Dropout's Dream Land: Generalization from Learned Simulators to Reality

Zac Wellmer and James T. Kwok

Department of Computer Science and Engineering
Hong Kong University of Science and Technology, Hong Kong
{zwwellmer,jamesk}@cse.ust.hk

Abstract. A World Model is a generative model used to simulate an environment. World Models have proven capable of learning spatial and temporal representations of Reinforcement Learning environments. In some cases, a World Model offers an agent the opportunity to learn entirely inside of its own dream environment. In this work we explore improving the generalization capabilities from dream environments to real environments (Dream2Real). We present a general approach to improve a controller's ability to transfer from a neural network dream environment to reality at little additional cost. These improvements are gained by drawing on inspiration from Domain Randomization, where the basic idea is to randomize as much of a simulator as possible without fundamentally changing the task at hand. Generally, Domain Randomization assumes access to a pre-built simulator with configurable parameters but oftentimes this is not available. By training the World Model using dropout, the dream environment is capable of creating a nearly infinite number of *different* dream environments. Previous use cases of dropout either do not use dropout at inference time or averages the predictions generated by multiple sampled masks (Monte-Carlo Dropout). Dropout's Dream Land leverages each unique mask to create a diverse set of dream environments. Our experimental results show that Dropout's Dream Land is an effective technique to bridge the reality gap between dream environments and reality. Furthermore, we additionally perform an extensive set of ablation studies.[1]

1 Introduction

Reinforcement learning [30] (RL) has experienced a flurry of success in recent years, from learning to play Atari [20] to achieving grandmaster-level performance in StarCraft II [32]. However, in all these examples, the target environment is a simulator that can be directly trained in. Reinforcement learning is often not a practical solution without a simulator of the environment.

Sometimes the target environment is expensive, dangerous, or even impossible to interact with. In these cases, the agent is trained in a simulated source environment. Approaches that train an agent in a simulated environment with

[1] The code is available at https://github.com/zacwellmer/DropoutsDreamLand

the hopes of generalization to the target environment experience a common problem referred to as the *reality gap* [13]. One approach to bridge the reality gap is Domain Randomization [31]. The basic idea is that an agent which can perform well in an ensemble of simulations will also generalize to the real environment [2, 21, 24, 31]. The ensemble of simulations is generally created by randomizing as much of the simulator as possible without fundamentally changing the task at hand. Unfortunately, this approach is only applicable when a simulator is provided and the simulator is configurable.

A recently growing field, World Models [9], focuses on the side of this problem when the simulation does not exist. World Models offer a general framework for optimizing controllers directly in *learned* simulated environments. The learned dynamics model can be viewed as the agent's dream environment. This is an interesting area because access to a learned dynamics model removes the need for an agent to train in the target environment. Some related approaches [10, 11, 15, 19, 25, 29] focus on an adjacent problem which allows the controller to continually interact with the target environment.

Despite the recent improvements [10,11,15,16,25] of World Models, little has been done to address the issue that World Models are susceptible to the reality gap. The learned dream environment can be viewed as the source domain and the true environment as the target domain. Whenever there are discrepancies between the source and target domains the reality gap can cause problems. Even though World Models suffer from the reality gap, none of the Domain Randomization approaches are directly applicable because the dream environment does not have easily configurable parameters.

In this work we present Dropout's Dream Land (DDL), a simple approach to bridge the reality gap from learned dream environments to reality (Dream2Real). Dropout's Dream Land was inspired by the first principles of domain randomization, namely, train a controller on a large set of *different* simulators which all adhere to the fundamental task of the target environment. We are able to generate a nearly infinite number of different simulators via the insight that dropout [27] can be understood as learning an ensemble of neural networks [3].

Our empirical results demonstrate that Dropout's Dream Land is an effective technique to cross the Dream2Real gap and offers improvements over baseline approaches [9,16]. Furthermore, we perform an extensive set of ablation studies which indicate the source of generalization improvements, requirements for the method to work, and when the method is most useful.

2 Related Works

2.1 Dropout

Dropout [27] was introduced as a regularization technique for feedforward and convolutional neural networks. In its most general form, each unit is dropped with a probability p during the training process. During training weights are scaled by $\frac{1}{1-p}$. Weight scaling ensures that for any hidden unit the *expected*

output is the same as the actual output at test time [27]. Recurrent neural networks (RNNs) initially had issues benefiting from dropout. Zaremba *et al.* [35] suggests not to apply dropout to the hidden state units of the RNN cell. Gal *et al.* [7] shortly after show that the mask can also be applied to the hidden state units, but the mask must be fixed across the sequence during training.

In this work, we follow the dropout approach from [7] when training the RNN. More formally, for each sequence, the Boolean masks $\mathbf{m}_{xi}$, $\mathbf{m}_{xf}$, $\mathbf{m}_{xw}$, $\mathbf{m}_{xo}$, $\mathbf{m}_{hi}$, $\mathbf{m}_{hf}$, $\mathbf{m}_{hw}$, and $\mathbf{m}_{ho}$ are sampled, then used in the following LSTM update:

$$\mathbf{i}_t = \mathbf{W}_{xi}(\mathbf{x}_t \odot \mathbf{m}_{xi}) + \mathbf{W}_{hi}(\mathbf{h}_{t-1} \odot \mathbf{m}_{hi}) + \mathbf{b}_i, \tag{1}$$

$$\mathbf{f}_t = \mathbf{W}_{xf}(\mathbf{x}_t \odot \mathbf{m}_{xf}) + \mathbf{W}_{hf}(\mathbf{h}_{t-1} \odot \mathbf{m}_{hf}) + \mathbf{b}_f, \tag{2}$$

