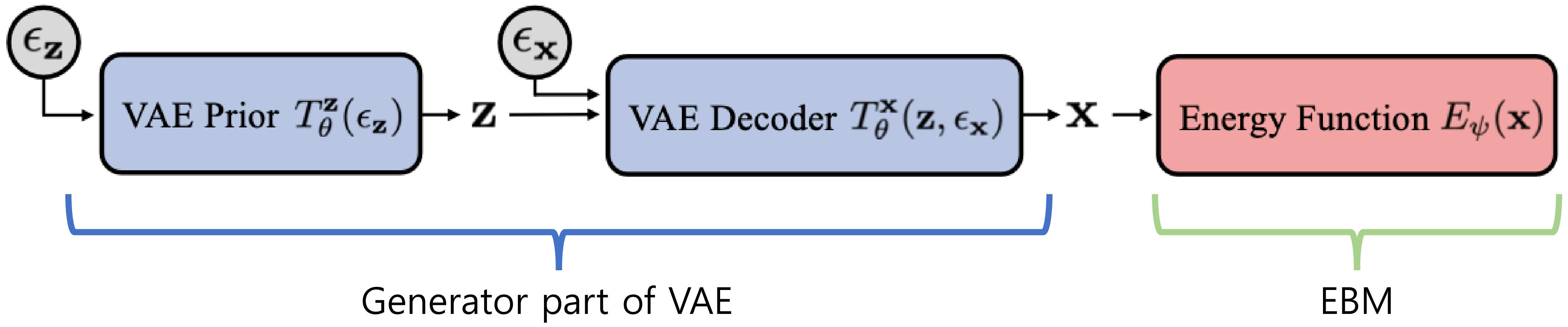
VAEBM: A Symbiosis between Variational Autoencoders and Energy-based Models (ICLR May 2021)

2021 VILab Summer Seminar

Daehyun Kim

About VAEBM



$\epsilon_{\mathbf{z}}$
VAE Prior $T_\theta^{\mathbf{z}}(\epsilon_{\mathbf{z}})$
$\mathbf{z}$
$\epsilon_{\mathbf{x}}$
VAE Decoder $T_\theta^{\mathbf{x}}(\mathbf{z}, \epsilon_{\mathbf{x}})$
$\mathbf{x}$
Energy Function $E_\psi(\mathbf{x})$
Generator part of VAE
EBM

- Make generative model which can make image very close to given dataset.

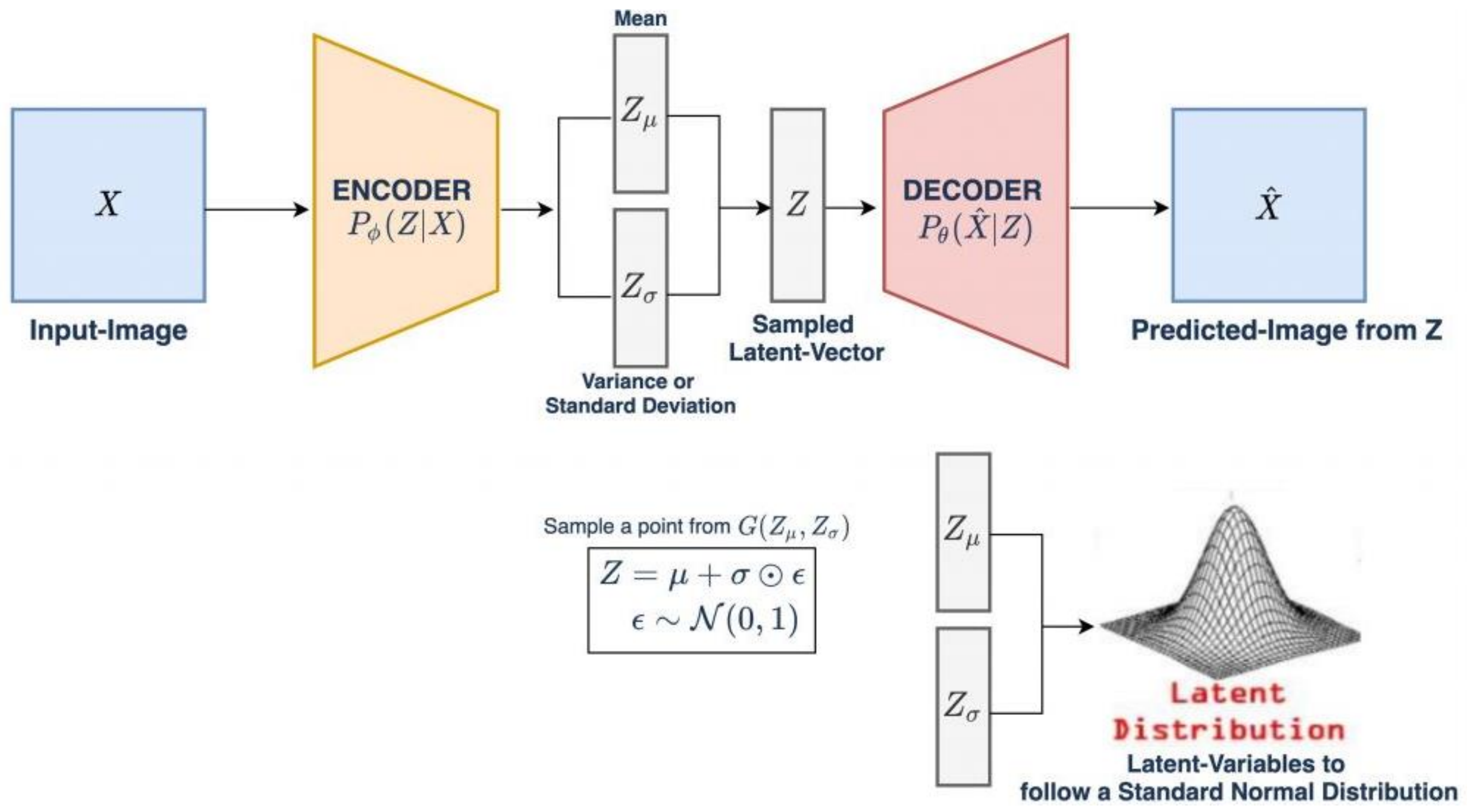
- VAEBM = VAE + EBM
  (VAE: variational autoencoder, EBM: energy-based model)

