AI Applications Lecture 13

Iterative Generation

SUZUKI, Atsushi[a]

[a] I thank my collaborator, Jing WANG, for cooperating in creating the teaching materials.

Outline

1. Introduction

2. Mathematical notation

3. What can we do? Why cannot we directly predict output?

4. Why iterative? How iterative?

5. What's the difference between reverse diffusion and flow matching?

6. Summary

Background

In Stable Diffusion 1.5, the reverse diffusion process is used to generate images in a low-dimensional latent space, whereas Qwen Image 2.1 and Hunyuan3D 2.1 use flow matching. Both are algorithms that iteratively apply a neural network (NN).

Background

In Stable Diffusion 1.5, the reverse diffusion process is used to generate images in a low-dimensional latent space, whereas Qwen Image 2.1 and Hunyuan3D 2.1 use flow matching. Both are algorithms that iteratively apply a neural network (NN).

Therefore, before studying NN architectures, we need to understand these iterative algorithms.

But

Background

In Stable Diffusion 1.5, the reverse diffusion process is used to generate images in a low-dimensional latent space, whereas Qwen Image 2.1 and Hunyuan3D 2.1 use flow matching. Both are algorithms that iteratively apply a neural network (NN).

Therefore, before studying NN architectures, we need to understand these iterative algorithms.

But why was it necessary to develop iterative algorithms in the first place? In other words, why not simply apply an NN once and directly regress the output using mean squared error?

Recap: pipeline components

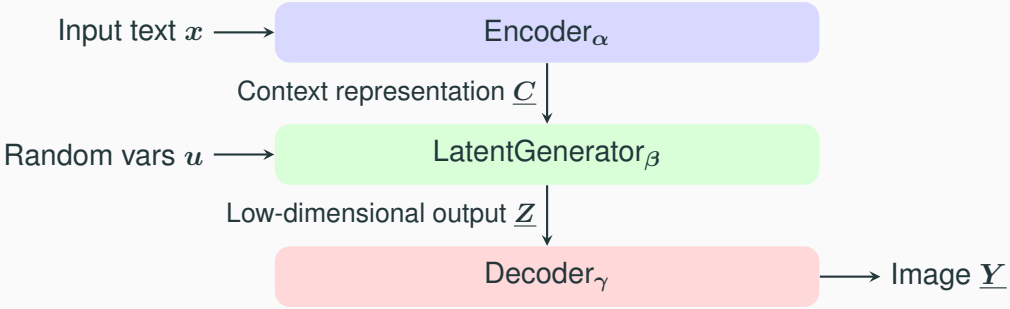


Figure 1: Latent generation pipeline.

Why not simply predict the low-dimensional output $\underline{Z}$ from the random variables u?

Learning outcomes

At the end of this lecture, you will be able to:

Learning outcomes

At the end of this lecture, you will be able to:

- **Explain** why direct conditional least-squares prediction collapses diversity and **justify** small-step updates.

Learning outcomes

At the end of this lecture, you will be able to:

- **Explain** why direct conditional least-squares prediction collapses diversity and **justify** small-step updates.
- **Construct** a general framework for iterative generation to avoid the issues in direct prediction.

Learning outcomes

At the end of this lecture, you will be able to:

- **Explain** why direct conditional least-squares prediction collapses diversity and **justify** small-step updates.
- **Construct** a general framework for iterative generation to avoid the issues in direct prediction.
- **Construct** practical iterative generators, including reverse diffusion (DDIM), flow matching (linear path + Euler).

Outline

1. **Introduction**
2. Mathematical notation
3. What can we do? Why cannot we directly predict output?
4. Why iterative? How iterative?
5. What's the difference between reverse diffusion and flow matching?
6. Summary

Outline

1. Introduction

2. Mathematical notation

3. What can we do? Why cannot we directly predict output?

4. Why iterative? How iterative?

5. What's the difference between reverse diffusion and flow matching?

6. Summary

Preliminaries: mathematical notations

Set:

- Sets: $\mathcal{A}$
- Membership: $x \in \mathcal{A}$
- Empty set: $\{\}$
- Roster notation: $\{a, b, c\}$
- Set-builder: $\{x \in \mathcal{A} \mid P(x)\}$
- Cardinality: $|\mathcal{A}|$
- Real numbers: $\mathbb{R}, \mathbb{R}_{>0}, \mathbb{R}_{\geq 0}$
- Integers: $\mathbb{Z}, \mathbb{Z}_{>0}, \mathbb{Z}_{\geq 0}$
- Integer range: $[1, k]_{\mathbb{Z}} \coloneqq \{1, \ldots, k\}$

Function:

- $f : \mathcal{X} \to \mathcal{Y}$: f maps from $\mathcal{X}$ to $\mathcal{Y}$.
- $y = f(x)$: The output of f for input x.

Definition:

- $(\mathrm{LHS}) \coloneqq (\mathrm{RHS})$: Left side is defined by the right side.

Preliminaries: random variables, time derivatives, and estimates

Random variables and their values

- Uppercase Z, C, W_t denote random variables (RVs).

Three related notations

- A dot denotes time differentiation: $\dot{a}(t) = da(t)/dt$.
- A bar denotes a true conditional mean: $\overline{z}_t(z, c) = \mathbb{E}[Z \mid W_t = z, C = c]$.
- A hat denotes an NN prediction: $\widehat{z}_\theta(t, z, c) \approx \overline{z}_t(z, c)$.

The same conventions apply to $\overline{\epsilon}_t, \overline{v}_t$ and to their learnable approximations $\widehat{\epsilon}_\theta, \widehat{v}_\theta$.

Outline

1. Introduction

2. Mathematical notation

3. What can we do? Why cannot we directly predict output?

4. Why iterative? How iterative?

5. What's the difference between reverse diffusion and flow matching?

6. Summary

Outline

1. Introduction

2. Mathematical notation

3. What can we do? Why cannot we directly predict output?

4. Why iterative? How iterative?

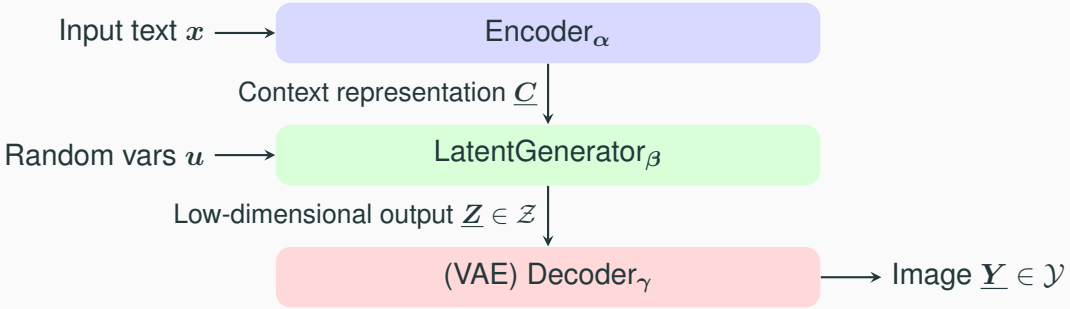
5. What's the difference between reverse diffusion and flow matching?

