Decomposing Temporal latent variable using Compositional Energy functions

GyuBin Lee
2019100019
Korea University
dlrbqls0521@korea.ac.kr

YoonSeok Oh
2018320203
Korea University
bd9983@korea.ac.kr

Sukyung Baek
2021320305
Korea University
sk100@korea.ac.kr

Paul Lee
2020130003
Korea University
egilnu2018@korea.ac.kr

Abstract

Compositionality is a crucial aspect for intelligent systems. Humans structure information into disentangled factors and create new concepts by recombining and reconfiguring these disentangled factors. Recently, various models enabling compositional generation have been proposed[2–4]. These models are all based on energy-based models(EBMs)[8] and generate images that include each learned factor by learning each factor as an energy function at training time separately and combining these energy functions at inference time. Furthermore, they trained the model that learn each factor in an unsupervised manner by just looking at images. We have tried with a model that learns to generate sequential data based on the framework of energy-based models that enable such structural compositionality. While previous studies focused on extracting factors within a single image without considering the temporal aspect, we configured the model to learn and generate the latent factors of transformations occurring over time. Following the flow of previous papers, we conducted the following three experiments: 1) First, learn and generate each energy function according to each factor. 2) Freeze the pre-trained encoder and share it while learning and generating energy functions for each factor. 3) Extract factors in an unsupervised manner and regenerate using the extracted factors. As a result, we observed excellent generative outcomes when handling color and scale factors, confirming the ability to manipulate factors for temporal data using EBM.

1. Introduction

Compositionality is important feature for intelligent systems. Representing compositionally means that each compositional unit must be independent of each other(we call Modularization). Because changing one factor in the model should not alter the others. Also, we need to well use those factorized units(we call Generalization). Therefore, for compositional representations, the following are necessary: 1. Disentangled representation which we call modularization 2. Selective policy mechanism using each disentangled factor modules for generalization. We focused on the first key issue, modularization, and structured the model accordingly. We replaced the second part which is the attention policy utilizing each module, with a selection mechanism of our own for simplicity for this project.

1.1. Generalizability

A representative example of compositionality is language. The meanings of individual words are loosely connected (family resemblance) while sufficiently separated(quantized), so that we can create infinite and complex meanings using simple and finite words. When encountering new concepts, we break them down into known components and combine the conceptual prototypes of each unit to understand them. This aids in generalization. Recent paper[2] has shown that using multiple simple generative models allows for good generalization beyond the training data[Figure 1]. It argues that training and combining multiple small and simple models can be a more effective and efficient approach than training a single large model end-to-end. Therefore, we conducted the project under the broad framework that compositionality is a desirable property. In our project, we aim to express this generalizability through the combination of individual energy models.

1.2. Modularization

Learning disentangled representations allows for the utilization of modularization. As mentioned earlier, if the sys-

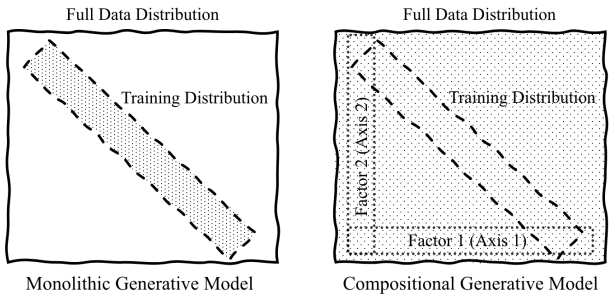

Figure 1. An image from [2]. This image illustrates that combining two orthogonal small models can generalize beyond the learned data.

tem is sufficiently modularized, changing one part should not affect the others(like language word). In our project, we aimed to introduce an inductive bias towards modularization by representing each modules as different energy functions with neural networks that have distinct weights. Furthermore, it was demonstrated in [4] that it is possible to extract relevant disentangled factors from images in an unsupervised manner. This means that instead of learning each energy function based on predefined concepts separately, we can learn the energy functions all at once, and they will naturally extract important disentangled factors that repeatedly appear in the data. Leveraging this characteristic, we also attempted to extract temporal factors in an unsupervised manner.

1.3. Purpose of this project

The purpose of this project is to build and train a toy model to understand compositional generation. In brief, we aim to apply the existing Compositional EBM framework to videos, training a model that can generate videos. Due to resource constraints, we constructed a small toy model, which will allow us to gain a deeper understanding of EBM (learned sampler) and compositionality.

