Guest Lecture COMP6258

Diffusion Models & Flow Matching

Introduction

Diffusion

Diffusion Overview

Diffusion models learn map from a Gaussian distribution to a distribution approximated with a dataset.

Multiple steps are used to execute this mapping.

ImageNet images generated using the EDM 2 diffusion model (FID = 1.91)

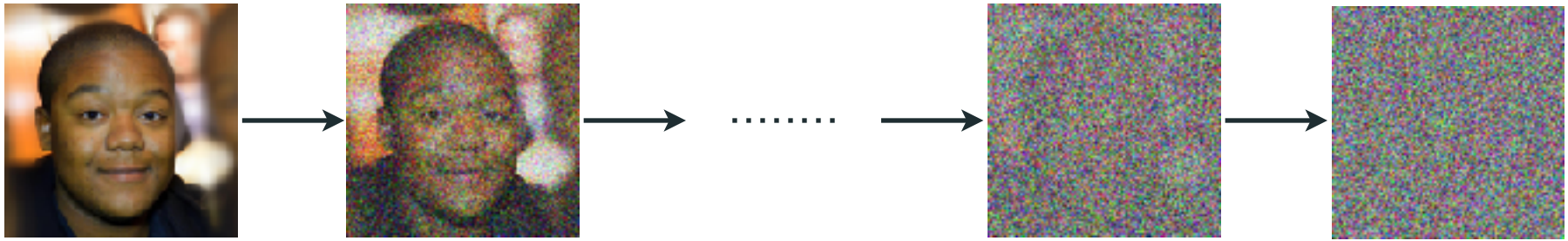
Forward Process

The forward process gradually adds noise to our starting image x_0:

$$x_t = \sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t}\epsilon, \qquad \alpha_t = \prod_{s=1}^{t} 1 - \beta_s$$

$$\epsilon \sim \mathcal{N}(0, I), \qquad \beta_t = [0.0001, \ldots, 0.02]$$

where $t \in [0,T]$ is the timestep.



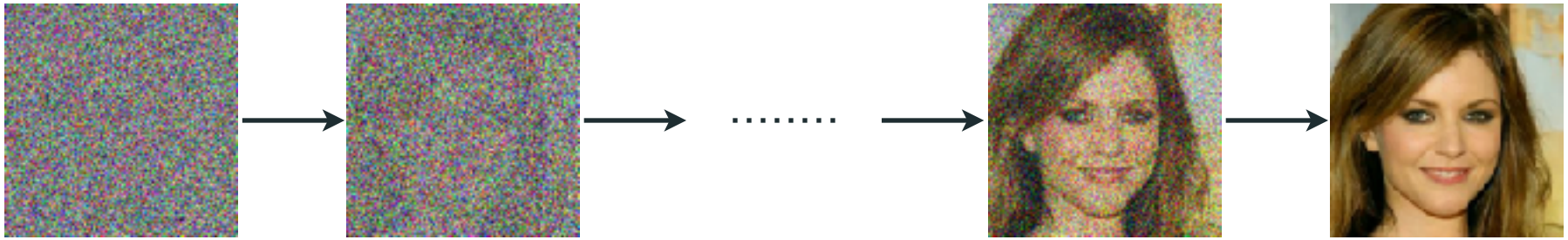
Reverse Process

We then use a network f to predict x_0, with parameters θ, to reverse this process:

$$p(x_T) = \mathcal{N}(0, I)$$

$$p(x_{t-1}|x_t) = \mathcal{N}(f_\theta(x_t, t), \sigma_t I)$$

where σ controls the amount of noise at each step.



Training

In practice, we typically learn the reverse process using a neural network D which predicts noise instead of x_0:

$$\mathcal{L}_{MSE} = \|D_\theta(x_t, t) - \epsilon\|^2$$

This indirectly maximizes the the log-likelihood of the training data w.r.t. the network parameters:

$$\underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{I} \log p(x_{0,i})$$

* See Chapter 18 of "Understanding Deep Learning" by Simon Prince for full loss derivation

Inference

To step through from x_T to x_0 we repeatedly apply the following equation:

