Unsupervised Learning: Image Generation - Casual Read

Earl Wong

Supervised vs Unsupervised

- In supervised learning, learning occurs using labeled training data.

- i.e. One learns a mapping for input x, to output y, via y = f(x).

- f( ) is a black box / model implemented using a decision tree, multilayer perceptron, convolutional neural network, transformer network, etc.

- Successful examples of supervised learning include: image classification, bounding boxes for object detection, etc.

- In all cases, labeled training data is required: $Data = (\vec{x}, \vec{y})$

Application

- Using the learned model, one can take an input image x, observe the output y, and determine if there is a car in the image.

- One could also determine the location of the car in the image, etc.

- Simply put, many useful applications result from an accurate input / output black box model.

- Example: Self driving cars are possible, because the black box models can detect pedestrians, track pedestrians, detect other cars, track other cars, etc.

Supervised vs Unsupervised

- In unsupervised learning, the goal is to learn the distribution of data.

- No labels are involved.

- For example, consider measuring the height of every person in San Francisco, and generating a histogram.

- The data is the height, while the learned distribution is represented by the histogram.

- In this example, the learned distribution p(x) would (most likely) be gaussian / normal.

Background

- In statistics, if the distribution is known, we can 1) sample from the distribution, 2) extract useful information from the distribution, etc.

- For example: If we sample from the distribution, it will be fair to say, that we will never obtain a sample where the height of a person is greater than 10 feet.

- For example: The median of the distribution tells us the midpoint of the heights.

- i.e. Half of the population in San Francisco will be taller than the median, while the other half of the population will be shorter than the median.

Application

- The learned distribution $p(x)$, can also represent natural images.
- Now, we can sample from the distribution of natural images.
- This is the basis for image generation, video generation, etc.
- In the remainder of this presentation, we will focus on image generation.

Image Generation

- A common thread exists between all image generation methods.

- **The training data determines the images the you can generate.**

- If the training data only contains images of people, you will not be able generate images of cars.

- **The model needs to have the capacity to support the distribution that you are trying to learn.**

- Why is this important?

Image Generation

- Imagine a manifold / surface, containing the distribution of all possible natural images.

- A very, very small subset of this manifold contains images of people.

- In order to generate image of people on bicycles, of people driving cars, of people sitting on park benches, etc., you ALSO need to learn how to generate bicycles, cars, park benches, etc.

- Image generation begins with “toy problems”: for example, generate MNIST digits.

- However, the holy grail is to generate any and all natural images.

Image Generation

In order to go from MNIST to natural images, you need:

- A lot more training data
- Training data with significantly richer information content
- A model that can capture / learn the information contained in the richer training data

Image Generation

- This is where language models become invaluable.
- By construction, language is used to describe natural images.
  - Kindergarten: dog, cat, pig, horse
  - First grade: A dog sitting on a pillow
  - Advanced 4th grader? - A dog sitting on a pillow in front of a fireplace, in the living room of house with green walls, with a picture of President Kennedy above the fireplace ….”

Image Generation

- The holy grail: lots of training data, a model with enough capacity to capture the information contained in the training data, a language model to describe the image.

- Goal: Learn a distribution $p_{\theta*}(x)$ that allows you generate the complete image manifold = every conceivable image that you can think of.

- However, let's begin with $p_{\theta}(x)$, where $p_{\theta}(x) \subset p_{\theta*}(x)$, representing a small subset of the manifold - say images of MNIST digits or images of human faces.

Overview

- Many competing techniques exist for image generation based on a explicit (or implicit) learned distribution $p_\theta(x)$

- Goodness Criteria: High quality output image, with minimal compute / low generation latency, etc.

- We will now discuss the fundamental concepts, with some details, for several different techniques.

Methods

- Auto regressive (AR) based image generation
- Flow based image generation
- Variational Auto Encoder (VAE) based image generation
- GAN based image generation
- Diffusion based image generation

Auto Regressive

High Level Concept

- Determine the next output, given the previous input.
- Example: The sun rises in the _____
- For image generation, we want to predict the next pixel, given knowledge of the previous pixels.
- Previous pixels means all pixels above and to the left of the current pixel.

