Diffusion/Score Models

Deep Learning
Bryan Pardo
Spring 2023

Diffusion Models became SOTA in 2021-ish

"Diffusion Models Beat GANs on Image Synthesis"
Dhariwal & Nichol, OpenAI, 2021

"Cascaded Diffusion Models for High Fidelity Image Generation"
Ho et al., Google, 2021

https://cvpr2022-tutorial-diffusion-models.github.io

DALL-E 2 (left)

"a teddy bear on a skateboard in times square"

"Hierarchical Text-Conditional Image Generation with CLIP Latents" Ramesh et al., 2022

IMAGEN (right)

A group of teddy bears in suit in a corporate office celebrating the birthday of their friend. There is a pizza cake on the desk.

"Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding", Saharia et al., 2022

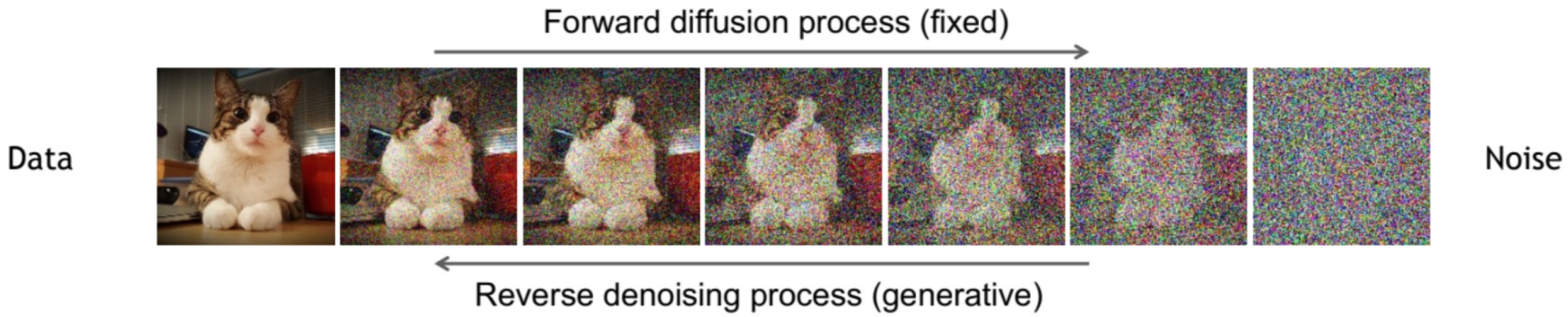
https://cvpr2022-tutorial-diffusion-models.github.io

Diffusion/Score models

- Assume a dataset $X = \{x_1, \ldots x_n\}$ where each x_i is an i.i.d. draw from a probability distribution $p(x)$.
- An example dataset would be pictures of cats.
- We want to learn $p(x)$, so we can make new things like the ones in the data set (i.e. more cat pictures)
- We do this by adding (Gaussian) noise to the samples in our data set and learning a function that can de-noise to generate things like our samples.

Two processes

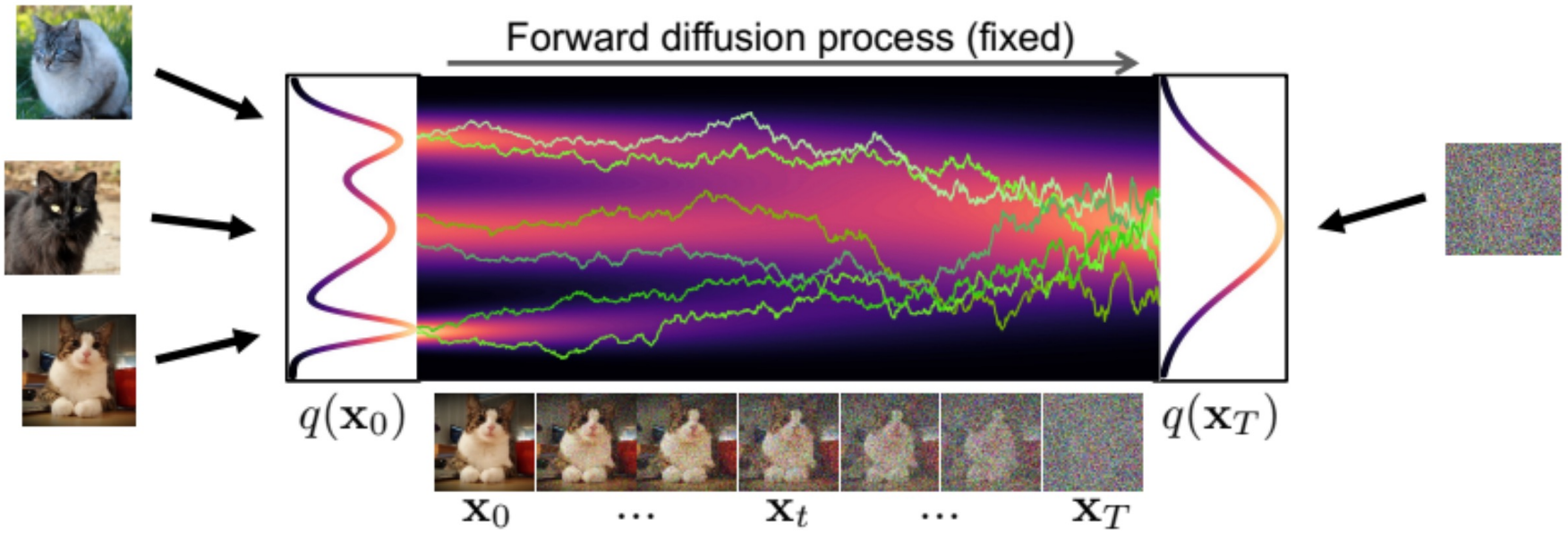
- Forward diffusion: add noise to a true data point
- Backward diffusion: denoise from a noise sample



Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015
Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021

https://cvpr2022-tutorial-diffusion-models.github.io

As t increases, we get more Guassian



Song et al., *ICLR*, 2021

Forward process (noising up 4 images)

