Overview of Generative AI Models in Computer Vision

Presented by: Yue Yu†

†: Statistics Club, Indiana University

Tentative Schedule of Diffusion Model Series

- **10/10: Overview of generative AI models in Computer Vision, including GANs (Goodfellow et al., 2014), VAEs (Kingma, 2013) and diffusion models (Sohl-Dickstein et al., 2015), and detailed introduction on GANs;**
- **10/17**: Detailed introduction on VAEs and intuition of diffusion models;
- **10/24**: DDPM (Ho et al., 2020), the cornerstone paper about diffusion models, which enables diffusion models to produce high-quality, realistic samples and to be competitive with generative models like GANs and VAEs;

Tentative Schedule of Diffusion Model Series Cont'

- **11/7**: Score-based generative modeling (Song et al., 2020b), which bridges the gap between DDPMs and score-matching, offering a more generalized framework for diffusion-based generative models; and DDIM (Song et al., 2020a), which presents a method to speed up the sampling process in diffusion models without sacrificing much in terms of sample quality;

- **11/21**: Recent years' improvements and applications of diffusion models from different aspects, e.g., Latent Diffusion Models (Stable Diffusion, Rombach et al. (2022)) and its subsequent works like conditional control to text-to-image diffusion (Zhang et al., 2023) and Stable Diffusion XL (Podell et al., 2023).

Today's Presentation Outline

- Development of Generative AI Models in the Past Decade;
- Detailed Introduction on Generative Adversarial Networks (Goodfellow et al., 2014).

Deep Dream by Google

Original Paper: Mordvintsev et al. (2015)

- **Concept:** Deep Dream uses convolutional neural networks (CNNs) to amplify patterns detected in images, creating surreal, dream-like visualizations.
- **Applications:** Artistic image generation, visualization of neural network features.
- **Significance:** Provided a way to interpret and visualize the patterns learned by deep neural networks.

Image Generated by Deep Dream (Nightmare?)

Figure: The Mona Lisa with DeepDream effect using VGG16 network trained on ImageNet, with other animals such as dog and turtle added to the original picture.

Generative Adversarial Networks (GANs)

Original Paper: Goodfellow et al. (2014)

- **Concept:** GANs consist of two neural networks, a generator and a discriminator, trained in a game-theoretic setting.
- **Applications:** Image generation, super-resolution, image-to-image translation.
- **Significance:** First successful architecture for generating high-quality synthetic images.

Architecture of GAN

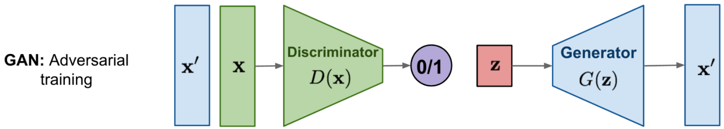

Figure: Architecture of a Generative Adversarial Network (GAN). The generator $G(z)$ takes a random noise vector z as input and produces synthetic data x'. The discriminator $D(x)$ aims to distinguish between real data x and generated data x', providing feedback to improve the generator. The adversarial training process encourages the generator to produce more realistic data over time while the discriminator becomes better at distinguishing real from fake.

https://yinglinzheng.netlify.app/diffusion-model-tutorial/ref$_8$

Image Generated by GAN

Figure: An image generated using StyleGAN (Karras et al., 2019) that looks like a portrait of a young woman. This image was generated by a StyleGAN based on an analysis of a large number of photographs.

Variational Autoencoders (VAEs)

Original Paper: Kingma (2013)

- **Concept:** VAEs encode data into a latent space, using a probabilistic framework to reconstruct images.
- **Applications:** Image denoising, anomaly detection, latent space exploration.
- **Significance:** Introduced a way to perform efficient latent space learning with a probabilistic approach.

