Data Generation with Diffusion Models

Andrew H. Fagg
Symbiotic Computing Laboratory
University of Oklahoma

Generative Adversarial Networks

- Formulated as a minimax problem: two competing networks with opposite objectives
  - Generator: translate a noise latent vector into an image
  - Discriminator: tell real from fake images
- Easy for one network to overtake the other & then never allow the other network to catch up
- Mode collapse: no matter the random input, generate the same output

Denoising Autoencoders

- Image to image translation technique
- Trained to remove noise from the input image
- Training set:
  - Corrupt each image & use as input
  - Use the original image as the target output

To what degree can we remove noise?

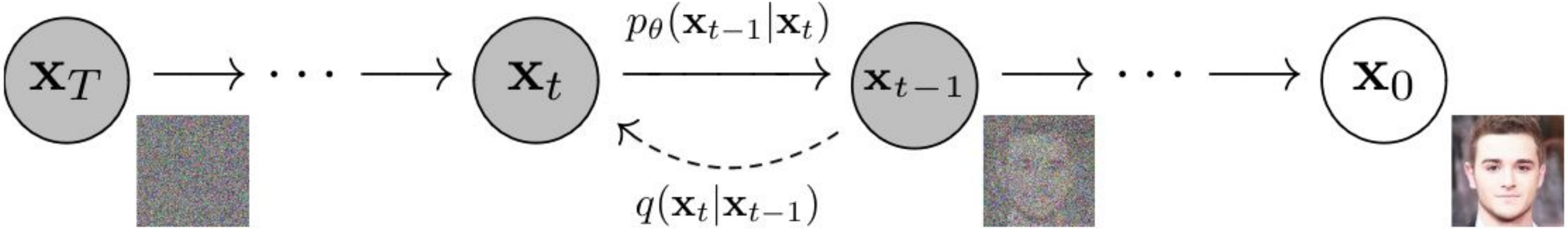
Denoising Diffusion Models

Ho, Jain & Abbeel (2020):

- Rather than removing all noise at once, we can remove the noise gradually over many steps
- Potential to remove a lot of noise
  - May even be able to start from an image that appears to contain only random noise

Denoising over Many Steps

- X_T: Start with very noisy image
- At each step, remove some amount of noise by sampling from $p(x_{t-1} \mid x_t)$
- We don't know this transformation – it must be learned!

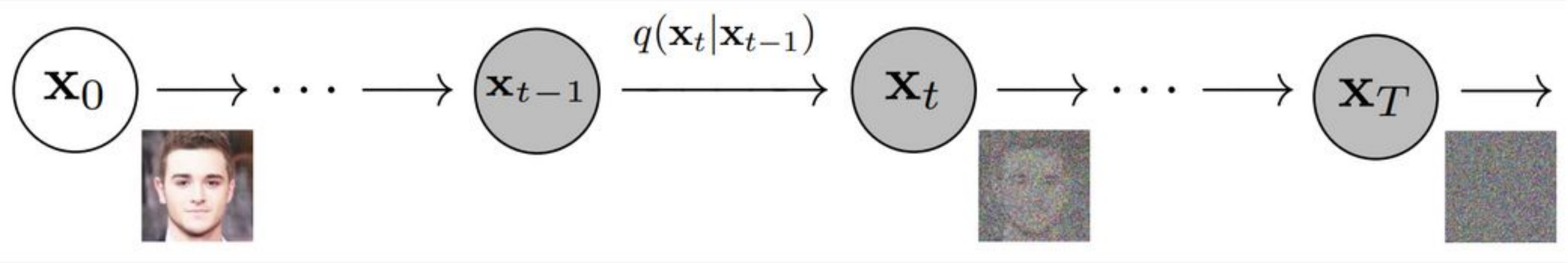


Ho, Jain & Abbeel (2020)