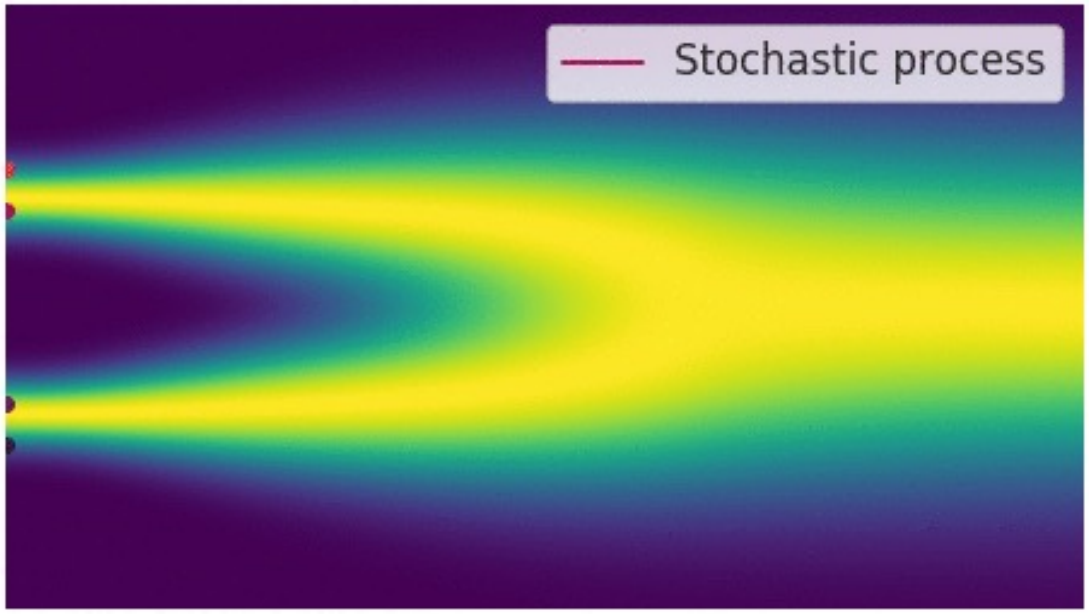
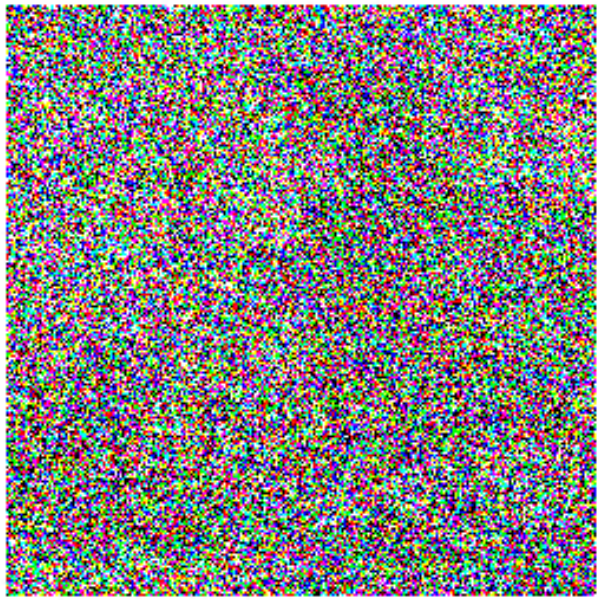


Image from https://yang-song.net/blog/2021/score/

Reverse (generating 4 images from noise)



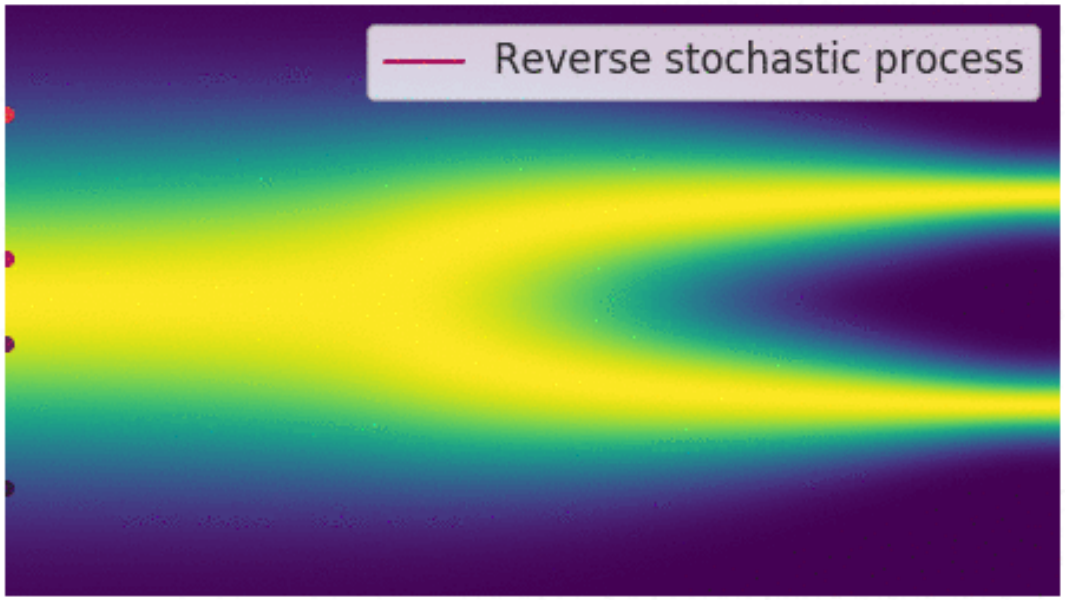


Image from https://yang-song.net/blog/2021/score/

Disclaimer: I’m going to focus on score-based

- Why? I like the math better.
- Don’t worry. Score models are the same thing as diffusion models.

The goal of our generative model

- Assume a dataset of items $\{x_1, x_2, \ldots, x_n\}$
- Each x_1 is drawn from an unknown distribution $p(x)$
- The goal is to make an estimate of $p(x)$such that drawing samples from the estimate gives things that seem like they came from $p(x)$.

Let's define our estimate function

- Define our estimate of probability function $p(x)$ as follows:

$$p(x) \approx f(x) = \frac{\tilde{p}(x)}{Z}$$

- …where Z is a normalizing constant that ensures $f(x)$ sums to 1

- In the end, we're going to want to lose Z and just learn $\tilde{p}(x)$

Learning our estimate

- Let's parameterize this by θ. These are the learned parameters.
- Approximate the true distribution by varying the parameters θ to maximize the probability of each sample x_i in the data.

$$\underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{N} \frac{\tilde{p}_\theta(x_i)}{Z_\theta}$$

There are 3 ways deal with Z_θ

RESTRICTING MODEL ARCHITECTURE TO MAKE CALCULATING POSSIBLE

causal convolutions in autoregressive models

GUESTIMATE Z_θ

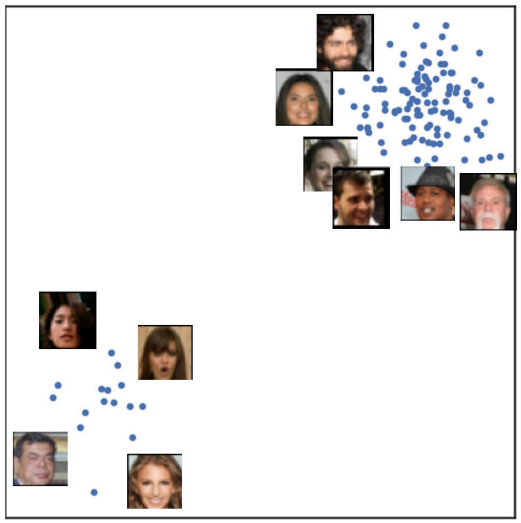
variational inference in VAEs

AVOID NEEDING TO KNOW Z_θ

Score models

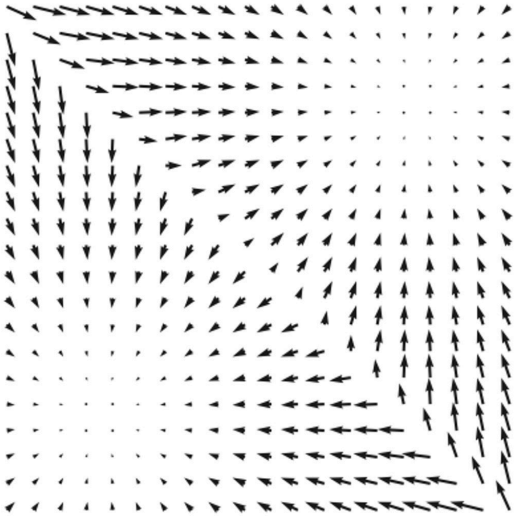
Score Function $\nabla_x p(x)$

- The score of a probability function is the gradient of that function.