6. Summary

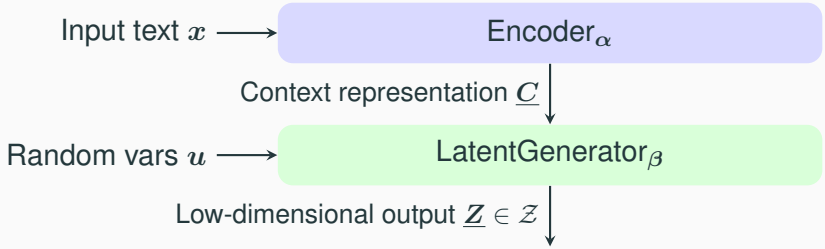
Question: In practical image generation, the generation process is complicated. Why

Question: In practical image generation, the generation process is complicated. Why don't we simply predict the output in image generation?

The pushforward approach



The pushforward approach



Objective: We learn the parameter β so that $\underline{Z} = \text{LatentGenerator}_{\beta}(u, \underline{C})$ follows a desired conditional distribution $p(\cdot \mid \underline{C})$, when we sample u from a source distribution λ.

What is a typical approach? Why can't we use the approach?

If we could construct a probability density function (PDF) p_β from LatentGenerator$_{\boldsymbol{\beta}}$ have a dataset $(\underline{\boldsymbol{Z}}_i, \underline{\boldsymbol{C}}_i)_{i=1}^m$, we could simply train β by the following minimization:

$$\underset{\boldsymbol{\beta}}{\text{Minimize}} -\frac{1}{m}\sum_{i=1}^{m} \log p_\beta(\underline{\boldsymbol{Z}}_i, \underline{\boldsymbol{C}}_i).$$

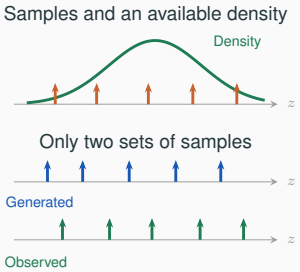
What is a typical approach? Why can't we use the approach?

If we could construct a probability density function (PDF) p_β from LatentGenerator$_\beta$ have a dataset $(\underline{\boldsymbol{Z}}_i, \underline{\boldsymbol{C}}_i)_{i=1}^m$, we could simply train β by the following minimization:

$$\underset{\boldsymbol{\beta}}{\text{Minimize}} -\frac{1}{m}\sum_{i=1}^{m} \log p_\beta(\underline{\boldsymbol{Z}}_i, \underline{\boldsymbol{C}}_i).$$

However, in the pushforward approach,

- We cannot construct a PDF.
- We can only sample outputs $\hat{\underline{\boldsymbol{Z}}}_i = \text{LatentGenerator}_\beta(\boldsymbol{u}, \underline{\boldsymbol{C}}_i)$ by sampling random variable $\boldsymbol{u}$.



Why not minimize a distance between the two distributions?

Now, we can only **sample** points from the real distribution and generated distribution.

Why not minimize a distance between the two distributions?

Now, we can only **sample** points from the real distribution and generated distribution.

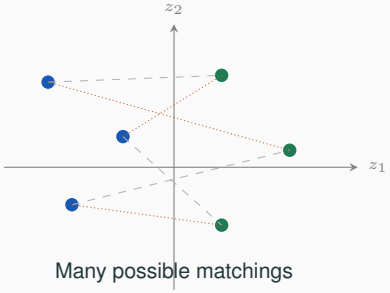
A naive approach is to minimize a distance between generated samples and observed samples. However,

Why not minimize a distance between the two distributions?

Now, we can only **sample** points from the real distribution and generated distribution.

A naive approach is to minimize a distance between generated samples and observed samples. However, simply computing the divergence is infeasible since it requires matching of points.

The matching is unknown



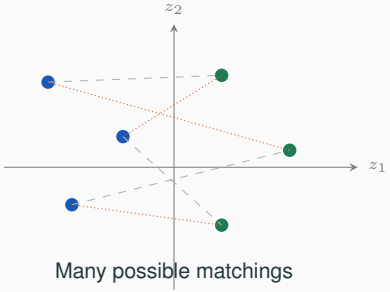
Many possible matchings

Why not minimize a distance between the two distributions?

Now, we can only **sample** points from the real distribution and generated distribution.

A naive approach is to minimize a distance between generated samples and observed samples. However, simply computing the divergence is infeasible since it requires matching of points.

The matching is unknown



Many possible matchings

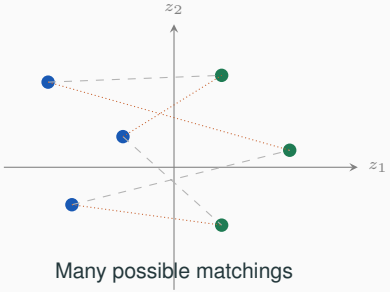
Recall that **GAN**'s idea [2] was to replace this distance computation by a NN discriminator. It works but adversarial optimization can exhibit instability [1].

Why not minimize a distance between the two distributions?

Now, we can only **sample** points from the real distribution and generated distribution.

A naive approach is to minimize a distance between generated samples and observed samples. However, simply computing the divergence is infeasible since it requires matching of points.

The matching is unknown



Many possible matchings

Recall that **GAN**'s idea [2] was to replace this distance computation by a NN discriminator. It works but adversarial optimization can exhibit instability [1].

Can't we construct a desired distribution without measuring distribution distance?

The safest regression target: conditional expectation

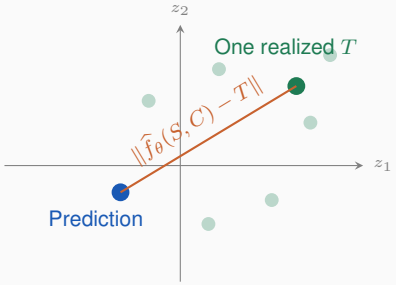
What we can natively do is regression. Specifically, given source RV S, context RV C, target RV T, we can construct a NN predicting T from S and C.