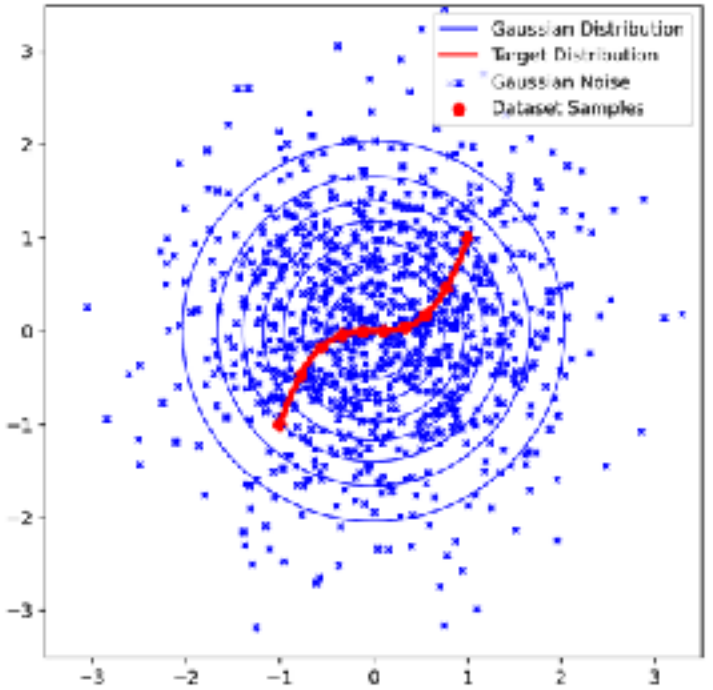
$$x_{t-1} = \sqrt{\alpha_{t-1}} \left(\frac{x_t - \sqrt{1-\alpha_t} D_\theta(x_t, t)}{\sqrt{\alpha_t}} \right) + \sqrt{1 - \alpha_{t-1} - \sigma} D_\theta(x_t, t) + \sigma \epsilon$$

where $\sigma = 0$ leads to deterministic diffusion and $\sigma > 0$ results in stochastic diffusion.

2D Toy Example

To train our model, we give a network a linear combination of x_T to x_0 and train it to predict x_T.

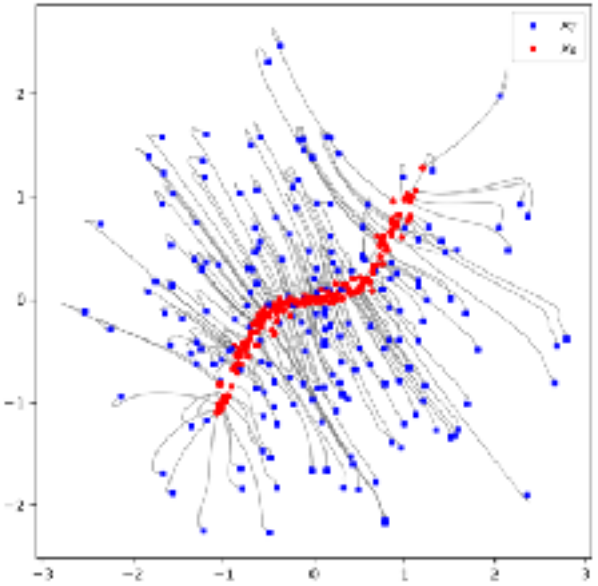
At inference time, we sample x_T, then repeated apply our network to step towards x_0.



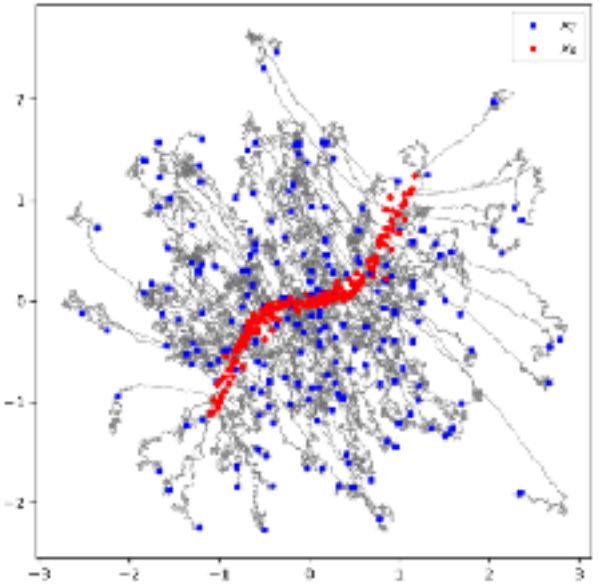
* https://github.com/harveymannering/boilerplate_code/blob/main/diffusion_toy_example.ipynb

2D Toy Example

Deterministic Sampling (σ = 0)



Stochastic Sampling (σ > 0)



* https://github.com/harveymannering/boilerplate_code/blob/main/diffusion_toy_example.ipynb

Algorithms

A summary of both training and inference can be seen below.

