Report for Generative Temporal Models with Spatial Memory for Partially Observed Environments

Xianyu Chen

Guangzhou, China

Tel: +86-135-8059-3702

Email: xianyuchen1992@outlook.com

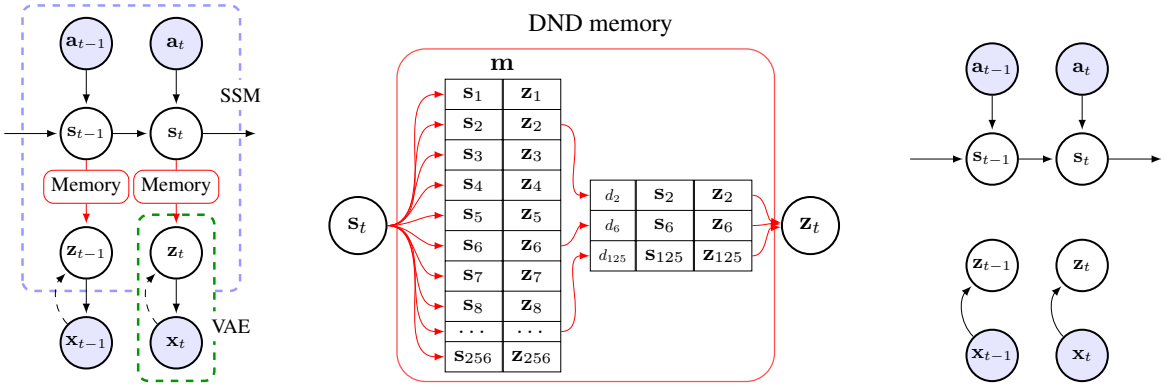
July 27, 2018

Contents

1 Definition

In this report, I would illustrate some definitions of variables from paper [1] to remind the reader about them.

I implement the image navigation experiment, and the relevant code is pushed to github https://github.com/chenxy99/Generative-Temporal-Models-with-Spatial-Memory.



(a) Generative model of the GTM-SM. Red arrows represent dependencies on the DND memory. Dashed arrows represent the VAE inference network.

(b) The dependence of $\mathbf{z}_t$ on $\mathbf{s}_t$ and the DND memory $\mathbf{m}$ in the VAE prior $p(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m})$. This corresponds to the red rectangle in Figure 1a. Here, the memory contains the first $\tau = 256$ time steps of the video, and we retrieve the $K = 3$ closest neighbours to $\mathbf{s}_t$ (d is the distance).

(c) Inference network for the GTM-SM.

Figure 1: Generative Temporal Model with Spatial Memory.

As shown in figure 1(a), in the Generative Temporal Model with Spatial Memory (GTM-SM), there are two sets of latent variables that disentangle visual and dynamics information. In the time step t, $\mathbf{z}_t$ is the output of the encoder whose input is the cropping of image $\mathbf{x}_t$, $3 \times 8 \times 8$, while the $\mathbf{s}_t$ is the latent variable that demonstrates the representation of an agent in the environment. Action $\mathbf{a}_t$ control the moving of the cropping window, so that the window can move right/left/up/down or stop.

In our simulation, there exist to be three different neural networks - encoder $q_\phi(\mathbf{z}_t|\mathbf{x}_t)$, decoder $p_\theta(\mathbf{x}_t|\mathbf{z}_t)$ and the non-linear transitions

$$\mathbf{s}_t = \mathbf{s}_{t-1} + \mathbf{M}\mathbf{a}_t \cdot \sigma_d(\mathbf{s}_{t-1} + \mathbf{M}\mathbf{a}_t) + \varepsilon_t. \tag{1}$$

During the observations in the memorization phase $t = 1 : \tau$, based on the encoder $q_\phi(\mathbf{z}_t|\mathbf{x}_t)$ and non-linear transitions, we write $\mathbf{m} = \{\mathbf{s}_t, \mathbf{z}_t\}$ into the dictionary. Then, in the prediction phase of training $t = \tau + 1 : T$, we minimize the objective function

$$\mathcal{F}(\theta, \phi) = -\sum_{t=\tau+1}^{T} \mathbb{E}_{q_\phi(\mathbf{z}_t|\mathbf{x}_t)}[\log p_\theta(\mathbf{x}_t|\mathbf{z}_t)] + \mathbb{E}_{p^*_\theta(\mathbf{s}_t)}[KL[q_\phi(\mathbf{z}_t|\mathbf{x}_t)||p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m})]], \tag{2}$$

where $p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m})$ is the VAE prior distribution obtained by creating the mixture Gaussian distribution from the differential neural dictionary, whose weights are inversely proportional to the square distances $d^{(k)}$ between $\mathbf{s}_t$ and the retrieved elements $\mathbf{s}^{(k)}$ which is the K nearest neighborhoods of $\mathbf{s}_t$ from the dictionary.

$$p_\theta(\mathbf{z}_t|\mathbf{s}_t, \mathbf{m}) = \sum_{k=1}^{K} \omega_k \mathcal{N}(\mathbf{z}_t|\mathbf{z}^{(k)}); \quad \omega_k \propto \frac{1}{d^{(k)^2} + \delta}, \tag{3}$$

where $\delta = 10^{-4}$ is added for stability.

