Learning Robust Dynamics through Variational Sparse Gating

Arnav Kumar Jain[1,2,*], **Shivakanth Sujit**[2,3], **Shruti Joshi**[2,3], **Vincent Michalski**[1,2]
Danijar Hafner[4,5], **Samira Ebrahimi-Kahou**[2,3,6]

Abstract

Learning world models from their sensory inputs enables agents to plan for actions by imagining their future outcomes. World models have previously been shown to improve sample-efficiency in simulated environments with few objects, but have not yet been applied successfully to environments with many objects. In environments with many objects, often only a small number of them are moving or interacting at the same time. In this paper, we investigate integrating this inductive bias of sparse interactions into the latent dynamics of world models trained from pixels. First, we introduce Variational Sparse Gating (VSG), a latent dynamics model that updates its feature dimensions sparsely through stochastic binary gates. Moreover, we propose a simplified architecture Simple Variational Sparse Gating (SVSG) that removes the deterministic pathway of previous models, resulting in a fully stochastic transition function that leverages the VSG mechanism. We evaluate the two model architectures in the BringBackShapes (BBS) environment that features a large number of moving objects and partial observability, demonstrating clear improvements over prior models.

1 Introduction

Latent dynamics models are models that generate agent's future states in the compact latent space without feeding the high-dimensional observations back to the model. They have shown promising results on various tasks like video prediction (Karl et al., 2016; Kalman, 1960; Krishnan et al., 2015), model-based Reinforcement Learning (RL) (Hafner et al., 2020; 2021; 2019; Ha and Schmidhuber, 2018), and robotics (Watter et al., 2015). Generating high dimensional inputs like images in latent space reduces the accumulating errors leading to more accurate long-term predictions. Additionally, having lower dimensionality leads to a lower memory footprint. Solving tasks in model-based RL involves learning a world model (Ha and Schmidhuber, 2018) that can predict outcomes of actions, followed by using them to derive behaviors (Sutton, 1991). Motivated by these benefits, the recently proposed DreamerV1 (Hafner et al., 2020) and DreamerV2 (Hafner et al., 2021) agents achieved state-of-the-art results on a wide range of visual control tasks.

Many complex tasks require reliable long-term prediction of dynamics. This is true especially in partially observable environments where only a subspace is visible to the agent, and it is usually required to accurately retain information over multiple time steps to solve the task. The Dreamer agents (Hafner et al., 2020; 2021) employ an Recurrent State-Space Model (RSSM) (Hafner et al., 2019) comprising of a Recurrent Neural Network (RNN). Training RNNs for long sequences is challenging as they suffer from optimization problems like vanishing gradients (Hochreiter, 1991; Bengio et al., 1994). Different ways of applying sparse updates in RNNs have been investigated (Campos et al., 2017; Neil et al., 2016; Goyal et al., 2019), enabling a subset of state dimensions to be constant

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

[1]Université de Montréal, [2]Mila- Quebec AI Institute, [3]École de technologie supérieure, [4]University of Toronto, [5]Google Brain, [6]CIFAR. [*]Correspondence to Arnav Kumar Jain <arnav-kumar.jain@mila.quebec>

during the update. A sparse update prior is also motivated by the fact that in the real world, many factors of variation are constant over extended periods of time. For instance, several objects in a physical simulation may be stationary until some force acts upon them. Additionally, this is useful in the partially observable setting where the agent observes a constrained viewpoint and has to keep track of objects that are not visible for many time steps. In this work, we introduce Variational Sparse Gating (VSG), a stochastic gating mechanism that sparsely updates the latent states at each step.

Recurrent State-Space Model (RSSM) (Hafner et al., 2019) was introduced in PLaNet where the model state was composed of two paths, a image representation path and a recurrent path. DreamerV1 (Hafner et al., 2020) and DreamerV2 (Hafner et al., 2021) utilized them to achieve state-of-the-art results in continuous and discrete control tasks (Hafner et al., 2019). While the image representation path which is stochastic accounts for multiple possible future states, the recurrent path which is deterministic is motivated by stable retention of information over multiple time steps to facilitate gradient-based optimization. The authors in PLaNet (Hafner et al., 2019) showed that both components were important for solving tasks, where the stochastic part was more important to account for partial observability of the initial states. By leveraging the proposed gating mechanism (Variational Sparse Gating (VSG)), we demonstrate that a purely stochastic model with a single component can achieve competitive results, and call it Simple Variational Sparse Gating (SVSG). To the best of our knowledge, this is the first work that shows that purely stochastic models achieve competitive performance on continuous control tasks when compared to leading agents.

Existing benchmarks (Bellemare et al., 2013; Chevalier-Boisvert et al., 2018; Tassa et al., 2018) for RL do not test the capability of agents in both partial observability and stochasticity. The Atari (Bellemare et al., 2013) benchmark comprises of 55 games but most of the games are deterministic and a lot of compute is required to train on them. Some tasks in the Atari and Minigrid benchmarks are partially-observable but either lack stochasticity or are hard exploration tasks. Also, these benchmarks do not allow for controlling the factors of variation. We developed a new partially-observable and stochastic environment, called BringBackShapes (BBS), where the task is to push objects to a predefined goal area. Solving tasks in BBS require agents to remember states of previously observed objects and avoid noisy distractor objects. Furthermore, VSG and SVSG outperformed leading model-based and model-free baselines. We also present studies with varying partial-observability and stochasticity to demonstrate that the proposed agents have better memory for tracking observed objects and are more robust to increasing levels of noise. Lastly, the proposed methods were also evaluated on existing benchmarks - DeepMind Control (DMC) (Tassa et al., 2018), DMC with Natural Background (Zhang et al., 2021; Nguyen et al., 2021b), and Atari (Bellemare et al., 2013). On the existing benchmarks, the proposed method performed better on tasks with changing viewpoints and sparse rewards.

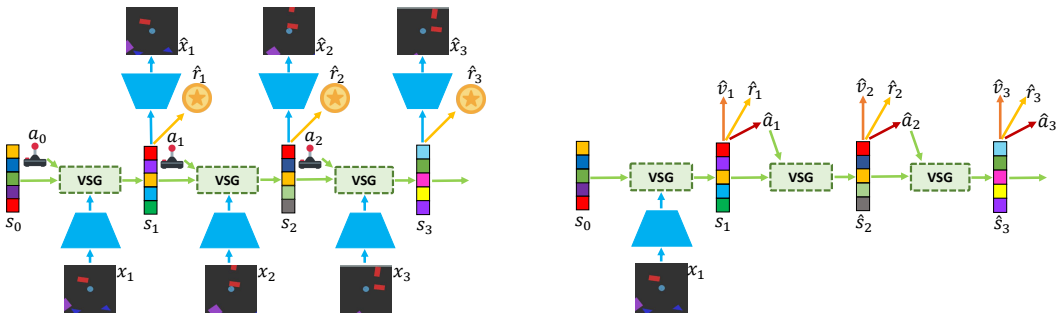
Our key contributions are summarized as follows:

- **Variational Sparse Gating**: We introduce Variational Sparse Gating (VSG), where the recurrent states are sparsely updated through a stochastic gating mechanism. A comprehensive empirical evaluation shows that VSG outperforms baselines on tasks requiring long-term memory.
- **Simple Variational Sparse Gating**: We also propose Simple Variational Sparse Gating (SVSG) which has a purely stochastic state, and achieves competitive results on continuous control tasks when compared with agents that also use a deterministic component.
- **BringBackShapes**: We developed the BringBackShapes (BBS) environment to evaluate agents on partially-observable and stochastic settings where these variations can be controlled. Our experiments show that the proposed agents are more robust such variations.

2 Variational Sparse Gating

Reinforcement Learning: The visual control task can be formulated as a Partially Observable Markov Decision Process (POMDP) with discrete time steps $t \in [1; T]$. The agent selects action $a_t \sim p(a_t|o_{\leq t}, a_{<t})$ to interact with the environment and receives the next observation and scalar reward $o_t, r_t \sim p(o_t, r_t|o_{<t}, r_{<t})$, respectively, at each time step. The goal is to learn a policy that maximizes the expected discounted sum of rewards $\mathbb{E}_p(\sum_{t=1}^{T} \gamma^t r_t)$, where γ is the discount factor.

Agent: Agent is composed of a world model and a policy (Fig. 1). World models (Sec. 2.1) encode a sequence of observations and actions into latent representations. Behaviors (Appendix B) are derived to maximize expected returns on the trajectories generated from the learned world model. While training, the world model is learned with collected experience, the policy is improved on



(a) (b)

Figure 1: (a) World Model: The input images x_t are encoded using Convolutional Neural Network (CNN) and Multi-layer Perceptron (MLP) layers. The VSG takes the previous model state s_{t-1} and action a_{t-1}, and outputs the updated model state at next step s_t, which is further used to reconstruct image $\hat{x}_t$ and reward $\hat{r}_t$. (b) Policy: Comprises of an actor to select optimal action $\hat{a}_t$ and critic to predict value $\hat{v}_t$ beyond the planning horizon. The world model is unrolled using the prior model state $\hat{s}_t$ which does not contain information about image x_t.

trajectories unrolled using the world model and new episodes are collected by deploying the policy in the environment. An initial set of episodes are collected using a random policy. As training progresses, new episodes are collected using the latest policy to further improve the world model.

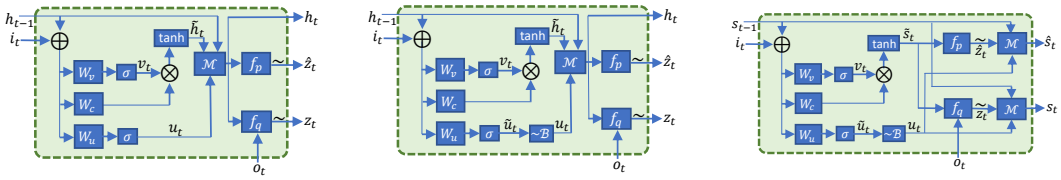
2.1 World Model

World Models (Ha and Schmidhuber, 2018) learn to mimic the environment, using the collected experience, to facilitate deriving behaviours in the abstract latent space. Given an abstract state of the world and an action, the model applies the learned transition dynamics to predict the resulting next state and reward. RSSM (Hafner et al., 2019) was introduced in PlaNet, where the model state was composed of two paths. The recurrent path consists of an RNN (See Figure 2 [a]), and is motivated with reliable long-term information preservation, while the image representation path samples from a learned distribution to account for multiple possible futures (Babaeizadeh et al., 2017). In this work, we introduce Variational Sparse Gating (VSG), where the recurrent path selectively updates a subset of the latent states at each step using a stochastic gating network. Sparse updates enables agents to have long-term memory and learn robust representations which is required for complex tasks.

