SUPPLEMENTARY MATERIAL FOR SUPPRESSING OUTLIER RECONSTRUCTION IN AUTOENCODERS FOR OUT-OF-DISTRIBUTION DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper presents the supplementary material for the paper Suppressing Outlier Reconstruction in Autoencoders for Out-of-Distribution Detection.

1 DERIVATION FOR THE LOG LIKELIHOOD GRADIENT

Here, we present the derivation for the gradient of log likelihood (Eq.(2) in the main manuscript).

$$\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\nabla_\theta \log p_\theta(\mathbf{x})] = -\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\nabla_\theta E(\mathbf{x})] + \mathbb{E}_{\mathbf{x}'\sim p_\theta(\mathbf{x})}[\nabla_\theta E(\mathbf{x}')]. \tag{1}$$

The gradient expression is well-known in energy-based model literature, but we provide its derivation to make the paper self-contained.

Recall that the model density function $p_\theta(\mathbf{x})$ is defined from an energy function $E_\theta(\mathbf{x})$ using Gibbs distribution: $p_\theta(\mathbf{x}) = \exp(-E_\theta(\mathbf{x}))/\Omega_\theta$ for the normalization constant $\Omega_\theta = \int \exp(-E_\theta(\mathbf{x}))\mathrm{d}\mathbf{x} < \infty$. The gradient for the log likelihood of a single datum $\mathbf{x}$ is given as follows:

$$\begin{aligned}
\nabla_\theta \log p_\theta(\mathbf{x}) &= -\nabla_\theta E(\mathbf{x}) - \nabla_\theta \log \Omega_\theta & (2)\\
&= -\nabla_\theta E(\mathbf{x}) - \frac{\nabla_\theta \Omega_\theta}{\Omega_\theta} & (3)\\
&= -\nabla_\theta E(\mathbf{x}) - \frac{1}{\Omega_\theta}\nabla_\theta \int \exp(-E_\theta(\mathbf{x}'))\mathrm{d}\mathbf{x}' & (4)\\
&= -\nabla_\theta E(\mathbf{x}) - \frac{1}{\Omega_\theta}\int \nabla_\theta \exp(-E_\theta(\mathbf{x}'))\mathrm{d}\mathbf{x}' & (5)\\
&= -\nabla_\theta E(\mathbf{x}) + \int \frac{1}{\Omega_\theta}\exp(-E_\theta(\mathbf{x}'))\nabla_\theta E_\theta(\mathbf{x}')\mathrm{d}\mathbf{x}' & (6)\\
&= -\nabla_\theta E(\mathbf{x}) + \mathbb{E}_{\mathbf{x}'\sim p_\theta(\mathbf{x}')}\left[\nabla_\theta E_\theta(\mathbf{x}')\right]. & (7)
\end{aligned}$$

Taking the expectation over the data density $p(\mathbf{x})$,

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}[\nabla_\theta \log p_\theta(\mathbf{x})] &= -\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\nabla_\theta E(\mathbf{x})\right] + \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\mathbb{E}_{\mathbf{x}'\sim p_\theta(\mathbf{x}')}\left[\nabla_\theta E_\theta(\mathbf{x}')\right]\right] & (8)\\
&= -\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}\left[\nabla_\theta E(\mathbf{x})\right] + \mathbb{E}_{\mathbf{x}'\sim p_\theta(\mathbf{x}')}\left[\nabla_\theta E_\theta(\mathbf{x}')\right]. & (9)
\end{aligned}$$

Thus, we obtain Eq.(2) of the main manuscript.

2 DATASETS

Here, we present details on the datasets used in the experiments.

In general, we follow Serrà et al. (2020) in preparing datasets. For MNIST, FashionMNIST, SVHN, and CIFAR-10, we use the predefined test splits as the test sets and randomly select 10% from the train splits as validation set. For CelebA, we randomly assign 10% of images for validation and test sets. Each image in CelebA is cropped in 160×160 and then resized into 32×32. When MNIST and Fashion MNIST are needed to be fed to a network trained on CIFAR-10, an image is resized into 32×32. The training set of ImageNet32 (Oord et al., 2016) is the randomly selected 80% from the

Table 1: Summary of dataset statistics.

