Learning Approximate Stochastic Transition Models

AAAI Press
Association for the Advancement of Artificial Intelligence
2275 East Bayshore Road, Suite 160
Palo Alto, California 94303

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 $R : S \times A \rightarrow \Re$.

Since the focus of this paper is not on decision making but on learning the dynamics, we simplify the transition function to $T(\bar{x}, a, x') = \mathbb{P}_r^{\bar{x}}(x')$, 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 $\mathbb{P}_r^{\bar{x}}(x')$, 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; Degris, 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 if enormous, however, because looping over all the 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 $\mathbb{P}_r^{\bar{x}}(x') > 0$. Such a representation can be used much more efficiently as it only needs to consider the non-zero entries of $\mathbb{P}_r^{\bar{x}}(x')$. 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 η 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, 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 $\bar{x}$ and random noise $\mathfrak{n}$ and generates a possible next state x_g,

$$x_g = G_\mu(\bar{x}, \mathfrak{n}), \tag{1}$$

where μ 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[1]. A second model D, the *discriminator*, takes in current state $\bar{x}$ 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

$$D_\theta(\bar{x}, x'), \tag{2}$$

where θ 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:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \sup_{\|f\|_L \le 1} \mathbb{E}_{x_r \sim \mathbb{P}_r}\left[f(x_r)\right] - \mathbb{E}_{x_g \sim \mathbb{P}_g}\left[f(x_g)\right], \tag{3}$$

where $\|\cdot\|_L \le 1$ 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[2] D^W_θ and maximizing the expression over parameters θ. This choice leads to the following expression, which encapsulates the optimization procedure for the discriminator and generator networks in the conditional setting:

$$\min_\mu \max_{\theta : \|D^W_\theta\|_L \le 1} \mathbb{E}_{x_r \sim \mathbb{P}_r}\left[D^W_\theta(\bar{x}, x_r)\right] - \mathbb{E}_{x_g \sim \mathbb{P}_{G_\mu}}\left[D^W_\theta(\bar{x}, x_g)\right]. \tag{4}$$

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 generated and real data are indistinguishable. The training procedure for this loss is performed by taking gradient steps to optimize the discriminator while holding the generator's parameters fixed, and the generator while holding the discriminator's parameters fixed.

Notice that the optimization of θ requires that D^W_θ is 1-Lipschitz throughout the process. The most popular way of enforcing this constraint is by introducing a penalty term (Gulrajani et al. 2017) in the discriminator's optimization steps, giving the updated expression:

$$\min_\mu \max_{\theta : \|D^W_\theta\|_L \le 1} \mathbb{E}_{x_r \sim \mathbb{P}_r}\left[D^W_\theta(\bar{x}, x_r)\right] - \mathbb{E}_{x_g \sim \mathbb{P}_{G_\mu}}\left[D^W_\theta(\bar{x}, x_g)\right]$$
$$- \lambda \mathbb{E}_{x_\tau \sim \mathbb{P}_{x_\tau}}\left[(\|\nabla_{x_\tau} D^W_\theta(\bar{x}, x_\tau)\| - 1)^2\right], \tag{5}$$

where $\lambda > 0$ is a hyperparameter controlling how strongly to enforce the penalty term and x_τ is a point drawn from somewhere in the space of $\mathbb{P}_r$ or $\mathbb{P}_{G_\mu}$. Specifically, x_τ is generated as an interpolation between a pair of real and generated samples: $x_\tau = \tau x_r + (1-\tau)x_g$ with $x_r \sim \mathbb{P}_r$, $x_g \sim \mathbb{P}_{G_\mu}$ and $\tau \sim \mathrm{U}(0,1)$. For a detailed derivation of this loss term, see Lemma 1 of Gulrajani et al. (2017). This method, because it combines a gradient penalty with the WGAN, is known as GP-WGAN.

