Tensor-Train Diffusion Models (Work-In-Progress)

M. Baddar , M. Eigel

May 8, 2024

1 Abstract

Denoising Diffusion Probabilistic Models(DDPMs) are one of the recently developed class of Generative Models. Yet to our knowledge, there is no comparative analysis for different classes of the applied function approximator, along with their optimizers, which can justify the current model setups. Furthermore, all experimentation work has been exclusively focusing on image-generation, neglecting high-dimensional data from other domains.

In this article, we will study the application of different Functional-Tensor-Trains(FTT) based approximators with Alternating Least Minimization (ALS) and Riemannian Optimization (RO) as the optimization algorithms. The study will compare these approximator to Neural Networks(NN) approximators optimized by Stochastic Gradient Descent(SGD).

The first goal is to comprehensively analyze DDPM's convergence and quality's sensitivity to architecture and optimization parameters over different toy and real world high-dimensional datasets from different domains. As a result, **the second goal** is to introduce a FTT-DDPM model which is more efficient than NN-DDPM counterparts.

2 Notion and Preliminaries

We present the terminology and notations to be used in this article.

2.1 Generative Model Notations

The vector $\mathbf{x}_0 \in \mathbb{R}^D$ denotes a single sample out of the training samples $X_0 \in \mathbb{R}^{N \times D}$, where D is the data-dimension, N is the number of training samples.

Statistically, the vector $\mathbf{x}_0$ is a random variable whose actual density $p(\mathbf{x}_0)$ is the target density we want to estimate. The generative modeling task is about search for the optimal parametric density $q_\theta \subset \mathcal{Q}$ such that $q_\theta(\mathbf{x}_0) \approx p(\mathbf{x}_0)$

As we focus on discrete non-stationary DDPM models,the term t denotes a discrete finite non-negative time point in the diffusion process such that $t \in \mathbb{Z}^*$ and $0 \leq t \leq T$, where T is the number of time points in the forward and reverse diffusion process.

2.2 Tensor Notations and Terminology

Tensor *order* is the number of dimensions in that tensor. A scaler (order-0 tensor) is denoted by a lowercase letter, e.g. a,while a vector (order-1 tensors) is denoted by a lowercase boldface letter, e.g. $\mathbf{a}$. An uppercase letter, denotes a matrix (order-2 tensors) e.g. A. A higher order tensor ($order - m$ tensor with $m \geq 3$) is denoted by uppercase boldface letters $\mathbf{A}$.

3 Background

3.1 Denoising Diffusion Probabilisitce Models (DDPM)

Score-based Generative models are one of the state-of-art models in generative models literature [SE20, SSDK$^+$21]. Denoising Diffusion Probabilistic Models (DDPM) can be thought of as the simple discrete counterpart of score-based models.

Given a random variable, $\mathbf{x}(0) \in \mathbb{R}^D \sim p(\mathbf{x}_0)$ the model's task is to find a set of parameters $\boldsymbol{\theta}$ for the approximate density function $q(\mathbf{x}(0); \boldsymbol{\theta}) \subset \mathcal{Q}$ such that $q(\mathbf{x}(0); \boldsymbol{\theta}) \approx p(\mathbf{x}(0))$. More formally, the objective is to maximize the log-likelihood $\log(q(\mathbf{x}(0); \boldsymbol{\theta}))$ where $\boldsymbol{x} \sim p(\mathbf{x}(0))$.

3.2 Forward and Reverse Process

In DDPM, the training happens in two phases:

Forward Process In this phase, a set of intermediate latent variables are generated $\mathbf{x}(t), t = 1, 2, .., T$, where $\mathbf{x}(0) \sim p(\mathbf{x}(0))$ is the input data sample, and $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad \mathbf{0} \in \mathbb{R}^D, \mathbf{I} \in \mathbb{R}^{D \times D}$ is noisy seed data.