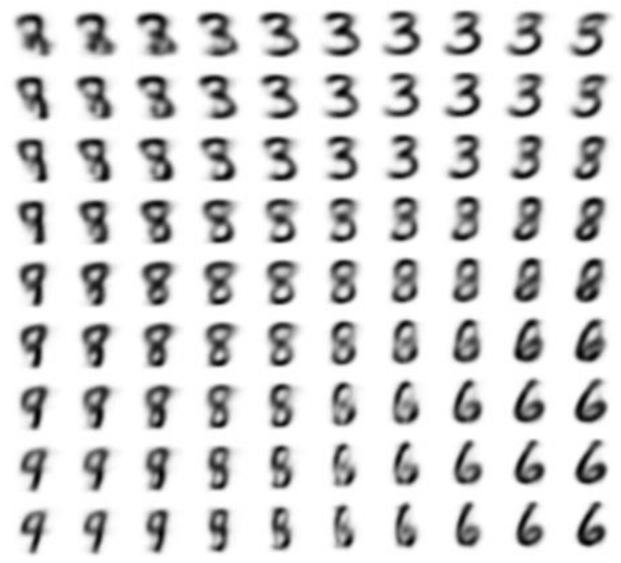
- ϵ_z, ϵ_x are noise. (The model take generator from pretrained VAE)

Content

Background

VAE (1)


Mean
Z_μ
X
ENCODER
$P_\phi(Z|X)$
Z
DECODER
$P_\theta(\hat{X}|Z)$
$\hat{X}$
Z_σ
Input-Image
Sampled Latent-Vector
Predicted-Image from Z
Variance or Standard Deviation
Sample a point from $G(Z_\mu, Z_\sigma)$
$Z = \mu + \sigma \odot \epsilon$
$\epsilon \sim \mathcal{N}(0,1)$
Z_μ
Z_σ
Latent Distribution
Latent-Variables to follow a Standard Normal Distribution



VAE (2)

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{\mathrm{KL}}\left[q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})\right]$$

The main idea of VAE is maximizing probability of generated image existence in data distribution.

- Strength
  - Computationally efficient
  - Fast traverse of the data manifold
- Weakness
  - Tend to assign high probability to regions in data space that have low probability under the data distribution.

EBM (1) -IGEBM

- Goal is same with VAE's. (generating samples)
- Need to know
  1. Energy function(E(x)) is a model and E(x) is a scalar.
  2. E(fake) > E(true)
  3. Energy function has Boltzman distribution.
  4. Use Langevin dynamics to generate sample images.

Boltzmann distribution

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

Langevin dynamics (method of MCMC)

$$\tilde{\mathbf{x}}^k = \tilde{\mathbf{x}}^{k-1} - \frac{\lambda}{2}\nabla_{\mathbf{x}} E_\theta(\tilde{\mathbf{x}}^{k-1}) + \omega^k,\ \omega^k \sim \mathcal{N}(0, \lambda)$$

Algorithm 1 Energy training algorithm

```
**Input:** data dist. $p_D(\mathbf{x})$, step size $\lambda$, number of steps $K$
$\mathcal{B} \leftarrow \varnothing$
while not converged do
    $\mathbf{x}_i^+ \sim p_D$
    $\mathbf{x}_i^0 \sim \mathcal{B}$ with 95% probability and $\mathcal{U}$ otherwise

    ▷ Generate sample from $q_\theta$ via Langevin dynamics:

    for sample step $k = 1$ to $K$ do
        $\tilde{\mathbf{x}}^k \leftarrow \tilde{\mathbf{x}}^{k-1} - \nabla_{\mathbf{x}} E_\theta(\tilde{\mathbf{x}}^{k-1}) + \omega, \quad \omega \sim \mathcal{N}(0, \sigma)$
    end for
    $\mathbf{x}_i^- = \Omega(\tilde{\mathbf{x}}_i^k)$

    ▷ Optimize objective $\alpha\mathcal{L}_2 + \mathcal{L}_{ML}$ wrt $\theta$:
    $\Delta\theta \leftarrow \nabla_\theta \frac{1}{N}\sum_i \alpha(E_\theta(\mathbf{x}_i^+)^2 + E_\theta(\mathbf{x}_i^-)^2) + E_\theta(\mathbf{x}_i^+) - E_\theta(\mathbf{x}_i^-)$
    Update $\theta$ based on $\Delta\theta$ using Adam optimizer

    $\mathcal{B} \leftarrow \mathcal{B} \cup \tilde{\mathbf{x}}_i$
end while
```

EBM (2) -IGEBM

- Strength
  - Model the unnormalized data density by assigning low energy to high-probability regions in the data space
  - Requires almost no restrictions on network architectures.
  - Robustness
  - Performance of OOD is high between likelihood models
- Weakness
  - At training and sampling requires MCMC which is computationally expensive.

