Lecture 2: Basic Concepts for Generative Models

University of Michigan, FA 2026

Instructor: Prof. Jeong Joon Park

Today

- Generative Models
  - Autoregressive Models
  - VAE
  - GANs
  - Diffusion Models
  - Flow-based Models

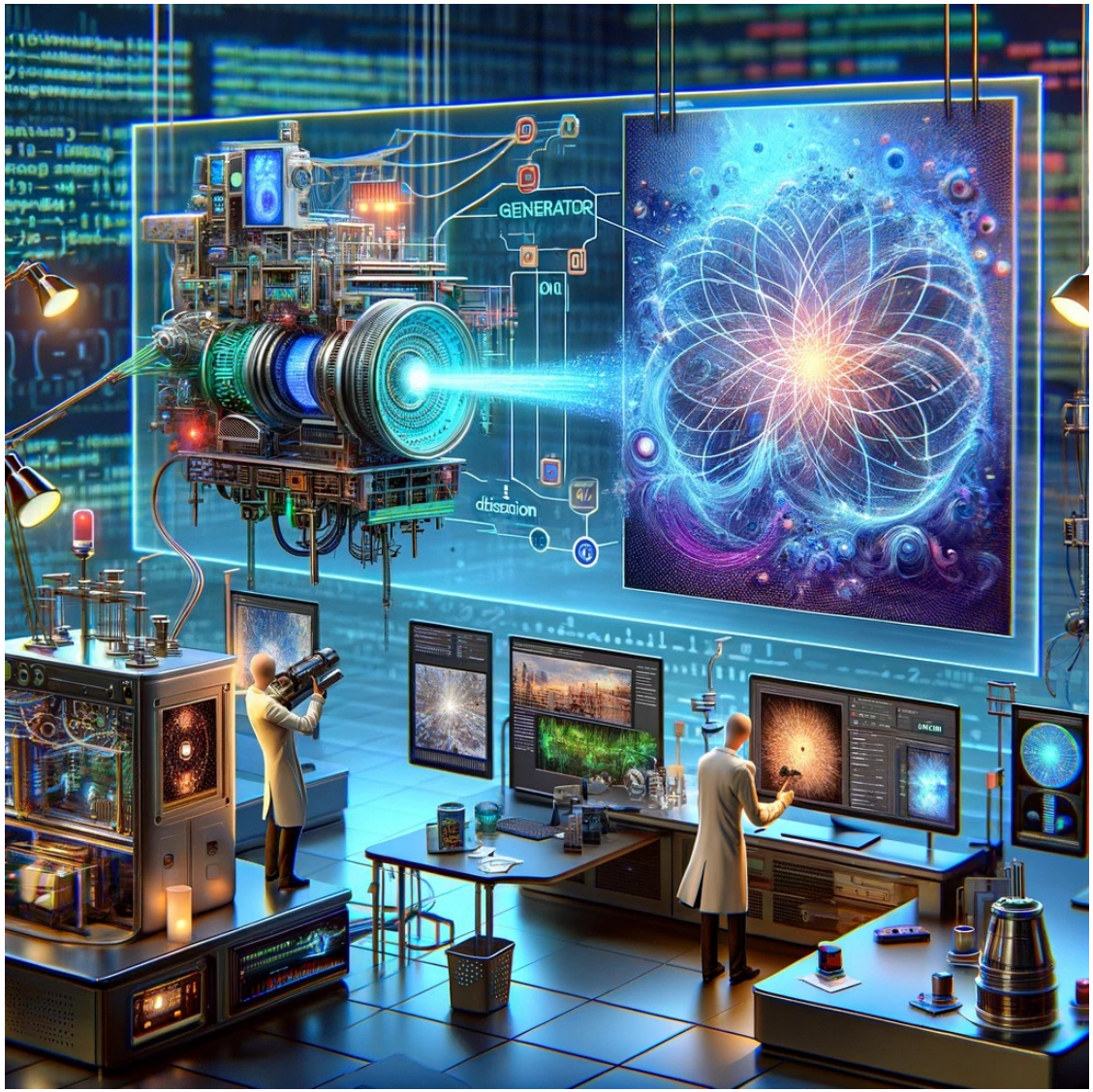
Administration

- Register for a paper to present
- 1~2 students per paper (third column will be removed)
- Needs to be registered by this Friday
- Also, sign up for Piazza immediately!

Visual Generative Models

- GANs
- Diffusion Models
- 3D Gen Models
- Automatic Content Generation!

→ Image By ChatGPT



Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Data: x

Label: y

Cat

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Data: x

Label: y

Cat

Probability Recap:

Density Function
p(x) assigns a positive number to each possible x; higher numbers mean x is more likely

Density functions are **normalized**:

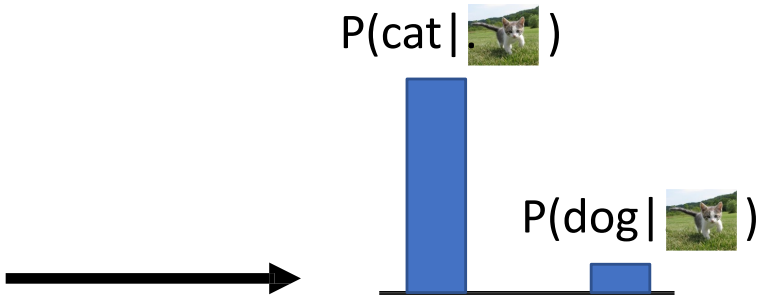
$$\int_X p(x)dx = 1$$

Different values of x **compete** for density

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Data: x



Generative Model: Learn a probability distribution p(x)

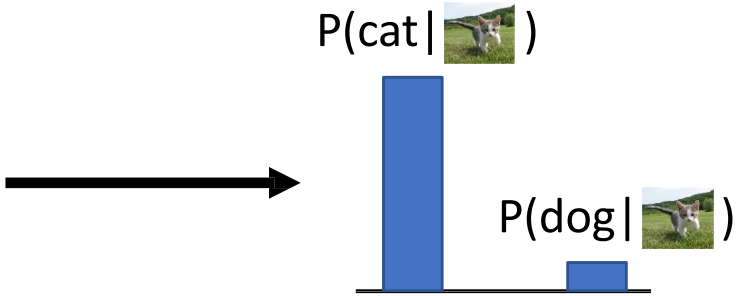
Conditional Generative Model: Learn p(x|y)

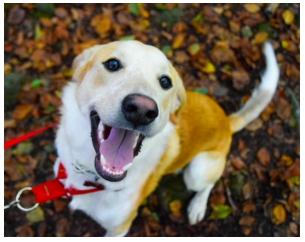
Discriminative vs Generative Models

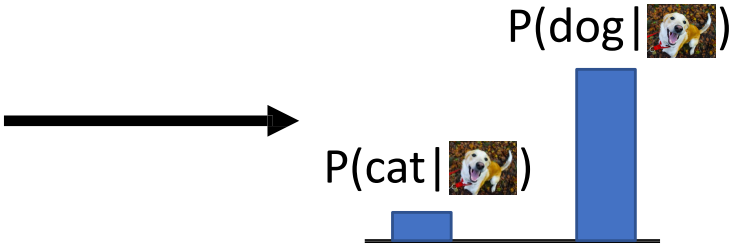
Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)







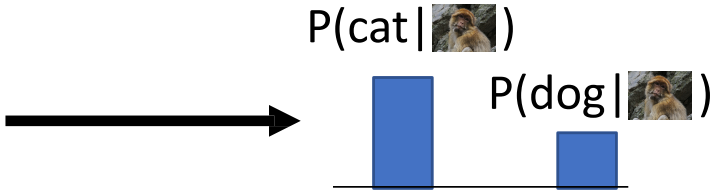
Discriminative model: the possible labels for each input "compete" for probability mass. But no competition between **images**

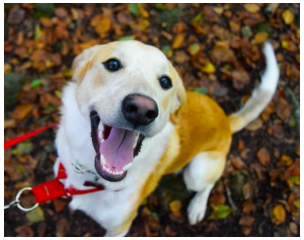
Discriminative vs Generative Models

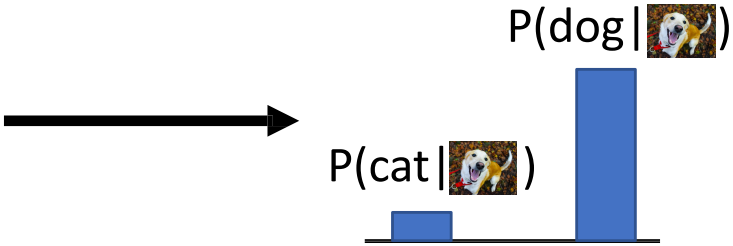
Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



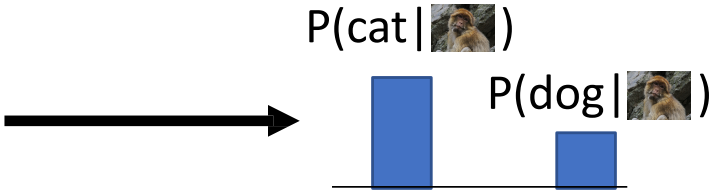




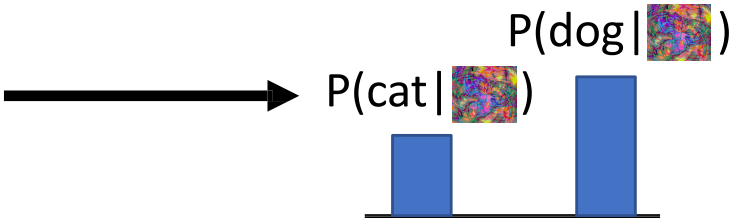
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)



Generative Model: Learn a probability distribution p(x)



Conditional Generative Model: Learn p(x|y)

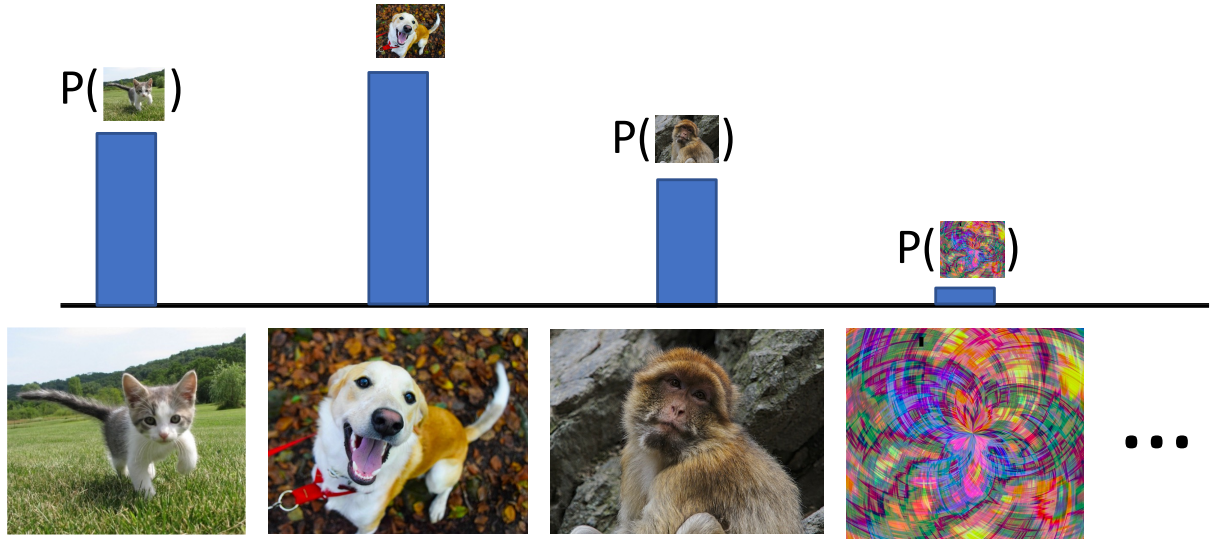
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Generative model: All possible images compete with each other for probability mass

Requires deep image understanding! Is a dog more likely to sit or stand? How about 3-legged dog vs 3-armed monkey?

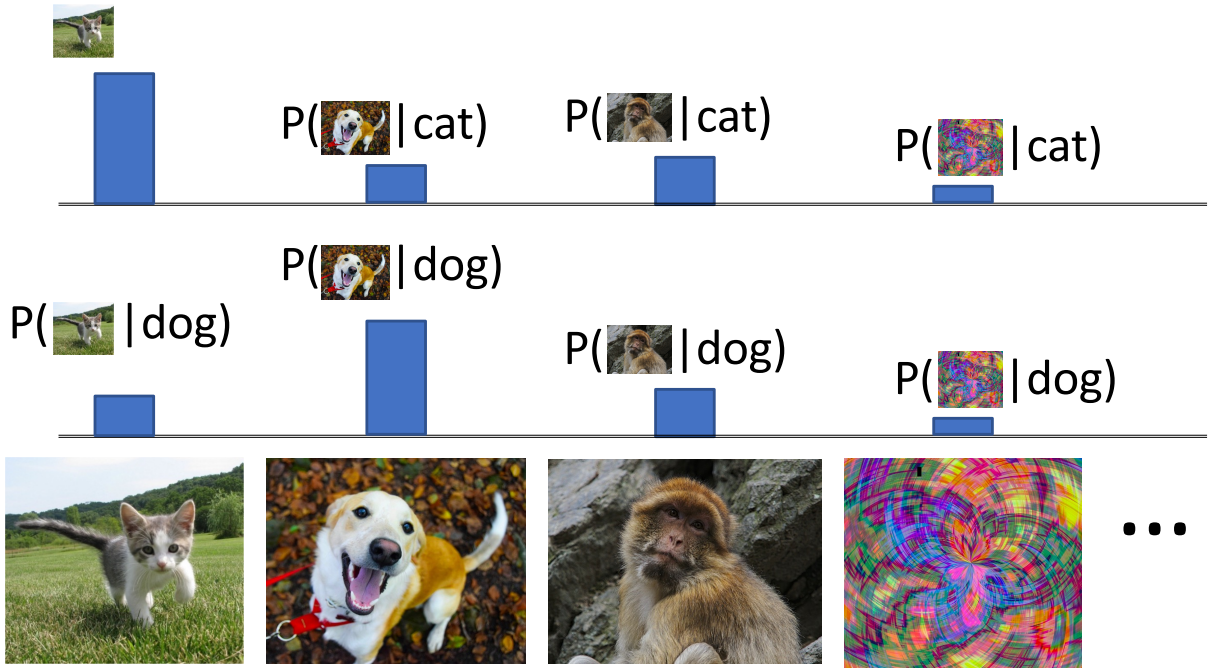
Model can “reject” unreasonable inputs by assigning them small values

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Conditional Generative Model: Each possible label induces a competition among all images

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Recall **Bayes' Rule:**

$$P(x \mid y) = \frac{P(y \mid x)}{P(y)} P(x)$$

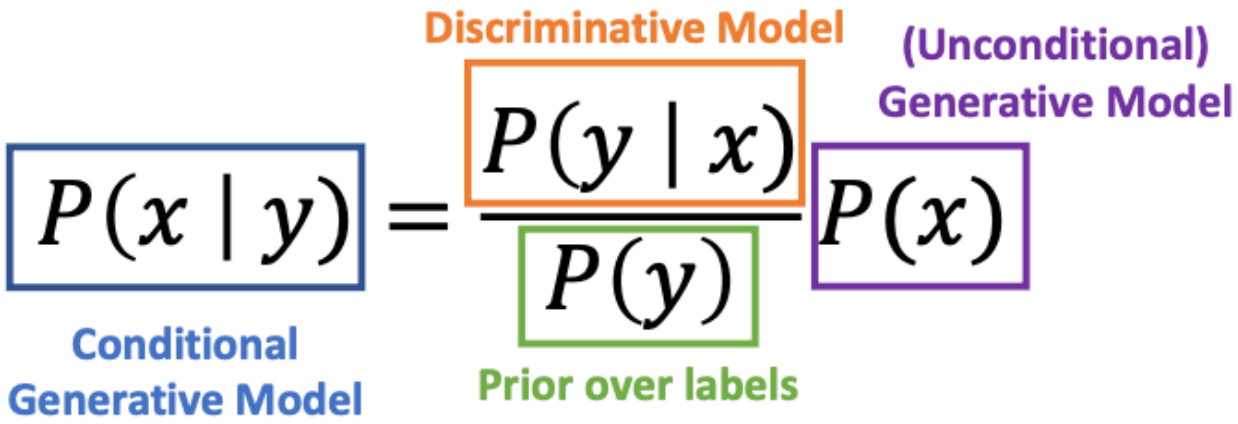
Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Recall **Bayes' Rule:**



We can build a conditional generative model from other components, etc.

Progress in Generative Models of Images

Slide credit: Ian Goodfellow, 2019

https://github.com/NVlabs/stylegan3 [Karras et al., “Alias-Free Generative Adversarial Networks”, 2021]

Images and Text

TEXT PROMPT an armchair in the shape of an avocado. . . .

AI-GENERATED IMAGES

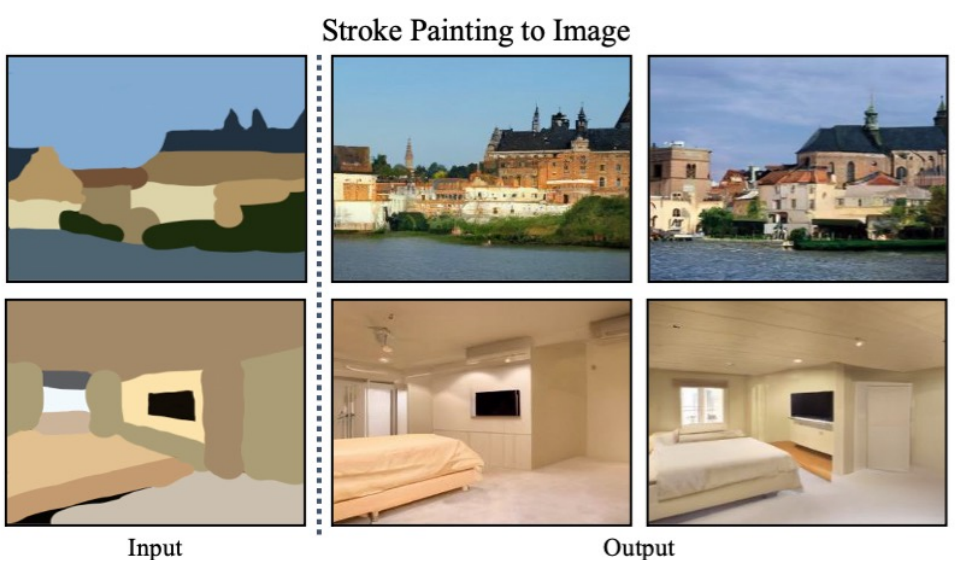
Edit prompt or view more images↓

P(image | caption)

TEXT PROMPT a store front that has the word ‘openai’ written on it. . . .

