Gradient Origin Predictive Coding

by

Mozes Jacobs

Supervised by Rajesh Rao

A senior thesis submitted in partial fulfillment of

the requirements for the degree of

Bachelor of Science
With Departmental Honors

Computer Science & Engineering

University of Washington

June 2022

Presentation of work given on May 31, 2022

Thesis and presentation approved by Rajesh P.N. Rao

Date June 10 2022

Abstract

Generative models are statistical models that describe how a dataset is generated. Traditional variational autoencoder (VAE)-based models view this data as being generated by some hidden variables and attempt to model the statistical properties of these variables. VAEs perform variational inference (VI) - inference of distributional parameters through minimizing the Kullback-Leibler divergence (KLD) between an approximate and target distribution - through the use of an image encoder. They are trained by minimizing the "variational loss", which upper bounds the aforementioned KLD. Gradient Origin Networks [1] (GONs), on the other hand, replace the encoder and perform VI using a one-step gradient update with respect to the origin using the variational loss. Building off of GONs, this thesis proposes an encoder-less generative model based on the predictive coding framework that learns latent representations of image sequences using only the gradient of the variational loss. Specifically, at each time step in an image sequence, the model predicts a latent encoding of the current frame. After comparing the prediction to the real frame, the model uses a one-step gradient update to generate a posterior prediction of the frame. On a variant of the Moving MNIST [2] dataset called Smooth Moving MNIST, the model is able to produce high-quality reconstructions of frames. In addition, it is able to successfully perform next-frame prediction using the prior, as well as predict (to varying degrees of accuracy) multiple time steps into the future. The proposed model is compared to an encoder-based version to assess the relative strength of the framework. Furthermore, an analysis of the limitations of the inference framework is provided.

Acknowledgements

I would like to thank Dr. Rajesh Rao for letting me join his lab and providing me with numerous opportunities to explore my interests in neuroscience and deep learning. I would like to give thanks to Preston Jiang for mentoring me for over two and a half years and helping me every step of the way. Lastly, I would like to thank everyone I work with in the Neural Systems Lab on a daily basis for the fun meetings and long work sessions we have had.

Contents

Chapter 1

Introduction

Predictive coding theories of cortical function [3, 4] view the brain as a predictive entity that learns a hierarchical generative model of the world. Each level ascending the cortical hierarchy learns neural representations at successively greater spatiotemporal scales and attempts to predict the neural activity of lower levels. At the very lowest level, the cortex predicts the incoming sensory stimuli such as visual input. Once sensory input is received, it can be compared to the predicted input to compute a “prediction error”. This prediction error can be used to update its predictions. The prediction and error correction happens at every level of the hierarchy. This theory can be seen as a neural implementation of kalman filtering [3, 5, 6].

Attempts have been made to incorporate predictive coding into deep learning models. PredNet [7] is a predictive-coding based model for video prediction. However, instead of successively predicting activations, its neurons predict prediction errors, meaning its generative model is unclear [8]. Dynamic Predictive Coding Networks (DPCNs) [8] are hierarchical predictive coding models in which higher level activations predict the transition dynamics of lower level activations using a hypernetwork. MAP inference of the latent variables is performed through a gradient descent loop that minimizes a joint prediction error loss function. However, the model is not probabilistic.

Building off of this theory of predictive processing, we aimed to answer the following question: can we build an efficient probabilistic generative model for video sequences using prediction errors?

To help answer this question, we will first look at some more existing methods in the realm of generative modeling and encoder-less learning and their drawbacks.

Generative adversarial networks [9] (GANs) are a type of network that can produce high fidelity images and learn without an encoder through the use of an adversarial game between a generator network and a discriminator network. However, they are quite difficult to train [10].

Disentangled sequential autoencoder [11] is a variational autoencoder (VAE) model that can completely disentangle object identity and motion in video sequences. Improved Conditional VRNN [12] is a hierarchical VAE model for high-quality video prediction. Clockwork VAE [13] is another hierarchical VAE model in which higher levels operate over longer time-scales, enabling extremely long-term dynamics to be modeled. However, each of these models uses an encoder.

DeepSDF [14] is an encoder-less model for learning a latent space for shapes. However, it performs Maximum-a-Posterior (MAP) inference with gradient descent, which is generally slow compared to an encoder.

Recent work on Gradient Origin Networks (GONs) [1] demonstrated that variational inference can be performed in one step using the gradient of the variational loss instead of an encoder. However, GONs were only designed for and evaluated on image data. If it could be shown that GONs perform well on video sequences, then it would be one example of an efficient probabilistic generative model built off of prediction errors.

In this work, I propose an extension of the variational GON framework to the video domain called "Gradient Origin Predictive Coding" (GOPC). In particular, the contributions of this thesis are as follows:

1. A working implementation of a single level GOPC model (GOPC-1) and demonstrations of its inference and generative capabilities

2. A comparison of GOPC-1 to an encoder-based version of the model (VAE-1)

3. An implementation of a 2-level, hierarchical GOPC model (GOPC-2)

4. An analysis of the limitations of the GON inference framework and some potential workarounds

Chapter 2

Background

2.1 Intro

This chapter will cover necessary background information for this project.

2.2 Variational Autoencoders

Suppose we have some dataset $\boldsymbol{X} = \{\boldsymbol{x}_i^n\}$, where each example $\boldsymbol{x}_i$ generated from some latent vector $\boldsymbol{z}_i$. For ease of notation, let $\boldsymbol{x}$ be some $\boldsymbol{x}_i$ and $\boldsymbol{z}$ be the corresponding $\boldsymbol{z}_i$. Inference of z aims to compute the posterior over $\boldsymbol{z}$:

$$p_\theta(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})}{p_\theta(\boldsymbol{x})}$$

However, this would require calculating $p_\theta(\boldsymbol{x})$:

$$p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{x}|\boldsymbol{z})p_\theta(\boldsymbol{z})dz$$

which is computationally intractable, since it would require marginalizing over all possible values of $\boldsymbol{z}$. Although it is possible to approximate the above integral by sampling many values of $\boldsymbol{z}$, this is an impractically slow solution, since it would require a huge number of calculations for every single data example.

Instead, VAEs [15] construct an approximate posterior distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and minimize the Kullback-

Leibler divergence (KLD) between this approximate posterior and the true posterior:

$$KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|\boldsymbol{x})) = E[log(q_\phi(\boldsymbol{z}|\boldsymbol{x}))] - E[log(p_\theta(\boldsymbol{z}|\boldsymbol{x}))]$$

Intuitively, this minimization is bringing the approximate posterior, $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, as close as possible to the true posterior, $p_\theta(\boldsymbol{z}|\boldsymbol{x})$. Furthermore, minimizing the KLD between the approximate and true posterior is equivalent (up to a constant) to maximizing the evidence lower bound (ELBO):

$$ELBO = E[log(p_\theta(\boldsymbol{x}|\boldsymbol{z}))] - KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}))$$

Maximizing $E[log(p_\theta(\boldsymbol{x}|\boldsymbol{z}))]$ maximizes the likelihood of the data, while minimizing $KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}))$ regularizes the latent space to be closes to the prior $p_\theta(\boldsymbol{z})$, allowing VAEs to generate new data samples from the prior.

In practice, an encoder performs inference of $q_\phi(\boldsymbol{z}|\boldsymbol{x})$. The "reparameterization trick" allows VAEs to sample from $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ using $\boldsymbol{z} = \mu(\boldsymbol{z}) + \sigma(\boldsymbol{z}) \bigodot \boldsymbol{\epsilon}$ while still training the network end-to-end with backpropagation. A decoder is used to reconstruct $\boldsymbol{x}$ using $\boldsymbol{z}$, parameterizing $p_\theta(\boldsymbol{x}|\boldsymbol{z})$.