Model Components: The world model comprises of an image encoder, a VSG model, and predictors for image, discount and reward. The image encoder generates representations o_t for the observation x_t using CNNs. The VSG model comprises of a recurrent network equipped with the stochastic gating mechanism, and is used to compute two stochastic image representation states. The posterior representation state z_t contains information about the current image x_t, while the prior state $\hat{z}_t$ does not observe x_t. For planning, sequences are generated in compact latent state by utilizing the prior representation state $\hat{z}_t$. This results in a lower memory footprint and enables predictions of thousands of trajectories in parallel on a single GPU. The representation states are sampled from a known distribution with learned parameters like Gaussian (Hafner et al., 2020) or Categorical (Hafner et al., 2021). The concatenation of outputs from the recurrent and image representation models gives the compact model state ($s_t = [h_t, z_t]$). The posterior model state is further used to reconstruct the original image $\hat{x}_t$, predict the reward $\hat{r}_t$, and discount factor $\hat{\gamma}_t$. The discount factor helps to predict the probability that an episode will end. The components of world model are as follows:

$$
\begin{aligned}
&\text{Recurrent model:} && h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}) \\
&\text{Representation model:} && z_t \sim q_\phi(z_t \mid h_t, x_t) \\
&\text{Transition predictor:} && \hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t) \\
&\text{Image predictor:} && \hat{x}_t \sim p_\phi(\hat{x}_t \mid h_t, z_t) \\
&\text{Reward predictor:} && \hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t, z_t) \\
&\text{Discount predictor:} && \hat{\gamma}_t \sim p_\phi(\hat{\gamma}_t \mid h_t, z_t).
\end{aligned}
\tag{1}
$$

Neural Networks: The representation model outputs the posterior image representation state z_t conditioned on the image encoding x_t and recurrent state h_t. The transition predictor provides the



(a) (b) (c)

Figure 2: Architectures of (a) Recurrent State-Space Model (RSSM), (b) Variational Sparse Gating (VSG), and (c) Simple Variational Sparse Gating (SVSG), respectively. σ and $\tanh$ denote the sigmoid and tanh non-linear activations, respectively. W_* and b_* are the corresponding weights and biases. $\sim$, $\oplus$ and $\otimes$ denote sampling, vector concatenation, and element-wise multiplication, respectively. $\mathcal{M}$ computes $x_t = u_t\tilde{x}_t + (1-u_t)x_{t-1}$, where $x_t = h_t$ is used for RSSM and VSG, and $x_t = s_t$ is used for SVSG. $\mathcal{B}$ denotes Bernoulli distribution. f_p and f_q denote the prior and posterior distributions with learned parameters, respectively (See Appendix I for more details).

prior image representation state $\hat{z}_t$. The image encoding o_t is obtained by passing the image x_t through CNN (LeCun et al., 1989) and MLP layers. In VSG, we propose to modify the Gated Recurrent Unit (GRU) used in RSSM to sparsely update the recurrent state at each step. The model state s_t, which is a concatenation of recurrent and image representation states is passed through several layers of MLP to predict the discount and reward, and transposed CNN layers are used to reconstruct the image. The Exponential Linear Unit (ELU) activation is used for training all the components of the world model (Clevert et al., 2015).

Sparse Gating: In light of training RNNs to capture long-term dependencies, different ways of applying sparse updates have been investigated (Campos et al., 2017; Neil et al., 2016; Goyal et al., 2019), enabling a subset of state dimensions to be constant during the update. They were found to alleviate the vanishing gradient problem by effectively reducing the number of sequential operations (Campos et al., 2017). Discrete gates may also improve long-term memory by avoiding the gradual change of state values introduced by repeated multiplication with continuous gate values in standard recurrent architectures. Previous works on sparsely updating hidden states (Campos et al., 2017; Neil et al., 2016) use a separate layer applied over the outputs of RNN, and do not modify the RNN in itself. However, in this work, we modify the update gate in GRU (Cho et al., 2014) to take binary values by sampling from a Bernoulli distribution (Fig. 2 [b] shows the architecture).

The input i_t to the recurrent model contains information about the action and is obtained by concatenating the previous image representation state z_{t-1} and action a_t followed by passing them through through a MLP layer. Similar to GRU (Cho et al., 2014), there are two gates- reset and update. The reset gate v_t decides the extent of information flow from the previous recurrent state and inputs, and the update gate u_t tells which parts of the recurrent state will be updated. The update gate takes only binary values, selecting whether the value will be updated or copied from previous time step. Binary values are obtained by sampling from a Bernoulli distribution where the probability of sampling is obtained using the previous recurrent state h_{t-1} and input i_t. Straight-through estimators (Bengio et al., 2013) were used for propagating gradients backwards for training. The update equations are:

$$
\begin{aligned}
v_t &= \sigma(W_v^T[h_{t-1}, i_t] + b_v) \\
\tilde{u}_t &= \sigma(W_u^T[h_{t-1}, i_t] + b_u) \\
\tilde{h}_t &= \tanh(v_t * (W_c^T[h_{t-1}, i_t] + b_c)) \\
u_t &\sim \text{Bernoulli}(\tilde{u}_t) \\
h_t &= u_t \odot \tilde{h}_t + (1-u_t) \odot h_{t-1},
\end{aligned}
\tag{2}
$$

where $\odot$ denotes element-wise multiplication, σ and $\tanh$ are the sigmoid and hyperbolic tangent activation function, and W_* and b_* denotes the weights and biases, respectively. To control the sparsity of updates, we have used KL divergence between probability of sampling the update gate $\tilde{u}_t$ and a fixed prior probability κ, where κ is a tunable hyperparameter.

Loss function: The predictors for image and reward produces Gaussian distributions with unit variance, whereas the discount predictor predicts a Bernoulli likelihood. The image representation states are sampled from a Gaussian (Hafner et al., 2020) or a Categorical (Hafner et al., 2021) distribution which are trained to maximize the likelihood of targets. In addition, there is a KL

Divergence term between the prior and posterior distributions and similar to DreamerV2 (Hafner et al., 2021), we have also used KL balancing with a factor of 0.8. We have also added a sparsity loss to regularize the number of updates in hidden state at each step. All the components of the world model are optimized jointly using the loss function given by:

$$\begin{aligned}\mathcal{L}(\phi) \doteq \mathrm{E}_{q_\phi(z_{1:T} \mid a_{1:T}, x_{1:T})} \Big[\sum_{t=1}^{T} & \underbrace{-\ln p_\phi(x_t \mid h_t, z_t)}_{\text{image log loss}} \underbrace{-\ln p_\phi(r_t \mid h_t, z_t)}_{\text{reward log loss}} \\ & \underbrace{-\ln p_\phi(\gamma_t \mid h_t, z_t)}_{\text{discount log loss}} \underbrace{+\beta \, \mathrm{KL}\big[q_\phi(z_t \mid h_t, x_t) \,\big\|\, p_\phi(z_t \mid h_t)\big]}_{\text{KL loss}} \underbrace{+\alpha \, \mathrm{KL}\big[\tilde{u}_t \,\big\|\, \kappa\big]}_{\text{sparsity loss}} \Big],\end{aligned} \tag{3}$$

where β and α are the scale for KL losses of the latent codes and the sparse update gates, respectively.

3 Simple Variational Sparse Gating

Stochastic State-Space Model (SSM) were proposed in PLaNet (Hafner et al., 2019), where it was discussed that it is not trivial to achieve competitive results without the deterministic recurrent path. Having a deterministic component was motivated to allow the transition model to retain information for multiple time steps as the stochastic component induces variance (Hafner et al., 2019). In this work, we show that having a purely stochastic component achieves comparable performance with DreamerV2 while significantly outperforming SSMs (refer to Appendix H for more details). We introduce a simplified version of VSG, called Simple Variational Sparse Gating (SVSG) where the world model has a model state s_t with single path to preserve information over multiple steps and also account for partial observability in future states (Fig. 2 [c] presents the SVSG architecture). The model state s_t is sparsely updated at each step using the stochastic gating mechanism proposed in VSG. In SVSG, the posterior state s_t is conditioned on the previous state s_{t-1}, input image x_t and the action a_t, whereas the prior state $\hat{s}_t$ is generated without using the image observation. We have used a diagonal Gaussian distribution for the stochastic state with a learnable mean and standard deviation. Similar to VSG, the posterior state is used to reconstruct the image, and predict the reward and discount factor. The components of world models in SVSG are:

$$\begin{aligned}&\text{Representation model:} && s_t \sim q_\phi(s_t \mid s_{t-1}, x_t, a_t) \\ &\text{Transition predictor:} && \hat{s}_t \sim p_\phi(\hat{s}_t \mid s_{t-1}, a_t) \\ &\text{Image predictor:} && \hat{x}_t \sim p_\phi(\hat{x}_t \mid s_t) \\ &\text{Reward predictor:} && \hat{r}_t \sim p_\phi(\hat{r}_t \mid s_t) \\ &\text{Discount predictor:} && \hat{\gamma}_t \sim p_\phi(\hat{\gamma}_t \mid s_t).\end{aligned} \tag{4}$$

The representation model q_ϕ and transition predictor p_ϕ are modified to output the posterior s_t and prior $\hat{s}_t$ states, respectively. The candidate state $\tilde{s}_t$ at each step is obtained using input i_t, reset gate v_t and previous state s_{t-1}. Similar to VSG, the update gate u_t is sampled from a Bernoulli distribution to sparsely update the latent states at each step, given by:

$$\begin{aligned}v_t &= \sigma(W_v^T[s_{t-1}, i_t] + b_v) \\ \tilde{u}_t &= \sigma(W_u^T[s_{t-1}, i_t] + b_u) \\ \tilde{s}_t &= \tanh(v_t \odot (W_c^T[s_{t-1}, i_t] + b_c)) \\ u_t &\sim \text{Bernoulli}(\tilde{u}_t),\end{aligned} \tag{5}$$

where $\odot$ denotes the element-wise multiplication, σ and $\tanh$ are the sigmoid and hyperbolic tangent activation functions, and W_* and b_* denote the weights and biases, respectively.