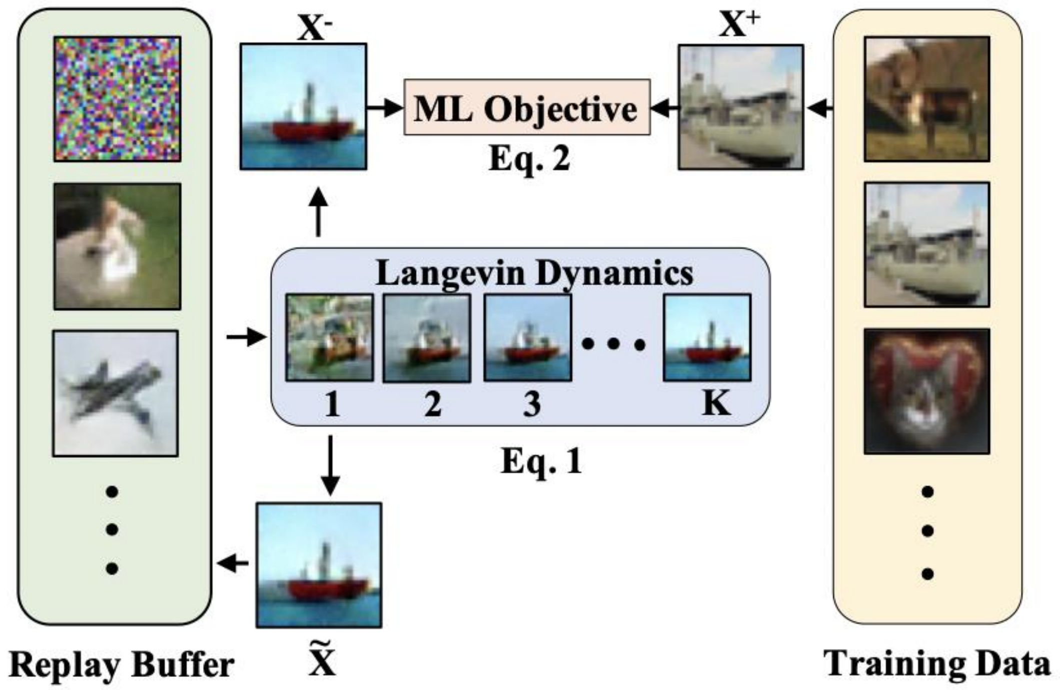
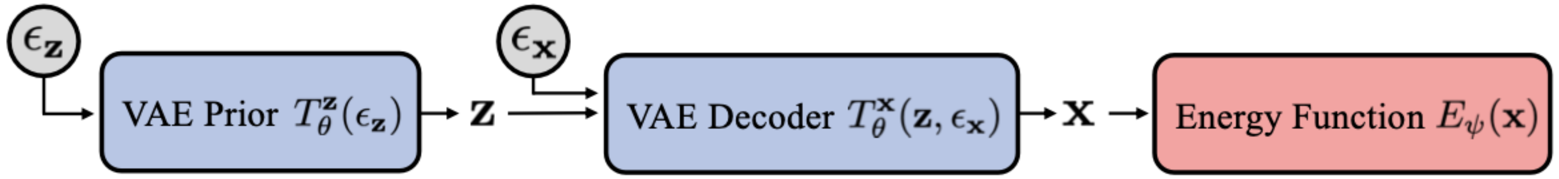


Figure 1: Overview of our method and the interrelationship of the components involved.

VAEBM

Structure

θ: *parameter of VAE*
ψ: *parameter of EBM*


$\epsilon_{\mathbf{z}}$
VAE Prior $T_\theta^{\mathbf{z}}(\epsilon_{\mathbf{z}})$
$\mathbf{z}$
$\epsilon_{\mathbf{x}}$
VAE Decoder $T_\theta^{\mathbf{x}}(\mathbf{z}, \epsilon_{\mathbf{x}})$
$\mathbf{x}$
Energy Function $E_\psi(\mathbf{x})$

The goal is also same with VAE. VAEBM works to maximizing probability of generated image.

Probability at VAE: $p_\theta(x)$

Probability at EBM: $p_\psi(x) = \exp\left(-E_\psi(x)\right)/Z_\psi$

Total probability: $h_{\psi,\theta}(x) = p_\theta(x)\exp\left(-E_\psi(x)\right)/Z_{\psi,\theta}$ (z is marginalized)

$$h_{\psi,\theta}(\mathbf{x}, \mathbf{z}) = \tfrac{1}{Z_{\psi,\theta}} p_\theta(\mathbf{x}, \mathbf{z}) e^{-E_\psi(\mathbf{x})}$$

$$h_{\psi,\theta}(\mathbf{x}) = \frac{1}{Z_{\psi,\theta}} \int p_\theta(\mathbf{x}, \mathbf{z}) e^{-E_\psi(\mathbf{x})} d\mathbf{z} = \frac{1}{Z_{\psi,\theta}} p_\theta(\mathbf{x}) e^{-E_\psi(\mathbf{x})}.$$

Object function

θ: *parameter of VAE*
ψ: *parameter of EBM*

- By training, need to maximize the marginal log-likelihood.

$$\log h_{\psi,\theta}(\mathbf{x}) = \log p_\theta(\mathbf{x}) - E_\psi(\mathbf{x}) - \log Z_{\psi,\theta} \tag{5}$$

$$\geq \underbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))}_{\mathcal{L}_{\text{vae}}(\mathbf{x},\theta,\phi)} \underbrace{-E_\psi(\mathbf{x}) - \log Z_{\psi,\theta}}_{\mathcal{L}_{\text{EBM}}(\mathbf{x},\psi,\theta)}, \tag{6}$$

- In the case of L_{VAE} there's no problem at doing backpropagation.
- In the case of L_{EBM}, when doing backpropagation training becomes computationally expensive.