Algorithm 1 SGAN Training Algorithm for learning to match an observed probability distribution. Default values: number of discriminator iterations per generator iteration $C = 5$; batch size $B = 32$; hyper-parameter $\delta = 1$; $l = 128$

```
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 $G_\mu$.
 1: while $\theta$ has not converged do
 2:   for $c = 1, \cdots, C$ do
 3:     for $b = 1, \cdots, B$ do
 4:       Sample real transition pair $(\bar{x}, x_r)$ from dataset
 5:       Sample noise vector $\mathfrak{n} \sim \mathrm{U}[0,1]^l$
 6:       $x_g \leftarrow G_\mu(\bar{x}, \mathfrak{n})$ //generate next state
 7:       $T_b \leftarrow \frac{\|x_r - x_g\|}{\delta}$
 8:       for $t = 1, \cdots, T_b$ do
 9:         Sample $\tau \sim \mathrm{U}[0,1]$
10:         $x_\tau \leftarrow \tau x_r + (1-\tau)x_g$
11:         $L^{(b,t)}_D \leftarrow (\|\nabla_{x_\tau} D^S_\theta(\bar{x}, x_\tau) - \frac{x_r - x_g}{\|x_r - x_g\|}\|)^2$
12:       end for
13:     end for
14:     $\theta \leftarrow \theta - \nabla_\theta \frac{1}{B}\sum_{b=1}^{B} \frac{1}{T_b}\sum_{t=1}^{T_b} L^{(b,t)}_D$
15:   end for
16:   for $b = 1, \cdots, B$ do
17:     Sample noise vector $\mathfrak{n} \sim \mathrm{U}[0,1]^l$.
18:     $L^b_G \leftarrow -D^S_\theta(G_\mu(\bar{x}, \mathfrak{n}))$
19:   end for
20:   $\mu \leftarrow \mu - \nabla_\mu \frac{1}{B}\sum_{b=1}^{B} L^b_G$
21: 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 $x_g = G_\mu(\bar{x}, \mathfrak{n})$, where the components of the noise vector $\mathfrak{n}$ are drawn from $\mathrm{U}[0,1]$ and G_μ denotes the generator with parameter μ.

[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.

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

- Train the discriminator D_θ^S with parameter θ to minimize L_D^{SGAN}, which we will define later.
- Train the generator G_μ with its loss $L_G = \mathbb{E}_{\mathfrak{n}\sim\text{U}[0,1]}\{-D_\theta^S(G_\mu(\bar{x},\mathfrak{n}))\}$.

Apart from the new loss function L_D^{SGAN}, the above procedure is shared with that of WGANs and GP-WGANs.

Now, we define the new discriminator loss function L_D^{SGAN},

$$L_D^{\text{SGAN}} = \mathbb{E}_{x_r\sim\mathbb{P}_r, x_g\sim\mathbb{P}_g, \tau\sim\text{U}[0,1]} \left[(\|\nabla_{x_\tau} D_\theta^S(\bar{x}, x_\tau) - \frac{x_r - x_g}{\|x_r - x_g\|}\|)^2\right], \quad (6)$$

where x_τ is

$$x_\tau = \tau x_r + (1-\tau)x_g, \quad (7)$$
$$\tau \sim \text{U}[0,1]. \quad (8)$$

But, unlike the way x_τ is sampled in the GP-WGAN, the SGAN samples x_τ for T times given each x_r, x_g pair. The value T is computed by

$$T = \frac{\|x_r - x_g\|}{\delta}, \quad (9)$$

where δ is a hyper-parameter of the algorithm.

Training the discriminator in SGANs

By executing Algorithm 1, our D_θ^S is modelling a different discriminator from the D_θ^W 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^*}:

$$D^{S^*}(x) = \int_0^x (\int_x^1 \mathbb{P}_r(\hat{x})d\hat{x} - \int_x^1 \mathbb{P}_g(\hat{x})d\hat{x})dx.$$

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_τ 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: $x_\tau \overset{T}{=} x_n$, defined to means we sample x_τ T times and the $x_\tau = x_n$ at least once. To be clear, x_τ x_r, x_g are all random variables while x_n represents a specific fixed value. Assuming*

$$T = |x_r - x_g|/\delta, \quad (10)$$

it follows that

$$P(x_\tau \overset{T}{=} x_n | x_r, x_g) \quad (11)$$
$$= \begin{cases} c & x_r < x_n < x_g, x_g < x_n < x_r \\ 0 & else, \end{cases} \quad (12)$$