Large Vision Language Models might already perform compositional generation by learning the correlation between text tokens and image patterns. Therefore, our proposed method may not be entirely novel and necessary. However, we believe that training multiple small and simple models and combining them is more effective and efficient approach than training a single large model. To summarize, the objectives of our project are:

1. To understand compositional generation through EBM.
2. To implement controllable generation by combining multiple small generation models.

1.4. Basic notations

In this paper, we have not distinguished notation between scalars and vectors, so in the expression $y = f(x)$, y can mean a real value or it can be a vector of real numbers. We also used the notation for deterministic variables and random variables without distinction.

$\mathbb{E}[x]$ denotes the expectation of a random variable x. $\mathbb{E}_{p(x)}[x]$ means that we take the expectation of x with respect to the probability distribution $p(x)$.

2. Related works

2.1. Autoencoder

Autoencoders are one of the earliest methods proposed to learn the semantic representation of an image in an unsupervised manner. It consists of two parts: one is a neural network called an encoder, which we will denote as $\text{Enc}_\phi(x)$, which embeds an image x into a lower-dimensional latent representation z. The other is a neural network called a decoder, which we will denote as $\text{Dec}_\theta(z)$, which attempts to recover the original image x from the latent representation z.

During training, the autoencoder tries to minimize the distance between the original images and the reconstructed images. More precisely, the autoencoder tries to minimize the expectation of the L2 distance between the original image x and the reconstructed image $\hat{x}$:

$$\mathcal{L}_{\text{AE}} = \mathbb{E}_{p_{\text{data}}(x)}[\|x-\hat{x}\|_2^2] = \mathbb{E}_{p_{\text{data}}(x)}[\|x-\text{Dec}_\theta(\text{Enc}_\phi(x))\|_2^2]$$

The autoencoder trained in this way typically provides a latent representation that is more compact and tractable than the image itself. Although traditional autoencoders are effective at finding a compressed representation of an image, the latent representation they find may not be helpful in generating new data that has never been seen during training. Roughly speaking, Autoencoder is designed to reconstruct the original x from z with a small error, so it may be inappropriate for the task of creating a new, never-before-seen sample by deliberately removing information from x and adding new information in its place.

The Variational Autoencoder(VAE)[7] is a generative model that adds a stochastic component to a traditional autoencoder. A VAE, similar to an autoencoder, consists of a probabilistic encoder $q_\phi(z|x)$ and a probabilistic decoder $\log p_\theta(x|z)$. Its loss function can be understood as the traditional reconstruction cost plus a new term, the prior matching cost.

$$\begin{aligned}\mathcal{L}_{\text{VAE}} &= -\mathbb{E}_{q_\phi(z|x)}\Big[\log p_\theta(x|z)\Big] + D_{\text{KL}}\big(q_\phi(z|x)||p(z)\big) \\ &= \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{prior}}\end{aligned}$$

VAE and its variants, such as beta-VAE, FactorVAE, and DDPAE, are more effective at learning a disentangled representation of an image than traditional autoencoders, and

we can trace the reason for this to the prior matching term. Minimizing KL divergence between $q_\phi(z|x)$ and $p(z)$ can be interpreted as removing some of the information in x from z, replacing it with information contained in $p(z)$, and then sampling x from z. This is because after training, during inference, we can sample z from $p(z)$, which has similar properties, instead of $q_\phi(z|x)$, and generate a never-before-seen x from that.

2.2. Flow Factorized Representation Learning

In the Flow Factorized Representation Learning[10] paper, the authors present a solution using Flow factorized VAE to the problem of predicting future video frames given $T+1$ video frames $x_0, x_1, x_2, ..., x_T$. Instead of directly predicting x_{t+1} in image space, they first embed x_t into z_t in latent space, and then transforms z_t to z_{t+1} using the transformation in latent space that corresponds to the transformation in the image space, and finally constructing x_{t+1} from z_{t+1}.

2.3. Diffusion Model

Diffusion models, especially Denoising Diffusion Probabilistic Models(DDPM)[6] and Classifier-Free Diffusion Guidance[6] provide important insights into the process of discovering compositional concepts. In short, these studies tell us that generating samples beyond the training distribution, or generating samples that are both realistic and consistent with a given concept, requires a process of iteratively removing information from an image x while simultaneously iteratively injecting new information.

For example, let's say x is an image of a human face and z_1, z_2 are composable concepts extracted from x. What constraints should be imposed on these concepts?

Let z_1 be a latent vector representing the color of this person's eyes and z_2 be a latent vector representing this person's skin color. Given an image X, these concepts can be completely determined. But now let's think about the reverse process: Given a person's eye color z_1, there still remains a large amount of uncertainty that makes it difficult to predict x, which is actually a good thing. This is because in order for a concept to generate samples well beyond the training distribution, the concept must lose some information about the training domain. To borrow a term from information theory, given an image x and the concepts $z_1, \cdots, z_n$, the mutual information $I(x; z_k)$ between x and z_k should not be too large for each k.

