symmetry

MDPI

Article

Discrete Space Deep Reinforcement Learning Algorithm Based on Support Vector Machine Recursive Feature Elimination

Chayoung Kim

Bright College, Hankyong National University, 327 ungang-ro, Anseong-si 17579, Gyeonggi-do, Republic of Korea; kimcha0@hknu.ac.kr; Tel.: +82-31-670-5670

Abstract: Algorithms for training agents with experience replay have advanced in several domains, primarily because prioritized experience replay (PER) developed from the double deep Q-network (DDQN) in deep reinforcement learning (DRL) has become a standard. PER-based algorithms have achieved significant success in the image and video domains. However, the exceptional results observed in images and videos are not as effective in many domains with simple action spaces and relatively small states, particularly in discrete action spaces with sparse rewards. Moreover, most advanced techniques may improve sampling efficiency using deep learning algorithms rather than reinforcement learning. However, there is growing evidence that deep learning algorithms cannot generalize during training. Therefore, this study proposes an algorithm suitable for discrete action space environments that uses the sample efficiency of PER based on DDQN but incorporates support vector machine recursive feature elimination (SVM-RFE) without enhancing the sampling efficiency through deep learning algorithms. The proposed algorithm exhibited considerable performance improvements in classical OpenAI Gym environments that did not use images or videos as inputs. In particular, simple discrete space environments with reflection symmetry, such as Cart–Pole, exhibited a faster and more stable learning process. These results suggest that the application of SVM-RFE, which leverages the orthogonality of support vector machines (SVMs) across learning patterns, can be appropriate when the data in the reinforcement learning environment demonstrate symmetry.

Keywords: support vector machine recursive feature elimination; deep reinforcement learning with double Q-learning; prioritized experience replay; deep reinforcement learning; reflection symmetry

Citation: Kim, C. Discrete Space Deep Reinforcement Learning Algorithm Based on Support Vector Machine Recursive Feature Elimination. *Symmetry* **2024**, *16*, 940. https://doi.org/10.3390/sym16080940

Academic Editor: Jie Yang

Received: 8 July 2024
Revised: 17 July 2024
Accepted: 18 July 2024
Published: 23 July 2024

1. Introduction

Since Mnih et al. [1] proposed a deep Q-network (DQN) and demonstrated human-level performance in the Atari 2600 games, deep reinforcement learning (DRL) has achieved significant success in areas such as path planning, autonomous robot navigation, and natural language processing [2,3]. Recently, techniques such as reinforcement learning from human feedback (RLHF) [4] have emerged, enabling AI agents to be trained to satisfy human standards. The RLHF, popularized by Christiano et al. in 2017 [5], is used in conjunction with large language models (LLMs) [4] that require an enormous number of training parameters. This study was pivotal in applying feedback-based approaches to the evolution of DRL with a focus on continual reinforcement learning, various improvements, and multilayered approaches within deep-learning models. However, these techniques have not yet been standardized, particularly in the domains of image and video DRL, which are increasingly reliant on advances in artificial neural networks, such as diffusion models [6], rather than on reinforcement learning itself.

Thus, it is necessary to consider whether DRL techniques developed for the image and video domains can be effectively applied to other fields. Although DRL techniques based on artificial neural networks are advancing, there is growing evidence that they cannot generalize during training [7]. A critical issue is the effective implementation of predictive strategies for effective sampling in complex environments; this poses practical challenges

to the widespread use of reinforcement learning [8,9]. DRL agents require billions of interactions with their environments, resulting in extremely long training times. As the training progresses, the probability of the agent's learning capacity diminishing increases significantly [7,9]. Thus, efforts to enhance the sampling efficiency of DRL algorithms continue to be a focal point.

Double DQN (DDQN) [10] improves sample efficiency by mitigating overestimation through the separation of action value functions and action evaluation functions whereas dueling DQN [11] enhances sample efficiency by quickly learning the values of new actions using state and advantage value functions, thereby reducing redundant or similar sample training. DDQN-based prioritized experience replay (PER) [12] further improves efficiency by assigning priority to experience replay buffer samples based on TD errors, making it easier to select more efficient samples. PER provides two options—selecting experiences to store and selecting experiences to replay—and improves the latter by prioritizing traces of samples stored in the replay buffer. However, PER calculates the sampling rates for each transition independently, which increases the likelihood of sampling highly biased and redundant transitions. The neural experience replay sampler was designed to mitigate PER's bias [8]. Study [13] emphasized recently observed data to prevent forgetting past data based on a soft actor–critic (SAC) [14,15], demonstrating a significantly higher sample efficiency by applying the algorithm [13] with PER [12] to SAC [14,15].

However, since experience replay has achieved significant success in DQN [1], more research has focused on DQN rather than actor–critic [16]. Study [9] was also based on a DDQN, which proposed a prioritized technique for selecting samples before placing them in the experience replay buffer. Similar to study [9], the algorithm proposed in this study aims to improve the sample selection for replay. The difference between the proposed algorithm and the previously developed algorithm [9] lies in addressing discrete space learning issues. The latter uses a DDQN with prior knowledge using the mean square error of n samples to select the replay samples based on the average return values of the n samples. Although the previously developed algorithm [9] showed improved results in the image and video domains, it did not exhibit the same improvements in discrete space learning issues.

Existing reinforcement learning algorithms in discrete space learning typically face the problems of slow convergence and reduced accuracy [17]. Despite the need for extensive interactions with the learning samples during training, DRL struggles to converge in discrete space learning and frequently falls into local minima [17]. The primary reason for the preference for deep learning algorithms is to automate representation learning [18]. Large-scale data are necessary to automatically extract features suitable for a target task, thereby improving the accuracy. However, often, only "little data" are available in many domains outside of discrete space learning, necessitating analysts to engage in complex feature engineering and a thorough understanding of the domain. With limited training data, there is a higher probability of partial samples not reflecting the entire population, which leads to biased learning results and significant errors in generalization.

Therefore, the proposed algorithm aims to achieve effects similar to those of feature engineering by selecting the data required for discrete space learning using machine learning techniques. Specifically, the proposed algorithm uses support vector machine recursive feature elimination (SVM-RFE) [19] from machine learning to select a subset of effective samples during DRL training to enhance the accuracy. Insights were gained from a previous study [17] that linked an SVM [20] with actor–critic [16] relationships. However, in contrast to study [17], which requires pre-training for the sample selection criteria and gradient descent optimization, the proposed algorithm constructs and selects useful feature subsets during agent training without pre-training. Although deep learning methods have improved sample efficiency, random sampling remains highly effective in small, discrete environments. The proposed algorithm hypothesizes that constructing a subset of samples with highly relevant features through machine learning, rather than identifying all potentially relevant training features, will contribute to the objective. The

insight that machine learning can enhance sampling efficiency better than deep learning has also been derived from a previous study [21]. Study [21] utilized a technique called "least squares".

The main contribution of this study is that random sampling guided by sample feature patterns is more effective than deep learning-based sample selection, particularly in small discrete environments. Among machine learning algorithms, SVM-RFE iteratively employs an SVM to eliminate unnecessary features. The SVM is renowned as an algorithm that identifies optimal boundaries. Significant performance improvements were observed, particularly in environments with reflection symmetry, such as the classic environment of the OpenAI Gym [22]. Reinforcement learning uses the Markov decision process (MDP) [23], which can exhibit symmetry [24]. The Cart–Pole [25] involves a system in which a pole is poised on a cart, and the goal is to move the cart left and right to prevent the pole from toppling. Left–right symmetry means that moving the cart to the left and right is modeled in a physically identical manner. In other words, applying the same magnitude of force in opposite directions resulted in the same response. However, in all application environments where reinforcement learning is applied, the data may not exhibit symmetry similar to that of the Cart–Pole. Nonetheless, in environments in which the data exhibit symmetry, the application of SVM-RFE, which leverages the orthogonality of the SVM model approximations across training patterns, is deemed appropriate.

