ITU/CSAR

Improving Sample Efficiency of Reinforcement Learning Control Using Autoencoders

Burak Er[1], **Mustafa Doğan**[2]

[1] Istanbul Technical University
[2] Istanbul Technical University
[3]

Abstract: This study presents a novel approach for improving the sample efficiency of reinforcement learning (RL) control of dynamic systems by utilizing autoencoders. The main objective of this research is to investigate the effectiveness of autoencoders in enhancing the learning process and improving the resulting policies in RL control problems. In literature, most applications use only the latent space of autoencoders during learning. This approach can cause loss of information, difficulty in interpreting latent space, difficulty in handling dynamic environments, and outdated representation. In this study, the proposed novel approach overcomes these problems and enhances sample efficiency by using both states and their latent space during learning. The methodology consists of two main steps. First, a denoising-contractive autoencoder is developed and implemented for RL control problems, focusing on state representation and feature extraction. The second step involves training a Deep Reinforcement Learning algorithm using the augmented states generated by the autoencoder. This algorithm is compared against a baseline Deep Q-Network (DQN) in the LunarLander environment, where observations are subject to Gaussian noise.

Keywords: Reinforcement Learning, Autoencoders, Control

Pekiştirmeli Öğrenme Kontrolde Otokodlayıcılar ile Örnek Verimliliğini Arttırma

Özet: Pekiştirmeli öğrenme yöntemlerinde eğitim sırasında çok sayıda örnek gerekmesi zaman kaybına, bütçesel problemlere neden olmakta ve gerçek dünya uygulamalarında risk alınmasına sebep olmaktadır. Bu çalışmada otokodlayıcılar yardımıyla sistemin durumlarının kodlanması ile elde edilen gizli uzay durumlarla birlikte kullanılarak herhangi bir pekiştirmeli öğrenme yönteminin eğitimi için örnek verimliliğinin arttırılması amaçlanmaktadır. Literatürde sadece gizli uzayın kullanıldığı durumlarda bilgi kaybı, sistem dinamiğinin değişmesi durumunda gizli uzayın durumların güncel olmayan bir temsilini içermesi gibi problemlere neden olmaktadır. Bu çalışmada sunulan yeni algoritma ile örnek olarak Derin Q-Ağı (DQN) yönteminin verimliliği arttırılmıştır. Temel DQN ve Otokodlayıcı ile Geliştirilmiş DQN algoritması OpenAI Gym Lunar Lander sisteminde denenerek birbirleriyle karşılaştırılmış, hem örnekleme verimliliğinin arttırdığı hem de önceki çalışmalardaki sorunların üstesinden gelindiği gösterilmiştir.

Anahtar Kelimeler: Pekiştirmeli Öğrenme, Otokodlayıcılar, Kontrol

RESEARCH PAPER

Corresponding Author: Burak Er, erb20@itu.edu.tr

Reference: B. Er, M. Doğan, (2024), Improving Sample Efficiency of Reinforcement Learning Control Using Autoencoders, ITU Computer Science AI and Robotics, 1, (1) 39–48.

Submission Date: 09, May, 2024
Acceptance Date: 15, May, 2024
Online Publishing: 20, Jul, 2024

1 Introduction

A subset of machine learning called reinforcement learning (RL) teaches agents to make series of decisions by environment interaction ([1]). These agents learn to optimize their decisions, or actions, over time to maximize cumulative reward. This concept has been particularly effective in several domains, including game playing, resource management, and control tasks ([2], [3]). Control of dynamic systems, which are systems that change over time, is one of the areas where RL has shown substantial promise. Dynamic systems are prevalent in various fields such as robotics, economics, and biology. They often involve complex, non-linear behaviors that are difficult to model and control with traditional methods. RL, with its capacity to learn complex behaviors from interaction with the environment, presents an attractive approach to these challenges. However, one major challenge in applying RL to dynamic system control is the issue of sample efficiency. To learn an efficient strategy, many interactions with the environment are frequently necessary for RL techniques. This is particularly problematic in dynamic systems, where each interaction can be expensive, time-consuming and risky ([4]).

Model-based RL provides a compelling avenue to address sample efficiency. It builds a model of the environment to simulate outcomes of various actions, reducing the necessity for extensive exploration in the actual environment. Dyna-Q ([5]) was one of the earliest attempts to combine model-free and model-based RL. It alternates between updating the model of the environment and the Q-values, leveraging simulated experience to learn the policy. Probabilistic Ensembles with Trajectory Sampling (PETS) ([6]) uses an ensemble of probabilistic models to make predictions about future states and rewards. This approach allows the uncertainty of the model's predictions to be incorporated into the policy, improving robustness and efficiency. AlphaNPI ([7]) combines model-based planning with imitation learning, using a model of the environment for lookahead search to train a neural program interpreter. CCFDM is a notable example of this approach.CCFDM trains a deep convolutional neural network-based image encoder using contrastive learning and a forward dynamics model. Given the current observation and action, the forward dynamics model predicts the following observation, and the prediction error is employed as an intrinsic reward. This encourages the agent to explore novel observations, thereby improving the sample efficiency of the RL algorithm ([8]).