Data samples

$$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p(\mathbf{x})$$



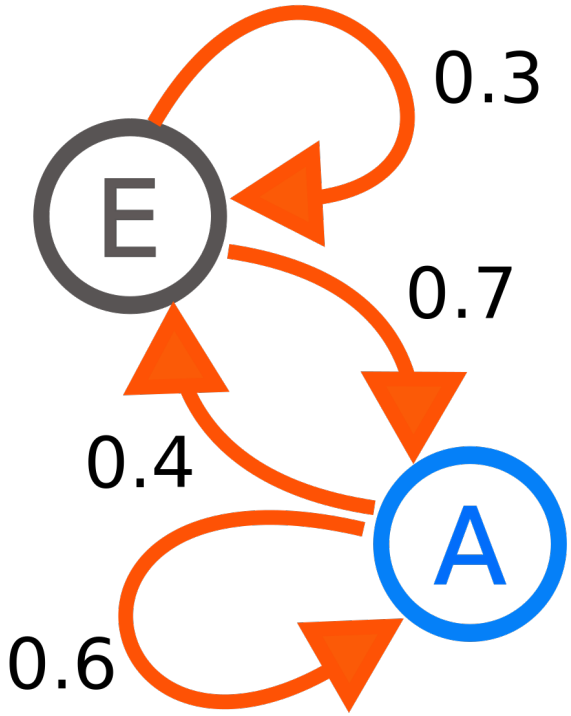
Scores

$$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$$

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

Markov Chain

- A stochastic model describing a sequence of possible events where the probability of each event depends only on the state attained in the previous step.
- What happens next depends only on the current state.

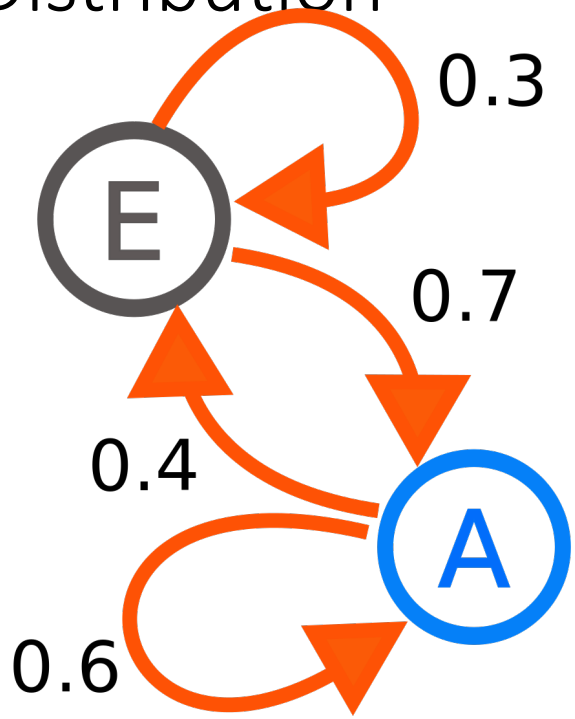


https://en.wikipedia.org/wiki/Markov_chain#/media/File:Markovkate_01.svg

Markov Chain Equilibrium Distribution

- If you sample from the Markov Chain over and over, you end up with this distribution

- Example: Four is the magic number (spelling letters chain)



https://en.wikipedia.org/wiki/Markov_chain#/media/File:Markovkate_01.svg

Markov chain Monte Carlo (MCMC)

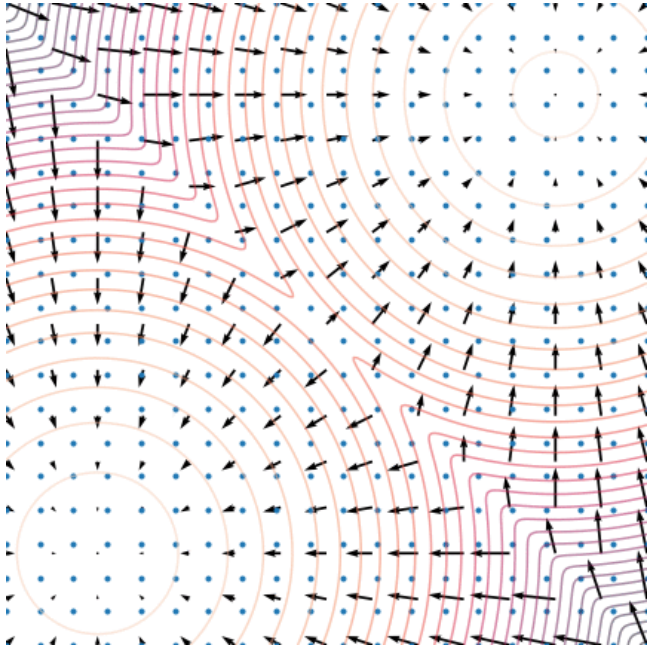
- MCMC algorithms are for sampling from a probability distribution you can’t model directly.

- A Markov Chain with the desired distribution as its equilibrium distribution, lets you sample of the desired distribution by recording states from the chain.

- To work, an MCMC algorithm need a way of ensuring each step gets closer to the desired distribution.

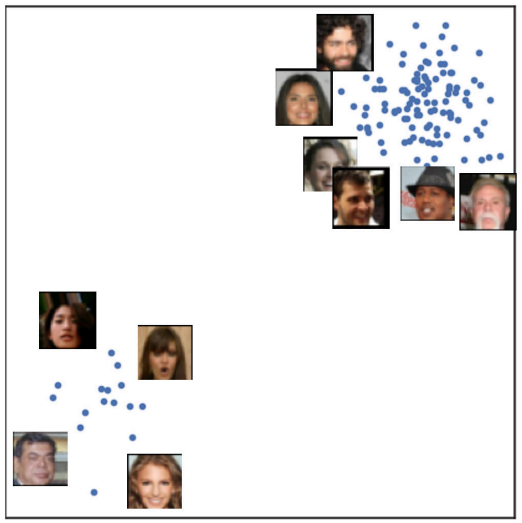
Langevin Dynamics

- Named for French physicist Paul Langevin (Lahn-je-vahn)
- Langevin dynamics provides an MCMC procedure to sample from a distribution using only its score function
- It initializes the chain from an arbitrary prior distribution and then iterates to converge on p(x)



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

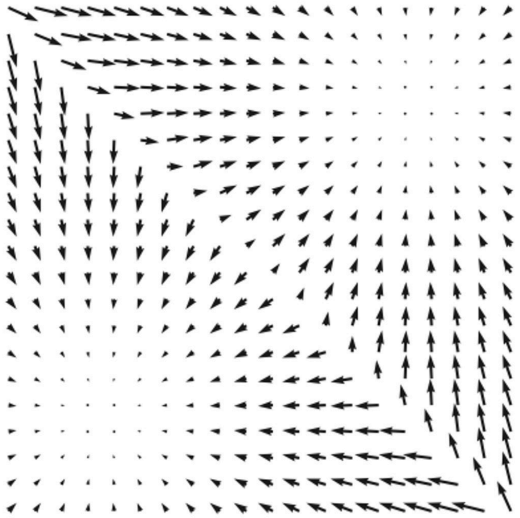
We do this by learning the gradient of p(x)



