Interpretable Style Swapping with a Symmetric Autonencoder

Rui Sun
Mentor: Borjan Geshkovski
Project suggested by: Laurent Demanet

Abstract

We investigate an encoder-decoder architecture proposed by Bharadwaj et al. (2022), which empirically disentangles two distinct but unknown factors of variation within a dataset. Dubbed *SymAE*, this auto-encoder proceeds by sequentially being fed datapoints corresponding to a unique signal and various styles into a couple of encoders: one which is invariant (*symmetric*) to permutations of the input's orderings to embed the signal, and another which scrambles all inputs by randomizing to embed the style, all with the goal of obtaining two latent spaces which are essentially orthogonal. This then allows for generating virtual data in a controllable way by signal and style swapping in latent space and decoding, with various applications in the applied sciences (therein called *redatuming*). Our goal in this paper is to describe the structure of the learned latent space and to investigate when disentanglement is possible. To this end, we focus on a task motivated by *multi-reference alignment* – a toy model for signals corrupted by styles given by group actions (here, the cyclic group). We demonstrate that SymAE's second encoder precisely learns the group action by plotting its latent space, which appears homomorphic to a torus. We also consider penalties that enhance the disentanglement of latent codes.

1. Introduction

Unknown and un-modeled nuisance variations are ubiquitous in experimental designs in the applied sciences. For example, in geophysics, specifically passive time-lapse seismic monitoring, the recorded seismic data are generated by uncontrollable sources related to tectonic stress changes in the subsurface (Aki and Richards (2002)) or ocean-wave activity (Montagner et al. (2020)). Similarly, in astronomy, the fluorescent emissions which characterize the lunar surface's chemical composition fluctuate depending on un-modeled solar flares (Tandberg-Hanssen and Emslie (1988); Narendranath et al. (2011)). Such designs introduce ambiguity when determining whether changes in data represent coherent information (a signal indicating changes related to the underlying physical state) or conversely represent nuisance information (incidental variations due to noise inherent to the data acquisition). Nevertheless, these uncontrollable experiments remain the only feasible avenue to measure and study certain physical phenomena. This motivates the development of algorithmic tools to reliably disentangle the nuisance noise from coherent signals in these settings.

A recent development in this direction is the encoder-decoder architecture proposed by Bharadwaj et al. (2022), which empirically disentangles coherent and nuisance information in the particular instance of passive time-lapse seismic monitoring. Datapoints here are shot records over multiple timeframes, which read as

$$u_{ij}(t, x_r) = \int_0^T G_i(\tau, x_r) w_j(t - \tau) d\tau, \tag{1}$$

for $(i,j) \in [n_1] \times [n_2]$ and $t \in (0,T)$, $x_r \in \mathbb{R}$. Here $G_i(\tau, x_r) = G(m_i; \tau, x_r, x_s)$ is the Green's function for the acoustic wave equation

$$\begin{cases} m_i(x)\partial_t^2 u(t,x) - \Delta u(t,x) = w_j(t)\delta_{x_s}(x) & \text{in } (0,T) \times \mathbb{R} \times \mathbb{R}_-, \\ (u, \partial_t u)_{|t=0} = (0,0), \end{cases} \tag{2}$$

where $m_i(x) = 1/v_i^2(x)$ and $v_i(x)$ is the wave-speed. In this example, the different sources w_j represent nuisance information, while the media $m_i(x)$ are the coherent information, which are sought after in seismic imaging.

This auto-encoder proceeds by sequentially being fed datapoints corresponding to a unique signal and various styles into a couple of encoders: one which is invariant (*symmetric*) to permutations of the input's orderings to embed the signal, and another which scrambles all inputs by randomizing to embed the style, all with the goal of obtaining two latent spaces.

Our main interest is to understand what lies under the hood of SymAE. SymAE is task-independent and can be applied to models beyond seismic monitoring, such as MNIST. Therefore, the latent space varies based on the task, and the decoder can learn a meaningful approximation of the actual operation. A direct application is to perform actual interpolation rather than just style swapping by drawing geodesics inspired by the latent space's geometry. The decoder performs well, allowing us to achieve continuous interpolation in the latent space and decode it to obtain interpolation between data points.

1.1. Related work

Independent component analysis (ICA) is a computational method for separating a multivariate signal in signal processing (Hyvärinen and Pajunen (1999)). ICA can be formulated as the estimation of the following generative model for the data:

$$x = f(s),$$

where $x \in \mathbb{R}^n$ is the vector of observed random variables and $s \in \mathbb{R}^n$ is the vector of latent random variables. The fundamental assumption in ICA is that the n components of s are mutually independent, and the goal of ICA is to estimate the realizations of the independent components of s together with the mixing function f only using observations $\{x^{(i)}\}_{i=1}^N$. The coherent states and the nuisances in SymAE can be viewed as two independent components of the s in ICA. Both SymAE and ICA seek to disentangle independent latent variables from observations.

Beyond ICA, β-VAE (Higgins et al. (2017)) is a type of variational autoencoder that seeks disentangled latent factors. Compared with β-VAE, SymAE is not a variational method, and we do not presume the distribution of the latent code, which is often set to be Gaussian distribution in β-VAE.

2. The SymAE architecture

We will work with the dataset

$$\{\boldsymbol{X}_i\}_{i=1}^{n_x},$$

which consists of n_x datapoints $\boldsymbol{X}_i \in \mathbb{R}^{n_\tau \times k}$. Each datapoint $\boldsymbol{X}_i$ contains n_τ instances, each of dimension k, repeatedly capturing the same physical state with different nuisances. The j-th instance of X_i is denoted as $\boldsymbol{X}_{i,j} \in \mathbb{R}^k$.

The autoencoder contains a couple of networks: an encoder and a decoder. We shall feed data sequentially by block columns of the matrix $\boldsymbol{X}$, namely we will minimize a loss of the form

$$\min_{Enc,Dec} \sum_{j=1}^{n_x} \|\boldsymbol{X}_j - Dec(Enc(\boldsymbol{X}_j))\|_2^2 \tag{3}$$

using gradient decent, where Enc and Dec are neural networks.

2.1. The encoder

The encoder Enc of SymAE contains two parts: a so-called *coherent encoder* $CEnc$, and a so-called *nuisance encoder* $NEnc$. The encoder $Enc : \mathbb{R}^{n_\tau \times k} \to \mathbb{R}^{p+n_\tau q}$ can be described as

$$Enc(\boldsymbol{X_i}) = \begin{bmatrix} CEnc(\boldsymbol{X}_i) \\ NEnc(\boldsymbol{X_{i,1}}) \\ \vdots \\ NEnc(\boldsymbol{X_{i,n_\tau}}) \end{bmatrix}$$

where

$$CEnc : \mathbb{R}^{n_\tau \times k} \to \mathbb{R}^p$$

and

$$NEnc : \mathbb{R}^k \to \mathbb{R}^q$$

are neural networks. The integers p and q are both fixed, controlling the size of the latent space. It is straightforward to see the dimension of the latent space is $p + n_\tau q$.

The coherent encoder $CEnc$ is built as follows:

$$CEnc(\boldsymbol{X_i}) = CEnc_2\left(\frac{1}{n_\tau}\sum_{j=1}^{n_\tau} CEnc_1(\boldsymbol{X_{i,j}})\right)$$

where $CEnc_1$ and $CEnc_2$ are both neural networks. The architecture of the coherent encoder ensures that the output of the coherent encoder is permutation-invariant under any permutation $\pi \in S_{n_\tau}$ of n_τ instances, where S_{n_τ} is the symmetric group of $\{1, 2, ..., n_\tau\}$:

$$CEnc\left(\begin{bmatrix} \boldsymbol{X_{i,1}} \\ \vdots \\ \boldsymbol{X_{i,n_\tau}} \end{bmatrix}\right) = CEnc\left(\begin{bmatrix} \boldsymbol{X_{i,\pi(1)}} \\ \vdots \\ \boldsymbol{X_{i,\pi(n_\tau)}} \end{bmatrix}\right).$$

This special structure encourages the latent coherent code

$$C_i = CEnc(\boldsymbol{X_i})$$

to be irrelevant to nuisances and to only depend on the state of $\boldsymbol{X_i}$ [1].

1. The key point of the symmetric encoder is not the sum, but rather its property of being permutation-invariant. Other permutation-invariant functions, such as max or min, are also acceptable.

2.2. The dropout

The so-called *stochastic regularization* [2] is implemented after the nuisance encoder $NEnc$. We denote

$$N_{i,j} = NEnc(\boldsymbol{X}_{i,j}). \tag{4}$$

A random noise is then added to the output of $NEnc$ by a layer

$$SR : \mathbb{R}^q \to \mathbb{R}^q$$

during the training procedure. TensorFlow provides several types of stochastic regularization [3]. We mainly consider three types: GaussianNoise, Droupout, and GaussianDropout (Srivastava et al. (2014)). They are defined as follows:

1. GaussianNoise.
$$[SR(x)]_i = x_i + \epsilon_i,\ \epsilon_i \sim \mathcal{N}(0, r^2)\ i.i.d.$$
2. Dropout.
$$[SR(x)]_i = x_i * \epsilon_i,\ \epsilon_i \sim Bernoulli(1-r)\ i.i.d.$$
3. GaussianDropout.
$$[SR(x)]_i = x_i * \epsilon_i,\ \epsilon_i \sim \mathcal{N}(1, \frac{r}{1-r})\ i.i.d.$$

where $r > 0$ is a hyper-parameter named dropout rate. GaussianNoise is an additive noise, while both Dropout and GaussianDropout are multiplicative noises. The dropout is only activated during the training procedure. Once we finish training the neural networks, the SR is fixed to be identity, namely $SR(x) = x,\ \forall x \in \mathbb{R}^q$.

