

Improving GAN Inference with EBM model

Vasiliy Viskov, Bogdan Alexandrov, Nikita Sukhorukov,
Nikolay Ivanov, Mark Nuzhnov
Skolkovo Institute of Science and Technology, Skolkovo, Russia

Skoltech
Skolkovo Institute of Science and Technology

Machine Learning, 2023

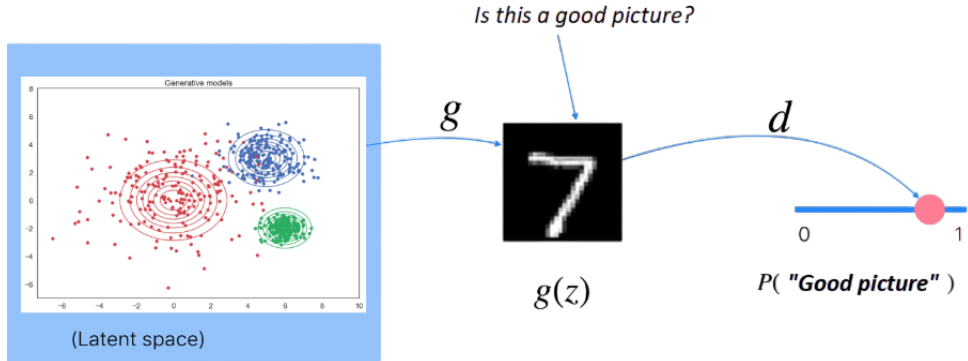
23.03.2023

Skoltech
Skolkovo Institute of Science and Technology

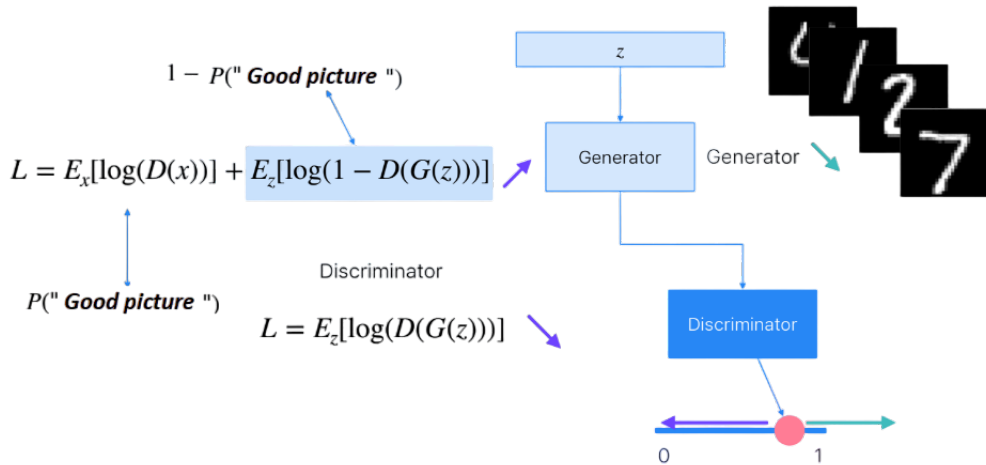
- 1 Introduction to GAN
- 2 Introduction to Energy-based models
- 3 GAN with EBM
- 4 Experiments

- ▶ Generative Adversarial Networks (GANs) are a type of deep learning model used for generating synthetic data.
- ▶ They consist of two parts: a generator that produces fake data and a discriminator that tries to distinguish real from fake data.
- ▶ The generator tries to fool the discriminator by generating data that is similar to real data, while the discriminator tries to correctly identify the fake data.
- ▶ GANs have shown impressive results in a variety of applications, including image and speech synthesis, data augmentation, and anomaly detection.