Dataset	Original Shape	Training	Validation	Test
Constant (Synthetic)	-	-	4,000	4,000
MNIST	1×28×28	54,000	6,000	10,000
FashionMNIST	1×28×28	54,000	6,000	10,000
SVHN	3×32×32	65,930	7,327	20,632
CIFAR-10	3×32×32	45,000	5,000	10,000
CelebA	3×178×218	162,079	20,260	20,260
ImageNet32	3×32×32	1,024,919	256229	49999
Noise (Synthetic)	3×32×32	-	4,000	4,000
HalfMNIST (Synthetic)	1×28×28	-	-	10,000
ChimeraMNIST (Synthetic)	1×28×28	-	-	10,000

original train split, and the rest 20% are used as the validation set. We use the original validation split as the test set in our experiment. All pixel values are normalized to interval $[0, 1]$.

Constant images are generated by first drawing a random number from interval $[0, 1]$, and then making a 1×28×28 or 3×32×32 array filled with the drawn number. An image in Noise dataset is created by randomly generating each pixel value from interval $[0, 1]$.

3 Implementation Detail

Here, we present details on implementing EBAE. The sample code is provided in the supplementary material.

Network architectures In all autoencoder-based methods, including EBAE, AE, DAE, WAE, VAE and DAGMM, we use the same architectures for encoder and decoder for fair comparison. The network architectures are described in Table 2, where we use the following notations to denote operations in networks.

- Conv2d(`in`, `out`, `kernel_size`, `stride`): A 2D convolutional operation with bias.
- ConvT2d(`in`, `out`, `kernel_size`, `stride`): A 2D transposed convolutional operation with bias.
- ReLU: A rectified linear unit activation function. $y = \max 0, x$.
- Sigmoid: A sigmoid activation function. $y = 1/(1 + \exp(-x))$.
- MaxPool(`kernel_size`): A max-pooling operator with the window of `kernel_size`×`kernel_size` and the stride of `kernel_size`.
- Bilinear(`scale_factor`): A bilinear upscaling operation that upsamples an input array into an `scale_factor` times larger array.
- FC(`in`, `out`): A fully-connected layer.
- Res(`in`, `hidden`, `out`): A residual block composed of two two fully-connected layers F_1 = FC(`in`, `hidden`) and F_2 = FC(`hidden`, `out`). It operates as $y = x + F_2(\text{ReLU}(F_1(\text{ReLU}(x))))$ given an input x and an output y.
- ResAtt(`in`, `hidden`, `out`): A residual attention block which has two residual blocks R_1 = Res(`in`, `hidden`, `out`) and R_2 = Res(`in`, `hidden`, `out`) and operates as $y = x + R_1(x) \odot \text{Sigmoid}(R_2(x))$, where $\odot$ denotes an element-wise product.

We devise two network architectures, Arch28 and Arch32, tailored for MNIST-like inputs and CIFAR-10-like inputs, respectively.

The optimal $D_{\mathbf{z}}$ is searched among $[2, 4, 8, 16, 32, 64, 128, 256]$. The best parameter is selected according to AUC of classifying a separate OOD dataset, as described in the main manuscript.

For ImageNet32 experiment, we use ResNet-based architecture for an encoder and a decoder. The encoder is the same as one used in Du & Mordatch (2019). The last representation layer is average-pooled. The decoder has the same architecture to the generator used in Miyato et al. (2018).

Table 2: Network architectures for autoencoders.