In literature unsupervised representation learning techniques are used to improve Deep Reinforcement Learning ([9]). But these methods are generally used for processes which have too high dimensions like images. Also only latent space is used in these techniques. This approach can cause loss of information, difficulty in interpreting latent space, difficulty in handling dynamic environments and outdated representation.

Improving exploration strategies is another approach to increase sample efficiency in RL. UCT ([10]) incorporates a measure of uncertainty into the selection of actions, improving exploration in domains with large action spaces. Intrinsic Motivation ([11]) introduces an additional reward, encouraging the agent to explore less familiar states. Random Network Distillation (RND) ([12]) measures novelty by comparing the output of two networks, one fixed and one learnable, and uses this to incentives exploration.

Transfer Learning and Meta-Learning can also enhance sample efficiency by utilizing prior knowledge or quickly adapting to new tasks. Progressive Networks ([13]) encapsulate knowledge from previous tasks in reusable modules, allowing the transfer of skills to new tasks. Model-Agnostic Meta-Learning (MAML) ([14]) trains a model to learn quickly from a few examples, making it suitable for new task adaptation. Reptile ([15]) simplifies the MAML approach, using a meta-gradient descent to improve efficiency and stability in meta-learning.

Offline RL, also known as batch RL, leverages pre-collected data to improve policies, making it a potentially efficient approach for sample-efficiency. Deep Q-Learning from Demonstrations (DQfD) ([16]) combines demonstration data with online interaction, enabling the RL agent to learn both from human demonstrations and exploration. Conservative Q-Learning (CQL) ([17]) augments the value function with an auxiliary loss to learn a conservative Q-function and prevent overestimation, thus stabilizing the learning process in offline RL. Reinforcement Learning with Augmented Data (RAD) ([18]) extends offline RL to visual domains, using data augmentations to leverage the full batch of data effectively.

Autoencoders are a type of Artificial Neural Networks used for encoding of input data. Autoencoders can develop the ability to represent high-dimensional states in a latent space with lesser dimensions, potentially simplifying the learning task for the RL agent. Embed to Control (E2C) ([19]) combines autoencoders with model-based RL for continuous control tasks. It uses a variational autoencoder to learn a low-dimensional representation of the environment, then learns a linear control model in this latent space. Ha and Schmidhuber ([20]) proposed World Models, incorporating VAEs to learn and generate new environments, which are then used to train the agent, thereby improving the efficiency of learning. However, the use of autoencoders in RL for dynamic systems control is an area that has not been thoroughly explored.

This study aims to investigate the application of autoencoders to improve the sample efficiency of reinforcement learning (RL) in controlling dynamic systems. By leveraging autoencoders' ability to efficiently represent complex, high-dimensional states, this research seeks to enhance RL agents' capabilities, enabling them to learn control policies more quickly and robustly with fewer samples and in the presence of observation noise. Additionally, this study aims to overcome the limitations associated with using only the latent space of autoencoders.

This paper structured into four main sections. The initial section covers the background information essential for understanding the field. This is followed by a detailed explanation of the methodology and implementation of simulation experiments. The subsequent section presents the results and discusses their implications. Finally, the paper concludes by summarizing the key findings and suggesting directions for future research.

2 Background Information

In reinforcement learning, agents learn by interacting with the environment and receives feedback in the form of reward. An agent is an entity that makes decisions and takes actions within an environment. In RL, the agent's objective is to identify a course of behavior that maximizes the total reward over time. The agent's operating space is known as the environment. In response to the agent's activities, the environment provides the states, observations, and rewards of the agent.

A state is complete description of the situation or configuration of the environment at a given time and represented as s. An agent's choice to change the environment is referred to as an action an represented as "a". A policy defines an agent's behavior by mapping states to actions and represented as "π".

If the probability distribution of the future states depends solely on the current state and is conditionally independent of the past states, a process is said to have the Markov property. The mathematical model for the agent-environment interaction in RL is called a Markov Decision Process (MDP) if process has Markov Property. In this study the process is assumed as MDP.

The agent's main objective is to maximize cumulative reward which is given in Equation 1.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{1}$$

.

Here G_t is discounted cumulative return, R_t is immediate return at time t and γ is discount factor.