The safest regression target: conditional expectation

What we can natively do is regression. Specifically, given source RV S, context RV C, target RV T, we can construct a NN predicting T from S and C.

By minimizing the population loss $\mathcal{L}(\theta) = \mathbb{E}\|\widehat{f}_\theta(S, C) - T\|^2$, we get the **conditional expectation (mean)** $\overline{f}(s, c) = \mathbb{E}[T \mid S = s, C = c]$.

Predicting one random variable

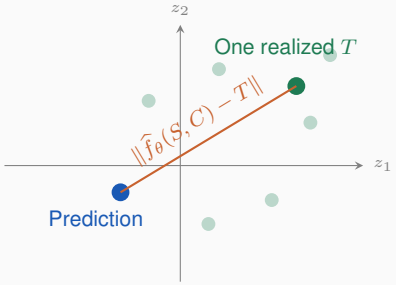


The safest regression target: conditional expectation

What we can natively do is regression. Specifically, given source RV S, context RV C, target RV T, we can construct a NN predicting T from S and C.

By minimizing the population loss $\mathcal{L}(\theta) = \mathbb{E}\|\widehat{f}_\theta(S, C) - T\|^2$, we get the **conditional expectation (mean)** $\overline{f}(s, c) = \mathbb{E}[T \mid S = s, C = c]$.

Predicting one random variable



Next question: Can we construct a desired distribution using conditional expectations?

Why do we avoid directly optimizing the output error?

We want to construct the data distribution using a NN f_θ by $\widehat{f}_\theta(\epsilon, c)$, where ϵ follows, e.g., the standard normal distribution.

Why do we avoid directly optimizing the output error?

We want to construct the data distribution using a NN f_θ by $\widehat{f}_\theta(\epsilon, c)$, where ϵ follows, e.g., the standard normal distribution.

Why not directly optimize $\mathbb{E}\|\widehat{f}_\theta(\epsilon, c) - Z\|^2$?



Why do we avoid directly optimizing the output error?

We want to construct the data distribution using a NN f_θ by $\widehat{f}_\theta(\epsilon, c)$, where ϵ follows, e.g., the standard normal distribution.



Why not directly optimize $\mathbb{E}\|\widehat{f}_\theta(\epsilon, c) - Z\|^2$?

Exercise

Let $\epsilon \sim \mathcal{N}(0, 1)$ and fix $C = c$,

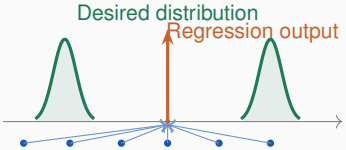
$$Z \sim \tfrac{1}{2}\mathcal{N}(1, 0.25^2) + \tfrac{1}{2}\mathcal{N}(5, 0.25^2),$$

independent. Train $\widehat{f}_\theta(\epsilon, c)$ with squared loss to predict Z and then push forward $\mathcal{N}(0, 1)$ by this fitted function. What is the generated distribution?

A one-step predictor can erase all diversity

Answer: Since Z and ϵ are independent conditional on C, for every e,

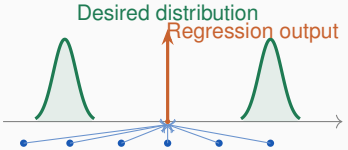
$$\begin{aligned}\mathbb{E}[Z \mid \epsilon = e, C = c] &= \int_{\mathbb{R}} z\, p_{Z|\epsilon,C}(z \mid e, c)\, dz \\ &= \int_{\mathbb{R}} z\, p_{Z|C}(z \mid c)\, dz \\ &= \tfrac{1}{2} \cdot 1 + \tfrac{1}{2} \cdot 5 = 3.\end{aligned}$$



A one-step predictor can erase all diversity

Answer: Since Z and ϵ are independent conditional on C, for every e,

$$\begin{aligned}\mathbb{E}[Z \mid \epsilon = e, C = c] &= \int_{\mathbb{R}} z \, p_{Z|\epsilon,C}(z \mid e, c)\, dz \\ &= \int_{\mathbb{R}} z \, p_{Z|C}(z \mid c)\, dz \\ &= \tfrac{1}{2} \cdot 1 + \tfrac{1}{2} \cdot 5 = 3.\end{aligned}$$

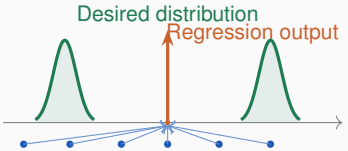


Therefore the population-optimal predictor is the constant $\widehat{f}_\theta(\epsilon, c) = 3$, and the pushforward of $\mathcal{N}(0, 1)$ is δ_3.

A one-step predictor can erase all diversity

Answer: Since Z and ϵ are independent conditional on C, for every e,

$$\begin{aligned}\mathbb{E}[Z \mid \epsilon = e, C = c] &= \int_{\mathbb{R}} z \, p_{Z|\epsilon,C}(z \mid e, c)\, dz \\ &= \int_{\mathbb{R}} z \, p_{Z|C}(z \mid c)\, dz \\ &= \tfrac{1}{2} \cdot 1 + \tfrac{1}{2} \cdot 5 = 3.\end{aligned}$$



Therefore the population-optimal predictor is the constant $\widehat{f}_\theta(\epsilon, c) = 3$, and the pushforward of $\mathcal{N}(0, 1)$ is δ_3.

Every source value is sent to the same expectation value. It **degenerates** the distribution.

From one-step predictor to iterative generator.

Why did it fail?

From one-step predictor to iterative generator.

Why did it fail?

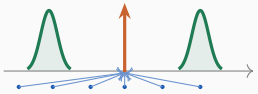
- It tried to approximate the non-localized conditional distribution by a single value, i.e., the conditional expectation.

From one-step predictor to iterative generator.

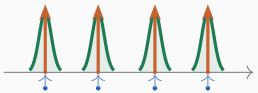
Why did it fail?

- It tried to approximate the non-localized conditional distribution by a single value, i.e., the conditional expectation.
- It tried to predict an independent RV. The conditional expectation was the same for all ϵ.

Predicting independent non-localized distribution



Predicting localized interdependent distributions



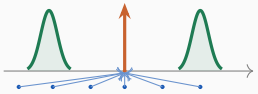
From one-step predictor to iterative generator.

Why did it fail?

