Independent Study final report on Energy Based Models

Chaitanya Kharyal
(20171208)

Contents

1 Introduction

1.1 Why Energy Based Models? (why not multilayer perceptrons?)

Before getting into what energy based models[1] are, let us have a look at why do we need anything other than a multilayer perceptron for making machines understand the world. Following are the two broad categories of problems that we face with the multilayer perceptron:

- **Complex Inference Procedure:** It can so happen that the inference procedure required for our task is more complex than calculating a stacked layer of weighted sum (MLP). Since the inference is more complex, we will need something more complex than a simple DL architecture to get desired results.

- **Multiple outputs for the same input:** Traditionally, the DL models give only one output for a single input which might be desirable in some cases while it might not be in some others. For example, while predicting future video frames from current and past frames, there can be multiple possible outputs (future frames) given the same inputs because there is an intrinsic uncertainty associated to the future. Moreover, if we use a simple MLP with Least Squares loss

function, the output future frames would be too blurry to be used meaningfully. Lets consider the experiment shown in the figure, we are asked to predict the

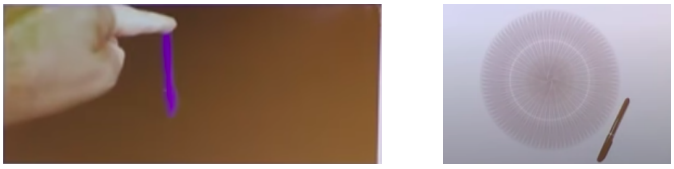

Figure 1: *The initial position (input) of the pen and the possible final outcomes (blurry) of its position on the table*

final position of the pen on a table when is allowed to fall from upright position. This situation can lead to multiple possible outcomes of pen falling in different directions. Now, if we train a Least Squares based DL model to predict this outcome, the best it can do (in order to minimise the MSE) is to output a very blurry image (as depicted by the right side picture). This can be further understood by a simpler example of text completion. Let's say we train a model to complete the phrase "I am a ... boy" and our training data has two type of completions for this particular phrase, "good" and "strong". Now, let the embeddings for them be $e(\text{good})$ and $e(\text{strong})$ respectively. Since the input phrase is the same, a normal MLP won't be able to incorporate the multiple outcomes for the same input and would end up giving something like $\frac{e(\text{good})+e(\text{strong})}{2}$ as the output in order to minimise the error for both of the training samples.

2 Energy Based Models

2.1 Energy and Inference

Unlike a DL model which gives an output y for an input x, the Energy based models try to model the compatibility between a given input-output pair. The main component of an EBM, as the name suggests, is energy and, intuitively, it describes the dependence on the input-output space. Formally, energy function (F) is,

$$F : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \tag{1}$$

The lesser the energy of a particular input-output pair, the better they fit together. Now, the inference procedure to output a particular output y once the energy is learnt becomes pretty straight forward,

$$\hat{y} = \underset{y}{\operatorname{argmin}}\, F(x, y) \tag{2}$$

If the dimension of $\mathcal{Y}$ is small, we can compute all the $F(x, y)$s and pick the y that minimises the energy. But if the dimension is large and it is not possible to compute the energy for all possible ys (say y is continuous), then we need to resort to the gradient based methods for our inference. For this, we would like to choose a differentiable energy function.

2.2 Latent Variable

If we are saying that we can have multiple outputs for a single input, this means that something other than the input itself too has control over the output, just that we don't know the value of this something else. We call this variable a latent variable z. The latent variables can provide us with auxiliary information about the task. In the previous case of determining the final position of the pen, one type of latent variable can be the direction and magnitude of airflow that might affect the direction of fall. Knowing such information would make the inference task easier.
Now, we want to minimise our energy function with respect to both the output and the latent variable. Therefore, the inference process becomes:

$$\hat{y}, \hat{z} = \underset{y,z}{\operatorname{argmin}} F(x, y, z) \tag{3}$$

Note that if we have any two of the three variables (x, y, z) and the energy function (F), we can infer the other variable by applying gradient descent on the energy function (if the variable is differential). Before we go any further into the energy based models, I want to introduce a paper which uses a similar concept.

