail randomly at different times. This is a representative example of a robotics or IoT application where the RL policy makes decisions based on sensor information. In `VizDoom` scenario, the observational space is image pixels from a video game, similar to the setting described in [3], [11]. This example serves as a convenient framework for visualizing and analyzing the reconstructed signals. Moreover, images are one of the most convenient and inexpensive ways of acquiring rich state information in most real-world applications (e.g., using cameras for robotics). By using these two representative scenarios, we want to highlight that our proposed method is applicable to a wide range of state vector signals, as long as a suitable sparse representation is attainable through an appropriate linear transformation.

A. LASSO for Observational Data Recovery

The observation state for `BipedalWalker-v3` consists of 10 lidar rangefinder measurements, different speed and velocity measurements as well as other sensor information (e.g., legs contact with ground). From our preliminary experiments and analysis, we observe that most of these sensor data exhibit some sort of periodicity and thus have sparse representations when using the Fourier transform. For `VizDoom`, given that the input image is sourced from video games, which also exhibits sparse representations in a Fourier basis, we employ the Discrete Cosine Transform (DCT) as our chosen transform matrix, denoted as $\mathbf{\Phi}$. We assume that the noise resulting from the loss of pixel information in the observation is entirely random. Consequently, we can consider our measurement matrix $\mathbf{C}$ to be incoherent with respect to the DCT transform matrix $\mathbf{\Phi}$. In other words, the product of $\mathbf{C}$ and $\mathbf{\Phi}$, denoted as $\mathbf{C\Phi} = \mathbf{\Theta}$, satisfies the Restricted Isometry Property (RIP) condition.

Building upon the concepts discussed in the theory section (Section III-A), we observe that it is possible to obtain a sparse vector, denoted as $\hat{\mathbf{s}}$, which is consistent with the measurements $\mathbf{y}$. Once $\hat{\mathbf{s}}$ is obtained, we can perform an inverse transform to derive an estimate, denoted as $\hat{\mathbf{x}}$, of the original observation signal $\mathbf{x}$. While there exist various convex optimization methods to solve for $\hat{\mathbf{s}}$, we specifically focus on the widely used least-squares regression technique called the Least Absolute Shrinkage and Selection Operator (LASSO) [16]. LASSO regression introduces an l_1 penalty term to regularize the problem, preventing overfitting and making it well-suited for our objectives. Formally, the optimization problem for LASSO with a given regularization parameter α can be expressed as:

$$\hat{\mathbf{s}} = \underset{\mathbf{s}}{\operatorname{argmin}}(\|\mathbf{\Theta s} - \mathbf{y}\|_2 + \alpha\|\mathbf{s}\|_1) \tag{2}$$

Here, the cost function is regulated by $\|\mathbf{s}\|_1 = \sum_n^{k=1} |s_k|$ to obtain the sparsest $\hat{\mathbf{s}}$.

In the case of LASSO regression, we transform our noisy 2D input image into a flattened column vector, which becomes our measurement $\mathbf{y}$ for the original image $\mathbf{x}$. Although the measurement matrix $\mathbf{C}$ is not explicitly provided, it can be inferred easily by identifying the rows of $\mathbf{y}$ containing zero values. Consequently, we determine $\mathbf{\Theta}$ as $\mathbf{C\Psi}$ by selectively extracting the rows of $\mathbf{\Psi}$ that correspond to non-zero rows in $\mathbf{y}$. Subsequently, we input $\mathbf{y}$ and $\mathbf{\Theta}$ into the LASSO optimizer to obtain the sparse vector solution, denoted as $\hat{\mathbf{s}}$. The LASSO optimizer uses coordinate descent algorithm to solve the underdetermined system of equation. Finally, we reconstruct the original observation $\hat{\mathbf{x}}$ by applying the inverse DCT to $\hat{\mathbf{s}}$ and appropriately reshaping the resulting vector (into a 2D image format for`VizDoom`) which is our reconstructed state observation.

