Information Sciences 537 (2020) 467–484

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Towards safe reinforcement-learning in industrial grid-warehousing

Per-Arne Andersen *, Morten Goodwin, Ole-Christoffer Granmo

Department of ICT, University of Agder, Jon Lilletuns vei 9, 4879 Grimstad, Norway

ARTICLE INFO

Article history:
Received 27 August 2019
Received in revised form 31 May 2020
Accepted 2 June 2020
Available online 21 June 2020

Keywords:
Model-based reinforcement learning
Neural networks
Variational autoencoder
Markov decision processes
Exploration
Safe reinforcement learning

ABSTRACT

Reinforcement learning has shown to be profoundly successful at learning optimal policies for simulated environments using distributed training with extensive compute capacity. Model-free reinforcement learning uses the notion of trial and error, where the error is a vital part of learning the agent to behave optimally. In mission-critical, real-world environments, there is little tolerance for failure and can cause damaging effects on humans and equipment. In these environments, current state-of-the-art reinforcement learning approaches are not sufficient to learn optimal control policies safely.

On the other hand, model-based reinforcement learning tries to encode environment transition dynamics into a predictive model. The transition dynamics describes the mapping from one state to another, conditioned on an action. If this model is accurate enough, the predictive model is sufficient to train agents for optimal behavior in real environments.

This paper presents the *Dreaming Variational Autoencoder* (DVAE) for safely learning good policies with a significantly lower risk of catastrophes occurring during training. The algorithm combines variational autoencoders, risk-directed exploration, and curiosity to train deep-q networks inside "dream" states. We introduce a novel environment, ASRS-Lab, for research in the safe learning of autonomous vehicles in grid-based warehousing. The work shows that the proposed algorithm has better sample efficiency with similar performance to novel model-free deep reinforcement learning algorithms while maintaining safety during training.

1. Introduction

Reinforcement learning has recently demonstrated a high potential to learn efficient strategies in environments where there are noisy or incomplete data [16]. We find these achievements in many domains, such as robotics [40], wireless networking [39], and game-playing [33,37]. The common denominator between these domains is that they can be computer-simulated with significant resemblance to real-world environments. For this reason, reinforcement learning algorithms train at accelerated rates without compromising the safety of real-world systems [17].

The goal of reinforcement learning algorithms is to learn a policy (or behavior) that stimulates optimal actions based on sensory input and feedback from an environment. A policy is a parameterized model that is constructed in (exact) tabular form or using an (approximation) neural network with algorithms such as gradient descent [36]. The algorithm performs an iterative process of (sampling) exploration, exploitation, and (learning) policy updates that move the policy toward

* Corresponding author.
E-mail addresses: per.andersen@uia.no, per@sysx.no (P.-A. Andersen).

https://doi.org/10.1016/j.ins.2020.06.010
0020-0255/

the desired behavior. Exploration is commonly performed using a separate policy, such as a (random sampling) Gaussian distribution. It is crucial that the algorithm balance exploration and exploitation with schemes such as ϵ-greedy so that the policy updated towards a generalization of the whole environment.

The problems of guaranteed safety during reinforcement learning are many. (1) It requires a tremendous amount of sampling to learn a good policy. (2) Stable and safe policies are challenging to achieve in non-deterministic and even deterministic, for fast-changing environments. (3) Conventional model-free exploration methods are not safe in mission-critical environments. (4) Reinforcement learning methods depend on negative feedback to avoid catastrophic states and hence, should be avoided for mission-critical systems. Most reinforcement learning techniques are not designed for safe learning, and therefore, few solutions exist for mission-critical real-world environments [18].

Automated Storage and Retrieval Systems (ASRS) are a modern method of performing warehouse logistics where the system is partially or fully automated [22]. In industry, including ASRS, it is common to rely on complex expert systems to perform tasks such as control, storage, retrieval, and scheduling. If-else statements and traditional pathfinding algorithms drive these tasks. The benefit of expert systems is that it is trivial to model operative safety bounds that limit the system from entering catastrophic states. The downside is that expert systems do not automatically adapt to changes and require extensive testing if the environment is modified. While it may be possible and perhaps trivial to construct safe routines with an expert system, it is inconceivable to expect optimal behavior due to the complexity of most real-world environments [29]. Reinforcement learning is perhaps the most promising approach to solve these problems because it can generalize well across many domains [33], and is designed to work in noisy environments with partial state-space visibility [32].

We propose *The Dreaming Variational Autoencoder* (DVAE), an algorithm for safer learning in real-world environments. DVAE is an improved version of previous work in Ref. [4] that emphasize on more reliable learning in mission-critical environments. The algorithm tries to address the problems (1, 2, 3, 4) that concern safe and sample efficient reinforcement learning. The algorithm does not require direct access to the real-world environment or prior knowledge to learn a stable policy. It is, however, possible to inject prior knowledge of catastrophic states to strengthen safer learning.

The key contributions of this paper are summarized as follows:

- DVAE, a predictive model for n-state predictions,
- safety constraints using a constrained MDP scheme,
- safe exploration through risk-directed exploration and curiosity,
- ASRS-Lab for industry near testing of the proposed approach,
- and analysis of empirical results.

The organization of the paper follows. Section 2 outlines progress in the field, including automated storage and retrieval systems, model-based reinforcement learning, and safe reinforcement learning. Section 3 presents the theoretical background of the proposed algorithm. Section 4 details the DVAE algorithm thoroughly and discuss the convergence guarantee for the algorithm. Section 5 introduces *The ASRS-Lab*, an industry-near learning environment that simulates real-world ASRS systems. The results are presented in Section 6 and show that the algorithm act safer than model-free reinforcement learning. The paper is finally summarized in Section 7 and proposes future work in safe reinforcement learning.

2. Related work

Recently, advancements in reinforcement learning have more frequent and included substantial performance improvements in numerous domains [6]. Many aspects play a role, but notwithstanding increased publicity, which attracts new institutions to work with reinforcement learning. This section presents work that relates to reinforcement learning in industry-near environments and research that attempts to address safety in these domains.

2.1. Reinforcement learning

Reinforcement learning is applied previously in industry-near environments, and perhaps the most widespread application is autonomous vehicles. The proposed method in this paper uses an auxiliary policy to label data for supervised training. With only 12 h of labeled data, Ref. [10] illustrates learning performant policies using a direct perception approach with convolutional neural networks. This approach is much like a variational autoencoder that simplifies the perception of the world significantly. This simplification of the input significantly speeds up inference, which enables the system to issue control commands more frequently. There are many other significant contributions to autonomous vehicle control, such as [30]. An in-depth survey on the topic of autonomous vehicle control that directly relates to ASRS environments are discussed thoroughly in [38].

2.2. Model-based RL

In model-based reinforcement learning, the goal is to learn state-transitions based on observations from the environment, the predictive model. If the predictive model is stable, with low variance and improves monotonically during training, it is, to some degree, possible to learn model-free agents to act optimally in environments that have never been observed directly.

Perhaps the most sophisticated algorithm for model-based reinforcement learning is the Model-based policy optimization (MBPO) algorithm, proposed by Janner et al. [27] The authors empirically show that MBPO performs significantly better in continuous control tasks compared to previous methods. MBPO proves to be monotonically improving, given that the following bounds hold:

$$\eta[\pi] \geqslant \hat{\eta}[\pi] - C$$

where $\eta[\pi]$ denotes the returns in the real environment under a policy whereas $\hat{\eta}[\pi]$ denotes the returns in the predicted model under policy π. Furthermore, the authors show that as long as they can improve the C, the performance will increase monotonically [27].

DARLA is an architecture for modeling the environment using β-VAE. The trained model was used to learn the optimal policy of the environment using algorithms such as DQN, A3C, and Episodic Control. DARLA is, to the best of our knowledge, the first algorithm to introduce learning without access to the ground-truth environment during training [26].

Chua et al. proposed *Probabilistic Ensembles with Trajectory Sampling* (PETS). The algorithm uses an ensemble of bootstrap neural networks to learn a dynamics model of the environment over future states. The algorithm then uses this model to predict the best action for future states. The authors show that the algorithm significantly lowers sampling requirements for the half-cheetah environment compared to SAC and PPO. [12].

Deep Planning Network (PlaNet) is a model-based agent that interprets the pixels of a state to learn a predictive model of an environment. The environment dynamics stores into a latent-space, where the agent sample actions based on the learned representation. The proposed algorithm showed significantly better sample efficiency compared to model-free algorithms such as A3C [23].

The Dreaming Variational Autoencoder (DVAE) is an end-to-end solution for predicting the probable future state $p(\hat{s}_{t+1}|s_t, a_t)$. The authors showed that the algorithm successfully predicted the next state in non-continuous environments and could, with some error, predict future states in continuous state-space environments such as the Deep Line Wars environment. In the experiments, the authors used DQN, PPO, and TRPO using an artificial buffer to feed states to the algorithms. In all cases, the DVAE algorithm was able to create buffers that were accurate enough to learn a near-optimal policy. [4].

In *Recurrent World Models Facilitate Policy Evolution*, a novel architecture for training RL algorithms using variational autoencoders. This paper showed that agents could successfully learn the environment dynamics and use this as an exploration technique requiring no interaction with the target domain. The architecture is mainly three components; vision, controller, and model, the vision model is a variational autoencoder that outputs a latent-space variable of an observation. The latent-space variable is processed in the model and feeds into the controller for action decisions. Their algorithms show state-of-the-art performance in self-supervised generative modeling for reinforcement learning agents [21].

Neural Differential Information Gain Optimization (NDIGO) algorithm by Azar et al. is a self-supervised exploration model that learns a world model representation from noisy data. The primary features of NDIGO are its robustness to noise due to their method to cancel out negative loss and to give positive learning more value. In their maze environment, the authors show that the model successfully converges towards an optimal world model even when introducing noise. The author claims that the algorithm outperforms the state-of-the-art, such as Recurrent World Models [7].

Gregor et al. proposed a scheme to train expressive generative models to learn belief-states of complex 3D environments with little prior knowledge. Their method effectively predicted multiple steps into the future (overshooting) and significantly improved sample efficiency. In the experiments, the authors illustrated model-free policy training in several environments, including DeepMind Lab. However, the authors found it difficult to use their predictive model in model-free agents directly [20].

For further details on model-based RL, we refer the reader to Ref. [5].

