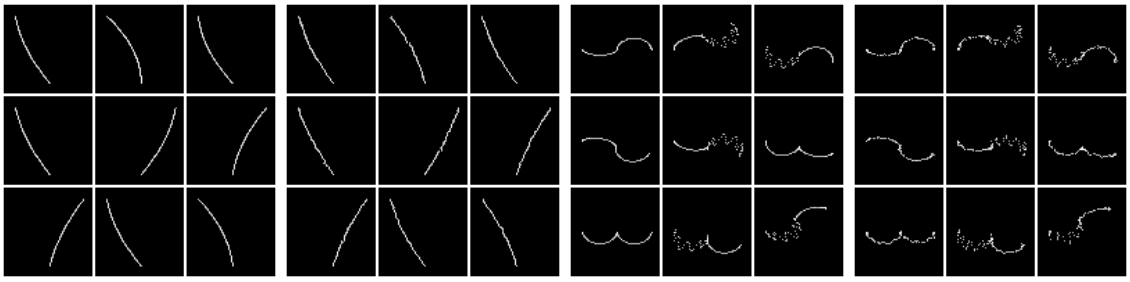
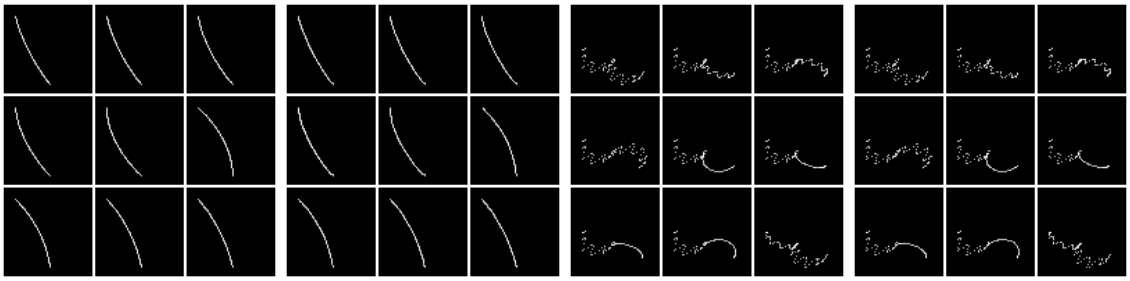
Appendix A. Reconstruction and Latent traversal

We show reconstruction and latent traversal results in Fig. 1 and Fig. 2. In 2D Reaching and 2D Wavy Reaching tasks, we picked the models with the highest MIGs and plotted the reconstruction and latent traverse. In each block of Fig. 1, the left is input (3x3) and the right is its reconstruction (3x3). Fig. 2 shows the latent traversal results (Vertical axis: z-dimension (the meaningless z-dim is omitted to save space. The bottom row z is meaningless z-dim), Horizontal axis: z-value (varied from -3.0 to 3.0)).

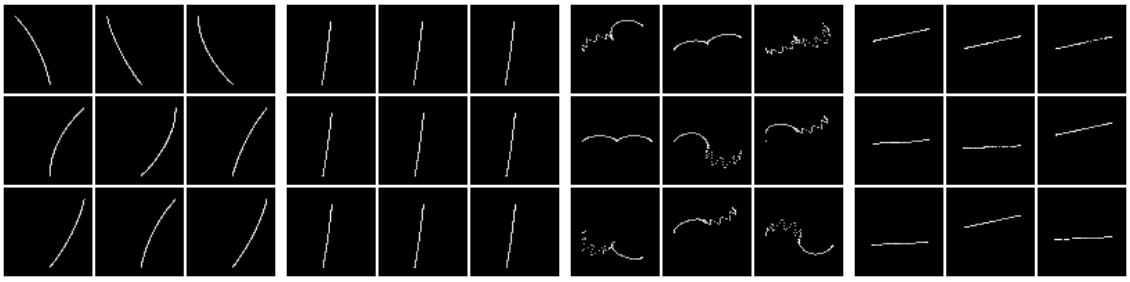
Thorough these results we can confirm in a qualitative manner that FAVAE (proposed) is capable of extracting dynamic factors (like wave patterns). DSAE fails to reconstruct and FHVAE cannot disentangle all factors (some factors are entangled).