2.2.1 Concept Learning

For this section, I will use [2] as my main reference which is also the first paper that I have read in the concept learning domain. Here, the author has used "concept" and "attention" as latent variables. Therefore, for some input state $\mathbf{x}$ (Each state contains a collection of N entities $\mathbf{x}^t = [x_0, \ldots, x_N]$ and each entry x_i^t can contain information such as position and colour of the entity), the Energy can be represented as $E(\mathbf{x}, \mathbf{a}, \mathbf{w})$ where $\mathbf{a}$ and $\mathbf{w}$ are attention and concept respectively. Here, attention is nothing but an attention mask over the entities in the state $\mathbf{x}$ which tells us which entities are we attending to, and $\mathbf{w}$ a concept like "square", "circle" etc as shown in the figure.

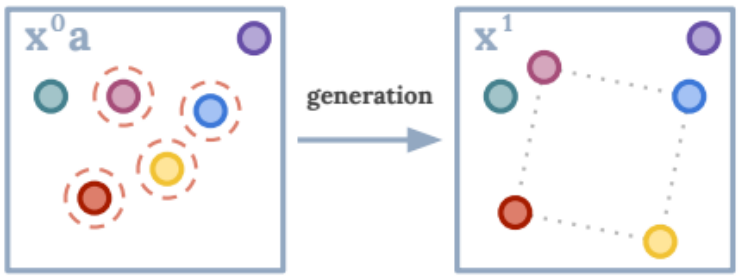

Figure 2: A visualisation for attention mask $\mathbf{a}$ over the state $\mathbf{x}^0(left)$, and the concept "square" (right)

Therefore, ideally, $E(\mathbf{x}, \mathbf{a}, \mathbf{w}) = 0$ when state trajectory $\mathbf{x}$ under attention mask $\mathbf{a}$ over entities satisfies the concept $\mathbf{w}$, and it is > 0 otherwise. Therefore, If we know concept $\mathbf{w}$, we can calculate one of $\mathbf{x}$ and $\mathbf{a}$ if we know the other using:

$$\mathbf{x}(\mathbf{a}) = \underset{x}{\operatorname{argmin}}\, E(\mathbf{x}, \mathbf{a}, \mathbf{w}) \qquad \mathbf{a}(\mathbf{x}) = \underset{a}{\operatorname{argmin}}\, E(\mathbf{x}, \mathbf{a}, \mathbf{w})$$

Both of these can be approximated using gradient descent.

2.2.1.1 Learning Procedure

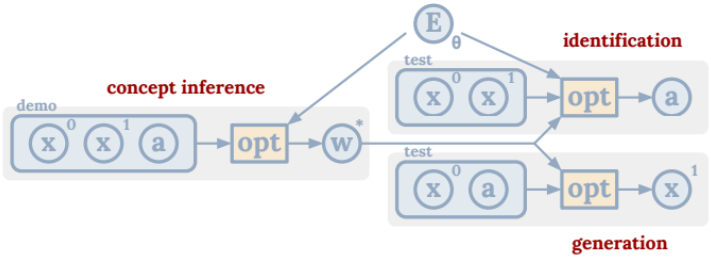

Figure 3: A visualisation for the learning procedure

First, a pre-generated training state and attention is fed to the energy function and the resultant $\mathbf{w}$ is obtained through optimisation (gradient descent etc.). Now, this $\mathbf{w}$ is used on another state and attention pair and predicted attention and state are obtained through optimisation again. Now, this attention and state are compared against the ground truth attention and state to generate the loss, and the error is back propagated.
There is a major pitfall in the training procedure. Unlike classical ML, here, gradient descent is a part of the inference procedure. This means that if the network for Energy is, say, 5 layers deep, and the optimisation module takes 10 steps for optimisation, it is equivalent to having a 50 layer deep network because every optimisation step will require a forward pass of the network. Moreover, if this optimization is happening 3 times (as is the case above), it is equivalent to having a 100 layer deep network (2 optimizations are in parallel). Therefore, even for a simple energy function (5 layers deep), we would require computation worth 100 layer network.