2.3. Safe reinforcement learning

A majority of established systems in the industry have an expert system that already acts as the controller for the environment. In real-world environments, the need for safe and stable learning is critical so that existing routines are not interrupted.

Similar to the proposed algorithm, [9] assumes a predictive model that learns the dynamics of the environment. The authors propose that the policy should be limited to a safe-zone, called the **R**egion **O**f **A**ttraction (ROA). Everything within the bounds of the ROA is "safe states" that the policy can visit, and during training, the ROA gradually expands by carefully exploring unknown states. The algorithm shrinks the ROA to ensure stability if the feedback indicates movement towards catastrophic states.

The proposed algorithm encodes the observations as latent embeddings using a variational autoencoder (VAE) similar to the View model in [21]. In the world model approach, the authors define three components. The (VAE) *view* encodes observations to a compact latent embedding. The *model* (MDM-RNN)[1] is the predictive model used to learn the (predictive model) world model. Finally, the (C) *controller* is a general framework that enables model-free algorithms to interact with the world model.

[1] Mixture Density Network combined with a Recurrent Neural Networks.

3. Background

The optimization problem is modeled as **M**arkov **D**ecision **P**rocesses (MDP). The MDP consists of the tuple (S, A, R, P, γ) where S is the set of possible states, A is the set of possible actions, $R : S \times A \times S \to \mathbb{R}$ is the reward function, $P : S \times A \times S \to [0, 1]$ is the transition probability function (where $P(s' \mid s, a)$ denotes the probability of transitioning to next state s' given that the agent takes action a in state s), and $\gamma \in [0, 1]$ is the discount factor for future rewards.

A policy (π) is a parameterized model that maps together (input) observations and (output) actions to form behavior. **The goal**of the reinforcement learning agent is to select actions in a way that maximizes future rewards.

$$G_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} \tag{1}$$

Eq. 1 denotes the discounted cumulative future rewards, often referred to as the *discounted return* in literature [36]. We assume that if the policy adjusts its parameters to find actions towards maximizing the return, the policy will ultimately converge optimally.

$$\pi^* = \arg\max_{\pi} V^{\pi}(s) \quad \forall s \in S \tag{2}$$

Eq. (2) denotes the optimal policy and is a policy that yields the highest attainable state-value $V^{\pi}(s)$ for all states while under the control of the policy π. The state-value function is denoted

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t | S_t = s] = \mathbb{E}_{\pi}[R_{t+1} + \gamma V^{\pi}(S_{t+1}) | S_t = s], \tag{3}$$

and quantifies how good it is for an agent to be in a particular state. Furthermore, the state-action function indicates how good it is for the agent to take any possible action being in state s, where

$$V^*(s) = \max_{a \in A} Q^*(s, a) \quad \forall s \in S \tag{4}$$

describes the relationship between the state-value and state-action function [33]. As long as the agent selects actions that maximize the Q-values, the state-value is also optimal [36]. Therefore,

$$\pi^* = \arg\max_{a \in A} Q^*(s) \quad \forall s \in S \tag{5}$$

the optimal policy is found when the agent always makes actions that maximize the Q-value.

Traditional RL learns the optimal policy according to an *optimization criterion*. This optimization criterion varies with different algorithms but is commonly implemented to minimize time or to maximize reward. The *return maximization criterion* is frequently used in Q-Learning, where

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a \in A} Q^{\pi}(s_{t+1}, a) - Q^{\pi}(s_t, a_t) \right] \tag{6}$$

backpropagates the Q-estimate of the following state to the former state.[2]

It becomes evident that there is **no safety guarantee** in the traditional view of reinforcement learning [9]. The primary focus is for the agent to find the policy that maximizes some feedback signal, and through dynamic programming, monte-carlo methods, or temporal-difference, find a way to learn by trial and error. For mission-critical environments, reinforcement learning is insufficient, and therefore, we seek a method to learn good policies while reducing the number of catastrophic states.

Fig. 1 illustrates a stochastic MDP in the view of a traditional RL agent (left) and an agent that is safety-aware (right). The MDP considers state-space $S = \{s_0 \dots s_9\}$ and an action-space $A = \{a_0 \dots a_2\}$ controlled with the probabilistic policy $\pi(a \mid s)$, with the probability of transitioning to the next state $P(s' \mid s, a)$ (stochastic transition). The traditional model-free RL agent must explore to learn a policy that would keep a distance from catastrophic states. This means that the agent would eventually take action a_0 in state s_0 and enter state s_1, which leads to a catastrophic outcome. The motivation for a safer learning system becomes evident, and the idea is to find a method to define good (green) and bad (red) state-space regions before the agent starts exploration.

3.1. Safe policy selection

Risk is a function that indicates the danger of making an action under the policy $\pi(a \mid s)$ [25]. It is founded on the uncertainty associated with future events and is inevitable since the consequences of actions are unknown when an action is made. There are numerous definitions of the term risk, namely *Risk-Sensitive Criterion*, *Worst Case Criterion*, and *Constrained Criterion* [31]. A policy that disregards risk evaluation is *risk-neutral*, and the learning objective is to maximize the expectation of the return,

[2] The equation illustrates the Q-Learning algorithm without any extensions and without deep learning considerations.

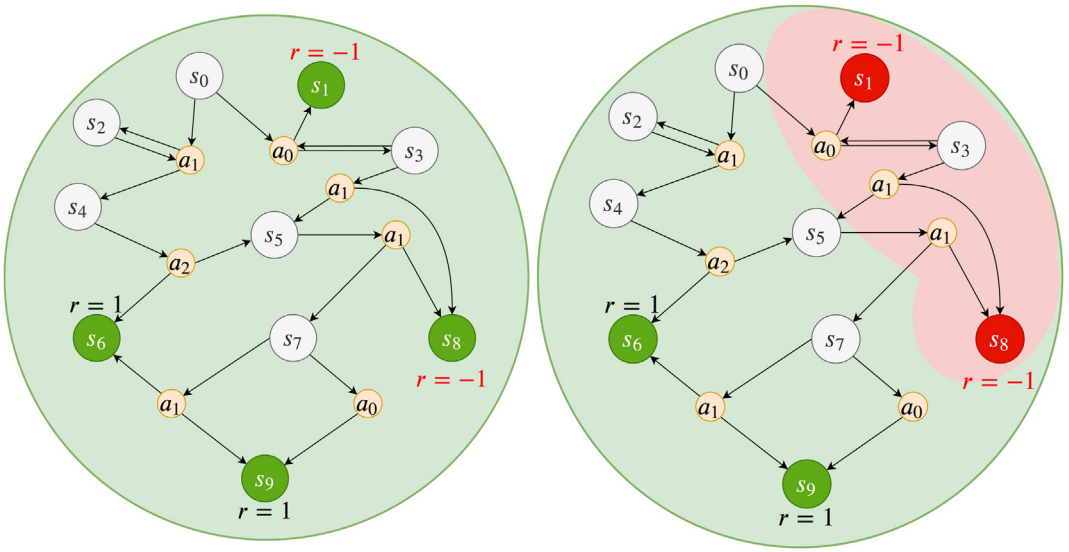


Fig. 1. Illustration of an MDP where actions are made according to a policy π. The colors are in the view of the policy where green is *safety*, red is danger, gray is non-terminal states, and orange is actions in a stochastic environment. On the left, the policy follows traditional RL optimization where trial and error occurs in order to map recognize bad states. On the right, the policy has some notion of danger (red area) for actions leading to states with negative feedback. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$\max_{\pi\in\Pi}\mathbb{E}_\pi(G) = \max_{\pi\in\Pi}\mathbb{E}_\pi\left(\sum_{t'=t}^{T}\gamma^{t'-t}r_{t'}\right) \tag{7}$$

where it becomes apparent that Eq. (7) is the same objective as Eq. (1). This gives motivation for modification of the objective function so that the policy is *risk-aware* when maximizing the return.

The **Constrained Criterion** is an appealing approach as it extends the standard MDP framework described as the tuple (S, A, R, P, γ, C), where C is a set of constraints applied to the policy. The goal of the constraint set is to, with high probability, eliminate policies such as the *unsafe* example in Fig. 1, similar to the work in Ref. [9]. The general form of the constrained criterion is defined,

$$\max_{\pi\in\Pi}\mathbb{E}_\pi(G)\ \textit{subjectto}\ c_i \in C, c_i = \left\{h_i \gtrless \alpha_i\right\} \tag{8}$$

where c_i is the i_{th} constraint in the set C that must be satisfied by the policy π. Additionally, h_i is a function related to the return G that is an upper or lower bound to the threshold value α_i. Consider all constraints satisfied, then the policy-space is reduced to a subset $\Gamma \subseteq \Pi$, and the policy exists only within this subset $\pi \in \Gamma$. The idea is that the constraints lead to a significantly smaller policy-space, where it is more likely that a safe solution is found, seen in Fig. 2. Given that the algorithm selects policies only from the *safe* subset Γ, the objective function can be written as

$$\max_{\pi\in\Gamma}E_\pi(G) \tag{9}$$

which is the standard notion of expected return from Eq. (1), but with respect to the subset of safe policies Γ.

Constraint Selection is a delicate user-defined process, which largely depends on a specific problem [19]. It is possible to form constraints from any metric originating from the MDP. Our approach attempts to use a general approach for safe policy updates across various domains. In the proposed algorithm, only a single constraint is formed using the error (uncertainty) of a predictive model [21]. The α parameter acts as a threshold for how much risk we allow when evaluating a policy. Higher the value, the constraint is more restrictive, and for lower values, more permissive [11].

3.2. Safe exploration

Policies with a constrained criterion do not guarantee safety in the short term because it is challenging to choose parameters within the subset of safe policies Γ initially. Therefore, we also consider **safer exploration** as a means to guide the agent towards making safe actions in the short term.

Risk-directed Exploration uses a notion risk to determine in which direction the agent should explore. We refer the reader to [13] for an in-depth definition. There are several ways to define risk, such as keeping below a variance threshold, but our approach uses *normalized expected return with weighted sum entropy* [13].

