Learning Dynamic Generator Model by Alternating Back-Propagation through Time

Jianwen Xie, Ruiqi Gao, Zilong Zheng, Song-Chun Zhu, Ying Nian Wu
1Hikvision Research Institute, Santa Clara, USA 2University of California, Los Angeles, USA

Abstract
This paper studies the dynamic generator model for spatial-temporal processes such as dynamic textures and action sequences in video data. In this model, each time frame of the video sequence is generated by a generator model, which is a non-linear transformation of a latent state vector, where the non-linear transformation is parametrized by a top-down neural network. The sequence of latent state vectors follows a non-linear auto-regressive model, where the state vector of the next frame is a non-linear transformation of the state vector of the current frame as well as an independent noise vector that provides randomness in the transition. The non-linear transformation of this transition model can be parametrized by a feedforward neural network. We show that this model can be learned by an alternating back-propagation through time algorithm that iteratively samples the noise vectors and updates the parameters in the transition model and the generator model. We show that our training method can learn realistic models for dynamic textures and action patterns.

1 Introduction
1.1 The model
Most physical phenomena in our visual environments are spatial-temporal processes. In this paper, we study a generative model for spatial-temporal processes such as dynamic textures and action sequences in video data. The model is a non-linear generalization of the linear state space model proposed by (Doretto et al. 2003) for dynamic textures. The model of (Doretto et al. 2003) is a hidden Markov model, which consists of a transition model that governs the transition probability distribution in the state space, and an emission model that generates the observed signal by a mapping from the state space to the signal space. In the model of (Doretto et al. 2003), the transition model is an auto-regressive model in the d-dimensional state space, and the emission model is a linear mapping from the d-dimensional state vector to the D-dimensional image. In (Doretto et al. 2003), the emission model is learned by treating all the frames of the input video sequence as independent observations, and the linear mapping is learned by principal component analysis via singular value decomposition. This reduces the D-dimensional image to a d-dimensional state vector. The transition model is then learned on the sequence of d-dimensional state vectors by a first order linear auto-regressive model.

Given the high approximation capacity of the modern deep neural networks, it is natural to replace the linear structures in the transition and emission models of (Doretto et al. 2003) by the neural networks. This leads to the following dynamic generator model that has the following two components. (1) The emission model, which is a generator network that maps the d-dimensional state vector to the D-dimensional image via a top-down deconvolution network. (2) The transition model, where the state vector of the next frame is obtained by a non-linear transformation of the state vector of the current frame as well as an independent Gaussian white noise vector that provides randomness in the transition. The non-linear transformation can be parametrized by a feedforward neural network or multi-layer perceptron. In this model, the latent random vectors that generate the observed data are the independent Gaussian noise vectors, also called innovation vectors in (Doretto et al. 2003). The state vectors and the images can be deterministically computed from these noise vectors.

1.2 The learning algorithm
Such dynamic models have been studied in the computer vision literature recently, notably (Tulyakov et al. 2017). However, the models are usually trained by the generative adversarial networks (GAN) (Goodfellow et al. 2014) with an extra discriminator network that seeks to distinguish between the observed data and the synthesized data generated by the dynamic model. Such a model may also be learned by variational auto-encoder (VAE) (Kingma and Welling 2014) together with an inference model that infers the sequence of noise vectors from the sequence of observed frames. Such an inference model may require a sophisticated design.

In this paper, we show that it is possible to learn the model on its own using an alternating back-propagation through time (ABPTT) algorithm, without recruiting a separate discriminator model or an inference model. The ABPTT algorithm iterates the following two steps. (1) Inferential back-propagation through time, which samples the sequence of noise vectors given the observed video sequence using the Langevin dynamics, where the gradient of the log posterior
distribution of the noise vectors can be calculated by back-propagation through time. (2) Learning back-propagation through time, which updates the parameters of the transition model and the emission model by gradient ascent, where the gradient of the log-likelihood with respect to the model parameters can again be calculated by back-propagation through time.

The alternating back-propagation (ABP) algorithm was originally proposed for the static generator network (Han et al. 2017). In this paper, we show that it can be generalized to the dynamic generator model. In our experiments, we show that we can learn the dynamic generator models using the ABPTT algorithm for dynamic textures and action sequences.

Two advantages of the ABPTT algorithm for the dynamic generator models are convenience and efficiency. The algorithm can be easily implemented without designing an extra network. Because it only involves back-propagations through time with respect to a single model, the computation is very efficient.

1.3 Related work

The proposed learning method is related to the following themes of research.

