Learning Approximate Stochastic Transition Models

Yuhang Song\textsuperscript{1}, Christopher Grimm\textsuperscript{2}, Xianming Wang\textsuperscript{3}, Michael L. Littman\textsuperscript{4}
\textsuperscript{1}Beijing University of Aeronautics and Astronautics, \textsuperscript{2}University of Michigan, \textsuperscript{3}Renmin University of China, \textsuperscript{4}Brown University
yuhangsong@buaa.edu.cn, crgrimm@umich.edu, wxm@ruc.edu.cn, michael_littman@brown.edu

Abstract

We examine the problem of learning mappings from state to state, suitable for use in a model-based reinforcement-learning setting, that simultaneously generalize to novel states and can capture stochastic transitions. We show that currently popular generative adversarial networks struggle to learn these stochastic transition models but a modification to their loss functions results in a powerful learning algorithm for this class of problems.

Introduction and Background

Model-based approaches separate the reinforcement-learning (RL) problem into two components. The first component learns a transition model that predicts the next state from the current state and action. The second component uses that model to make decisions by looking ahead to predict the consequences of different courses of actions. This paper focuses on the first problem of acquiring the model, specifically addressing the development of a mechanism for learning to approximate a stochastic transition function.

A Markov decision process (MDP) model of an environment consists of a set of states $S$ and actions $A$, a transition function $T : S \times A \rightarrow \Pi(S)$ mapping state–action pairs to a probability distribution over next states, and a reward function $R : S \times A \rightarrow \mathbb{R}$.

Since the focus of this paper is not on decision making but on learning the dynamics, we simplify the transition function to $T(\bar{x}, a, x') = \mathbb{P}_{T}(x' | a, \bar{x})$, which represents the probability that state $x'$ will follow $\bar{x}$. A separate function can be learned for each action $a \in A$. Although some authors have found there to be an advantage to representing the transitions jointly for all actions (Oh et al. 2015), this issue is orthogonal to the representation issue we address here.

To review methods for learning $\mathbb{P}_{T}(x')$, we begin by separating out three representations for transition models. A query model is one that can answer, for any $\bar{x}, x'$ pair, the probability of $x'$ given $\bar{x}$. Such a model can be represented as a table if the state space is relatively small (Kearns and Singh 2002). It can also be captured by a dynamic Bayesian network (Kearns and Koller 1999; Degr"{i}s, Sigaud, and Wuillemin 2006). Some types of planners, such as ones based on policy iteration (Puterman 1994), require access to these probabilities to compute expected values. Query models can be very challenging to work with and learn when the size of the state space is enormous, however, because looping over all possible values of $x'$ can be too expensive. It is especially problematic when most $\bar{x}, x'$ pairs have zero probability, since considering each of them is expensive and pointless.

A sparse model is a refinement of the query model that takes a state $\bar{x}$ as input and returns a list of states $N(\bar{x})$ such that $x' \in N(\bar{x})$ if and only if $\mathbb{P}_{T}(x' | a, \bar{x}) > 0$. Such a representation can be used much more efficiently as it only needs to consider the non-zero entries of $\mathbb{P}_{T}(x')$. Tabular and DBN methods can be used in this setting, but a general approach has yet to be articulated. In addition, they provide no advantage over the query model in environments in which $|N(\bar{x})|$ is intractably large.

An alternative is a generative model, which is a function $G$ that, given, $\bar{x}$, produces $x'$. Learning a generative model is closely related to classical supervised learning problems. Given examples of $\bar{x}, x'$ pairs, they learn a mapping such that $G(\bar{x})$ produces $x'$. When the target mapping is deterministic, many learning algorithms can be brought to bear to learn the transition model (Atkeson, Moore, and Schaal 1997). These learning algorithms can be applied to stochastic transition models, but, as we show, there are significant pitfalls to doing so. An exception is in control problems where the transition model has the form $x' \sim G(x) + \eta$, where the $\eta$ is state independent and typically small and zero-mean noise, so that planning using $G(x)$ results in an approximation to planning with the stochastic model—the noise can be safely ignored during planning (Bradtke 1993).