Algorithm 1 Training

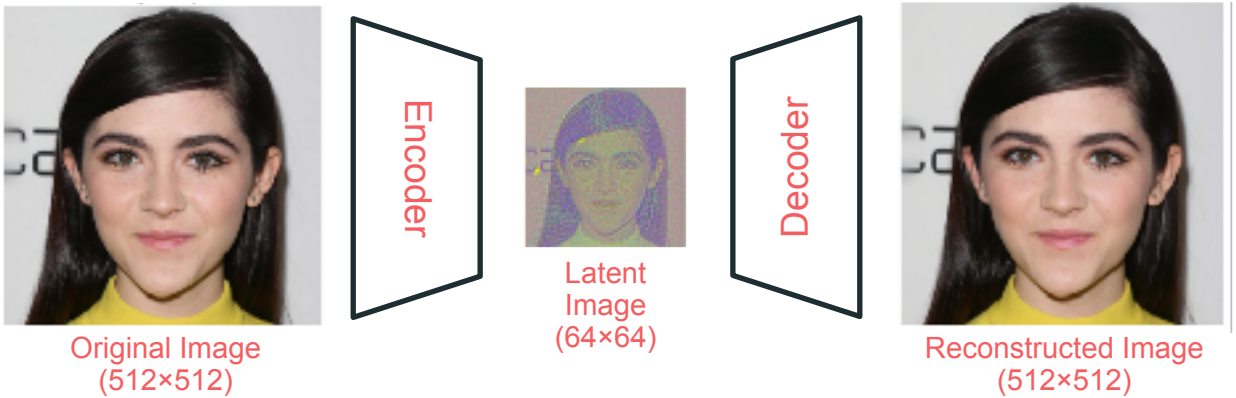
1. **repeat**
2. $x_0 \sim q(x_0)$
3. $t \sim \mathcal{U}(\{1, \dots, T\})$
4. $\epsilon \sim \mathcal{N}(0, I)$
5. $x_t = \sqrt{\alpha_t} x_0 + \sqrt{1 - \alpha_t}\epsilon$
6. Take gradient step on
7. $\nabla_\theta \| D_\theta(x_t, t) - \epsilon \|^2$

Algorithm 2 Sampling

1. $x_T \sim \mathcal{N}(0, I)$
2. **for** $t = T, \dots, 1$ **do**
3. $\epsilon \sim \mathcal{N}(0, I)$ if $t > 1$, else $\epsilon = 0$
4. $x_{t-1} = \sqrt{\alpha_{t-1}} \left(\frac{x_t - \sqrt{1-\alpha_t} D_\theta(x_t, t)}{\sqrt{\alpha_t}} \right) + \sqrt{1 - \alpha_{t-1} - \sigma_t} D_\theta(x_t, t) + \sigma_t \epsilon$
5. **end for**
6. **return** x_0

Practical Considerations

- The architecture must have the same input/output dimensions (e.g. U-Net). We must condition the network on the timestep *t.*
- Both training and inference can be very slow. Latent Diffusion significantly speeds up both by doing diffusion/flow matching in the latent space of a VAE, effectively reducing the image dimensions.



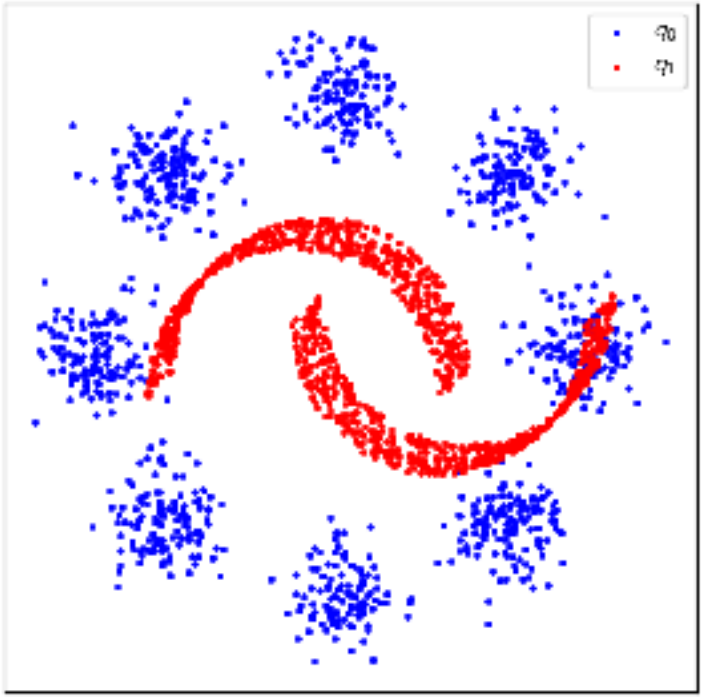
Flow Matching

Flow Matching

The problem we are trying to solve:

> Given samples two distributions, q_0 and q_1, can we find a mapping that transforms q_0 in to q_1?

For this mapping, we can use a vector field $u_t(x)$ where t is between 0 and 1, and x is sample from the probability path p, that runs between q_0 and q_1.



