Perception Updating Networks: On architectural constraints for interpretable video generative models

Eder Santana & Jose C Principe [*]
Department of Electrical and Computer Engineering
University of Florida
Gainesvillle, FL 32611, USA
edersantana@ufl.edu, principe@cnel.ufl.edu

Abstract

We investigate a neural network architecture and statistical framework that models frames in videos using principles inspired by computer graphics pipelines. The proposed model explicitly represents "sprites" or its percepts inferred from maximum likelihood of the scene and infers its movement independently of its content. We impose architectural constraints that forces resulting architecture to behave as a recurrent what-where prediction network.

1 Introduction

The current computer graphics pipelines are the result of efficient implementations required by limited hardware and high frequency output requirements. These requirements were also achieved with the use of explicit physics and optic constraints and modeling with constantly improving data structures (Shirley et al., 2015).

In machine learning on the other hand, for a long time image (Olshausen et al., 1996) and video (Hurri & Hyvärinen, 2003) generative models had been investigated with statistical approaches that model images down to the pixel level (Simoncelli & Olshausen, 2001), sometimes assuming neighborhood statistical dependencies (Osindero & Hinton, 2008). In video prediction, the current state of the art uses variations of deep convolutional recurrent neural networks (Kalchbrenner et al., 2016) (Lotter et al., 2016) (Finn et al., 2016).

Parallel to the classic machine learning approach to image and video interpretation and prediction, there is a growing trend in the deep learning literature for modeling vision as inverse graphics (Kulkarni et al., 2015)(Rezende et al., 2016)(Eslami et al., 2016). These approaches can be interpreted into two groups: supervised and unsupervised vision as inverse graphics. The supervised approach assumes that during training an image is provided with extra information about its rotation, translation, illumination, etc. The goal of the supervised model is to learn an auto-encoder that explicitly factors out the content of the image and its physical properties. The supervised approach is illustrated by Kulkarni et al. (2015).

The unsupervised approach requires extra architectural constraints, similar to those assumed in computer graphics. For example, Reed et al. (2016) modeled the content of a scene with a Generative Adversarial Network (Goodfellow et al., 2014) and its location with Spatial Transformer Networks (STN) (Jaderberg et al., 2015). The full model is adapted end-to-end to generate images whose appearance can be changed by independently modifying the "what" and/or "where" variables. A similar approach was applied to video generation with volumetric convolutional neural networks (Vondrick et al., 2016).In two papers by Google DeepMind (Rezende et al., 2016) (Eslami et al., 2016) they improved the "where" representations of the unsupervised approach and modeled the 3D geometry of the scene. This way they explicitly represented object rotation, translation, camera pose, etc. Their approaches were also trained end-to-end with REINFORCE-like stochastic gradients to backpropagate through non-differentiable parts of the graphics pipeline (Rezende et al.,

[*]Code repository: https://github.com/cnel/PUN

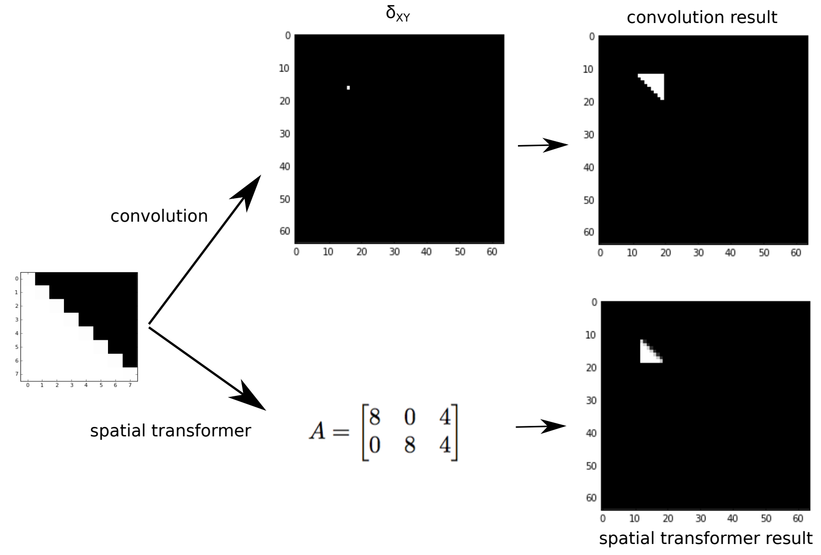
δ_XY
convolution result
convolution
spatial transformer
$A = \begin{bmatrix} 8 & 0 & 4 \\ 0 & 8 & 4 \end{bmatrix}$
spatial transformer result