In our work\footnote{Code to reproduce this work is publicly available online for facilitating future research: https://github.com/YuhangSong/SGAN.}, we capture the transition function adopting a generative adversarial network (GAN) perspective (Goodfellow et al. 2014). In the next section, we present GANs and derive our novel variant that is more effective at capturing detailed probability distributions. We provide empirical comparisons between GANs and other approaches to learning, showing that our GAN approach can learn to generalize probabilistic functions effectively.
Modeling Stochastic Transitions with GANs

The GAN approach to modeling stochastic transition functions focuses on creating a generator \( G \), which takes in current state \( x \) and random noise \( n \) and generates a possible next state \( x_g \),

\[
x_g = G_\mu(x, n),
\]

where \( \mu \) is a set of parameters defining \( G \) to be set by learning. We assessed the error in \( G \) by the L1 distance between the distribution produced by \( G \) and the true distribution \( P \).

A second model \( D \), the discriminator, takes in current state \( x \) and next state \( x' \). Here, \( x' \) is either a state \( x_g \) generated by \( G \) or a real state \( x_r \) from the observed data. The output of the discriminator is interpreted as a score of whether it believes \( x' = x_r \). We write

\[
\begin{aligned}
D_\theta(x, x'),
\end{aligned}
\]

where \( \theta \) is the parameters defining \( D \).

**WGANs and GP-WGANs**

In the Wasserstein GAN or WGAN (Arjovsky, Chintala, and Bottou 2017), the generator and discriminator attempt to minimize a metric known as the Earth Mover’s distance between the generated probability distribution and the real probability distribution:

\[
W(P_r, P_g) = \sup_{\|f\|_{L^1} \leq 1} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_g}[f(x)],
\]

where \( \| \cdot \|_L \leq 1 \) denotes the space of 1-Lipschitz functions.

Computing the supremum over 1-Lipschitz functions in Equation (3) is computationally intractable. However, Arjovsky, Chintala, and Bottou (2017) showed how to approximate this operation by optimizing the function \( f \) as the discriminator \( D_\theta^W \) and maximizing the expression over parameters \( \theta \). This choice leads to the following expression, which encapsulates the optimization procedure for the discriminator and generator networks in the conditional setting:

\[
\min_{\mu} \max_{\|\theta\|_{D_\theta^W} \leq 1} \mathbb{E}_{x \sim P_r}[D_\theta^W(x, x_r)] - \mathbb{E}_{x \sim P_g}[D_\theta^W(x, x_g)].
\]

In words, we are looking for the generator such that even the most discriminating discriminator is unable to assign real data high scores and generated data low scores—the generator’s fixed, and the generator while holding the discriminator’s parameters fixed.

Notice that the optimization of \( \theta \) requires that \( D_\theta^W \) is 1-Lipschitz throughout the process. The most popular way of enforcing this constraint is by introducing a penalty

\[
\min_{\mu} \max_{\|\theta\|_{D_\theta^W} \leq 1} \mathbb{E}_{x \sim P_r}[\mathbb{E}_{x \sim P_g}[D_\theta^W(x, x_g)]] - \mathbb{E}_{x \sim P_g}[D_\theta^W(x, x_g)],
\]

where \( \lambda > 0 \) is a hyperparameter controlling how strongly to enforce the penalty term and \( x_r \) is a point drawn from somewhere in the space of \( P_r \) or \( P_g \). Specifically, \( x_r \) is generated as an interpolation between a pair of real and generated samples: \( x_r = \tau x_r + (1 - \tau) x_g \) with \( x_r \sim P_r, x_g \sim P_g \), and \( \tau \sim [0, 1] \).

We choose L1 because it is common to measure the error in transition functions this way—a bound in L1 error can be translated to a bound in the reward obtained via the simulation lemma (Kearns and Singh 2002). Of course, other measures of distribution difference are also valid.

Because the function assigns scores, it is sometimes referred to as the critic instead of the discriminator.