$$\mathbf{w}_t = \mathbf{W}_{xw}(\mathbf{x}_t \odot \mathbf{m}_{xw}) + \mathbf{W}_{hw}(\mathbf{h}_{t-1} \odot \mathbf{m}_{hw}) + \mathbf{b}_w, \tag{3}$$

$$\mathbf{o}_t = \mathbf{W}_{xo}(\mathbf{x}_t \odot \mathbf{m}_{xo}) + \mathbf{W}_{ho}(\mathbf{h}_{t-1} \odot \mathbf{m}_{ho}) + \mathbf{b}_o, \tag{4}$$

$$\mathbf{c}_t = \sigma(\mathbf{i}_t) \odot \tanh(\mathbf{w}_t) + \sigma(\mathbf{f}_t) \odot \mathbf{c}_{t-1}, \tag{5}$$

$$\mathbf{h}_t = \sigma(\mathbf{o}_t) \odot \tanh(\mathbf{c}_t), \tag{6}$$

where $\mathbf{x}_t$, $\mathbf{h}_t$, and $\mathbf{c}_t$ are the input, hidden state, and cell state, respectively, $\mathbf{W}_{xi}$, $\mathbf{W}_{xf}$, $\mathbf{W}_{xw}$, $\mathbf{W}_{xo} \in \mathbb{R}^{d \times r}$ $\mathbf{W}_{hi}$, $\mathbf{W}_{hf}$, $\mathbf{W}_{hw}$, $\mathbf{W}_{ho} \in \mathbb{R}^{d \times d}$ are the LSTM weight matrices, and $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_w$, $\mathbf{b}_o \in \mathbb{R}^d$ are the LSTM biases. The masks are fixed for the entire sequence, but may differ between sequences in the mini-batch.

Monte-Carlo (MC) Dropout [6] runs multiple forward passes with independently sampled masks. In related works [14], Monte-Carlo (MC) Dropout [6] has been used to approximate the mean and variance of output predictions from an ensemble. We emphasize that Dropout's Dream Land does not use MC Dropout. Details are in Section 3.2.

2.2 Domain Randomization

The goal of Domain Randomization [24, 31] is to create many different versions of the dynamics model with the hope that a policy generalizing to all versions of the dynamics model will do well on the true environment. Figure 1 illustrates many simulated environments ($\hat{e}^j$) overlapping with the actual environment (e^*). Simulated environments are often far cheaper to operate in than the actual environment. Hence, it is desirable to be able to perform the majority of interactions in the simulated environments.

Randomization has been applied on observations (e.g., lighting, textures) to perform robotic grasping [31] and collision avoidance of drones [24]. Randomization has also proven useful when applied to the underlying dynamics of simulators [23]. Often, both the observations and simulation dynamics are randomized [1].

Domain randomization generally uses some pre-existing simulator which then injects randomness into specific aspects of the simulator (e.g., color textures, friction coefficients). Each of the simulated environments in Figure 1 can be thought of as a noisy sample of the pre-existing simulator. To the best of our

Algorithm 1 World Models: Training in dreams.

1: Initialize parameters of V, M, and C
2: Collect N trajectories $\mathbf{o}$, d, and $\mathbf{a}$ from e^*
3: Optimize V on observations $\mathbf{o}$
4: Generate embeddings $\mathbf{z}$ for $\mathbf{o}$ with V
5: Optimize M on $\mathbf{z}$ and d
6: Generate dream environment $\hat{e}$ from M
7: **for** iteration=1, 2, ... **do**
8: Optimize C via interactions with $\hat{e}$

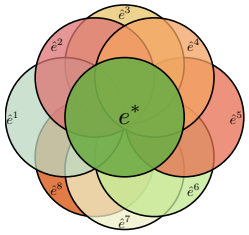


Fig. 1. e^* is the actual environment, and $\hat{e}^j$'s are randomized variants of the simulated environment.

knowledge, Domain Randomization has yet to be applied to entirely learned simulators.

2.3 World Models

The world model [9] has three modules trained separately: (i) vision module (V); (ii) dynamics module (M); and (iii) controller (C). A high-level view is shown in Algorithm 1. The vision module (V) is a variational autoencoder (VAE) [17], which maps an image observation ($\mathbf{o}$) to a lower-dimensional representation $\mathbf{z} \in \mathbb{R}^n$.

The dynamics model (M) is a mixture density network recurrent neural network (MDN-RNN) [8, 9]. The MDN-RNN models the dynamics of the environment, so modifying the parameters changes the dynamics of the learned simulated environment. It is implemented as an LSTM followed by a fully-connected layer outputting parameters for a Gaussian mixture model with k components. Each feature has k different π parameters for the logits of multinomial distribution, and (μ, σ) parameters for the k components in the Gaussian mixture. At each timestep, the MDN-RNN takes in the state $\mathbf{z}$ and action $\mathbf{a}$ as inputs and predicts $\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\sigma}$. To draw a sample from the MDN-RNN, we first sample the multinomial distribution parameterized by $\boldsymbol{\pi}$, which indexes which of the k normal distributions in the Gaussian mixture to sample from. This is then repeated for each of the n features. Depending on the experiments, Ha and Schmidhuber [9] also include an auxiliary head to the LSTM which predicts whether the episode terminates (d).

The controller (C) is responsible for deciding what actions to take. It takes features produced by the encoder V and dynamics model M as input (not the raw observations). The simple controller is a single-layer model which uses an evolutionary algorithm (CMA-ES [12]) to find its parameters. Depending on the problem setting, the controller (C) can either be optimized directly on the target environment (e^*) or on the dream environment ($\hat{e}$). This paper is focused on the case of optimizing exclusively in the dream environment.