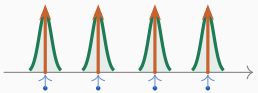
- It tried to approximate the non-localized conditional distribution by a single value, i.e., the conditional expectation.
- It tried to predict an independent RV. The conditional expectation was the same for all ϵ.

Let's do the opposite!

Predicting independent non-localized distribution



Predicting localized interdependent distributions

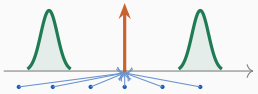


From one-step predictor to iterative generator.

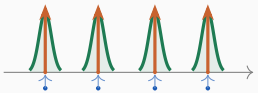
Why did it fail?

- It tried to approximate the non-localized conditional distribution by a single value, i.e., the conditional expectation.
- It tried to predict an independent RV. The conditional expectation was the same for all ϵ.

Predicting independent non-localized distribution



Predicting localized interdependent distributions



Let's do the opposite!

Idea: Construct the objective distribution by a **chain of localized and interdependent conditional distributions**.

Idea: chain localized conditional distributions

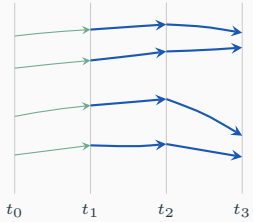
Construct a sequence of random variables $W_0 = \epsilon, W_{t_1}, W_{t_2}, ..., W_{t_{K-1}}, W_1 = Z$ such that $W_{t_k}, W_{t_{k+1}}$ are strongly interdependent and the conditional distribution is localized.

Idea: chain localized conditional distributions

Construct a sequence of random variables $W_0 = \epsilon, W_{t_1}, W_{t_2}, ..., W_{t_{K-1}}, W_1 = Z$ such that $W_{t_k}, W_{t_{k+1}}$ are strongly interdependent and the conditional distribution is localized.

Then we can expect the pushforward by the conditional expectation $\overline{w}_{t_{k+1}|t_k}(w, c) = \mathbb{E}[W_{t_{k+1}} \mid W_{t_k} = w, C = c]$ to approximate the conditional distribution $p_{W_{t_{k+1}}|W_{t_k},C}$.

Chain of conditional expectations

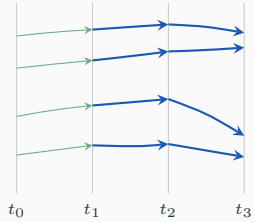


Idea: chain localized conditional distributions

Construct a sequence of random variables $W_0 = \epsilon, W_{t_1}, W_{t_2}, ..., W_{t_{K-1}}, W_1 = Z$ such that $W_{t_k}, W_{t_{k+1}}$ are strongly interdependent and the conditional distribution is localized.

Then we can expect the pushforward by the conditional expectation $\overline{w}_{t_{k+1}|t_k}(w, c) = \mathbb{E}[W_{t_{k+1}} \mid W_{t_k} = w, C = c]$ to approximate the conditional distribution $p_{W_{t_{k+1}}|W_{t_k},C}$.

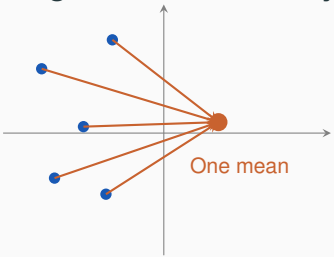
Chain of conditional expectations



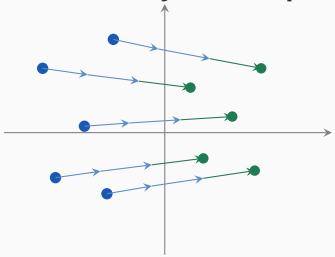
As a result, the chain of the conditional expectations may approximate the conditional distribution $p_{W_1|W_0,C} = p_{Z|\epsilon,C} = p_{Z|C}$.

One long mean jump versus a chain of local mean jumps

One long conditional-mean jump



Chain of nearby mean updates



Long regression collapses independent branches; nearby dependent states allow the mean updates to retain a spread of samples.

One step prediction error minimization does not work - Answer

Question: Why don't we simply predict the output in image generation?

Answer: If we minimize the one step prediction error, it degenerates and cannot represent the objective distribution well. This is because,

One step prediction error minimization does not work - Answer

Question: Why don't we simply predict the output in image generation?

Answer: If we minimize the one step prediction error, it degenerates and cannot represent the objective distribution well. This is because, the input and output are independent, and the output distribution is not localized.

Outline

1. Introduction

2. Mathematical notation

3. What can we do? Why cannot we directly predict output?

4. Why iterative? How iterative?

5. What's the difference between reverse diffusion and flow matching?

6. Summary

Outline

1. Introduction

2. Mathematical notation

3. What can we do? Why cannot we directly predict output?

4. Why iterative? How iterative?

5. What's the difference between reverse diffusion and flow matching?

6. Summary

Question: A direct regression fails. Why can an iterative generation avoid the issues?

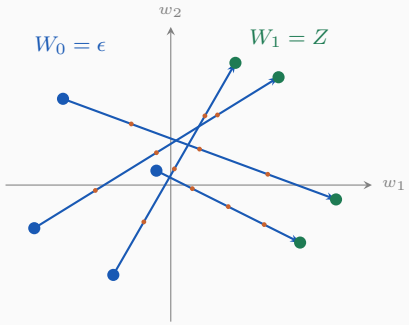
Construct intermediate distributions with available samples

From the target RV Z and source noise ϵ, we design a smooth path:

$$W_t = a(t)\epsilon + b(t)Z, \quad W_0 = \epsilon, \quad W_1 = Z.$$

Here $a(0) = 1$, $b(0) = 0$, $a(1) = 0$, $b(1) = 1$ and both functions a and b are smooth so that for nearby times:

Two-dimensional sample paths



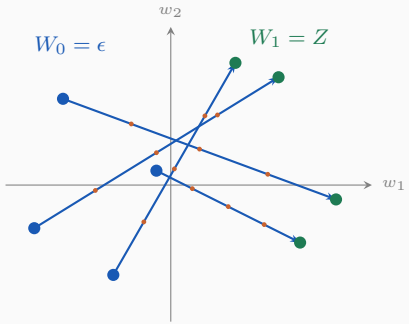
Construct intermediate distributions with available samples

From the target RV Z and source noise ϵ, we design a smooth path:

$$W_t = a(t)\epsilon + b(t)Z, \quad W_0 = \epsilon, \quad W_1 = Z.$$

Here $a(0) = 1$, $b(0) = 0$, $a(1) = 0$, $b(1) = 1$ and both functions a and b are smooth so that for nearby times:

Two-dimensional sample paths



- The samples are strongly interdependent, and
- The conditional distribution is localized.