2.2.2 Free Energy

We can redefine our energy function to not depend on the latent variable z by marginalizing over it:

$$F_\beta(x, y) = -\frac{1}{\beta} \int_z e^{-\beta E(x,y,z)} \tag{4}$$

This new energy function is called free energy and is derived from the Boltzmann distribution. Another popular way to make the energy function independent of the latent variable is by minimizing over z.

$$F_\infty(x, y) = \min_z E(x, y, z) \tag{5}$$

This can also be interpreted as $\beta \to \infty$ Boltzmann distribution.

2.3 Generative Models

The energy based generative models are divided into two main categories,

2.3.1 Implicit Generative Models

Definition 1 (IGM)**.** *[2] [3] IGM is a family of probability distributions $\mathbb{G}_\theta$ parametrized by a learnable generator function $G : \mathcal{Z} \to \mathcal{X}$ that maps latent samples z from a fixed latent distribution η to the data space $\mathcal{X}$. The latent distribution η is required to have a density over the latent space $\mathcal{Z}$ and is often easy to sample from. Thus, Sampling from $\mathbb{G}$ is simply achieved by first sampling z from η then applying G. Formally,*

$$x \sim \mathbb{G} \iff x = G(z),\ z \sim \eta$$

Generative Adversarial Networks (GANs) [4] and Metropolis Hastings GANs [5] can be examples of Implicit Generative Models. Let us briefly look into the working of GANs:

2.3.1.1 Generative Adversarial Networks

I will be using [4] as my main reference for this section. Two differentiable functions (MLPs) $\mathbb{G}_{\theta_g}$ and $\mathbb{D}_\theta$ (generator and discriminator) depending on parameters θ_g and θ_d respectively are defined. Here, $\mathbb{G}$ is a function such that,

$$\mathbb{G} : \mathcal{Z} \to \mathcal{X}$$

Where $\mathcal{Z}$ is the latent space with the associated probability distribution η and $\mathcal{X}$ is the output space (data space) space with the generator probability distribution p_g. We want this p_g to be as close as possible to the true data distribution p_x. $\mathbb{D}(x)$ represents the probability that x came from the distribution p_x rather than p_g,

$$\mathbb{D} : \mathcal{X} \to [0, 1]$$

While training, we train $\mathbb{D}$ to maximize the probability of assigning the correct label to both training examples and samples from $\mathbb{G}$. We simultaneously train $\mathbb{G}$ to try and maximise the discriminator error. Since during training both the Discriminator and Generator are trying to optimize opposite loss functions, they can be thought of two agents playing a minimax game with value function $V(\mathbb{G}, \mathbb{D})$. Formally,

$$\min_{\mathbb{G}} \max_{\mathbb{D}} V(\mathbb{G}, \mathbb{D}) = \mathbb{E}_{x \sim p_x}[\log \mathbb{D}(x)] + \mathbb{E}_{z \sim \eta}[(1 - \log \mathbb{D}(\mathbb{G}(z)))] \tag{6}$$

Therefore, the gradient for stochastic gradient ascent for discriminator becomes:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log \mathbb{D}(x_i) + \log(1 - \mathbb{D}(\mathbb{G}(z_i)))] \tag{7}$$