3 Dropout's Dream Land

In this work we introduce Dropout's Dream Land (DDL). Dropout's Dream Land is the first work to offer a strategy to bridge the *reality gap* between learned neural network dynamics models and reality. Traditional Domain Randomization generates many *different* dynamics models by randomizing configurable parameters of a given simulation. This approach does not apply to neural network dynamics models because they generally do not have configurable parameters (such as textures and friction coefficients). In Dropout's Dream Land, the controller can interact with billions[2] of dream environments, whereas previous works [9, 16] only use one dream environment. A naive way to go about this would be to train a population of neural network world models. However, this would be computationally expensive.

To keep the computational cost low, we go about this by applying dropout to the dynamics model in order to form different dynamics models. Crucially, dropout is applied at **both** training and inference of the dynamics model M. Each unique dropout mask applied to M can be viewed as a different environment. Similar to the spirit of Domain Randomization, an agent is expected to perform well in the real environment if it can perform well in a variety of simulated environments.

3.1 Learning the Dream Environment

The Dropout's Dream Land environments are built around the dynamics model M. The controller interactions during training are described by Figure 2, in which $\hat{r}$, $\hat{d}$, and $\hat{\mathbf{z}}$ are generated entirely by M. In this work, M is an LSTM where $\mathbf{x} = [\mathbf{z}^\top, \mathbf{a}^\top]^\top$ from equations (1)-(4). The LSTM is followed by multiple heads for predictions of the latent state ($\hat{\mathbf{z}}$), reward ($\hat{r}$) and termination ($\hat{d}$). The reward and termination heads are simple fully-connected layers. Latent state prediction is done with a MDN-RNN [8, 9], but this could be replaced by any other neural network that supports dropout (e.g., GameGAN [16]).

Loss Function The dynamics model M jointly optimizes all three heads. The loss of a single transition is defined as:

$$\mathcal{L}^M = \mathcal{L}^z + \alpha_r \mathcal{L}^r + \alpha_d \mathcal{L}^d. \tag{7}$$

Here, $\mathcal{L}^z = -\sum_{i=1}^{n} \log(\sum_{j=1}^{k} \hat{\pi}_{i,j} \mathcal{N}(z_i | \hat{\mu}_{i,j}, \hat{\sigma}_{i,j}^2))$ is a mixture density loss for the latent state predictions, where n is the size of the latent feature vector z, $\hat{\pi}_{i,j}$ is the jth component's probability for the ith feature, $\hat{\mu}_{i,j}, \hat{\sigma}_{i,j}$ are the corresponding mean and standard deviation. $\mathcal{L}^r = (r - \hat{r})^2$ is the square loss on rewards, where r and $\hat{r}$ are the true and estimated rewards, respectively. $\mathcal{L}^d = -d \log(\hat{d}) - (1 - d) \log(1 - \hat{d})$ is the cross-entropy loss for termination

[2] In practice we are bounded by the total number of steps instead of every possible environment.

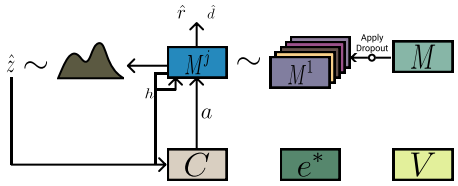


Fig. 2. Interactions with the dream environment. A dropout mask is sampled at every step yielding a new M^j.

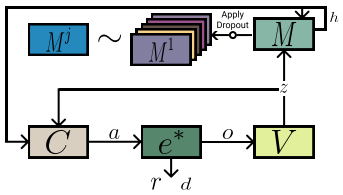


Fig. 3. Interactions with the real environment. The controller being optimized only interacts with the real environment during the final testing phase.

prediction, where d and $\hat{d}$ are the true and estimated probabilities of the episode ending, respectively. Constants α_d and α_r in (7) are for trading off importance of the termination and reward objectives. The loss ($\mathcal{L}^M$) is aggregated over each sequence and averaged across the mini-batch.

Training Dynamics Model M with Dropout At training time of M (Algorithm 1, Line 5), we apply dropout [7] to the LSTM to simulate different random environments. For each input and hidden unit, we first sample a Boolean indicator with probability p_{train}. If the indicator is 1, the corresponding input/hidden unit is masked. Masks $\mathbf{m}_{xi}$, $\mathbf{m}_{xf}$, $\mathbf{m}_{xw}$, $\mathbf{m}_{xo}$, $\mathbf{m}_{hi}$, $\mathbf{m}_{hf}$, $\mathbf{m}_{hw}$, and $\mathbf{m}_{ho}$ are sampled independently (Equations (1)-(4)). When training the RNN, each mini-batch contains multiple sequences. Each sequence uses an independently sampled dropout mask. We fix the dropout mask for the entire sequence as this was previously found to be critically important [7].

Training the RNN with many different dropout masks is critical in order to generate multiple different dynamics models. At the core of Domain Randomization is the requirement that the randomizations do not fundamentally change the task. This constraint is violated if we do not train the RNN with dropout but apply dropout at inference (explored further empirically in Section 4.3). After optimizing the dynamics model M, we can use it to construct dream environments (Section 3.2) for controller training (Section 3.3).

In this work, we never sample masks to apply to the action ($\mathbf{a}$). We do not zero out the action because in some environments this could imply the agent taking an action (e.g., moving to the left). This design choice could be changed depending on the environment, for example, when a zerod action corresponds to a no-op or a sticky action.