In the training phase, the DDPM is trained to first remove a large amount of information from x_t and then predict information about x_0 using the decoder $\epsilon_\theta(x_t, t)$. At the inference step, DDPM gradually adds information to the noisy sample x_t using a decoder $\epsilon_\theta(x_t, t)$ to restore x_0. Also, in Classifier-Free Diffusion Guidance, the method used is to iteratively inject information about condition c during the process of generating an image from noise to produce an image that satisfies the condition. In the inference phase, both are using a method of iteratively injecting new information into the image rather than generating it in a single step, which is one of the hallmarks of energy-based models, as we will discuss below.

2.4. Energy Based Model

Energy-based models[8] are a way to understand probabilistic and deterministic models within a unified framework. The energy-based model framework encompasses discriminative models, VAEs, and diffusion models. In general, the energy $E_\theta(x)$ of a random vector x can be treated as a kind of cost imposed on that vector.

Therefore, training an energy-based model somehow involves minimizing the energy function from specific training samples. Typically, energy functions are designed to learn the unnormalized log-probability of training data. In other words, the energy function $E_\theta(x)$ is usually defined as follows:

$$p_\theta(x) = \frac{\exp(-E_\theta(x))}{Z(\theta)} \text{ where } Z(\theta) = \int \exp(-E_\theta(x))dx$$

$$\log p_\theta(x) = -E_\theta(x) - \log Z(\theta)$$

A key feature of energy-based models is that there are many ways to compose multiple energy-based models to create a new energy-based model. This is possible because the energy function outputs a scalar value, unlike discriminative models, which typically output vectors, and because there is no constraint that the integration of the energy function must be 1, unlike the probability distribution that generative models learn.
The simplest way to combine energy functions is to take a weighted sum. Intuitively, we can think of each energy function $E_\theta(x)$ as implicitly imposing a constraint on x and assigning a low energy value to x that satisfies that constraint well and a high energy value to x that does not. So, if we take the weighted sum of two energy functions, we can think of the new energy function $E(x) = \alpha_1 E_1(x) + \alpha_2 E_2(x)$ as imposing a constraint similar to the constraint imposed by taking weighted AND operation on the constraint imposed by $E_1(x)$ the constraint imposed by $E_2(x)$.

3. Main Method

First, in section 3.1, we will introduce the general framework used in this experiment in the order of the reason why we use the Energy Based Model framework, model architecture, formulation, and algorithm. Then, in sections 3.2,

3.3, and 3.4, we will sequentially explain how we conducted experiments by applying the general framework from section 3.1 in slightly different ways.

3.1. Compositional Energy functions

3.1.1 Why we use Energy Based Model

We chose to use energy based model(EBM) framework because energy based frameworks have significant advantage of being able to combine multiple models during the inference or sampling phase as discussed at section 2.4. We believe this is what sets energy based models apart from other models(though diffusion models also share this advantage. However, we noted that the score function learned by diffusion models is similar to the gradient of the energy function in energy-based models. Thus, for this project, we decided to focus solely on energy-based models.)

As mentioned earlier, main advantage is that it allows us to separate the training and inference phases, enabling compositional generation. Additionally, we understand this project from the perspective of inference as constraint optimization, where each energy function learns key constraints, and the optimization process continues until data satisfying all constraints is sampled[5]. Previous paper [3] has shown that this series of processes operates similarly to an "AND" operator. While the paper also proposed "OR" and "NOT" operators, this project focuses solely on the "AND" property.

3.1.2 Dataset

To understand how well we can adjust the desired factors, we needed a minimal dataset excluding other factors. For this purpose, we based our work on the simplest MNIST dataset and added factors such as Color, Angle, and Scale. Additionally, to measure the ability of the Energy Function to predict the adjustment of these factors, we modified the dataset to allow for continuous variations in each factor. Based on the existing MNIST dataset, the number of channels was increased to accommodate the color factor. Using the 'Kornia'[9] library, the data was augmented. This augmentation created sequences with 18 levels of variation in angle, color, and scale from the original data. The angle was augmented from -180 degrees to 180 degrees in 20-degree increments, color was augmented from 0 to 6, and scale was augmented from 0.1x to 1.8x. Experiments were conducted using these factors individually and in combination. These sequences were then used in experiments through the train loader. An example of this SeqMNIST dataset can be seen in Figure 2.