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 ε. 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

$$P(\check{x}_\tau \overset{1}{=} \check{x}_n | \check{x}_r, \check{x}_g) = \begin{cases} \frac{1}{|\check{x}_r - \check{x}_g|/\varepsilon} & \check{x}_r < \check{x}_n < \check{x}_g, \check{x}_g < \check{x}_n < \check{x}_r \\ 0 & \text{otherwise.} \end{cases} \quad (13)$$

If we sample $\check{x}_\tau$ for T times,

$$P(\check{x}_\tau \overset{T}{=} \check{x}_n | \check{x}_r, \check{x}_g) \quad (14)$$
$$= 1 - (1 - P(\check{x}_\tau \overset{1}{=} \check{x}_n | \check{x}_r, \check{x}_g))^T$$
$$= \begin{cases} 1 - (1 - \frac{1}{d/\varepsilon})^{d/\delta} & \check{x}_r < \check{x}_n < \check{x}_g, \check{x}_g < \check{x}_n < \check{x}_r \\ 0 & \text{otherwise.} \end{cases}$$

We relate ε to δ via a positive integer multiple z:

$$\delta = z\varepsilon, \quad (15)$$

where $z \in Z^+$. To connect back to Equation (14), we consider following limit,

$$\begin{aligned} &\lim_{\delta = z\varepsilon, \varepsilon\to 0} (1 - \frac{1}{d/\varepsilon})^{d/\delta} \\ &= \lim_{\delta = z\varepsilon, \varepsilon\to 0} e^{d/\delta \ln(1 - \frac{1}{d/\varepsilon})} \\ &= \lim_{\delta = z\varepsilon, \varepsilon\to 0} e^{\frac{\ln(\frac{d-\varepsilon}{d})}{\delta/d}} \\ &= \lim_{\varepsilon\to 0} e^{\frac{\frac{d}{d-\varepsilon}\frac{-1}{d}}{z/d}} \\ &= e^{-1/z} \end{aligned} \quad (16)$$

Substituting Equation (16) into Equation (14) and taking the limit as $\varepsilon \to 0$, we have

$$P(x_\tau \overset{T}{=} x_n | x_r, x_g) \quad (17)$$
$$= \lim_{\delta = z\varepsilon, \varepsilon\to 0} P(\check{x}_\tau \overset{T}{=} \check{x}_n | \check{x}_r, \check{x}_g)$$
$$= \begin{cases} 1 - e^{-1/z} & x_r < x_n < x_g, x_g < x_n < x_r \\ 0 & \text{otherwise,} \end{cases} \quad (18)$$

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_θ^S with loss*

$$L = (|\nabla_{x_\tau} D_\theta^S(\bar{x}, x_\tau) - \frac{x_r - x_g}{|x_r - x_g|}|)^2, \quad (19)$$

then $D_\theta^S(x)$ approaches

$$D_\theta^S(x) = c\, D^{S^*}(x), \quad (20)$$

for an undefined constant c.

Proof of Theorem 1: Equation (19) encourages $\nabla_{x_\tau} D_\theta^S(x_\tau)$ to approach $\frac{x_r - x_g}{|x_r - x_g|}$. Since x_r and x_g are random variables, $\nabla_{x_\tau} D_\theta^S(x_\tau)$ is updated toward $+1$ and -1 randomly. As a result, we should consider the learned value of $\nabla_{x_\tau} D_\theta^S(x_\tau)$ as it relates to the probability that it gets different updates. Let us take a look at $\nabla_{x_\tau} D_\theta^S(x_\tau)$ at an arbitrary point x_n:

$$\begin{aligned}
&\mathbb{E}_{x_r \sim \mathbb{P}_r, x_g \sim \mathbb{P}_g, \tau \sim \mathrm{U}[0,1]} \left[\nabla_{x_\tau = x_n} D_\theta^S(x_\tau = x_n, \delta)\right] \\
&= \mathbb{E}_{x_r \sim \mathbb{P}_r, x_g \sim \mathbb{P}_g, \tau \sim \mathrm{U}[0,1]} \left[\frac{x_r - x_g}{|x_r - x_g|}\right] \\
&= P(x_\tau \overset{T}{=} x_n | x_g < x_n < x_r) P(x_g < x_n < x_r) \\
&\quad -P(x_\tau \overset{T}{=} x_n | x_r < x_n < x_g) P(x_r < x_n < x_g).
\end{aligned} \tag{21}$$

From Lemma 1, we know that

$$P(x_\tau \overset{T}{=} x_n | x_g < x_n < x_r) = c \tag{22}$$

and

$$P(x_\tau \overset{T}{=} x_n | x_r < x_n < x_g) = c, \tag{23}$$

for some hyper-parameter controlled constant c. In the context of Equation (21), we have

$$\begin{aligned}
&\nabla_{x_\tau = x_n} D_\theta^S(x_\tau = x_n) \\
&= [P(x_g < x_n < x_r) - P(x_r < x_n < x_g)]c \\
&= [P(x_g < x_n)P(x_n < x_r) \\
&\quad -P(x_r < x_n)P(x_n < x_g)]c \\
&= [\int_0^{x_n} \mathbb{P}_g(x)dx \int_{x_n}^1 \mathbb{P}_r(x)dx \\
&\quad - \int_0^{x_n} \mathbb{P}_r(x)dx \int_{x_n}^1 \mathbb{P}_g(x)dx]c \\