3.2 Interacting with Dropout's Dream Land

Interactions with the dream environment (Algorithm 1, Line 8) can be characterized as training time for the controller (C) and inference time of the dynamics model (M). An episode begins by generating the initial latent state vector $\hat{\mathbf{z}}$ by

either sampling from a standard normal distribution or sampling from the starting points of the observed trajectories used to train M [9]. The hidden cell ($\mathbf{c}$) and state ($\mathbf{h}$) vectors are initialized with zeros.

The controller (C) decides the action to take based on $\hat{\mathbf{z}}$ and $\mathbf{h}$. In Figure 2, the controller also observes $\hat{r}$ and $\hat{d}$, but these are exclusively used for the optimization process of the controller. The controller then performs an action $\mathbf{a}$ on a dream environment.

A new dropout mask is sampled (with probability p_{infer}) and applied to M. We refer to the masked dynamics model as M^j and the corresponding Dropout's Dream Land environment as $\hat{e}^j$. The current latent state $\hat{\mathbf{z}}$ and action $\mathbf{a}$ are concatenated, and passed to M^j to perform a forward pass. The episode terminates based on a sample from a Bernoulli distribution parameterized by $\hat{d}$. The dream environment then outputs the latent state, LSTM's hidden state, reward, and whether the episode terminates.

It is crucial to apply dropout at inference time (of the dynamics model M) in order to create *different* versions of the dream environment for the controller C. Our experiments (Sections 4.2 and 4.3) consider an extensive set of ablation studies as to how and when dropout should be applied.

Dropout's Dream Land is not Monte-Carlo Dropout The only work we are aware of that applies dropout at inference time is Monte-Carlo Dropout [7]. In Section 4.1 we include a Monte-Carlo Dropout World Model baseline because DDL can easily be misinterpreted as an application of Monte-Carlo Dropout. This baseline passes the expected hidden ($\tilde{\boldsymbol{h}}_t$) and cell ($\tilde{\boldsymbol{c}}_t$) state to the next time-step, in which the expectation is over dropout masks from Equations (1)-(4). In practice we follow a similar approach to previous work [7] and approximate the expectation by performing multiple forward passes (each forward pass samples a new dropout mask), and averages the results. At each step, the expected Mixture Model parameters ($\tilde{\boldsymbol{\pi}}$, $\tilde{\boldsymbol{\mu}}$, $\tilde{\boldsymbol{\sigma}}$), reward ($\tilde{r}$), and termination ($\tilde{d}$) are used. Maximizing expected returns from the Monte-Carlo Dropout World Model is equivalent to maximizing expected returns on a *single* dream environment, the average dynamics model. On the other hand, the purpose of DDL's approach to dropout is to generate many *different* versions of the dynamics model. More explicitly, the controller is trained to maximize expected returns across many different dynamics models in the ensemble, as opposed to maximizing expected returns on the ensemble average.

Dropout has also traditionally been used as a model regularizer. Dropout as a model regularizer is only applied at training time but not at inference time. In this work, this approach would regularize the dynamics model M. The usual trade-off is lower test loss at the cost of higher training loss [7, 27]. However, DDL's ultimate goal is not to lower test loss of the World Model (M). The ultimate goal is providing dream environments to a controller so that the optimal policy in Dropout's Dream Land also maximizes expected returns in the target environment (e^*).

3.3 Training the Controller

Training with CMA-ES We follow the same controller optimization procedure as was done in World Models [9] and GameGAN [16] on their DoomTakeCover experiments. We train the controller with CMA-ES [12]. At every generation CMA-ES [12] spawns a population (of size N_{pop}) of agents. Each agent in the population reports their mean returns on a set of N_{trials} episodes generated in the dream environments. As controllers in the population do not share a dream environment, the probability of controllers interacting with the same sequence of dropout masks is vanishingly small. Let $N_{\text{max\_ep\_len}}$ be the maximum number of steps in an episode. In a single CMA-ES iteration, the population as a whole can interact with $N_{\text{pop}} \times N_{\text{trials}} \times N_{\text{max\_ep\_len}}$ *different* environments. In our experiments, $N_{\text{pop}} = 64$, $N_{\text{trials}} = 16$, and $N_{\text{max\_ep\_len}}$ is 1000 for CarRacing and 2100 for DoomTakeCover. This potentially results in $> 1,000,000$ different environments at each generation.

Dream Leader Board After every fixed number of generations (25 in our experiments), the best controller in the population (which received the highest average returns across its respective N_{trials} episodes) is selected for evaluation [9,16]. This controller is evaluated for another $N_{\text{pop}} \times N_{\text{trials}}$ episodes in the Dropout's Dream Land environments. The controller's mean across $N_{\text{pop}} \times N_{\text{trials}}$ trials is logged to the Dream Leader Board. After 2000 generations, the controller at the top of the Dream Leader Board is evaluated in the real environment.

Interacting with the Real Environment In Figure 3 we illustrate the controller's interaction with the real target environment (e^*). Interactions with e^* do not apply dropout to the input or hidden units of M. The controller only interacts with the target environment during testing. These interactions are never used to modify parameters of the controller. At test time r, d, and o are generated by e^*, and $\mathbf{z}$ is the embedding of o from the VAE (V). The only use of M when interacting with the target environment is producing $\mathbf{h}$ as a feature for the controller.

4 Experiments

Broadly speaking, our experiments are focused on either evaluating the dynamics model (M) or the controller (C). Architecture details of V, M, and C are in Appendix A.1. Experiments are performed on the DoomTakeCover-v0 [22] and CarRacing-v0 [18] environments from OpenAI Gym [4]. These have also been used in related works [9,16]. Even though both baseline target environments are simulators we still consider this "reality" because we do not leverage knowledge about the simulator mechanics to learn the source environment (M).