Dynamic textures. The original dynamic texture model (Doretto et al. 2003) is linear both in the transition model and the emission model. Our work is concerned with a dynamic model with non-linear transition and emission models. See also (Tesfaldet, Brubaker, and Derpanis 2018) and references therein for some recent work on dynamic textures.

Chaos modeling. The non-linear dynamic generator model has been used to approximate chaos in a recent paper (Pathak et al. 2017). In the chaos model, the innovation vectors are given as inputs, and the model is deterministic. In contrast, in the model studied in this paper, the innovation vectors are independent Gaussian noise vectors, and the model is stochastic.

GAN and VAE. The dynamic generator model can also be learned by GAN or VAE. See (Tulyakov et al. 2017) (Saito, Matsumoto, and Saito 2017) and (Vondrick, Pirsiavash, and Torralba 2016) for recent video generative models based on GAN. However, GAN does not infer the latent noise vectors. In VAE (Kingma and Welling 2014), one needs to design an inference model for the sequence of noise vectors, which is a non-trivial task due to the complex dependency structure. Our method does not require an extra model such as a discriminator in GAN or an inference model in VAE.

Models based on spatial-temporal filters or kernels. The patterns in the video data can also be modeled by spatial-temporal filters by treating the data as 3D (2 spatial dimensions and 1 temporal dimension), such as a 3D energy-based model (Xie, Zhu, and Wu 2017) where the energy function is parametrized by a 3D bottom-up ConvNet, or a 3D generator model (Han et al. 2019) where a top-down 3D ConvNet maps a latent random vector to the observed video data. Such models do not have a dynamic structure defined by a transition model, and they are not convenient for predicting future frames.

1.4 Contribution

The main contribution of this paper lies in the combination of the dynamic generator model and the alternating back-propagation through time algorithm. Both the model and algorithm are simple and natural, and their combination can be very useful for modeling and analyzing spatial-temporal processes. The model is one-piece in the sense that (1) the transition model and emission model are integrated into a single latent variable model. (2) The learning of the dynamic model is end-to-end, which is different from (Han et al. 2017)'s treatment. (3) The learning of our model does not need to recruit a discriminative network (like GAN) or an inference network (like VAE), which makes our method simple and efficient in terms of computational cost and model parameter size.

2 Model and learning algorithm

2.1 Dynamic generator model

Let $X = (x_t, t = 1, ..., T)$ be the observed video sequence, where $x_t$ is a frame at time $t$. The dynamic generator model consists of the following two components:

$$s_t = F_\alpha(s_{t-1}, \xi_t),$$  \hspace{1cm} (1)

$$x_t = G_\beta(s_t) + \epsilon_t,$$  \hspace{1cm} (2)

where $t = 1, ..., T$. (1) is the transition model, and (2) is the emission model. $s_t$ is the $d$-dimensional hidden state vector. $\xi_t \sim N(0, I)$ is the noise vector of a certain dimensionality. The Gaussian noise vectors $\xi_t$ are independent of each other. The sequence of $\xi_t$ follows a non-linear auto-regressive model, where the noise vector $\xi_t$ encodes the randomness in the transition from $s_{t-1}$ to $s_t$ in the $d$-dimensional state space. $F_\alpha$ is a feedforward neural network or multi-layer perceptron, where $\alpha$ denotes the weight and bias parameters of the network. We can adopt a residual form (He et al. 2016) for $F_\alpha$ to model the change of the state vector. $x_t$ is the $D$-dimensional image, which is generated by the $d$-dimensional hidden state vector $s_t$. $G_\beta$ is a top-down convolutional network (sometimes also called deconvolution network), where $\beta$ denotes the weight and bias parameters of this top-down network. $\epsilon_t \sim N(0, \sigma^2 I)$ is the residual error. We let $\theta = (\alpha, \beta)$ denote all the model parameters.

Let $\xi = (\xi_t, t = 1, ..., T)$. $\xi$ consists of the latent random vectors that need to be inferred from $X$. Although $x_t$ is generated by the state vector $s_t$, $S = (s_t, t = 1, ..., T)$ are generated by $\xi$. In fact, we can write $X = H_\theta(\xi) + \epsilon$, where $H_\theta$ composes $F_\alpha$ and $G_\beta$ over time, and $\epsilon = (\epsilon_t, t = 1, ..., T)$ denotes the observation errors.