2.3. The decoder

The decoder $Dec : \mathbb{R}^{p+n_\tau q} \to \mathbb{R}^{n_\tau k}$ is responsible for reconstructing the input $\boldsymbol{X}_i$ from the encoded information, which is constructed by repeating a smaller neural network $dec : \mathbb{R}^{p+q} \to \mathbb{R}^k$ n_τ-times:

$$Dec\left(\begin{bmatrix} C_i \\ SR(N_{i,1}) \\ \vdots \\ SR(N_{i,n_\tau}) \end{bmatrix}\right) = \begin{bmatrix} dec(C_i, SR(N_{i,1})) \\ \vdots \\ dec(C_i, SR(N_{i,n_\tau})) \end{bmatrix}.$$

We refer to $\hat{X}_i$ as the reconstruction result

$$\hat{X}_i = \begin{bmatrix} dec(C_i, SR(N_{i,1})) \\ \vdots \\ dec(C_i, SR(N_{i,n_\tau})) \end{bmatrix}.$$

Then the loss function can be written as

$$L_{rec}(Enc, Dec) = \frac{1}{n_x} \sum_{i=1}^{n_x} ||\hat{\boldsymbol{X}}_i - \boldsymbol{X}_i||_2^2 \tag{5}$$

2. The goal of using a stochastic regularization layer is not for regularization, but for removing information.
3. The documentation of different layers can be found at https://www.tensorflow.org/api_docs/python/tf/keras/layers.

2.4. Redatuming

We hope that a trained SymAE learns a representation with disentangled coherent and nuisance code in the latent space. This will allow us to generate virtual instances that have not been seen by the autoencoder during training. This procedure is called *redatumining* (Bharadwaj et al. (2022)). Redatumed virtual data $\hat{X}_{i\to i',j}$ generated by swapping latent codes following

$$\hat{X}_{i\to i',j} = dec\big(C_{i'}, N_{i,j}\big).$$

We perform redatuming after training, so there is no dropout SR in the definition of redatuming. We denote the ground truth of $\hat{\boldsymbol{X}}_{i\to i',j}$ as $\boldsymbol{X}_{i\to i',j}$, which is generated using the state from $\boldsymbol{X}_{i'}$ and the nuisance from $\boldsymbol{X}_{i,j}$. The goal of SymAE is to achieve

$$\hat{\boldsymbol{X}}_{i\to i',j} = \boldsymbol{X}_{i\to i',j},\ \forall i, i' \in [n_x], j \in [n_\tau]. \tag{6}$$

If Equation (6) is achieved, we could say SymAE successfully disentangles coherent information and nuisance information.

Remark 1 *The Equation* (6) *is equivalent to*

$$dec(C_i, N_{i,j}) = dec(C_{i'}, N_{i,j}),\ \forall i, i' \in [n_x], j \in [n_\tau]$$

3. Numerical results

3.1. (Idealized) multi-reference alignment

Consider a slightly idealized version of Multi-reference alignment (MRA). MRA is a toy model of three-dimensional molecule reconstruction in Cryo-Electron Microscopy (cyro-EM) (Perry et al. (2019)). It is one of the simplest models of nuisant group actions. Group actions are a good benchmark for latent space interpretability, since we should be able to recover the group in latent space, in some shape or form.

In MRA, the coherent states are vectors $a_1, a_2, ..., a_{n_\varepsilon} \in \mathbb{R}^k$, and the nuisances are the cyclic shifts $\{R_\ell\}_{\ell=1}^k$. Here, R_ℓ is a cyclic shift by an number ℓ of coordinates, defined as

$$\begin{gathered} R_\ell : \mathbb{R}^k \to \mathbb{R}^k \\ (x_1, x_2, ..., x_k) \mapsto (x_{1+\ell}, x_{2+\ell}, ..., x_{k+\ell}), \end{gathered}$$

where a subscript is viewed as its modulus to k if it exceeds k. In the setting of SymAE, the dataset is sampled from $\{R_\ell a_c : c \in [n_\epsilon], l \in [k]\}$ following Algorithm 1.

In our experiments, we fix the length of the signals to be $k = 100$ and the number of states to be $n_\varepsilon = 6$. We set $p = 2$, $q = 10$, $n_x = 18$, $n_\tau = 30$, and use the Dropout layer for SR. The six states are defined using the following functions:

$$\begin{aligned} &\phi_1(x) = \mathbb{1}\{x > 0.5\}, && \phi_2(x) = 0.5 + 0.5\cos(2\pi x), && \phi_3(x) = (1-2x)\mathbb{1}\{x < 0.5\} \\ &\phi_4(x) = 2\phi_1(x) - 1, && \phi_5(x) = 2\phi_2(x) - 1, && \phi_6(x) = 2\phi_3(x) - 1. \end{aligned}$$

The $n_\varepsilon = 6$ states $a_i \in \mathbb{R}^k$ are defined as:

$$[a_i]_j = \phi_i(\frac{j}{k}), \quad \forall i \in [n_\varepsilon], j \in [k]. \tag{7}$$

Algorithm 1 Generate dataset $\boldsymbol{X}$

```
Input  : Integers $n_x$, $n_\tau$, $k$, $n_\varepsilon$, vectors $\{a_i\}_{i=1}^{n_\varepsilon}$.
Output : Tensor $\boldsymbol{X} \in \mathbb{R}^{n_x \times n_\tau \times k}$.
for $i \leftarrow [n_x]$ do
    c $\leftarrow$ a random integer sampled uniformly from 1 to $n_\varepsilon$.
    for $j \leftarrow [n_\tau]$ do
        $\ell \leftarrow$ a random integer sampled uniformly from 1 to $k$, without replacement.
        $X[i, j, :] \leftarrow R_\ell(a_c)$
    end
end
return $\boldsymbol{X}$
```

The redatuming result is displayed in 6×10 subplots. The subplot located at (i, j) has the i-th state and the shift of $10j$, where $i \in [6]$ and $j \in [10]$. Each subplot exhibits three lines: the green line represents the real instance $R_{10j}(a_i)$; the yellow line represents the reconstruction instance, i.e. the output of the autoencoder; the blue line represents the redatuming instance. Redatuming instance is generated by mixing the latent coherent code from the i-th state, with an arbitrary shift, and the latent nuisance code from the $\varphi(i)$-th state, with a shift of $10j$, where

$$\varphi : 1 \mapsto 2 \mapsto 3 \mapsto 1,\ 4 \mapsto 6 \mapsto 5 \mapsto 4.$$

Remark 2 (What is a successful redatuming in MRA)

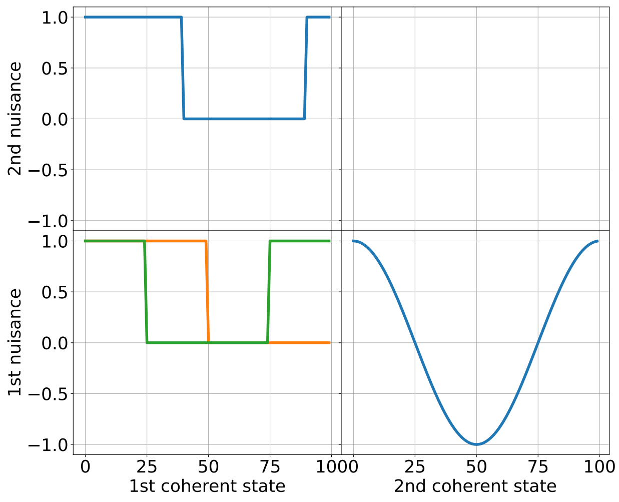

Figure 1: Given the square wave and the cosine wave depicted as blue lines, both the orange and green lines are valid redatuming results. See Remark 2.

For example, Figure 2 shows a relatively successful redatuming. It is important to note that in the particular setting of MRA, achieving $\hat{X}_{i\to i',j} = X_{i\to i',j}$ is meaningless.

To better understand this, let's imagine asking humans instead of SymAE to perform redatuming. We give a human two signals, for example, the square wave and the cosine wave depicted as blue lines in Figure 1. Then we ask this human to generate a new signal with the same state as the square wave and the same shift as the cosine wave. There is no way to know the "real shift" of the cosine wave. It could have a nuisance shift of 0, in which case the state would be $\phi(x) = cos(2\pi x)$. Alternatively, it could have a nuisance shift of 25, in which case the state would be $\phi(x) = cos(2\pi x + \frac{\pi}{2})$.

Suppose we received two different answers from two different humans, represented by the orange and green lines in Figure 1. Both of these answers are acceptable for redatuming purpose. Instead, we should allow a shift between the redatuming result and the ground truth.

The aforementioned issue also presents in similar group action problems, including cryo-EM.

We present some numerical results below.

Example 1 *We set parameters $p = 5, q = 10, r = 0.5, n_x = 18, n_\tau = 30$. The redatuming result is shown in Figure 2.*

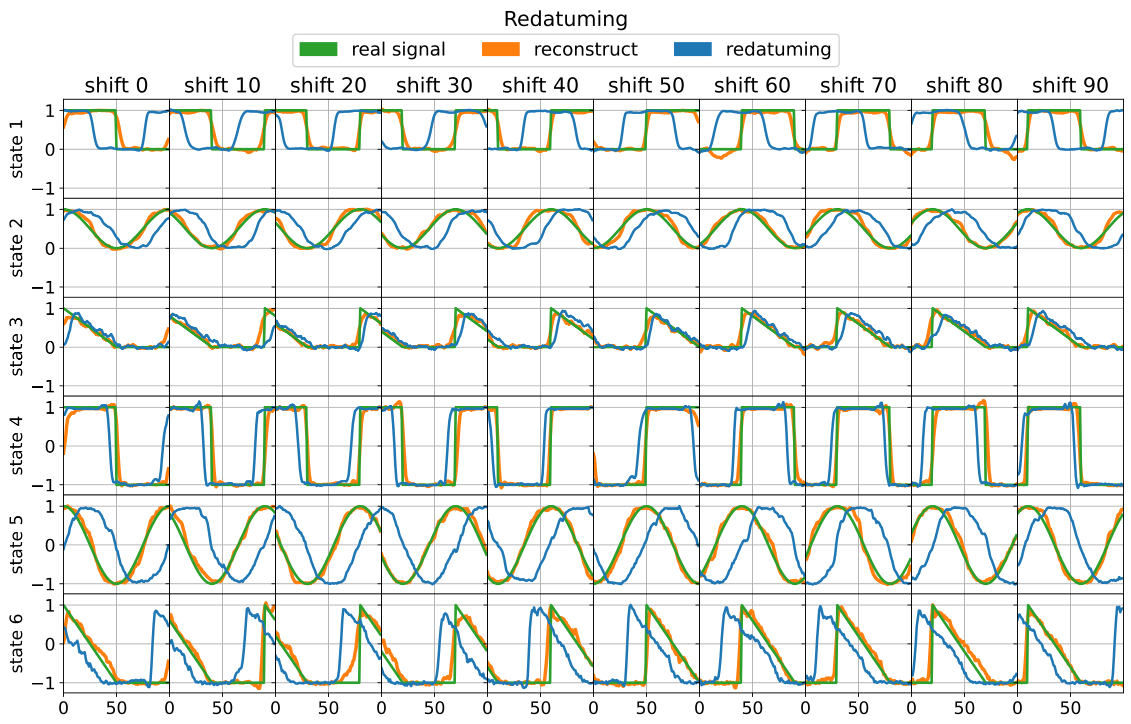

Figure 2: Redatuming of Example 1. Hyperparameters are p=5, q=10, n_x=18, n_τ=30. We use Dropout for the dropout layer, with a dropout rate $r = 0.5$. The blue, green, and yellow lines have the same shape, and the shift between the blue and green lines remains the same, so redatuming is successful (see Remark 2).