Architecture of VAE

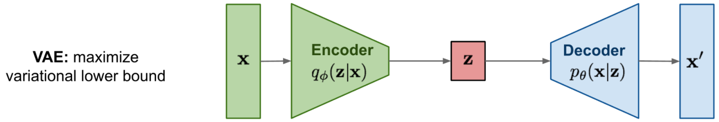
VAE: maximize variational lower bound
$\mathbf{x}$
Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
$\mathbf{z}$
Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\mathbf{x}'$

Figure: Architecture of a Variational Autoencoder (VAE). The encoder $q_\phi(z|x)$ maps input data x to a latent representation z, while the decoder $p_\theta(x|z)$ reconstructs x from z, generating new sample x'. The model is trained to maximize the variational lower bound, which encourages the learned latent space to approximate the true posterior distribution and reconstruct the data effectively.

https://yinglinzheng.netlify.app/diffusion-model-tutorial/ref$_8$

Image Generated by VAE

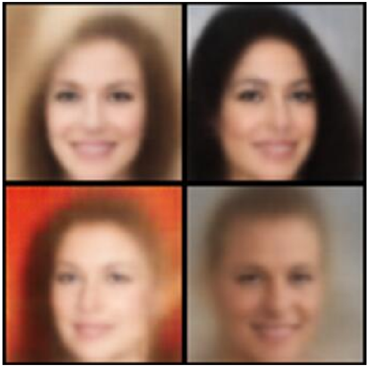

Figure: Images of human face generated by VAE (He, 2023). Autoencoders usually give out blurry images because of their nature of compressing information. While retaining important information like the object of reference fairly good, the loss due to compression is seen by its inability to reconstruct the background and image quality.

Diffusion Models

Original Paper: Sohl-Dickstein et al. (2015)

- **Concept:** Diffusion models progressively add noise to data and learn to reverse the process, generating high-quality images.
- **Applications:** Image synthesis, inpainting, super-resolution.
- **Significance:** Achieved state-of-the-art image quality, competing with GANs in terms of output fidelity.

Architecture of (Basic) Diffusion Model

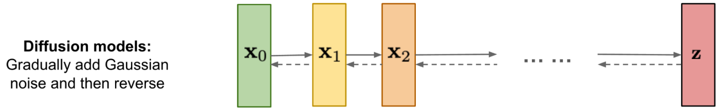

Figure: Architecture of a (basic) diffusion model. It describe the process where Gaussian noise is gradually added to data, transitioning from x_0 to z. The model then learns to reverse this process, reconstructing data by denoising step-by-step, moving from noisy latent states back to the original data distribution.

https://yinglinzheng.netlify.app/diffusion-model-tutorial/ref[8]

Image Generated by (Enhanced) Diffusion Model

Figure: Image of "an Astronaut Riding a Horse" text prompt, generated by Stable Diffusion (Rombach et al., 2022). Diffusion models can be powerful when combined with VAEs and text encoders like CLIP (Radford et al., 2021).

Architecture of DALL-E 2

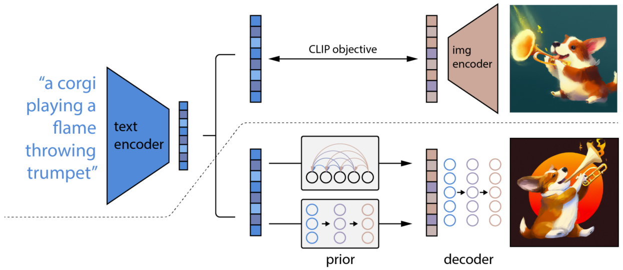

Figure: Image generation process of a DALL-E 2 model (Ramesh et al., 2022). Above the dotted line depicts the CLIP training process, through which we learn a joint representation space for text and images. Below the dotted line, it shows the text-to-image generation process: a CLIP text embedding is first fed to an autoregressive or diffusion prior to produce an image embedding, and then this embedding is used to condition a diffusion decoder which produces a final image. Note that the CLIP model is frozen during training of the prior and decoder.

Problem: Transforming a Certain Distribution to a Normal Distribution

Problem: Given a random variable X, following some distribution, how can we transform it to produce values following a normal distribution $N(0,1)$?