Moreover, during the testing phase, without the information of $\mathbf{x}_t$ the cropping windows can acquire, the cost function will be modified as

$$LOSS = -\sum_{t=\tau+1}^{T} \mathbb{E}_{p_\theta(\mathbf{z}_t|\mathbf{s}_t,\mathbf{m})}[\log p_\theta(\mathbf{x}_t|\mathbf{z}_t)] \tag{4}$$

2 Implementation

At the beginning of this section, I will illustrate the architecture of neural networks in my simulation.

The encoder $q_\phi(\mathbf{z}_t|\mathbf{x}_t)$ is used to produce the multi-Gaussian mean value and variance, which is make up of two layers convolution neural networks followed by a full connection layer.

```
self.enc_zt = nn.Sequential(
    Preprocess_img(),
    nn.Conv2d(3, 64, kernel_size=2, stride=2),
    nn.LeakyReLU(0.01),
    nn.Conv2d(64, 16, kernel_size=2, stride=2),
    nn.LeakyReLU(0.01),
    Flatten())
self.enc_zt_mean = nn.Sequential( nn.Linear(64, z_dim))
self.enc_zt_std = nn.Sequential( nn.Linear(64, z_dim), Exponent())
```

The decoder $p_\theta(\mathbf{x}_t|\mathbf{z}_t)$ is used to produce the prediction images, which is make up of a full connection layer followed by two layers deconvolution neural networks.

```
self.dec = nn.Sequential(  nn.Linear(z_dim, 64),
    nn.ReLU(),
    Unflatten(-1, 16, 2, 2),
    nn.ConvTranspose2d(in_channels=16, out_channels=64, kernel_size=2, stride=2),
    nn.ReLU(),
    nn.ConvTranspose2d(in_channels=64, out_channels=3, kernel_size=2, stride=2),
    nn.Tanh(),
    Deprocess_img())
```

Finally, the non-linearity transition function whose variable is $\mathbf{s}_t$ can be modeled as

```
self.enc_st_matrix = nn.Sequential( nn.Linear(a_dim, s_dim, bias=False))
self.enc_st_sigmoid = nn.Sequential(
    nn.Linear(s_dim, 10),
    nn.ReLU(),
    nn.Linear(10, 5),
    nn.ReLU(),
    nn.Linear(5, 1),
    nn.Sigmoid())
```

It is tricky to implement GTM-SM, so it would be ineffective to directly train the whole model. In the rest of this section, I will show how I fulfill all the training processes step by step.

1 . At first, I start with very simple experiments and make them progressively harder. I used linear transitions, which means that there is no "wall" in my data generation. So the sigmoid neural network in (1) can be eliminated. The objective function in this phase is (2).

2 . From the results achieved from step 1, we find that in the inferred states, the up/down vectors and left/right vectors are not mutually perpendicular. Also, the amplitude of up/down vectors is not equal to that of the left/right vectors. So, to make up/down vectors and left/right vectors mutually perpendicular and have the same norm, I add a

penalty term containing the orthogonal item and norm constraint, which is

$$PENALTY = \lambda_1[(\mathbf{M}_{:,0}^T\mathbf{M}_{:,2})^2 + (\mathbf{M}_{:,1}^T\mathbf{M}_{:,3})^2] + \lambda_2\sum_{i=0}^{3}(\|\mathbf{M}_{:,i}\|_2 - \frac{D}{8})^2, \tag{5}$$

where λ_1 and λ_2 are the hyper-parameters, $\mathbf{M}$ is the transition matrix, D is set to 1.5. The objective function in this phase is the sum of (2) and (5).

3 . Furthermore, I will add the sigmoid neural network. In prior, we know that this sigmoid neural network acts like a wall which means if the agent is on the right side of the picture, the action - moving right will make it hold still. So that I will generate a 17×17 grids, the center of which is the position that the agent can reach, the size of which is 9×9. I set the label for the center of the grids 1, while the rest of them will be 0. It is a simple binary classification problem and I add this cost function as the regularization term of (2). The training set of this regularization term is $(\mathbf{x}_{j,k}, y_{j,k}),\ j,k = -8,-7,\cdots,7,8$, where $\mathbf{x}_{j,k} = j\mathbf{M}_{:,0} + k\mathbf{M}_{:,2},\ j,k = -8,-7,\cdots,7,8$, and $y_{j,k} = 1,\ j,k = -4,-3,\cdots,3,4$, while other $y_{j,k}$ is equal to 0. So the regularization term is

$$REG = -\lambda_3\sum_{j=-8}^{8}\sum_{k=-8}^{8}(y_{j,k}\log(\sigma_d(\mathbf{x}_{j,k})) + (1-y_{j,k})\log(1-\sigma_d(\mathbf{x}_{j,k}))), \tag{6}$$

where λ_3 is the hyper-parameters. The objective function in this phase is the sum of (2) , (5) and (6). I will train it under the scenario of non-linearity transition function.