2.3 Gradient Origin Networks

GONs [1] are a type of generative model like VAEs, but do not require encoders. To understand how they do this, suppose, as VAEs do, that we want to model $p_\theta(\boldsymbol{z}|\boldsymbol{x})$, where θ are the generative parameters. Let ϕ denote inference parameters. Furthermore, suppose we have some noisy observation $\boldsymbol{z}_0$ of $\boldsymbol{z}$ such that $\boldsymbol{z_0} = \boldsymbol{z} + \mathcal{N}(\boldsymbol{0}, \boldsymbol{I_d})$. Then, we can calculate the least squares estimate $\hat{\boldsymbol{z}}_{\boldsymbol{x}}$ of $\boldsymbol{z}$:

$$\begin{aligned}
\hat{\boldsymbol{z}}_{\boldsymbol{x}}(\boldsymbol{z_0}) &= \int \boldsymbol{z} p_\theta(\boldsymbol{z}|\boldsymbol{z}_0, \boldsymbol{x}) d\boldsymbol{z} && \text{Bayes Least Squares} \\
&= \boldsymbol{z_0} + \nabla_{\boldsymbol{z_0}} log p_\theta(\boldsymbol{z_0}|\boldsymbol{x}) && \text{Proof in [1]} \\
&= \boldsymbol{z_0} + \nabla_{\boldsymbol{z_0}} (log p_\theta(\boldsymbol{x}|\boldsymbol{z_0}) + log p(\boldsymbol{z_0}) - log p_\theta(\boldsymbol{x})) && \text{Bayes Rule} \\
&= \boldsymbol{z_0} + \nabla_{\boldsymbol{z_0}} (log p(\boldsymbol{x}|\boldsymbol{z_0}) + log p(\boldsymbol{z_0})) && log p_\theta(\boldsymbol{x}) \text{ constant}
\end{aligned}$$

Essentially, $\hat{\boldsymbol{z}}_{\boldsymbol{x}}$ is a function of the noisy observation $\boldsymbol{z_0}$ and a gradient with respect to $\boldsymbol{z_0}$ that maximizes the likelihood of the $\boldsymbol{x}$ and regularizes to the prior over $\boldsymbol{z_0}$.

If we assume a standard normal prior for $p_\theta(\boldsymbol{z})$, then we have:

$$\hat{\boldsymbol{z}}_{\boldsymbol{x}}(\boldsymbol{z_0}) = \boldsymbol{z_0} + \nabla_{\boldsymbol{z_0}} (log p_\theta(\boldsymbol{x}|\boldsymbol{z_0}) + log \mathcal{N}(\boldsymbol{z_0}; \boldsymbol{0}, 2\boldsymbol{I_d}))$$

If some neural network parameterizes $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, as well as the calculation of μ_ϕ and σ_ϕ, the reparamaterization trick yields us the following inference equation:

$$\hat{\boldsymbol{z}}_{\boldsymbol{x}}(\boldsymbol{z_0}) = \boldsymbol{z_0} + \nabla_{\boldsymbol{z_0}} (log p(\boldsymbol{x}|\boldsymbol{z_0}) - KL(\mathcal{N}(\boldsymbol{\mu}_\phi(\boldsymbol{z_0}), \boldsymbol{\sigma}_\phi(\boldsymbol{z_0}))||\mathcal{N}(\boldsymbol{0}, \boldsymbol{I_d})))$$

Once we have $\hat{\boldsymbol{z}}_{\boldsymbol{x}}$, we can

- re-estimate the posterior distribution via $\mathcal{N}(\boldsymbol{\mu}_\phi(\hat{\boldsymbol{z}}_{\boldsymbol{x}}), \boldsymbol{\sigma}_\phi(\hat{\boldsymbol{z}}_{\boldsymbol{x}}))$

- and reconstruct the input through $p_\theta(\boldsymbol{x}|\hat{\boldsymbol{z}}_{\boldsymbol{x}})$

To train the weights of the network, we maximize:

$$p_\theta(\boldsymbol{x}|\hat{\boldsymbol{z}}_x) - KLD(\mathcal{N}(\boldsymbol{\mu}(\hat{\boldsymbol{z}}_{\boldsymbol{x}}), \boldsymbol{\sigma}(\hat{\boldsymbol{z}}_{\boldsymbol{x}})^2)||\mathcal{N}(\boldsymbol{0}, \boldsymbol{I_d}))$$

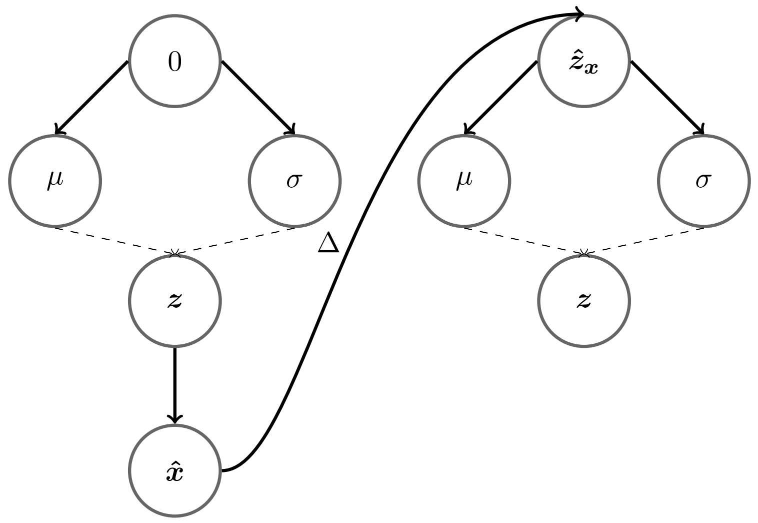


Figure 2.1: Variational GON Inference Procedure. Dashed lines indicate sampling. Straight lines indicate deterministic transformations. Inference is performed to calculate $\hat{\boldsymbol{z}}_{\boldsymbol{x}}$, which yields the parameters of q_ϕ: $\boldsymbol{\mu}(\hat{\boldsymbol{z}}_{\boldsymbol{x}})$ and $\boldsymbol{\sigma}(\hat{\boldsymbol{z}}_{\boldsymbol{x}})^2$. $\boldsymbol{z}$ is sampled from q_ϕ using the reparameterization trick.

Thus, the first-order gradient with respect to $\boldsymbol{z}_0$:

$$\nabla_{\boldsymbol{z_0}}(logp(\boldsymbol{x}|\boldsymbol{z_0}) - KL(\mathcal{N}(\boldsymbol{\mu}_\phi(\boldsymbol{z_0}), \boldsymbol{\sigma}_\phi(\boldsymbol{z_0}))||\mathcal{N}(\boldsymbol{0}, \boldsymbol{I_d})))$$

essentially acts as the encoder to infer q_ϕ. Furthermore, note that backpropagating through this network requires calculating second-order derivatives, meaning that the Hessian matrix must be computed using $\nabla_{\boldsymbol{z_0}}(logp(\boldsymbol{x}|\boldsymbol{z_0}) - KL(\mathcal{N}(\boldsymbol{\mu}_\phi(\boldsymbol{z_0}), \boldsymbol{\sigma}_\phi(\boldsymbol{z_0}))||\mathcal{N}(\boldsymbol{0}, \boldsymbol{I_d})))$. As such, GONs seem to benefit from using activation functions with non-zero second-derivatives, such as the ELU non-linearity [16].