The risk of taking action in a particular state is given by

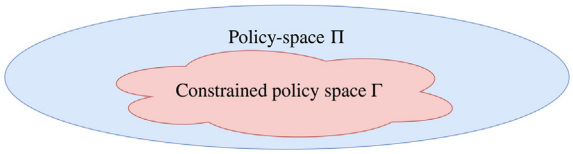


Fig. 2. The policy-space (blue) Π and the subset of policies (red) $\Gamma \subseteq \Pi$, where each policy $\pi \in \Gamma$ must satisfy the constraints $c_i \in C$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$Risk(s,a) = \mathscr{R}(s,a) = wH(s,a) - (1-w)\frac{\mathbb{E}[G]}{\max_{a \in A} |\mathbb{E}[G]|}, \tag{10}$$

where H is the policy entropy, and the second term describes how good the action is for the particular state. The weight $w \in [0 \dots 1]$ determines the balance between entropy and return, where higher values of w indicate *more risk* due to less deterministic behavior. The utility function is then updated as follows

$$Utility_{risk}(s,a) = U_{ri}(s,a) = \alpha \mathscr{R}(s,a) + (1-\alpha)\pi(a \mid s). \tag{11}$$

where $\alpha \in [0 \dots 1]$ controls the risk-awareness of the agent. At $\alpha = 0$, the agent does not perform risk-directed exploration but considers safety more as $\alpha \to 1$. The risk function $\mathscr{R}(s,a)$ outputs a vector describing the risk of each action in the action space. The risk vector adds to the probabilities and state-action values for the current state ($\pi(a \mid s)$). The updated utility function $U_{ri}(s,a)$ ensures that sampling is performed in favor of safe and conservative (less exploratory) actions, depending on the weight parameter w, and risk aversion parameter α [13].

3.3. Safe predictive model

In model-based reinforcement learning, the goal is to efficiently learn a predictive model that accurately learns the environment dynamics to predict future states given the current state and action [23]. During the learning of a predictive model, explorative agents are frequently used. However, in a real-world environment where catastrophic states exist, it is little room for errors. However, these environments are often eligible for the deployment of expert systems. Therefore, it is possible to collect observations, with a sub-optimal agent for a user-specified amount of time. The collected observations significantly increase the accuracy of the predictive model, enabling the use of the concept of *curiosity* [34] to create constraints to increase safety. Curiosity-driven exploration is composed of two rewards, extrinsic (the environment) and intrinsic (curiosity) reward, where the agent is encouraged to enter unexplored states. For this work, we **negate** this effect and encourage the agent to stay in states where the predictive model has low uncertainty. For each evaluation using the predictive model, we can calculate the error, which is the difference between the predicted state and the actual state (the state observed by the agent). For predicted states with high error, the model knows little about the consequences of doing the action, indicating that the action will lead to a catastrophic state. Curiosity is the mean squared error of the predicted future state features $\widehat{\mathscr{M}}(\hat{s_{t+1}}|s_t, a_t)$ and the ground truth future state $\mathscr{M}(s_{t+1}|s_t, a_t; P)$ where

$$C_u\left(\mathscr{M}, \widehat{\mathscr{M}}\right) = \frac{1}{2} \mid\mid \widehat{\mathscr{M}}(\hat{s_{t+1}}|s_t, a_t) - \mathscr{M}(s_{t+1}|s_t, a_t; P) \mid\mid_2^2 \tag{12}$$

defines the curiosity vector. In curiosity-driven exploration, the goal is to pursue states that maximize curiosity, but, we aim to minimize C_u for actions with high uncertainty for safe exploration. In our approach, the weighted curiosity vector adds to the action probability distribution such that

$$U(s,a) = U_{ri+Cu}(s,a) = U_{ri} + \alpha C_u \tag{13}$$

where α is the risk-aversion parameter previously defined in Eq. (11).

The updated utility is then compatible with Q-Learning updates using neural network function approximator with weight θ ad the Q-Network. The network is trained by sequentially minimizing the loss function $L_i(\theta_i)$ where i denotes the iteration, such that,

$$L_i(\theta_i) = \mathbb{E}_{s,a \sim p(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right], \tag{14}$$

where $y_i = \mathbb{E}_{s' \sim \mathscr{E}}[U + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a]$ is the target for iteration i and $p(s,a)$ is the behavior distribution [33]. Finally, the standard differentiated loss, w.r.t to the weights θ denoted,

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim p(\cdot); s' \sim \mathscr{E}}\left[\left(U + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s,a;\theta_i)\right) \nabla_{\theta_i} Q(s,a;\theta_i)\right], \tag{15}$$

where U is the modified reward from Eq. (13).

4. Safe dreaming

This paper aims to increase the safety of agents that act in environments with catastrophic state outcomes. The *Dreaming Variational Autoencoder* (DVAE) is a model-based reinforcement learning approach for safe and efficient learning. A predictive model learns the dynamics of the environment and acts as a safety precaution as the agent learn fully offline from the real environment. Additionally, the algorithm models the problem as a constrained MDP with a combination of risk-directed exploration and negated curiosity. The algorithm performs learning three steps, *predictive model learning*, *RL training*, and *transition to the real environment*, and describes as follows.

A predictive model learns the transition dynamics of the real environment. To learn these transitions, the model gathers experience through observation of an expert system. An expert system quite regularly in use already; hence, minimal effort is required to train the predictive model. The expert system already makes safe decisions because of hand-crafted features, but it often operates at a sub-optimal performance. Therefore, a reinforcement learning algorithm is well-suited for decision-making in industry-near environments, as it can improve performance and safety with a predictive model's learning guidance.

Training model-free RL algorithms using the learned predictive model is safe and efficient in terms of sampling efficiency. Deep Q-Learning from Ref. [33] is well suited because it converges with off-policy data. A combination of the DVAE algorithm and model-free reinforcement learning ensures that learning is performed safely without the risk of entering catastrophic states or cause damage to the real-world environment.

Training duration depends on the problem, and should, therefore, rely on some mechanic to determine the learning stopping criteria. As the algorithm learns an optimal policy for the predictive model, it gradually transitions to make actions in the real environment based on the rate it enters catastrophic states. At such a time, when the agent interacts directly with the real environment, it is possible to enter catastrophic states. The algorithm adds a negated curiosity bonus to reduce the exploration of state-space regions with high uncertainty. This way, the fully deployed algorithm will behave cautiously when the movement towards novel states appears, or if the environment is changed dynamically.

The training procedure illustrated in Fig. 3 works as follows. (1) The predictive model observes and learns the real environment using a sensor model. The same sensor model is the interface that the expert system uses for making actions. (2) The intelligent agent (i.e., a reinforcement learning agent) interact with the predictive model and improve its policy. (3) When the intelligent agent is sufficiently trained, it can replace existing expert systems with comparable performance. (4) If desirable, the intelligent agent can train further in the real-world environment.

The execution graph of DVAE is shown in Fig. 4 and works as follows. The policy $\pi(a \mid s)$ predicts the best action a for the observed state s. The first action is sent to the real environment to produce initial state $s_t = s_0$. The initial state s_t and initial action a_t is processed by the predictive model M and outputs predicted future state $\hat{s}_{t+1}$ and reward $\hat{r}_{t+1}$. The reward is used for policy updates during training and the state for further action prediction. The policy predicts a future state $\hat{s}_{t+1}$ and is sent to the predictive model, now to predict the two-step predicted state $\hat{s}_{t+2}$. The procedure continues until the algorithm meets the stopping criteria.

A detailed illustration of the DVAE-architecture is shown in Fig. 5, including the encoder, decoder, policy, and environment. Initially, the interaction is between the predictive model M and the policy $\pi(s \mid a)$. The encoder takes a state s_t and predicted action a_t as input and outputs the embedding z_t. An embedding is a compression of the input and leads to faster training and better performance. Considering that the replay-buffer RB holds millions of embedding, the memory footprint is significantly reduced. The replay-buffer generates sequential batches of embeddings that are input to the t-encoder enc_t.

The s-encoder is responsible for transforming raw input data into a meaningful and compact feature embedding. DVAE uses a variational autoencoder primarily for this task, but other methods are also applicable, such as generative adversarial networks (GAN). It is possible to visualize the embedding $z_x \in \mathbb{Z}$ by manually altering its values depending on the environ-

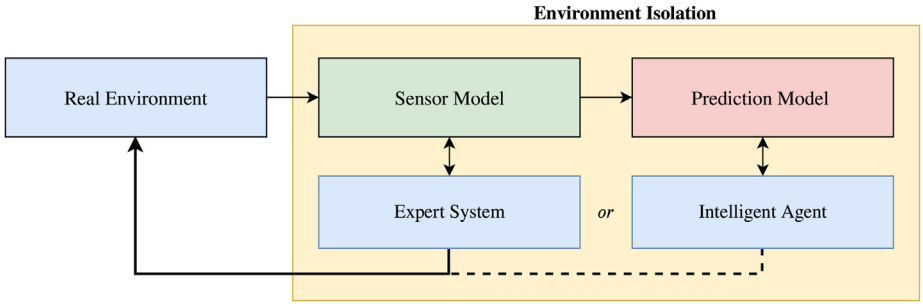


Fig. 3. Isolation of the real environment. The general idea of DVAE is to isolate the agent training to reduce the risk of catastrophic behavior in the real environment. The predictive model observes the sensors of the real environment and estimates its transition function. The intelligent agent uses the predictive model to train in an offline setting, without the risk of making mistakes in the real world. After training, the algorithm is deployed to the real environment, with significantly less chance of entering catastrophic states. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

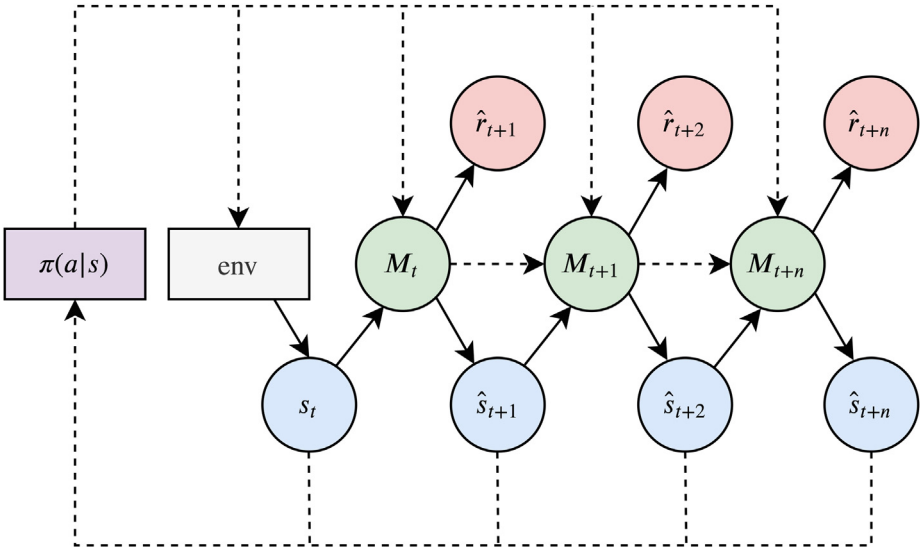


Fig. 4. Prediction of future states. For each timestep, the agent observes a state from the environment or the predictive model. The agent makes an action that results in a transition to the next state with the corresponding reward. M denotes the predictive model where $\hat{s}$ and $\hat{r}$ is the predicted state and reward.