&= \left[\int_{x_n}^1 \mathbb{P}_r(x)dx - \int_{x_n}^1 \mathbb{P}_g(x)dx\right] c.
\end{aligned} \tag{24}$$

Further, based on Equation (24),

$$\begin{aligned}
&\mathbb{E}_{x_r \sim \mathbb{P}_r, x_g \sim \mathbb{P}_g, \tau \sim \mathrm{U}[0,1]} \left[D_\theta^S(x_\tau = x_n)\right] \\
&= \left[\int_0^{x_n} \left(\int_x^1 \mathbb{P}_r(\hat{x})d\hat{x} - \int_x^1 \mathbb{P}_g(\hat{x})d\hat{x}\right) dx\right] c \\
&= \left[D^{S^*}(x_\tau = x_n)\right] c,
\end{aligned} \tag{25}$$

completing the proof.

Experiments

This section presents experimental results.

Comparison Algorithms

We compare SGAN against a tabular learner, a deterministic deep network and the state-of-the-art[3] GP-WGAN.

Given a set $\mathbb{S}$ of $\langle \bar{x}, x_r \rangle$ samples, our tabular learner simply memorizes all of them. It then estimates $\mathbb{P}_r^{\bar{x}}(x') = |\{\langle \bar{x}, x' \rangle \in \mathbb{S}\}| / |\{\langle \bar{x}, \cdot \rangle \in \mathbb{S}\}$. 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 keep the network structure uniform as possible. For all 3D convolutional neural networks (3D-CNN) layers, 3D deconvolutional neural networks (3D-DCNN) 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 $0 \times 1 \times 1$, with sizes reported as Depth $\times$ Height $\times$ Width. So we denote one of such 3D-CNN layer as, 3D-CNN^D, where D denotes different depth in kernel size $D \times 4 \times 4$. Similarly, we denote such 3D-DCNN layer as, 3D-DCNN^D. For all fully connected (FC) layers, we used LeakyReLU activation with a negative slope of 0.001. So, we denote a FC layer mapping size a to size b as FC$^{a \to b}$. We first define a few special terms to describe the network structure,

- *Squeeze layer*. This layer always appears after a 3D-CNN layer, and it first flattens the output of the 3D-CNN layer, then uses a FC layer mapping the flattened vector to 512.
- *Concatenate Layer*. This layer always appears after a FC layer of size 512, it concatenates the output of the FC layer with noise vector $\mathfrak{n}$, which means it is a FC layer mapping from (512+l) to 512, and $l = 128$. Note that deterministic deep network replace $\mathfrak{n}$ with a zero vector of the same size.
- *Unsqueeze layer*. This layer appears after a FC layer of size 512 and before a 3D-DCNN layer, it first uses an FC layer that maps 512 to the size of the input of the following 3D-DCNN layer, then reshape the output vector to the shape of the input of the following 3D-DCNN layer.
- *Linear output layer*. This layer appears after a FC layer of size 512, it is a linear layer that maps 512 to 1.
- $[*] \to [*] \to [*]$. A sequence of layers.

Network structure for all grid domains. For vector representations, we denote the size of vector as V. Deterministic deep network, G of GP-WGAN and G of SGAN, use the same structure of:

- *Input Layers*. [FC$^{V \to 512}$] $\to$ [FC$^{512 \to 512}$]
- *Squeeze layer*.
- *Concatenate Layer*.
- *Unsqueeze layer*.
- *Output Layers*. [FC$^{512 \to 512}$] $\to$ [FC$^{512 \to V}$]

For vector representations, D of GP-WGAN and D of SGAN, use the same structure of:

- *Input Layers*. [FC$^{2V \to 512}$] $\to$ [FC$^{512 \to 512}$]
- *Squeeze layer*.
- *Linear output layer*.

For image representations, deterministic deep network, G of GP-WGAN and G of SGAN, use the same structure of:

- *Input Layers*. [3D-CNN^1] $\to$ [3D-CNN^1]
- *Squeeze layer*.

[3]In preliminary work, we evaluated GAN and WGAN and found they were consistently worse than GP-WGAN.