The remainder of this paper is structured as follows. Section 2 discusses the background. Section 3 introduces the proposed algorithm, which combines SVM-RFE with DDQN influenced by the sample efferent algorithm (PER). Section 4 summarizes the experimental results, and Section 5 concludes the study.

2. Background

2.1. Research Trends in Reinforcement Learning Based on LLMs

The RLHF [4] is a key technique used to train machine learning models for difficult-to-specify objectives and align AI systems with human goals [5,26,27]. The RLHF is a fundamental component in training state-of-the-art large language models (LLMs), such as Google Gemini [28], OpenAI's GPT-4o [29], and Meta's Llama 3 [30]. The RLHF has a relative advantage over manually engineered reward functions and other learning methods because it facilitates the identification of desirable behaviors. Christiano et al. [5] popularized the current standard methodology for the RLHF in 2017. However, models fine-tuned with the RLHF face ongoing challenges, such as sensitivity to private data [31], jailbreaks (breaking the constraints under which the system should operate normally), and the generation of hallucinated misinformation [32]. Therefore, further research is required to standardize and generalize these models. The study "Do as I can, not as I say" [33] investigates how to extract knowledge from LLMs so that embodied agents, such as reinforcement learning robots, can follow high-level textual instructions. Robots have a repertoire of learned skills for "atomic" actions and can determine whether their individual skills are likely to make progress toward completing advanced instructions beyond simply interpreting LLM instructions. The research "Eureka" [34] leveraged LLMs for high-level semantic frameworks in sequential decision-making tasks but showed that using LLMs for complex low-level manipulation tasks, such as proficient pen spinning, remains unresolved. Eureka aimed to bridge these gaps by proposing an LLM-driven human-level reward design algorithm using state-of-the-art LLMs such as GPT-4o to optimize reward codes and presented a new gradient-free in-context learning approach for RLHF. In addition, Eureka suggested the possibility of developing a universal reward programming algorithm using cutting-edge coding LLMs, such as GPT-4o.

2.2. Diffusion Models in Reinforcement Learning

Pearce et al. [35] proposed using expressive and stable diffusion models to better simulate human behavior. A diffuser [36] uses diffusion models as trajectory generators, with the entire trajectory of state–action pairs constituting a single sample for the diffusion

model. A separate return model is trained to predict the cumulative reward for each trajectory sample. The guidance of the return model is then applied to the reverse sampling step. The research in [37] uniquely utilizes diffusion models in reinforcement learning, applying them to the action space and transforming them into conditional diffusion models given the state. The diffusion model samples one action at a time and injects a Q-value function guidance during training to provide good empirical performance, particularly when approaching policy optimization from an offline reinforcement learning perspective.

2.3. Studies on Q-Learning

The goal of reinforcement learning is to learn an optimal policy, which is an algorithm that optimizes accumulated future rewards. Q-learning [38] is the most renowned reinforcement learning algorithm. It learns the expected value of the rewards, such as the Q-value, as represented in Equation (1).

$$Q_\pi(s,a) \equiv \mathrm{E}[R_1 + \gamma R_2 + \cdots | S_0 = s, A_0 = a, \pi] \tag{1}$$

A policy specifies the probability of performing action a in state s for all states and actions. Gradient descent is used to learn the parameter θ of the Q-value function, as expressed in Equation (2).

$$\theta_{t+1} = \theta_t + \alpha(\mathrm{R}_{t+1} + \gamma max_a Q(S_{t+1}, a; \theta_t) - Q(S_t, A_t; \theta_t))\nabla_{\theta_t} Q(S_t, A_t; \theta_t) \tag{2}$$

The target Y for Q-learning is expressed in Equation (3).

$$Y_t^Q \equiv R_{t+1} + \gamma max_a Q(S_{t+1}, a; \theta_t) \tag{3}$$

DQN [1] combines Q-learning with deep neural networks, parameterizing the Q-value function with θ, the network's parameters, to represent state s and action values. Two key concepts improve DQN's performance: (1) the use of a target network, which has the same structure but different parameters θ^- copied from the original network at step τ, and (2) the experience replay buffer, storing observed transitions (state, action, next_state, and reward) and sampling randomly when the buffer exceeds batch_size, thus removing temporal correlations among samples. Equation (4) expresses the inclusion of the target network.

$$Y_t^{DQN} \equiv R_{t+1} + \gamma max_{a'} Q^{\wedge}(S_{t+1}, a'; \theta_t^-) \tag{4}$$

However, Q-learning tends to overestimate Q-values owing to the "estimate of optimal future value". This occurs because Q-learning and DQN use a greedy policy with the max operator, estimating target Y with the same parameters θ and leading to overestimation. To address this issue, double Q-learning [39], expressed in Equation (5), separates action selection and evaluation with different parameters, using θ_t for action selection and θ' for evaluation, thus reducing overestimation.

$$Y_t^{DoubleQ} \equiv R_{t+1} + \gamma Q(S_{t+1} argmax_a Q(S_{t+1}, a; Q_t); Q_t') \tag{5}$$

Similar to how the DQN applies deep learning to Q-learning, the DDQN [10] applies deep learning to double Q-learning, as expressed in Equation (6).

$$Y_t^{DoubleDQN} \equiv R_{t+1} + \gamma Q(S_{t+1} argmax_{a'} Q^{\wedge}(S_{t+1}, a'; Q_t), \theta_t^-) \tag{6}$$

A DDQN uses a target network to evaluate actions, minimize the propagation of overestimated action values, and resolve overestimation issues.

3. Proposed Algorithm

Figure 1 shows the structure of the proposed algorithm, which combines SVM-RFE with PER based on DDQN to further enhance the sample efficiency. Most algorithms devel-

oped for the PER aim to enhance learning efficiency by prioritizing the experience replay buffer samples in the DDQN. Study [9] also expands on the PER by updating the priorities of TD errors and sample rewards during deep learning training. Similarly, the proposed algorithm uses the TD errors of the PER. However, a machine learning technique called SVM-RFE was used to construct a subset of the samples with the most relevant features for training before randomly inserting samples into the experience replay buffer. This approach aims to achieve the desired effects of feature engineering by effectively selecting samples for inclusion in an experience replay buffer before sampling for effective learning.

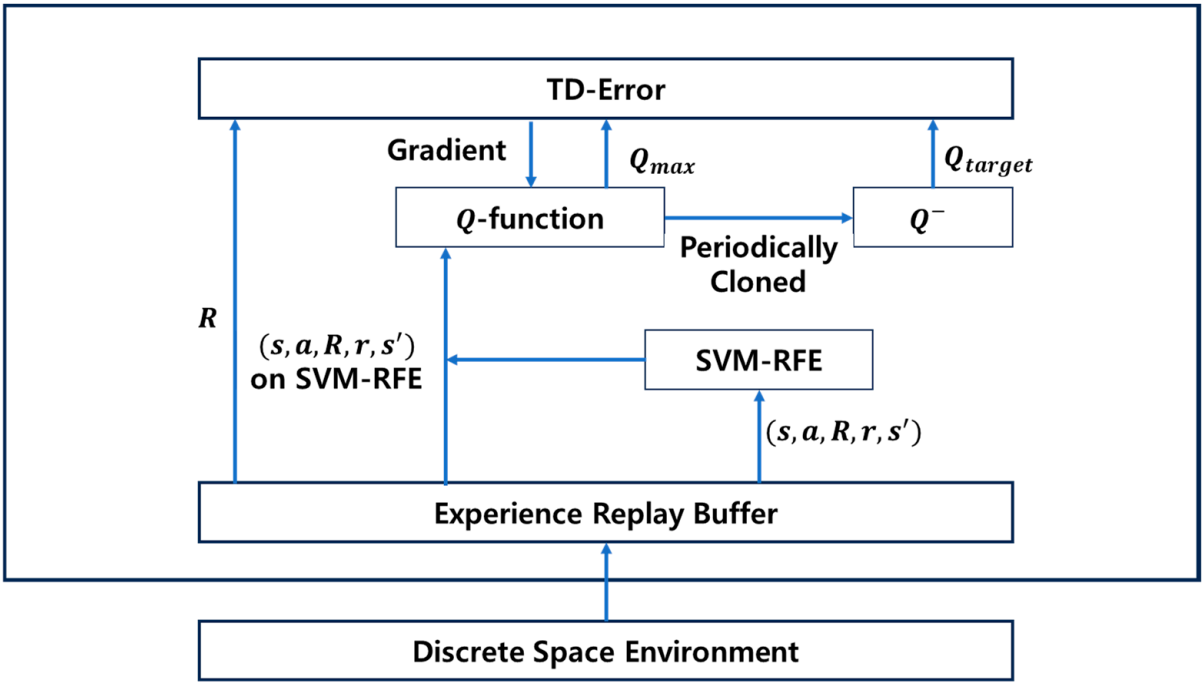

Figure 1. The structure of the proposed algorithm, which combines SVM-RFE with PER based on DDQN.