Auto Regressive

The Math

- Let the family of nxn face images be represented by the following (unrolled) distribution:

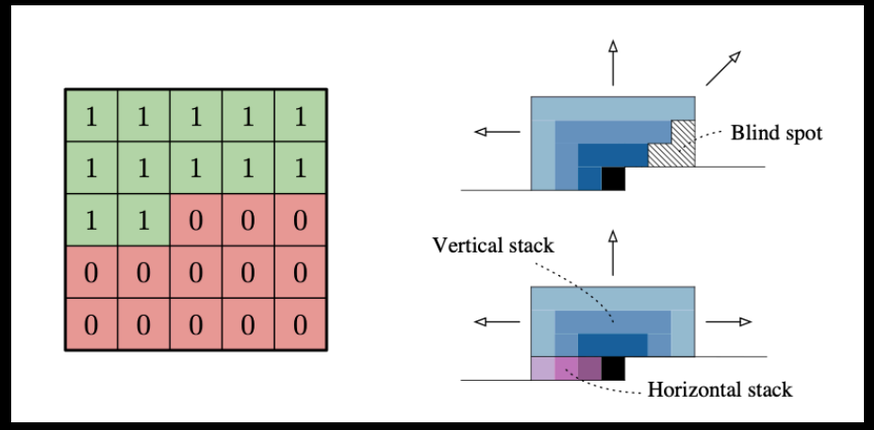
$$p(\overrightarrow{x}) = p(x_1, x_2, \ldots, x_{n*n}) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \ldots = \prod_{i=1}^{n*n} p(x_i \mid x_1, x_2, \ldots, x_{i-1})$$

- By construction, we observe the auto regressive nature, baked into each term of the equation.

- i.e. Each subsequent pixel depends on all previous pixels.

Auto Regressive

- How is this accomplished in practice?
- Originally, recurrent neural networks were used.
- However, convolutional neural networks (CNN) accomplished the same functionality, but in significantly faster runtime.



A convolution kernel with the auto regressive property, is shown to the left.

Conditional Image Generation with PixelCNN Decoders, van den Oord, et. al.

- However, stacked convolutional filters (done to expand CNN the receptive field), resulted in blind spots.
- To address this problem, the ar convolutional filters were decomposed into horizontal and vertical components (shown as vertical and horizontal stack in the picture).

Auto Regressive

Training

- The entire image is consumed by the CNN.
- The depth of the CNN is dictated by the needed receptive field coverage / size of the input image.
- At each pixel location, 256 logits are computed.
- The logits are converted to probabilities via softmax.
- At each pixel location, the probability (associated with the 1 hot vector for the training image intensity) is back propagated into the network.

Auto Regressive

Generation

- The image is generated pixel by pixel.
- An nxn zero matrix is input into the network, sans position x_1
- x_2 is output.
- An new nxn zero matrix is input into the network, sans positions x_1 and x_2
- etc.

Flow

High level Concept

- Flow methods result from a statistical theorem used to transform a random variable X into a random variable Z.
- i.e. $Z = f(X)$
- By definition, f needs to be invertible and differentiable.

Flow

The Math

- Let X be a continuous random variable with density function $p_x(x)$
- Let f be a continuous and differentiable function where $Z = f(X)$
- Now $p_Z(z) = p_X(x) \left| \det J_f(x) \right|^{-1}$
- J represents the Jacobian of the function f

Flow

The Math (for images)

- X and Z become: $\vec{x} = (x_1, x_2, \ldots x_n)\ to\ \vec{z} = (z_1, z_2, \ldots z_n)$ and $p_x(\vec{x}) \rightarrow p_z(\vec{z})$

- Where $(z_1, z_2, \ldots z_n) = f(x_1, x_2, \ldots x_n)$

- For an nxn input image, the dimension of the Jacobian matrix becomes nxn.

Flow

- How is this accomplished in practice?
- The main computational constraint, resides in computing the determinant of the Jacobian.
- If the Jacobian is upper or lower triangular, the calculation simplifies to computing the product of the diagonal terms in J.
- f is realized via: $f = f_1 \circ f_2 \circ f_3 \ldots \circ f_m \quad or \quad f = f_1(f_2(f_3(\ldots f_m)))(\vec{x})$
- Each f_i is invertible, and the Jacobian matrix for each f_i is upper or lower triangular.

