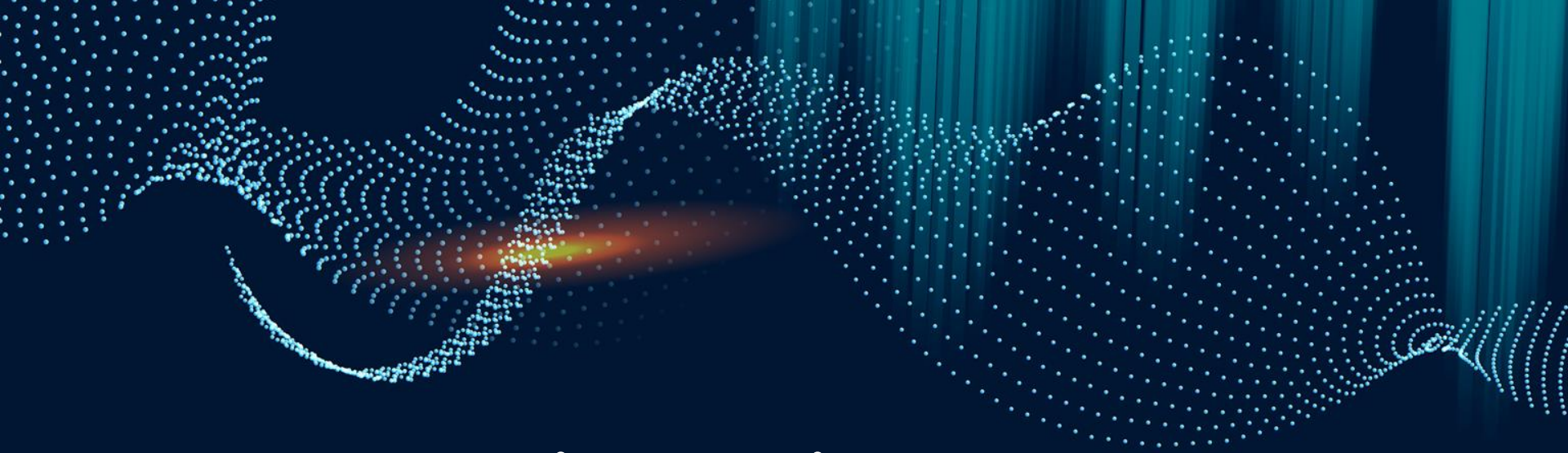
Generative point-net deep energy based learning

Mehrdad Baradaran
Katayoun Kobraei

Neural Network Presentation

CONTENTS OF THIS TEMPLATE

There's what you'll find in this **Slides** :



An introduction to GAN

Generative adversarial networks as an exiting innovation in machine learning

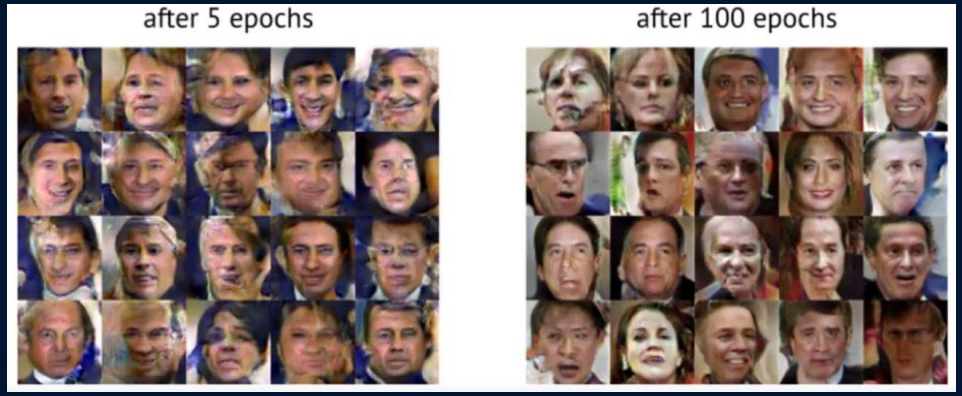
What is GAN?

They were first introduced by *Ian Goodfellow et al.* in 2014.

Can learn to draw samples from a model that is similar to data that we give them.

Un-supervised training of two models.
Learning a model without seeing the data from the feedbacks of other models (using seeing the real data and learning how to recognize the correctness and fakeness of the generated data)

They have achieved some incredible results:

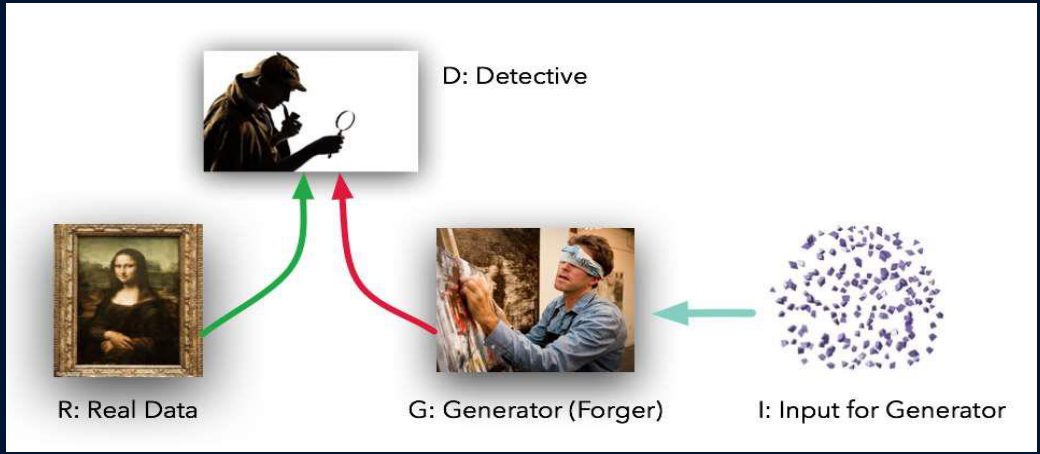


In 2015, the first times this idea was applied to faces, the results were like this.
Of course, in those years, they thought that these results were very good and desirable.

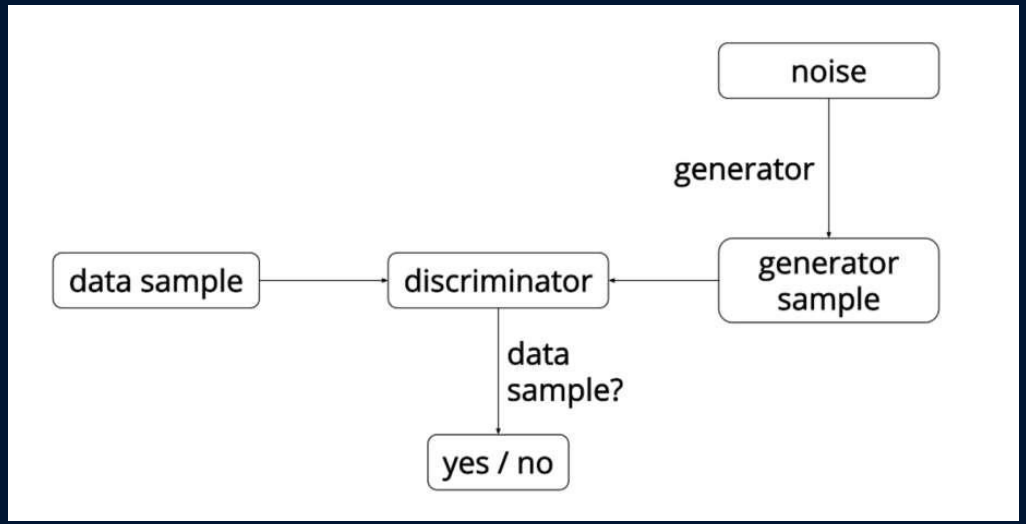
Since this model requires more processing time, and the initial models and random generators are important, NVIDIA, which has more hardware at its disposal, worked on these figures until finally in 2017, it reached these results.