3.2. Swapping content and style of Images

The SymAE architecture can be used to swap styles and contents among images. We present the MNIST dataset as an example. Here, the coherent state of an image is its content, which is a digit in $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$; the nuisance of an image is its handwriting style.

Given the MNIST dataset, we are able to generate new images using redatuming. Some redatuming results are shown in Figure 3.

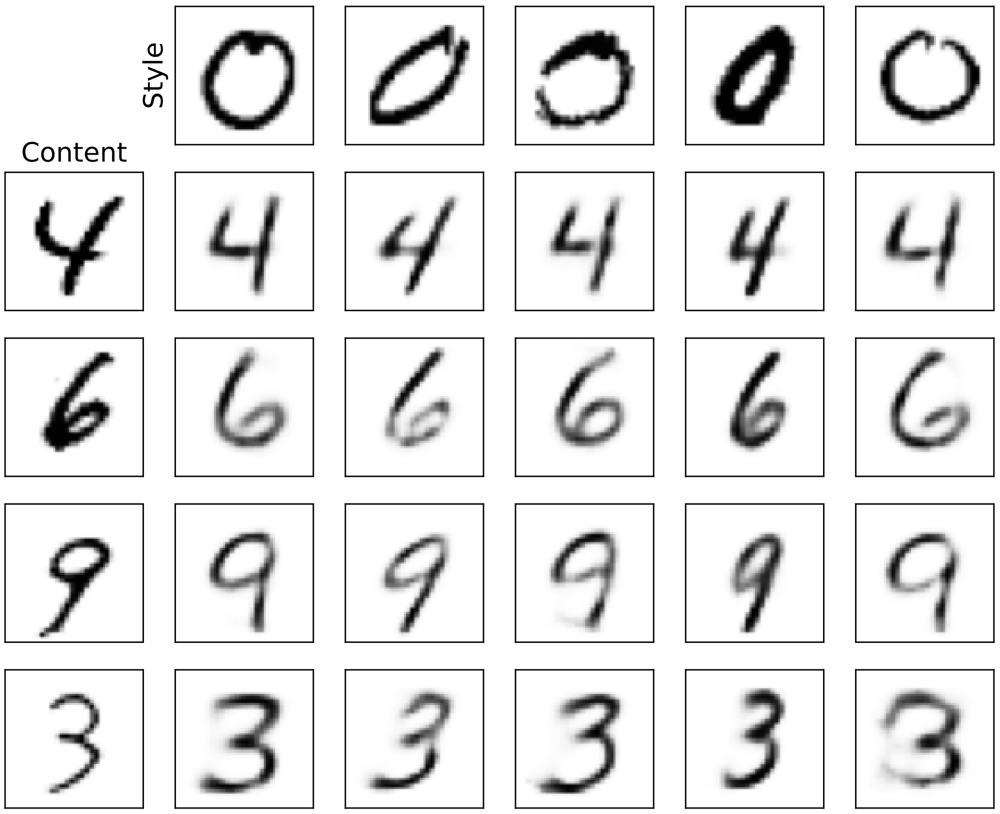

Figure 3: There are 20 redatuming images displayed in this figure. Each redatuming image is generated by mixing the coherent code from the left-most image (with content 4, 6, 9, and 3), with the nuisance code from the top-most image (with content 0). The images in each column exhibit similar writing styles (note that in the MNIST task, there is no ground truth for redatuming).

4. Discussion

Figure 4 show a typical distribution of latent coherent codes C_i and latent nuisance codes $N_{i,j}$, using principal component analysis (PCA). From Figure 4 (left), we observe that the symmetric encoder works well, and the points are well clustered according to their states. So phenomenologically, we could assume that the real state uniquely determines the latent coherent code. SymAE might not

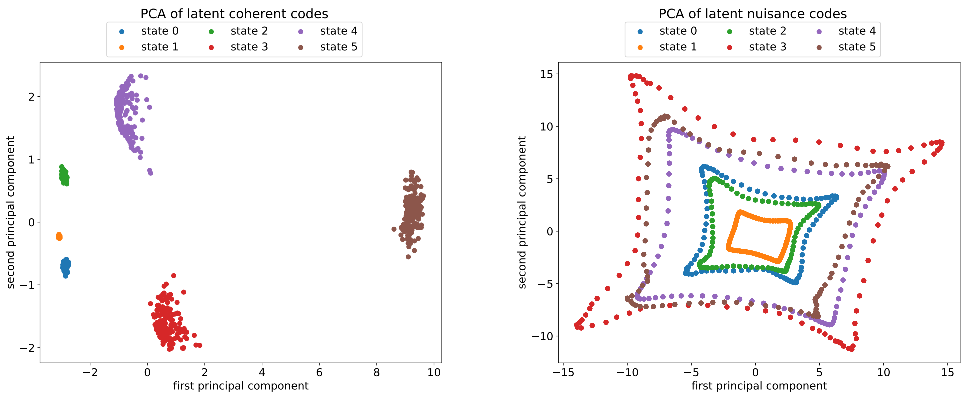

Figure 4: PCA of a typical distribution of latent coherent and nuisance codes. We use the same parameters as Example 1

perform well because the dropout layer cannot remove all coherent information from the nuisance code.

4.1. Interpolation in latent space

By decoding latent codes obtained by interpolating between datapoints, the autoencoder can produce an output that semantically mixes characteristics from different instances (Berthelot et al. (2018)). For the trained SymAE in Example 1, we draw a continuous line by interpolating the latent nuisance code of a datapoint using a cubic spline. Then, we generated some outputs by decoding virtual points that are on the line but were not seen in the training set. The result is shown in Figure 5.

4.2. Energy distance as a penalty

Although Example 1 achieves relatively good redatuming results, the performance of redatuming remains uncontrollable and actually requires fine-tuning of the neural networks. Some non-perfect examples are shown in Appendix B. In this subsection, we attempt an explicit way to encourage disentanglement.

4.2.1. MOTIVATION OF USING THE ENERGY DIVERGENCE

To further improve SymAE, we hope to add a regularizer on the nuisance codes C_i and $N_{i,j}$, to penalize how strong the nuisance codes depend on their states. We hope to construct a ”distance” function $D(\cdot,\cdot)$ between random variables, such that $D(X,Y) = 0$ if and only if X and Y are independent. There are many ways to measure whether two random variables are independent. A possible way is to first build up a divergence between two probability distributions. There are many ways to measure the difference between two probability distributions, such as KL-divergence and Wasserstein distance.

Why not Kullback–Leibler divergence (KL divergence). It is not possible to naturally approximate the Kullback–Leibler divergence from discrete samples. Indeed, the quantity is almost always

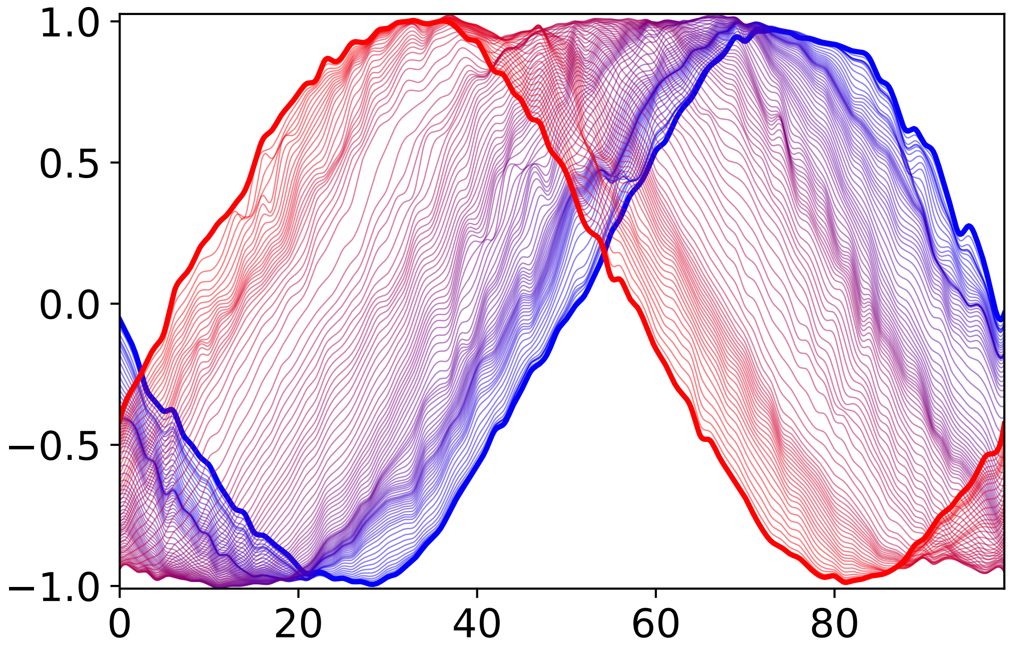

Figure 5: Interpolation in latent space, see Section 4.1. Using a cubic spline, we interpolate the latent nuisance codes of the n_τ instances of a specific datapoint (in this example, a sine wave with state 5). We then generate some outputs by decoding virtual points that lie on the interpolated line but were not seen in the training set. The color, varying from blue to red, represents the position of an output's latent nuisance code on the interpolated line.