Figure 1: How to get similar results using convolutions with delta-functions and spatial transformers. Input sprite is 8×8 pixels and the outputs are 64×64 pixels. Note that in the convolution the result shape is rotated 180 degrees and its center is where the delta equals to one at pixel $(x = 16, y = 16)$. Note also that the edges of the spatial transformer results are blurred due to bilinear interpolation. A matrix can be read as “zoom-out” 8 times and translate up and left in a quarter of the resulting size.

2016) or to count the number of objects in the scene (Eslami et al., 2016). Those papers also used STNs to model the position of the objects in the scene, but they extended STN to 3D geometry so it could also model rotation and translation in a volumetric space.

Other approaches inspired by the graphics pipeline and computer vision geometry in machine learning uses the physics constraints to estimate the depth of each pixel in the scene and camera pose movements to predict frames in video (Mahjourian et al., 2016) (Godard et al., 2016).

The present paper is closer to the unsupervised approach of vision as inverse graphics. More precisely, here we investigate frame prediction in video. Contrary to the work by Reed et al. (2016) here we first limit ourselves to simple synthetic 2D datasets and learning models whose representations can be visually interpreted. This way we can investigate exactly what the neural network is learning and validate our statistical assumptions. Also, we investigate the behavior of Spatial Transformer Networks and question it as the default choice when limited compute resources are available and no scale invariance is required.

First in the next Section we will pose a statistical model that is appropriate for machine learning but inspired by the graphics pipeline.

2 A 2D STATISTICAL GRAPHICS PIPELINE

This section starts with a high level description of the 2D graphics pipeline, followed by a discussion of how to implement it with neural network modules, and finally we define a formal statistical model.

The 2D graphics pipeline starts from geometric primitives and follows with modeling transformations, clipping, viewing transformations and finally scan conversion for generating an image. Here, we will deal with previously rasterized bitmaps, i.e. sprites, and will model the translation transformations, rotation and clipping with differential operations. This way, the steps in the pipeline can be defined as layers of a neural network and the free parameters can be optimized with backpropagation.

For our neural network implementation, we assume a finite set of sprites (later we generalize it to infinite sprites) that will be part of the frames in the video. The image generation network selects a sprite, s, from a memorized sprite database $S_{i\in\{1,\ldots,K\}}$ using an addressing signal c:

$$\begin{aligned} s &= \sum_j c_j S_j, \quad \text{where} \\ \sum_j c_j &= 1. \end{aligned} \tag{1}$$

For interpretable results it would be optimal to do one-hot memory addressing where $c_j = 1$ for $S_j = S$ and $c_j = 0$ otherwise. Note that (1) is differentiable w.r.t to both c_j and S_j so we can learn the individual sprites from data. We can guarantee c_j adds up to 1 using the softmax nonlinearity. This approach was inspired by the recent deep learning literature on attention modules (Bahdanau et al., 2014) (Graves et al., 2014).

When the number of possible sprites is too large it is more efficient to do a compressed representation. Instead of using an address value c we use a content addressable memory where the image generator estimates a code z that is then decoded to the desired sprite with a (possibly nonlinear) function $d(z)$. If we interpret the addressing value z as a latent representation and the content addressable memory $d(z)$ as a decoder, we can use the recent advances in neural networks for generative models to setup our statistical model. We will revisit this later in this section.

The translation transformation can be modeled with a convolution with a Delta function or using spatial transformers. Note that the translation of an image $I(x, y)$ can be defined as

$$I(x - \tau_x, y - \tau_y) = I(x, y) \star \delta(x - \tau_x, y - \tau_y), \tag{2}$$

where $\star$ denotes the image convolution operation. Clipping is naturally handled in such a case. If the output images have finite dimensions and $\delta(x - \tau_x, y - \tau_y)$ is non-zero near its border, the translated image $I(x - \tau_x, y - \tau_y)$ will be clipped. Another way to implement the translation operation is using Spatial Transformer Networks (STN) (Jaderberg et al., 2015). STN can be defined in two steps: resampling and bilinear interpolation. Resampling is defined by moving the position of the pixels (x, y) in the original image to new positions $(\tilde{x}, \tilde{y})$ using a linear transform:

$$\begin{aligned} \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} &= A \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad \text{where} \\ A &= \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix}. \end{aligned} \tag{3}$$