A natural choice is to initialize $\boldsymbol{z}_0$ at the origin, given it is the mean of $p_\theta(\boldsymbol{z})$ and provides a constant point to perform inference from.

2.4 Hypernetworks

Hypernetworks [17] are networks that create the weights of other neural networks. For example, suppose we had some neural network F such that that: $\boldsymbol{x} = F(\boldsymbol{z})$, where $\boldsymbol{x}$ and $\boldsymbol{z}$ are some vectors. A hypernetwork H could be used to generate the weights of F: $\boldsymbol{W_1^H}, \boldsymbol{W_2^H}, \boldsymbol{W_3^H} = H(\boldsymbol{z})$. Let F_H be the neural network F, but with it's weight matrices replaced by $H(\boldsymbol{z})$. Then, we could predict x:

$$\boldsymbol{x} = F_H(\boldsymbol{z})$$

This is a trivial example, but hypernetworks have been used extensively in models with temporal components, such as recurrent neural networks.

2.5 Dynamic Predictive Coding Networks

In DPCNs, lower level latent variables generate each observation in an image sequence, and a higher level latent variable generates the transition dynamics of the lower level variables.

To begin, suppose we have some dataset of n image sequences of length T: $\boldsymbol{X} = \{\boldsymbol{x}^n_{1:T}\}$. Furthermore, suppose the observation $\boldsymbol{x}_t$ at time t is generated by a lower level latent variable $\boldsymbol{r}_t$. Let $\boldsymbol{r}^H$ be a higher level latent variable that conditions all the lower level vectors.

We could factorize the generative model as follows:

$$p(\boldsymbol{X}_{1:T}, \boldsymbol{r}_{1:T}, \boldsymbol{r}^h) = p(\boldsymbol{r}^h)p(\boldsymbol{r}_1)\prod_{t=1}^{T} p(\boldsymbol{x}_t|\boldsymbol{r}_t)\prod_{t=2}^{T} p(\boldsymbol{r}_t|\boldsymbol{r}_{t-1}, \boldsymbol{r}^h)$$

Suppose that $p(\boldsymbol{r}_t|\boldsymbol{r}_{t-1}, \boldsymbol{r}^h)$ is implemented by a hypernetwork H that takes in $\boldsymbol{r^h}$ and generates a transition function V for $\boldsymbol{r}_{t-1}$:

$$V = H(\boldsymbol{r}^H)$$

Then, $\boldsymbol{r^h}$ is generating the transition dynamics of $\boldsymbol{r}_{1:T}$.

MAP inference is performed in this model to infer the latent variables by minimizing the following loss function with respect to $\boldsymbol{r}_t$ and $\boldsymbol{r^h}$:

$$L = \sum_{t=1}^{T} ||\boldsymbol{x}_t - U(\boldsymbol{r}_t)||_2^2 + \sum_{t=2}^{T} ||\boldsymbol{r}_t - V(\boldsymbol{r}_{t-1})||_2^2$$

where U is some decoding function used to reconstruct the frame using the lower level latent vector. This inference is performed at each time step through a gradient descent loop.

2.6 Kalman Filters

Suppose we have some dataset of n image sequences of length T: $\boldsymbol{X} = \{\boldsymbol{x}^n_{1:T}\}$. Furthermore, suppose the observation $\boldsymbol{x}_t \in \mathcal{R}^d$ at time t is generated by some latent vector $\boldsymbol{r}_t \in \mathcal{R}^k$. In addition, suppose that all computations are performed using linear transformations, meaning that all latent variables and observations are drawn from Gaussian distributions and all probabilities are Gaussian distributions.

In a kalman filter [18], we have a dynamical model for temporal transitions:

$$\boldsymbol{r}_{t+1} = V\boldsymbol{r}_t + GW_t$$

where V and G are some matrices and $W_t \sim \mathcal{N}(0, Q)$.

We can also derive the mean $\hat{\boldsymbol{r}}_{t+1}$ and variance $\P_{t+1}$ over this distribution:

$$\hat{\boldsymbol{r}}_{t+1} = V\boldsymbol{r}_t$$
$$P_{t+1} = VP_tV + GQG^T$$

There exists an observation model, $\boldsymbol{x}_t = C\boldsymbol{r}_t + \boldsymbol{v}_t$, where $v \sim \prime, \mathcal{R}$. Computing the mean and variance:

$$E(\boldsymbol{x}_{t+1}|\boldsymbol{x}_{1:t}) = C\hat{\boldsymbol{r}}_{t+1}$$
$$E((\boldsymbol{x}_{t+1} - \hat{\boldsymbol{x}}_{t+1})(\boldsymbol{x}_{t+1} - \hat{\boldsymbol{x}}_{t+1})|\boldsymbol{x}_{1:t}) = CP_{t+1}C^T + R$$

Suppose that the Kalman filter has created a prediction with mean $\hat{\boldsymbol{r}}_{t+1}$ using the dynamical model. The

Kalman filter will then use an update to fully infer $\hat{\boldsymbol{r}}_{t+1}$ and P_{t+1}:

$$\hat{\boldsymbol{r}}_{t+1} = \hat{\boldsymbol{r}}_{t+1} + K_{t+1}(\boldsymbol{x}_{t+1} - C\hat{\boldsymbol{r}}_{t+1})$$
$$P_{t+1} = P_{t+1} - KCP_{t+1}$$

where $K_{t+1} = P_{t+1}C^T(CP_{t+1}C^T + R)^{-1}$ is the Kalman gain matrix and scales the prediction error.

Essentially, the Kalman filter first makes a prediction of the latent variable $\hat{\boldsymbol{r}}_{t+1}$ using the dynamical model. Then, it uses the observational model to predict the image, which results in a prediction error $\boldsymbol{x}_{t+1} - C\hat{\boldsymbol{r}}_{t+1}$. This prediction error is weighted by the Kalman gain and then used to update $\hat{\boldsymbol{r}}_{t+1}$.

For more details on Kalman filters (much of the math was omitted from this derivation), see [18] (which this derivation was modeled using).

Chapter 3

Single-Level GOPC Model

In this chapter, we examine the capabilities of a single-level GOPC model and compare it to an encoder-based version of the model.

Let $\boldsymbol{x}_{1:T}$ denote one video sequence of T frames. Suppose that each frame $\boldsymbol{x}_t$ is generated by some latent vector $\boldsymbol{r}_t$.

3.1 Generative Model

The generative model factorizes as follows:

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{r}_{1:T}) = \boldsymbol{r}_1 \prod_{t=1}^{T} p_\theta(\boldsymbol{x}_t|\boldsymbol{r}_t) \prod_{t=2}^{T} p_\theta(\boldsymbol{r}_t|\boldsymbol{r}_{1:t-1})$$

We chose $\boldsymbol{r}_1$ to be drawn from a standard gaussian distribution: $\boldsymbol{r}_1 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. θ denotes the parameters of the generative model. We parameterize the decoder $p_\theta(\boldsymbol{x}_t|\boldsymbol{r}_t)$ with a transpose convolutional network and the temporal transitions $p_\theta(\boldsymbol{r}_t|\boldsymbol{r}_{1:t-1})$ using three LSTM cells [19] stacked together. The latent vector on time t is dependent on all of the previous latent vectors, while the generation of the frame at time t is dependent only on the latent vector for the current timestep. Refer to Figure 3.1 for a diagram of the generative model and Algorithm 1 for the full generative process.

