[aeproj.tex]subfiles

Appendix

In the case of AE, reconstruction loss is a mean squared error (MSE) between an input x and output x' of the network: $\mathcal{L}(x, x') = \|x - x'\|^2$. The network can be trained using standard machine learning techniques for training such as backpropagation. We utilise a regularized version of a standard AE which is known as *Sparse Autoencoder*. While L1 regularization is applied to the weight matrix of the final dense layer of the encoder which produced the latent vectors to make it sparse, L2 regularization is utilized for output of this layer to prevent its growth and overfitting [25].

Another version of considered autoencoders is *Sliced-Wasserstein Autoencoder (SWAE)*. This is a generative model with a simple implementation and which does not require adversarial training [20]. SWAE objective consists of a Wasserstein distance W_c between the distribution of input p_X and a decoder $p_{X'}$, and is regularized with the sliced-Wasserstein distance SW_c between the distribution of encoded training samples p_Z and, in our experiments, a uniform distribution in the embedding space q_Z:

$$arg\min_{\phi,\theta} \quad W_c(p_X, p_{X'}) + \lambda SW_c(p_Z, q_Z), \tag{1}$$

where ϕ and θ are the parameters of probabilistic encoder and decoder respectively.

Finally, we consider *VAE* and *β-VAE* in our autoencoder study for feature extraction. In the case of VAE [19], variational lower bound is:

$$\mathcal{L}(\theta, \phi; x) = -D_{KL}(q_\phi(z|x)||p_\theta(z)) + \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)], \tag{2}$$

assuming that the prior $p_\theta(z)$ is a unit Gaussian distribution $\mathcal{N}(0, I)$ and the approximate posterior $q_\phi(z|x)$ is a Gaussian $\mathcal{N}(\mu, \sigma^2)$ with parameters μ and σ as outputs of the encoder. The lower bound $-\mathcal{L}(\theta, \phi; x)$ must be minimized w.r.t. ϕ and θ. We can notice in the right hand side the regularization term in the form of KL divergence and the reconstruction term in the form of expected likelihood.

In the case of β-VAE [13], the beta-variational loss can be defined with one Lagrangian multiplier hyperparameter β:

$$\mathcal{L}(\theta, \phi; x, z, \beta) = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \beta D_{KL}(q_\phi(z|x)||p(z)). \tag{3}$$

The smaller values of β, less than one, encourage the expression to be in a form of an autoencoder, with the value $\beta = 1$ being a standard VAE explained above, the greater values restrict the representation capacity of the latent space. We tested the values of β in a range from 0.1 to 100.

Method type	**Convolution**	**Latent dim.**	**Neighborhood hit**	**Silhouette**	**Calinski-Harabasz**	**Davies-Bouldin**
Sparse AE	2D	64	0.9850.005	-0.0060.047	282.766.7	4.21.8
Sparse AE	2D	128	0.9780.005	-0.0240.01	122.848.8	9.22.8
Sparse AE	2D	256	0.9740.004	-0.040.044	90.970.3	15.112.6
SWAE	2D	2	**0.9940.001**	0.3050.049	1131.1145.4	1.50.2
SWAE	2D	4	0.9410.005	-0.0630.034	27.113.5	15.19.1
SWAE	2D	8	0.9740.004	0.040.035	431.3179.3	2.50.7
SWAE	2D	16	0.9650.005	-0.0060.043	302.178.9	4.71.5
SWAE	2D	32	0.9570.006	-0.0470.032	150.178.5	5.12.3
SWAE	2D	64	0.9560.005	-0.0610.061	109.9132.3	9.84.8
SWAE	2D	128	0.9740.003	0.0510.077	416.6223.5	2.60.6
VAE	2D	128	0.9930.001	0.1310.068	375.8170.8	3.52.6
VAE	2D	256	**0.9940.001**	0.0990.089	305.7190.3	4.41.9
β(2)-VAE	2D	32	0.9910.003	0.1950.055	560.7177.7	2.51.0
β(2)-VAE	2D	64	0.9920.001	0.1410.106	426.8141.0	2.91.2
β(2)-VAE	2D	128	0.9930.001	0.1130.062	359.6123.3	3.20.8
β(2)-VAE	2D	256	0.9920.001	0.1710.064	453.885.5	3.11.4
β(4)-VAE	2D	128	0.9860.003	0.2260.049	794.4264.5	2.31.3
β(4)-VAE	2D	256	0.9780.006	0.2580.095	1277.7610.9	1.40.3
β(6)-VAE	2D	128	0.9730.004	0.2870.028	1135.1266.5	1.30.3
β(6)-VAE	2D	256	0.9210.027	0.2090.039	986.0246.0	1.60.4
β(8)-VAE	2D	32	0.9470.009	**0.3170.036**	**1504.8399.1**	1.30.6
β(8)-VAE	2D	64	0.9640.01	0.2870.083	1284.5491.7	1.50.6
β(8)-VAE	2D	128	0.9120.027	0.2720.036	1278.4154.8	1.30.2
β(8)-VAE	2D	256	0.8670.043	0.2470.03	1284.8206.5	1.40.2
β(10)-VAE	2D	128	0.8880.028	0.2920.029	1382.1236.7	1.40.3
β(10)-VAE	2D	256	0.7750.028	0.2110.043	1108.4231.3	1.40.5
β(20)-VAE	2D	32	0.3460.269	0.0390.132	282.0629.1	76.751.4
β(20)-VAE	2D	64	0.8140.014	0.2570.022	1303.7218.9	**1.20.3**
β(20)-VAE	2D	128	0.7420.037	0.2250.041	1062.9183.9	1.30.2
β(20)-VAE	2D	256	0.5880.037	0.1020.03	638.398.6	2.20.7
β(100)-VAE	2D	128	0.2280.003	-0.0270.004	2.62.7	64.342.1
β(100)-VAE	2D	256	0.2270.003	-0.0220.005	0.80.4	118.4102.8
Baseline	–	–	0.977±0.008	-0.061±0.033	90.7±63.1	8.8±5.8