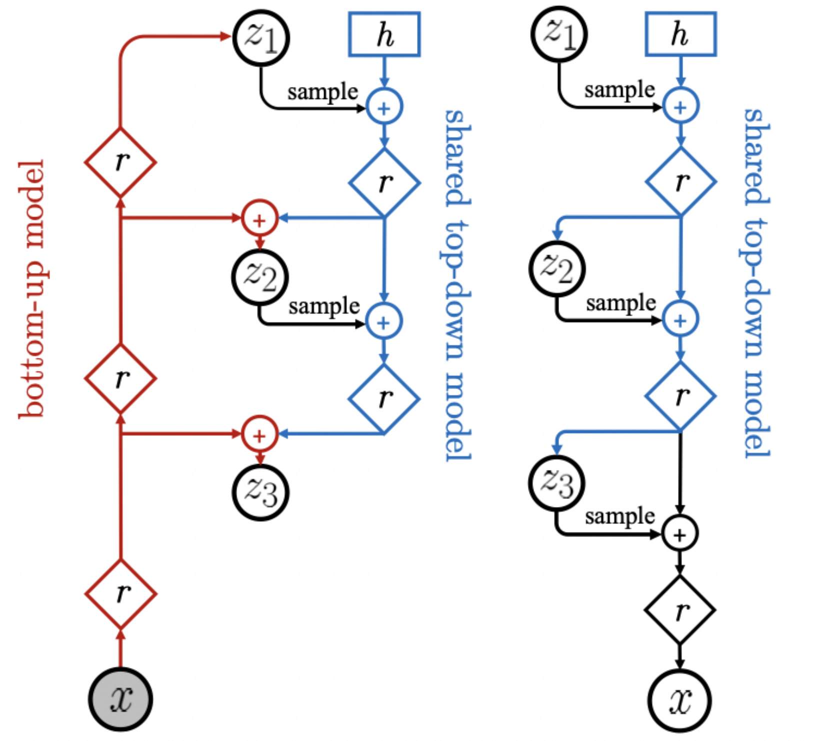
Gradient of $\log Z_{\psi,\theta}$: $\partial_\psi \log Z_{\psi,\theta} = \mathbb{E}_{\mathbf{x}\sim h_{\psi,\theta}(\mathbf{x},\mathbf{z})}\left[-\partial_\psi E_\psi(\mathbf{x})\right]$ and $\boxed{\partial_\theta \log Z_{\psi,\theta} = \mathbb{E}_{\mathbf{x}\sim h_{\psi,\theta}(\mathbf{x},\mathbf{z})}\left[\partial_\theta \log p_\theta(\mathbf{x})\right]}$

- To find the second term of the gradient, need to do sampling from VAEBM using MCMC.

(can check details in Appendix. A)

Two-stage training (1)

- The first stage
  - Train the VAE model in VAEBM by maximizing $L_{VAE}(x, \theta, \phi)$.
  - Parameters θ and ϕ are trained using the reparameterized trick. (NVAE)
- After the first stage, $\partial_\theta \log Z_{\psi,\theta} = \mathbb{E}_{\mathbf{x}\sim h_{\psi,\theta}(\mathbf{x},\mathbf{z})}\left[\partial_\theta \log p_\theta(\mathbf{x})\right]$ becomes zero, cause $p_\theta(x)$ will be fixed.



(a) Bidirectional Encoder (b) Generative Model

Figure 2: The neural networks implementing an encoder $q(\boldsymbol{z}|\boldsymbol{x})$ and generative model $p(\boldsymbol{x}, \boldsymbol{z})$ for a 3-group hierarchical VAE. $\diamond r$ denotes residual neural networks, $\oplus$ denotes feature combination (e.g., concatenation), and $\boxed{h}$ is a trainable parameter.

Two-stage training (2)

θ: *parameter of VAE*
ψ: *parameter of EBM*

- $L_{EBM}(x,\psi,\theta) = -E_{\psi}(x) - logZ_{\psi,\theta}$

$L(\psi) = \mathbb{E}_{\mathbf{x}\sim p_d}\left[\mathcal{L}_{\text{EBM}}(\mathbf{x},\psi,\theta)\right]$ w.r.t. ψ is:

becomes zero

$$\partial_{\psi}L(\psi) = \underbrace{\mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}\left[-\partial_{\psi}E_{\psi}(\mathbf{x})\right]}_{\text{positive phase}} + \underbrace{\mathbb{E}_{\mathbf{x}\sim h_{\psi,\theta}(\mathbf{x},\mathbf{z})}\left[\partial_{\psi}E_{\psi}(\mathbf{x})\right]}_{\text{negative phase}} \boxed{-\mathbb{E}_{x\sim h_{\psi,\theta}(x,z)}[\partial_{\theta} log\, p_{\theta}(x)]}$$

- Positive phase need no sampling to find the value.
- Negative phase need sampling to find the value.