Let:

$$
\begin{aligned}
&\beta_t, \quad 0 \leq \beta_t \leq 1, \beta \in \mathbb{R} \\
&\alpha_t = 1 - \beta_t \\
&\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i
\end{aligned} \tag{1}
$$

The latent variable $\mathbf{x}(t)$ has the following distribution:

$$
\begin{aligned}
q\left(\mathbf{x}(t) \mid \mathbf{x}(t-1)\right) &= \mathcal{N}\left(\mathbf{x}(t); \sqrt{1-\beta_t}\mathbf{x}(t-1), \beta_t \mathbf{I}\right) \\
q\left(\mathbf{x}(1:T) \mid \mathbf{x}(0)\right) &= \prod_{t=1}^{T} q\left(\mathbf{x}(t) \mid \mathbf{x}(t-1)\right)
\end{aligned} \tag{2}
$$

Which can be rewritten as

$$
q\left(\mathbf{x}(t) \mid \mathbf{x}(0)\right) = \mathcal{N}\left(\mathbf{x}(t); \sqrt{\bar{\alpha}_t}\mathbf{x}(0), \left(1-\bar{\alpha}_t\right)\mathbf{I}\right) \tag{3}
$$

TBD : Add details for deriving the forward sampling equation

Accordingly, in the forward phase for each time $t = 1, 2, \ldots T$, a corresponding $\mathbf{x}(t)$ is generated by sampling from density in eq.(3). In another word, $\mathbf{x}(t)$ can be sampled using the following equation (based on the Re-parameterization trick[KW22, The])

$$
\mathbf{x}(t) = \sqrt{\bar{\alpha}_t}\mathbf{x}(0) + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \tag{4}
$$

Where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

Reverse Process This the process in which we try to train a model the reconstructs $\hat{\mathbf{x}}_0 \sim q_{\boldsymbol{\theta}}$ f rom normal noise $\mathbf{x}_T$, where $q_{\boldsymbol{\theta}}(\mathbf{x}(0)) \approx p(\mathbf{x}(0))$

Add loss function derivation

The loss function for DDPM, given a sample $\mathbf{x} \sim q(\mathbf{x}_0)$ can be written as:

$$
L_{\text{simple}}\left(\theta\right) = \mathbb{E}_{t, \mathbf{x}(t) \mid \mathbf{x}_0, \boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\mathbf{x}(t), t\right)\right\|^2\right] \tag{5}
$$

Where $\mathbf{x}(t)$ is generated as in (4),$t = 1, 2, \ldots, T$.(See eq (14) in[HJA20]) and the re-parameterization section in this article[Wha].

How the loss is calculated for each batch of the optimization loop (assuming an iterative optimization algorithm is applied):

1. The parameter values $\boldsymbol{\theta}$ are given, either as the initial or the updated values during optimization.

2. Sample t uniformly from $1, 2, \dots T$
3. Sample $\mathbf{x}(t)$ based on eq. 4
4. Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5. Calculate $\boldsymbol{\epsilon}_\theta(\mathbf{x}(t), t)$
6. Calculate the Loss as in eq.(5)

3.3 Modeling $\epsilon_\theta(\mathbf{x}(t), t)$

The core modeling task for Diffusion Models is to, a first design an *ansatz* for $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$, then develop an optimization algorithm to find the optimal $\boldsymbol{\theta}$ That minimizes the loss in eq.(5).

In this subsection, we will show the different Neural Network architecture used for this ansatz realization. Also, we will go through the Positional Time Encoding used to embed time into such a model,

3.3.1 Architecture modeling $\epsilon_\theta(\mathbf{x}(t), t)$

In this section we present different Neural Network architectures applied in the literature for modeling $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ [Dif, HJA20, Wha].