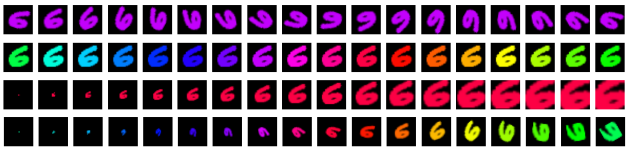

Figure 2. Example of SeqMNIST datasets. This dataset represents, from top to bottom, Angle, Color, Scale and Combine all.

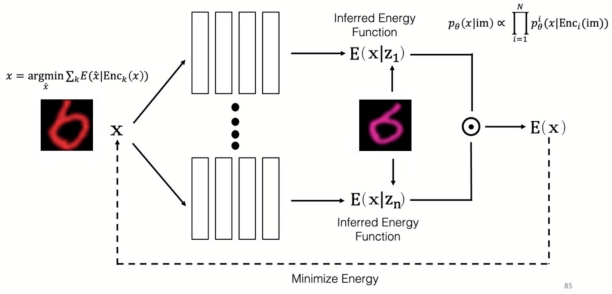

Figure 3. An image from [1]. This image illustrates the general framework of this project.

3x3 Conv2d
ResBlock
ResBlock
Linear
Linear → Latents

Table 1. The model architecture used for encoder.

3x3 Conv2d
AvgPool
ResBlock
ResBlock
ResBlock
Flatten
Linear → energy

Table 2. The model architecture used for energy network.

3.1.3 Model

Our model is inspired by COMET[4], in the way of using Energy-based models to represent underlying components and compose them to control generation.

The overall architecture is described in Figure Figure 3. Given the SeqMNIST dataset, our model aims to learn different energy functions that corresponds to underlying factors-color, scale and angle-of the dataset.

To illustrate, the input sequence is passed through the encoder to achieve latent z that represents core information of components. Then using latent z, we infer energy function with energy network that consists of residual blocks. The specific architecture of encoder and energy network is described in TableTable 1,Table 2. The whole network is trained to minimize energy.

3.1.4 Formulation

Inspired by the [4] paper, we can formulate the following training equation. The key difference from the paper is that our approach aims to predict future scenes. Currently, we have k energy functions E_{θ_i} , $i = 1, .., k$. Using these k energy functions, we will learn the underlying factors inherent in the SeqMNIST dataset.

First, from SeqMNIST dataset $\mathcal{D} = \{traj_1, ..., traj_N\}$, we sample a trajectory $traj \sim \mathcal{D}$. we sample $t \sim$ Unif$(0, T)$. Then we also sample 2 images x_t and x_{t+l}(practically set $l = 2$) from trajectory $traj$. The energy functions E_{θ_k} are trained to take x_t and latent embedding $z_i = \text{chunk}(Enc_\theta(x_t), k)[i]$, $i = 1, .., k$ as input and predict x_{t+l}. Specifically, x_t is used to create a latent variable z_k, which is then chunked into k parts and fed into each energy function E_{θ_k}(In practice, we use 0^{th} Energy model class's Embed network to predict $Z = \{z_1, ..., z_k\}$). The final model's prediction is obtained using energy functions, and it is trained by calculating the L2 loss with x_{t+l}. The objective function used in this project is :

$$\mathcal{L}(\theta) = \| \arg\min_x (\sum_k E_{\theta_k}(x; Enc_\theta(x_t))) - x_{t+l} \|^2 \quad (1)$$

In practice, evaluating $\arg\min_x(\sum_k E_{\theta_k}(x; Enc_\theta(x_t)))$ is computationally intractable. We thus approximate the $\arg\min$ operation with respect to x via N steps of gradient optimization, with an approximate optimum x_t^N obtained as :

$$x_t^N = x_t^{N-1} - \lambda \nabla_x \sum_k E_{\theta_k}(x_t^{N-1}; Enc_\theta(x_t)) \quad (2)$$

Additionally, due to the smoothness assumption, we expect the number of optimization iterations to be proportional to l(temporal distance). Therefore, l is set to be proportional to N, $l \propto N$. We are training the model so that by following the gradient of the implicitly learned multiple energy functions, it obtains the next time step image. We can also expect that the energy functions will autonomously learn the latent transformations which Datasets inherently possess(Color, Scale, Angle). In other words, we hope E_{θ_1} will extract Color transformation, E_{θ_2} will extract Scale transformation, and E_{θ_3} will extract Angle transformation.

3.1.5 Algorithm

Training Below is a general pseudo Training algorithm. There may be slight differences in experimental data, step size, and sampling frequency for each experiment we conducted.