We assume the coordinates in the original image are integers $0 \leq x < M$ and $0 \leq y < N$, where $M \times N$ is the size of the image I. Once the new coordinates are defined, we can calculate the values of the pixels in the new image $\tilde{I}$ using bilinear interpolation:

$$\begin{aligned} \tilde{I}(\tilde{x}, \tilde{y}) = w_{x_1,y_1} I(x_1, y_1) + w_{x_1,y_2} I(x_1, y_2) + \\ w_{x_2,y_1} I(x_2, y_1) + w_{x_2,y_2} I(x_2, y_2) \end{aligned} \tag{4}$$

where (x_1, x_2, y_1, y_2) are integers, $x_1 \leq \tilde{x} < x_2$, $y_1 \leq \tilde{y} < y_2$ and

$$\begin{aligned} w_{x_1,y_1} &= (\lfloor \tilde{x} \rfloor - \tilde{x})(\lfloor \tilde{y} \rfloor - \tilde{x}) \\ w_{x_1,y_2} &= (\lfloor \tilde{x} \rfloor - \tilde{x})(\lfloor \tilde{y} \rfloor + 1 - \tilde{y}) \\ w_{x_2,y_1} &= (\lfloor \tilde{x} \rfloor + 1 - \tilde{x})(\lfloor \tilde{y} \rfloor - \tilde{y}) \\ w_{x_2,y_2} &= (\lfloor \tilde{x} \rfloor - \tilde{x})(\lfloor \tilde{y} \rfloor + 1 - \tilde{y}) \end{aligned} \tag{5}$$

To avoid sampling from outside the image we clip the values $\lfloor \tilde{x} \rfloor$ and $\lfloor \tilde{x} \rfloor + 1$ between 0 and M and the values $\lfloor \tilde{y} \rfloor$ and $\lfloor \tilde{y} \rfloor + 1$ between 0 and N. We did not write the clipping operation in (5) for conciseness. Note that (4) is piecewise differentiable w.r.t I.

We can define translation through operations with

$$A = \begin{bmatrix} 1 & 0 & \tau_x \\ 0 & 1 & \tau_y \end{bmatrix}. \tag{6}$$

Also, we can rotate the image ρ radians counter clockwise with

$$A = \begin{bmatrix} \cos\rho & \sin\rho & 0 \\ -\sin\rho & cos\rho & 0 \end{bmatrix}. \tag{7}$$

Image rescaling is achieved by rescaling in the right square submatrix $A_{1:2,1:2}$. We illustrate in Fig. 1 how to get similar results using convolutions with a delta-function and spatial transformers.

We can now define a statistical model of 2D images that explicitly represents sprites and their positions in the scene. We can use the free energy of this statistical model to optimize a neural network. Let us start with the statistical model of a single static frame and later generalize it to video.

Let an image $I \sim p_\theta(I)$ be composed of a sprite $s \sim p_\theta(s)$ centered in the (x, y) coordinates in the larger image I. Denote the coordinates as a random variable $\delta_{xy} \sim p_\theta$, where θ represent the model parameters. $p_\theta(\delta_{xy})$ can be factored in two marginal categorical distributions $Cat(\delta_x)$ and $Cat(\delta_y)$ that model the probability of each coordinate of the sprite independently. For the finite sprite dataset, $p_\theta(s)$ is also a categorical distribution conditioned on the true sprites. For this finite case the generative model can be factored as

$$p_\theta(I, s, \delta) = p_\theta(s)p_\theta(\delta_{xy})p(I|s, \delta_{xy}), \tag{8}$$

assuming that "what", s, and "where", δ_{xy}, are statistically independent. Also, in such case the posterior

$$p_\theta(s, \delta|I) = p_\theta(s|I)p(\delta_{xy}|I) \tag{9}$$

is tractable. One could use for instance Expectation-Maximization or greedy approaches like Matching Pursuit to alternate between the search for the position and fitting the best matching shape. For the infinite number of sprites case, we assume that there is a hidden variable z from which the sprites are generated as $p(s, z) = p_\theta(z)p\theta(s|z)$. In such case our full posterior becomes

$$\begin{aligned} p_\theta(z, s, \delta|I) = p_\theta(z, s|I)p(\delta_{xy}|I) = \\ p_\theta(z|I)p_\theta(s|I, z)p(\delta_{xy}|I). \end{aligned} \tag{10}$$

