Diffusion

- decomposing the mapping into a sequence of small, incremental transformations
  - from a simple distribution (e.g., Gaussian noise)
  - to a complex data distribution
- learn to reverse a gradual noising process
  - instead of directly learning a single complex mapping from noise to data

Denoising Diffusion Probabilistic Models (DDPM)

- probabilistic diffusion model that defines a Markovian forward noising process
  - data distribution → gradually corrupted by Gaussian noise → normal Gaussian
  - close-form transition in the noising (forward) process
- learns to reverse the noising process step by step using a neural network
  - gradually denoising Gaussian noise estimated by the network (denoiser)

linear Gaussian Markov chain

$q(x_t|x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$ Gaussian

$q(x_{t-1}|x_t, x_0) \approx p_\theta(x_{t-1}|x_t)$ infinitesimal change → Gaussian noise

Gaussian (tractable) ← approximate

Denoising Diffusion Implicit Models (DDIM)

- assume a non-Markovian forward process
- defines a deterministic mapping from x_t to x_{t-1}
  - prediction → decomposed into clean data x_0 + noise vector to $t-1$ (+ stochastic term)

- skipping steps during sampling → faster inference with fewer steps (accelerating)

Classifier-Free Guidance (CFG)

$\epsilon = \epsilon_{\text{uncond}} + s \cdot (\epsilon_{\text{cond}} - \epsilon_{\text{uncond}})$

- conditioning technique for diffusion models (conditional generation)
  - not require an external classifier
- by Bayesian theorem
  - $p(x_t \mid y) \propto p(y \mid x_t)\, p(x_t)$
  - $\nabla_{x_t} \log p(x_t \mid y) = \nabla_{x_t} \log p(x_t) + \nabla_{x_t} \log p(y \mid x_t)$
  - interpreted as approximating the Bayesian posterior ($\nabla_{x_t} \log p(x_t \mid y)$)
    - by amplifying the likelihood term ($\nabla_{x_t} \log p(y \mid x_t)$) without an explicit
- how it works
  - train the model with and without conditioning
  - during sampling → combine conditional and unconditional predictions
  - guidance scale s ↑ → stronger conditioning
    - too large scale ↑ → diversity ↓
- pusing samples toward regions that better satisfy the condition
  - interpolating between unconditional and conditional noise predictions

Score Function

- $\nabla_x \log p(x)$: gradient of log density
  - how to move a noisy sample toward higher-density regions of the data distribution.

Tweedie's formula

$$\mathbb{E}[x_0 \mid x] = x + \sigma^2 \nabla_x \log p(x)$$

- posterior mean estimation using variance-scaled score function
  - denoising ≃ estimating the posterior mean using the score of the noisy distribution
  - in the diffusion models → how far I should move in that direction to recover the clean signal

---

Flow Matching

- reframes generative modeling as learning a continuous transport
  - from a simple distribution to the data distribution push forward
  - velocity regression without explicitly optimizing likelihoods or solving expensive ODEs during training
- variants dependent on how to define a conditional flow mapping (transformation)
  - diffusion models: a special case of flow matching
    - where the target velocity corresponds to the score-induced reverse-time dynamics

Normalizing Flow

composition of invertible transforms

transformation 꼬이 invertible 해야 함

Jacobian의 determinant 계산이 어려움 $O(n^3)$

- learns a ~~single~~ invertible mapping (discrete)
  - from a simple distribution (e.g., Gaussian) to the data distribution
- uses the change-of-variables formula: 삼각 행렬 elementwise 곱 구함 (∵ complexity ↓)
  - $\log p_1(x_1) = \log p_0(\Phi^{-1}(x_1)) + \log \left|\det(J)\right|$
  - the transformation Φ must be invertible
  - the Jacobian J determinant must be tractable
- limitation: computing Jacobians is expensive

locally linear approximation

determinant → 변환의 scaling

$J = \frac{\partial \Phi^{-1}(x_1)}{\partial x_1}$

확률 밀도 보존

$p_X(x)\,dx = p_Z(z)\,dz$

$p_X(x) = p_Z(z)\left|\det\left(\frac{dz}{dx}\right)\right|$

부피 변화율에 의해 밀도 재조정

질량 보존

coupling layers: 여러 층의 transformation 쌓아 적용

NLL 최소화 (log likelihood 최대화)