$+\infty$. Suppose we have two empirical density functions

$$p = \frac{1}{N}\sum_{i=1}^{N}\delta_{x_i}, \qquad q = \frac{1}{N}\sum_{i=1}^{N}\delta_{y_i}. \tag{8}$$

We hope the KL divergence $D_{KL}(p||q)$ would measure the difference between p and q. Let $\mathcal{X} = \{x_i\}$ and $\mathcal{Y} = \{y_i\}$. Then, formally,

$$\begin{aligned} D_{KL}(p||q) &= \int_{\mathbb{R}} \log\left(\frac{p(x)}{q(x)}\right) p(x)dx \\ &= \frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{p(x_i)}{q(x_i)}\right) \\ &= \frac{1}{N}\sum_{i=1}^{N}\log\left(\frac{\delta(0)}{N\cdot q(x_i)}\right) \end{aligned}$$

The first problem is the $\delta(0)$. As the numerator, the $\delta(0)$ is not even a real number. Informally, let's temporarily think of $\delta(0)$ as a fixed large constant. Evaluate the denominator, we have

$$q(x_i) = \begin{cases} \frac{1}{N}\delta(0), & \text{if } x_i \in \mathcal{Y} \\ 0, & \text{if } x_i \notin \mathcal{Y} \end{cases}$$

Thus, informally

$$\begin{aligned} D_{KL}(p||q) &= \frac{1}{N}\left[|\mathcal{X} \cap \mathcal{Y}| \cdot \log\left(\frac{\delta(0)}{\delta(0)}\right) + |\mathcal{X}\backslash\mathcal{Y}| \cdot \log\left(\frac{\delta(0)}{0}\right)\right] \\ &= \frac{1}{N} \cdot |\mathcal{X}\backslash\mathcal{Y}| \cdot (+\infty) \end{aligned}$$

We see that $D_{KL}(p||q)$ does measure the difference between $\mathcal{X}$ and $\mathcal{Y}$ in some sense. But in fact $D_{KL}(p||q)$ is always infinity unless $|\mathcal{X}\backslash\mathcal{Y}| = 0$. Another essential problem is that the gradient of $D_{KL}(p||q)$ is always zero, making it impossible to use gradient descent to update x_i and y_i.

Instead, to use the KL divergence, it is required to use a density estimator to somehow smooth the discrete empirical measures and replace them with densities by using a kernel density estimation (KDE) (Peyré and Cuturi (2019)).

Suppose we still have the same empirical density functions as defined in 8. The Gaussian kernel is defined as

$$\varphi_\varepsilon(x) = \frac{1}{\sqrt{2\pi\varepsilon}} e^{-\frac{x^2}{2\varepsilon^2}}$$

The density estimator for p and q is defined using a convolution against this kernel

$$p(x) = \frac{1}{N}\sum_{i=1}^{N} \varphi_\varepsilon(x - x_i), \qquad q(y) = \frac{1}{N}\sum_{i=1}^{N} \varphi_\varepsilon(y - y_i)$$

where ε should be adapted to the number N of samples and to the dimension d. Then the KL divergence $D_{KL}(p||q)$ becomes well-defined.

However, even though we apply the KDE on the empirical discrete distribution, the KL-divergence between p and q still has no explicit expression. Evaluating the KL-divergence requires computing the integral in $\mathbb{R}^n$. An efficient approach could be Monte-Carlo integration, but it would still remain an approximate approach. All in all, it is not natural to use the KL divergence on a discrete distribution.

Why not Wasserstein distance. The Wasserstein metric originates from optimal transport. Suppose $M = \mathbb{R}^n$ is the Euclidean space while $d(\cdot,\cdot)$ denotes the Euclidean distance on M. The Wasserstein distance between two probability measures α and β on M is defined as

$$\mathcal{W}_p(\alpha,\beta) = \inf_{\gamma\in\Gamma(\alpha,\beta)} \left(\int_{M\times M} d(x,y)^p d\gamma(x,y)\right)^{\frac{1}{p}}$$

where $\Gamma(\alpha,\beta)$ denotes the collection of all measures on $M \times M$ with marginals α and β on the first and second factors respectively. The set $\Gamma(\alpha,\beta)$ is also called the set of all couplings of α and β.

The Wasserstein metric has an explicit expression when $M = \mathbb{R}$. Let α and β be probability measures on $\mathbb{R}$, and denote their cumulative distribution functions as F_α and F_β. Then the p-Wasserstein distance between α and β is

$$\mathcal{W}_p(\alpha, \beta) = \left(\int_0^1 |F_\alpha^{-1}(q) - F_\beta^{-1}(q)|^p \delta q \right)^{\frac{1}{p}}.$$

Furthermore, if both α and β are sum of delta functions, say $\alpha = \frac{1}{N}\sum_{i=1}^N \delta_{x_i}$ and $\beta = \frac{1}{N}\sum_{i=1}^N \delta_{y_i}$ with $x_1 \leq x_2 ... \leq x_N$ and $y_1 \leq y_2 ... \leq y_N$, then

$$\mathcal{W}_p(\alpha, \beta) = \left(\frac{1}{N} \sum_{i=1}^N |x_i - y_i|^p \right)^{\frac{1}{p}}.$$

In general, cases, evaluating Wasserstein distance is costly. Wasserstein distance has no closed form expression for higher dimension $M = \mathbb{R}^n$ of $n \geq 2$. An efficient algorithm to compute the Wasserstein distance is the Sinkhorn algorithm. Suppose $\alpha = \frac{1}{N}\sum_{i=1}^N \delta_{x_i}$ and $\beta = \frac{1}{N}\sum_{i=1}^N \delta_{y_i}$. Sinkhorn computes an ϵ-approximate solution of the Wasserstein distance $\mathcal{W}_p(\alpha, \beta)$ in $O(N^2 \log(N)\epsilon^{-3})$ operations (Altschuler et al. (2017)).

4.2.2. Energy distance

Definition 3 (Energy distance) *Let $\mathcal{P}(\mathbb{R}^n)$ denote the collection of all probability measures on the measurable space $(\mathbb{R}^n, \mathcal{B}_{\mathbb{R}^n})$, where $\mathcal{B}_{\mathbb{R}^n}$ is the Borel σ-algebra on $\mathbb{R}^n$. Then the energy distance $\mathcal{E} : \mathcal{P} \times \mathcal{P} \to \mathbb{R}$ between two probability measures μ and ν is defined as*

$$\begin{aligned} \mathcal{E}(\mu, \nu) := \int_{\mathbb{R}^n \times \mathbb{R}^n} & |x - y| \cdot d\mu(x) d\nu(y) \\ & - \frac{1}{2} \int_{\mathbb{R}^n \times \mathbb{R}^n} |x_1 - x_2| \cdot d\mu(x_1) d\mu(x_2) \\ & - \frac{1}{2} \int_{\mathbb{R}^n \times \mathbb{R}^n} |y_1 - y_2| \cdot d\nu(y_1) d\nu(y_2), \end{aligned}$$

where $|\cdot|$ denotes the Euclidean distance.

If X and Y are two $\mathbb{R}^n$-valued random variables defined on probability space $(\Omega, \mathcal{F}, P)$, we also write $\mathcal{E}(f_X, f_Y) := \mathcal{E}(X, Y) := \mathcal{E}(\mu_X, \mu_Y)$, where $\mu_X = P \circ X^{-1}$ and $\mu_Y = P \circ Y^{-1}$ are laws of X and Y and f_X, f_Y are densities of X and Y.

Theorem 4 *The energy distance $\mathcal{E}(X, Y)$ is always non-negative, and equal to zero if and only if X and Y are identically distributed.*

Proof See an available proof in Székely and Rizzo (2013). ■

4.2.3. NEW "MUTUAL INFORMATION"

Definition 5 *Let (X, Y) be a continuous random variable, and suppose X is $\mathbb{R}^n$-valued. We define*

$$J(X;Y) := \int_{\mathbb{R}^n} \mathcal{E}\big(f_{Y|X}(\cdot|x), f_Y\big) f_X(x) dx,$$

where f_X and f_Y are the probability density function of X and Y, and $f_{Y|X}$ is the conditional density function. Similarly, for discrete random variable (X, Y), we define

$$J(X;Y) := \sum_x \mathcal{E}\big(f_{Y|X}(\cdot|x), f_Y\big) P(X = x)$$

where f_Y and $f_{Y|X}$ are in the form of sum of delta functions.