3.3.2 Modeling $\epsilon_\theta(\mathbf{x}(t), t)$ with ResNets

One of the simple approaches for modeling $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}$ is using Res-Net like architecture. The code presentedhere[Jmk], has successfully applied ResNet-like architecture for the density estimation of 2D swissroll dataset[sklb]. The equation set describing the applied ResNet model of depth L can be written as follows (in order from left to right in the model implementation)

$$\begin{aligned}
\mathbf{u}(t) &= \mathbf{x}(t) + \mathbf{g}(t) \\
\mathbf{z}(0) &= \mathbf{A}_1\mathbf{u}(0) \quad \textit{Dimension expansion} \\
\mathbf{a}(l) &= \mathbf{B_2}(\boldsymbol{\sigma}(\mathbf{B}_1(\mathbf{z}(l-1)) \quad l = 1, 2, \dots, L \quad \textit{Non-Linearity} \\
\mathbf{z}(l) &= \frac{\mathbf{a}(l) - E(\mathbf{a}(l))}{\sqrt{Var(\mathbf{a}(l)) + \epsilon}} * \gamma + \beta \quad \textit{Normalization Layer} \\
\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}(t), t) &= \mathbf{A}_2\mathbf{z}(L) \quad \textit{Dimension contraction}
\end{aligned} \tag{6}$$

Where $\mathbf{g}(t) : \mathbb{Z}^* \to \mathbb{R}^D$ is the time Positional Encoding function. A detailed explanation for this encoding approach is presented in subsection (3.3.4).

A more sophisticated model for approximating the parametric noise $\boldsymbol{\epsilon_\theta}(\mathbf{x}(t), t)$ is U-Net, which is a Neural Network architecture composed of several ResNet blocks in U shape, and designed especially for image segmentation problems[RFB15]. UNet also applies Positional Encoding for time conditioning [Wha] This architecture has been used in DDPM [HJA20] and the more generic counterpart of DDPMs: the score-based generative models [SE20, SSDK$^+$21, Son21] In the next subsection, we provide more details about this architecture.

3.3.3 Modeling $\epsilon_\theta(\mathbf{x}(t), t)$ with U-Net

TBD: [HJA20, SDWMG15]

3.3.4 Positional Time Embedding

Positional Encoding is a time-embedding technique that has been successfully applied in Transformers architectures[LWLQ21]. Let $t = 0, 1, 2, \ldots, T$ be a time-point in the diffusion process; Positional-Encoding is a function that maps the scalar-values time-point t to a vector-valued time-embedding $\mathbf{g}(t)$ based on the position of t in the sequence $1, 2, \ldots, T$. Formally, let $let : \mathbf{g}(t) = [g(t; j)]_{j=1}^{d} : \mathbb{Z}^* \to \mathbb{R} :$ where d is the dimension of the output embedding

$$\begin{aligned} g(t; 2i) &= \sin\left(\frac{t}{n^{2i/d}}\right) \\ g(t; 2i+1) &= \cos\left(\frac{t}{n^{2i/d}}\right) \end{aligned} \tag{7}$$

Where:

- t is the time steps for the DDPM model, $t = 0, 1, 2, \ldots, T$
- i is the index for the project time-embedding dimension d where $i = 0, 1, 2, .., \frac{d}{2} - 2$. For calculation convenience, d is usually selected to be an even number.
- n User defined scalar, usually set to 10,000 as mentioned in
- $P(t, j)$ is the projection of the time scalar t to the j^{th} projected dimension where $j = 0$

3.4 Functional Tensor-Trains (FTT)

Let $\mathbf{y} = \mathbf{h}(\mathbf{x})$ be a vector-valued function where [Pau] $\mathbf{h} : \mathbb{R}^{D_x} \to \mathbb{R}^{D_y}$. A tensor-train-based ansatz $\mathbf{f} \approx \mathbf{h}$ can be formulated as:

$$(y_1, y_2, \ldots, y_j, y_{D_y}) = \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_j(\mathbf{x}), \ldots, f_{D_y}(\mathbf{x})) \tag{8}$$