None of these figures have external existence

Generative Adversarial Networks



By giving a latent space or noise space, the generator creates fake data, and the discriminator must use real data to label them fake or real.
The discriminator is updated using the number of correct recognition errors with gradient descent. (At this time, the generator is frozen)
And the generator is updated using the failure to fool the discriminator (only fake data labels are detected.)
At this stage, the discriminator is frozen and the generator is updated with gradient ascend.
So, we have two training steps every time we take a mini batch.



Reza Zadeh @Reza_Zadeh · 15m

GANs in nature: Cuckoos (Generator) lay eggs in Warbler (Discriminator) bird's nests, tricking warblers to raise cuckoo chicks. Warblers are evolving to recognize cuckoo eggs to destroy them. Simultaneously, cuckoos evolve to produce more warbler-like eggs cam.ac.uk/research/featu…

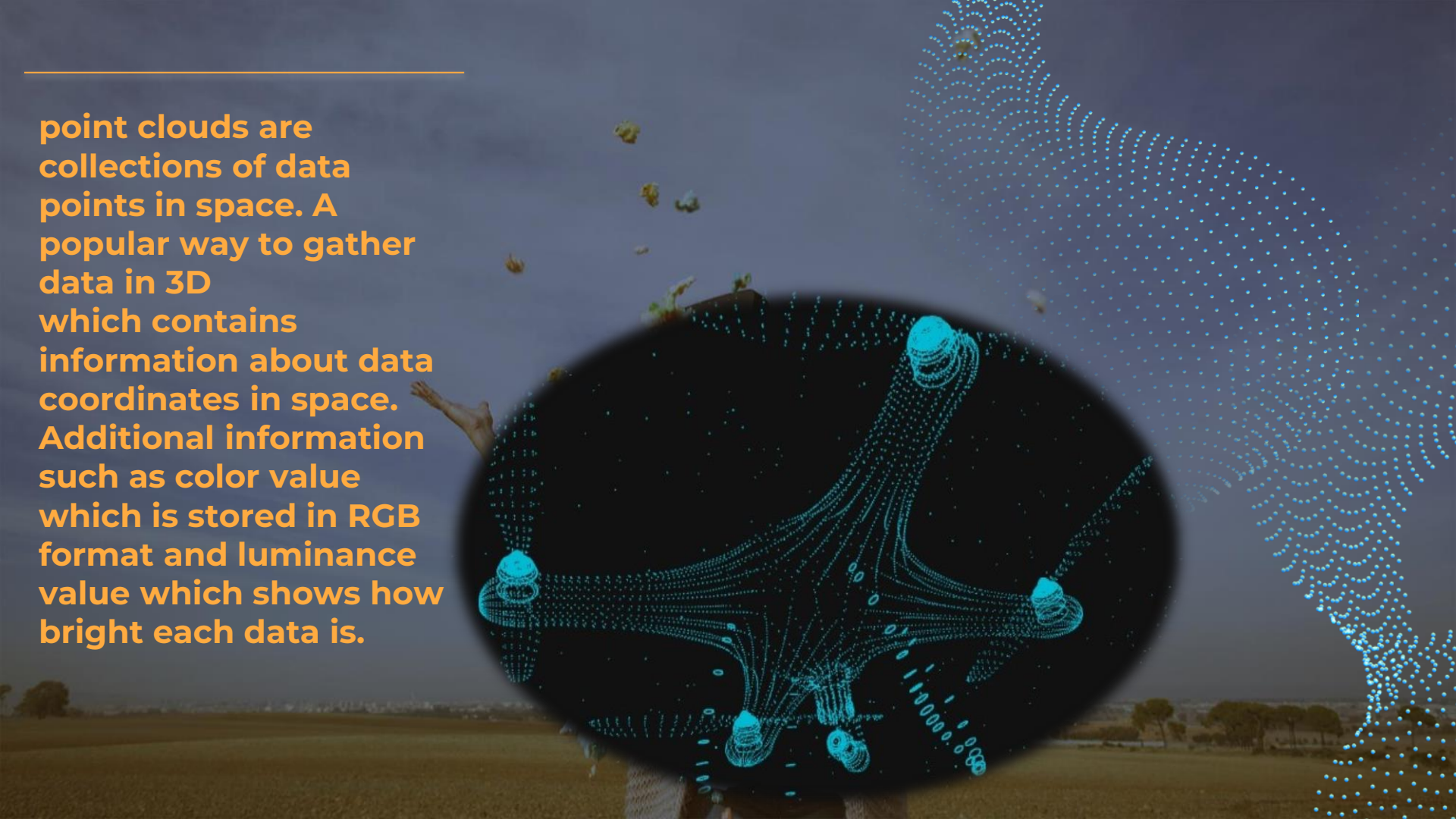
GAN Lab

https://poloclub.github.io/ganlab/

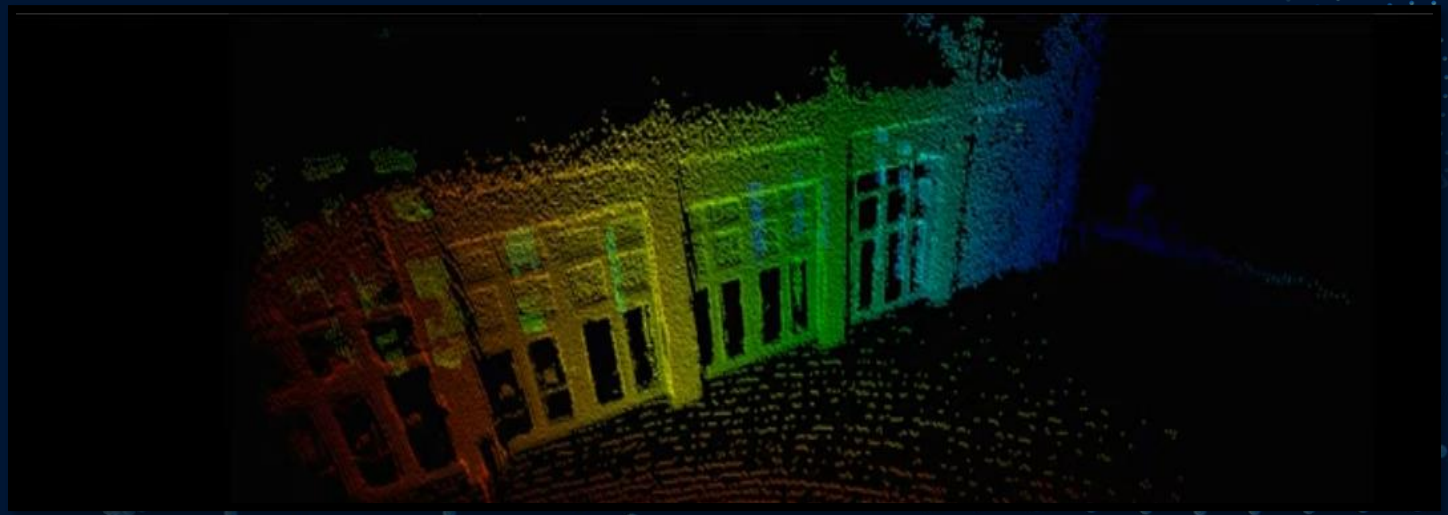
Point clouds

One of the simplest and
most cost-effective ways
to retrieve spatial data in 3D

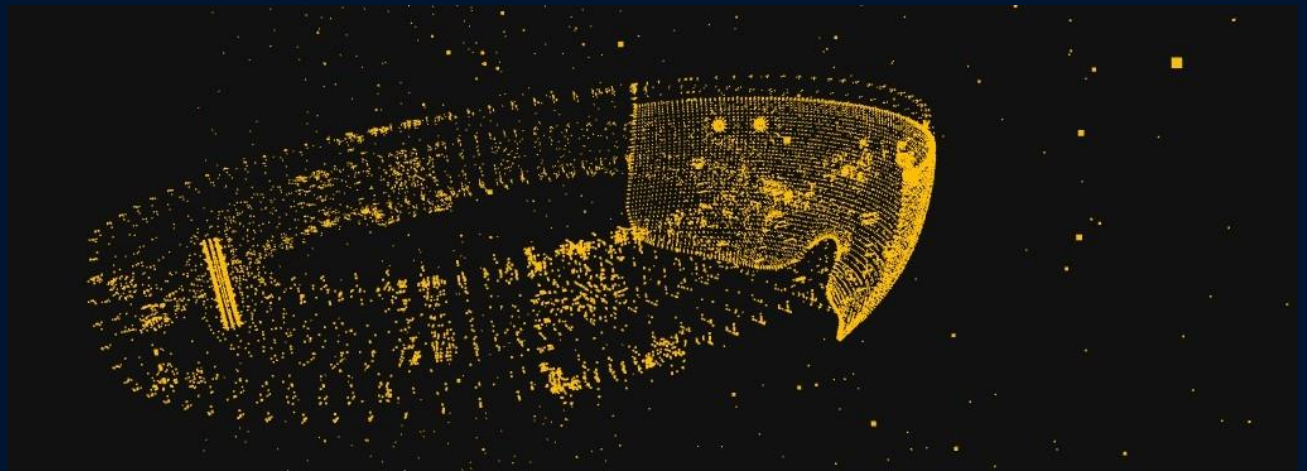
point clouds are collections of data points in space. A popular way to gather data in 3D