Data samples

$$\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \overset{\text{i.i.d.}}{\sim} p(\mathbf{x})$$

score matching



Scores

$$\mathbf{s}_\theta(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p(\mathbf{x})$$

Langevin dynamics

New samples

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

Learning the score: $\nabla_x \tilde{p}(x)$

- Assume we can approximate data distribution p(x), up to some normalizing factor Z:

$$p(x) = \frac{\tilde{p}(x)}{Z}$$

Take the log:

$$\log\big(p(x)\big) = \log\left(\frac{\tilde{p}(x)}{Z}\right)$$
$$= \log(\tilde{p}(x)) - \log(Z)$$

Rearrange:

$$\log\big(p(x)\big) + \log(Z) = \log(\tilde{p}(x))$$

Learning the score: $\nabla_x \tilde{p}(x)$

From the previous slide: $\log\big(p(x)\big) + \log(Z) = \log(\tilde{p}(x))$

…Now let's take gradients

Gradient=0 w.r.t. x

$$\nabla_x(\log\big(p(x)\big) + \log(Z)) = \nabla_x \log\big(\tilde{p}(x)\big)$$

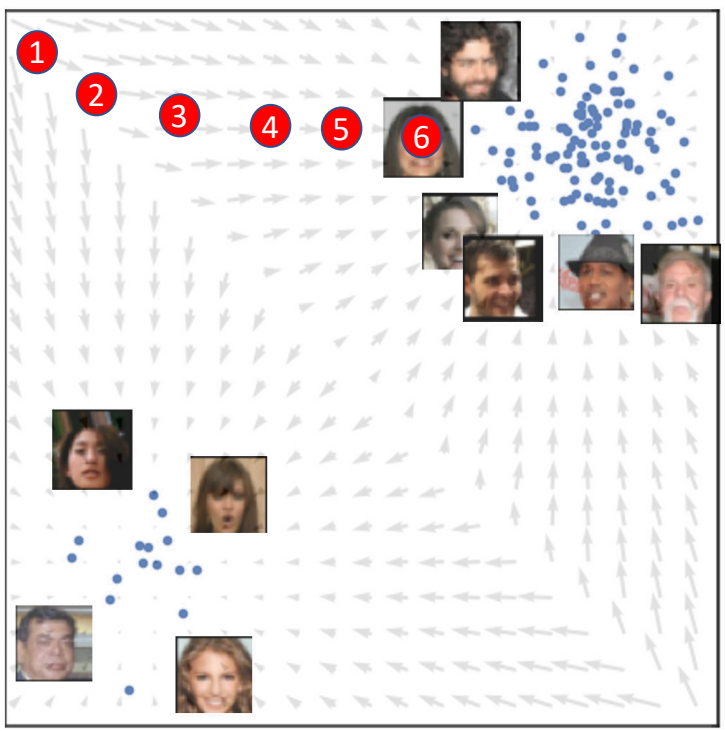
$$\nabla_x \log\big(p(x)\big) = \nabla_x \log(\tilde{p}(x))$$

Z drops out, since it is a constant with respect to x.

So, we can skip learning Z….which was the intractable part.

Using$\nabla_x p(x)$ to generate a new example

- Start at a random point.
- Iteratively follow gradient $\nabla_x p(x)$ to reach a point that looks like our training data.
- Voila! a new sample is generated!
- All we need is an estimate of $\nabla_x p(x)$.

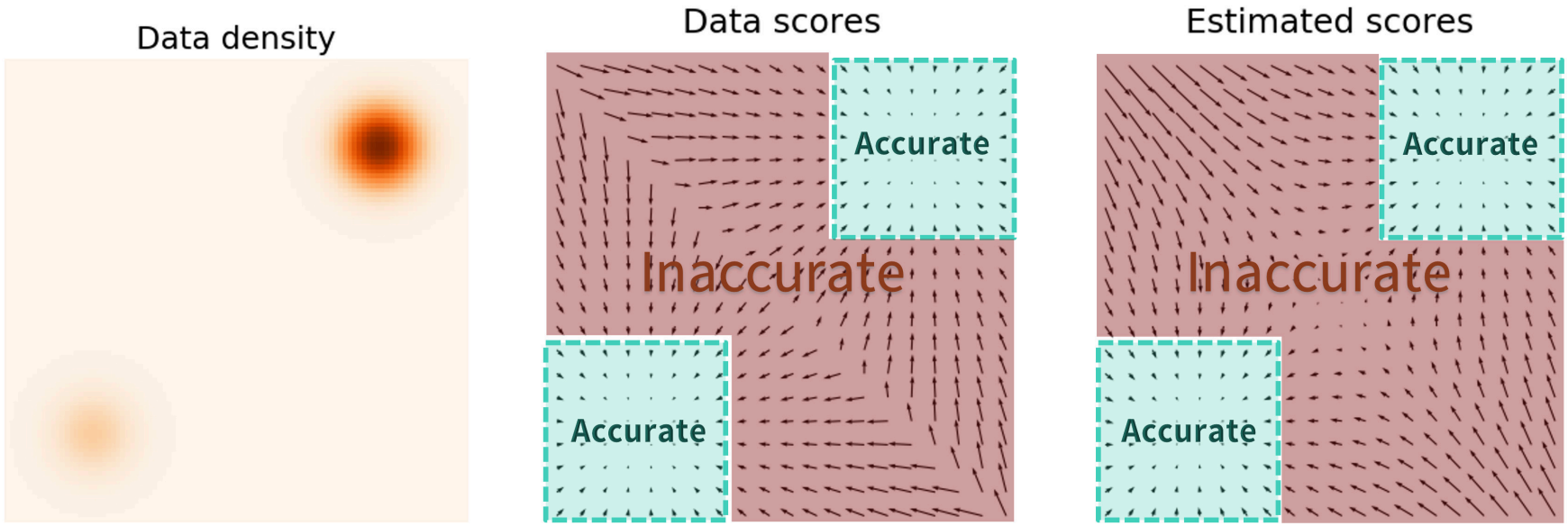


Modified image from https://yang-song.net/blog/2021/score/

Using $\nabla_x p(x)$ vs $\nabla_x \log(p(x))$

- We want the gradient $\nabla_x p(x)$ of the probability function $p(x)$ because it lets us start anywhere in the space and iteratively follow gradient $\nabla_x p(x)$ to reach a point that looks like our training data.
- So…learning $\nabla_x p(x)$ it is as good as learning $p(x)$
- The log of the gradient $\nabla_x \log(p(x))$ also lets us play the same game.