2.2 Learning and inference algorithm

Let $p(\xi)$ be the prior distribution of $\xi$. Let $p_\theta(X|\xi) \sim N(H_\theta(\xi), \sigma^2 I)$ be the conditional distribution of $X$ given $\xi$, where $I$ is the identity matrix whose dimension matches that of $X$. The marginal distribution is $p_\theta(X) = \int p(\xi)p_\theta(X|\xi)d\xi$ with the latent variable $\xi$ integrated out. We estimate the model parameter $\theta$ by the maximum likelihood method that maximizes the observed-data log-likelihood $\log p_\theta(X)$, which is analytically intractable. In contrast, the complete-data log-likelihood
log \ p_\theta(\xi, X), \text{ where } p_\theta(\xi, X) = p(\xi)p_\theta(X|\xi), \text{ is analytically tractable. The following identity links the gradient of the observed-data log-likelihood log \ p_\theta(X) \text{ to the gradient of the complete-data log-likelihood log } p_\theta(\xi, X):}

\frac{\partial}{\partial \theta} \log p_\theta(X) = \frac{1}{p_\theta(X)} \frac{\partial}{\partial \theta} \log p_\theta(X) = \frac{1}{p_\theta(X)} \left[ \int \frac{\partial}{\partial \theta} \log p_\theta(\xi, X) \right] p_\theta(\xi, X) d\xi = E_{p_\theta(\xi|X)} \left[ \frac{\partial}{\partial \theta} \log p_\theta(\xi, X) \right], \tag{3}

where p_\theta(\xi|X) = p_\theta(\xi, X)/p_\theta(X) is the posterior distribution of the latent \xi given the observed X. The above expectation can be approximated by Monte Carlo average. Specifically, we sample from the posterior distribution p_\theta(\xi|X) using the Langevin dynamics:

\xi^{(\tau+1)} = \xi^{(\tau)} + \frac{\delta^2}{2} \log p_\theta(\xi^{(\tau)}|X) + \delta z_\tau, \tag{4}

where \tau indexes the time step of the Langevin dynamics (not to be confused with the time step of the dynamics model, \tau), z_\tau \sim N(0, I) where I is the identity matrix whose dimension matches that of \xi, and \xi^{(\tau)} = (\xi_1^{(\tau)}, t = 1, ..., T) denotes all the sampled latent noise vectors at time step \tau. \delta is the step size of the Langevin dynamics. We can correct for the finite step size by adding a Metropolis-Hastings acceptance-rejection step. After sampling \xi \sim p_\theta(\xi|X) using the Langevin dynamics, we can update \theta by stochastic gradient ascent

\Delta \theta \propto \frac{\partial}{\partial \theta} \log p_\theta(\xi, X), \tag{5}

where the stochasticity of the gradient ascent comes from the fact that we use Monte Carlo to approximate the expectation in (3). The learning algorithm iterates the following two steps. (1) Inference step: Given the current \theta, sample \xi from p_\theta(\xi|X) according to (4). (2) Learning step: Given \xi, update \theta according to (5). We can use a warm start scheme for sampling in step (1). Specifically, when running the Langevin dynamics, we start from the current \xi, and run a finite number of steps. Then we update \theta in step (2) using the sampled \xi. Such a stochastic gradient ascent algorithm has been analyzed by (Younes 1999).

Since \frac{\partial}{\partial \xi} \log p_\theta(\xi|X) = \frac{\partial}{\partial \xi} \log p_\theta(\xi, X), both steps (1) and (2) involve derivatives of

\log p_\theta(\xi, X) = -\frac{1}{2} \left[ \|\xi\|^2 + \frac{1}{\sigma^2} \|X - H_\theta(\xi)\|^2 \right] + \text{const},

where the constant term does not depend on \xi or \theta. Step (1) needs to compute the derivative of \log p_\theta(\xi, X) with respect to \xi. Step (2) needs to compute the derivative of \log p_\theta(\xi, X) with respect to \theta. Both can be computed by back-propagation through time. Therefore the algorithm is an alternating back-propagation through time algorithm. Step (1) can be called inferential back-propagation through time. Step (2) can be called learning back-propagation through time.