AI-GENERATED IMAGES

Inverse Problems with GenAI

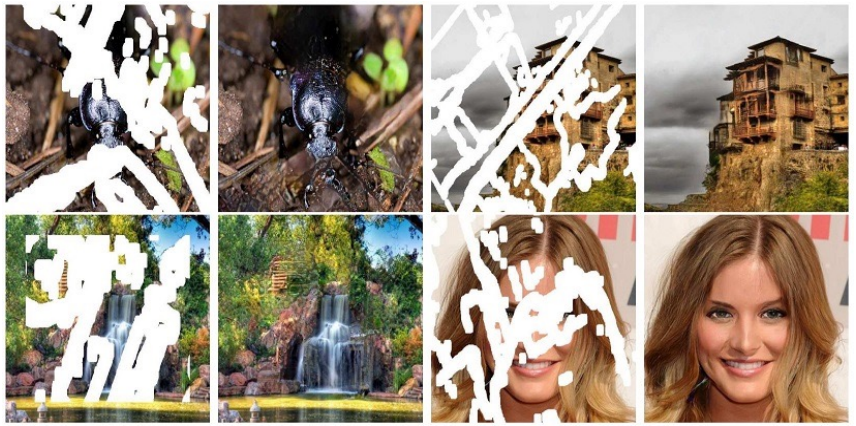


Inverse Problems with GenAI

P(high resolution | low resolution)

Menon et al, 2020

P(full image| mask)

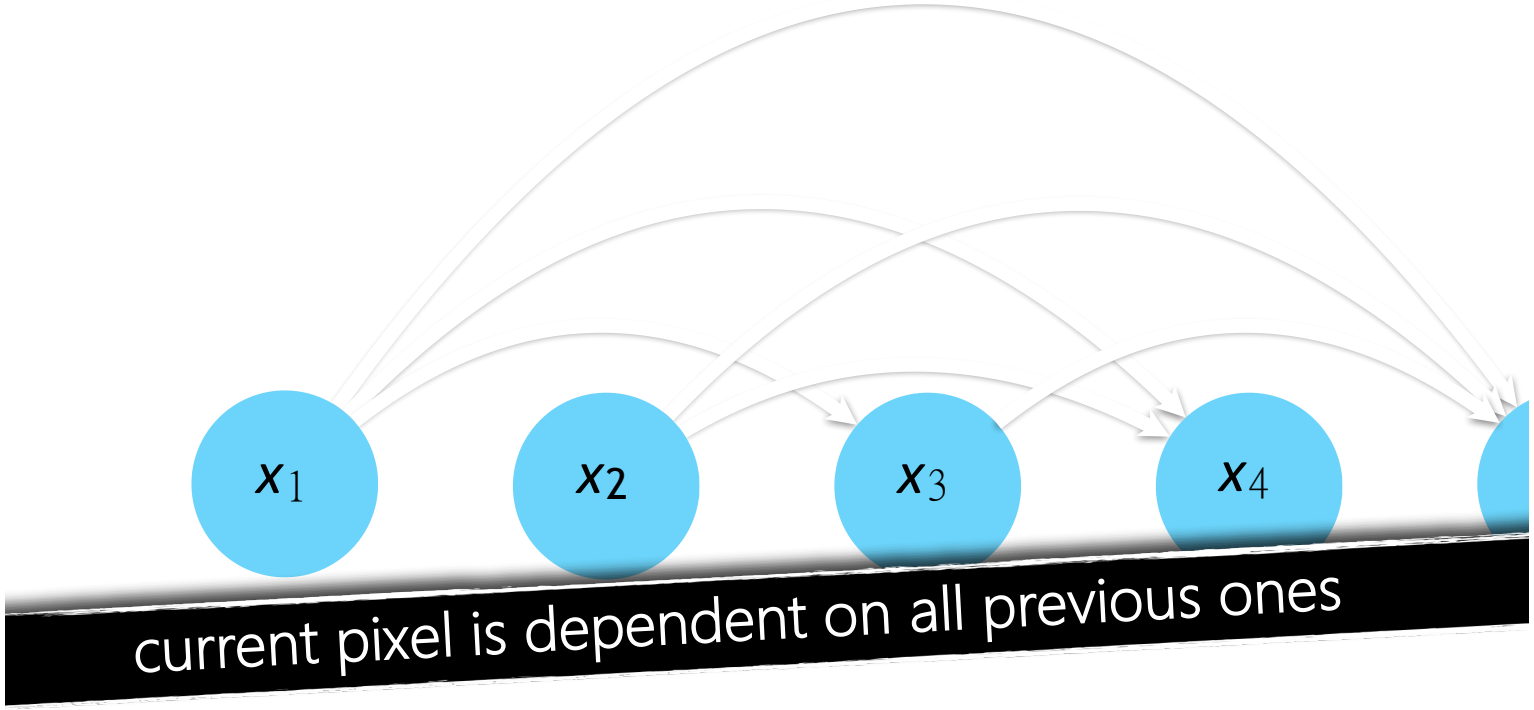


Liu al, 2018

P(color image| greyscale)

Antic, 2020

Autoregressive models: PixelRNN & PixelCNN


x_1
x_2
x_3
x_4
current pixel is dependent on all previous ones

p

use chain rule to decompose

$$p_{\text{model}}(\mathbf{x}) = \prod_{i=1}^{N} p(x_i | x_1, \ldots, x_{i-1})$$

$$p_{\text{model}}(\mathbf{x}) = \prod_{i=1}^{N} p(x_i | x_1, \ldots, x_{i-1})$$

probability of ith pixel given all previous pixels

$$p_{\text{model}}(\mathbf{x}) = \prod^{N} p(x_i | x_1, \ldots, x_{i-1})$$

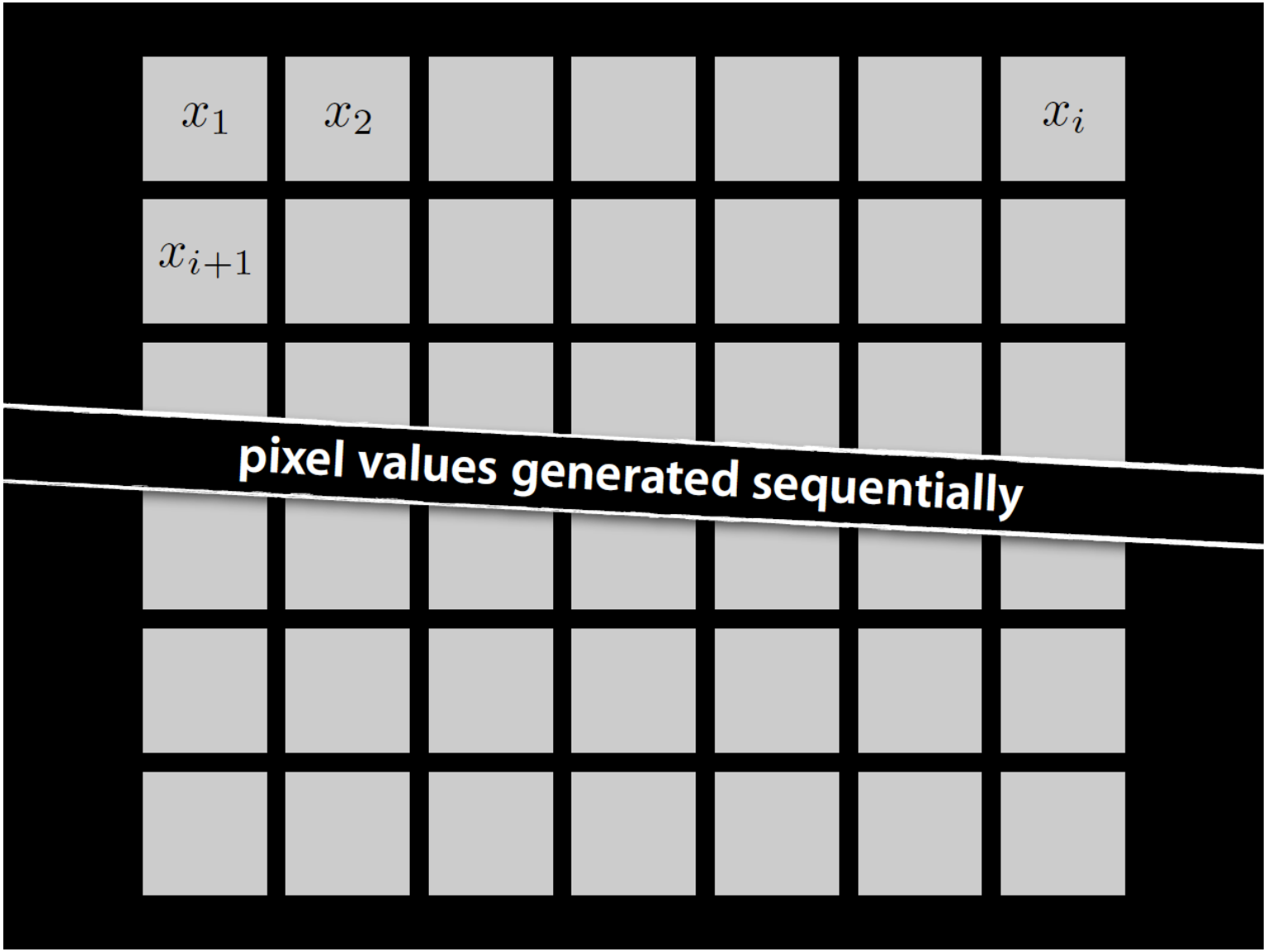
express distribution over pixel values as neural network

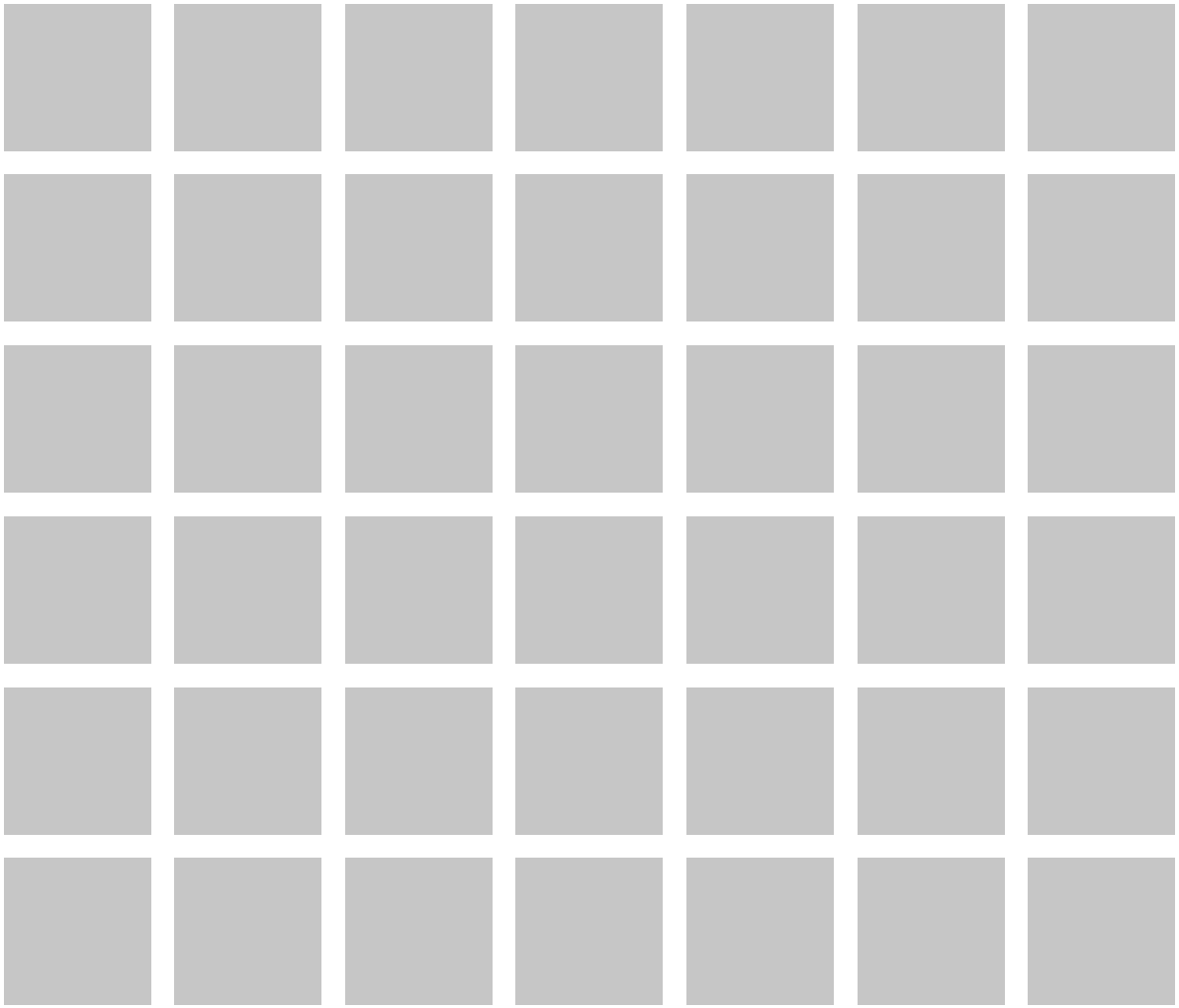
$$p_{\text{model}}(\mathbf{x}) = \prod_{i=1}^{N} p(x_i | x_1, \ldots, x_{i-1})$$

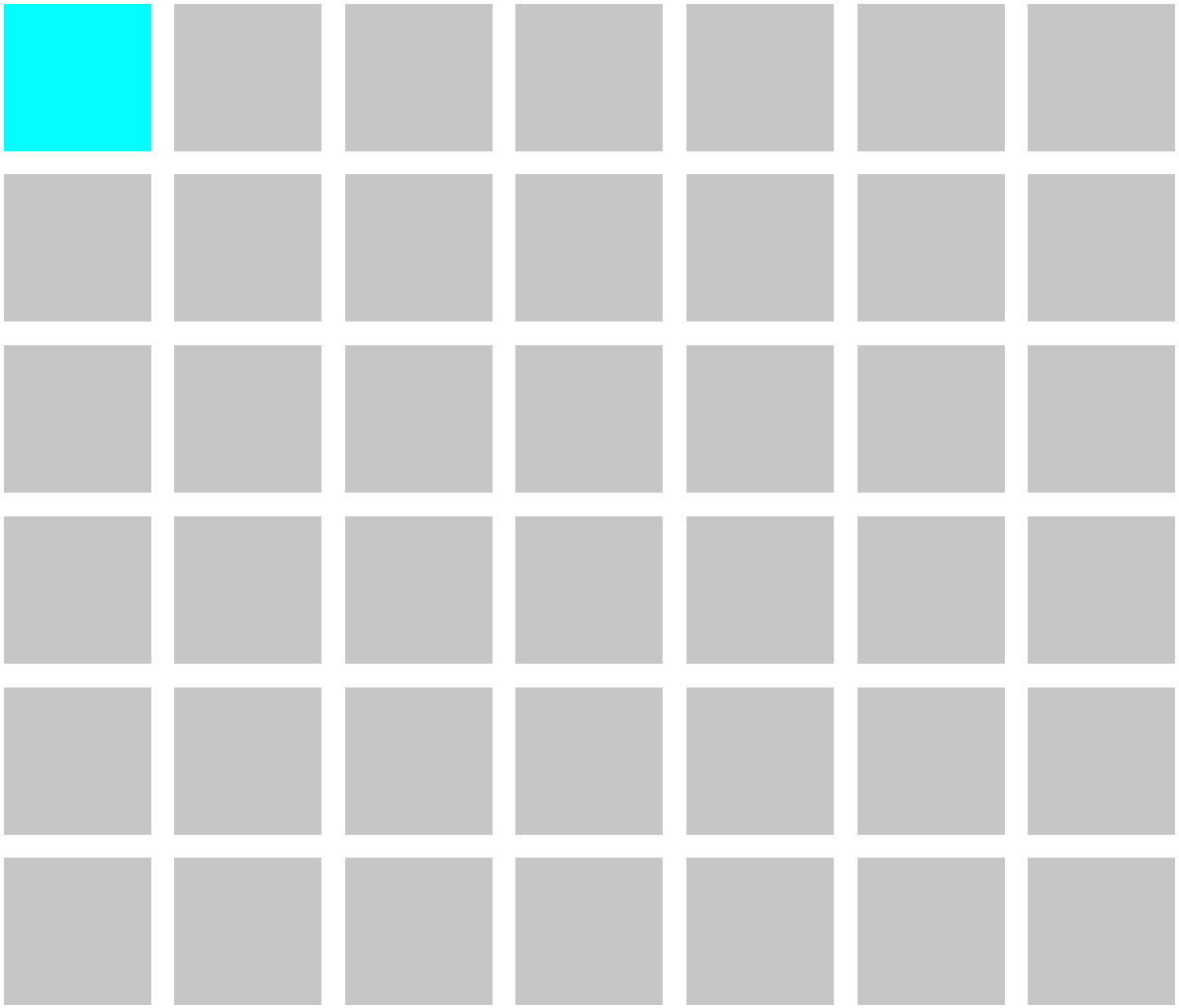
maximize likelihood of training data with respect to network weights

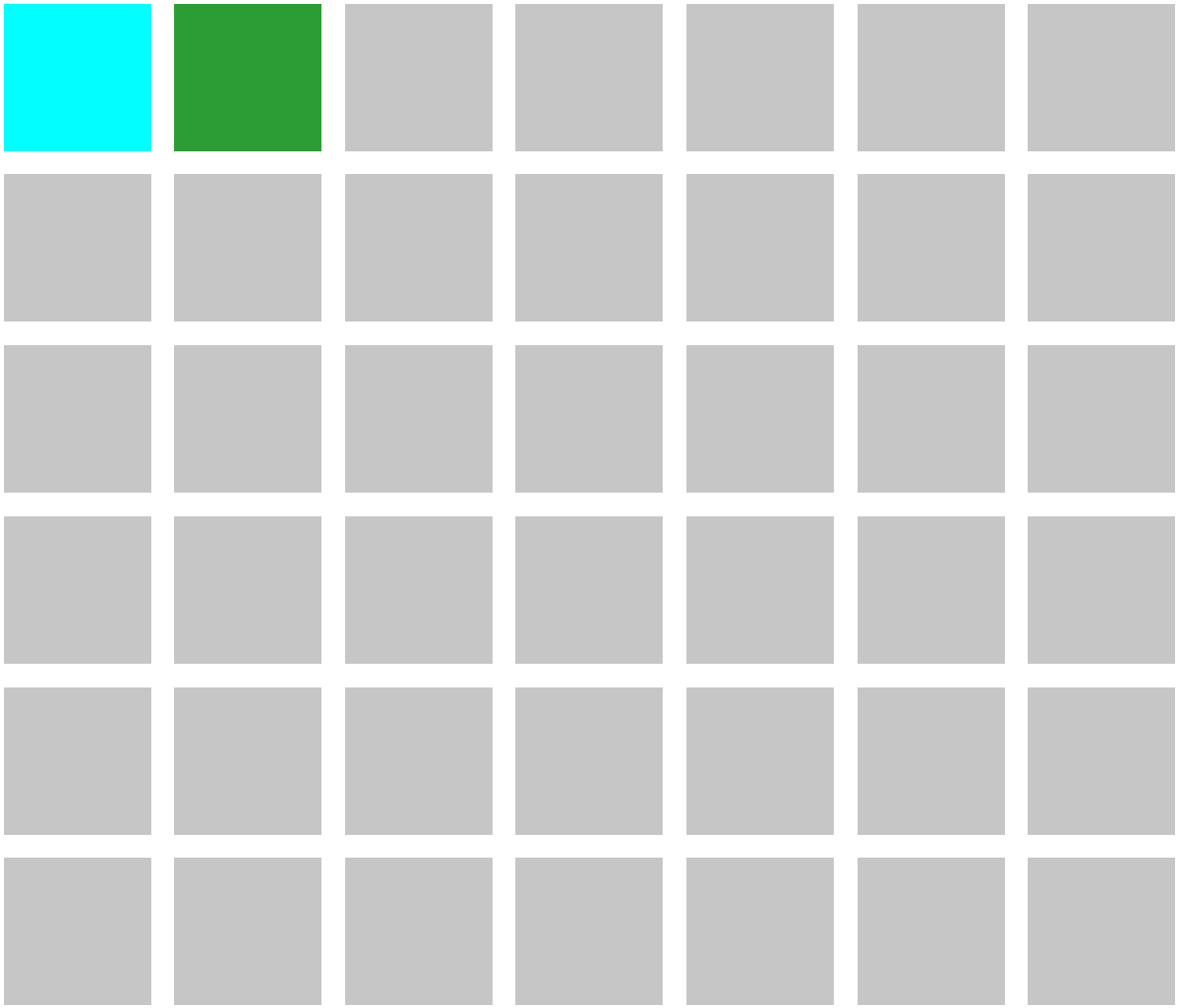
Maximum Likelihood estimate: $arg\max_{\theta} \; log f_{model}(x;\theta)$

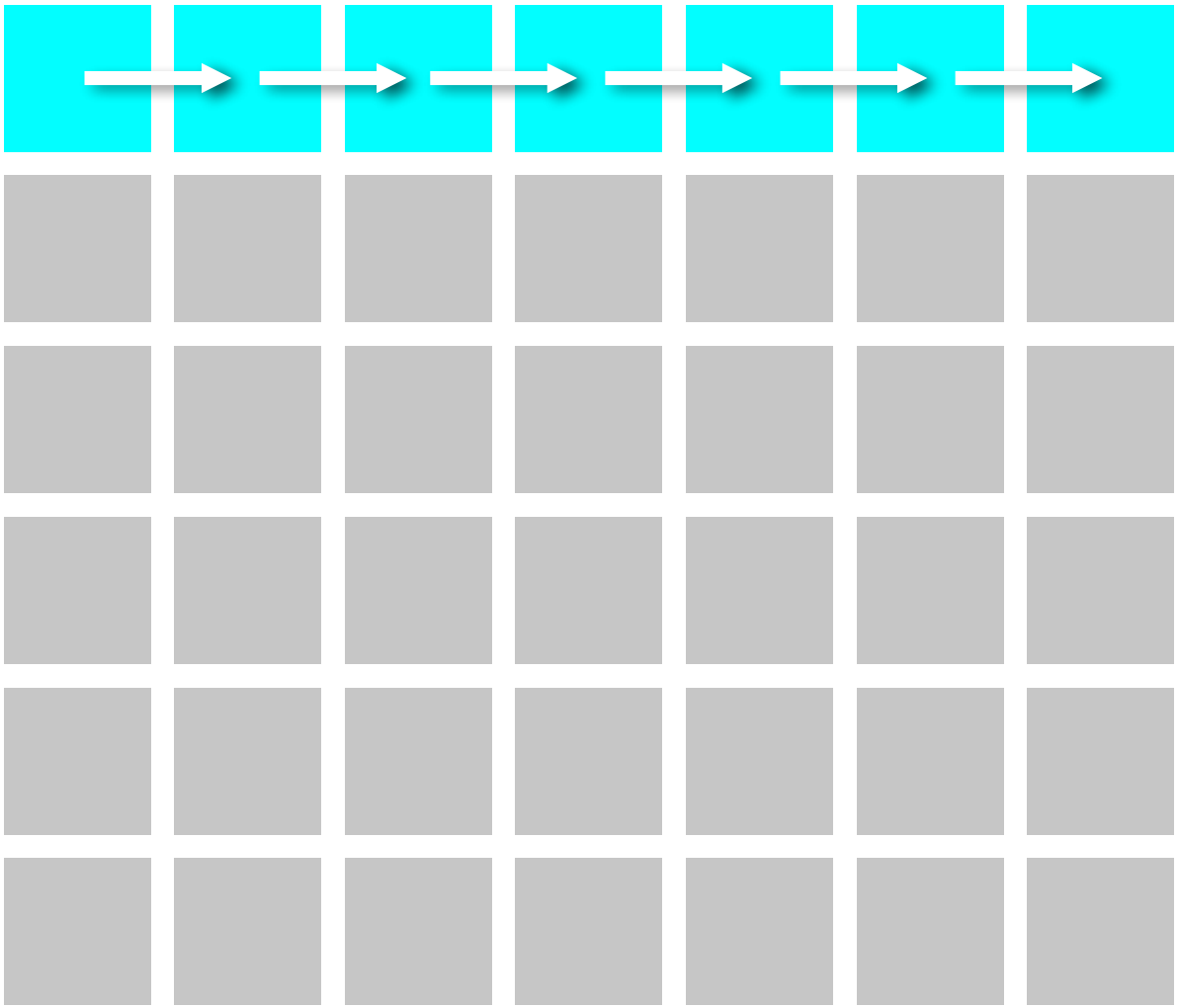
- Maximize this estimator w.r.t. neural network parameter $\boldsymbol{\theta}$.
- In practice we minimize negative of the above estimator as our loss function.

