Lecture 14:
Generative Models (part 2)

Administrative

- Assignment 3 due on 5/30
- Project Report due on 6/4

Last Time: Generative vs Discriminative 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

Density Function

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

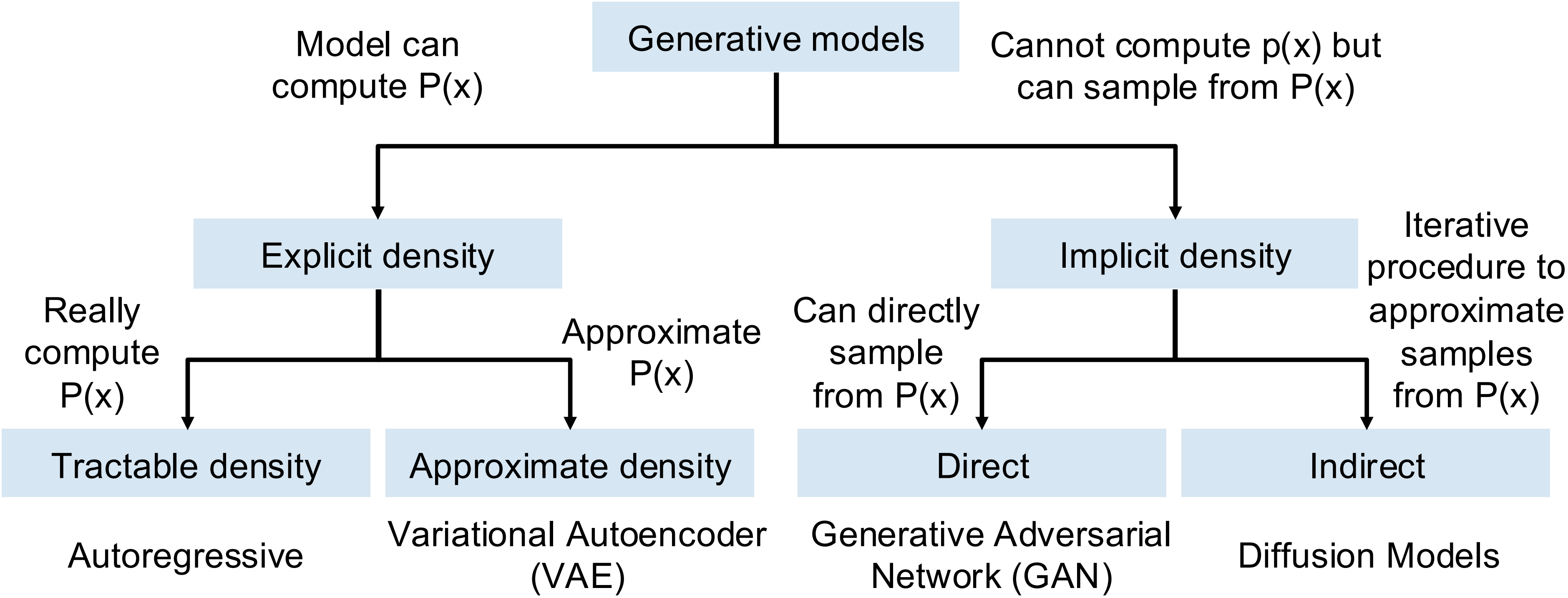
Density functions are **normalized:**

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

Different values of x **compete** for density

Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017

Last Time: Generative Models



Generative models

Model can compute P(x)

Cannot compute p(x) but can sample from P(x)

Explicit density

Implicit density

Really compute P(x)

Approximate P(x)

Can directly sample from P(x)

Iterative procedure to approximate samples from P(x)

Tractable density

Approximate density

Direct

Indirect

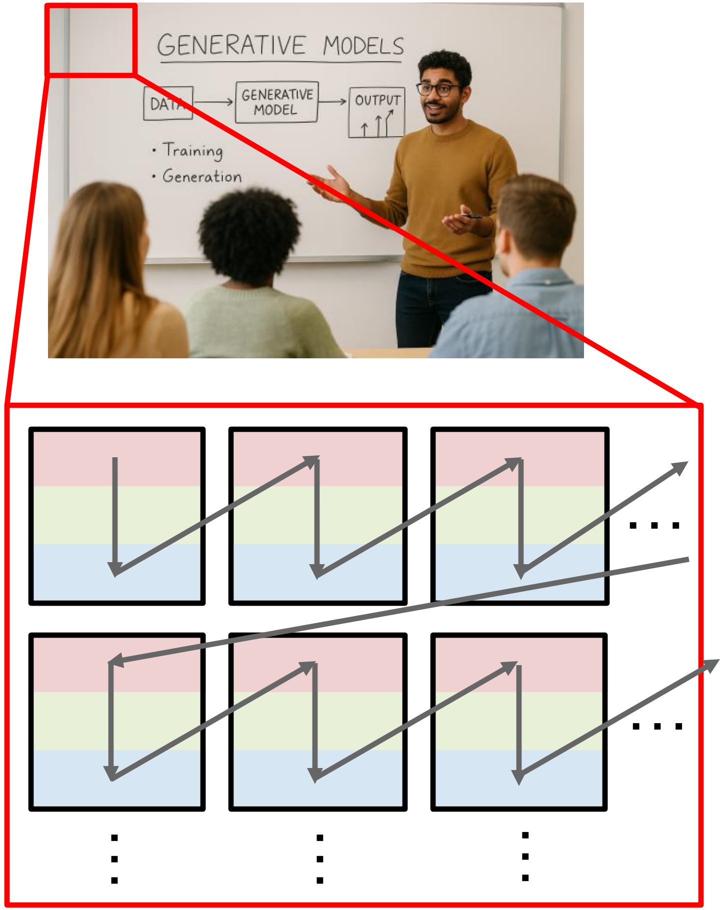
Autoregressive

Variational Autoencoder (VAE)

Generative Adversarial Network (GAN)

Diffusion Models

Last Time: Autoregressive Models



Treat data as a sequence
(e.g. image as sequence of pixels)

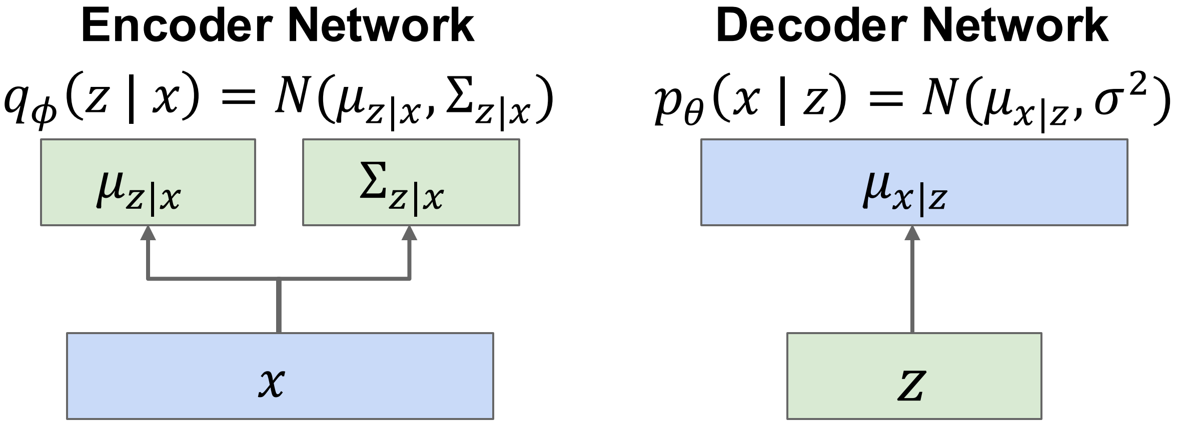
$$
\begin{aligned}
p(x) &= p(x_1, x_2, \ldots, x_N) \\
&= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \ldots \\
&= \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})
\end{aligned}
$$

Model with an RNN or Transformer

Last Time: Variational Autoencoders

Jointly train **encoder** q and **decoder** p to maximize the **variational lower bound** on the data likelihood
Also called **Evidence Lower Bound** (**ELBo**)

$$\log p_\theta(x) \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$



Encoder Network

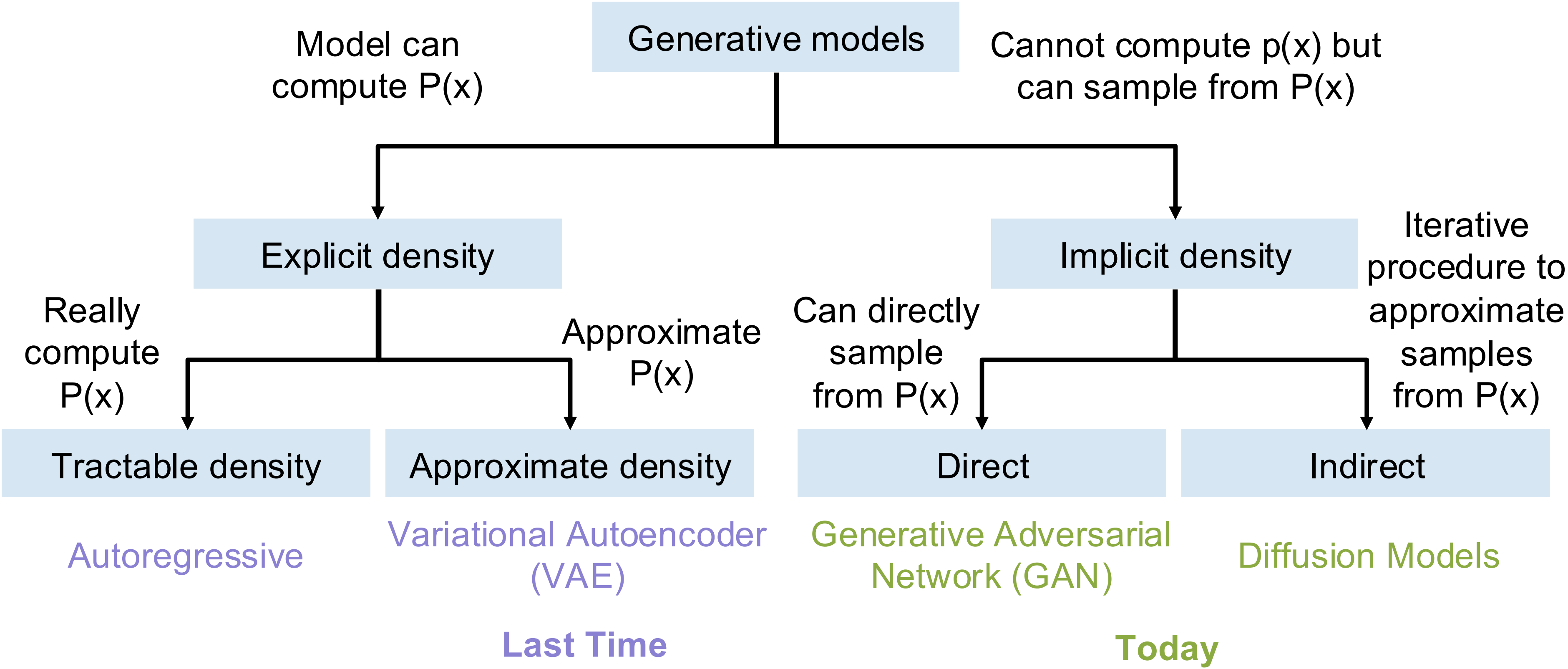
$q_\phi(z \mid x) = N(\mu_{z|x}, \Sigma_{z|x})$

Decoder Network

$p_\theta(x \mid z) = N(\mu_{x|z}, \sigma^2)$

Today: More Generative Models

Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017



Generative models

Model can compute P(x)

Cannot compute p(x) but can sample from P(x)

Explicit density

Implicit density

Really compute P(x)

Approximate P(x)

Can directly sample from P(x)

Iterative procedure to approximate samples from P(x)

Tractable density

Approximate density

Direct

Indirect

Autoregressive

Variational Autoencoder (VAE)

Generative Adversarial Network (GAN)

Diffusion Models

Last Time

Today

Generative Adversarial Networks (GANs)

Generative Models So Far

Autoregressive Models directly maximize likelihood of training data:

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

Variational Autoencoders introduce a latent z, and maximize a lower bound:

$$p_\theta(x) = \int_Z p_\theta(x|z)p(z)dz \geq E_{Z\sim q_\phi(Z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

Generative Models So Far

Autoregressive Models directly maximize likelihood of training data:

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

Variational Autoencoders introduce a latent z, and maximize a lower bound:

$$p_\theta(x) = \int_Z p_\theta(x|z)p(z)dz \geq E_{Z \sim q_\phi(Z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

Generative Adversarial Networks give up on modeling p(x), but allow us to draw samples from p(x)

Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{data}(x)$. Want to sample from p_{data}

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{data}(x)$. Want to sample from p_{data}

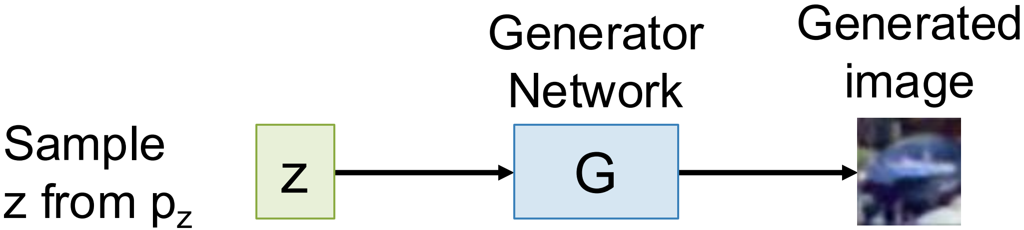
Idea: Introduce a latent variable z with simple prior p(z) (e.g. unit Gaussian)
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** p_G. Want $p_G = p_{data}$!

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{data}(x)$. Want to sample from p_{data}

Idea: Introduce a latent variable z with simple prior p(z) (e.g. unit Gaussian)
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** p_G. Want $p_G = p_{data}$!



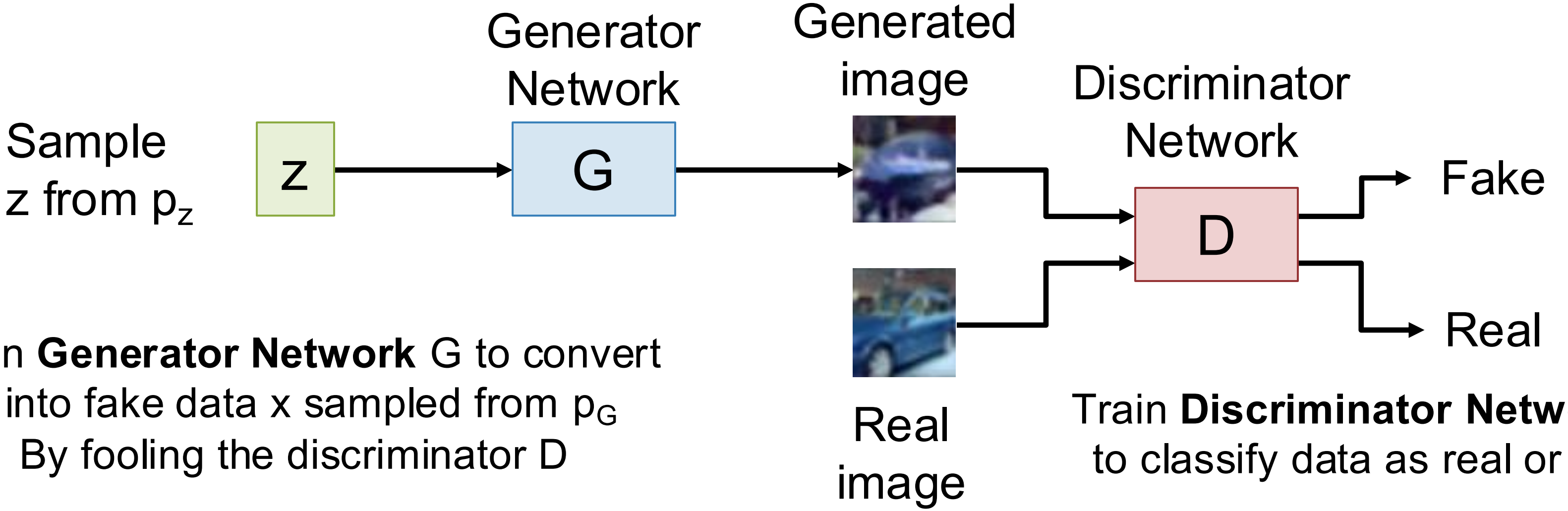
Train **Generator Network** G to convert z into fake data x sampled from p_G

Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{data}(x)$. Want to sample from p_{data}

Idea: Introduce a latent variable z with simple prior p(z) (e.g. unit Gaussian)
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** p_G. Want $p_G = p_{data}$!



Train **Generator Network** G to convert
z into fake data x sampled from p_G
By fooling the discriminator D

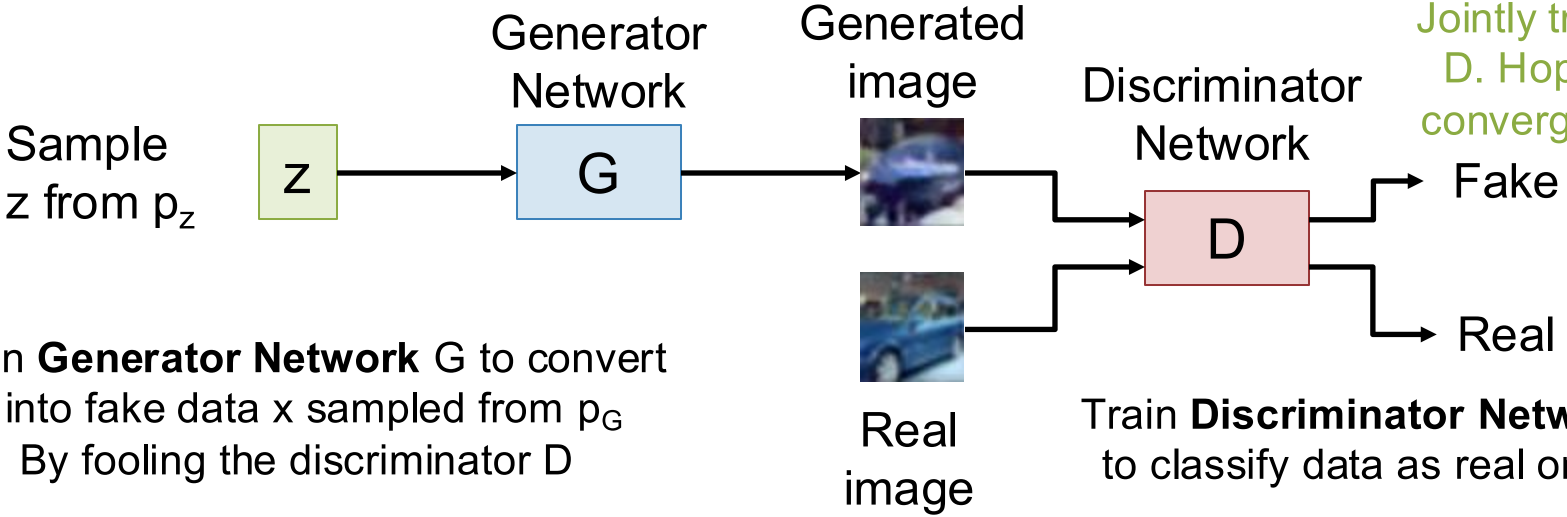
Train **Discriminator Network** D
to classify data as real or fake

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

Generative Adversarial Networks

Setup: Have data x_i drawn from distribution $p_{data}(x)$. Want to sample from p_{data}

Idea: Introduce a latent variable z with simple prior p(z) (e.g. unit Gaussian)
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** p_G. Want $p_G = p_{data}$!



Jointly train G and D. Hopefully p_G converges to p_{data}

Train **Generator Network** G to convert z into fake data x sampled from p_G By fooling the discriminator D

Train **Discriminator Network** D to classify data as real or fake

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

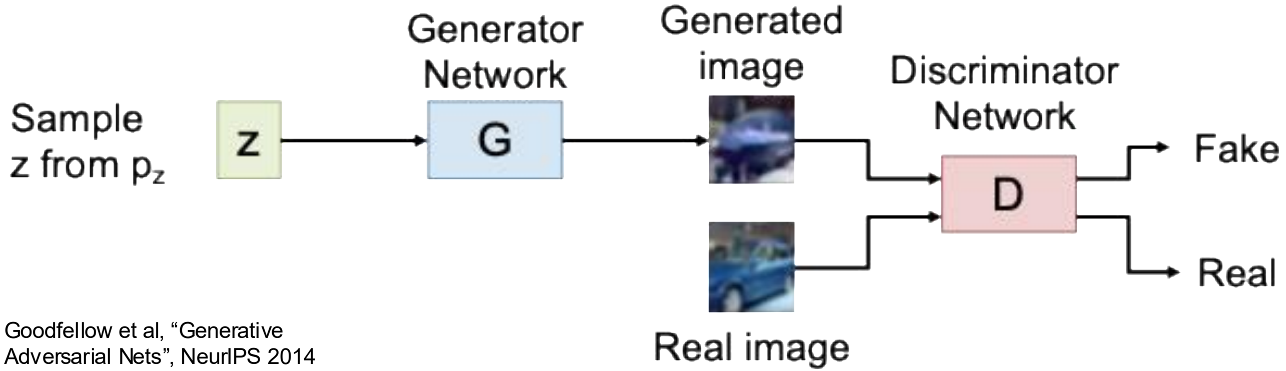
$$\min_{G}\max_{D}\left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]\right)$$

Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_{G} \max_{D} \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log\left(1 - D(G(z)) \right) \right] \right)$$



D(x) = P(x is real)
D(x) = 0 => fake
D(x) = 1 => real

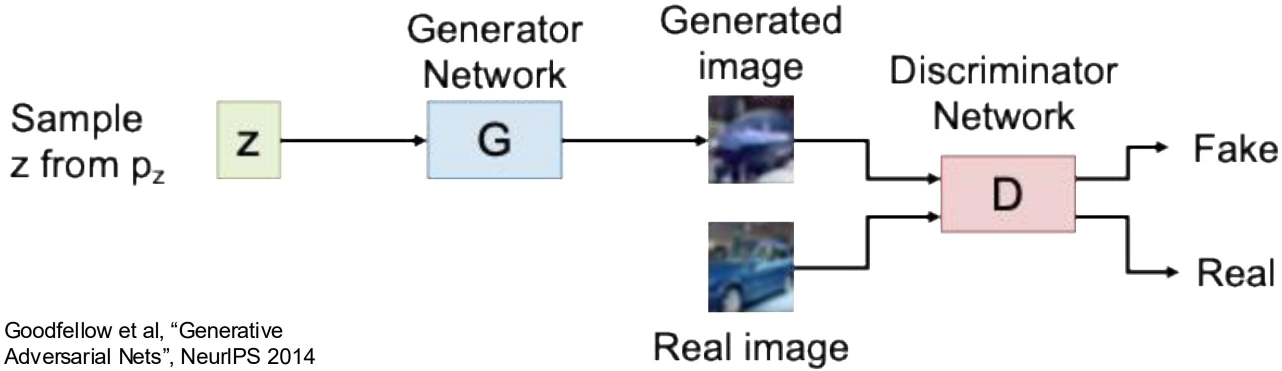
Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Imagine fixing **G**

$$\min_{G}\max_{D}\left(\overbrace{E_{x\sim p_{data}}[\log D(x)]}^{\text{Discriminator wants } D(x) = 1 \text{ for real data}} + E_{z\sim p(z)}\overbrace{\left[\log\left(1-D\left(G(z)\right)\right)\right]}^{\text{Discriminator wants } D(x) = 0 \text{ for fake data}}\right)$$



D(x) = P(x is real)
D(x) = 0 => fake
D(x) = 1 => real

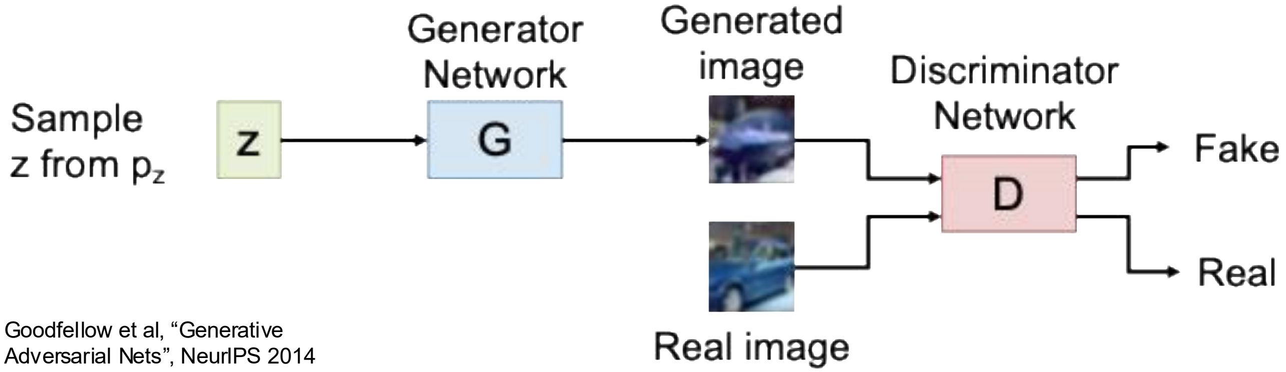
Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Imagine fixing **D**

$$\min_{G} \max_{D} \left(\underbrace{E_{x\sim p_{data}}[\log D(x)]}_{\text{This term does not depend on G}} + \underbrace{E_{z\sim p(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]}_{\text{Generator wants D(x) = 1 for fake data}} \right)$$



D(x) = P(x is real)
D(x) = 0 => fake
D(x) = 1 => real

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train **G** and **D** using alternating gradient updates

$$\min_{G} \max_{D} \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log \left(1 - D\left(G(z)\right) \right) \right] \right)$$

$$= \min_{G} \max_{D} V(G, D)$$

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train **G** and **D** using alternating gradient updates

$$\min_{G} \max_{D} \left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\left[\log\left(1 - D(G(z))\right)\right]\right)$$

$$= \min_{G} \max_{D} V(G, D)$$

While True:

$$D = D + \alpha_D \frac{dV}{dD}$$

$$G = G - \alpha_G \frac{dV}{dG}$$

Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train **G** and **D** using alternating gradient updates

$$\min_{G} \max_{D} \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log \left(1 - D(G(z)) \right) \right] \right)$$

$$= \min_{G} \max_{D} V(G, D)$$

While True:

$$D = D + \alpha_D \frac{dV}{dD}$$

$$G = G - \alpha_G \frac{dV}{dG}$$

We are not minimizing any overall loss! No training curves to look at!

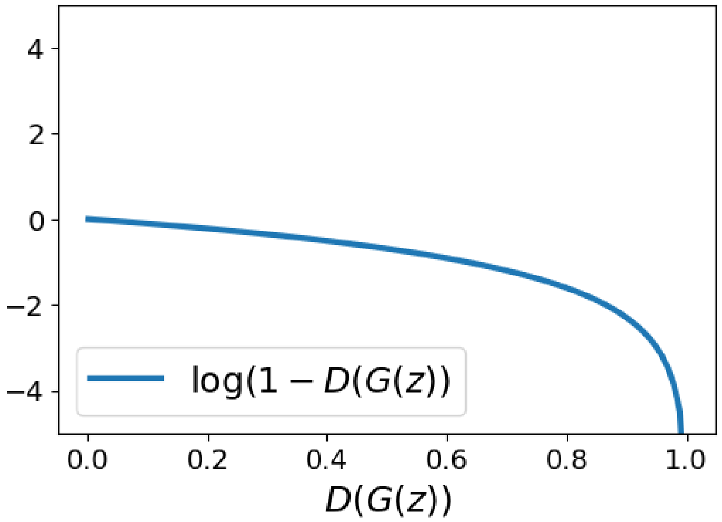
Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_{G} \max_{D} \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log \left(1 - D(G(z)) \right) \right] \right)$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so D(G(z)) close to 0



Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

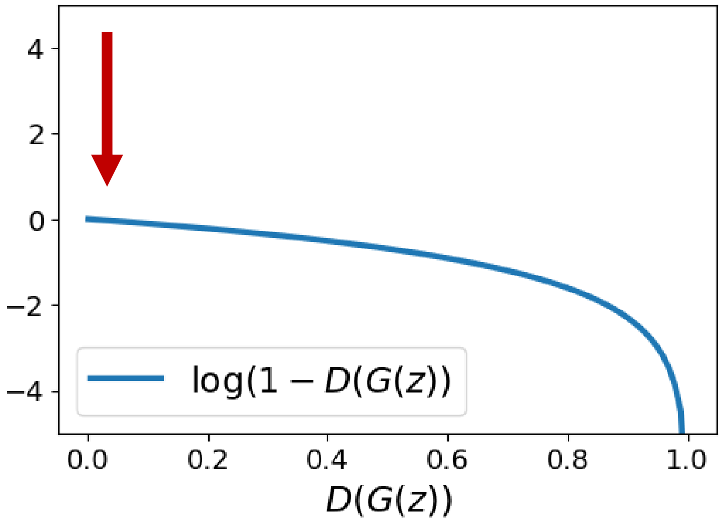
Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_{G}\max_{D}\left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\Big[\log\Big(1 - D\big(G(z)\big)\Big)\Big]\right)$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so D(G(z)) close to 0

Problem: Gradients for G are close to 0



Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

Generative Adversarial Networks: Training Objective

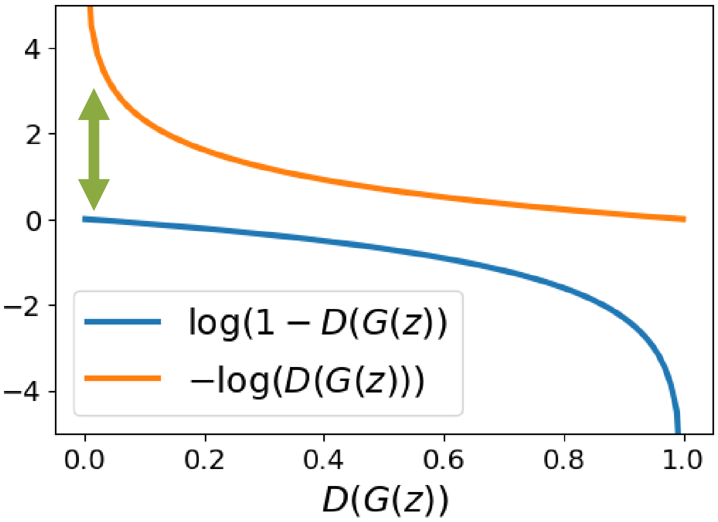
Jointly train generator G and discriminator D with a **minimax game**

$$\min_{G} \max_{D} \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log \left(1 - D(G(z)) \right) \right] \right)$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so D(G(z)) close to 0

Problem: Gradients for G are close to 0

Solution: Generator wants D(G(z)) = 1. Train generator to minimize –log(D(G(z)) and discriminator to maximize log(1-D(G(z)) so generator gets strong gradients at start



Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_{G} \max_{D} \left(E_{x \sim p_{data}}[\log D(x)] + E_{z \sim p(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]\right)$$

Why is this a good objective?

Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_{\boldsymbol{G}} \max_{\boldsymbol{D}} \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log \left(1 - D\left(G(z) \right) \right) \right] \right)$$

Why is this a good objective?

Inner objective is maximized by

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

(for any p_G)

(Proof omitted)

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_{G} \max_{D} \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log \left(1 - D(G(z)) \right) \right] \right)$$

Why is this a good objective?

Inner objective is maximized by

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

(for any p_G)

Outer objective is then minimized by $p_G(x) = p_{data}(x)$

(Proof omitted)

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_{G}\max_{D}\left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\Big[\log\big(1 - D(G(z))\big)\Big]\right)$$

Why is this a good objective?

Inner objective is maximized by

$$D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

(for any p_G)

(Proof omitted)

Outer objective is then minimized by

$$p_G(x) = p_{data}(x)$$

Caveats:

1. Neural nets with fixed capacity may not be able to represent optimal D and G
2. This tells us nothing about convergence to the solution with finite data

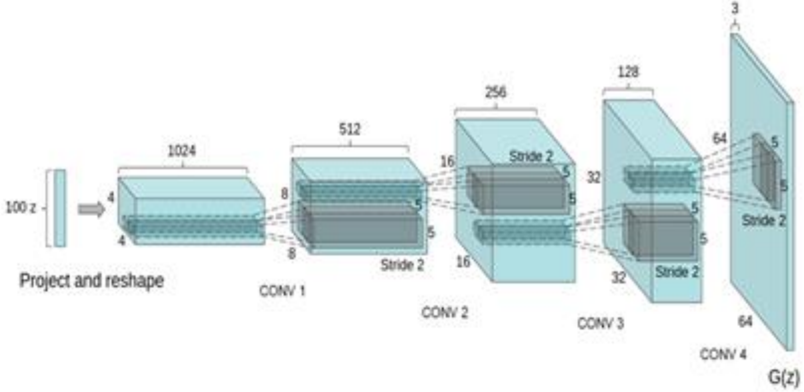
Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

GAN Architectures: DC-GAN

Generator G and discriminator D are both neural networks

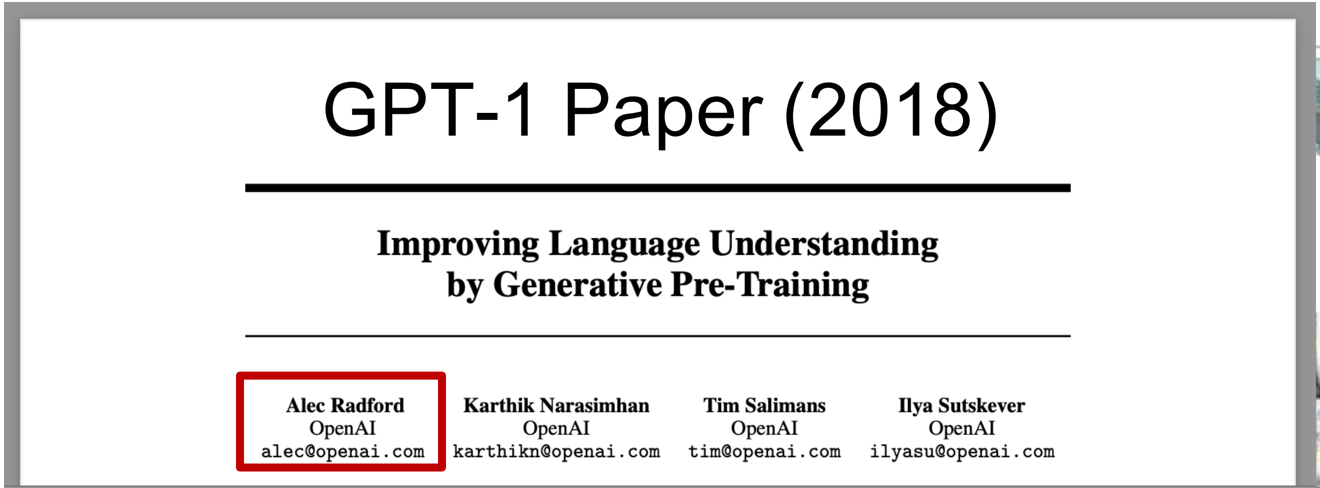
Usually CNNs … GANs fell out of favor before ViT became popular

DC-GAN was the first GAN architecture that worked on non-toy data



Radford et al, ICLR 2016

GAN Architectures: DC-GAN



GPT-1 Paper (2018)

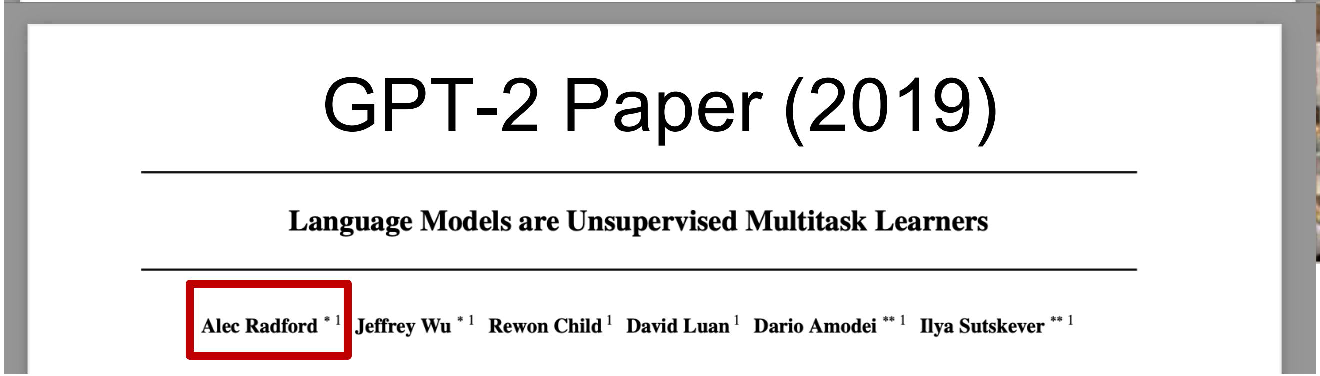
Improving Language Understanding by Generative Pre-Training

Alec Radford
OpenAI
alec@openai.com

Karthik Narasimhan
OpenAI
karthikn@openai.com

Tim Salimans
OpenAI
tim@openai.com

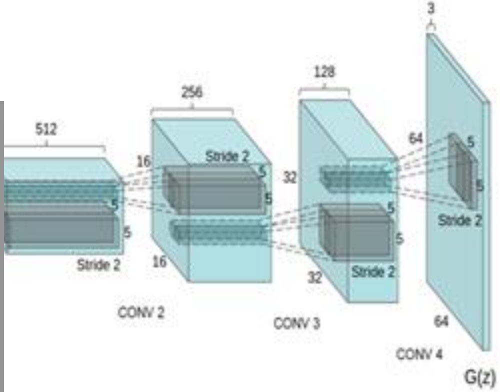
Ilya Sutskever
OpenAI
ilyasu@openai.com



GPT-2 Paper (2019)

Language Models are Unsupervised Multitask Learners

Alec Radford [*1] **Jeffrey Wu** [*1] **Rewon Child** [1] **David Luan** [1] **Dario Amodei** [**1] **Ilya Sutskever** [**1]



Radford et al, ICLR 2016

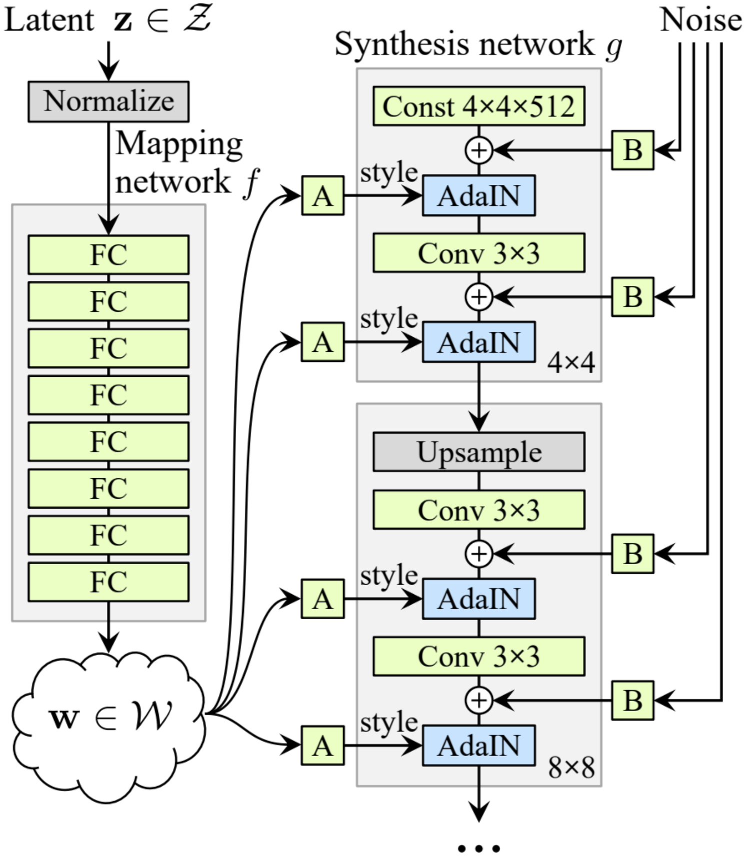
GAN Architectures: StyleGAN

Generator G and discriminator D are both neural networks

StyleGAN uses a more complex architecture that injects noise via **adaptive normalization**.

At each layer predict a scale w and shift b the same shape as x:

$$AdaIN(x, w, b)_i = w_i \frac{x_i - \mu(x)}{\sigma(x)} + b_i$$



Karras et al, "A Stye-Based Generator Architecture for Generative Adversarial Networks", CVPR 2019

GANs: Latent Space Interpolation

Latent space is **smooth**.

Given latent vectors z_0 and z_1, we can **interpolate** between them:

$$z_t = tz_o + (1-t)z_1$$
$$x_t = G(z_t)$$

The resulting image x_t smoothly interpolate between samples!

GANs: Latent Space Interpolation

Latent space is **smooth**.

Given latent vectors z_0 and z_1, we can **interpolate** between them:

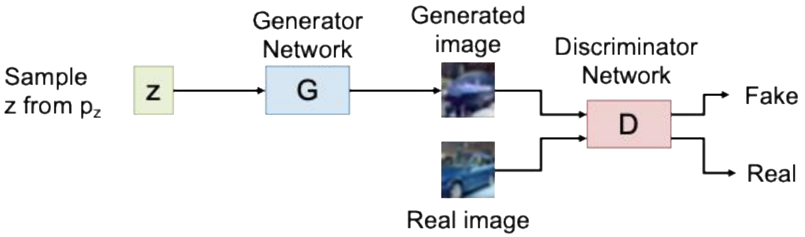
$$z_t = tz_o + (1 - t)z_1$$
$$x_t = G(z_t)$$

The resulting image x_t smoothly interpolate between samples!

Karras et al, ”Alias-Free Generative Adversarial Networks”, NeurIPS 2021

Generative Adversarial Networks: Summary

Jointly train Generator and Discriminator with a minimax game



Pros:
- Simple formulation
- Very good image quality

Cons:
- No loss curve to look at
- Unstable training
- Hard to scale to big models + data

These were the go-to generative models from ~2016 – 2021

Diffusion Models

Sohl-Dickstein et al, “Deep Unsupervised Learning using noneuilibrium thermodynamics”, ICML 2015
Song and Ermon, “Generative modeling by estimnating gradients of the data distribution”, NaurIPS 2019
Ho et al, “Denoising Diffusion Probabalistic Models”, NeurIPS 2020
Song et al, “Score-Based Generative Modeling through Stochastic Differential Equations”, ICLR 2021
Song et al, “Denoising Diffusion Implicit Models”, ICLR 2021

Diffusion Models

Warning: Terminology and notation in this area is a mess!

There are many different mathematical formalisms; tons of variance in terminology and notation between papers.

We'll just cover the basics of a modern "clean" implementation (Rectified Flow)

Diffusion Models: Intuition

Pick a **noise distribution** $z \sim p_{noise}$
(Usually unit Gaussian)

Diffusion Models: Intuition

Pick a **noise distribution** $z \sim p_{noise}$
(Usually unit Gaussian)

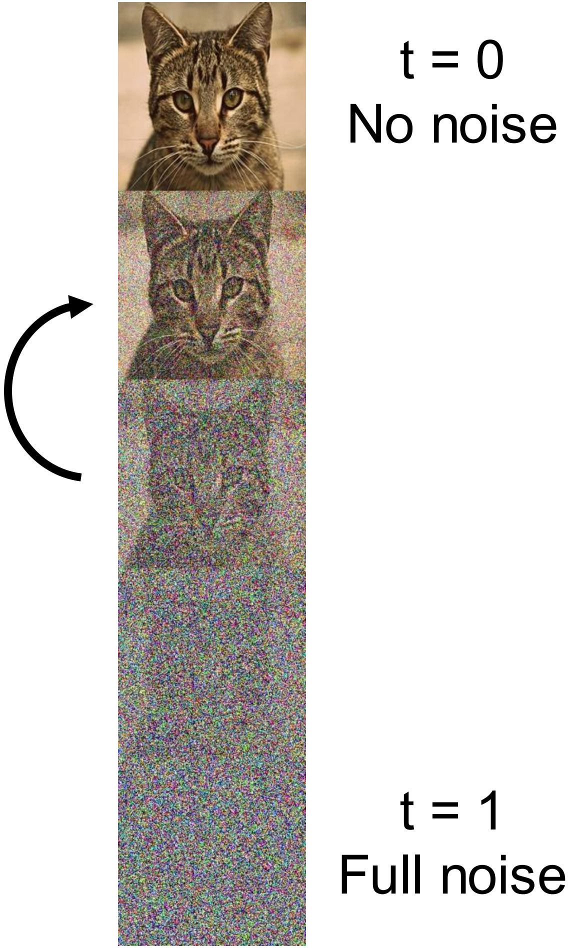
Consider data x corrupted under varying **noise levels** t to give noisy data x_t

Diffusion Models: Intuition

Pick a **noise distribution** $z \sim p_{noise}$ (Usually unit Gaussian)

Consider data x corrupted under varying **noise levels** t to give noisy data x_t

Train a neural network to **remove a little bit of noise**: $f_\theta(x_t, t)$



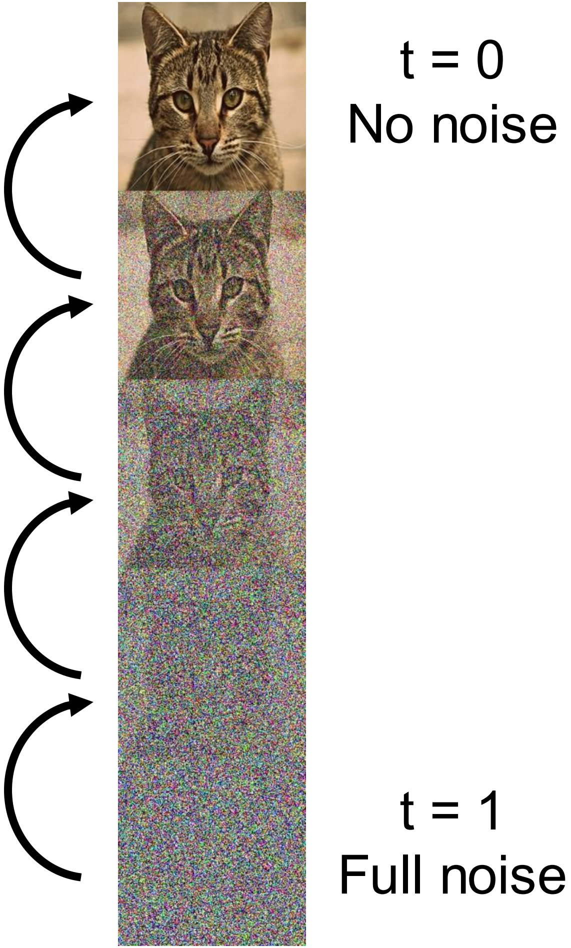
Diffusion Models: Intuition

Pick a **noise distribution** $z \sim p_{noise}$ (Usually unit Gaussian)

Consider data x corrupted under varying **noise levels** t to give noisy data x_t

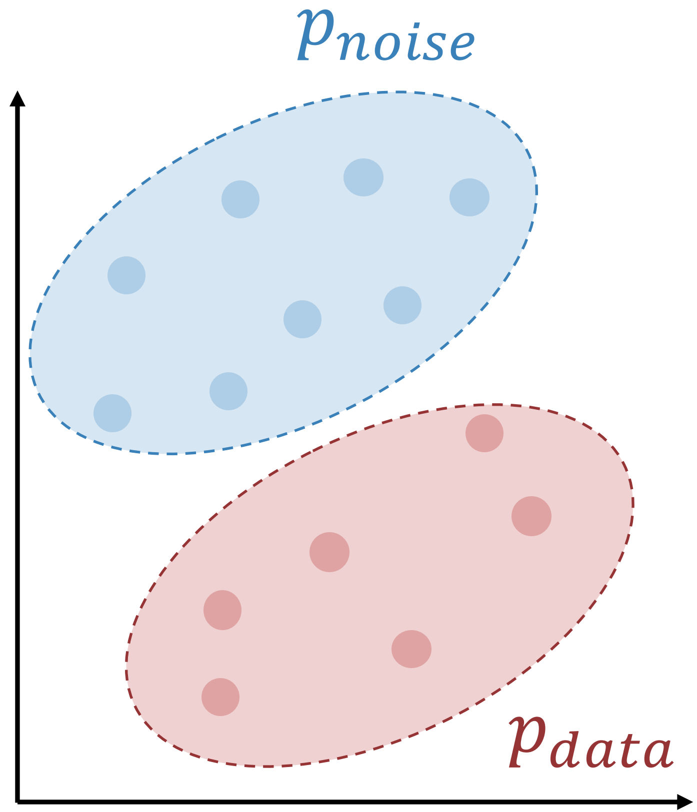
Train a neural network to **remove a little bit of noise**: $f_\theta(x_t, t)$

At inference time, sample $x_1 \sim p_{noise}$ and apply f_θ many times in sequence to generate a noiseless sample x_0



Diffusion Models: Rectified Flow

Suppose we have a simple $\mathbf{p}_{\text{noise}}$ (e.g. Gaussian) and samples from $\mathbf{p}_{\text{data}}$



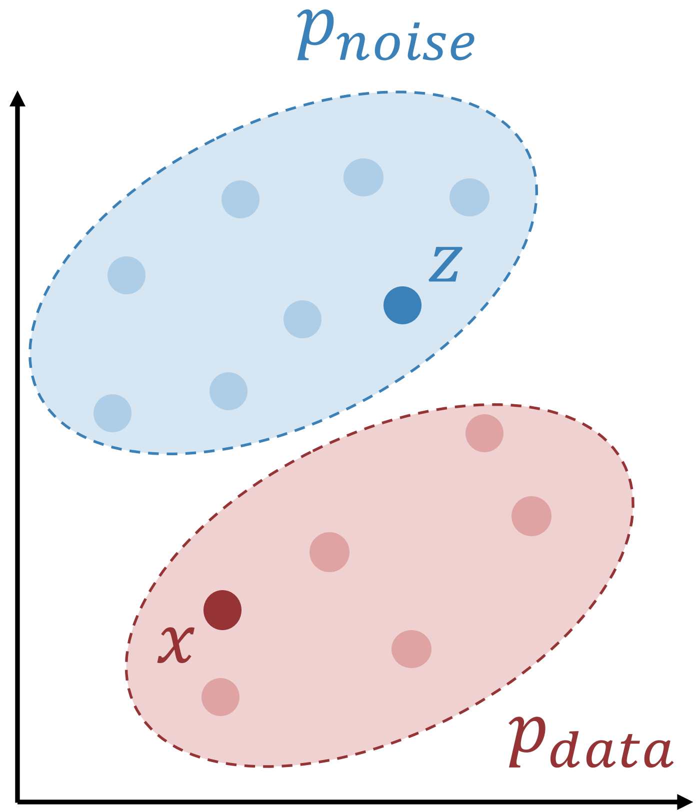
Liu et al, “Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow”, 2022
Lipman et al, “Flow Matching for Generative Modeling”, 2022

Diffusion Models: Rectified Flow

Suppose we have a simple $\mathbf{p}_{\text{noise}}$ (e.g. Gaussian) and samples from $\mathbf{p}_{\text{data}}$

On each training iteration, sample:

$z \sim p_{noise}$ $\quad x \sim p_{data}$ $\quad t \sim Uniform[0,1]$



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

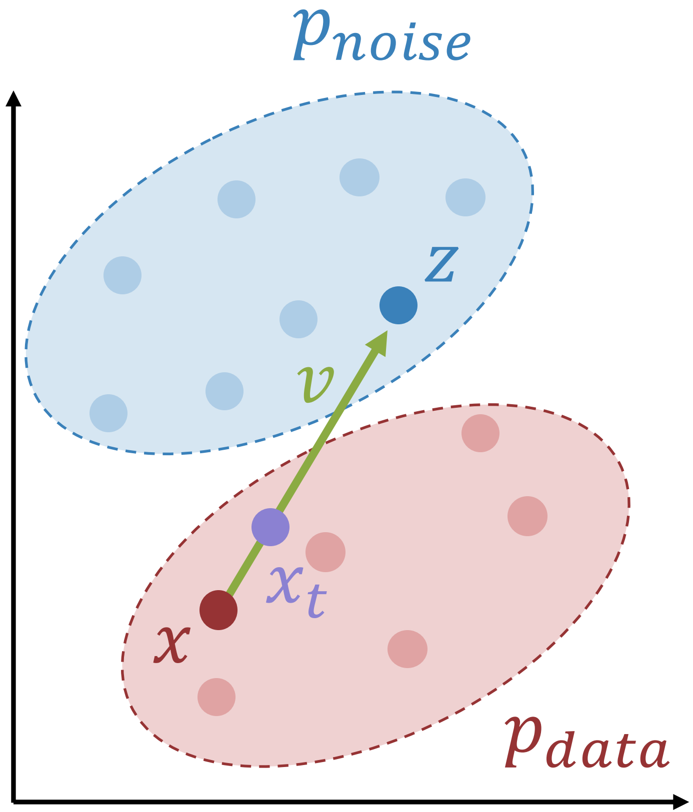
Diffusion Models: Rectified Flow

Suppose we have a simple **p_{noise}** (e.g. Gaussian) and samples from **p_{data}**

On each training iteration, sample:

$z \sim p_{noise}$ $\quad x \sim p_{data}$ $\quad t \sim Uniform[0, 1]$

Set $x_t = (1-t)x + tz$, $v = z - x$



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Diffusion Models: Rectified Flow

Suppose we have a simple $\mathbf{p}_{\mathbf{noise}}$ (e.g. Gaussian) and samples from $\mathbf{p}_{\mathbf{data}}$

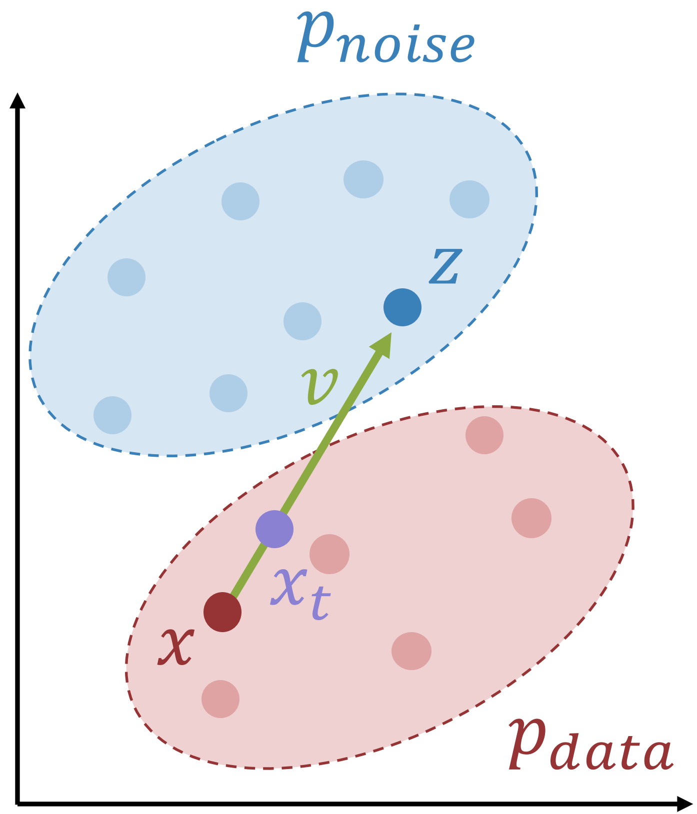
On each training iteration, sample:

$z \sim p_{noise} \quad x \sim p_{data} \quad t \sim Uniform[0,1]$

Set $x_t = (1-t)x + tz$, $v = z - x$

Train a neural network to predict v:

$$L = \|f_\theta(x_t, t) - v\|_2^2$$

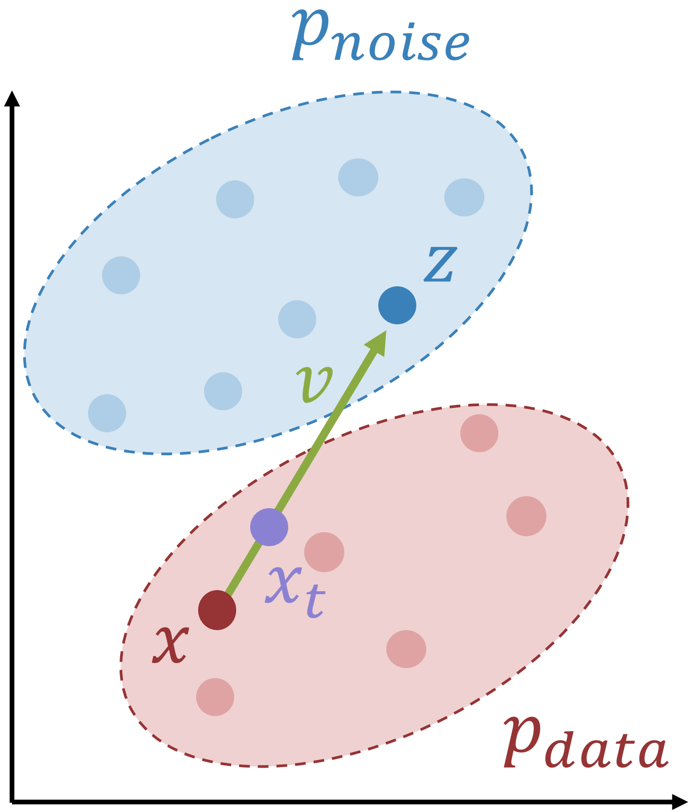


Liu et al, “Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow”, 2022
Lipman et al, “Flow Matching for Generative Modeling”, 2022

Diffusion Models: Rectified Flow

Core training loop is just a few lines of code!