We can simplify (10) assuming $p_\theta(z|s) = p_\theta(z|I)$ for simple images without ambiguity and no sprite occlusion. For a scalable inference in the case of unknown θ and z and intractable $p_\theta(z|s)$ we can use the auto-encoding variational Bayes (VAE) approach proposed by Kingma & Welling (2013). Using VAE we define an approximate recognition model $q_\phi(z|s)$. In such case, the log-likelihood of the i.i.d images I is $\log p_\theta(I_1, \ldots, I_T) = \sum_i^T \log p_\theta(I_i)$ and

$$\begin{aligned} \log p_\theta(I_i) = D_{KL}(q_\phi(z|s_i)||p_\theta(z|s_i)) + \\ D_{KL}(p_\theta(z|s_i)||p_\theta(z|I_i)) + \\ \mathcal{L}(\theta, \phi, \delta_{xy}, I_i). \end{aligned} \tag{11}$$

Again, assume that the approximation $p_\theta(z|s) = p_\theta(z|I)$ we have $D_{KL}(p_\theta(z|s_i)||p_\theta(z|I_i)) = 0$ and the free energy (or variational lower bound) term equal to

$$\begin{aligned} \mathcal{L}(\theta, \phi, \delta, I) = -D_{KL}(q_\phi(z|s_i)||p_\theta(z)) + \\ E_{q_\phi(z|s,\delta)p_\theta(\delta|I)}[\log p_\theta(I|z, \delta)], \end{aligned} \tag{12}$$

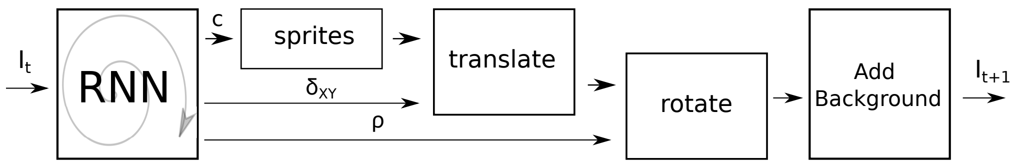

Figure 2: A schematic block diagram for a Perception Updating Network. This configuration uses both convolutions with delta functions for translation and spatial transformers for rotation. It also shows the optional background underlay.

where we dropped the subindices xy and i to avoid clutter. Here we would like to train our model by maximizing the lower bound (12) in the VAE way. We can do so using the reparametrization trick assuming $q_\phi(z|s)$ and the prior $p_\theta(z)$ to be Gaussian and sampling

$$z = m_\phi(I) + v_\phi(I) \cdot \xi, \tag{13}$$

where $\xi \sim \mathcal{N}(0, \sigma\mathbb{I})$, $\mathbb{I}$ is the identity matrix, the functions $m(I)$ and $v(I)$ are deep neural networks learned from data.

One can argue that given z and a good approximation to the posterior q_ϕ, estimating δ is still tractable. Nevertheless, we preemptively avoid Expectation-Maximization or other search approaches and use instead neural network layers l_x and l_y:

$$\delta_{xy} = \text{softmax}(l_x(I)) \otimes \text{softmax}(l_y(I)), \tag{14}$$

with $\otimes$ denoting the outer product of marginals. We also experiment using STNs. Such amortized inference is also faster in training and test time than EM and will also cover the case where I is itself a learned low dimensional or latent representation instead of an observable image. Bear this in mind while we use this approach even in simple experiments such as those with moving shapes in the Experiments section. This will help us to understand what can be learned from this model.

We extend the model above to videos, i.e. sequences of images $I(t) = \{I(0), I(1), \ldots\}$, assuming that the conditional log-likelihood $\log p_\theta(I_t|H_{I_t}) = logp_\theta(I_t|H_{\delta_t}, H_{z_t})$ follows (11), where H_{I_t} is the history of video frames prior to time point t. Also H_{δ_t} and H_{z_t} are the history of position coordinates and the history of latent variables of the sprites respectively. We should observe that one can make the assumption that the sprites don't change for a given video $I(t)$ and only estimate one sprite $s_{t=0}$ or hidden variable $z_{t=0}$. This assumption can be useful for long term predictions, but requires that the main object moving in the scene doesn't change.