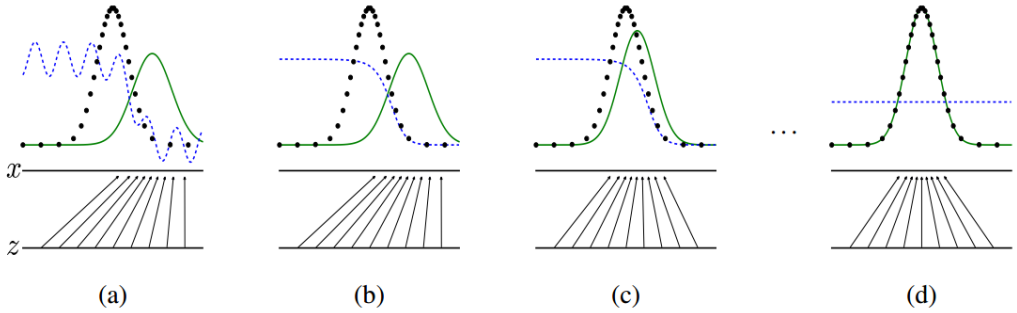

Figure 4: Training process of GAN

And that for the gradient descent for generator becomes:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} (\log(1 - \mathbb{D}(\mathbb{G}(z_i))) \tag{8}$$

As you can see in the figure, we sample from the latent space $\mathcal{Z}$ using η (uniform here), and map it to space $\mathcal{X}$ with resulting probability distribution p_g (green). Through training, we try to bring p_g and the real data distribution p_x (black, dotted) closer. During the independent study, I tried to train a GAN on MNIST dataset [6]. Following are some results:

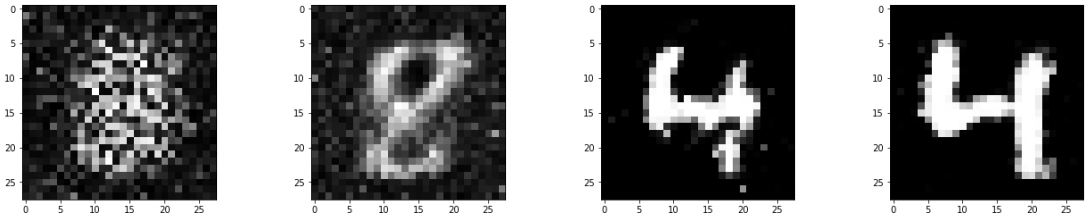

Figure 5: *Outputs of GAN after 1, 10, 50 and 200 iterations (left to right) respectively*

One major problem that I faced during training GANs is their well known issue with convergence. I tried using TraVeLGAN [7] and CycleGAN [8] for style transfer task on a custom dastaset made in Unity Game Engine and ADVIO dataset [9], but wasn't able to make them converge.

2.3.2 Sampling Based Methods

Suppose we have trained the energy function for our generative model i.e., the energy function can tell us successfully if some sample is compatible with the given dataset. Now, we want it to generate some samples compatible to the dataset. We can, then, think of the probability distribution originating from the energy function as,

$$\begin{aligned} p_E(x) &\propto e^{-E(x)} \\ &= \frac{e^{-E(x)}}{Z} \end{aligned} \tag{9}$$

Where, Z is the partition function defined as,

$$Z = \int_x e^{-E(x)}\, dx \tag{10}$$

Now, the sampling methods can be used in three ways:

- Approximating Z. As you might have noticed by now that if dimension of x is large, calculating the integral might be intractable and therefore, sampling methods are used here.
- Sampling from the distribution p_E. If the distribution is complex, we might want to use sampling methods.
- Sampling from the latent space. From out previous discussion of GAN we know that there we sampled from the latent space $\mathcal{Z}$ using some simple distribution like uniform distribution. But what if we don't want that latent distribution to be that simple? [3]

Let us see that what methods can be used for sampling in the scenarios listed above:

2.3.2.1 Sampling Methods

In this section, I am going to very briefly introduce sampling methods, their limitations and possible solutions.