Quality of the dynamics model is evaluated against a training and testing set of trajectories (described below). Quality of the controller is measured by returns in the target environments. For all experiments the controller is trained

exclusively in the dream environment (Section 3.2) for 2,000 generations. The controller only interacts with the target environments for testing (Section 3.3). The target environment is never used to update parameters of the controller. Means and standard deviations of returns achieved by the best controller (Section 3.3) in the target environment are reported based on 100 trials for CarRacing and 1000 trials for DoomTakeCover.[3]

DoomTakeCover Environment DoomTakeCover is a control task in which the goal is to dodge fireballs for as long as possible. The controller receives a reward of +1 for every step it is alive. The maximum number of frames is limited to 2100.

For all tasks on this environment, we collect a training set of $10,000$ trajectories and a test set of 100 trajectories. A trajectory is a sequence of state ($\mathbf{z}$), action ($\mathbf{a}$), reward (r), and termination (d) tuples. Both datasets are generated according to a random policy. Following the same convention as World Models [9], on the DoomTakeCover environment we concatenate $\boldsymbol{z}$, $\boldsymbol{h}$, and $\boldsymbol{c}$ as input to the controller. In (7), we set $\alpha_d = 1$ and $\alpha_r = 0$ because the Doom reward function is determined entirely based on whether the controller lives or dies.

CarRacing Environment CarRacing is a continuous control task to learn from pixels. The race track is split up into "tiles". The goal is to make it all the way around the track (i.e., crossing every tile). We terminate an episode when all tiles are crossed or when the number of steps exceeds 1,000. Let N_{tiles} be the total number of tiles. The simulator [18] defines the reward r_t at each timestep as $\frac{100}{N_{\text{tiles}}} - 0.1$ if a new tile is crossed, and -0.1 otherwise. The number of tiles is not explicitly set by the simulator. We generated 10,000 tracks and observed that the number of tiles in the track appears to follow a normal distribution with mean 289. To simplify the reward function, we fix N_{tiles} to 289 in the randomly generated tracks, and call the modified environment CarRacingFixedN.

For all tasks on this environment, the training set contains $5,000$ trajectories and the test set contains 100 trajectories. Both datasets are collected by following an expert policy with probability 0.9, and a random policy with probability 0.1. The expert policy was trained directly on the CarRacing environment and received an average return of 885 ± 63 across 100 trials. In comparison, the performance of the random policy is -53 ± 41. This is similar to the setup in GameGAN [16] on the Pacmanenvironment which also used an expert policy. For this environment, we set $\alpha_d = \alpha_r = 1$ in (7).

4.1 Comparison with Baselines

Dropout's Dream Land (DDL) is compared against World Models (WM), Monte-Carlo Dropout World Models (MCD-WM), and a uniform random policy on the CarRacing and DoomTakeCover environments. The Monte-Carlo Dropout World Models baseline uses $p_{\text{train}} = 0.05$, $p_{\text{infer}} = 0.1$, and 10 samples. On the Doom environment, we also compare with GameGAN [16] and Action-LSTM [5][4]. All controllers are trained entirely in dream environments.

[3] 100 trials are used for the baselines GameGAN and Action-LSTM.

[4] Results on GameGAN and Action-LSTM returns are from [16].

Table 1. Returns from baseline methods and DDL ($p_{\text{train}} = 0.05$ and $p_{\text{infer}} = 0.1$) on the DoomTakeCover environment.

	DoomTakeCover
random policy	210 ± 108
GameGAN	765 ± 482
Action-LSTM	280 ± 104
WM	849 ± 499
MCD-WM	798 ± 464
DDL	$\mathbf{933 \pm 552}$

Table 2. Returns from baseline methods and DDL ($p_{\text{train}} = 0.05$ and $p_{\text{infer}} = 0.1$) on the CarRacingFixedN and the original CarRacing environments.

	CarRacingFixedN	CarRacing
random policy	-50 ± 38	-53 ± 41
WM	399 ± 135	388 ± 157
MCD-WM	-56 ± 31	-53 ± 32
DDL	$\mathbf{625 \pm 289}$	$\mathbf{610 \pm 267}$

Results on the target environments are in Tables 1 and 2. The CarRacing results appear different from those found in World Models [9] because we are not performing the same experiment. In this paper, we train the controller entirely in the dream environment and only interact with the target environment during testing. In World Models [9], the controller was trained directly in the CarRacing environment.

In Tables 1 and 2, we observe that DDL offers performance improvements over all the baseline approaches in the target environments. We suspect this is because the WM dream environments were easier for the controller to exploit errors between the simulator and reality. Forcing the controller to succeed in many different dropout environments makes it difficult to exploit discrepancies between the dream environment and reality. This leads us to the conclusion that forcing the controller to succeed in many different dropout environments is an effective technique to cross the Dream2Real gap.

The DoomTakeCover returns in the target environment as reported by the temperature-regulated variant[5] in [9] are higher than the returns we obtain from DDL, which does not use temperature. However, we emphasize that adjusting temperature is only useful for a limited set of dynamics models. For example, it would not be straightforward to apply temperature to any dynamics model which does not produce a probability density function (e.g., GameGAN); whereas the DDL approach of generating many *different* dynamics models is useful to any learned neural network dynamics model. Moreover, even though the temperature-regulated variant increases uncertainty of the dream environment, it is still only capable of creating *one* dream environment.

4.2 Inference Dropout and Dream2Real Generalization

In this experiment, we study the effects of dropout on the World Model. First, we evaluate the relationship between dropout and World Model accuracies. Second,

[5] We were unable to reproduce the temperature results in [9].