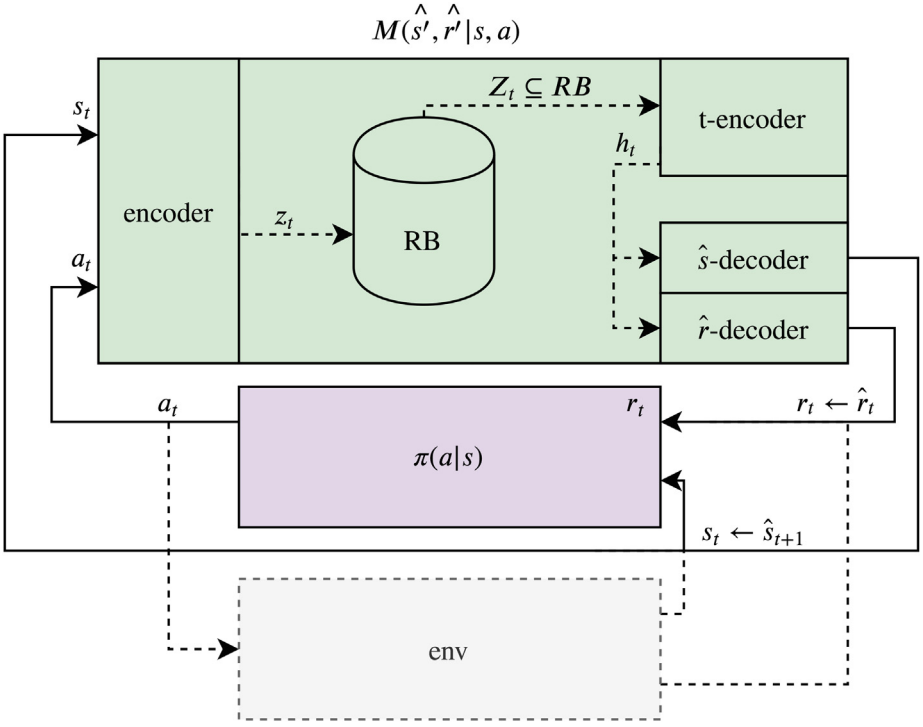
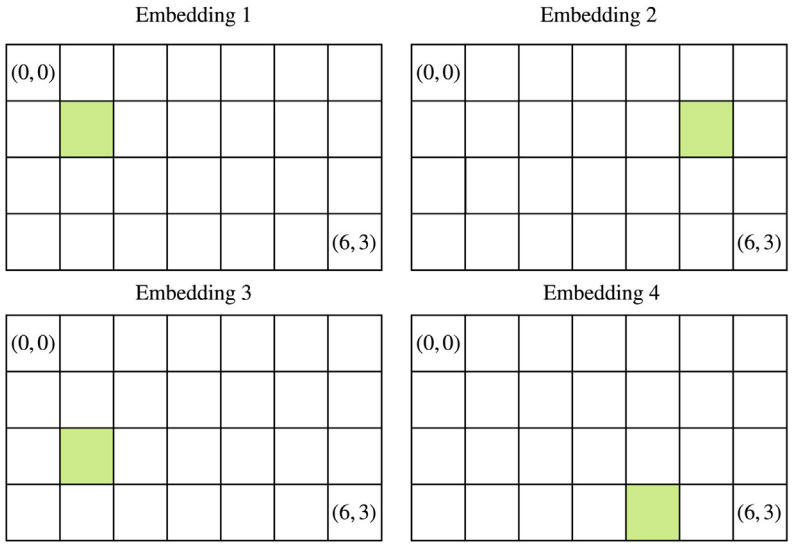


Fig. 5. Architecture overview of DVAE.The architecture of the proposed predictive model. The predictive model M takes action a and state s as input to the encoder. The input to the encoder is transformed into the embedding z_t and is stored in the replay-buffer RB. The t-encoder (temporal-encoder) retrieve $Z_t \subseteq RB$ (size determined by hyper-parameter) to learn the transition dynamics h_t w.r.t time. The $\hat{s} - decoder$ and $\hat{r} - decoder$ decode h_t into a predicted future state and reward, which feeds into the policy for decision making and training. The dotted lines illustrate the standard reinforcement learning interaction between agent and environment [36], which the algorithm uses after training.

ment and the input data. In Fig. 6, we illustrate this with a (green) agent in an empty grid-world. The embedding layer consists of two neurons where the first and second neuron learns the vertical and horizontal location, respectively.

The t-encoder learns the time dependency between states, or in MDP terms, the transition function $T : S \times A \rightarrow S$. The t-encoder model computes the future state embedding z^{π}_{t+1} based on a batch of previous embeddings from the view



Embedding	z_0	z_1
1	0.285	0.285
2	0.857	0.285
3	0.285	0.750
4	0.714	1.000

Fig. 6. Learned features of the encoder. The figure illustrates a two-parameter embedding of a grid-world environment where the learned embedding refers to the player's location. The idea is that the state information, an $n \times m$ grid, is compressed significantly and can be retrieved by decoding the embedding.

$Z_t^{\pi} = \{z_{t-n} \dots z_t\}^{\pi}$. The π denotes the policy which DVAE operates under. In DVAE, *long short-term memory* (LSTM) performed best when learning the future state embedding.

The control policy $\pi(s \mid a)$ is responsible for interaction with the environment and the predictive model (s-encoder and t-encoder). The control is the primary model for making actions that are safe and progress the learning in the right direction. In DVAE, we consider Deep Q-Networks (DQN) using a **constrained optimization criterion** and for exploration, **risk-directed exploration**, and **negated curiosity**. The negated curiosity act as the **constrained criterion** for the MDP. The input to the algorithm is a raw-state, commonly a high-dimensional data structure that is difficult to interpret spatial information. The benefit of the DVAE architecture is that the t-model finds an embedding that can represent the state with the order of magnitudes less complexity. The DVAE algorithm also enables initial training fully offline in a *dream* version of the real environment.

Algorithm 1. DVAE with Deep Q-Learning

1: Initialize policy $\pi_\theta(s_t|a_t)$
2: Initialize predictive model $M_\psi(\hat{s}', \hat{r}' \mid s, a_\pi)$
3: Initialize encoder $enc(z_t|s_t, a_t)$
4: Initialize replay-buffer $RB(Z_t \mid \{z_t \dots z_{t+n}\})$
5: Initialize t-encoder $enc_t(h_t|Z_t)$,
$\hat{s}$-decoder $dec_s(\hat{s}_t|h_t)$,
$\hat{r}$-decoder $dec_r(\hat{r}_t|h_t)$
Training of the predictive model
6: predictive model needs training; episode = 1, E do
7: Make decisions using predefined expert system policy
8: Store transition (s_t, a_t, r_t, s_{t+1}) in buffer D
9: Train predictive model M_ψ on data batch $d \subseteq D$ using *MLE* loss
Training of the Deep Q-Network (or similar RL algorithm)
10: for episode = 1, E do
11: Sample initial state s_0 from D
12: Predict action using policy $\pi(a \mid s; \theta)$
13: Predict future state using the predictive model M_ψ where,
14: Encode input state and action to embedding $enc(z_t \mid s_t, a_t)$
15: Store z_t in RB and form sequential subset of n-elements $Z_t \subseteq RB$
16: Encode sequence of embeddings w.r.t time $enc_t(h_t \mid Z_t)$
17: Decode future state and reward $enc_{s+r}(\hat{s}', \hat{r}' \mid h_t)$
18: Update policy π_θ with pairs of $(\hat{s}_t, a_t, \hat{r}_t, \hat{s}_{t+1})^{\pi_\theta}$ according to Eq. (15)

The definition of the DVAE algorithm, seen in Algorithm 1, has the following procedure. **(Line 1–5)** The π_θ, predictive model M_ψ with corresponding encoder $enc(z_t|s_t, a_t)$, replay-buffer $RB(Z_t \mid \{z_t \dots z_{t+n}\})$ t-encoder $enc_t(h_t|Z_t), \hat{s}$-decoder $dec_s(\hat{s}_t|h_t)$ where each component is a function approximator, is initialized with random weights.

(Line 6) starts the training procedure of the predictive model for E episodes. **(Line 7–8)** A expert system algorithm makes decisions and is recorded into buffer D. This step is primarily for observation of the environment to learn the sensor model transition dynamics.**(Line 9)** initiates training of the predictive model using mini-batch stochastic gradient descent with *MLE*-loss.

(Line 10) initiates a for loop of E episodes to train the reinforcement learning algorithm using the learned predictive model from the procedure at line numbers 6–9. **Line 11–12** is similar to the standard reinforcement learning loop, but instead of taking actions in the real environment, the decision is sent to the predictive model M_ψ. **(Line 13)** the predictive model outputs the estimated future state from the agent decision with the following steps. **(Line 14)** encode current state s_t and action a_t into the embedding z_t. **(Line 15)** The embedding is stored in the replay buffer and is retrieved in batches of n-elements to form Z_t. **(Line 16)** encode the sequential batch of embeddings to capture transition dynamics between states, yielding h_t. **(Line 17)** The $dec_{\hat{s}+\hat{r}}$ decoder outputs the predicted future state and the corresponding reward. Finally, using the predicted values, the reinforcement learning policy is updated using Eq. (15) **(Line 18)**, similar to Ref. [33]. The process is repeated separately for the predictive model and the reinforcement learning algorithm until reaching an acceptable convergence threshold.