(a) Reconstruction in FAVAE.

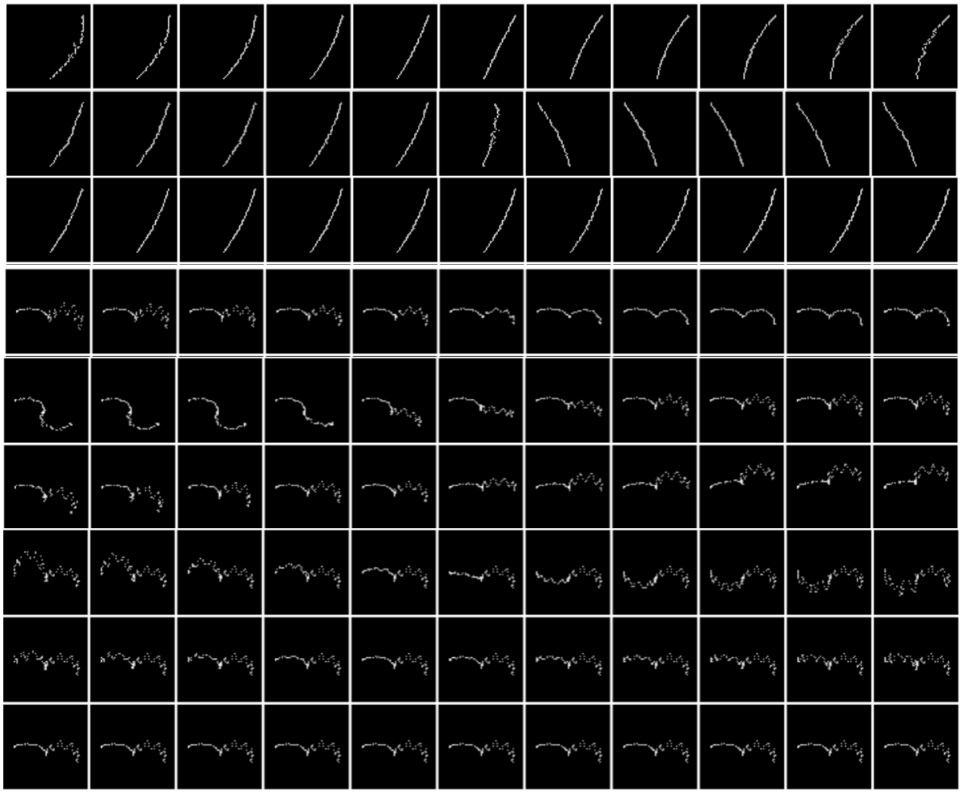


(b) Reconstruction in FHVAE.

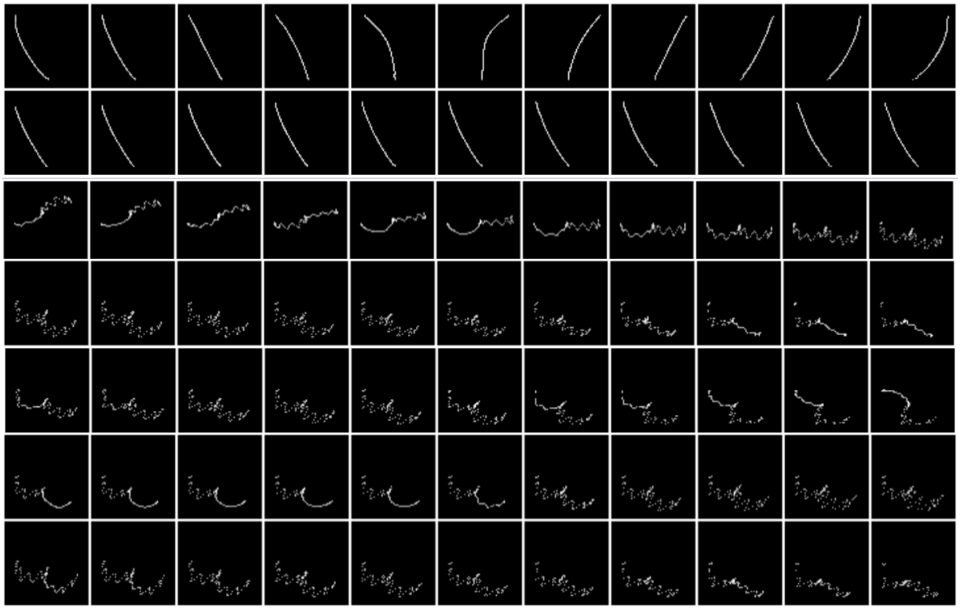


(c) Reconstruction in DSAE.

