Tensor-Train Diffusion Models

M. Baddar , M. Eigel

April 29, 2024

1 Abstract

Score-based Generative Models are one of the recently developed class of Generative Models. Yet, to our knowledge, there is no much work on analyzing the convergence of these models for different classes of density functions. Furthermore, there is no comparative analysis for different architectures of function approximator, along with their corresponding optimizers, which can justify the current model setups.

In this article, we focus on a simplified discrete version of Score-Based Generative Models, called Denoising Diffusion Probabilistic Models(DDPM).

The core task in DDPMs is finding the optimal functional approximator for $\boldsymbol{\epsilon}_\theta(\mathbf{x}(t), t)$ where $\boldsymbol{\theta}^* = argmin_{\boldsymbol{\theta}} \mathbb{E}_{t,\mathbf{x}(t),\boldsymbol{\epsilon}} \left(\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}(t), t)\|_2^2 \right)$

In literature, different Neural-Network (NN) architecture like ResNet and UNet are applied along with Gradient Descent as the optimization algorithm.

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 NN-Gradient Descent counterparts. **The primary goal** is to comprehensively analyze DDPM's convergence and its sensitivity to architecture and optimization parameters. The analysis will include experimentation with toy and real-world densities.

As an outcome of this analysis, the **secondary goal** is to introduce a FTT-DDPM model which is more efficient than NN-DDPM counterparts.

2 Notion and Preliminaries

For consistency, 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 $q_\theta(\mathbf{x}_0) \approx (\mathbf{x}_0)$

As we focus on discrete non-stationary DDPM models,the term t denotes a discrete finite non-negative time point in the diffusion process $t \in \mathbb{Z}^*$ 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 orders 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)

DDPM models are one of the state-of-the-art generative models[HJA20]. Given a random variable $\mathbf{x}_0 \in \mathbb{R}^D$ with density $q(\mathbf{x}_0)$. The task is to find a set of parameters $\boldsymbol{\theta}$ for the approximate density function $p(\mathbf{x}_0; \boldsymbol{\theta})$ such that the log-likelihood $\log(p(\mathbf{x}_0; \boldsymbol{\theta}))$ is maximized.

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 $\mathbf{x}_t \quad t = 0, 1, 2, .., T$, where $\mathbf{x}_t$is the input data and $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad \mathbf{0} \in \mathbb{R}^D, \mathbf{I} \in \mathbb{R}^{D \times D}$.

Given the following set of constants:

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

The latent variable $\mathbf{x}_t$ has the following distribution:

$$
\begin{aligned}
q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right) &= \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}\right) \\
q\left(\mathbf{x}_{1:T} \mid \mathbf{x}_0\right) &= \prod_{t=1}^{T} q\left(\mathbf{x}_t \mid \mathbf{x}_{t-1}\right)
\end{aligned}
\tag{2}
$$

TBD : Add details for deriving the forward sampling equation: Which can be rewritten as

$$
q\left(\mathbf{x}_t \mid \mathbf{x}_0\right) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, \left(1-\bar{\alpha}_t\right)\mathbf{I}\right) \tag{3}
$$

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 $\mathbf{x}_0^{'} \sim q$ f rom normal noise $\mathbf{x}_T$.

Add loss function derivation

The loss function for DDPM, given a sample $\mathbf{x0} \sim q(\mathbf{x}_0)$ 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 eq. (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 and architecture for a model that approximates $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$, 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 different realizations of Diffusion Models and how they are optimized. 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 presented here, has successfully applied ResNet-like architecture for the density estimation of 2D swissroll dataset [skl]. 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)

$$\begin{aligned}
\mathbf{x}_t^{'} &= \mathbf{x}_t + t^{'} \\
\mathbf{z}_0 &= \mathbf{A}_1 \mathbf{x}^{'} \quad \text{Dimension expansion} \\
\mathbf{u} &= \mathbf{B_2}(\boldsymbol{\sigma}(\mathbf{B}_1(\mathbf{z}_{l-1})) \quad l = 1, 2, \dots, L \quad \text{Non-Linearity} \\
\mathbf{z}_l &= \frac{\mathbf{u} - E(\mathbf{u})}{\sqrt{Var(\mathbf{u}) + \epsilon}} * \gamma + \beta \quad \text{Normalization Layer} \\
\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) &= \mathbf{A}_2 \mathbf{z}_L \quad \text{Dimension contraction}
\end{aligned} \tag{6}$$

Where $t^{'}$ is encoded time based on the Positional Encoding function $f : \mathbb{R} \to \mathbb{R}^D$ (A detailed explanation for this encoding approach is presented in subsection (3.3.4).