Continuous Normalizing Flow

ODE + log density

- extends normalizing flows to continuous time
  - as the integration of infinitesimal changes (instantaneous)
- models the transformation as an ODE
  - $dx/dt = u_t(x)$
  - density evolution follows the continuity equation:
    - $\partial p_t(x)/\partial t = -\nabla \cdot (p_t(x)u_t(x))$
  - log-density change:
    - $d \log p(x_t)/dt = -\nabla \cdot u_t(x_t) = -\mathrm{Tr}\left(\frac{\partial u_t}{\partial x}\right)$
- advantages
  - more flexible transformations
  - avoids explicit Jacobian determinants
- drawbacks
  - training requires solving ODEs repeatedly
    - computationally expensive

model이 학습하는 건 변환 그 자체가 아니라 ODE 학습

timestep t에서 x지점에서 어디로 이동할지

timestep t에서 x 지점에서 확률밀도 p가 어떻게 변할지

trace : 각 축 방향으로의 infinitesimal scaling을 더한 값

divergence를 시간에 대해 적분

integral 요구됨

$O(d^2)$ → 근사하여 확률적 추정

$x_{t+\Delta t} \approx x_t + v_\theta(x_t, t)\Delta t$

log likelihood 최대화로 풀거나

vector regression으로 풀거나

Flow Matching

transport

우리가 path를 어떻게 설정하느냐에 따라 target velocity 구할 수 있음

- matching vector fields instead of densities
  - avoids likelihood-based training altogether
- learns a neural velocity field $v_\theta(x, t)$ to match a target velocity $u_t(x)$
- objective
  - $\mathcal{L} = E[||v_\theta(x_t, t) - u_t(x_t)||^2]$
- advantages
  - no need to compute log-densities
  - no need to solve ODEs during training
- drawbacks
  - the marginal velocity field u_t(x) is generally intractable

GT velocity field

Conditional Flow Matching

- learn a conditional velocity field $u_t(x|x_0, x_1)$
  - instead of learning a marginal velocity field $u_t(x)$
- samples are paired:
  - x_0 ~ base distribution
  - x_1 ~ data distribution
- learns how to transport x_0 to x_1 over time
- objective
  - $\mathcal{L} = E[||v_\theta(x_t, t) - u_t(x_t|x_0, x_1)||^2]$
  - theoretically grounded by the continuity equation
- benefits
  - simpler target vector field
  - stable training
  - avoids density estimation entirely

Rectified Flow

transport path를 어떻게 정의하는지에 따라 velocity가 달라짐

- special case of conditional flow matching
- defines straight-line paths between x_0 and x_1:
  - $x_t = (1-t)x_0 + tx_1$

e.g. diffusion-style

$x_t = \alpha(t)\, x_1 + \sigma(t)\, x_0$

↳ 얼마나 원본 signal을 보존할지

↳ noise를 얼마나 추가할지

$u_t(x_t|x_1) = \frac{dx_t}{dt} = x_1 - x_0$

- corresponding velocity field is simple and deterministic
- advantages
  - extremely simple training
  - fast sampling
  - strong empirical performance
- widely used in practice due to its simplicity

L2 비용 함수 (cost function)의 OT에서는

각 점이 직선으로 이동하는 geodesic에 가까움

↳ 직선 → optimal transport에 가깝도록

coupling에 따라 비용 달라짐

OT-optimal coupling이어야 직선이 OT

RF에서는 OT-optimal coupling 찾는 건 아니지만

더 일관된 transport → 더 OT에 가까운 성질 유도

Diffusion Transformer (DiT)

- latent diffusion — VAE from SD — ∵ pixel DDPM → computationally expensive
- patchify → token sequence — patch size ⇓ → # of tokens ⇑ → computational cost, memory ⇑ / quality ⇑
- positional embedding (sin/cos)