Rejection Sampling

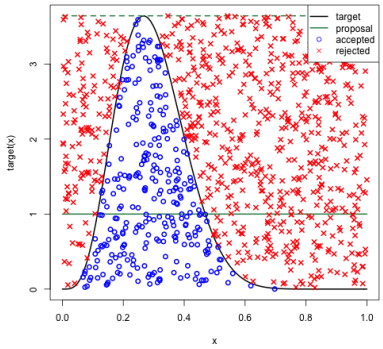

Figure 6: Rejection sampling

One of the obvious ways to sample from a given probability distribution is by drawing the points uniformly from under the probability curve. This is because the probability of a point being in a region is proportional to the height of the curve in that region. But how to sample a point when sampling from under the curve is not possible analytically? In such cases we use rejection sampling, i.e., we define a curve which we know (and can sample from) analytically such that the defined curve (scaled) is always above the distribution that we want to sample from. Now, we sample the points from under the defined curve but reject all samples that don't lie under the required distribution.

Importance Sampling
Let's say that I wanted to compute the integral,

$$\mathbb{E}[f(x)] = \int_x f(x)p(x)\ dx$$

But I don't know how to sample from the distribution p, therefore, I can't apply the Monte Carlo approximation as,

$$\mathbb{E}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i),\ \ x_i \sim p$$

But, let's say that we have another distribution q that we can sample from, now we can apply Monte Carlo method as,

$$\begin{aligned}\mathbb{E}[f(x)] &= \int_x f(x)p(x)\ dx \\ &= \int_x f(x)\frac{p(x)}{q(x)}q(x)\ dx \\ &= \frac{1}{N}\sum_{i=1}^{N} f(x_i)\frac{p(x_i)}{q(x_i)},\ \ x_i \sim q\end{aligned}$$

Limitations [10]
As expected, the rejection sampling algorithm doesn't work for higher dimensions as the acceptance rate falls exponentially in dimension. Therefore, we end up rejecting a lot of proposals before accepting a few of them.
In importance sampling, the issue is a bit more complex as, in it, we are weighting each sample with $\frac{p(x)}{q(x)}$ and we want this weight to be higher for more important regions in p. But, as the dimension increases, the variance of this weight increases exponentially, and hence, we end up weighting less important regions more which degrades our approximation.
Markov Chain Monte Carlo
Since the theory behind it is very deep, I am going to just mention one such MCMC algorithm, i.e. Metropolis Algorithm [11]. Let $\mathcal{X}$ be a Markov Chain from which we have to sample with probability distribution p, let $J(x, y)$ be the transition probability (from the graph itself) from node x to node y. We want $K(x, y)$ as the new transition probability which allows us to mix to the stationary distribution p. According to the algorithm,

$$K(x,y) = \begin{cases} J(x,y) & \text{if } x \neq y \text{ and } A(x,y) \geq 1 \\ J(x,y)A(x,y) & \text{if } x \neq y \text{ and } A(x,y) < 1 \\ J(x,y) + \sum_{z:A(x,z)<1} J(x,z)(1 - A(x,z)) & \text{if } x = y \end{cases} \tag{11}$$

Where, the acceptance probability (A) is defined as,

$$A(x,y) = \frac{p(y)J(y,x)}{p(x)J(x,y)} \tag{12}$$

2.3.2.2 Generalized Energy Based Models

I will be using [3] as my main reference in this section ([12] also presents a similar idea). This paper addresses one problem with the vanilla GAN that once the generator is trained, the discriminator is thrown away (not used for the generation part). The main claim of this paper is that the discriminator/critic contains important information about the real data distribution which can be used during sampling.