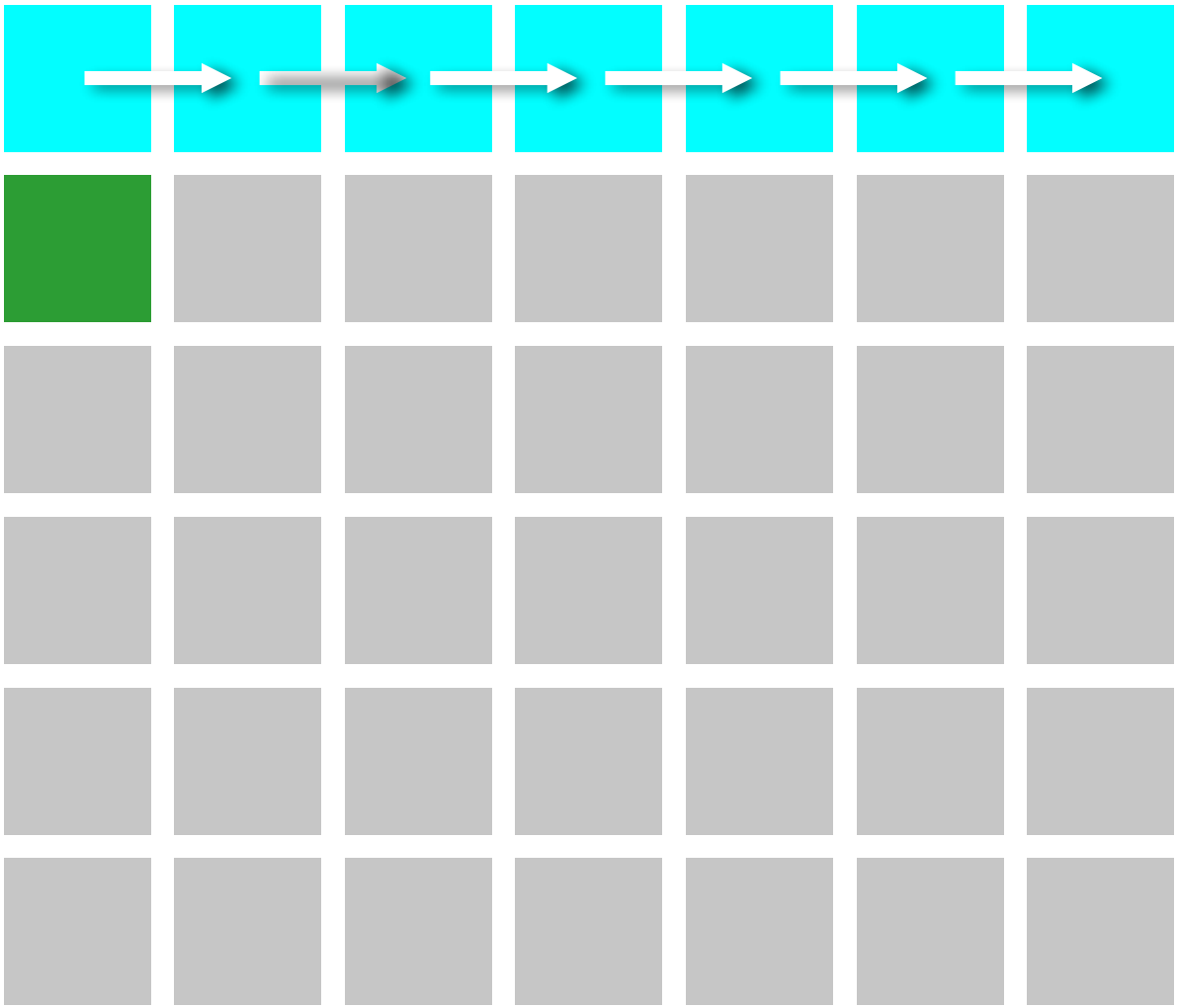

x_1
x_2
x_i
x_{i+1}
pixel values generated sequentially

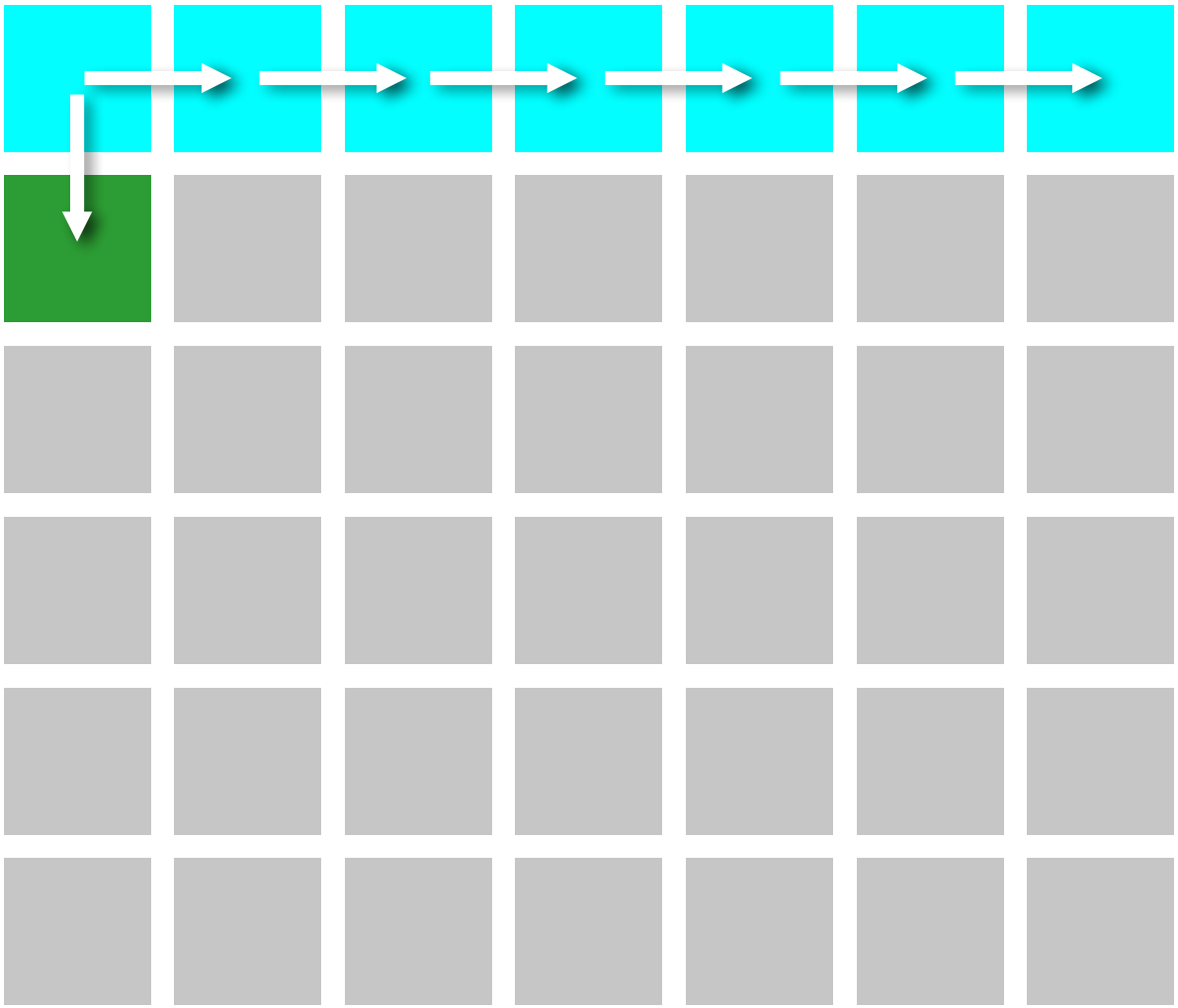


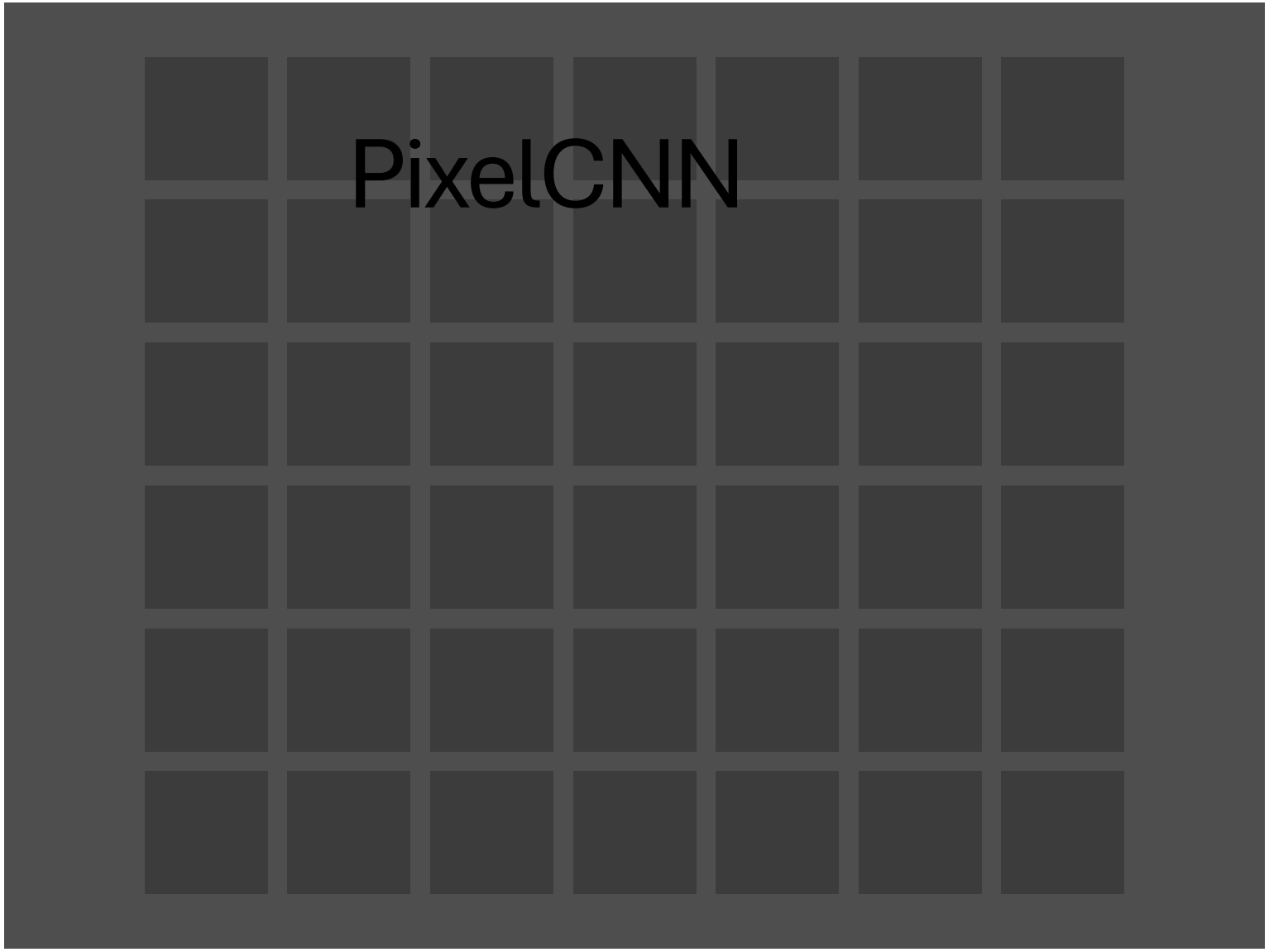


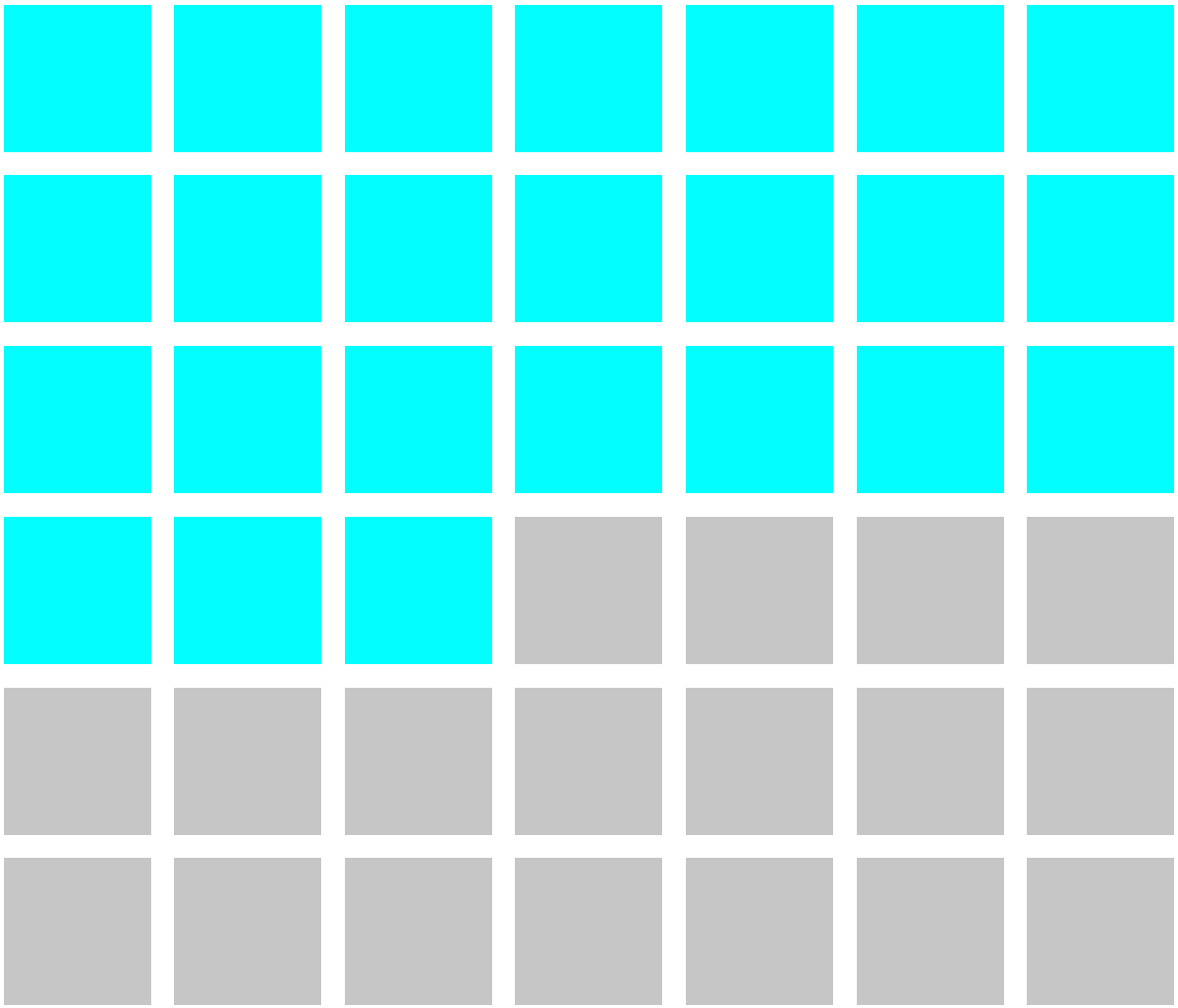


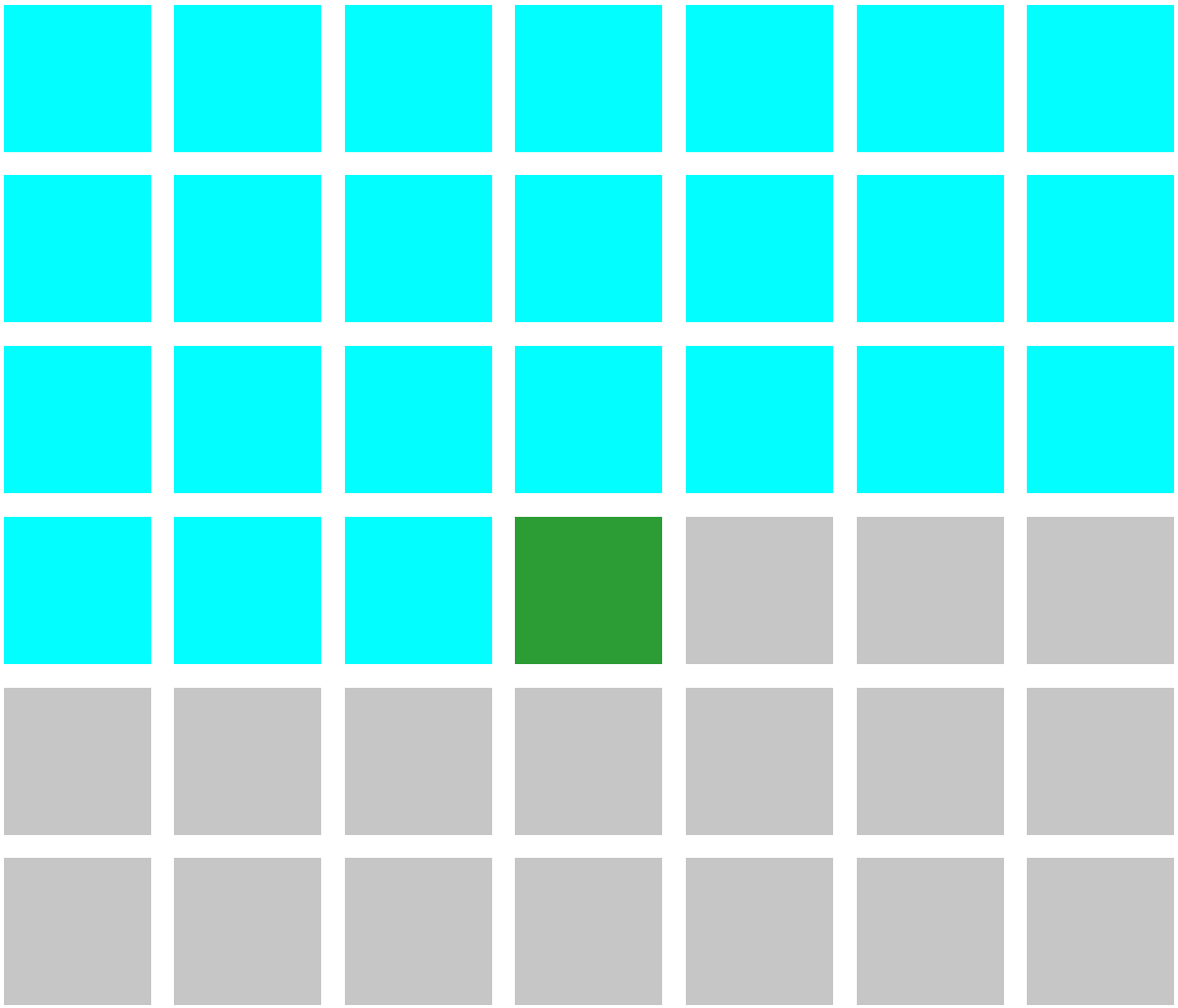


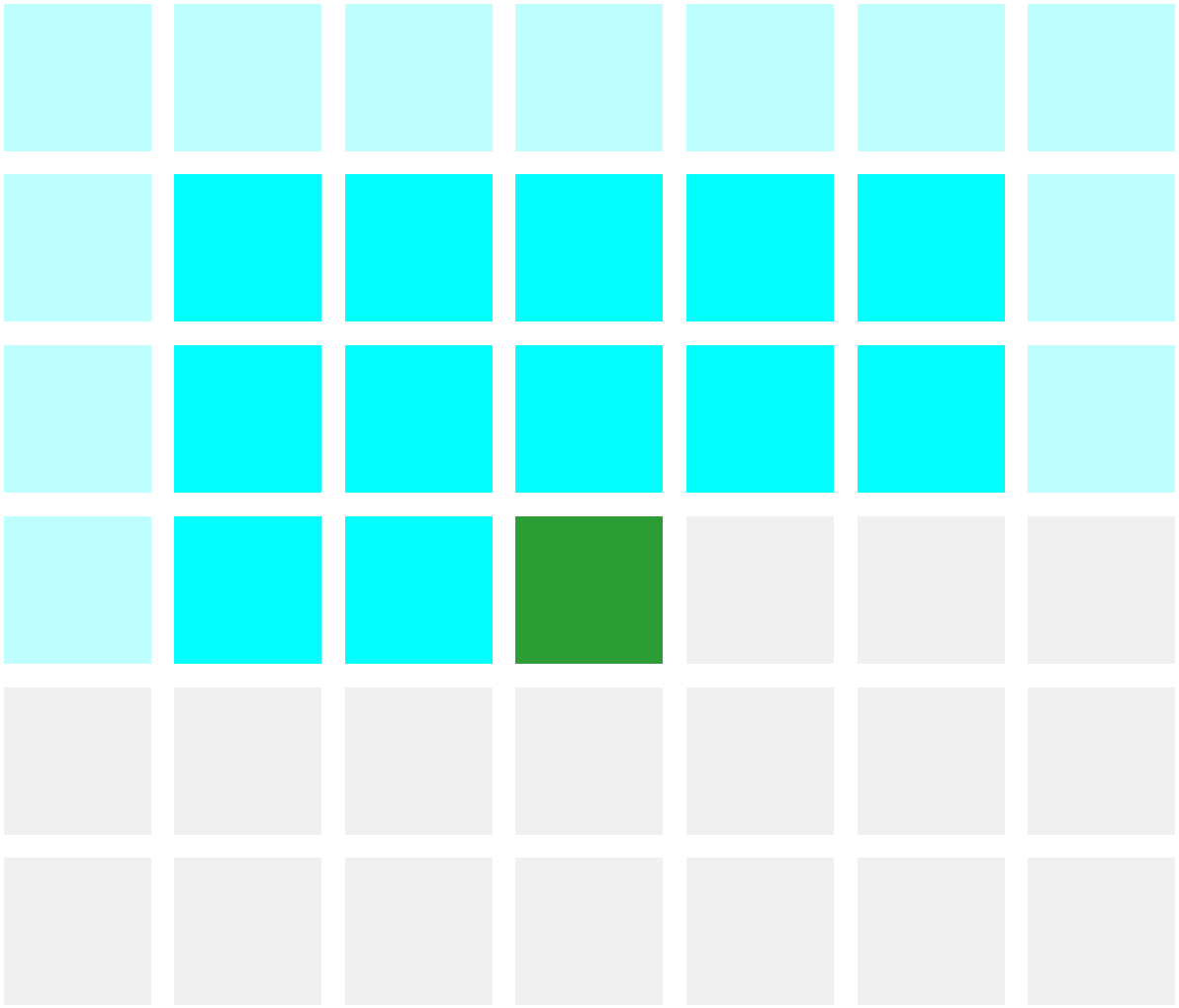


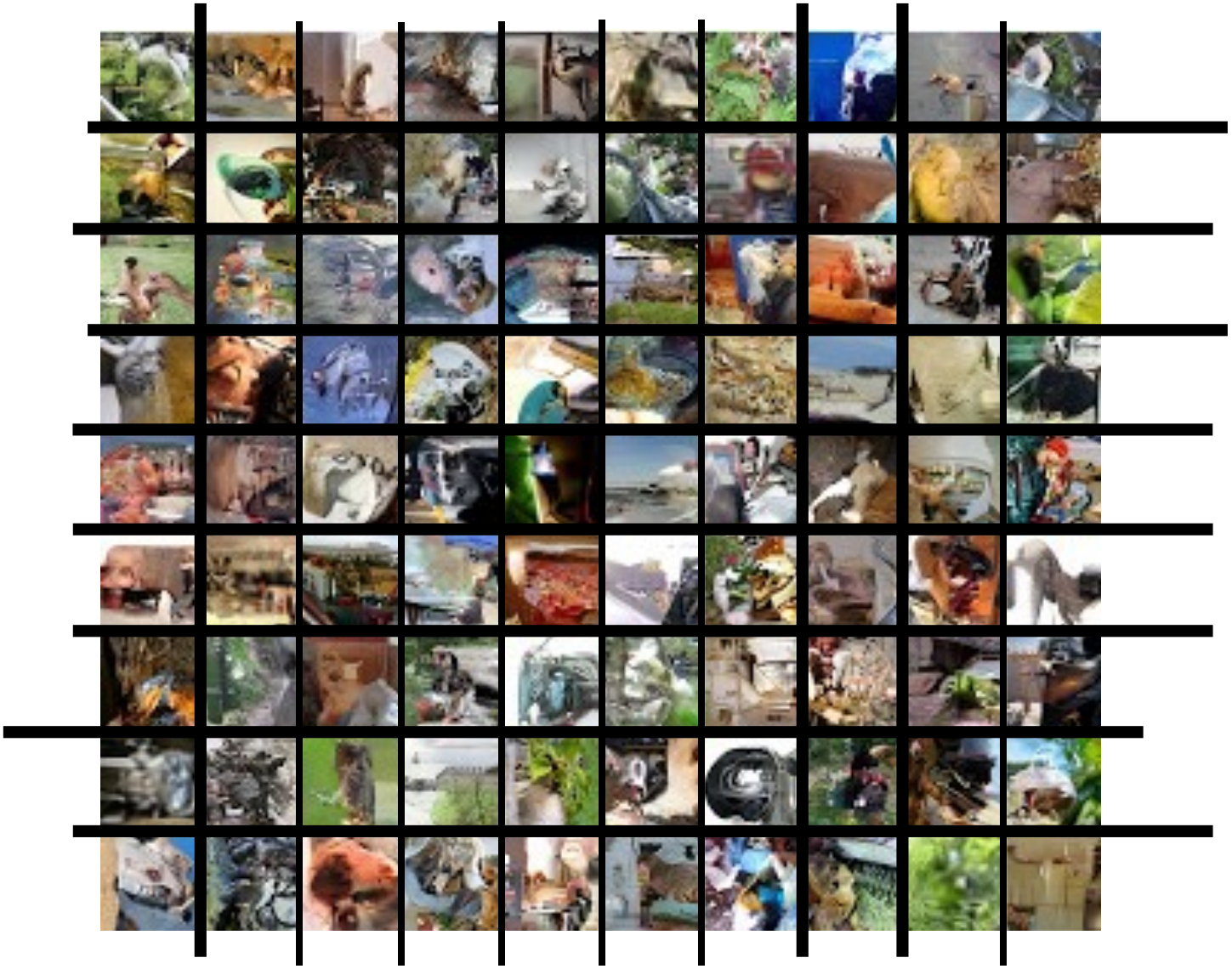


