Webinar
Generative AI for 3D content

Athanasios (Thanos) Malamos
Member of Web3D

Hellenic Mediterranean University
Crete-Greece
amalamos@hmu.gr

Web3D Webinar Series - Learn X3D

Fundamentals

What Are Generative Models?

Generative models are a class of machine learning models that **learn to generate new data** similar to the data they were trained on.

Unlike discriminative models (which predict labels), generative models **understand the underlying data distribution**.

Key Features

- **Unsupervised learning:** They often don't require labeled data.
- **Creativity:** Can generate completely new samples that never existed before.
- **Complex distributions:** Capable of modeling intricate patterns and dependencies in data.

Why Are They Important?

- **Data Augmentation:** Generate synthetic data for training other models.
- **Anomaly Detection:** By learning the normal data distribution, they can detect unusual data.

Main Types

Generative models are typically split into **two main categories**:

Likelihood-Based Models

- ➔ Learn the **probability distribution of data**.
- ➔ Optimize a likelihood function (maximize how probable the observed data is).
- ➔ Examples:
  - ◆ **Variational Autoencoders (VAEs)**
  - ◆ **Normalizing Flows (NFs)**
  - ◆ **Diffusion Probabilistic Models (DPMs)**
  - ◆ **Energy-Based Models (EBMs)**

Likelihood-Free Models

- ➔ Do **not directly optimize likelihood**.
- ➔ Generate data via a competition or adversarial process.
- ➔ Example: **Generative Adversarial Networks (GANs)**

Likelihood-based: Try to “fit” a model so it matches the real data distribution as closely as possible.

Likelihood-free: Use a game-like mechanism where one model tries to fool another (GANs) to produce realistic data.

Generative Adversarial Networks (GANs)

Consist of **two neural networks**:

1. **Generator (G):** Creates synthetic data from a latent code z~pz **Goal:** produce realistic data to **fool the discriminator**.

2. **Discriminator (D):** Distinguishes real data x~px from generated data G(z) **Goal:** correctly label generated data as fake and real data as real.

Formulation as a Min-Max Game:

The interaction can be expressed mathematically as a **two-player optimization**

$$\mathcal{L}_D = -\mathbb{E}_{\mathbf{x}\sim p_{\mathbf{x}}}[\log(D(\mathbf{x}))] - \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))],$$
$$\mathcal{L}_G = -\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(D(G(\mathbf{z})))].$$

Discriminator tries to maximize its accuracy in distinguishing real vs fake. Generator tries to minimize the discriminator's success (i.e., generate realistic data).

Nash equilibrium:

- The generator produces realistic enough data that the discriminator cannot reliably tell real from fake.
- Neither network can improve its outcome without the other changing strategy.

Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) are a type of **Deep Latent Variable Model (DLVM)**.

VAEs are **encoder-decoder models** with a probabilistic twist:

1. **Encoder ($q\phi(z|x)$):**
   Takes input data x and maps it to a latent variable z (a latent variable is a hidden variable that represents underlying features or factors of the data that are not directly observed)
   Approximates the intractable posterior $p\theta(z|x)$

2. **Decoder ($p\theta(x|z)$)**
   Takes z and reconstructs the input .
   The reconstruction is compared with the original input to measure performance.

3. **Loss function**

$$\mathcal{L}_{VAE} = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) + \mathbb{E}_{\mathbf{z}\sim q_{\phi(\mathbf{z}|\mathbf{x})}} \log(p_\theta(\mathbf{x}|\mathbf{z}))$$

KL divergence: keeps the latent space close to the prior.

Reconstruction term: encourages accurate reconstruction.

Generating New Data with VAEs

Suppose we have a 3D point cloud representing an object (e.g., a chair). Each point in the cloud has **coordinates (x, y, z)**.

1. **Latent Variables (z):**
    - The VAE encoder maps the point cloud to a **latent vector z**.
    - z captures the **underlying features** of the object, such as:
        - Overall **shape** (e.g., chair with four legs vs stool)
        - **Size or scale**
        - **Orientation or tilt**
        - **Curvature or surface details**
2. **Generating New Point Clouds:**
    - Sample a new $z \sim p_z(z)$ from the prior.
    - The decoder maps z back to a **3D point cloud**.
    - The generated point cloud is **similar to training objects** but unique.

Adjustments via z

Small changes in z lead to variations in:

- Leg thickness or length
- Backrest height
- Overall proportions

These "adjustments" are **implicitly encoded in the latent vector**; the decoder automatically translates them into the point cloud coordinates.