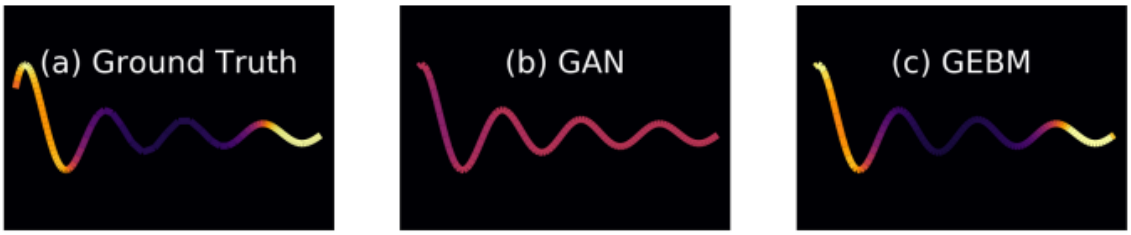

Figure 7: Comparing results from GAN and GEBM

As it can be seen in the figure, GAN and GEBM both learn the support of the data well, but the GEBM improves on the distribution of the data by incorporating the critic.
The paper introduces the Generalized Energy Based Model $Q_{B_\theta,E}$, where B_θ is the generator and $E \in \mathcal{E}$ is the class of energy functions Such that (as we saw in the implicit models),

$$X \sim Q_\theta \iff X = B_\theta(z), \;\; z \sim \eta$$

The energy function is then used to re-weight the samples based on their importance weights (similar to free energy),

$$f_{Q,E}(x) = \frac{e^{-E(x)}}{Z_{Q_\theta,E}} \tag{13}$$

Where,

$$Z_{Q_\theta,E} = \int e^{-E(x)} \, dQ_\theta$$

Note that GEBM is also a bit different than the classical EBMs in the sense that in EBMs, the energy is defined on the data space $\mathcal{X}$ instead of the latent/support space $\mathcal{Z}$ in the case of GEBM.
The training of the energy function (and generator) in GEBM is somewhat similar to the training of discriminator in the GAN and they try to minimize the Generalized log-likelihood,

$$\mathcal{L}_{P,Q}(E) := \int \log f_{Q,E} \, dP = -\int E \, dP - \log Z_{Q,E}$$

Where, P is the real data distribution. As you might have noticed, calculating Z in intractable and they overcome this problem by using a lower bound on the generalized log-likelihood,

$$\mathcal{L}_{P,Q}(E) \geq \int (E+c)dP - \int e^{-E+c}dQ_\theta + 1 := \mathcal{F}(P, Q_\theta; \mathcal{E}, \mathbb{R}) \tag{14}$$

Where $c \in \mathbb{R}$ is a constant that we optimize over along with E.
Now that we have trained our Energy function and the generator, we can define the sampling method to sample the images from our model. For some test function g, its expectation with respect to Q is defined as,

$$\int g(x)\, dQ_{B,E} = \int g(B(z)) f_{B,E}(B(z))\eta(z)\, dz \tag{15}$$

Therefore, we can define our posterior latent distribution from the importance weights as,

$$\nu_{B,E}(z) = \eta(z) f_{B,E}(B(z)) \tag{16}$$

Now, we can sample from the latent space using the distribution ν using the sampling methods and apply the generator B to get the samples.

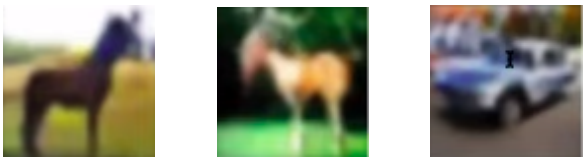

Figure 8: *Outputs generated by GEBM on CIFAR-10 dataset*

Why does it make sense to change the latent distribution?
I am using [12] as my main reference in this subsection. This paper uses the vanilla GAN and presents it as an Energy Based Model, where the energy is defined using the output of the discriminator network. From [4], we know that the optimal discriminator is the one which gives its output as,

$$D(x) = \frac{p_d(x)}{p_d(x) + p_g(x)}$$

Where, p_d and p_g are the real and generated data probabilities respectively. Now, we define $d(x)$ as the logit of $D(x)$,

$$D(x) = \frac{p_d(x)}{p_d(x) + p_g(x)} = \frac{1}{1 + \frac{p_g(x)}{p_d(x)}} \approx \frac{1}{1 + e^{-d(x)}} \tag{17}$$