Although the approach of selecting a portion of the samples before inserting them into the experience replay buffer is similar to that in [9], which uses the mean square error of rewards with a DDQN and updates it based on a predefined reward mean square error to overcome its limitations, the proposed algorithm forms a subset of samples based on the criteria of the sample feature patterns determined via SVM-RFE. Therefore, in this study, the criteria for sample selection can be changed each time the SVM-RFE is executed in a batch, and this approach can use domain knowledge to achieve the effects of feature engineering. As can be observed from the results of this study, using a specific sample pattern as a criterion demonstrated much better sample efficiency in discrete environments with fewer samples compared with the reward mean square error of [9]. In the proposed algorithm, samples were used for agent training from the experience replay buffer if their reward values exceeded the SVM-RFE criterion; otherwise, they were not used.

Study [17] provided insights into our approach by incorporating support vector machines into reinforcement learning to address the traditional issues of DRL in small discrete spaces, such as "little data", not-easy algorithm convergence, and susceptibility to local minima. However, Ref. [17] pre-trained the advantage function to optimize the parameters of the support vector machine and combined actor–critic [16] with support vector machine classification, which might not aim to improve sampling efficiency during agent training but rather address the shortcomings of deep learning training. Study [21] was also predicated on the actor–critic and utilized a "least square" in machine learning to further enhance sample efficiency. Unlike this study, Ref. [21] targeted an online learning environment, meaning it could only use data obtained through interactions with the

environment. However, this study has the capability of offline learning, allowing for policy convergence by exploiting only pre-collected data or batches when interactions are not feasible, such as when no additional samples are provided. Because both online and offline learning have their own advantages and disadvantages, it is not a question of promoting one learning algorithm over the other. Instead, the determination of the learning algorithm can be adapted according to the specific circumstances of the environment.

In this study, SVM-RFE was used during agent training to change the criteria for selecting the data samples in each experience batch. It employed gradient descent through a DDQN, as used in the PER.

3.1. Preliminaries

3.1.1. SVM-RFE

An SVM [20] is a binary classification method that determines the optimal hyperplane to satisfy the classification requirements. Given a training set (x_i, y_i), $i = 1, 2, \ldots, l$, $x \in R_n$, $y \in \{\pm 1\}$ in a space with feature patterns, the hyperplane is represented as $(\omega \cdot x + b) = 0$. To ensure a classification margin, the constraint in Equation (7) is required.

$$y_i(\omega \cdot x_i + b) \geq 1, \; i = 1, 2, \ldots, l \tag{7}$$

Thus, the margin maximization problem is expressed in Equation (8), as follows [20]:

$$min_{\omega,b} \frac{1}{2} ||\omega||_2^2 \tag{8}$$

Typically, the ideal goal of learning is to approximate a cost function calculated from the training examples only. Therefore, the cost function is defined in Equation (9) as follows [20]:

$$cost - fuction \; (J) = \sum_{x \in X} ||\omega \cdot x - y||^2 \tag{9}$$

The OBD algorithm [40] approximates this by declaring the cost function used in the SVM, as shown in Equation (10):

$$Optimizing \; (J) = (\frac{1}{2}) \frac{\delta^2 J}{\delta \omega_i^2} \tag{10}$$

Equation (10) is then defined as in [19], by minimizing the following in Equation (11):

$$\left(\frac{1}{2}\right) ||\omega||^2 \tag{11}$$

While extracting good feature patterns does not necessarily provide the best subset ranking criteria, $(\omega_i)^2$ creates the effect of eliminating unnecessary features one at a time from the cost objective function and can be a reasonably effective method to obtain a subset of effective features for training by eliminating multiple useless feature patterns at once. This method is known as recursive feature elimination and can be used to iteratively remove unnecessary feature patterns as follows:

1. Train the classifier (optimize the weights ω_i of the objective function).
2. Compute the ranking criterion for $(\omega_i)^2$.
3. Remove the feature patterns with the smallest criteria.

3.1.2. Previous Sampling Algorithms

The PER uses a DDQN. Therefore, the majority of sampling studies have been based on the DDQN. In contrast to the DQN, the DDQN uses convolutional neural networks in the Q-network to approximate action values and a separate target network to calculate Q-network updates, thus separating the deep learning training components. In a DQN, the

maximum operation determines the optimal value, whereas in a DDQN, the action value of the optimal strategy is determined.

The loss functions for the DQN and DDQN are given in Equations (12) [1] and (13) [10], respectively.

$$\text{Loss(DQN)} = r + \gamma max_{a'} Q^{\wedge}(s', a'; \theta^{-}) - Q(s, a; \theta) \quad (12)$$

$$\text{Loss(DDQN)} = r + \gamma Q(s', arg\ max_{a'} Q(s'; a'; Q^{\wedge}); \theta^{-}) - Q(s, a; \theta) \quad (13)$$

where θ represents the parameters of the Q-network and θ^{-} represents the parameters of the target network. Generally, when training an AI agent, the goal is to minimize the difference between the target and the predicted values.

The PER is based on the DDQN to address the issue of nonstationary distribution. In the first stage, the samples resulting from the interaction between the agent and environment were stored in an experience replay buffer. In the second stage, small batch samples for training were selected based on TD errors, and different levels of importance were assigned to each experience; this increases the probability of significant experiences being sampled, thereby significantly enhancing the efficiency of agent training. A DQN obtains past experiences through random sampling, which assigns equal probability to all experiences in the experience replay buffer, potentially neglecting the varying importance of different experiences. Therefore, the PER based on the DDQN aims to overcome the shortcomings of a single DQN by employing a prioritized experience replay; this minimizes the probability that low-priority transitions cannot be extracted using a robust prioritization technique. This method ensures that significant experience is likely to influence the training process, thereby enhancing the overall robustness and efficiency of the reinforcement learning agent. The difference between the DQN-based and DDQN-based approaches based on their loss functions is illustrated in Figure 2.

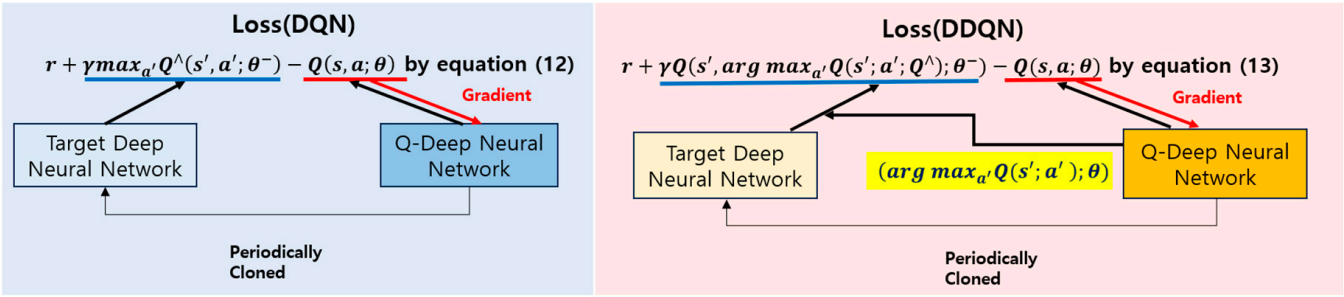

Figure 2. The difference between the DQN-based approach and the DDQN-based approach.