which contains information about data coordinates in space. Additional information such as color value which is stored in RGB format and luminance value which shows how bright each data is.



point clouds are a standard 3d acquisition format widely uses by devices like face sensors and.... The data format of the point cloud is a set of 3d coordinates. It's an unordered stat so that changing a point order will not affect the shape.



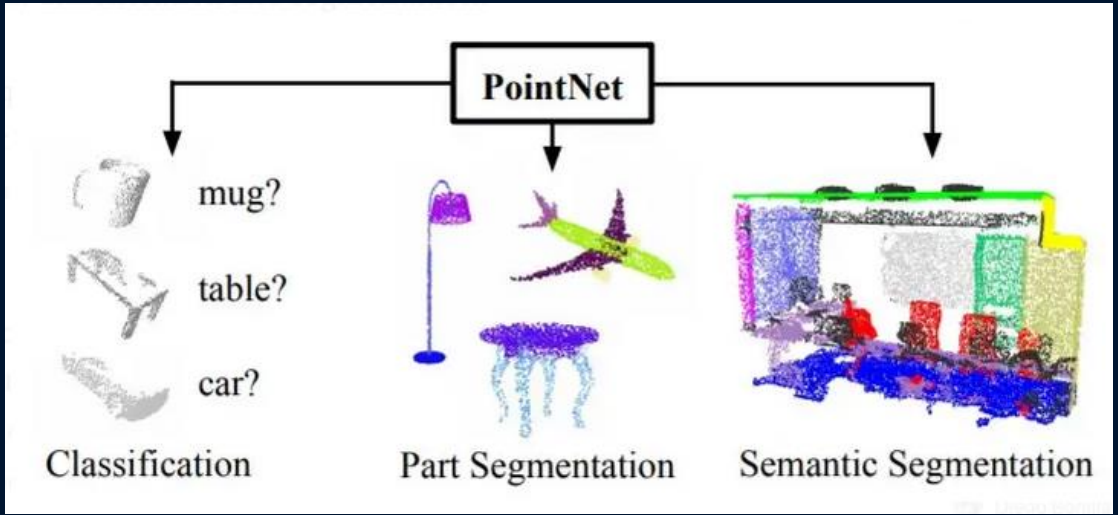
This cloud space is obtained by scanning objects or their structure.
For example, by using laser scan, we can send light pulses and calculate the amount of time each pulse takes for the lasers to return to the scanner.
And this finally specifies the exact position of each object data.
And these data will eventually create the point cloud.



There are two major types of problems that are common for doing machine learning on point clouds:

Classification and Segmentation

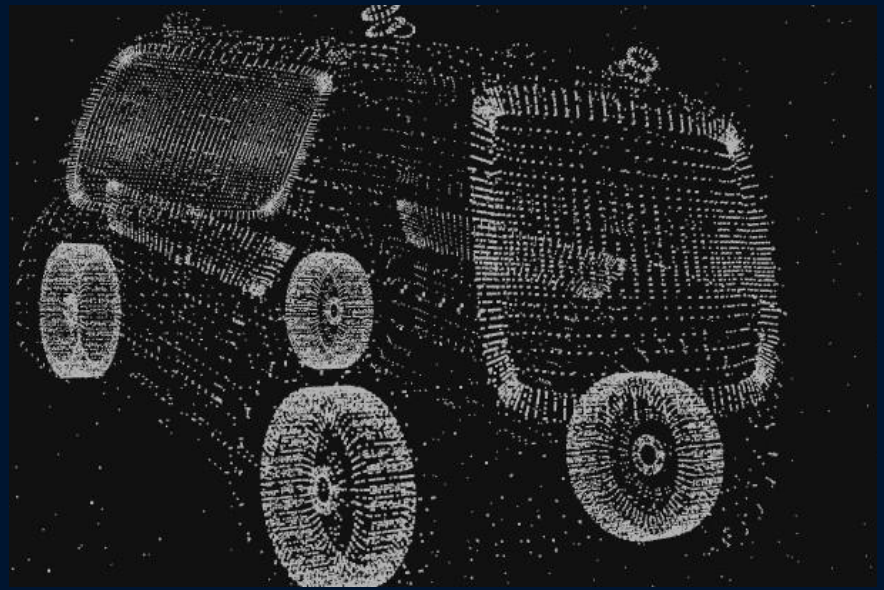
Classification asks the question: *What type of object is this?* The goal is to classify the entire point cloud with one label. There can be two labels (i.e. is this data of a cat or a dog?) or multiple (i.e. is this data of a car, plane, boat, or bike?).