Hence, chaining $\overline{w}_{t_{k+1}|t_k}(w, c) := \mathbb{E}[W_{t_{k+1}} \mid W_{t_k} = w, C = c]$ hopefully works well!

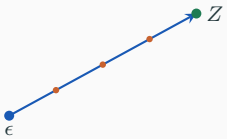
Example of paths

Linear path (flow matching)

A standard choice in flow matching is

$$W_t = (1-t)\epsilon + tZ.$$

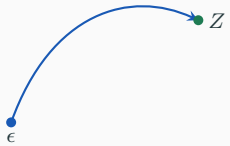
For each sampled pair (ϵ, Z), the trajectory is a straight line from ϵ to Z.



Cosine–sine path (diffusion models)

A constant square weight sum path:

$$W_t = \cos\left(\frac{\pi t}{2}\right)\epsilon + \sin\left(\frac{\pi t}{2}\right)Z.$$



Advantage of the constant square weight sum path

A constant square weight sum path:

$$W_t = \cos\left(\frac{\pi t}{2}\right)\epsilon + \sin\left(\frac{\pi t}{2}\right)Z.$$

If $\epsilon \perp (Z, C)$, $\mathrm{Cov}(\epsilon) = I$, and $\mathrm{Cov}(Z \mid C = c) = \mathbf{\Sigma}$, then

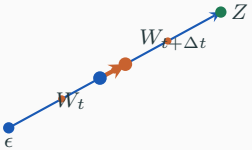
$$\mathrm{Cov}(W_t \mid C = c) = \cos^2\left(\frac{\pi t}{2}\right) I + \sin^2\left(\frac{\pi t}{2}\right)\mathbf{\Sigma}.$$

Thus, if $\mathbf{\Sigma} = I$, the covariance is I for every t.

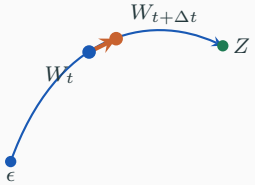
Nearby times gives nearby samples on a smooth path

For a smooth path, when Δt is small, W_t and $W_{t+\Delta t}$ are close.

Linear path
(default in flow matching)



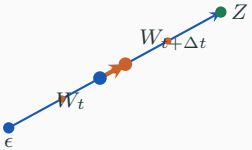
Cosine–sine path
(default in reverse diffusion)



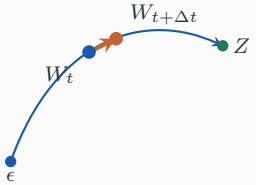
Nearby times gives nearby samples on a smooth path

For a smooth path, when Δt is small, W_t and $W_{t+\Delta t}$ are close.

Linear path
(default in flow matching)



Cosine–sine path
(default in reverse diffusion)



Hence, we can expect the conditional distribution to be localized and can be well approximated by the pushforward to the conditional expectation.

What should a NN learn?

Does a NN need to learn $\overline{w}_{t_{k+1}|t_k}(w, c) := \mathbb{E}[W_{t_{k+1}} \mid W_{t_k} = w, C = c]$ for all k?

What should a NN learn?

Does a NN need to learn $\overline{w}_{t_{k+1}|t_k}(w,c) := \mathbb{E}[W_{t_{k+1}} \mid W_{t_k} = w, C = c]$ for all k?

Not necessary! If you can construct

- $\overline{\epsilon}_t(w,c) := \mathbb{E}[\epsilon \mid W_t = w, C = c]$ and
- $\overline{z}_t(w,c) := \mathbb{E}[Z \mid W_t = w, C = c]$,

then you can compute everything you want! Specifically,

$$\mathbb{E}[r\epsilon + qZ \mid W_t = w, C = c] = r\overline{\epsilon}_t(w,c) + q\overline{z}_t(w,c).$$

What should a NN learn?

Does a NN need to learn $\overline{w}_{t_{k+1}|t_k}(w,c) := \mathbb{E}[W_{t_{k+1}} \mid W_{t_k} = w, C = c]$ for all k?

Not necessary! If you can construct

- $\overline{\epsilon}_t(w,c) := \mathbb{E}[\epsilon \mid W_t = w, C = c]$ and
- $\overline{z}_t(w,c) := \mathbb{E}[Z \mid W_t = w, C = c]$,

then you can compute everything you want! Specifically,

$$\mathbb{E}[r\epsilon + qZ \mid W_t = w, C = c] = r\overline{\epsilon}_t(w,c) + q\overline{z}_t(w,c).$$

Exercise

Find $\overline{w}_{t_{k+1}|t_k}$ from $\overline{\epsilon}_t$, $\overline{Z}_t$, a, and b. (Recall $W_t = a(t)\epsilon + b(t)Z$).

We can compute conditional expectation

Answer: Recall that $\overline{w}_{t_{k+1}|t_k}(w,c) := \mathbb{E}[W_{t_{k+1}} \mid W_{t_k} = w, C = c]$.

Since $W_t = a(t)\epsilon + b(t)Z$, linearity of conditional expectation gives

$$\overline{w}_{t_{k+1}|t_k}(w,c) = a(t_{k+1})\overline{\epsilon}_{t_k}(w,c) + b(t_{k+1})\overline{z}_{t_k}(w,c).$$

What are reverse diffusion and flow matching doing?

In fact, both reverse diffusion and flow matching construct a NN to estimate the noise expectation $\overline{\epsilon}_{t_k}(w, c)$ and data expectation $\overline{z}_{t_k}(w, c)$.

Here,

What are reverse diffusion and flow matching doing?

In fact, both reverse diffusion and flow matching construct a NN to estimate the noise expectation $\overline{\epsilon}_{t_k}(w, c)$ and data expectation $\overline{z}_{t_k}(w, c)$.

Here, estimating both is redundant. The difference between reverse diffusion and flow matching is in how to estimate the minimal information to construct both $\overline{\epsilon}_{t_k}(w, c)$ and $\overline{z}_{t_k}(w, c)$.

The advantage of iterative generation.

Question: A direct regression fails. Why can an iterative generation avoid the issues?

The advantage of iterative generation.

Question: A direct regression fails. Why can an iterative generation avoid the issues?

Answer: Construct a smooth path from the noise to the data. Then, nearby points are strongly interdependent and the conditional distribution is strongly localized. Hence, the iterative pushforward to the conditional expectation can approximate the conditional distribution well.