$$\mathbf{z} = T^{\mathbf{z}}_{\theta}(\boldsymbol{\epsilon}_{\mathbf{z}}), \quad \mathbf{x} = T^{\mathbf{x}}_{\theta}(\mathbf{z}(\boldsymbol{\epsilon}_{\mathbf{z}}),\boldsymbol{\epsilon}_{\mathbf{x}}) = T^{\mathbf{x}}_{\theta}(T^{\mathbf{z}}_{\theta}(\boldsymbol{\epsilon}_{\mathbf{z}}),\boldsymbol{\epsilon}_{\mathbf{x}}).$$

$$h_{\psi,\theta}(x,z) = h_{\psi,\theta}(\boldsymbol{\epsilon}_{\mathbf{x}},\boldsymbol{\epsilon}_{\mathbf{z}}) \propto e^{-E_{\psi}(T^{\mathbf{x}}_{\theta}(T^{\mathbf{z}}_{\theta}(\boldsymbol{\epsilon}_{\mathbf{z}}),\boldsymbol{\epsilon}_{\mathbf{x}}))} p_{\boldsymbol{\epsilon}}(\boldsymbol{\epsilon}_{\mathbf{x}},\boldsymbol{\epsilon}_{\mathbf{z}})$$

- By doing MCMC sampling in a standard Normal distribution $p_{\epsilon}(\epsilon_x,\epsilon_z)$, can make sa me result to sampling from $p_{\theta}(x,z)$!!
- This reduces calculation of sampling!!

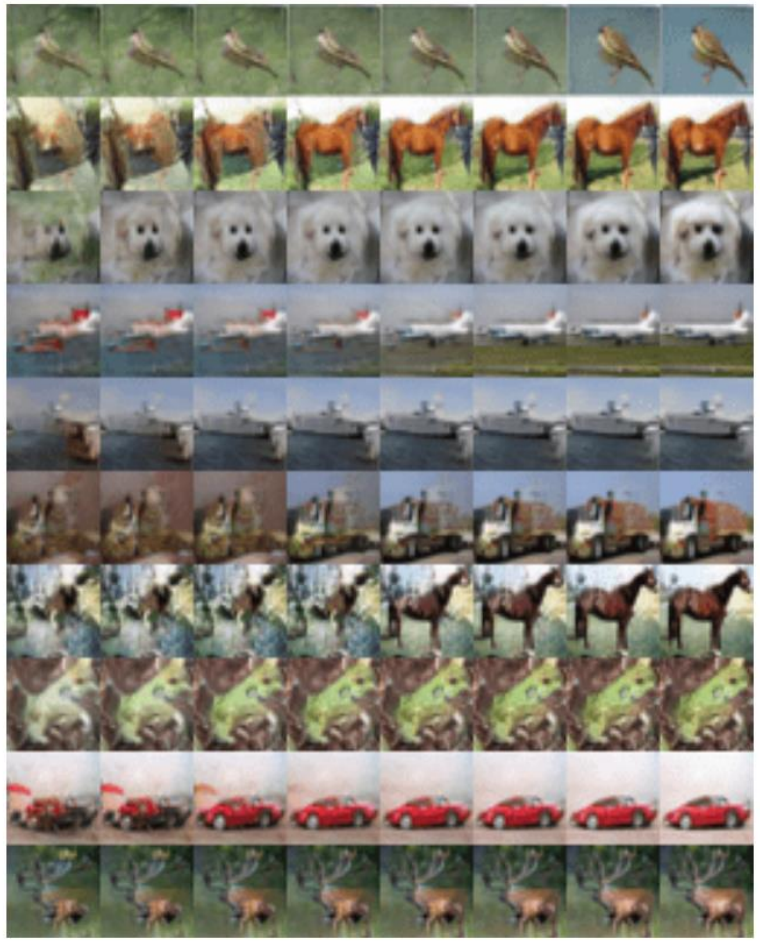
Contribution

1. Propose a new generative model using the product of a VAE generator and an EBM defined in the data space.
2. Show how training this model can be decomposed into training the VAE first, and then training the EBM component.
3. Show how MCMC sampling from VAEBM can be pushed to the VAE's latent space, accelerating sampling.
4. It is a likelihood model which confirms complete mode converge, and show strong out-of-distribution detection performance.