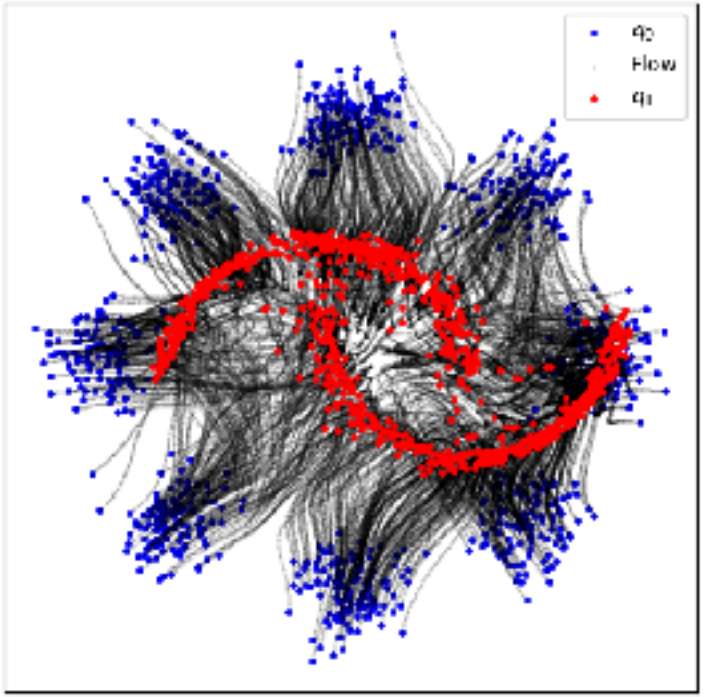
Flow Matching

Think of $u_t(x)$ as an Ordinary Differential Equation (ODE).

Flow matching tries to learning the $u_t(x)$ with a neural network v_θ

$$\mathcal{L}_{\mathrm{FM}}(\theta) := \mathbb{E}_{t\sim\mathcal{U}(0,1),x\sim p_t(x)}\|v_\theta(t,x) - u_t(x)\|^2$$

Unfortunately, calculating $u_t(x)$ in not tractable.



Conditional Flow Matching

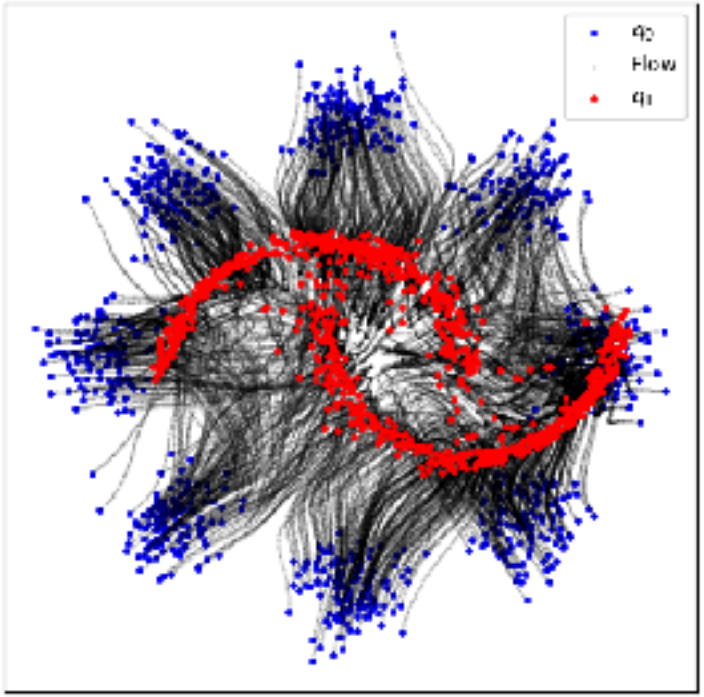
However, $u_t(x|z)$ is tractable to calculate, giving the loss function

$$\mathcal{L}_{\text{CFM}}(\theta) := \mathbb{E}_{t,q(z),p_t(x|z)} \| v_\theta(t,x) - u_t(x|z) \|^2$$

where z is a conditioning variable.

Furthermore, both loss function have the same gradients, meaning optimizing the latter, also optimizes the former.

Once we have learnt our network v_θ, we can use an ODE solver to map from q_0 and q_1.

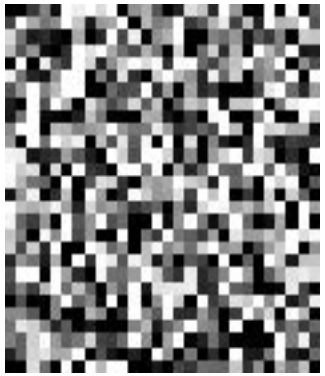


Conditional Flow Matching

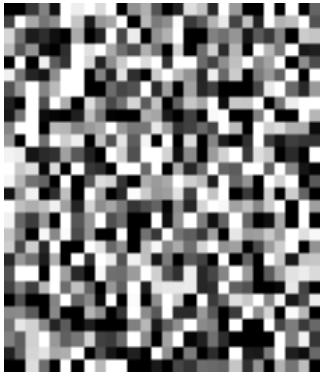
We can scale this up to higher dimensional distributions, such as image distributions.