- conditioning — timestep t, condition c
  - in-context — concatenate t, c into input token sequences
  - cross-attention
  - adaLN — layer norm의 scaling · shift ← from condition (regression)
  - adaLN-Zero — residual block — initially zero (gating) — training stability ⇑

GFLops — # of parameters 보다 지표로서 나음 (good proxy)
forward pass — ∵ attention은 token 수에 민감 (이미지 해상도에 따라 요구되는 연산 달라질 수 있음)

SDE-based Score Model

forward process 정의 : data → noise

$$dx = \underbrace{f(x,t)\,dt}_{\text{drift}} + \underbrace{g(t)}_{\text{diffusion coefficient}}\underbrace{dW_t}_{\text{Brownian motion}} \quad (\text{diffusion (stochastic term)}) \qquad dW_t \sim N(0, I)$$

↳ reverse process

reverse-time SDE

$$dx = \underbrace{\left[f(x,t) - g(t)^2 S_t(x)\right]dt}_{\text{drift}} + g(t)\,d\bar{W}_t \qquad S_t(x) = \nabla_x \log p_t(x)$$

score function

probability flow ODE

$$\frac{dx}{dt} = F(x,t) = f(x,t) - \frac{1}{2}g(t)^2 S_t(x) \qquad \text{deterministic}$$

" diffusion sampling은 ODE/SDE를 적분하는 문제 "

VP-SDE (Variance-Preserving)

forward 과정에서 x_t의 variance가 폭발하지 않도록

$$SNR(t) = \frac{Var(signal)}{Var(noise)} = \frac{\bar{\alpha}_t}{1-\bar{\alpha}_t}$$

forward SDE : $dx = -\frac{1}{2}\beta(t)x\,dt + \sqrt{\beta(t)}\,dW_t$

drift : $f(x,t) = -\frac{1}{2}\beta(t)x$

시간에 따라 signal 감소

diffusion : $g(x,t) = \sqrt{\beta(t)}$

DDPM의 discrete-time forward → continuous version

$$x_t = \sqrt{\bar{\alpha}_t}\,x_0 + \sqrt{1-\bar{\alpha}_t}\,\varepsilon$$

$$Var(x_t) = \bar{\alpha}_t\,Var(x_0) + (1-\bar{\alpha}_t)\,Var(\varepsilon) = 1$$

t 증가에 따라 signal의 분산 ↓

timestep별 SNR 편차 심함 → 학습 신호 불균형

낮은 SNR 구간에서 예측 어려움

reverse 시 PF-ODE : $\frac{dx}{dt} = -\frac{1}{2}\beta(t)x - \frac{1}{2}\beta(t)S_\theta(x,t)$

S_θ: noise로부터 유도 가능

$$S_\theta(x,t) \propto -\frac{1}{\sigma(t)}\varepsilon_\theta(x,t)$$

VE-SDE (Variance-Exploding)

forward 과정에서 x_t의 variance ↑

$$SNR(t) \propto \frac{1}{\sigma(t)^2}$$

forward : $dx = g(t)\,dW_t$

drift : $f(x,t) = 0$ diffusion : $g(x,t) = \sqrt{\frac{d}{dt}\sigma(t)^2}$

NCSN

$x_t = x_0 + \sigma(t)\,\varepsilon$

perturbed data

여러 각 σ_i에서 score 학습

annealed Langevin Dynamics

X – Prediction

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon \qquad \varepsilon \sim N(0, I)$$

* ε-prediction — score matching과 정합 (noise ≒ score function의 scaled 역방향)

  noise $\varepsilon_\theta(x_t, t)$ 자체를 예측 — timestep 따라 loss scale 달라질 수 있음 (∵ SNR)