4.1. Exploration and policy update constraints

There are significant improvements to the exploration and policy update for finding safe policies in DVAE. During sampling, the policy uses a *risk-directed exploration* bonus [13]. The bonus is added to the probability distribution over actions before sampling is performed, as described in Section 3. The **policy updates** are constrained to a set of criteria defined as follows. During the learning of the predictive model, feedback is received from the real-world environment. All actions are bound to some feedback even though only 1 of these are received depending on which action the agent performed. In our model, we assume that all actions that were not chosen by the agent are considered unsafe. This way, the algorithm gradually maps the unsafe policy space, as illustrated in Fig. 1. It is important to note that this mapping does not influence the agent's choices when learning the predictive model. When the agent revisits a state, the agent may select another action, which will label the state safe. Depending on how much the expert system behavior is observed, the better understanding the predictive model gets, as well as the state-risk mapping of the state-space.

4.2. Analysis of convergence guarantees

The Dreaming Variational Autoencoder combines several approaches that previous work has shown to have convergence properties. The algorithm model the problem as an MDP, which is proven to have convergence properties in several works [15]. The Markov property is especially interesting, and the proof is detailed well in Ref. [24]. The DVAEalgorithm use *constrained MDP* and is proved to have convergence properties for the discounted case used in Ref. [1].

Tabular Q-Learning is known to converge as time goes towards T, but deep learning variants, specifically neural network estimated Deep Q-Networks, primarily have empirical success. There are efforts such as Ref. [14] that prove theoretical convergence for simplified DQN, but no proof for the general case. In regards to using neural network estimators for the predictive model, the proposed approach is based primarily on empirical observations. DVAE uses a similar approach to Refs. [21,23] where the predictive model encoder constructs a variational bound on the data log-likelihood:

$$lnM_d(s_1 : T) \triangleq ln \int \prod_{t=1}^{T} M(s_t \mid s_{t-d}) M(s_t \mid s_t^o) ds_{1:T} \geqslant \sum_{t=1}^{T} \underbrace{\left(\mathbb{E}_{q(s_t|s_t^o)}\right)\left[lnM(s_t^o|s_t)\right]}_{\text{reconstruction}} - \underbrace{M(s_{t-1} \mid s_{t-d}) q(s_{t-d}| \mid s^o \leqslant t-d) \mathbb{E}\left[KL\left[q(s_t \mid s_t^o \leqslant t) \| M(s_t \mid s_{t-1})\right]\right)}_{\text{multi-step prediction}} \tag{16}$$

where s^o denote unprocessed states. We refer the reader to Ref. [23] for the derivation. The curiosity bonus used in the proposed algorithm is shown to work well empirically, but there is no proof of convergence to the best of our knowledge. Through trial and error, the proposed algorithm converges empirically, but theoretical convergence remains future work.

5. The ASRS Lab

Safety during RL learning has been a less prevalent priority in recent years compared to improving the performance of existing non-safe algorithms. We argue that this may be due to the high cost of physical systems to experiment on and that RL research is primarily tested in games, which naturally encourages to maximize the agent performance by trial and error. In this section, we propose the **A**utomated **S**torage and **R**etrieval **S**ystem **L**ab (ASRS-Lab), a flexible and industry-near envi-

ronment for reinforcement learning research. The ASRS-Lab environment focuses on how RL algorithms can learn good policies with minimal negative feedback and enable auxiliary policies to create a predictive model that can be used for safe offline training of reinforcement learning algorithms.

5.1. Motivation

It is well known that the training of algorithms in real-world environments is complicated for several quintessential reasons, which causes non-deterministic side-effects. **First**, in real-world environments, there is no option to accelerate the sampling speed to increase training speed since the training speed depends on real-world time. **Second**, reinforcement learning builds on trial and error, which is not applicable for many mission-critical systems as an error can have catastrophic consequences. **Third**, in real-world environments, there are additional uncertainty factors that can alter the state-space. Most RL algorithms can adapt sufficiently to slight changes, but with the risk of policy collapse for drastic changes. **Fourth**, a system which in simulation is deterministic will because of the side-effects mentioned above, in a real-world environment becomes stochastic. All of these factors cause challenges to guarantee **safety** during training in real-world environments.

5.2. Implementation

With safe reinforcement learning in mind, the ASRS-Lab is implemented with flexible options for state, action, and reward-representations. There are many categories of ASRS systems in the real world, and to build an environment flexible

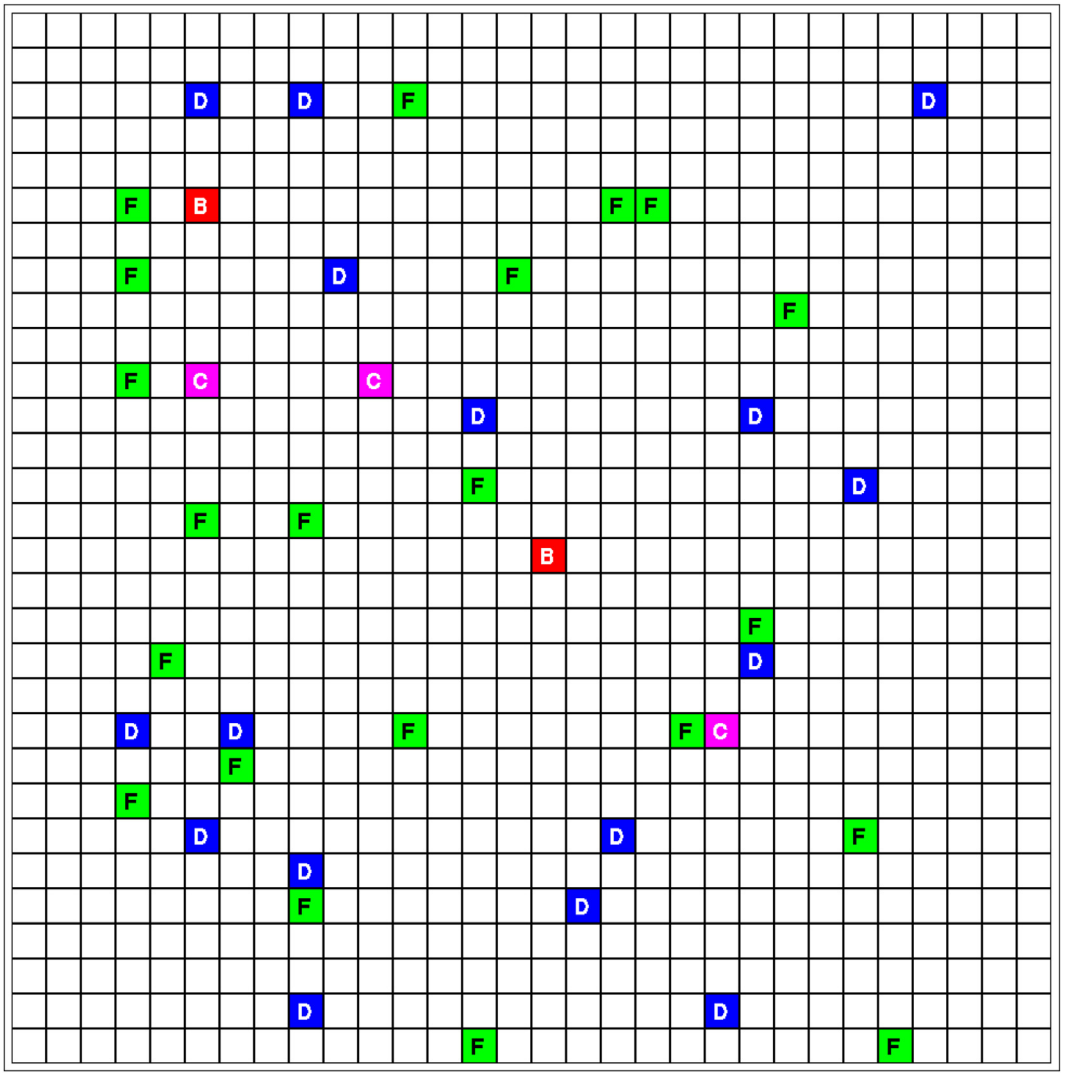
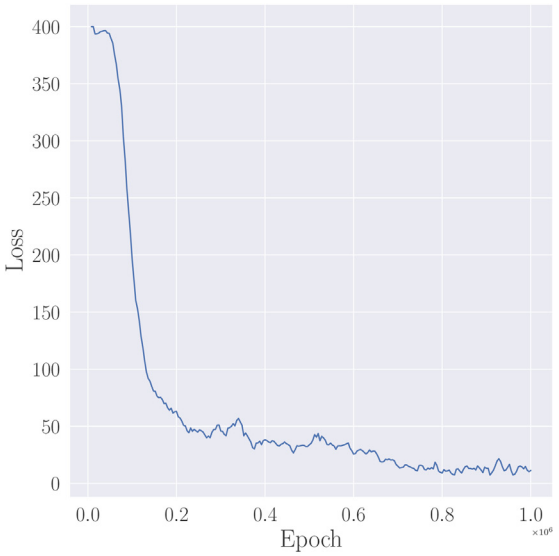
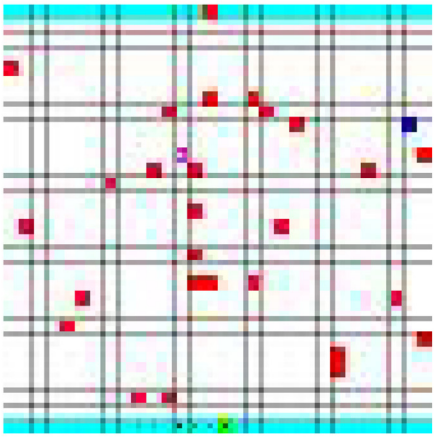


Fig. 7. Visual observation of the ASRS-Lab environment using cube-based ASRS configuration.



(a) **Average predictive model M loss.**



(b) **Decoder output after training.**

Fig. 8. The MSE loss between real and predicted state for ASRS-Lab is illustrated on the left. The x-axis describes the epoch number ranging from 1 to 1,000,000. The right figure shows the decoder output of the predictive model after 1,000,000 epochs of training.

enough to accommodate all requirements for all systems was unfeasible. However, the ASRS-Lab could successfully reconstruct shuttle-based, aisle-based, and grid-based warehouses. For this paper, we consider the grid-based architecture.