GAN pipeline



Loss function



Train the discriminator

- 1 Generate a fake image
- 2 Concatenate fake image and an image from the batch. Assign labels: 0's and 1's
- 3 Calculate BCE and update weights discriminator (do not touch the generator)

Train the generator

- 1 Generate a new fake image and invert its label from 0 to 1
- 2 Calculate BCE and update generator weights (do not touch the discriminator)

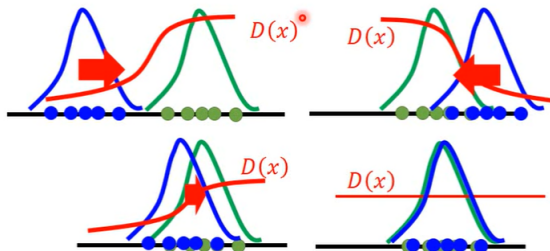
$$\min_G \max_D V(D, G) = \min_G \max_D (\mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (1)$$

Here, G represents the generator, D represents the discriminator, p_{data} is the distribution of real data, p_z is the distribution of noise input, and x and z are the input samples. The objective function tries to maximize the log probability of the discriminator being correct on real data and fake data, respectively.

Original Idea

- Discriminator
- Data (target) distribution
- Generated distribution

- Discriminator leads the generator



- ▶ Energy-based models are a class of probabilistic models that use energy functions to measure the compatibility between input data and model parameters.
- ▶ In contrast to GANs, which rely on adversarial training, EBMs use a different training paradigm called contrastive divergence.
- ▶ This involves repeatedly sampling from the model's energy function and updating the parameters to reduce the difference between the model's distribution and the true distribution of the data.

The fundamental idea of energy-based models is to turn any function that predicts values larger than zero into a probability distribution by dividing by its volume. Imagine we have a neural network, which has as output a single neuron, like in regression. We can call this network $E_\theta(\mathbf{x})$, where θ are our parameters of the network, and \mathbf{x} the input data (e.g. an image). The output of $E_\theta(\mathbf{x})$ is a scalar value between $-\infty$ and ∞ . Now, we can use basic probability theory to normalize the scores of all possible inputs:

$$q_\theta(\mathbf{x}) = \frac{\exp(-E_\theta(\mathbf{x}))}{Z_\theta} \quad \text{where} \quad Z_\theta = \begin{cases} \int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x})), d\mathbf{x} & \text{if } \mathbf{x} \text{ is continuous} \\ \sum_{\mathbf{x}} \exp(-E_\theta(\mathbf{x})) & \text{if } \mathbf{x} \text{ is discrete} \end{cases} \quad (2)$$

The \exp -function ensures that we assign a probability greater than zero to any possible input. We use a negative sign in front of E because we call E_θ to be the energy function: data points with high likelihood have a low energy, while data points with low likelihood have a high energy. Z_θ is our normalization terms that ensures that the density integrates/sums to 1. We can show this by integrating over $q_\theta(\mathbf{x})$:

$$\int_{\mathbf{x}} q_\theta(\mathbf{x}) d\mathbf{x} = \int_{\mathbf{x}} \frac{\exp(-E_\theta(\mathbf{x}))}{\int_{\tilde{\mathbf{x}}} \exp(-E_\theta(\tilde{\mathbf{x}})) d\tilde{\mathbf{x}}} d\mathbf{x} = \frac{\int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x})) d\mathbf{x}}{\int_{\tilde{\mathbf{x}}} \exp(-E_\theta(\tilde{\mathbf{x}})) d\tilde{\mathbf{x}}} = 1 \quad (3)$$

Note that we call the probability distribution $q_\theta(\mathbf{x})$ because this is the learned distribution by the model, and is trained to be as close as possible to the true, unknown distribution $p(\mathbf{x})$.

How to calculate?