Constructing Training Data

Reverse the direction of the process

- Formulate as a Markov chain: the next step only depends on the previous step
- Model $q(x_t \mid x_{t-1})$ as a Gaussian distribution



https://www.assemblyai.com/blog/diffusion-models-for-machine-learning-introduction/

Constructing Training Data

Model $q(x_t \mid x_{t-1})$ as a Gaussian distribution:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

- x_t is a full image (each DOF is within +/- 1)
- Variance schedule: $\beta_1, \ldots, \beta_T$
  - Start small; becomes larger with each step. Always < 1
- Mean: original mean, but scaled toward zero
- Variance: no cross-terms

Constructing Training Data

Markov chain implies that x_t ONLY depends on x_{t-1} (and is independent of x_{t-2}, …)

When a single step is modeled as a Gaussian:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

we can model the likelihood of the entire sequence as a product of the individual steps:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

Constructing Training Data

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Key implication of this formulation:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

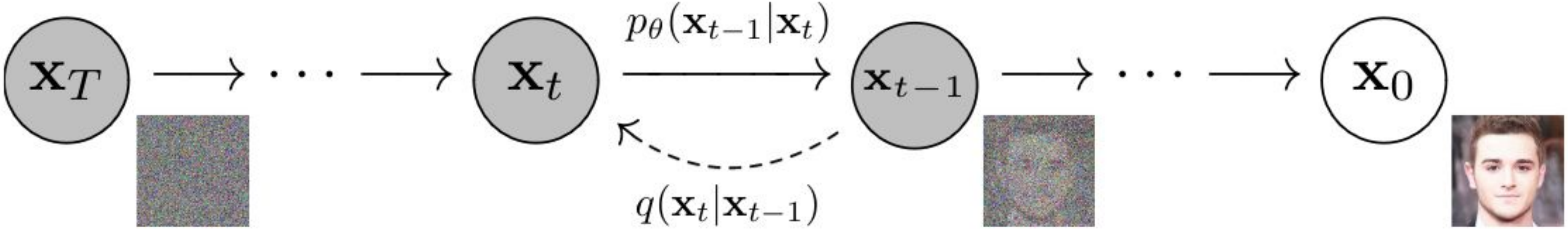
- The product of a subsequence can be computed in one shot:

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

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

- Easy to generate training samples with differing numbers of steps

Model



- Must learn the reverse conditional distribution
- Also model as a Gaussian:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

- The parameters of the Gaussian are learned functions

Model

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$$

- Simplifying assumption: covariance matrix is diagonal only with the same variance for every pixel
- Then, just need to estimate the mean as a function of x_t and t
  - Each pixel component has its own mean
- Implement using a U-net
  - This gives us some sharing of information across pixels

Training

- Noising process is a Gaussian (with fixed, defined parameters)
- Denoising process is also a Gaussian (learned means; fixed covariance)

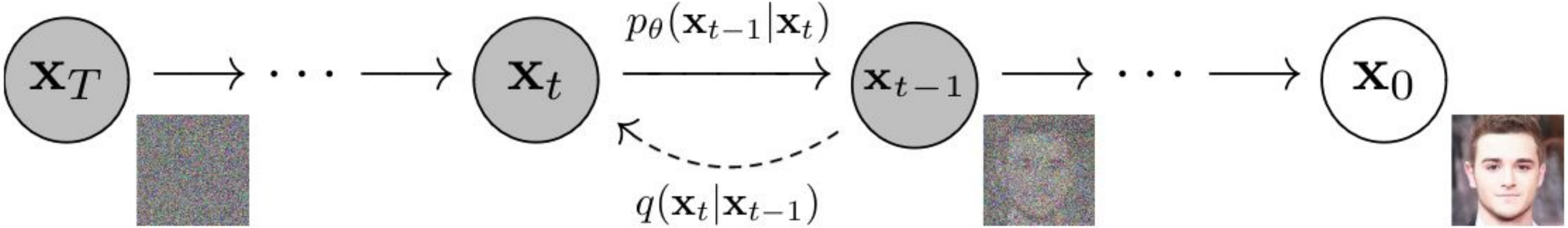
Loss function intuition: the forward and backward processes should produce the same distribution of images

- KL divergence can be used to compare the distributions
    - KL of two Gaussians has a closed-form solution!