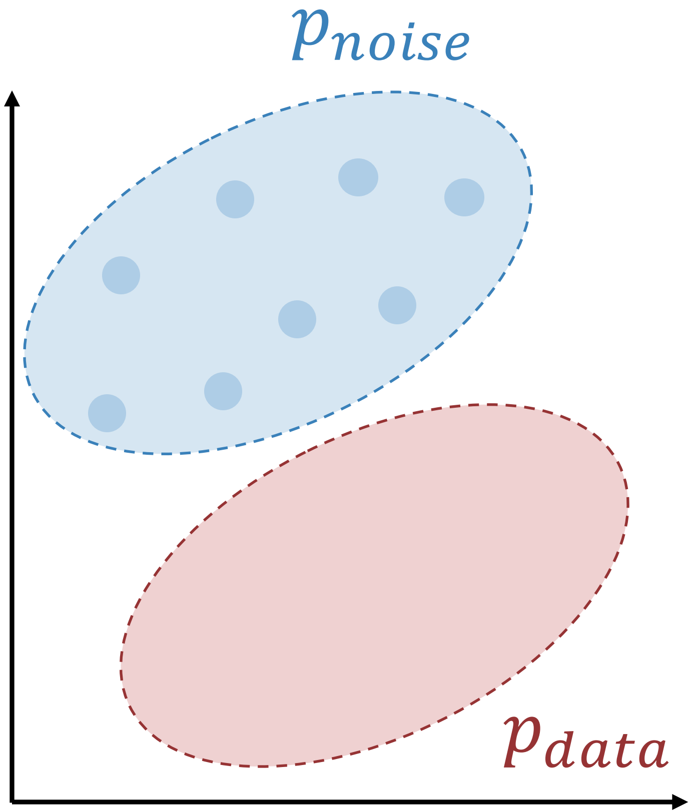
```
for x in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    v = model(xt, t)
    loss = (z - x - v).square().sum()
```



Liu et al, “Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow”, 2022
Lipman et al, “Flow Matching for Generative Modeling”, 2022

Rectified Flow: Sampling

Choose number of steps T (often T=50)

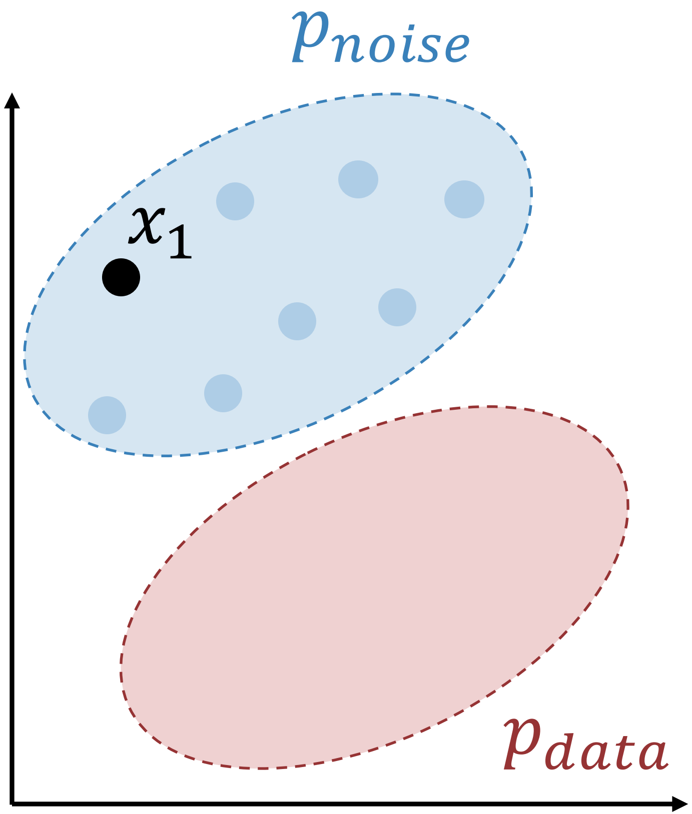


Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Rectified Flow: Sampling

Choose number of steps T (often T=50)

Sample $x \sim p_{noise}$



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

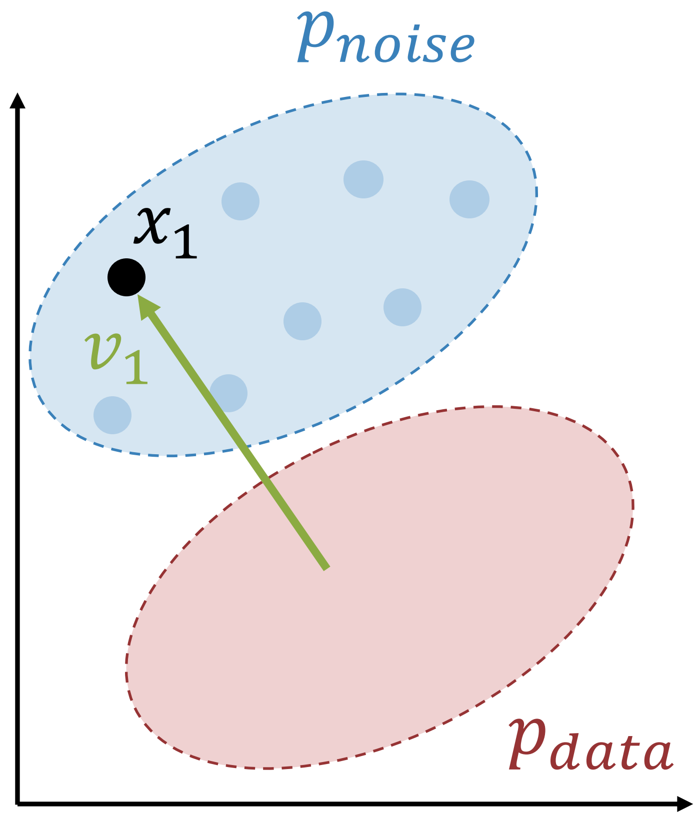
Rectified Flow: Sampling

Choose number of steps T (often T=50)

Sample $x \sim p_{noise}$

For t in $[1, 1 - \frac{1}{T}, 1 - \frac{2}{T}, \dots, 0]$:

  Evaluate $v_t = f_\theta(x_t, t)$

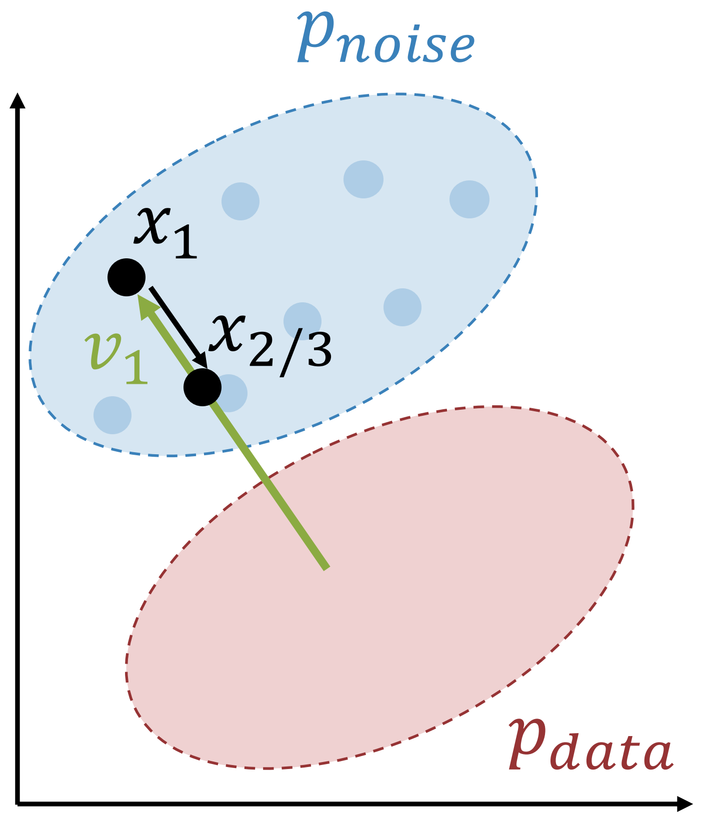


Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Rectified Flow: Sampling

Choose number of steps T (often T=50)

```
Sample $x \sim p_{noise}$
For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
    Evaluate $v_t = f_\theta(x_t, t)$
    Step $x = x - v_t/T$
```

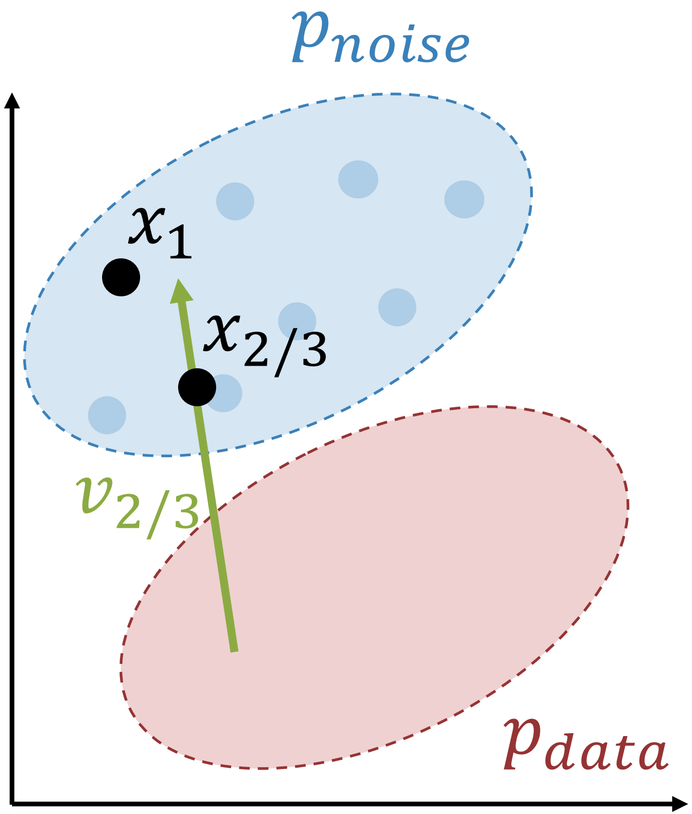


Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Rectified Flow: Sampling

Choose number of steps T (often T=50)

```
Sample $x \sim p_{noise}$
For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
    Evaluate $v_t = f_\theta(x_t, t)$
    Step $x = x - v_t/T$
```

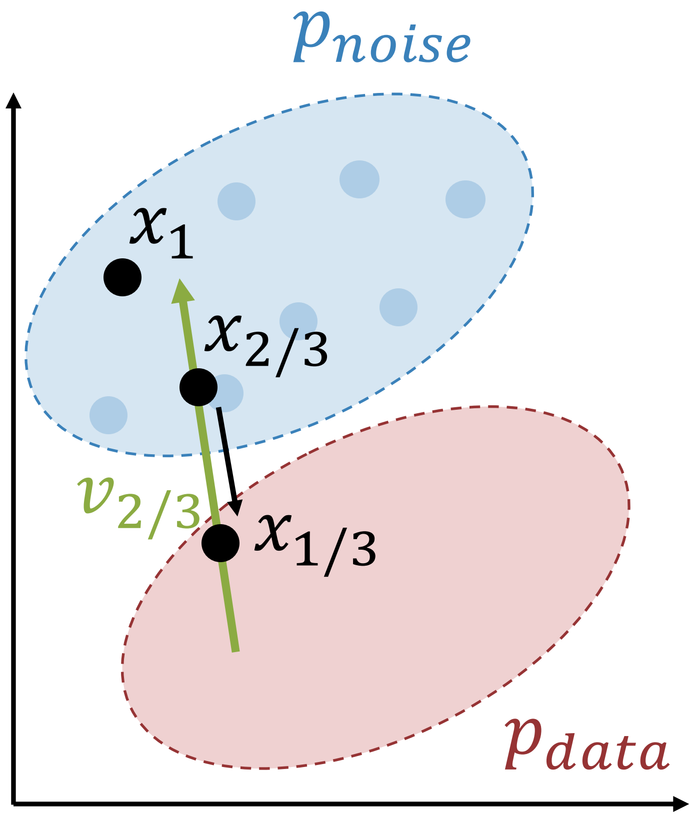


Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Rectified Flow: Sampling

Choose number of steps T (often T=50)

```
Sample $x \sim p_{noise}$
For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
    Evaluate $v_t = f_\theta(x_t, t)$
    Step $x = x - v_t/T$
```

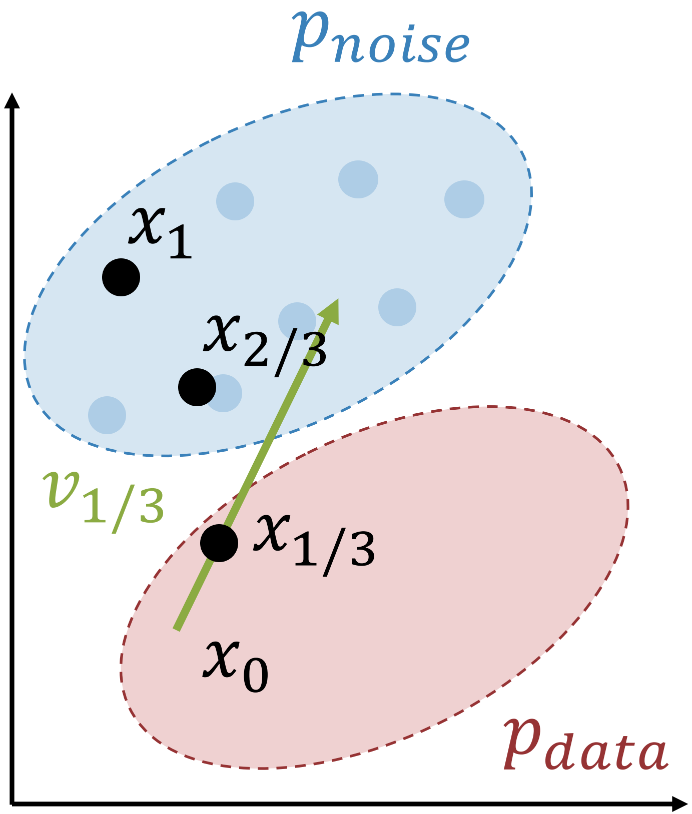


Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Rectified Flow: Sampling

Choose number of steps T (often T=50)

```
Sample $x \sim p_{noise}$
For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
    Evaluate $v_t = f_\theta(x_t, t)$
    Step $x = x - v_t/T$
```

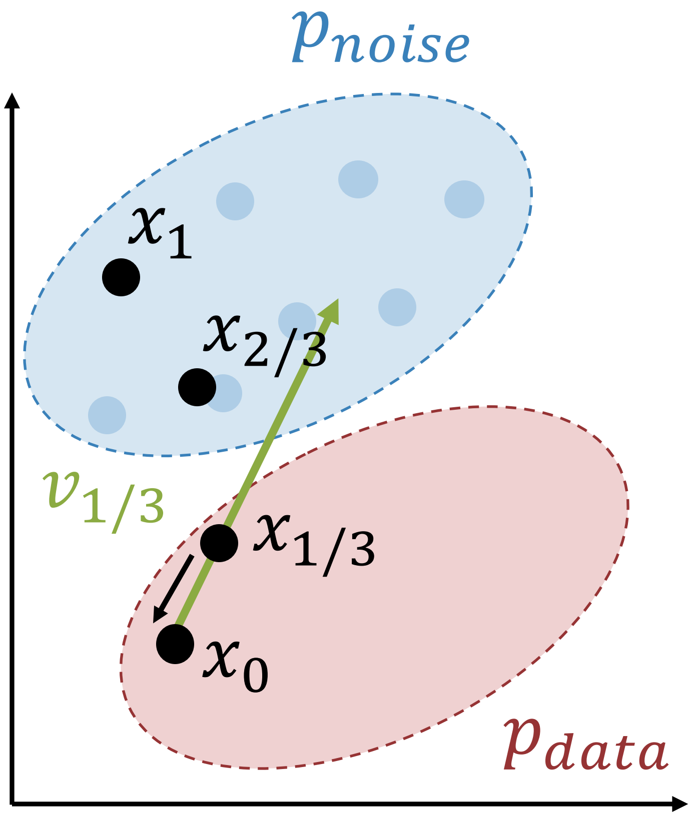


Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Rectified Flow: Sampling

Choose number of steps T (often T=50)

```
Sample $x \sim p_{noise}$
For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \dots, 0]$:
    Evaluate $v_t = f_\theta(x_t, t)$
    Step $x = x - v_t/T$
```

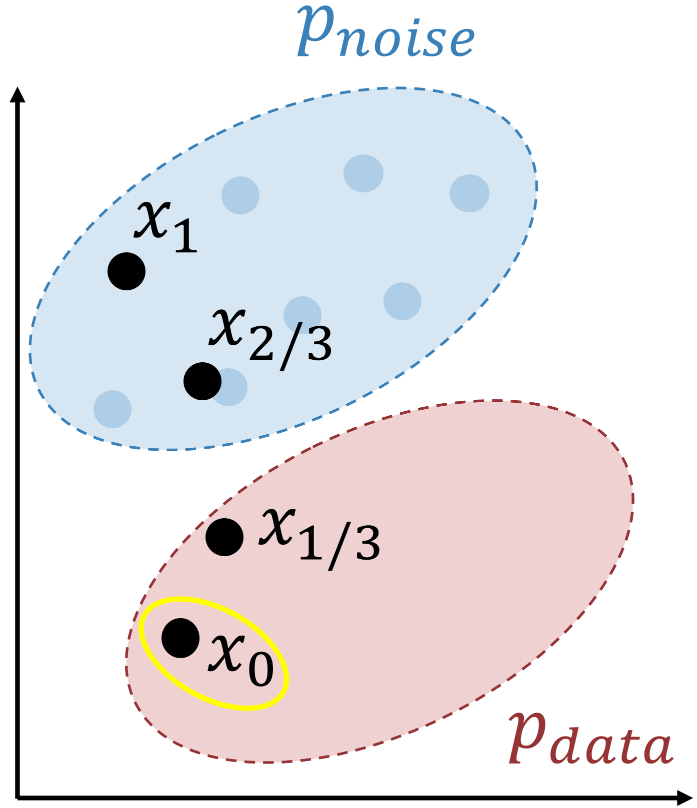


Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Rectified Flow: Sampling

Choose number of steps T (often T=50)

```
Sample $x \sim p_{noise}$
For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
    Evaluate $v_t = f_\theta(x_t, t)$
    Step $x = x - v_t/T$
Return x
```

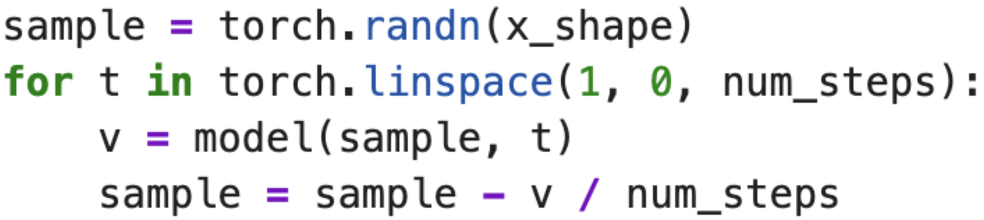


Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

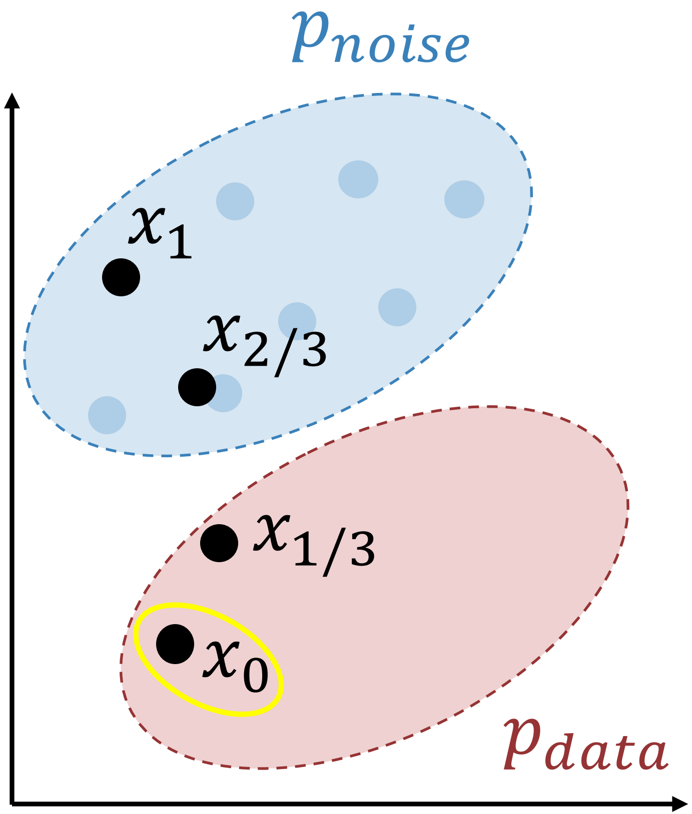
Rectified Flow: Sampling

Choose number of steps T (often T=50)

```
Sample $x \sim p_{noise}$
For t in $[1, 1-\frac{1}{T}, 1-\frac{2}{T}, \ldots, 0]$:
    Evaluate $v_t = f_\theta(x_t, t)$
    Step $x = x - v_t/T$
Return x
```



```
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, t)
    sample = sample - v / num_steps
```



Liu et al, “Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow”, 2022
Lipman et al, “Flow Matching for Generative Modeling”, 2022

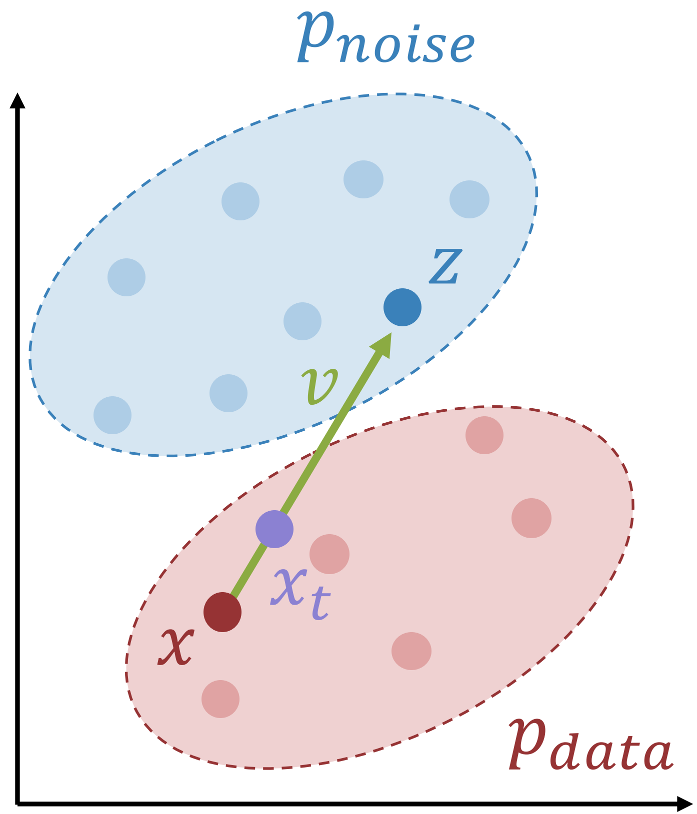
Rectified Flow: Summary

Training

```
for x in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    v = model(xt, t)
    loss = (z - x - v).square().sum()
```

Sampling

```
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, t)
    sample = sample - v / num_steps
```



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

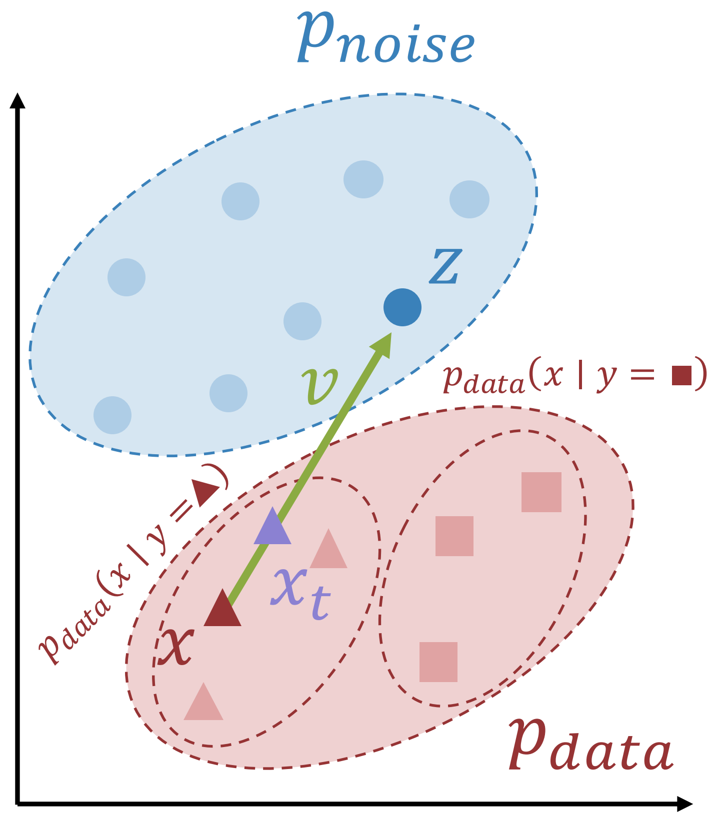
Conditional Rectified Flow

Training

```
for x in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    v = model(xt, t)
    loss = (z - x - v).square().sum()
```

Sampling

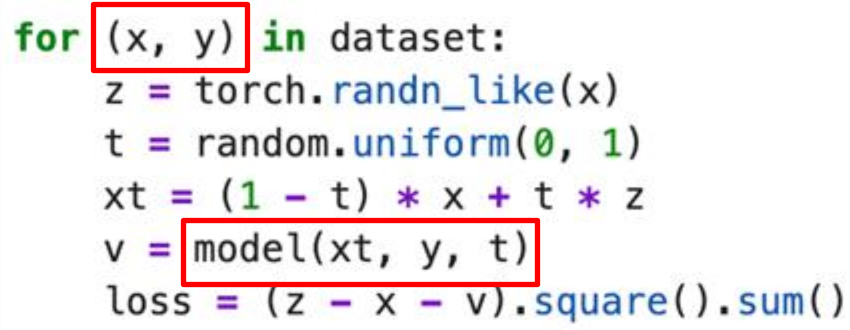
```
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, t)
    sample = sample - v / num_steps
```