Table 3. RNN's loss with and without dropout ($p_{\text{train}} = 0.05$ and $p_{\text{infer}} = 0$) during training.

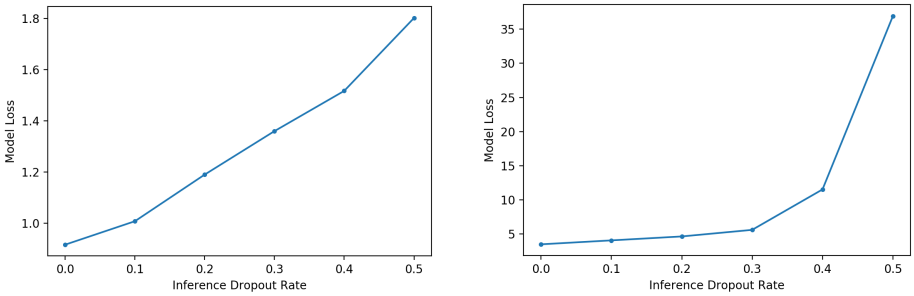
	DoomTakeCover		CarRacingFixedN	
	training loss	test loss	training loss	test loss
without dropout	0.89	**0.91**	2.36	**3.10**
with dropout	0.93	**0.91**	3.19	3.57

we evaluate the relationship between dropout and generalization from the World Model to the target environment. Model loss is measured by the loss in (7) on the test sets. Returns in the target environment are reported based on the best controller (Section 3.3) trained with varying levels of inference dropout. The same training and test sets described at the beginning of Section 4 are used.

Standard use cases of dropout generally observe a larger training loss but lower test loss relative to the same model trained without dropout [6, 27]. In Table 3, we do not observe any immediate performance improvements of the World Model trained with dropout ($p_{\text{train}} = 0.05$ and $p_{\text{infer}} = 0$). In fact, we observe worse results on the test set. The poor performance of both DDL RNNs (Table 3) indicates a clear conclusion about the results from Tables 1 and 2. The improved performance of DDL relative to World Models comes from forcing the controller to operate in many different environments and not from a single more accurate dynamics model M.

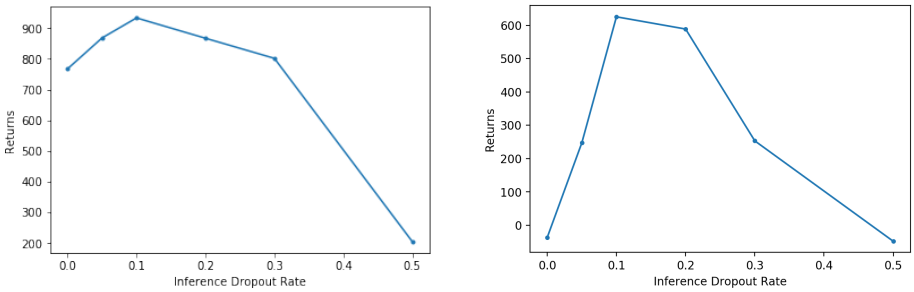
Next we take a World Model trained with dropout and evaluate the model loss on a test set across varying levels of inference dropout (p_{infer}). As expected, in Figure 4 we observe that as the inference dropout rate is increased the model loss increases. In Figure 5 we observe that increasing the inference dropout rate improves generalization to the target environment. We believe that the boost in returns on the target environments comes from an increase in capacity to distort the dynamics model. Figures 4 and 5 suggest that we can sacrifice accuracy of the dream environments to better cross the Dream2Real gap between dream and target environments. However, this should only be useful up to the point where *the task at hand is fundamentally changed*. Figure 5 suggests this point is somewhere between 0.1 and 0.2 for p_{infer}, though we suspect in practice this will be highly dependent on network architecture and the environment.

In Figure 5 we observe relatively weak returns on the real CarRacingFixedN environment when the inference dropout rate is zero. Recall from Table 3 that the dropout variant has a much higher test loss than the non-dropout variant on CarRacingFixedN. This means that when $p_{\text{infer}} = 0$, the *single* environment DDL is able to create is relatively inaccurate. It is easier for the controller to exploit any discrepancies between the dream environment and target environment because only a single dream environment exists. However, as we increase the inference dropout rate it becomes harder for the controller to exploit the dynamics model, suggesting that DDL is especially useful when it is difficult to learn an accurate World Model.



(a) DoomTakeCover. (b) CarRacingFixedN.

Fig. 4. Loss of DDL dynamics model ($p_{\text{train}} = 0.05$) at different inference dropout rates.



(a) DoomTakeCover. (b) CarRacingFixedN.

Fig. 5. DDL ($p_{\text{train}} = 0.05$) returns at different inference dropout rates in the target environments.

4.3 When Should Dropout Masks be Randomized During Controller Training?

In this ablation study we evaluate when the dropout mask should be randomized during training of C. We consider two possible approaches of when to randomize the masks. The first case only randomizes the mask at the beginning of an episode (*episode randomization*). The second case samples a new dropout mask at every step (*step randomization*). We also consider if it is effective to only apply dropout at inference time but not during M training (i.e., $p_{\text{infer}} > 0, p_{\text{train}} = 0$).

As can be seen in Table 4, randomizing the mask at each step offers better returns on both target environments. Better returns in the target environment when applying step randomization comes from the fact that the controller is exposed to a much larger number ($> 1000\times$) of dream environments. We also observe that applying step randomization without training the dynamics model with dropout yields a weak policy on the target environment. This is due to the randomization fundamentally changing the task. Training the dynamics model

with dropout ensures that at inference time the masked model (M^j) is meaningful.