Mathematically, a deterministic policy is represented as in Equation 2

$$\pi(s) = a, \tag{2}$$

Finding an optimal policy π^* that maximizes the expected discounted cumulative reward for each state s is the agent's objective. The formal definition of this is given in Equation 3.

$$\pi^* = \arg\max_{\pi} E[G_t | s_t = s, \pi], \tag{3}$$

In RL, the value (V) and action-value (Q) functions play a critical role in helping the agent estimate the expected future rewards and make decisions. These functions represent the expected return for each state and state-action pair, respectively, and are used to evaluate the quality of the agent's current policy. This functions are given in Equations 4-5.

$$V(s) = E[G_t | s_t = s, \pi] \tag{4}$$

$$Q(s,a) = E[G_t | s_t = s, a_t = a, \pi] \tag{5}$$

The Bellman Equations provide mathematical relationship between the value and action-value functions of a given state or state-action pair and their successors. These equations are used to evaluate and update the value and action-value functions, and given in Equations 6-7.

$$V(s) = \sum_a \pi(a|s) \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right) \tag{6}$$

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s') Q(s',a') \tag{7}$$

Bellman optimality equations for V(s) and Q(s,a) are given in Equations 8-9.

$$V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right) \tag{8}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \tag{9}$$

A family of model-free reinforcement learning algorithms known as Temporal Difference (TD) learning combines the principles of Dynamic Programming with Monte Carlo approaches ([1]). TD learning algorithms update value and action value function estimates using value of next state and observed reward. This enables TD learning algorithms to learn from incomplete episodes and update value estimates online, in contrast Monte Carlo methods, which require complete episodes. TD error is calculated as in Equation 10.

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{10}$$

The V(s) is updated using the TD error as in Equation 11.

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t \tag{11}$$

This update rule is called one step TD as it updates the value function based on one-step look ahead.

Q-learning is a TD control method which action-value function Q(s,a) is iteratively updated similar to updating V(s) with TD error. This refines the policy in response to the new action-value estimates. TD error for Q(s,a) is calculated as in Equation 12.

$$\delta_t = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \tag{12}$$

The Q(s,a) is updated using the TD error as in Equation 13.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\delta_t \tag{13}$$

In Deep Q-Network (DQN) algorithm ([21]), action value Q(s,a) is estimated using deep neutral networks. DQN introduces approaches like experience replay and target networks to reduce instability and divergence problems caused by carelessly merging deep learning and reinforcement learning. The training algorithm for baseline DQN algorithm is given in Algorithm 1.

Algorithm 1: Baseline DQN Training

```
Initialize Replay Memory R
Initialize action-value function Q with random weights $\theta$
Initialize target action-value function $\hat{Q}$ with random weights $\theta^-$
Initialize $\varepsilon$
for $episode = 1, N$ do
    Reset Environment
    while not done do
        With probability $\varepsilon$ select random action $a_t$
        otherwise select action $a_t = argmax_a Q(s_t, a; \theta)$
        Execute action $a_t$ on environment and observe next state $s_{t+1}$ and
          reward $r_t$
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in R
        if time for learn and enough samples available then
            Sample random minibatch $(s_k, a_k, r_k, s_{k+1})$ from R
            Set $y_k = r_k$ if episode terminates at step k+1
            otherwise set $y_k = r_k + \gamma \max_{a'} \hat{Q}(s_{k+1}, a'; \theta^-)$
            Perform gradient descent on $(y_k - Q(s_k; \theta))^2$
            $\theta^- = \tau * \theta + (1 - \tau) * \theta^-$
        end
        Set $s_t = s_{t+1}$
        Decay $\varepsilon = \varepsilon * \delta\varepsilon$
    end
end
```

Autoencoders are used for the unsupervised learning of efficient encodings using neural networks. An autoencoder learns a representation (encoding) of a input data, often to get lower dimensionality or noise. Autoencoders can extract meaningful features from input data to have better representations. Encoder part encodes input to latent space and decoder part rebuilds this input to original dimensions. ([22]). An example of Autoencoder is given in Figure 1.

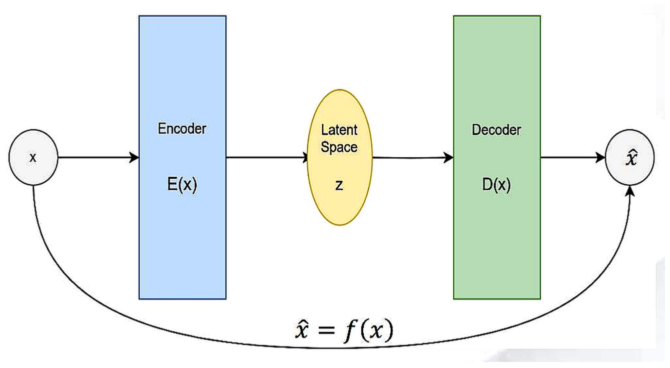

Fig. 1 AutoEncoder

The autoencoder's learning process includes minimizing a loss function, also known as reconstruction error. The most popular form of loss function for autoencoders is mean squared error. This evaluates the distance between the original input and the reconstructed output. This error function is given in Equation 14

$$L(x, \hat{x}) = \|x - \hat{x}\|^2 \tag{14}$$

where $\hat{x}$ is reconstructed version of input x. For recovering input data from a noisy version, denoising autoencoders can be used. The reconstruction loss calculated using noiseless version of x in this type of autoencoders. Also a regularization term can be added to the loss function in Equation 14 which promotes the model to learn a function that is resistant to minor changes in input values. To get the overall loss, the Frobenius norm of the encoder's Jacobian as a contractive loss can be added to reconstruction loss as in Equation 15 ([23])

$$L(x, \hat{x}) = \|x - \hat{x}\|^2 + \lambda\|\nabla_x h(x)\|_F^2 \tag{15}$$

where λ is a hyper-parameter, h is latent layer and x is sampled batch of input.