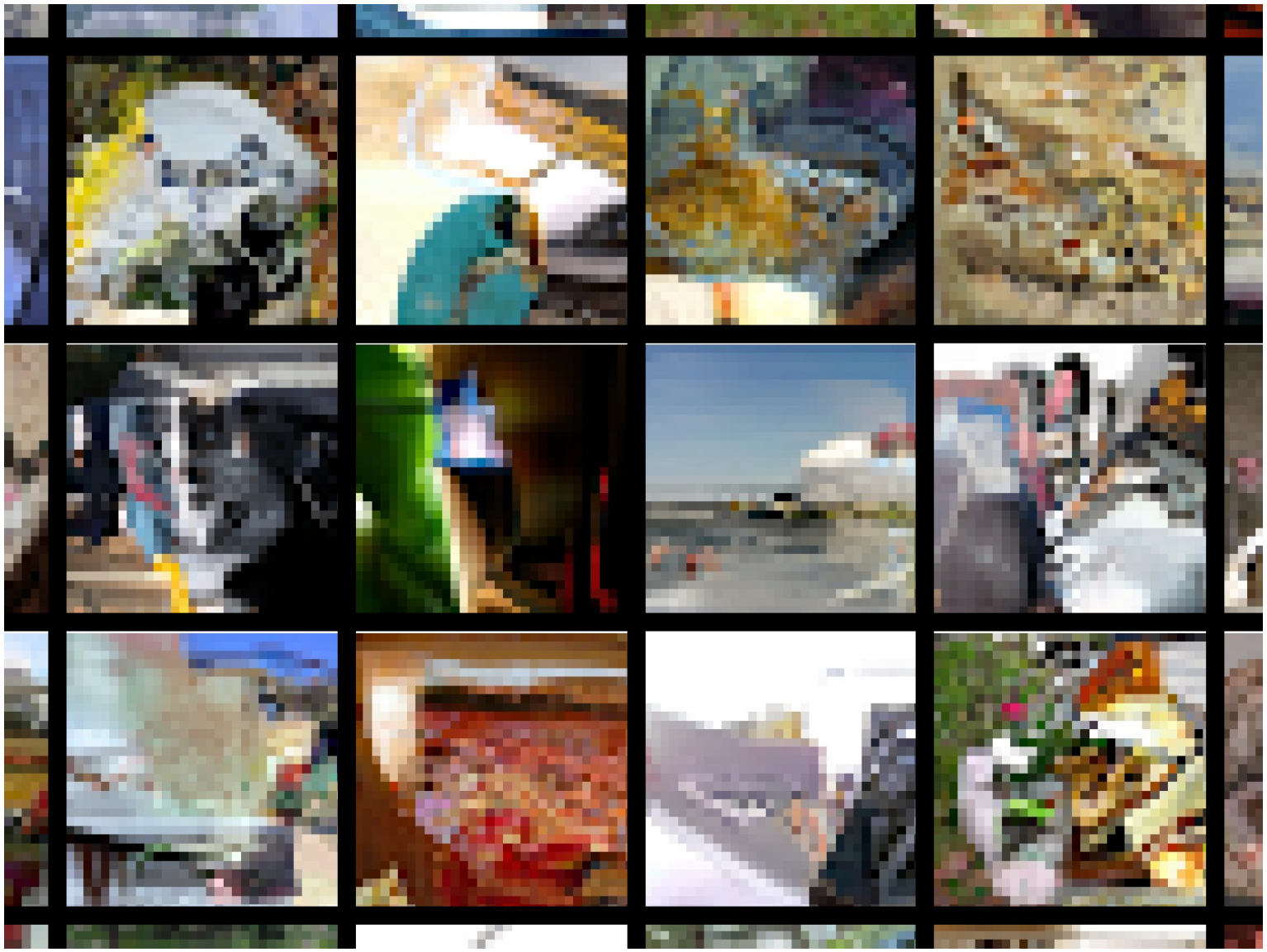

PixelCNN

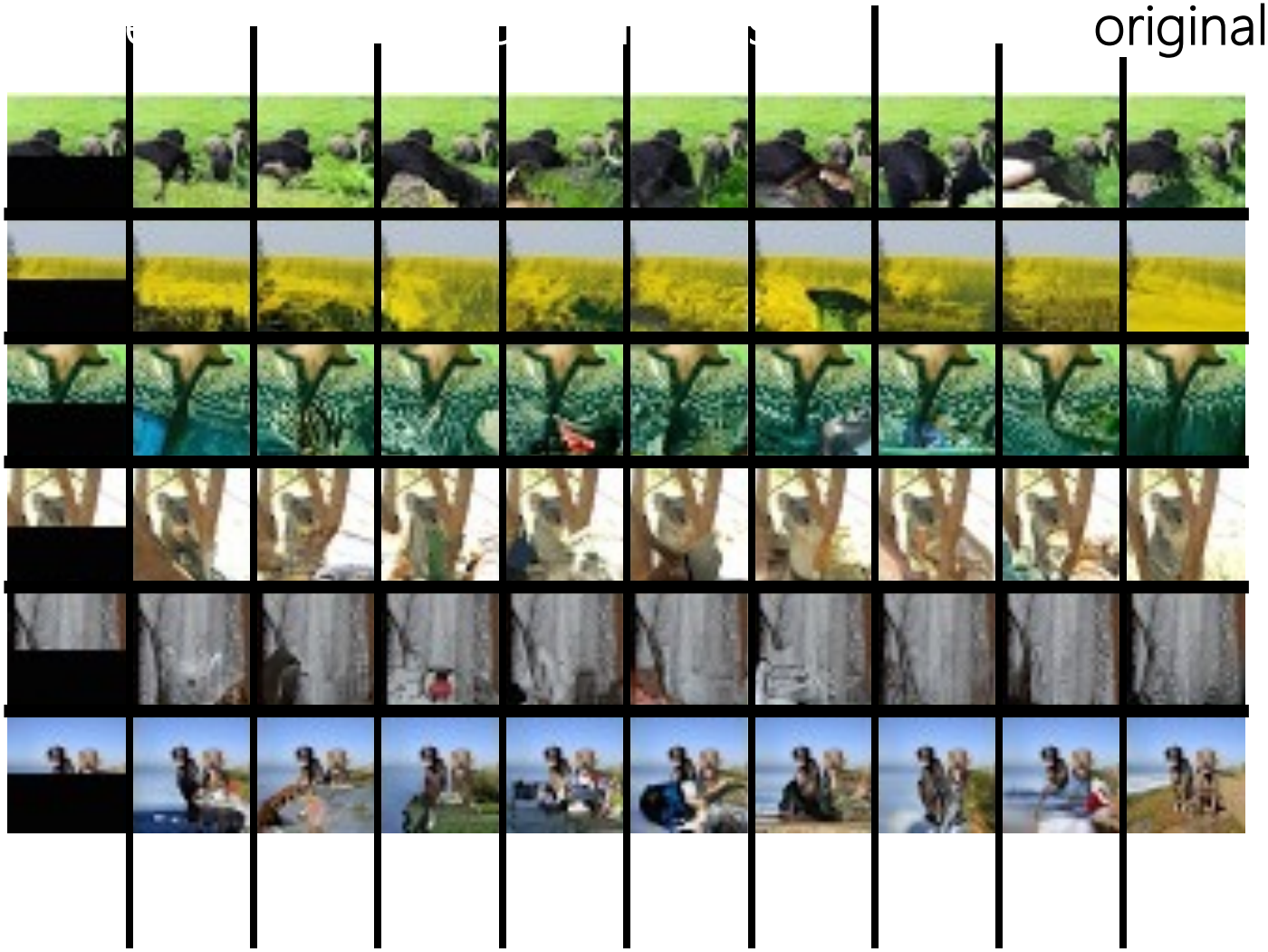




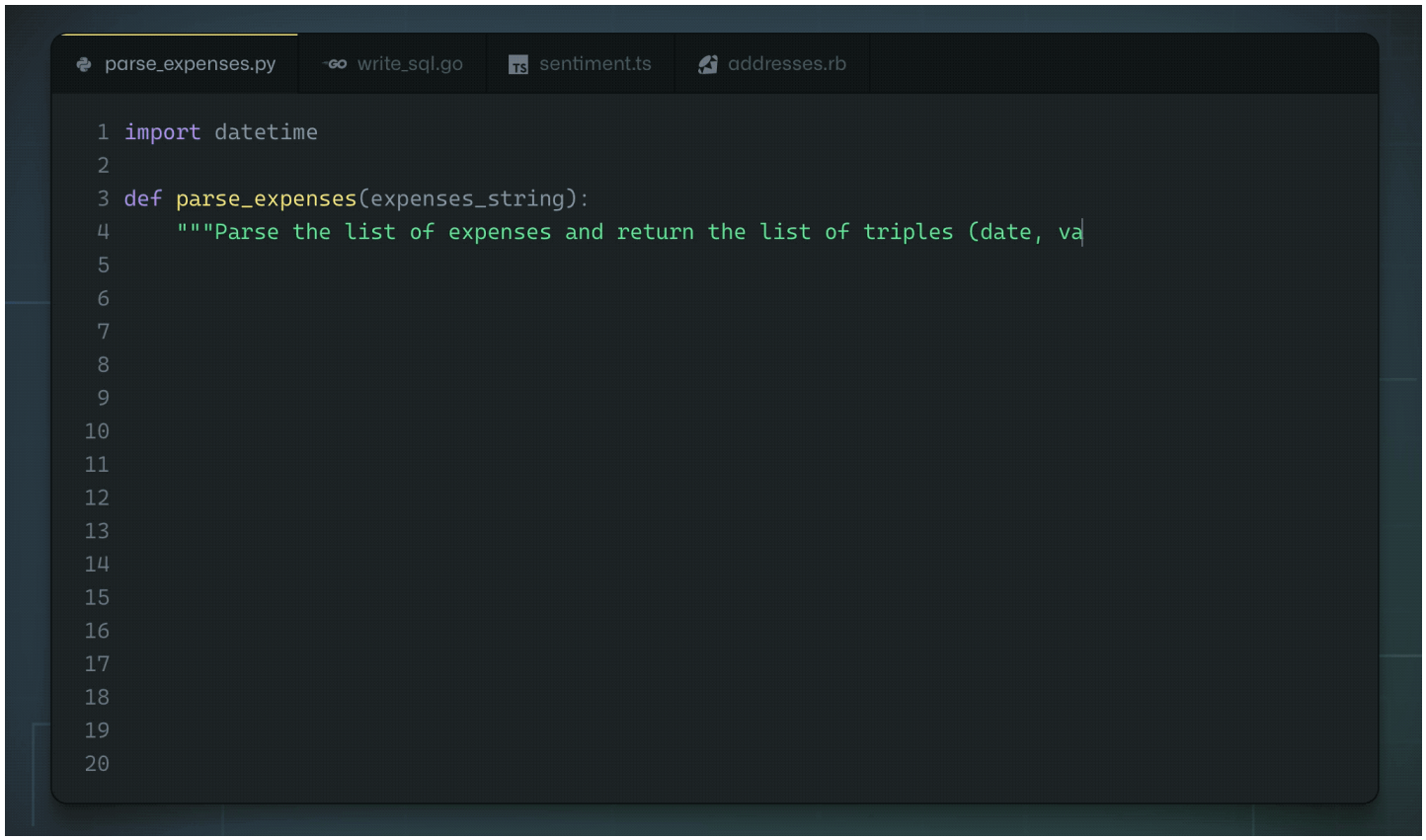







original

Code Generation



OpenAI Codex

sequential pixel generation is SLOW

pixel generation order matters

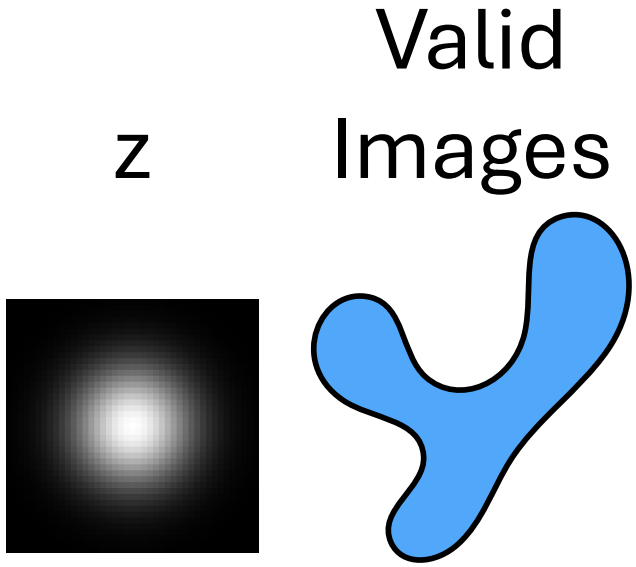
Latent Variable Models

How Many Images Are There?