Theorem 6 *The $J(X;Y) = 0$ holds if and only if X and Y are independent.*

Proof If X and Y are independent, then the conditional distribution of Y given X equals the unconditional distribution of Y. By theorem 4, for any x we have $\mathcal{E}\big(f_{Y|X}(\cdot|x), f_Y\big)$. Thus $J(X;Y) = 0$. Conversely, if $J(X;Y) = 0$, then $\mathcal{E}\big(f_{Y|X}(\cdot|x), f_Y\big) = 0$ holds almost surely. Thus for almost all the x, we have $f_{Y|X}(\cdot|x) \overset{a.e.}{=} f_Y$. Thus X and Y are independent. ■

4.2.4. HOW TO APPLY IT IN SYMAE

The advantage of the energy distance is that we can directly evaluate it using empirical distributions. Suppose the latent codes are $\{(C_i, N_{i,j})\}_{i\in[n_x], j\in[n_\tau]}$. Consider the empirical probability density functions

$$\begin{aligned} p_i &= \frac{1}{n_\tau} \sum_{j=1}^{n_\tau} \delta_{N_{i,j}} \\ p &= \frac{1}{n_x n_\tau} \sum_{i=1}^{n_x} \sum_{j=1}^{n_\tau} \delta_{N_{i,j}}. \end{aligned} \tag{9}$$

The regularizer, which will encourage disentanglement of the latent codes, is constructed as

$$\begin{aligned} \mathcal{J}(NEnc) =& \frac{1}{n_x} \sum_{i=1}^{n_x} \mathcal{E}(p_i, p) \\ =& \frac{1}{n_x} \sum_{i=1}^{n_x} \Bigg(\frac{2}{n_x n_\tau^2} \sum_{j=1}^{n_\tau} \sum_{i'=1}^{n_x} \sum_{j'=1}^{n_\tau} |N_{i,j} - N_{i',j'}| \\ & - \frac{1}{n_\tau^2} \sum_{j=1}^{n_\tau} \sum_{j'=1}^{n_\tau} |N_{i,j} - N_{i,j'}| \\ & - \frac{1}{n_x^2 n_\tau^2} \sum_{i_1=1}^{n_x} \sum_{j_1=1}^{n_\tau} \sum_{i_2=1}^{n_x} \sum_{j_2=1}^{n_\tau} |N_{i_1,j_1} - N_{i_2,j_2}| \Bigg). \end{aligned}$$

Remark 7 *If $\mathscr{J} = 0$, then according to the definition of $\mathscr{J}$, we have*

$$\mathcal{E}(p_i, p) = 0, \quad \forall i.$$

Consequently, $p_i = \frac{1}{n_\tau} \sum_{j=1}^{n_\tau} \delta_{N_{i,j}}$ does not depend on i, and thus the set of nuisance codes $N_{i,j} j \in [n\tau]$ is identical for different i. This indicates that the nuisance codes do not depend on their states.

The total loss function is

$$L(Enc, Dec) = L_{rec}(Enc, Dec) + \lambda \cdot \mathscr{J}(NEnc),$$

where L_{rec} is the reconstruction loss as defined in (5).

Remark 8 *When $\lambda = 0$, L goes back to L_{rec}.*

We present a numerical result below.

Example 2 *We set parameters $p = 5, q = 10, n_x = 6, n_\tau = 97$. The redatuming result and PCA of the latent nuisance space are shown in Figure 6 and 7.*

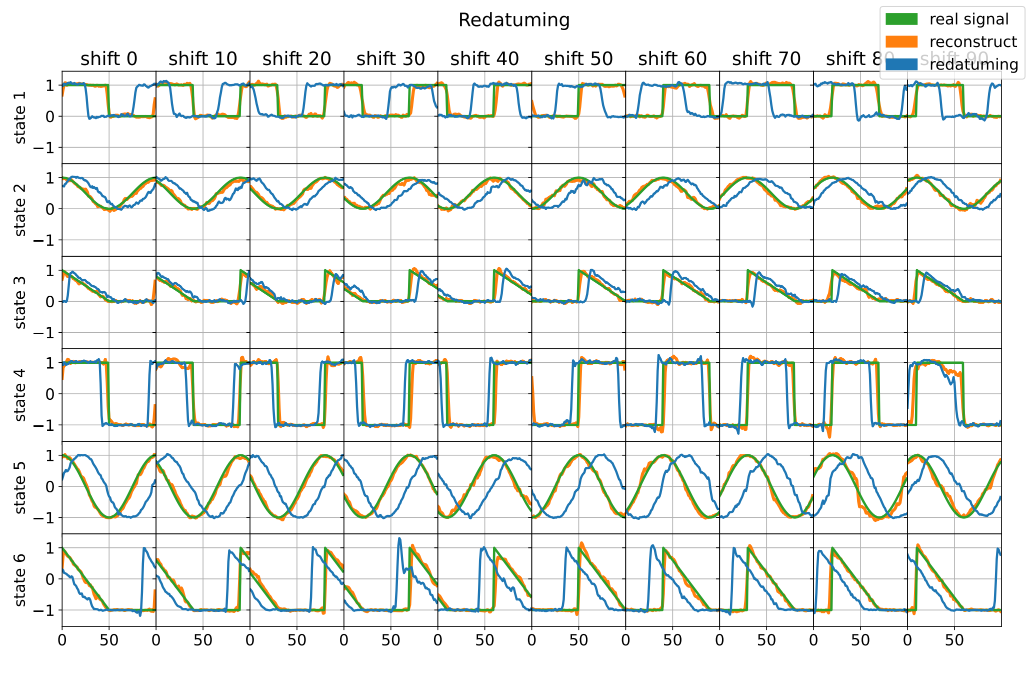

Figure 6: Redatuming of of Example 2. We set parameters $p = 5, q = 10, n_x = 6, n_\tau = 97$. Use a penalty of λ=10 and no dropout. The redatuming result is good as all redatuming successfully reconstructs the shape of the state. The shifts between the blue and green lines remain the same in any fixed row, so redatuming successfully reconstructs the shift (see Remark 2).

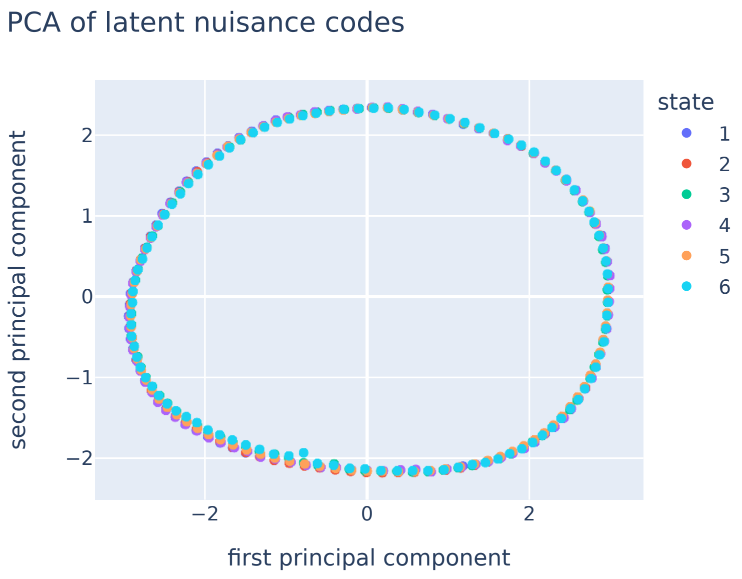

Figure 7: Latent nuisances (after PCA) of Example 2. We set parameters $p = 5, q = 10, n_x = 6, n_\tau = 97$. Use a penalty of λ=10 and no dropout. All six tori overlapped, meaning the latent nuisance code does not depend on the state. So it achieves disentanglement.

4.3. A simple concatenation model

4.3.1. INTRODUCTION

We consider a much simpler model in this section. Let states $\mathcal{A} \subset \mathbb{R}$ and nuisances $\mathcal{B} \subset \mathbb{R}$ all be one dimensional. The dataset $X = \{(s, l) | s \in \mathcal{A}, l \in \mathcal{B}\}$ contains all $n = |\mathcal{A}| \cdot |\mathcal{B}|$ datapoints. There is no structure of n_τ here, or say $n_\tau = 1$. We name the datapoints in X as $X = \{x_i\}_{i \in [n]}$.

SymAE still has four components: symmetric encoder $CEnc$, nuisance encoder $NEnc$, dropout layer SR, and the decoder Dec. The model has $3m + 2$ parameters in total, denoted as

$$\theta = (\alpha, \beta, a_1, b_1, c_1, ..., a_m, b_m, c_m)$$

where m is a fixed given constant, denoting the width of the neural network in the decoder. The simplified SymAE is as follows:

- The symmetric encoder has no parameter but simply chooses the first coordinate of a datapoint and its dimension $p = 1$, defined as

$$\begin{aligned} CEnc(\cdot; \theta) : \mathbb{R}^2 &\to \mathbb{R} \\ (s, l) &\mapsto s. \end{aligned}$$

- The nuisance encoder is linear, has two parameters α and β, defined as

$$
\begin{aligned}
NEnc(\cdot;\theta) : \mathbb{R}^2 &\to \mathbb{R}^2 \\
(s, l) &\mapsto (\alpha s + \beta l, \alpha s + \beta l)
\end{aligned}
$$

  Then the latent nuisance code has dimension $q = 2$. By the definition of the symmetric and nuisance encoders, an input of (s, l) will give a latent code of $(s, \alpha s + \beta l, \alpha s + \beta l)$.

- The decoder is a two-layer neural network (without bias) which has $3m$ parameter, defined as

$$
\begin{aligned}
Dec(\cdot\,;\theta) : \mathbb{R}^3 &\to \mathbb{R} \\
(s, N_1, N_2) &\mapsto \sum_{j=1}^{m} \sigma\big(a_j s + b_j N_1 + c_j N_2\big)
\end{aligned}
$$

  where σ is the ReLU activation.

- The dropout is not implemented through a random Bernoulli mask on nuisance codes. Instead, we eliminate the randomness by changing the random loss function into its expectation.

$$
\begin{aligned}
L(\theta) = &\sum_{i=1}^{n}\left(\sum_{j=1}^{m} \sigma\big(a_j s_i + (b_j + c_j)(\alpha s_i + \beta l_i)\big) - l_i\right)^2 \\
&+ \sum_{i=1}^{n}\left(\sum_{j=1}^{m} \sigma\big(a_j s_i + b_j(\alpha s_i + \beta l_i)\big) - l_i\right)^2 \\
&+ \sum_{i=1}^{n}\left(\sum_{j=1}^{m} \sigma\big(a_j s_i + c_j(\alpha s_i + \beta l_i)\big) - l_i\right)^2 .
\end{aligned} \tag{10}
$$

Here, since $q = 2$ is small, we discard the term where the mask is $(0, 0)$, where all nuisance information is discarded, making reconstruction impossible (this happens with probability $1/2^q = 1/4$).