V. Evaluation Methodology

A. BipedalWalker-v3 Environment

We train the RL agent to solve `BipedalWalker-v3` using the default SAC network architecture provided by the `stable-baselines3` library [17]. This network has fully connected layers with 24 neurons in its input layer (state size) shared by both actor and critic networks. The actor contains 8 neurons in its output layer (action space size) with two hidden layers, each with 256 neurons. The critic has similar architecture to the actor but only contains a single node that outputs the values of the input state. The RL is trained for a total of 1×10^6 timesteps with a learning rate of 3×10^{-4} and a batch size of 256. Network updates are performed at every timestep.

While training is carried out in a noise-free environment, the inference (or test) is carried out in a noisy environment (assuming faulty sensors). To emulate this, the state data for some dimensions are randomly zeroed out with some probability given by the `noise` parameter. From our preliminary experiments, we observe that only a certain set of sensor signals are compressible `BipedalWalker-v3` i.e., they

have a sparse representation in the Fourier frequency domain. We analyze the effect of noise and subsequent reconstruction using CS only for these sensor signals. Other state signals are assumed to be not affected by noise.

Since CS requires the temporal history of each of the compressible state observation signals, we store a record of previous observations for a fixed time window. In our experiments, a window size of 25 proved sufficient, consuming approximately only 4KB of extra memory. Thus, at each timestep, if any of the state dimensions is missing the state information, we use CS to fill in this information using the history of that particular state signals.

B. ViZDoom Environment

`ViZDoom` environment emulates a simplified version of a 3D first-person shooter game called Doom [3]. For our experiments, we choost the `BASIC` scenario map which features a rectangular room with gray walls, ceiling, and floor. In this scenario, the shooter agent spawns along one of the longer walls in the center, while a circular monster randomly spawns somewhere along the opposite wall. The primary objective of the agent is to eliminate the monster by shooting it as quickly as possible without missing.

We deliberately choose this scenario due to its simplicity and straightforward objectives. The agent's task involves identifying the monster's location, maneuvering left or right until it is in front of the monster, and then firing. Therefore, the underlying RL problem is relatively straightforward. However, the introduction of noise through the removal of pixels from the observation frame creates difficulties for the agent in learning the necessary features to recognize the monster. Consequently, any performance degradation resulting from missing pixels can be directly attributed to the lack of accurate state information rather than limitations in the learning algorithm itself (e.g., sparse rewards, partial observation, or inadequate exploration).

To optimize computational resources and improve the efficiency of our approach, we modify the rendering settings of the original game engine. We choose to render the game in grayscale frames with a resolution of 160×320 pixels, as opposed to the default RGB24 format. Additionally, we crop the frames to a size of 50×100 pixels, removing non-essential elements such as the Heads Up Display (HUD), floor, ceiling, and walls. This not only reduces the parameter size of the RL network but also accelerates the LASSO image reconstruction process.

Furthermore, to optimize computational costs and enhance the learning process, we select every fourth frame while skipping the three frames in between. This downsampling approach helps reduce compute requirements while preserving important temporal information, and may even accelerate learning as reported in [18].

The RL agent has to choose between three actions at each timestep: `shoot`, `go-left`, and `go-right`. Each episode is constrained to a maximum of 300 timesteps. Throughout the training process, the agent receives a reward of -1 at each timestep. If the agent successfully shoots and eliminates the monster, it obtains a reward of $+101$. Conversely, if the agent fails to hit the monster, it incurs a penalty of -5. The episode concludes either when the monster is killed or when the episode reaches the maximum timestep limit. To ensure training stability, we scale the rewards by a factor of 0.01. Consequently, the highest achievable reward is 1, indicating that the agent successfully eliminates the monster in the first shot. Conversely, the lowest possible reward is -15, which occurs when the agent fails to kill the monster within the 300 timesteps but continuously fires at each timestep.