Flow

$$
\begin{aligned}
y_{1:d} &= x_{1:d} \\
y_{d+1:D} &= x_{d+1:D} \odot \exp\big(s(x_{1:d})\big) + t(x_{1:d}),
\end{aligned}
$$

A coupling layer.

Density Estimation Using Real NVP, Dinh, et. al.

- To create a diagonal J for each f_i , a coupling layer is employed.

- The input image is split into two halves - half of the image is passed through without any processing.

- The other half undergoes a learned scaling and translation = affine transform.

- The process then repeats, ping ponging between the two halves, for subsequent coupling layers.

Flow

Training

- An input image is consumed by the network.
- The target output / desired output is a multivariate gaussian distribution with independent variables: $p_z(\vec{z}) = N(0, I)$
- A loss is computed, with this target in mind.
- The loss is back propagated into the network, to learn the scalings and translations for each coupling layer.
- A detailed “cut and paste” workflow is given on the next slide.

Flow

- Training

3. Training: maximum likelihood

We **don't have target z values**. Instead, we **train the flow so that the transformed z follows the Gaussian**.

Use the **change-of-variables formula**:

$$\log p_X(x) = \log p_Z(z) + \log \left| \det J_{f_\theta}(x) \right|$$

Where:

- $p_Z(z)$ is the Gaussian probability of z:

$$\log p_Z(z) = -\frac{1}{2} \sum_i z_i^2 - \frac{n^2}{2} \log(2\pi)$$

- $J_{f_\theta}(x)$ is the Jacobian of the flow transformation. For flows like **coupling layers**, $\det J$ is easy to compute.

The **loss** is:

$$\mathcal{L}(\theta) = -\log p_X(x) = -\left(\log p_Z(z) + \log |\det J_{f_\theta}(x)| \right)$$

- Minimize this loss with gradient descent.
- Intuition: the flow learns to **warp the data space** x into a **simple Gaussian latent space** z.

Flow

Generation

- Generation is simple, since (by construction) a 1-1 bijective transform exists between the latent space $\vec{z} = (z_1, z_2, \ldots z_n)$ and the image space $\vec{x} = (x_1, x_2, \ldots x_n)$

- i.e. We sample from a multivariate gaussian distribution in the latent space, and run the sampled input through $f^{-1}(\ )$

GANS

High Level Concept

- AN = adversarial network
- Two networks compete against each other.
  - A generator generates an image.
  - A discriminator tries to discern whether it is real or fake.
  - Over time, the generator gets better at producing more realistic images; the discriminator gets better at discerning between real and fake images.

GANS

The Math

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Generative Adversarial Networks, Goodfellow, et. al.

- If the discriminator gets too strong early on, the resulting signal to the generator might be small.

- To address this problem:

$$\max_D \mathbb{E}_{x\sim q_{\textbf{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]$$

$$\min_G -\mathbb{E}_{z\sim p(z)}[\log D(G(z))]$$

GANS

Training

- Sample real images
- Sample random noise, and generate fake images.
- Maximize $\max_D \mathbb{E}_{x\sim q_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]$ to train the discriminator.
- Minimize $\min_G - \mathbb{E}_{z\sim p(z)}\left[\log D(G(z))\right]$ to train the generator.
- Repeat.

GANS

Generation

- Generation is straightforward.
- Once the network is trained, we sample from the latent space: $\hat{x} = G(z), \quad z \sim N(0, I)$

Diffusion

High Level Concept

- Two processes are run: a forward process and a reverse process.

- The forward process adds an increasing (and known amount) of gaussian noise to a noise free input image.

- The process stops, when the noise free image becomes a random noise image.

- The reverse process incrementally “de-noises” the random noise image back into a clean, noise free image.

Diffusion

The Details

Forward diffusion

- A specific amount of (increasing) gaussian noise is added to noise free input images for a specific number of time steps. [0, 1, 2, … N] according to some fixed noise schedule.
- At time step N, the original images are no longer discernible, and resemble random noise images.
- The images generated at all time steps, comprise the training data.

Reverse diffusion

- A random noise image is input. [Timestep N image]
- The network estimates an incremental “noise image”.
- An updated image is generated, by subtracting the incremental “noise image” and (then) adding a small amount of gaussian noise (for image generation diversity) [Timestep N-1 image].
- The process repeats until time step 0, resulting in a noise free image.