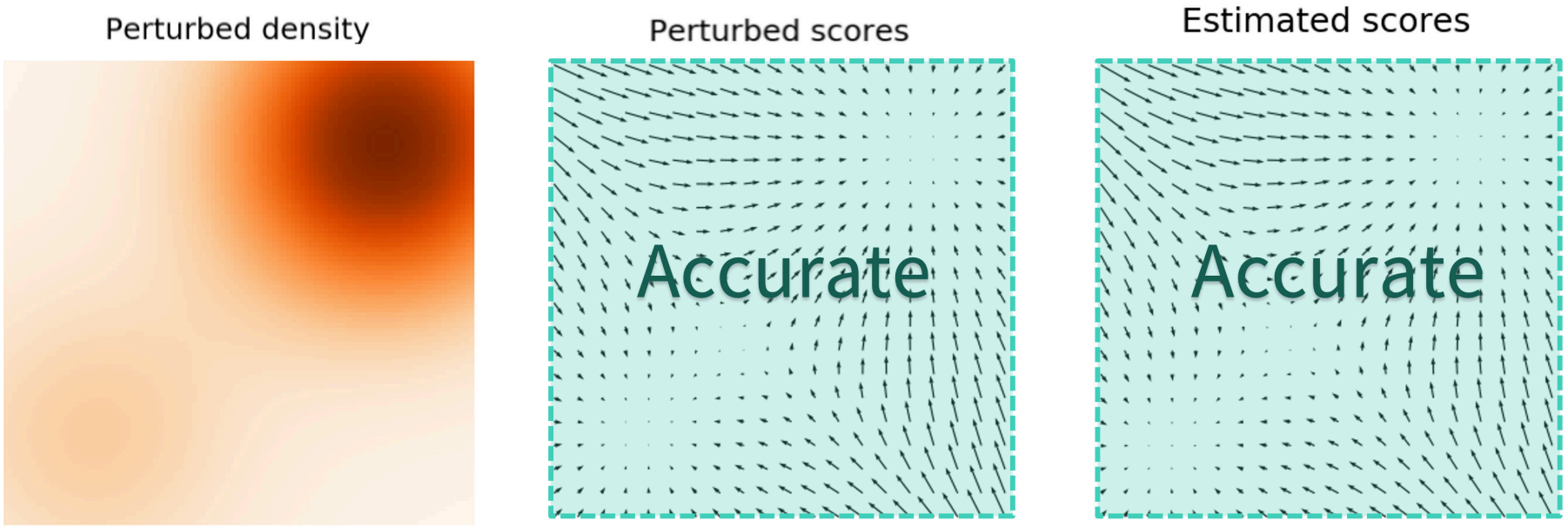
Don’t worry, it will be imperfect.



- Our estimate $\nabla_x \tilde{p}(x)$ is only accurate where we have lots of data.
- For 1000 by 1000 pixel images, this 10^6 D space is mostly empty.
- Ergo, the estimate is mostly bad.

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

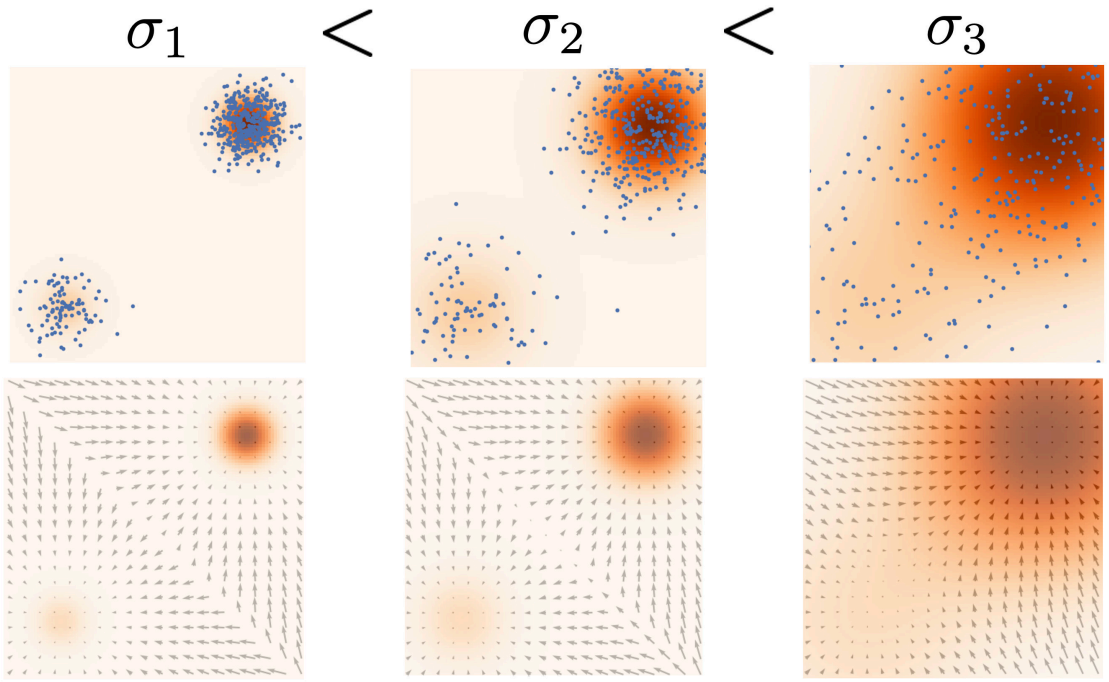
What if we made more data to fill in the space?



- If we add perturbed data, our sample would cover more of the space.
- If we perturb using a parametric noise distribution (e.g. Gaussian noise) we could account for it when we learn the gradient.

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

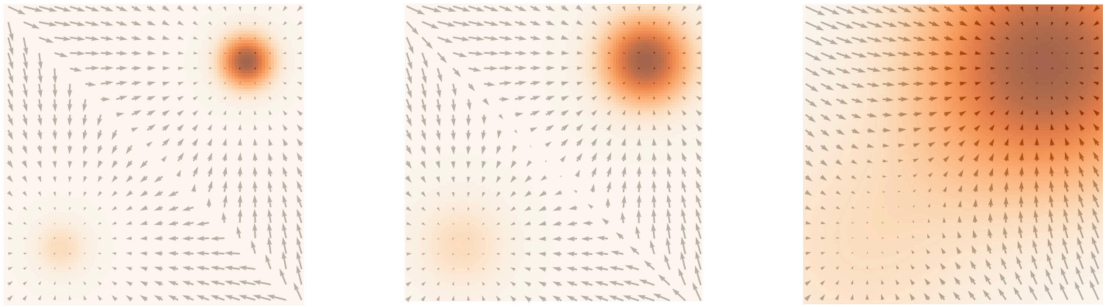
Perturb at multiple scales



Add (Gaussian) noise at different scales until your data fills the space

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

Walk it back



Start from the most perturbed (essentially an isotropic Gaussian)

Walk backwards from that to the 2nd most perturbed….then the 3rd most…until you reach what you hope is the original p(x)

Learn these steps with a neural network that outputs the noise gradient at each step.

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

Relating this to the other view of diffusion

- When we typically talk diffusion, we discuss learning the mean μ and covariance matrix Σ for each Gaussian at each step in the diffusion process.

- If you know μ and Σ of a Gaussian, you can derive the gradient.

- The score is the gradient.

- So these are two views of the same thing.

What we'd hope to see.

Start from the most perturbed (essentially an isotropic Gaussian)

Walk backwards from that to the 2nd most perturbed….then the 3rd most…until you reach what you hope is a sample from p(x)

We learn these steps with a neural network that outputs the noise gradient (AKA the score) at each step.

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

What we're minimizing

The number of levels in our noise scaling.