Segmentation asks the question: *Can you separate this object into distinguished parts?* If we have a point cloud describing a bike, maybe we want to separate the wheels, handles, and seat (Part Segmentation). Segmentation is also used for handling complex point clouds that describe an entire environment rather than a single object. For example, we may have a point cloud describing a traffic intersection, and want to distinguish each individual car, person, and stoplight

How to solve?

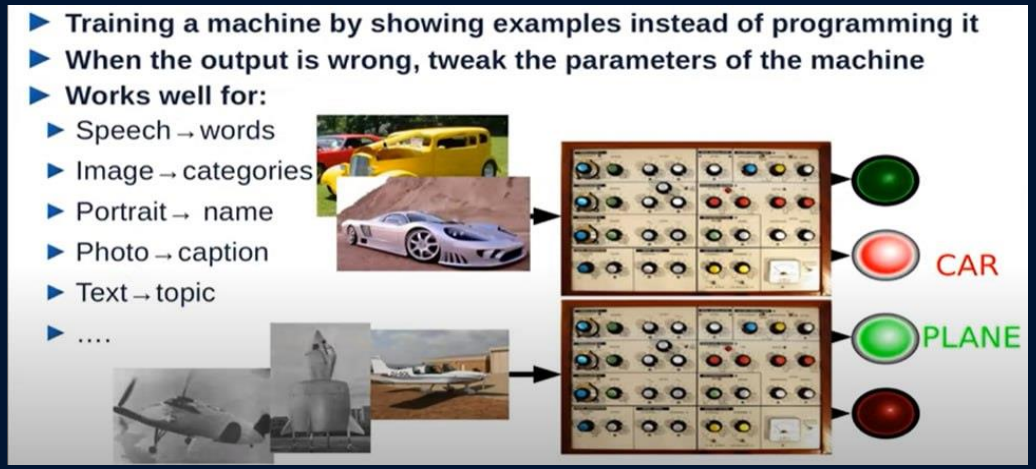
Basic idea : What we do have is a metric, which tells us the distance between points. This allows us to group together points that are close to each other in tiny pockets, which are then compressed into a single point. We can repeatedly apply this principle to summarize the geometric information and ultimately label the entire point cloud.



Energy-based learning & energy-based gan model

views the discriminator as an energy function that attributes low energies to the regions near the data manifold and higher energies to other regions

classical way to do machine learning



- all based on something called <u>supervised running</u>
- you want to build the machine that classifies
- if it produces the right answer don't touch the buttons and if it doesn't produce you touch the buttons so that the answer gets closer to the one you want.
- works well but it requires a lot of data and label data where the desired output provided by humans.

backpropagation as lagransian optimization

Here is an interesting connection with classical mechanics. You can drive backpropagation as basically as if we're a Lagrangian problem which is directly lifted out of classical mechanics.

- **Loss**

$$L(x, y, w) = C(z_K, y) \text{ such that } z_{k+1} = g_k(z_k, w_k), \quad z_0 = x$$

- **Lagrangian for optimization under constraints**

$$L(x, y, z, \lambda, w) = C(z_K, y) + \sum_{k=0}^{K-1} \lambda_k^T (z_{k+1} - g_k(z_k, w_k))$$

You have the less function which is something that measures the discrepancy between the output you want and the output you get at the end of your chain functions. But you have a constraint to satisfy which is that the input to block number k+1 must be equal to the output of block number k and then you know z zero is equal to the input. So, you can write this as a Lagrangian.

- **Loss**

$$L(x, y, w) = C(z_K, y) \text{ such that } z_{k+1} = g_k(z_k, w_k), \quad z_0 = x$$

- **Lagrangian for optimization under constraints**

$$L(x, y, z, \lambda, w) = C(z_K, y) + \sum_{k=0}^{K-1} \lambda_k^T (z_{k+1} - g_k(z_k, w_k))$$

Here is the Lagrangian function of all kinds of variables. C is the cost you want to optimize and sigma is the set of constraints that means the input of stage k+1 equals the output of stage k multiplied by a Lagrange multiplier which you know is a vector because in the parentheses we have a vector.

And then you write the optimality conditions:

- **Optimality conditions:**

$$\frac{\partial L(x,y,z,\lambda,w)}{\partial z_k} = 0, \quad \frac{\partial L(x,y,z,\lambda,w)}{\partial \lambda_k} = 0, \quad \frac{\partial L(x,y,z,\lambda,w)}{\partial w_k} = 0$$

then we have dL / dz(k) and forward propagation will be:

$$\frac{\partial L(x,y,z,\lambda,w)}{\partial \lambda_k} = z_{k+1} - g_k(z_k, w_k) = 0 \Longrightarrow z_{k+1} = g_k(z_k, w_k)$$

And then we have dL dz(k) . The interesting formula here if you rewrite it basically tells you this is backpropagation:

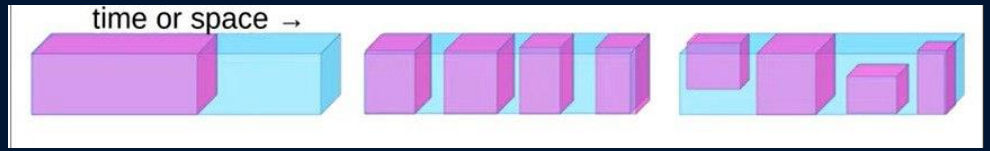
- **Backprop!**
  - Lambda is the gradient

$$\lambda_{k-1} = \frac{\partial g_{k-1}(z_{k-1}, w_{k-1})}{\partial z_k}^T \lambda_k$$

$$\frac{\partial L(x, y, z, \lambda, w)}{\partial w_k} = \lambda_{k+1}^T \frac{\partial g_k(z_k, w_k)}{\partial w_k}$$

And this is just classical mechanics and nothing more.