If q_0 is a Gaussian and q_1 is an image distribution, we now have a generative model.

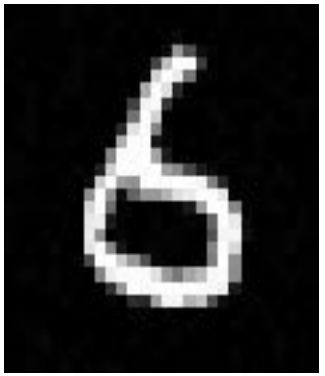
$x_0 \sim q_0$



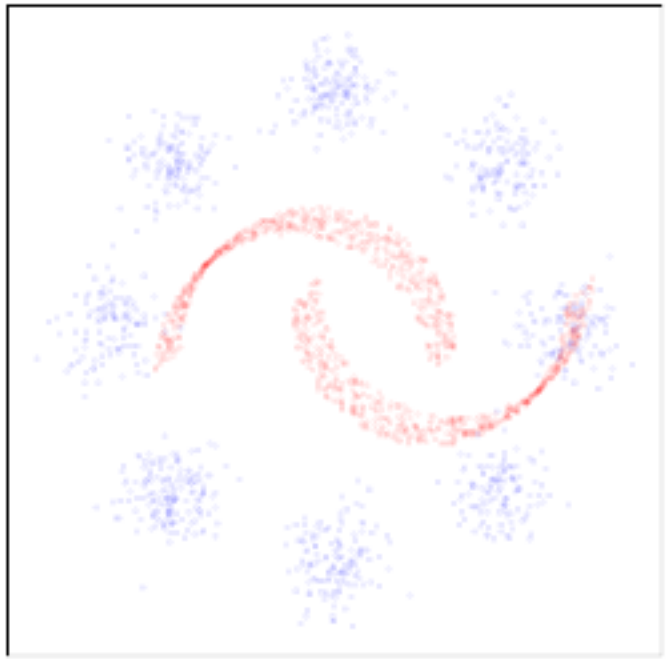
x



$x_1 \sim q_1$

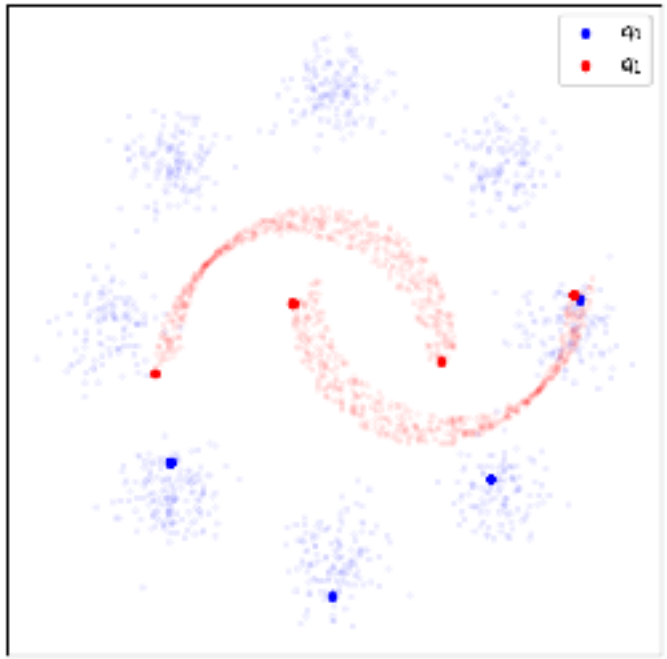


Basic Form a CFM



So how to we calculate x and u_t?