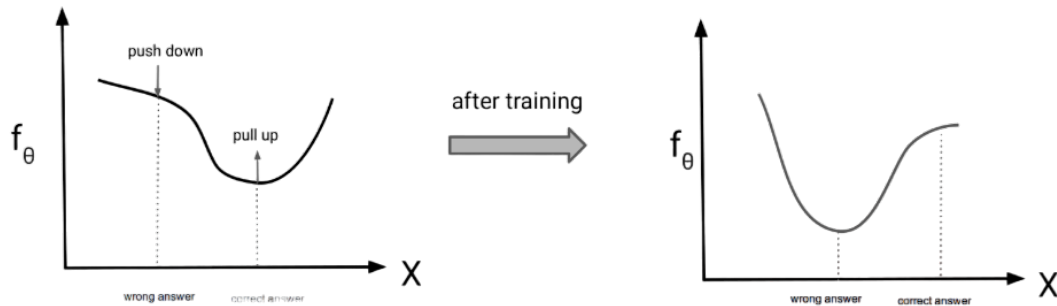
There is no chance that we can calculate Z_θ analytically for high-dimensional input and/or larger neural networks, but the task requires us to know Z_θ . Although we can't determine the exact likelihood of a point, there exist methods with which we can train energy-based models. Thus, we will look next at “Contrastive Divergence” for training the model.

When we train a generative model, it is usually done by maximum likelihood estimation. We cannot just maximize the un-normalized probability $\exp(-E_\theta(x_{train}))$ because there is no guarantee that Z_θ stays constant, or that x_{train} is becoming more likely than the others. However, if we base our training on comparing the likelihood of points, we can create a stable objective. Namely, we can re-write our maximum likelihood objective where we maximize the probability of x_{train} compared to a randomly sampled data point of our model:

$$\nabla_\theta \mathcal{L}_{MLE}(\theta; p) = -\mathbb{E}_{p(\mathbf{x})} [\nabla_\theta \log q_\theta(\mathbf{x})] = \mathbb{E}_{p(\mathbf{x})} [\nabla_\theta E_\theta(\mathbf{x})] - \mathbb{E}_{q_\theta(\mathbf{x})} [\nabla_\theta E_\theta(\mathbf{x})] \quad (4)$$

Note that the loss is still an objective we want to minimize. Thus, we try to minimize the energy for data points from the dataset, while maximizing the energy for randomly sampled data points from our model (how we sample will be explained below). Although this objective sounds intuitive, how is it actually derived from our original distribution $q_\theta(x)$? The trick is that we approximate Z_θ by a single Monte-Carlo sample. This gives us the exact same objective as written above. Visually, we can look at the objective as follows

Visual explanation



f_θ represents $\exp(-E_\theta(x))$ in our case. The point on the right, called “correct answer”, represents a data point from the dataset (i.e. x_{train}), and the left point, “wrong answer”, a sample from our model (i.e. x_{sample}). Thus, we try to “pull up” the probability of the data points in the dataset, while “pushing down” randomly sampled points. The two forces for pulling and pushing are in balance iff $q_\theta(x) = p(x)$

How to sample?

For sampling from an energy-based model, we can apply a Markov Chain Monte Carlo using Langevin Dynamics. The idea of the algorithm is to start from a random point, and slowly move towards the direction of higher probability using the gradients of E_θ . Nevertheless, this is not enough to fully capture the probability distribution. We need to add noise ω at each gradient step to the current sample. Under certain conditions such as that we perform the gradient steps an infinite amount of times, we would be able to create an exact sample from our modeled distribution. However, as this is not practically possible, we usually limit the chain to K steps (K is a hyperparameter that needs to be finetuned).

Algorithm 1 Sampling from an energy-based model

- 1: Sample $\tilde{\mathbf{x}}^0$ from a Gaussian or uniform distribution;
 - 2: **for** sample step $k = 1$ to K **do** ▷ Generate sample via Langevin dynamics
 - 3: $\tilde{\mathbf{x}}^k \leftarrow \tilde{\mathbf{x}}^{k-1} - \eta \nabla_{\mathbf{x}} E_{\theta}(\tilde{\mathbf{x}}^{k-1}) + \omega$, where $\omega \sim \mathcal{N}(0, \sigma)$
 - 4: **end for**
 - 5: $\mathbf{x}_{\text{sample}} \leftarrow \tilde{\mathbf{x}}^K$
-