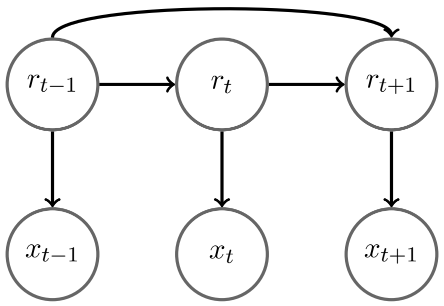


Figure 3.1: Single-Level Level GOPC Generative Model. The latent vector $\boldsymbol{r}_{t+1}$ is dependent on $\boldsymbol{r}_{1:t}$, while the frame $\boldsymbol{x}_{t+1}$ is dependent only on $\boldsymbol{r}_{t+1}$.

Algorithm 1 Generative Algorithm

```
 1: $\boldsymbol{r}_1 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
 2: $\boldsymbol{x}_1 = G(\boldsymbol{r}_1)$
 3: $h_0, c_0 = \mathbf{0}, \mathbf{0}$
 4: $t = 2$
 5: while $t \leq T$ do
 6:     $\boldsymbol{h}_t = LSTM(\boldsymbol{r}_{t-1}, \boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1})$
 7:     Calculate $\boldsymbol{\mu}_t$ and $log(\boldsymbol{\sigma}_t^2)$ from $\boldsymbol{h}_t$
 8:     Get $r_t$ through reparameterization trick on $\boldsymbol{\mu}_t$ and $log(\boldsymbol{\sigma}_t^2)$
 9:     $\boldsymbol{x}_t = G(\boldsymbol{r}_t)$
10:     $t = t + 1$
```

3.2 Inference Model

Inference aims to compute the following posterior using a one-step gradient update:

$$q_\phi(\boldsymbol{r}_{1:T}|\boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} q_\phi(\boldsymbol{r}_t|\boldsymbol{x}_t, \boldsymbol{r}_{1:t-1})$$

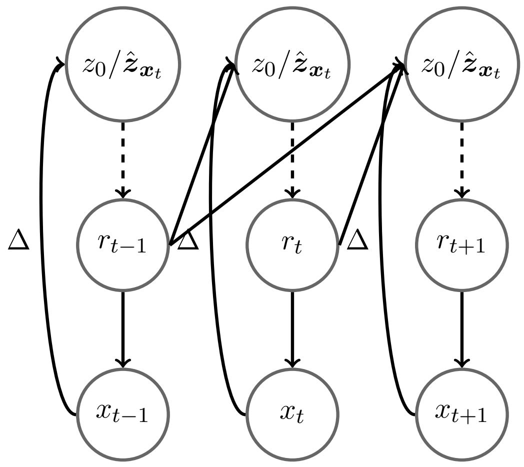
In practice, inference is performed using a one-step gradient update:

$$\hat{\boldsymbol{z}}_{\boldsymbol{x}_t}(\boldsymbol{z_0}) = \boldsymbol{z_0} + \nabla_{\boldsymbol{z_0}}(logp(\boldsymbol{x}_t|\boldsymbol{r_0}) - KL(\mathcal{N}(\boldsymbol{\mu}(\boldsymbol{z_0}), \boldsymbol{\sigma}(\boldsymbol{z_0})^2)||\mathcal{N}_P(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2)))$$
$$\boxed{= \boldsymbol{z_0} + \nabla_{\boldsymbol{z_0}}(logp(\boldsymbol{x}_t|\boldsymbol{r_0}) - KL(q_\phi(\boldsymbol{\mu}(\boldsymbol{z_0}), \boldsymbol{\sigma}(\boldsymbol{z_0})^2)||p_\theta(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2)))}$$

where $\boldsymbol{r}_0 \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{z_0}), \boldsymbol{\sigma}(\boldsymbol{z_0}))$.

The influence of $\boldsymbol{r}_{1:t-1}$ comes in through the KLD term, $KL(q_\phi(\boldsymbol{\mu}(\boldsymbol{z_0}), \boldsymbol{\sigma}(\boldsymbol{z_0})^2)||p_\theta(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2))$. By minimizing this KLD term, we condition $\boldsymbol{r}_t$ on $\boldsymbol{r}_{1:t-1}$ in some capacity.

At each time step t during inference, we initialize a latent vector, $\boldsymbol{z}_0$, at the origin. Using $\boldsymbol{z}_0$, we calculate an initial estimate over the posterior distribution. After sampling $\boldsymbol{r}_0$ from this distribution, we feed it to the decoder to create an initial estimate of the current frame $\boldsymbol{x}_t$. Then, we update $\boldsymbol{z}_0$ using the gradient of the full ELBO to get $\hat{\boldsymbol{z}}_{\boldsymbol{x}_t}$. Now, we can recalculate the posterior using $\hat{\boldsymbol{z}}_{\boldsymbol{x}_t}$, and then create the corrected posterior estimate of the current frame. Refer to Algorithm 2 for the full inference process, and Figure 3.2 for a diagram of the inference process.



$z_0/\hat{\boldsymbol{z}}_{\boldsymbol{x}_t}$
$z_0/\hat{\boldsymbol{z}}_{\boldsymbol{x}_t}$
$z_0/\hat{\boldsymbol{z}}_{\boldsymbol{x}_t}$
Δ
r_{t-1}
Δ
r_t
Δ
r_{t+1}
x_{t-1}
x_t
x_{t+1}

Figure 3.2: 1 Level GOPC Inference Model. Dashed lines indicate sampling. Straight lines indicate deterministic transformations. At each time step t, $\boldsymbol{z}_0$ is initialized at the origin and $\boldsymbol{r}_t$ (this initial estimate is also referred to as $\boldsymbol{r}_0$) is sampled from $q_\phi(\boldsymbol{\mu}(\boldsymbol{z}_0), \boldsymbol{\sigma}(\boldsymbol{z}_0)^2)$. The frame $\boldsymbol{x}_t$ is predicted and $\hat{\boldsymbol{z}}_{\boldsymbol{x}}$ is calculated using $\nabla_{\boldsymbol{z}_0} ELBO$. Then, the updated $\boldsymbol{r}_t$ is sampled from $q_\phi(\boldsymbol{\mu}(\hat{\boldsymbol{z}}_{\boldsymbol{x}_t}), \boldsymbol{\sigma}(\hat{\boldsymbol{z}}_{\boldsymbol{x}_t})^2)$.

This model can be though of as a nonlinear Kalman filter in which temporal dynamics are modeled through the LSTM prior (from the generative model) and the Kalman update occurs through the one-step gradient update.