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

Conditional Rectified Flow

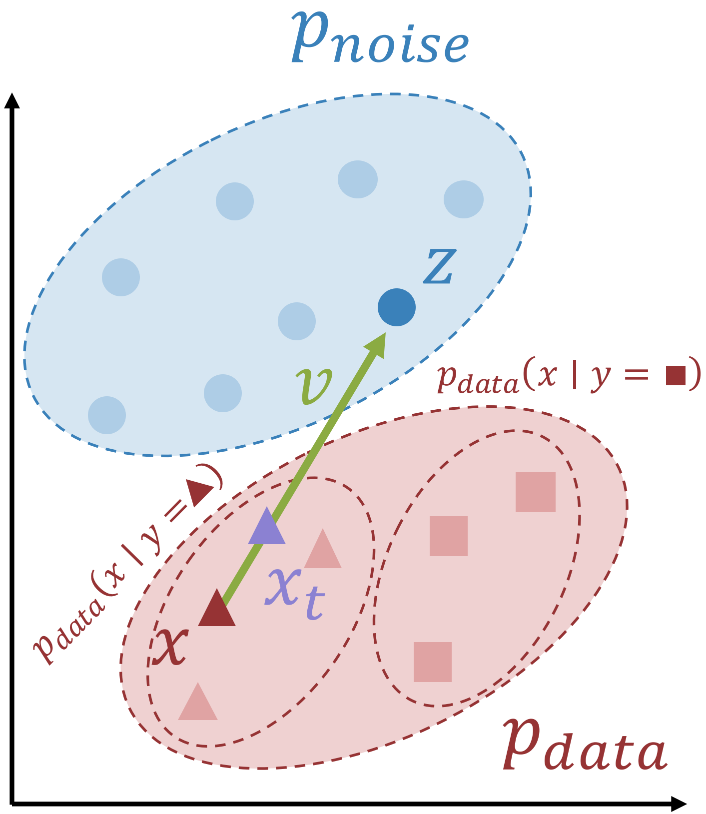
Training



```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Sampling

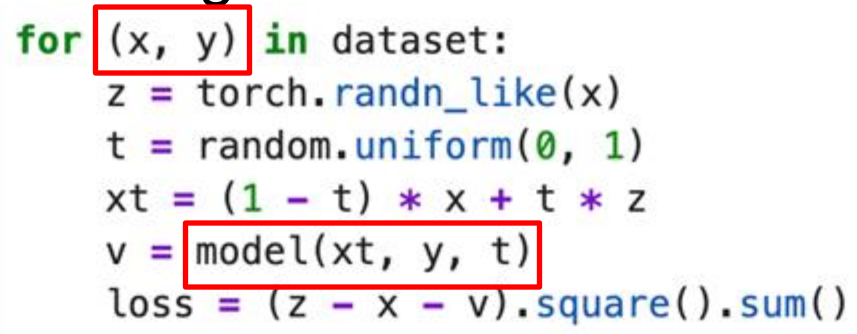
```
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, t)
    sample = sample - v / num_steps
```



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

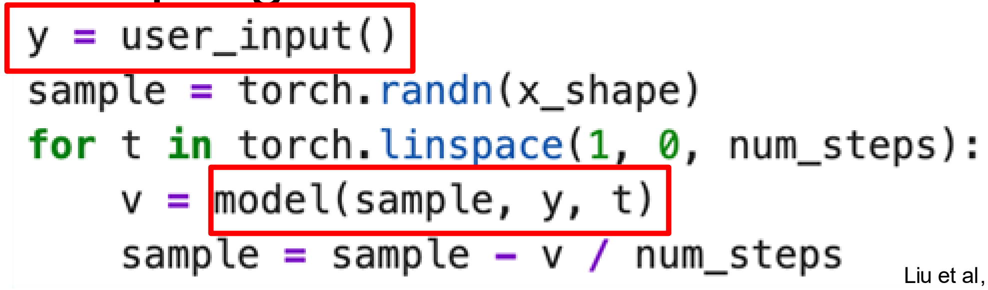
Conditional Rectified Flow

Training

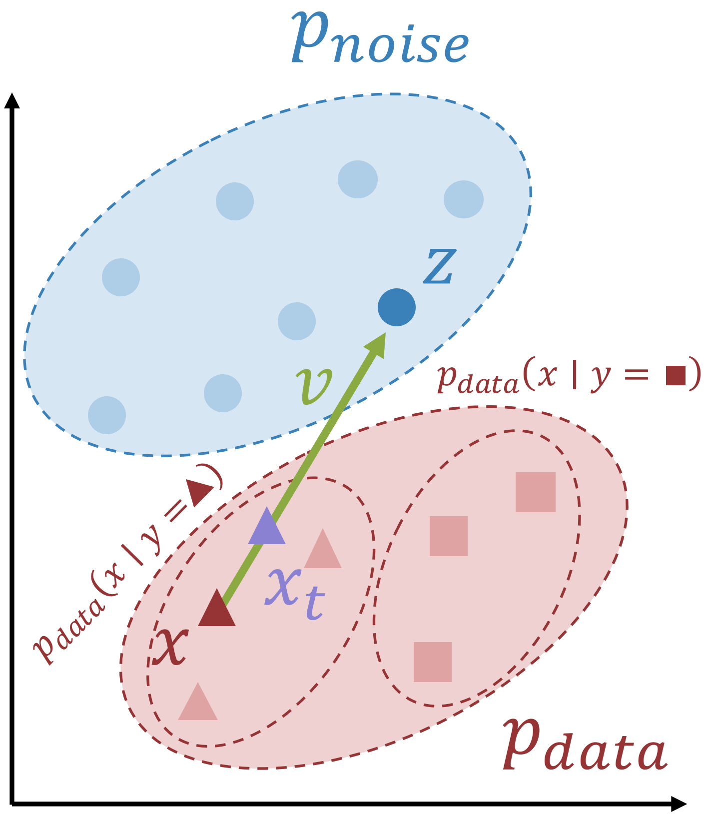


```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Sampling



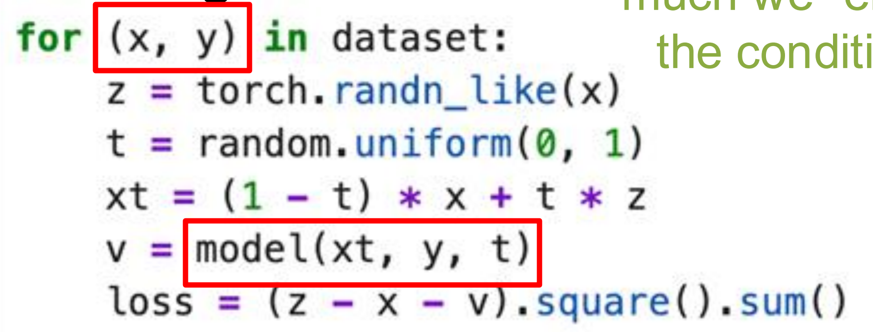
```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, y, t)
    sample = sample - v / num_steps
```



Liu et al, “Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow”, 2022
Lipman et al, “Flow Matching for Generative Modeling”, 2022

Conditional Rectified Flow

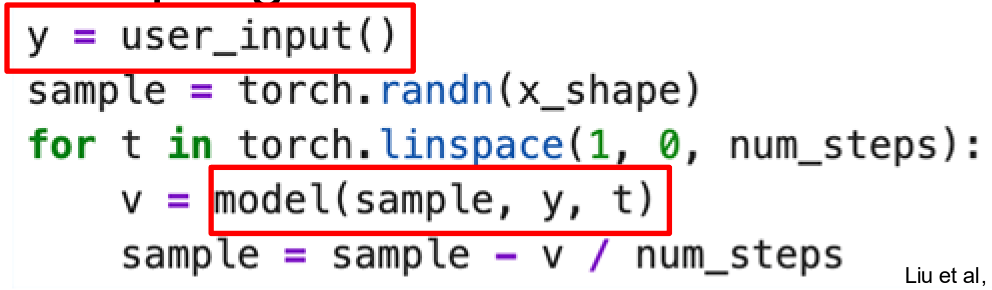
Training



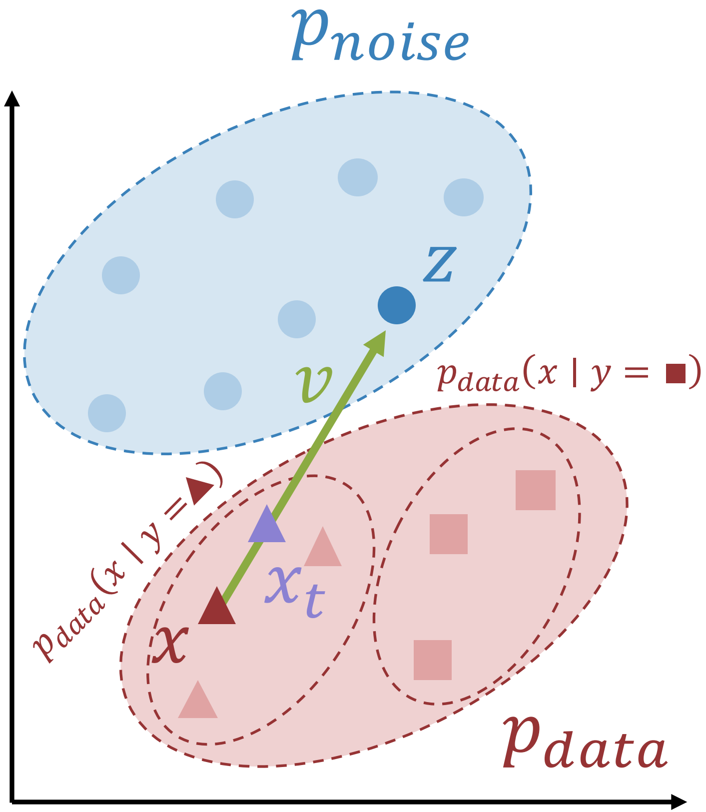
```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y?

Sampling



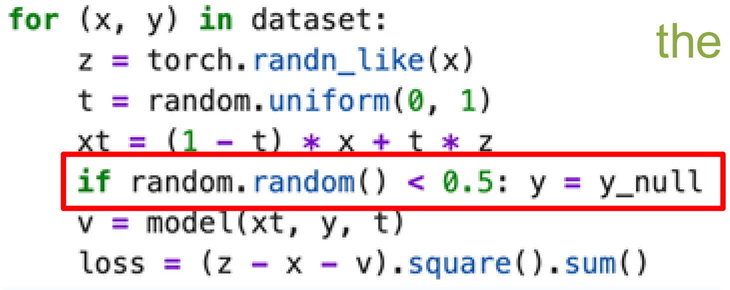
```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    v = model(sample, y, t)
    sample = sample - v / num_steps
```



Liu et al, "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow", 2022
Lipman et al, "Flow Matching for Generative Modeling", 2022

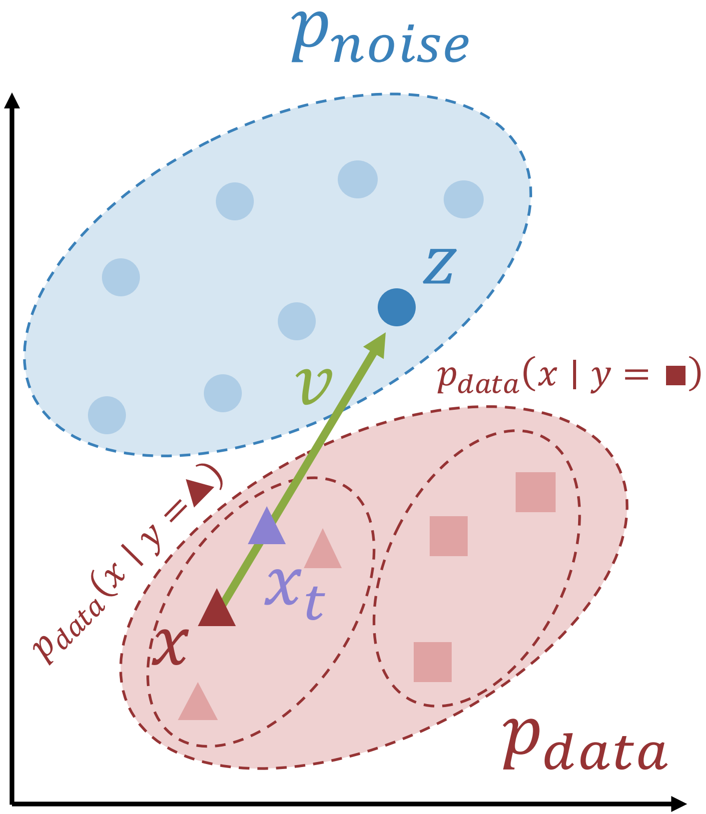
Classifier-Free Guidance (CFG)

Training



```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y?



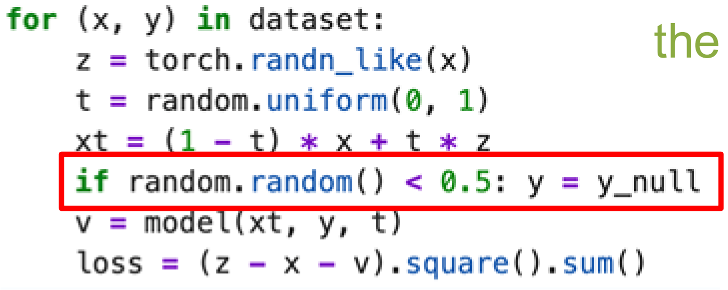
Randomly drop y during training.

Now the same model is conditional and unconditional!

Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Classifier-Free Guidance (CFG)

Training

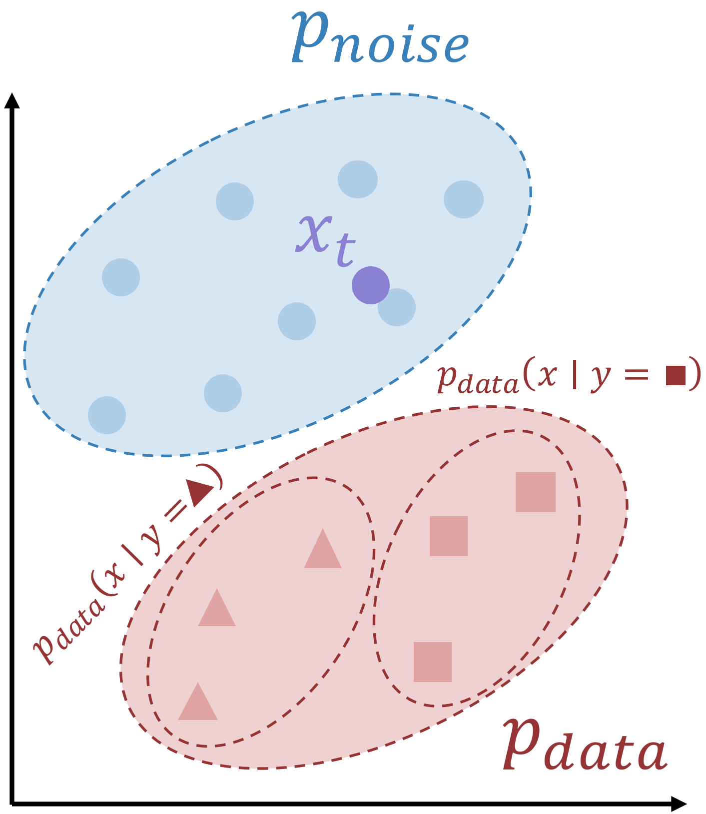


```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y?

Randomly drop y during training.

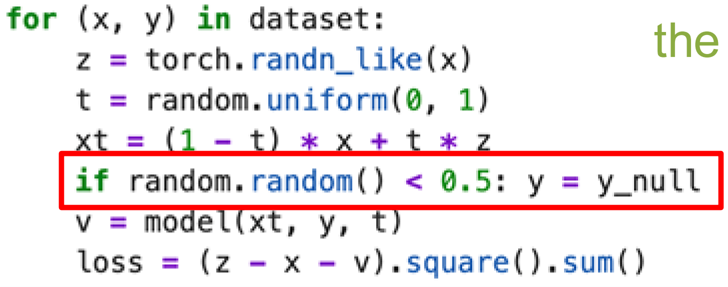
Now the same model is conditional and unconditional!
Consider a noisy x_t:



Ho and Salimans, "Classifier-Free Diffusion Guidance", arXiv 2022

Classifier-Free Guidance (CFG)

Training

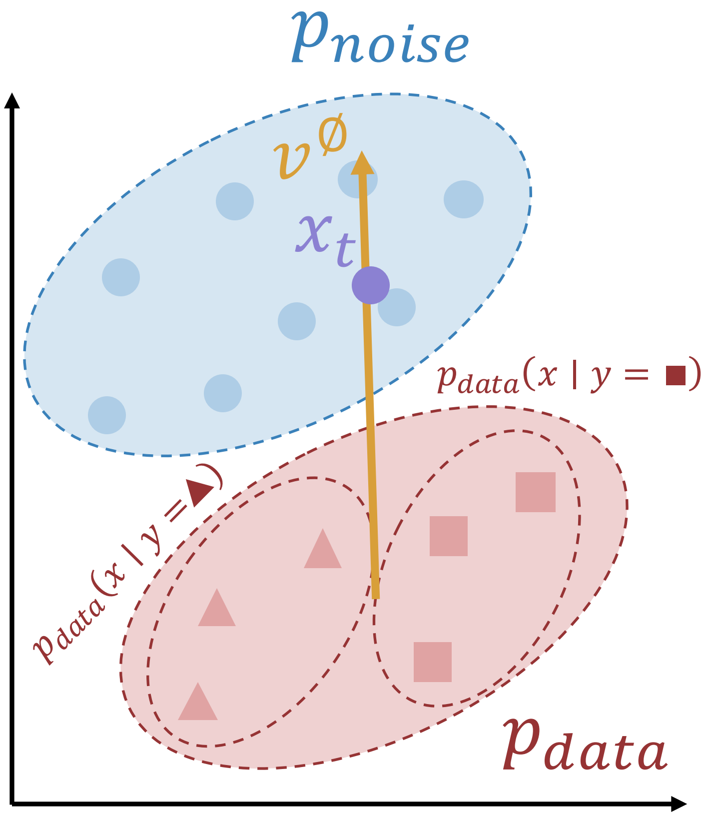


```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y?

Randomly drop y during training.

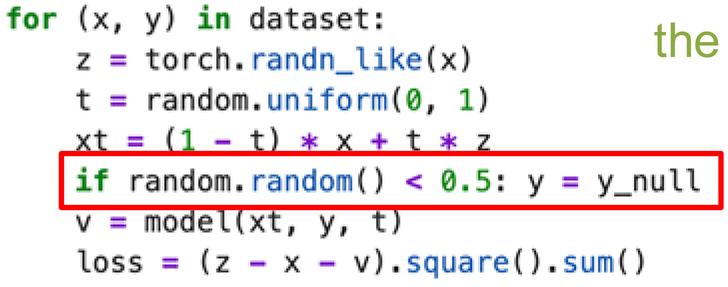
Now the same model is conditional and unconditional!
Consider a noisy x_t:
$v^{\emptyset} = f_\theta(x_t, y_\emptyset, t)$ points toward $p(x)$



Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Classifier-Free Guidance (CFG)

Training



```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y?

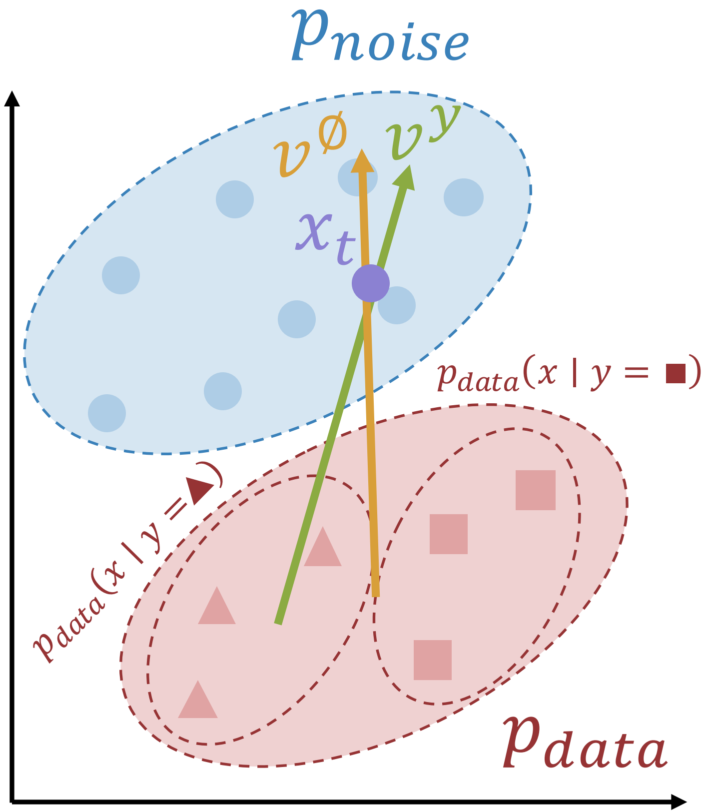
Randomly drop y during training.

Now the same model is conditional and unconditional!

Consider a noisy x_t:

$v^{\emptyset} = f_\theta(x_t, y_\emptyset, t)$ points toward $p(x)$

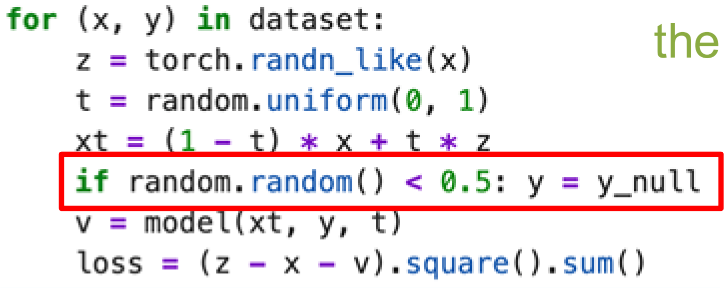
$v^{y} = f_\theta(x_t, y, t)$ points toward $p(x \mid y)$



Ho and Salimans, "Classifier-Free Diffusion Guidance", arXiv 2022

Classifier-Free Guidance (CFG)

Training



```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y?

Randomly drop y during training.

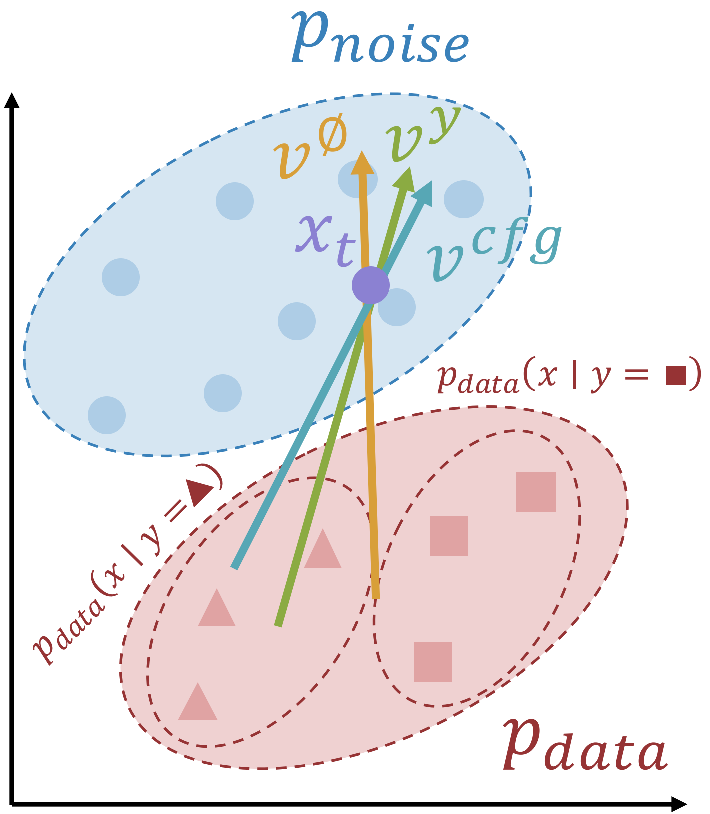
Now the same model is conditional and unconditional!

Consider a noisy x_t:

$v^{\emptyset} = f_\theta(x_t, y_\emptyset, t)$ points toward $p(x)$

$v^{y} = f_\theta(x_t, y, t)$ points toward $p(x \mid y)$

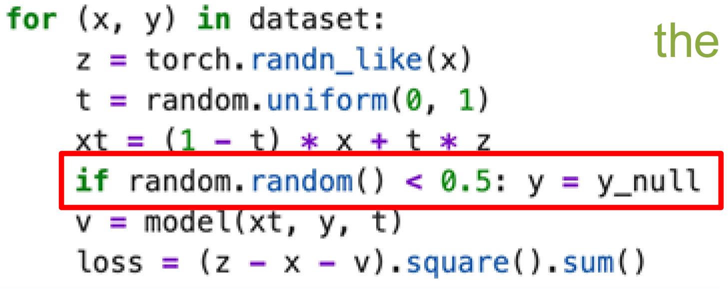
$v^{cfg} = (1 + w)v^{y} - wv^{\emptyset}$ points more toward $p(x \mid y)$



Ho and Salimans, "Classifier-Free Diffusion Guidance", arXiv 2022

Classifier-Free Guidance (CFG)

Training

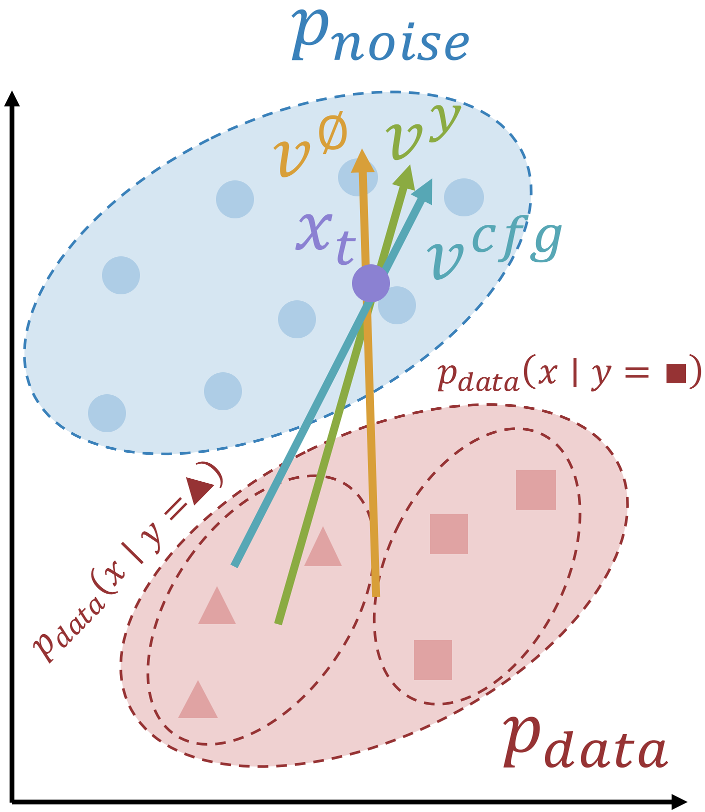


```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we "emphasize" the conditioning y?

Randomly drop y during training.

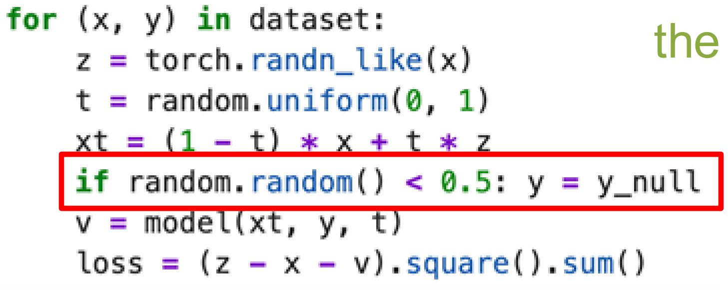
Now the same model is conditional and unconditional!
Consider a noisy x_t:
$v^{\emptyset} = f_\theta(x_t, y_\emptyset, t)$ points toward $p(x)$
$v^{y} = f_\theta(x_t, y, t)$ points toward $p(x \mid y)$
$v^{cfg} = (1+w)v^{y} - wv^{\emptyset}$ points more toward $p(x \mid y)$
During sampling, step according to v^{cfg}



Ho and Salimans, "Classifier-Free Diffusion Guidance", arXiv 2022

Classifier-Free Guidance (CFG)

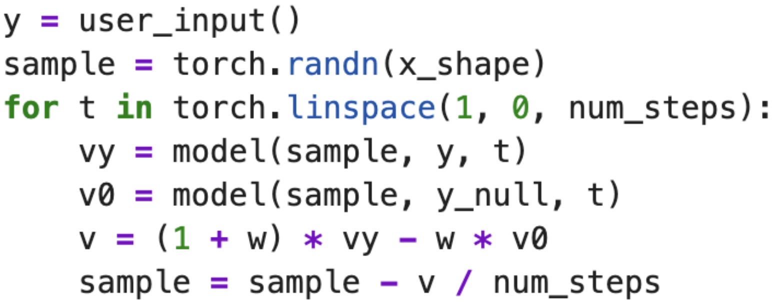
Training



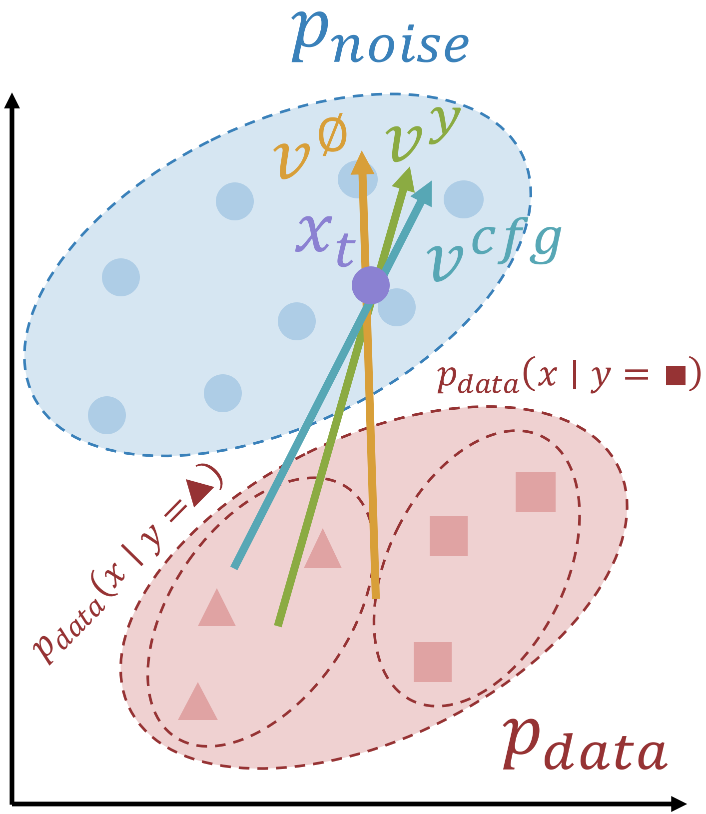
```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y?