In the next section, we propose a neural network architecture for maximizing our approximate variational lower bound 2D videos.

3 Perception Updating Networks

This section proposes a family of neural architectures for optimizing the lower bound (12). A schematic diagram is represented in Fig. (2). The core of our method is a Recurrent Neural Network (RNN) augmented with task specific modules, namely a sprite addressable memory and modeling transformations layers. RNNs augmented with task specific units were popularized by Graves et al. (2014) in the context of learning simple differentiable algorithms and served as inspiration for us as well. Here since we explicitly model the perceived sprites as s or z and update it and its location and/or rotation though time we decided to call our method simply Perception Updating Networks.

An input frame at time t, I_t, is fed to the RNN that emits 2 signals: a memory address that selects a relevant sprite and transformation parameters. If we are doing the translation transformation using convolutions and delta functions this output is equal to (14). If using STN, the translation operation returns the matrix A used in (3). Note that we could use both, letting convolutions with δ to the translation is constraining A as in (7) to do rotation transformations only. We describe the general case where both δ_{xy} and STNs are used in Algorithm 1.

Beyond deciding between STNs vs δ_{xy}, a few other free parameters of our method are the type of RNN (e.g. vanilla RNN, LSTM, GRU, ConvRNN, etc), the number of neurons in the hidden state of the RNN and neural network architectures that infer the correct sprite and modeling transformation parameters. Our hyperparameter choices are investigated separately in each experiment in the next Section.

```
Data: input videos $I_t, t \in \{0, 1, 2, \ldots\}$, initial RNN state $h_0$, neural network layers $m_\phi, v_\phi, d, l, f$
Result: video predictions $I_t, t \in \{1, 2, 3, \ldots\}$
for $t \in \{0, 1, \ldots\}$ do
    $h_t \leftarrow RNN(I_t, h_{t-1})$
    $\delta_{xy} = \text{softmax}(l_x(h_t)) \otimes \text{softmax}(l_y(h_t))$
    $\rho = f(h_t)$
    $A = \begin{bmatrix} \cos\rho & \sin\rho & 0 \\ -\sin\rho & \cos\rho & 0 \end{bmatrix}$
    $\xi \sim p_\theta(z)$
    $z_t = m_\phi(h_t) + v_\phi(h_t) \cdot \xi$
    $s_t = d(z_t)$
    $a_t = STN(s_t, A)$
    $\tilde{I}_{t+1} = a_t \star \delta_{xy}$
    $I_{t+1} = \mu \tilde{I}_{t+1} + (1 - \mu)B$
end
```

Algorithm 1: Perception Updating Networks. STN denotes spatial transformer operator (3)-(4) and $\star$ denotes convolution. We experimented with several variations of this algorithm, mainly changing if and how the "where" modules δ_{xy} and STN are used. Also changing how the sprite s_t is calculated and not using a background B when not necessary.

In the next section we present experiments with the proposed architecture on synthetic datasets.

4 Experiments

In this section we experiment with several implementations of the proposed Perception Updating Networks. We start with a simple synthetic dataset made of videos where one of 3 moving shapes moves with constant speed bouncing in the edges of an image. This illustrates the working of the finite memory and the addressing scheme in (1). Afterwards we show results on the moving MNIST dataset (Srivastava et al., 2015) commonly used in the literature of generative neural network models of videos.

4.1 Bouncing Shapes

In this first experiment we generate videos of one of three shapes moving on a non-zero background. The shapes are a square, triangle and cross. The image size is 20×20 pixels and the shapes are 8×8 pixels. The pixel values are between 0 and 1. The shapes are picked with equal probability and they move at constant speed of 1 pixel per frame. The shapes start from random initial positions and move following a random direction vector.