Training

Minimize expected value:

$$\mathbb{E}_q\left[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\parallel p(\mathbf{x}_T))}_{L_T} + \sum_{t>1}\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}}\underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\right]$$



Training Algorithm

Repeat:

- Sample an image
- Randomly pick the number of steps (T)
- Sample x_t from a noise source
- Compute d L / d theta
- Update theta to reduce L

$$\mathbb{E}_q\left[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)\parallel p(\mathbf{x}_T))}_{L_T}+\sum_{t>1}\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}}\underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\right]$$

Sampling Process

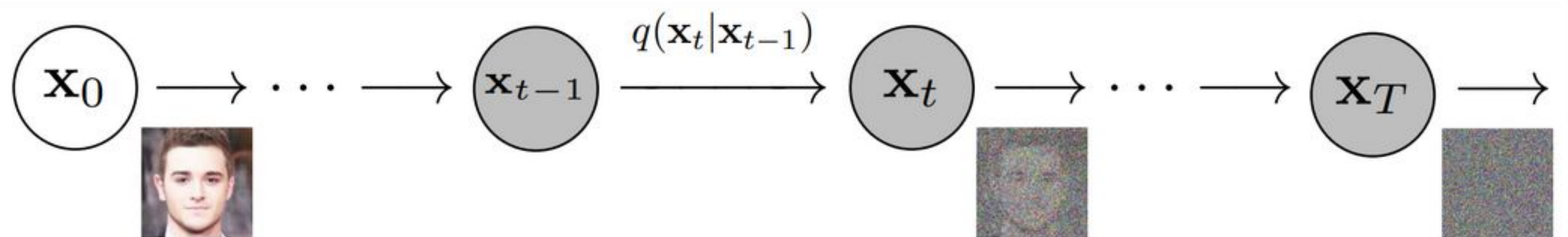
- We don’t have to sample all timesteps for a given image - we can just touch one for any given image!
- It is better to predict the noise given the input image and then subtract this noise out
  - As compared to predicting the cleaned up image directly

Noise Schedule

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

β_t: increase variance with time

- Want injected noise to be large enough by t=T such that the result is N(0, `I`)
- Provided code: increase linearly with t



Jumping Directly from 0 to t

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$

Key implication of this formulation:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