Table 1: Metrics scores of all models performing feature extraction on the MCMC ensemble.

Method type	**Convolution**	**Latent dim.**	**Neighborhood hit**	**Silhouette**	**Calinski-Harabasz**	**Davies-Bouldin**
Sparse AE	2D	64	0.6770.006	-0.0440.022	735.7284.5	4.62.1
Sparse AE	2D	128	0.6710.017	-0.0630.029	642.0193.7	3.70.8
Sparse AE	2D	256	0.6730.01	-0.0520.015	657.6131.5	6.85.0
VAE	2D	128	0.6330.001	-0.0250.014	617.983.8	5.91.8
VAE	2D	256	0.590.01	-0.0280.01	608.295.1	4.71.1
β(4)-VAE	2D	128	0.4830.014	-0.0610.011	549.719.0	7.33.3
AE	3D	64	0.7750.008	-0.0910.011	408.484.8	5.72.7
AE	3D	128	0.7720.01	-0.1290.016	353.7115.2	5.71.0
AE	3D	256	**0.7820.009**	-0.0850.015	458.848.2	4.90.9
Sparse AE	3D	256	0.770.006	-0.1110.02	339.1100.7	6.33.3
SWAE	3D	32	0.7730.014	-0.0830.008	655.6104.0	4.71.9
SWAE	3D	64	0.7730.01	-0.1010.015	583.685.7	5.31.4
SWAE	3D	128	0.7620.018	-0.0930.026	552.8152.3	4.71.1
β(0.1)-VAE	3D	256	0.7230.011	**-0.0050.03**	**831.3224.1**	**3.60.3**
VAE	3D	256	0.5920.024	-0.020.016	797.258.9	9.62.4
β(2)-VAE	3D	256	0.5140.011	-0.050.014	640.680.3	10.52.2
β(4)-VAE	3D	256	0.4210.012	-0.080.017	583.960.8	15.73.7
β(10)-VAE	3D	256	0.3010.004	-0.0940.011	400.221.3	15.63.5
Baseline	–	–	0.6410.011	-0.1120.029	449.3156.8	9.23.9

Table 2: Metrics scores of all models performing feature extraction on the Drop Dynamics ensemble.

Layer type	**Output Shape**	**Details**
Input	(batch size, 3, h, w, 1)	height = h, width = w
Conv3D	(batch size, 1, h/2, w/2, 64)	kernel size = (3, 3, 3), stride = (3, 2, 2)
Conv3D	(batch size, 1, h/4, w/4, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Conv3D	(batch size, 1, h/8, w/8, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Conv3D	(batch size, 1, h/16, w/16, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Flatten	(batch size, 1, (h/16) · (w/16) · 64)	reshape before dense layer
Dense	(batch size, num. of units)	first dense layer of encoder
AE: Dense	(batch size, latent dimension)	second dense layer
VAE: Dense $(\mu, \log\sigma)$	(batch size, latent dimension)	two parallel dense layers for VAE
VAE: Sample z	(batch size, latent dimension)	reparameterization trick for VAE
Dense	(batch size, 1, (h/16) · (w/16) · 64)	first dense layer of decoder
Reshape	(batch size, 1, (h/16) · (w/16) · 64)	reshape before deconvolutions
Conv3DTranspose	(batch size, 1, h/8, w/8, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Conv3DTranspose	(batch size, 1, h/4, w/4, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Conv3DTranspose	(batch size, 1, h/2, w/2, 64)	kernel size = (1, 3, 3), stride = (1, 2, 2)
Conv3DTranspose	(batch size, 1, h, w, 64)	kernel size = (3, 3, 3), stride = (3, 2, 2)
Conv3DTranspose	(batch size, 1, h, w, 1)	kernel size = (3, 3, 3), stride = (1, 1, 1)

Table 3: 3D AE/VAE architecture used on Drop Dynamics ensemble. The difference is highlighted in italics in the bottleneck.