Diffusion

The Math

Forward Diffusion

$$q(x_t \mid x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t}\, x_0, (1 - \bar{\alpha}_t)I\right)$$

$$\alpha_t := 1 - \beta_t, \quad \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$$

$$\beta_1 = 10^{-4} \quad \textbf{to} \quad \beta_T = 0.02$$

$$t \in [0,1,2,....1000]$$

Reverse Diffusion

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \text{ if } t > 1, \text{ else } \mathbf{z} = \mathbf{0}$$

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$$

Denoising Diffusion Probabilistic Models, Ho, et. al.

Diffusion

- How is this accomplished in practice?
- The forward diffusion process is straightforward.
- However, the reverse diffusion process should be intriguing:

$$\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right)$$

- Specifically, $\epsilon_\theta(x_t, t)$ is a noise image estimate, output by the trained network, for some noisy input image.
- Over the course of the reverse diffusion process, these noise image estimates are used to “de-noise” the random noise image.

Diffusion

- However, denoising is not occurring in the classical, traditional sense.
- Instead, $\epsilon_\theta(x_t, t)$ provides a well defined "movement direction" for each pixel in the image.
- The final destination, is a noise free image from the original training data.
- Because $\sigma_t z$ is added to the reverse process, image diversity is created.

Diffusion

Training

- A noisy input image is consumed by the network.
- By construction, the applied noise is known = gaussian with known variance.
- The network is trained to produce an output image with a similar noise level.
- MSE is used, to back propagate the observed quantity, with the desired quantity.
- Each pixel location is IID $N(0, k * I)$ for $k \in [0,1]$

Diffusion

Generation

- The earned, reverse diffusion is applied to a random noise input image.

- The “denoising” occurs incrementally, over many steps (T=1000).

- However, $\epsilon_\theta(x_t, t)$ is not “subtracting noise”, but instead, performing learned, incremental trajectory guiding.

- If no diversity term was added at each iteration, the process would re-generate a noise free image, present in the original training set.

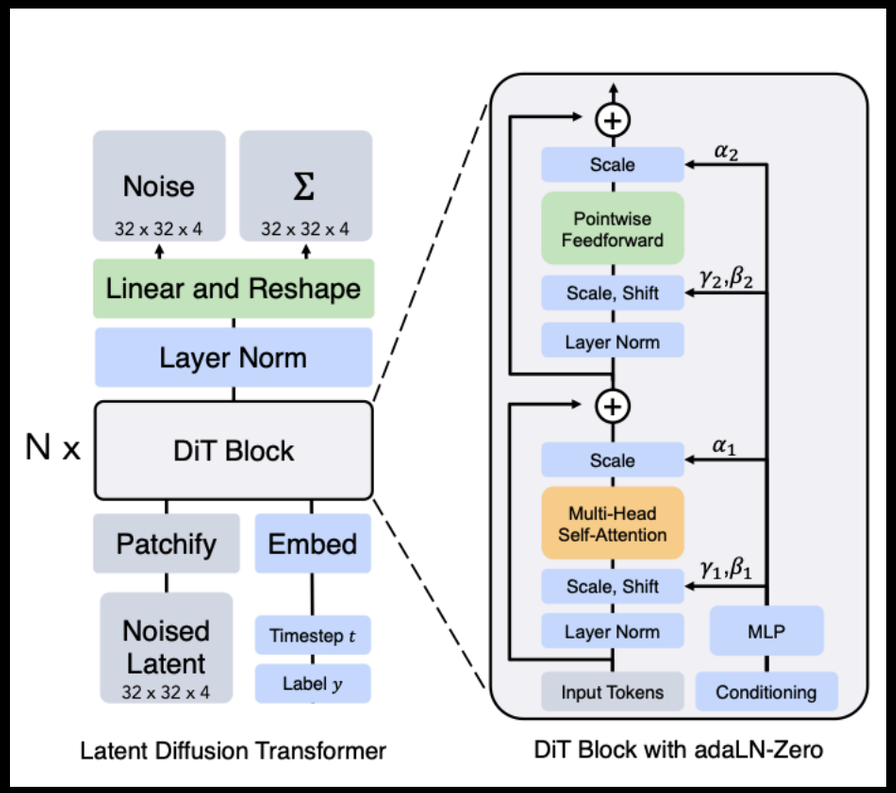
Diffusion

Generation

- Inference can be slow, for T = 1000.
- As a result, T is subsampled, and the effect at each time step is scaled accordingly.
- Because fewer time steps are used, diversity will suffer.
- Because fewer time steps are used, modeling errors and non-linearities will become more noticeable and problematic.