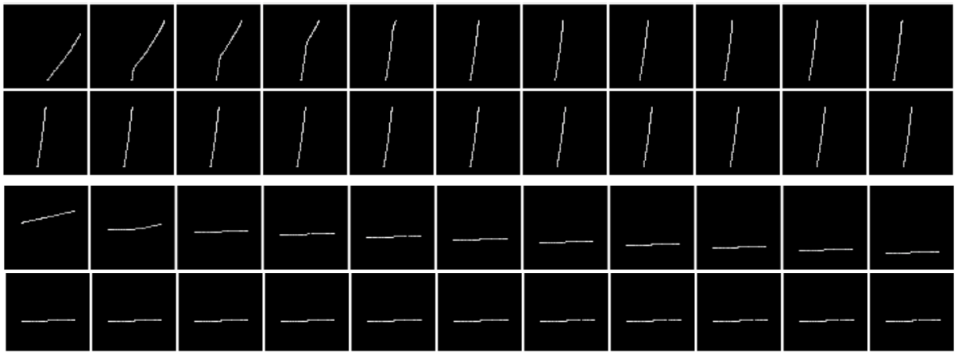
Figure 1: Reconstruction in 2D Reaching and 2D Wavy Reaching.



(a) Latent traversal in FAVAE.



(b) Latent traversal in FHVAE.



(c) Latent traversal in DSAE.

Figure 2: Latent traversal in 2D Reaching and 2D Wavy Reaching.

Appendix B. Evaluation of FHVAE with time convolution

To check the difference between the time convolution and LSTM, we performed an experiment in which we replaced the LSTM of FHVAE with the time convolution.

data	MIG	Rec
2D Reaching (length=100)	0.11(7)	0.63(133)
2D Reaching (length=1000)	0.14(6)	1.392(1449)
2D Wavy Reaching (length=100)	0.05(4)	3.476(2551)
2D Wavy Reaching (length=1000)	0.17(10)	5.292(12149)

Table 1: FHVAE with time convolution.

Appendix C. Detail of FAVAE (proposed)

This section shows the model details of FAVAE with hyperparameter search. Listing 1, Listing 2 and Listing 3 show the architecture of FAVAE by using "print(model)" in PyTorch (Paszke et al., 2019). The neural network of FAVAE is compose of following blocks (e.g. time_encode, time_decode, fc_encode, fc_decode, time_decode_not_last, gate) in 2D Reaching, 2D Wavy Reaching. The Encoder used time_encode and fc_encode. The Decoder used time_decode, fc_decode and time_decode_not_last (except the last layer in upper ladder). The gate is used at the junction of the ladder network in the decoder. The dimension of latent variables whcih are Gaussian distributions are 8, 4, 2 in lower, middle and higher ladder. When we used without ladder network, we disconnect the gate and only used upper ladder for reconstruction and KL divergence losses.

In Sprites, we add pre_conv, pre_conv_fc, post_deconv_fc and post_deconv blocks for image dataset as Listing 2. The input data is calculated in order of pre_conv, pre_conv_fc, time_encode and fc_encode, latent variable, fc_decode, time_decode_not_last, post_deconv_fc and post_deconv.

In Timit, we modify the channel in Conv1d and ConvTranspose1d as Listing 3.

