Generative Models Taxonomy

- **Generative Models**
  - **Traditional**
    - GMM
    - HMM
    - Naive Bayes
  - **Deep Learning**
    - VAE
    - GAN
    - Autoregressive

VAE vs GAN

Output Quality	GAN ↑
Training Stability	VAE ↑
Mode Coverage	VAE ↑
Sharpness	GAN ↑

1. GMM (Gaussian Mixture Model)

📌 Basic Principle

GMM is a probabilistic model that assumes data is generated from a **mixture of multiple Gaussian distributions (normal distributions)**. Each Gaussian component represents a specific cluster in the data, and the model calculates the probability that each data point was generated from which component.

$$p(x) = \Sigma \pi_k \cdot N(x \mid \mu_k, \Sigma_k)$$

Here, π_k is the mixing coefficient of the k-th component, μ_k is the mean, and Σ_k is the covariance matrix. Parameters are learned using the EM (Expectation-Maximization) algorithm.

🎯 Main Applications

- **Clustering:** Modeling more flexible cluster shapes than K-means
- **Image Segmentation:** Segmenting images into multiple regions
- **Anomaly Detection:** Detecting data that deviates from normal data distribution
- **Speech Recognition:** Phoneme modeling
- **Density Estimation:** Approximating complex data distributions

💡 **Advantages:** Soft clustering (probabilistic assignment), capable of modeling elliptical clusters
⚠️ **Disadvantages:** Number of components must be specified in advance, performance degrades in high-dimensional data

2. HMM (Hidden Markov Model)

📌 Basic Principle

HMM is a probabilistic model for **time series data**, assuming that observable events are generated by hidden states. The system transitions between invisible states and generates observations from each state.

Core Components:

- **Transition Probability (A):** Probability of moving from one state to another
- **Emission Probability (B):** Probability of an observation occurring in a specific state
- **Initial State Probability (π):** Probability of each state at the start

$$P(O|\lambda) = \Sigma\ P(O|Q, \lambda) \cdot P(Q|\lambda)$$

Uses **Forward-Backward algorithm** (inference), **Viterbi algorithm** (optimal path search), and **Baum-Welch algorithm** (learning).

🎯 Main Applications

- **Speech Recognition:** Recognizing words from phoneme sequences
- **Natural Language Processing:** Part-of-speech (POS) tagging
- **Bioinformatics:** Gene sequence analysis, DNA pattern recognition
- **Gesture Recognition:** Recognizing motion patterns in video
- **Financial Time Series:** Stock pattern and market state analysis

💡 **Advantages:** Models temporal dependencies, can handle incomplete observation data

⚠️ **Disadvantages:** Markov assumption (present depends only on previous state), difficulty in choosing number of states

3. Naive Bayes

📌 Basic Principle

Based on Bayes' theorem, it makes the "naive" assumption that all features are **conditionally independent**. While this assumption is unrealistic, it works surprisingly well in practice.

$$P(y|x_1,...,x_n) = P(y) \cdot \prod P(x_i|y) \;/\; P(x_1,...,x_n)$$

Given class y, it assumes each feature x_i is independent. This allows decomposing high-dimensional joint probabilities into simple products of conditional probabilities, making computation very efficient.

Main Variants:

- **Gaussian NB:** For continuous data, features follow normal distribution
- **Multinomial NB:** For text classification, based on word frequency
- **Bernoulli NB:** For binary features, word presence/absence

🎯 Main Applications

- **Text Classification:** Spam filtering, sentiment analysis, news categorization
- **Document Classification:** Email classification, document categorization

- **Recommendation Systems:** Predicting user preferences
- **Medical Diagnosis:** Calculating disease probability based on symptoms
- **Real-time Prediction:** Classification tasks requiring fast speed

💡 **Advantages:** Very fast and efficient, can learn with small data, easy to interpret
⚠️ **Disadvantages:** Feature independence assumption is unrealistic, ignores correlations between features

4. VAE (Variational Autoencoder)

📌 Basic Principle

VAE is a deep learning generative model that combines an **autoencoder architecture** with **variational inference**. It encodes data into a low-dimensional latent space and reconstructs data from this latent representation.