3 Methodology and Implementation

The Lunar Lander simulation environment, provided by OpenAI's Gym ([24]), serves as the primary experimental setup for this study. This environment simulates a lunar module that is tasked with the objective of landing safely on the moon's surface. The module starts each episode in mid-air and is required to navigate to the landing zone, which is always at zero x and y coordinates.

An 8-dimensional continuous space serves as the environment's state space for the Lunar Lander. It includes the X and Y coordinates of the lander, its velocity in both the X and Y directions, two boolean flags indicating if the lander's left and right legs have touched the ground, the angle and angular velocity. A summary of observation space of Lunar Lander environment is given in Table 1.

Table 1 Observations of LunarLander Environment

Observations	Constraints
x,y coordinates	[-90, 90]
x,y speeds	[-5, 5]
angle	[-3.14, 3.14]
angular velocity	[-5, 5]
two leg contact bool	[0, 1]

The action space, however, is a discrete space with four actions: no action, firing the bottom (main) engine, firing left engine, or firing the right engine. Each action affects the state of the lander consequently, the received reward. A summary of action space of Lunar Lander environment is given in Table 2.

Table 2 Actions of LunarLander Environment

Actions	Action Number
No Action	0
Engine Firing: Left	1
Engine Firing: Main	2
Engine Firing: Right	3

The rewards of the environment is designed to encourage safe and efficient landings. For a successful landing reward is +100 but for crashing penalty is -100. There are also intermediate rewards and penalties for actions such as firing the engines or moving away from the landing pad. A summary of rewards of Lunar Lander environment is given in Table 3.

Table 3 Rewards of LunarLander Environment

Case	Rewards for Each Step
Crashing	-100
Safe Landing	+100
Leg Contact	+10
Side Engine Firing	-0.03
Main Engine Firing	-0.3
x,y coordinates	$-100\sqrt{x^2+y^2}$
x,y speeds	$-100\sqrt{\dot{x}^2+\dot{y}^2}$
angle	$-100\|angle\|$

An autoencoder is used to process the observations from the environment, helping the reinforcement learning model make sense of data. There is no need to interact with the environment to train an autoencoder. It is sufficient to have knowledge of the environment's states and their constraints. By uniformly randomly sampling from these states, a dataset can be created to train autoencoder. The structure of creating dataset for training autoencoder is given in Figure 2.

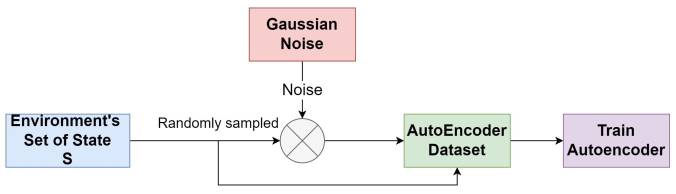

Fig. 2 Creating Dataset To Train AutoEncoder

The design of the autoencoder used in this study is a feed-forward neural network, which contains fully linked layers for both the encoder and decoder. The autoencoder's encoder component is made up of three layers. The input data is transformed into a hidden representation with the dimensionality determined by the hidden dimension parameter in the first layer, which is a linear layer. The activation function that bring non-linearity into the model, the Rectified Linear Unit (ReLU), is then applied. The second layer is another linear layer that further transforms the data, followed by another ReLU activation. A linear layer that maps the data into the latent space makes up the encoder's last layer. The latent dimension parameter determines how many dimensions there are in the latent space.

The decoder part of the autoencoder mirrors the architecture of the encoder but in reverse order. It starts with a linear layer that takes the latent representation and transforms it back to the hidden dimension, followed by a ReLU activation. The second layer has same architecture. The final layer of the decoder is a linear layer that reconstructs the data back to its original dimensionality.

The autoencoder used in this study is also a denoising-contractive mixed autoencoder ([25]). The difference from a conventional autoencoder is, this autoencoder trained using noise corrupted observations and a contractive loss. The reconstruction, encouraging the autoencoder to generate a reconstruction that closely matches the original data. Contrarily, the contractive loss pushes the autoencoder to develop a reliable and consistent representation of the data in the latent space, which reduces its sensitivity to minor changes in the input data. Training algorithm for autoencoder is given in Algorithm 2. The algorithm begins with creating a dataset as shown in Figure 2. To prevent encoder and decoder stuck in local minima, their neural network weights should be initialized randomly. Then using all data from dataset, the reconstruction loss which was given in Equation 15 can be calculated. By performing backpropagation on this loss function, optimal weights of encoder and decoder can be obtained.