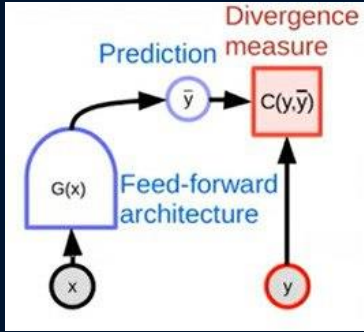
Self-supervised learning : running by prediction running by filling in the blanks



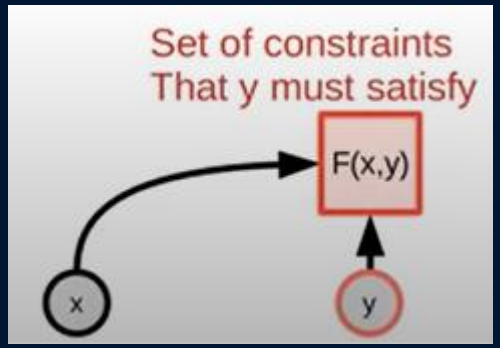
You have a partial view of a piece of data eventually you see the whole thing and you can correct your model of the world to predict what was actually observed. that can be used for two things:

- Learning hierarchical representations of the world
- Learning predictive (forward) models of the world

But the big question here is we're just talking about this within is how do we represent uncertainty and multimodality in the prediction. That's were energy based models com in.



you just get an blury mess as an prediction.



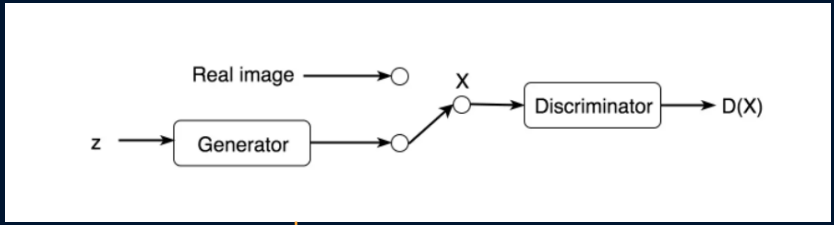
What we are going to do is we're going to replace g(c) by an implicit function an energy function . We are going to connect an energy function f to f of x y and the role of this energy function is to measure the compatibility between x and y. If y is not a good continuation for x f of x y would be large (high energy). So f measures the incompatibility between x and y. The inference process if you really want to make a prediction for y would consist in given an x finding a y that minimizes this energy function.

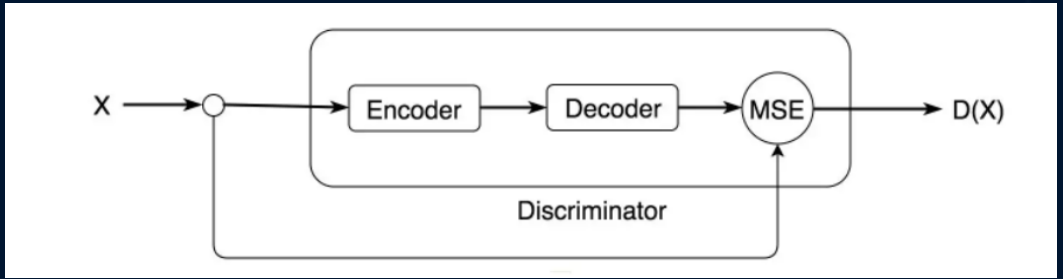
Energy-based model

GAN is a competitive game in which opponents take action to undercut the other. This makes GAN much harder to converge, if it ever happens, than other deep learning models.

sometimes, the generator can hardly fool the discriminator , so it rebuilds itself and learns until the generator cost is so low that the photos it produces are all the same. That is, in order to keep its cost low, it continuously produces only one photo and the discriminator cannot recognize it anymore.

That means although GAN produces data, its range is very limited and its features are not distributed because its cost is so low that discriminator cannot give it a penalty.





Instead of designing a discriminator similar to a classifier, the discriminator uses an **autoencoder** which extracts latent features of the input image by an encoder and reconstruct it again with the decoder.

This discriminator outputs the reconstruction error (mean square error MSE) between the input image and the recreated image instead of a probability value in the original GAN.

$$D(x) = ||Dec(Enc(x)) - x||$$

we use real images to train the autoencoder to reconstruct images. For a poorly generated image, the reconstruction has a very high error because it misses all the latent features needed by the decoder. If the generated image looks very real, its reconstruction cost will be low. Even it is less obvious, the reconstruction cost is an indirect but good metric to critic the generated images.

Objective function :

To train the discriminator, its cost function composes of two goals:

- a good autoencoder: we want the reconstruction cost ***D(x)*** for real images to be low.
- a good critic: we want to penalize the discriminator if the reconstruction error for generated images drops below a value ***m***.

For the generator, we want to lower the reconstruction error for the generated images. Here is the cost function for the discriminator and the generator.

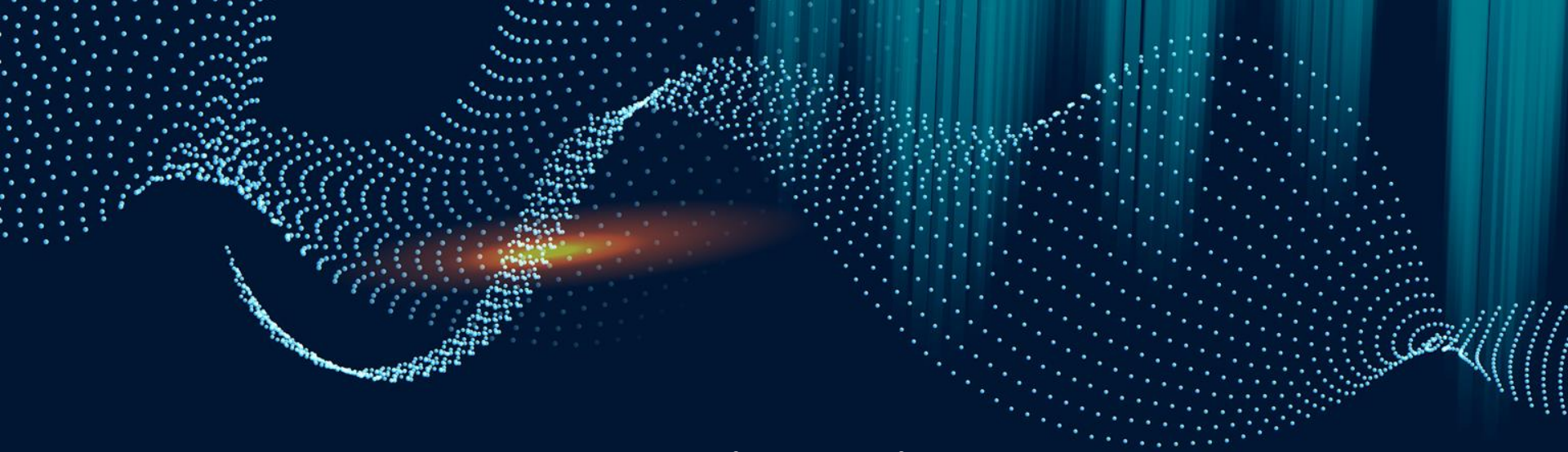
$$\mathcal{L}_D(x, z) = D(x) + [m - D(G(z))]^+$$
$$\mathcal{L}_G(z) = D(G(z))$$

where $[\,u\,]^+ = max(0, u)$

D *is the reconstruction loss and* ***L*** *is the loss function. Don't confuse by them.*

At first, the reconstruction errors for generated and real images are high. The loss in the discriminator comes from the reconstruction loss of the real images since the hinge loss ([u]+) is zero. The discriminator concentrates on creating a good autoencoder for real images. In parallel, the generator learns from the reconstruction errors from the generated images to improve itself. This reconstruction error will gradually drop.