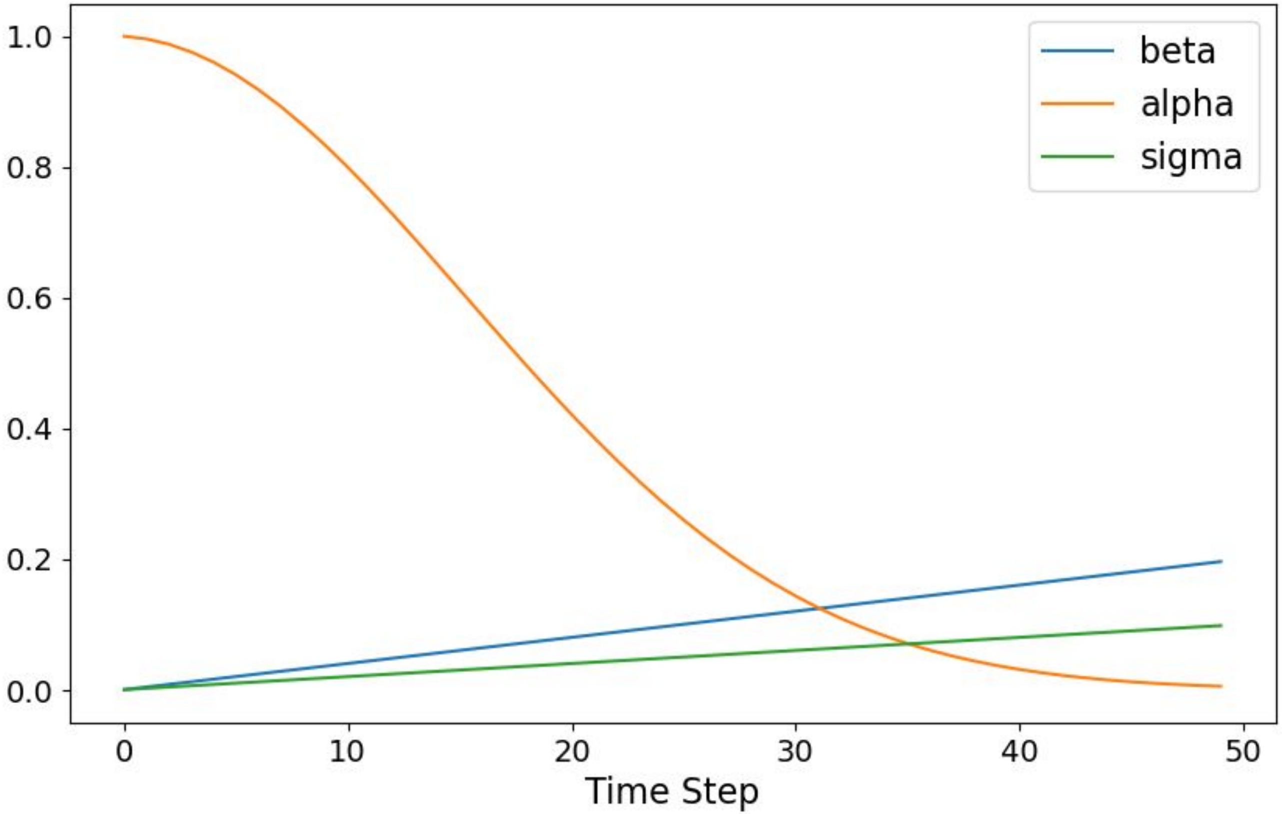
- The product of a subsequence can be computed in one shot:

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

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

- Easy to generate training samples with differing numbers of steps

Noise Scaling



Training (from book)

```
For each image x / label L in a batch:

    t ~ uniform([0, …, T-1])

    noise ~ N(0,I)

    x_noised = sqrt(a[t]) * x + sqrt(1 - a[t]) * noise
```