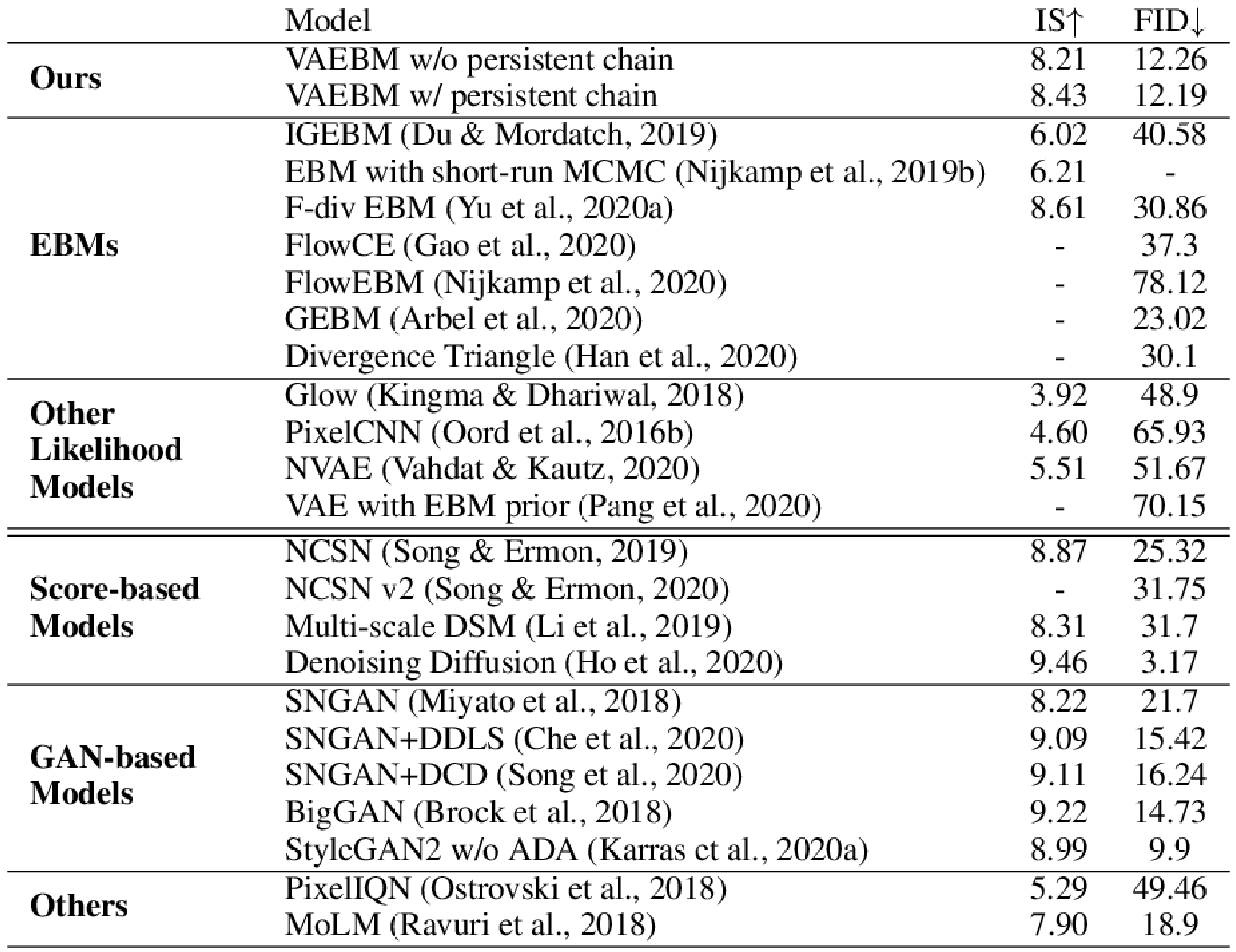
Experiments

Experiments (CIFAR-10)

(a)


(b)

Experiments (CIFAR-10)



	Model	IS↑	FID↓
Ours	VAEBM w/o persistent chain	8.21	12.26
	VAEBM w/ persistent chain	8.43	12.19
EBMs	IGEBM (Du & Mordatch, 2019)	6.02	40.58
	EBM with short-run MCMC (Nijkamp et al., 2019b)	6.21	-
	F-div EBM (Yu et al., 2020a)	8.61	30.86
	FlowCE (Gao et al., 2020)	-	37.3
	FlowEBM (Nijkamp et al., 2020)	-	78.12
	GEBM (Arbel et al., 2020)	-	23.02
	Divergence Triangle (Han et al., 2020)	-	30.1
Other Likelihood Models	Glow (Kingma & Dhariwal, 2018)	3.92	48.9
	PixelCNN (Oord et al., 2016b)	4.60	65.93
	NVAE (Vahdat & Kautz, 2020)	5.51	51.67
	VAE with EBM prior (Pang et al., 2020)	-	70.15
Score-based Models	NCSN (Song & Ermon, 2019)	8.87	25.32
	NCSN v2 (Song & Ermon, 2020)	-	31.75
	Multi-scale DSM (Li et al., 2019)	8.31	31.7
	Denoising Diffusion (Ho et al., 2020)	9.46	3.17
GAN-based Models	SNGAN (Miyato et al., 2018)	8.22	21.7
	SNGAN+DDLS (Che et al., 2020)	9.09	15.42
	SNGAN+DCD (Song et al., 2020)	9.11	16.24
	BigGAN (Brock et al., 2018)	9.22	14.73
	StyleGAN2 w/o ADA (Karras et al., 2020a)	8.99	9.9
Others	PixelIQN (Ostrovski et al., 2018)	5.29	49.46
	MoLM (Ravuri et al., 2018)	7.90	18.9

Experiments (CelebA, LSUN)

(a) CelebA 64

(b) LSUN Church 64

(c) CelebA HQ 256

Experiments (CelebA, LSUN)

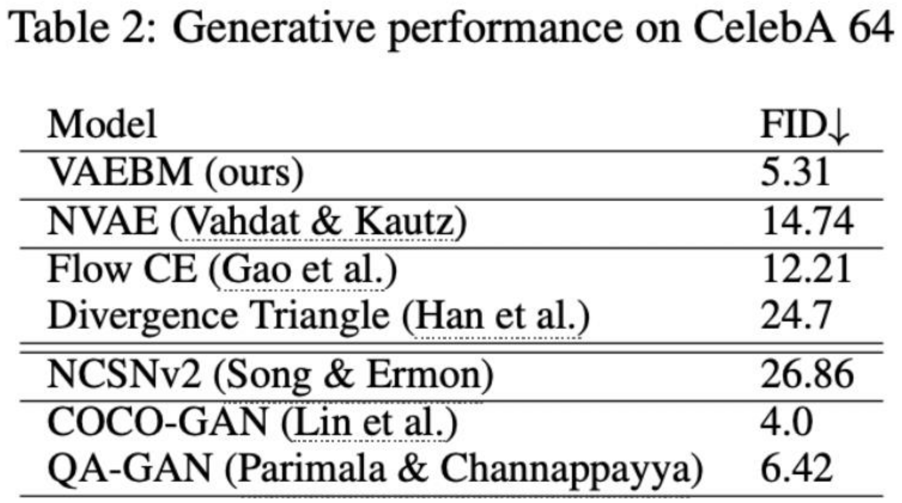


Table 2: Generative performance on CelebA 64

Model	FID↓
VAEBM (ours)	5.31
NVAE (Vahdat & Kautz)	14.74
Flow CE (Gao et al.)	12.21
Divergence Triangle (Han et al.)	24.7
NCSNv2 (Song & Ermon)	26.86
COCO-GAN (Lin et al.)	4.0
QA-GAN (Parimala & Channappayya)	6.42