Algorithm 2 Training an energy-based model for generative image modeling

```
1: Initialize empty buffer  $B \leftarrow \emptyset$ 
2: while not converged do
3:   Sample data from dataset:  $\mathbf{x}_i^+ \sim p_{\mathcal{D}}$ 
4:   Sample initial fake data:  $\mathbf{x}_i^0 \sim B$  with 95% probability, else  $\mathcal{U}(-1, 1)$ 
5:   for sample step  $k = 1$  to  $K$  do ▷ Generate sample via Langevin dynamics
6:      $\tilde{\mathbf{x}}^k \leftarrow \tilde{\mathbf{x}}^{k-1} - \eta \nabla_{\mathbf{x}} E_{\theta}(\tilde{\mathbf{x}}^{k-1}) + \omega$ , where  $\omega \sim \mathcal{N}(0, \sigma)$ 
7:   end for
8:    $\mathbf{x}^- \leftarrow \Omega(\tilde{\mathbf{x}}^K)$  ▷  $\Omega$ : Stop gradients operator
9:   Contrastive divergence:  $\mathcal{L}_{CD} = 1/N \sum_i E_{\theta}(\mathbf{x}_i^+) - E_{\theta}(\mathbf{x}_i^-)$ 
10:  Regularization loss:  $\mathcal{L}_{RG} = 1/N \sum_i E_{\theta}(\mathbf{x}_i^+)^2 + E_{\theta}(\mathbf{x}_i^-)^2$ 
11:  Perform SGD/Adam on  $\nabla_{\theta}(\mathcal{L}_{CD} + \alpha \mathcal{L}_{RG})$ 
12:  Add samples to buffer:  $B \leftarrow B \cup \mathbf{x}^-$ 
13: end while
```

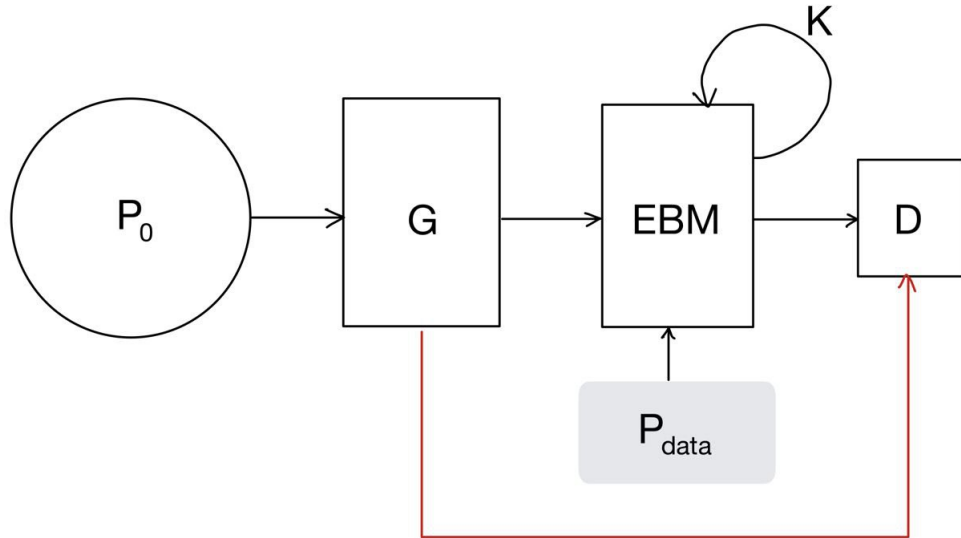
► Training:

- 1 Fit D replacing p_{data} with K steps of MCMC sampling from $G(z)$, $z \sim \mathcal{N}(0, 1)$
- 2 Fit G as in GAN training pipeline
- 3 Fit EBM model replacing fake data with $G(z)$, $z \sim \mathcal{N}(0, 1)$