To be more specific, the complete-data log-likelihood log \ p_\theta(\xi, X) can be written as (up to an additive constant, assuming \sigma^2 = 1)

L(\theta, \xi) = -\frac{1}{2} \sum_{t=1}^T \left[ \|x_t - G_\beta(s_t)\|^2 + \|\xi_t\|^2 \right]. \tag{6}

The derivative with respect to \beta is

\frac{\partial L}{\partial \beta} = \sum_{t=1}^T \left( x_t - G_\beta(s_t) \right) \frac{\partial G_\beta(s_t)}{\partial \beta} \frac{\partial s_t}{\partial \beta}, \tag{7}

The derivative with respect to \alpha is

\frac{\partial L}{\partial \alpha} = \sum_{t=1}^T \left( x_t - G_\beta(s_t) \right) \frac{\partial G_\beta(s_t)}{\partial \alpha} \frac{\partial s_t}{\partial \alpha} - \xi_t, \tag{8}

where \frac{\partial s_{t+1}}{\partial \xi_t} can again be computed recursively. To infer \xi, for any fixed time point t_0,

\frac{\partial L}{\partial \xi_{t_0}} = \sum_{t=t_0+1}^T \left( x_t - G_\beta(s_t) \right) \frac{\partial G_\beta(s_t)}{\partial \xi_t} \frac{\partial s_t}{\partial \xi_{t_0}} - \xi_{t_0}, \tag{9}

A minor issue is the initialization of the transition model. We may assume that s_0 \sim N(0, I). In the inference step, we can sample s_0 together with \xi using the Langevin dynamics.

It is worth mentioning the difference between our algorithm and the variational inference. While variational inference is convenient for learning a regular generator network, for the dynamic generator model studied in this paper, it is not a simple task to design an inference model that infers the sequence of latent vectors \xi = (\xi_t, t = 1, ..., T) from the sequence of X = (x_t, t = 1, ..., T). In contrast, our learning method does not require such an inference model and can be easily implemented. The inference step in our model can be done via directly sampling from the posterior distribution p_\theta(\xi|X), which is powered by back-propagation through time. Additionally, our model directly targets maximum likelihood, while model learning via variational inference is to maximize a lower bound.

### 2.3 Learning from multiple sequences

We can learn the model from multiple sequences of different appearances but of similar motion patterns. Let X^{(i)} = (x_t^{(i)}, t = 1, ..., T) be the i-th training sequence, i = 1, ..., n.

We can use an appearance (or content) vector a^{(i)} for each sequence to account for the variation in appearance. The model is of the following form

s_t^{(i)} = F_\alpha(s_{t-1}^{(i)}, \xi_t^{(i)}), \tag{10}

x_t^{(i)} = G_\beta(s_t^{(i)}, a^{(i)}) + e_t^{(i)}, \tag{11}

where a^{(i)} \sim N(0, I), and e^{(i)} is fixed over time for each sequence i. To learn from such training data, we only need to add the Langevin sampling of a^{(i)}. If the motion sequences are of different motion patterns, we can also introduce another vector m^{(i)} \sim N(0, I) to account for the variations
of motion patterns, so that the transition model becomes $s_t^{(i)} = F_{\alpha}(s_{t-1}^{(i)}, \varepsilon_t^{(i)}, m^{(i)})$ with $m^{(i)}$ fixed for the sequence $i$.

Recently (Tulyakov et al. 2017) studies a similar model where the transition model is modeled by a recurrent neural network (RNN) with another layer of hidden vectors. (Tulyakov et al. 2017) learns the model using GAN. In comparison, we use a simpler Markov transition model and we learn the model by alternating back-propagation through time. Even though the latent state vectors follow a Markovian model, the observed sequence is non-Markovian.

Algorithm 1 Learning and inference by alternating back-propagation through time (ABPTT)

Input: (1) training sequences $\{X^{(i)} = (x_t^{(i)}, t = 1, ..., n)\}$
(2) number of Langevin steps $l$
(3) number of learning iterations $N$.

Output: (1) learned parameters $\theta = (\alpha, \beta)$
(2) inferred noise vectors $\xi^{(i)} = (\xi_t^{(i)}, t = 1, ..., T)$.

1: Initialize $\theta = (\alpha, \beta)$. Initialize $\xi^{(i)}$ and $a^{(i)}$. Initialize $k = 0$.
2: repeat
3: Inferential back-propagation through time: For $i = 1, ..., n$, sample $\xi^{(i)}$ and $a^{(i)}$ by running $l$ steps of Langevin dynamics according to (4), starting from their current values.
4: Learning back-propagation through time: Update $\alpha$ and $\beta$ by gradient ascent according to (8) and (7).
5: Let $k \leftarrow k + 1$
6: until $k = N$

Algorithm 1 summarizes the learning and inference algorithm for multiple sequences with appearance vectors. If we learn from a single sequence such as dynamic texture, we can remove the appearance vector $a^{(i)}$, or simply fix it to a zero vector.

3 Related models

In this section, we shall review related models of spatial-temporal processes in order to put our work into the big picture.