Listing 1: The architecture of FAVAE in 2D Reaching and 2D Wavy Reaching

```
FAVAE(
  (time_encode): ModuleDict(
    (ladder0): Sequential(
      (0): Conv1d(2, 4, kernel_size=(3,), stride=(2,), padding=(1,))
      (1): BatchNorm1d(4, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): Conv1d(4, 8, kernel_size=(3,), stride=(2,), padding=(1,))
      (4): BatchNorm1d(8, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (5): LeakyReLU(negative_slope=0.01)
    )
    (ladder1): Sequential(
      (0): Conv1d(8, 16, kernel_size=(3,), stride=(2,), padding=(1,))
      (1): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): Conv1d(16, 32, kernel_size=(3,), stride=(2,), padding=(1,))
      (4): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (5): LeakyReLU(negative_slope=0.01)
    )
    (ladder2): Sequential(
      (0): Conv1d(32, 64, kernel_size=(3,), stride=(2,), padding=(1,))
      (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): Conv1d(64, 128, kernel_size=(3,), stride=(2,), padding=(1,))
      (4): BatchNorm1d(128, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (5): LeakyReLU(negative_slope=0.01)
    )
  )
  (time_decode): ModuleDict(
    (ladder0): Sequential(
      (0): ConvTranspose1d(8, 4, kernel_size=(3,), stride=(2,), padding
         =(1,), output_padding=(1,))
      (1): BatchNorm1d(4, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): ConvTranspose1d(4, 2, kernel_size=(3,), stride=(2,), padding
         =(1,), output_padding=(1,))
    )
    (ladder1): Sequential(
      (0): ConvTranspose1d(32, 16, kernel_size=(3,), stride=(2,), padding
         =(1,))
      (1): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): ConvTranspose1d(16, 8, kernel_size=(3,), stride=(2,), padding
         =(1,))
    )
    (ladder2): Sequential(
      (0): ConvTranspose1d(128, 64, kernel_size=(3,), stride=(2,), padding
         =(1,), output_padding=(1,))
      (1): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): ConvTranspose1d(64, 32, kernel_size=(3,), stride=(2,), padding
         =(1,))
    )
```

```
  )
  (fc_encode): ModuleDict(
    (ladder0): Sequential(
      (0): Linear(in_features=200, out_features=16, bias=True)
    )
    (ladder1): Sequential(
      (0): Linear(in_features=224, out_features=8, bias=True)
    )
    (ladder2): Sequential(
      (0): Linear(in_features=256, out_features=4, bias=True)
    )
  )
  (fc_decode): ModuleDict(
    (ladder0): Sequential(
      (0): Linear(in_features=8, out_features=200, bias=True)
      (1): BatchNorm1d(200, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (ladder1): Sequential(
      (0): Linear(in_features=4, out_features=224, bias=True)
      (1): BatchNorm1d(224, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (ladder2): Sequential(
      (0): Linear(in_features=2, out_features=256, bias=True)
      (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
  )
  (time_decode_not_last): ModuleDict(
    (ladder1): Sequential(
      (0): BatchNorm1d(8, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (1): LeakyReLU(negative_slope=0.01)
    )
    (ladder2): Sequential(
      (0): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (1): LeakyReLU(negative_slope=0.01)
    )
  )
  (gate): ParameterDict(
      (ladder0): Parameter containing: [torch.FloatTensor of size 8x25]
      (ladder1): Parameter containing: [torch.FloatTensor of size 32x7]
      (ladder2): Parameter containing: [torch.FloatTensor of size 128x2]
  )
  (fc_decode_param): Linear(in_features=2, out_features=2, bias=True)
)
```