**Algorithm 1 SGAN Training Algorithm for learning to match an observed probability distribution**

- **Require:** \( C, B, \alpha, \beta_1, \beta_2, \delta \), dataset containing multiple transition pairs \((\bar{x}, x_r)\).
- **Require:** Initial parameters \( \theta_0 \) for discriminator \( D^S_\theta \), initial parameters \( \mu_0 \) for generator \( G_\mu \).

\[
\begin{align*}
1: & \textbf{while } \theta \text{ has not converged do} \\
2: & \textbf{for } c = 1, \ldots, C \text{ do} \\
3: & \textbf{for } b = 1, \ldots, B \text{ do} \\
4: & \text{Sample real transition pair } (\bar{x}, x_r) \text{ from dataset} \\
5: & \text{Sample noise vector } n \sim [0, 1]^l \\
6: & x_g \leftarrow G_\mu(\bar{x}, n) \text{ //generate next state} \\
7: & T_b \leftarrow \frac{\|x_r - x_g\|}{\delta} \\
8: & \textbf{for } t = 1, \ldots, T_b \text{ do} \\
9: & x_r \leftarrow \tau x_r + (1 - \tau) x_g \\
10: & L_D^{(b,t)} \leftarrow (\|\nabla_{x_r} D^S_\theta(\bar{x}, x_r) - \|x_r - x_g\|)^2 \\
11: & \textbf{end for} \\
12: & \textbf{end for} \\
13: & \theta \leftarrow \theta - \nabla \theta \frac{1}{B} \sum_{b=1}^B \frac{1}{T_b} \sum_{t=1}^{T_b} L_D^{(b,t)} \\
14: & \textbf{end for} \\
15: & \textbf{for } b = 1, \ldots, B \text{ do} \\
16: & \text{Sample noise vector } n \sim [0, 1]^l \\
17: & L_G \leftarrow -D^S_\theta(G_\mu(\bar{x}, n)) \\
18: & \mu \leftarrow \mu - \nabla \mu \frac{1}{B} \sum_{b=1}^B L_G^{(b)} \\
19: & \textbf{end for} \\
20: & \mu \leftarrow \mu - \nabla \mu \frac{1}{B} \sum_{b=1}^B L_G^{(b)} \\
21: & \textbf{end while}
\]

**SGANs**

As we demonstrate, WGAN and GP-WGANs struggle to learn generators that closely match the observed transition data. In response, we propose our SGAN in Algorithm 1. Its basic steps are:

- Sample a real transition pair \((\bar{x}, x_r)\) from the dataset.
- Generate a transition pair \((\bar{x}, x_g)\) with \( x_g = G_\mu(\bar{x}, n) \), where the components of the noise vector \( n \) are drawn from \([0, 1]^l\) and \( G_\mu \) denotes the generator with parameter \( \mu \).
• Train the discriminator $D^S_\theta$ with parameter $\theta$ to minimize $L^S_D$, which we will define later.
• Train the generator $G_\mu$ with its loss $L_G = \mathbb{E}_{n \sim [0,1]} \{-D^S_\theta(G_\mu(x,n))\}$.

Apart from the new loss function $L^S_D$, the above procedure is shared with that of WGANs and GP-WGANs.

Now, we define the new discriminator loss function $L^S_D$,
\[
L^S_D = \mathbb{E}_{x_r, x_g \sim \mathbb{P}_n, \tau \sim [0,1]} \left(\|\nabla_x D^S_\theta(x, x_r) - \frac{x_r - x_g}{\|x_r - x_g\|}\|^2\right),
\]
where $x_r$ is
\[
x_r = \tau x_r + (1 - \tau)x_g, \quad \tau \sim [0,1].
\]
But, unlike the way $x_r$ is sampled in the GP-WGAN, the SGAN samples $x_r$ for $T$ times given each $x_r, x_g$ pair. The value $T$ is computed by
\[
T = \frac{\|x_r - x_g\|}{\delta}.
\]
Here, $\delta$ is a hyper-parameter of the algorithm, the choice of which is discussed in the experiment section.

**Training the discriminator in SGANs**

By executing Algorithm 1, our $D^S_\theta$ is modelling a different discriminator from the $D^S_\theta$ in WGAN—we refer to our new discriminator as the SGAN discriminator. Following the derivation of the optimal WGAN discriminator, we now express the optimal SGAN discriminator, denoted by $D^S^*$:
\[
D^S^*(x) = \int_0^x \left(\int_0^1 \mathbb{P}_r(\tilde{x})d\tilde{x} - \int_0^1 \mathbb{P}_g(\tilde{x})d\tilde{x}\right)dx.
\]

This section focuses on proving the SGAN algorithm can minimize $D^S^*$. Our argument proceeds in two steps:
• Lemma 1 shows an important property arising from sampling $x_r$ for $T$ times in the SGAN algorithm.
• Building on this property, Theorem 1 shows that the loss function of $D$ in the SGAN algorithm (Equation 6) leads to $D^S^*$.