We train the RL agent for `ViZDoom` in the `BASIC` scenario map for a total of 1×10^6 steps using the default CNN architecture for PPO algorithm provided by `stable-baselines3` library. The feature extractor is composed of three CNN layers with ReLU activation that feeds into the actor and critic networks. Both are fully connected layers with 512 nodes. During training, the learning rate is fixed at 1×10^{-4} and a batch size of 32 is used. Network updates are performed every 4096 timesteps.

Similar to `BipedalWalker-v3`, the training is carried out in a noise-free environment while the testing assumes a noisy environment. To emulate missing sensor information, we zero out random pixels in each frame. The missing pixel probability is represented by the `noise` parameter. A high `noise` means a higher likelihood of a pixel being zeroed out i.e. being unsampled.

C. Experiment Setup

All experiments were run on a DGX-H100 machine that contains NVIDIA H100 GPUs and Intel Xeon 8480C processors (112 cores). For both `BipedalWalker-v3` and `ViZDoom`, we train five agents with different initialization seeds in a noise free environment. These performance of these agents are then assessed in noisy environments. The objective is to analyze the impact of missing observation data on the policies learned by the agents. Each agent (for both environments) is evaluated for 20 game episodes under varying levels of noise in the testing environment. The mean and standard error of the rewards obtained in each episode are recorded.

We then investigate the effect of employing CS for state recovery on the RL agents' performance. To achieve this, we select the agents trained in noise-free environments and evaluate their performance on environments with different values of `noise` while utilizing compressed sensing to reconstruct the image prior to inputting it into the RL agent. The mean and standard deviation of rewards over 20 episodes are analyzed. In our experiments, we employ the LASSO optimizer from the `scikit-learn` Python package, which utilizes the coordinate descent algorithm. The regularization parameter is set to $\texttt{alpha} = 1 \times 10^{-3}$.

VI. Results

A. Effect of missing data during training

One popular method to make RL agents robust to noise is to train the RL agents in noisy environments. To analyze

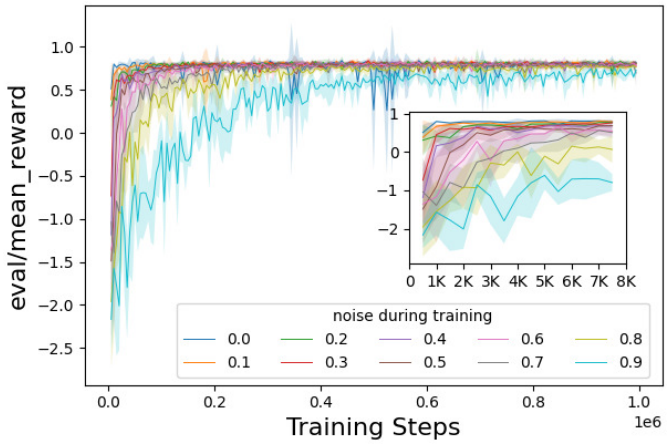

Fig. 2. The RL agent takes longer to learn good policies as the noise is increased.

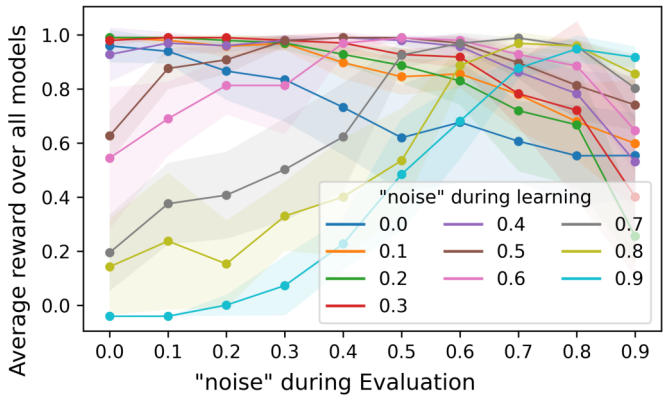

Fig. 3. Agents trained in environments with lower levels of noise exhibit poorer performance in environments with higher levels of noise. Interestingly, agents trained in highly noisy environments perform even worse when the noise is eliminated during evaluation. This observation indicates that the introduction of noise during training does not necessarily enhance the agents' robustness to noise during evaluation.

the effectiveness of this method, we train the `ViZDoom` RL agent in noisy environments for different values of the `noise` parameter. During training, we log the performance of the learned policy every 5000 training steps for 10 episodes. (We have not shown similar analysis for noise sensitivity during training for `BipedalWalker-v3` in the interest of space.)