Table 4. Returns of the controller with different frequencies to randomize the dropout mask.

	DoomTakeCover	CarRacingFixedN
episode randomization ($p_{\text{train}} = 0.05$, $p_{\text{infer}} = 0.1$)	786 ± 469	601 ± 197
step randomization ($p_{\text{train}} = 0.05$, $p_{\text{infer}} = 0.1$)	$\mathbf{933 \pm 552}$	$\mathbf{625 \pm 289}$
step randomization ($p_{\text{train}} = 0$, $p_{\text{infer}} = 0.1$)	339 ± 90	-43 ± 52

4.4 Comparison to Standard Regularization Methods

In this experiment we compare Dropout's Dream Land with standard regularization methods. First, we consider applying the standard use case of dropout ($0 < p_{\text{train}} < 1$ and $p_{\text{infer}} = 0$). Second, we consider a noisy variant of M when training C. The Noisy World Model uses exactly the same parameters for M as the baseline World Model. When training the controller, a small amount of Gaussian noise is added to z at every step.

In Table 5, we observe that DDL is better at generalizing from the dream environment to the target environment than the standard regularization methods. Dropout World Models can be viewed as a regularizer on M. Noisy World Models can be viewed as a regularizer on the controller C. The strong returns on the target environment by DDL suggest that it is better at crossing the Dream2Real gap than standard regularization techniques.

Table 5. Returns from World Models, Dropout World Models ($p_{\text{train}} = 0.05$ and $p_{\text{infer}} = 0.0$), Noisy World Models, and DDL ($p_{\text{train}} = 0.05$ and $p_{\text{infer}} = 0.1$) on the CarRacingFixedN and the original CarRacing environments.

	CarRacingFixedN	CarRacing
World Models	399 ± 135	388 ± 157
Dropout World Models	-36 ± 19	-36 ± 20
Noisy ($\mathcal{N}(0, 1)$) World Models	147 ± 121	180 ± 132
Noisy ($\mathcal{N}(0, 10^{-2})$) World Models	455 ± 171	442 ± 171
Dropout's Dream Land	$\mathbf{625 \pm 289}$	$\mathbf{610 \pm 267}$

4.5 Comparison to Explicit Ensemble Methods

In this experiment we compare Dropout's Dream Land with two other approaches for randomizing the dynamics of the dream environment. We consider

using an explicit ensemble of a population of dynamics models. Each environment in the population was trained on the same set of trajectories described at the beginning of Section 4 with a different initialization and different mini-batches. With the population of World Models we train a controller with Step Randomization and a controller with Episode Randomization. Note that the training cost of dynamics models and RAM requirements at inference time scale linearly with the population size. Due to the large computational cost we consider a population size of 2.

In Table 6, we observe that neither Population World Models (PWM) Step Randomization or Episode Randomization substantially close the Dream2Real gap. Episode Randomization does not dramatically improve results because the controller is forced to understand the hidden state ($\mathbf{h}$) representation of every M in the population. Step Randomization performs even worse than Episode Randomization because on top of the previously stated limitations, each dynamics model in the population is also forced to be compatible with the hidden state ($\mathbf{h}$) representation of all other dynamics models in the population. DDL does not suffer from any of the previously stated issues and is also computationally cheaper because only one M must be trained as opposed to an entire population.

Table 6. Returns from World Models, PWM Episode Randomization, PWM Step Randomization, and DDL ($p_{\text{train}} = 0.05$ and $p_{\text{infer}} = 0.1$) on the CarRacingFixedN and the original CarRacing environments.

	CarRacingFixedN	CarRacing
World Models	399 ± 135	388 ± 157
PWM Episode Randomization	398 ± 126	402 ± 142
PWM Step Randomization	-78 ± 14	-77 ± 13
Dropout's Dream Land	$\mathbf{625 \pm 289}$	$\mathbf{610 \pm 267}$

5 Conclusion

Dropout's Dream Land introduces a novel technique to improve controller generalization from dream environments to reality. This is accomplished by taking inspiration from Domain Randomization and training the controller on a large set of different simulators. A large set of different simulators are generated at little cost by the insight that dropout can be used to generate an ensemble of neural networks. To the best of our knowledge this is the first work to bridge the reality gap between learned simulators and reality. Previous work from Domain Randomization [31] is not applicable to learned simulators because they often do not have easily configurable parameters. Future direction for this work could be modifying the dynamics model parameters in a targeted manner [28, 33, 34]. This simple approach to generating different versions of a model could also be useful in committee-based methods [25, 26].