	Arch28 (For 1×28×28 input)	Arch32 (For 3×32×32 input)
Encoder	Conv2d(1, 32, 3, 1)-ReLU- Conv2d(32, 64, 3, 1)-ReLU-MaxPool(2)- Conv2d(64, 64, 3, 1)-ReLU- Conv2d(64, 128, 3, 1)-ReLU-MaxPool(2)- Conv2d(128, 1024, 4, 1)-ReLU- FC(1024, $D_{\mathbf{z}}$)	Conv2d(3, 32, 4, 2)-ReLU- Conv2d(32, 64, 4, 2)-ReLU- Conv2d(64, 128, 4, 2)-ReLU- Conv2d(128, 256, 2, 2)-ReLU- Conv2d(256, $D_{\mathbf{z}}$, 1, 1)-ReLU- ResAtt($D_{\mathbf{z}}$, 1024, $D_{\mathbf{z}}$)
Decoder	ConvT2d($D_{\mathbf{z}}$, 128, 4, 1)-ReLU-Bilinear(2)- ConvT2d(128, 64, 3, 1)-ReLU- ConvT2d(64, 64, 3, 1)-ReLU-Bilinear(2)- ConvT2d(64, 32, 3, 1)-ReLU- ConvT2d(32, 1, 4, 1)-Sigmoid	ConvT2d($D_{\mathbf{z}}$, 256, 6, 1)-ReLU- ConvT2d(256, 128, 4, 2)-ReLU- ConvT2d(128, 64, 4, 2)-ReLU- ConvT2d(64, 3, 3, 1)-Sigmoid

Sampling parameters For the latent chain, the step size $\lambda_{\mathbf{z}}$ is 0.4 for Arch28 and 2.0 for Arch32. The standard deviation of the noise $\epsilon_{\mathbf{z}}$ is set to 0.05 and 0.02 for Arch28 and Arch32, respectively. The length of a latent chain is 10 for Arch32 and 50 for Arch32. We use the sample replay buffer as described in Du & Mordatch (2019) with the buffer size of 10,000 and the replay ratio of 95%. In other words, a starting point of a latent chain is given as a random point for probability of 5% or a previous sample for probability of 95%.

In the visible chain, the step size $\lambda_{\mathbf{x}}$ is 20 and the gradient $\nabla_{\mathbf{x}} \log p_\theta(\mathbf{x})$ is clipped at 0.01, similarly to Du & Mordatch (2019). The length of a visible chain is 50. We let the standard deviation of $\epsilon_{\mathbf{x}}$ decay as the visible chain proceeds: std($\epsilon_{\mathbf{x}}$) $= 0.05/(1 +$ step) for step $= 0, \cdots, 49$.

Learning parameters We use Adam optimizer Kingma & Ba (2014) with learning rate 1×10^{-5}. Also, for EBAE, we regularize L_2 norm of the encoder with a coefficient of 1×10^{-4}.

Pre-training of EBAE is done for 100 epochs for MNIST and 120 epochs for CIFAR-10. The weight with the minimal validation loss is retrieved and used as the initial weights for EBAE training. EBAE training is done for 50 epochs.

Implementing baselines PixelCNN++ Salimans et al. (2017) is implemented based on an open-source code base[1]. We use set parameters as `nr_resnet`=5 and `nr_filters`=80. Input values are scales to $[-1, 1]$ only when running PixelCNN++.

Glow Kingma & Dhariwal (2018) is also implemented based on the open-source repository[2]. We set $K = 12$, $L = 1$, $hidden\_channels = 64$.

DAGMM Zong et al. (2018) is implemented based on the public repository[3]. We failed to train it on CIFAR-10 such that it produces AUC of 1.0 for Noise dataset, and therefore we exclude the result from the Table 2 of the main manuscript.

WAE-MMD is implemented. We use median heuristic to determine the kernel parameter of RBF kernel. Regularization coefficient is searched between 0.001 and 0.1

For DAE, we use Gaussian noise with standard deviation of 0.3.

4 Experimental Detail for Figure 2

We first define two 2D uniform distributions. The first distribution spans $[-2, -1]$ in the horizontal coordinate and $[-0.5, 0.5]$ in the vertical coordinate. Likewise, the second distribution spans $[1, 2]$ and $[-0.5, 0.5]$ in the horizontal and the vertical coordinate, respectively. 200 independent samples are drawn from each uniform distribution, resulting in 400 samples for training.