Fig. 7 illustrates the observable state-space from a two-dimensional point of view. In a simple cube-based ASRS system, the environment consists of (B) passive and (C) active delivery-points, (D) pickup-points, and (F) taxis.**The goal** of the environment is to find a positive terminal state using minimal time with a limited set of actions. One episode of the environment is defined as follows. The (taxi) agent starts at an arbitrary position on the plane. At the same time, the agent receives a *retrieve order* from the ASRS *scheduling system*. This order describes a target location for goods to be retrieved. The agent must now reach the target location in minimal time using its controls. Considering that there are many other agents on the plane, the control task is challenging to learn because each action has a significant risk of collision with other agents, as well as the outer bounds of the grid system. When an agent enters a target position, it is rewarded and is assigned a *delivery task* from the scheduling system. The agent must now move to the designated location described by the delivery task. When the agent reaches its destination, a large reward is given.

A taxi can move using a **discrete** or a **continuous controller**. In the discrete mode, the agent can increase and decrease thrust and move in either direction, including the diagonals. For the continuous mode, all of these actions are floating-point numbers between (off) 0 and (on) 1, giving a significantly harder action-space to learn. The simulator also features a continuous mode for the state-space, where actions are performed asynchronously to the game loop. The environment supports custom modules for mechanisms such as the scheduling system, agent controllers, and fitness scoring.

5.3. Benefits

A notable benefit of the ASRS-Lab is that it can accurately model real-world warehouse environments at high speed. Compared to real-world systems, the ASRS-Lab environment runs an order of magnitudes faster on a single high-end processing unit. The performance is measured by comparing the number of actions a taxi performs in the real environment versus the virtual environment. The environment can be distributed on many processing units to increase the performance further. In our benchmarks, the simulator was able to collect 1 million samples per second during the training of deep learning models using high-performance computing (HPC).

6. Results

In state-of-the-art model-free reinforcement learning algorithms, it is common to perform a (random) Gaussian-based exploration method to map the return to states. These algorithms are excellent at finding an average point on the optimization plane that generalizes well across multiple domains. The issue, however, is that there are no guarantees that the learned policy avoids catastrophic states. In this section, we show that DVAE is capable of learning an accurate predictive model for model-free algorithms and learn good policies while behaving safely during exploration. We apply the proposed constrained criterion to the policy updates and use risk-directed exploration to enforce safer actions as described in Section 3. DVAE inte-

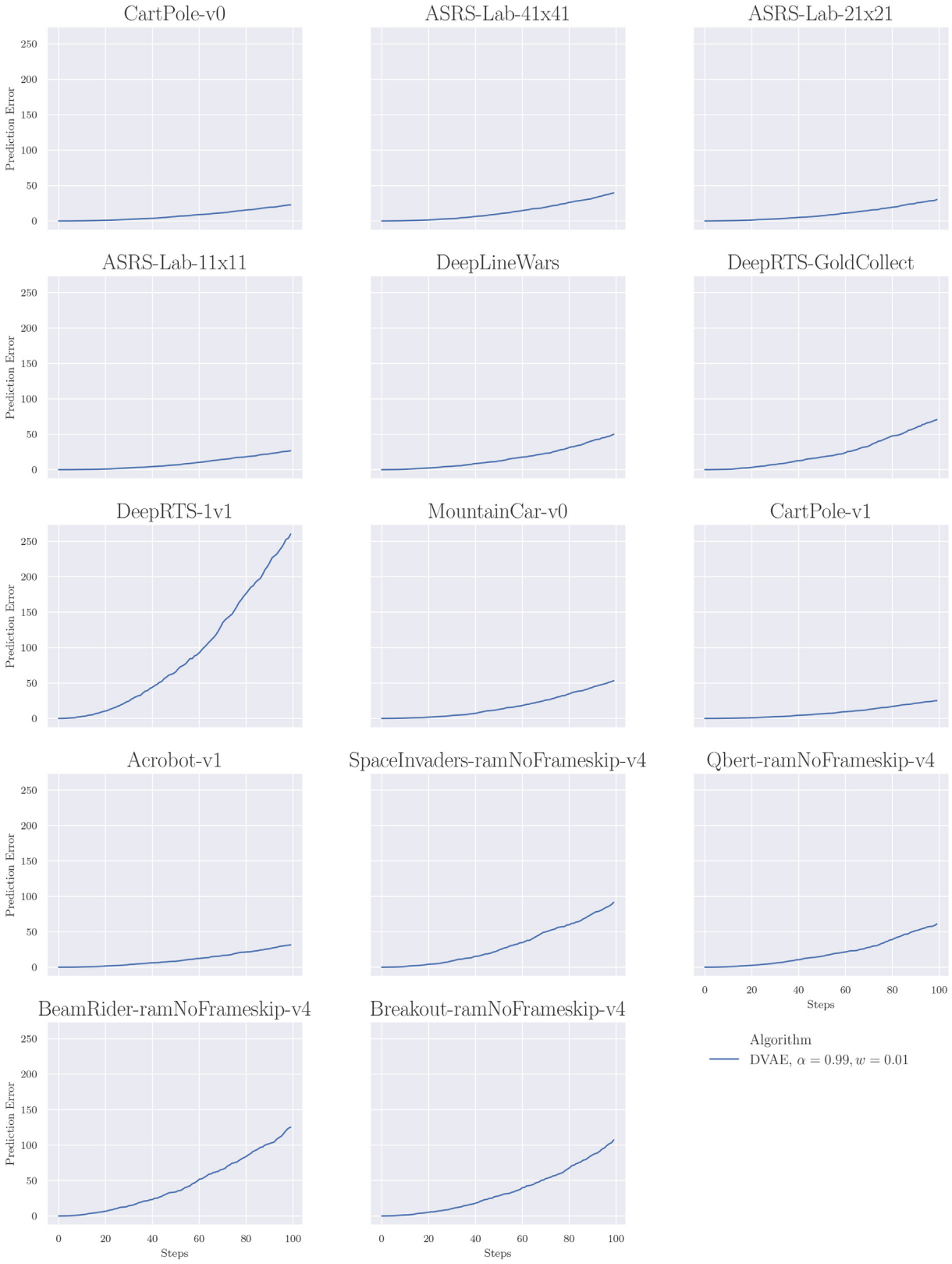


Fig. 9. Cumulative Prediction Error The y-axis shows the pixel error where each whole number represents a 2-dimensional error. For example, an error of 32 means that 32×32 pixels have incorrect values. The x-axis is how many predictions in the future is made without interaction with the real environment (how many states in the future has the algorithm "dreamed").

Table 1
Exponential cumulative prediction error. Depending on the environment, the cumulative prediction error increase exponentially for all environments. The table shows that the exponential growth is consistently less extreme for simple environments. The numbers in the header present the state n-th in the future.

Environment	10	25	50	75	100
ASRS-Lab-11x11	0.34	2.08	7.47	14.91	25.10
ASRS-Lab-21x21	0.36	1.91	7.53	16.62	28.75
ASRS-Lab-41x41	0.43	2.98	10.50	25.06	43.02
Acrobot-v1	0.42	2.04	9.15	19.56	34.86
BeamRider-ramNoFrameskip-v4	1.77	9.33	33.99	75.94	135.49
Breakout-ramNoFrameskip-v4	1.52	7.20	28.84	67.37	110.48
CartPole-v0	0.31	1.52	6.13	13.62	22.58
CartPole-v1	0.35	1.50	6.66	14.53	26.01
DeepLineWars	0.54	3.81	13.73	29.41	50.58
DeepRTS-1v1	2.72	16.29	65.98	143.02	255.99
DeepRTS-GoldCollect	0.69	4.55	19.75	46.17	78.00
MountainCar-v0	0.65	3.66	15.31	30.73	49.95
Qbert-ramNoFrameskip-v4	0.77	4.20	18.23	38.69	63.38
SpaceInvaders-ramNoFrameskip-v4	1.04	4.98	21.96	53.11	89.28

grates well with **Deep Q-Network** (DQN) and compared against DQN (Rainbow) and **Proximal Policy Optimization** (PPO). The algorithm tests across various environments, including popular Atari 2600 games [8], Deep RTS [2], Deep Line Wars [3], and the industry-near environment, ASRS-Lab [5].

6.1. Predictive model

The prediction model's objective is to learn about environment dynamics and features so that it can accurately mimic the environment's behavior. Fig. 8a illustrates the average loss for all tested environments. The predictive model trains using specially crafted **expert systems**that perform well in each of the tested environments. The trend is for the loss to start high, and quickly reduce to only minor weight adjustments during training. These minor weight adjustments play a significant role in learning an accurate embedding, as illustrated by Fig. 8b.

A way to measure the accuracy of the predictive model is to investigate the cumulative prediction error. Fig. 9 illustrates this cumulative prediction error for all tested environments. The experiments show that the prediction error tends towards exponential growth when the predictive model makes predictions for longer time horizons. As seen in Table 1, the predictive model has an error of 284 (the decoded state is 284×284) for the Deep RTS environment at predictions done for 100 time-steps in the future. This means that every pixel in the predicted state is incorrect, and hence, difficult to use for training model-free algorithms.

It is sensible to limit the prediction horizon for environments that are too advanced or difficult to extract the dynamics from. The downside of limiting the prediction horizon is that the algorithm is not able to train fully offline. However, the algorithm reduces the volume of **real** training data needed to converge model-free approaches by magnitudes successfully.

The predictive model successfully learns several environments sufficiently, including ASRS-Lab-21x21, CartPole, and Deep Line Wars. It is likely that tuning the α, w, and learning-rate would improve accuracy for other environments, but parameters remain problem-specific and must be **carefully tuned**.

6.2. Agent failure rate

The failure rate is measured by counting the number of negative rewards the agent receives during an episode while training. The environment has a negative reward for catastrophic states and positive on the contrary. Recall that the algorithm should interpret the MDP with constraints and label catastrophic states accordingly, see Fig. 1.

Fig. 10 illustrates the failure-rate for DVAE with three hyper parameter configurations, $\alpha = 0.99,\ w = 0.01,\ \alpha = 0.7,\ w = 0.3,\ \alpha = 0.5,\ w = 0.5$. Recall that higher α and lower w values account for safe-aware behavior. Safer configuration of DVAE clearly impacts the rate by which the algorithm makes mistakes.