- *Concatenate Layer.*
- *Unsqueeze layer.*
- *Output layers.* [3D-DCNN^1] $\rightarrow$ [3D-DCNN^1]

For image representations, D of GP-WGAN and D of SGAN, use the same structure of:

- *Input Layers.* [3D-CNN^2] $\rightarrow$ [3D-CNN^1]
- *Squeeze layer.*
- *Linear output layer.*

Network structure for marble domain. Deterministic deep network, G of GP-WGAN and G of SGAN, use the same structure of:

- *Input Layers.* [3D-CNN^2] $\rightarrow$ [3D-CNN^1] $\rightarrow$ [3D-CNN^1]
- *Squeeze layer.*
- *Concatenate Layer.*
- *Unsqueeze layer.*
- *Output layers.* [3D-DCNN^1] $\rightarrow$ [3D-DCNN^1] $\rightarrow$ [3D-DCNN^1]

D of GP-WGAN and D of SGAN, use the same structure of:

- *Input Layers.* [3D-CNN^2] $\rightarrow$ [3D-CNN^2] $\rightarrow$ [3D-CNN^1]
- *Squeeze layer.*
- *Linear output layer.*

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 find 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 how often the model produces meaningful samples. When possible, we resample after an uninterpretable 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 use block size of 4, which means every box is 4×4 in pixel size.

Our test domains include simple and complex domains, described next.

Simple Domains

We first investigate SGAN on two simple domains. States in the *1D Grid* domain consist of n-bit vectors in which exactly one component has the value of 1. (zzzs: then may be we should remove image representations on 1D grid?) The dynamics are a transition of $\frac{1}{3}$ probability of moving left and $\frac{2}{3}$ probability moving right. At the far left and right of the vector, it bounces. zzzs: it bounces, I do not know if I can use this word, so I leave this zzzs here We ran experiments for $n \in \{5, 10, 20\}$.

States in the *2D Grid* domain consisted of two-dimensional images with a grid size of 5×5. We examined two different

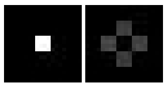


Figure 1

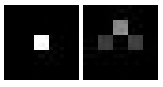


Figure 2

Figure 3: Transition pairs modelled by deterministic deep network on 5×5 *2D Grid*. For every subfigure, the image on the left side is the start state, and the image on the right side is the modelled next state. Subfigure 1 shows the learned transition pair under the dynamic of a uniform random walk on the 4 cardinal directions $(0.25 : 0.25 : 0.25 : 0.25)$. Subfigure 1 shows the learned transition pair under the dynamic of Russell-Norvig (Russell and Norvig 1994) grid dynamics $(0.8 : 0.1 : 0.0 : 0.1)$.

transition dynamics for these grids: a uniform random walk on the 4 cardinal directions $(0.25 : 0.25 : 0.25 : 0.25)$ and the Russell-Norvig (Russell and Norvig 1994) grid dynamics $(0.8 : 0.1 : 0.0 : 0.1)$ with the intended movement direction being north. Transition that would take the agent out of the grid result in no state change.

As shown in Table 2, our SGAN performs well for most of the simple domains. The average improvement of SGAN over GP-WGAN was 0.128 on L1 loss and 3% on sample validity.

Unsurprisingly, SGAN performed worse than the tabular learners in these domains—it is effectively matching ground truth in these cases as there is more than enough data to tabulate all possible inputs and their associated outputs.

As show in Figure 3, the deterministic deep net cannot generate any valid samples despite the simplicity of these problems. Actually, we can see that deterministic models the transitions into 'Gray', determined by the probability of occurance.

Complex Domains

We also evaluate our SGAN in three more complicated domains.