3.2. Proposed Efficient Sample Subset Based on SVM-RFE

This study used the TD errors of the PER in a DDQN, positing that random sampling remained more effective than deep learning-based sample selection in small discrete environments. As shown in [9]—a study focusing on sample selection for experience replay buffers based on PER—the prioritization of samples using TD errors did not demonstrate significant advantages in discrete space environments. The primary contribution of this study is to show that random sampling based on sample feature patterns can outperform deep learning-based sample selection, particularly in small discrete environments.

With insights from previous research [17], this study aimed to enhance the sample efficiency in discrete environments without relying on deep learning. Instead, support vector machine recursive feature elimination (SVM-RFE) was used for feature engineering during the DDQN agent training. The samples stored in the experience replay buffer were selectively chosen based on the SVM-RFE criteria to ensure that only subsets of effective training samples were included. Subsequently, samples were randomly selected. Random sampling ensures sampling efficiency [1]. In the PER, the transition of priority and rank-based prioritization based on the TD errors is used for sample selection. This study, as well

as TD errors, formed sample subsets based on TD error characteristics using SVM-RFE, rather than TD error ranking. It is assumed that TD errors exhibit certain patterns, and samples (s, a, R, r, s') are set with reward R as y and other vector values as x, similar to [17]. Then, x is input into SVM-RFE. Future studies will incorporate special processing layers for the input vectors, as described in [41].

In this study, SVM-RFE did not iterate until a single feature pattern remained. Instead, a criterion was randomly selected from a small number of remaining feature patterns for random sampling. Furthermore, the ξ -greedy strategy was employed to facilitate random sampling for DDQN, ensuring some level of randomness within the SVM-RFE-based sample subset to minimize the probability of exclusion. In the PER, samples are stored in the experience replay buffer, with priorities assigned based on the absolute value of TD errors. Reference [9] proposed tracking updates to increase the probability of sample selection with high TD errors. This study adopted a similar approach by periodically re-executing SVM-RFE on sample batches to continually update the criterion, achieving effects similar to those of priority tracking updates. However, enhancing the PER priority using deep learning, similar to [9], posed challenges in small discrete environments owing to issues such as convergence accuracy deficiency and susceptibility to local minima [17]. Reference [17] attempted to address these issues by excluding the deep learning structure, linking SVM with actor–critic models, and applying gradients to the "advantage function", which was directly suggested to solve discrete environment problems. Reference [21] was also predicated on the actor–critic and utilized a machine learning algorithm for engineering effects on the data. This study gained insight from [17], which proposed a method using machine learning, such as SVM-RFE, for feature engineering effects while using gradient methods with a DDQN without a specifically suggested function.

Algorithm 1 describes the proposed method. The command "Store transitions (S_{t-1}, A_{t-1}, R_t, γ_t, S_t) in V" stores samples in the buffer for use in SVM-RFE. $R_{svm-rfe}$ is the criterion for SVM-RFE, and the samples are stored in the replay memory H based on this criterion. In "For $j = 1$ to κ do," TD errors are calculated. In "Call $R_{svm-rfe}$," the criterion for stored samples is periodically recalculated. The procedure $R_{svm-rfe}$ sets y to one if the reward is greater than zero for each sample. Sample feature patterns are calculated based on the SVM model's weights ω, and one feature pattern is randomly selected as the criterion if multiple patterns remain.

Algorithm 1: DDQN influenced by PER with SVM-RFE.

Input: *minibatch* κ, *step* $-$ *size* η, *replay period* K, *size* K,
V for $SVM - RFE$, *exponent* α, β, *budget* T
Initialize *replay memory* H $= \varnothing$, $\Delta = 0$, $P_1 = 1$, $\xi = 0.1$, $random(b)$,
$Set_{SVM} = 0$
Observe S_0 and $A_0 \sim \pi_\theta(S_0)$
For $i = 1$ to T do
  Observe S_t, R_t, γ_t
  Store transition trajectories (S_{t-1}, A_{t-1}, R_t, γ_t, S_t) in V
  If $Set_{SVM} = 1$ and $R_t > R_{svm-rfe}$ and $b > \xi$ then
Store transition trajectories (S_{t-1}, A_{t-1}, R_t, γ_t, S_t) in H
with $P_t = max_{i<t} P_i$
  Else
Store transition trajectories (S_{t-1}, A_{t-1}, R_t, γ_t, S_t) in H
with $P_t = max_{i<t} P_i$
  If $t \equiv 0$ *mod* K then
For $j = 1$ to κ do
Do sample transition trajectories $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
Do compute $\omega_j = (N \cdot P(j)) - \beta / max_i \omega_i$
Do compute $TD - error$,

$$\delta_j = R_j + \gamma_j Q_{target}\left(S_j, a_{argmax} Q\left(S_j, a\right)\right) - Q\left(S_{j-1}, A_{j-1}\right)$$

Algorithm 1: *Cont.*

Do update $P_j \leftarrow \left|\delta_j\right|$
Do accumulate $\Delta \leftarrow \Delta + \omega_j \cdot \delta_j \cdot \nabla_\theta Q\left(S_{j-1}, A_{j-1}\right)$
End for
Call Procedure $R_{svm-rfe}$
Set $Set_{SVM} \leftarrow 1$
Update $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
Set $\theta_{target} \leftarrow \theta$
End if
Choose $A_t \sim \pi_\theta(S_0)$
End for

Procedure $R_{svm-rfe}$
$Surviving\_List[1 \ldots K]$, V for $SVM - RFE$
For $j = 1$ to K
Set x_j as $(S_j, A_j, \gamma_j, S_{j+1})$ in V
If $R_j > 0, y_j = 1$ in V
Else $y_j = -1$ in V
For $j = 1$ to $K - 1$
Train a model of SVM for ω_j
Compute the ranking criterion, ω_j^2
Remove the smallest ranking criterion, $Surviving\_List[j-1]$
Return one out of a few smallest $R_{svm-rfe} = Surviving\_List[1]$

4. Results

This study used the OpenAI Gym [22] platform as the environment for comparative experiments. This implementation was based on the DDQN [10] and integrated supervised learning through SVM-RFE to demonstrate the superiority of prioritized experience replay specifically tailored for discrete action space environments without relying on deep learning methods.

In this study, the "Classic Control" [42] environments of OpenAI Gym, namely Cart–Pole [25,43,44], Acrobot [45–47], and MountainCar [48–51], were selected as the experimental settings. The experiments used the well-known DDQN influenced by the PER and the proposed SVM-RFE integrated prioritized experience replay. The superior performance of our approach was validated through various comparisons, including reward metrics and optimal action percentages.

Classic control environments representative of reinforcement learning are modeled as MDPs based on the Bellman equations [23]. Symmetry can appear in MDPs, and the Cart–Pole exhibits reflection symmetry [24]. Notably, the results for Cart–Pole and Acrobot effectively demonstrate the advantages of the proposed algorithm. However, MountainCar provided rewards only when the car reached the top of the hill, presenting fewer sample states and sparser reward returns than Cart–Pole and Acrobot. Consequently, while the SVM-RFE-based prioritized experience replay proposed in this study was optimized for sample rewards, the improvement in returns was marginal and less pronounced than that in Cart–Pole or Acrobot when compared with the DDQN influenced by the PER.

4.1. Cart–Pole

The Cart–Pole [25] is a representative of "Classic Control". The dynamics of Cart–Pole exhibit reflection symmetry with respect to the vertical axis. A pole is attached to a cart moving along the track. The pole was positioned at a 90° angle to the cart, and the objective was to maintain the balance of the pole as the cart moved left and right. Table 1 lists two action options, $\{0, 1\}$, that represent the direction of the force applied to the cart. The observed values were the position and velocity of the cart. During an episode, the cart position ranged between -4.8° and $+4.8^\circ$, and the pole angle ranged between -0.418

and +0.418 rad. A reward was given for maintaining the pole upright, as represented by a positive value of 1. The reward threshold for the cart pole was 500. An episode terminates when one of the following conditions is met: (1) the pole angle exceeds -12° to $+12^{\circ}$, (2) the position of the cart exceeds −2.4 to +2.4 (i.e., the cart reaches the edge of the display), or (3) the episode length exceeds 500 steps (this condition can be optional). The movements of the cart pole demonstrated symmetry with respect to a plane perpendicular to the direction of the movement of the cart in the state–action space. Therefore, evaluating the improvement effects based on reward increases is suitable for environments such as the Cart–Pole.