Diffusion Models (DMs)

A Markov chain is a type of mathematical model that describes a sequence of events where the probability of each event depends only on the state attained in the previous event (not on the full history).

Basic Idea

- Parameterized by a **Markov chain**.
- Gradually **adds noise** to input data x0 over **T time steps** with a noise schedule β1:T.
- Theoretically, as T→∞, the final noisy state xT becomes a **normal Gaussian distribution**.

Forward Process (Adding Noise): Represents how noise is added step by step to the data.

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}).$$

Reverse Process (Generating Data)

Goal: reconstruct the original data by reversing the noise process (True posterior q(xt−1|xt) is intractable).

Solution: approximate with a parametric model pθ(xt−1|xt).

Energy-Based Models (EBMs)

Basic Idea

Use an energy function Eθ(x) to model the probability distribution of the data:

$$p(\mathbf{x}) = \frac{\exp(-E_\theta(\mathbf{x}))}{\int_{\mathbf{x}} \exp(-E_\theta(\mathbf{x}))}$$

Low energy → high probability data points (likely data = keep / accept).
High energy → low probability data points (reject or modify).

Example: How EBMs Generate a 3D Chair Point Cloud

1. Training Phase
    - The EBM learns an energy function $E\theta(x)$ from the training 3D chair point clouds.
    - Low energy → realistic chairs, high energy → unrealistic shapes.
2. Sampling / Generation Phase
    - Start with a random point cloud.
    - Use Markov Chain Monte Carlo (MCMC) to iteratively modify the points:
        - Propose small changes to points (e.g., moving a vertex slightly).
        - Evaluate energy of the new configuration.
        - Accept changes if they lower the energy (make it more realistic).
3. Resulting Point Cloud
    - After multiple iterations, the points converge to a configuration with low energy, representing a new, realistic chair.
    - Features like legs, backrest, and seat are implicitly shaped by the energy function learned from the training data.

Voxel Grids

Voxel Grids

Definition

Voxel grids are like 3D images, where each voxel stores information such as:

- Geometry occupancy
- Signed distance values
- Density values

These representations define the shape surface as a level-set function.

The regular grid structure of voxels makes them suitable for 3D CNNs and early deep learning tasks, such as 3D classification, detection, and segmentation.

Voxel Grids and Data Generation (1/2)

Generative Use Case in 3D ShapeNets

- **Step 1:** Input conversion
  A depth map is taken as input and converted into a 3D voxel grid (like a 3D pixel representation of the scene).

- **Step 2:** Processing by Deep Belief Network
  This voxel grid is fed into a convolutional deep belief network (DBN). A DBN is a probabilistic generative model that explicitly learns a distribution over possible 3D shapes.

- **Step 3:** Learning a distribution
  Because the DBN models the probability distribution of voxel configurations, it doesn't just recognize or classify shapes — it learns the underlying likelihood of different 3D structures.

- **Step 4:** Generation
  Once trained, the model can sample from this learned distribution, generating entirely new 3D voxel-based shapes.
  This makes ShapeNets a likelihood-based generative model.

Voxel Grids and Data Generation (2/2)

3D-R2N2

- **Step 1: Input image**
  The model takes a 2D image of an object as input.

- **Step 2: Encoding**
  A 2D CNN processes the image and encodes it into a latent vector, a compact representation that captures shape information.

- **Step 3: Decoding into voxels**
  This latent vector is passed into a 3D CNN decoder, which predicts a 3D voxel grid representing the full shape.

- **Step 4: Generative use**
  Since the model works in latent space, it isn't restricted to reconstructing from input images. We can also sample new latent vectors (instead of encoding from a real image).

- **Step 5: New data generation**
  These sampled latent vectors can then be decoded into voxel grids, producing new 3D shapes.

Generative Rendering Tasks with Voxel Grids

Early methods

- Store high-dimensional feature vectors at each voxel.
- These feature vectors encode the scene geometry and appearance.
- The stored information is then interpreted into color images using projection and 2D CNNs.
- This allows the generation of rendered images from voxel-based 3D data.

Neural Volumes

- Uses CNNs to predict RGB-Alpha volumes.
- The predicted volumes are synthesized into images through volume rendering techniques.
- Enables generating realistic 2D images directly from volumetric data.