3.1 Two related spatial-temporal models

Let $X = (x_t, t = 1, ..., T)$ be the observed sequence. We have studied the following energy-based model (Xie, Zhu, and Wu 2017):

$$p(X; \theta) = \frac{1}{Z(\theta)} \exp \left[ f_\theta(X) \right],$$

where $f_\theta(X)$ is a function of the whole sequence $X$, which can be defined by a bottom-up network that consists of multiple layers of spatial-temporal filters that capture the spatial-temporal patterns in $X$ at multiple layers. $\theta$ collects all the weight and bias parameters of the bottom-up network. The model can be learned by maximum likelihood, and the learning algorithm follows an “analysis by synthesis” scheme. The algorithm iterates (1) Synthesis: generating synthesized sequences from the current model by Langevin dynamics. (2) Analysis: updating $\theta$ based on the difference between the observed sequences and synthesized sequences. The two steps play an adversarial game with $f_\theta$ serving as a critic. The synthesis step seeks to modify the synthesized examples to increase $f_\theta$ scores of the synthesized examples, while the analysis step seeks to modify $\theta$ to increase the $f_\theta$ scores of the observed examples relative to the synthesized examples.

We have also studied the following generator model (Han et al. 2019)

$$s \sim N(0, I_d), \quad X = g_\theta(s) + \epsilon,$$

where the latent state vector $s$ is defined for the whole sequence, and is assumed to follow a prior distribution which is $d$-dimensional Gaussian white noise. The whole sequence is then generated by a function $g_\theta(s)$ that can be defined by a top-down network that consists of multiple layers of spatial-temporal kernels. $\theta$ collects all the weight and bias parameters of the top-down network. $\epsilon$ is the Gaussian noise image sequence. This generator model can be learned by maximum likelihood, and the learning algorithm follows the alternating back-propagation method of (Han et al. 2017).

In (Xie et al. 2018), we show that we can learn the above two models simultaneously using a cooperative learning scheme. We can also cooperatively train the energy-based model (12) and the dynamic generator model (1) and (2) studied in this paper simultaneously, where the dynamic generator model serves as an approximate sampler of the energy-based model.

Unlike the dynamic generator model (1) and (2) studied in this paper, the above two models (12) and (13) are not of a dynamic or causal nature in that they do not directly evolve or unfold over time.

3.2 Action, control, policy, and cost

If we observe the sequence of actions $a = (a_t, t = 1, ..., T)$ applied to the system, we can extend the forward dynamic model (1) to

$$s_t = F_\theta(s_{t-1}, a_t, \xi_t).$$

The model can still be learned by alternating back-propagation through time. With a properly defined cost function, we can optimize the sequence $a = (a_t, t = 1, ..., T)$ for control. We may also learn a policy $\pi(a_t | s_{t-1})$ directly from demonstrations by expert controllers. We may call the resulting model that consists of both dynamics and control policy as the controlled dynamic generator model.

We can also learn the cost function from expert demonstrations by inverse reinforcement learning (Ziebart et al. 2008) (Abbeel and Ng 2004), where we can generalize the above energy-based model (12) to $p_\theta(X; a) = \frac{1}{Z(\theta)} \exp[f_\theta(X, a)]$, where $-f_\theta(X, a)$ can be interpreted as the total cost. We can learn both the cost function and the policy cooperatively as in (Xie et al. 2018), where we fit the energy-based model to the demonstration data while using the controlled dynamic generator model as an approximate sampler of the energy-based model.
3.3 Velocity field, optical flow, and physics

In our work, the training data are image frames of video sequences. If we are given the velocity fields over time, we can also learn the dynamic generator model from such data, such as turbulence. Even with raw image sequences, it may still be desirable to learn a model that generates the velocity fields or optical flows over time, which in turn generate the image frames over time. This will lead to a more physically meaningful model of motion, which can be considered the mental physics. In our recent work on deformable generator network (Xing et al. 2018), we model the deformations explicitly. We can combine the deformable generator model and the dynamic generator model. We may also consider restricting the transition model to be linear to make the state vector close to real physical variables.