In Figure 2, we present the learning performance of PPO agents during the training process which displays the average reward for the different agents, with shaded areas indicating one standard deviation. As expected, the agent trained in a non-noisy environment (`noise` $= 0.0$, blue) quickly learns an effective policy, achieving a high reward within 1000 training steps. This is expected since the `BASIC` scenario is relatively simple, with limited actions and straightforward features, requiring minimal exploration. However, when pixels are missing from the observation frame, the learning performance deteriorates. In the case of extreme noise with a 90% missing pixel probability (`noise` $= 0.9$, cyan), the learning performance declines significantly. In this scenario, the agent requires nearly 500K additional steps to achieve even half the reward obtained in the non-noisy environment. This outcome is not surprising as obfuscated pixels make it more challenging for the agent to learn informative state representations for deriving an effective policy. However, even when the noise level is extremely high, the agent can still learn a reasonable policy, albeit suboptimal. This is attributed to the effectiveness of the CNN feature extractor network in filtering out noise through its convolutional kernels and max-pooling operations.

In Figure 3, we compare the performance of different `ViZDoom` agents in various noise settings. The plot displays the average reward achieved by agents trained in a specific noisy environment when evaluated in different environments with varying levels of noise. The blue line represents an agent trained in a non-noisy environment. As the noise parameter increases in the evaluation environment, we observe a degradation in performance. Additionally, the policies of these agents become less stable, as indicated by the wider error ranges (shaded areas), as the evaluation noise increases. This outcome is expected because the agent struggles to handle the noise introduced by missing pixels, resulting in suboptimal state representations and compromised policies.

On the other hand, agents trained in slightly noisy environments (`noise` $= 0.1, 0.2, 0.3$) demonstrate robustness to noise in the evaluation environment. However, when the noise in the evaluation environment surpasses the level experienced during training, their performance also deteriorates. This suggests that introducing noise during training can enhance the robustness of RL agents, but only up to a certain threshold. For agents trained in highly noisy environments (`noise` $= 0.4, ..., 0.9$), we observe a degradation in performance even when evaluated in less noisy environments. These agents achieve their best performance when the evaluation noise matches the noise level of their training environment. This implies that these agents have learned policies that are specifically tailored to the noise profile of their training environment. Consequently, we cannot expect RL agents to acquire noise-robust policies solely by introducing noise during training.

B. Enhancing RL performance with Compressed Sensing

We have seen that unexpected noise caused by missing pixels negatively impacts the performance of RL agents and merely training the agent in a noisy environment does not lead to the acquisition of noise-robust policies. This approach would require prior knowledge of the noise characteristics *and* that the noise remain constant in the environment, both of which are not generally possible. Thus, to mitigate performance degradation resulting from missing pixel information, we employ CS techniques to reconstruct the state from the noisy state information.

In Figure 4, we compare the performance of RL agents (which were trained in non-noisy environment) in various noisy environments with and without state reconstruction. The green bars represent the performance of agents without any state reconstruction and serves as the baseline (error bars represent one standard deviation). It is clear from the

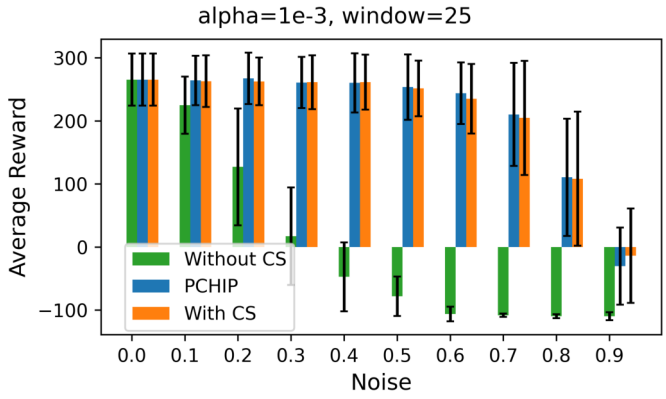

Fig. 4. The performance of `BipedalWalker-v3` decreases dramatically as noise increases (green bars). However, by using CS, we can recover the state information and dramatically restore the performance (orange bars).