Core Architecture:

- **Encoder ($q_\varphi(z|x)$):** Maps input data x to probability distribution of latent variable z (mean μ, variance σ^2)
- **Reparameterization Trick:** Samples $z = \mu + \sigma \odot \varepsilon$ ($\varepsilon \sim N(0,1)$) to enable backpropagation
- **Decoder ($p_\theta(x|z)$):** Reconstructs original data from latent variable z

📊 Loss Function (ELBO)

VAE is trained by maximizing the Evidence Lower Bound (ELBO), which balances two terms:

$$L = E[\log p_\theta(x|z)] - KL(q_\varphi(z|x) \,||\, p(z))$$

First term (Reconstruction Loss): How well the decoder reconstructs the original data

Second term (KL Divergence): Difference between the learned distribution and prior distribution (usually N(0,1))

🔑 **Key Insight:** The KL term regularizes the latent space to make it smooth and continuous, enabling meaningful interpolation when generating new samples.

🎨 Properties of Latent Space

VAE's latent space has the following important properties:

- **Continuity:** Similar data points are located close in latent space
- **Interpolability:** Smooth movement between two points generates meaningful intermediate samples
- **Structured:** Specific dimensions control specific attributes (e.g., facial expression, angle)
- **Regularity:** Regularization to prior distribution reduces empty spaces

🎯 Main Applications

- **Image Generation:** Generating new faces, landscapes, artworks
- **Data Augmentation:** Generating transformed images to expand training data
- **Anomaly Detection:** Identifying samples with high reconstruction error as anomalies
- **Dimensionality Reduction:** Capturing non-linear relationships better than t-SNE, PCA
- **Image Editing:** Changing specific attributes by manipulating latent vectors
- **Semi-supervised Learning:** Learning representations with limited labels
- **Drug Design:** Exploring latent space of molecular structures

⚖️ Detailed VAE vs GAN Comparison

VAE Strengths:

- **Stable Training:** Single objective function, no mode collapse
- **Complete Probabilistic Model:** Explicit probability density estimation
- **Interpretable Latent Space:** Learning structured and meaningful representations
- **Mode Coverage:** Better capturing diverse data distributions

VAE Weaknesses:

- **Blurry Output:** Reconstruction loss (MSE) generates averaged images
- **Lower Sample Quality:** Generated images are less sharp than GAN's
- **Prior Distribution Assumption:** Usually assumes Gaussian, which may differ from actual distribution

💡 **Practical Tip:** Use β-VAE (weight β on KL term) to adjust balance between reconstruction and regularization. β > 1 learns more disentangled representations but may reduce reconstruction quality.

🔬 Main Variants

- **β-VAE:** KL weight adjustment for disentangled representation learning
- **Conditional VAE (CVAE):** Adding label information for conditional generation
- **VQ-VAE:** Using discrete latent space, high-quality image/audio generation
- **Hierarchical VAE:** Modeling complex structures with multi-level latent variables

📐 Input/Output Calculation Example

Let's examine a VAE that encodes MNIST handwritten digit images (28×28) into a 2-dimensional latent space.

Network Structure:

- **Input:** 28×28 = 784-dimensional image vector
- **Encoder:** 784 → 400 → (μ: 2-dim, $\log \sigma^2$: 2-dim)
- **Latent space:** 2-dimensional (good for visualization)
- **Decoder:** 2 → 400 → 784
- **Output:** 28×28 reconstructed image

🔄 Forward Pass Calculation Process

Step 1: Input Image

```
x ∈ ℝ^(784) = [0.1, 0.9, 0.8, ..., 0.0] (normalized pixel values)
```

Step 2: Encoder - Hidden Layer

```
h = ReLU(W₁·x + b₁) ∈ ℝ^(400)
```

Example: Using $W_1 \in \mathbb{R}^{(400\times784)}$, $b_1 \in \mathbb{R}^{(400)}$ to compress 784 dimensions to 400 dimensions

Step 3: Encoder - Computing Mean and Variance

```
μ = W_μ·h + b_μ ∈ ℝ² = [1.2, -0.8]
log σ² = W_σ·h + b_σ ∈ ℝ² = [-0.5, -1.2]
```

We predict $\log \sigma^2$ for numerical stability.