Theoretical Approach:

- Assume that X follows the probability density function (PDF) of $f(\cdot)$.
- Transform X using some function $Y = g(X)$ such that $Y \sim N(0,1)$.
- Then we have:

$$f(x)\,dx = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right)\,dy$$

- Integrating to find the cumulative distribution function (CDF):

$$\int_{-\infty}^{x} f(t)\,dt = \int_{-\infty}^{y} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{t^2}{2}\right)\,dt = \Phi(y)$$

Problem Cont'

- Here, $\Phi(y)$ represents the CDF of the standard normal distribution, and Φ^{-1} is its inverse.
- Thus, the transformation is:

$$y = \Phi^{-1}\left(\int_{-\infty}^{x} f(t)\,dt\right)$$

Note: This transformation is computationally intensive due to the need for numerical integration. It is commonly used for generating normal distributions from uniform distributions.

Mapping Any Distribution to a Target Distribution

General Problem: How do we find a mapping $Y = g(X)$ to transform a given random variable X into another random variable, which follows specified distribution?

Example with GANs:

- GANs aim to map uniform noise to a specific target distribution, represented by a set of "real" samples $X = (x_1, x_2, \ldots, x_N)$.
- The goal is not to learn the transformation from noise to samples directly, but to learn how to generate samples that follow the specified distribution.
- If successful, a random noise input will be transformed into samples resembling the target distribution.

General Problem Cont'

Role of Neural Networks:

- The mapping function g is typically complex and lacks an analytical solution.
- We can use a neural network $G(X, \theta)$ with parameters θ to approximate this mapping.
- Once θ is trained, we can assume $Y = G(X, \theta)$.

Question: How do we evaluate if $Y = G(X, \theta)$ closely matches the target distribution?

Problem Restatement: Distribution Mapping

Problem: Given a dataset $X = (x_1, x_2, \ldots, x_N)$ sampled from a target distribution, find a neural network $Y = G(X, \theta)$ that maps uniform random variables X to the target distribution.

Focus: Compare the closeness between two distributions, not the difference between individual samples.

Approach: Use a distance metric like KL divergence or JS divergence to measure the difference between the distributions.

KL Divergence

KL Divergence: Measures the difference between two probability distributions $p_1(x)$ and $p_2(x)$.

$$KL\left(p_1(x) \parallel p_2(x)\right) = \int p_1(x) \log \frac{p_1(x)}{p_2(x)}\, dx$$

Properties:

- Not symmetric: $KL(p_1 \parallel p_2) \neq KL(p_2 \parallel p_1)$.
- Non-negative: $KL(p_1 \parallel p_2) \geq 0$.
- Equals zero if and only if $p_1 = p_2$.

JS Divergence

Jensen-Shannon (JS) Divergence: A symmetric variant of KL divergence.

$$JS\left(p_1(x), p_2(x)\right) = \frac{1}{2} KL\left(p_1(x) \parallel m(x)\right) + \frac{1}{2} KL\left(p_2(x) \parallel m(x)\right)$$

where $m(x) = \frac{1}{2}(p_1(x) + p_2(x))$.

Significance: More stable and symmetric compared to KL divergence, making it suitable for comparing distributions.

Estimating Probability Densities

Problem: How to estimate the probability densities p_x and p_y?

Approach:

- Divide the real domain into K disjoint intervals $I_1, I_2, \ldots, I_K$.
- Estimate $p_x(I_i)$:

$$p_x(I_i) = \frac{1}{N} \sum_{j=1}^{N} \mathbb{1}(x_j \in I_i)$$

- Generate M uniform samples $x_1, \ldots, x_M$ and obtain $y_1, \ldots, y_M$ from $Y = G(X, \theta)$.
- Estimate $p_y(I_i)$:

$$p_y(I_i) = \frac{1}{M} \sum_{j=1}^{M} \mathbb{1}(y_j \in I_i)$$

Calculating the Distance

Loss Function: Using the JS divergence between p_y and p_x:

$$\text{Loss} = JS\left(p_y(I_i), p_x(I_i)\right)$$

Optimization: Adjust the parameters θ of the neural network $G(X, \theta)$ to minimize the loss, thereby making Y closely match the target distribution.

Note: Since y_j values are generated by $G(X, \theta)$, the estimation of p_y depends on the parameters θ.

Challenges in High-Dimensional Mapping

Problem: When transforming high-dimensional noise into complex distributions, such as generating MNIST digits, the previous approaches may become impractical.