Listing 2: The architecture of FAVAE in Sprites

```
CNNFAVAE(
  (time_encode): ModuleDict(
    (ladder0): Sequential(
      (0): Conv1d(48, 96, kernel_size=(3,), stride=(2,), padding=(1,))
```

```
    (1): BatchNorm1d(96, eps=1e-05, momentum=0.1, affine=True,
       track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): Conv1d(96, 192, kernel_size=(3,), stride=(2,), padding=(1,))
    (4): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
       track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.01)
  )
  (ladder1): Sequential(
    (0): Conv1d(192, 192, kernel_size=(3,), stride=(2,), padding=(1,))
    (1): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
       track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): Conv1d(192, 192, kernel_size=(3,), stride=(2,), padding=(1,))
    (4): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
       track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.01)
  )
  (ladder2): Sequential(
    (0): Conv1d(192, 192, kernel_size=(3,), stride=(2,), padding=(1,))
    (1): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
       track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): Conv1d(192, 192, kernel_size=(3,), stride=(2,), padding=(1,))
    (4): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
       track_running_stats=True)
    (5): LeakyReLU(negative_slope=0.01)
  )
)
(time_decode): ModuleDict(
  (ladder0): Sequential(
    (0): ConvTranspose1d(192, 96, kernel_size=(3,), stride=(2,), padding
       =(1,), output_padding=(1,))
    (1): BatchNorm1d(96, eps=1e-05, momentum=0.1, affine=True,
       track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): ConvTranspose1d(96, 48, kernel_size=(3,), stride=(2,), padding
       =(1,), output_padding=(1,))
  )
  (ladder1): Sequential(
    (0): ConvTranspose1d(192, 192, kernel_size=(3,), stride=(2,), padding
       =(1,))
    (1): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
       track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): ConvTranspose1d(192, 192, kernel_size=(3,), stride=(2,), padding
       =(1,), output_padding=(1,))
  )
  (ladder2): Sequential(
    (0): ConvTranspose1d(192, 192, kernel_size=(3,), stride=(2,), padding
       =(1,))
    (1): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
       track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.01)
    (3): ConvTranspose1d(192, 192, kernel_size=(3,), stride=(2,), padding
       =(1,))
  )
)
(fc_encode): ModuleDict(
  (ladder0): Sequential(
    (0): Linear(in_features=384, out_features=16, bias=True)
  )
```

```
    (ladder1): Sequential(
      (0): Linear(in_features=192, out_features=8, bias=True)
    )
    (ladder2): Sequential(
      (0): Linear(in_features=192, out_features=4, bias=True)
    )
  )
  (fc_decode): ModuleDict(
    (ladder0): Sequential(
      (0): Linear(in_features=8, out_features=384, bias=True)
      (1): BatchNorm1d(384, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (ladder1): Sequential(
      (0): Linear(in_features=4, out_features=192, bias=True)
      (1): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (ladder2): Sequential(
      (0): Linear(in_features=2, out_features=192, bias=True)
      (1): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
  )
  (time_decode_not_last): ModuleDict(
    (ladder1): Sequential(
      (0): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (1): LeakyReLU(negative_slope=0.01)
    )
    (ladder2): Sequential(
      (0): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (1): LeakyReLU(negative_slope=0.01)
    )
  )
  (gate): ParameterDict(
      (ladder0): Parameter containing: [torch.FloatTensor of size 192x2]
      (ladder1): Parameter containing: [torch.FloatTensor of size 192x1]
      (ladder2): Parameter containing: [torch.FloatTensor of size 192x1]
  )
  (fc_decode_param): Linear(in_features=48, out_features=48, bias=True)
  (pre_conv): Sequential(
    (0): Conv2d(3, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Conv2d(128, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1)
       , bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
       track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2)
    (5): Dropout2d(p=0.5, inplace=False)
    (6): Conv2d(128, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1)
       , bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
       track_running_stats=True)
    (8): LeakyReLU(negative_slope=0.2)
    (9): Dropout2d(p=0.5, inplace=False)
    (10): Conv2d(128, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1,
       1), bias=False)
```

```
      (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (12): LeakyReLU(negative_slope=0.2)
      (13): Dropout2d(p=0.5, inplace=False)
    )
    (pre_conv_fc): Sequential(
      (0): Linear(in_features=2048, out_features=48, bias=True)
      (1): BatchNorm1d(48, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (post_deconv_fc): Sequential(
      (0): Linear(in_features=48, out_features=2048, bias=True)
      (1): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (post_deconv): Sequential(
      (0): ConvTranspose2d(128, 128, kernel_size=(4, 4), stride=(2, 2),
         padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
      (3): Dropout2d(p=0.5, inplace=False)
      (4): ConvTranspose2d(128, 128, kernel_size=(4, 4), stride=(2, 2),
         padding=(1, 1), bias=False)
      (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (6): LeakyReLU(negative_slope=0.2)
      (7): Dropout2d(p=0.5, inplace=False)
      (8): ConvTranspose2d(128, 128, kernel_size=(4, 4), stride=(2, 2),
         padding=(1, 1), bias=False)
      (9): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (10): LeakyReLU(negative_slope=0.2)
      (11): Dropout2d(p=0.5, inplace=False)
      (12): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2),
         padding=(1, 1))
      (13): Tanh()
    )
  )
```