$$\implies p_d(x) = p_g(x)e^{d(x)} \tag{18}$$

Adding the normalizing constant,

$$p_d^*(x) = \frac{p_g(x)e^{d(x)}}{Z_0} \tag{19}$$

Theorem 1. *When the Discriminator is optimal, $p_d^* = p_d$.*

Proof. We know that for an Ideal discriminator,

$$D(x) = \frac{p_d(x)}{p_d(x) + p_g(x)}$$

We also know that,

$$\begin{aligned} D(x) &= \sigma(\log p_d - \log p_g) \\ \implies d(x) &= \log p_d - \log p_g \\ \implies p_d^*(x) &= \frac{p_g(x) e^{\log p_d - \log p_g}}{Z_0} \\ \implies p_d^* &= \frac{p_d}{Z_0} \end{aligned}$$

Since Z_0 is the normalization constant, it must be 1 since p_d is itself a probability distribution.

$$\implies p_d^* = p_d$$

Now, we define the energy function in the latent space as,

$$E(z) = -\log p_0(z) - d(G(z)) \tag{20}$$

And the corresponding Boltzmann distribution becomes,

$$P_t(z) = \frac{e^{-E(z)}}{Z} \tag{21}$$

Where, p_0 is the initial latent distribution (for example uniform in the case of GAN).

Theorem 2. *If we sample $z \sim p_t$, and $x = G(z)$ for some G, then $x \sim p_d^*$, i.e. the induced probability measure $G \circ p_t = p_d^*$*

Proof. Ideally, we know the distribution p_g. To generate samples from p_d^*, we can do rejection sampling with acceptance probability,

$$\frac{p_d^*(x)}{M p_g(x)} = \frac{e^{d(x)}}{M Z_0}$$

Where, M is a constant. Now, this can be thought of as doing rejection sampling in the latent space with the acceptance probability,

$$r(z) = \frac{p_d^*(G(z))}{M p_g(G(z))} = \frac{e^{d(G(z))}}{M Z_0}$$

This induces a new probability distribution in the latent space as,

$$p_t(z) = \frac{p_0(z) r(z)}{C} = \frac{e^{-E(z)}}{C}$$

Where, C is a constant. Therefore, sampling from p_d is equivalent to sampling from $p_t(z)$ and then applying G on it.
This tells us that the main purpose of the generator is to learn the support of the data, and with the discriminator, one can learn how to distribute the data on that support and in-turn sample from it.

References

[1] Yann LeCun, Sumit Chopra, Raia Hadsell, Fu Jie Huang, and et al. A tutorial on energy-based learning. In *PREDICTING STRUCTURED DATA*. MIT Press, 2006.

[2] Igor Mordatch. Concept learning with energy-based models, 2018.

[3] Michael Arbel, Liang Zhou, and Arthur Gretton. Generalized energy based models, 2020.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[5] Ryan Turner, Jane Hung, Eric Frank, Yunus Saatci, and Jason Yosinski. Metropolis-hastings generative adversarial networks, 2019.

[6] Feiyang Chen, Nan Chen, Hanyang Mao, and Hanlin Hu. Assessing four neural networks on handwritten digit recognition dataset (mnist), 2019.

[7] Matthew Amodio and Smita Krishnaswamy. Travelgan: Image-to-image translation by transformation vector learning, 2019.

[8] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.

[9] Santiago Cortés, Arno Solin, Esa Rahtu, and Juho Kannala. Advio: An authentic dataset for visual-inertial odometry, 2018.

[10] Art B. Owen. *Monte Carlo theory, methods and examples.* 2013.

[11] Christian P. Robert. The metropolis-hastings algorithm, 2016.

[12] Tong Che, Ruixiang Zhang, Jascha Sohl-Dickstein, Hugo Larochelle, Liam Paull, Yuan Cao, and Yoshua Bengio. Your gan is secretly an energy-based model and you should use discriminator driven latent sampling, 2020.