We tested two implementations of the proposed architecture: one using only convolutions, referred to as convolutional PUN in the figures, and another using using spatial transformers, called spatial transformer PUN. For the parameters of the convolutional PUN the RNN used was a Long Short Term Memory (LSTM) with 100 cells. The RNN in the Spatial Transformer PUN had 256 cells. We used the finite addressable memory described in (1). The background is also learned from data with backpropagation. This background served to make the task more difficult and force the network to avoid just exploiting any non-zero value. After the convolutional composition $I_t = s_t \star \delta_{xy}$, we added the background to form a new image using $\tilde{I}_t = \mu \cdot I_t + (1 - \mu)B$, where μ is a differentiable mask that accounts for the "transparency" of the image I_t. B is the learned 20×20 pixels background image. For complex shapes this mask shape could be calculated as another module in the network, similarly to the approach in Vondrick et al. (2016). Here we used $\tilde{I}_t$ itself as the mask.

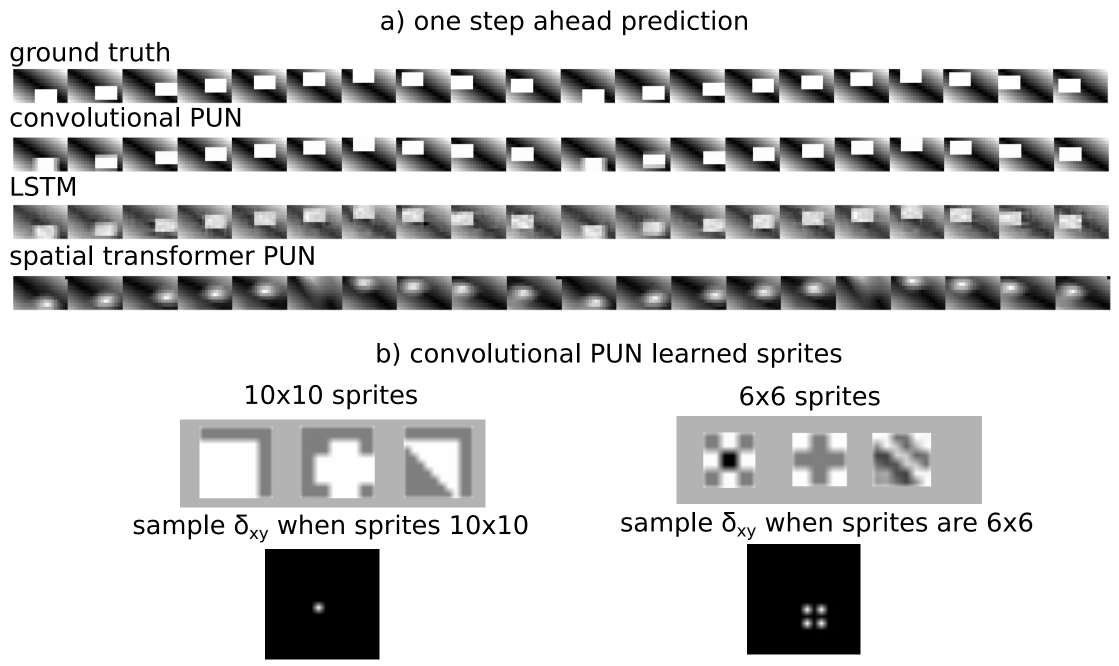
a) one step ahead prediction
ground truth
convolutional PUN
LSTM
spatial transformer PUN
b) convolutional PUN learned sprites
10x10 sprites
6x6 sprites
sample δ_{xy} when sprites 10x10
sample δ_{xy} when sprites are 6x6

Figure 3: Results on the Bouncing Shapes dataset. Three 8x8 sprites (a square, a cross and a triangle) were used to generate videos. The shapes move in a 20x20 pixels canvas with a Toeplitz background and bounce on the corners. a) We show one step ahead predictions with the compared methods. b) We also show the learned sprites for the convolutional implementation of the proposed Perception Updating Networks when we over- and under-estimate the size of the desired sprites.

In the following experiments, the training videos were 10 frames long. At test time the network is fed the first 10 frames of a video and asked to predict the next 10. Results for the compared methods are shown in Fig. 4. For the baseline method, we did a hyperparameter search on conventional LSTMs with a single linear output layer until we found one that had comparable results at test time. That network had 256 hidden cells. Also, note that although the scale of the mean square error is the same, the results from our proposed architecture look smoother than those learned by the LSTM as shown in Fig. 3.