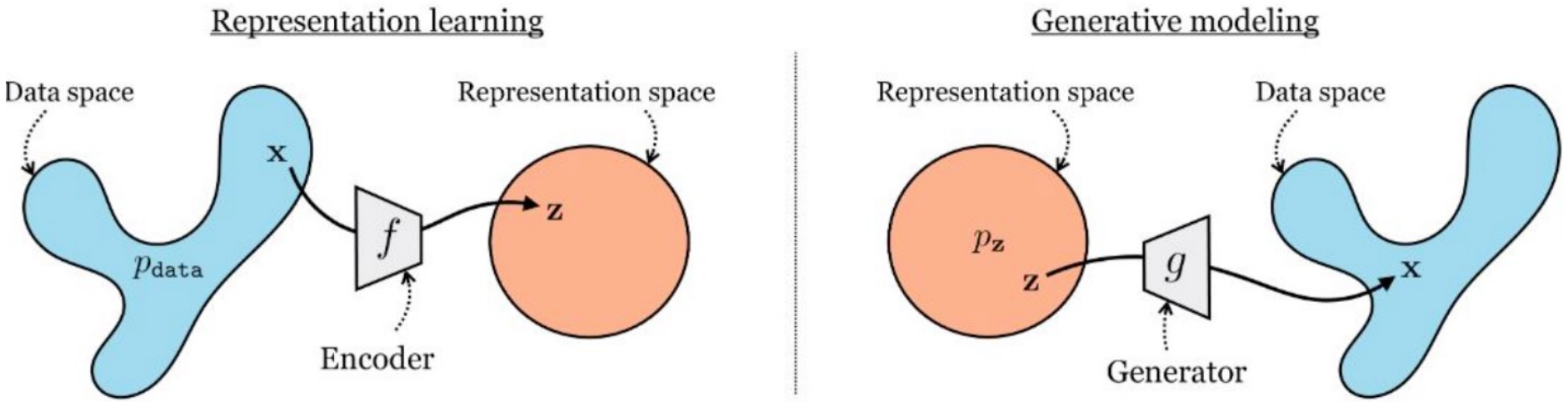
- Set height and width to 1024
- Assume 256^3 (aka 2^{24}) values per pixel
- **How many images can I create?**
- (256^3)~1M
- **Why might it be quite a bit less?**

Learning a Mapping

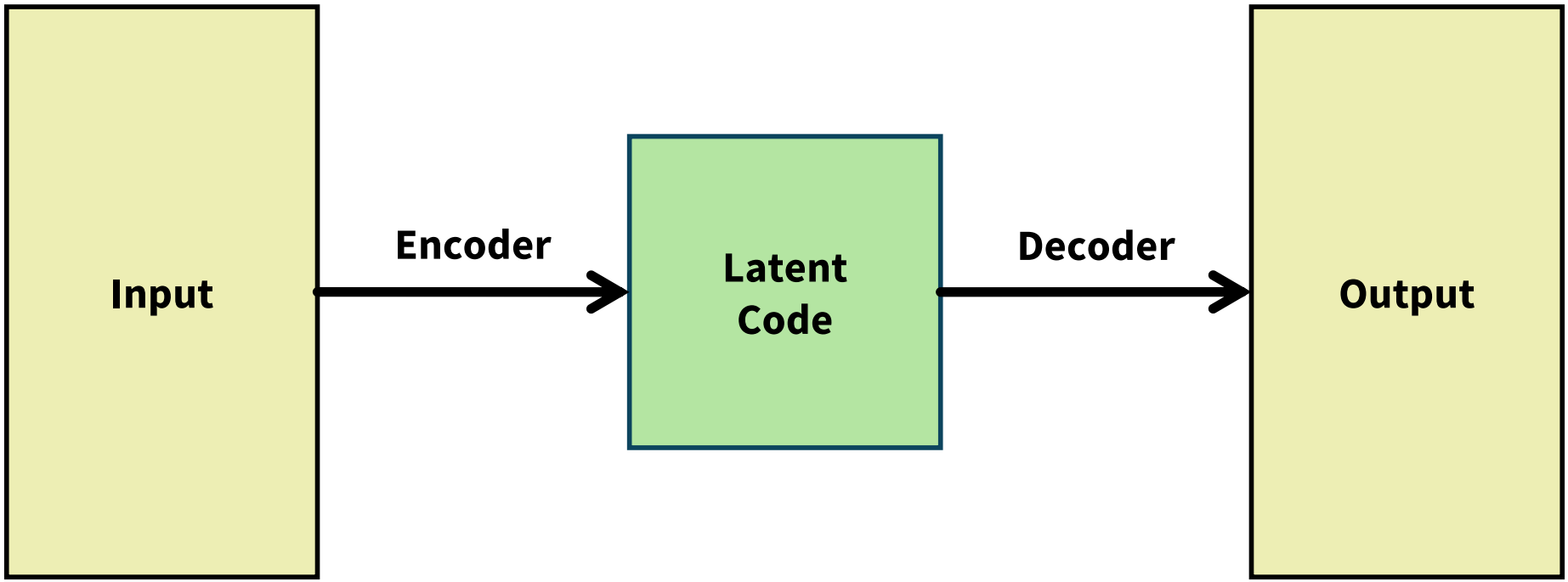
- Want to learn a mapping
- **From:** a “latent” space z, often assume to be the result of sampling N-D Gaussian noise
- **To:** the space of valid images
- Latent Variable Model

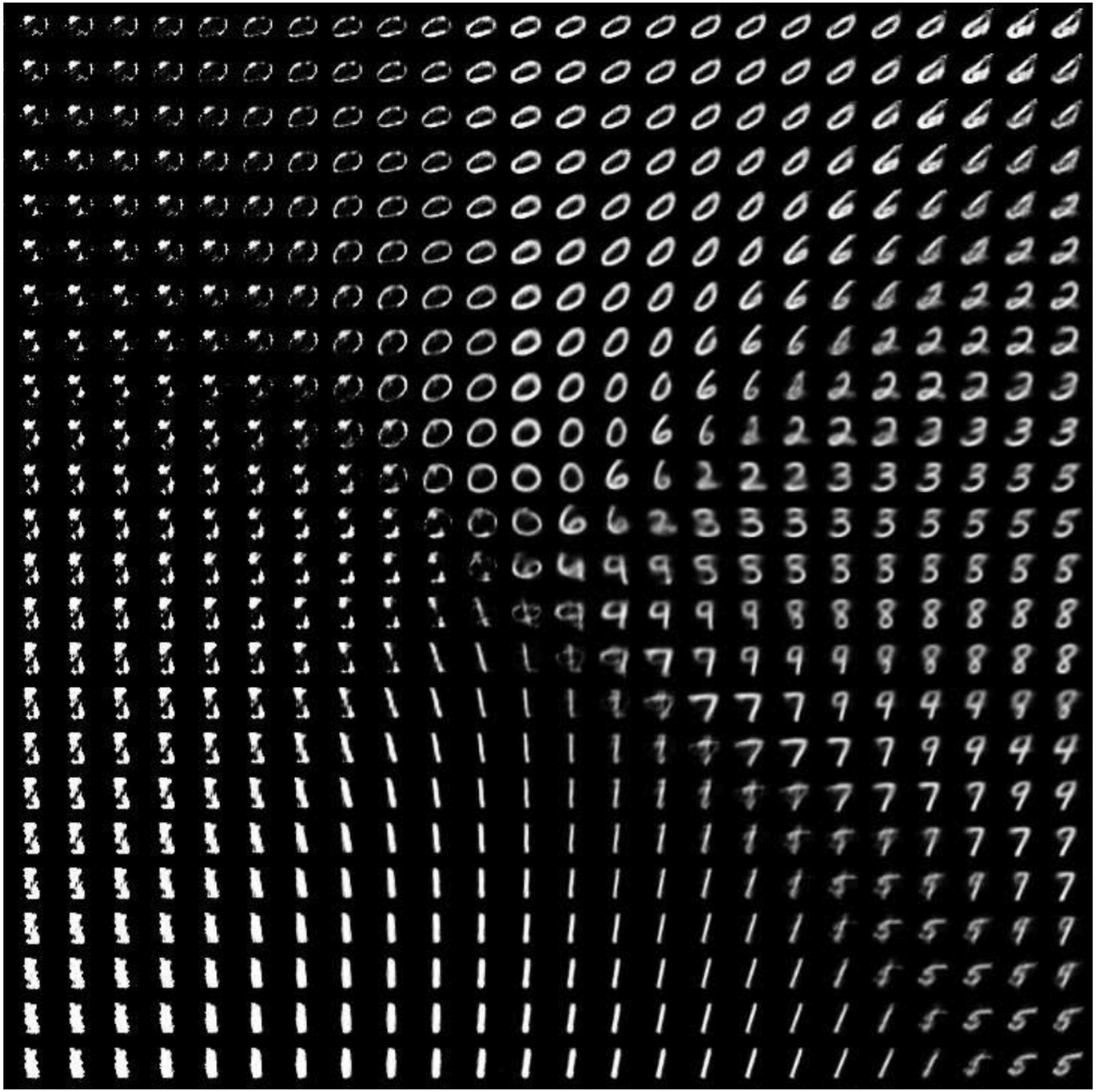


Variational Auto Encoder



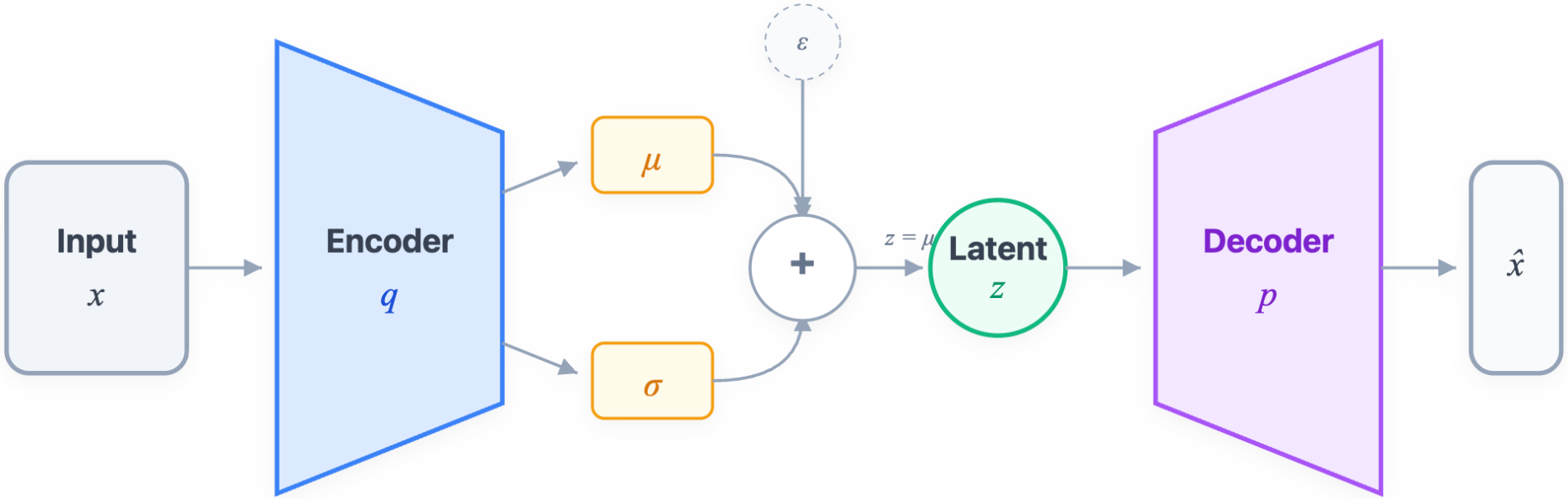
Auto Encoder





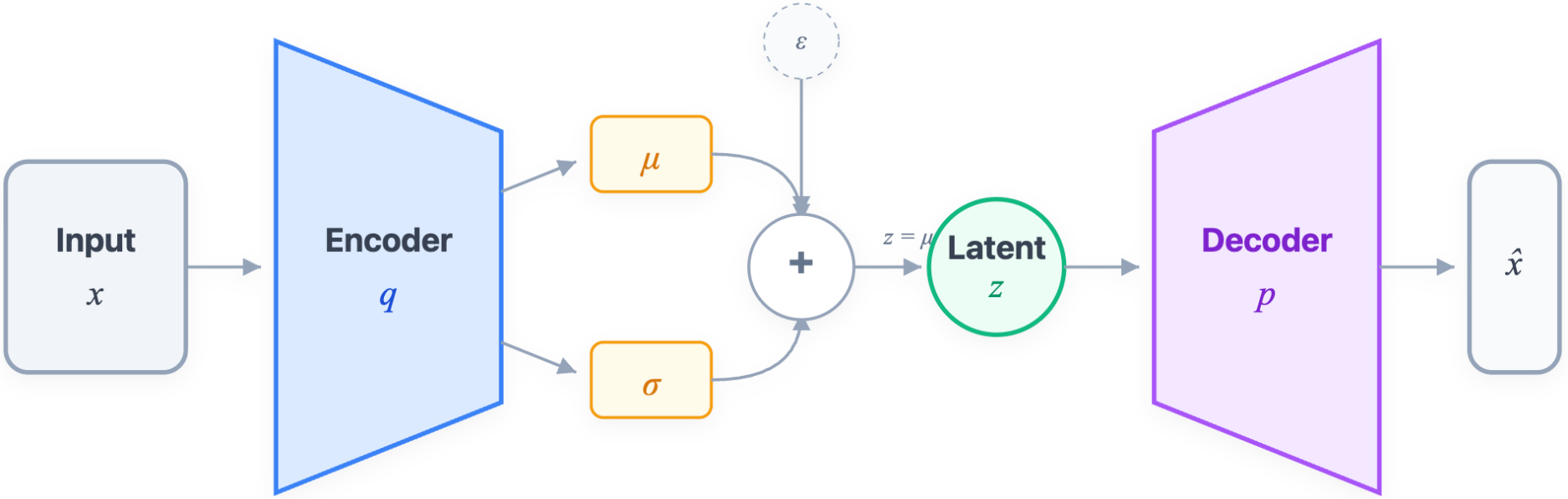
Variational Auto Encoder

- Regularizing the latent space
- Allows random sampling, interpolation, and optimization in the latent space



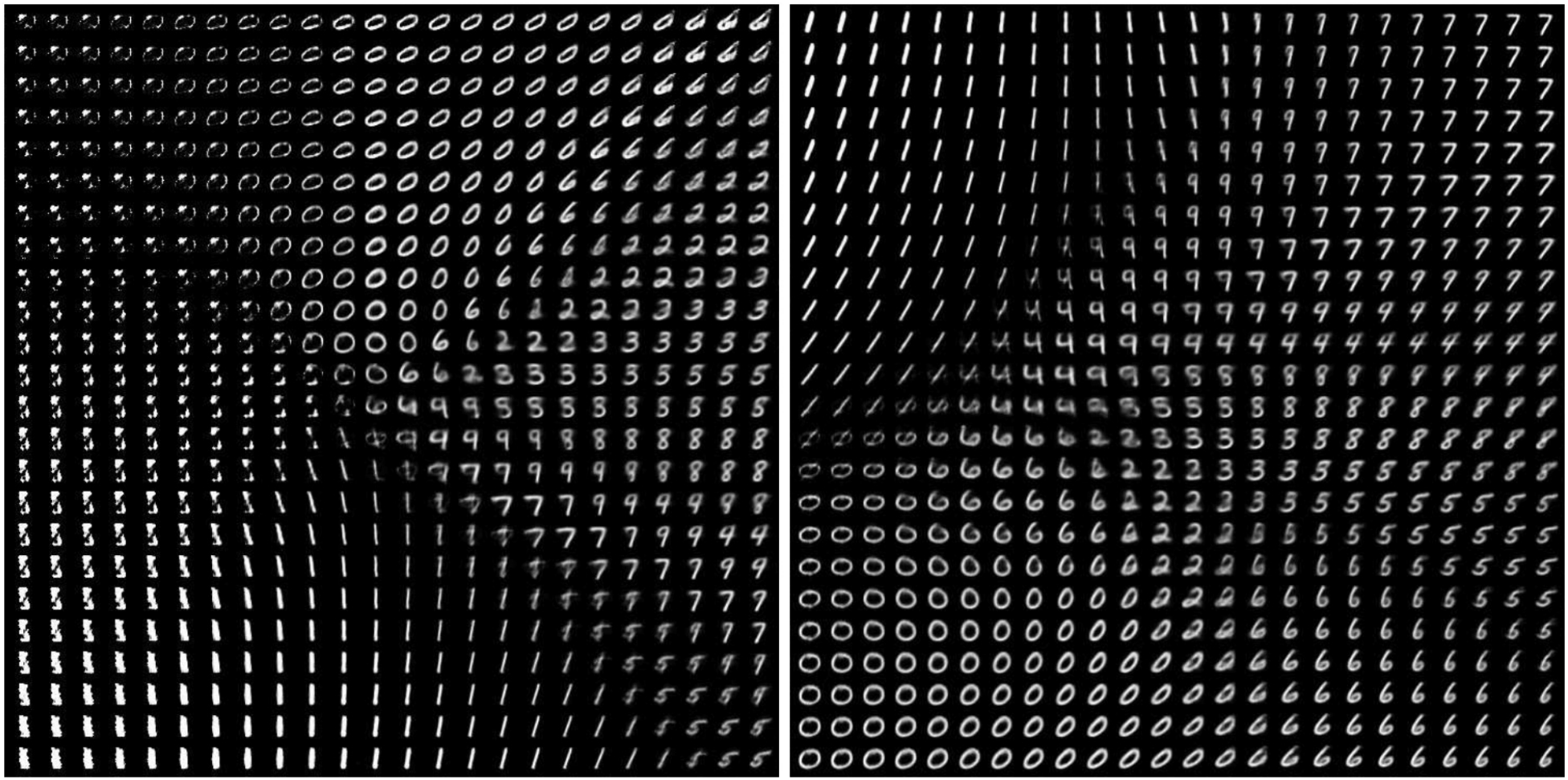
VAE Loss Function

- Minimize the reconstruction loss
- Regularize that z follows N(0,1) distribution
- More on theoretical derivation here



$$L = MSE(x, \hat{x}) - \frac{1}{2}\sum(1 + \log\sigma^2 - \mu^2 - \sigma^2)$$

Latent Space Visualization

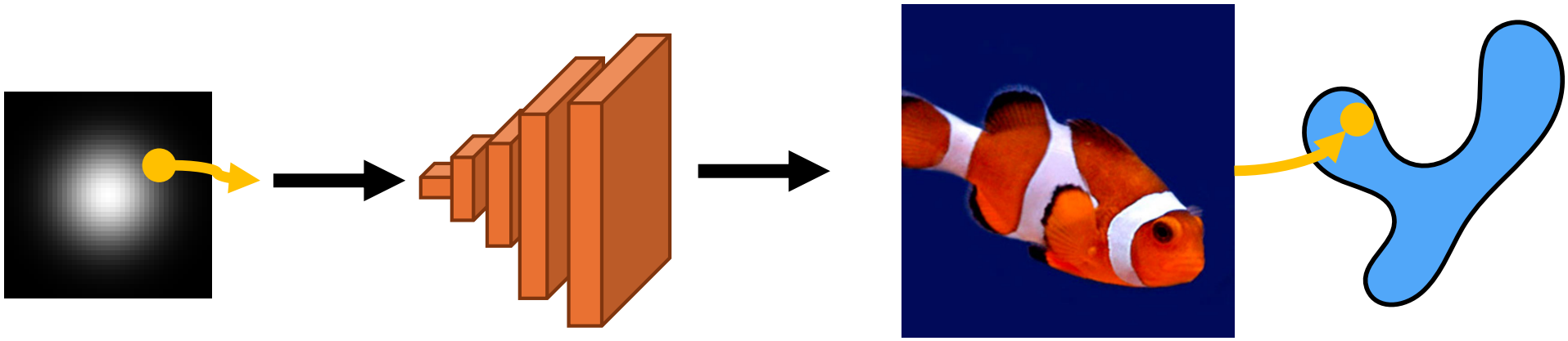


(a) Auto Encoder (AE)

(b) Variational Auto Encoder (VAE)

Generating Data

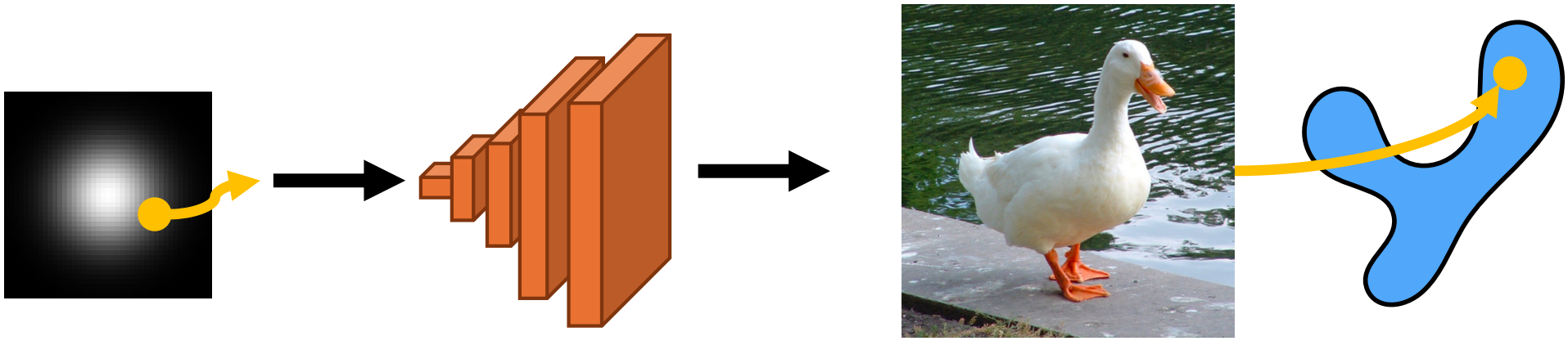
z G(·) x Valid Images



Remake of Slides by Isola, Freeman, Torralba, taken from Owens; diagram, duck + clownfish pictures from IFT.