4.3.2. Experiment

States are $\mathcal{A} = [10]$ and nuisances are $\mathcal{B} = [20]$. The width of the neural network is fixed to be $m = 100$. The initialization of the decoder follows the default initialization in TensorFlow, the Glorot uniform initializer (Glorot and Bengio (2010)). We run through different initializations $\alpha_0 = \alpha(t = 0)$ and $\beta_0 = \beta(t = 0)$. For each initialization (α_0, β_0), the neural network is trained for 10000 epochs.[4] Then we plot the value of $\alpha_\infty = \alpha(t = \infty)$ and $\beta_\infty = \alpha(t = \infty)$ with respect to different initialization (α_0, β_0) in Figure 8. For most (α_0, β_0), as shown in Figure 8, the latent codes will be $(s, \alpha_\infty s + \beta_\infty l, \alpha_\infty s + \beta_\infty l) \approx (s, \pm l, \pm l)$ and achieve perfect disentanglement.

For comparison, we use 0.5 GaussianDropout as the mask. Since GaussianDropout does not suffer from the problem of throwing away all information, we still use the loss function:

$$
L(\theta) = \sum_{i=1}^{n}\left(\sum_{j=1}^{m} \sigma\big(a_j s_i + (t_j^{(1)} b_j + t_j^{(2)} c_j)(\alpha s_i + \beta l_i)\big) - l_i\right)^2 ,
$$

4. Note that we feed in all the data, so there's no overfitting in this sense. Instead, we want to see if the autoencoder learns to disentangle coherent information and nuisance information in the latent space.

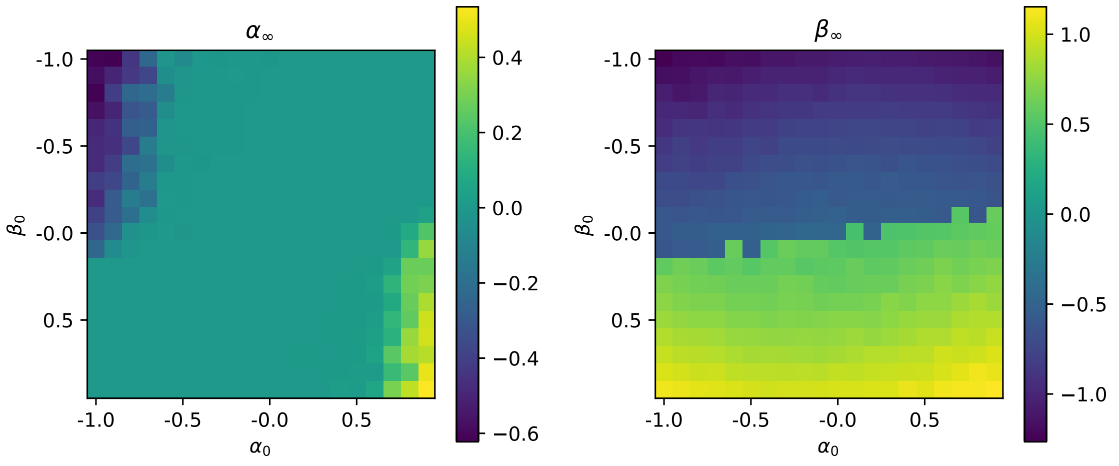

Figure 8: The contours shows the values of α_∞ and β_∞ as a function of initialization (α_0, β_0). We test all possible initialization of (α_0, β_0) in $(0.1\mathbb{Z} \cap [-1, 1]) \times (0.1\mathbb{Z} \cap [-1, 1])$, with the decoder following Glorot initialization. The left figure displays a vast green region, meaning that most of the (α_0, β_0) combinations eventually converge to $\alpha_\infty \approx 0$. The right figure displays a sharp boundary $\beta_0 \approx 0$ between the two regions $\alpha_\infty \approx \pm 1$. So for most initializations, the latent codes will be $(s, \pm l, \pm l)$ and achieve perfect disentanglement.

where $t_j^{(1,2)}$ are sampled from $\mathcal{N}(1, 1)$ and we re-sample them in each epoch . However, as shown in Figure 9, it completely fails to disentangle. In contrast to Figure 8, where α_∞ tends to be zero, in Figure 9 the absolute value of α_∞ is large.

4.3.3. DISCUSSION

The universal approximation theorem of a two-layer neural network gives that for any $(\alpha_\infty, \beta_\infty) \in \mathbb{R}^2$ with $\beta_\infty \neq 0$, as $m \to \infty$, $\min_{a_i, b_i, c_i} L(\theta) \to 0$. Figure 8 demonstrates that by running gradient descent on Equation (10), with a high probability (under the initialization $(\alpha_0, \beta_0) \sim \mathcal{U}([-1, 1]^2)$), from many different global minimizers of $L(\theta)$, the gradient descent algorithm will always converge to a specific one of them (the one with $\alpha_\infty = 0$). Additionally, disentanglement completely fails when using Gaussian dropout as shown in Figure 9, which is still not fully understood.

A similar phenomenon called ”implicit regularization,” which refers to the preference of optimization algorithms to converge to certain minima, has been studied for overparameterized models. However, even for a depth-2 neural network, implicit regularization exhibits ”complicated data-dependent behavior” (Vardi and Shamir (2021)).

Furthermore, in the particular setting of this section, the dropout layer does not play the role of ”throwing away information,” since we remove the randomness of the dropout layer. SymAE works might because the dropout layer changes the landscape of the loss function.

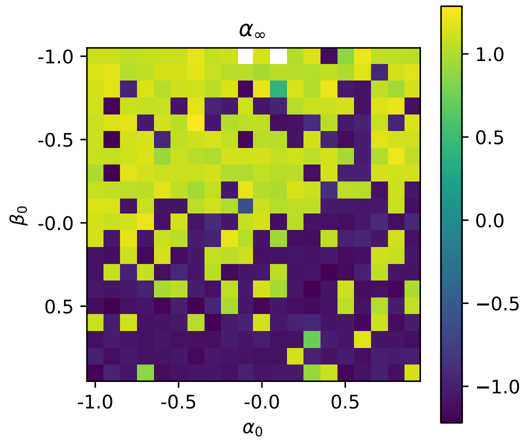

Figure 9: Using the 0.5 GaussianDropout as the mask. The contour shows the values of α_∞ as a function of initialization (α_0, β_0). We test all possible initializations of (α_0, β_0) in $(0.1\mathbb{Z} \cap [-1, 1]) \times (0.1\mathbb{Z} \cap [-1, 1])$. The absolute value of α_∞ is large, which means the autoencoder fails to disentangle.

References

Keiiti Aki and Paul G Richards. Quantitative seismology. sausalito. *Calif: University Science Books*, 2002.

Jason Altschuler, Jonathan Niles-Weed, and Philippe Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/491442df5f88c6aa018e86dac21d3606-Paper.pdf.

David Berthelot, Colin Raffel, Aurko Roy, and Ian Goodfellow. Understanding and improving interpolation in autoencoders via an adversarial regularizer. *arXiv preprint arXiv:1807.07543*, 2018.

Pawan Bharadwaj, Matthew Li, and Laurent Demanet. Redatuming physical systems using symmetric autoencoders. *Phys. Rev. Research*, 4:023118, May 2022. doi: 10.1103/PhysRevResearch.4.023118. URL https://link.aps.org/doi/10.1103/PhysRevResearch.4.023118.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*, 2017.

Aapo Hyvärinen and Petteri Pajunen. Nonlinear independent component analysis: Existence and uniqueness results. *Neural networks*, 12(3):429–439, 1999.

Jean-Paul Montagner, Anne Mangeney, and Eléonore Stutzmann. Seismology and environment. *Encyclopedia of Solid Earth Geophysics*, pages 1–8, 2020.

S Narendranath, PS Athiray, P Sreekumar, BJ Kellett, L Alha, CJ Howe, KH Joy, M Grande, J Huovelin, IA Crawford, et al. Lunar x-ray fluorescence observations by the chandrayaan-1 x-ray spectrometer (c1xs): Results from the nearside southern highlands. *Icarus*, 214(1):53–66, 2011.

Amelia Perry, Jonathan Weed, Afonso S. Bandeira, Philippe Rigollet, and Amit Singer. The sample complexity of multireference alignment. *SIAM Journal on Mathematics of Data Science*, 1(3):497–517, 2019. doi: 10.1137/18M1214317. URL https://doi.org/10.1137/18M1214317.

Gabriel Peyré and Marco Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11 (5-6):355–602, 2019. URL https://arxiv.org/abs/1803.00567.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Gábor J. Székely and Maria L. Rizzo. Energy statistics: A class of statistics based on distances. *Journal of Statistical Planning and Inference*, 143(8):1249–1272, 2013. ISSN 0378-3758. doi: https://doi.org/10.1016/j.jspi.2013.03.018. URL https://www.sciencedirect.com/science/article/pii/S0378375813000633.

Einar Tandberg-Hanssen and A Gordon Emslie. *The physics of solar flares*, volume 14. Cambridge University Press, 1988.

Gal Vardi and Ohad Shamir. Implicit regularization in relu networks with the square loss. In Mikhail Belkin and Samory Kpotufe, editors, *Proceedings of Thirty Fourth Conference on Learning Theory*, volume 134 of *Proceedings of Machine Learning Research*, pages 4224–4258. PMLR, 15–19 Aug 2021. URL https://proceedings.mlr.press/v134/vardi21b.html.

Appendix A. SymAE architecture and training procedure

Our implementation is available on https://github.com/Sr-11/SymAE, where the reader can look for more details. All MRA examples are trained with 3000 epochs with the default Adam optimizer in TensorFlow. The autoencoder is built primarily with convolutional layers.

The architecture in Example 1 is described below. The first part of the symmetric encoder $CEnc_1$ consists of two blocks of two consecutive convolutional layers with 32 filters and kernel size 5, followed by a 2 max pooling operation. All convolutional layers use an ELU nonlinearity, and all layers in this part are implemented within the TimeDistributed wrapper. The output values from each block are then averaged to obtain the input for the second part $CEnc_2$, which does not use the TimeDistributed wrapper.