4 Experiments

4.1 Experiment 1: Learn to generate dynamic textures

We first learn the model for dynamic textures, which are sequences of images of moving scenes that exhibit stationarity in time. We learn a separate model from each example. The video clips for training are collected from DynTex++ dataset of (Ghanem and Ahuja 2010) and the Internet. Each observed video clip is prepared to be of the size 64 pixels \( \times \) 64 pixels \( \times \) 60 frames. We implement our model and learning algorithm in Python with Tensorflow (Abadi et al. 2015). The transition model is a feedforward neural network with three layers. The network takes a 100-dimensional state vector \( s_{t-1} \) and a 100-dimensional noise vector \( \xi_t \) as input and produces a 100-dimensional vector \( r_t \), so that \( s_t = \tanh(s_{t-1} + r_t) \). The numbers of nodes in the three layers of the feedforward neural network are \( \{20, 20, 100\} \). The emission model is a top-down deconvolution neural network or generator model that maps the 100-dimensional state vector (i.e., \( 1 \times 1 \times 100 \)) to the image frame of size 64 \( \times \) 64 \( \times \) 3 by 6 layers of deconvolutions with kernel size of 4 and up-sampling factor of 2 from top to bottom. The numbers of channels at different layers of the generator are \( \{512, 512, 256, 128, 64, 3\} \). Batch normalization (Ioffe and Szegedy 2015) and ReLU layers are added between deconvolution layers, and tanh activation function is used at the bottom layer to make the output signals fall between deconvolution layers, and tanh activation function is used at the bottom layer to make the output signals fall between 0 and 1.

We compare our model with three baseline methods, such as LDS (linear dynamic system) (Doretto et al. 2003), TwoStream (Tesfaldet, Brubaker, and Derpanis 2018), and MoCoGAN (Tulyakov et al. 2017), for dynamic texture synthesis in terms of “fooling” rate on 12 dynamic texture videos (e.g., waterfall, burning fire, waving flag, etc.). LDS represents dynamic textures by a linear autoregressive model; TwoStream method synthesizes dynamic textures by matching the feature statistics extracted from two pre-trained convolutional networks between synthesized and observed examples; and MoCoGAN is a motion and content decomposed generative adversarial network for video generation.

Figure 2 summarizes the comparative result by showing the “fooling” rate as a function of exposure time across methods. We can find that as the given exposure time becomes longer, it becomes easier for the users to observe the difference between the real and synthesized dynamic textures. More specifically, the “fooling” rate decreases as exposure time increases, and then remains at the same level for longer exposures. Overall, our method can generate more
realistic dynamic textures than other baseline methods. The result also shows that the linear model (i.e., LDS) outperforms the more sophisticated baselines (i.e., TwoStream and MoCoGAN). The reason is because when learning from a single example, the MoCoGAN may not fit the training data very well due to the unstable and complicated adversarial training scheme as well as a large number of parameters to be learned, and the TwoStream method has a limitation that it cannot handle dynamic textures that have structured background (e.g., burning fire heating a pot).

4.2 Experiment 2: Learn to generate action patterns with appearance consistency

We learn the model from multiple examples with different appearances by using a 100-dimensional appearance vector. We infer the appearance vector and the initial state via a 15-step Langevin dynamics within each iteration of the learning process. We learn the model using the Weizmann action dataset (Gorelick et al. 2007), which contains 81 videos of 9 people performing 9 actions, including jacking, jumping, walking, etc, as well as an animal action dataset that includes 20 videos of 10 animals performing running and walking collected from the Internet. Each video is scaled to 64×64 pixels × 30 frames. We adopt the same structure of the model as the one in Section 4.1, except that the emission model takes the concatenation of the appearance vector and the hidden state as input. For each experiment, a single model is trained on the whole dataset without annotations. The dimensions of the hidden state $s$ and the Gaussian noise $\xi$ are set to be 100 and 50 respectively for the Weizmann action dataset, and 3 and 100 for the animal action dataset.

Figure 3 shows some synthesized results for each experiment. To synthesize video, we randomly pick an appearance vector inferred from the observed video and generate new motion pattern for that specified appearance vector by the learned model with a noise sequence of $\{\xi_t, t = 1, ..., T\}$ and an initial state $s_0$ sampled from Gaussian white noise. We show two different synthesized motions for each appearance vector. With a fixed appearance, the learned model can generate diverse motions with consistent appearance.

Figure 4 shows two examples of video interpolation by
interpolating between appearance vectors of videos at the two ends. We conduct these experiments on some videos selected from categories “blooming” and “melting” in the dataset of (Zhou and Berg 2016). For each example, the videos at the two ends are generated with the appearance vectors inferred from two observed videos. Each video in the middle is obtained by first interpolating the appearance vectors of the two end videos, and then generating the videos using the dynamic generator. All the generated videos use the same set of noise sequence \( \{ \xi_t \} \) and \( s_0 \) randomly sampled from Gaussian white noise. We observe smooth transitions in contents and motions of all the generated videos and that the intermediate videos are also physically plausible.