Big Picture Summary

- In 2023, diffusion based image generation outperformed all of the other image generation approaches. (Will Peebles, PhD Dissertation, Ch. 2)
- Instead of using a CNN UNet as a model, a transformer network is used.



Class-Conditional ImageNet 512×512					
Model	FID↓	sFID↓	IS↑	Precision↑	Recall↑
BigGAN-deep [56]	8.43	8.13	177.90	0.88	0.29
StyleGAN-XL [69]	2.41	4.06	267.75	0.77	0.52
ADM [4]	23.24	10.19	58.06	0.73	0.60
ADM-U	9.96	5.62	121.78	0.75	**0.64**
ADM-G	7.72	6.57	172.71	**0.87**	0.42
ADM-G, ADM-U	3.85	5.86	221.72	0.84	0.53
DiT-XL/2	12.03	7.12	105.25	0.75	**0.64**
DiT-XL/2-G (cfg=1.25)	4.64	5.77	174.77	0.81	0.57
DiT-XL/2-G (cfg=1.50)	**3.04**	**5.02**	**240.82**	0.84	0.54

Table 2.3: **Benchmarking class-conditional image generation on ImageNet 512×512.** Note that prior work [4] measures Precision and Recall using 1000 real samples for 512 × 512 resolution; for consistency, we do the same.

Big Picture Summary

- Next, let’s look at the big picture limitations, for each of the approaches.

Big Picture Summary

- Auto Regressive - Although a nice mathematical model exists, one must ask: “Is causality the best property for image generation? For image in painting?”

- Also, from an engineering standpoint, mathematical models are often “approximated”, to obtain the needed tradeoffs for productization.

- In the process, the desired output can suffer.

- We will revisit the above statement for Flow.

Big Picture Summary

- Flow - The method is rooted in rigorous mathematics.
- The achilles heel, is the need to compute the determinant of the Jacobian.
- To make the process feasible for large images, constraints are (then) placed on f, reducing the efficacy of the approach.

Big Picture Summary

- GANS - The method is rooted in mathematical optimization. i.e. the minimax problem.

- By construction, the image generation distribution is learned implicitly from the training data.

- i.e. The generator tries to generate images that will fool the discriminator.

- i.e. The discriminator is shown fake, generated images and real images from a training test set.

- GANS are notoriously difficult to train, since the generator and discriminator training needs to be “balanced”, so that neither network gets too far ahead of the other.

Big Picture Summary

- Diffusion - A nice forward and reverse diffusion process exists, that is succinctly described by the mathematics.

- However, the network is not estimating noise (in the traditional sense) for the reverse process, but instead, pixel trajectory**.

- As a result, without an additional noise contribution term for image diversity, the reverse process would regenerate the original noise free image.

Big Picture Summary

- ** If you take a sequence of Gaussian IID random variables and add or subtract them, the noise variance will not go to zero.

- **Hence, the reverse process in denoising diffusion is not denoising the image in the classical sense.

- Classical sense = denoising by temporal or spatial averaging.

Big Picture Summary

- This brings us full circle, back to the early statements made at the beginning of this presentation:

- **The training data determines the images the you can generate.**

- **The model needs to have the capacity to support the distribution that you are trying to learn.**

- By construction, transformers have an advantage over CNN's (UNets) because of the attention vs receptive field tradeoff.

Big Picture Conclusion

- **Flow based image generation may be the best approach (sans runtime issues).**
- Unlike diffusion, there is no need for incremental steps.
- Unlike GANS, it does not have a training stability issue.
- Unlike auto regressive, it is 1-1 per image vs pixel based.
- => Image generation from the latent space is the correct framework / diversity is free = built in.

Big Picture Conclusion

- Unfortunately, the author has (yet) to encounter any published results showing the lower bound performance for Flow, when a full Jacobian / non affine transform network is used / trained.