We restrict our argument to a one-dimensional setting for simplicity even though the algorithm is implemented in tested in high dimensional problems.

**Lemma 1** Consider an event denoted by: $x_r \overset{T}{\sim} x_n$, defined to means we sample $x_r$ $T$ times and the $x_r = x_n$ at least once. To be clear, $x_r, x_n, x_g$ are all random variables while $x_n$ represents a specific fixed value. Assuming $T = |x_r - x_g|/\delta$, it follows that
\[
P(x_r \overset{T}{=} x_n | x_r, x_g) = \begin{cases} c & x_r < x_n < x_g, x_g < x_n < x_r \\ 0 & \text{else} \end{cases}
\]
where $c$ is a constant independent of the values of the random variables.

**Proof of Lemma 1:** We begin with a derivation in which we have discretized the one-dimensional space into intervals of size $\varepsilon$. Notationally, the discretized versions of the variables are marked with a check over the variable name. Later on, we will derive what happens to these expressions as we take the limit of $\varepsilon \to 0$, bringing us back to statements about continuous space. We have
\[
P(\tilde{x}_r = \check{x}_n | \check{x}_r, \check{x}_g) = \begin{cases} \frac{1}{|x_r - x_g|/\varepsilon} & \check{x}_r < \check{x}_n < \check{x}_g, \check{x}_g < \check{x}_n < \check{x}_r \\ 0 & \text{otherwise} \end{cases}
\]
If we sample $\check{x}_r$ for $T$ times,
\[
P(\check{x}_r \overset{T}{=} \check{x}_n | \check{x}_r, \check{x}_g) = \begin{cases} 1 - (1 - P(\check{x}_r = \check{x}_n | \check{x}_r, \check{x}_g))^T & \check{x}_r < \check{x}_n < \check{x}_g, \check{x}_g < \check{x}_n < \check{x}_r \\ 0 & \text{otherwise} \end{cases}
\]
We relate $\varepsilon$ to $\delta$ via a positive integer multiple\(^4\) $z$:
\[
\delta = z\varepsilon
\]
where $z \in \mathbb{Z}^+$. To connect back to Equation (14), we consider following limit,
\[
\lim_{\delta = z\varepsilon, \varepsilon \to 0} (1 - \frac{1}{d/\varepsilon})\frac{d}{\delta} = \lim_{\delta = z\varepsilon, \varepsilon \to 0} e^{d/\delta \ln(1 - 1/d\varepsilon)}
\]
\[
= \lim_{\delta = z\varepsilon, \varepsilon \to 0} e^{\ln(d/z\varepsilon)}
\]
\[
= \lim_{\varepsilon \to 0} e^{\frac{d/\varepsilon}{z}}
\]
\[
= e^{-1/z}
\]
Substituting Equation (16) into Equation (14) and taking the limit as $\varepsilon \to 0$, we have
\[
P(\check{x}_r \overset{T}{=} \check{x}_n | \check{x}_r, \check{x}_g)
\]
\[
= \lim_{\delta = z\varepsilon, \varepsilon \to 0} P(\check{x}_r \overset{T}{=} \check{x}_n | \check{x}_r, \check{x}_g)
\]
\[
= \begin{cases} 1 - e^{-1/z} & \check{x}_r < \check{x}_n < \check{x}_g, \check{x}_g < \check{x}_n < \check{x}_r \\ 0 & \text{otherwise} \end{cases}
\]
where we have switched back to the continuous space and finished the proof.

**Theorem 1** Under all the assumptions in Lemma 1, if we update $D^S_\theta$ with loss
\[
L = (\|\nabla_x D^S_\theta(\check{x}, x_r) - \frac{x_r - x_g}{|x_r - x_g|}\|^2),
\]
then $D^S_\theta(x)$ approaches
\[
D^S_\theta(x) = c D^S^*(x),
\]
for an undefined constant $c$.