The algorithm does not always learn good policies, such as in the DeepRTS environment. The reason is perhaps that the reward function does not represent the goal, and further investigations discovered that this is the case for DeepRTS. For the DeepRTSGold environment, DVAE outperformed PPO and DQN significantly.

DVAE increases safety significantly for the majority of the environments tested in this paper. The results from Fig. 10 shows a consistent decrease in failures when increasing the safety-awareness sensitivity using the α and w hyper-parameter. The benefits of having high safety-awareness increase action safety, but at the cost of **slower convergence** or local minima problems.

6.3. Agent performance

DVAE has comparable performance to DQN and PPO in terms of accumulating reward during training. Fig. 11 shows the performance **after** the DVAE algorithm is pre-trained on the predictive model. We perform these tests on DeepLineWars,

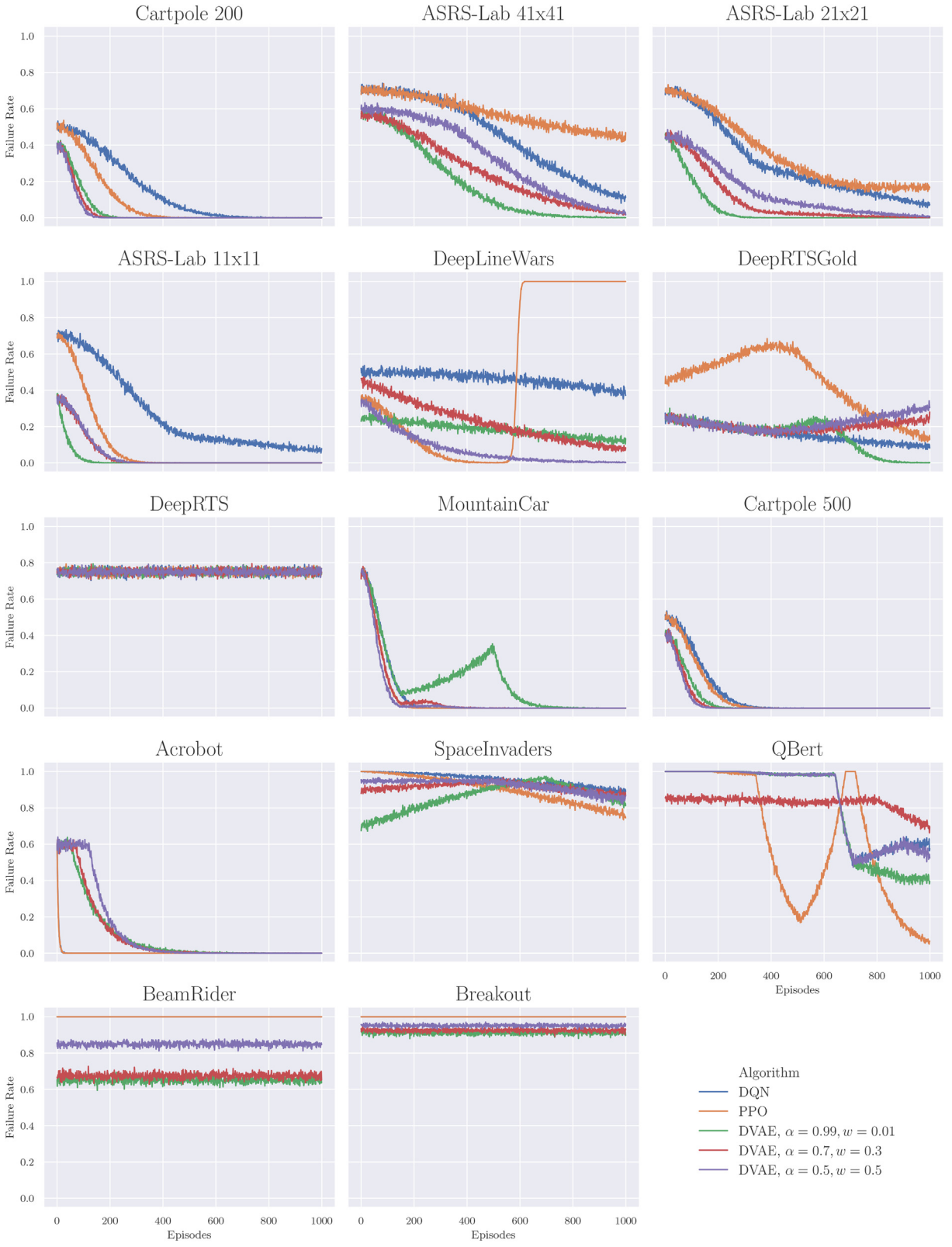


Fig. 10. Agent failure rate. We evaluate the rate of which an agent fails during trials across various environments where the x-axis illustrates the episode number, and the y-axis the rate in percentage. Each environment is averaged over 100 trials for 1000 episodes. We compare three safety configurations of DVAE against DQN [33] and PPO.

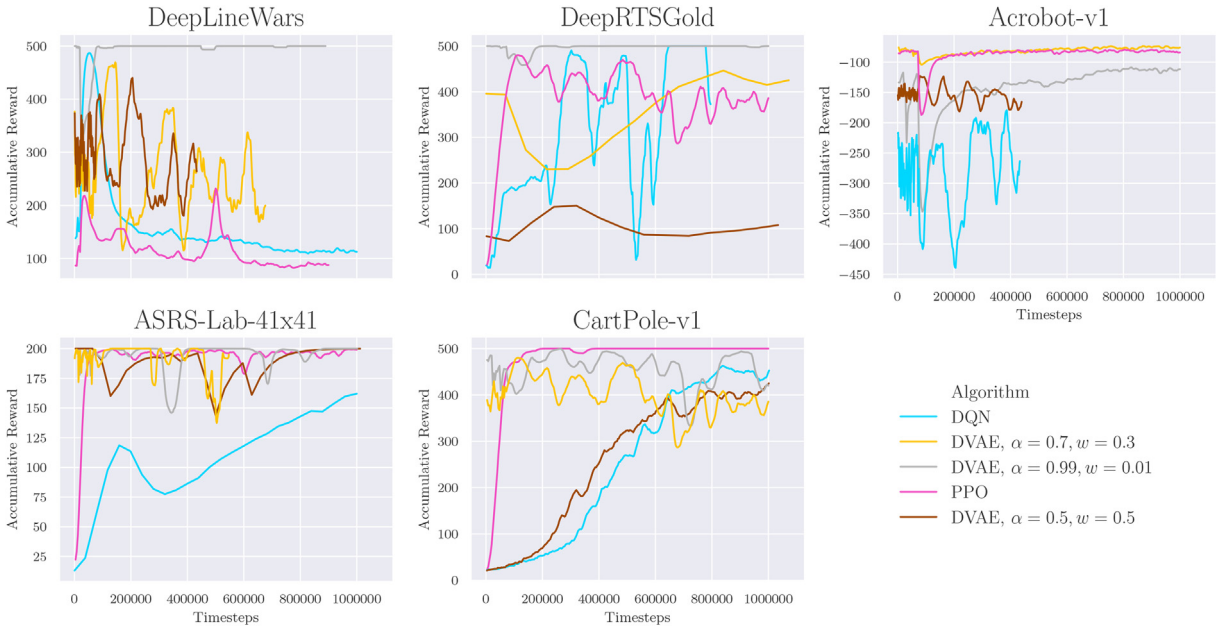


Fig. 11. Behavioral agent performance. DVAEshows good performance when accumulating reward (y-axis) during training for 1 million timesteps (x-axis). The experiment was averaged across 100 runs and was limited to only a subset of the environments due to execution time.

DeepRTSGold, Acrobot, CartPole, and ARS-Lab-41x41. The figure clearly shows that DVAE successfully trains the algorithm to a sufficient level of behavior, and can improve further when training on the real environment.

DVAE is not always stable when training in complex environments, as seen in the DeepLineWars plot. Out of 100 trials, the DVAE configuration using $\alpha = 0.7$ and $\alpha = 0.5$ diverged, and hence, was stopped before reaching 1 million timesteps.

The pretraining was done using a horizon of 40 frames for 2 million timesteps. In practice, this only results in 50000 timesteps in the real environment, resulting in magnitudes lower risk of failures.

However, **sensitivity to hyperparameters** is a significant issue that limits the algorithm from functioning well throughout all tests, without extensive tuning.

7. Conclusion and future work

The Dreaming Variational Autoencoder increases safety during the training of reinforcement learning agents. Section 6 shows that,

1. Agents have significantly lower failure rates when pretraining using the predictive model,
2. has similar performance, in terms of accumulative reward, compared to state-of-the-art algorithms, including DQN and PPO, and
3. can predict longer time-horizons with good training quality. However, the prediction error grows exponentially.

Although DVAEis less stable than model-free approaches and is the biggest challenge of using a dream model for safety–critical tasks, the sample efficiency is significantly improved. As is common in many other models, the proposed algorithm requires significant hyperparameter tuning to function well, and it could be difficult to find general parameters that work across many environments. However, we found that $\alpha = 0.99,\ w = 0.01,\ \alpha = 0.7,\ w = 0.3$, and $\alpha = 0.5,\ w = 0.5$ to perform best during the experiments.

The most considerable achievement is that **DVAEimproves sample efficiency** significantly when using the predictive model when pretraining the agent. The algorithm can predict future-state sequences of up to 100 frames with an accuracy sufficient for pretraining. This reduces the need for interacting with the real environment and hence defeats the potential risk of entering catastrophic states.