Inference(Sampling) For inference, we input the starting point image x_t into the model and repeatedly perform the optimization steps that were used during training. By doing so, we can obtain the model's prediction. We experimented with various step sizes and optimization iterations, but the proportional relationship remains consistent which is $N \propto l$.

Training algorithm

$\mathcal{D} = \{\text{traj}_1, \text{traj}_2, \text{traj}_3, \ldots, \text{traj}_J\}$
$j \sim \text{Unif}(0, J)$
$x_t, x_{t+l} \sim \text{traj}_j$
$z_k = \text{Chunk}(Enc_\theta(x_t))[k]$
find $\arg\min$ with optimization
$x_t^N = x_t^{N-1} - \lambda \nabla_x \sum_k E_\theta(x_t^{N-1}; z_k)$
prediction
$\check{x}_{t+l} \leftarrow x_t^N$
optimize $\mathcal{L}$ w.r.t θ
$\Delta\theta \leftarrow \nabla_\theta \| \check{x}_{t+l} - x_{t+l} \|^2$
update θ based on $\Delta\theta$

3.2. Learning individual energy functions

First, we learn individual energy functions for underlying factors of the dataset. For this we separately train encoders to embed latent, and energy functions to output energy with respect to dataset that involves the component in question, by sequentially applying each transformation.

3.2.1 Setting

SeqMNIST dataset includes transformations of color, scale, and angle. As we want to learn individual energy functions corresponding to the specific underlying factor, we prepare dataset respectively for each factor.

We apply color, scale, and angle transformation separately, ending up with 3 sets of data $\mathcal{D}_{\textbf{Color}}$,$\mathcal{D}_{\textbf{Scale}}$,$\mathcal{D}_{\textbf{Angle}}$. This can be done simply by turning on the flags of desired transformation true and the rest false. For each dataset, We train encoder separately that output latent z from the given sequential input x_t.

Feeding latent z_1, for instance, obtained from input x_t of $\mathcal{D}_{\textbf{Color}}$ to corresponding energy function E_{θ_1}, yields energy value of the input. By training 3 energy networks, we can achieve 3 energy function E_{θ_1},E_{θ_2},E_{θ_3} associated with each of the underlying factors in a supervised manner.

3.2.2 Result

This work was conducted to verify if the energy functions can be learned effectively. Here, l was set to 1 and the number of optimization steps was set to 4. Thus, N was set to 4 for the experiments($N = 4 \cdot l$). Below is the result of learning individual energy functions. For each row, the left most image is the original input and each column is the result of

optimization step sampling using eqation that we have proposed in section 3.1.4 Formulation. The fifth column in the images represent the predicted value for the next frame.

Color transformation : E_{θ_1} Figure 4 shows sample results using color transformation data $\mathcal{D}_{\textbf{color}}$. The red box on the left is the input provided to the model, and the results sampled sequentially are shown on the right. It can be seen that color transformation works well, since it is a relatively easy task.

Figure 4. result for Color Dataset $\mathcal{D}_{\text{color}}$

Scale transformation : E_{θ_2} Figure 5 shows sample results using data with Scale transformations $\mathcal{D}_{\textbf{scale}}$. Similarly, the red box on the left is the input, and we can observe that for every sampling to the right, the image slightly scales up.

Figure 5. result for Scale Dataset $\mathcal{D}_{\text{scale}}$

Angle transformation : E_{θ_3} Figure 6 shows results using data with angle transformations $\mathcal{D}_{\textbf{angle}}$. We can observe the relative rotation of digit compared to the left most input, though some artifacts occur due to the difficulty of the task.

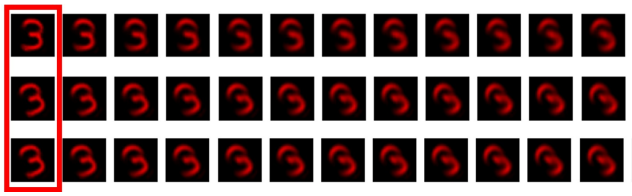

Figure 6. result for Angle Dataset $\mathcal{D}_{\text{angle}}$

3.3. Sharing embedding latent network to train each energy functions

In section 3.2, we trained each energy function on separate transformation datasets, learning each embedded latent and energy function end-to-end. However, to use the method from section 3.2, we needed to train three embedding latent networks and use all networks during sampling, which posed a bottleneck. Therefore, this time, we shared a single embedding latent network to obtain the embedding latent Z. Additionally, we used a pre-trained embedding latent model to train the energy functions.