We compare with MoCoGAN and TGAN (Saito, Matsumoto, and Saito 2017) by training on 9 selected categories (e.g., PlayingCello, PlayingDaf, PlayingDhol, PlayingFlute, PlayingGuitar, PlayingPiano, PlayingSitar, PlayingTabla, and PlayingViolin) of videos in the UCF101 (Soomro, Zamir, and Shah 2012) database and following (Saito, Matsumoto, and Saito 2017) to compute the inception score. Table 1 shows comparison results. Our model outperforms the MoCoGAN and TGAN in terms of inception score.

<table>
<thead>
<tr>
<th>Reference</th>
<th>ours</th>
<th>MoCoGAN</th>
<th>TGAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.05±0.16</td>
<td>8.21±0.09</td>
<td>4.40±0.04</td>
<td>5.48±0.06</td>
</tr>
</tbody>
</table>

4.3 Experiment 3: Learn from incomplete data

Our model can learn from videos with occluded pixels and frames. We adapt our algorithm to this task with minimal modification involving the computation of \( \sum_{t=1}^{T} ||x_t - G_\beta(s_t)||^2 \). In the setting of learning from fully observed videos, it is computed by summing over all the pixels of the video frames, while in the setting of learning from partially visible videos, we compute it by summing over only the visible pixels of the video frames. Then we can continue to use the alternating back-propagation through time (ABPTT) algorithm to infer \( \{ x_t \} \) and \( s_0 \), and then learn \( \beta \) and \( \alpha \). With inferred \( \{ x_t \} \) and \( s_0 \), and learned \( \beta \) and \( \alpha \), the video with occluded pixels or frames can be automatically recovered by \( G_\beta(s_t) \), where the hidden state can be recursively computed by \( s_t = F_\alpha(s_{t-1}, x_t) \).

Eventually, our model can achieve the following tasks: (1) recover the occluded pixels of training videos. (2) synthesize new videos by the learned model. (3) recover the occluded pixels of testing videos using the learned model. Different from those inpainting methods where the prior model has already been given or learned from fully observed training data, our recovery experiment is about an unsupervised learning task, where the ground truths of the occluded pixels are unknown in training the model for recovery. It is also
We have two types of occlusions: (1) single region mask occlusion, where a $60 \times 60$ mask is randomly placed on each $150 \times 150$ image frame of each video. (2) missing image frames, where 50% of the image frames are randomly blocked in each video. For each type of occlusion experiment, we measure the recovery errors by the average per pixel difference between the recovered video sequences and the original ones (The range of pixel intensities is $[0, 255]$), and compare with STGCN (Xie, Zhu, and Wu 2017), which is a spatial-temporal deep convolutional energy-based model that can recover missing pixels of videos by synthesis during the learning process. We also report results obtained by generic spatial-temporal Markov random field models with potentials that are $\ell_1$ or $\ell_2$ difference between pixels of nearest neighbors that are defined in both spatial and temporal domains, and the recovery is accomplished by synthesizing missing pixels via Gibbs sampling. Table 2 shows the comparison results. Some qualitative results for recovery by our models are displayed in Figure 5.

**Table 2: Recovery errors in occlusion experiments**

<table>
<thead>
<tr>
<th></th>
<th>ours</th>
<th>STGCN</th>
<th>MRF-$\ell_1$</th>
<th>MRF-$\ell_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>7.8782</td>
<td>8.1636</td>
<td>10.6586</td>
<td>12.5300</td>
</tr>
<tr>
<td>fountain</td>
<td>5.6988</td>
<td>6.0323</td>
<td>11.8299</td>
<td>12.1696</td>
</tr>
<tr>
<td>ocean</td>
<td>3.3966</td>
<td>3.4842</td>
<td>8.7498</td>
<td>9.8078</td>
</tr>
<tr>
<td>playing</td>
<td>4.9251</td>
<td>6.1375</td>
<td>15.6296</td>
<td>15.7085</td>
</tr>
<tr>
<td>sea world</td>
<td>5.6596</td>
<td>5.8850</td>
<td>12.0297</td>
<td>12.2868</td>
</tr>
<tr>
<td>windmill</td>
<td>6.6827</td>
<td>7.8858</td>
<td>11.7355</td>
<td>13.2036</td>
</tr>
<tr>
<td>Avg.</td>
<td>5.7068</td>
<td>6.2681</td>
<td>11.7722</td>
<td>12.6177</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>ours</th>
<th>STGCN</th>
<th>MRF-$\ell_1$</th>
<th>MRF-$\ell_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>5.0874</td>
<td>5.5992</td>
<td>10.7171</td>
<td>12.6317</td>
</tr>
<tr>
<td>fountain</td>
<td>5.5669</td>
<td>8.0311</td>
<td>19.4331</td>
<td>13.2251</td>
</tr>
<tr>
<td>ocean</td>
<td>3.3666</td>
<td>4.0428</td>
<td>9.0838</td>
<td>9.3913</td>
</tr>
<tr>
<td>playing</td>
<td>5.2563</td>
<td>7.6103</td>
<td>22.8287</td>
<td>17.5692</td>
</tr>
<tr>
<td>sea world</td>
<td>4.0682</td>
<td>5.4348</td>
<td>13.5101</td>
<td>12.9305</td>
</tr>
<tr>
<td>windmill</td>
<td>6.9267</td>
<td>7.5346</td>
<td>13.3364</td>
<td>12.9911</td>
</tr>
<tr>
<td>Avg.</td>
<td>5.0454</td>
<td>6.3791</td>
<td>14.7272</td>
<td>13.2065</td>
</tr>
</tbody>
</table>