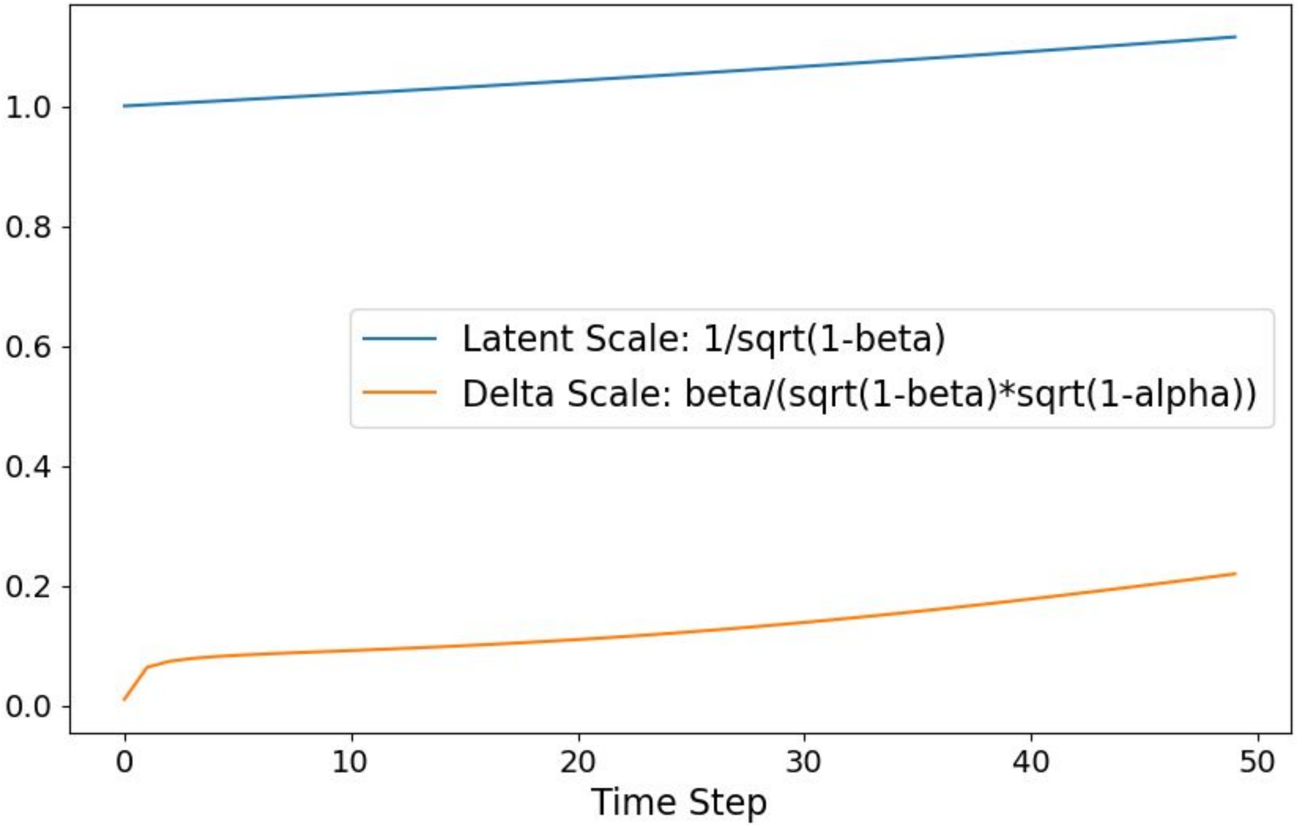
Want model g(x_noised, t, L) to predict the noise

- This becomes a “straightforward” supervised learning prob

Inference (from book)

```
z[T] ~ N(0,I)

for t in [T-1, T-2, … 0]:

    delta = model.predict(z[t+1], t, L)

    epsilon ~ N(0,I)

    z[t] = 1/sqrt(1-b[t]) * z[t+1]

           - (b[t]/[sqrt(1-a[t])sqrt(1-b[t])]) * delta

           + sigma * epsilon                    // not used in last step
```