3.3 Learning

To do the gradient update, we minimize the following loss function:

$$MSE(\boldsymbol{x}_t|\hat{\boldsymbol{x}}_{\boldsymbol{r}_0}) + KL(q_\phi(\boldsymbol{\mu}(\boldsymbol{z_0}), \boldsymbol{\sigma}(\boldsymbol{z_0})^2)||p_\theta(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2))$$

Algorithm 2 Inference Algorithm

```
1: $t = 1$
2: **while** $t \leq T$ **do**
3:     $\boldsymbol{z}_0 = \mathbf{0}$
4:     Calculate $\boldsymbol{\mu}_t$ and $log(\boldsymbol{\sigma}_t^2)$ from $\boldsymbol{z}_0$
5:     Get initial estimate of $\boldsymbol{r}_t$ ($\boldsymbol{r}_0$) through reparameterization trick on $\boldsymbol{\mu}_t$ and $log(\boldsymbol{\sigma}_t^2)$
6:     $\hat{\boldsymbol{x}}_{\boldsymbol{r}_0} = G(\boldsymbol{r}_0)$
7:     $ELBO = (\boldsymbol{x}_t - \hat{\boldsymbol{x}}_{\boldsymbol{r}_0})^2 + KL(q_\phi||p_\theta)$
8:     $\hat{\boldsymbol{z}}_{\boldsymbol{x}_t} = \boldsymbol{z}_0 - \nabla_{\boldsymbol{z}_0} ELBO$
9:     Recalculate $\boldsymbol{\mu}_t$ and $log(\boldsymbol{\sigma}_t^2)$ from $\hat{\boldsymbol{z}}_{\boldsymbol{x}_t}$
10:    Get corrected posterior estimate of $\boldsymbol{r}_t$ through reparameterization trick on $\boldsymbol{\mu}_t$ and $log(\boldsymbol{\sigma}_t^2)$
11:    $\hat{\boldsymbol{x}}_{\boldsymbol{r}_t} = G(\boldsymbol{r}_t)$
12:    $t = t + 1$
```

We train the weights of the model through backpropagation to minimize the ELBO loss:

$$MSE(\boldsymbol{x}_t|\hat{\boldsymbol{x}}_{\boldsymbol{r}_t}) + KL(q_\phi(\boldsymbol{\mu}(\hat{\boldsymbol{z}}_{\boldsymbol{x}_t}), \boldsymbol{\sigma}(\hat{\boldsymbol{z}}_{\boldsymbol{x}_t})^2)||p_\theta(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2))$$

where $\boldsymbol{r}_t$ is sampled from $q_\phi(\boldsymbol{\mu}(\hat{\boldsymbol{z}}_{\boldsymbol{x}_t}), \boldsymbol{\sigma}(\hat{\boldsymbol{z}}_{\boldsymbol{x}_t})^2)$.

The weights of the network are updated using the ELBO calculated after doing the gradient update. Note that the hessian matrix must be calculated since second-order derivatives are used. Refer to 3 for the full learning algorithm.

Algorithm 3 Learning Algorithm

```
1: **for** each minibatch **do**
2:     $t = 1$
3:     $total\_loss = 0$
4:     **while** $t \leq T$ **do**
5:         Calculate prior distribution, $p$
6:         Calculate posterior distribution, $q$
7:         Get $\hat{\boldsymbol{x}}_t$ using sample from $q$
8:         Calculate $ELBO = (\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t)^2 + KL(q||p)$
9:         $total\_loss = total\_loss + ELBO$
10:        $t = t + 1$
11:    Backprop $total\_loss$ and update parameters of model
```

3.4 Experiments

We trained our models using the Adam [20] optimizer, a learning rate of 1e-3, and batch size of 50. We trained each model for a total of 50 epochs. We evaluate our models on the Smooth Moving MNIST dataset. We use an instantiation of the dataset in which each video contains one digit moving across a frame of size 64x64 pixels for 20 timesteps.

3.4.1 GOPC-1 Results

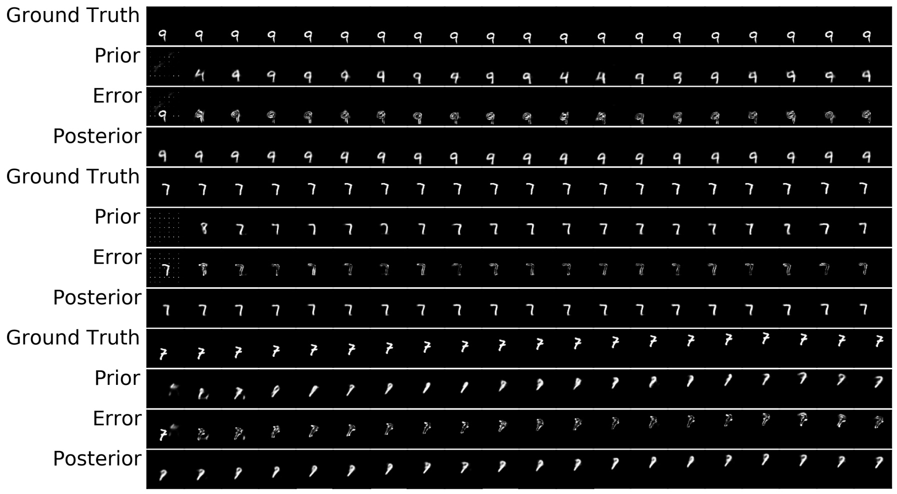


Figure 3.3: GOPC Inference Results. Three sequences are shown. For each sequence, we show the ground truth sequence, the prediction by the prior (prediction before gradient update), the error between the prior prediction and the ground truth sequence, and the posterior correction.

Figure 3.3 demonstrates how the model performs when it can correct itself with the prediction error. When the error of the sequence predicted by the prior is high (i.e. at the beginning two timesteps when information about dynamics is known), the posterior is able to produce much better reconstructions than the prior.

Figure 3.4 demonstrates how well the model performs when predicting into the future using the prior (so no error correction is present). After a few timesteps, the lack of prediction error means the model gets confused and starts changing the digit identity (i.e. a “9” to a “4”).

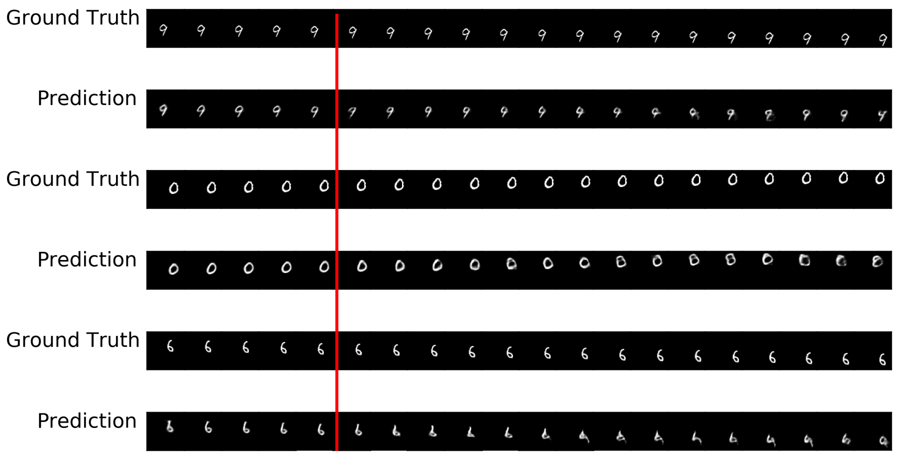


Figure 3.4: GOPC Predictions Conditioned on Five Context Frames. Three sequences are shown. For each sequence, we show the ground truth sequence and predictions. Predictions before the red line are using the posterior (so after the gradient update). After the red line, we turn off the error correction and sample for 15 time steps into the future using the prior.