\(^4\)This is true when consider $\varepsilon \to 0$ is the minimal value a computer can operate.
We compare SGAN against a tabular learner, a deterministic deep network and the state-of-the-art\(^5\) GP-WGAN.

Given a set \( S \) of \((\bar{x}, x_r)\) samples, our tabular learner simply memorizes all of them. It then estimates \( P_{\text{tab}}(x') = \frac{1}{|\{(\bar{x}', \cdot) \in S\}|} \cdot \frac{1}{|\{(\bar{x}, \cdot) \in S\}|} \). If \( \bar{x} \) was not observed, it returns a default value that is interpreted as an error in our experiments.

For all deep neural network based methods, that is, deterministic deep network, GP-WGAN and SGAN, we used the same Adam optimizer with parameters \( \alpha = 0.0001, \beta_1 = 0.0, \) and \( \beta_2 = 0.9 \). We kept the network structure as uniform as possible. For all 3D convolutional neural networks (C) layers and 3D deconvolutional neural networks (DC) layers (Ji et al. 2013), we used LeakyRelu activation with a negative slope of 0.001 and kernel size \( D \times 4 \times 4 \), stride \( 1 \times 2 \times 2 \), and padding \( 0 \times 1 \times 1 \) (sizes are sizes reported as Depth \( \times \) Height \( \times \) Width). We denote a C layer as \([C^D] \), where \( D \) denotes different kernel depth \( D \times 4 \times 4 \). Similarly, we denote a DC layer as \([DC^D] \). For all fully connected (F) layers, we used LeakyRelu activation with a negative slope of 0.001. We denote a F layer mapping size \( a \) to size \( b \) as \( F_a \to b \).

To be able to precisely describe the networks in our experiments, we define a few special terms:

- **Squeeze layer** [S]. This layer always appears after a C layer, and it first flattens the output of the C layer, then uses a F layer mapping the flattened vector to 512.

- **Concatenate Layer** [CL]. This layer always appears after a F layer of size 512. It concatenates the output of the F layer with noise vector \( n \), which means it is a F layer mapping from \((512+n)\) to 512, and the dimensionality of the noise vector is \( l = 128 \). For uniformity, we run deterministic deep networks using the same structure, replacing \( n \) with a zero vector of the same size.

- **Unsqueeze layer** [U]. This layer appears after a F layer of size 512 and before a DC layer. It first uses a F layer that maps 512 to the size of the input of the following DC layer, then reshapes the output vector to the shape of the input of the following DC layer.

- **Linear output layer** [L]. This layer appears after a F layer of size 512. It is a linear layer that maps 512 to 1.

- **Layer sequence**. We use arrows to show how layers are connected: \( * \to | \to * \).

**Network structure for all grid domains.** For vector-based domains, we denote the size of the vector as \( V \). Our deterministic deep network, \( G \) of GP-WGAN and \( G \) of SGAN used the same structure:

\[
\begin{align*}
[F^V \to 512] & \to [F^{512} \to 512] \to [S] \to [CL] \to [U] \to [F^{512} \to V] \\
\end{align*}
\]

For vector-based domains, \( D \) of GP-WGAN and \( D \) of SGAN use the same structure of:

\[
\begin{align*}
[F^{2V} \to 512] & \to [F^{512} \to 512] \to [S] \to [L] \\
\end{align*}
\]

For image representations, deterministic deep network, \( G \) of GP-WGAN and \( G \) of SGAN, used the same structure of:

\[
\begin{align*}
[C^1] & \to [C^1] \to [S] \to [CL] \to [U] \to [DC^1] \to [DC^1] \\
\end{align*}
\]

For image representations, \( D \) of GP-WGAN and \( D \) of SGAN used the same structure of:

\[
\begin{align*}
[C^2] & \to [C^1] \to [S] \to [L] \\
\end{align*}
\]

\( ^5\)In preliminary work, we evaluated GAN and WGAN and found they were consistently worse than GP-WGAN.
Network structure for marble domain. The deterministic deep network, $G$ of GP-WGAN, and $G$ of SGAN used the same structure of:
- $[C^2] \rightarrow [C^1] \rightarrow [C^1] \rightarrow [S] \rightarrow [CL] \rightarrow [U] \rightarrow [DC^1] \rightarrow [DC^1] \rightarrow [DCL]$