The candidate state $\tilde{s}_t$ was passed through MLP layers to get the prior and posterior distributions. The image encoding x_t was used to get posterior distribution, whereas the prior distribution was predicted without it. After sampling the prior $\hat{z}_t$ and posterior z_t candidate states, the update gate sparsely modifies the previous latent state and outputs the prior $\hat{s}_t$ and posterior s_t model state at each step, respectively. The update equations are given by:

$$\begin{aligned}\hat{z}_t &\sim f_{\mathrm{p}}(\tilde{s}_t) \\ z_t &\sim f_{\mathrm{q}}(\tilde{s}_t, x_t) \\ \hat{s}_t &= u_t \odot \hat{z}_t + (1 - u_t) \odot s_{t-1} \\ s_t &= u_t \odot z_t + (1 - u_t) \odot s_{t-1},\end{aligned} \tag{6}$$

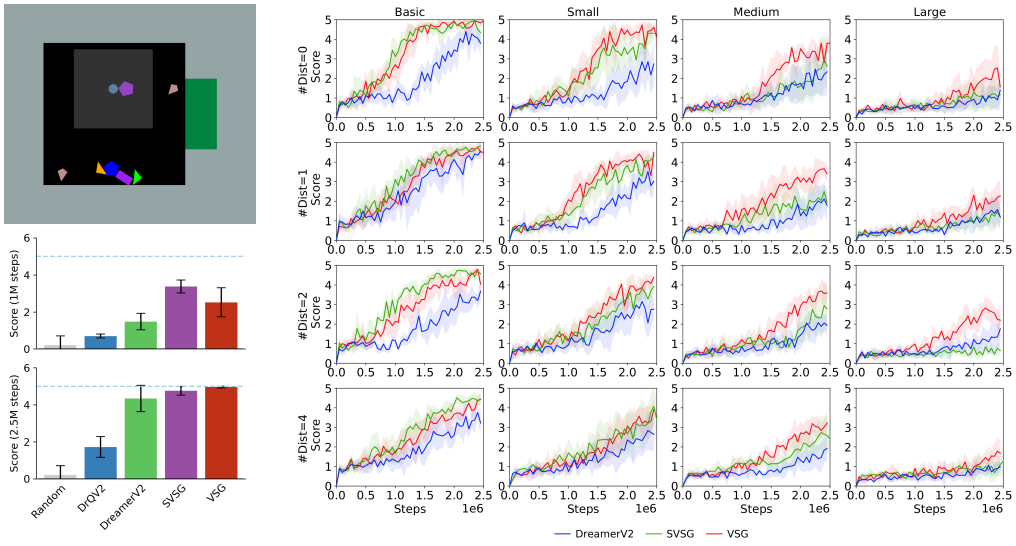


Figure 3: a) (Top Left) Full arena of the BringBackShapes (BBS) where the gray region around agent shows the partial view received by it. The circular agent is located in the center of partial view and is of teal blue color. The task is to push objects in the green goal region on the right side of arena. b) (Bottom left) Scores obtained on BBS with Basic size and no distractors at 1M and 2.5M steps. c) (Right) Performance (results over 5 seeds are reported) at different sizes of arena and number of distractor objects (#Dist). VSG and SVSG outperforms DreamerV2 significantly in most scenarios.

where f_{p} and f_{q} denotes functions that output a distribution with learnable parameters for prior and posterior, respectively. $\sim$ denotes sampling from the learned distribution. For SVSG, Categorical latents (Hafner et al., 2021) were not performing well on our tasks. We attribute this to the fact that samples from a categorical distribution are binary vectors and it is difficult to accurately reconstruct with such sparse latent representations. Lastly, we observed that the sparse gating mechanism introduced in VSG was important for the convergence of SVSG.

For training the SVSG model, we replace the KL loss term between prior and posterior distributions in Eq. 3 with a masked KL loss that penalizes the state dimensions that were updated in the corresponding time step, i.e. those for which the corresponding element in u_t is equal to 1. We found this to be necessary, since the original, unmasked KL loss did not yield good performance, presumably due to its effect on state dimensions that were not updated. We have used the same loss function as described in Sec. 2 and policy is similar to used in VSG (described in Appendix B).

4 Experiments

4.1 BringBackShapes

In this work, we developed the BringBackShapes (BBS) environment to test the ability of agents to solve tasks in partially-observable and stochastic scenarios (see Fig. 3 [a]). The task is to push the objects within the arena into a pre-specified goal area. Moreover, rewards are sparse and is +1 for successfully pushing an object into the goal. At each time step, the agent only receives an obfuscated view of the arena centered around its current position. This requires agents to efficiently explore the arena to find new objects as well as remember states of previously observed objects. The objects can collide with each other and the walls, which further requires the agent to account for these events while updating its state. Stochasticity was introduced in the environment by using random distractor objects. The distractor objects follow Brownian motion in any direction and can't be pushed into the goal area. Additionally, they add noise to the reward signal as they might push objects into the goal, causing a reward that is not or only partially related to the agent's behavior. Due to partial-observability, such instances might not be visible to the agent, making the task even more challenging. Refer to Appendix A for further description of the environment.

Quantitative Results: Fig. 3 [b] compares the proposed methods VSG and SVSG with the leading RL agents- DreamerV2 (Hafner et al., 2021) and DrQ-V2 (Yarats et al., 2022). The score indicates how many objects on average were scored within a episode. As discussed in Section **??**, we

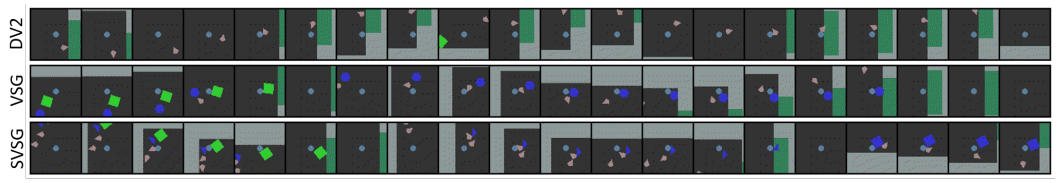


Figure 4: Learned behaviors of DreamerV2 (DV2), VSG and SVSG agents on BringBackShapes on Basic size and with 2 distractor objects at different steps. DreamerV2 fails to capture that distractors (whitish cones) are noisy objects and tries to push them towards the goal, whereas VSG and SVSG learn to avoid the noisy objects and carefully maneuvers the right objects towards the goal.

	First-Visit Time	Episode Length	Objects Not Visited (%)	Visited Objects Not Scored (%)	First-Visit Time	Episode Length	Objects Not Visited (%)	Visited Objects Not Scored (%)
	Basic, #Distractors=0				Medium, #Distractors=0			
DV2	500.25	2503.55	4.6	14.67	1800.79	2994.91	50.2	33.68
VSG	**276.20**	**1881.80**	**0.08**	**1.38**	**1360.32**	**2953.21**	32.6	**24.98**
SVSG	365.08	2196.37	2.00	5.15	1375.99	2964.85	**30.00**	38.53
	Basic, #Distractors=2				Medium, #Distractors=2			
DV2	593.53	2908.73	6.40	35.90	1669.42	2998.51	40.80	45.48
VSG	330.74	2482.40	**0.4**	9.67	**1051.29**	**2944.34**	**16.4**	**32.12**
SVSG	**313.95**	**2292.00**	0.6	**7.61**	1484.09	2988.79	31.80	51.42

Table 1: Average values of the first-visit time, episode length, % of objects not visited, and % of objects visited but not scored within an episode for trained agents on Basic and Medium environments, and with 0 and 2 distractor objects respectively. Metrics were calculated for 50 episodes for each of the 5 seeds. VSG and SVSG significantly outperformed DreamerV2 on most statistics.

trained SVSG with Gaussian latents only. Upon evaluation at 2.5M timesteps, DreamerV2 achieves competitive scores when compared to the proposed methods. Whereas at 1M steps, DreamerV2 does not perform as well as VSG, which has mean score of **4.9**. This shows that learning with sparsity priors helps improve the sample efficiency. Furthermore, performance of SVSG is better than DreamerV2 but similar to VSG, demonstrating that a purely stochastic model can achieve similar performance.

Varying Partial-Observability and Stochasticy: We also study the effect of partial observability and stochasticity. For partial-observability, we increased the size of the arena while reducing the portion visible to the agent. We consider 4 configurations of the arena- Basic, Small, Medium and Large. For stochasticity, we increase the number of distractor objects using values 0, 1, 2, and 4. Fig. 3 [c] presents the plots of models trained at 2.5M steps at different sizes of the arena and number of distractor objects. It can be observed that increasing the size of arena makes it harder to score objects. VSG was found to outperform DreamerV2 across all arena sizes. However, SVSG did not perform well on larger arena sizes. Adding noisy distractor objects led to drop in final performance of all models. But VSG and SVSG still outperformed DreamerV2, indicating that the sparsity prior helps in ignoring the noisy objects in the arena while solving the task.

Ablation Studies: In this work, we also report statistics to describe the behavior of learned agents. First-visit time is the number of episode steps taken to visit an object (when object is completely visible in the agent's view). Lower first-visit time for all objects tells that an agent is able to quickly discover all the objects in the arena. Another metric is Episode Length which denotes the number of steps taken by the agent to complete the task. The maximum of these scores was set to 3000. We also report the percentage of objects that were visited within an episode which represents the ability of agents to explore all parts of the arena to find novel objects. Lastly, we report the percentage of visited objects that were not scored which tells us that the agent might not be remembering positions of objects seen previously. Thus, agents might have to explore the arena again to find them leading to an increase in time taken to finish the task. Table 1 presents the results on different settings of the environment and it can be observed that VSG and SVSG outperform DreamerV2 significantly.

Qualitative Results: We also observed the maneuvers taken by the agents to push the objects to the goal. The DreamerV2 agent was able to recognize objects and go behind them to push, but

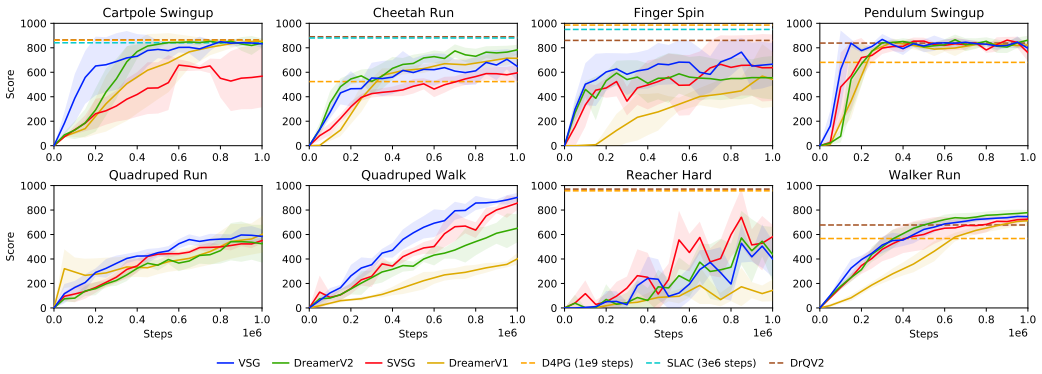


Figure 5: Comparison of VSG and SVSG with DreamerV1 (Hafner et al., 2020) and DreamerV2 (Hafner et al., 2021) on the DeepMind Control Suite. VSG converges faster on many tasks as demonstrated by evaluation curves. Even with a single stochastic path, SVSG achieves performance competitive to the models that use a combination of multiple paths.

did not follow a smooth trajectory and was spending more time around an object to push it in the goal. However, the proposed methods had comparatively much smoother trajectories and were more efficient at pushing objects successfully in the goal area. Additionally, the proposed methods learned to avoid noisy distractor objects whereas the DreamerV2 agent was colliding with them and was trying to push them to the goal (see Fig. 4 and supplementary material for more videos).