* x_0-prediction

  clean data $\hat{x}_0$ 자체를 예측 — 최종 objective와 정합

  작은 SNR 구간에서는 x_0 추정 어려움 → 불안정

* v-prediction — ε과 x_0의 선형 결합 — 두 신호 섞어서 SNR 균형 맞춤

$$v = \sqrt{\bar{\alpha}_t}\, \varepsilon - \sqrt{1-\bar{\alpha}_t}\, x_0$$

Noise Scheduler

forward process의 각 t에서 α_t, $\bar{\alpha}_t$ 정함 → noise 양 결정

(α_t, $\bar{\alpha}_t$: coefficient)

$$SNR = \frac{\bar{\alpha}_t}{1-\bar{\alpha}_t}$$

$\bar{\alpha}_t \downarrow \rightarrow$ noise $\Uparrow \rightarrow$ SNR $\Downarrow \rightarrow$ ε 예측 어려움

loss를 단순 MSE로 두면 unstable

cosine schedule — SNR이 너무 빨리 붕괴하는 구간 막음

Sampler와 함께 고려되어야 함 (∵ 어떤 구간에서는 abruptly · rapidly 변화 가능)

오차가 민감한 구간에 step 더 배치 (SNR을 고려해서)

Sampler — Update 규칙

DDPM forward process : Markovian chain + linear Gaussian transitions + $q(x_t|x_0)$ closed form

reverse process : stochastic reverse → noise 추가

ancestral sampling $p(x_{t-1}|x_t) = N(\mu_\theta(x_t, t), \sigma_t^2 I)$ (각 step에서 분포에서 적정 뽑는 형태) (Markovian chain)

이산적으로 sampling 하는 게 핵심

DDIM forward process : forward process 자체는 marginal에 관해 DDPM과 동일 (실무에서는 transition 자체도 DDPM처럼) $q(x_t|x_0)$

reverse process : implicit reverse process가 Markovian chain이 아님 (x_0로 가는 게 고려되므로 joint distribution)

implicit trajectory (mapping) x_0 예측 + ϵ_t로 가는 방향 예측 with scaling + stochastic term

stochasticity 조절 가능

probability flow ODE로 표현 가능 (ODE sampling) 0 이면 deterministic

deterministic 경로 solver step ↓ → accelerate ↑ but error ↑

DDPM과 같은 marginals $q(x_t|x_0) = N(\sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$ 을 유지하는 diffusion family 만듦

이를 만들기 위해 posterior $q_\sigma(x_{t-1}|x_t, x_0)$ 분포를 정의 (여러 개가 있지만 그중 하나)

DDPM의 forward처럼 쓰면 결국 q_σ는 유도되는 거 아닌가? → 핵심은 marginal $q(x_t|x_0)$을 맞추는 건이지

DDPM처럼 $q(x_t|x_{t-1})$을 Markov transition으로 고정하는 게 X

(실무에서 forward 자체는 DDPM처럼 but reverse process에서는

ancestral Markov chain 라는 다른 implicit trajectory 사용)

요약) DDIM은 DDPM과 동일한 forward marginals $q(x_t|x_0)$을 유지하는 diffusion family 고려

그 family 안에서 경로의 stochasticity를 고려하는 파라미터 σ_t를 두고, 그에 맞는 $q_\sigma(x_{t-1}|x_t, x_0)$을 정의

정의를 전개하면 sampling update가 deterministic term + stochastic term

Solver

ODE/SDE를 step-by-step으로 근사해 푸는 수치 해석

Sampler의 dynamics를 기반으로 얼마나 적분할지 (주로 DDIM/ODE/SDE 기반의 가속 sampling 시 사용)

deterministic → probability flow ODE로 고려 가능

model의 예측값으로부터 drift 구함 → sampler 규칙에 수치적으로 근사

1st Order Solver (Euler 계열)

한 step에서 현재 시점의 기울기 (derivative)만 보고 직선으로 한 번 이동 (denoising process)

≃ drift

ODE $\frac{dx}{dt} = F(x,t)$ 일때 Euler $x_{k+1} = x_k + h\,F(x_k, t_k)$

denoised, step size 주로 timestep 차이

e.g. DDIM PF-ODE 관점 reverse process 시

$$\frac{dx}{dt} = F(x,t) = -\frac{1}{2}\beta(t)\,x - \frac{1}{2}\beta(t)\,s_\theta(x,t)$$