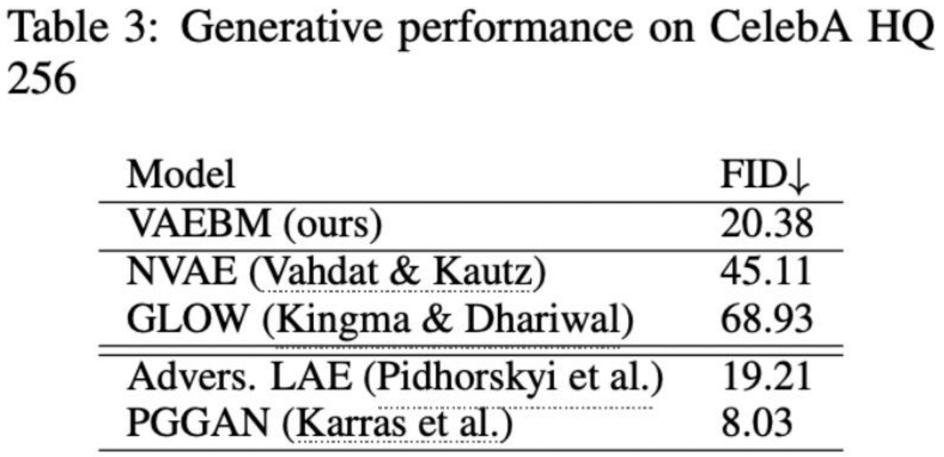
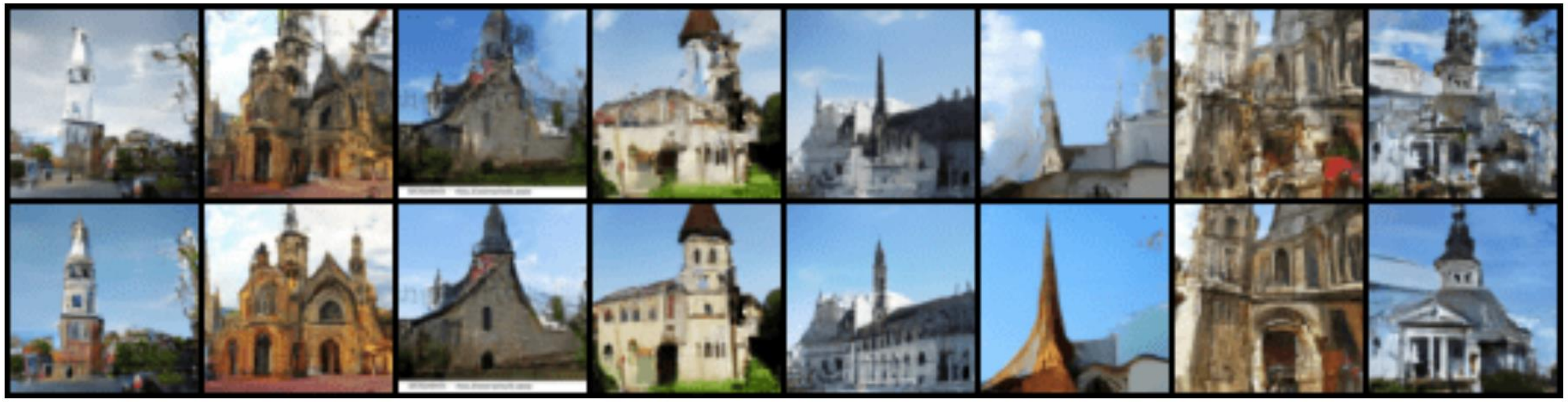
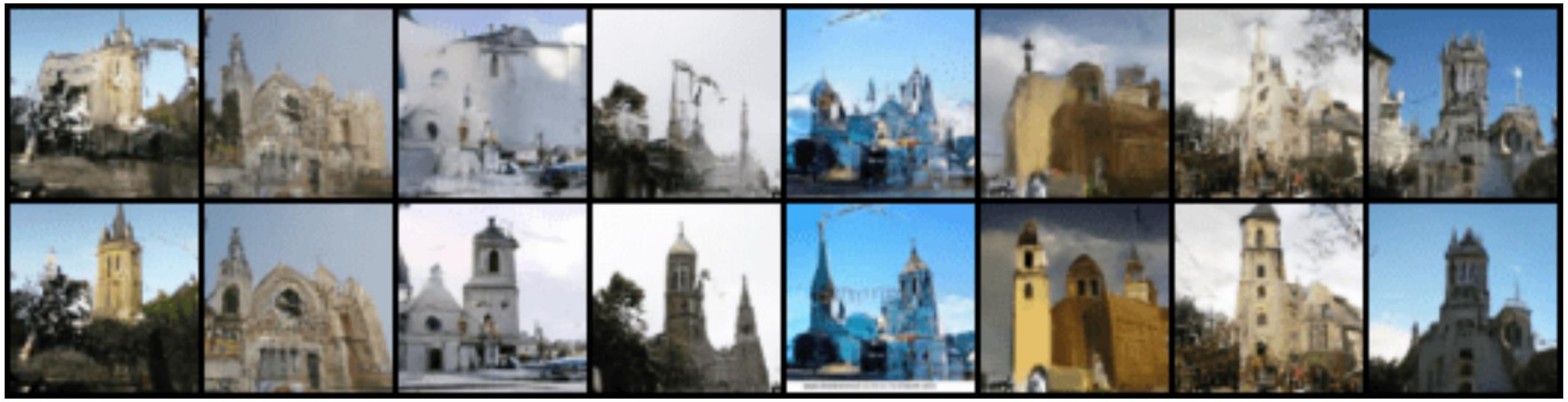