Networks $D$ of GP-WGAN and $D$ of SGAN used the same structure of:
- $[C^2] \rightarrow [C^2] \rightarrow [C^1] \rightarrow [S] \rightarrow [L]$

SGAN Parameter Selection
The SGAN hyper-parameter $\delta$ sets a weak trade-off between training speed and optimizing the SGAN objective function. Smaller values of $\delta$ will guarantee SGAN is achieved but result in slow training speed as many $x_r$ values end up being sampled. We found that, as long as $T$ is kept above 3, $\delta$ can be made as small as desired. As such, we recommend keeping $T$ around 3, since it reduces computational cost without sacrificing suboptimality. In our experiments, we used $\delta = 0.3$ for all vector-based domains and $\delta = 1$ for all image-based domains.

When setting $\delta$ in a new domains, we recommend a simple way to find the maximal $\delta$ that keeps $T$ above 3. If the state can be decomposed into a deterministic piece (like maintaining the background) and a stochastic piece (like deciding which local move to make), let $s^d$ be a state that highlights the deterministic aspect (such as a pure black background in our 2D grid domain) and let $s^a$ be a state that highlights the stochastic aspect (such as a black background with a white agent on it), and set $\delta = \frac{\|s^d - s^a\|}{3}$.

Evaluation metric
As has been mentioned, we evaluate the error in $G$ by the L1 distance between the distribution produced by $G$ and the true distribution. In evaluating the distributions, we found that the GANs sometimes produce states that are not meaningful in the context of the domain under study. In these cases, we also include an evaluation of sample validity—how often the model produces meaningful samples. When possible, we resample after an invalid sample is produced. Our rule for separating valid from invalid samples is to call an output valid if its pixel-wise deviation to a corresponding real state is less than 0.1.

To make comparisons as fair as possible, all methods are evaluated after being trained with 100,000 iterations with a batch size of 32 for each iteration. For all grid domains represented by images, we used a block size of 4, which means every cell in the grid is $4 \times 4$ pixels.

Simple Domains
We first investigate SGAN on two simple domains. The state in the 1D Grid domain is the location of an agent in a $1 \times n$ hallway. The dynamics are that the agent transitions with a $\frac{1}{3}$ probability to the left and a $\frac{2}{3}$ probability to the right. If a transition would cause the agent to exit the $1 \times n$ hallways, it remains in place. We represented the state of the domain in two different ways. In the vector representation, the learner was presented an $n$-bit vector with a 1 at the

![Figure 1: Transition pairs modelled by a deterministic deep network on the 5 x 5 2D Grid, where the image on the left side is the start state, and the image on the right side is the generated next state. (a) The learned transition pair under the uniform random walk dynamics (0.25 : 0.25 : 0.25 : 0.25). (b) The learned transition pair under the Russell-Norvig grid dynamics (0.8 : 0.1 : 0.0 : 0.1).](image)
### Table 1: Results on Simple Domains.

| Representation | Size | Dynamic | Tabular learner | Deterministic deep network | GP-WGAN | SGAN |
|----------------|------|---------|----------------|----------------------------|---------|------|
| 1D Grid Vector | 5    | 1/3:2/3 | 0.001/100\%    | /0\%                       | 0.231/92\% | 0.046/99\% |
|                | 10   | 1/3:2/3 | 0.001/100\%    | /0\%                       | 0.103/99\% | 0.038/99\% |
|                | 20   | 1/3:2/3 | 0.001/100\%    | /0\%                       | 0.089/98\% | 0.035/98\% |
| 1D Grid Image  | 5    | 1/3:2/3 | 0.001/100\%    | /0\%                       | 0.149/97\% | 0.054/97\% |
|                | 10   | 1/3:2/3 | 0.001/100\%    | /0\%                       | 0.221/94\% | 0.106/97\% |
|                | 20   | 1/3:2/3 | 0.001/100\%    | /0\%                       | 0.152/94\% | 0.076/93\% |
| 2D Grid Image  | 5    | 0.8:0.1:0:0.1 | 0.018/100\% | /0\%                       | 0.140/92\% | 0.109/92\% |
|                | 5    | 0.25:0.25:0.25:0.25 | 0.018/100\% | /0\%                       | 0.450/77\% | 0.082/90\% |
| Overall        |      |         | 0.005/100\%    | /0\%                       | 0.196/93\% | 0.068/96\% |