Listing 3: The architecture of FAVAE in Timit

```
FAVAE(
  (time_encode): ModuleDict(
    (ladder0): Sequential(
      (0): Conv1d(80, 160, kernel_size=(3,), stride=(2,), padding=(1,))
      (1): BatchNorm1d(160, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): Conv1d(160, 192, kernel_size=(3,), stride=(2,), padding=(1,))
      (4): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (5): LeakyReLU(negative_slope=0.01)
    )
    (ladder1): Sequential(
      (0): Conv1d(192, 192, kernel_size=(3,), stride=(2,), padding=(1,))
```

```
      (1): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): Conv1d(192, 192, kernel_size=(3,), stride=(2,), padding=(1,))
      (4): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (5): LeakyReLU(negative_slope=0.01)
    )
    (ladder2): Sequential(
      (0): Conv1d(192, 192, kernel_size=(3,), stride=(2,), padding=(1,))
      (1): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): Conv1d(192, 192, kernel_size=(3,), stride=(2,), padding=(1,))
      (4): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (5): LeakyReLU(negative_slope=0.01)
    )
  )
  (time_decode): ModuleDict(
    (ladder0): Sequential(
      (0): ConvTranspose1d(192, 160, kernel_size=(3,), stride=(2,), padding
         =(1,), output_padding=(1,))
      (1): BatchNorm1d(160, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): ConvTranspose1d(160, 80, kernel_size=(3,), stride=(2,), padding
         =(1,), output_padding=(1,))
    )
    (ladder1): Sequential(
      (0): ConvTranspose1d(192, 192, kernel_size=(3,), stride=(2,), padding
         =(1,))
      (1): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): ConvTranspose1d(192, 192, kernel_size=(3,), stride=(2,), padding
         =(1,), output_padding=(1,))
    )
    (ladder2): Sequential(
      (0): ConvTranspose1d(192, 192, kernel_size=(3,), stride=(2,), padding
         =(1,))
      (1): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
      (3): ConvTranspose1d(192, 192, kernel_size=(3,), stride=(2,), padding
         =(1,), output_padding=(1,))
    )
  )
  (fc_encode): ModuleDict(
    (ladder0): Sequential(
      (0): Linear(in_features=25728, out_features=16, bias=True)
    )
    (ladder1): Sequential(
      (0): Linear(in_features=6528, out_features=8, bias=True)
    )
    (ladder2): Sequential(
      (0): Linear(in_features=1728, out_features=4, bias=True)
    )
  )
  (fc_decode): ModuleDict(
    (ladder0): Sequential(
      (0): Linear(in_features=8, out_features=25728, bias=True)
```

```
      (1): BatchNorm1d(25728, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (ladder1): Sequential(
      (0): Linear(in_features=4, out_features=6528, bias=True)
      (1): BatchNorm1d(6528, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
    (ladder2): Sequential(
      (0): Linear(in_features=2, out_features=1728, bias=True)
      (1): BatchNorm1d(1728, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.01)
    )
  )
  (time_decode_not_last): ModuleDict(
    (ladder1): Sequential(
      (0): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (1): LeakyReLU(negative_slope=0.01)
    )
    (ladder2): Sequential(
      (0): BatchNorm1d(192, eps=1e-05, momentum=0.1, affine=True,
         track_running_stats=True)
      (1): LeakyReLU(negative_slope=0.01)
    )
  )
  (gate): ParameterDict(
      (ladder0): Parameter containing: [torch.FloatTensor of size 192x134]
      (ladder1): Parameter containing: [torch.FloatTensor of size 192x34]
      (ladder2): Parameter containing: [torch.FloatTensor of size 192x9]
  )
  (fc_decode_param): Linear(in_features=80, out_features=80, bias=True)
)
```

C.1. Hyperparameters

The optimizer is Adam with learning rate is 10^{-3} and batch size is 128. We search the β for [0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0], [0.0, 1.0, 20.0, 40.0, 60.0, 80.0, 100.0, 120.0, 140.0], [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4], [0.0, 1.0, 2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0] and [0.000001, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, 45.0, 50.0] in 2D reaching with length 1000, 2D Wavy Reaching with length 1000, 2D Reaching with length 100, 2D Wavy Reaching with length 100, Sprites and Timit. We choosed the best parameter for maximizing the disentangle metric score (MIG). Table 2 show the best C in all dataset.