$\epsilon_\theta(x,t)$ 예측 → score $s_\theta(x,t)$ 구함 → drift $F(x,t)$ 계산 → 다음 state update

SDE: Euler – Maruyama

$$dx = f(x,t)\,dt + g(t)\,dW_t$$

$$x_{k+1} = x_k + h\,f(x_k, t_k) + g(t_k)\sqrt{h}\,z \qquad z \sim N(0, I)$$

Higher-Order Solver

한 step에서 한 기울기만 보는 게 아니라 중간 or 다음 시점도 한 번 더 보조적 보정

e.g. $x_{k+1} = x_k + \frac{h}{2}\left(F(x_k, t_k) + F(\tilde{x}_k, t_{k+1})\right)$

step 수 v.s. NFE (Number of Function Evaluations)

n차 solver → 호출 더 늘어남

PLMS (Pseudo Linear Multistep)

DDIM update 기반 이전 step들의 ϵ 등 예측 저장 → 다음 update 정확히

DPM-Solver

diffusion의 ODE를 단순 Euler로 풀지 말고, 해당 ODE 구조에 맞춘 고차 근사식 사용

10~20 step → well

CFG scale ⇑ → drift ⇑ → step 배치 촘촘하게 or 안정적인 solver

ADM (Diffusion Models Beat GANs)

Classifier Guidance classifier의 $\nabla_x \log p_\phi(y|x)$ 를 이용

reverse variance 학습

$p_\theta(x_{t-1}|x_t) = (\mu_\theta(x_t, t), \Sigma_\theta^2(x,t)I)$ 범위 clip하여 제한해서 학습 $[\log \tilde{\beta}_t, \log \beta_t]$

→ 더 적은 step으로도 품질 유지 가능

Diffusion Likelihood

많은 step의 작은 transition의 곱 → exact likelihood 계산 어려움

- discrete ELBO
- Continuous score-SDE/ probability flow ODE CNF 처럼 likelihood 계산

Efficient DiT

Flash Attention

attention　query key value → query key similarity (softmax) → weighted sum (value)

log-sum-exp　sequence 전체 기준으로 query key sim의 최댓값을 모두 빼 줌 → overflow 방지

GPU　SRAM ↕ HBM　이동 너무 많음　이동 시간 ↑ → 왕복 회수

query key value를 block 단위로 쪼갬

SRAM에 올려 뒀을 때 similarity → softmax → weighted sum 한번에

log sum exp를 위해 online으로 최댓값 · 합 update

$$m_{:j}^{new} = \max(m_{:i}, m_{:j})$$

$$\ell = e^{x_0 - m:2} \times \frac{e^{-m:4}}{e^{-m:2}} + e^{x_1 - m:2} \times \frac{e^{-m:4}}{e^{-m:2}} + e^{x_2 - m:4} + e^{x_3 - m:4}$$

(새로 update) (새로 추가)

KV Cache

W_Q → sequence 내에서 의존성 없이 병렬적임

query는 이전 sample 의존성 X

W_K

W_V　key와 value의 수는 같음

$$z_i = \frac{\exp\langle q_i, k_i \rangle}{\sum_j \exp\langle q_i, k_j \rangle} v_i$$

L^2 개의 가중치 → query key 내적　$O(L^2 d)$　query, key, value 가중치 계산 시

Caching　key, value sequence 저장해 둠 (한 번은 전체 계산 필요함)

다음 token부터는 전체 sequence가 아닌 마지막 sample 하나만 예측

맨 마지막 token으로만 다음 token 예측　$O(LD)$

key value는 저장해둔 것으로 전체 sequence 필요

prefill　prefill_chunk

LLM에서 system prompt는 이미 알고 있는 것 → 미리 계산하여 KV cache에 채워 둠　prefill

$O(L_{prompt}^2)$　$O(L_{user}^2)$ → but 입력 길이는 가변적

↳ 미리 계산할 수 X

chunk로 나눠서 그 안에서 미리 KV cache 계산

prefill_chunk　chunk 단위식 KV caching → 계산 상한량