Input example at step t in the Lavengian process

.....The score function.....

$$\sum_{i=1}^{L} \lambda(i) \mathbb{E}_{p_{\sigma_i(\mathbf{x})}} \left\| s_\theta(\mathbf{x}_t, t) - \nabla_{x_t} \log q_t(\mathbf{x}_t | \mathbf{x}_0) \right\|_2^2$$

How much weight to give this level in the noise scaling, when calculating loss

The function determining the scale of noise being applied

learned network with paramters θ

What we're minimizing

The number of levels in our noise scaling.

Input example at step t in the Langevin process

…..The score function…..

$$\sum_{i=1}^{L} \lambda(i) \mathbb{E}_{p_{\sigma_i(\mathbf{x})}} \left\| s_\theta(\mathbf{x}_t, t) - \nabla_{x_t} \log q_t(\mathbf{x}_t | \mathbf{x}_0) \right\|_2^2$$

How much weight to give this level in the noise scaling, when calculating loss. Often a function of σ

learned network with paramters θ

The function determining the scale of noise being applied. Here, σ is our standard deviation

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

We're minimizing this difference

$$\left\| \underbrace{s_\theta(\mathbf{x}_t, t)}_{\text{neural network}} - \underbrace{\nabla_{x_t} \log q_t(\mathbf{x}_t|\mathbf{x}_0)}_{\text{score of diffused data sample}} \right\|_2^2$$

If we train s_θ on the data, we hope to learn the gradient for each time step EVERYWHERE, not just at the points in the training data.

We know this because we have each data sample $\mathbf{x_0}$ and we defined the parameters of the distribution q_t applied at step t.