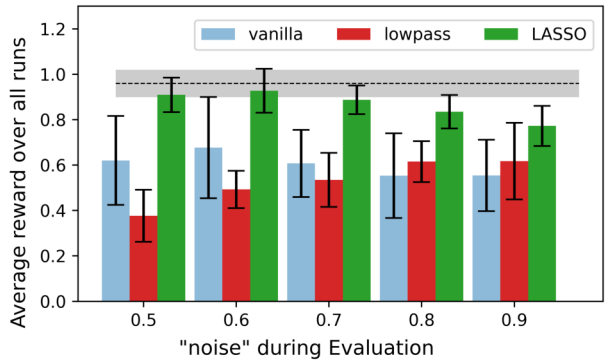

Fig. 5. Comparison of `ViZDoom` RL agent performance using low pass filter and compressed sensing for noise mitigation. The agent was trained in a non-noisy environment. The dashed line and the gray shaded area represent the mean and standard deviation of the rewards received by the agent when evaluated in a non-noisy environment.

figure that the performance of the agents degrade rapidly with increasing environmental noise. When noise increases above 30%, the reward becomes negative and this shows that the robot has a hard time even standing upright. However, when we use CS to reconstruct the state information, we observe that we can almost recover the agent to its original performance (orange bars) even when there is a 50% chance of sensor failure. Furthermore, the recovered state information does not cause instability in the agents' policy as evidenced by the relatively low standard deviation. We also investigate how the performance of RL agent is affected when we use Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) interpolation (applicable for only for 1-D signals) to fill in the missing state information. This is shown by the blue bars in Figure 4. We observe that PCHIP and CS methods have similar reconstruction fidelity and are able to help the RL agent recover its performance quite significantly. At very high noise levels, CS cannot reconstruct the signal with high fidelity because the number of samples is not enough (Equation (1) due to the small window size of 25.

We observer similar advantages of using CS for `ViZDoom`.

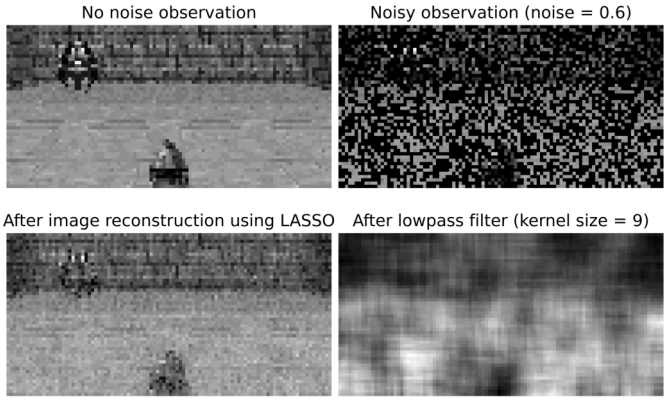

Fig. 6. Comparison of image reconstruction using LASSO and lowpass filter.