Multi-Plane Image (MPI)

- Treated as a variation of voxel grids.
- The 3D space is divided into several depth planes, each with an RGB-Alpha image.
- By reducing the number of voxels along the depth dimension, computational cost is lowered.
- Still supports generative rendering but in a more efficient way.

3D GANs for Voxel Grids

Generator

- Maps a high-dimensional latent vector to a 3D voxel cube.
- Describes the synthesized object in voxel space.
- Constructed using a sequence of 3D convolution blocks to handle 3D voxel data.

Discriminator

- Built with **3D convolution blocks**.
- Evaluates the realism of voxel grids (real VS generated).

Encoder (VAE-GAN-like design)

- Maps a **2D image** into the latent space of the generator.
- Trained with multiple losses:
  1. **Adversarial loss** – ensures realism.
  2. **Reconstruction loss** – ensures the encoder-decoder reproduces shapes correctly.
  3. **KL divergence loss** – constrains the output distribution of the encoder.

Capabilities

- Can **recover 3D shapes from 2D images**.
- Learned **discriminator can be transferred** to other 3D downstream tasks.

Comparison to 2D GANs

- Generator and discriminator are **3D convolutional networks**, instead of 2D.
- Encoder added for mapping 2D images into latent space, resembling a **VAE-GAN**.

VAE Approaches for Voxel Grids

Encoder-Decoder Architecture

- **Encoder**:
  - Consists of four 3D convolutional layers followed by a fully connected layer.
  - Maps the input voxel grid into a latent vector and then the 3D decoder with an identical but inverted architecture to transform a latent vector into a 3D voxel.
  - Uses stride convolutions as a downsampler to reduce spatial dimensions.

- **Decoder**:
  - Mirrors the encoder with an inverted 3D convolutional architecture.
  - Converts the latent vector back into a 3D voxel grid.
  - Uses stride convolutions as an upsampler to reconstruct the voxel shape

- **Objective Function (Loss)**

  Two components:

  1. **KL divergence** on the latent codes – ensures the latent space follows a desired distribution.
  2. **Binary Cross-Entropy (BCE) for voxel reconstruction** – measures how well the decoder reconstructs the input.

VQ-VAE + Autoregressive Models for Voxel Grids

Motivation

- Standard VAEs use a single latent vector for the input, which can produce blurry voxel outputs.
- VQ-VAE addresses this by projecting the high-dimensional 3D shape into a lower-dimensional discrete latent space.
- This latent space is optimized during training, unlike the fixed latent of standard VAEs.

Autoregressive Modeling

- Once the latent space is trained, a transformer is used to autoregressively model the non-sequential data.
- The model maximizes the likelihood of latent representations using randomized orders for an autoregressive generation.

Generative Capabilities

- Can perform shape completion and text-guided shape generation.
- Incorporation of conditional information into the autoregressive model allows straightforward fusion with the model.

Point Clouds

Point Clouds

Overview

Point clouds are unstructured sets of 3D points representing the surface of an object.

They are direct outputs of depth scanners and widely used in 3D scene understanding.

Advantages over voxel grids:

1. Require less GPU memory for processing.
2. Suitable for normalizing flows and diffusion models.

GAN-based Methods for Point Clouds

Overview

- GANs are used to learn the **distribution of 3D point clouds**.
- Pioneer work: **Achlioptas et al. [27]**, introducing two types of GANs:

1. **r-GAN (raw point cloud GAN)**
   Generator: MLP with 5 fully connected layers Maps a randomly sampled noise vector to a point cloud with 2048 points. Discriminator: PointNet backbone.
   Limitation: Hard to generate high-quality point clouds. A plausible reason is that GANs are hard to converge to a good point.

2. **l-GAN (latent-space GAN)**
   Solution to r-GAN limitations.
   Trains a GAN to model the latent space of a pre-trained auto-encoder.
   Advantage: Better performance and convergence than r-GAN.

AAE, VAE-based Methods and Normalizing Flow Methods for Point Clouds

3D VAE and Adversarial Auto-Encoder (AAE):

- Encoder: PointNet-like network.
  Decoder: MLP network.
- Losses: Set-to-set matching (Earth Mover's, Chamfer distance) and KL divergence for latent space supervision.
- **AAE** model adopts an alternative approach by learning the latent space through adversarial training

3D AAE model outperforms the 3D VAE model in terms of performance

Normalizing Flow Methods

- **PointFlow**:
  - Models point clouds as a distribution of distributions.
  - Samples points from a generic prior (Gaussian), then applies conditional continuous normalizing flow (CNF) which produces a vector field to move the sampled points to generate the shape.
  - Uses an additional CNF to model shape priors for better performance.
- **SoftFlow**:
  - Perturbs points with noise and uses conditional normalizing flows to map points to latent variables

Diffusion and Denoising Methods for Point Clouds

Overview

- Point cloud generation can be formulated as a denoising process.
- Models are trained to output vector fields that gradually move points from a generic prior distribution (e.g., noise) toward the target point cloud distribution.