And for each $y_j \quad j = 1, 2, \ldots, D_y$, the component function [Pau] $f_j(\mathbf{x}) = f_j(x_1, \ldots, x_i, \ldots, x_{D_x}) : \mathbb{R}^{D_x} \to \mathbb{R}$ can be modeled as a FTT of order D_x, which can be formulated as:

$$f(\mathbf{x}) = \boldsymbol{A}\Phi(\mathbf{x}) \tag{9a}$$

$$\boldsymbol{A}[\boldsymbol{k}] = \sum_{\boldsymbol{s}}^{\boldsymbol{r}} G_1[s_0, k_1, s_1] \ldots G_i[s_{i-1}, k_i, s_i] \ldots G_{D_x}[s_{D_x-1}, k_D, s_{D_x}] \tag{9b}$$

$$\boldsymbol{\Phi}(\mathbf{x}) = \bigotimes_{i=1}^{D_x} \boldsymbol{\phi}_i(x_i) \tag{9c}$$

Where

$\mathbf{A}, \boldsymbol{\Phi} \in \mathbb{R}^{m_1 \times \cdots \times m_i \times \cdots \times m_{D_x}}$

$i = 1, 2, \ldots, D$ *Index over Tensor-Train dimensions*

$\boldsymbol{k} = (k_1, k_2, \ldots, k_i, \ldots, k_{D_x}) \quad k_i = 1, 2, \ldots, m_i$ *Index over Tensor-Train elements*

$\boldsymbol{r} = (r_0, r_1, \ldots, r_i, \ldots, r_{D_x})$ *Tensor train operator ranks*

$r_0 = r_{D_x} = 1$

$\boldsymbol{s} = (s_0, s_1, \ldots, s_i, \ldots, s_{D_x})$ *Index over the ranks*

$G_i \in \mathbb{R}^{r_{i-1} \times m_i \times r_i}$

And the tensor-valued basis function $\Phi(\mathbf{x})$ is modeled by dimensional decomposition as the tensor-product of a set of rank-1 basis functions

$$\Phi(\mathbf{x}) = \bigotimes_{i=1}^{D_x} \boldsymbol{\phi}_i(x_i) \tag{10}$$

Where $\boldsymbol{x} = [x_i]_{i=1}^{D_x}$ and $\boldsymbol{\phi}_i(x_i) : \mathbb{R} \to \mathbb{R}^{m_i}$.

Finally, the contraction algorithm in (9a) is:

1. Calculate $C_i(x_i) = \sum_{k_i=1}^{m_i} G_i(s_{i-1}, k_i, s_i)\boldsymbol{\phi}(x_i)$ for all $i = 1, 2 \ldots, D$. Note that $C_i \in \mathbb{R}^{r_{i-1} \times r_i}$
2. $f(\mathbf{x}) = \sum_{i=1}^{D} \sum_{s_{i-1}, s_i}^{r_{i-1}, r_i} C(x_i)$

4 FTT-DDPM Model

In this section, we present the architecture and the optimization algorithm for the proposed FTT ansatz.

4.1 Architecture

Given a sample from the training data $\mathbf{x}(0) \in p(\mathbf{x}(0))$,the main DDPM loss function under study is:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{t,\mathbf{x}(t)|\mathbf{x}_0,\boldsymbol{\epsilon}} \left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}} (\mathbf{x}(t), t)\|^2 \right] \tag{11}$$

Where

$$\begin{aligned} \mathbf{x}(t) &\in \mathbb{R}^D \quad \forall t = 0, 1, 2, \ldots, T \\ \boldsymbol{\epsilon}_{\boldsymbol{\theta}} (\mathbf{x}_t, t) &: \mathbb{R}^D \times \mathbb{R} \to \mathbb{R}^D \\ \boldsymbol{\epsilon} &\sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \mathbf{0} \in \mathbb{R}^D, \mathbf{I} \in \mathbb{R}^{D \times D} \\ \mathbf{x}(t) &= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon} \\ t &\sim U(1, T) \end{aligned}$$

See eq.(1,2,4) for more details.

There are many TT-based approaches for density estimation in literature[HOL24, NPO22]. This proposed architecture in (**??**) based on the TTDE model introduced in [NPO22]. However, the core difference is the TTDE directly approximates the density $q_\theta(\mathbf{x}(0))$. On the other hand,our work approximates $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)$ by minimizing the $\mathcal{L}_2$ loss function in eq(5), which is a proxy for maximizing the log-likelihood $\log q_\theta(\mathbf{x}_0)$.