3.4.2 Comparison to Using an Encoder

In this subsection, we build a version of the GOPC-1 model that uses an encoder to perform inference instead of the one-step gradient update. We will refer to this model as VAE-1. The generative model is the same as GOPC-1:

$$p(\boldsymbol{x}_{1:T}, \boldsymbol{r}_{1:T}) = \boldsymbol{r}_1 \prod_{t=1}^{T} p_\theta(\boldsymbol{x}_t|\boldsymbol{r}_t) \prod_{t=2}^{T} p_\theta(\boldsymbol{r}_t|\boldsymbol{r}_{1:t-1})$$

The inference model is quite similar:

$$q_\phi(\boldsymbol{r}_{1:T}|\boldsymbol{x}_{1:T}) = \prod_{t=1}^{T} q_\phi(\boldsymbol{r}_t|\boldsymbol{x}_t)$$

There are two primary differences. First, the inference model in VAE-1 uses a convolutional neural network to encode each frame. Second, the inference model in VAE-1 conditions $\boldsymbol{r}_t$ only on the current latent vector. GOPC-1 conditions $\boldsymbol{r}_t$ on $\boldsymbol{r}_{1:t-1}$ through the KLD term $KL(q_\phi(\boldsymbol{\mu}(\boldsymbol{z_0}), \boldsymbol{\sigma}(\boldsymbol{z_0})^2)||p_\theta(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2))$. However, we cannot represent this error minimization in the same way using an encoder, since there is no prediction error. Although the encoder is implicitly doing something similar (since its weights are trained through minimizing both reconstruction error and KLD), it is not directly doing this in the same manner on each example at test time. It would be possible to condition $\boldsymbol{r}_t$ on $\boldsymbol{r}_{1:t-1}$ using some sort of recurrent neural network in the posterior; however, this would add extra parameters to VAE-1 and also lead to further architectural differences, which would further exacerbate issues of unfairness in the comparison. We view this current architecture as the fairest way to do a comparison between VAE-1 and GOPC-1, with the note that VAE-1 may lose some representational power since its encoder gets no temporal information.

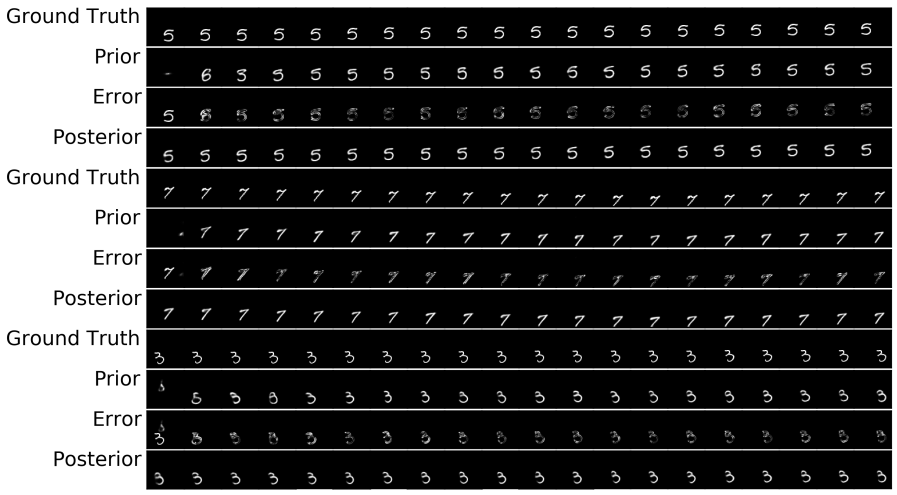


Figure 3.5: VAE-1 Inference Results. Three sequences are shown. For each sequence, we show the ground truth sequence, the prediction by the prior, the error between the prior prediction and the ground truth sequence, and the posterior prediction (using an encoder).

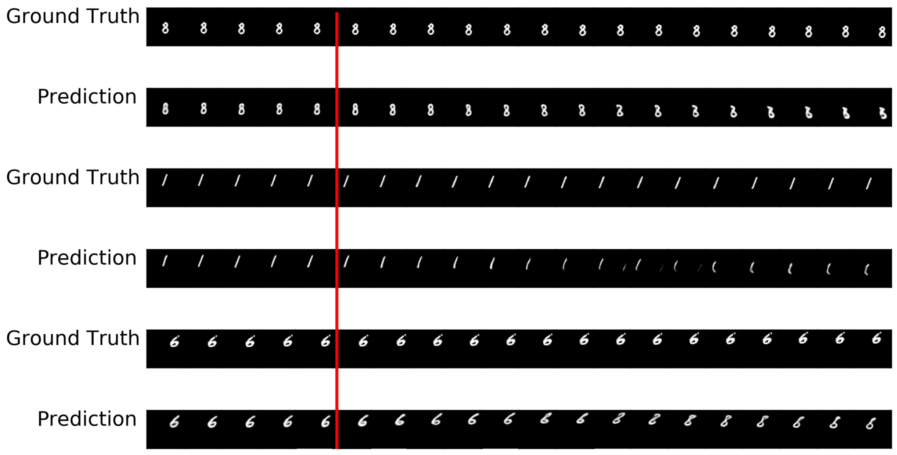


Figure 3.6: VAE-1 Predictions Conditioned on Five Context Frames. Three sequences are shown. For each sequence, we show the ground truth sequence and predictions. Predictions before the red line are using the posterior (so using an encoded frame). After the red line, we sample for 15 time steps into the future using the prior.

Model	Number of Parameters	Mean (↓)	STD (↓)
GOPC-1	**3532610**	0.0252	0.0086
VAE-1	3993986	**0.0206**	**0.0064**
DIFFERENCE	461376	0.0046	0.0022

Table 3.1: Comparison between posterior predictions of GOPC-1 and VAE-1 for full video sequences. We compare the number of parameters in GOPC-1 and VAE-1, as well as the mean and standard deviation (STD) of reconstruction error across the test set.

Model	Number of Parameters	Mean (↓)	STD (↓)
GOPC-1	**3532610**	0.0393	0.0108
VAE-1	3993986	**0.0374**	**0.0102**
DIFFERENCE	461376	0.0019	0.0006

Table 3.2: Comparison between prior predictions of GOPC-1 and VAE-1 for full video sequences. We compare the number of parameters in GOPC-1 and VAE-1, as well as the mean and standard deviation (STD) of reconstruction error across the test set.

Figure 3.5 demonstrates how the model performs when it reconstructs the frame using an encoder. Figure 3.6 demonstrates how well the model performs when predicting into the future using the prior (so it can no longer use encoded frames to assist itself). It seems that digit identity is maintained better while predicting into the future with VAE-1 than GOPC-1, suggesting that VAE-1 has learned a better prior.

Refer to Table 3.1 for a quantitative assessment of the posterior prediction capabilities of GOPC-1 and VAE-1 across the test set. Even though GOPC-1 saves parameters, VAE-1 is consistently better at reconstructing samples using the posterior.

Refer to Table 3.2 for a quantitative assessment of the prior prediction capabilities of GOPC-1 and VAE-1 across the test set. Again, even though GOPC-1 saves parameters, VAE-1 is consistently better at the task of next-frame prediction using the prior.

Chapter 4

Hierarchical GOPC Model

In this chapter, we look at implementing a GOPC version of DPCN, which was described in Chapter 2.5. We will refer to this hierarchical GOPC model as “GOPC-2”.

4.1 GOPC-2 Implementation

In the GOPC version of DPCN, the generative model remains the same:

$$p(\boldsymbol{X}_{1:T}, \boldsymbol{r}_{1:T}, \boldsymbol{r}^h) = p(\boldsymbol{r}^h)p(\boldsymbol{r}_1)\prod_{t=1}^{T} p(\boldsymbol{x}_t|\boldsymbol{r}_t)\prod_{t=2}^{T} p(\boldsymbol{r}_t|\boldsymbol{r}_{t-1}, \boldsymbol{r}^h)$$

where $p(\boldsymbol{r}_t|\boldsymbol{r}_{t-1}, \boldsymbol{r}^h)$ implemented by a hypernetwork so that $\boldsymbol{r}^{\boldsymbol{h}}$ is generating the transition dynamics of $\boldsymbol{r}_{1:T}$.