3.3.1 Setting

The pre-trained model was taken from the embedding latent module of a model trained for 16 epochs in section 3.4. As will be explained in section 3.4, if our assumption is correct and each energy function learns a distinct latent transformation, then each embedded latent will contain useful representation such as each latent transformation factors which helps to predict the next future frame very well. Empirically, the Section 3.4 results showed that the three energy functions operated in a manner similar to division of labor, with each module determining each factor of what it would be the next frame image.

We trained all energy functions with freezed pre-trained share latent embedding model(in practice, we used 0^{th} Energy function class's embedding latent module to output the 3 latent variables(input x_t, pre-trained encoder's output Z which is $[z_1, z_2, z_3]$)). we feed x_t and z_k into each Energy function $E_{\theta_k}, k = 1, .., 3$ and get scalar energy value which we can use while sampling. We trained each energy functions with each latent transformation applied Dataset. In practice, we train E_{θ_1} to predict Color transformation, E_{θ_2} to predict Scale transformation and E_{θ_3} to predict Rotation transformation and trained with each generated SeqMNIST dataset $\mathcal{D}_{\text{color}}$,$\mathcal{D}_{\text{Scale}}$, $\mathcal{D}_{\text{Angle}}$. For efficiency, we also initialized pre-trained Energy functions that we obtained in section 3.4 to make learning easier. so, we can view this section's method pre-train and fine-tune framework. we expect each energy functions can be fine-tuned on each latent transformation specifically while utilizing pre-trained features.

3.3.2 Result

Below are the results of fine-tuning on color, scale, and angle data, respectively.

Color transformation : E_{θ_1} First, Figure 7 shows the fine-tuning results using color transformation data $\mathcal{D}_{\text{color}}$. The red box on the left is the input provided to the model, and the results sampled sequentially to the right are shown. Since color transformation is considered a relatively easy task, we fine-tuned it for a total of 1 epoch.

Scale transformation : E_{θ_2} Figure 8 shows the fine-tuning results using data with Scale transformations $\mathcal{D}_{\text{scale}}$.

Figure 7. result for fine-tuning on Color Dataset $\mathcal{D}_{\text{color}}$

Similarly, the red box on the left represents the input, and the sampled results are shown sequentially to the right. we fine-tuned it for a total of 2 epoch. Although the shape of the digits is not perfect, we could observe that the size of the digits relatively increases.

Figure 8. result for fine-tuning on Scale Dataset $\mathcal{D}_{\text{scale}}$

Angle transformation : E_{θ_3} Figure 9 shows the fine-tuning results using data with angle transformations $\mathcal{D}_{\text{angle}}$. We considered angle transformation to be the most challenging task, so we fine-tuned the model for a total of 8 epochs. The results were not ideal, as the model tended to eliminate thinner parts of the digits and only reconstruct larger segments. However, as seen in Figure 9, the model was able to recognize and sample based on angle transformations.

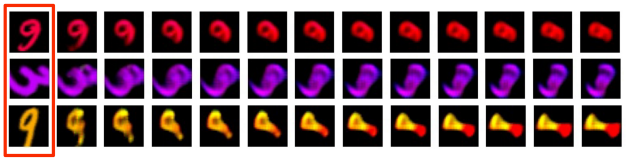

Figure 9. result for fine-tuning on Angle Dataset $\mathcal{D}_{\text{angle}}$

3.3.3 Control Generation

Next, we used the fine-tuned energy functions to control image generation. If the models are perfectly trained, we should be able to control the results as desired. For example, by inputting "For the initial steps, increase the size, and for the later steps, change the color." we would first sample using E_{scale} and then switch to sampling using E_{color}, thereby achieving the desired outcome.

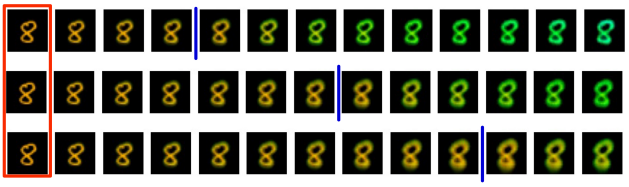

Figure 10. Result image for Section 3.4.3. controlling generation using E_{θ_1} and E_{θ_2}

'First, scale up for a while, and then change the color for the remaining steps.' Figure 10 shows the result images of scaling the size for a few steps and then changing the color. The leftmost column is the input image. From top to bottom, the sequences are as follows: 3 steps Scaling and 9 steps Coloring, 6 steps Scaling and 6 steps Coloring, and 9 steps Scaling and 3 steps Coloring. In other words, the initial steps were sampled using E_{θ_2}, and the remaining steps were sampled using E_{θ_1}.