Generating Data

z G(·) x Valid Images

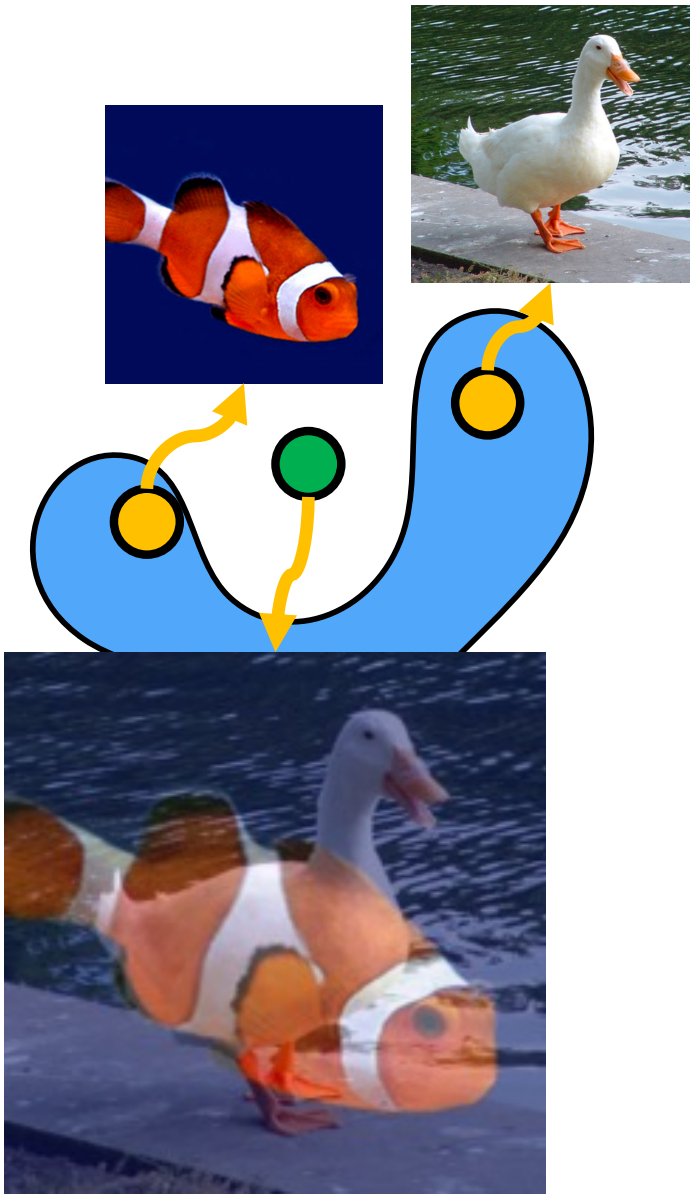


Why The Funny Shape?

Given two valid images, what about their average?

Key things to remember

- Linear combinations of images aren't valid images.
- Explains funny shape ("manifold") and need for a deep network



Generative Adversarial Networks

- Generator tries to make fake images – accepts noise and makes an image
- Discriminator tries to identify fakes – outputs p(real)

Remake of Slides by Isola, Freeman, Torralba, taken from Owens; diagram, duck + clownfish pictures from IFT.

Overall

Discriminator

$$\arg\max_{\mathrm{D}} E_{z,x}[\log\Big(1 - D\big(G(z)\big)\Big) + \log\big(D(x)\big)]$$

Generator

$$\operatorname{argmin}_{\mathrm{G}} E_z[-\log\Big(D\big(G(z)\big)\Big)$$

Slide Credit: Roni Sengupta

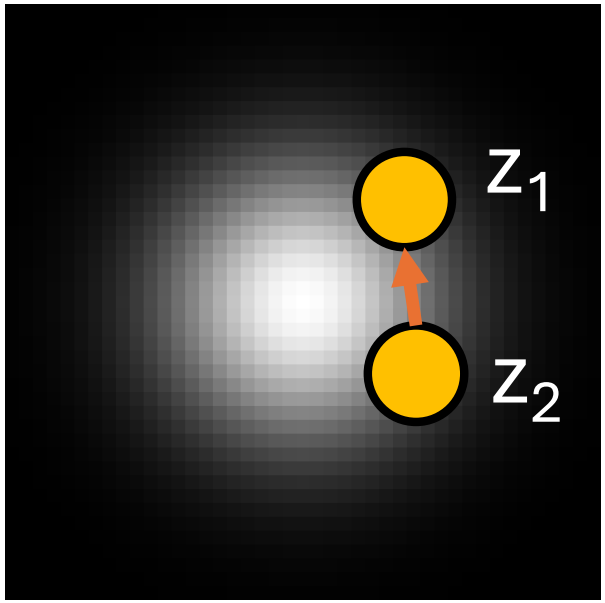
Typically for GAN Training

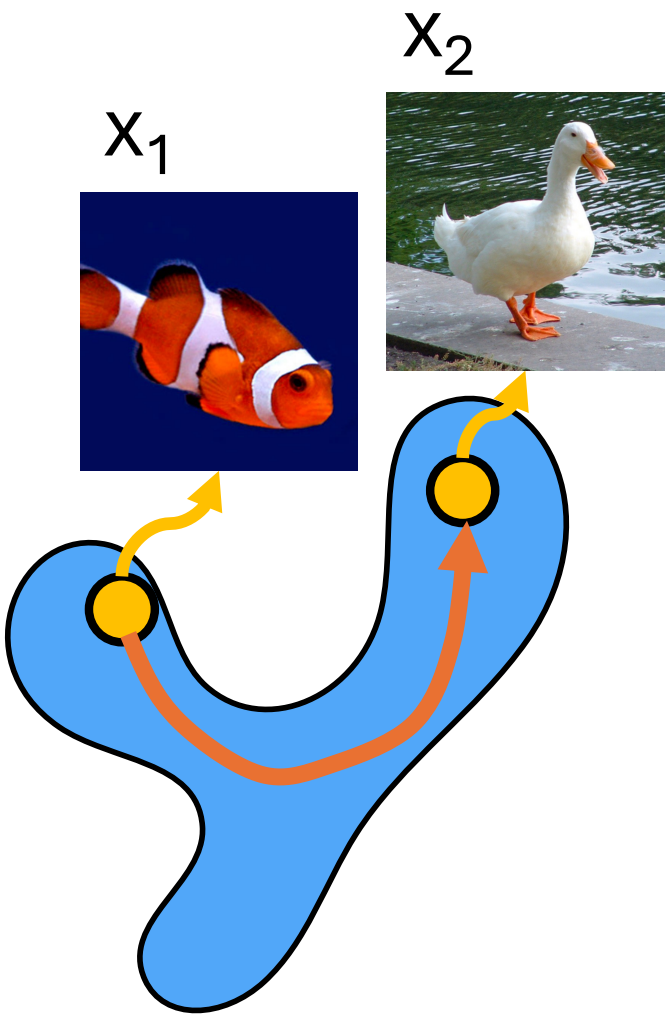
- Very often unstable
- Need regularizations or more stable losses (out of scope)
- Need balance between D and G
- Training discriminator is easier than generator (more people can appreciate art vs # of artists)

Revisiting Averages

Can use z to walk latent space

$G(\alpha z_1+(1-\alpha)z_2)$ for α in [0,1]





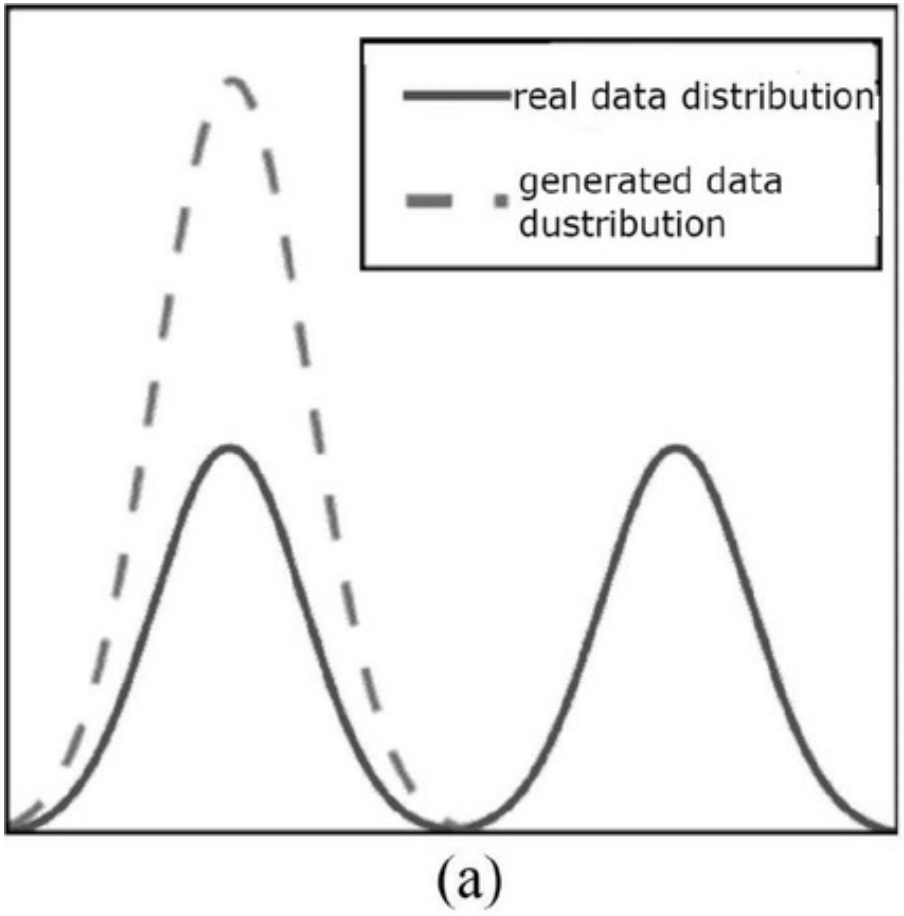
[Karras et al., “Alias-Free Generative Adversarial Networks”, 2021]

[Karras et al., “Alias-Free Generative Adversarial Networks”, 2021]

GANs→Great Results!

However,
Training is sensitive and difficult
Mode Collapse!

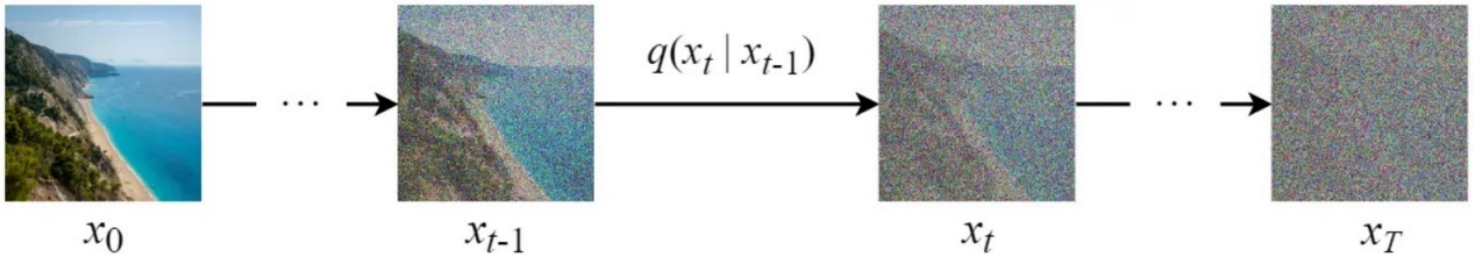
Mode Collapse



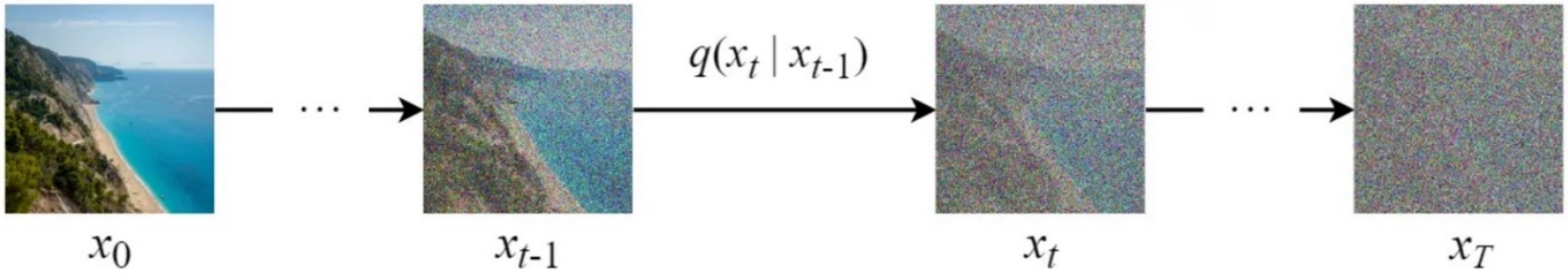
Diffusion Models to Rescue

- Generates better data than any other visual generative models
- Easier to train than generative adversarial networks and does not suffer from mode collapse

Forward Diffusion Process



Forward Diffusion Process



Distribution of the noised images — Output — Mean μ_t — Variance Σ_t

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t;\ \sqrt{1-\beta_t}x_{t-1},\ \beta_t I)$$

Notations:

t : time step (from 0 to T)

x_0 : a data sampled from the real data distrition $q(x)$ (i.e. $x_0 \sim q(x)$)

β_t : variance schedule ($0 \le \beta_t \le 1$, and β_0 = small number, β_T = large number)

I : identity matrix

Stochastic Transformation

- reparameterization trick:
  - When $x \sim P(x) = N(x|\mu, \sigma^2)$
    then $x = \sigma\epsilon + \mu$ where $\epsilon \sim N(\epsilon|0,1)$

- Since $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = N\left(\boldsymbol{x}_t\middle|\sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t\boldsymbol{I}\right)$
  Then $\boldsymbol{x}_t = \sqrt{1-\beta_t}\boldsymbol{x}_{t-1} + \sqrt{\beta_t}\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim N(\boldsymbol{\epsilon}|\boldsymbol{0}, \boldsymbol{I})$

Stochastic Transformation

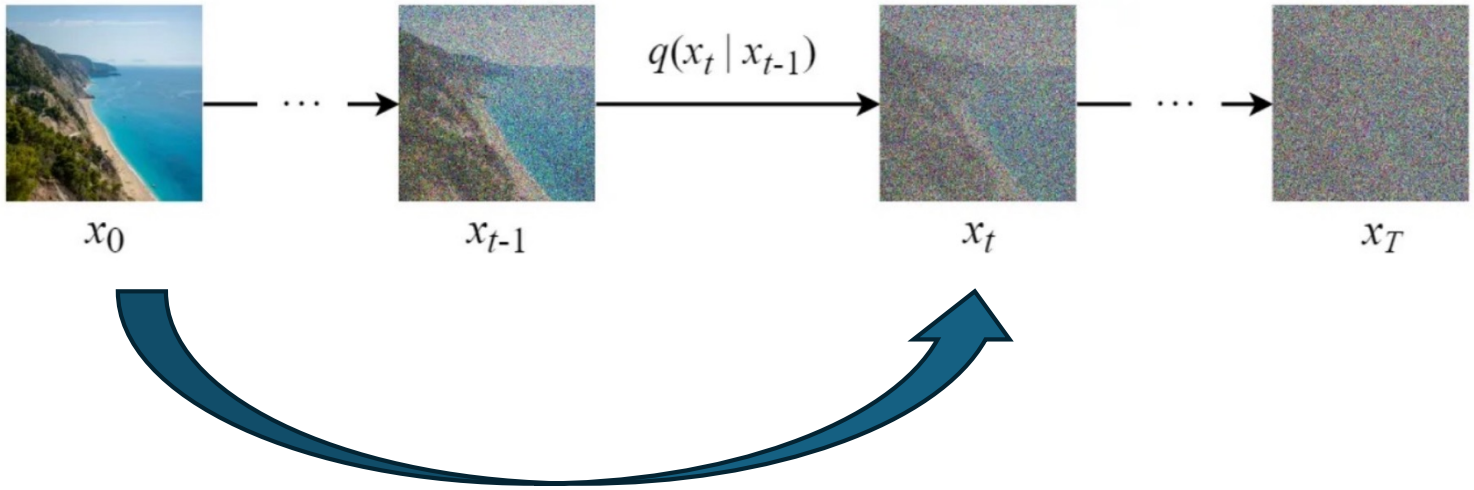
- We can analytically compute x_t in one step. (instead of multi-step)

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon} \sim N(\boldsymbol{\epsilon}|\mathbf{0},\mathbf{1})$

$$\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i \text{ and } \alpha_i = 1 - \beta_i$$

Forward Diffusion Process



Analytically compute in one step!

Stochastic Transformation

- We can analytically compute x_t in one step.

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon} \sim N(\boldsymbol{\epsilon}|\mathbf{0},\mathbf{1})$

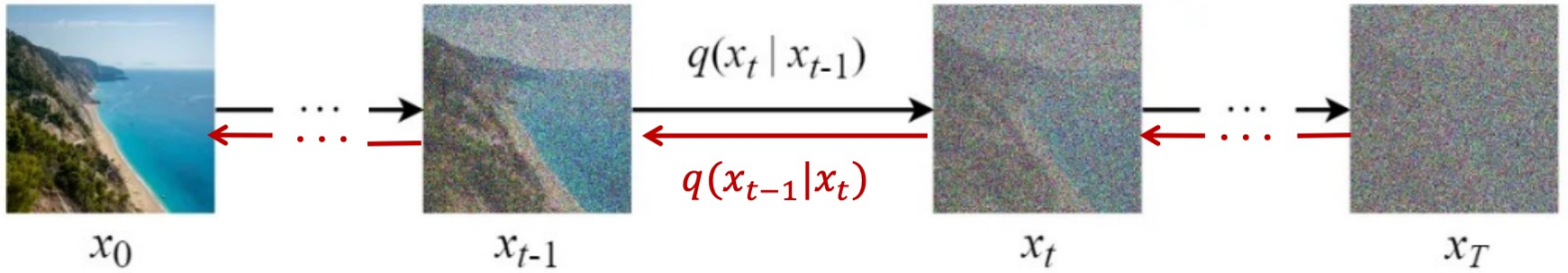
$$\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i \text{ and } \alpha_i = 1 - \beta_i$$

$$\lim_{t\to\infty} \boldsymbol{x}_t = \sqrt{\bar{\alpha}_\infty}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_\infty}\boldsymbol{\epsilon} = \boldsymbol{\epsilon} \text{ since } \bar{\alpha}_\infty \to 0$$

- In the limit, $\boldsymbol{x}_\infty$ is a random vector from an isotropic Gaussian

Reverse Denoising Process

From Steins (medium.com)



Assuming the noise difference between t-1 and t is small,
We can approximate $q(x_{t-1}|x_t)$ with a Gaussian

$$q(\boldsymbol{x_{t-1}}|\boldsymbol{x_t}) = N(\boldsymbol{x_{t-1}}|\tilde{\mu}_t(\boldsymbol{x_t}, t), \sigma_t \boldsymbol{I})$$

We let $\boldsymbol{\sigma_t}$ to follow fixed values, and only estimate the mean $\tilde{\mu}_t(\boldsymbol{x_t}, t)$with a neural network.
(Typically, $\sigma_t = \sqrt{\beta_t}$)

Reverse Denoising Process

- The reverse conditional $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = N(\boldsymbol{x}_{t-1}|\tilde{\mu}_t(\boldsymbol{x}_t, t), \sigma_t \boldsymbol{I})$ does not have a closed form, but Ho, Jain and Abbeel (2020) derived the following approximation for $\tilde{\mu}_t$:

$$\tilde{\mu}_t(\boldsymbol{x}_t, t) \approx \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right)$$

where $\boldsymbol{\epsilon}_t = N(\boldsymbol{\epsilon}|\boldsymbol{0}, \boldsymbol{I})$ is the noise introduced at step t

Reverse Denoising Process

- The reverse conditional $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = N(\boldsymbol{x}_{t-1}|\tilde{\mu}_t(\boldsymbol{x}_t, t), \sigma_t \boldsymbol{I})$ does not have a closed form, but Ho, Jain and Abbeel (2020) derived the following approximation for $\tilde{\mu}_t$:

$$\tilde{\mu}_t(\boldsymbol{x}_t, t) \approx \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right)$$

where $\boldsymbol{\epsilon}_t = N(\boldsymbol{\epsilon}|\boldsymbol{0}, \boldsymbol{I})$ is the noise introduced at step t

- We do not know $\boldsymbol{\epsilon}_t$, but we can train a neural network $\epsilon_\theta(\boldsymbol{x}_t, t)$ to approximate it:

$$\text{Minimize } L(\theta) = \left|\left|\boldsymbol{\epsilon}_t - \epsilon_\theta(\boldsymbol{x}_t, t)\right|\right|_2^2$$

Tractable Posterior via Bayes' Rule

The true reverse process $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is intractable. However, conditioning on the clean image $\mathbf{x}_0$ makes it a tractable Gaussian. We apply Bayes' theorem:

$$
\begin{aligned}
q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) &= \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0) q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \\
&= \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1}) q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)}
\end{aligned}
$$

(The second step uses the Markov property: $\mathbf{x}_t$ only depends on $\mathbf{x}_{t-1}$).