Algorithm 2: Denoising-Contractive Autoencoder Training

```
Create training dataset T with randomly sampled and noise added states
Create validation dataset V with randomly sampled and noise added states
Initialize Encoder E with random weights $\Theta$
Initialize Decoder D with random weights $\Phi$
for $t$ = 1, N do
    Sample train batch noiseless $x_t$ and noisy $\tilde{x}_t$ from T
    Calculate contractive loss $\lambda\|\nabla_{x_t}\mathbf{h}(x_t)\|_F^2$ where h is hidden layer of E
    Calculate reconstruted sample $\hat{x}_t = D(E(\tilde{x}_t;\Theta);\Phi)$
    Calculate reconstruction loss $L(x_t,\hat{x}_t)$
    Perform gradient descent on total loss $L(x_t,\hat{x}_t)+\lambda\|\nabla_{x_t}\mathbf{h}(x_t)\|_F^2$ Sample
     validation batch and calculate total loss in similar way if minimum
     validation loss don't change for 1000 steps then
        Early stop
    end
end
```

The RL model used in this study is a Deep Q-Network (DQN) agent. The methodology also involves the use of a replay buffer for storing and sampling experiences, which allows the model to learn from past experiences and improve its performance over time. Trajectories are stored in memory and randomly sampled in batches when needed. A structural comparison of the proposed and baseline reinforcement learning algorithms is provided in Figure 3. The baseline reinforcement learning control algorithms use states directly, whereas the proposed algorithms augment these states with latent spaces obtained by encoding the states. The training algorithm for the baseline DQN is presented in Algorithm 1, while the training algorithm for the proposed technique is detailed in Algorithm 3. The difference between Algorithm 3 and Algorithm 1 is in Algorithm 3 there is an encoder which was pretrained using Algorithm 2. This pretrained encoder is used to find latent states of the system and then given as an extra input to a baseline DRL algorithm alongside with normal states.

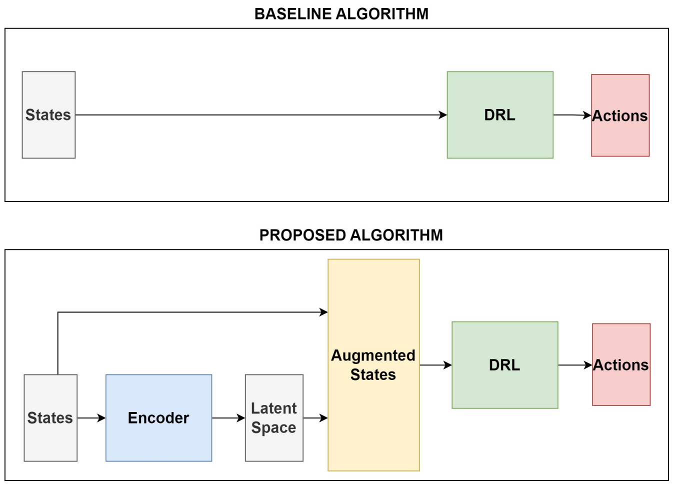

Fig. 3 Baseline and Proposed Controllers

```
Algorithm 3: Encoder Augmented DQN Training
  Pretrain Encoder E
  Initialize Replay Memory R
  Initialize action-value function Q with random weights $\theta$
  Initialize target action-value function $\hat{Q}$ with random weights $\theta^{-}$
  Initialize $\varepsilon$
  for episode = 1, N do
      Reset Environment
      while not done do
          With probability $\varepsilon$ select random action $a_t$
          Find latent states $z_t = E(s_t)$
          otherwise select action $a_t = argmax_a Q(s_t, z_t, a; \theta)$
          Execute action $a_t$ on environment and observe next state $s_{t+1}$ and
            reward $r_t$
          Store transition $(s_t, a_t, r_t, s_{t+1})$ in R
          if time for learn and enough samples available then
              Sample random minibatch $(s_k, a_k, r_k, s_{k+1})$ from R
              Find latent states $z_k = E(s_k)$ and $z_{k+1} = E(s_{k+1})$
              Set $y_k = r_k$ if episode terminates at step k+1
              otherwise set $y_k = r_k + \gamma \max_{a'} \hat{Q}(s_{k+1}, z_{k+1}, a'; \theta^{-})$
              Perform gradient descent on $(y_k - Q(s_k, z_k; \theta))^2$
              $\theta^{-} = \tau * \theta + (1 - \tau) * \theta^{-}$
          end
          Set $s_t = s_{t+1}$
          Decay $\varepsilon = \varepsilon * \delta\varepsilon$
      end
  end
```

4 Results and Discussion

The goal of this study's experimental setting was to investigate the potential of combining a denoising-contractive autoencoder with a DQN within the Lunar Lander simulation environment. The autoencoder, designed with three layers for both encoding and decoding, featured a hidden layer dimension of 64 and a variable latent layer dimension for comparison. Implemented as a denoising contractive autoencoder—known for its efficacy in learning robust representations—the settings included a contractive lambda of 10^{-4}, a learning rate of 10^{-3}, and ReLU as the activation function. The training batch size was set to 64, with a Gaussian noise of zero mean and 0.01 standard deviation added to the observations for the denoising process.