Appendix D. Detail of FHVAE (baseline model)

This section shows the model details of factorized hierarchical VAE (FHVAE) with hyperparameter search. For the FHVAE experiments, we used the original code from the implementation at Hsu et al. (2017)[1]. We used the recurrent setting with LSTM encoders and a decoder with unit size=256. The dimensions of latent variables z_1 and z_2 were 7 for each to be a fair comparison with z-dimension

1. The code is available at https://github.com/wnhsu/FactorizedHierarchicalVAE

Table 2: Best β and C in all datasets.

Dataset	Ladder option	C option	best β	best C
2D Reaching with length 100	–	–	0.6	
2D Reaching with length 100	L	–	0.4	
2D Reaching with length 100	–	C	0.2	[0]
2D Reaching with length 100	L	C	0.2	[0, 0, 0.99]
2D Wavy Reaching with length 100	–	–	2	
2D Wavy Reaching with length 100	L	–	4	
2D Wavy Reaching with length 100	–	C	2	[7.84]
2D Wavy Reaching with length 100	L	C	12	[5.61, 0.25, 4.08]
2D Reaching with length 1000	–	–	2.52	
2D Reaching with length 1000	L	–	3.5	
2D Reaching with length 1000	–	C	2	[1.17]
2D Reaching with length 1000	L	C	2	[1.35, 0.00, 1.29]
2D Wavry Reaching with length 1000	–	–	40	
2D Wavry Reaching with length 1000	L	–	60	
2D Wavry Reaching with length 1000	–	C	60	[18.6]
2D Wavry Reaching with length 1000	L	C	120	[16.37, 3.72, 4.1]
Sprites	–	–	25	
Sprites	L	–	15	
Sprites	–	C	30	[20]
Sprites	L	C	15	[17, 0.031, 0.08]
Timit	–	–	0	
Timit	L	–	10	
Timit	–	C	20	[35]
Timit	L	C	5	[19, 12, 7]

14 FAVAE. We used Adam optimizer. The batch size is 80 in 2D Reaching, 2D Wavy Reaching, Sprites. In Timit, we used batch size is 256 to match the original.

D.1. Hyperparameters

We search the alpha for [0.01, 0.1, 1, 10], the learning rate [10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}] and the distribution of decoder ['gauss', 'mean square error']. We choosed the best parameter for maximizing the disentangle metric score (MIG). The best parameters of alpha are 1, 0, 0.1, 1 in 2D Reaching, 2D Wavy Reaching, Sprites and Timit. The best parameters of learning rate are 10^{-2}, 10^{-3}, 10^{-3}, 10^{-3} in 2D Reaching, 2D Wavy Reaching, Sprites and Timit.

Appendix E. Detail of DSAE (baseline model)

For the Disentangled Sequential Autoencoder (DSAE) experiments, we used the code from the implementation at the github repository[2]. We used the dimension of time dependent latent variable

2. The code is available at https://github.com/yatindandi/Disentangled-Sequential-Autoencoder

z is 7, and time independent latent variable z_s is 7 to be a fair comparison with z-dimension 14 FAVAE. The batch size is 25 and the optimizer is Adam.

E.1. Hyperparameters

We search the learning rate for [10^{-7}, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}] for for maximizing the disentangle metric score (MIG). The best parameters of learning rate are 10^{-4}, 10^{-4}, 10^{-3}, 10^{-3} in 2D Reaching, 2D Wavy Reaching, Sprites and Timit.

References

Wei-Ning Hsu, Yu Zhang, and James Glass. Unsupervised learning of disentangled and interpretable representations from sequential data. In *Advances in Neural Information Processing Systems*, pages 1878–1889, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.