The second part of the symmetric encoder $CEnc_2$ also consists of two blocks of two consecutive convolutional layers with 32 filters and kernel size 5, followed by a 2 max pooling operation. All convolutions are zero-padded to maintain equal input and output height and width. After this, a BatchNormalization layer, an ELU activation layer, another 2 max pooling, and a dense layer (with no activation function) are applied consecutively. The final output from this dense layer serves as the latent coherent code.

The nuisance encoder $NEnc$ is composed of 4 blocks, each containing two consecutive convolutional layers with 32 filters and kernel size 5, followed by a 2 max pooling operation. The activation function used in all convolutional layers is ELU. The number of channels is maintained at 32 throughout the encoder. After the 4 blocks, a BatchNormalization layer, an ELU activation layer, a 2 max pooling layer, and a dense layer (without activation) are applied consecutively. All layers in the nuisance encoder are implemented using the TimeDistributed wrapper. The final output from this sequence of layers is used as the latent nuisance code.

The decoder dec is built as

$$dec(z) = dec_3(dec_1(z) \odot dec_2(z))$$

where dec_1, dec_2, dec_3 are neural nets and $\odot$ denotes element-wise product. The net dec_1 consists of a dense layer with a width of 50, two 1D convolutional layers, a 2 upsampling layer, another two 1D convolutional layers, a batch normalization layer, two more convolutional layers, and finally a convolutional layer. Except for the last convolutional layer, which has only 1 filter and with no activation, all other convolutional layers have 32 filters, kernel size 5, and ELU activation. The network dec_2 is a 3-layer dense network, where all hidden layers have a width of 200 and use the ELU activation function. The output layer has a width of 100 and has no activation function. The network dec_3 has the same architecture as dec_1.

Appendix B. More numerical results

Example 3 (Concatenation Decoder) *Compared to Example 1, a different decoder architecture is used here. The latent coherent code and the latent nuisance code are concatenated and then fed into a convolutional neural network, see more detail in our GitHub repository. The redatuming result and PCA of the latent nuisance space are shown in Figure 10 and 11.*

Example 4 (Additive GaussianNoise) *The redatuming result and PCA of the latent nuisance space are shown in Figure 12 and 13. The only effect of the GaussianNoise is scaling all the latent codes. The shapes of the latent torus of different epochs remain the same, while the range of latent codes increases from around 5 to around 100. The additive noise does not improve redatuming.*

Example 5 (Spectral Bias) *Compared to Example 1, a different decoder architecture is used here. The decoder is built with dense nets but not convolutional nets, see more detail in our GitHub repository. The redatuming result is shown in Figure 14.*

Example 6 *We present one more result on MNIST in Figure 15.*

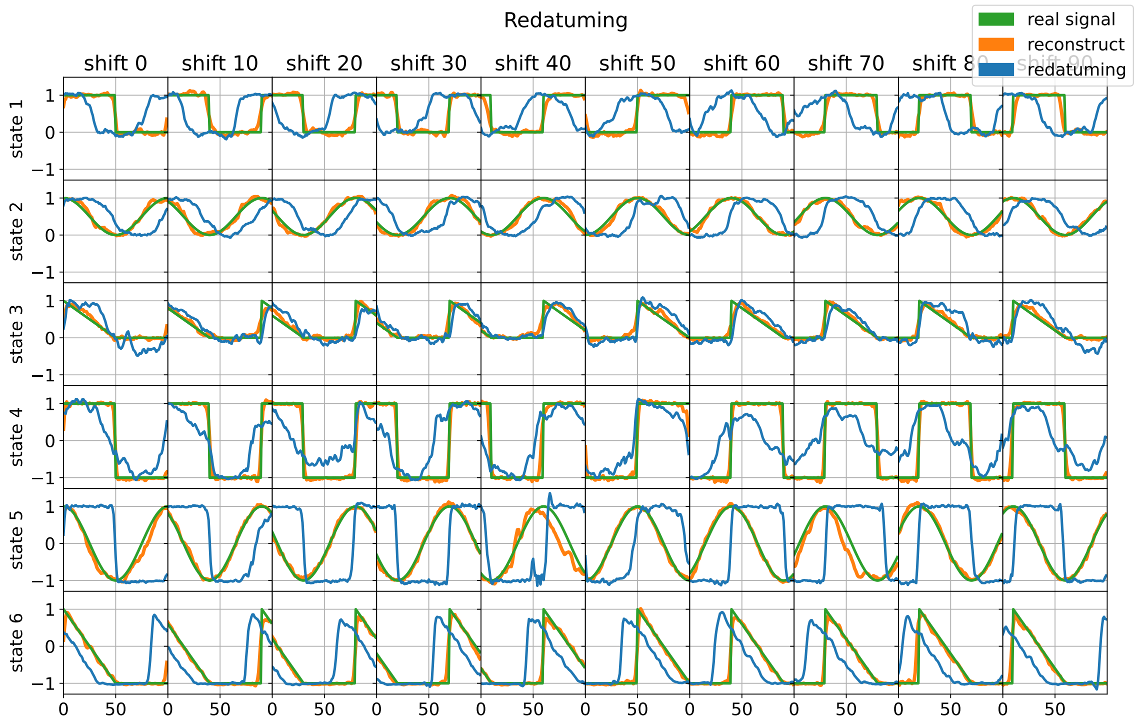

Figure 10: Redatuming of Example 3. Parameters are p=5, q=10, n_x=18, n_τ=30. We still use Dropout for the dropout layer, with a dropout rate $r = 0.5$. The redatuming of states 1, 2, 3, and 6 are good. But redatuming at state 4 and state 5 are bad.

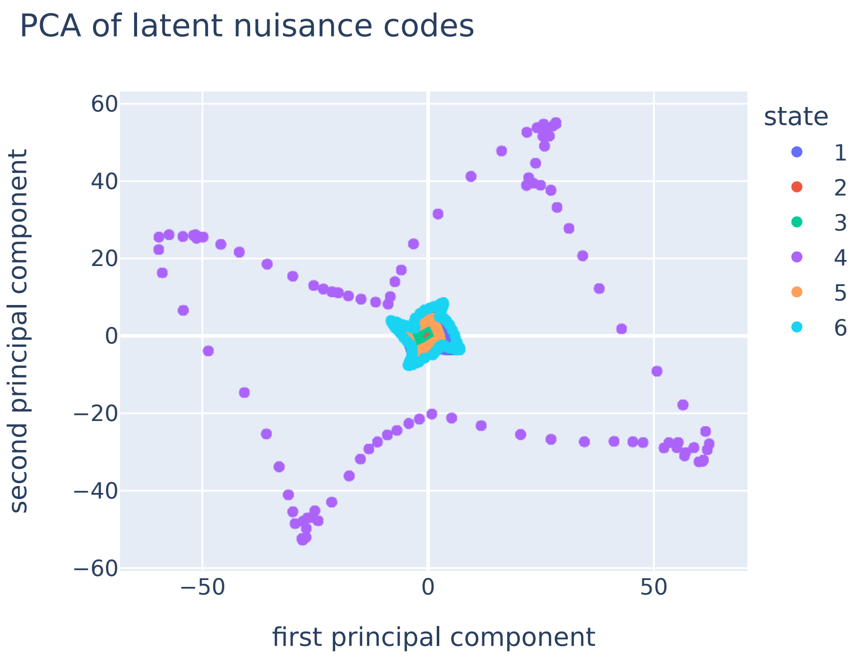

Figure 11: Latent nuisances of Example 3. Parameters are p=5, q=10, n_x=18, n_τ=30. We still use Dropout for the dropout layer, with a dropout rate $r = 0.5$. These 6 tori have similar shapes, and they do not overlap.

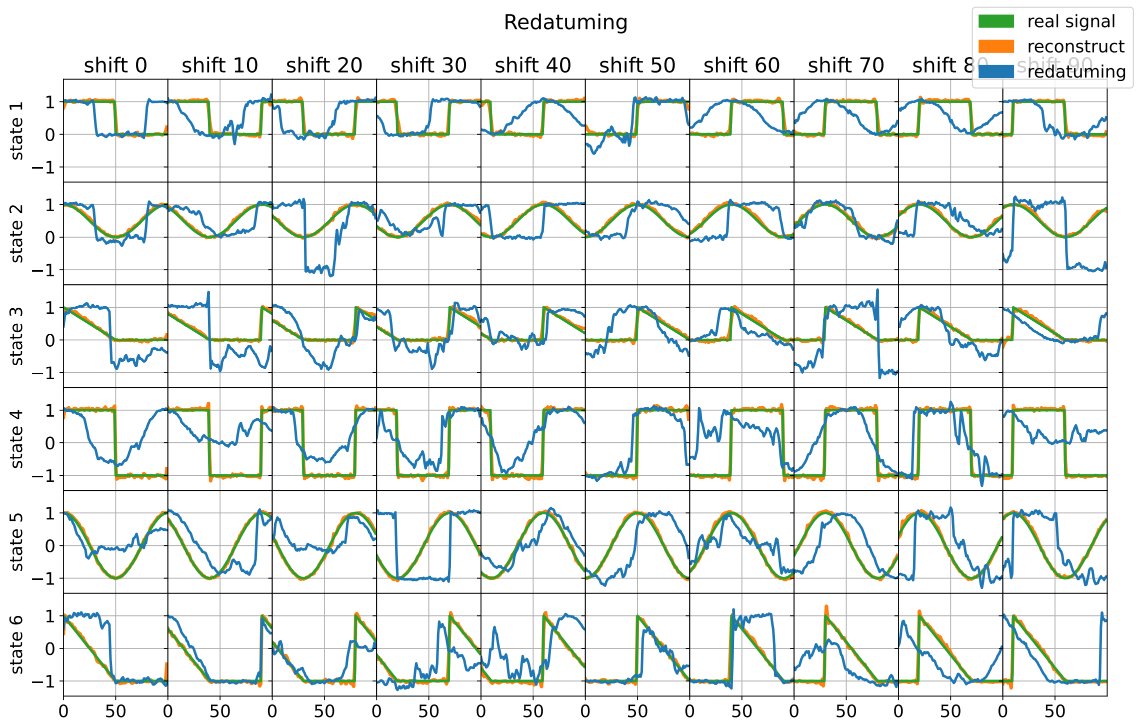

Figure 12: Redatuming of Example 4. Hyperparameters are p=5, q=10, n_x=18, n_τ=30. Using additive GaussianNoise. The redatuming is very bad that there is no subplot in which the green and blue lines have the same shape.