Sampling



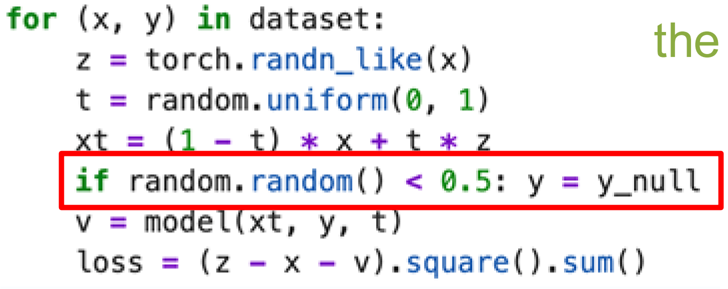
```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```



Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

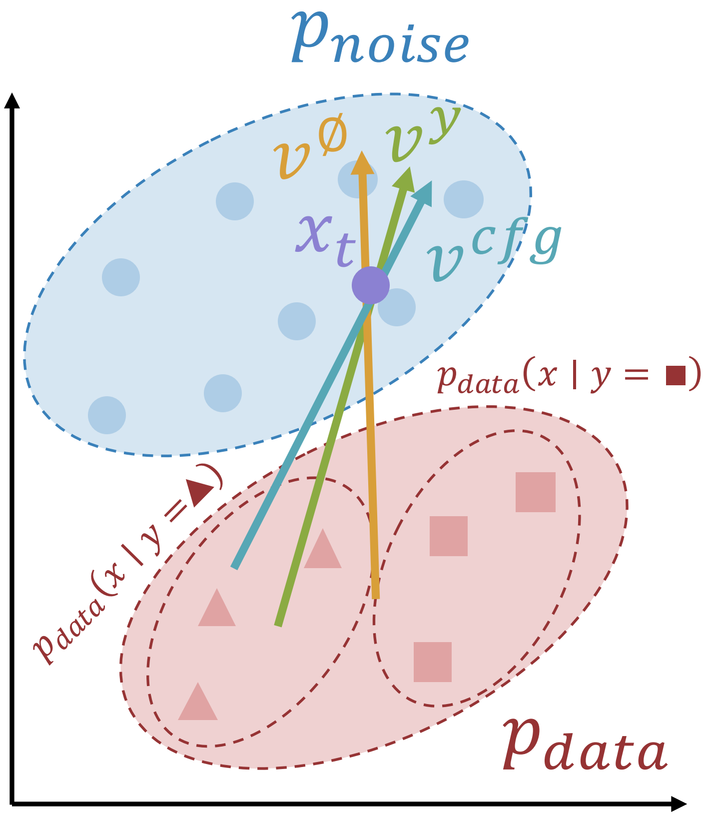
Classifier-Free Guidance (CFG)

Training

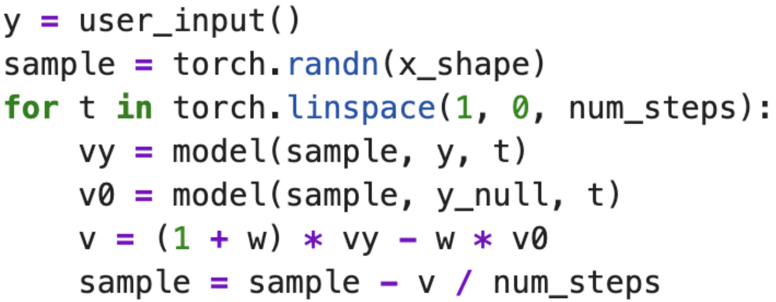


```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y?



Sampling

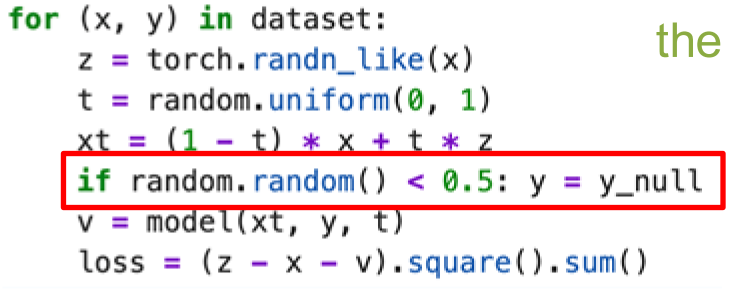


```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Classifier-Free Guidance (CFG)

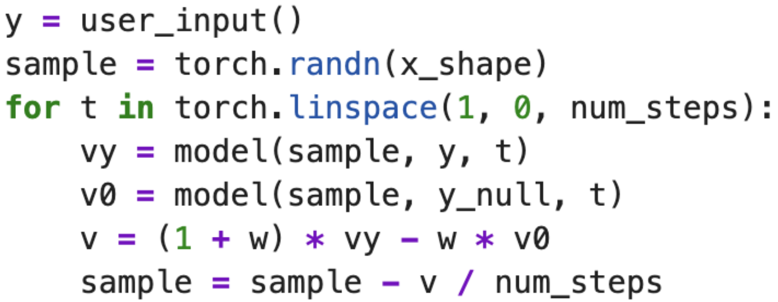
Training



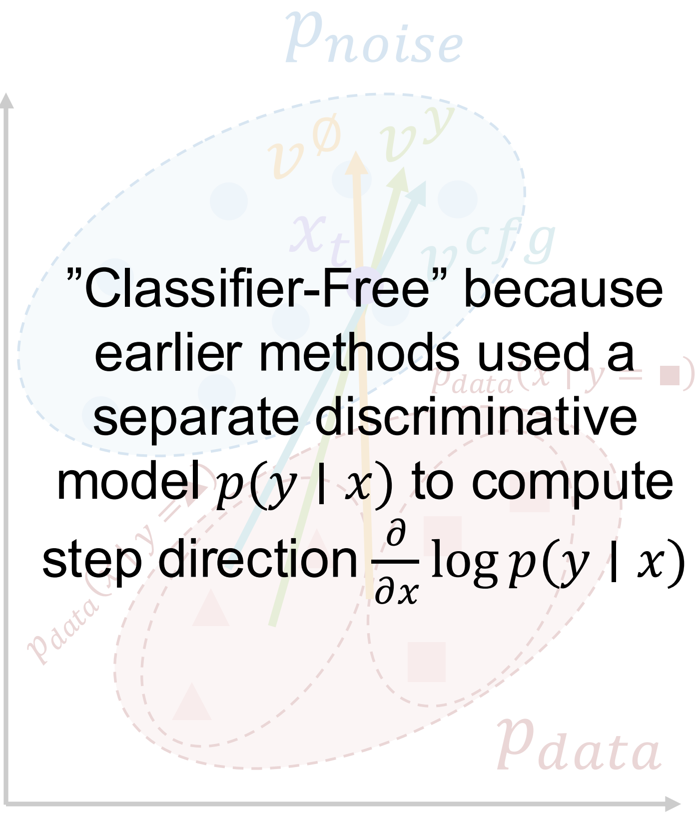
```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y?

Sampling



```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```



”Classifier-Free” because earlier methods used a separate discriminative model $p(y \mid x)$ to compute step direction $\frac{\partial}{\partial x} \log p(y \mid x)$

Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Classifier-Free Guidance (CFG)

Training

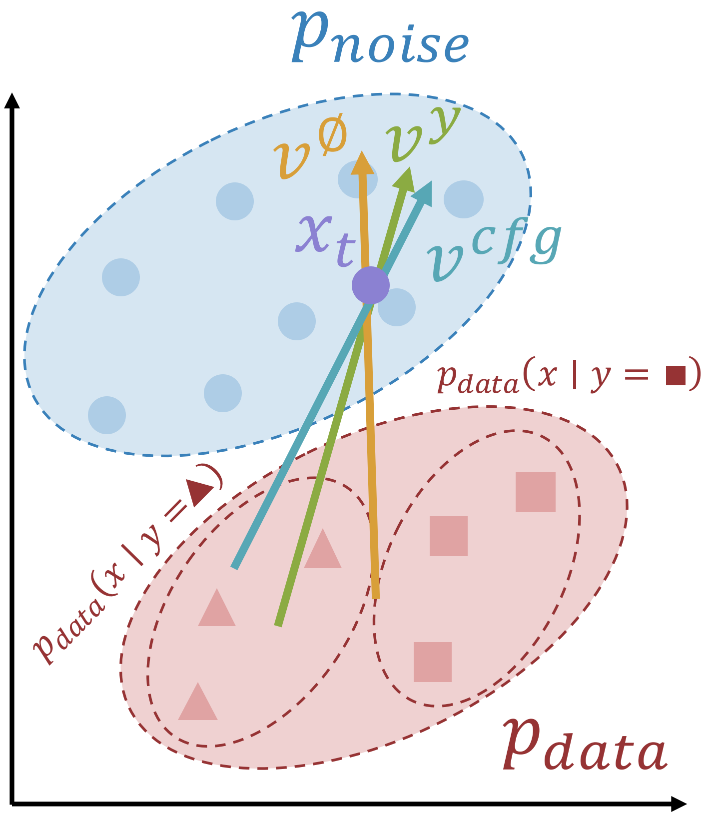
```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y?

Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

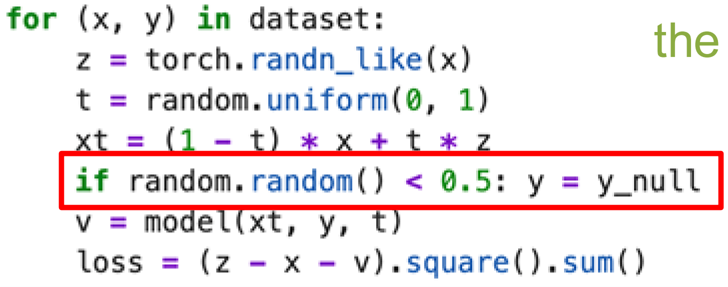
Used everywhere in practice! Very important for high-quality outputs



Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Classifier-Free Guidance (CFG)

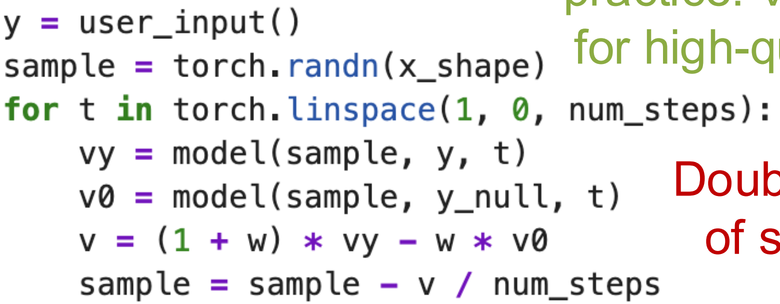
Training



```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Can we control how much we “emphasize” the conditioning y?

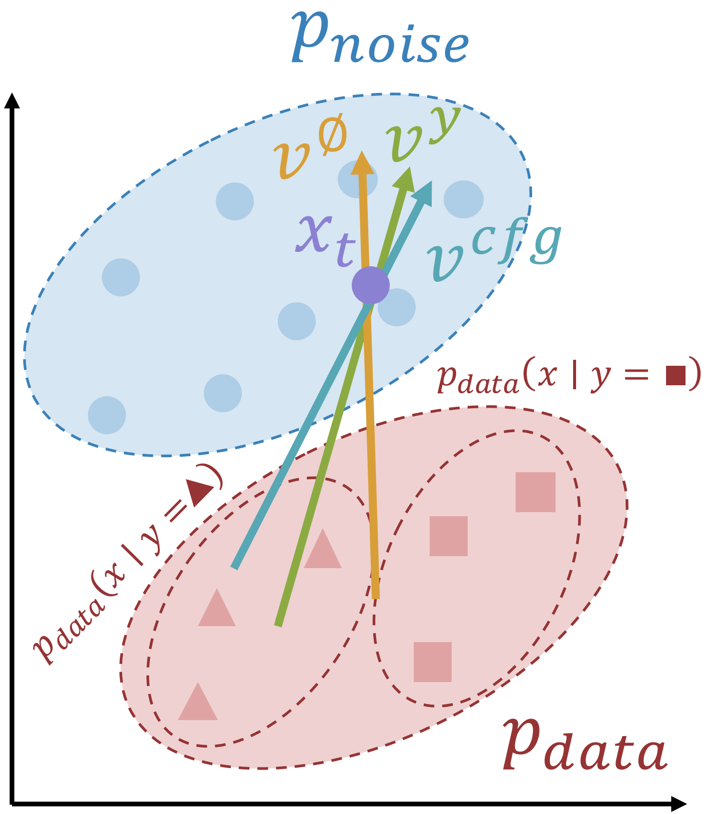
Sampling



```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

Used everywhere in practice! Very important for high-quality outputs

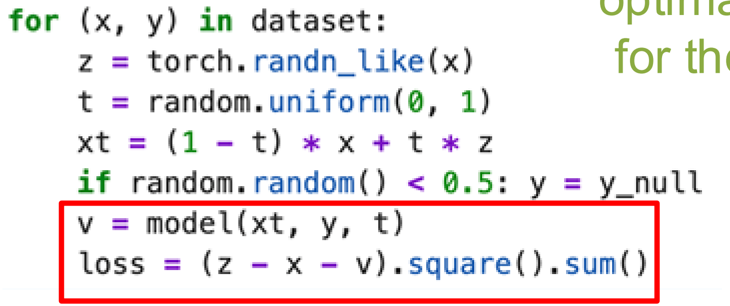
Doubles the cost of sampling…



Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

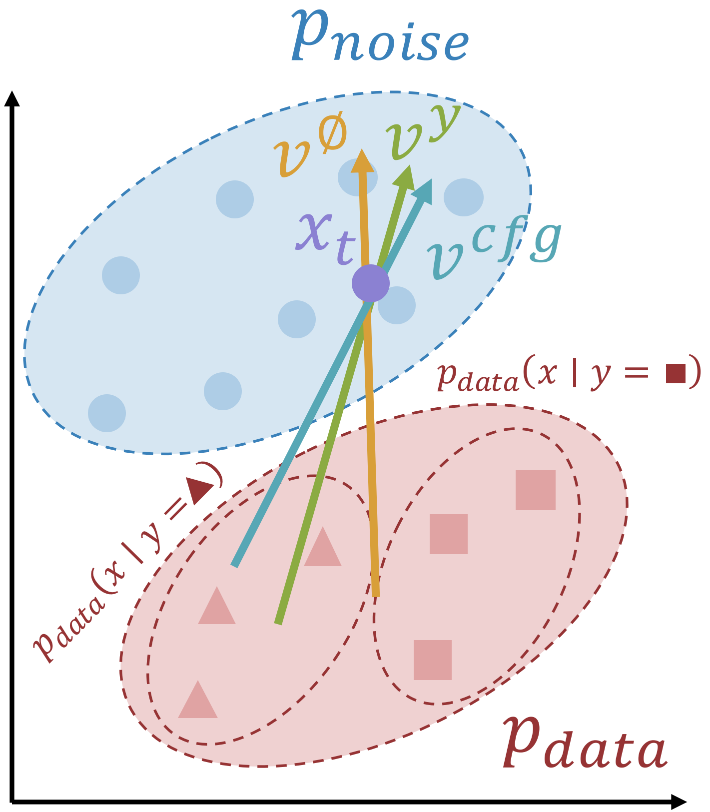
Optimal Prediction

Training



```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

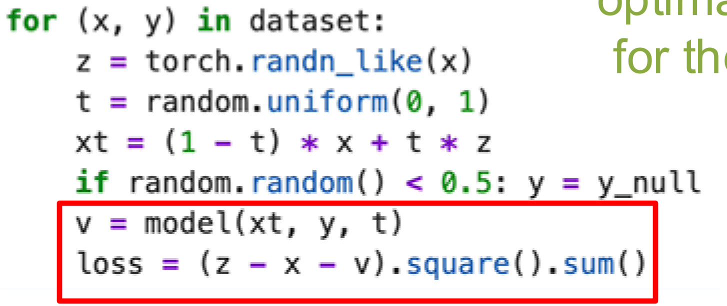
Q: What is the optimal prediction for the network?



Dhariwal and Nichol, "Diffusion Models beat GANs on Image Synthesis", arXiv 2021
Ho and Salimans, "Classifier-Free Diffusion Guidance", arXiv 2022

Optimal Prediction

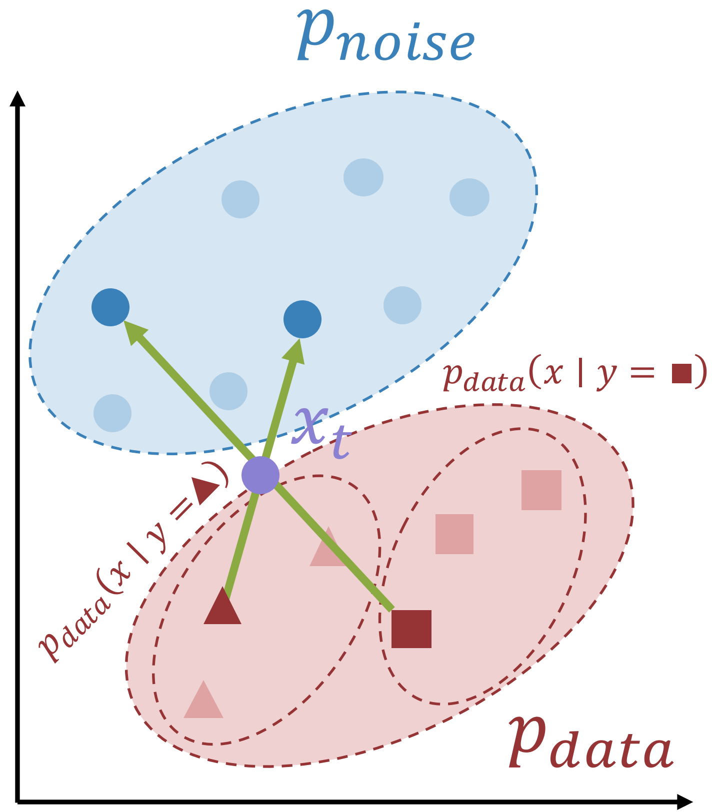
Training



```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

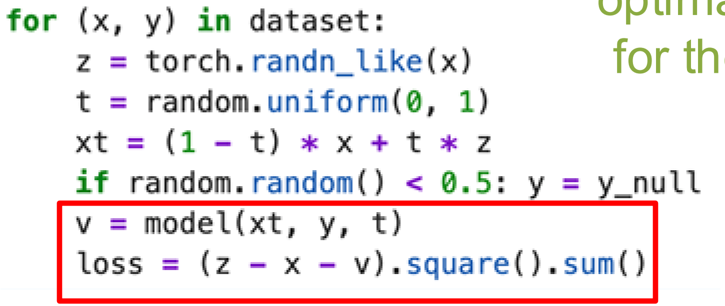
There may be many pairs (x, z) that give the same x_t; network must average over them



Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Optimal Prediction

Training

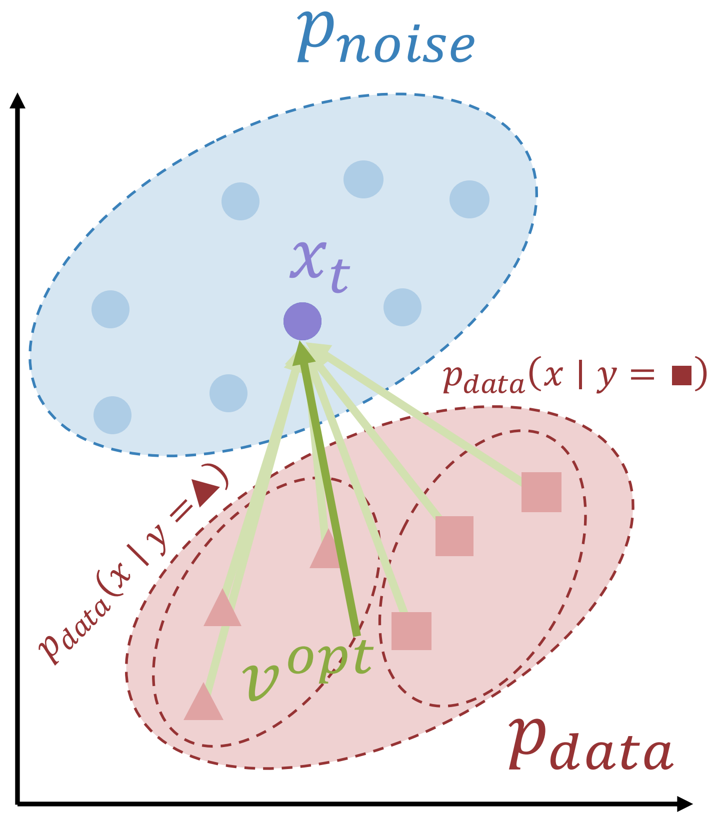


```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

There may be many pairs (x, z) that give the same x_t; network must average over them

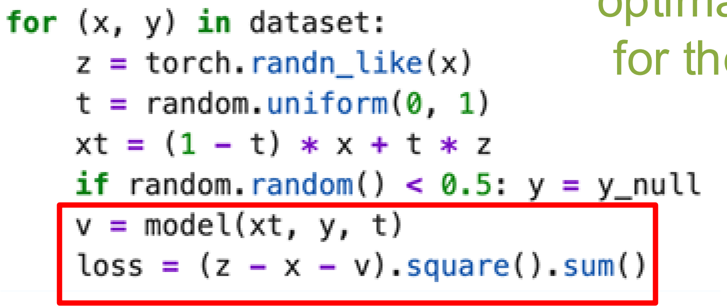
Full noise (t=1) is easy: optimal v is mean of p_{data}



Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Optimal Prediction

Training

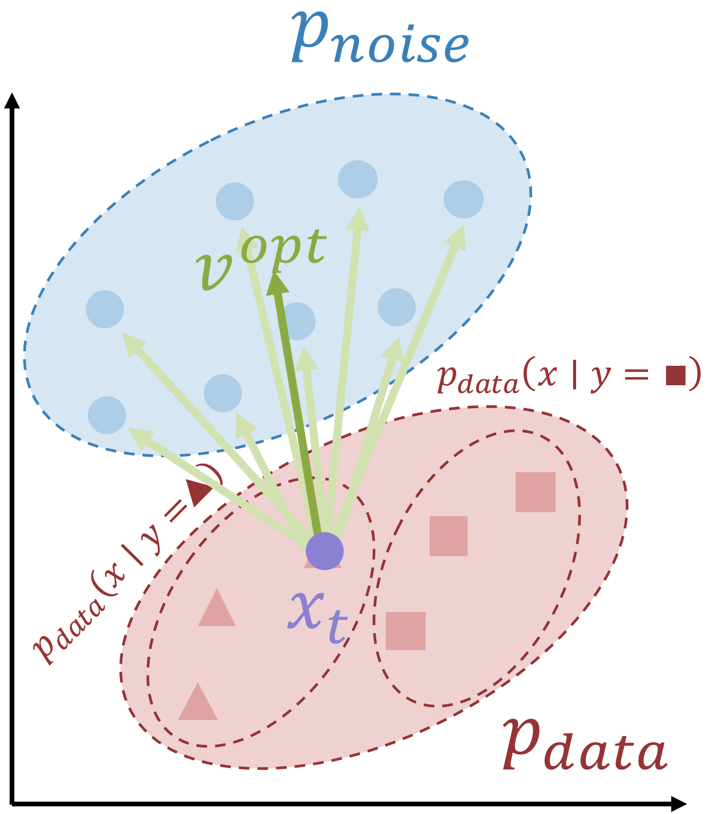


```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

There may be many pairs (x, z) that give the same x_t; network must average over them

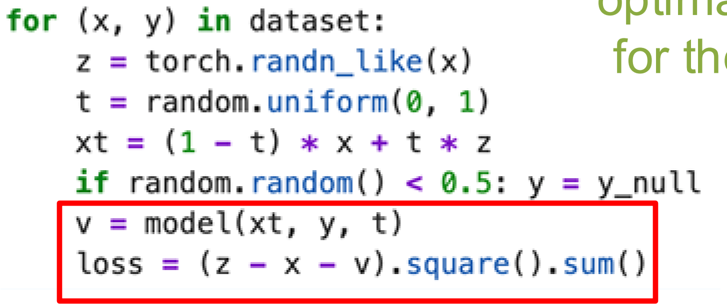
Full noise (t=1) is easy: optimal v is mean of p_{data}
No noise (t=0) is easy: optimal v is mean of p_{noise}



Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Optimal Prediction

Training

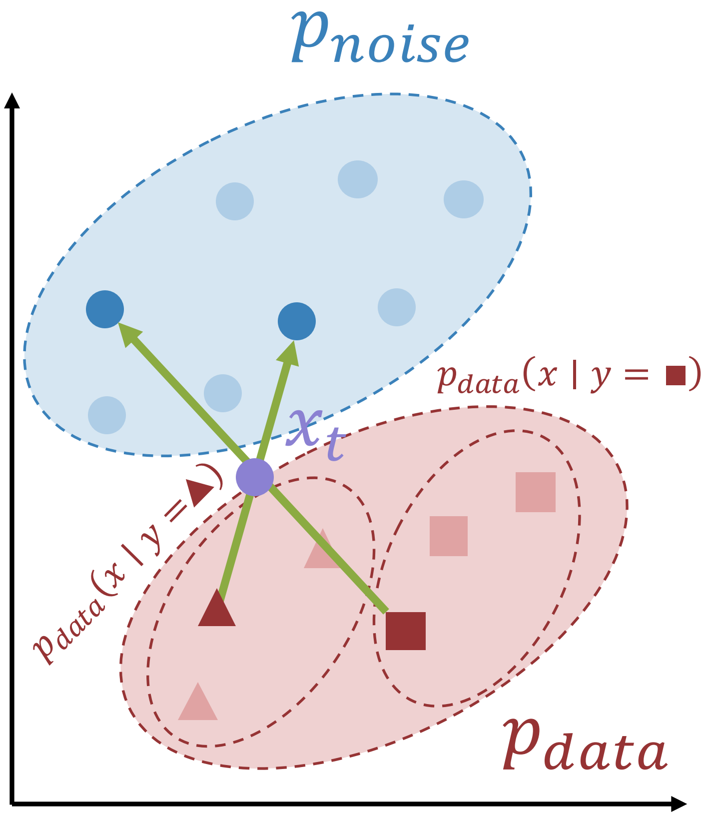


```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

There may be many pairs (x, z) that give the same x_t; network must average over them

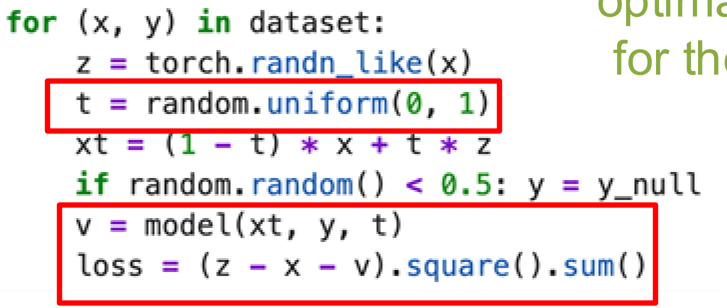
Full noise (t=1) is easy: optimal v is mean of p_{data}
No noise (t=0) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous



Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Optimal Prediction

Training

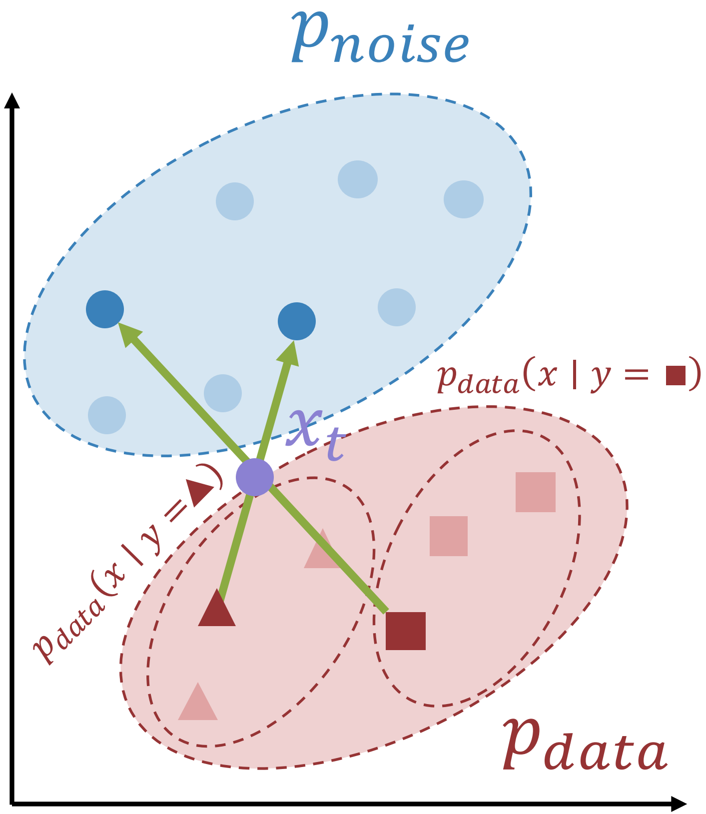


```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

There may be many pairs (x, z) that give the same x_t; network must average over them

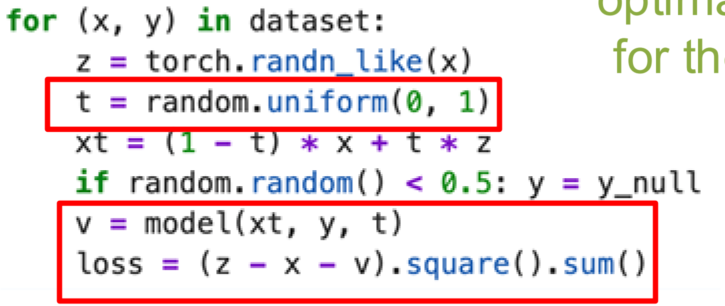
Full noise (t=1) is easy: optimal v is mean of p_{data}
No noise (t=0) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous
But we give equal weight to all noise levels!



Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Optimal Prediction

Training

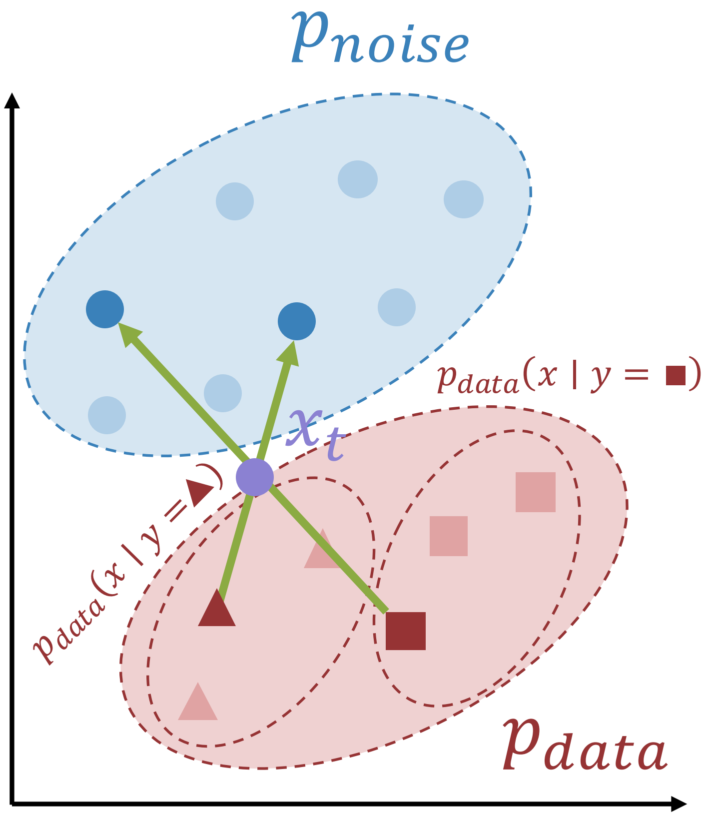


```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Q: What is the optimal prediction for the network?

There may be many pairs (x, z) that give the same x_t; network must average over them

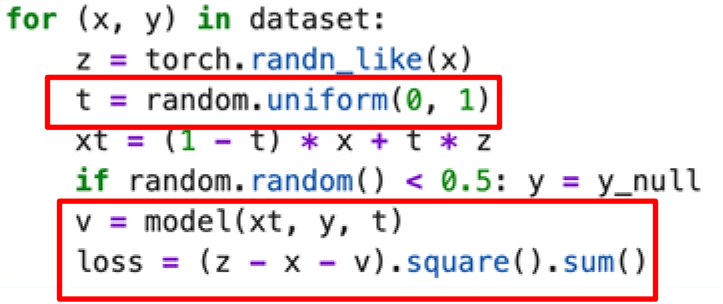
Full noise (t=1) is easy: optimal v is mean of p_{data}
No noise (t=0) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous
But we give equal weight to all noise levels!
Solution: Use a non-uniform noise schedule



Dhariwal and Nichol, “Diffusion Models beat GANs on Image Synthesis”, arXiv 2021
Ho and Salimans, “Classifier-Free Diffusion Guidance”, arXiv 2022

Noise Schedules

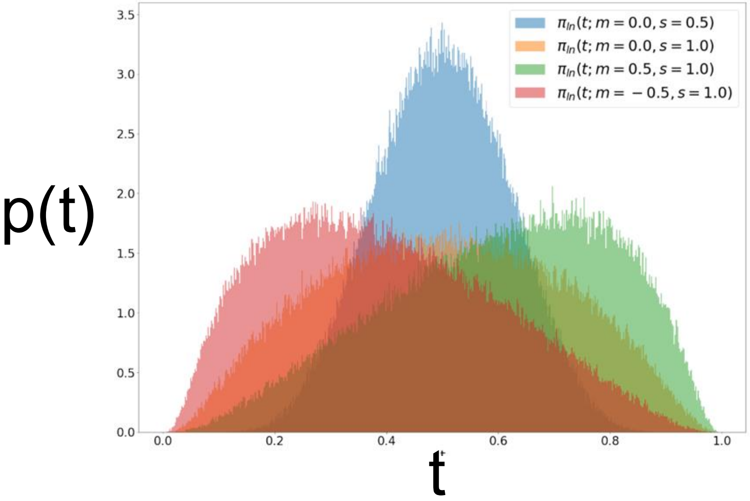
Training



```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = random.uniform(0, 1)
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

There may be many pairs (x, z) that give the same x_t; network must average over them

Full noise (t=1) is easy: optimal v is mean of p_{data}
No noise (t=0) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous
But we give equal weight to all noise levels!
Solution: Use a non-uniform noise schedule

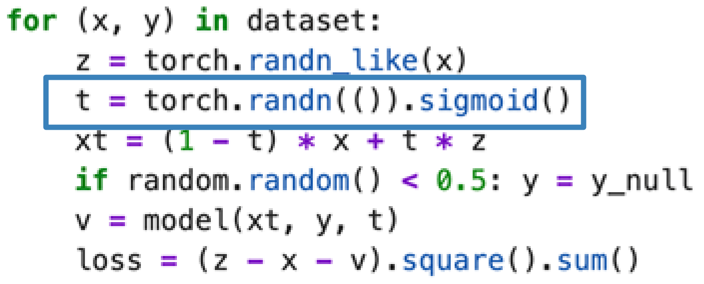


Put more emphasis on middle noise

Esser et al, "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis", arXiv 2024

Noise Schedules

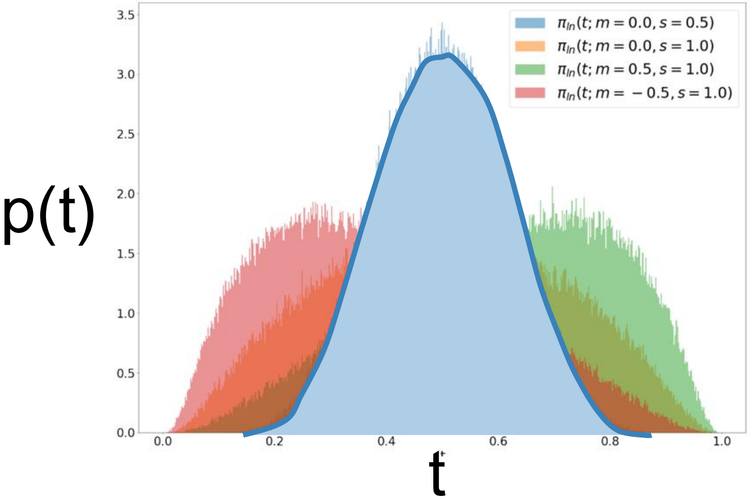
Training



```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = torch.randn(()).sigmoid()
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

There may be many pairs (x, z) that give the same x_t; network must average over them

Full noise (t=1) is easy: optimal v is mean of p_{data}
No noise (t=0) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous
But we give equal weight to all noise levels!
Solution: Use a non-uniform noise schedule



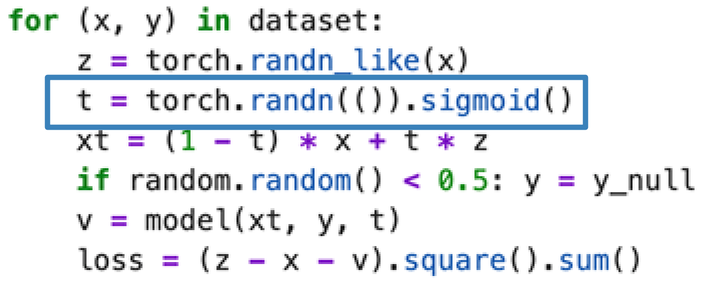
Put more emphasis on middle noise

Common choice: **logit-normal sampling**

Esser et al, “Scaling Rectified Flow Transformers for High-Resolution Image Synthesis”, arXiv 2024

Noise Schedules

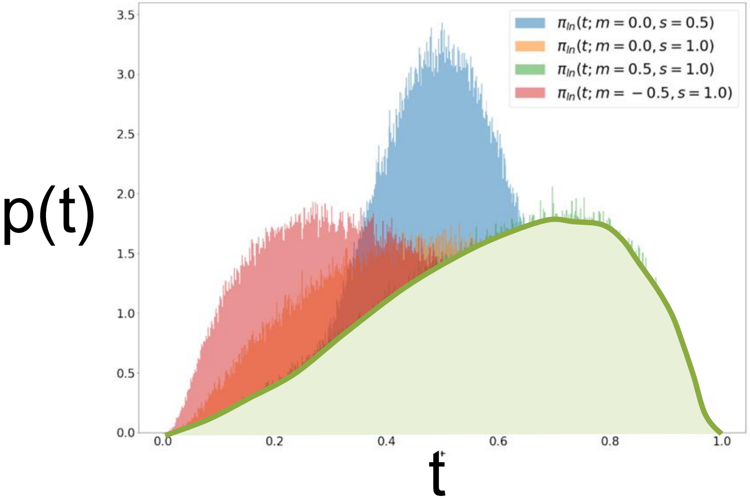
Training



```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = torch.randn(()).sigmoid()
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

There may be many pairs (x, z) that give the same x_t; network must average over them

Full noise (t=1) is easy: optimal v is mean of p_{data}
No noise (t=0) is easy: optimal v is mean of p_{noise}
Middle noise is hardest, most ambiguous
But we give equal weight to all noise levels!
Solution: Use a non-uniform noise schedule



Put more emphasis on middle noise

Common choice: **logit-normal sampling**

For high-res data, often **shift to higher noise** to account for pixel correlations

Esser et al, "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis", arXiv 2024

Diffusion: Rectified Flow

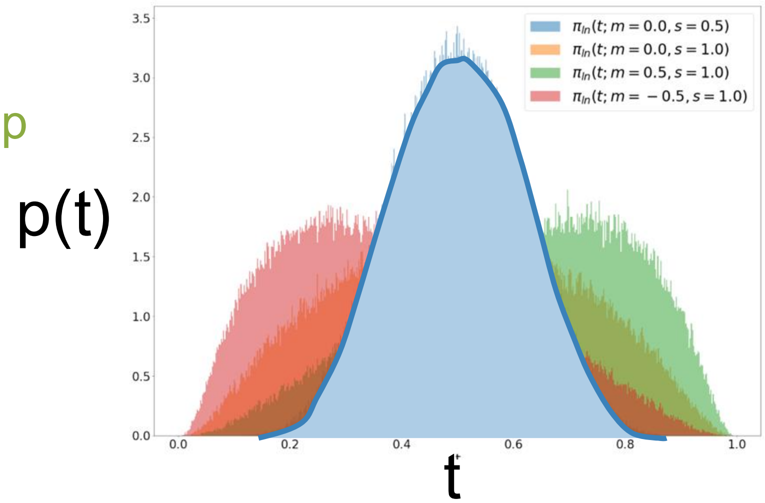
Training

```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = torch.randn(()).sigmoid()
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Simple and scalable setup for many generative modeling problems!

Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```



Put more emphasis on middle noise

Common choice: **logit-normal sampling**

For high-res data, often **shift to higher noise** to account for pixel correlations

Esser et al, "Scaling Rectified Flow Transformers for High-Resolution Image Synthesis", arXiv 2024

Diffusion: Rectified Flow

Training

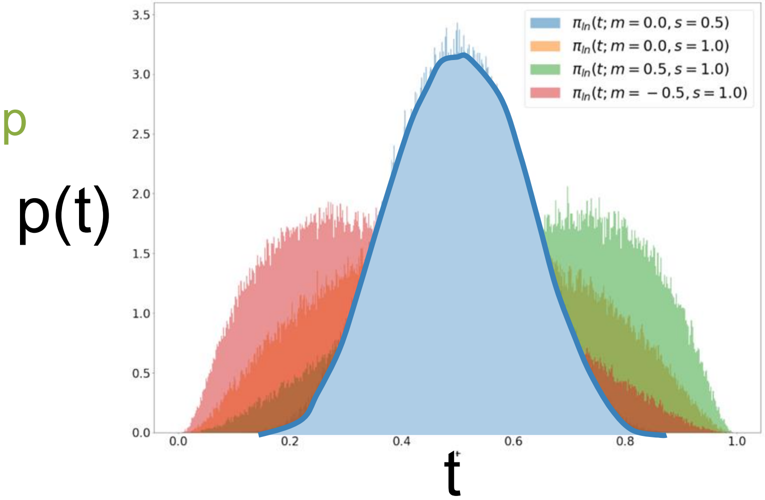
```
for (x, y) in dataset:
    z = torch.randn_like(x)
    t = torch.randn(()).sigmoid()
    xt = (1 - t) * x + t * z
    if random.random() < 0.5: y = y_null
    v = model(xt, y, t)
    loss = (z - x - v).square().sum()
```

Simple and scalable setup for many generative modeling problems!

Sampling