4.2 DeepMind Control Suite

The proposed method is evaluated on a few tasks from DeepMind Control Suite (Tassa et al., 2018). Observations for the agents are high dimensional images of shape $64 \times 64 \times 3$, actions range between 1 to 12 dimensions, and episodes last for 1000 steps. An action repeat (Mnih et al., 2016) of 2 was used. The model was implemented using Tensorflow Probabability (Dillon et al., 2017) and trained on a single NVIDIA V100 GPU with 16GB memory in less than 6 hours. The agents were trained for 1M environment steps. Baselines include DreamerV1 (Hafner et al., 2020), DreamerV2 (Hafner et al., 2021), DrQ-v2 (Yarats et al., 2022), D4PG (Barth-Maron et al., 2018), and A3C (Mnih et al., 2016). Except A3C, all baselines learn policies from high dimensional pixel inputs. DreamerV2 was trained using the implementation provided by the authors. For other baselines, the metrics provided by the respective authors were used for comparison. Lastly, returns averaged across 5 seeds were reported.

Figure 5 presents the comparison of VSG and SVSG with the aforementioned baselines (Refer to Appendix D for comparison on a few more tasks). It can be observed that on most tasks, VSG performs comparable to or better than DreamerV2 (Hafner et al., 2021). Notably, VSG significantly outperforms DreamerV2 on Quadruped and Finger-Spin tasks. Furthermore, it can be observed that SVSG with a purely stochastic component has similar performance to DreamerV2, outperforming on Finger Spin and Quadruped tasks and performing worse on Cartpole Swingup and Cheetah Run tasks. In addition, SVSG significantly outperforms DreamerV1 on many tasks which also used Gaussian latents and RSSM with multiple paths. Lastly, we also present the importance of sparsity loss in VSG (See Appendix I.3), and of KL Masking and Sparsity loss in SVSG (see Appendix E.2).

In BBS, we added noise in the environment by having distractor objects. We also experimented with other form of noise where natural videos is used in the background (Zhang et al., 2021; Nguyen et al., 2021b). Since reconstruction-free model based RL (Deng et al., 2021; Nguyen et al., 2021a) perform better than reconstruction based agents (Hafner et al., 2021) in such scenarios, we updated the RSSM block in DreamerPro (Deng et al., 2021) with VSG and call it VSGPro, and trained it on DMC with natural background (Refer to Appendix F). Lastly, we also compare VSG on discrete control tasks from Atari benchmark in Appendix G and VSG performed better on task with changing viewpoints.

5 Related Work

Latent Dynamics Models: Recent advancements in deep learning have allowed learning expressive latent dynamics models using stochastic backpropagation (Kingma and Welling, 2013; Chung et al., 2015; Krishnan et al., 2015; Karl et al., 2016). Recurrent State-Space Model (RSSM) (Hafner

et al., 2019) comprised of stochastic and deterministic components. VideoFlow (Kumar et al., 2019) predicted future values of the latent state by normalizing flows for robotic object interactions. Hierarchical latent models for video prediction were proposed in CWVAEs (Saxena et al., 2021) with levels ticking at different intervals in time. (Franceschi et al., 2020; Donà et al., 2021) disentangled dynamic and static factors where 2-5 initial observations was used to estimate the static component.

RL for Visual Control: Deep Reinforcement Learning (DRL) methods fall into one of two categories: 1) *Model-Based* — where an explicit model of the environment and its dynamics are learned (Ha and Schmidhuber, 2018; Hafner et al., 2019; 2020; 2021), and 2) *Model-Free* — where a policy is learned directly from the raw observations (Srinivas et al., 2020; Kostrikov et al., 2020b; Lillicrap et al., 2015; Yarats et al., 2022; Schwarzer et al., 2021). CURL (Srinivas et al., 2020) uses contrastive losses to learn discriminative representations. DrQ (Kostrikov et al., 2020a) and DrQ-v2 (Yarats et al., 2022) employed data augmentation techniques and do not use auxiliary losses or pre-training. RSSMs was introduced in PlaNet (Hafner et al., 2019) and was employed for online planning in the latent space. DreamerV1 (Hafner et al., 2020) and DreamerV2 (Hafner et al., 2021) achieved state-of-the-art results on DMC (Tassa et al., 2018) and Atari (Bellemare et al., 2013), respectively.

Sparsity in RNN: Neural networks have widely adopted sparsity to reduce the memory footprint of weights and activations (LeCun et al., 1990; Chen et al., 2015; Han et al., 2015). Several works have explored sparsity in RNNs. Campos et al. (2017) introduced a mechanism in RNNs that learns to skip state updates, effectively reducing the number of sequential operations on the latent state, thereby alleviating the problem of vanishing gradients in training on long sequences. Goyal et al. (2019) presented Recurrent Independent Mechanism (RIM), an architecture that consists of separate recurrent modules which are sparsely updated using a learned attention mechanism. In contrast to RIM, the number of updated state variables in VSG algorithm is not fixed.

6 Discussion

In this work, we introduce VSG and SVSG, two latent dynamics models leveraging sparse state updates. The sparse update prior was found to facilitate more efficient behaviors in tasks requiring long-horizon planning. Furthermore, SVSG is a purely stochastic model with a single component in the model state. We show that VSG and SVSG can outperform leading agents on the proposed BringBackShapes task, a challenging partially-observable and stochastic environment. BBS allows for controlling different factors of variation like stochasticity and partial-observability. Experiments conducted on various variations in BBS demonstrate that the proposed agents are more robust to noise in the environment and can better retain information of seen objects. Some limitations and potential research directions for future research are as follows:

- In the current implementation of VSG, the latent space does not exhibit disentanglement which could be an interesting direction for future research. Gating mechanisms in VSG can also be combined with other recurrent architectures like RIM (Goyal et al., 2019).
- In this work, BBS was explored with only 2 factors of variation: partial-observability and stochasticity. More controllable factors like the nature of entities (shape, size, color of objects), underlying physics (mass, friction, elasticity), or procedural background generation can be introduced to further study generalization capabilities of RL agents.
- SVSG being a purely stochastic model can further be used to estimate state uncertainty by marginalizing over multiple samples paths to efficiently explore in an unknown environment.
- A 3D version of the BBS environment would be interesting as the viewpoints may also change with rotation of agent and the underlying physics will make the task more challenging.
- We have used small latent dimensions and it would be interesting to train such models with larger architectures and on more complex tasks. Scaling the current architecture would also require optimizing the implementation to make them computationally feasible.
- Categorical latents outperformed Gaussian latents as the stochastic states of RSSM (Hafner et al., 2021), especially for discrete control tasks. However, SVSG was not found to work well with Categorical latents and we believe that sampled sparse states are hard to optimize.
- Model-based RL for visual control is still in early stages. However, a major challenge with deploying such models in the real world is safety especially during exploration. This would require an accurate world model that allows learning policies with stringent safety constraints that avoid mistakes when deployed in the real world. Such algorithms will rely on models that are robust when transferred from simulation to the real world.

Acknowledgements

The authors would like to thank David Meger and Ankesh Anand for their valuable feedback and discussions. The text also benefited from discussions with Abhinav Agarwalla, Rupali Bhati and Vineet Jain. The authors are also grateful to CIFAR for funding and the Digital Research Alliance of Canada for computing resources.

References

Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. Stochastic variational video prediction. *arXiv preprint arXiv:1710.11252*, 2017.

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Víctor Campos, Brendan Jou, Xavier Giró-i Nieto, Jordi Torres, and Shih-Fu Chang. Skip rnn: Learning to skip state updates in recurrent neural networks. *arXiv preprint arXiv:1708.06834*, 2017.

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294. PMLR, 2015.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31, 2018.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Fei Deng, Ingook Jang, and Sungjin Ahn. Dreamerpro: Reconstruction-free model-based reinforcement learning with prototypical representations. *arXiv preprint arXiv:2110.14565*, 2021.

Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. *arXiv preprint arXiv:1711.10604*, 2017.

Jérémie Donà, Jean-Yves Franceschi, Sylvain Lamprier, and Patrick Gallinari. {PDE}-driven spatiotemporal disentanglement. In *International Conference on Learning Representations*, 2021.

Jean-Yves Franceschi, Edouard Delasalles, Mickaël Chen, Sylvain Lamprier, and Patrick Gallinari. Stochastic latent residual video prediction. In *International Conference on Machine Learning*, pages 3233–3246. PMLR, 2020.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/hafner19a.html.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1lOTC4tDS.

Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0oabwyZbOu.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020a.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020b.

Rahul G Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. Videoflow: A flow-based generative model for video. *arXiv preprint arXiv:1903.01434*, 2(5), 2019.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. *arXiv preprint arXiv:1610.09513*, 2016.

Tung D Nguyen, Rui Shu, Tuan Pham, Hung Bui, and Stefano Ermon. Temporal predictive coding for model-based planning in latent space. In *International Conference on Machine Learning*, pages 8130–8139. PMLR, 2021a.

Tung D Nguyen, Rui Shu, Tuan Pham, Hung Bui, and Stefano Ermon. Temporal predictive coding for model-based planning in latent space. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8130–8139. PMLR, 18–24 Jul 2021b. URL https://proceedings.mlr.press/v139/nguyen21h.html.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR. URL https://proceedings.mlr.press/v32/rezende14.html.

Reuven Y Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

Vaibhav Saxena, Jimmy Ba, and Danijar Hafner. Clockwork variational autoencoders. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=fU7-so5RRhW.

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uCQfPZwRaUu.

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=_SJ-_yyes8.

Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=-2FCwDKRREu.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Described in the section with limitations.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The code for the model and dataset is in the supplementary material. There is a README file with the instructions to run them.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We have specified the hyperparameters in Appendix C.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All the results and plots presented in Section 4 and Appendix were obtained after training on multiple seeds ranging from 3-5.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We have mentioned about the GPUs and time taken to run on a single seed in the implementation details for each environment.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] We used the implementation of DreamerV2, DreamerPro and DrQ-v2 provided by the authors with MIT license. For comparison with DBC, we thank the authors for sharing the evaluation logs.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The code for the new environment (BBS) is included in the supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A BringBackShapes

The environment has a circular blue agent which can move in any direction. The shapes and colors of the objects are uniformly sampled from a predefined set with 5 shapes and 5 colors, respectively. As there are 5 objects in the arena for each episode, there are $25^5 \sim 9.8$M possible combinations. The initial positions of the agent and objects are randomly chosen within the arena. The elasticity of the objects and the agent is 1.0, while the walls have an elasticity of 0.7. There is a damping factor of 0.3 applied to the velocities of all objects and the agent. In Figure 3 [a], we show a full view of the whole arena at the beginning of an episode, and the gray region around the agent is its view. It can be observed that the agent might see none or all of the objects in its view and needs to explore in the environment to look for the objects in order to push them towards the goal (green region in Figure 3 [a]). The agent receives image observations of size $64 \times 64 \times 3$ from the environment. The action space is continuous and comprises of the angle and magnitude of force applied by the agent. The rewards are sparse and agent receives a reward of +1 for successfully pushing an object into the goal area. An episode terminates once all objects are pushed into the goal or if 3000 steps are completed. In this work, agents were trained for 2.5M environment steps and an action repeat of 4 was used. Evaluation was performed across 5 seeds with 10 episodes for each seed, and means and standard deviations across the seeds are reported.

Our motivation behind creatng BBS was to have a simple benchmark where the factors of variation can be controlled. For instance, in the current version, we add stochasticity and partial-observability. We believe future work can test for generalization to differences in the controllable factors between training and testing. These factors of variation specify the context of the MDP formalism of the task that the agent is trying to solve. If we have control over varying the context, we can define the training and the test distributions and this can enable us to formalize the class of generalisation problems we are focusing on. Furthermore, the environment can also be extended for open-ended learning where the agent has to learn an ever-increasing set of behaviours and abilities. Lastly, this can be further extended to multi-agent setting where the behaviour we expect to see is the emergence of some kind of cooperation between the two agents.

B Behavior Learning

The policy is trained by generating trajectories in the latent space obtained from the learned world model. The policy comprises of a stochastic actor and a deterministic critic to learn behaviours in the latent space. The actor learns to choose the most optimal actions conditioned on the model state ($\hat{a}_t \sim p_\psi(\hat{a}_t \mid \hat{s}_t)$). The critic estimates the discounted sum of future rewards that are beyond the planning horizon ($v_\xi(\hat{s}_t) \approx \mathrm{E}_{p_\phi, p_\psi}\left[\sum_{\tau \geq t} \hat{\gamma}^{\tau-t} \hat{r}_\tau\right]$). To obtain the latent trajectories, the initial model state is extracted from the collected data. The actor network provides the action $\hat{a}_t$, which is used to obtain the prior states $\hat{z}_t$ at each step. Since the agent does not act using these actions in the environment, the prior distributions are used to sample the state and reward predictor provides the reward $\hat{r}_t$. Furthermore, the value network provides the discounted sum of future rewards from that state. The actor and critic optimize different objectives:

Critic Loss: Temporal Difference learning is used to update the parameters of the critic. The target is estimated by combining the predicted rewards from latent model states and value estimates from critic. The weighted average of n-step returns (V_λ) proposed in DreamerV1 (Hafner et al., 2020) is used. The critic parameters (ξ) are optimized using the mean-squared error (MSE) between the predicted value and the λ-target over all the states in a trajectory, given by:

$$\mathcal{L}(\xi) \doteq \mathrm{E}_{p_\phi, p_\psi}\left[\frac{1}{H-1} \sum_{t=1}^{H-1} \frac{1}{2}\left(v_\xi(\hat{s}_t) - \mathrm{sg}(V_t^\lambda)\right)^2\right], \tag{7}$$

where sg denotes stopping gradients at the target while updating the critic, and H denotes the length of the planning horizon in latent space which was kept to 15 in our experiments. Furthermore, the targets are computed using a copy of the critic which is updated after every 100 gradient steps.

Actor Loss: The actor is trained to maximize the λ-return computed for training the critic. The reparameterization trick (Hafner et al., 2020; Kingma and Dhariwal, 2018; Rezende et al., 2014) was

used to backpropagate gradients from the value estimate. The entropy of the actor distribution is also regularized to encourage exploration. For training, $\eta_d = 1.0$ and the entropy regularizer $\eta_e = 10^{-4}$ was used. The loss for training the actor parameters (ϕ) is given by:

$$\mathcal{L}(\psi) \doteq \mathrm{E}_{p_\phi, p_\psi}\left[\frac{1}{H-1}\sum_{t=1}^{H-1}\Big(\underbrace{-\eta_d V_t^\lambda}_{\substack{\text{dynamics}\\ \text{backprop}}}\ \underbrace{-\eta_e \,\mathrm{H}[a_t|\hat{s}_t]}_{\text{entropy regularizer}}\Big)\right]. \tag{8}$$

C Hyper Parameters

Name	VSG	SVSG
World Model		
Batch Size	16	16
Sequence Length	50	50
Recurrent state dimensions	1024	1024
Image Representation num classes	32	-
Image Representation class dimension	32	-
KL Loss Scale	1.0	1.0
KL Balancing	0.8	0.8
Sparsity Loss Scale	0.1	0.1
Prior gate probability κ	0.3 / 0.4	0.3 / 0.4
World Model learning rate	3×10^{-4}	8×10^{-4}
Reward transformation	Identity	Identity
Behavior		
Imagination Horizon	15	15
Discount	0.99	0.99
λ-target parameter	0.95	0.95
Actor Gradient Mixing	0.1	0.1
Actor Entropy Loss Scale	1×10^{-4} / 2×10^{-3}	1×10^{-4} / 2×10^{-3}
Actor Learning Rate	8×10^{-5}	8×10^{-5}
Critic Learning Rate	8×10^{-5}	8×10^{-5}
Slow critic update inverval	100	100
Common		
Environment steps per update	5	5
MLP number of layers	4	4
MLP number of units	400	400
Gradient clipping	100	100
Adam epsilon	1×10^{-5}	1×10^{-5}
Weight decay	1×10^{-6}	1×10^{-6}
Total Parameters	32.3M	30.8M

Table 2: Hyper parameters of VSG and SVSG. When parameters are separated by /, the left hand side value is for BBS and the right hand side value is for other environments. When tuning the agent for a new task, we recommend searching over the KL loss scale $\beta \in \{0.1, 0.3, 1, 3\}$, prior gate probability $\kappa \in \{0.3, 0.4, 0.5\}$ and the discount factor $\gamma \in \{0.99, 0.999\}$.

D Scores on Deepmind Control Suite

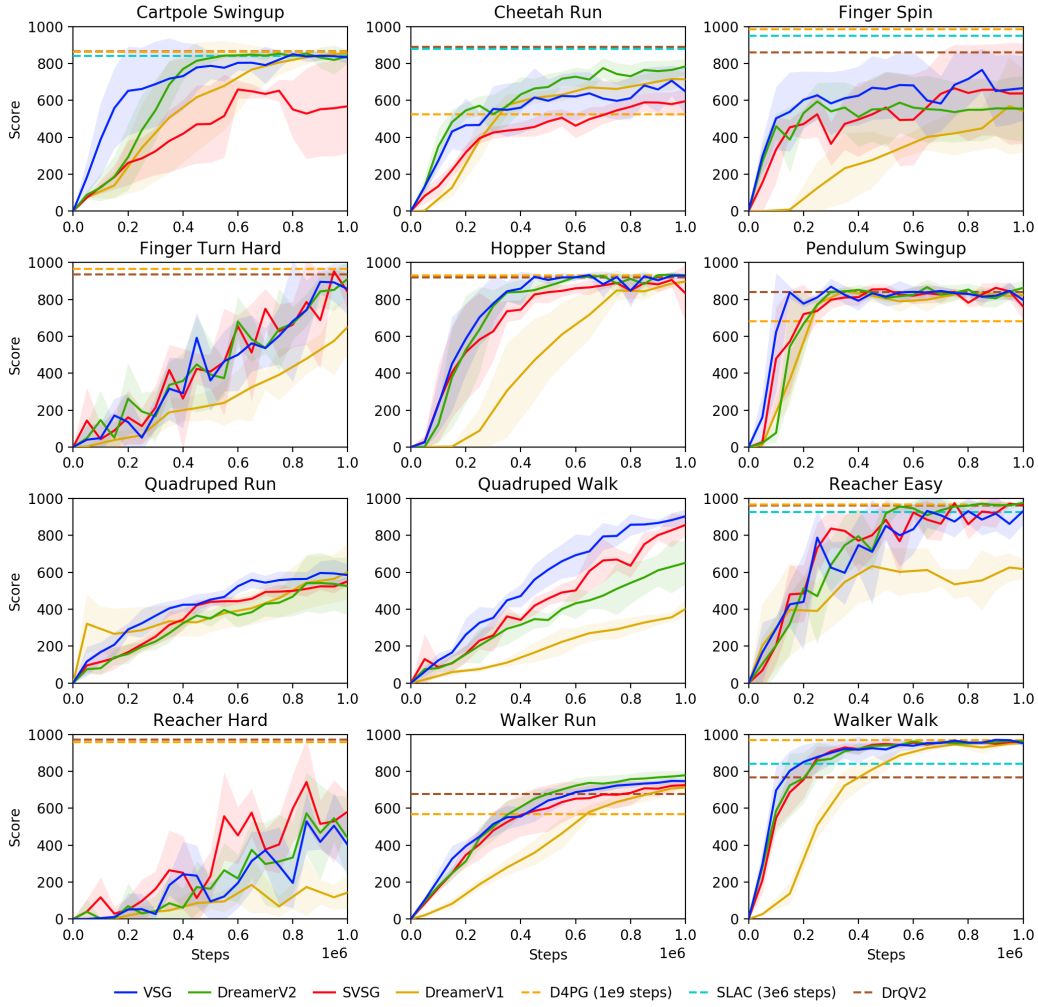


Figure 6: Comparison of VSG and SVSG with leading algorithms like DreamerV1 (Hafner et al., 2020), DreamerV2 (Hafner et al., 2021) and DrQ-V2 (Yarats et al., 2022) on tasks from the DeepMind Control Suite.