Outline

1. Introduction

2. Mathematical notation

3. What can we do? Why cannot we directly predict output?

4. Why iterative? How iterative?

5. What's the difference between reverse diffusion and flow matching?

6. Summary

Outline

1. Introduction

2. Mathematical notation

3. What can we do? Why cannot we directly predict output?

4. Why iterative? How iterative?

5. What's the difference between reverse diffusion and flow matching?

6. Summary

Question: We want to estimate the conditional expectation $\overline{w}_{t_{k+1}|t_k}(w, c)$ on a smooth path. Both reverse diffusion and flow matching seem to do this, but

Question: We want to estimate the conditional expectation $\overline{w}_{t_{k+1}|t_k}(w, c)$ on a smooth path. Both reverse diffusion and flow matching seem to do this, but what's the difference?

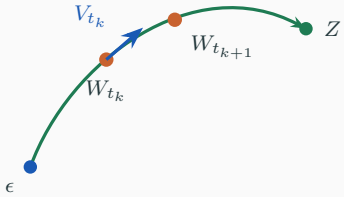
Reverse diffusion and flow matching do the same thing in different ways

Diffusion model
NN estimates
the noise $\overline{\epsilon}_{t_k}$

Flow matching
NN estimates
the velocity $\overline{v}_{t_k}$

Construct endpoint
conditional means
$\overline{\epsilon}_{t_k}(w, c),\ \overline{z}_{t_k}(w, c)$

Construct the next-
state conditional mean
$\overline{w}_{t_{k+1}|t_k}(w, c)$



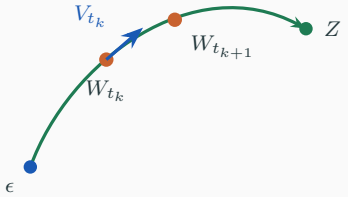
Reverse diffusion and flow matching do the same thing in different ways

Diffusion model
NN estimates
the noise $\overline{\epsilon}_{t_k}$

Flow matching
NN estimates
the velocity $\overline{v}_{t_k}$

Construct endpoint
conditional means
$\overline{\epsilon}_{t_k}(w, c),\ \overline{z}_{t_k}(w, c)$

Construct the next-
state conditional mean
$\overline{w}_{t_{k+1}|t_k}(w, c)$



Note: In implementations, those algorithms skip redundant parts.
The above diagram just explains the concept.

Reverse diffusion from a noise estimator - The noise expectation suffices!

We don't need both $\bar{\epsilon}_t$ and $\overline{Z}_t$, since we can construct $\overline{Z}_t$ from $\bar{\epsilon}_t$.

Reverse diffusion from a noise estimator - The noise expectation suffices!

We don't need both $\overline{\epsilon}_t$ and $\overline{Z}_t$, since we can construct $\overline{Z}_t$ from $\overline{\epsilon}_t$.

Exercise

Given $W_t = a(t)\epsilon + b(t)Z$, use conditional expectation to express $\overline{z}_t(w, c) = \mathbb{E}[Z \mid W_t = w, C = c]$ in terms of w, $a(t)$, $b(t)$, and $\overline{\epsilon}_t(w, c)$, assuming $b(t) \neq 0$.

Also express $\overline{w}_{t_{k+1}|t_k}(w, c) = \mathbb{E}[W_{t_{k+1}} \mid W_{t_k} = w, C = c]$ in terms of $\overline{\epsilon}_{t_k}$, a, and b.

Hint: $Z = [W_t - a(t)\epsilon]/b(t)$.

Reverse diffusion from a noise estimator - The noise expectation suffices!

Answer: Taking the conditional expectation of $W_t = a(t)\epsilon + b(t)Z$ given $W_t = w, C = c$ gives $w = a(t)\overline{\epsilon}_t(w, c) + b(t)\overline{z}_t(w, c)$.

Reverse diffusion from a noise estimator - The noise expectation suffices!

Answer: Taking the conditional expectation of $W_t = a(t)\epsilon + b(t)Z$ given $W_t = w, C = c$ gives $w = a(t)\overline{\epsilon}_t(w, c) + b(t)\overline{z}_t(w, c)$.

Therefore, for $b(t) \neq 0$,

$$\overline{z}_t(w, c) = \frac{w - a(t)\overline{\epsilon}_t(w, c)}{b(t)}.$$

Reverse diffusion from a noise estimator - The noise expectation suffices!

Answer: Taking the conditional expectation of $W_t = a(t)\epsilon + b(t)Z$ given $W_t = w, C = c$ gives $w = a(t)\overline{\epsilon}_t(w,c) + b(t)\overline{z}_t(w,c)$.

Therefore, for $b(t) \neq 0$,

$$\overline{z}_t(w,c) = \frac{w - a(t)\overline{\epsilon}_t(w,c)}{b(t)}.$$

Consequently, for $b(t_k) \neq 0$,

$$\begin{aligned}
\overline{w}_{t_{k+1}|t_k}(w,c) &= a(t_{k+1})\overline{\epsilon}_{t_k}(w,c) + b(t_{k+1})\overline{z}_{t_k}(w,c) \\
&= a(t_{k+1})\overline{\epsilon}_{t_k}(w,c) + \frac{b(t_{k+1})}{b(t_k)}\big[w - a(t_k)\overline{\epsilon}_{t_k}(w,c)\big] \\
&= \big[a(t_{k+1}) - \frac{a(t_k)b(t_{k+1})}{b(t_k)}\big]\overline{\epsilon}_{t_k}(w,c) + \frac{b(t_{k+1})}{b(t_k)}w
\end{aligned}$$

Reverse diffusion using a noise estimator NN

The idea of reverse diffusion using a noise estimator NN $\widehat{\epsilon}_\theta$ is the following (DDIM):

Reverse diffusion using a noise estimator NN

The idea of reverse diffusion using a noise estimator NN $\widehat{\epsilon}_\theta$ is the following (DDIM):

- Train a NN that satisfies $\widehat{\epsilon}_\theta(w, c, t) \approx \overline{\epsilon}_t(w, c) := \mathbb{E}[\epsilon \mid W_t = w, C = c]$.
- For each step $t_k \to t_{k+1}$, construct $\widehat{w}_{t_{k+1}|t_k}(w, c) = \left[a(t_{k+1}) - \frac{a(t_k)b(t_{k+1})}{b(t_k)}\right]\widehat{\epsilon}_\theta(w, c, t) + \frac{b(t_{k+1})}{b(t_k)}w$.
- Starting from a noisy initial state, repeatedly apply $w \leftarrow \widehat{w}_{t_{k+1}|t_k}(w, c)$ to obtain a generated latent Z.