https://cvpr2022-tutorial-diffusion-models.github.io

After training, we run in reverse order

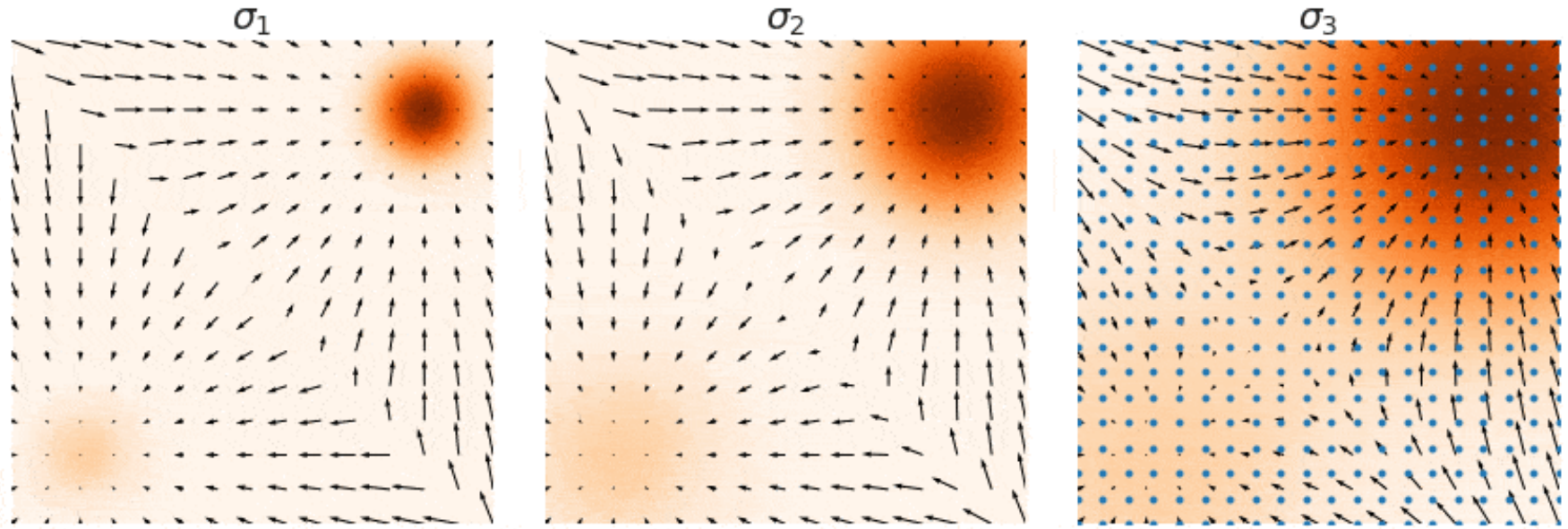


Image from https://yang-song.net/blog/2021/score/

Score network for CelebA

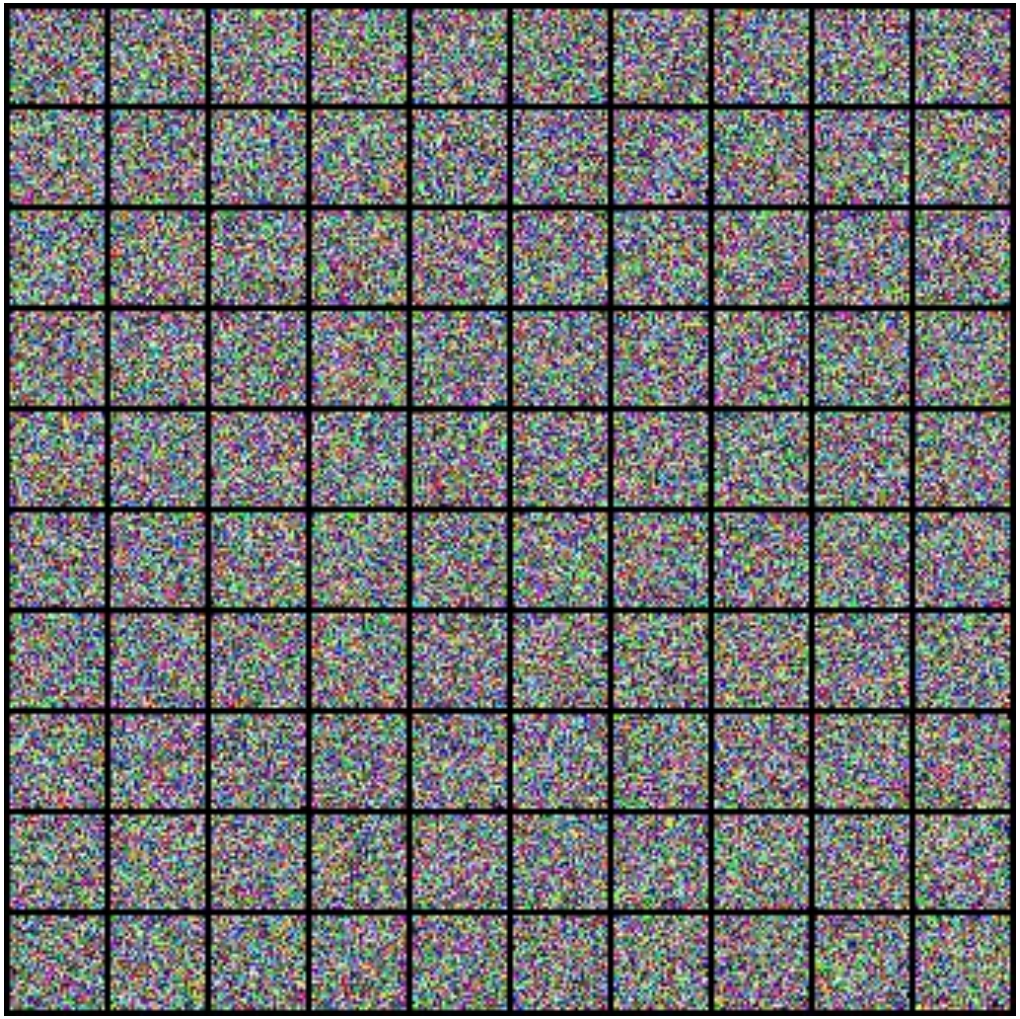


Image from https://yang-song.net/blog/2021/score/

Score network for CIFAR-10

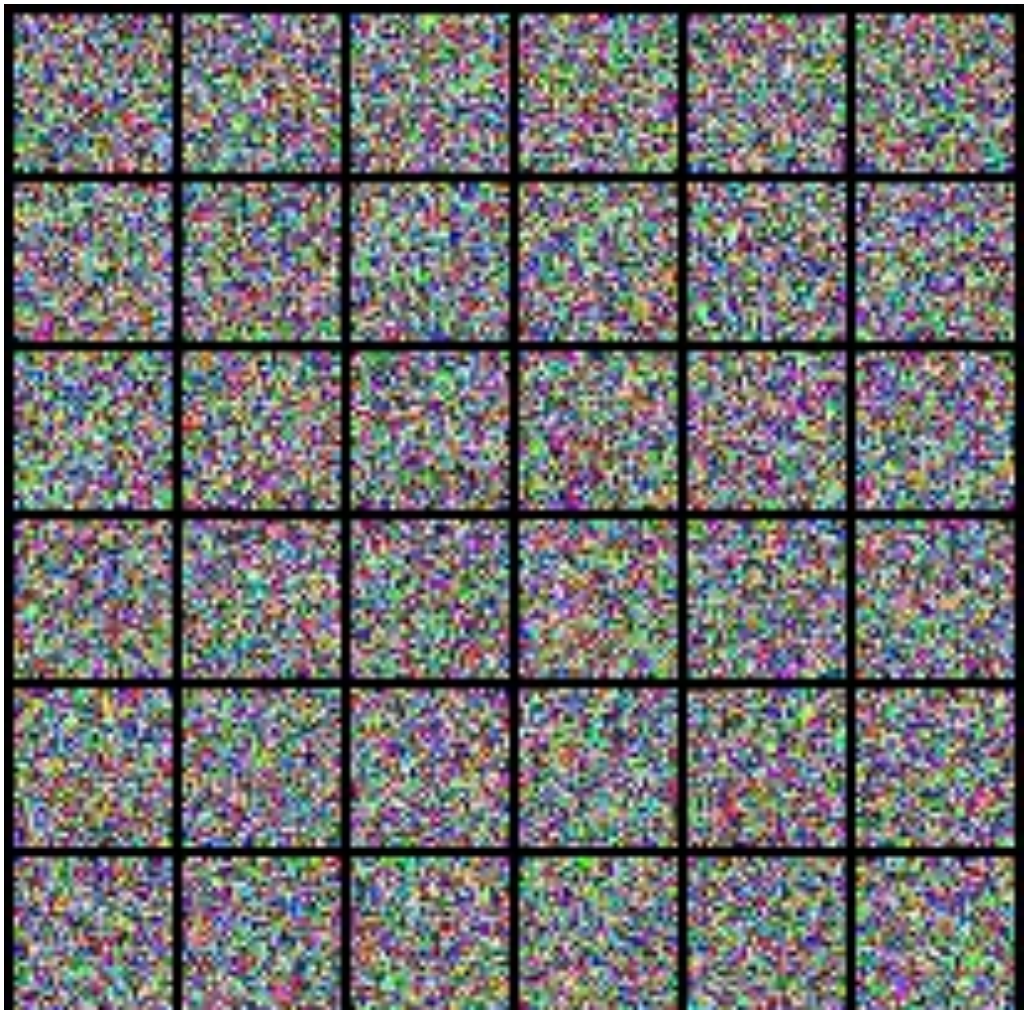
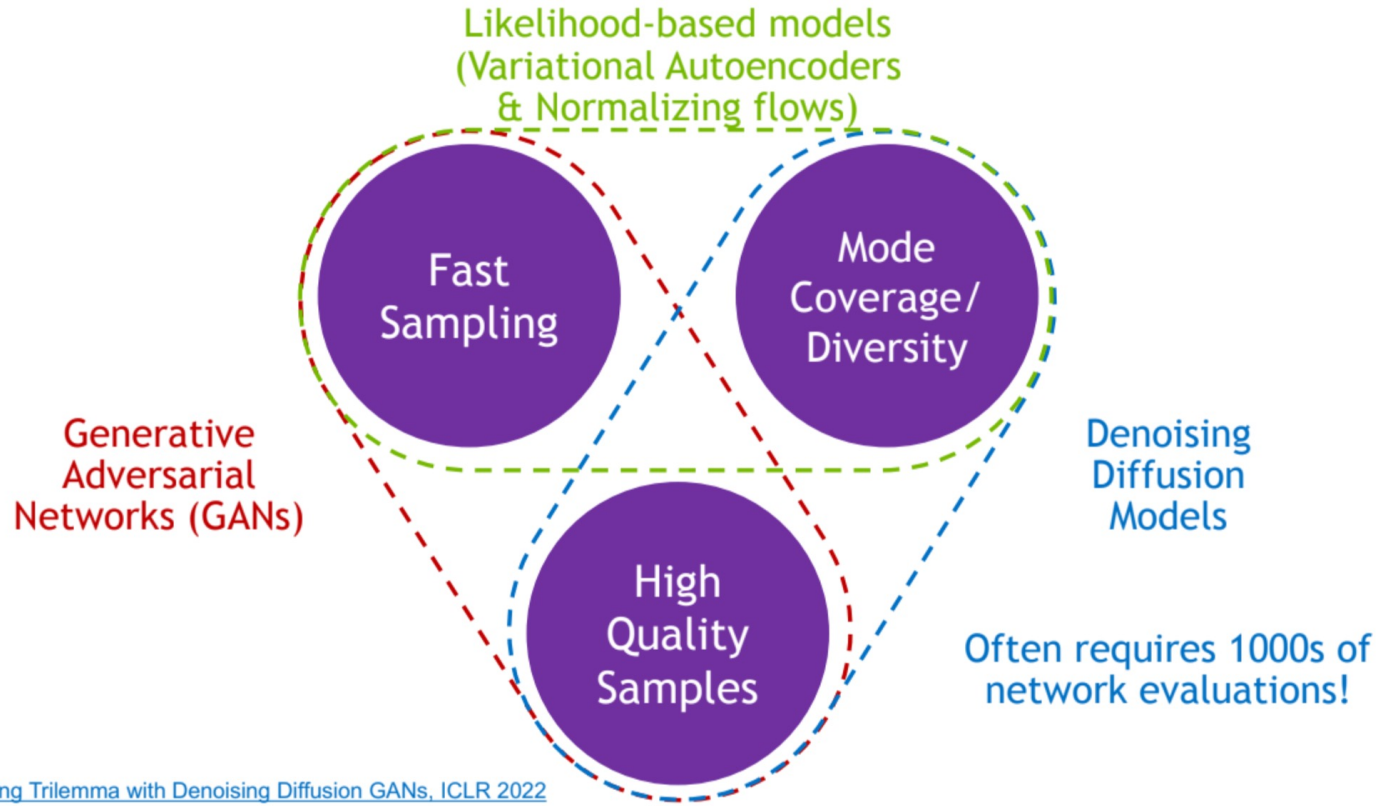


Image from https://yang-song.net/blog/2021/score/

Where we are

- We can learn $\nabla_x \log(\tilde{p}(x))$ with a diffusion model.
- This lets us start anywhere in the space & walk the gradients to make a sample similar to one from the original distribution.
- This can take thousands of steps at generation time.