[Comments]

1 . During the training, if I simultaneously optimize all of these three neural networks under the scenario of non-linearity transition function, failure will occur. So I have to train this model with the linearity transition function and the elimination of the sigmoid neural network. And then I achieve the satisfactory result under the aforementioned conditions.

2 . After obtaining the parameters from the first phase, I try to train all the neural network under the scenario of non-linearity transition function and the use of sigmoid neural network. But frustratingly, I found that the performance degrades after a several updates. So I try to add the information of the inferred states as a regularization term showed in (5) and (6). As a result, after some epochs, the validation error will decrease again.

Last but no least, Fig 2 shows the corresponding result obtained form the aforementioned steps.

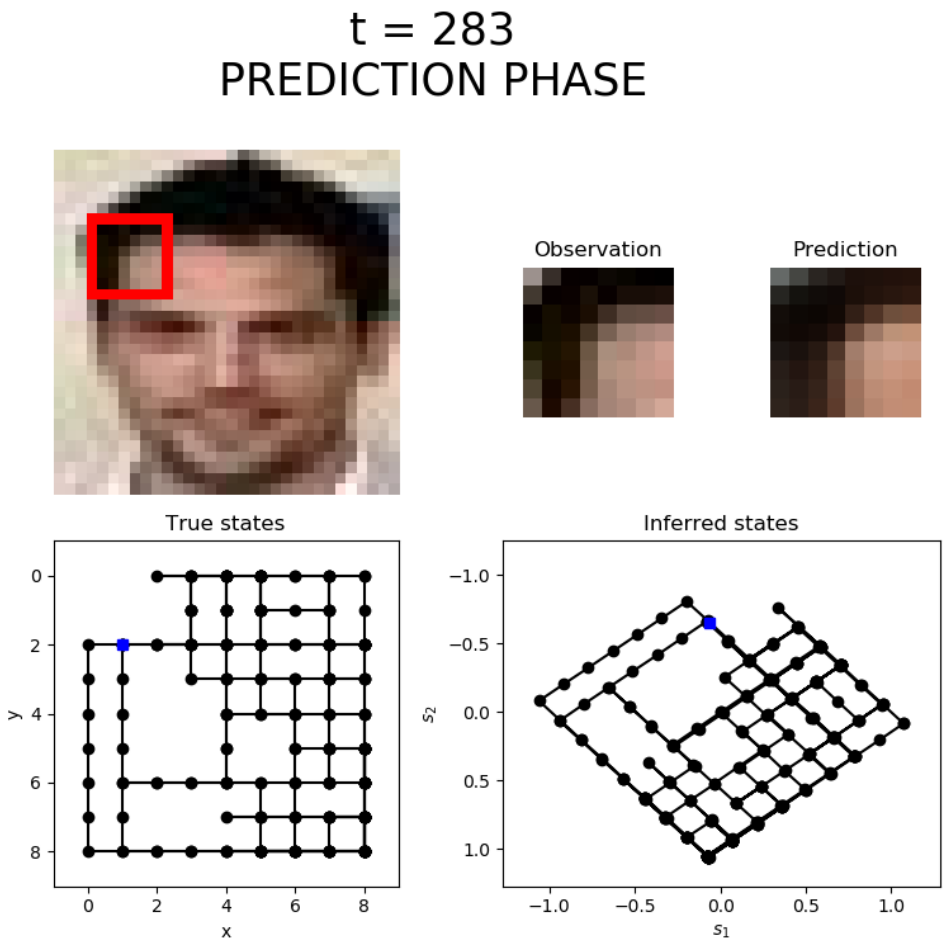


Figure 2: Image Navigation Experiment Result.

3 Conclusions

In this report, we implement the image navigation experiment and achieve the similar result as the origin paper [1] showed. During the training, I meet a lot of problems, one of which is that if I directly train the model, it cannot work as expected. I have to dismantle them into some small and easily solvable problems as I show in the previous section. Furthermore, I find that I need to add the regularization term to ensure the performance of the validation set.

References

[1] M. Fraccaro, D. Jimenez Rezende, Y. Zwols, A. Pritzel, S. M. A. Eslami, and F. Viola, "Generative temporal models with spatial memory for partially observed environments," *ArXiv e-prints*, Apr. 2018.