Figure 5 presents a comparison of the `ViZDoom` RL agents' performance with and without CS image reconstruction. We also compare against the performance of the agents when applying a lowpass filter to remove the high-frequency noise due to missing pixels (similar to PCHIP for 1D signals). In Figure 5, the average performance of the five agents trained in a non-noisy environment and evaluated in environments with different noise levels is shown. The horizontal dashed black line and shaded gray region represents the average reward and standard error obtained by these agents when evaluated over 20 episodes in a non-noisy environment. The bar plots depict the rewards achieved when evaluated in noisy environments using various state reconstruction techniques. The grayish blue bars (labeled as `vanilla`) indicate the agent's performance in noisy environments without any reconstruction. It is observed that increasing the noise level significantly degrades the performance by up to 40%. However, by utilizing compressive sensing techniques and employing LASSO for state reconstruction, we can recover the performance to nearly its original value (green bars labeled as `LASSO`). Even in the extreme case where pixels have a 90% chance of being lost, employing CS techniques can enhance the performance by up to 20% compared to the baseline.

Additionally, we investigated whether filtering out the noise using image filters can improve performance. The red bars (`lowpass`) represent the RL agent's performance when the noisy state is preprocessed with a lowpass filter (kernel size = 9). Interestingly, we observe that using such a filter does not necessarily improve performance and may even decrease it. The lowpass filter shows only a slight improvement in performance only in highly noisy environments. We also experimented with other common digital image processing filters, such as Gaussian blur and Median blur. However, even after extensive manual hyperparameter tuning (e.g., kernel size), these filters did not improve the performance of the RL agent. Figure 6 compares examples of reconstruction results using LASSO and lowpass filter. We can observe that the LASSO method is able to reconstruct the gun and the monster with enough fidelity that the CNN can recognize them and pass

on the feature activations to the policy network.

C. Computation cost of Compressed Sensing

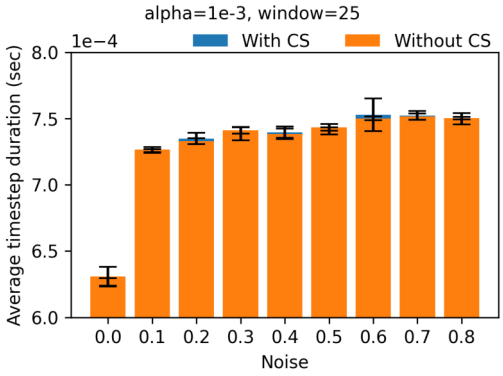

Fig. 7. LASSO computational time for reconstructing the sensor signals for BipedalWalker is negligible and consumes almost no overhead because the sensor data has only a few dimensions (window size = 25).

Figure 7 shows the average time required to process each timestep in the environment for `BipedalWalker-v3`. This involves generating and applying the noise to the observations received from the environment API. This is a simulation overhead (represented by orange bars) and is not related to the computation cost of CS. The actual cost of CS is negligibly small as evidenced by the slightly visible blue bars stacked on the orange bars. This low CS computation time is possible because the size of the window over which we store the temporal history of different sensor signals is very small.

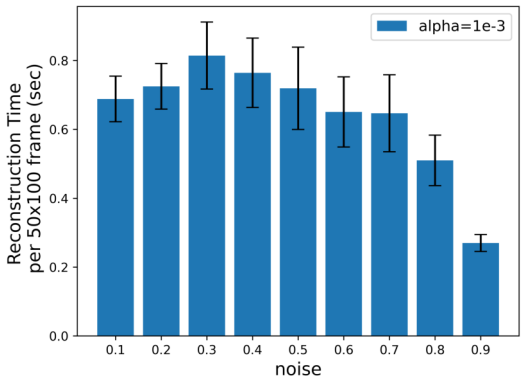

Fig. 8. Computational time for reconstructing a 50×100 image from noisy input with missing pixels for `ViZDoom` for various values of `noise`.

However, the process of reconstructing the state from noisy and incomplete input data using LASSO can be computationally expensive when the input data has a large number of dimensions as in `ViZDoom`. Figure 8 shows the LASSO reconstruction time per frame for `ViZDoom`. The LASSO algorithm is implemented using a highly optimized standard `scikit-learn` library in a CPU using FP64 format. We observe that the reconstruction takes hundreds of milliseconds which is a significant overhead for a game that can run as fast as 7000 fps. This is because image construction is achieved using coordinate descent which require looping over each element of the solution vector *sequentially* (for `ViZDoom`, the solution vector has a size of 5000). We also observe that the reconstruction time peaks at `noise` $= 0.3$ and decreases as we increase the noise. This is because at low noise, the reconstruction error is not very large and so the algorithm converges quickly. Also when the `noise` is very high (e.g., `noise` $= 0.9$), the measurement vector $\mathbf{y}$ becomes shorter, requiring fewer computations and iterations to find a solution. However, these solutions tend to be non-optimal.