► Inference

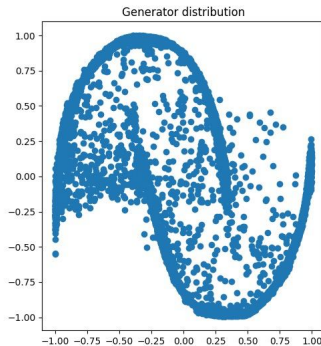
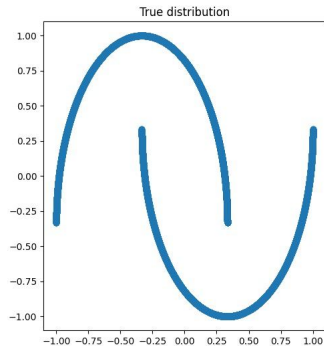
- 1 Sample $z \sim \mathcal{N}(0, 1)$, apply K steps of MCMC sampling to $G(z)$

GAN with EBM: pipeline



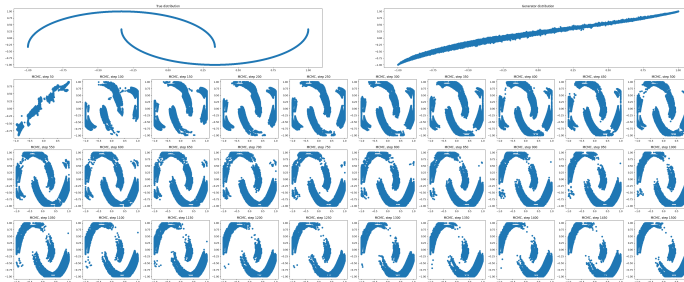
- ▶ Stabilization of GAN learning
- ▶ After learning our model we get not only sampler (generator) from learned distribution, but it PDF (unnormalized)

Experiment (1)



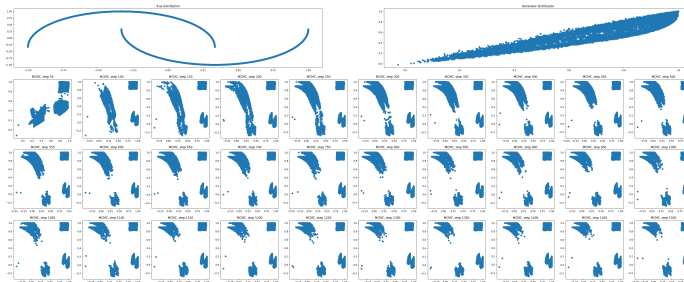
- Make Moons from Scikit-learn
- Vanilla GAN

Experiment (2)



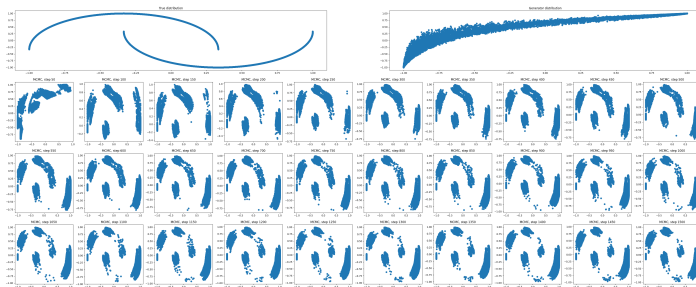
- Make Moons from Scikit-learn
- Vanilla GAN
- EBM
- Steps: 60, step_size: 3

Experiment (3)



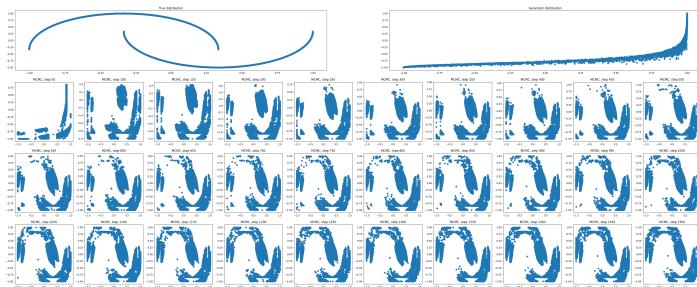
- Make Moons from Scikit-learn
- Vanilla GAN
- EBM
- Steps: 30, step_size: 5