In the reinforcement learning framework, the DQN was configured with a discount factor of 0.99 to control the weighting of potential rewards. The soft update tau, controlling the rate at which the target network is updated, was set to 10^{-3}. The DQN's batch size and learning rate were also set to 64 and $5x10^{-4}$, respectively.

The loss function is a numerical measure that reflects the norm between the batch of states and the reconstructed batch of states from the autoencoder. A lower loss number indicates that the autoencoder is better capturing the essential features of the input data. The results showed a steady decrease in the loss value over the training epochs, suggesting that the autoencoder was effectively learning the representations of the environment's state. For different latent dimensions, autoencoder training and validation losses are given in Figure 4 - 6.

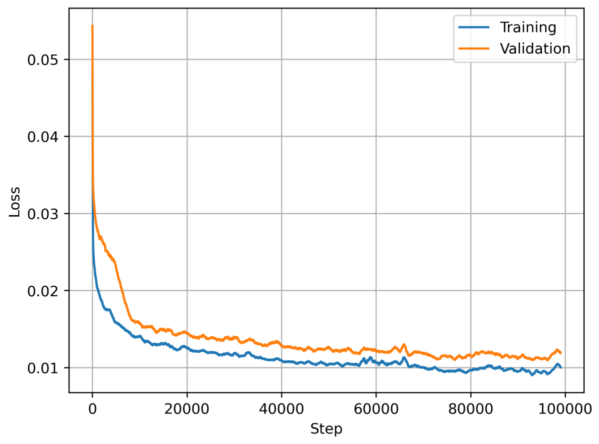

Fig. 4 AutoEncoder Losses for Latent Dim 2

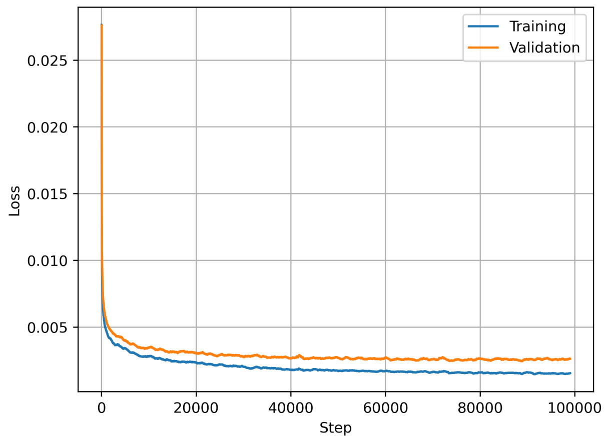

Fig. 5 AutoEncoder Losses for Latent Dim 4

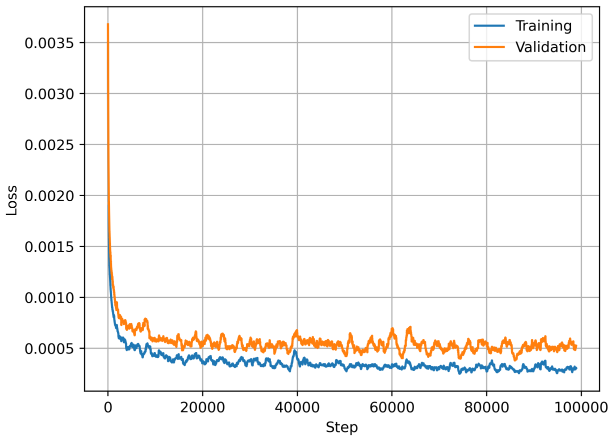

Fig. 6 AutoEncoder Losses for Latent Dim 12

The experiments were conducted over a series of episodes, with variations in the size of the latent space and the seed, each episode representing a complete game run in the Lunar Lander environment. The cumulative reward earned for each episode and the difference between the target action-value and the expected action-value in DQN were used to assess the model's performance. The cumulative reward is a numerical measure that reflects the success of the agent in achieving the game's objective, which is to land the lunar module safely. The training losses of agent are given in Figure 7 - 9 and the moving average rewards of episodes are given in Figures 10 - 12.

For both algorithms, in the initial episodes, the model's performance, as indicated by the cumulative reward, was relatively low. This initial low performance is expected, as the model begins with no prior knowledge about the environment and learns through trial and error. However, as the number of episodes increased, we observed a significant improvement in the model's performance. The cumulative reward per episode showed an increasing trend, indicating that the model was learning from its experiences and improving its strategy over time.