Reverse diffusion using a noise estimator NN - Training on square distance

More specifically, the basic training phase is as follows.

Reverse diffusion using a noise estimator NN - Training on square distance

More specifically, the basic training phase is as follows.

- Prepare training dataset $(c_b, z_b)_{b=1}^B$.
- For each b, independently sample $t_b \sim \mathrm{Uniform}(0,1)$ and $\epsilon_b \sim \mathcal{N}(0, I)$; set $w_b = a(t_b)\epsilon_b + b(t_b)z_b$.
- Update θ by minimizing the squared loss $\frac{1}{B}\sum_{b=1}^B \|\widehat{\epsilon}_\theta(w_b, c_b, t_b) - \epsilon_b\|^2$.
- Repeat the sampling and parameter updates.

Reverse diffusion using a noise estimator NN - Iterative generation

Then, in the inference phase, we conduct the following.

Reverse diffusion using a noise estimator NN - Iterative generation

Then, in the inference phase, we conduct the following.

- **Input**: The context tensor c, schedule $0 < t_0 < \cdots < t_K = 1$ with $b(t_k) \neq 0$ for $k < K$, coefficients a, b, and trained $\widehat{\epsilon}_\theta$.
- Initialize $w \sim \mathcal{N}(0, I)$, approximating the distribution of W_{t_0} for small $t_0 > 0$.
- **For** $k \leftarrow 1, 2, \ldots, K$:
- $\quad w \leftarrow \frac{b(t_k)}{b(t_{k-1})} w + \left[a(t_k) - \frac{a(t_{k-1}) b(t_k)}{b(t_{k-1})} \right] \widehat{\epsilon}_\theta(w, c, t_{k-1})$.
- **Output**: w as the generated latent Z.

Flow matching - The velocity expectation suffices!

In flow matching, we consider the path velocity $V_t = \dot{W}_t = \dot{a}(t)\epsilon + \dot{b}(t)Z$, where the dot indicates the time derivative.

Flow matching - The velocity expectation suffices!

In flow matching, we consider the path velocity $V_t = \dot{W}_t = \dot{a}(t)\epsilon + \dot{b}(t)Z$, where the dot indicates the time derivative.

Similar to the reverse diffusion, we can construct $\overline{\epsilon}_t$ and $\overline{Z}_t$,
$\overline{v}_t = \mathbb{E}[V_t \mid W_t = w, C = c]$.

Flow matching - The velocity expectation suffices!

In flow matching, we consider the path velocity $V_t = \dot{W}_t = \dot{a}(t)\epsilon + \dot{b}(t)Z$, where the dot indicates the time derivative.

Similar to the reverse diffusion, we can construct $\overline{\epsilon}_t$ and $\overline{Z}_t$,
$\overline{v}_t = \mathbb{E}[V_t \mid W_t = w, C = c]$.

First, let's consider the linear path $W_t = (1-t)\epsilon + tZ$.

Flow matching - The velocity expectation suffices!

In flow matching, we consider the path velocity $V_t = \dot{W}_t = \dot{a}(t)\epsilon + \dot{b}(t)Z$, where the dot indicates the time derivative.

Similar to the reverse diffusion, we can construct $\overline{\epsilon}_t$ and $\overline{Z}_t$, $\overline{v}_t = \mathbb{E}[V_t \mid W_t = w, C = c]$.

First, let's consider the linear path $W_t = (1-t)\epsilon + tZ$.

Exercise

For the linear path above, express $\overline{\epsilon}_t(w, c)$ and $\overline{z}_t(w, c)$ in terms of w and $\overline{v}_t(w, c)$. Then express $\overline{w}_{t_{k+1}|t_k}(w, c)$ in terms of $\overline{v}_{t_k}$.

Hint: $V_t = Z - \epsilon$, so $\epsilon = W_t - tV_t$ and $Z = W_t + (1-t)V_t$.

Flow matching - The velocity expectation suffices!

Answer: Since $W_t = (1-t)\epsilon + tZ$ and $V_t = Z - \epsilon$, conditioning on $W_t = w, C = c$ gives $w = (1-t)\overline{\epsilon}_t + t\overline{z}_t$ and $\overline{v}_t = \overline{z}_t - \overline{\epsilon}_t$.

Flow matching - The velocity expectation suffices!

Answer: Since $W_t = (1-t)\epsilon + tZ$ and $V_t = Z - \epsilon$, conditioning on $W_t = w, C = c$ gives $w = (1-t)\overline{\epsilon}_t + t\overline{z}_t$ and $\overline{v}_t = \overline{z}_t - \overline{\epsilon}_t$.

Therefore,

$$\overline{\epsilon}_t(w,c) = w - t\overline{v}_t(w,c), \qquad \overline{z}_t(w,c) = w + (1-t)\overline{v}_t(w,c).$$

Flow matching - The velocity expectation suffices!

Answer: Since $W_t = (1-t)\epsilon + tZ$ and $V_t = Z - \epsilon$, conditioning on $W_t = w, C = c$ gives $w = (1-t)\overline{\epsilon}_t + t\overline{z}_t$ and $\overline{v}_t = \overline{z}_t - \overline{\epsilon}_t$.

Therefore,

$$\overline{\epsilon}_t(w,c) = w - t\overline{v}_t(w,c), \qquad \overline{z}_t(w,c) = w + (1-t)\overline{v}_t(w,c).$$

Consequently, writing $t = t_k$ and $s = t_{k+1}$,

$$\begin{aligned}\overline{w}_{s|t}(w,c) &= (1-s)\overline{\epsilon}_t(w,c) + s\overline{z}_t(w,c) \\ &= w + (s-t)\overline{v}_t(w,c).\end{aligned}$$

Flow matching using the velocity estimator (linear path case)

The idea of flow matching using a velocity estimator NN $\widehat{v}_\theta$ is the following:

Flow matching using the velocity estimator (linear path case)

The idea of flow matching using a velocity estimator NN $\widehat{v}_\theta$ is the following:

- Train a NN that satisfies $\widehat{v}_\theta(w, c, t) \approx \overline{v}_t(w, c) := \mathbb{E}[Z - \epsilon \mid W_t = w, C = c]$.
- For each step $t_k \to t_{k+1}$, construct $\widehat{w}_{t_{k+1}|t_k}(w, c) = w + (t_{k+1} - t_k)\widehat{v}_\theta(w, c, t_k)$ (Euler update).
- Starting from $w \sim \mathcal{N}(0, I)$ at $t_0 = 0$, repeatedly apply $w \leftarrow \widehat{w}_{t_{k+1}|t_k}(w, c)$ to obtain a generated latent Z at $t_K = 1$.