Table 1. Environment of Cart–Pole [25].

Cases	**Action Spaces**		
0	Move the cart to the left		
1	Move the cart to the right		
Cases	**Observation Spaces**	**Minimum**	**Maximum**
0	Cart's Position	−4.8	+4.8
1	Cart's Velocity	−Inf	+Inf
2	Pole's Angle	−0.418 *rad* (−24°)	+0.418 *rad* (+24°)
3	Pole's Angular Speed	−Inf	−Inf
Each Step		**The Reward**	
To maintain the pole in an upright position for the maximum duration		+1	
Cases	**Termination**		
0	Pole′s angle is more than $\pm 12^{\circ}$		
1	Cart′s position is more than ± 2.4		
2	Episode duration is more than 500		

For a better solution, specific considerations such as outputting "the best test reward" at each episode step were considered. In some cases, if the episode ends without reaching the maximum value, a negative reward, −1, is assigned, indicating that extending the episode does not always increase "the best test reward". Figure 3 shows the average instances of "the best test reward" over 100 episodes. The proposed SVM-RFE-integrated sample-efficient method was compared with the well-known DDQN based on the PER. Similar results were obtained in this study. However, the existing method exhibited greater instability in the expected rewards. Hence, in Figure 4, only the worst-case results among various execution examples were compared, confirming that the existing method was much more unstable than the average cases of "the best test reward". This indicates that the sample-efficient method can achieve reasonable results at an earlier stage, owing to the sample efficiency rather than significantly improving it by extending the episode length.

4.2. Acrobot

Acrobot [45] is also a part of "Classic Control", similar to Cart–Pole. The system consists of a chain structure linearly connected by two links, with one end of the chain fixed. As listed in Table 2, a joint operates between the two links. The objective of the task was to apply torque to the joint (actuator) such that the chain, starting from the initial state of hanging downward, swung or rotated its free end above a given height (the black horizontal line above the system). The two blue links were connected by two green joints. Therefore, the action space appeared to be activated by applying a torque to operate the joint between the two links. As shown, θ_1 is the angle of the first joint, and θ_2 is the angle of the second joint relative to the first link. An angle of 0 for θ_1 indicates that the first link is pointing downward, and an angle of 0 for θ_2 indicates that the angle between the two links is the same. θ_1 and θ_2 are velocity-limited to $\pm 4\pi$ and $\pm 9\pi$ rad/s, respectively. A reward was received when the free end reached the specified target height (the black horizontal

line above the system) in as few steps as possible. If the target was not reached, a negative reward of -1 was assigned, and the episode was terminated with a reward of zero upon reaching the target. The reward threshold was -100. The episode typically started with both links pointing downward and ended if one of the following conditions was met: the free end satisfied $-\cos\theta_1 - \cos(\theta_2 + \theta_1) > 1.0$ or the episode length exceeded 500 steps.

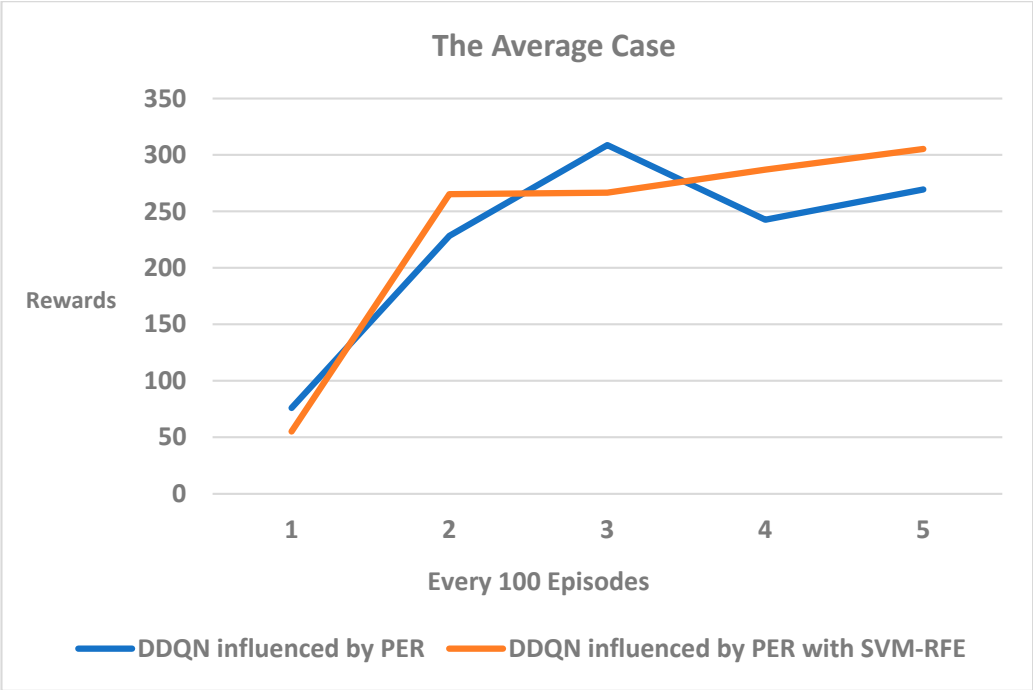

Figure 3. Average case of "The best test reward" over 100 episodes.

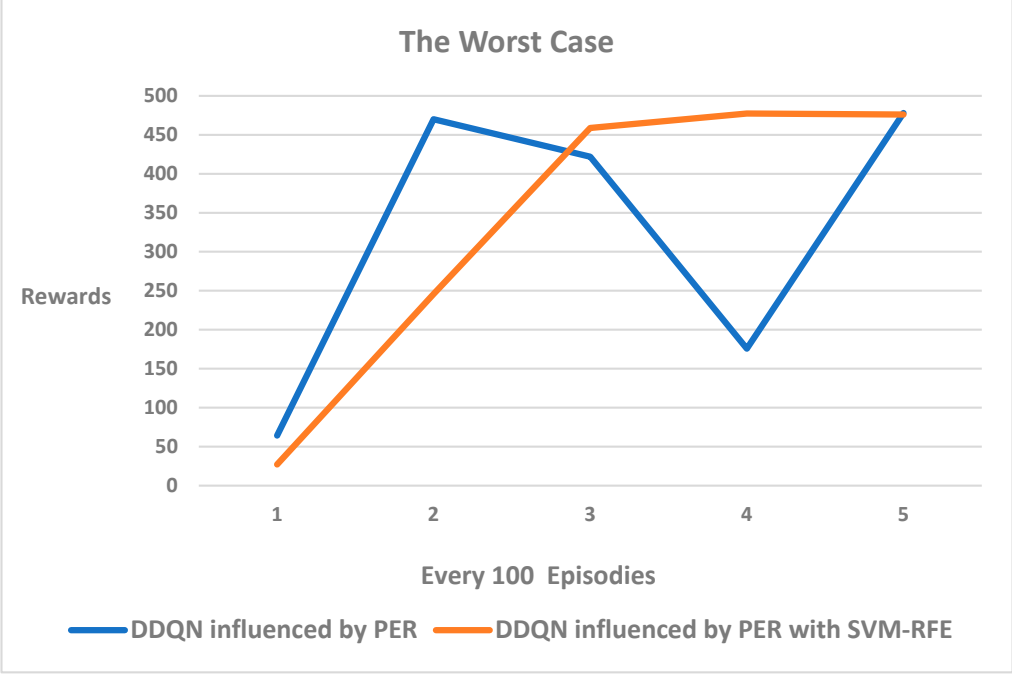

Figure 4. The worst case over 100 episodes.