Once, it falls below ***m***, the discriminator will also focus on locating features that discriminate generated images. These criteria focus on extracting latent features of real images while locating what generated images are missing. It is less vulnerable to greedy optimizations that draw too much conclusion on a small set of features. To lower the cost of the discriminator, it needs to capture enough features to reconstruct real images.



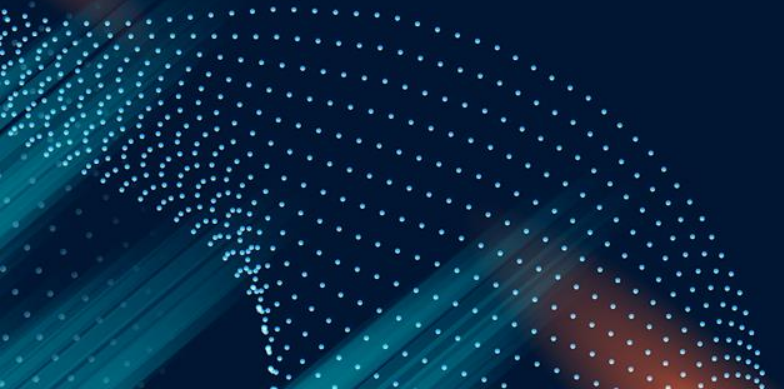
Generative pointnet

Deep Energy-Based Learning on Unordered Point Sets for 3D Generation, Reconstruction and Classification

Overview of our model

This works trying to learn an energy-based model on point clouds data. This is the first deep generating model that provides an explicit density function for point clouds data. It's also the first energy-based model that can perform point cloud synthesis, reconstruction and interpolation. Compared to other generative models, this model does not rely on any extra assisting network for training. It can be derived from the discriminative point net. It also unifies explicit and implicit representation of point clouds in a single framework that is the energy-based model and a later variable model

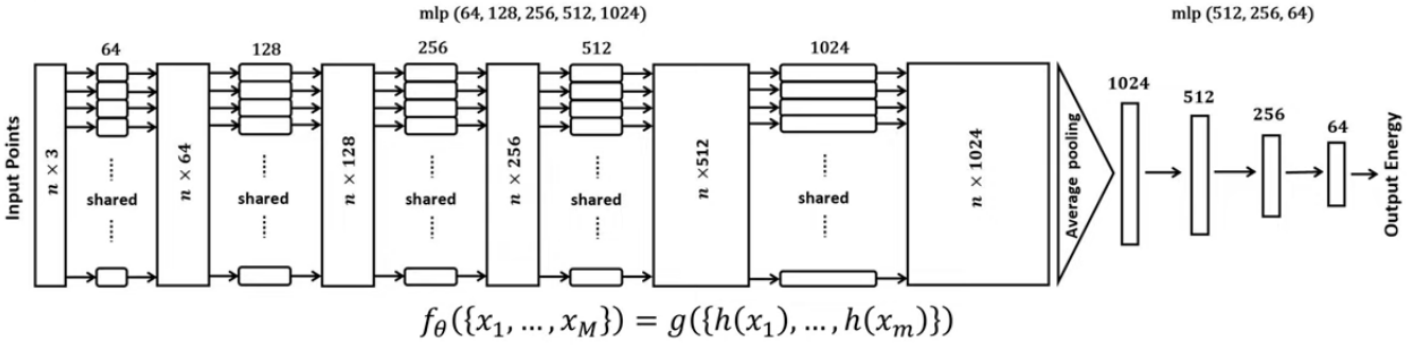
In experiments our model performs competitive performance. With fewer parameters compared to other generating models.



Energy-based model

$$p_\theta(X) = \frac{1}{Z(\theta)} \exp f_\theta(X)\, p_0(X)$$

We define an explicit probability distribution of 3d point cloud as energy-based model l. The probability p is equal to the exponential of the score function times the reference Gaussian distribution divided by an intractable normalization constant. the score function f is parameterized by
a button-up input pigmentation environment neural network
shown in the bottom. it first performed a nonlinearity transformation on point and then followed by a symmetric function g. which is typically an average pooling
Followed by multi-layer perception network.



Maximum likelihood estimate :

$$l(\theta) = E_{q_{data}}[\log p_\theta(X)] \approx \frac{1}{n}\sum_{i=1}^{n} \log p_\theta(X_i)$$

Equivalent to minimize the kullback-leibler divergence :

to train this model we perform maximum likelihood estimation. which is equivalent to minimize the kullback-leibler
divergence between the data distribution q and the ebm to be trained the gradient
of the chao divergence is approximately eguals to
the average score function gradient over training data minus that over samples from the ebm

$$\frac{\partial}{\partial\theta} KL\left(q_{data}(X) \| p_\theta(X)\right)$$

Use MCMC sampling

$$= E_{q_{data}}\left[\frac{\partial}{\partial\theta} f_\theta(X)\right] - E_{p_\theta}\left[\frac{\partial}{\partial\theta} f_\theta(X)\right] \approx \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial\theta} f_\theta(X_i) - \frac{1}{n}\sum_{i=1}^{n} \frac{\partial}{\partial\theta} f_\theta(\tilde{X}_i)$$


$\sim p_{data}$


$\sim p_\theta$

Sampling – langevin dynamics

to perform sampling we use large wind dynamics and MCMC sampling measured it starts from a noise initialization followed by a k-step transformation with noise since the shuffle MCMC is not convergent the sample x is highly dependent to its initialization z so we can regard a sharper MCMC procedure as a k-layer flow based generator model or a latent variable model with z being a continuous latent variable in the reconstruction we can reconstruct x by finding z to minimize the reconstruction error that is the l2 norm between the data and the sampled one

$$X_0 = N(0, \sigma^2)$$

$$X_{\tau+1} = X_\tau - \frac{\delta^2}{2}\frac{\partial}{\partial X}\mathcal{E}_\theta(X_\tau) + \delta U_\tau$$

Transformation Noise

where $U_\tau \sim N(0,1)$;

$$\tilde{X} = M_\theta(Z, \xi), \qquad Z \sim p_0(Z)$$

$$L(Z) = \|X - M_\theta(Z)\|^2$$

Learning process:

The model can be learned by maximum likelihood estimation (MLE), which tries to find model parameters that can assign higher scores (i.e., lower energies) to those observed training point clouds and assign lower scores (i.e., higher energies) to those unobserved ones. The gradient of the log-likelihood is given by

$$-\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{p_{\text{data}}} \left[\nabla_\theta f_\theta(X)\right] - \mathbb{E}_{p_\theta} \left[\nabla_\theta f_\theta(X)\right]$$

The second expectation term is intractable due to the intractable normalizing constant, therefore the learning algorithm relies on Markov chain Monte Carlo (MCMC) sampling, such as Langevin dynamics, to approximate the expectation by the sample average.