In Fig. 6, we present the scores on 12 tasks from DMControl Suite. VSG was found to perform better on 4 tasks and was competitive on 7 tasks when compared with DreamerV2. Furthermore, SVSG when compared with DreamerV1 which also used Gaussian latents, was found to perform better on 8 tasks and was competitive on 2 tasks. This demonstrates that using Gaussian latents with a single path and sparse gating mechanism can achieve competitive results when compared to leading methods and is better than previous methods using Gaussian latents.

E Ablations Studies

In this section, we present ablation experiments on the DeepMind Control Suite (Tassa et al., 2018).

E.1 Sparsity Loss in VSG

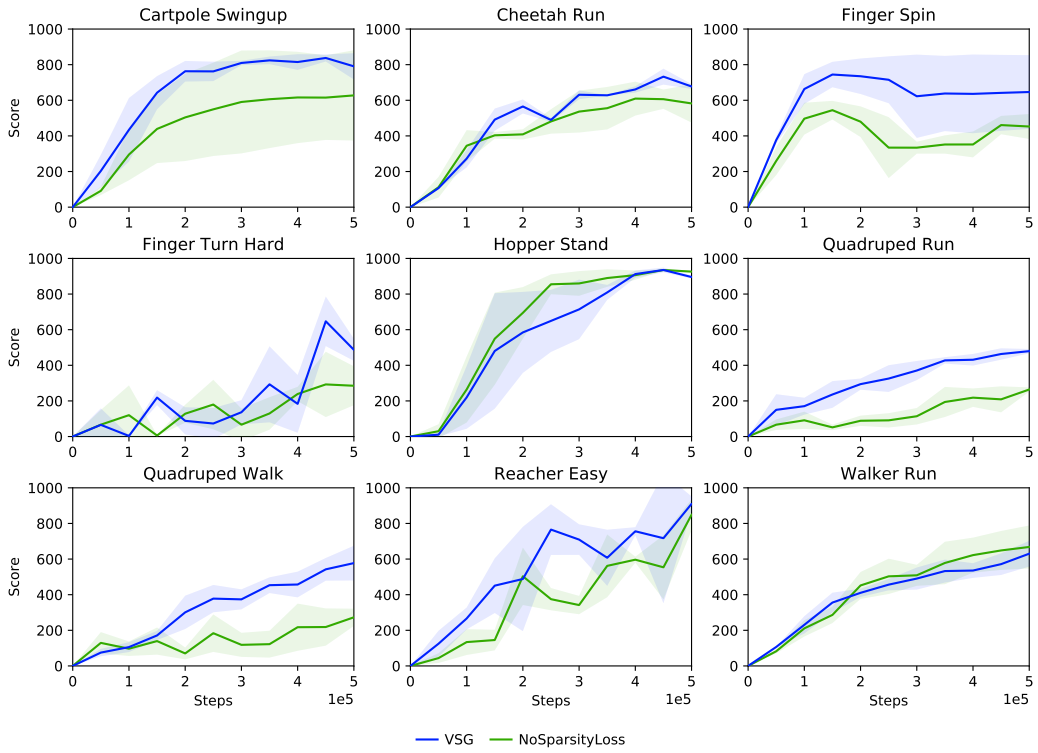


Figure 7: Ablation study showing the performance of VSG on 9 tasks tasks from DMC trained with (*VSG*) and without the sparsity penalty (*NoSparsityLoss*). VSG without the sparsity loss on update gate probabilities was found to significantly underperform on 5 out of 9 tasks from DM Control Suite.

E.2 Sparsity Loss and KL Mask in SVSG

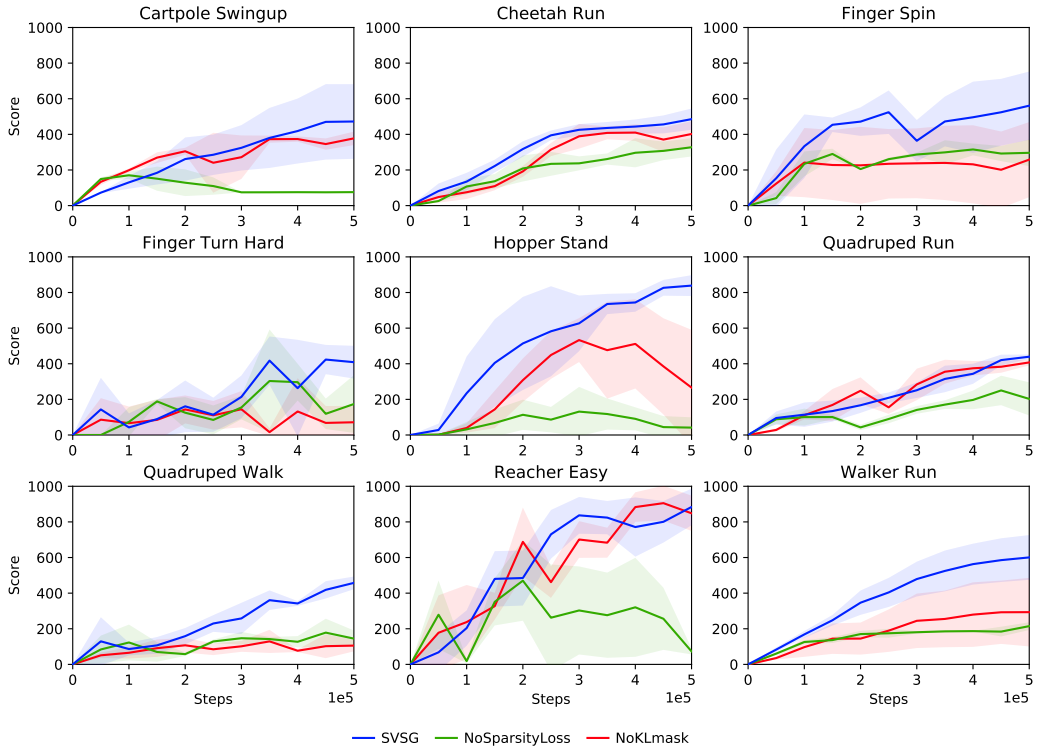


Figure 8: Ablation study comparing the performance of different SVSG models on DMC. We compare training with both KL masking and sparsity penalty (*SVSG*), with only sparsity penalty (*NoKLmask*), and with only KL masking (*NoSparsityLoss*).

F DMC with Natural Background

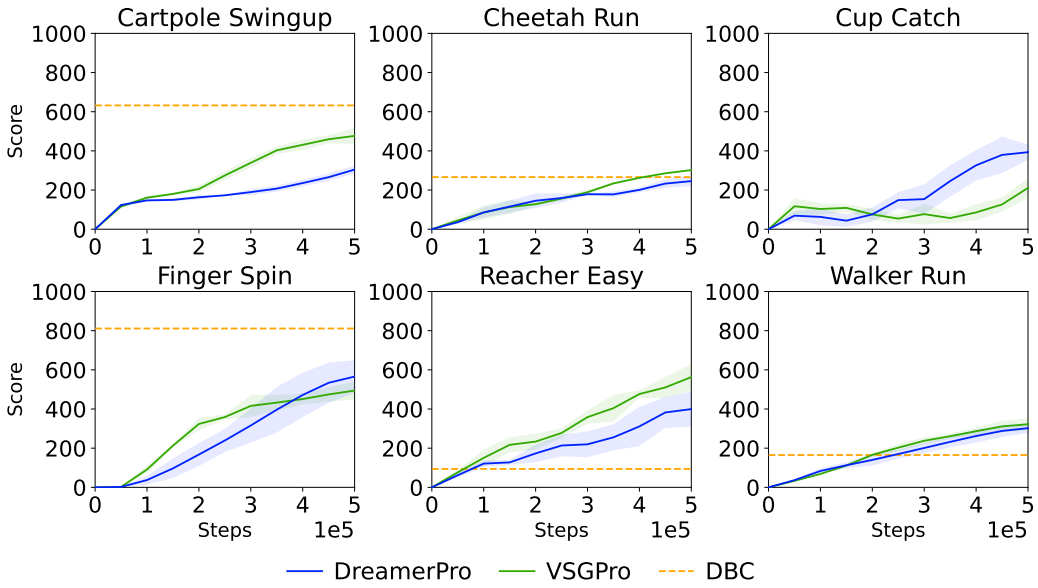


Figure 9: Results on DMC with Natural background setting.

To evaluate the efficacy of sparse gating mechanism in another setting with noise, we also experimented on DMC with noisy background (Zhang et al., 2021; Nguyen et al., 2021a). Reconstruction free model-based RL have been found to perform better on tasks with distractive backgrounds as they don't have reconstructive loss to generate the noisy frames. We updated the RSSM in DreamerPro (Deng et al., 2021) with VSG and call it VSGPro. VSGPro was found to work better or similar to DreamerPro. We also compare with DBC (Zhang et al., 2021), which uses bisimulation metrics to learn efficient encoders that can filter noise and focus on task relevant details. Upon evaluation at 500K environment steps, VSGPro performs better on 3 tasks and comparable on 2 tasks. Furthermore, VSGPro was found to perform similar to or better than DreamerPro. This demonstrates that the proposed gating mechanism also helps to learn efficient representations in settings with background noise.