Given the formulation of FTT in (9),then the proposed FTT ansatz for $\boldsymbol{\epsilon}_{\boldsymbol{\theta}} (\mathbf{x}_t, t)$ can be formulated as follows:

Let:

$$\begin{aligned} \mathbf{u}(t) &= \mathbf{x}(t) + \mathbf{g}(t) \\ \boldsymbol{\epsilon}_{\boldsymbol{\theta}} (\mathbf{x}_t, t) &= \mathbf{y}(t) = [y_j(t)]_{j=1}^D \end{aligned} \tag{12}$$

Then each component $y_j(t) \in \mathbb{R}, j = 1, 2, \ldots, D$ can be written as an FTT ansatz, by replacing $\boldsymbol{x}$ with $\boldsymbol{u}(t)$ in (9)

$$y_j(t) = \mathbf{A}\mathbf{\Phi}(\mathbf{u}(t)) \tag{13a}$$
$$\mathbf{A} = G_1 G_2 \dots G_i \dots G_D \tag{13b}$$
$$\mathbf{\Phi}(\boldsymbol{u}(t)) = \bigotimes_{i=1}^{D} \boldsymbol{\phi}_i(\boldsymbol{u}_i(t)) \tag{13c}$$

Where

- $\mathbf{g}(t)$ is the positional-encoding function as in (7)
- $\mathbf{u}(t)$ is the time-embedded image for latent variable $\mathbf{x}(t)$ by adding it to the positional-encoded time $\mathbf{g}(t)$.
- $G_i \in \mathbb{R}^{r_{i-1} \times m_i \times r_i}$ is the i^{th} core of the tensor-train operator $\boldsymbol{A}$

See (9) for the details of FTT-approximations.

Basis Functions

Many classes of basis functions have been applied to FTT-approximations like B-Splines, Fourier Series, [NPO22] and Legendre Polynomials[SGK$^+$24]. Initially, we focus on applying two classes of basis functions: Legendre Polynomials and B-Splines. The rationale behind this selection is that both have been successfully applied in the domain of generative modeling [NPO22, SGK$^+$24].

Legendre Polynomials The Bonnel's recursion formula for Legendre-Polynomials can be written as

$$P_k(x) = \begin{cases} X^k & k = 0, 1 \\ \frac{(2k+1)xP_{k-1}(x) - kP_{k-2}(x)}{(k+1)} & k = 2, 3, \dots \end{cases} \tag{14}$$

Accordingly the basis function $\boldsymbol{\phi}(u_i(t)) : \mathbb{R} \to \mathbb{R}^{m_i}$ can be defined as

$$\begin{aligned} \boldsymbol{\phi}_i(u_i(t)) &= [\phi_i^k(u_i(t))]_{k=0}^{m_i - 1} \\ \phi_i^k(u_i(t)) &= P_k(u_i(t)) \end{aligned} \tag{15}$$

Where $\boldsymbol{u}(t) = [u_i(t)]_{i=1}^{D}$

B-Splines Basis Function A B-spline function has two parameters: order p and number of knots K. The function at $B(x;k,p) \in \mathbb{R} \to \mathbb{R}$ for each knot with index $k = 1, 2, \ldots, K$ can be written as:

$$
\begin{aligned}
B(x;k,0) &= \begin{cases} 1 & \text{if } o_k \leq x < o_{k+1} \\ 0 & \text{otherwise} \end{cases} \\
B(x;k,p) &= \frac{x - o_k}{o_{k+p} - o_k} B(x;k,p-1) + \frac{o_{k+p+1} - x}{o_{k+p+1} - o_{k+1}} B(x;k+1,p-1)
\end{aligned}
\tag{16}
$$

How B-splines are used as basis functions? Each $\boldsymbol{\phi}_i(u_i)$ is modeled as B-splines function with m_i knots which are uniformly distributed over u_i's support. The degree is fixed to $p = 2$[NPO22].