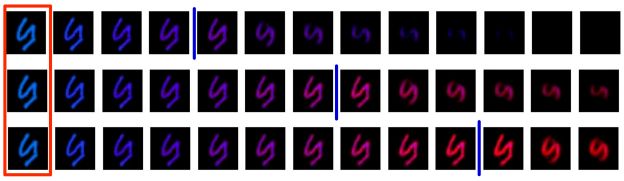

Figure 11. Result image for Section 3.4.3. controlling generation using E_{θ_1} and E_{θ_3}

'First, change the color for a while, and then change the angle for the remaining steps.' Figure 11 shows the result images of changing the color for a few steps and then changing the angle. The leftmost column is the input image. From top to bottom, the sequences are as follows: 3 steps Coloring and 9 steps Rotating, 6 steps Coloring and 6 steps Rotating, and 9 steps Coloring and 3 steps Rotating. In other words, the initial steps were sampled using E_{θ_1}, and the remaining steps were sampled using E_{θ_3}.

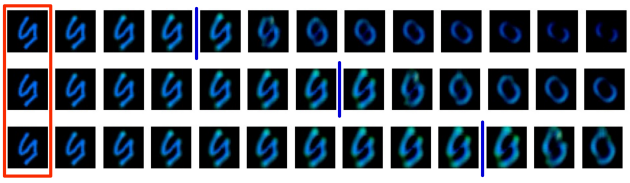

Figure 12. Result image for Section 3.4.3. controlling generation using E_{θ_2} and E_{θ_3}

'First, scale up for a while, and then change the angle for the remaining steps.' Figure 12 shows the result images

of scaling the size for a few steps and then changing the angle. The leftmost column is the input image. From top to bottom, the sequences are as follows: 3 steps Scaling and 9 steps Rotating, 6 steps Scaling and 6 steps Rotating, and 9 steps Scaling and 3 steps Rotating. In other words, the initial steps were sampled using E_{θ_2}, and the remaining steps were sampled using E_{θ_3}.

Although the results are not perfect, the significance lies in the ability to generate various combinations using the learned energy functions.

3.4. Unsupervised Factor decomposition

Initially, our goal was to develop a method to autonomously extract the transformations inherent in an image sequence by only observing the sequence of images. This approach is similar to the Variational Auto Encoder (VAE)[7] in that it aims to learn the latent variables through the structure of an encoder and decoder.

3.4.1 Setting

We previously constructed the SeqMNIST Dataset and augmented MNIST dataset using three types of transformations(Color, Scale, Angle). Our hypothesis is that by learning through three energy functions, each energy function will independently learn each of the transitions, allowing us to use each as a module.

Therefore, unlike the previous two experiments, this experiment block is purely unsupervised. If this method is scalable and learns well, we can expect to extract orthogonal transitions present in high-dimensional videos through pre-specified energy functions and this can be also viewed as some actions or actions in RL sense.

This block directly uses the equations employed in section 3.1.3 above. The dataset is a 3-latent dataset with all transformations applied. And we use 3 energy models and hope for each Energy model to extract latent transition. So we set $k = 3$ and each latent size is 1 ($z_i \in \mathbb{R}^1$), the predict frame distance l is set to 2, and the optimizing step size N is set to 4. So that, the model is trained to predict 1 step ahead frame with 2 optimization steps($N = 2 \cdot l$). We trained all energy functions with share latent embedding model(in practice, we used 0^{th} Energy function class module to output the 3 latent variabls(input x_t, output Z which is $[z_1, z_2, z_3]$)). we feed x_t and z_k into each Energy function E_{θ_k} , $k = 1, .., 3$ and get scalar energy value which we can use while sampling.

3.4.2 Result

In this section, we present the results of sampling using all three energy functions all at once and sequentially show the values sampled by each energy function to verify if the model has successfully extracted the latent factors or not.

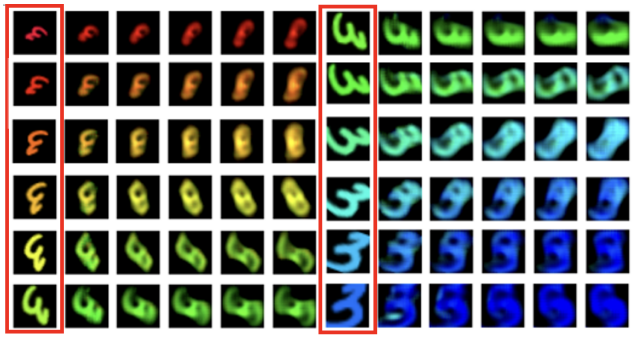

Figure 13. Result image for Section 3.4. samples using all energy function

Sample using all Energy functions at once Figure 13 is the sampled images using validation set data. The red-boxed area represents the input x_t fed into the model E_Θ, where $\Theta = [\theta_1, \theta_2, \theta_3]$. The results of moving a total of 5 steps to the right are displayed sequentially. In this experiment, the model was trained to predict the image two steps ahead. so, for example, third column from the left is the next frame prediction $\hat{x}_{t+1}$ from the model. Although the images appear slightly blurry, it appears that the model is learning to some extent.