The learning algorithm follows “analysis by synthesis”, which iterates the sampling step and the learning step. To be specific, in the sampling step, Langevin dynamics is used to draw samples from the distribution. With the synthesized examples and the observed examples, the model parameters are updated by a gradient-based optimizer in the learning step.

Sampling:

$$\tilde{X}_{t+\Delta t} = \tilde{X}_t - \frac{\Delta t}{2}\nabla_X \mathcal{E}_\theta(\tilde{X}_t) + \sqrt{\Delta t}e_t$$

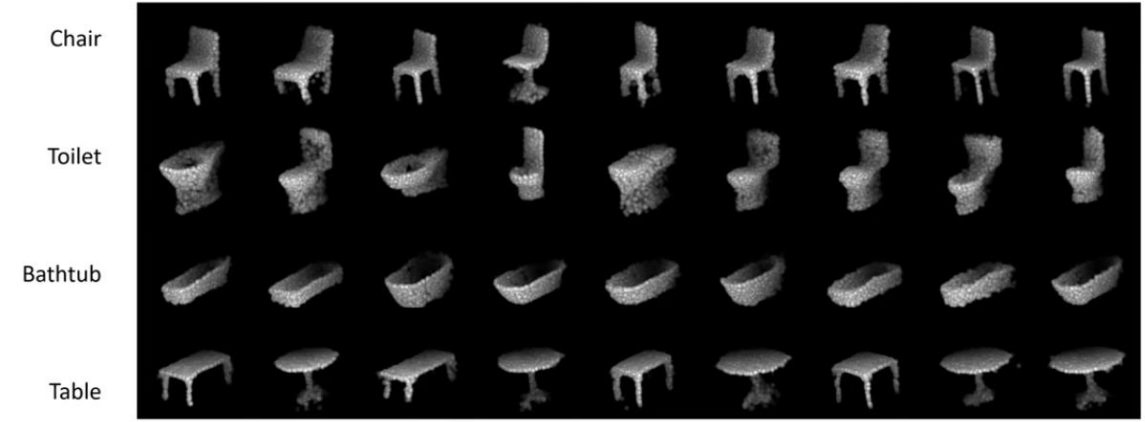
where $\mathcal{E}_\theta(X) = -f_\theta(X) + \|X\|^2/2s^2$ is the energy function.

Learning:

$$\theta_{t+1} = \theta_t + \eta_t \left[\frac{1}{n}\sum_{i=1}^{n}\nabla_\theta f_\theta(X_i) - \frac{1}{\tilde{n}}\sum_{i=1}^{\tilde{n}}\nabla_\theta f_\theta(\tilde{X}_i)\right]$$

There are different implementations of the Langevin dynamics sampling step: (i) **Persistent chain**: runs a finite-step MCMC from the synthesized examples generated from the previous epoch. (ii) **Contrastive divergence**: runs a finite-step MCMC from the observed examples. (iii) **Non-persistent short-run MCMC**: runs a finite-step MCMC from Gaussian white noise. The GPointNet model adopts short-run MCMC to train the model, such that it can unify the generation, reconstruction, and interpolation into a single framework.

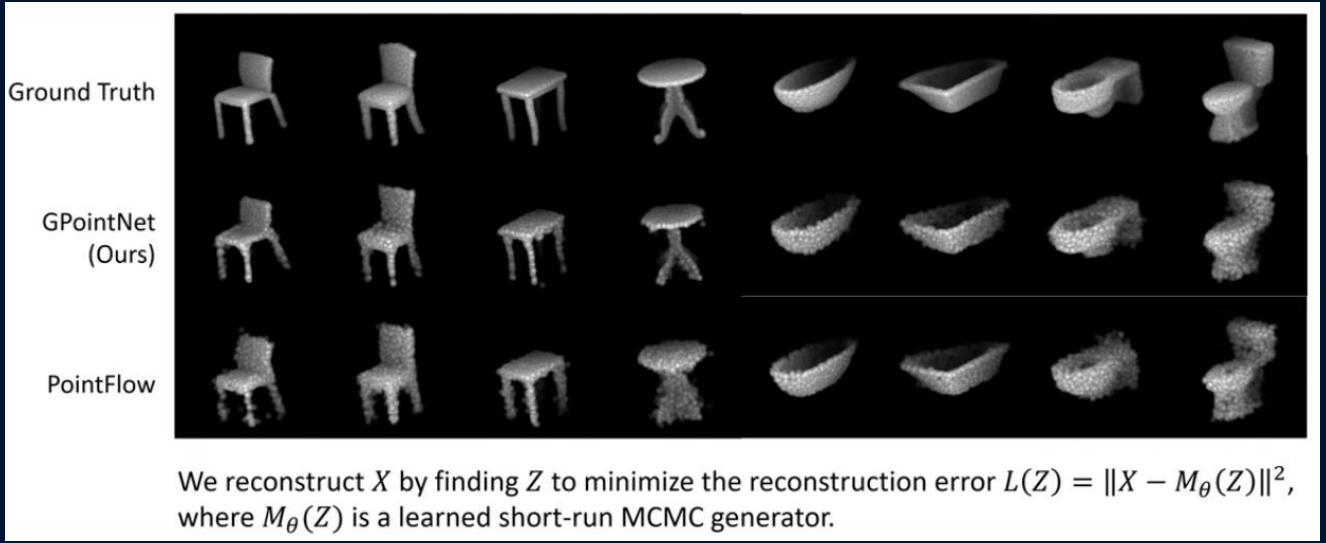
generation



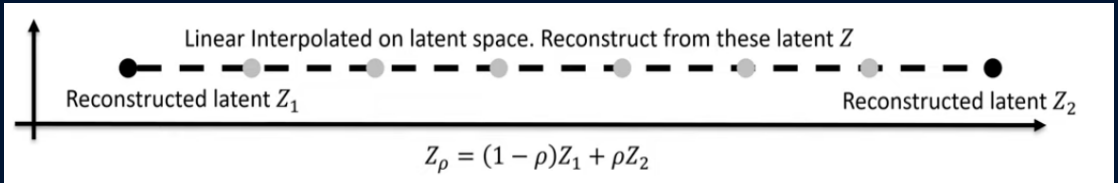
here is our experiment result on generation. we synthesize 3d point clouds by shutter MCMC sampling from the learn model by the modelnet10.