### Table 2: Results on Complex Domains.

| Representation | Size | Dynamic | Tabular learner | Deterministic deep network | GP-WGAN | SGAN |
|----------------|------|---------|----------------|----------------------------|---------|------|
| 2D Grid with Obstacles Image 5 0.8:0.1:0:0.1 | 0.021/100\% | /0\% | 0.099/96\% | 0.098/97\% |
| 2D Grid with Random Background Image 5 0.8:0.1:0:0.1 | 2.000/100\% | /0\% | 0.255/90\% | 0.118/93\% |
| Overall        |      |         | 1.009/100\%    | /0\%                       | 0.281/87.35\% | 0.124/94\% |
| Marble         | Image | /       | /              | /                          | Up:Down:Invalid | / |
|                |      |         |                | 100\%:0\%:0\%            | Up:Down:Invalid | / |
|                |      |         |                | 80\%:0\%:20\%            | Up:Down:Invalid | / |

The 2D Grid with Random Backgrounds domain is another 2D Grid domain. Unlike the previous domain, obstacles can appear at any location. Each grid cell is represented by two kinds of features—a ‘fence feature’ denoting the presence of an obstacle and an ‘agent feature’ denoting the presence of the agent. Since the space of possible backgrounds is enormous, models have to generalize from their limited data set to learn the underlying rules governing the dynamics. The size of the dataset we use for this domain is a $10^{-6}$ fraction of the total number of possible transition pairs. As such, the tabular learner performs extremely poorly in this task.

For these complex domains, we find that it is common for networks to need to learn two things: How to copy the background features and how to capture the probabilistic aspects of the transitions. Learning one of these can interfere with learning the other. A common failure mode is for the output layers of the $G$ network to lose their connection to $n$ when it learns the deterministic part of the transition. Once those aspects are learned, it can be difficult to recover the connections to the noise inputs $n$. To encourage the network to retain these connections, we have an optional additional loss in $G$:

$$L_n^G = -\log(1 + \|\nabla_n G_\mu(x, n)\|),$$

where the purpose of the $\log(1 + \cdot)$ operation is to restrain from becoming too large. We combine this loss with the original loss of $G$ by simple addition with a weighting coefficient of $\rho = 1.0$. We only implement this technique on the three complex domains, as the deterministic part in the transition is relatively complicated and important. In practice, we found this additional loss of $G$ gives a significant improvement on all GAN-related methods.

As shown in Table 2, our SGAN results in improvements in the most complex domains in terms of L1 loss and sample validity. Once again, the deterministic deep net cannot generate any valid samples. Figure 2 gives a visualization from the deterministic deep network, GP-WGAN, and SGAN on the 2D Grid with Random Backgrounds domain. From this fig-

---

The 2D Grid with Random Backgrounds domain is another 2D Grid domain. Unlike the previous domain, obstacles can appear at any location. Each grid cell is represented by two kinds of features—a ‘fence feature’ denoting the presence of an obstacle and an ‘agent feature’ denoting the presence of the agent. Since the space of possible backgrounds is enormous, models have to generalize from their limited data set to learn the underlying rules governing the dynamics. The size of the dataset we use for this domain is a $10^{-6}$ fraction of the total number of possible transition pairs. As such, the tabular learner performs extremely poorly in this task.

For these complex domains, we find that it is common for networks to need to learn two things: How to copy the background features and how to capture the probabilistic aspects of the transitions. Learning one of these can interfere with learning the other. A common failure mode is for the output layers of the $G$ network to lose their connection to $n$ when it learns the deterministic part of the transition. Once those aspects are learned, it can be difficult to recover the connections to the noise inputs $n$. To encourage the network to retain these connections, we have an optional additional loss in $G$:

$$L_n^G = -\log(1 + \|\nabla_n G_\mu(x, n)\|),$$

where the purpose of the $\log(1 + \cdot)$ operation is to restrain from becoming too large. We combine this loss with the original loss of $G$ by simple addition with a weighting coefficient of $\rho = 1.0$. We only implement this technique on the three complex domains, as the deterministic part in the transition is relatively complicated and important. In practice, we found this additional loss of $G$ gives a significant improvement on all GAN-related methods.