Accordingly, we can write the B-Splines realization of the rank-1 basis $\boldsymbol{\phi}_i(u_i(t)) : \mathbb{R} \to \mathbb{R}^{m_i}$ as follows:

Let

$$
\begin{aligned}
&\mathbf{u} = [u_i]_{i=1}^{D} \\
&u_i \in [u_{i,min}, u_{i,max}] \quad \forall i = 1, 2, \ldots D \text{ (Support for } u_i) \\
&\Delta = \lfloor \frac{u_{i,max} - u_{i,min}}{m_i} \rfloor \text{ (Step for the knots)} \\
&o_k = u_{i,min} + (k-1)\Delta \quad \forall k = 1, 2, \ldots, m_i \text{ (Knots values)}
\end{aligned}
$$

Then the basis function $\boldsymbol{\phi}_i(u_i)$ can be formulated as:

$$
\begin{aligned}
&\boldsymbol{\phi}_i(u_i) = [\phi_i^k(u_i)]_{k=1}^{m_i} \quad \boldsymbol{\phi}_i(u_i) : \mathbb{R} \to \mathbb{R}^{m_i}, \phi_i^k(u_i) : \mathbb{R} \to \mathbb{R} \\
&\phi_i^k(u_i) = B(u_i;k,p)
\end{aligned}
\tag{17}
$$

4.2 Optimization

In this subsection, an overview of different optimization methods for the FTT model in (**??**). Add details for each algorithm, in the corresponding subsection

4.2.1 Alternating Linear Schema

4.2.2 Riemannian Optimization

5 Experiments

In this section, we provide the experimentation details and results for optimizing the model in (**??**) by applying different optimization methods in the subsection 4.2.

5.1 Datasets

We apply the proposed method to a set of toy and real world datasets (as in [NPO22]).

Toy Datasets

- **Multivariate Gaussian Distribution**: Each target sample y can be written as $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $\mathbf{y} \in \mathbb{R}^D$
- **Multivariate Gaussian Mixtures**: [Sri] $\mathrm{p}(\boldsymbol{y}) = \sum_{k=1}^{K} \boldsymbol{\pi}_k N\left(\boldsymbol{y} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right) where$ $0 \leq \boldsymbol{\pi}_k \leq 1$, $\sum_{i=1}^{k} \boldsymbol{\pi}_k = 1$, $y \in \mathbb{R}$
- sklearn moon(2D), swissroll(2D and 3D) and circles(2D) [API]
- sklearn digits dataset [skla]

Real-World Datasets As our work is based on [NPO22], we use the same dataset which is basically the same dataset used in [PPM18]. All these datasets are based on UCI-data repository [Dat]

- **POWER** : Individual Household Electric Power Consumption [HB12] After postprocessing the data, we work with a 6-dimensional reduced version of the dataset
- **GAS** : The data set contains the recordings of 16 chemical sensors exposed to two dynamic gas mixtures at varying concentrations After removing correlated attributes, the reduced version will have 8-dimensions
- **HEPMASS** : One of the famous , non-image datasets used for benchmarking generative models, from the domain of physics [UCI, BLP$^+$20]

5.2 Quality Measures

The main metric we will use is Wasserstein-distance [PZ19]:

$$W_p(\pi, \nu) = \inf_{\substack{\mathbf{x}\sim\pi \\ \mathbf{y}\sim\nu}} (\mathbb{E}\|\mathbf{x} - \mathbf{y}\|^p)^{1/p}, \quad p \geq 1, \tag{18}$$

where the infimum is taken over all pairs of d-dimensional random vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^D$ are marginally distributed as π and ν, respectively. For convenience, the terms $W_p(\pi, \nu)$ and $W_p(\mathbf{x}, \mathbf{y})$ will be used interchangeably. Also, the term *Wasserstein-distance* is called *Optimal-Transport* in literature[Opt].

The calculation of Wasserstein-distance is not feasible for higher dimensional data. Accordingly, we apply the *sinkhorn* algorithm [Cut13] to get a regularized approximation for Wasserstein-Distance. TBD : Add more details about the sinkhorn algorithm.