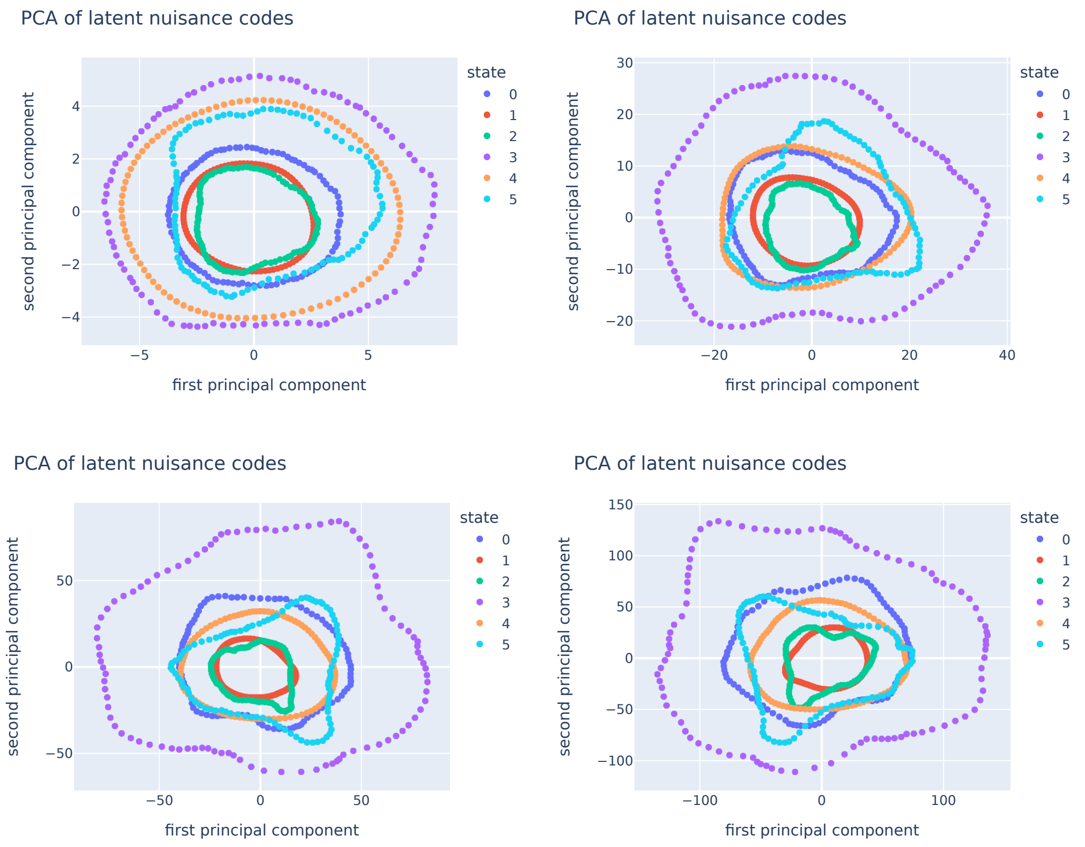

Figure 13: Latent nuisances of of Example 4, using additive GaussianNoise. The plots are the snapshots of latent nuisance space of different epochs. The upper-left, upper-right, bottom-left, and bottom-right subplots are taken at 100, 500, 2000, and 4000 epochs, respectively.

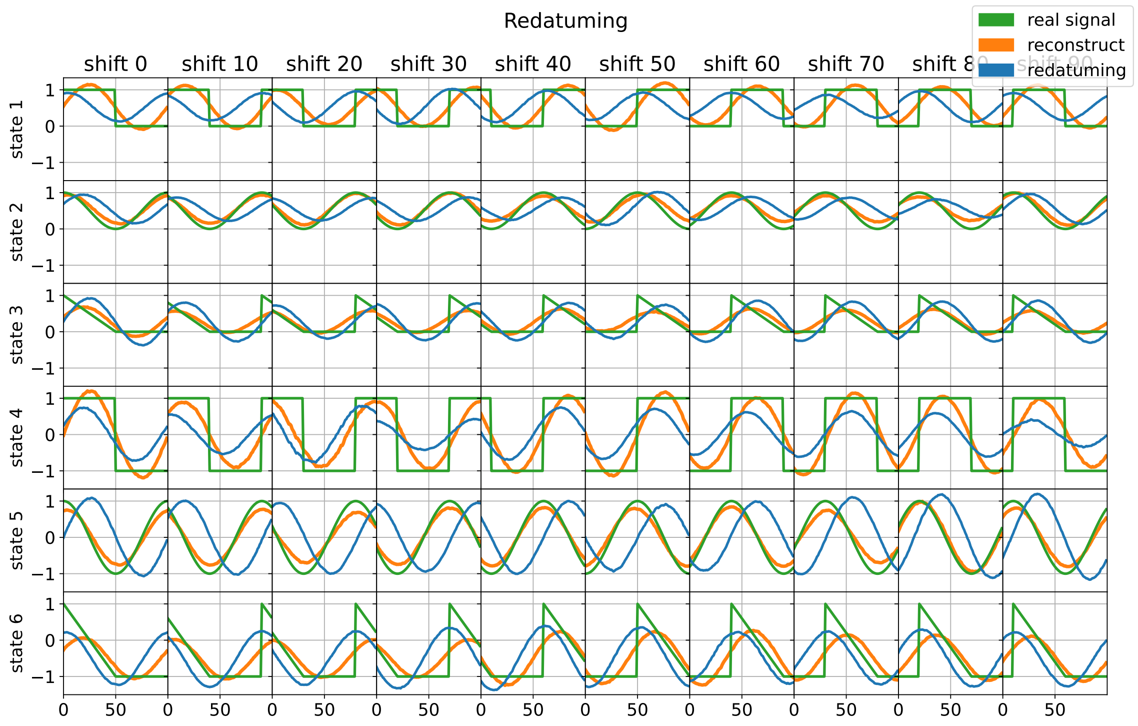

Figure 14: Redatuming of Example 5. The autoencoder can only learn low-frequency components of the signals.

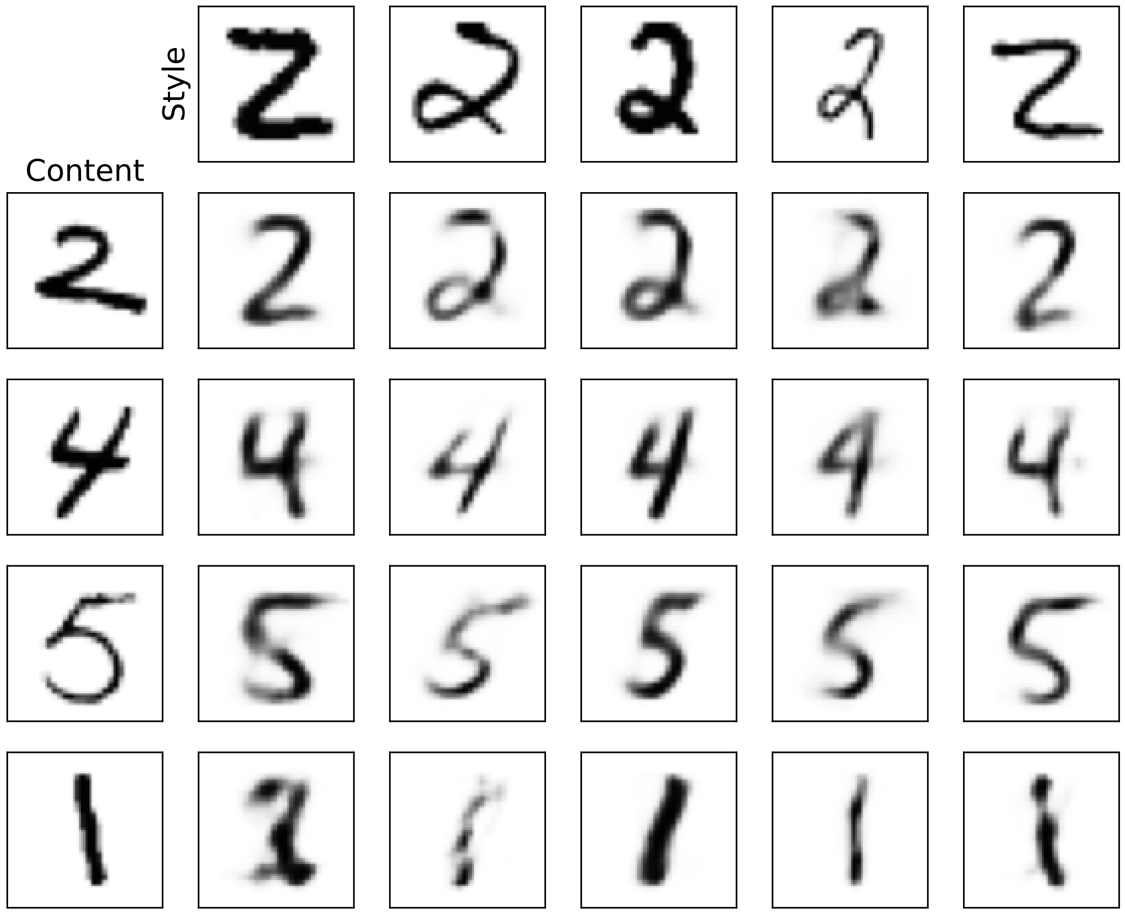

Figure 15: There are 20 redatuming images displayed in this figure. Each redatuming image is generated by mixing the coherent code from the left-most image (with contents 2, 4, 5, and 1), with the nuisance code from the top-most image (with content 2). The redatuming results are good, as the images in each column exhibit similar writing styles (note that in the MNIST task, there is no ground truth for redatuming).