Flow matching using a velocity estimator NN - Training on square distance

More specifically, the basic training phase is as follows.

Flow matching using a velocity estimator NN - Training on square distance

More specifically, the basic training phase is as follows.

- Prepare training dataset $(c_b, z_b)_{b=1}^B$.
- For each b, independently sample $t_b \sim \mathrm{Uniform}(0,1)$ and $\epsilon_b \sim \mathcal{N}(0, I)$; set $w_b = (1-t_b)\epsilon_b + t_b z_b$ and $v_b = z_b - \epsilon_b$.
- Update θ by minimizing the squared loss $\frac{1}{B}\sum_{b=1}^B \|\widehat{v}_\theta(w_b, c_b, t_b) - v_b\|^2$.
- Repeat the sampling and parameter updates.

Flow matching using a velocity estimator NN - Iterative generation

Then, in the inference phase, we conduct the following.

Flow matching using a velocity estimator NN - Iterative generation

Then, in the inference phase, we conduct the following.

- **Input**: The context tensor c, schedule $0 = t_0 < \cdots < t_K = 1$, and trained $\widehat{v}_\theta$.
- Initialize $w \sim \mathcal{N}(0, I)$, the distribution of W_{t_0}.
- **For** $k \leftarrow 1, 2, \ldots, K$:
- $\quad w \leftarrow w + (t_k - t_{k-1})\widehat{v}_\theta(w, c, t_{k-1})$.
- **Output**: w as the generated latent Z.

Flow matching - General cases

Let's consider a general path.

Exercise

Given $W_t = a(t)\epsilon + b(t)Z$ and $V_t = \dot{a}(t)\epsilon + \dot{b}(t)Z$, express $\overline{\epsilon}_t(w, c)$ and $\overline{z}_t(w, c)$ using w and $\overline{v}_t(w, c)$. Then express $\overline{w}_{t_{k+1}|t_k}(w, c)$ using $\overline{v}_{t_k}(w, c)$ and the path coefficients. Assume $\Delta_t := a(t)\dot{b}(t) - \dot{a}(t)b(t) \neq 0$.

Hint: Take conditional expectations of the equations for W_t and V_t, and solve the resulting two linear equations.

Flow matching - The velocity expectation suffices!

Answer: Conditioning on $W_t = w, C = c$ gives $w = a(t)\overline{\epsilon}_t + b(t)\overline{z}_t$ and $\overline{v}_t = \dot{a}(t)\overline{\epsilon}_t + \dot{b}(t)\overline{z}_t$. Hence, for $\Delta_t \neq 0$,

$$\overline{\epsilon}_t(w,c) = \frac{\dot{b}(t)w - b(t)\overline{v}_t(w,c)}{\Delta_t}, \qquad \overline{z}_t(w,c) = \frac{a(t)\overline{v}_t(w,c) - \dot{a}(t)w}{\Delta_t}.$$

Consequently, writing $t = t_k$ and $s = t_{k+1}$,

$$\begin{aligned}\overline{w}_{s|t}(w,c) &= a(s)\overline{\epsilon}_t(w,c) + b(s)\overline{z}_t(w,c) \\ &= \frac{a(s)\dot{b}(t) - b(s)\dot{a}(t)}{\Delta_t}\, w + \frac{a(t)b(s) - b(t)a(s)}{\Delta_t}\, \overline{v}_t(w,c).\end{aligned}$$

Flow matching using the velocity estimator

The idea of flow matching using a velocity estimator NN $\widehat{v}_\theta$ is the following:

- Train a NN that satisfies $\widehat{v}_\theta(w, c, t) \approx \overline{v}_t(w, c) := \mathbb{E}[V_t \mid W_t = w, C = c]$.
- For each step $t_k \to t_{k+1}$, with $t = t_k$, $s = t_{k+1}$ and $\Delta_t \neq 0$, construct

$$\widehat{w}_{s|t}(w, c) = \frac{a(s)\dot{b}(t) - b(s)\dot{a}(t)}{\Delta_t}\, w + \frac{a(t)b(s) - b(t)a(s)}{\Delta_t}\, \widehat{v}_\theta(w, c, t).$$

- Starting from a noisy initial state, repeatedly apply $w \leftarrow \widehat{w}_{s|t}(w, c)$ to obtain a generated latent Z.

Reverse diffusion and flow matching - Answer

Question: What's the difference between reverse diffusion and flow matching?

Answer:

Reverse diffusion and flow matching - Answer

Question: What's the difference between reverse diffusion and flow matching?

Answer:

- Reverse diffusion constructs a NN estimating the conditional expectation of the noise ϵ.
- Flow matching constructs a NN estimating the conditional expectation of the path velocity V_t.

Both can construct the estimate of the expectation of any linear combination of ϵ and Z.

Outline

1. Introduction

2. Mathematical notation

3. What can we do? Why cannot we directly predict output?

4. Why iterative? How iterative?

5. What's the difference between reverse diffusion and flow matching?

6. Summary

Outline

1. Introduction

2. Mathematical notation

3. What can we do? Why cannot we directly predict output?

4. Why iterative? How iterative?

5. What's the difference between reverse diffusion and flow matching?

6. Summary

Summary

Summary

- The pushforward by a map constructed by the one-step error minimization does not work since it degenerates.

Summary

- The pushforward by a map constructed by the one-step error minimization does not work since it degenerates.
- By constructing a “chain” of RVs along a smooth path, we can have a chain of localized conditional distributions so that the chain of pushforwards by the conditional expectation approximates them well.

Summary

- The pushforward by a map constructed by the one-step error minimization does not work since it degenerates.
- By constructing a “chain” of RVs along a smooth path, we can have a chain of localized conditional distributions so that the chain of pushforwards by the conditional expectation approximates them well.
- Reverse diffusion constructs a NN estimating the conditional expectation of the noise ϵ, while flow matching constructs a NN estimating the conditional expectation of the path velocity V_t.

References i

[1] Martin Arjovsky, Soumith Chintala, and Leon Bottou.
Wasserstein generative adversarial networks.
In International Conference on Machine Learning, 2017.

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio.
Generative adversarial nets.
In Advances in Neural Information Processing Systems, 2014.