- MNIST images have $28 \times 28 = 784$ dimensions.
- Each pixel is treated as a random variable, leading to $2^{784} \approx 10^{236}$ possible intervals when comparing probability distributions.
- Calculating KL or JS divergence directly between such high-dimensional distributions becomes computationally infeasible.

A Neural Network Approach to Distance Calculation

Solution: Use a neural network to approximate the distance between generated samples $\{y_i\}_{i=1}^M$ and target samples $\{x_i\}_{i=1}^N$:

$$L\left(\{y_i\}_{i=1}^M, \{x_i\}_{i=1}^N, \Theta\right)$$

- The network directly processes y_i and x_i samples to compute the distance.
- This approach allows the model to learn an implicit distance metric without needing to explicitly define KL or JS divergence.
- By training the parameters Θ, we can optimize this loss function.

Ordering and Distribution Comparison

Key Insight: We are comparing distributions, not individual samples.

- The order of samples y_i does not affect the distributional comparison.
- A possible loss function for comparing distributions is:

$$L = \frac{1}{M!} \sum_{\text{all permutations of } y_1,\ldots,y_M} D(y_1, \ldots, y_M, \Theta)$$

- This approach considers all possible orderings of the samples, though it is computationally infeasible.

Simplifying the Loss Function

Practical Approach: Use the simplest approximation:

$$L = \frac{1}{M} \sum_{i=1}^{M} D(y_i, \Theta)$$

- This formulation treats the distance as the average of distances over individual samples.
- It is computationally simpler and avoids the factorial complexity of averaging over all permutations.
- This method approximates the distance between the generated distribution and the target distribution effectively.

Training the Distance Function $D(Y, \Theta)$

Problem: How do we train the distance function $D(Y, \Theta)$?

- $G(X, \theta)$ has not been fully trained yet.
- A new divergence $D(Y, \Theta)$ is introduced, making the training more complex.
- We aim for the final loss L to separate the two distributions as much as possible.

Goal: Minimize L when mapping noise to the target distribution, bringing the distributions closer together.

Adversarial Training Concept

Intuition: An adversarial training approach arises:

- $D(Y, \Theta)$ represents the difference between generated samples Y and target samples X.
- The goal is for L to be as small as possible when comparing X with D, but as large as possible when comparing Y with D, since X are given target samples.

Optimization Objectives:

$$\Theta = \arg\min_{\Theta} L = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} D(x_i, \Theta)$$

$$\Theta = \arg\max_{\Theta} L = \arg\max_{\Theta} \frac{1}{M} \sum_{i=1}^{M} D(y_i, \Theta)$$

Balancing the Objectives

Challenge: Balancing the two objectives is difficult.

- Use a mini-batch size B for simultaneous training.
- For each batch, optimize:

$$\Theta = \arg\max_{\Theta} L_1 = \arg\max_{\Theta} \frac{1}{B} \sum_{i=1}^{B} [D(y_i, \Theta) - D(x_i, \Theta)]$$

- Update θ to minimize:

$$\theta = \arg\min_{\theta} L_2 = \arg\min_{\theta} \frac{1}{B} \sum_{i=1}^{B} D(G(x_i, \theta), \Theta)$$

Summary on GANs (So Far)

Summary:

- $G(X, \theta)$ aims to generate samples that closely match the target samples.
- Training involves alternating between optimizing Θ and θ.
- The adversarial approach helps achieve a balance between learning a strong discriminator D and improving the generator G.

Result: Through this method, $G(X, \theta)$ learns to generate samples that are increasingly close to the real target distribution.

Addressing the Need for Regularization

Observation: Without constraints, training D can lead to unstable behavior and poor optimization of the loss function.

Solution: Introduce constraints on the output of D to stabilize training.

- One approach is to restrict the range of D, for example, applying a Sigmoid activation to limit its output.
- However, such constraints can introduce difficulties during training (values restricted to $[0, 1]$, which makes it difficult for backpropagation in G) and might affect the final result.

The Principle of Finding Constraints

Why Constraints are Needed:

- A distance function D measures how different two objects are, which implies that it should vary smoothly with changes in input.
- We need to ensure that the distance function D is stable, minimizing abrupt changes.