```
y = user_input()
sample = torch.randn(x_shape)
for t in torch.linspace(1, 0, num_steps):
    vy = model(sample, y, t)
    v0 = model(sample, y_null, t)
    v = (1 + w) * vy - w * v0
    sample = sample - v / num_steps
```

Problem: Does not work naively on high-resolution data



Put more emphasis on middle noise

Common choice: **logit-normal sampling**

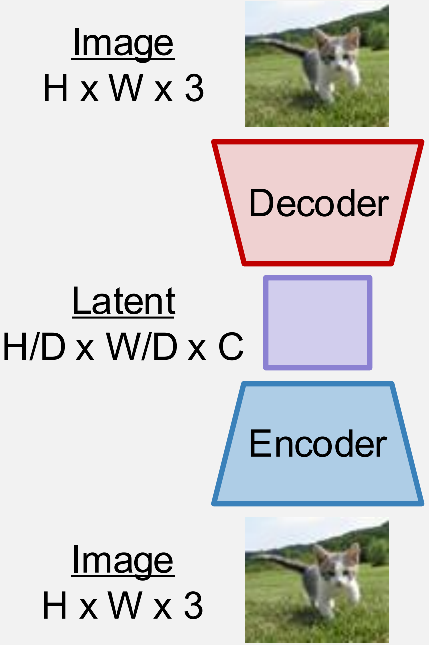
For high-res data, often **shift to higher noise** to account for pixel correlations

Esser et al, “Scaling Rectified Flow Transformers for High-Resolution Image Synthesis”, arXiv 2024

Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

Latent Diffusion Models (LDMs)

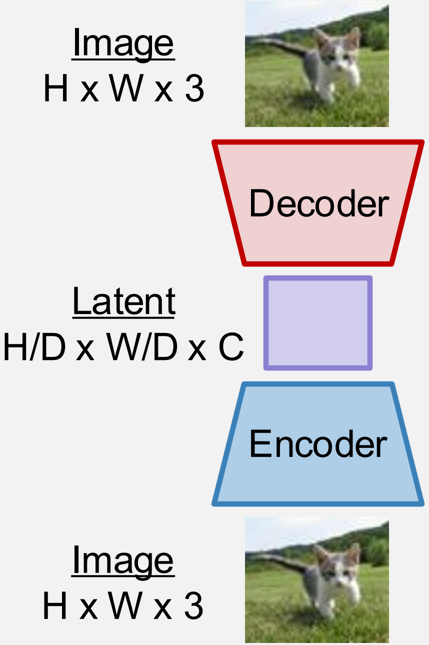
Train **encoder** + **decoder** to convert images to **latents**



Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**



Common setting: D=8, C=16

Image: 256 x 256 x 3
=> **Latent**: 32 x 32 x 16

Encoder / Decoder are CNNs with attention

Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**

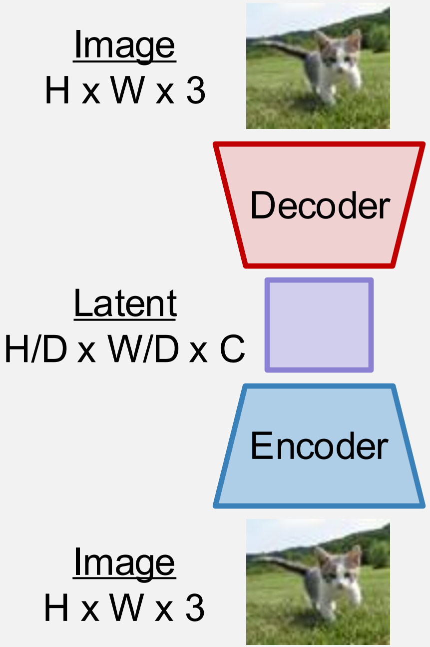


Image
H x W x 3

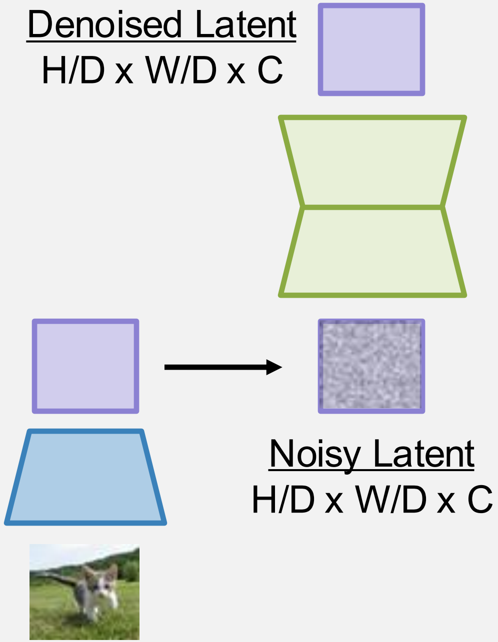
Decoder

Latent
H/D x W/D x C

Encoder

Image
H x W x 3

Train **diffusion model** to remove noise from **latents** (**Encoder** is frozen)



Denoised Latent
H/D x W/D x C

Noisy Latent
H/D x W/D x C

Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**

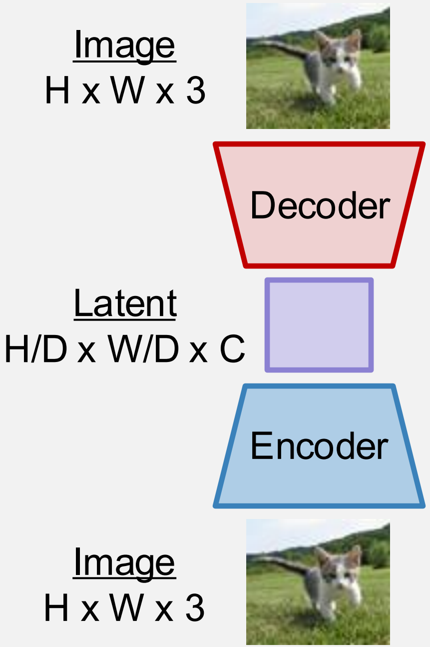


Image
H x W x 3

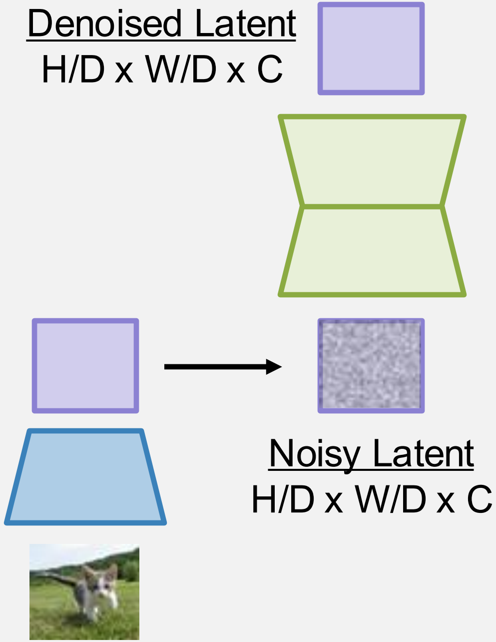
Decoder

Latent
H/D x W/D x C

Encoder

Image
H x W x 3

Train **diffusion model** to remove noise from **latents** (**Encoder** is frozen)



Denoised Latent
H/D x W/D x C

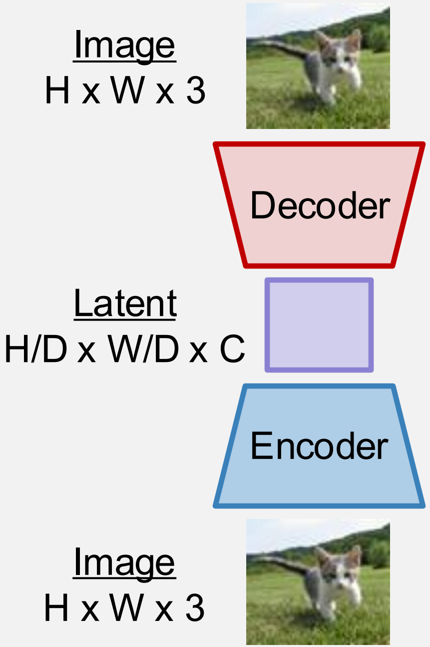
Noisy Latent
H/D x W/D x C

After training:

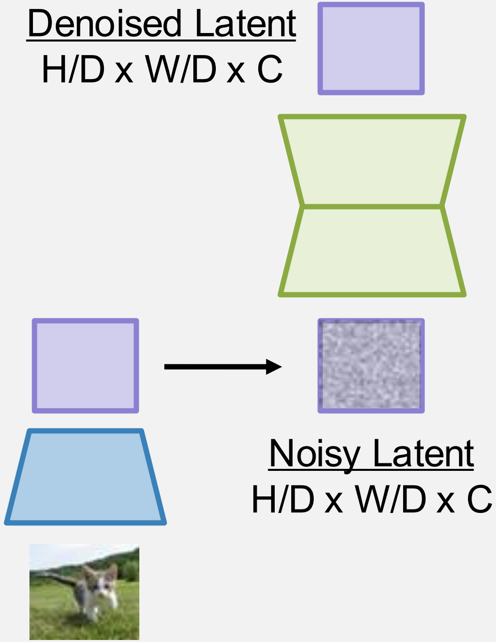
Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**

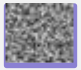


Train **diffusion model** to remove noise from **latents** (**Encoder** is frozen)



After training:

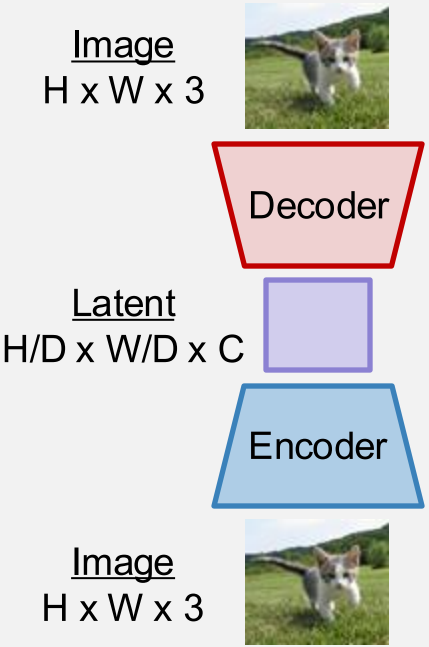
Sample random **latent**



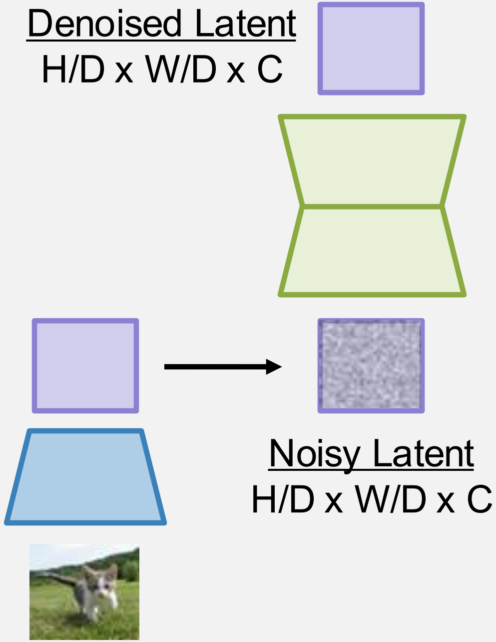
Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**



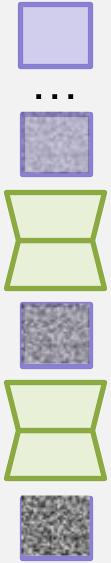
Train **diffusion model** to remove noise from **latents** (**Encoder** is frozen)



After training:

Sample random **latent**

Iteratively apply **diffusion model** to remove noise



Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**

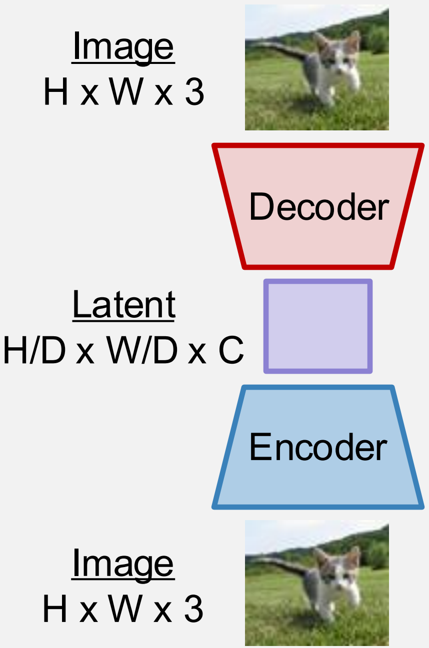


Image
H x W x 3

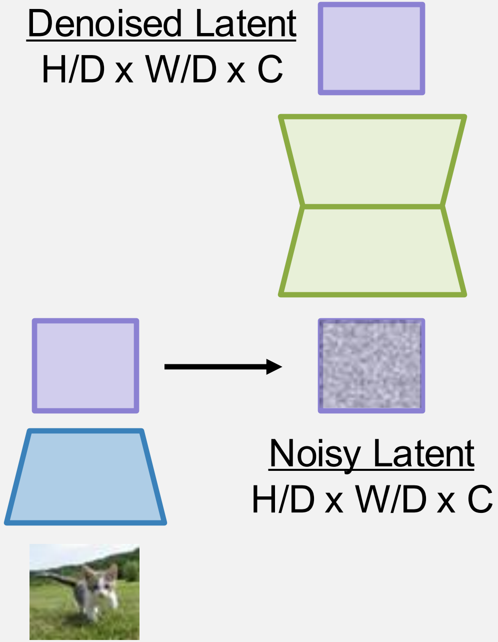
Decoder

Latent
H/D x W/D x C

Encoder

Image
H x W x 3

Train **diffusion model** to remove noise from **latents** (**Encoder** is frozen)


Denoised Latent
H/D x W/D x C

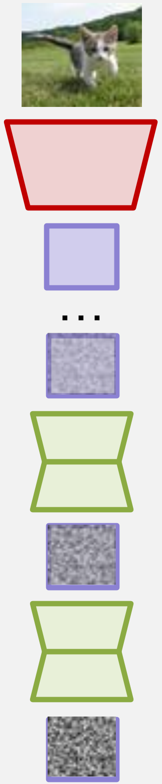
Noisy Latent
H/D x W/D x C

After training:

Sample random **latent**

Iteratively apply **diffusion model** to remove noise

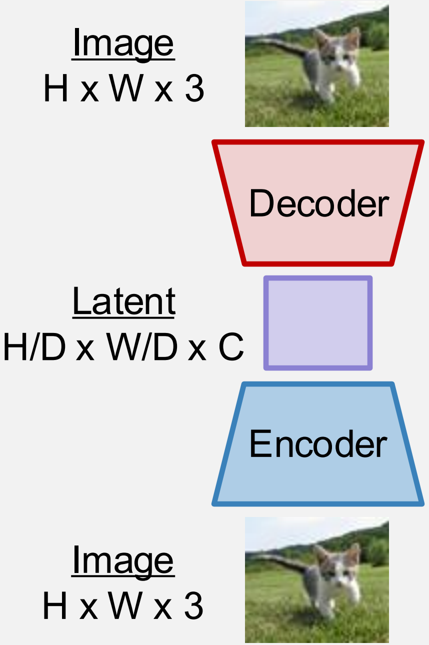
run **decoder** to get **image**



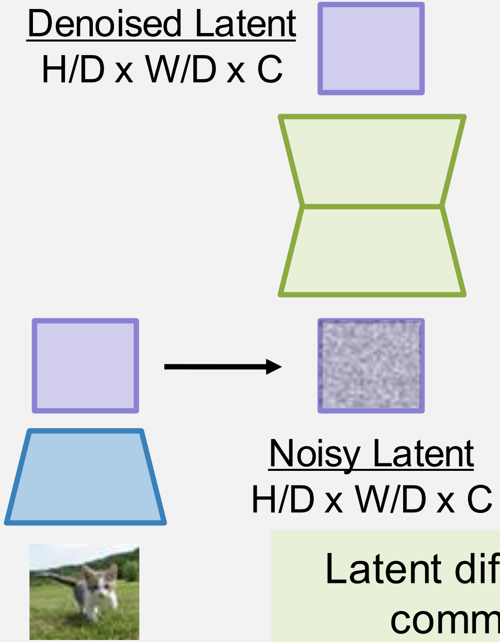
Rombach et al, “High-Resolution Image Synthesis with Latent Diffusion Models”, CVPR 2022

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**



Train **diffusion model** to remove noise from **latents** (**Encoder** is frozen)

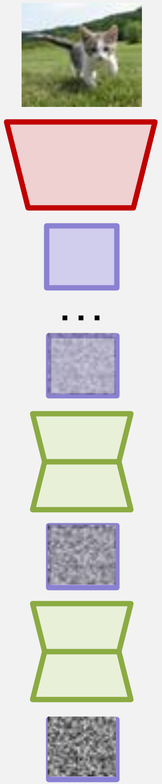


After training:

Sample random **latent**

Iteratively apply **diffusion model** to remove noise

run **decoder** to get **image**

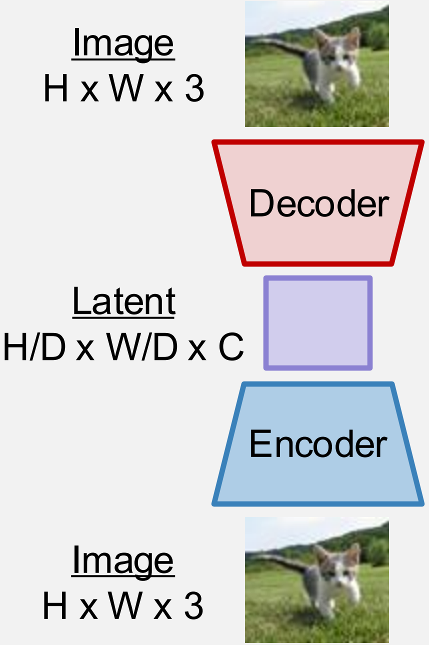


Latent diffusion is the most common form today

Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**

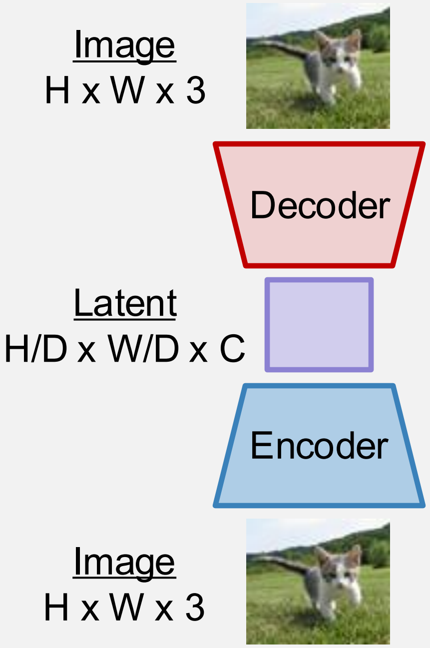


How do we train the encoder+decoder?

Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**

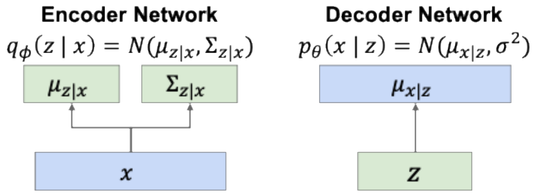


How do we train the encoder+decoder?

Solution: It's a VAE! Typically with very small KL prior weight

Recall: VAE

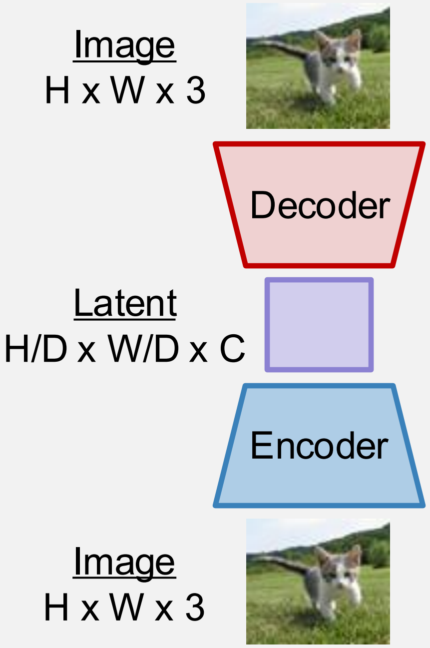
$$\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$



Rombach et al, “High-Resolution Image Synthesis with Latent Diffusion Models”, CVPR 2022

Latent Diffusion Models (LDMs)

Train **encoder** + **decoder** to convert images to **latents**



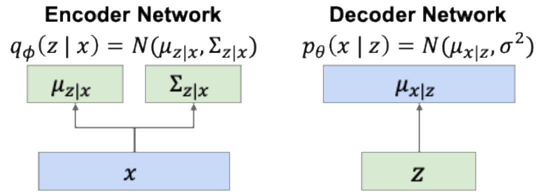
How do we train the encoder+decoder?

Solution: It’s a VAE! Typically with very small KL prior weight

Problem: Decoder outputs often blurry

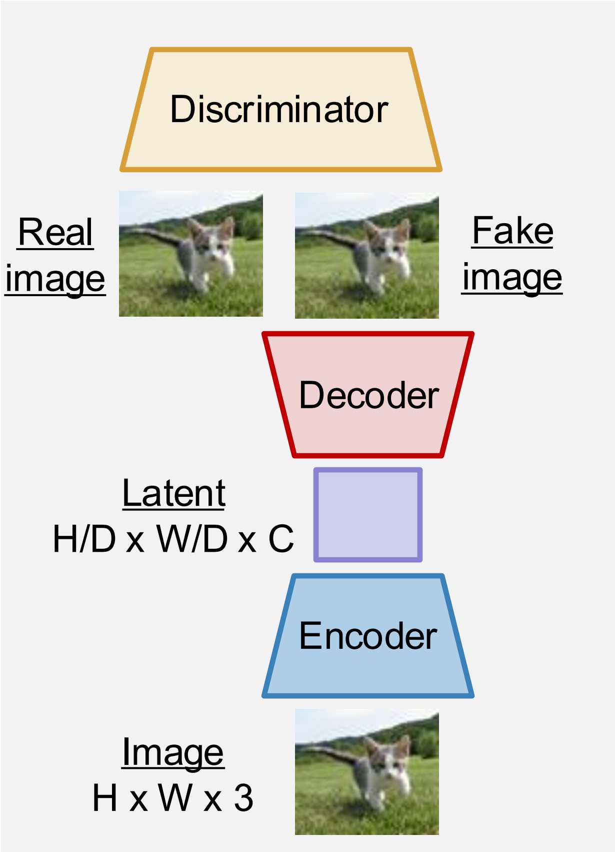
Recall: VAE

$$\log p_\theta(x) \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$



Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

Latent Diffusion Models (LDMs)



How do we train the encoder+decoder?

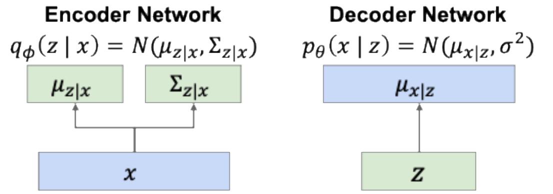
Solution: It's a VAE! Typically with very small KL prior weight

Problem: Decoder outputs often blurry

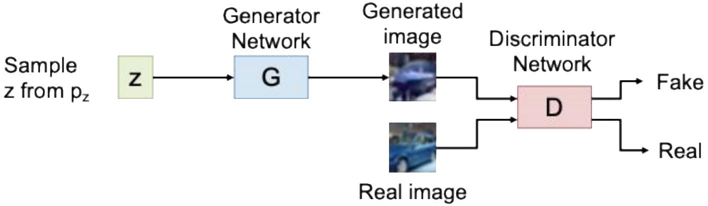
Solution: Add a discriminator!

Recall: VAE

$$\log p_\theta(x) \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

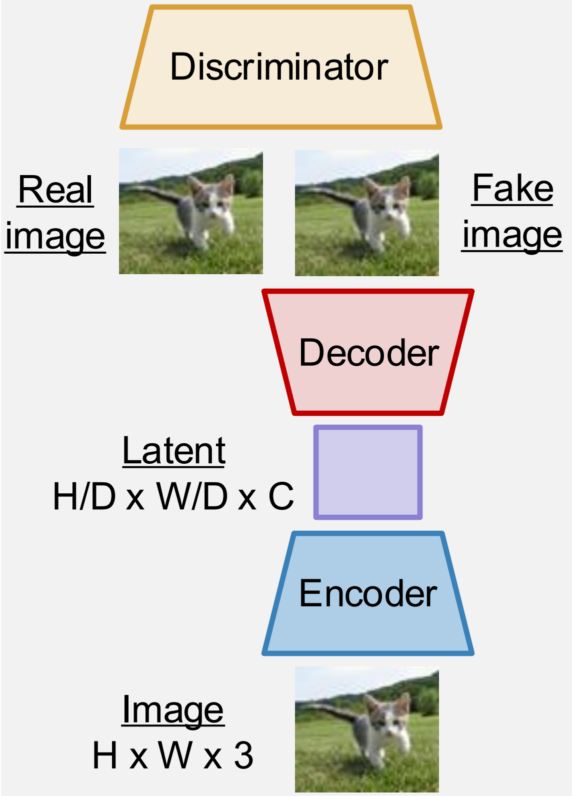


Recall: GAN

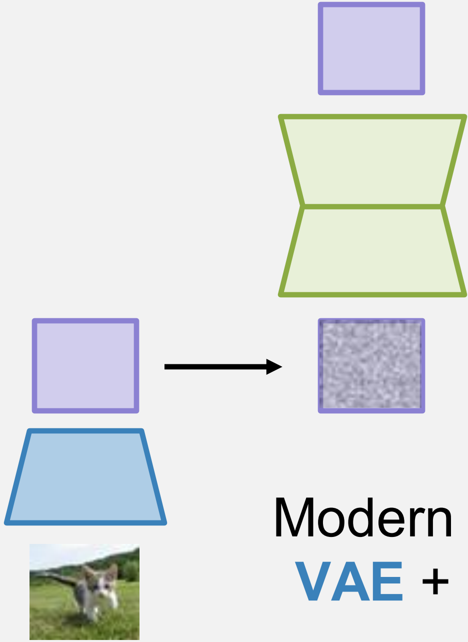


Rombach et al, "High-Resolution Image Synthesis with Latent Diffusion Models", CVPR 2022

Latent Diffusion Models (LDMs)



Train **diffusion model** to remove noise from **latents** (**Encoder** is frozen)

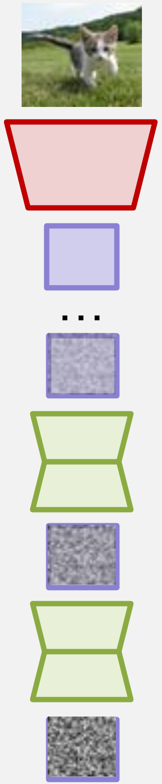


After training:

Sample random **latent**

Iteratively apply **diffusion model** to remove noise

run **decoder** to get **image**

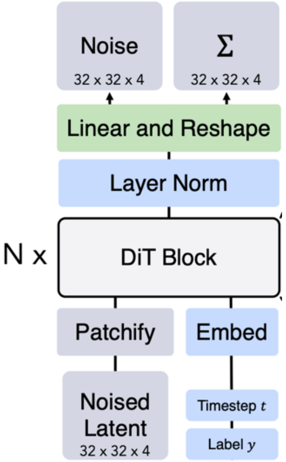


Modern LDM pipelines use **VAE** + **GAN** + **diffusion**!

Peebles and Xie, "Scalable Diffusion Models with Transformer", ICCV 2023

Diffusion Transformer (DiT)

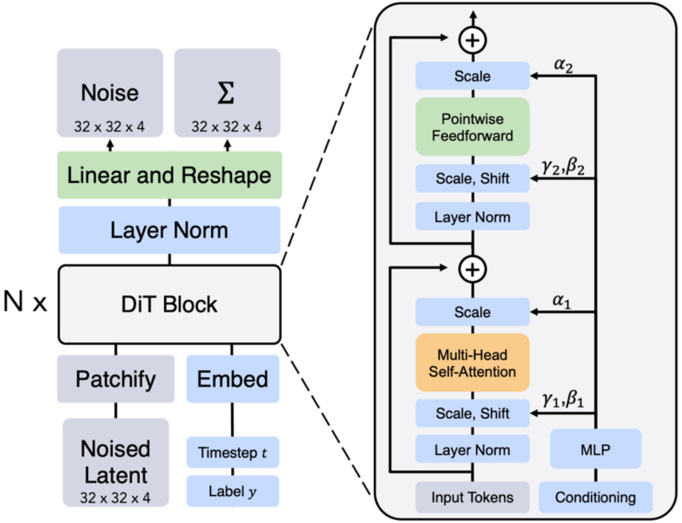
Diffusion uses standard Transformer blocks!
Main question: How to inject conditioning (timestep t, text, …)



Peebles and Xie, "Scalable Diffusion Models with Transformer", ICCV 2023

Diffusion Transformer (DiT)

Diffusion uses standard Transformer blocks!
Main question: How to inject conditioning (timestep t, text, …)

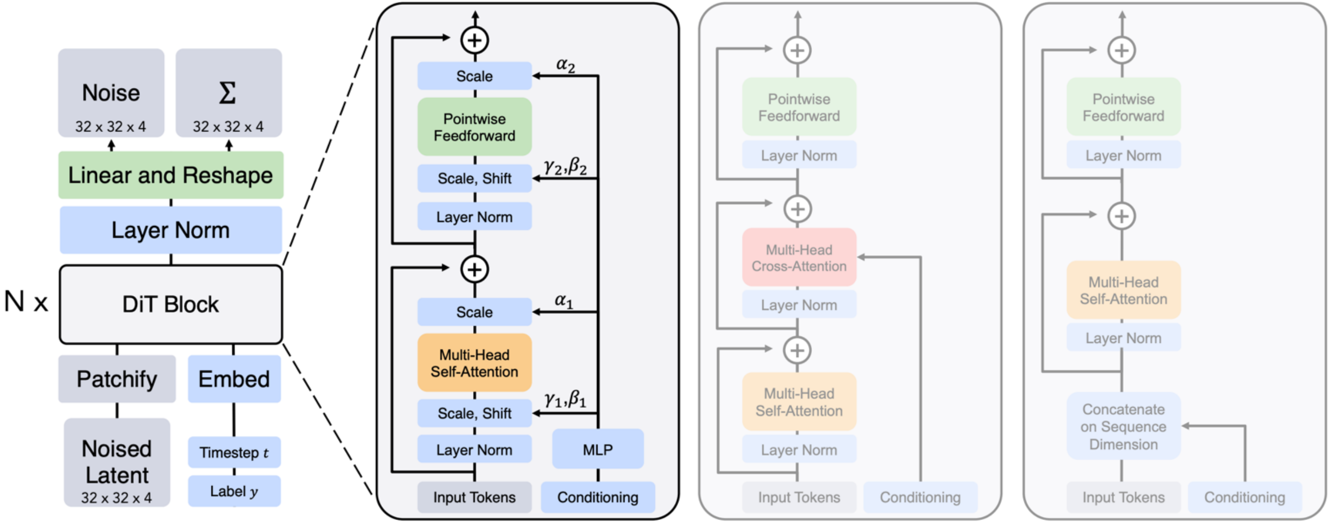


Predict scale/shift:
Most common for diffusion timestep t

Peebles and Xie, "Scalable Diffusion Models with Transformer", ICCV 2023

Diffusion Transformer (DiT)

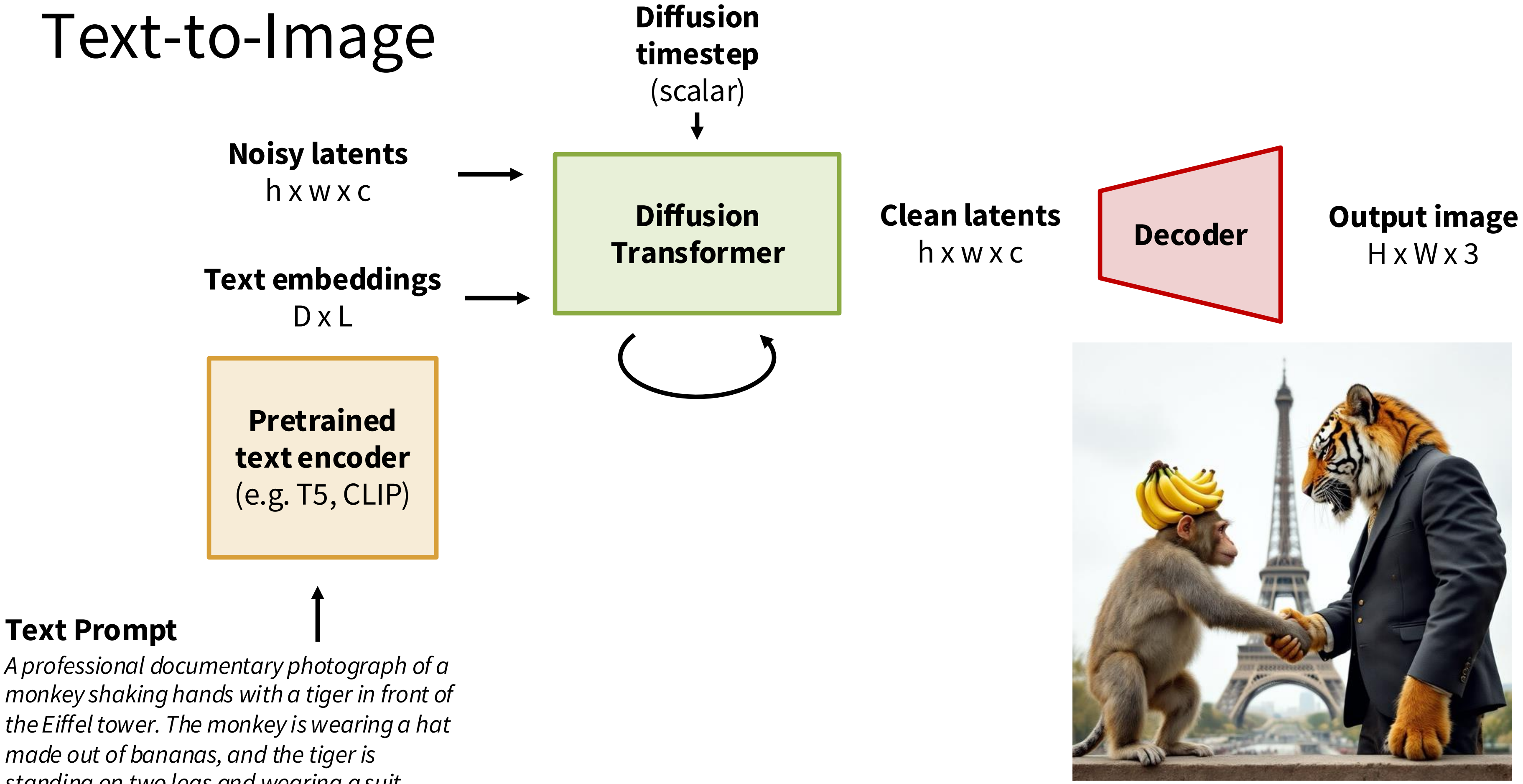
Diffusion uses standard Transformer blocks!
Main question: How to inject conditioning (timestep t, text, …)



Predict scale/shift: Most common for diffusion timestep t

Cross-Attention / Joint Attention: Common for text, image, etc conditioning

Text-to-Image



Diffusion timestep
(scalar)

Noisy latents
h x w x c

Text embeddings
D x L

Diffusion Transformer

Clean latents
h x w x c

Decoder

Output image
H x W x 3

Pretrained text encoder
(e.g. T5, CLIP)

Text Prompt

A professional documentary photograph of a monkey shaking hands with a tiger in front of the Eiffel tower. The monkey is wearing a hat made out of bananas, and the tiger is standing on two legs and wearing a suit.

Text-to-Image

Diffusion timestep
(scalar)

Noisy latents
128 x 128 x 16

Text embeddings
D x L

Diffusion Transformer

Clean latents
128 x 128 x 16

Decoder

Output image
1024 x 1024 x 3

Pretrained text encoder
(e.g. T5, CLIP)

Text Prompt
A professional documentary photograph of a monkey shaking hands with a tiger in front of the Eiffel tower. The monkey is wearing a hat made out of bananas, and the tiger is standing on two legs and wearing a suit.

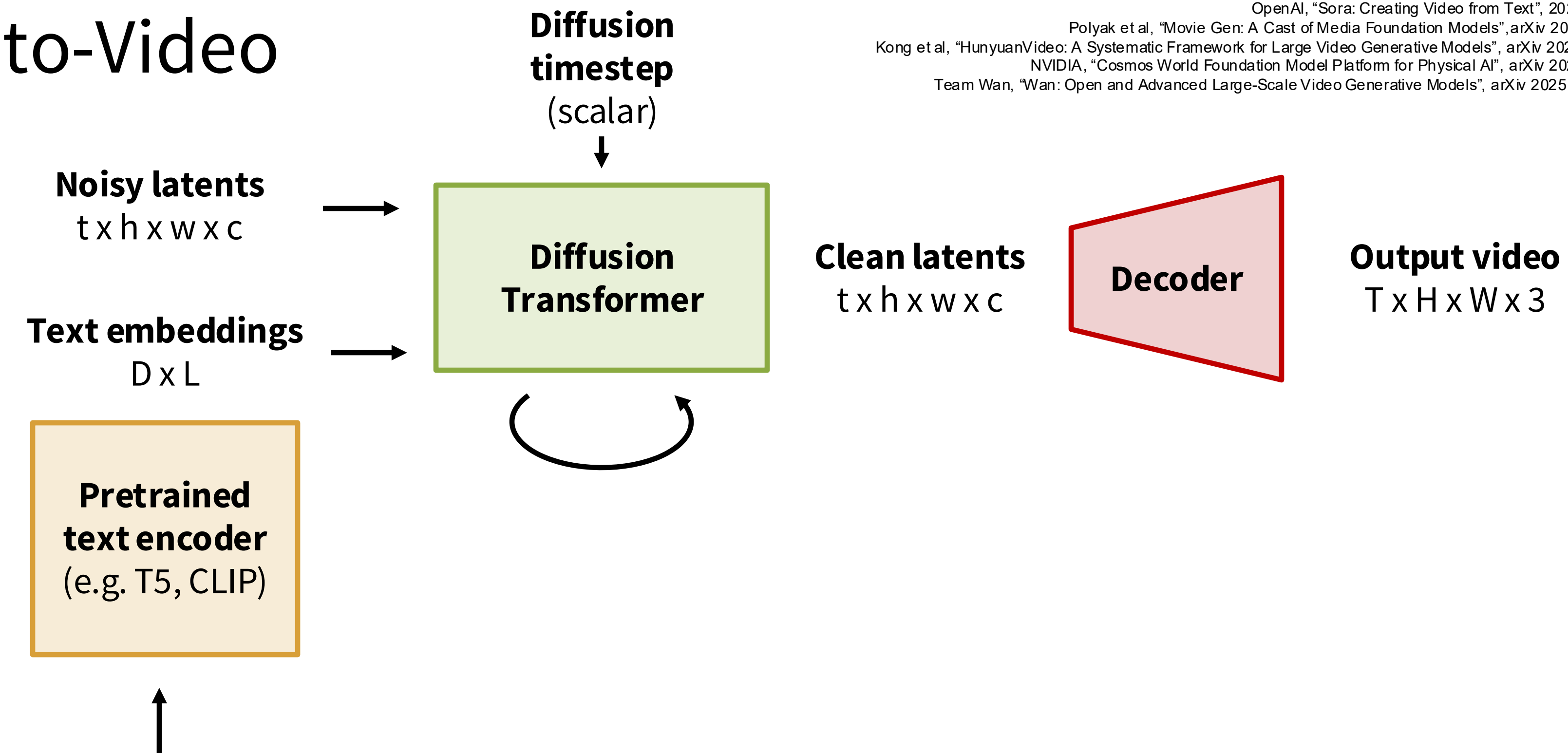
Example: FLUX.1 [dev]

Text Encoder: T5 + CLIP
Encoder/Decoder: 8x8 downsampling
Diffusion model: 12B parameter model
2x2 patchify => 64x64 = 1024 image tokens

https://github.com/black-forest-labs/flux

Text-to-Video

Gupta et al, "Photorealistic Video Generation with Diffusion Models", arXiv 2023 (Dec)
OpenAI, "Sora: Creating Video from Text", 2024 (Feb)
Polyak et al, "Movie Gen: A Cast of Media Foundation Models",arXiv 2024 (Oct)
Kong et al, "HunyuanVideo: A Systematic Framework for Large Video Generative Models", arXiv 2024 (Dec)
NVIDIA, "Cosmos World Foundation Model Platform for Physical AI", arXiv 2025 (Jan)
Team Wan, "Wan: Open and Advanced Large-Scale Video Generative Models", arXiv 2025 (March)



Diffusion timestep
(scalar)

Noisy latents
t x h x w x c

Text embeddings
D x L

Diffusion Transformer

Clean latents
t x h x w x c

Decoder

Output video
T x H x W x 3

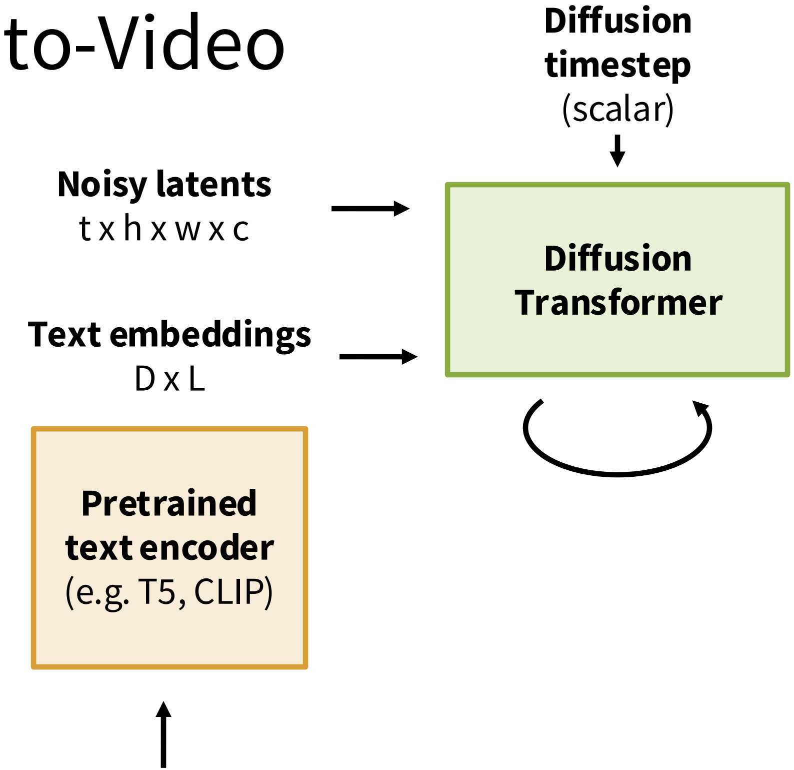
Pretrained text encoder
(e.g. T5, CLIP)

Text Prompt

A red-faced monkey with white fur is bathing in a natural hot spring. The monkey is playing in the water with a miniature sail ship in front of it, made of wood with a white sail and a small rudder. The hot spring is surrounded by lush greenery, with rocks and trees.

Text-to-Video

Gupta et al, "Photorealistic Video Generation with Diffusion Models", arXiv 2023 (Dec)
OpenAI, "Sora: Creating Video from Text", 2024 (Feb)
Polyak et al, "Movie Gen: A Cast of Media Foundation Models", arXiv 2024 (Oct)
Kong et al, "HunyuanVideo: A Systematic Framework for Large Video Generative Models", arXiv 2024 (Dec)
NVIDIA, "Cosmos World Foundation Model Platform for Physical AI", arXiv 2025 (Jan)
Team Wan, "Wan: Open and Advanced Large-Scale Video Generative Models", arXiv 2025 (March)



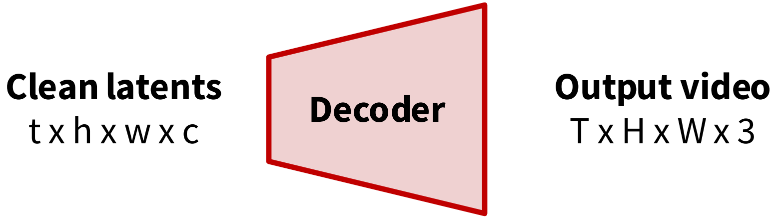
Diffusion timestep
(scalar)

Noisy latents
t x h x w x c

Diffusion Transformer

Text embeddings
D x L

Pretrained text encoder
(e.g. T5, CLIP)



Clean latents
t x h x w x c

Decoder

Output video
T x H x W x 3

Text Prompt

A red-faced monkey with white fur is bathing in a natural hot spring. The monkey is playing in the water with a miniature sail ship in front of it, made of wood with a white sail and a small rudder. The hot spring is surrounded by lush greenery, with rocks and trees.

Video from Meta Movie Gen
(https://ai.meta.com/research/movie-gen/)

Text-to-Video

Gupta et al, "Photorealistic Video Generation with Diffusion Models", arXiv 2023 (Dec)
OpenAI, "Sora: Creating Video from Text", 2024 (Feb)
Polyak et al, "Movie Gen: A Cast of Media Foundation Models", arXiv 2024 (Oct)
Kong et al, "HunyuanVideo: A Systematic Framework for Large Video Generative Models", arXiv 2024 (Dec)
NVIDIA, "Cosmos World Foundation Model Platform for Physical AI", arXiv 2025 (Jan)
Team Wan, "Wan: Open and Advanced Large-Scale Video Generative Models", arXiv 2025 (March)

Diffusion timestep (scalar)

Noisy latents
32 x 128 x 72 x 16

Text embeddings
D x L

Diffusion Transformer

Clean latents
33 x 128 x 72 x 16

Decoder

Output video
257 x 1024 x 576 x 3

Pretrained text encoder (e.g. T5, CLIP)

Text Prompt

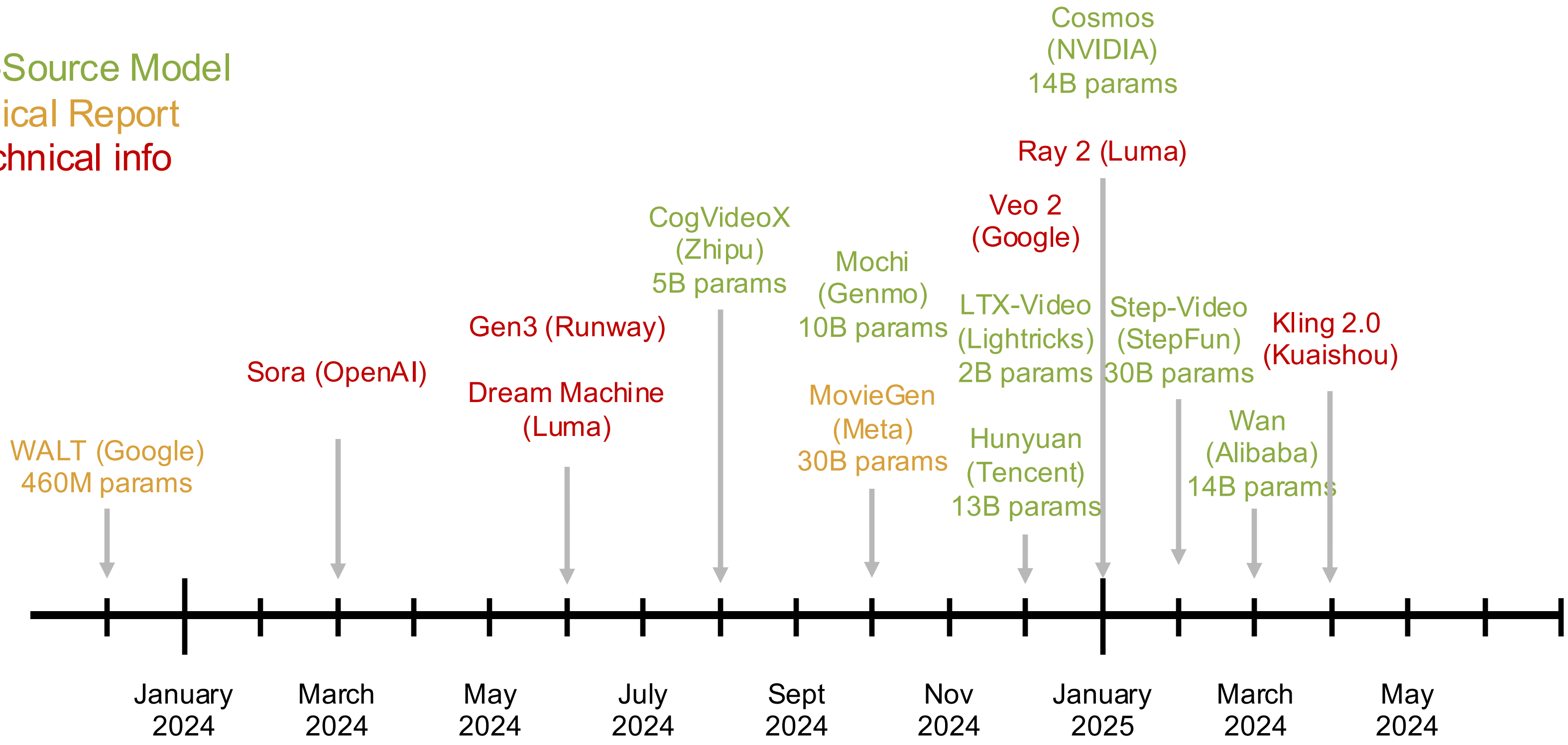