Minimize: $D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))$

Substituting Forward Distributions

We know the forward process transitions are defined as Gaussian distributions:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\mathbf{I})$$
$$q(\mathbf{x}_{t-1}|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0, (1-\bar{\alpha}_{t-1})\mathbf{I})$$

Because the denominator $q(\mathbf{x}_t|\mathbf{x}_0)$ does not depend on $\mathbf{x}_{t-1}$, the posterior distribution is proportional to the product of the numerators:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \propto q(\mathbf{x}_t|\mathbf{x}_{t-1})q(\mathbf{x}_{t-1}|\mathbf{x}_0)$$

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Expanding the Densities

We substitute the Gaussian probability density functions. Ignoring factors independent of $\mathbf{x}_{t-1}$, the exponent is proportional to:

$$\exp\left(-\frac{1}{2}\left[\frac{(\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_{t-1})^2}{1-\alpha_t} + \frac{(\mathbf{x}_{t-1} - \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0)^2}{1-\bar{\alpha}_{t-1}}\right]\right)$$

To find the distribution of $\mathbf{x}_{t-1}$, we will expand these squares and group the terms into a standard quadratic form: $-\frac{1}{2}(A\mathbf{x}_{t-1}^2 - 2B\mathbf{x}_{t-1} + C)$.

Expanding the Squares

We treat $\mathbf{x}_t$ and $\mathbf{x}_0$ as constants and expand the numerators to isolate $\mathbf{x}_{t-1}$:

$$(\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_{t-1})^2 = \alpha_t \mathbf{x}_{t-1}^2 - 2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1} + \mathbf{x}_t^2$$
$$(\mathbf{x}_{t-1} - \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0)^2 = \mathbf{x}_{t-1}^2 - 2\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0\mathbf{x}_{t-1} + \bar{\alpha}_{t-1}\mathbf{x}_0^2$$

Now we will plug these expansions back into our fractions and group the coefficients for $\mathbf{x}_{t-1}^2$ (to find A) and $\mathbf{x}_{t-1}$ (to find B).

The Tractable Posterior Mean

For a Gaussian distribution in the form $\exp(-\frac{1}{2}[Ax^2 - 2Bx + C])$, the mean is precisely $\mu = \frac{B}{A}$.

Dividing our B by our A algebraically yields the precise formula for the mean of the conditional reverse process:

$$\begin{aligned}\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) &= \frac{B}{A} \\ &= \frac{\sqrt{\alpha_t}(1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}(1 - \alpha_t)}{1 - \bar{\alpha}_t}\mathbf{x}_0\end{aligned}$$

Expressing $\mathbf{x}_0$ in terms of $\mathbf{x}_t$

During the reverse inference process, we do not know the clean image $\mathbf{x}_0$. Using the reparameterization trick from the forward process, we define $\mathbf{x}_t$ as:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}_t$$

We rearrange this equation to solve for the unknown $\mathbf{x}_0$:

$$\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}\left(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}_t\right)$$

Algebraic Simplification

Distributing and finding a common denominator for the $\mathbf{x}_t$ term:

$$\begin{aligned}\frac{\alpha_t(1-\bar{\alpha}_{t-1})+(1-\alpha_t)}{\sqrt{\alpha_t}(1-\bar{\alpha}_t)}\mathbf{x}_t &= \frac{1-\bar{\alpha}_t}{\sqrt{\alpha_t}(1-\bar{\alpha}_t)}\mathbf{x}_t \\ &= \frac{1}{\sqrt{\alpha_t}}\mathbf{x}_t\end{aligned}$$

Multiplying the remaining parts for the $\boldsymbol{\epsilon}_t$ term:

$$-\frac{1-\alpha_t}{\sqrt{\alpha_t}(1-\bar{\alpha}_t)}\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t = -\frac{1-\alpha_t}{\sqrt{\alpha_t}\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t$$

The Final Approximation

Combining our simplified terms and factoring out $\frac{1}{\sqrt{\alpha_t}}$ gives the final approximation for the reverse conditional mean derived by Ho, Jain, and Abbeel (2020):

$$\tilde{\mu}_t(\mathbf{x}_t, t) \approx \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_t \right)$$

We do not know $\boldsymbol{\epsilon}_t$, but we can train a neural network $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ to approximate it:

$$\text{Minimize } L(\theta) = \left\| \boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right\|_2^2$$

Training Algorithm

Repeat

- $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$
- $t \sim uniform(\{1, \dots, T\})$
- $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \boldsymbol{I})$
- $\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}$
- $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \nabla_{\boldsymbol{\theta}} \left|\left|\epsilon - \epsilon_\theta(\boldsymbol{x}_t, t)\right|\right|_2^2$

Until convergence

Noise Scheduling

- Linear:

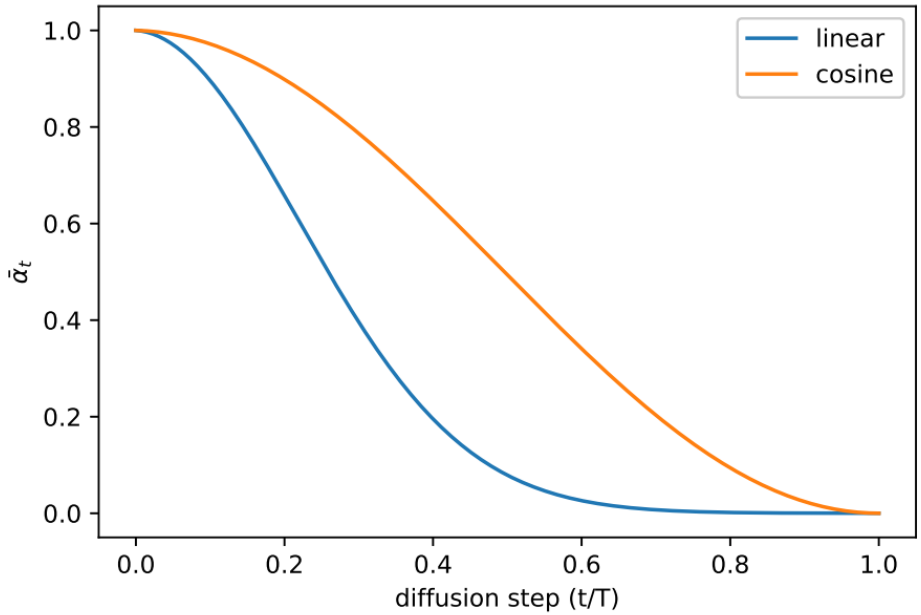
  $\beta_t = \beta_{start} + (\beta_{end} - \beta_{start}) * (t / T)$

  Typically, $\beta_{start} = 0.0001$, $\beta_{start} = 0.02$, $T = 1000$

- Cosine (Improved Denoising Diffusion Probabilistic Models. 2021):

  $\beta_t = 1 - \cos^2(t/T * \pi/2) / \cos^2((t-1)/T * \pi/2)$

  $\beta_t = \text{clip}(\beta_t, \beta_{start}, \beta_{end})$



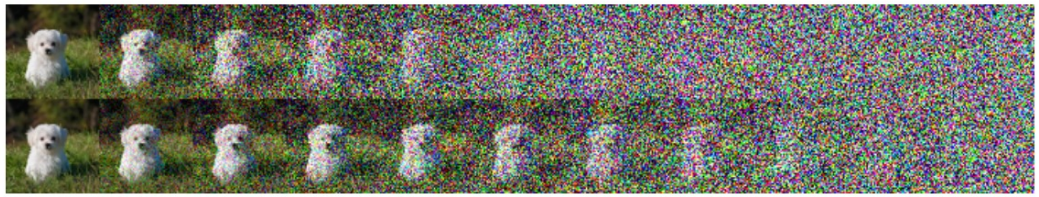
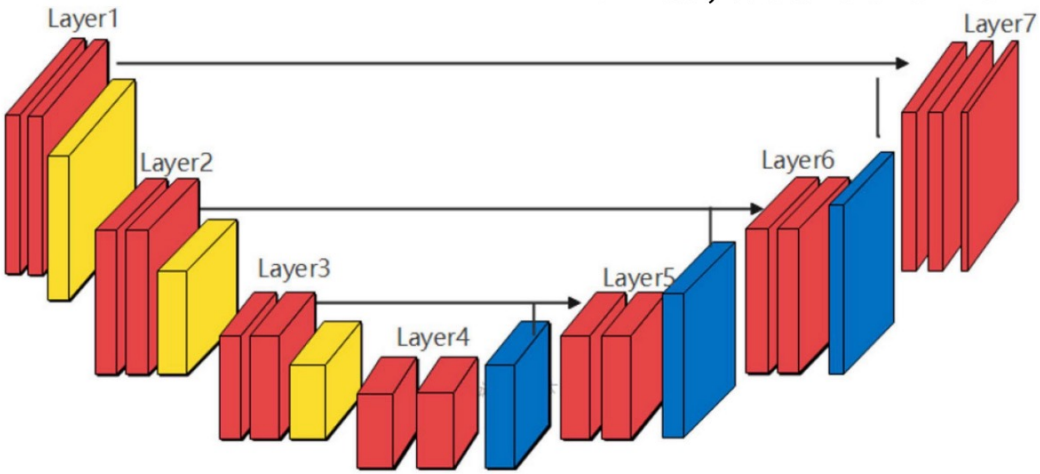


Figure 3. Latent samples from linear (top) and cosine (bottom) schedules respectively at linearly spaced values of t from 0 to T. The latents in the last quarter of the linear schedule are almost purely noise, whereas the cosine schedule adds noise more slowly

UNet

From Cai, Wu and Chen 2022



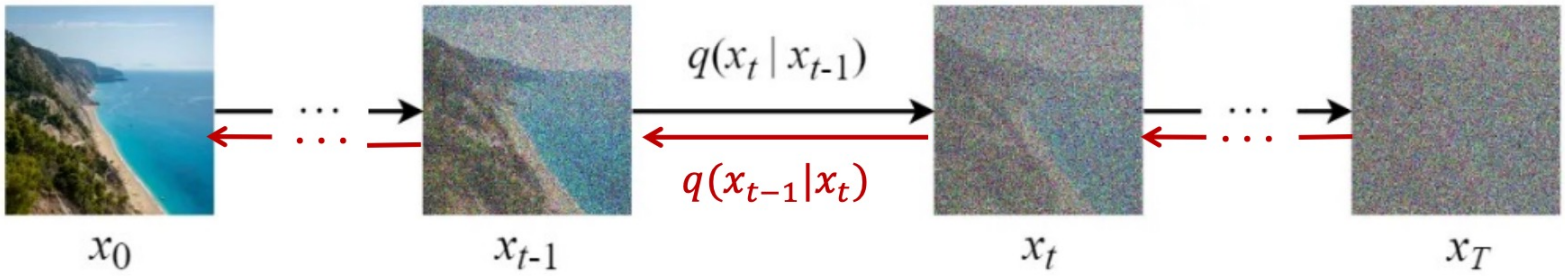
Conv+BatchNormalization+ReLU

Pooling operation

Upsampling Layer

Skip-Connection

Reverse Denoising Process



- We do not know $\boldsymbol{\epsilon}_t$, but we can train a neural network $\epsilon_\theta(\boldsymbol{x}_t, t)$ to approximate it:

$$\text{Minimize } L(\theta) = \left|\left|\boldsymbol{\epsilon}_t - \epsilon_\theta(\boldsymbol{x}_t, t)\right|\right|_2^2$$

Data Generation Alg

$$\tilde{\mu}_t(\mathbf{x}_t, t) \approx \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\right)$$

$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = N(\boldsymbol{x}_{t-1}|\tilde{\mu}_t(\boldsymbol{x}_t, t), \sigma_t \boldsymbol{I})$$

- $\boldsymbol{x}_T \sim N(\boldsymbol{x}|\boldsymbol{0}, \boldsymbol{I})$
- For $t = T, \ldots, 1$ do
  - $\boldsymbol{\epsilon} \sim N(\boldsymbol{\epsilon}|\boldsymbol{0}, \boldsymbol{I})$ if $t > 1$, else $\boldsymbol{\epsilon} = \boldsymbol{0}$
  - $\boldsymbol{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(\boldsymbol{x}_t, t)\right) + \sqrt{\sigma_t}\boldsymbol{\epsilon}$
- Return $\boldsymbol{x}_0$

1. Sample a Gaussian noise

$x_T \sim N(0, \mathrm{I})$

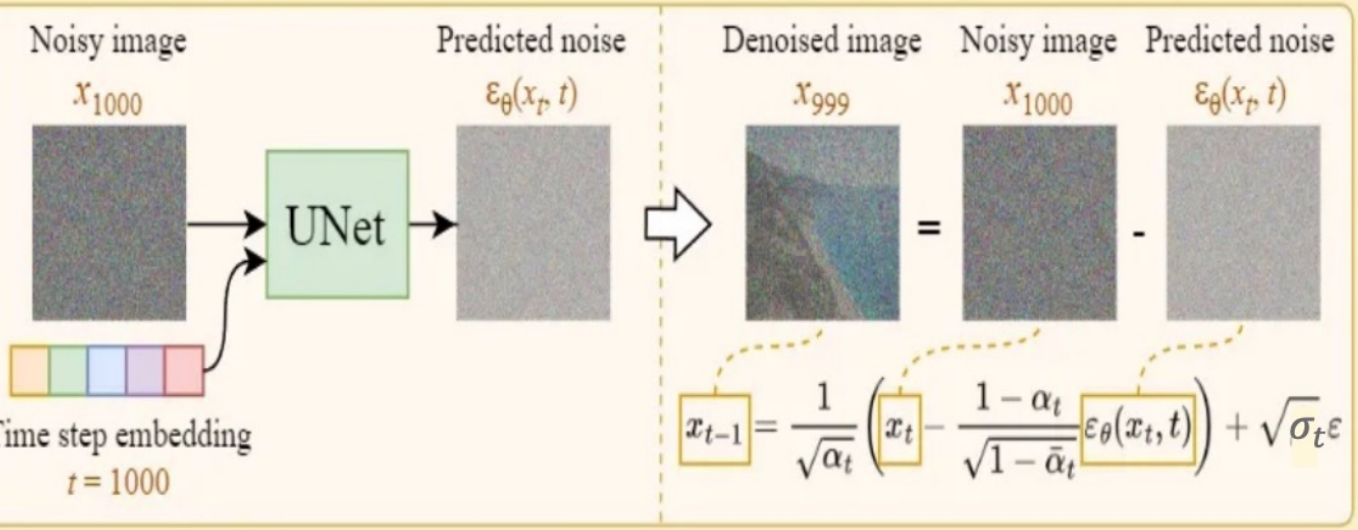
E.g. $T = 1000$

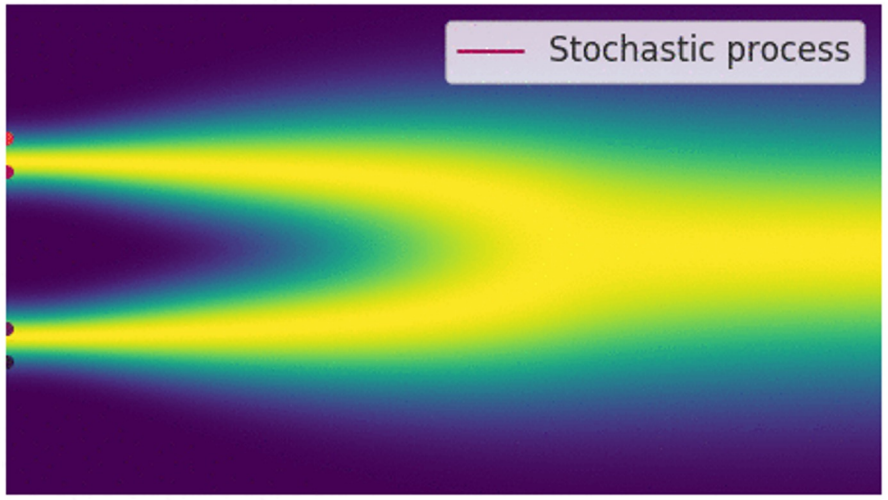
$x_{1000} \sim N(0, \mathrm{I})$

3. Output the denoised image

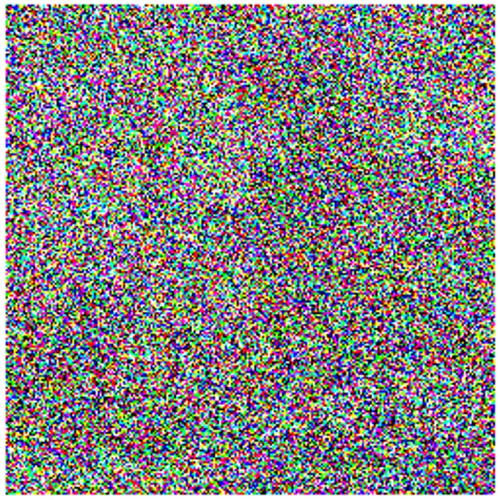
Denoised image x_0

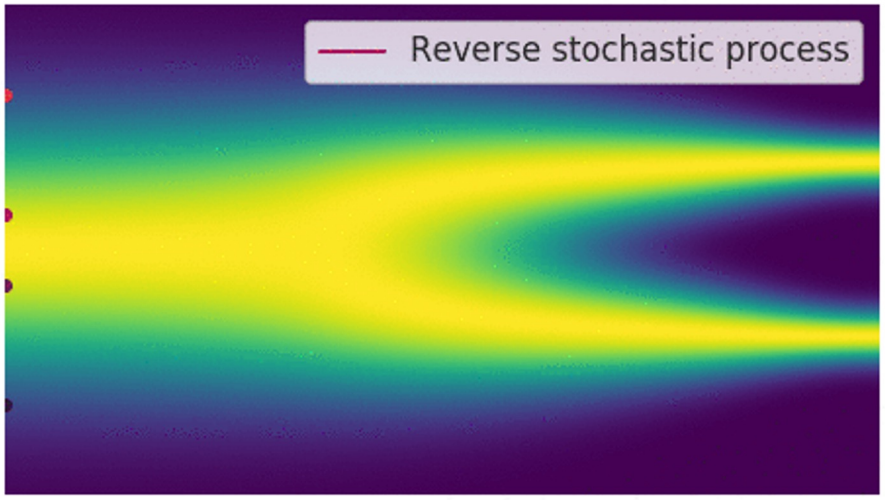
2. Iteratively denoise the image





Forward process: converting the image distribution to pure noise



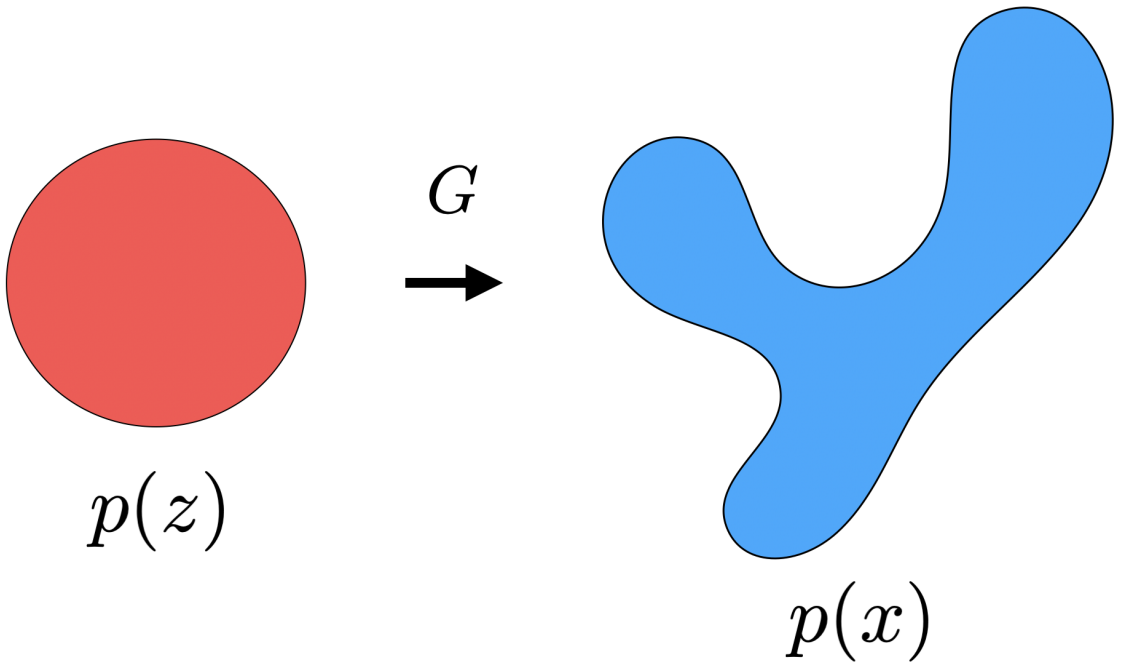


Reverse process: sampling from the image distribution, starting with pure noise

Sampling from Noise

Source distribution

Target distribution



Score-based Models

- Identical in practice with Diffusion model
- But, coming from a different theoretical angle