Table 2. Environment of Acrobot [45].

Cases	Action Spaces		
0	Apply a torque of -1 to the driven joint		
1	Apply a torque of 0 to the driven joint		
2	Apply a torque of $+1$ to the driven joint		
Cases	**Observation Spaces**	**Minimum**	**Maximum**
0	$cos(\theta_1)$	-1	$+1$
1	$sin(\theta_1)$	-1	$+1$
2	$cos(\theta_2)$	-1	$+1$
3	$sin(\theta_2)$	-1	$+1$
4	Angular velocity of θ_1	$\sim -12.567(-4 * pi)$	$\sim 12.567(4 * pi)$
5	Angular velocity of θ_2	$\sim -28.274(-9 * pi)$	$\sim 28.274(9 * pi)$
Each Step		**The Reward**	
To reach the target height before termination		$0\ (threshold : -100)$	
If the target is not reached before termination		-1	
Cases		**Termination**	
1		The free end reaches the target height: $-cos(\theta_1) - cos(\theta_2 + \theta_1) > 1.0$	
2		Episode duration is more than 500	

Figure 5 shows the average number of episodes for lengths less than or equal to 150, 200, and 500. Similar to the case of Cart–Pole, the proposed sample-efficient method integrating SVM-RFE was compared with the well-known DDQN influenced by the PER. The proposed method showed a higher frequency of episodes that ended with shorter lengths. However, for a maximum episode length of 500, both the proposed and the well-known methods performed similarly. The results indicate that the proposed method generally reached conclusions faster on average because the worst-case scenario, that is, the longest episode length, occurred less frequently. Figure 6 presents a box plot comparing the optimal action percentages for episodes of average length for a more qualitative comparison. Although the box plot was small, the proposed SVM-RFE-integrated sample-efficient method exhibited significant differences. The proposed method demonstrated a lower variability at the median value because of the relatively small difference between the minimum and maximum values. These results indicate that the proposed method was considerably more stable in terms of agent learning.

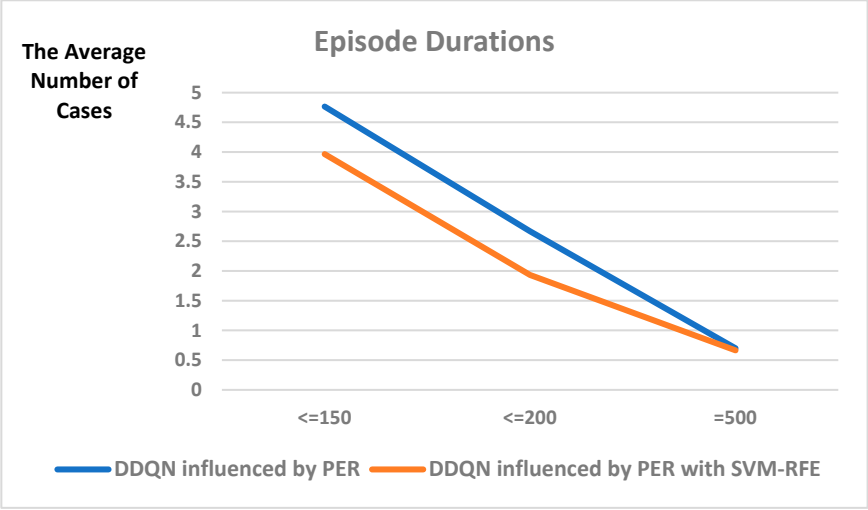

Figure 5. Average number of episodes for 150, 200, and 500, respectively.

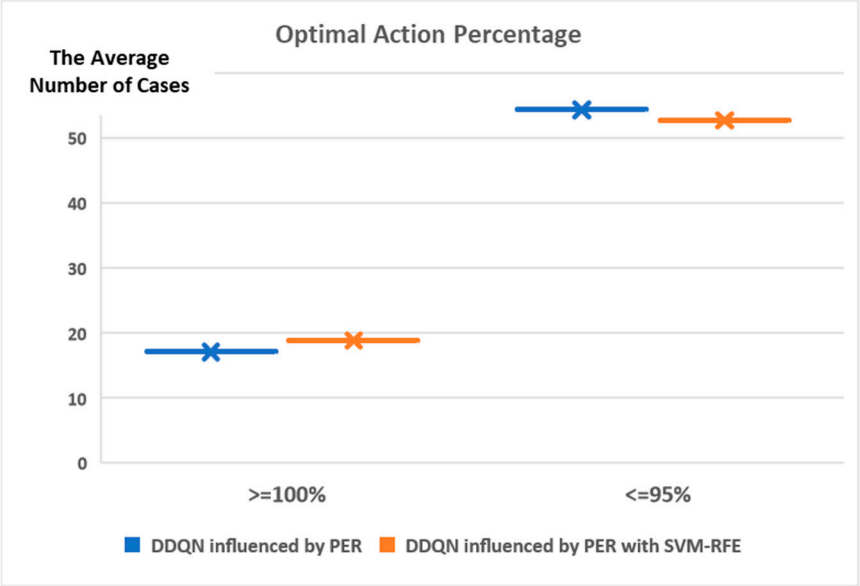

Figure 6. Optimal action percentage for episodes of average length.

4.3. MountainCar

MountainCar [48] is an example of an environment with extremely sparse rewards [52]. Similar to Acrobot [45] and Cart–Pole [25], MountainCar [48] is a classic discrete environment in OpenAI Gym [22]. The MountainCar comprises a car placed at the bottom of a sine curve (MDP) [23]. As listed in Table 3, the only possible control is acceleration, which can be applied in different directions. The objective of this environment was to tactically accelerate the car to reach the top of the right hill. Because the car engine was not powerful enough to climb a hill in one step, the only way to gain momentum was to move back and forth. Initially, the car started at rest at the bottom of the valley between the hills (approximately at the −0.5 mark). The episode ended when the car reached the flag position (approximately +0.5) or after 200 steps. The available actions for the cars were 1. push left, 2. push right, and 3. do nothing. A penalty of −1 was applied for each step taken until the goal was reached.

Table 3. Environment of MountainCar [48].

Cases	**Action Spaces**		
0	Speed up to the left		
1	Do not speed up		
2	Speed up to the right		
Cases	**Observation Spaces**	**Minimum**	**Maximum**
0	Position of the car along the *x*-axis	−1.2	0.6
1	Velocity of the car	−0.07	+0.07
Each Step		**The Reward**	
To arrive at the flag atop the right hill in the shortest time possible		−1 (When a penalty is applied)	
Cases	**Termination**		
1	The car's position is at least 0.5, which is the target location On the peak of the right hill.		
2	Episode duration is more than 200		

Figure 7 shows a comparison of the changes in rewards from episodes 500 to 1000. When comparing the proposed sample-efficient method integrating SVM-RFE with the

well-known DDQN based on the PER, it was difficult to identify significant changes in the reward. Therefore, we selected only the worst-case scenario in Figure 8 and the best-case scenario in Figure 9 for comparison. In the worst-case scenario, the proposed and existing methods were very similar. However, in the best-case scenario, the proposed method exhibited a slight improvement, although the improvement was not significant. In other words, the reward of the agent determines "how quickly the agent can learn within the expected period". Unlike Cart–Pole or Acrobot, ensuring rapid and accurate learning on MountainCar is difficult. MountainCar is representative of a sparse environment compared with Cart–Pole and Acrobot. Therefore, instead of focusing solely on rewards, it may be beneficial to consider other aspects, such as those in the Bayesian study [53], to confirm improvements in the agent's learning from the loss function. In addition, the agent has been extensively explored in sparse environments, which has led to significant variability. In such cases, determining the effectiveness of various rare samples using supervised learning methods may be challenging. Therefore, in sparse environments, selecting a specific model and accumulating knowledge using a prior model, similar to other studies [53], may be more effective. The next step was to investigate various techniques that could demonstrate the meaningful effects of the sample-efficient method, even in sparse environments.