G Atari

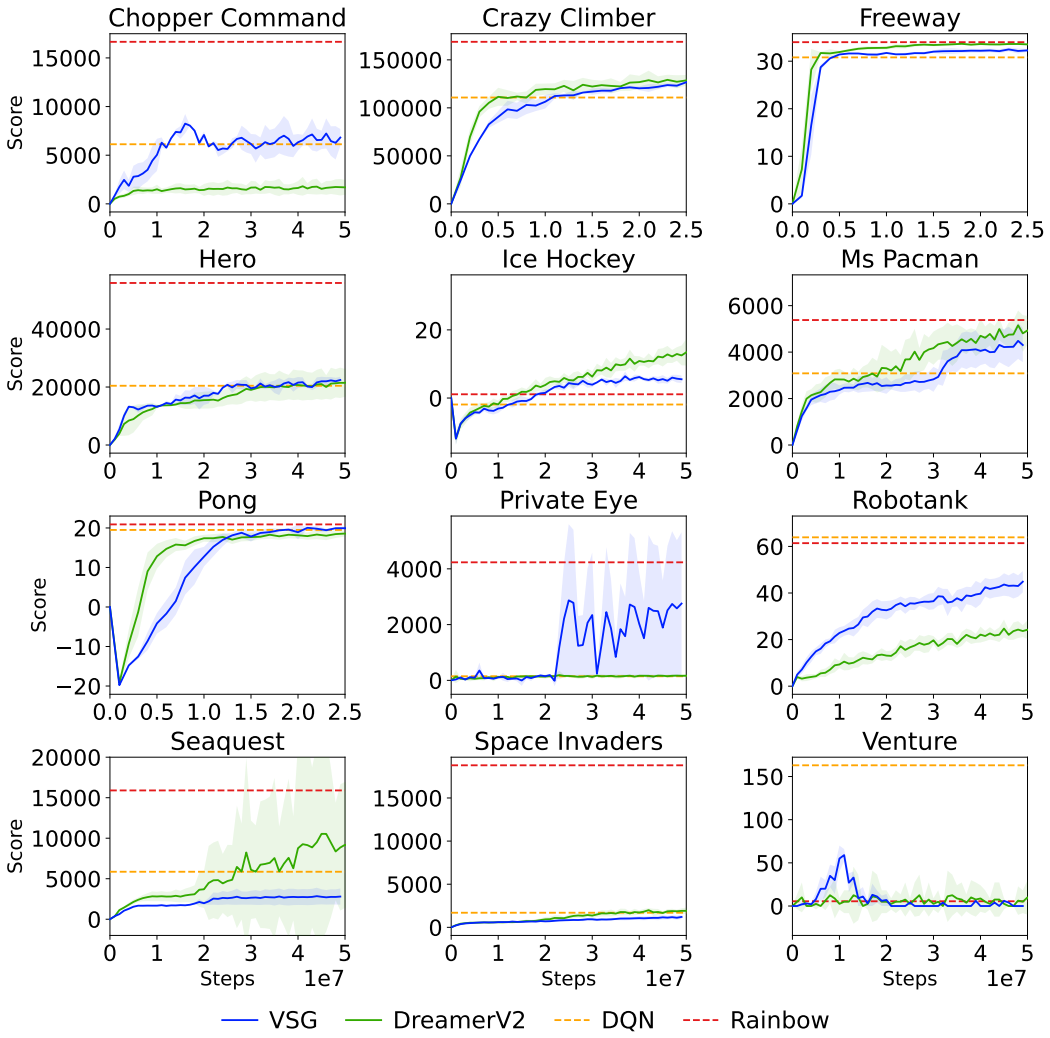


Figure 10: Results on a few tasks from the Atari benchmark trained for upto 50M environment steps.

In this work, we also present results on 12 tasks from the Atari benchmark (Bellemare et al., 2013). We trained the models for upto 50M environment steps which took around 2 days on a single NVIDIA A100 GPU for each seed. We also present results of Rainbow and DQN which were trained for 200M environment steps. For this experiment, we used the hyperparameters mentioned in the DreamerV2 paper, and use the same parameters for the gating mechanism as mentioned in Table 2. It can be observed that on most tasks, the proposed method VSG performs similar to DreamerV2. However, we observe performance gains on Chopper Command and Robotank, and we believe that this was because in those games the viewpoint of the agent changes with movement. For example, in Robotank environment, the agent can rotate around to search for enemy tanks to shoot. VSG was performing worse than the baseline on stochastic environments- Seaquest and Ms Pacman. Also, DreamerV2 was not performing well on the Private Eye environment. However, VSG was able to learn to solve the task for a few seeds as the environment is partially-observable and agent has to enter and exit different parts of the game. We ran with 3 more seeds and a similar trend was observed where some of the seeds were failing, whereas on a few of them the model learned to solve the task.

H Comparison with Stochastic State-Space Models (SSM)

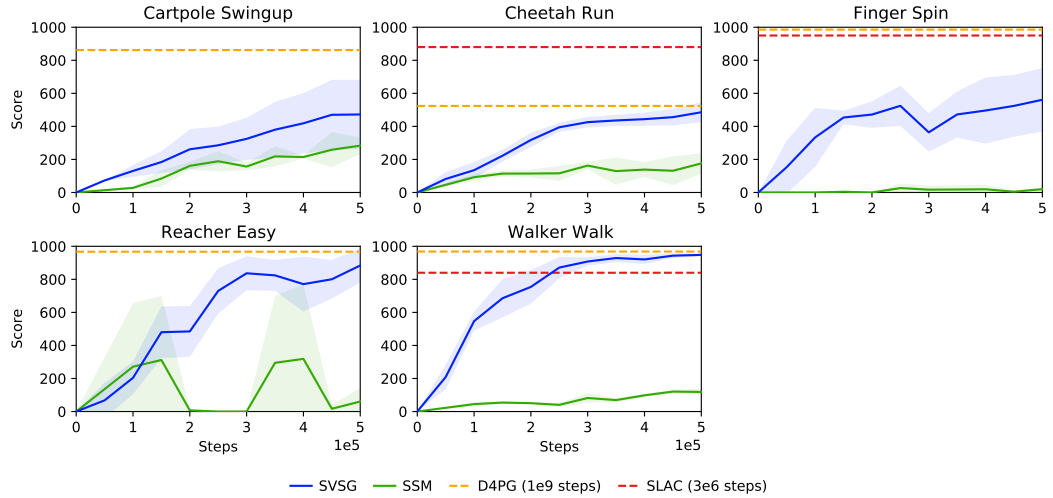


Figure 11: Comparison of SVSG and SSMs on a few tasks from the DMC trained for 500K environment steps.

Stochastic State-Space Models (SSMs) were discussed in PlaNet (Hafner et al., 2019) where the authors showed that SSMs do not achieve comparable performance when compared with RSSMs. We have shown that SVSG with a purely stochastic path can achieve comparable performance and outperform RSSMs on continuous control tasks with partial-observability and stochasticity. We also compare with SSMs as a baseline with a pure stochastic path only. In PLaNet, Cross Entropy Method (CEM) (Rubinstein, 1997; Chua et al., 2018) was used for planning. Since Dreamer agents improve upon PLaNet by having actor-critic network with learnable parameters in the policy and having KL-balancing in the training objective, we also implemented SSMs with those modifications. However, SSMs with those modifications were not found to work well as the actor was diverging. We also tried increasing the size of the stochastic state to larger values as it is 30 in the original implementation. We believe that sparse update prior is enabling the SVSG model to have large state sizes. Thus, we use the original implementation of SSMs from PLaNet for comparison. We experimented with a few tasks from the DMControl Suite and used 3 seeds for each task. As discussed earlier, SVSG was found to significantly outperform SSMs on all the tasks.

I Comparison of Architectures

I.1 Recurrent State-Space Model (RSSM)

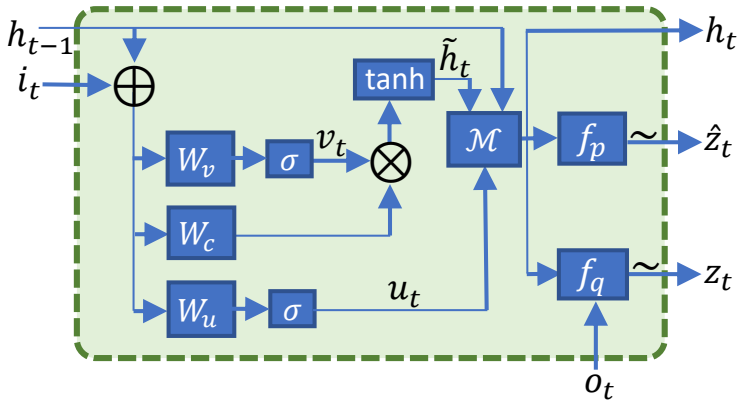


Figure 12: Architecture of Recurrent State-Space Model (RSSM). σ and $\tanh$ denote the sigmoid and hyperbolic tangent non-linear activation, respectively. W_* and b_* are the corresponding weights and biases. $\sim$, $\oplus$ and $\otimes$ denote sampling, vector concatenation, and element-wise multiplication, respectively. $\mathcal{M}$ computes $h_t = u_t\tilde{h}_t + (1-u_t)h_{t-1}$. f_p and f_q denote the prior and posterior distributions with learned parameters, respectively.

The Recurrent State-Space Model (RSSM) comprises of a recurrent path and an image representation path. Similar to VSG, the input i_t to the recurrent model contains information about the action and is obtained by concatenating the previous image representation state z_{t-1} and action a_t followed by passing them through through a MLP layer. RSSM uses a Gated Recurrent Unit (GRU) (Cho et al., 2014) for the recurrent state where the module has two gates- reset and update. The reset gate v_t controls the flow of information from the previous state and input, and the update gate u_t controls the extent of update of the recurrent state. Unlike VSG, the update gate in RSSM is not binary and can have values between 0 and 1. The equations are as follows:

$$
\begin{aligned}
v_t &= \sigma(W_v^T[h_{t-1}, i_t] + b_v) \\
u_t &= \sigma(W_u^T[h_{t-1}, i_t] + b_u) \\
\tilde{h}_t &= \tanh(v_t * (W_c^T[h_{t-1}, i_t] + b_c)) \\
h_t &= u_t \odot \tilde{h}_t + (1-u_t) \odot h_{t-1},
\end{aligned}
\tag{9}
$$

where $\odot$ denotes element-wise multiplication, σ and $\tanh$ are the sigmoid and hyperbolic tangent non-linear activation, and W_* and b_* denotes the weights and biases, respectively. The recurrent state is further used to obtain the posterior z_t and prior states $\hat{z}_t$ by passing it through MLP layers with and without observation o_t, respectively.

I.2 Variational Sparse Gating (VSG)

We present the zoomed in architecture of Variational Sparse Gating (VSG). For details, refer to Section 2 in the main paper.

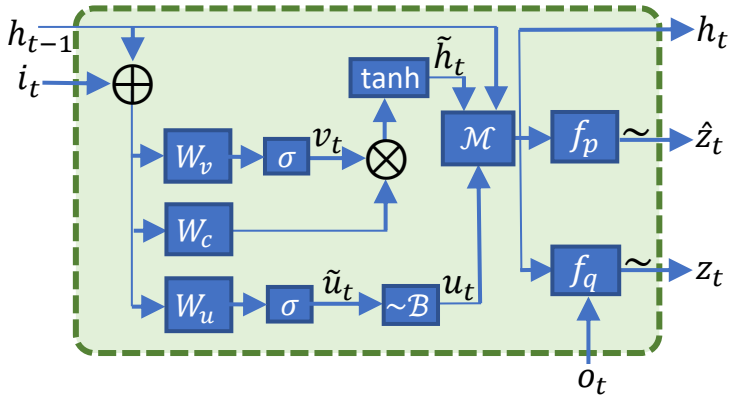


Figure 13: Architecture of Variational Sparse Gating (VSG). σ and $\tanh$ denote the sigmoid and tanh non-linear activations, respectively. W_* and b_* are the corresponding weights and biases. $\sim$, $\oplus$ and $\otimes$ denote sampling, vector concatenation, and element-wise multiplication, respectively. $\mathcal{M}$ computes $h_t = u_t\tilde{h}_t + (1-u_t)h_{t-1}$. $\mathcal{B}$ denotes Bernoulli distribution. f_p and f_q denote the prior and posterior distributions with learned parameters, respectively.