The parameters vector can be written as $\boldsymbol{\theta} = [vec(\mathbf{A}_1), vec(\mathbf{A}_2), vec(\mathbf{B}_1), vec(\mathbf{B}_2)), \gamma, \beta]$ where $\mathbf{z}_l \in \mathbb{R}^{D_1}, \mathbf{A}_1 \in \mathbb{R}^{D \times D_1}, \mathbf{A}_2 \in \mathbb{R}^{D_1 \times D}, \mathbf{B}_1 \in \mathbb{R}^{D_1 \times D_2}, \mathbf{B}_2 \in \mathbb{R}^{D_2 \times D_1}$.

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

TBD : [Dem, AGe, VSP$^+$23]

3.4 Functional Tensor-Trains (FTT)

In this section and overview of Functional Tensor Trains is introduced. The details of how tensor-trains can approximate a given function, along with different optimization algorithms, will be illustrated TBD : Add details

3.5 Optimization of Function Tensor Trains

In this subsection, an overview of different optimization methods for Functional Tensor Trains.

Add details for each algorithm, in the corresponding subsection

3.5.1 Alternating Linear Schema

3.5.2 DMRG

[]

3.5.3 Riemannian Optimization

4 Model

4.1 Architecture

Given a sample from the training data $\mathbf{x}_0 \in q(\mathbf{x}_0)$,the main optimization task we will study is:

$$\mathcal{L}_{\text{simple}}\left(\boldsymbol{\theta}\right)) = \mathbb{E}_{t,\mathbf{x}_t|\mathbf{x}_0\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}\left(\mathbf{x}_t, t\right)\right\|^2\right] \tag{7}$$

Where

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

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

Let $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{\prod_{i=1}^{D} m}$ be a tensor-train of order D whose elements can be written as

$$\boldsymbol{\mathcal{A}} = G_1[., i_1, .]G_2[., i_2, .] \dots G_D[., i_D, .] \tag{8}$$

Where each core G_i is a 3-order real matrix: $G_i \in \mathbb{R}^{r_{i-1}\times m_i \times r_i}$ and $r_0 = r_D = 1$.

There are many TT-based approaches for density estimation in literature [HOL24, NPO22]. This architecture is based on the TTDE model introduced in [NPO22]. However, the core difference is the TTDE directly approximates the density $q_\theta(\mathbf{x}_0)$, but on the other hands, our work approximates $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}(t), t)$ by minimizing the $L2$ loss function in eq(5). Minimizing the $L2$ loss is a proxy for maximizing the log-likelihood $\log q_\theta(\mathbf{x}_0)$.

Then the proposed FTT approximation model for $\boldsymbol{\epsilon}_{\boldsymbol{\theta}}\left(\mathbf{x}_t, t\right)$ can be for-

mulated as:

$$\mathbf{u}(t) = \mathbf{x}(t) + \mathbf{t}' \tag{9a}$$

$$y_i(t) = \mathbf{A}\mathbf{\Phi}(\mathbf{u}(t)) \tag{9b}$$

$$= \sum_{i_1=1,i_2=1,\ldots,i_D=1}^{m,\ldots,m} G_1[.,i_1,.]G_2[.,i_2.]\ldots G_D[.,i_D,.]\boldsymbol{\phi}_1(u_1(t))\boldsymbol{\phi}_2(u_2(t))...\boldsymbol{\phi}_D(u_D(t)) \tag{9c}$$

$$\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t) = \mathbf{y}(t) = [y_i(t)]_{i=1}^{D} \tag{9d}$$

Where $\mathbf{u}$ is the time-embedded image for latent variable $\mathbf{x}(t)$ by adding it to the Positional Encoding time. $\mathbf{t}'$.

We apply dimensional-decomposition for both input $\mathbf{u}(t)$ and output $\mathbf{y}(t)$. For input, we apply a tensor-valued basis function $\mathbf{\Phi}(\mathbf{u}(t))$ to introduce non-linearity in the model. The tensor basis $\mathbf{\Phi}(\mathbf{u}(t))$ is decomposed to the tensor product of a set of rank-1 basis function:

$$\mathbf{\Phi}(\mathbf{u}(t)) = \boldsymbol{\phi}_{\mathbf{1}} \otimes (u_1(t)) \otimes \boldsymbol{\phi}_{\mathbf{2}}(u_2(t)) \cdots \otimes \boldsymbol{\phi}_{\boldsymbol{D}}(u_D(t))$$

Where $\mathbf{\Phi}(\mathbf{u}(t)) \in: \mathbb{R}^D \to \mathbb{R}^{m_1 \times m_2 \cdots \times m_D}$ and $\boldsymbol{\phi}_i(u_i) \in \mathbb{R} \to \mathbb{R}^{m_i}$.