Imposing Lipschitz Regularity

Defining Stability: To ensure stability, D should satisfy the following constraint:

$$\|D(y, \Theta) - D(y', \Theta)\| \leq C\|y - y'\|^{\alpha}$$

where $\alpha > 0$. A simpler form is:

$$\|D(y, \Theta) - D(y', \Theta)\| \leq C\|y - y'\|$$

Interpretation: This is known as a Lipschitz constraint, a common mathematical condition that ensures small changes in input lead to small changes in output.

Practical Implications of Lipschitz Regularity

Why Lipschitz Regularity?

- This constraint ensures that D has a bounded gradient, improving the stability of the training process.
- Although it's a sufficient condition, not a necessary one, it provides a clear and effective way to regularize D.
- A practical condition for D to satisfy Lipschitz regularity is:

$$\left\| \frac{\partial D(y, \Theta)}{\partial y} \right\| \leq C$$

Outcome: Enforcing this condition helps achieve smoother transitions between different states, making D more robust during training.

Adding the Constraint to the Model

Question: How do we integrate the Lipschitz constraint into the model?

Regularization Term: A simple approach is to add a penalty term:

$$\Theta = \arg\min_{\Theta} \frac{1}{B} \sum_{i=1}^{B} [D(x_i, \Theta) - D(y_i, \Theta)] + \lambda \max\left(\left\| \frac{\partial D(y, \Theta)}{\partial y} \right\|, C \right)$$

- λ controls the strength of the penalty.
- This ensures that the gradient of D remains bounded, improving training stability.

Understanding the Soft Constraint

Soft Constraint: The regularization term does not guarantee strict adherence to the Lipschitz condition but penalizes deviations:

- C is a bound constant, not necessarily fixed at 1 (though often chosen to be), but it should remain limited.
- This approach results in a more relaxed Lipschitz constraint, allowing some flexibility.

How to Choose y?

- Noticed that theoretically, it would be best to compute $\left\|\frac{\partial D(y,\Theta)}{\partial y}\right\|$ for all y (the entire space) and then take the average. Clearly, this is impractical. Thus, we resort to a compromise: compute it only for the real samples x_i and the generated samples y_i.
- However, this constraint range seems too narrow, so instead, we interpolate randomly between real samples and generated samples. We hope that in this way the constraint can "cover" the space between real samples x_i and generated samples y_i:

$$y = \varepsilon_i y_i + (1 - \varepsilon_i) x_i,$$

where ε_i is sampled from $U[0, 1]$.

Gradient Penalty Approach: WGAN-GP

WGAN-GP: An approach by Arjovsky et al. (2017), known as Wasserstein GAN - Gradient Penalty.

Formulation: A slightly modified penalty function:

$$\Theta = \arg\min_{\Theta} \frac{1}{B} \sum_{i=1}^{B} [D(x_i, \Theta) - D(y_i, \Theta)] \\ + \lambda \sum_{i=1}^{B} \max \left(\left\| \frac{\partial D(y, \Theta)}{\partial y} \right\|_{y=\varepsilon_i y_i + (1-\varepsilon_i) x_i} - C \right)^2$$

- This approach enforces smooth transitions between real and generated samples.

Final Remarks on GANs

- Many researchers focus on developing different penalty terms to modify GANs, to improve the quality of generated images.
- The choice of regularization should match the specific problem and dataset characteristics.
- In GANs, we actually do not have strong control over two adversarial neural networks, i.e., generator $G(X, \theta)$ and discriminator $D(Y, \Theta)$. It's more like black-box problems (implicit generative models), which means that they do not explicitly model the likelihood function nor provide a means for finding the latent variable corresponding to a given sample, unlike alternatives such as VAEs.

Questions?

Thank you!

References I

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

L. He. Comparison of improved variational autoencoder models for human face generation. In *Journal of Physics: Conference Series*, volume 2634, page 012042. IOP Publishing, 2023.

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

References II

T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

D. P. Kingma. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

A. Mordvintsev, C. Olah, and M. Tyka. Inceptionism: Going deeper into neural networks. *Google research blog*, 20(14):5, 2015.

D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

References III

A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 1(2):3, 2022.

R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020a.

References IV

Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020b.

L. Zhang, A. Rao, and M. Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023.