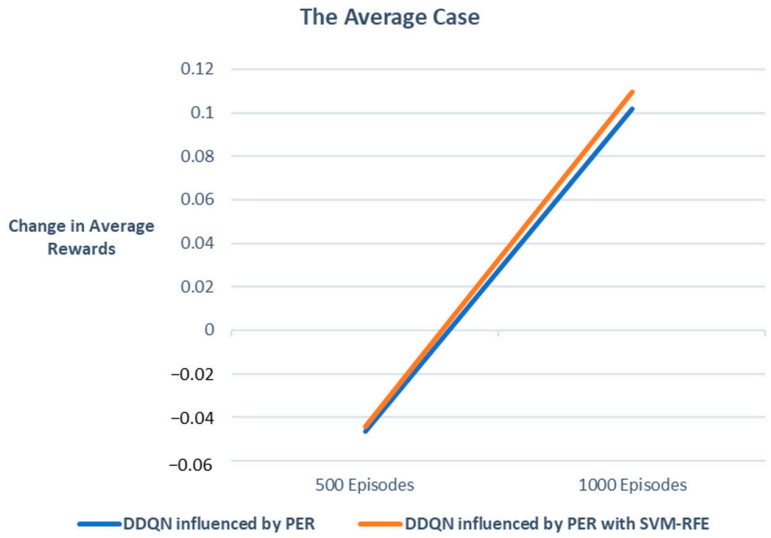

Figure 7. Changes in reward from 500 to 1000.

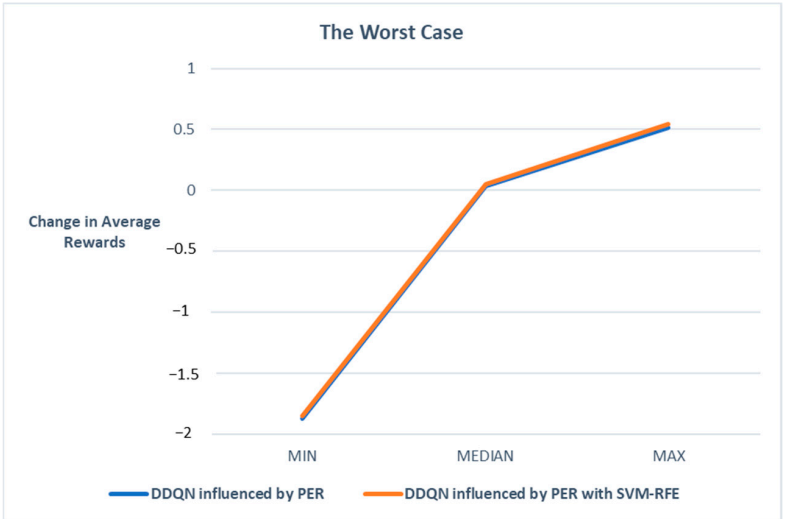

Figure 8. Worst case in minimum, median, and maximum.

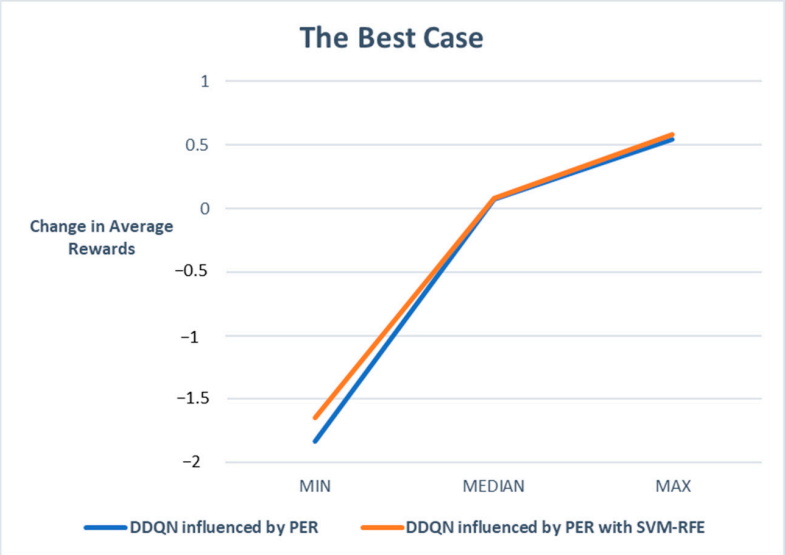

Figure 9. Best case in minimum, median, and maximum.

5. Discussion

When receiving rewards in environments like MountainCar is more infrequent compared to Cart–Pole or Acrobot, it can be observed that the results for MountainCar do not significantly improve compared to Acrobot or Cart–Pole; this indicates that the data for feature engineering effects must be more specific and diverse. This study was based on the premise that random sampling is much more effective than deep learning in small discrete environments. Based on previous studies [17,21], we integrated a machine learning method to refine the random sampling more precisely. The premise of this study is not to assert that machine learning is superior to deep learning but rather that the environments in which deep learning and machine learning algorithms are applied can differ. This suggests that the engineering effects of machine learning must be more detailed and vary in sparse environments. In sparse environments, the more the agent explores, the greater the variation. Therefore, a loss function can be suggested for sparse environments, as proposed in study [53]. In particular, the loss function of the PER based on the DDQN used in this study will be re-proposed based on a previous study [53], and demonstrating more effective sampling compared to various algorithms will be the next step of the research.

6. Conclusions

In this study, we proposed an algorithm for selecting a subset of effective samples for learning in discrete space environments using SVM-RFE, a machine learning technique. The goal was to achieve feature-engineering-like effects that would improve the rewards and accuracy of agent learning within a relatively short period. Although leveraging prior knowledge to improve learning often requires specialized application structures or deep learning enhancements, this study aims to construct a subset of samples with the most relevant features for training without such structures. The objective was to use machine learning to build this subset, thereby improving learning efficiency.

The experimental results of this study revealed that random, informed by sample feature patterns, is more effective than deep learning-based sample selection, particularly in small discrete environments. Compared to the existing DDQN influenced by the PER, significant differences were observed in the reflection symmetry environments. Since reinforcement learning is based on MDPs with symmetrical properties, the data used for agent learning can exhibit symmetry. SVM-RFE can effectively leverage the symmetrical characteristics of data by repeatedly utilizing the orthogonality of SVM approximations. However, despite being based on the MDPs, it is difficult to observe significant differences in sparse environments. Therefore, future studies should focus on proposing loss functions

that can demonstrate the meaningful effects of sample-efficient methods, even in sparse environments, and devise various techniques to handle these functions.