Table 3: Generative performance on CelebA HQ 256

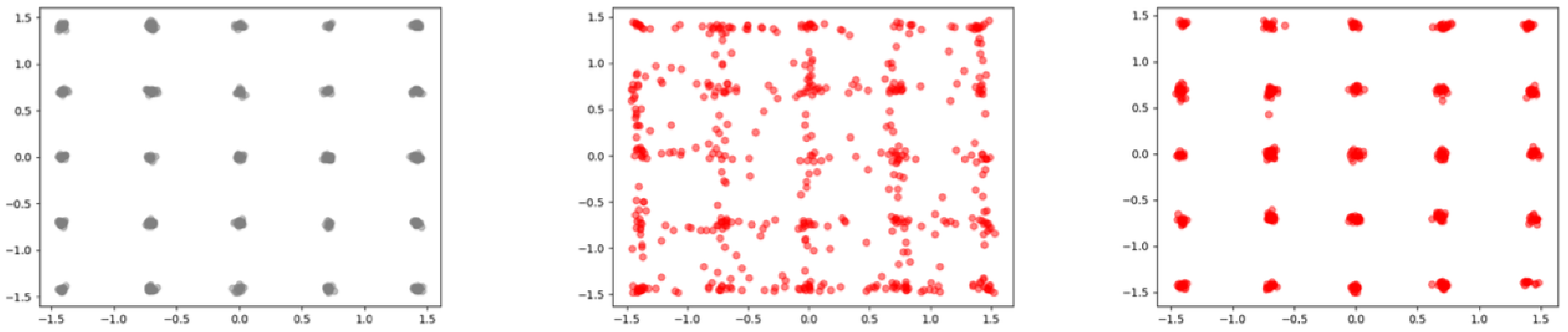
Model	FID↓
VAEBM (ours)	20.38
NVAE (Vahdat & Kautz)	45.11
GLOW (Kingma & Dhariwal)	68.93
Advers. LAE (Pidhorskyi et al.)	19.21
PGGAN (Karras et al.)	8.03

Experiments (LSUN vae/vaebm)





Experiments (2D toy data)



(a) Samples from the true distribution (b) Samples from VAE (c) Samples from VAEBM

Figure 4: Qualitative results on the 25-Gaussians dataset

Settings

Settings

WGAN initialized with NVAE decoder: We initialize the generator with the pre-trained NVAE decoder, and the discriminator is initialized by a CIFAR-10 energy network with random weights. We use spectral normalization and batch normalization in the discriminator as we found them necessary for convergence. We update the discriminator using the Adam optimizer with constant learning rate $5e{-5}$, and update the generator using the Adam optimizer with initial learning rate $5e{-6}$ and cosine decay schedule. We train the generator and discriminator for 40k iterations, and we reach convergence of sample quality towards the end of training.

EBM on x, w/ or w/o initializing MCMC with NVAE samples: We train two EBMs on data space similar to Du & Mordatch (2019), where for one of them, we use the pre-trained NVAE to initialize the MCMC chains that draw samples during training. The setting for training these two EBMs are the same except for the initialization of MCMC. We use spectral normalization in the energy network and energy regularization in the training objective as done in Du & Mordatch (2019) because we found these modifications to improve performance. We train the energy function using the Adam optimizer with constant learning rate $1e{-4}$. We train for 100k iterations, and we reach convergence of sample quality towards the end of training. During training, we draw samples from the model following the MCMC settings in Du & Mordatch (2019). In particular, we use persistent sampling and sample from the sample replay buffer with probability 0.95. We run 60 steps of Langevin dynamics to generate negative samples and we clip gradients to have individual value magnitudes of less than 0.01. We use a step size of 10 for each step of Langevin dynamics. For test time sampling, we generate samples by running 150 steps of LD with the same settings as during training.

VAEBM with $D_{\mathrm{KL}}(p_\theta(\mathbf{x})||h_{\psi,\theta}(\mathbf{x}))$ loss: We use the same network structure for E_ψ as in VAEBM. We find persistent sampling significantly hurts the performance in this case, possibly due to the fact that the decoder is updated and hence the initial samples from the decoder change throughout training. Therefore, we do not use persistent training. We train the energy function using the Adam optimizer with constant learning rate $5e{-5}$. We draw negative samples by running 10 steps of LD with step size $8e{-5}$. We update the decoder with the gradient in Eq. 21 using the Adam optimizer with initial learning rate $5e{-6}$ and cosine decay schedule. For test time sampling, we run 15 steps of LD with step size $5e{-6}$.

VAEBM: The training of VAEBM in this section largely follows the settings described in Appendix E. We use the same energy network as for CIFAR-10, and we train using the Adam optimizer with constant learning rate $5e{-5}$. Again, we found that the performance of VAEBM with or without persistent sampling is similar. We adopt persistent sampling in this section because it is faster. The setting for the buffer is the same as in Appendix E. We run 5 steps of LD with step size $8e{-5}$ during training, and 15 steps of LD with the same step size in testing.

References

- VAEBM: A Symbiosis between Variational Autoencoders and Energy-based Models
- Implicit Generation and Generalization in Energy-Based Models
- Auto-Encoding Variational Bayes
- Energy based model code : https://github.com/rosinality/igebm-pytorch