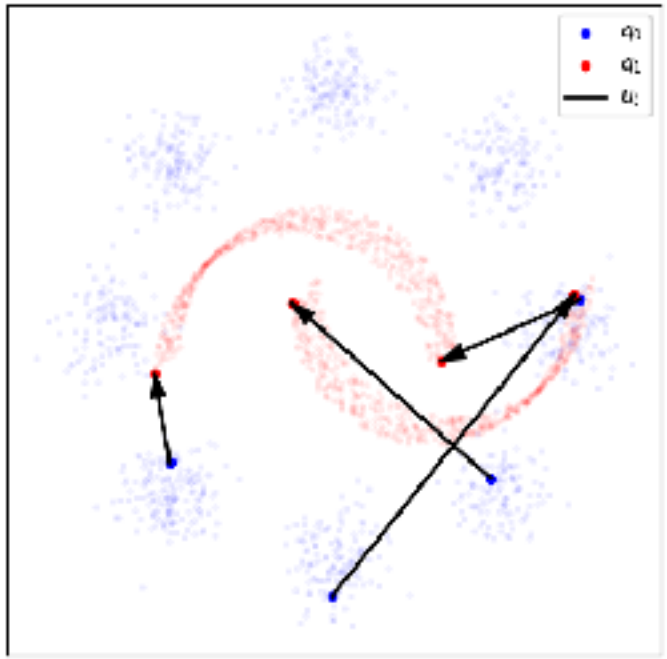
Basic Form a CFM



So how to we calculate x and u_t?

Firstly, we draw samples from $x_0 \sim q_0$ and $x_1 \sim q_1$ at random.

Basic Form a CFM

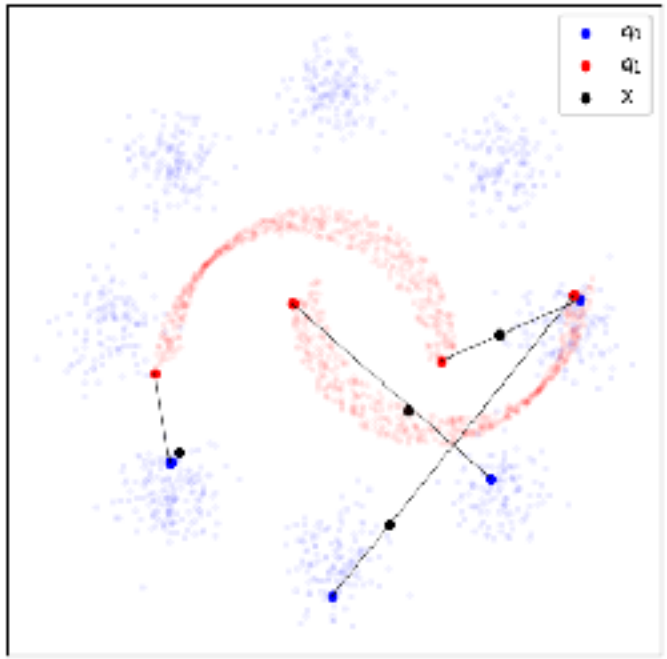


So how to we calculate x and u_t?

Firstly, we draw samples from $x_0 \sim q_0$ and $x_1 \sim q_1$ at random.

$$u_t(x|z) = x_1 - x_0.$$

Basic Form a CFM



So how to we calculate x and u_t?

Firstly, we draw samples from $x_0 \sim q_0$ and $x_1 \sim q_1$ at random.

$$u_t(x|z) = x_1 - x_0.$$
$$p_t(x|z) = \mathcal{N}(x \mid tx_1 + (1-t)x_0, \sigma^2)$$

where σ is a small amount of noise (≈0.01).

Basic Form a CFM

Algorithm 1 Conditional Flow Matching

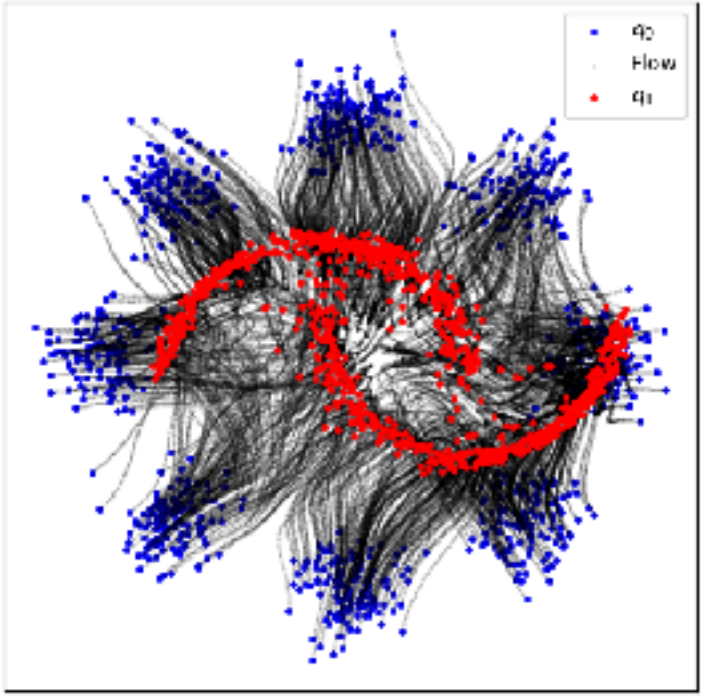
Input: Efficiently samplable $q(z)$, $p_t(x|z)$, and computable $u_t(x|z)$ and initial network v_θ.
while Training **do**
  $z \sim q(z)$; $t \sim \mathcal{U}(0,1)$; $x \sim p_t(x|z)$
  $\mathcal{L}_{\text{CFM}}(\theta) \leftarrow \|v_\theta(t,x) - u_t(x|z)\|^2$
  $\theta \leftarrow \text{Update}(\theta, \nabla_\theta \mathcal{L}_{\text{CFM}}(\theta))$
return v_θ

* https://github.com/harveymannering/boilerplate_code/blob/main/flow_matching_toy_example.ipynb

CFM

If we train a model like this, then use an ODE solver to map from we get something like this.