**4.4 Experiment 4: Learn to remove content**

The dynamic generator model can be used to remove undesirable content in the video for background inpainting. The basic idea is as follows. We first manually mask the undesirable moving object in each frame of the video, and then learn the model from the masked video with the recovery algorithm that we used in Section 4.3. Since there are neither clues in the masked video nor prior knowledge to infer the occluded object, it turns out to be that the recovery algorithm will inpaint the empty region with the background.

Figure 6 shows two examples of removals of (a) a walking person and (b) a moving boat respectively. The videos are collected from Braham and Van Droogenbroeck (2016). For each example, the first row displays 5 frames of the original video. The second row shows the corresponding frames with masks occluding the target to be removed. The third row presents the inpainting results by our algorithm. The video shows...
2, and numbers of channels deconvolutions with kernel size of 4, up-sampling factor of
in Section 4.3, and an emission model that has 7 layers of
example (b). We adopt the same transition model as the one
the subsequent frames. We incorporate an extra encoder
ically, we learn a mapping from a static image frame to
be used for video prediction given a static image. Specif-
A conditional version of the dynamic generator model can
generate a sequence of images by the dynamic generator.
it the appearance vector and the initial state vector, which
transition model and emission model) together by alternat-
We learn both the encoder and the dynamic generator (i.e.,
model and emission model) together by alternat-
back-propagation through time. The appearance vector
and the initial state are no longer hidden variables that need
to be inferred in training. Once the model is learned, given
a testing static image, the learned encoder $E_s$ extracts from
it the appearance vector and the initial state vector, which
generate a sequence of images by the dynamic generator.

We test our model on burning fire dataset (Xie, Zhu, and
Wu 2017), and MUG Facial Expression dataset (N. Aifanti
and Delopoulos 2010). The encoder has 3 convolutional lay-
ers with numbers of channels \{64, 128, 256\}, filter sizes
\{5, 3, 3\} and sub-sampling factors \{2, 2, 1\} at different lay-
ers, and one fully connected layer with the output size equal
to the dimension of the appearance vector (100) plus the di-
ension of the hidden state (80). The dimension of $\xi$ is 20.
The other configurations are similar to what we used in Sec-
4.2. We qualitatively display some results in Figure 7,
where each row is one example of image-to-video predic-
tion. For each example, the left image is the static image
frame for testing, and the rest are 6 frames of the predicted
video sequence. The results show that the predicted frames
by our method have fairly plausible motions.

5 Conclusion

This paper studies a dynamic generator model for spatial-
temporal processes. The model is a non-linear generalization
of the linear state space model where the non-linear trans-
formations in the transition and emission models are param-
erized by neural networks. The model can be conveniently
and efficiently learned by an alternating back-propagation
through time (ABPTT) algorithm that alternatively samples
from the posterior distribution of the latent noise vectors and
then updates the model parameters. The model can be gen-
eralized by including random vectors to account for various
sources of variations, and the learning algorithm can still ap-
ply to the generalized models.

Project page

The code and more results can be found at http://www.stat.ucla.
edu/~jxie/DynamicGenerator/DynamicGenerator.html

Acknowledgement

Part of the work was done while R. G. was an intern at Hikvision
Research Institute during the summer of 2018. She thanks Director
Jane Chen for her help and guidance.
Figure 7: Image-to-video prediction. For each example, the first image is the static image frame, and the rest are 6 frames of the predicted sequence.

The work is supported by DARPA XAI project N66001-17-2-4029; ARO project W911NF1810296; ONR MURI project N00014-16-1-2007; and a Hikvision gift to UCLA.

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References