Given such a simple experiment, it is elucidating to visualize values learned by each piece of the network. As expected the sprite memory learned the 3 investigated shapes in transposed order since they are reverted by the convolution operation to compose the frame, similar to 3.b-left. We also experimented with choosing the size of the learned sprites s_t smaller and larger than the true shapes. We observed that for larger shapes such as 10×10 the sprites converge to the correct shapes but just using part of the pixels. For smaller shapes such as 6×6 pixels, instead of learning a part of the correct shape, the convolutional Perception Updating Network learned to compensate for the lack of enough pixels with more than one non-zero value in the location operation δ_{xy} (see Fig. 3). This allow us to suggest to the interested practitioner that in order to get interpretable results it is better to use sprites larger than the expected true size than smaller.

For the spatial transformer PUN the image is calculated as (see Algorithm 1 for context):

$$\begin{aligned} A &= f(h_t), \\ I_{t+1} &= STN(s_t, A). \end{aligned} \tag{15}$$

We noticed that the spatial transformer PUN was not able to learn the training videos using an equivalent architecture to the convolutional PUN one. We had to use multiple layers to define the function $f(h_t)$. In other words, in the convolution based method δ_{xy} can be estimated by a single affine transformation of the state h_t, but A cannot. We also had to use smaller learning rates to

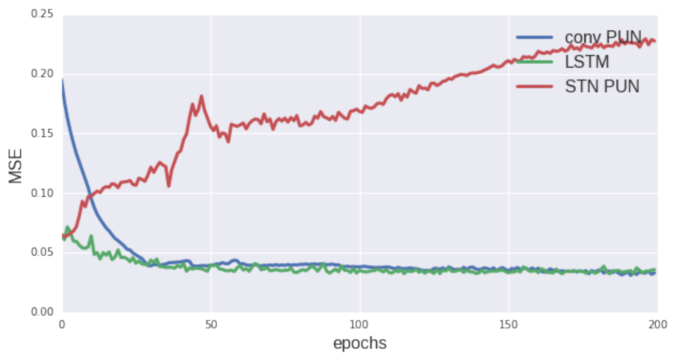

Figure 4: Learning curves in the test task of two implementations of the proposed architecture (conv PUN and STN PUN) and an equivalent LSTM baseline. Note that the spatial transformer based PUN was not able to generalize to the test set, i.e. they did not work well for generating videos when getting its own previous outputs as next step inputs.

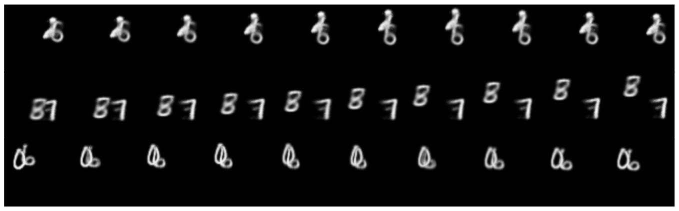

Figure 5: Sample rollouts of a 2 layer LSTM convolutional Perception Updating Network with .

guarantee convergence: 0.0001 for STN while the δ_{xy}-based model worked with a value 10 times larger.

It is interesting to observe that under this framework the "what" and "where" are both elements of commutative operation and can only be distinguished if we impose architectural constraints. As a note on rotation, we ran experiments where the sprite are rotated by a random angle before being placed in the image. This new type of videos cannot be learned using only convolutional based Perception Updating Networks unless we increase the number of sprites proportionally to the number of possible angles. Spatial transformer based Perception Updating Networks can handle this new type of video naturally. Nevertheless, if the number of rotation angles is finite or can be discretized we found that we could learn to generate the videos faster if we combined the convolutional approach with a mechanism to select the appropriate angle from a set of possibilities. Results on this experiment are not shown in this paper due to space constraints but they can be reproduced with the companion code.

4.2 Moving MNIST

The Moving MNIST benchmark uses videos generated by moving 28×28 pixel images of hand written digits in a 64×64 pixels canvas. Just like in the Bouncing Shapes dataset, the digits move with different different speeds in different directions and can bounce in the walls. Unlike the Bouncing Shapes dataset, there are 60000 different sprites for training and 10000 for test, making it impractical to use a discrete memory module. Instead, we use the memory representation denoted by (13) followed by $s_t = d(z_t)$ as written in Algorithm 1.