However, flow paths are curved, meaning we need more steps in for our ODE solver to accurately map between q_0 and q_1.



* https://github.com/harveymannering/boilerplate_code/blob/main/flow_matching_toy_example.ipynb

Connection to Rectified Flows

- CFM is equivalent to rectified flows, the only difference being that $\sigma=0$.
- State of the art text-to-image models currently use Rectified Flows (e.g. Stable Diffusion 3, FLUX.1, InstaFlow)

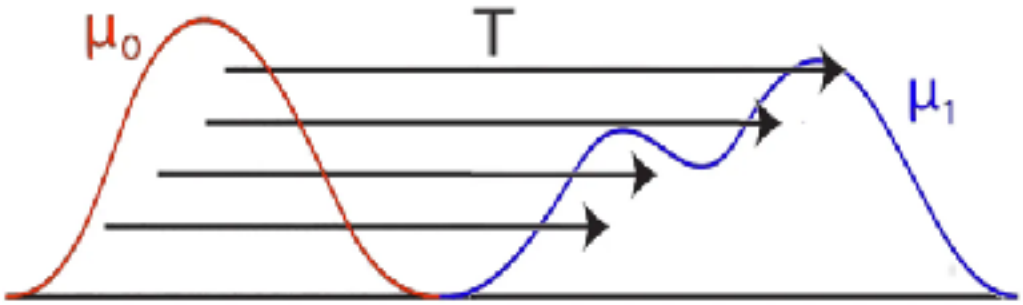
Image generated using FLUX.1 model

Optimal Transport

Optimal Transport

How can map one distribution to another in such a way that minimises some displacement cost?

2-Wasserstein distance is used to measure the transport cost between distributions. This distance can be expressed using our vector field u_t.

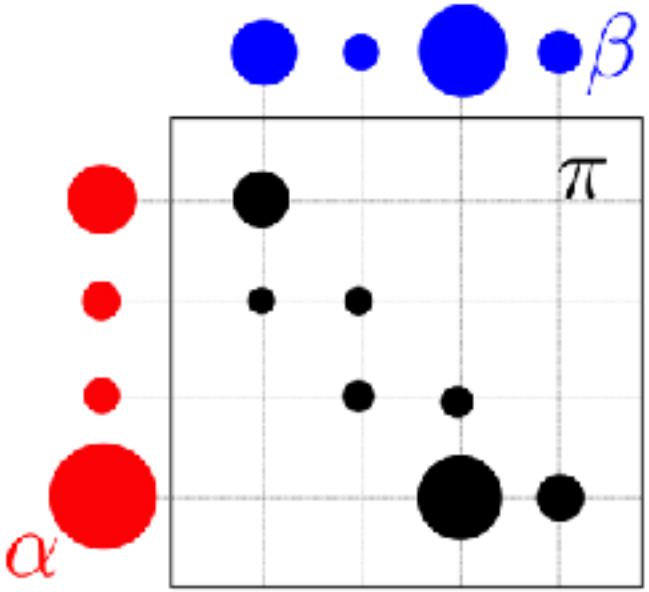


Optimal Transport

The coupling π, describes for every combination of point between two distributions, the optimal amount of mass to move between the two distributions.

π is sometimes called as a transport plan.

Sinkhorn algorithm can be used to fine π.