Reconstruction results :

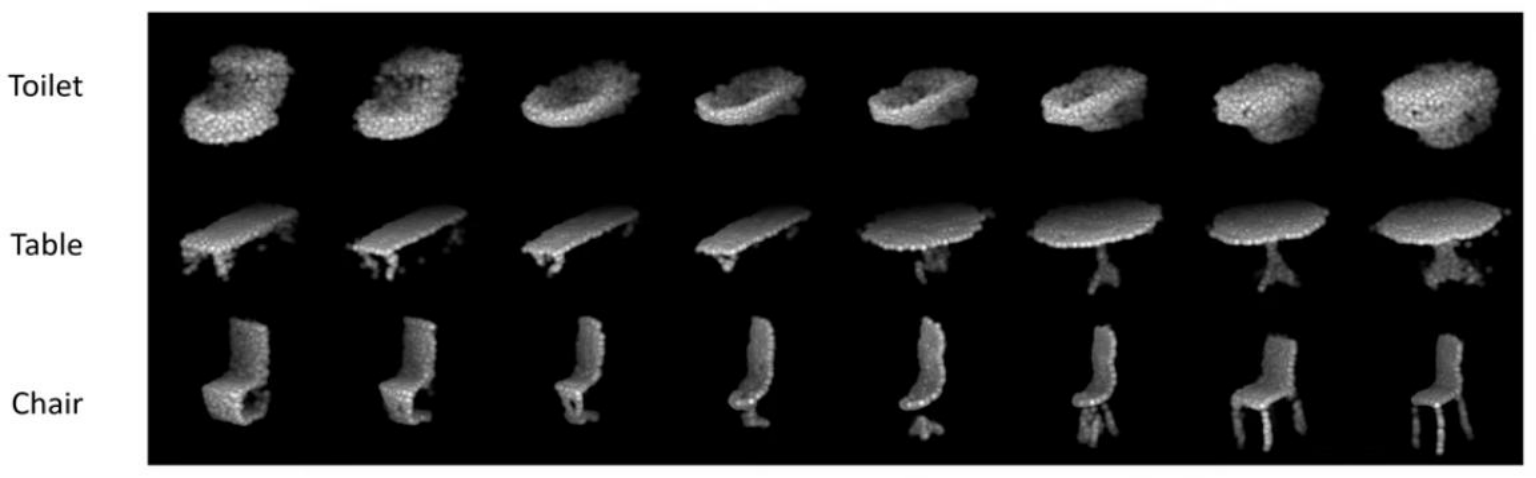
here is the reconstruction result we reconstruct it by finding z to minimize the reconstruction error. the first line is the ground truth the second line is our method compared to the third line which is the output of the point flow. we can see our model is more realistic.



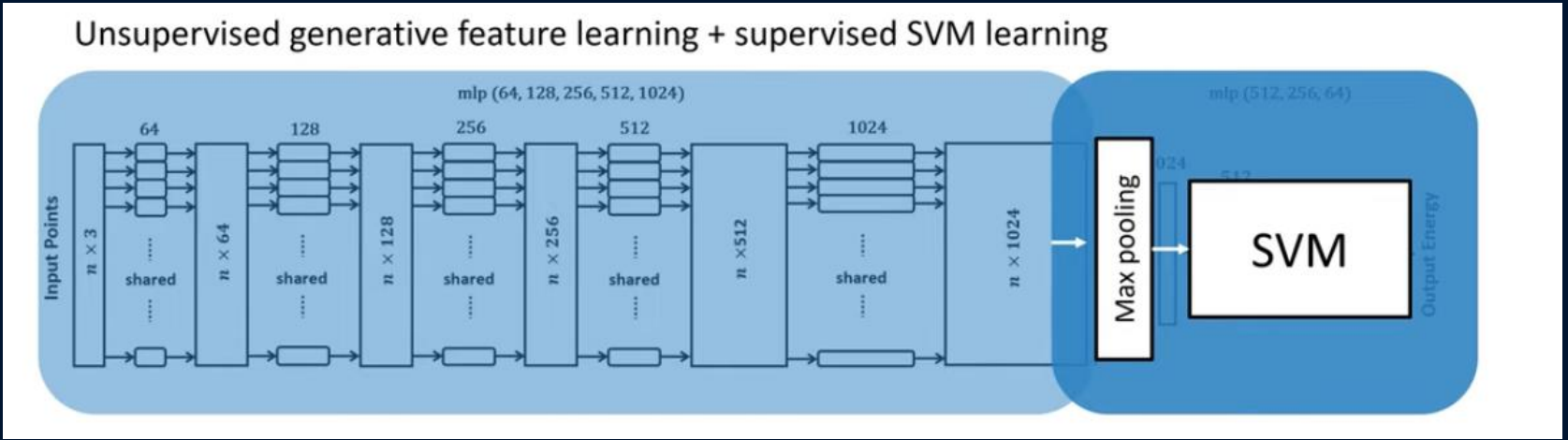
interpolation



our model also have the interpolation abilities. we first sample two noise z1 and z2 then we perform linear interpolation over the latent space L. we can see the smooth transition from left to right it suggests that the generator learns smooth latent space for point cloud embeddings.



classification

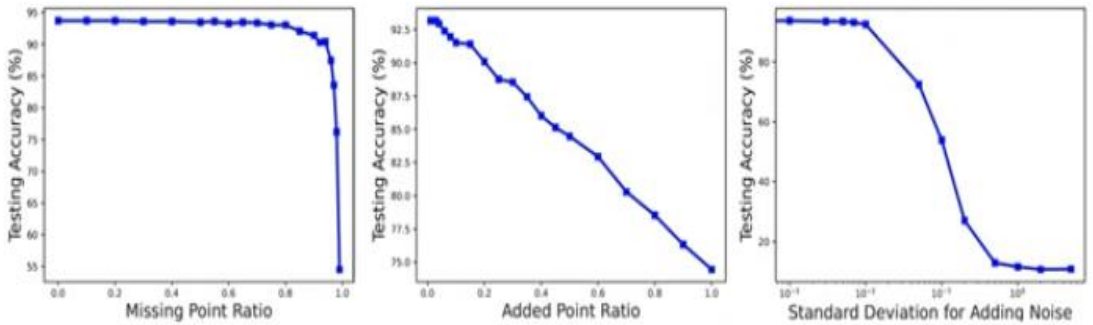


we then provide the classification experiments. we first do unsupervised generative feature learning by learning this ebm. Then, we do supervise svm learning on the learn feature. the result our modelnet10 is comparable to other generated models. we also connect experiments to test the robotness of the classifier we can achieve over 90 accuracy even when using only six percent of the point in a shape or add another 25 point which is produced from the random noise or per tube each point by 0.01 noise

Results on ModelNet10

Method	Accuracy
SPH [18]	79.8%
LFD [4]	79.9%
PANORAMA-NN [33]	91.1%
VConv-DAE [34]	80.5%
3D-GAN [38]	91.0%
3D-WINN [16]	91.9%
3D-DescriptorNet [44]	92.4%
Primitive GAN [19]	92.2%
FoldingNet [51]	94.4%
1-GAN [1]	95.4%
PointFlow [50]	93.7%
Ours	93.7%

Robustness test



Resources :

- https://arxiv.org/abs/2004.01301
- https://jasonlee1980hk.medium.com/generative-pointnet-a-deep-energy-based-models-for-point-clouds-801705411cc0
- https://www.youtube.com/watch?v=4lthJd3DNTM
- http://yann.lecun.com/exdb/publis/pdf/lecun-06.pdf
- https://towardsdatascience.com/understanding-machine-learning-on-point-clouds-through-pointnet-f8f3f2d53cc3