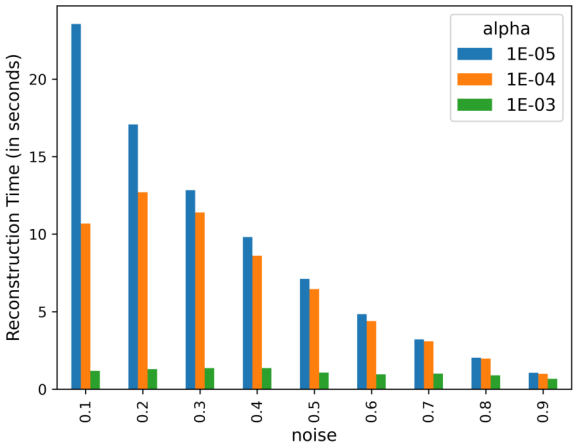

Fig. 9. The LASSO computational time for reconstructing for various values of `alpha` (regularization strength).

The fidelity of the reconstructions can be controlled to some extent by using the regularization parameter `alpha`. We observe from our experiments that higher `alpha` values yield better but grainier reconstructions. We also observe that higher alpha decreases the LASSO computation time as shown in Figure 9. A higher `alpha` gives less weight to accurate optimization (reconstruction) and more weight to sparsity, resulting in less iterations and shorter computation times. Lower `alpha` encourage the optimizer to minimize reconstruction error and therefore requires more reconstruction time. Empirical analysis indicates that an `alpha` value of 1×10^{-3} strikes a good balance between reconstruction quality and computation time.

Furthermore, the memory cost for implementing LASSO increases exponentially with the size of the solution vector. For `ViZDoom`, the image size is 50×100 and thus requires the transform matrix $\mathbf{\Phi}$ to be a 5000×5000 dense matrix. This requires about $\approx$ 200 MB in FP64 representation. When we use the original frame (without cropping), the image size is 120×160. In this case, $\mathbf{\Phi}$ is a square matrix with dimensions of size 19,200 and requires 2.7GB of memory. Obviously, the computation time required for optimization also increases drastically. Once solution to overcome this limitation would be to offload the memory and calculations to a GPU accelerator.

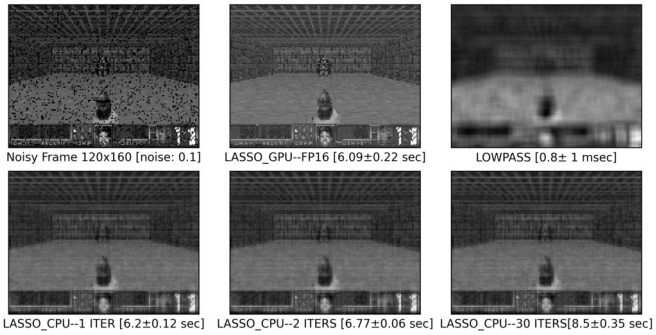

Fig. 10. Image reconstructions and average time overhead required for 120×160 image.

Figure 10 shows the reconstructed images and the average reconstruction time for a 120×160 frame. The CPU implementation uses the `scikit-learn` library which has been optimized for FP64 operations using Fortran-contiguous numpy array allocation for the $\mathbf{\Phi}$ matrix. This takes about 6-8 seconds depending on the number of optimization iterations. We observe that increase the number of iterations does not necessarily improve the quality of the image.