I.3 Simple Variational Sparse Gating (SVSG)

We present the zoomed in architecture of Simple Variational Sparse Gating (SVSG). For details, refer to Section 3 in the main paper.

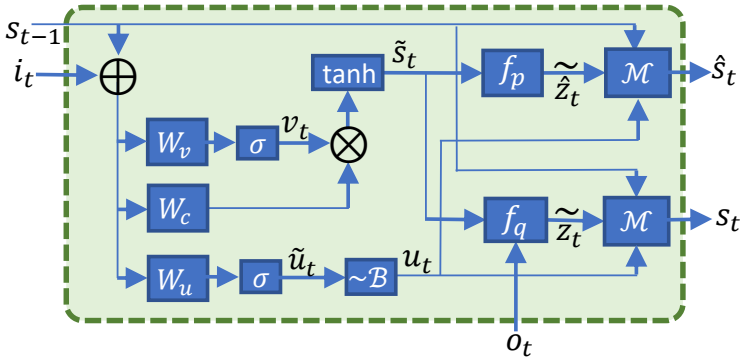


Figure 14: Architecture of Simple Variational Sparse Gating (SVSG). σ and $\tanh$ denote the sigmoid and tanh non-linear activations, respectively. W_* and b_* are the corresponding weights and biases. $\sim$, $\oplus$ and $\otimes$ denote sampling, vector concatenation, and element-wise multiplication, respectively. $\mathcal{M}$ computes $s_t = u_t\tilde{s}_t + (1-u_t)s_{t-1}$. $\mathcal{B}$ denotes Bernoulli distribution. f_p and f_q denote the prior and posterior distributions with learned parameters, respectively.

J Latent Transition

In this section, we give the learned world model of corresponding agents the first 15 frames and ask it to imagine 5 different rollouts in the latent space for the next 35 frames. The sequence of actions is kept fixed across imagined trajectories. The model state from imagined trajectories was passed through the decoder to reconstruct the frames. It can be observed from the GIFs of VSG and SVSG that the model is able to remember the color and location of objects, and is also cognizant about the goal location and walls. Furthermore, for GIFs of DreamerV2, it can be observed that there is a distortion in the shapes and the model also modifies the color of the objects after a few timesteps. This also means that the proposed mechanism is helping the model to retain information for longer time steps.

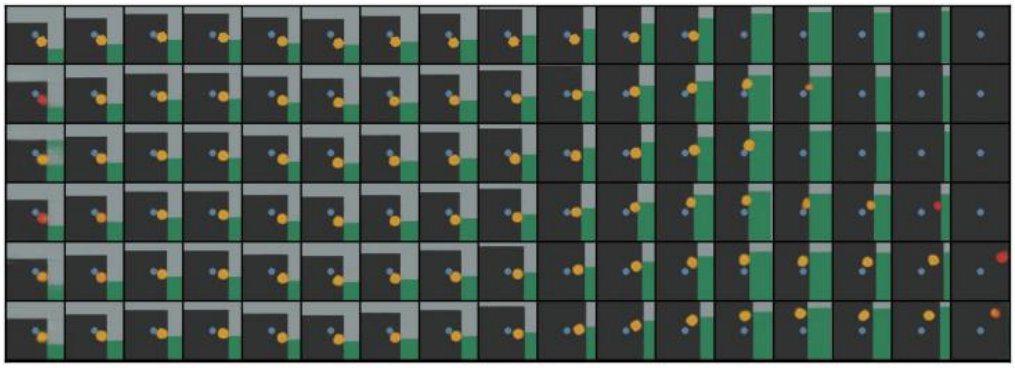


Figure 15: Imagined trajectories of an evaluation episode using learned DreamerV2 agent. The top row is the ground truth and the next 5 rows are different roll outs given the same first 15 frames and action sequence. In this figure, we can observe that the model changes the color of the object towards the end of the episode.

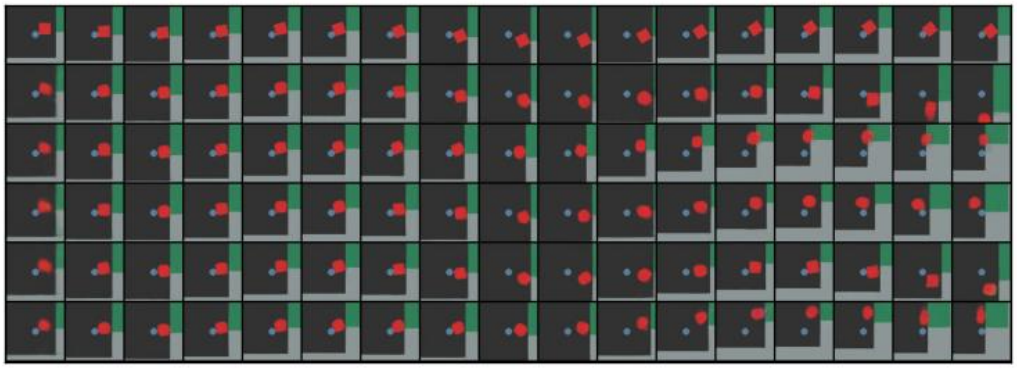


Figure 16: Imagined trajectories of an evaluation episode using learned DreamerV2 agent. The top row is the ground truth and the next 5 rows are different rollouts given the same first 15 frames and action sequence. It can be observed that the agent is distorting the shape of the block towards the end of the episode. For example, the last frame in the last row shows an oval object although the agent retains the red color.

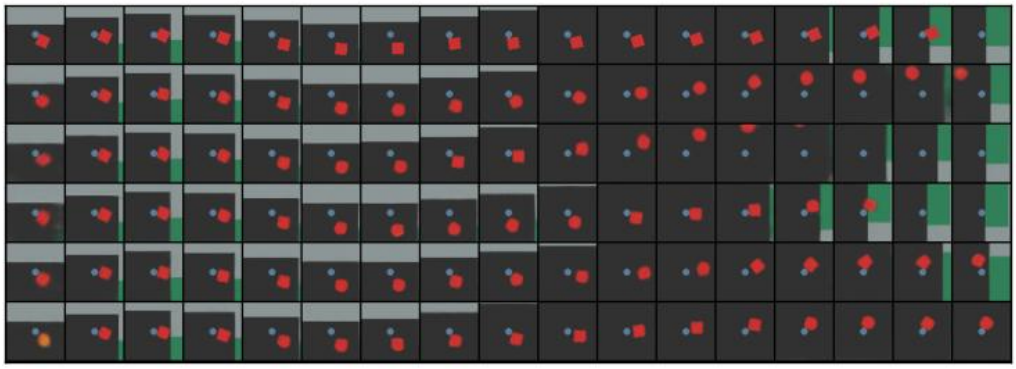


Figure 17: Imagined trajectories of an evaluation episode using learned VSG agent. The top row is the ground truth and the next 5 rows are different roll outs given the same first 15 frames and action sequence. Here, we observe that the agent is able to retain the color and shape of red block, and also reaches the final position which is close to the goal on 4 trajectories.

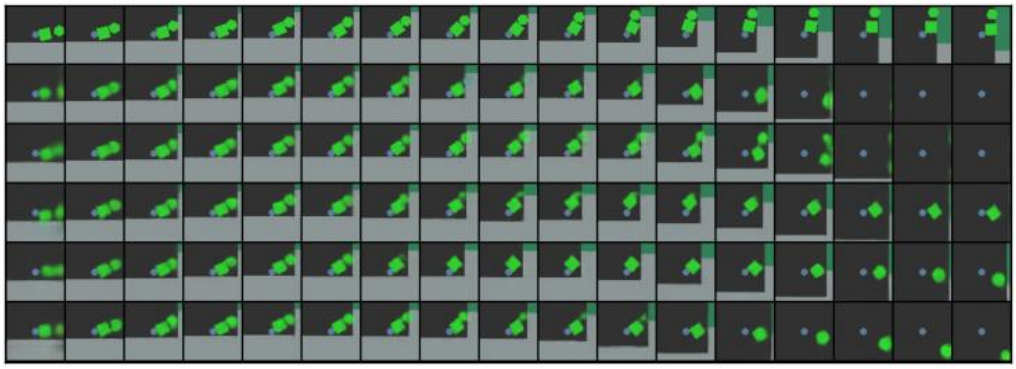


Figure 18: Imagined trajectories of an evaluation episode using learned VSG agent. The top row is the ground truth and the next 5 rows are different roll outs given the same first 15 frames and action sequence. This shows a failure case of VSG where there are 2 objects of the same color and the model merge them to a single shape.

Figure 19: Imagined trajectories of an evaluation episode using learned SVSG agent. The top row is the ground truth and the next 5 rows are different roll outs given the same first 15 frames and action sequence. We can see that SVSG is also able to retain the shape and color of objects.

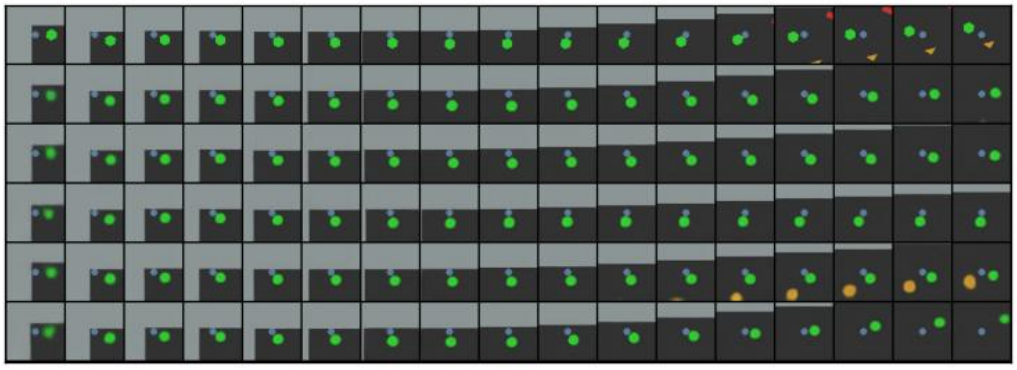


Figure 20: Imagined trajectories of an evaluation episode using learned SVSG agent. The top row is the ground truth and the next 5 rows are different roll outs given the same first 15 frames and action sequence.