Continued research of this work is dedicated to better combine proximal policy optimization with the presented methods for safe reinforcement learning. In the DVAEt-model, we would like to investigate if temporal convolutional networks [28] could further improve the performance of learning the predictive model. We hope to experiment with the recent *vector quantized variational autoencoder* [35] for more accurate latent space (embedding) encoding. While this paper contributes new findings in safe reinforcement learning, it is still room for improvement, in which we hope to contribute more in the future.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] E. Altman, Constrained Markov Decision Processes, first ed., CRC Press, Boca Raton, FL, USA, 1999.
[2] P. Andersen, M. Goodwin, O. Granmo, Deep RTS: A game environment for deep reinforcement learning in real-time strategy games, in: 2018 IEEE Conference on Computational Intelligence and Games (CIG), 2018, pp. 1–8, https://doi.org/10.1109/CIG.2018.8490409.
[3] P.A. Andersen, M. Goodwin, O.C. Granmo, Towards a deep reinforcement learning approach for tower line wars, in: M. Bramer, M. Petridis, (Eds.), Artificial Intelligence XXXIV, Springer International Publishing, Cham, CH, 2017, pp. 101–114, doi: 10.1007/978-3-319-71078-5_8.
[4] P.A. Andersen, M. Goodwin, O.C. Granmo, The dreaming variational autoencoder for reinforcement learning environments, in: Bramer Max, M. Petridis, (Eds.), Artificial Intelligence XXXV. xxxv ed. Springer, Cham, vol. 11311, 2018, pp. 143–155, doi: 10.1007/978-3-030-04191-5_11.
[5] P.A. Andersen, M. Goodwin, O.C. Granmo, Towards model-based reinforcement learning for industry-near environments, in: M. Bramer, M. Petridis, (Eds.), Artificial Intelligence XXXVI, Springer International Publishing, Cham, CH, 2019, pp. 36–49, doi: 10.1007/978-3-030-34885-4_3.
[6] K. Arulkumaran, M.P. Deisenroth, M. Brundage, A.A. Bharath, Deep reinforcement learning: A brief survey, IEEE Signal Processing Magazine 34 (2017) 26–38, https://doi.org/10.1109/MSP.2017.2743240.
[7] M.G. Azar, B. Piot, B.A. Pires, J.B. Grill, F. Altché, R. Munos, World Discovery Models, 2019, ArXiv e-prints arXiv:1902.07685.
[8] M.G. Bellemare, Y. Naddaf, J. Veness, M. Bowling, The arcade learning environment: an evaluation platform for general agents, Journal of Artificial Intelligence Research 47 (2013) 253–279, https://doi.org/10.1613/jair.3912.
[9] F. Berkenkamp, M. Turchetta, A.P. Schoellig, A. Krause, Safe model-based reinforcement learning with stability guarantees, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates Inc, Long Beach, CA, USA, 2017, pp. 908–918, URL:https://papers.nips.cc/paper/6692-safe-model-based-reinforcement-learning-with-stability-guarantees.
[10] C. Chen, A. Seff, A. Kornhauser, J. Xiao, DeepDriving: learning affordance for direct perception in autonomous driving, in: Proc. International Conference on Computer Vision, ICCV'2015, IEEE, Santiago, CL, 2015, pp. 2722–2730, https://doi.org/10.1109/ICCV.2015.312.
[11] Y. Chow, M. Ghavamzadeh, L. Janson, M. Pavone, Risk-constrained reinforcement learning with percentile risk criteria, Journal of Machine Learning Research 18 (2017) 6070–6120, URL:http://www.jmlr.org/papers/volume18/15-636/15-636.pdf.
[12] K. Chua, R. Calandra, R. McAllister, S. Levine, Deep reinforcement learning in a handful of trials using probabilistic dynamics models, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems 31. Curran Associates, Inc., 2018, pp. 4754–4765. https://papers.nips.cc/paper/7725-deep-reinforcement-learning-in-a-handful-of-trials-using-probabilistic-dynamics-models.
[13] L.L. Edith, C. Melanie, P. Doina, R. Bohdana, Risk-directed exploration in reinforcement learning, in: IJCAI 2005 Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains, 2005.
[14] J. Fan, Z. Wang, Y. Xie, Z. Yang, A theoretical analysis of deep Q-learning, 2019, Technical Report. Princeton University.arXiv:1901.00137.
[15] E.A. Feinberg, M.E. Lewis, On the convergence of optimal actions for Markov decision processes and the optimality of (s, S) inventory policies, Naval Research Logistics 65 (2018) 619–637, https://doi.org/10.1002/nav.21750.
[16] R. Fox, A. Pakman, N. Tishby, Taming the noise in reinforcement learning via soft updates, in: Proc. 32nd Conference on Uncertainty in Artificial Intelligence, UAI'16, AUAI Press, Arlington, VA, USA, 2016, pp. 202–211, doi: 10.5555/3020948.3020970.
[17] N. Fulton, A. Platzer, Safe reinforcement learning via formal methods: toward safe control through proof and learning, in: Proc. 32nd Conference on Artificial Intelligence, AAAI'18, AAAI Press, New Orleans, LA, USA, 2018, pp. 2669–2678, URL:https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/download/17376/16225.
[18] J. Garcia, F. Fernandez, Safe exploration of state and action spaces in reinforcement learning, Journal of Artificial Intelligence Research 45 (2012) 515–564, https://doi.org/10.1613/jair.3761.
[19] P. Geibel, F. Wysotzki, Risk-sensitive reinforcement learning applied to control under constraints, Journal of Artificial Intelligence Research 24 (2005) 81–108, https://doi.org/10.1613/jair.1666.
[20] K. Gregor, D. Jimenez Rezende, F. Besse, Y. Wu, H. Merzic, A. van den Oord, Shaping belief states with generative environment models for RL, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates Inc, Vancouver, BC, CA, 2019, pp. 13475–13487, URL:http://papers.nips.cc/paper/9503-shaping-belief-states-with-generative-environment-models-for-rl.pdf.
[21] D. Ha, J. Schmidhuber, Recurrent world models facilitate policy evolution, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems 31, Curran Associates Inc, Montréal, QC, CA, 2018, pp. 2450–2462, URL: http://papers.nips.cc/paper/7512-recurrent-world-models-facilitate-policy-evolution.pdf.
[22] S.T. Hackman, M.J. Rosenblatt, J.M. Olin, Allocating items to an automated storage and retrieval system, IIE Transactions 22 (1990) 7–14, https://doi.org/10.1080/07408179008964152.
[23] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, J. Davidson, Learning latent dynamics for planning from pixels, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proc. 36th International Conference on Machine Learning, ICML'18, PMLR, Long Beach, CA, USA, 2019, pp. 2555–2565, URL:http://proceedings.mlr.press/v97/hafner19a/hafner19a.pdf.
[24] M. Hairer, Convergence of Markov processes, 2016, Technical Report. Mathematics Department, University of Warwick.
[25] M. Heger, Consideration of risk in reinforcement learning, in: W.W. Cohen, H. Haym, (Eds.), Proc. 11th International Conference on Machine Learning, ICML'94, Elsevier, New Brunswick, NJ, USA, 1994, pp. 105–111, doi: 10.1016/B978-1-55860-335-6.50021-0.
[26] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, A. Lerchner, beta-VAE: Learning basic visual concepts with a constrained variational framework, in: Proc. 5th International Conference on Learning Representations, ICLR'17, Toulon, FR, 2017, URL:https://openreview.net/forum?id=Sy2fzU9gl.
[27] M. Janner, J. Fu, M. Zhang, S. Levine, When to trust your model: model-based policy optimization, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates Inc, Vancouver, BC, CA, 2019, pp. 12519–12530, URL:http://papers.nips.cc/paper/9416-when-to-trust-your-model-model-based-policy-optimization.pdf.
[28] C. Lea, R. Vidal, A. Reiter, G.D. Hager, Temporal convolutional networks: a unified approach to action segmentation, in: G. Hua, H. Jégou, (Eds.), Proc. 14th European Conference on Computer Vision, Springer International Publishing, Amsterdam, NL, 2016, pp. 47–54, URL:http://link.springer.com/10.1007/978-3-319-49409-8_7, doi: 10.1007/978-3-319-49409-8_7.
[29] S. Leo Kumar, Knowledge-based expert system in manufacturing planning: state-of-the-art review, International Journal of Production Research 57 (2019) 4766–4790, https://doi.org/10.1080/00207543.2018.1424372.
[30] J. Li, L. Yao, X. Xu, B. Cheng, J. Ren, Deep reinforcement learning for pedestrian collision avoidance and human-machine cooperative driving, Information Sciences 532 (2020) 110–124, https://doi.org/10.1016/j.ins.2020.03.105.
[31] T. Mannucci, E.J. van Kampen, C. de Visser, Q. Chu, Safe exploration algorithms for reinforcement learning controllers, IEEE Transactions on Neural Networks and Learning Systems 29 (2018) 1069–1081, https://doi.org/10.1109/TNNLS.2017.2654539.
[32] R. McAllister, C.E. Rasmussen, Data-efficient reinforcement learning in continuous state-action Gaussian-POMDPs, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates Inc, Long

Beach, CA, USA, pp. 2040–2049, URL:http://papers.nips.cc/paper/6799-data-efficient-reinforcement-learning-in-continuous-state-action-gaussian-pomdps.pdf.
[33] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533, https://doi.org/10.1038/nature14236.
[34] D. Pathak, P. Agrawal, A.A. Efros, T. Darrell, Curiosity-driven exploration by self-supervised prediction, in: Proc. 34th International Conference on Machine Learning, ICML'17, JMLR.org, Sydney, NSW, AU, 2017, pp. 2778–2787, URL:https://dl.acm.org/doi/10.5555/3305890.3305968.
[35] A. Razavi, A. van den Oord, O. Vinyals, Generating diverse high-fidelity images with VQ-VAE-2, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates Inc, Vancouver, BC, CA, 2019, pp. 14837–14847, URL:http://papers.nips.cc/paper/9625-generating-diverse-high-fidelity-images-with-vq-vae-2.
[36] R.S. Sutton, A.G. Barto, Reinforcement Learning: An introduction, 2 ed., A Bradford Book, Cambridge, MA, USA, 2018, URL:https://dl.acm.org/doi/book/10.5555/3312046.
[37] X.S. Wang, Y.H. Cheng, J.Q. Yi, A fuzzy Actor-Critic reinforcement learning network, Information Sciences 177 (2007) 3764–3781, https://doi.org/10.1016/j.ins.2007.03.012.
[38] X. Xu, L. Zuo, Z. Huang, Reinforcement learning algorithms with function approximation: Recent advances and applications, Information Sciences 261 (2014) 1–31, https://doi.org/10.1016/j.ins.2013.08.037.
[39] C. Zhang, P. Patras, H. Haddadi, Deep learning in mobile and wireless networking: a survey, IEEE Communications Surveys & Tutorials 21 (2019) 2224–2287, https://doi.org/10.1109/COMST.2019.2904897.
[40] C. Zhou, Robot learning with GA-based fuzzy reinforcement learning agents, Information Sciences 145 (2002) 45–68, https://doi.org/10.1016/S0020-0255(02)00223-2.