The *2D Grid with Obstacle* domain is identical to the 2D Grid domain, except an impassable object is included in the middle of the grid. Note that in *2D Grid with Obstacle* domain, agent is represented by pixel value to be 1, and obstacle is represented by pixel value to be 0.5.

The *2D Grid with Random Backgrounds* domain is another 2D Grid domain in which each background box is either empty or an obstacle randomly. Each grid cell is represented by two kind of features—'fence feature' denotating the presence of an obstacle and 'one feature' denotating the presence of the agent. Figure 7 illustrates the domain, for each column on the left side of each subfigure, they are start states with the agent always at the same place in the middle ,represented by the 'one feature', and different collections of backgrounds represented by 'fence feature'. For The *2D Grid with Random Backgrounds*, the next state includes the backgrounds and therefore the learner needs to learn how to copy it forward. Since the possibility of backgrounds is enormous, models have to generalize with limited data set. Note the the size

L1 Loss/Valid Percentage	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%
	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	0.8:0.1:0.0:0.1	**0.018/100%**	/0%	0.180/92%	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 1: Results on Simple Domains.

L1 Loss/ Valid Percentage	Representation	Size	Dynamic	Tabular learner	Deterministic deep network	GP-WGAN	SGAN
2D Grid with Obstacles	Image	5	0.8:0.1:0.0:0.1	**0.021/100%**	/0%	0.099/96%	0.098/97%
			0.25:0.25:0.25:0.25	**0.018/100%**	/0%	0.151/92%	0.120/94%
2D Grid with Random Background	Image	5	0.8:0.1:0.0:0.1	2.000/100%	/0%	0.255/90%	**0.118/93%**
		5	0.25:0.25:0.25:0.25	2.000/100%	/0%	0.619/71.4%	**0.161/91%**
	Overall			1.009/100%	/0%	0.281/87.35%	**0.124/94%**
Marble	Image	/	/	/	Up:0% Down:100% Invalid:0%	Up:80% Down:0% Invalid:20%	**Up:43%** **Down:34%** **Invalid:23%**

Table 2: Results on Complex Domains.

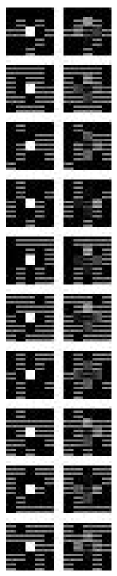


Figure 4

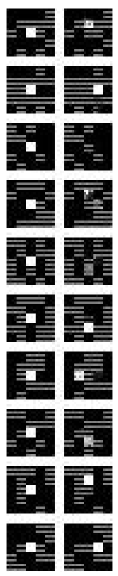


Figure 5

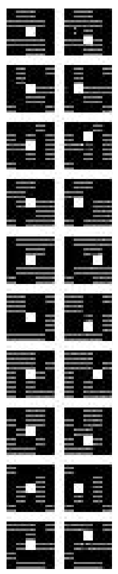


Figure 6

Figure 7: Transition pairs generated by deterministic deep network (Figure 4), WG-WGAN (Figure 5) and SGAN (Figure 6) on 5×5 *2D Grid with Random Backgrounds* domain under the dynamic of a uniform random walk ($0.25 : 0.25 : 0.25 : 0.25$). For every subfigure, the column if images on the left side is the start state (agent fixed to the same position for evaluation but backgrounds are random), and the column of images on the right side is the modelled next state based on that start state.

of the data set we use for this domain is 10^{-6} of the total number of possible transition pairs, which makes memorize things impossible.

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 connection to $\mathfrak{n}$ 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 $\mathfrak{n}$. To encourage the network to retain these connections, we have an optional additional loss in G:

$$L_G^{\mathfrak{n}} = -\log(1 + \|\nabla_{\mathfrak{n}} G_\mu(\bar{x}, \mathfrak{n})\|), \tag{26}$$

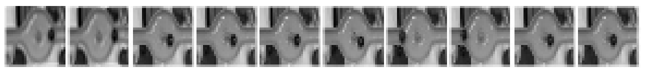
where the $\log(1 + \cdot)$ operation is to restrain this loss when it gets big. We combine this loss with the original loss of G by simple addition with a weighting coefficient of $\rho = 1.0$. We only implement this technique on the three complicated domains, when the deterministic part in the transition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