The GPU implementation of LASSO is a straightforward naive implementation and there is still much room for machine-specific optimization. However even with such a simple implementation, we were able to match the performance of the highly optimized CPU implmentation. This was possible because we were able to use the large amount of available GPU memory and leverage the use of accelerated FP16 computation to decrease the computation time. From Figure 10, we observe that the FP16 GPU reconstruction has a much better fidelity that the FP64 CPU reconstruction. This shows that there is a possibility of decreasing the computation time and memory overhead for CS by using GPUs.

VII. Conclusion and Future Directions

In conclusion, this research work explored the use of CS techniques to enhance the performance of RL agents in environments with insufficiently sampled and missing state information. To address the performance degradation caused by missing pixel information, CS methods were employed to reconstruct the state from noisy and subsampled inputs. The results showed that CS techniques could significantly improve the performance of RL agents in environments with limited state information. However, the reconstruction process using LASSO can get computationally expensive especially for higher-dimensional input reconstructions but this can be overcome to some degree by using more suitable transform matrices, convex optimization algorithms and GPU acceleration. Overall, the findings of this research highlight the potential of CS techniques to enhance the robustness and performance of RL agents in challenging environments, opening doors to new opportunities and applications.

Acknowledgment

We thank Wojciech Roga and Baptiste Chevalier for their helpful discussions. This work was supported by the Center of Innovations for Sustainable Quantum AI (JST Grant Number JPMJPF2221).

References

[1] S. Shresthamali, M. Kondo, and H. Nakamura, "Multi-objective reinforcement learning for energy harvesting wireless sensor nodes," in *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 2021, pp. 98–105.

[2] Y. Shen, L. Schreuders, A. Pathania, and A. D. Pimentel, "Thermal management for 3d-stacked systems via unified core-memory power regulation," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 5s, pp. 1–26, 2023.

[3] M. Wydmuch, M. Kempka, and W. Jaśkowski, "ViZDoom Competitions: Playing Doom from Pixels," *IEEE Transactions on Games*, vol. 11, no. 3, pp. 248–259, 2019, the 2022 IEEE Transactions on Games Outstanding Paper Award.

[4] N. Karim, A. Zaeemzadeh, and N. Rahnavard, "Rl-ncs: Reinforcement learning based data-driven approach for nonuniform compressed sensing," in *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2019, pp. 1–6.

[5] G. Liu, R. Ma, and Q. Hao, "A reinforcement learning based design of compressive sensing systems for human activity recognition," in *2018 IEEE SENSORS*. IEEE, 2018, pp. 1–4.

[6] J. Wang, Y. Liu, and B. Li, "Reinforcement learning with perturbed rewards," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 6202–6209.

[7] C. Tessler, Y. Efroni, and S. Mannor, "Action robust reinforcement learning and applications in continuous control," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6215–6224.

[8] H. Van Hasselt, Y. Doron, F. Strub, M. Hessel, N. Sonnerat, and J. Modayil, "Deep reinforcement learning and the deadly triad," *arXiv preprint arXiv:1812.02648*, 2018.

[9] A. Bora, A. Jalal, E. Price, and A. G. Dimakis, "Compressed sensing using generative models," in *International Conference on Machine Learning*. PMLR, 2017, pp. 537–546.

[10] Y. Wu, M. Rosca, and T. Lillicrap, "Deep compressed sensing," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6850–6860.

[11] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Improving sample efficiency in model-free reinforcement learning from images," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021, pp. 10 674–10 681.

[12] D. L. Donoho, "Compressed sensing," *IEEE Transactions on information theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[13] E. J. Candes, J. K. Romberg, and T. Tao, "Stable signal recovery from incomplete and inaccurate measurements," *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, vol. 59, no. 8, pp. 1207–1223, 2006.

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

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

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

[17] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[18] S. Kalyanakrishnan, S. Aravindan, V. Bagdawat, V. Bhatt, H. Goka, A. Gupta, K. Krishna, and V. Piratla, "An analysis of frame-skipping in reinforcement learning," *arXiv preprint arXiv:2102.03718*, 2021.