Generative Modeling by Estimating Gradients of the Data Distribution

Yang Song
Stanford University
yangsong@cs.stanford.edu

Stefano Ermon
Stanford University
ermon@cs.stanford.edu

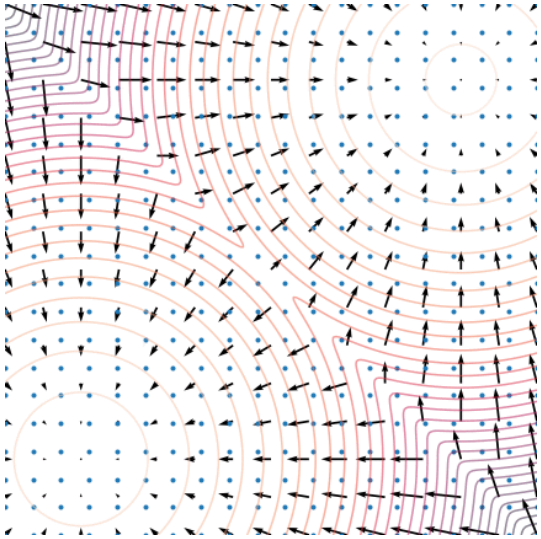
Score-based Models

- Data $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$ from distribution p(x)
- We can define pdf $p_\theta(\mathbf{x}) = \frac{e^{-f_\theta(\mathbf{x})}}{Z_\theta} \quad \int p_\theta(\mathbf{x}) \mathrm{d}\mathbf{x} = 1$
- Here $Z_\theta > 0$ is a normalizing constant, and $f_\theta(\mathbf{x})$ is energy-based model

Material from https://yang-song.net/blog/2021/score/

Score-based Models

- $Z_\theta > 0$ is mostly impossible to compute
- Thus we can model the gradient of pdf instead: $\nabla_{\mathbf{x}} \log p(\mathbf{x})$
- $\mathbf{s}_\theta(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_\theta(\mathbf{x}) = -\nabla_{\mathbf{x}} f_\theta(\mathbf{x}) - \underbrace{\nabla_{\mathbf{x}} \log Z_\theta}_{=0} = -\nabla_{\mathbf{x}} f_\theta(\mathbf{x})$



Diffusion Models Beats GANs

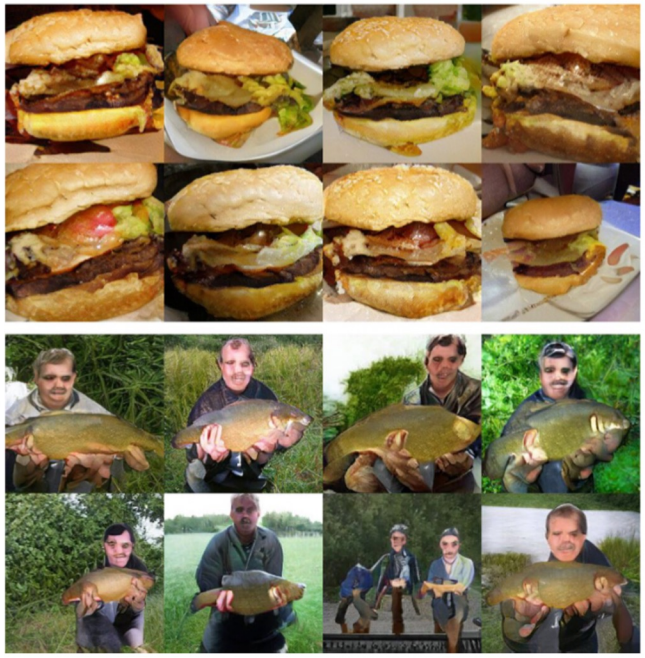
BigGAN

Diffusion

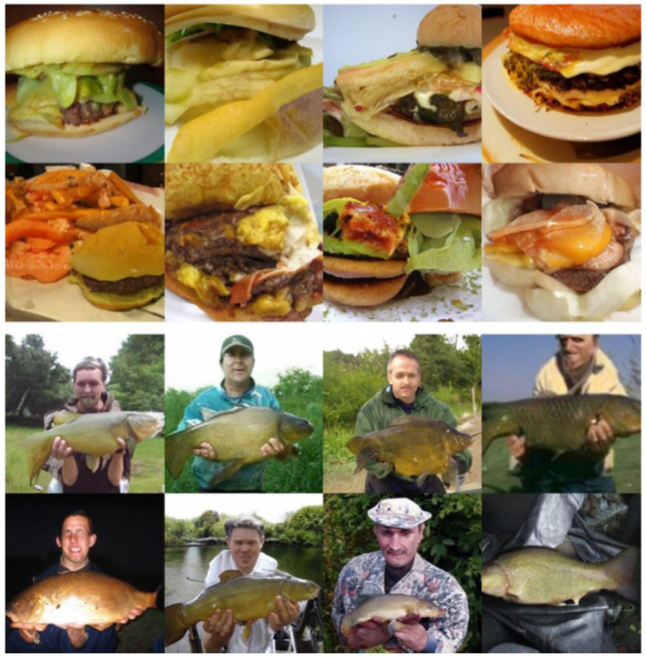
Training Set

Diffusion Models Beats GANs

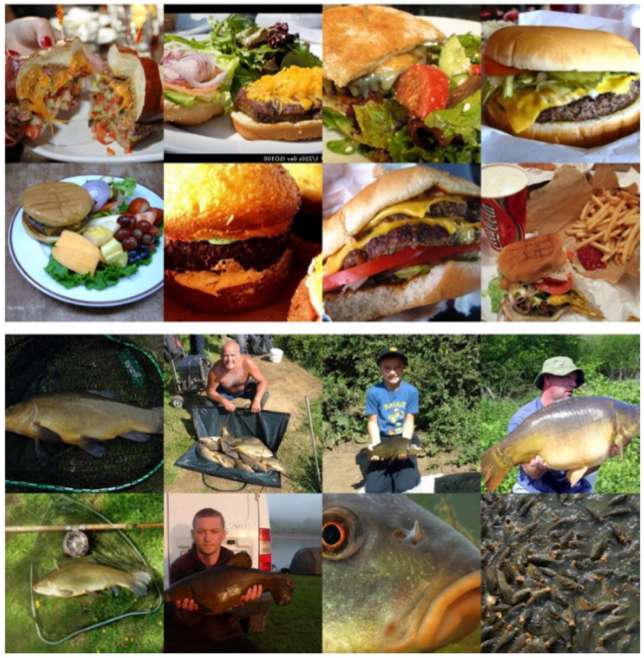
BigGAN



Diffusion

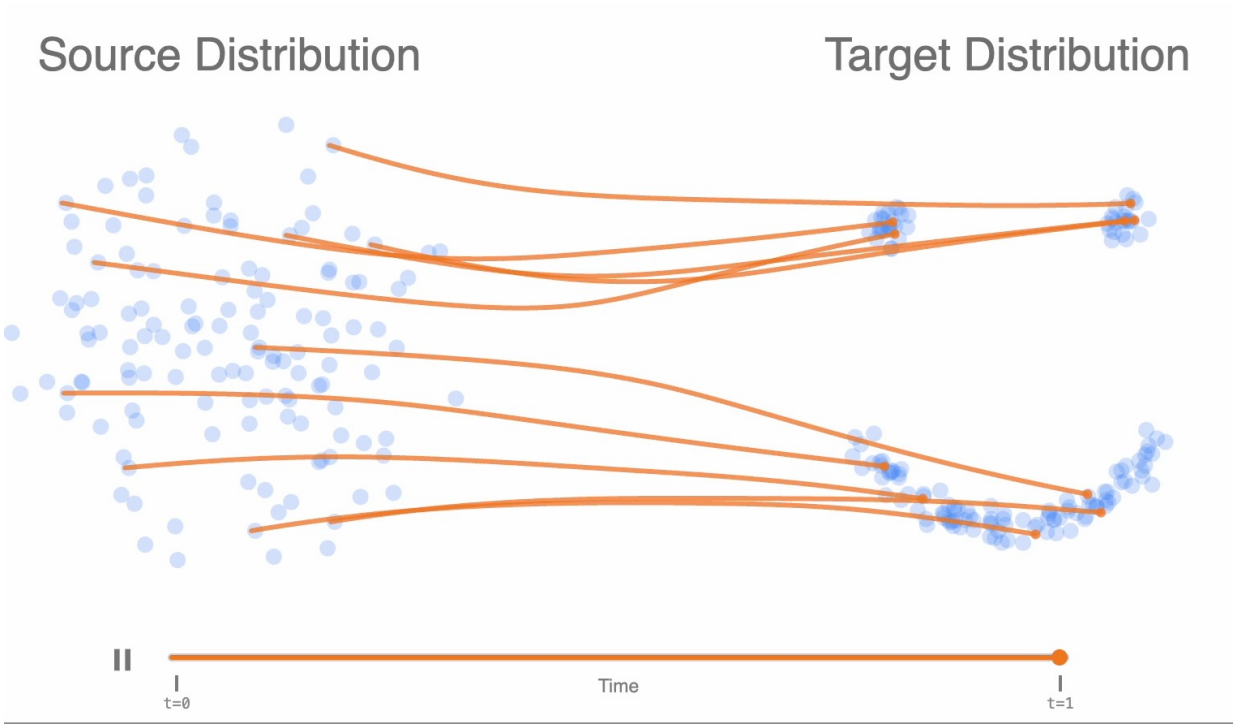


Training Set



Flow Matching

- Generative modeling: Source to Target distribution
- X_0 to X_1
- Lipman et al. 2022 (Flow Matching). Liu et al. 2022. (Rectified Flow)
- Blog post.
- Generalization of Diffusion Model
- Usually performs better. Newer models use FM.



Flow and Velocity Fields

- $\psi_t(x)$ is the trajectory that goes from x_0 to x_1
- $t \in [0,1]$
- We want to learn the velocity field at time t and x

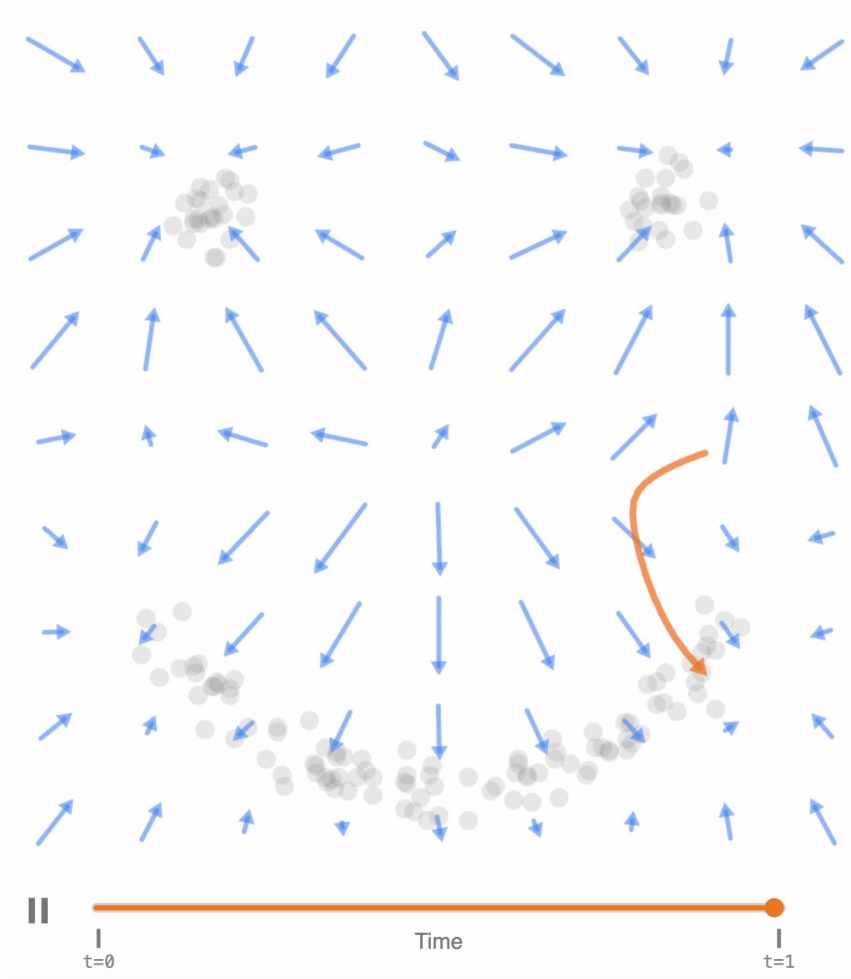
$$\frac{d}{dt}\psi_t(x) = v_t(x), \quad \psi_0(x) = x.$$

- Once we know $v_t(x)$, we integrate numerically (e.g., Euler) to get $\psi_t(x)$

$$x_{t+\Delta t} = x_t + \Delta t \cdot v_t(x_t)$$

Flow and Velocity Fields

- Velocity inference and integration



$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$$

Diffusion Path

Flow Matching

- Flow matching can be broken down into two key steps:
- We need to define our probability path $p_t(x)$ for interpolating between our source p and target distribution q.
- We need to train a velocity field $v_t^\theta(x)$ that generates the path p_t through regression.

- For Flow Matching, they introduced a simpler **Linear Path**
  - $X_t = (1-t)X_1 + tX_0 \sim p_t(x)$

Now we can train with a flow matching loss:

$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t, X_t \sim p_t} ||v_t(X_t) - v_t^\theta(X_t)||^2$$

Flow Matching

$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t, X_t \sim p_t} ||v_t(X_t) - v_t^\theta(X_t)||^2$$

- However, $v_t^\theta(x)$ not tractible and impossible to compute.
- Hence the authors use a conditional version:

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t, X_0, X_1} ||v_t(X_t|X_1) - v_t^\theta(X_t)||^2$$

$$\psi_t = (1-t)x_1 + tx_0$$

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t, X_0, X_1} ||(X_1 - X_0) - v_t^\theta(X_t)||^2$$

Flow Matching

$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t, X_t \sim p_t} ||v_t(X_t) - v_t^\theta(X_t)||^2$$

- However, $v_t^\theta(x)$ not tractible and impossible to compute.
- Hence the authors use a conditional version:

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t, X_0, X_1} ||v_t(X_t|X_1) - v_t^\theta(X_t)||^2$$

$$\psi_t = (1-t)x_1 + tx_0$$

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t, X_0, X_1} ||(X_1 - X_0) - v_t^\theta(X_t)||^2$$

- Authors proved that $\nabla_\theta \mathcal{L}_{CFM}(\theta) = \nabla_\theta \mathcal{L}_{FM}(\theta)$

Flow Matching

- So in practice, train with randomly sampled x_0, x_1, x_t:

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t,X_0,X_1}||(X_1 - X_0) - v_t^\theta(X_t)||^2$$

- Inference with numerical integration (eg. Euler)

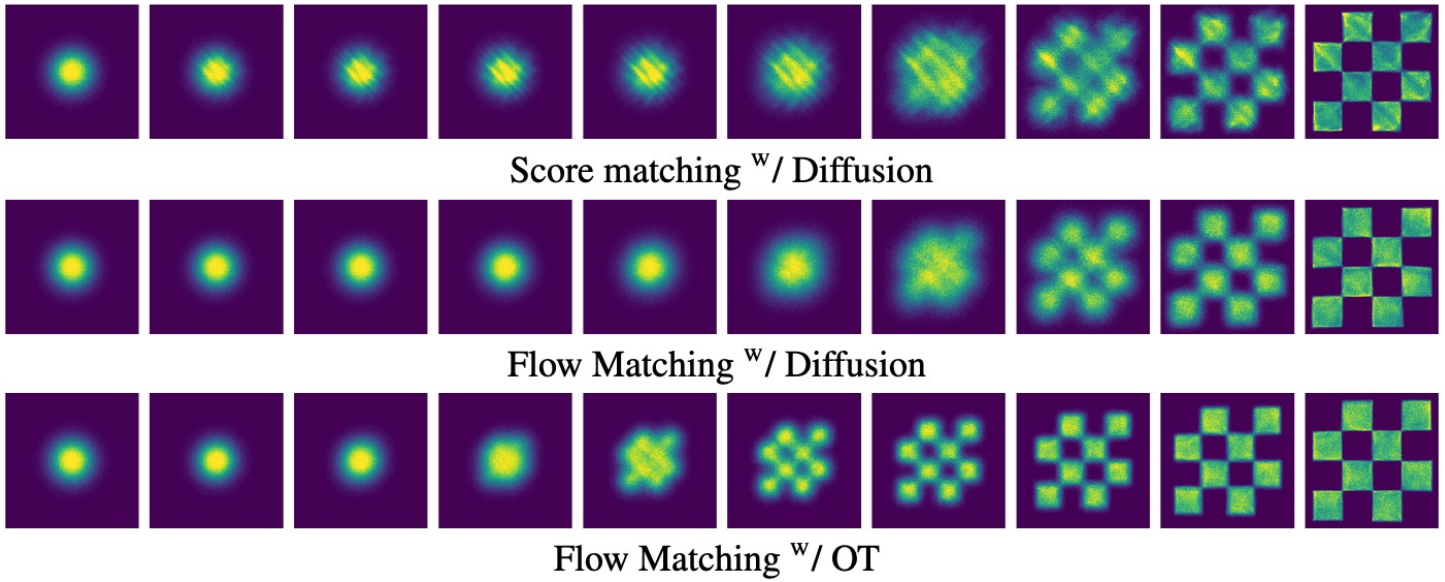
$$x_{t+\Delta t} = x_t + \Delta t \cdot v_t(x_t)$$

Flow Matching

- So in practice, train with randomly sampled x_0, x_1, x_t:

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t,X_0,X_1}||(X_1 - X_0) - v_t^\theta(X_t)||^2$$

- Inference with numerical integration (eg. Euler)



Today

- Generative Models
  - Autoregressive Models
  - VAE
  - GANs
  - Diffusion Models
  - Flow-based Models

Next Week

- Monday is Labor day: have fun!
- Wednesday we will start from ConvNets and Recognition papers.
- Please sign up for papers
- Uploaded project examples. Find topics and teammates!

Discussion

1. The outputs of VAEs are known to be blurry. Why would this be?
2. For diffusion models, what could be the advantage modeling the noise $\boldsymbol{\epsilon}_t$ instead of the less noisy image $\boldsymbol{x}_{t-1}$ or the clean image $\boldsymbol{x}_0$
3. Why is diffusion training usually more stable than GAN training?
4. Can you come up with regularization techniques to make GAN training more stable?
5. What's the latent space of DM/FM? Why do you think they outperform GANs or VAEs?
6. Can we train DM/FM from partial data? E.g., a data set of random patches instead of full images.