Experiment (4)



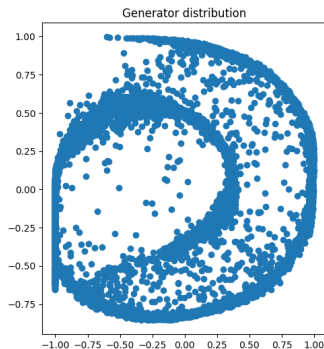
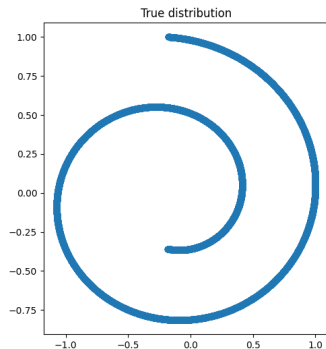
- Make Moons from Scikit-learn
- Vanilla GAN
- EBM
- Steps: 10, step_size: 10

Experiment (5)



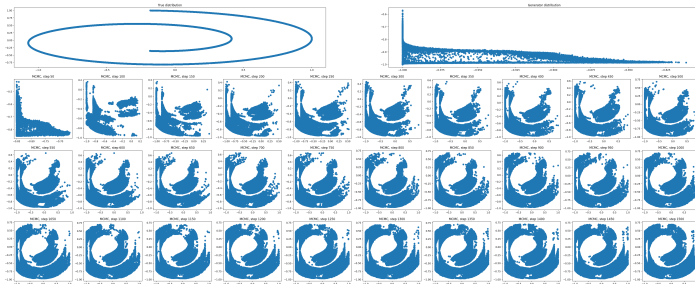
- Make Moons from Scikit-learn
- Vanilla GAN
- EBM
- Steps: 40, step_size: 5

Experiment (6)



- Swiss roll from Scikit-learn
- Vanilla GAN

Experiment (7)



- Swiss roll from Scikit-learn
- Vanilla GAN
- EBM
- Steps: 60, step_size: 3

Experiment (8)



Figure: Comparison of Vanilla GAN and Vanilla GAN with EBM on MNIST

Experiment (9)

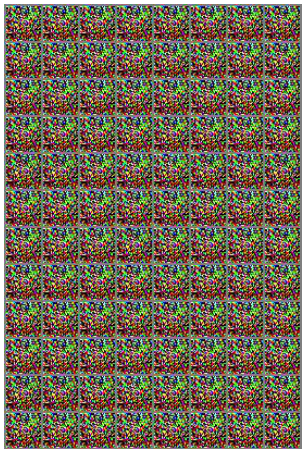


Figure: Comparison of Vanilla GAN and Vanilla GAN with EBM on CIFAR10

- ▶ We learned how GANs work
- ▶ We learned how EBMs work
- ▶ We implemented simultaneously GAN and EBM for 2D datasets (Make Moons, Swiss Roll) and MNIST
- ▶ We compared different step sizes, MCMC step count and model architectures with their parameters

- ▶ Use another types of GAN (Conditional, infoGAN, etc.)
- ▶ Think of using various heuristics as a playout algorithm

- ▶ Vasiliy Viskov - project mastermind and code guru
- ▶ Bogdan Alexandrov - theory, code, presentation
- ▶ Nikita Sukhorukov - theory, code, presentation
- ▶ Nikolay Ivanov - theory, code, presentation
- ▶ Mark Nuzhnov - theory, code, presentation

- ▶ ECCV2022. Learning Energy-Based Models With Adversarial Training. Xuwang Yin, Shiyong Li, Gustavo K. Rohde
- ▶ GAN-Tutorial
- ▶ [Goodfellow et. al., 2014] "Generative Adversarial Nets"
- ▶ [Nijkamp et. al., 2019] "On the Anatomy of MCMC-Based Maximum Likelihood Learning of Energy-Based Models"

Thank you for attention!
Questions? Link to Github