A red-faced monkey with white fur is bathing in a natural hot spring. The monkey is playing in the water with a miniature sail ship in front of it, made of wood with a white sail and a small rudder. The hot spring is surrounded by lush greenery, with rocks and trees.

Example: Meta MovieGen

Text Encoder: UL2, ByT5, MetaCLIP
Encoder/Decoder: 8x8x8 downsample
Diffusion model: 30B param DiT
1x2x2 patchify => 76K tokens

The Era of Video Diffusion Models

Open-Source Model
Technical Report
No technical info



WALT (Google)
460M params

Sora (OpenAI)

Gen3 (Runway)

Dream Machine
(Luma)

CogVideoX
(Zhipu)
5B params

Mochi
(Genmo)
10B params

MovieGen
(Meta)
30B params

Veo 2
(Google)

LTX-Video
(Lightricks)
2B params

Hunyuan
(Tencent)
13B params

Cosmos
(NVIDIA)
14B params

Ray 2 (Luma)

Step-Video
(StepFun)
30B params

Wan
(Alibaba)
14B params

Kling 2.0
(Kuaishou)

January
2024

March
2024

May
2024

July
2024

Sept
2024

Nov
2024

January
2025

March
2024

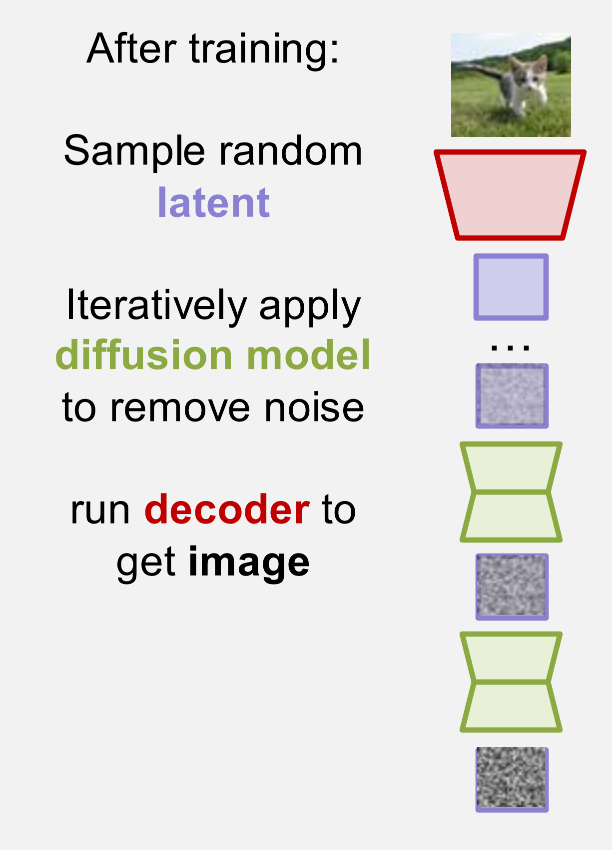
May
2024

Gupta et al, "Photorealistic Video Generation with Diffusion Models", arXiv 2023 (Dec)
OpenAI, "Sora: Creating Video from Text", 2024 (Feb)
Polyak et al, "Movie Gen: A Cast of Media Foundation Models", arXiv 2024 (Oct)
Kong et al, "HunyuanVideo: A Systematic Framework for Large Video Generative Models", arXiv 2024 (Dec)
NVIDIA, "Cosmos World Foundation Model Platform for Physical AI", arXiv 2025 (Jan)
Team Wan, "Wan: Open and Advanced Large-Scale Video Generative Models", arXiv 2025 (March)

Diffusion Distillation

During sampling we need to run the diffusion model many times (~30 – 50 for rectified flow)

This is really slow!



After training:

Sample random **latent**

Iteratively apply **diffusion model** to remove noise

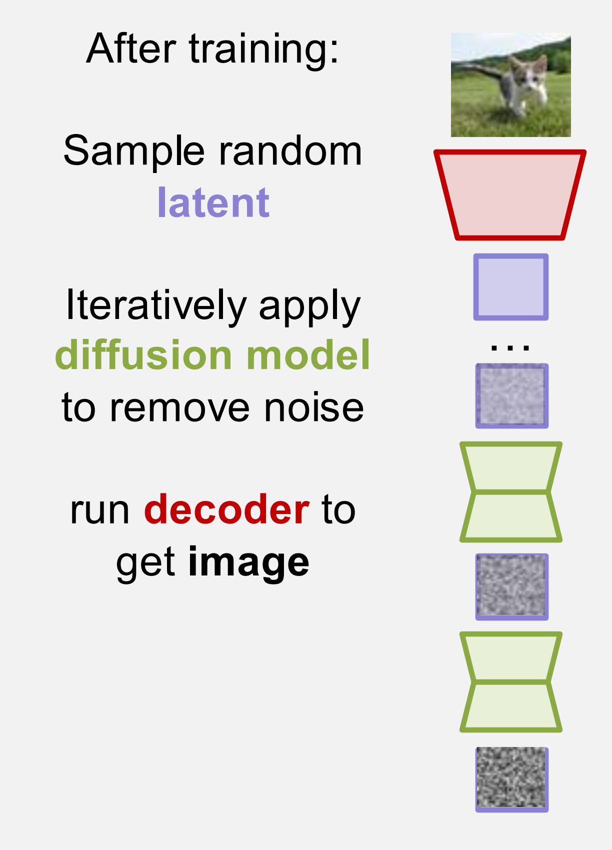
run **decoder** to get **image**

Diffusion Distillation

During sampling we need to run the diffusion model many times (~30 – 50 for rectified flow)

This is really slow!

Solution: **distillation** algorithms reduce the number of steps (sometimes all the way to 1), can also bake in CFG


After training:

Sample random **latent**

Iteratively apply **diffusion model** to remove noise

run **decoder** to get **image**

Salimans and Ho, "Progressive Distillation for Fast Sampling of Diffusion Models", ICLR 2022
Song et al, "Consistency Models", ICML 2023
Sauer et al, "Adversarial Diffusion Distillation", ECCV 2024
Sauer et al, "Fast High-Resolution Image Synthesis with Latent Adversarial Diffusion Distillation", arXiv 2024
Lu and Song, "Simplifying, Stabilizing and Scaling Consistency Models", ICLR 2025
Salimans et al, "Multistep Distillation of Diffusion Models via Moment Matching", NeurIPS 2025

Generalized Diffusion

Rectified Flow

Sample $x \sim p_{data}$, $z \sim p_{noise}$
Sample $t \sim p_t$
Set $x_t = (1 - t)x + tz$
Set $v_{gt} = z - x$
Compute $v_{pred} = f_\theta(x_t, t)$
Compute loss $\left\| v_{gt} - v_{pred} \right\|_2^2$

Generalized Diffusion

Rectified Flow

Sample $x \sim p_{data}, z \sim p_{noise}$
Sample $t \sim p_t$
Set $x_t = (1-t)x + tz$
Set $v_{gt} = z - x$
Compute $v_{pred} = f_\theta(x_t, t)$
Compute loss $\|v_{gt} - v_{pred}\|_2^2$

→

Generalized Diffusion

Sample $x \sim p_{data}, z \sim p_{noise}$
Sample $\boldsymbol{t} \sim p_t$
Set $x_t = \boldsymbol{a}(\boldsymbol{t})x + \boldsymbol{b}(\boldsymbol{t})z$
Set $y_{gt} = \boldsymbol{c}(\boldsymbol{t})x + \boldsymbol{d}(\boldsymbol{t})z$
Compute $y_{pred} = f_\theta(x_t, \boldsymbol{t})$
Compute loss $\|y_{gt} - y_{pred}\|_2^2$

Generalized Diffusion

Rectified Flow

Sample $x \sim p_{data}, z \sim p_{noise}$
Sample $t \sim p_t$
Set $x_t = (1 - t)x + tz$
Set $v_{gt} = z - x$
Compute $v_{pred} = f_\theta(x_t, t)$
Compute loss $\left\| v_{gt} - v_{pred} \right\|_2^2$

$$\boldsymbol{a}(\boldsymbol{t}) = 1 - t$$
$$\boldsymbol{b}(\boldsymbol{t}) = t$$
$$\boldsymbol{c}(\boldsymbol{t}) = -1$$
$$\boldsymbol{d}(\boldsymbol{t}) = 1$$

→

Generalized Diffusion

Sample $x \sim p_{data}, z \sim p_{noise}$
Sample $\boldsymbol{t} \sim p_t$
Set $x_t = \boldsymbol{a}(\boldsymbol{t})x + \boldsymbol{b}(\boldsymbol{t})z$
Set $y_{gt} = \boldsymbol{c}(\boldsymbol{t})x + \boldsymbol{d}(\boldsymbol{t})z$
Compute $y_{pred} = f_\theta(x_t, \boldsymbol{t})$
Compute loss $\left\| y_{gt} - y_{pred} \right\|_2^2$

Generalized Diffusion

Variance Preserving (VP)

$$\boldsymbol{a}(\boldsymbol{t}) = \sqrt{\sigma(t)}$$

$$\boldsymbol{b}(\boldsymbol{t}) = \sqrt{1 - \sigma(t)}$$

If x and z are independent and variance=1, then x_t also has variance=1

Generalized Diffusion

Sample $x \sim p_{data}$, $z \sim p_{noise}$

Sample $\boldsymbol{t} \sim p_t$

Set $x_t = \boldsymbol{a}(\boldsymbol{t})x + \boldsymbol{b}(\boldsymbol{t})z$

Set $y_{gt} = \boldsymbol{c}(\boldsymbol{t})x + \boldsymbol{d}(\boldsymbol{t})z$

Compute $y_{pred} = f_\theta(x_t, \boldsymbol{t})$

Compute loss $\|y_{gt} - y_{pred}\|_2^2$

Song et al, "Score-Based Generative Modeling through Stochastic Differential Equations", ICLR 2021

Generalized Diffusion

Variance Exploding (VE)

$\boldsymbol{a}(\boldsymbol{t}) = 1$
$\boldsymbol{b}(\boldsymbol{t}) = \sigma(t)$

$\sigma(1)$ Needs to be big enough to drown out all signal in x

Generalized Diffusion

Sample $x \sim p_{data}, z \sim p_{noise}$
Sample $\boldsymbol{t} \sim p_t$
Set $x_t = \boldsymbol{a}(\boldsymbol{t})x + \boldsymbol{b}(\boldsymbol{t})z$
Set $y_{gt} = \boldsymbol{c}(\boldsymbol{t})x + \boldsymbol{d}(\boldsymbol{t})z$
Compute $y_{pred} = f_\theta(x_t, \boldsymbol{t})$
Compute loss $\left\|y_{gt} - y_{pred}\right\|_2^2$

Song et al, "Score-Based Generative Modeling through Stochastic Differential Equations", ICLR 2021
Karras et al, "Elucidating the Design Space of Diffusion-Based Generative Models", NeurIPS 2022

Generalized Diffusion

x-prediction

$$y_{gt} = x \quad [\boldsymbol{c}(\boldsymbol{t}) = 1;\ \boldsymbol{d}(\boldsymbol{t}) = 0]$$

ε-prediction

$$y_{gt} = \boldsymbol{z} \quad [\boldsymbol{c}(\boldsymbol{t}) = 0;\ \boldsymbol{d}(\boldsymbol{t}) = 1]$$

v-prediction

$$y_{gt} = \boldsymbol{b}(\boldsymbol{t})z - \boldsymbol{a}(\boldsymbol{t})x \qquad [\boldsymbol{c}(\boldsymbol{t}) = \boldsymbol{b}(\boldsymbol{t});\ \boldsymbol{d}(\boldsymbol{t}) = -\boldsymbol{a}(\boldsymbol{t})]$$

Generalized Diffusion

Sample $x \sim p_{data}, z \sim p_{noise}$
Sample $\boldsymbol{t} \sim p_t$
Set $x_t = \boldsymbol{a}(\boldsymbol{t})x + \boldsymbol{b}(\boldsymbol{t})z$
Set $y_{gt} = \boldsymbol{c}(\boldsymbol{t})x + \boldsymbol{d}(\boldsymbol{t})z$
Compute $y_{pred} = f_\theta(x_t, \boldsymbol{t})$
Compute loss $\left\| y_{gt} - y_{pred} \right\|_2^2$

Salimans and Ho, "Progressive Distillation of Diffusion Models", ICLR 2022
Ho et al, "Imagen Video: High Definition Video Generation with Diffusion Models", arXiv 2022

Generalized Diffusion

How do we choose these functions?

Usually through some **mathematical formalism**

Generalized Diffusion

Sample $x \sim p_{data}, z \sim p_{noise}$

Sample $\boldsymbol{t} \sim p_t$

Set $x_t = \boldsymbol{a}(\boldsymbol{t})x + \boldsymbol{b}(\boldsymbol{t})z$

Set $y_{gt} = \boldsymbol{c}(\boldsymbol{t})x + \boldsymbol{d}(\boldsymbol{t})z$

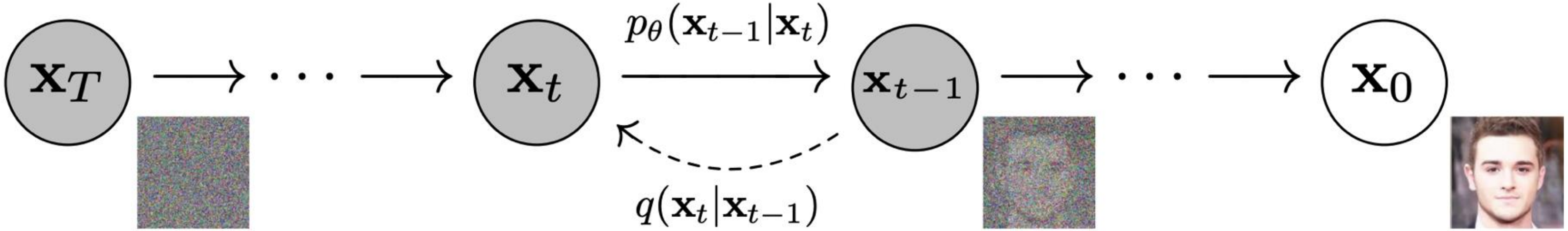
Compute $y_{pred} = f_\theta(x_t, \boldsymbol{t})$

Compute loss $\left\| y_{gt} - y_{pred} \right\|_2^2$

Salimans and Ho, "Progressive Distillation of Diffusion Models", ICLR 2022
Ho et al, "Imagen Video: High Definition Video Generation with Diffusion Models", arXiv 2022

Diffusion is a Latent Variable Model

We know the forward process: Add Gaussian noise



Learn a network to approximate the backward process

Optimize variational lower bound (same as VAE)

Sohl-Dickstein et al, "Deep Unsupervised Learning using Nonequilibrium Thermodynamics", NeurIPS 2015
Figure from Ho et al, "Denoising Diffusion Probabilistic Models", NeurIPS 2020

Diffusion Learns the Score Function

For any distribution $p(x)$ over $x \in \mathbb{R}^N$ the **score function**

$$s: \mathbb{R}^N \to \mathbb{R}^N \qquad s(x) = \frac{\partial}{\partial x} \log p(x)$$

Is a vector field pointing toward areas of high probability density

Diffusion learns a neural network to approximate the score function of p_{data}

Song and Ermon, "Generative Modeling by Estimating Gradients of the Data Distribution", NeurIPS 2019
Ho et al, "Denoising Diffusion Probabilistic Models", NeurIPS 2020

Diffusion Solves Stochastic Differential Equations

We can describe a continuous noising process as an SDE

$$d\boldsymbol{x} = f(\boldsymbol{x}, t)dt + g(t)d\boldsymbol{w}$$

Gives a relationship between infinitesimal changes in data x, time t, and noise w.

Diffusion learns a neural network to approximately solve this SDE

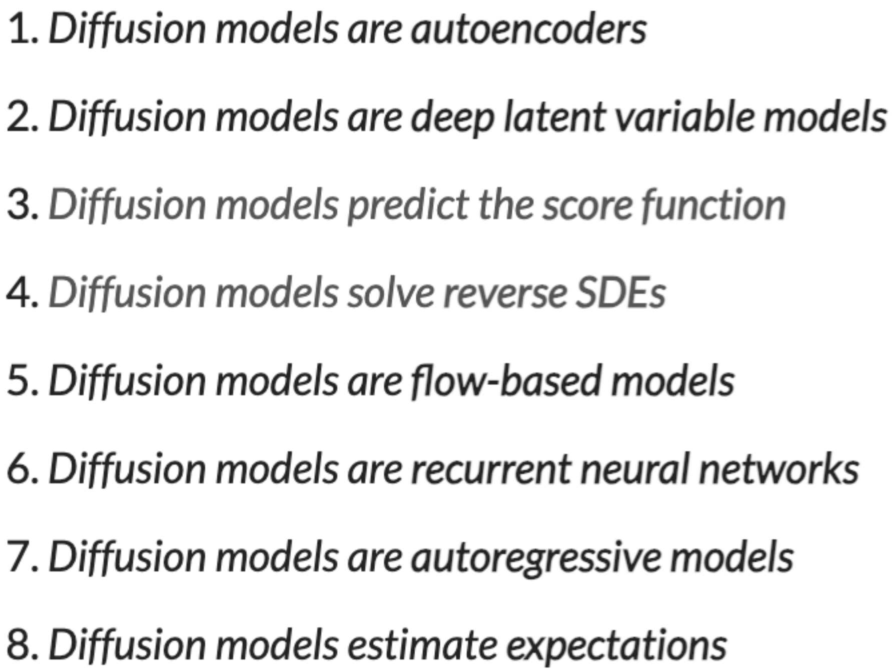
Song et al, "Score-Based Generative Modeling through Stochastic Differential Equations", ICLR 2021

Perspectives on Diffusion

Great blog post by Sander Dieleman:

https://sander.ai/2023/07/20/perspectives.html

(All his blog posts are great)

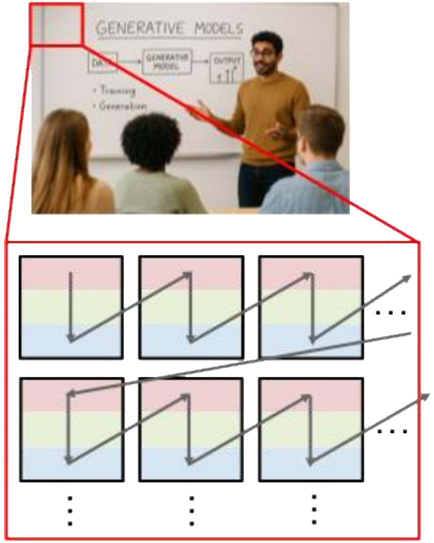


1. *Diffusion models are autoencoders*
2. *Diffusion models are deep latent variable models*
3. *Diffusion models predict the score function*
4. *Diffusion models solve reverse SDEs*
5. *Diffusion models are flow-based models*
6. *Diffusion models are recurrent neural networks*
7. *Diffusion models are autoregressive models*
8. *Diffusion models estimate expectations*

Autoregressive Models Strike Back

Recall autoregressive models

Too slow on raw pixels



They work great on (discrete) latents!

Autoregressive Models Strike Back

Train **encoder** + **decoder** to convert images to **discrete latents**

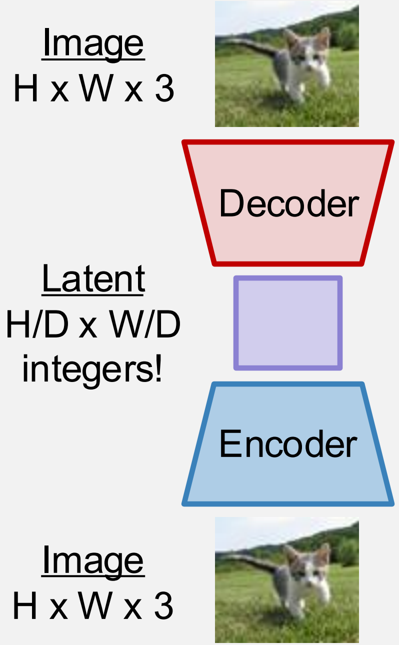


Image
H x W x 3

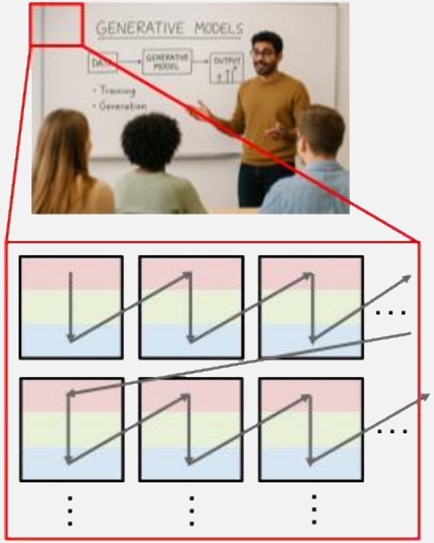
Decoder

Latent
H/D x W/D
integers!

Encoder

Image
H x W x 3

Train **autoregressive model** to model sequences of **discrete latents**

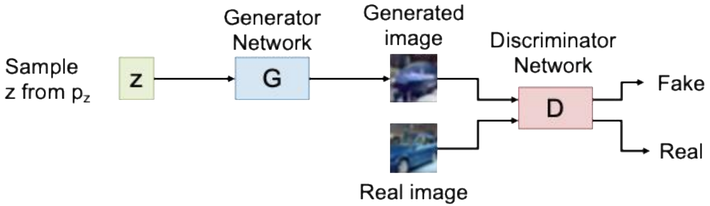


Sample **discrete latents** from the **autoregressive model**, pass to **decoder** to get an image

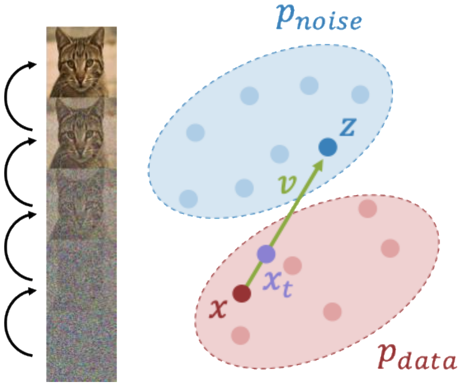
van den Oord et al, "Neural Discrete Representation Learning", NeurIPS 2017
Razavi et al, "Generating Diverse High-Fidelity Images with VQ-VAE-2", NeurIPS 2019
Esser et al, "Taming Transformers for High-Resolution Image Synthesis", CVPR 2021
Yu et al, "Scaling Autoregressive Models for Content-Rich Text-to-Image Generation", arXiv 2022

Summary

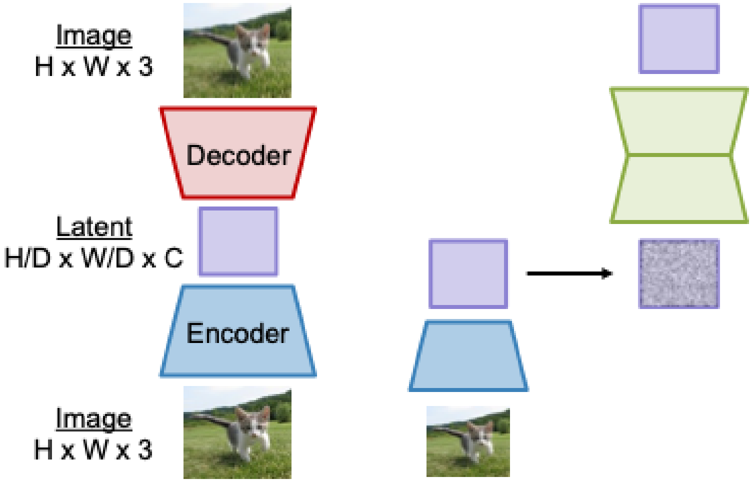
Generative Adversarial Networks



Diffusion Models



Latent Diffusion Models



Next Time:
Vision + Language