There are many existing implementations for sinkhorn algorithm in python like *POT*[FCG+21, POT], and *GemLoss* [Geo] libraries.

References

[API] sklearn datasets module. `https://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets`. (Accessed on 05/04/2024).

[BLP+20] Siavash A. Bigdeli, Geng Lin, Tiziano Portenier, L. Andrea Dunbar, and Matthias Zwicker. Learning generative models using denoising density estimators, 2020.

[Cut13] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[Dat] Datasets - uci machine learning repository. `https://archive.ics.uci.edu/datasets`. (Accessed on 05/04/2024).

[Dif] Diffusion/diffusion/ddpm at main · jmkernes/diffusion. `https://github.com/Jmkernes/Diffusion/tree/main/diffusion/ddpm`. (Accessed on 04/27/2024).

[FCG+21] RÃ©mi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, AurÃ©lie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, LÃ©o Gautheron, Nathalie T.H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

[Geo] Geomloss: Geometric loss functions between sampled measures, images and volumes — geomloss. `https://www.kernel-operations.io/geomloss/`. (Accessed on 05/07/2024).

[HB12] Georges Hebrail and Alice Berard. Individual Household Electric Power Consumption. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C58K54.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[HOL24] Fuqun Han, Stanley Osher, and Wuchen Li. Tensor train based sampling algorithms for approximating regularized wasserstein proximal operators, 2024.

[Jmk] Jmkernes/diffusion: Everything related to diffusion models! `https://github.com/Jmkernes/Diffusion/tree/main`. (Accessed on 05/07/2024).

[KW22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[LWLQ21] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers, 2021.

[NPO22] Georgii S. Novikov, Maxim E. Panov, and Ivan V. Oseledets. Tensor-train density estimation, 2022.

[Opt] Optimal transport and wasserstein distance. `https://www.stat.cmu.edu/~larry/=sml/Opt.pdf`. (Accessed on 05/07/2024).

[Pau] Paul. Calculus iii - vector functions. https://tutorial.math.lamar.edu/classes/calciii/VectorFunctions.aspx. (Accessed on 05/02/2024).

[POT] Pot: Python optimal transport — pot python optimal transport 0.9.3 documentation. https://pythonot.github.io/. (Accessed on 05/07/2024).

[PPM18] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation, 2018.

[PZ19] Victor M. Panaretos and Yoav Zemel. Statistical aspects of wasserstein distances. *Annual Review of Statistics and Its Application*, 6(1):405–431, March 2019.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[SDWMG15] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.

[SE20] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution, 2020.

[SGK+24] David Sommer, Robert Gruhlke, Max Kirstein, Martin Eigel, and Claudia Schillings. Generative modelling with tensor train approximations of hamilton–jacobi–bellman equations, 2024.

[skla] sklearn digits module. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html#sklearn.datasets.load_digits. (Accessed on 05/04/2024).

[sklb] sklearn.datasets.make_swiss_roll — scikit-learn 1.4.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_swiss_roll.html. (Accessed on 04/27/2024).

[Son21] Yang Song. Generative modeling by estimating gradients of the data distribution — yang song. https://yang-song.net/blog/2021/score/, 5 2021. (Accessed on 04/27/2024).

[Sri] Sargur Srihari. Mixtures of gaussians. https://cedar.buffalo.edu/~srihari/CSE574/Chap9/Ch9.2-MixturesofGaussians.pdf. (Accessed on 05/04/2024).

[SSDK+21] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations, 2021.

[The] The reparameterization trick. https://gregorygundersen.com/blog/2018/04/29/reparameterization/#kingma2013auto. (Accessed on 04/26/2024).

[UCI] Uci hepmass benchmark (density estimation) — papers with code. https://paperswithcode.com/sota/density-estimation-on-uci-hepmass. (Accessed on 05/04/2024).

[Wha] What are diffusion models? — lil'log. https://lilianweng.github.io/posts/2021-07-11-diffusion-models/. (Accessed on 04/25/2024).