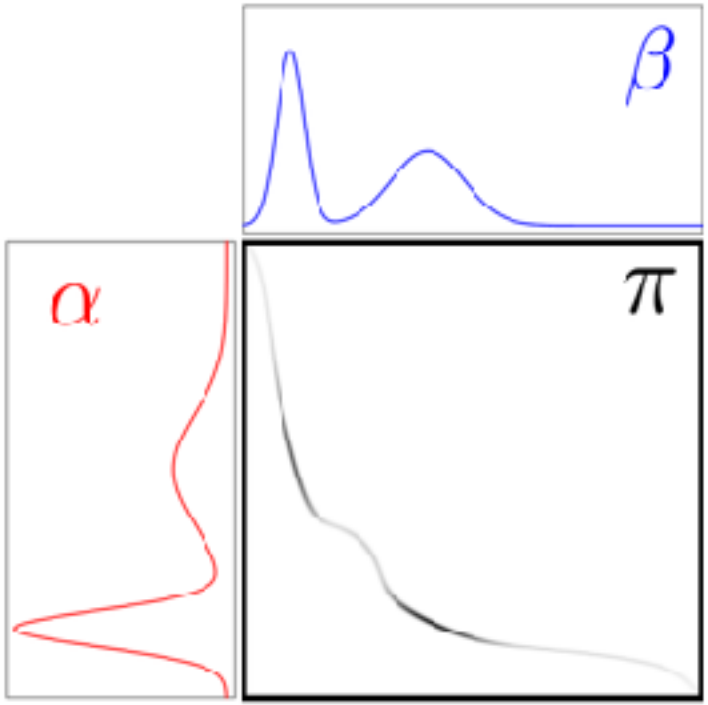
Peyré and Marco. "Computational optimal transport". 2019

Optimal Transport

The coupling π, describes for every combination of point between two distributions, the optimal amount of mass to move between the two distributions.

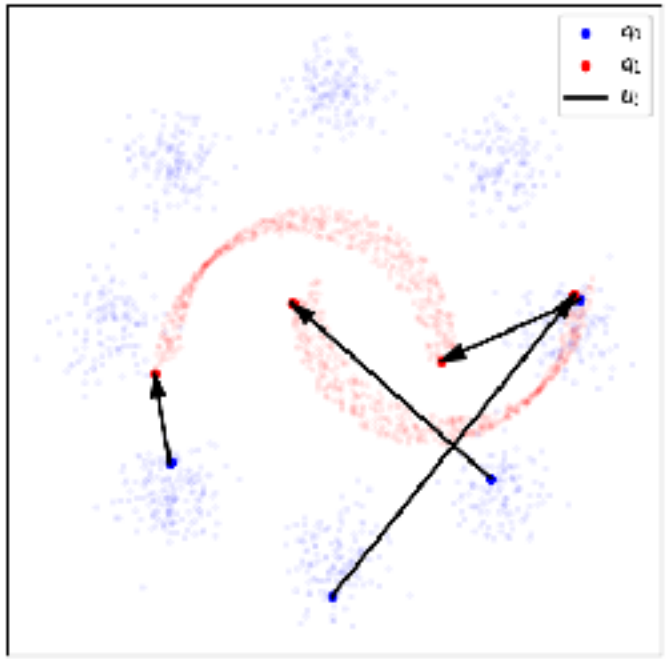
π is sometimes called as a transport plan.

Sinkhorn algorithm can be used to fine π.



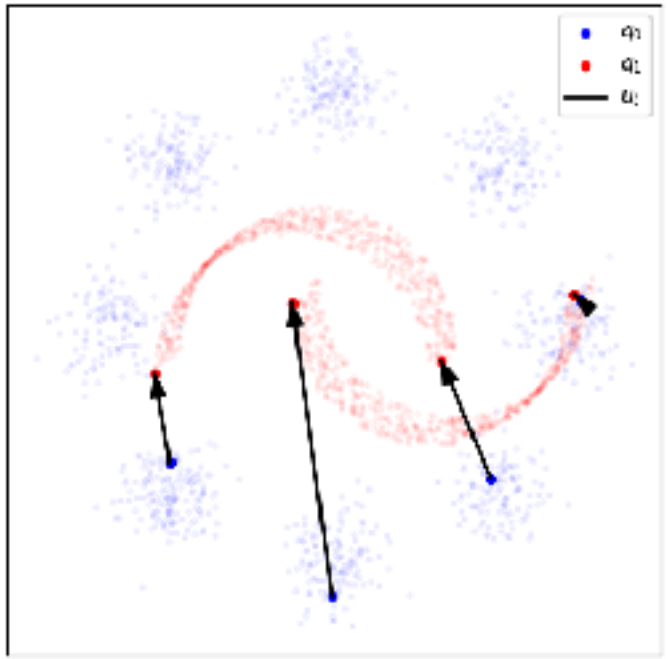
Peyré and Marco. "Computational optimal transport". 2019

OT-CFM



CFM randomly couples samples from $x_0 \sim q_0$ and $x_1 \sim q_1$.

OT-CFM



CFM randomly couples samples from $x_0 \sim q_0$ and $x_1 \sim q_1$.

Optimal Transport CFM (OT-CFM) uses the Sinkhorn algorithm to find optimal couplings between q_0 and q_1.

The Sinkhorn algorithm involve finding a transport plan π. This will be a matrix that how different parts of the distribution should be rearranged.

OT-CFM

OT-CFM straightens flow paths.

This makes training and inference more efficient.

If your flow paths are perfectly straight, you will only need one ODE step to map between q_0 and q_1.