We trained a convolutional Perception Updating Network using 2 layer LSTMs each one with 1024 cells for 200 epochs, with 10000 gradient updates per epoch. The latent variable z had 100 dimensions and the decoder $d(\cdot)$ was a single hidden layer MLP with 1000 hidden neurons and softplus activation function. The output layer of this MLP has 784 neurons, which is the size of an MNIST image, and sigmoid activation function. In the test set we obtained a negative log-likelihood of 239 nats with the proposed architecture, while a 2 layer LSTM baseline had 250 nats. Note that the our method was optimized to minimize the lower bound (12), not only the negative likelihood. These results are not as good as those obtained by the Video Pixel Networks (Kalchbrenner et al., 2016) that obtained 87 nats on the test set. Nevertheless, both approaches are not mutually exclusive and instead of a fully connected decoder we could use a similar PixelCNN decoder to generate sprites with higher likelihood. In this first paper we decided instead to focus in defining the statistical framework and interpretable "what" and "where" decoupling.

When running the proposed method in rollout mode, feeding the outputs back as next time step inputs, we were able to generate high likelihood frames for more time steps than with a baseline LSTM. Also, since the sprite to be generated and its position in the frame are decoupled, in rollout mode we can fix the sprite and only use the δ_{xy} coming from the network. This way we can generate realistic looking frames for even longer. On the other hand, after a few frames we observed the digits stopped moving or moved in wrong directions for some videos. This means that the LSTM RNN was not able to maintain its internal dynamics for too long, thus, there is still room for improvement in the proposed architecture.

In Fig. 5 we show sample rollout videos. The network was fed with 10 frames and asked to generate 10 more getting its own outputs back as inputs.

This experiment also suggests several improvements in the proposed architecture. For example, we assumed that the internal RNN has to calculate a sprite at every time step, which is inefficient when the sprites don't change in the video. We should improve the architecture with an extra memory unity that snapshots the sprites and avoid the burden of recalculating the sprites at every step. We believe this would be a possible way to free representation power that the internal RNN could use to model the video movement for more time steps.

5 Conclusions

This paper introduced a statistical framework for modeling video of 2D scenes inspired by graphics pipelines and variational auto-encoding Bayes. From this statistical framework we derived a variational lower bound that decouples sprites and their movement in a video. To optimize this lower bound, we suggested a family of architectures called Perception Updating Networks that can take advantage of this decoupled representation by independently memorizing sprites or their percepts, and updating its location in a scene. We showed that this architecture could generate videos that are interpretable and are better suited than baseline RNNs for long video generation.

Acknowledgments

We thank Ryan Burt for several suggestions to the first draft. This work was partially funded by the University of Florida Graduate Student Fellowship and ONR N00014-14-1-0542.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

SM Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, Koray Kavukcuoglu, and Geoffrey E Hinton. Attend, infer, repeat: Fast scene understanding with generative models. *arXiv preprint arXiv:1603.08575*, 2016.

Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. *arXiv preprint arXiv:1605.07157*, 2016.

Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. *arXiv preprint arXiv:1609.03677*, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Jarmo Hurri and Aapo Hyvärinen. Simple-cell-like receptive fields maximize temporal coherence in natural video. *Neural Computation*, 15(3):663–691, 2003.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pp. 2017–2025, 2015.

Nal Kalchbrenner, Aaron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. *arXiv preprint arXiv:1610.00527*, 2016.

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

Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pp. 2539–2547, 2015.

William Lotter, Gabriel Kreiman, and David Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

Reza Mahjourian, Martin Wicke, and Anelia Angelova. Geometry-based next frame prediction from monocular video. *arXiv preprint arXiv:1609.06377*, 2016.

Bruno A Olshausen et al. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

Simon Osindero and Geoffrey E Hinton. Modeling image patches with a directed hierarchy of markov random fields. In *Advances in neural information processing systems*, pp. 1121–1128, 2008.

Scott Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw. *arXiv preprint arXiv:1610.02454*, 2016.

Danilo Jimenez Rezende, SM Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. *arXiv preprint arXiv:1607.00662*, 2016.

Peter Shirley, Michael Ashikhmin, and Steve Marschner. *Fundamentals of computer graphics*. CRC Press, 2015.

Eero P Simoncelli and Bruno A Olshausen. Natural image statistics and neural representation. *Annual review of neuroscience*, 24(1):1193–1216, 2001.

Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. *CoRR, abs/1502.04681*, 2, 2015.

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. *arXiv preprint arXiv:1609.02612*, 2016.