However, rather than use a gradient descent loop to infer the latent variables, inference at time t is performed in two steps. The first step uses a gradient update to infer the lower level latent variable $\boldsymbol{r}_t$ in the same manner as GOPC-1:

$$q_\phi(\boldsymbol{r}_t|\boldsymbol{x}_t) = q_\phi(\boldsymbol{r}_t|\boldsymbol{x}_t, \boldsymbol{r}_{1:t-1})$$

After inferring $\boldsymbol{r}_t$, GOPC-2 updates $\boldsymbol{r}^H$ using the prediction error between the actual $\boldsymbol{r}_t$ and the prior prediction $\hat{\boldsymbol{r}}_t$, as well as a KLD term to regularize the latent space of $\boldsymbol{r}^H$. This gradient update is implemented as follows:

$$\boldsymbol{r}^H = \boldsymbol{r}^H - \nabla_{r^H}(||\boldsymbol{r}_t - \hat{\boldsymbol{r}}_t||_2^2 + KL(q_\phi(\boldsymbol{r}^H)||\mathcal{N}(\mathbf{0}, \boldsymbol{I})))$$

where $\hat{\boldsymbol{r}}_t$ is predicted using the transition function generated by H($\boldsymbol{r}^H$) (H refers to the hypernetwork).

4.2 Instability

I was not able to successfully train GOPC-2; within a few iterations, gradients would explode to extremely large values, resulting in "Not a Number" (NaN) values appearing and crashing the model. It appears the performing GON-style inference through a hypernetwork is very unstable, as the hessian matrix must be computed numerous times through a hypernetwork, which has a huge number of multiplications. Using more gradient steps to infer $\boldsymbol{r}^H$ simply exacerbated the gradient explosion problem. Regularizing the weights of the predicted transition function, adding layer normalization [21] or batch normalization [22] to the hypernetwork, applying a TanH activation function, layer normalization, or batch normalization to the weights of the transition function, and clipping the gradients of $\nabla_{\boldsymbol{r}^H}$ each did not not solve the problem. Tuning the learning rate of the model did not help either.

4.3 Stabilizing Techniques

Only two techniques seemed to ease some of the problems with gradient explosion: applying a sigmoid activation function to the vector predicted by the transition function, or feeding that predicted vector through a layer normalization layer.

The problem of gradient explosion still occurred sometimes on the full Smooth Moving MNIST dataset. However, when using just a single digit or a few digits, it was possible to train the network without it generating NaN values and crashing.

However, the model was not able to produce quality predictions using the prior, suggesting that these techniques significantly weakened what the prior was able to learn.

Chapter 5

Limitations

There are three general limitations I have identified with the GON inference procedure:

- Learning and inference are unstable when using a hypernetwork and can lead to exploding gradients that quickly crash the model, making it very difficult to train.
- Although less parameters are required compared to an encoder-based model, it is significantly slower.
- Difficult to implement with multiple latent variables.

Chapter 4.3 already discussed the issue of instability. In this chapter, we will focus on the issues of training speed and implementation difficulty.

5.1 Speed of Model

Although less parameters are required in a model using GON inference without the need of an encoder, the model has to use second-order derivatives in its optimization, since partial derivatives must be calculated with respect to the inferred vector: $\nabla_{\boldsymbol{z_0}}(logp(\boldsymbol{x}_t|\boldsymbol{r_0}) - KL(q_\phi(\boldsymbol{\mu}(\boldsymbol{z_0}), \boldsymbol{\sigma}(\boldsymbol{z_0})^2)||p_\theta(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2)))$. As such, the Hessian matrix using $\nabla_{\boldsymbol{z_0}}(logp(\boldsymbol{x}_t|\boldsymbol{r_0}) - KL(q_\phi(\boldsymbol{\mu}(\boldsymbol{z_0}), \boldsymbol{\sigma}(\boldsymbol{z_0})^2)||p_\theta(\boldsymbol{\mu}_t, \boldsymbol{\sigma}_t^2)))$ must be calculated, which is computationally more demanding than a feedforward encoder.

I evaluated the training speed of GOPC-1 and VAE-1 used a training set of 17500 data samples with a batch size of 50 (350 batches total). I evaluated how long it took each model to train for one epoch on

Model	Number of Parameters	Time (seconds) (↓)	Batches / Second (↑)
GOPC-1	**3532610**	206	1.7
VAE-1	3993986	**138**	**2.5**
DIFFERENCE	461376	68	0.8

Table 5.1: Comparison between between training speed of GOPC-1 and VAE-1. We compare the number of parameters in GOPC-1 and VAE-1, how long each model took to train for one epoch on the training set (17500 samples, batch size 50, 350 batches), and the training rate in batches per second.

the entire training set (including adjusting the weights via backpropagation). My implementation of the model is on PyTorch, and I evaluated the model using a GTX 1080 Ti GPU. Refer to 5.1 for the results of the comparison. In short, although GOPC-1 saves 461376 parameters, VAE-1 trains around runs through a batches around 33% faster.

5.2 Difficult to Implement

Furthermore, it can be quite difficult to implement hierarchical generative models with multiple latent variables using GON inference. For example, consider the generative model from Clockwork VAE (CWVAE) [13], shown in Figure 5.1:

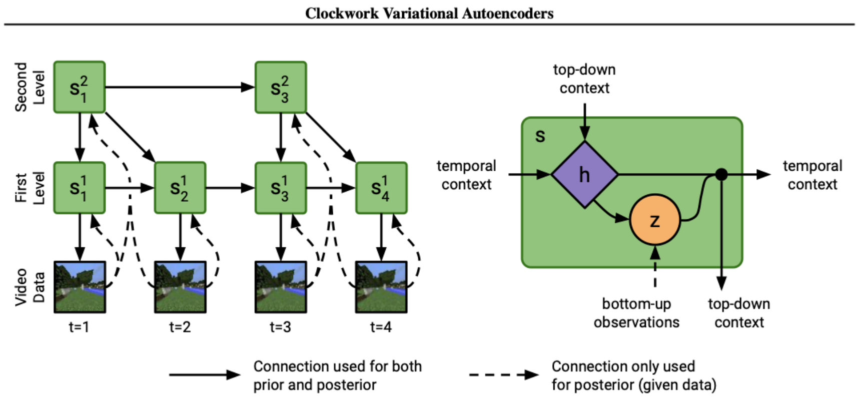


Figure 5.1: Clockwork VAE Generative Model. Figure taken directly from [13].

In this generative model, higher-level latent variables operate over longer timescales than the levels below it. For example, the same variable on the second level may condition two variables on the first level, meaning that the second level variable needs to be inferred using two input frames. Each of the first-level variables only need one input frame to infer.

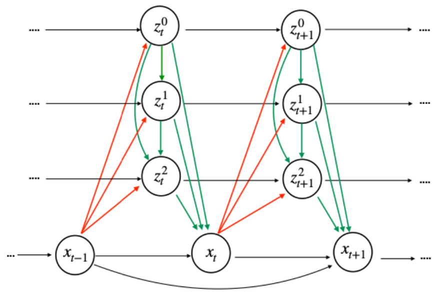


Figure 5.2: Improved Conditional VRNNs Generative Model. Figure taken directly from [12].