Funding: This study received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.; Veness, J.; Bellemare, M.; Graves, A.; Riedmiller, M.; Fidjeland, A.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
2. Kober, J.; Peters, J. Reinforcement learning in robotics: A survey. *Int. J. Robot. Res.* **2013**, *32*, 1238–1274. [CrossRef]
3. Mirowski, P.; Pascanu, R.; Viola, F.; Soyer, H.; Ballard, A.; Banino, A.; Denil, M.; Goroshin, R.; Sifre, L.; Kavukcuoglu, K.; et al. Learning to navigate in complex environments. *arXiv* **2016**, arXiv:1611.03673.
4. Casper, S.; Davies, X.; Shi, C.; Gilbert, T.K.; Scheurer, J.; Rando, J.; Freedman, R.; Korbak, T.; Lindner, D.; Freire, P.; et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv* **2023**, arXiv:2307.15217.
5. Christiano, P.F.; Leike, J.; Brown, T.; Martic, M.; Legg, S.; Amodei, D. Deep reinforcement learning from human preferences. In Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, 4–9 December 2017; pp. 4302–4310.
6. Ho, J.; Jain, A.; Abbeel, P. Denoising diffusion probabilistic models. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS2020), Virtual Conference, 6–12 December 2020; pp. 6840–6851.
7. D'Oro, P.; Schwarzer, M.; Nikishin, E.; Bacon, P.; Bellemare, M.G.; Courville, A.C. Sample-Efficient Reinforcement Learning by Breaking the Replay Ratio Barrier. In notable-top-5%. In Proceedings of the International Conference on Learning Representations (ICLR 2023), Kigali, Rwanda, 1–5 May 2022.
8. Oh, Y.; Lee, K.; Shin, J.; Yang, E.; Hwang, S.J. Learning to sample with local and global contexts in experience replay buffer. *arXiv* **2020**, arXiv:2007.07358.
9. Wang, X.; Xiang, H.; Cheng, Y.; Yu, Q. Prioritized experience replay based on sample optimization. *J. Eng. IET J.* **2020**, *2020*, 298–302.
10. Van Hasselt, H.; Guez, A.; Silver, D. Deep Reinforcement Learning with Double Q-learning. In Proceedings of the AAAI'16 Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; pp. 2094–2100.
11. Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; Freitas, N. Dueling network architectures for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning (ICML 2016), New York, NY, USA, 19–24 June 2016; pp. 1995–2003.
12. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. *arXiv* **2015**, arXiv:1511.05952.
13. Wang, C.; Ross, K. Boosting soft actor-critic: Emphasizing recent experience without forgetting the past. *arXiv* **2019**, arXiv:1906.04009.
14. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv* **2018**, arXiv:1801.01290.
15. Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. Soft actor-critic algorithms and applications. *arXiv* **2018**, arXiv:1812.05905.
16. Konda, V.; Tsitsiklis, J. Actor-Critic Algorithms. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NIPS1999), Denver, CO, USA, 29 November–4 December 1999; pp. 1008–1014.
17. An, Y.; Ding, S.; Shi, S.; Li, J. Discrete space reinforcement learning algorithm based on support vector machine classification. *Pattern Recognit. Lett.* **2018**, *111*, 30–35. [CrossRef]
18. SAMSUNG SDS. Available online: https://www.samsungsds.com/kr/insights/ai_automl.html (accessed on 1 July 2024).
19. Guyon, I.; Weston, J.; Barnhill, S.; Vapnik, V. Gene selection for cancer classification using Support Vector Machines. *Mach. Learn.* **2002**, *46*, 389–422. [CrossRef]
20. Boser, B.; Guyon, I.; Vapnik, V. A training algorithm for optimal margin classifiers. In Proceedings of the Fifth Annual Workshop on Computational Learning Theory, Pittsburgh, PA, USA, 27–29 July 1992; pp. 144–152.
21. Ren, J.; Lan, Y.; Xu, X.; Zhang, Y.; Fang, Q.; Zeng, Y. Deep reinforcement learning using least-squares truncated temporal-difference. *CAAI Trans. Intell. Technol.* **2024**, *9*, 425–439. [CrossRef]
22. OpenAI Gym. Available online: https://gymnasium.farama.org/environments/classic_control/ (accessed on 1 July 2024).
23. Bellman, R. A Markovian Decision Process. *J. Math. Mech.* **1957**, *6*, 679–684. [CrossRef]
24. Van Der Pol, E. Symmetry and Structure in Deep Reinforcement Learning. Ph.D. Dissertation, Universiteit van Amsterdam, Amsterdam, The Netherlands, 12 July 2023.
25. CartPole. Available online: https://gymnasium.farama.org/environments/classic_control/cart_pole/ (accessed on 1 July 2024).
26. Ziegler, D.M.; Stiennon, N.; Wu, J.; Brown, T.B.; Radford, A.; Amodei, D.; Christiano, P.; Irving, G. Fine-tuning language models from human preferences. *arXiv* **2019**, arXiv:1909.08593.

27. Bai, Y.; Jones, A.; Ndousse, K.; Askell, A.; Chen, A.; DasSarma, N.; Drain, D.; Fort, S.; Ganguli, D.; Henighan, T.; et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv* **2022**, arXiv:2204.05862.
28. Google. Gemini. 2024. Available online: https://deepmind.google/technologies/gemini/ (accessed on 1 July 2024).
29. OpenAI. Gpt-4o. 2024. Available online: https://openai.com/index/hello-gpt-4o/ (accessed on 1 July 2024).
30. Meta's Llama 3. 2024. Available online: https://llama.meta.com/ (accessed on 1 July 2024).
31. Li, H.; Guo, D.; Fan, W.; Xu, M.; Song, Y. Multi-step jailbreaking privacy attacks on chatgpt. *arXiv* **2023**, arXiv:2304.05197.
32. Zhang, M.; Press, O.; Merrill, W.; Liu, A.; Smith, N.A. How language model hallucinations can snowball. *arXiv* **2023**, arXiv:2305.13534.
33. Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv* **2022**, arXiv:2204.01691.
34. Ma, Y.J.; Liang, W.; Wang, G.; Huang, D.A.; Bastani, O.; Jayaraman, D.; Zhu, Y.; Fan, L.; Anandkumar, A. Eureka: Human-level reward design via coding large language models. *arXiv* **2023**, arXiv:2310.12931.
35. Pearce, T.; Rashid, T.; Kanervisto, A.; Bignell, D.; Sun, M.; Georgescu, R.; Macua, S.V.; Tan, S.Z.; Momennejad, I.; Hofmann, K.; et al. Imitating human behavior with diffusion models. *arXiv* **2023**, arXiv:2301.10677.
36. Janner, M.; Du, Y.; Tenenbaum, J.B.; Levine, S. Planning with diffusion for flexible behavior synthesis. *arXiv* **2022**, arXiv:2205.09991.
37. Wang, Z.; Hunt, J.J.; Zhou, M. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv* **2022**, arXiv:2208.06193.
38. Watkins, C.J.C.H.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
39. Van Hasselt, H. Double Q-learning. In Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS 2010), Vancouver, BC, Canada, 6–9 December 2010.
40. Cun, Y.L.; Denker, J.S.; Solla, S.A. Optimal brain damage. In *Advances in Neural Information Processing Systems (NIPS 1989)*; Morgan Kaufmann: San Francisco, CA, USA, 1990; pp. 598–605.
41. Kim, K.; Joo, C. Agnostic Architecture for Heterogeneous Multi-Environment Reinforcement Learning. In Proceedings of the NeurIPS 2023 Foundation Models for Decision Making Workshop, New Orleans, LA, USA, 15 December 2023.
42. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd ed.; MIT Press: Cambridge, MA, USA, 2018.
43. CartPole DQN. Available online: https://github.com/rlcode/reinforcement-learning-kr-v2/tree/master/2-cartpole/1-dqn (accessed on 1 July 2024).
44. CartPole DDQN. Available online: https://github.com/amirmirzaei79/CartPole-DQN-And-DDQN (accessed on 1 July 2024).
45. Acrobot. Available online: https://gymnasium.farama.org/environments/classic_control/acrobot/ (accessed on 1 July 2024).
46. Acrobot DQN. Available online: https://github.com/Slobodian17/Q-learning-Acrobot-v1/blob/main/main.py (accessed on 1 July 2024).
47. Acrobot DDQN. Available online: https://github.com/JustinStitt/acrobotDDQN/blob/master/DDQN.py (accessed on 1 July 2024).
48. MountainCar. Available online: https://gymnasium.farama.org/environments/classic_control/mountain_car/ (accessed on 1 July 2024).
49. MountainCar DQN. Available online: https://github.com/shivaverma/OpenAIGym/blob/master/mountain-car/MountainCar-v0.py (accessed on 1 July 2024).
50. MountainCar DQN. Available online: https://colab.research.google.com/drive/1T9UGr7vdXj1HYE_4qo8KXptIwCS7S-3v (accessed on 1 July 2024).
51. MountainCar DDQN. Available online: https://github.com/DanielPalaio/MountainCar-v0_DeepRL/tree/main/DuelingDQN (accessed on 1 July 2024).
52. Noel, A.D.; Van Hoof, C.; Millidge, B. Online reinforcement learning with sparse rewards through an active inference capsule. *arXiv* **2021**, arXiv:2106.02390.
53. Kim, C. Deep Q-Learning Network with Bayesian-Based Supervised Expert Learning. *Symmetry* **2022**, *14*, 2134. [CrossRef]