References

1. Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al.: Learning dexterous in-hand manipulation. International Journal of Robotics Research **39**(1), 3–20 (2020)
2. Antonova, R., Cruciani, S., Smith, C., Kragic, D.: Reinforcement learning for pivoting task. Preprint arXiv:1703.00472 (2017)
3. Baldi, P., Sadowski, P.J.: Understanding dropout. In: Advances in Neural Information Processing Systems. pp. 2814–2822 (2013)
4. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. Preprint arXiv:1606.01540 (2016)
5. Chiappa, S., Racaniere, S., Wierstra, D., Mohamed, S.: Recurrent environment simulators. Preprint arXiv:1704.02254 (2017)
6. Gal, Y., Ghahramani, Z.: Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In: International Conference on Machine Learning. pp. 1050–1059 (2016)
7. Gal, Y., Ghahramani, Z.: A theoretically grounded application of dropout in recurrent neural networks. In: Advances in Neural Information Processing Systems. pp. 1019–1027 (2016)
8. Graves, A.: Generating sequences with recurrent neural networks. Preprint arXiv:1308.0850 (2013)
9. Ha, D., Schmidhuber, J.: Recurrent world models facilitate policy evolution. In: Advances in Neural Information Processing Systems. pp. 2450–2462 (2018)
10. Hafner, D., Lillicrap, T., Ba, J., Norouzi, M.: Dream to control: Learning behaviors by latent imagination. In: International Conference on Learning Representations (2020)
11. Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., Davidson, J.: Learning latent dynamics for planning from pixels. In: International Conference on Machine Learning. pp. 2555–2565 (2019)
12. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary computation **9**(2), 159–195 (2001)
13. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: The use of simulation in evolutionary robotics. In: European Conference on Artificial Life. pp. 704–720. Springer (1995)
14. Kahn, G., Villaflor, A., Pong, V., Abbeel, P., Levine, S.: Uncertainty-aware reinforcement learning for collision avoidance. Preprint arXiv:1702.01182 (2017)
15. ukasz Kaiser, Babaeizadeh, M., Mios, P., Osiski, B., Campbell, R.H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., Michalewski, H.: Model based reinforcement learning for Atari. In: International Conference on Learning Representations (2020)
16. Kim, S.W., Zhou, Y., Philion, J., Torralba, A., Fidler, S.: Learning to simulate dynamic environments with GameGAN. In: IEEE Conference on Computer Vision and Pattern Recognition. pp. 1231–1240 (2020)
17. Kingma, D.P., Welling, M.: Auto-encoding variational Bayes. Preprint arXiv:1312.6114 (2013)
18. Klimov, O.: Carracing-v0 (2016), https://gym.openai.com/envs/CarRacing-v0/
19. Kurutach, T., Clavera, I., Duan, Y., Tamar, A., Abbeel, P.: Model-ensemble trust-region policy optimization. Preprint arXiv:1802.10592 (2018)
20. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C.,

Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
21. Mordatch, I., Lowrey, K., Todorov, E.: Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5307–5314 (2015)
22. Paquette, P.: Doomtakecover-v0 (2017), https://gym.openai.com/envs/DoomTakeCover-v0/
23. Peng, X.B., Andrychowicz, M., Zaremba, W., Abbeel, P.: Sim-to-real transfer of robotic control with dynamics randomization. In: IEEE International Conference on Robotics and Automation. pp. 1–8 (2018)
24. Sadeghi, F., Levine, S.: Cad2rl: Real single-image flight without a single real image. Preprint arXiv:1611.04201 (2016)
25. Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., Pathak, D.: Planning to explore via self-supervised world models. Preprint arXiv:2005.05960 (2020)
26. Settles, B.: Active learning literature survey. Tech. rep., University of Wisconsin-Madison Department of Computer Sciences (2009)
27. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**(1), 1929–1958 (2014)
28. Such, F.P., Rawal, A., Lehman, J., Stanley, K.O., Clune, J.: Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. Preprint arXiv:1912.07768 (2019)
29. Sutton, R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: International Conference on Machine learning. pp. 216–224 (1990)
30. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press (2018)
31. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. In: IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 23–30 (2017)
32. Vinyals, O., Babuschkin, I., Czarnecki, W.M., Mathieu, M., Dudzik, A., Chung, J., Choi, D.H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J.P., Jaderberg, M., Vezhnevets, A.S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T.L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wunsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., Silver, D.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575**(7782), 350–354 (2019)
33. Wang, R., Lehman, J., Clune, J., Stanley, K.O.: Paired open-ended trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions. Preprint arXiv:1901.01753 (2019)
34. Wang, R., Lehman, J., Rawal, A., Zhi, J., Li, Y., Clune, J., Stanley, K.O.: Enhanced POET: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. Preprint arXiv:2003.08536 (2020)
35. Zaremba, W., Sutskever, I., Vinyals, O.: Recurrent neural network regularization. Preprint arXiv:1409.2329 (2014)

A Appendix

A.1 Architecture Details

We follow the same architecture setup as World Models. We adopt the following notation [16] to describe the VAE architecture.

Conv2D(a, b, c): 2D-Convolution layer with output channel size **a**, kernel size **b**, and stride **c**. All use valid padding and ReLU activations.

T.Conv2D(a, b, c): Transposed 2D-Convolution layer with output channel size **a**, kernel size **b**, stride **c**. The final layer uses a sigmoid activation but every other layer uses ReLU activations.

LSTM(a): LSTM layer with **a** units.

Dense(a): Fully Connected layer with output size **a** followed by a ReLU activation.

Linear(a): Linear layer with output size **a**.

Reshape(a): Reshape input to output size **a**.

Table 7. Architectures DoomTakeCover and CarRacing.

DoomTakeCover	CarRacing
VAE (V)	
Conv2D(32, 4, 2)	Conv2D(32, 4, 2)
Conv2D(64, 4, 2)	Conv2D(64, 4, 2)
Conv2D(128, 4, 2)	Conv2D(128, 4, 2)
Conv2D(256, 4, 2)	Conv2D(256, 4, 2)
Reshape(1024)	Reshape(1024)
Linear(32), Linear(32)	Linear(64), Linear(64)
Dense(1024)	Dense(1024)
Reshape(1, 1, 1024)	Reshape(1, 1, 1024)
T.Conv2D(128, 5, 2)	T.Conv2D(128, 5, 2)
T.Conv2D(64, 5, 2)	T.Conv2D(64, 5, 2)
T.Conv2D(32, 6, 2)	T.Conv2D(32, 6, 2)
T.Conv2D(3, 6, 2)	T.Conv2D(3, 6, 2)
World Model (M)	
LSTM(512)	LSTM(256)
Dense(961)	Dense(482)
Controller (C)	
Linear(1)	Linear(3)