Actual standard deviation: $\sigma = \exp(0.5 \cdot \log \sigma^2) = [0.78, 0.55]$

Step 4: Reparameterization Trick

$$\varepsilon \sim N(0, I) \in \mathbb{R}^2 = [0.3, -0.7] \text{ (random sampling)}$$
$$z = \mu + \sigma \odot \varepsilon = [1.2, -0.8] + [0.78, 0.55] \odot [0.3, -0.7]$$
$$z = [1.2 + 0.234, -0.8 - 0.385] = [1.434, -1.185]$$

💡 **Key Point:** By sampling ε, backpropagation becomes possible. We can calculate gradients with respect to μ and σ.

Step 5: Decoder - Hidden Layer

$$h' = \mathrm{ReLU}(W_2 \cdot z + b_2) \in \mathbb{R}^{(400)}$$

Example: Using $W_2 \in \mathbb{R}^{(400\times2)}$, $b_2 \in \mathbb{R}^{(400)}$ to expand 2 dimensions to 400 dimensions

Step 6: Decoder - Output (Reconstruction)

$$\hat{x} = \sigma(W_3 \cdot h' + b_3) \in \mathbb{R}^{(784)}$$

σ is the sigmoid function that constrains output to [0, 1] range.

$\hat{x}$ = [0.09, 0.88, 0.82, ..., 0.01] (reconstructed image)

📊 Loss Calculation

Reconstruction Loss:

$$L\_recon = -\Sigma_i [x_i \cdot \log(\hat{x}_i) + (1-x_i) \cdot \log(1-\hat{x}_i)] \text{ (Binary Cross-Entropy)}$$

Or MSE can be used:

$$L\_recon = \Sigma_i (x_i - \hat{x}_i)^2 / 784$$

Example: L_recon = 0.025 (average error per pixel)

KL Divergence:

$$KL = -0.5 \cdot \Sigma_j \, [1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2]$$

Calculating per dimension:

```
j=1: -0.5 · [1 + (-0.5) - 1.2^2 - e^(-0.5)] = -0.5 · [0.5 - 1.44 - 0.61] = 0.775
j=2: -0.5 · [1 + (-1.2) - 0.8^2 - e^(-1.2)] = -0.5 · [-0.2 - 0.64 - 0.30] = 0.570
KL_total = 0.775 + 0.570 = 1.345
```

Total Loss:

```
L_total = L_recon + KL = 0.025 + 1.345 = 1.370
```

🎯 **Interpretation:**

- Low reconstruction loss (0.025) → Good reconstruction of image
- High KL (1.345) → Learned distribution deviates significantly from standard normal
- As training progresses, both losses will balance out

🎨 Generating New Images

After training is complete, to generate new images:

```
1. Sample from standard normal: z_new ~ N(0, I) = [-0.5, 1.2]
2. Use decoder only: x_new = Decoder(z_new)
3. Generate new handwritten digit image!
```

Interpolation in latent space is also possible:

$$z\_interpolate = \alpha \cdot z_1 + (1-\alpha) \cdot z_2, \quad \alpha \in [0, 1]$$

$$\text{Example: When } \alpha=0.5,\ z = 0.5 \cdot [1.4, -1.2] + 0.5 \cdot [-0.8, 0.9] = [0.3, -0.15]$$

Practical Examples:

- Generate from z = [2.0, 0.0] → Image of '1' with thick lines
- Generate from z = [0.0, 2.0] → Round '0' image
- Generate from z = [1.0, 1.0] → Intermediate form mixing both styles

💡 **Parameter Count Calculation:**

• Encoder: 784×400 + 400 + 400×2 + 2 + 400×2 + 2 = 315,604

• Decoder: 2×400 + 400 + 400×784 + 784 = 315,184

• Total parameters: ~630K (relatively small model)