4.1.1 Basis Functions

Many classes of basis functions have been applied to FTT-approximations like B-Splines, Fourier Series, [NPO22] Legendre Polynomials[SGK$^+$24]. As the proposed architecture is based on the TTDE model in [NPO22], we focus on B-Splines basis function.

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

$$B(x; k, 0) = \begin{cases} 1 & \text{if } s_k \leq x < s_{k+1} \\ 0 & \text{otherwise} \end{cases}$$

$$B(x; k, p) = \frac{x - s_k}{s_{k+p} - s_k} B(x; k, p-1) + \frac{s_{k+p+1} - x}{s_{k+p+1} - s_{k+1}} B(x; k+1, p-1) \tag{10}$$

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 $\boldsymbol{\phi}_i(u_i(t))$ as follows:
Let

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

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

$$
\begin{aligned}
&\boldsymbol{\phi}_i(u_i) = [\phi_i^k(u_i)]_{k=1}^{m_i} \quad \boldsymbol{\phi}_i(u_i) : \mathbb{R} \to \mathbb{R}^{m_i}, \phi_i^k(u_i) : \mathbb{R} \to \mathbb{R} \\
&\phi_i^k(u_i) = B(u_i; k, p) \\
&B(u_i; k, 0) = \begin{cases} 1 & \text{if } s_k \leq x < s_{k+1} \\ 0 & \text{otherwise} \end{cases} \\
&B(u_i; k, p) = \frac{u_i - s_k}{s_{k+p} - s_k} B(u_i; k, p-1) + \frac{s_{k+p+1} - u_i}{s_{k+p+1} - s_{k+1}} B(u_i; k+1, p-1) \quad \forall k = 1, 2, \dots, m_i
\end{aligned}
\tag{11}
$$

4.2 Optimization

We focus on L2 opt for FTT models. For each a method 1. line or two of text 2. eqn applied to problem 3. opt algorithm

5 Experiments

5.1 Plan

1. Isolation experiments for parametric noise model
2. Use a ResNet, Unet for training such data (for validation)
3. Apply TT opt to isolated data coming from different distributions (the effect of distribution on tt opt)
4. The effect of rank and number of cores,
5. The effect of opt algorithm parameters

6. Drawing a loss landscape for the original DDPM and TT ones
7. Analyzing the convergence of each optimization method
8. Analyzing the number of computations with each method
9. Analyze the memory footprint for each
10. As a plan B , if the TT-DDPM model is not better. This work can result in a comparative analysis to different TT-OPT methods applied to the context of Diffusion Models examples : https://arxiv.org/pdf/2404.16154 , https://arxiv.org/pdf/2404.06455 , https://arxiv.org/pdf/2212.10797

convergence analysis examples https://arxiv.org/abs/2208.05314 https://openreview.net/pdf?id=v https://www.igpm.rwth-aachen.de/Download/reports/pdf/IGPM423.pdf https://www.jstor.org/stable https://youtu.be/4WDedaz$_T V4?si = ksqwnfZMYNF_U595https://www.cis.upenn.edu/cis6100/Ri Wirth - optim - Riemann.pdf$**https://arxiv.org/abs/1712.09913**

5.2 Analysis of Gradient Descent Optimization with Neural Networks Model

5.3 Optimization with Gradient Descent for Tensor-Train Model

5.4 Optimization with Alternating Linear Scheme for Tensor-Train Model

5.5 Optimization with Riemannian Gradient Descent for Tensor-Train Model

Why ? OPTIMIZATION METHODS ON RIEMANNIAN MANIFOLDS AND THEIR APPLICATION TO SHAPE SPACE https://www.uni-muenster.de/AMM/num/wirth/fi
"Even if an embedding is known, one might hope that a Riemannian optimization method performs more efficiently since it exploits the underlying geometric structure of the manifold. For this purpose, various methods have been devised, from simple gradient descent on manifolds [25] to sophisticated trust region methods [5]."

5.6 Results

References

[AGe] A gentle introduction to positional encoding in transformer models, part 1 - machinelearningmas-

tery.com. https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1 (Accessed on 04/26/2024).

[Dem] Demystifying diffusion models. generative models learns a... — by rucha apte — medium. https://ruchaa.medium.com/demystifying-diffusion-models-c7e0689a7ca8. (Accessed on 04/26/2024).

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

[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.

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

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

[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.

[skl] 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).

[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).

[VSP+23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

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

Appendix A - FTT architecture experimentation

Appendix B - FTT optimization experimentation