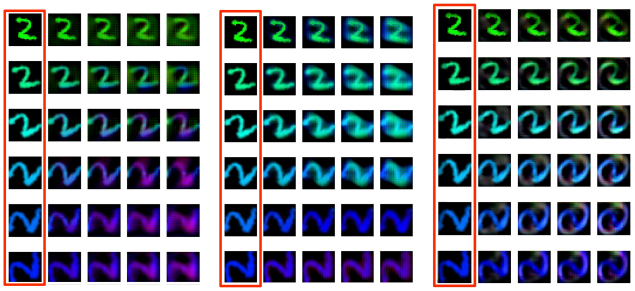

Figure 14. Result samples image using energy function E_1

Figure 15. Result samples image using energy function E_2

Figure 16. Result samples image using energy function E_3

Sample using each Energy functions The generated images are not perfect. But we can observe that Figure 14 relatively learns information related to color, Figure 15 learns information related to scale, and Figure 16 learns information related to angle which we have expected before.

Similarly, for each Figures 14 to 16, the image within the red box is the input x_t, and the results of a total of four sampling steps are shown sequentially to the right. The 2-step optimization result is the predicted image of the next frame.

The results above were all obtained through unsupervised learning manner. We did not add additional regularization to ensure that each latent encoder operates indepen-

dently in the model.

Our initial goal was to unsupervisedly learn the factors and then recombine them to achieve controllable generation as we done at Section 3.3. However, the Section 3.4 results did not meet our expectations to the extent that we could achieve controllable generation. Therefore, these generation experiments are concluded satisfactorily in section 3.3.

4. Conclusion

We conducted a project to implement compositional generation based on energy-based models. A key advantage of energy-based functions is the ability to separate training and inference, and we utilized this benefit. Although the results were not perfect, the project aimed to verify the initial hypothesis we had set as our goal.

5. Limitation and Future Works

The limitations of this work are that the training is very slow and unstable. To improve training stability, we followed previous papers and truncated the gradient of the N-step optimization. Additionally, we have doubts about the scalability of this method. We could consider using ideas from R-CNN[TODO] or Latent Diffusion[TODO] to leverage feature encoder. Another limitation is that the energy decomposition did not produce perfect results. Adding inhibitory connections between the latent variables to ensure orthogonality between them could be another approach to try. And also to solve the blurry result issues we can try some new loss functions(we tried L1 loss for experiments but the result was same) such as adding additional energy function that implicitly contain some "realistic view" constraint so that it can penalize(high energy) the unrealistic images and descent to the realistic images valley. Lastly, a major limitation of this project is that we did not compare our method with other baselines. Although we conducted qualitative evaluations, the lack of quantitative evaluations is the most significant flaw.

References

[1] Yilun Du. Yilun du - implicit learning with energy-based models — nuro technical talks, 2023. Accessed: 2024-06-23. 4

[2] Yilun Du and Leslie Kaelbling. Compositional generative modeling: A single model is not all you need, 2024. 1, 2

[3] Yilun Du, Shuang Li, and Igor Mordatch. Compositional visual generation and inference with energy based models, 2020. 4

[4] Yilun Du, Shuang Li, Yash Sharma, B. Joshua Tenenbaum, and Igor Mordatch. Unsupervised learning of compositional energy concepts. In *Advances in Neural Information Processing Systems*, 2021. 1, 2, 4, 5

[5] Yilun Du, Shuang Li, Joshua Tenenbaum, and Igor Mordatch. Learning iterative reasoning through energy minimization. In *International Conference on Machine Learning*, pages 5570–5582. PMLR, 2022. 4

[6] Ajay Jain Jonathan Ho and Pieter Abbeel. Denoising diffusion probabilistic models. 2020. 3

[7] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022. 2, 8

[8] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and Fujie Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006. 1, 3

[9] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3674–3683, 2020. 4

[10] Nicu Sebe Max Welling Yue Song, T.Anderson Keller. Flow factorized representation learning, 2023. 3