The generative modeling “tri-lemma”



Tackling the Generative Learning Trilemma with Denoising Diffusion GANs, ICLR 2022

https://cvpr2022-tutorial-diffusion-models.github.io

Many approaches to speeding up diffusion

- Teacher-student models to learn bigger steps
- Run diffusion on VAE latent space
- Learn variable step size in training, then take bigger steps in generation
- Do continuous (infinite steps) and then…well…there's lots there. No time to talk about it all, really.

Conditioning Using a Classifier

Where we are

- We can learn $\nabla_x \log(\tilde{p}(x))$ with a diffusion model.
- This lets us start anywhere in the space & walk the gradients to make a sample similar to one from the original distribution.
- We have no way of controlling which part of the original distribution.

So…how do we get to this?

DALL·E 2

"a propaganda poster depicting a cat dressed as french emperor napoleon holding a piece of cheese"

IMAGEN

"A photo of a raccoon wearing an astronaut helmet, looking out of the window at night."

https://cvpr2022-tutorial-diffusion-models.github.io

Adding a conditioning variable y

DOG

Gradient=0
w.r.t. x

$$p(x \mid y) = \frac{p(y \mid x) \cdot p(x)}{p(y)}$$

$$\implies \log p(x \mid y) = \log p(y \mid x) + \log p(x) - \log p(y)$$

$$\implies \nabla_x \log p(x \mid y) = \nabla_x \log p(y \mid x) + \nabla_x \log p(x),$$

From Sander Dielman's https://benanne.github.io/2022/05/26/guidance.html

...continued

Get this from any previously- trained classifier

Get this from our previously-trained diffusion model

$$\nabla_x \log p(x \mid y) = \nabla_x \log p(y \mid x) + \nabla_x \log p(x),$$

Gradient of the class-conditioned distribution

Gradient of the classifier's label, given, example x

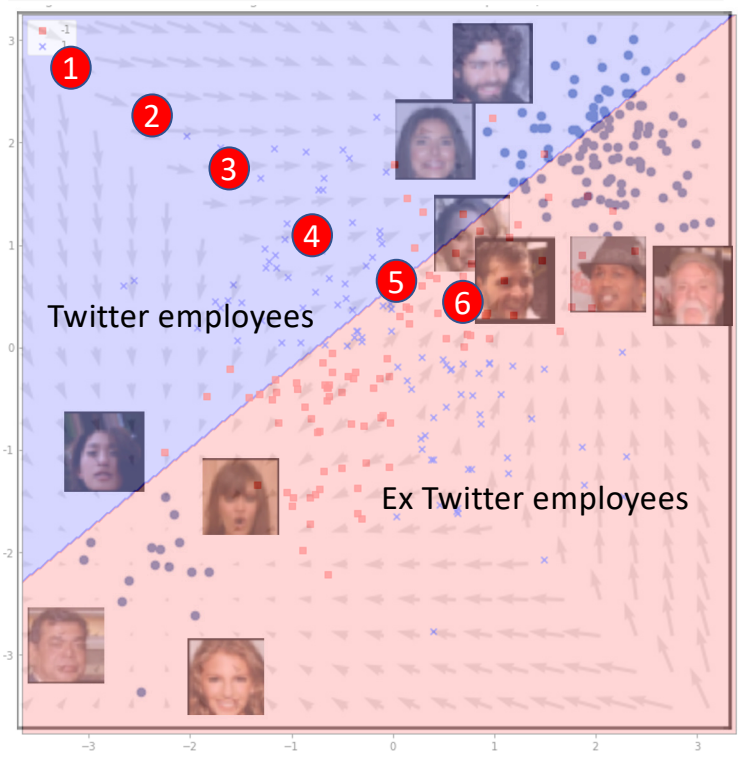
Gradient of the unconditional example x

- This lets us walk the gradients to make a sample like one from the original distribution that also would get labeled as a "dog" (or "cat" or whatever) by the classifier.

From Sander Dielman's https://benanne.github.io/2022/05/26/guidance.html

Use $\nabla_x \tilde{p}(x) + \nabla_x p(y|x)$ to condition generation

- Start at a random point.
- Iteratively follow the gradient $\nabla_x \tilde{p}(x)$ to reach a point like our training data.
- Iteratively follow $\nabla_x p(y|x)$ to reach a point with class y



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

Weighting our classifier “Guidance”

$$\nabla_x \log p_\gamma(x \mid y) = \nabla_x \log p(x) + \gamma \nabla_x \log p(y \mid x)$$

$\gamma = 1$ Class: Pembroke Welsh corgi $\gamma = 10$

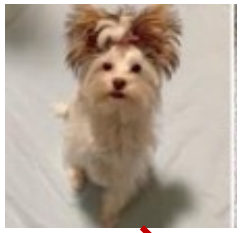
Image from Dhariwal, Nichol, “Diffusion Models Beat GANs on Image Synthesis”, Neural Information Processing Systems, 2021

The catch

- If the classifier can't handle noisy inputs, it can't provide good guidance
- Practically speaking, this can mean the classifier needs to be trained on the data generated by the diffusion process
- This means we can't necessarily use any off-the-shelf classifier for guidance
- So what now???
- …and how does CLIP play into things?
- We'll see with the GLIDE paper.

Classifier-free Conditioning

Let’s run Bayes the other way…



$$p(y \mid x) = \frac{p(x \mid y) \cdot p(y)}{p(x)}$$

$$\implies \log p(y \mid x) = \log p(x \mid y) + \log p(y) - \log p(x)$$

$$\implies \nabla_x \log p(y \mid x) = \nabla_x \log p(x \mid y) - \nabla_x \log p(x).$$

A little math

- Our formula for classifier-guided generation was this:

$$\nabla_x \log p_\gamma(x \mid y) = \nabla_x \log p(x) + \gamma \nabla_x \log p(y \mid x)$$

- From the previous slide we get:

$$\nabla_x \log p(y \mid x) = \nabla_x \log p(x \mid y) - \nabla_x \log p(x)$$

- Let’s plug that into the guided generation formula.

Weighted self-guidance

- This gives us:

$$\nabla_x \log p_\gamma(x \mid y) = \nabla_x \log p(x) + \gamma \left(\nabla_x \log p(x \mid y) - \nabla_x \log p(x)\right)$$

- Or equivalently:

$$\nabla_x \log p_\gamma(x \mid y) = (1-\gamma)\nabla_x \log p(x) + \gamma \nabla_x \log p(x \mid y)$$

Using this formula

- Train a class-conditional diffusion model
- Around 20% of the time, randomly replace the true label with the ”anything” class label.
- Use this “anything” conditioning as a substitute for unconditioned generation
- At generation time, use this formula for guidance:

$$\nabla_x \log p_\gamma(x \mid y) = (1-\gamma)\nabla_x \log p(x) + \gamma \nabla_x \log p(x \mid y)$$

Next time

- We'll see how CLIP and guidance lead to GLIDE