Scale Factors

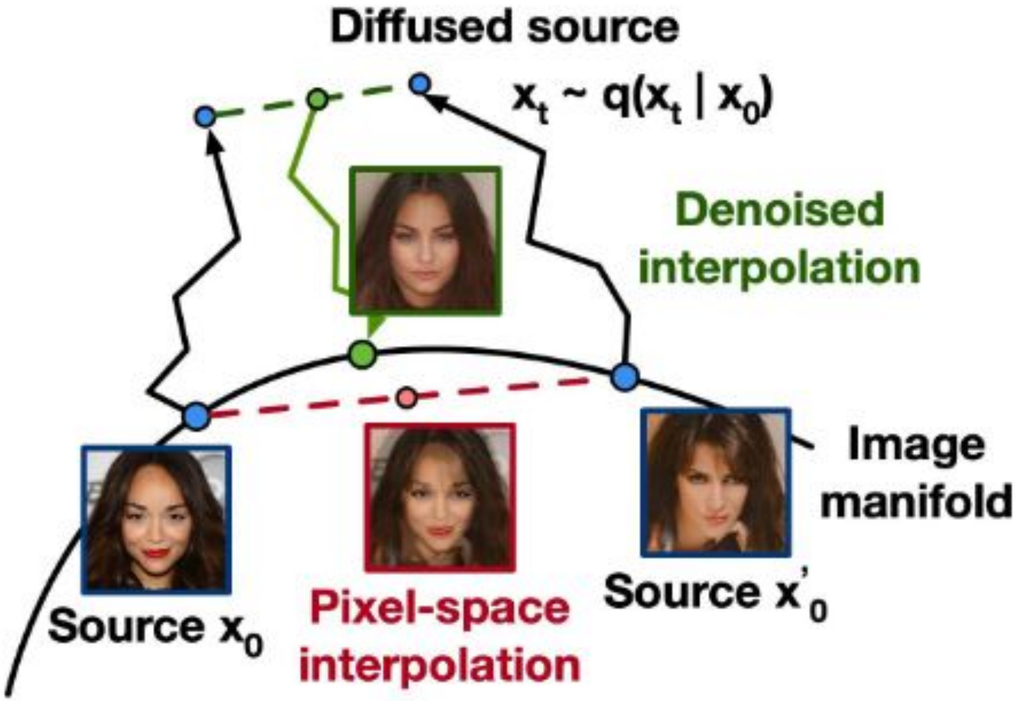


Experiment Details

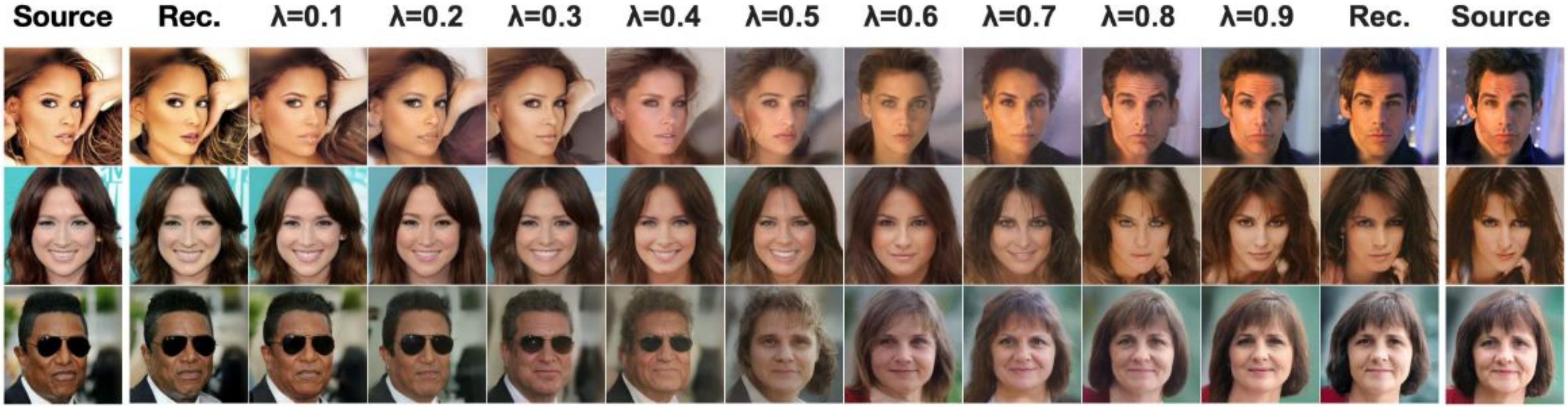
- T=1000 steps
  - But, technically, could stop any time
- Beta = 0.0001 (beginning) … 0.02 (end)
- Pixel values all scaled to +/-1