ShapeGF

- Treats the shape as a distribution.
- Assumes points on the shape surface have high densities.
- For any 3D point:
  - Trains a network to predict the point's gradient.
  - This gradient is used to move points toward high-density areas on the shape surface.

Reverse Diffusion Methods

- Generation is formulated as a reverse diffusion process, opposite to diffusion in non-equilibrium thermodynamics.
- Implemented as a Markov chain:
  - Transforms the distribution of points from noise to the target distribution.
  - Each transition step uses an autoencoder that:
    - Takes a point cloud as input.
    - Outputs displacements of points to move them closer to the target shape.

Neural Fields

Neural Fields

Overview

Neural fields use neural networks, mostly MLPs, to predict properties for any point in 3D space.

Advantages:

- Can model shapes of arbitrary spatial resolution.
- More memory efficient than voxel grids or point clouds.

Challenges:

- Applying standard generative models like GANs is not straightforward.
- Issues: lack of ground-truth data in neural representation format and difficulty in processing neural representations directly.

GAN-Based Methods on Neural Fields (1/2)

Latent-Space GANs

- **Concept:** Since raw neural fields don't have ground-truth 3D grids, GANs are applied not directly on the field, but in a latent feature space.
- **Workflow:**

    1. **Auto-encoder:**
        - Encoder maps a set of 3D shapes to latent vectors (features).
        - Can use a 3D CNN or PointNet for the encoder.
        - Decoder is typically an MLP network that reconstructs the shape from latent vectors.

    2. **Latent GAN Training:**
        - GAN is trained to generate new latent vectors that resemble those encoded from real shapes.
        - The discriminator distinguishes between real latent codes (from encoder) and fake latent codes (from generator).

- **Benefit:** Works around the lack of explicit ground-truth voxel/point data and allows standard GAN training in latent space.

GAN-Based Methods on Neural Fields (2/2)

Hybrid Representations

Concept: To improve spatial control and expressiveness, latent representations can combine **voxel grids + implicit fields**.

Mechanism:

- Each cell in the hybrid representation corresponds to a portion of the shape.
- Latent-GAN is trained on these hybrid features.

Advantage: Provides spatially aware generation, letting the GAN control detailed parts of the shape more effectively.

Signed Distance Field (SDF) Discriminators

Two types of discriminators for training GANs on neural fields:

1. **Voxel-based discriminator:**
   - Queries a **fixed number of points** in the SDF to evaluate the generated shapes.
   - Useful when working with structured grid representations.

2. **Point-based discriminator:**
   - Can handle **arbitrary points** for evaluating SDF values.
   - Suitable for unstructured or irregular shape representations.

Meshes

Challenges with Mesh Generation

Non-Euclidean structure: Meshes cannot be directly processed by standard convolutional neural networks.

Vertex connectivity: Generating plausible connections between mesh vertices is difficult, which is crucial for realistic shapes.

Generator Models for Meshes

Multi-chart structure:
Defines landmark points on a base mesh to parameterize the surface as image-like charts. By doing so, the generator can use image GAN techniques to synthesize meshes through these chart-based mappings.

SDM-Net:
Uses a variational auto-encoder (VAE) framework where the decoder acts as the generator, implemented with convolutional operators on meshes, to reconstruct meshes aligned with a unit cube mesh.

TM-Net:
Extends SDM-Net with a generator that not only reconstructs mesh geometry but also synthesizes texture maps using a CNN-based VAE decoder, combining both to output textured meshes.

PolyGen:

Employs an auto-regressive generator with a transformer-based network.

- Vertex transformer: generates vertices sequentially (sorted along vertical axis).
- Face transformer: conditioned on generated vertices, predicts mesh faces.

ShapeAssembly:

Uses a sequence VAE where the decoder is the generator. It applies a GRU to sequentially generate program line features, which are mapped via MLPs into program instructions that build meshes.

Thank you

Athanasios (Thanos) Malamos
Member of Web3D

Hellenic Mediterranean University
Crete-Greece
amalamos@hmu.gr