[1]`https://github.com/pclucas14/pixel-cnn-pp`
[2]`https://github.com/chaiyujin/glow-pytorch`
[3]`https://github.com/danieltan07/dagmm`

We build an autoencoder with an encoder, FC(2, 200)-ReLU-FC(200, 200)-ReLU-FC(200, 1)-tanh, and a decoder, FC(1, 200)-ReLU-FC(200, 200)-ReLU-FC(200, 2). The autoencoder is first trained as a conventional autoencoder by minimizing reconstruction error of the training data. The stochastic gradient descent is performed via Adam with the learning rate of 0.001. The training is carried for 200 epochs with batch size of 128.

After training as an autoencoder (and after drawing the first panel in Figure 2 in the main manuscript), the decoder weights are fixed and only encoder is trained via the gradients of EBAE. Both the latent chain and the visible chain have 10 steps with step size of 0.1. During the latent chain, if the state $\mathbf{z}$ of the chain moves outside of the interval $[-1, 1]$, the state is pushed back to the interval.

5 Ablation Study

To demonstrate the contribution of learning techniques and components used in EBAE, we train EBAEs with altered configurations on the MNIST 9 detection task. Following MNIST hold-out experiment in Section 6.2 of the main manuscript, we use digit 9 in MNIST as the anomaly class and use digits from 0 to 8 as the normal class. We mainly investigate the effect of three components: latent LMC chain, sample replay buffer, and spherical latent space. Table 3 shows different configurations experimented and their anomaly detection performance in AUC. Experiment No.1 represents the original EBAE configuration used in the experiments in the main manuscript.

Table 3: Ablation results

No.	Latent chain length	Sample replay buffer	Spherical latent space	AUC
1	10	Yes	Yes	0.957
2	**0**	**No**	Yes	0.839
3	**50**	Yes	Yes	0.944
4	10	**No**	Yes	0.936
5	**50**	**No**	Yes	0.952
6	10	Yes	**No**	0.662

- The use of latent chain is substantially beneficial, as shown from the comparison between No.1 and No.2.
- A longer latent chain is needed for a better result. Comparing No.2, No.4, and No.5, the performance increases monotonically with the chain length. Note that longer chain length requires proportionally longer time in training.
- The sample replay buffer helps to reduce the length of the latent chain required in training. We see that with the sample replay buffer, the latent chain with 10 steps (No.1) can achieve performance comparable to that of 50 latent chain steps (No.5). In other words, using sample replay buffer may reduce the training time up to five times without the loss of performance.
- The gain from a longer latent chain saturates. Especially when a sample replay buffer is used, a chain longer than a certain threshold does not improve the performance (No.1 & No.3).
- The use of spherical latent space is essential in achieving a good performance.

6 Computational Properties of EBAE

The inference step of EBAE is as fast as that of a conventional autoencoder, requiring no more than a single forward pass. On the other hand, VAE requires multiple passes of its decoder to reliably evaluate its likelihood and reconstruction error.

The Langevin Monte Carlo (LMC) is the most time consuming procedure in EBAE training. A single draw of sample involves dozens of transition steps, and each transition step requires multiple backward passes one of which is comparable to a single training step for an autoencoder. However, the number of transition step required is less than 100 in our experiments, and it is feasible to run it on a modern hardware. Training of EBAE takes about 5 hours for MNIST, and 10 hours for CIFAR-10 on a single Tesla V100 GPU.

REFERENCES

Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alche-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 3608–3618. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/8619-implicit-generation-and-modeling-with-energy-based-models.pdf`.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pp. 10215–10224, 2018.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

Joan Serrà, David Álvarez, Vicenç Gómez, Olga Slizovskaia, José F. Núñez, and Jordi Luque. Input complexity and out-of-distribution detection with likelihood-based generative models. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SyxIWpVYvr`.

Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=BJJLHbb0-`.