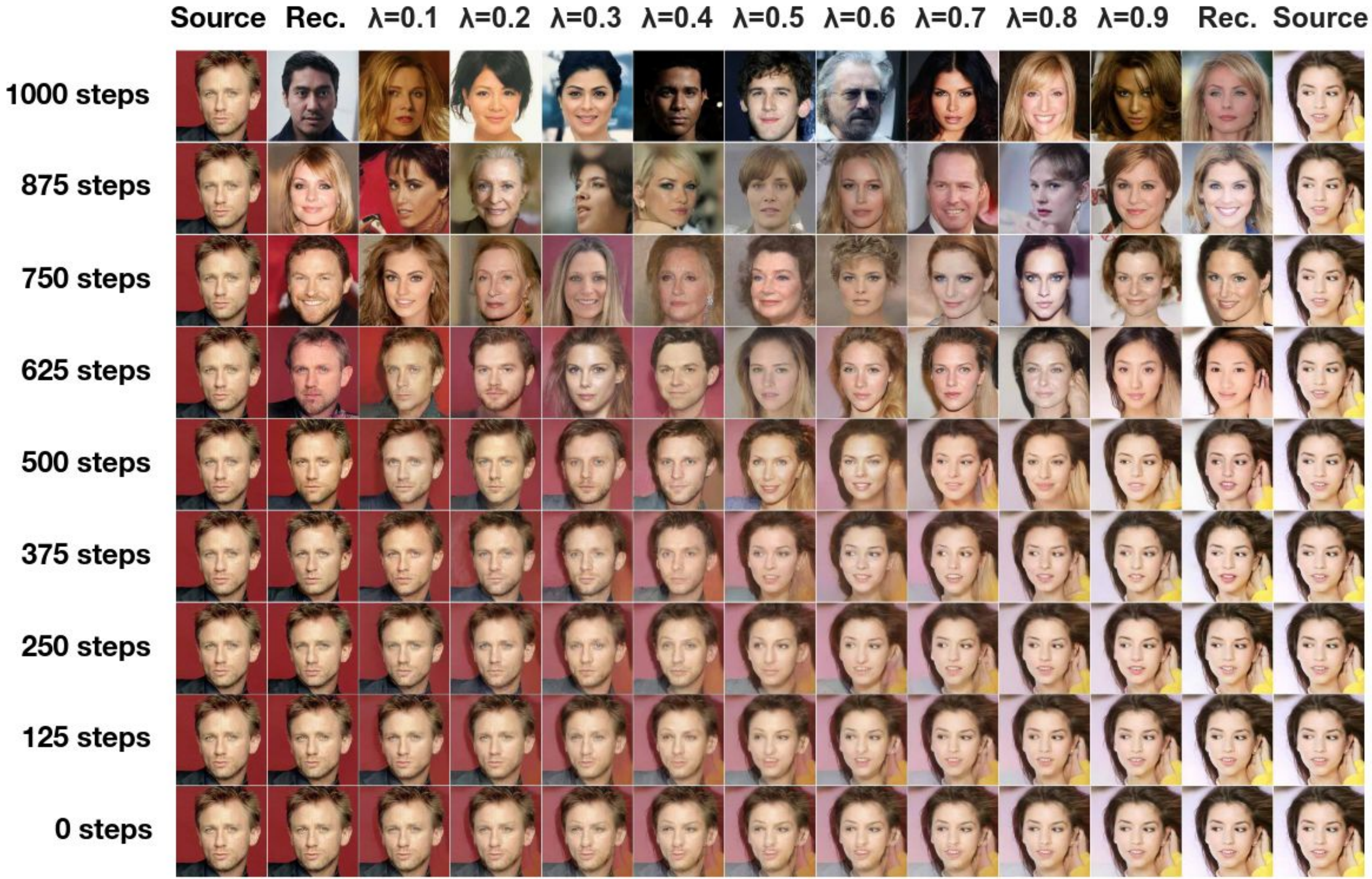
Progressive Sampling

Interpolation in the Latent Space



Interpolation Examples





Source Rec. λ=0.1 λ=0.2 λ=0.3 λ=0.4 λ=0.5 λ=0.6 λ=0.7 λ=0.8 λ=0.9 Rec. Source
1000 steps
875 steps
750 steps
625 steps
500 steps
375 steps
250 steps
125 steps
0 steps

Other Notes

- Tends to work best if the p() process just predicts the noise that needs to be removed
- Can then sample from p() and subtract this noise from the current image estimate