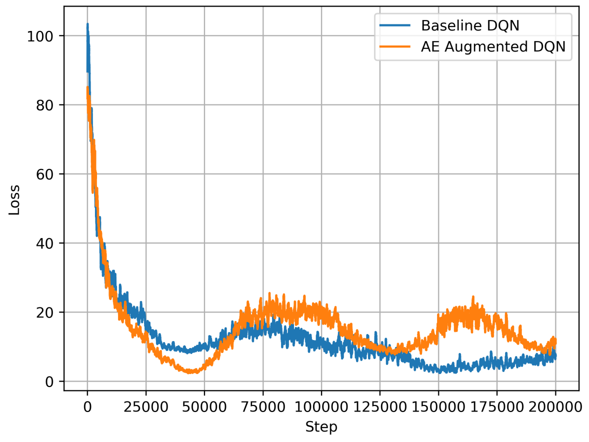

Fig. 7 Training Losses of Baseline DQN and AE DQN with Latent Dim 2

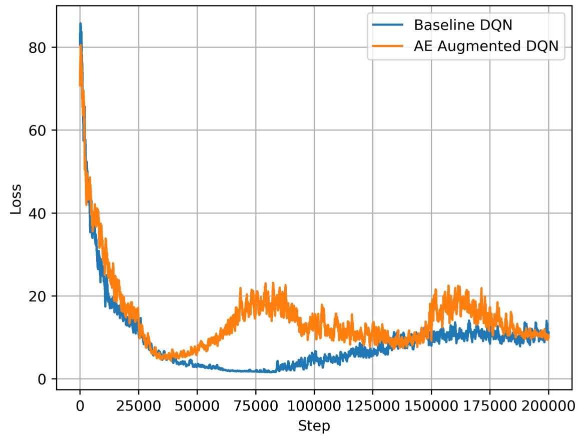

Fig. 8 Training Losses of Baseline DQN and AE DQN with Latent Dim 4

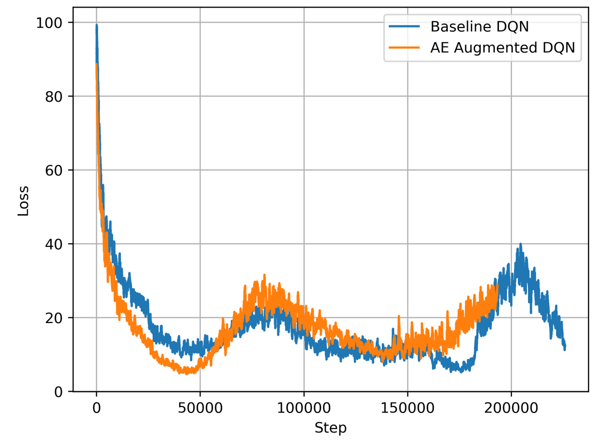

Fig. 9 Training Losses of Baseline DQN and AE DQN with Latent Dim 12

As can be seen from the Figures 7 - 9, the training losses of the proposed algorithm have decreased more rapidly in first 50000 step than those of the baseline algorithm for a latent dimension of 12. However, for some latent dimensions, they remain higher. Also it can be observed that their performances are similar at the and of steps. So proposed algorithm has advantage at the early steps. Optimizing the number of latent dimensions or designing an early step criteria could yield better performance for proposed algorithm.