The question arises: how do we infer these latent variables using GON inference? Do we use multiple zero vectors, infer the high-level variable first using the prediction error from two input frames, and then infer the two lower-level variables that are conditioned on this higher-level variable?

For example, we could infer $\hat{\boldsymbol{z}}^2_{\boldsymbol{x}_1,\boldsymbol{x}_2}$ (and then sample $\boldsymbol{s}_1^2$ from the resulting posterior parameters) using:

$$\hat{\boldsymbol{z}}^2_{\boldsymbol{x}_1,\boldsymbol{x}_2}(\boldsymbol{z}_0^2) = \nabla_{\boldsymbol{z}_\mathbf{0}^\mathbf{2}}(logp_\theta(\boldsymbol{x}_1|\boldsymbol{s}_1^1,\boldsymbol{s}_1^2) + logp(\boldsymbol{x}_2|\boldsymbol{s}_2^1,\boldsymbol{s}_1^2)) - KLD(q_\phi(\boldsymbol{\mu}(\boldsymbol{z}_\mathbf{0}^\mathbf{2}),\boldsymbol{\sigma}(\boldsymbol{z}_\mathbf{0}^\mathbf{2})^2)||p_\theta(\boldsymbol{\mu}_{\boldsymbol{s}_1^2},\boldsymbol{\sigma}^2_{\boldsymbol{s}_1^2})))$$

Then, we could infer $\hat{\boldsymbol{z}}^1_{\boldsymbol{x}_1}$ (and then sample $\boldsymbol{s}_1^1$):

$$\hat{\boldsymbol{z}}^1_{\boldsymbol{x}_1}(\boldsymbol{z}_0^1) = \nabla_{\boldsymbol{z}_\mathbf{0}^1}(logp_\theta(\boldsymbol{x}_1|\boldsymbol{s}_1^1,\boldsymbol{s}_1^1) - KLD(q_\phi(\boldsymbol{\mu}(\boldsymbol{z}_\mathbf{0}^\mathbf{1}),\boldsymbol{\sigma}(\boldsymbol{z}_\mathbf{0}^\mathbf{2})^2)||p_\theta(\boldsymbol{\mu}_{\boldsymbol{s}_1^1},\boldsymbol{\sigma}^2_{\boldsymbol{s}_1^1}|\boldsymbol{s}_1^2)))$$

Then, we could infer $\hat{\boldsymbol{z}}^1_{\boldsymbol{x}_2}$ (and then sample $\boldsymbol{s}_2^1$):

$$\hat{\boldsymbol{z}}^1_{\boldsymbol{x}_2}(\boldsymbol{z}_0^1) = \nabla_{\boldsymbol{z}_\mathbf{0}^1}(logp_\theta(\boldsymbol{x}_2|\boldsymbol{s}_2^1,\boldsymbol{s}_1^2) - KLD(q_\phi(\boldsymbol{\mu}(\boldsymbol{z}_\mathbf{0}^\mathbf{1}),\boldsymbol{\sigma}(\boldsymbol{z}_\mathbf{0}^\mathbf{2})^2)||p_\theta(\boldsymbol{\mu}_{\boldsymbol{s}_2^1},\boldsymbol{\sigma}^2_{\boldsymbol{s}_2^1}|\boldsymbol{s}_1^2)))$$

This inference procedure would require three gradient updates (one to arrive at $\boldsymbol{s}_1^2$, one to arrive at $\boldsymbol{s}_1^1$, and one to arrive at $\boldsymbol{s}_2^1$), which would significantly slow down the model. Other implementations may require even more gradient updates.

Consider the generative model from [12] (Figure 5.2) as another example. In this generative model, each higher-level latent vector conditions all of the latent vectors below it. In addition, each latent vector is fed into the decoder to predict the image.

The question again arises: how would we implement this using the GON inference framework? Would we use multiple gradient steps? Would we use multiple zero vectors? Would we just one zero vector at the highest level and perform top-down inference? Would we get rid of the zero vector and choose to update something else using the gradient?

In summary, there exist issues in the GON inference procedure when it is unclear what the noisy "observation" should be set to in cases where there are multiple hierarchical or sequential latent variables. In the case of both CWVAE and Improved Condititional VRNNs, I believe it it is significantly simpler to use an encoder for inferring the posterior of the latent variables implementing amortized variational inference.

Chapter 6

Conclusion

In this thesis, I have proposed Gradient Origin Predictive Coding and demonstrated its strengths and weaknesses through two different parameterization design choices.

In particular, I have provided a working implementation of a single level GOPC model (GOPC-1) and demonstrations of its inference and generative process and performances. I have also compared GOPC-1 to an encoder-based version of the model (VAE-1).

Furthermore, I have provided an implementation of a two-level, hierarchical GOPC model (GOPC-2).

Lastly, I have provided an analysis of the limitations of the GON inference framework and some potential fixes.

Future work could focus on stabilizing the training process of the proposed hypernetwork model GOPC-2. In addition, it would be interesting to see if it is possible to use the GON inference procedure in other hierarchical latent variable models such as CWVAE.

Bibliography

1. Bond-Taylor, S. & Willcocks, C. G. Gradient origin networks. *arXiv preprint arXiv:2007.02798* (2020).
2. Srivastava, N., Mansimov, E. & Salakhudinov, R. *Unsupervised learning of video representations using lstms* in *International conference on machine learning* (2015), 843–852.
3. Rao, R. P. & Ballard, D. H. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience* **2,** 79–87 (1999).
4. Huang, Y. & Rao, R. P. Predictive coding. *Wiley Interdisciplinary Reviews: Cognitive Science* **2,** 580–593 (2011).
5. Bryson, A. E. & Ho, Y.-C. Applied optimal control, revised printing. *Hemisphere, New York* (1975).
6. Rao, R. Robust Kalman filters for prediction, recognition, and learning (1996).
7. Lotter, W., Kreiman, G. & Cox, D. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104* (2016).
8. Jiang, L. P., Gklezakos, D. C. & Rao, R. P. Dynamic predictive coding with hypernetworks. *bioRxiv* (2021).
9. Goodfellow, I. *et al.* Generative adversarial nets. *Advances in neural information processing systems* **27** (2014).
10. Mescheder, L., Geiger, A. & Nowozin, S. *Which training methods for GANs do actually converge?* in *International conference on machine learning* (2018), 3481–3490.
11. Li, Y. & Mandt, S. Disentangled sequential autoencoder. *arXiv preprint arXiv:1803.02991* (2018).

12. Castrejon, L., Ballas, N. & Courville, A. *Improved conditional vrnns for video prediction* in *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), 7608–7617.

13. Saxena, V., Ba, J. & Hafner, D. Clockwork variational autoencoders. *Advances in Neural Information Processing Systems* **34** (2021).

14. Park, J. J., Florence, P., Straub, J., Newcombe, R. & Lovegrove, S. *Deepsdf: Learning continuous signed distance functions for shape representation* in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2019), 165–174.

15. Kingma, D. P. & Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

16. Clevert, D.-A., Unterthiner, T. & Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

17. Ha, D., Dai, A. & Le, Q. V. Hypernetworks. *arXiv preprint arXiv:1609.09106* (2016).

18. Naik, A., D, B. & Chen, B. *Lecture 11: Kalman Filtering and Topic Models* Feb. 2019.

19. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural computation* **9,** 1735–1780 (1997).

20. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

21. Ba, J. L., Kiros, J. R. & Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

22. Ioffe, S. & Szegedy, C. *Batch normalization: Accelerating deep network training by reducing internal covariate shift* in *International conference on machine learning* (2015), 448–456.