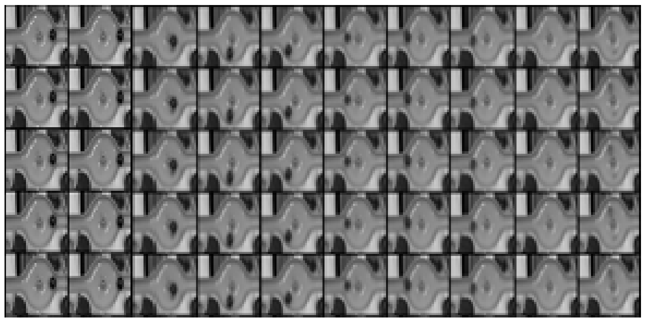
As shown in Table 2, our SGAN results in improvements in most of the domains in terms of L1 loss and percentage of valid outputs. In these complex domains, SGAN outperforms the tabular learner, specifically because generalization is absolutely necessary for capturing the large number of possible states. Once again, the deterministic deep net cannot generate any valid samples. Figure 7 gives a visualization from deterministic deep network, GP-WGAN and SGAN on the *2D Grid with Random Backgrounds* domain. From this figure we can see that all neural network base method can learn copying background forward, even though provide with insufficient data, which shows the neural network based method all have the ability to generalize with insufficient data. But only GP-WGAN and SGAN can model stochastic transitions and SGAN is doing a better job.

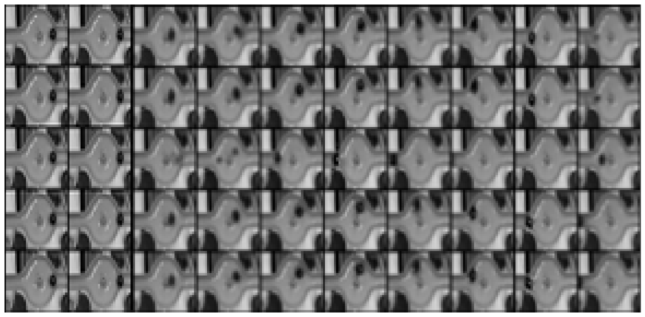
Physical Domain

In addition to these simulated video-game-like domains, we built a real-world experimental apparatus where the stochasticity derives 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 with FPS=30. To focus on the interesting part of the stochastic transitions, we filtered out the frames without marble in it. This filter is simply accomplished by compute the pixel-wised variance of frames and set a threshold to judge if there is a marble in images moving around. After that, we have 22427 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. Figure 8 shows several $(\bar{x}, x_r)$ pairs from this domain. We generate a sequence of marble images starting from two frames. Figure 8 shows one real sequence and multiple generated sequences from deterministic deep net, GP-WGAN, and SGAN, respectively. We can see from these figures that the deterministic deep net can generate valid next sequences but the sequence is always identical, which means it fails to model stochastic transitions. GP-WGAN can generate multiple outputs coming from the sequences, but the sequences are relatively more deterministic, compared to that of SGAN. The SGAN can generate multiple sequences of diverse transitions, which means it works best to model this stochastic dynamic system. As shown in the last row of Table 2, we also reported a manually counted result on whether the generated sequences shows the marble going up or down or invalid over 30 random generated samples. Once can also see that our SGAN perform best to model this stochastic transition. However, the generated images from SGAN are of lower quality than those from the deterministic deep net or GP-WGAN. We currently have no insight into why this happens and we leave it to future research to study in detail.


(a) Real marble sequence


(b) Generated Marble sequence from deterministic deep network.


(c) Generated Marble sequence from GP-WGAN.

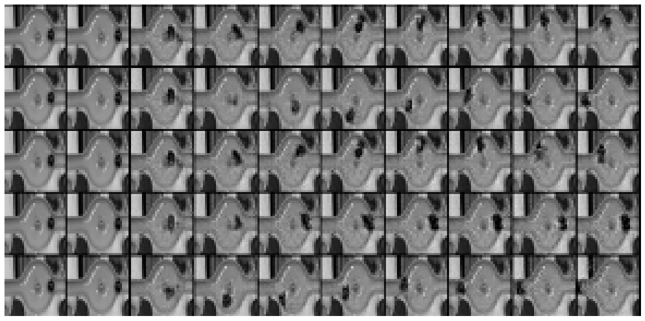

(d) Generated Marble sequence from SGAN.

Figure 8: One real marble sequence and multiple generated sequences from deterministic deep net, GP-WGAN, and SGAN, respectively. Each row is a generated sequence. Time increases from left to right. We report 5 sequences for each methods. Every later images is generated based on previous two images.

Future Work

We are planning using the proposed SGAN to model stochastic transitions in model-based reinforcement learning. As we as we will investigate more on why SGAN performs worse in real world domain when consider the quality of generated images.

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.