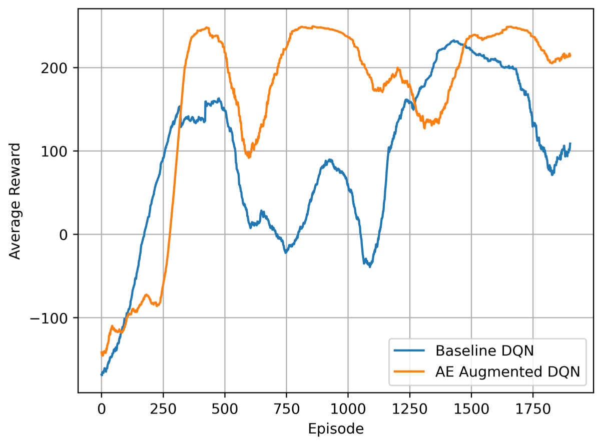

Fig. 10 Rewards of Baseline DQN and AE DQN with Latent Dim 2

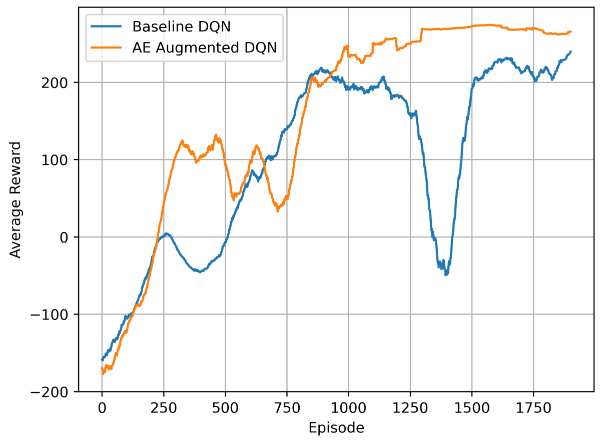

Fig. 11 Rewards of Baseline DQN and AE DQN with Latent Dim 4

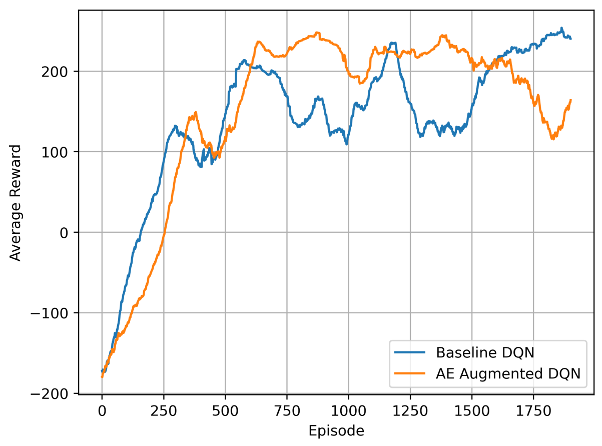

Fig. 12 Rewards of Baseline DQN and AE DQN with Latent Dim 12

The results in Figures 10 - 12 showed that, in terms of landing accuracy, stability, and total score, the suggested model outperformed the baseline DQN model in most applications. The suggested model's cumulative rewards reach high levels faster with fewer samples (episodes), and generally have higher and more stable rewards than the baseline model. This indicates that the inclusion of the autoencoder for state representation learning improved the model's performance. The autoencoder was able to effectively extract important features from the encoding of states to latent space, which seemed to facilitate the model's decision-making process. Performance varies with different latent space sizes, suggesting that this can be a critical design parameter.

5 Conclusion

The pursuit of this research project began with the objective to investigate the capabilities of an autoencoder-augmented DQN model in the context of enhancing sample efficiency. The literature typically focuses on using only the latent space in algorithms, which has several limitations such as information loss, dependence on the quality of the autoencoder, and difficulty in handling dynamic environments due to outdated representations. In this study, a novel algorithm is proposed, where states augmented with the latent space from their encoding are utilized in deep reinforcement learning (DRL) algorithms. This approach aims to overcome these limitations and enhance sample efficiency. A denoising-contractive autoencoder was designed to encode states for this purpose. The results showed that the proposed algorithm outperformed baseline algorithms by achieving higher rewards with fewer episodes and obtaining more stable rewards. Since proposed algorithm has advantage at the early steps and the number of latent dimensions have effect on performance, optimizing the number of latent dimensions and designing an early step criteria can be future research opportunities.

References

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

[2] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] G. Dulac-Arnold, D. Mankowitz, and T. Hester, *Challenges of real-world reinforcement learning*, 2019. arXiv: `1904.12901 [cs.LG]`.

[5] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[6] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *Advances in neural information processing systems*, vol. 31, 2018.

[7] R. Agarwal, C. Liang, D. Schuurmans, and M. Norouzi, "Learning to generalize from sparse and underspecified rewards," in *International conference on machine learning*, PMLR, 2019, pp. 130–140.

[8] T. Nguyen, T. M. Luu, T. Vu, and C. D. Yoo, "Sample-efficient reinforcement learning representation learning with curiosity contrastive forward dynamics model," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 3471–3477.

[9] N. Botteghi, M. Poel, and C. Brune, *Unsupervised representation learning in deep reinforcement learning: A review*, 2022. arXiv: `2208.14226 [cs.LG]`.

[10] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*, Springer, 2006, pp. 282–293.

[11] P.-Y. Oudeyer and F. Kaplan, "What is intrinsic motivation? a typology of computational approaches," *Frontiers in neurorobotics*, vol. 1, p. 108, 2007.

[12] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[13] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, *et al.*, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[14] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*, PMLR, 2017, pp. 1126–1135.

[15] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," *arXiv preprint arXiv:1803.02999*, 2018.

[16] T. Hester, M. Vecerik, O. Pietquin, *et al.*, "Deep q-learning from demonstrations," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[17] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.

[18] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *Advances in neural information processing systems*, vol. 33, pp. 19 884–19 895, 2020.

[19] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller, "Embed to control: A locally linear latent dynamics model for control from raw images," *Advances in neural information processing systems*, vol. 28, 2015.

[20] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, *Playing atari with deep reinforcement learning*, 2013. arXiv: `1312.5602 [cs.LG]`.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[23] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, “Contractive auto-encoders: Explicit invariance during feature extraction,” in *Proceedings of the 28th international conference on international conference on machine learning*, 2011, pp. 833–840.

[24] O. Gym, *Openai gym*, `https://www.gymlibrary.dev`.

[25] F.-q. Chen, Y. Wu, G.-d. Zhao, J.-m. Zhang, M. Zhu, and J. Bai, *Contractive de-noising auto-encoder*, 2014. arXiv: `1305.4076 [cs.LG]`.