As shown in Table 2, our SGAN results in improvements in the most complex domains in terms of L1 loss and sample validity. Once again, the deterministic deep net cannot generate any valid samples. Figure 2 gives a visualization from the deterministic deep network, GP-WGAN, and SGAN on the 2D Grid with Random Backgrounds domain. From this fig-
ure, we can see that all the neural network-based methods are able to learn how to copy the background forward, showing their generalization capability. Of the three, however, only GP-WGAN and SGAN can model stochastic transitions, and SGAN does a better job of capturing the mapping.

Physical Domain
In addition to these simulated video-game-like domains, we built a real-world experimental apparatus where stochasticity stems from low-level physical interactions. Our Marble domain consists of a self-resetting marble track. We used a video camera to capture the interactions on one particular part of the track consisting of a bowl-like space that is split by a pillar in the middle. We found that the marble, encountering the pillar from the left side, randomly heads in one of two directions, up or down. We collected 259 minutes of video on this single split at 30 frames per second. To focus on the interesting part of the stochastic transitions, we filtered out the frames that did not contain the marble by computing the pixel-wise variance of frames and setting a threshold to judge if there is a marble present. This process left 22,427 transition pairs in total.

To evaluate this domain, the $\bar{x}$ for all methods was set to two consecutive frames and the learner’s job was to predict the succeeding frame. We generate a sequence of marble images starting from two frames. Figure 3 shows one real sequence and multiple generated sequences from a deterministic deep net, GP-WGAN, and SGAN, respectively.

As shown in the last row of Table 2, we manually generated a statistic concerning whether the generated sequences show the marble going up or down or following an invalid path based 30 randomly generated samples. We can see from these figures that the deterministic deep net can generate valid next states but the sequence is always identical—it fails to model the stochastic transitions. In contrast, GP-WGAN can generate multiple outputs coming from the sequences, but the sequences are still quite deterministic compared to that of SGAN. The SGAN can generate multiple sequences of diverse transitions—it is the best at modeling this stochastic dynamical system.

One caveat is that we note that the generated images from the SGAN are generally of a lower quality than those from the deterministic deep net or GP-WGAN. We currently have no insight into why the SGAN is not generating clear images—we leave it to future research to study in detail.

Conclusion and Future Work
The SGAN approach provides a promising way of learning transition functions that require both generalization and complex stochastic models. We plan to continue to improve the robustness and accuracy of the method, returning to the original motivation of learning stochastic transitions for model-based reinforcement learning.

References
Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein gan. arXiv preprint arXiv:1701.07875.
Atkeson, C. G.; Moore, A. W.; and Schaal, S. 1997. Locally weighted learning for control. *Artificial Intelligence Review* 11:75–113.

Bradtke, S. J. 1993. Reinforcement learning applied to linear quadratic regulation. In Hanson, S. J.; Cowan, J. D.; and Giles, C. L., eds., *Advances in Neural Information Processing Systems* 5, 295–302. San Mateo, CA: Morgan Kaufmann.

Degris, T.; Sigaud, O.; and Wuillemin, P.-H. 2006. Learning the structure of factored markov decision processes in reinforcement learning problems. In *Proceedings of the 23rd international conference on Machine learning*, 257–264. ACM.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, 2672–2680.

Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; and Courville, A. 2017. Improved training of Wasserstein GANs. arXiv preprint arXiv:1704.00028.

Ji, S.; Xu, W.; Yang, M.; and Yu, K. 2013. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence* 35(1):221–231.

Kearns, M. J., and Koller, D. 1999. Efficient reinforcement learning in factored MDPs. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, 740–747.

Kearns, M. J., and Singh, S. P. 2002. Near-optimal reinforcement learning in polynomial time. *Machine Learning* 49(2–3):209–232.

Oh, J.; Guo, X.; Lee, H.; Lewis, R. L.; and Singh, S. 2015. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, 2863–2871.

Puterman, M. L. 1994. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. New York, NY: John Wiley & Sons, Inc.

Russell, S. J., and Norvig, P. 1994. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall.