Faster Training and Improved Performance of Diffusion Models via Parallel Score Matching

Etrit Haxholli [1] **Marco Lorenzi** [1]

Abstract

The modeling of the score evolution by a single time-dependent neural network in Diffusion Probabilistic Models (DPMs) requires long training periods and potentially reduces modeling flexibility and capacity. In order to mitigate such shortcomings, we propose to leverage the independence of the learning tasks at different time points in DPMs. More concretely, we split the learning task by employing independent networks, each of which only learns the evolution of scores in a time sub-interval. Furthermore, motivated by residual flows, we take this approach to the limit by employing separate networks independently modeling the score at each single time point. As demonstrated empirically on synthetic and image datasets, not only does our approach greatly speed up the training process, but it also improves the density estimation performance as compared to the standard training approach for DPMs.

1. Introduction

Forward Diffusion Processes (FDPs) are Markov chains that gradually transform the data distribution into a standard multivariate normal one, by slowly corrupting the data samples. The machine learning task in the framework of Diffusion Probabilistic Models (DPMs) consists of training a neural network in order to model the reverse dynamics of this process (Sohl-Dickstein et al., 2015). To this end, different approaches have been developed, notably the denoising (Vincent, 2011; Ho et al., 2020) and the sliced score matching loss (Hyvärinen, 2005; Song et al., 2019). In both cases, the observed samples are used to train a function approximator in order to model the score, that is, the gradient field of the log-likelihood function from which the data points originate.

The performance of such models improves as the number of steps in the FDP increases. The optimum is naturally reached at the limit when the number of steps tends to infinity. In this case, the FDP can be represented as a Stochastic Differential Equation (SDE). This SDE defines a distinct distribution at every time point of the diffusion process, with the initial and terminal ones being the data and the standard Gaussian distributions respectively. Given samples from the data distribution, for any given time t_i, one can generate samples from the distribution at that time efficiently. Such generated samples can be used to train a neural net to approximate the score at time t_i. The common approach (Ho et al., 2020; Song et al., 2021; Rombach et al., 2021) is to train a single time-varying neural network to model all scores, i.e., to model the evolution of the score (Figure 1).

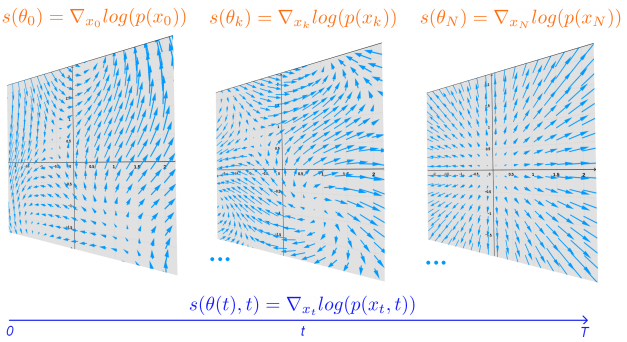

Figure 1. Instead of using a time dependent network to learn the scores for all times (below: blue), at each time point we can use a model per score (above: orange), naturally allowing parallel training of diffusion models.

The Fokker-Planck equation establishes the connection between SDE diffusion models and continuous normalizing flows (Song et al., 2021). Indeed, the estimated scores provide the dynamics of the continuous normalizing flow (CNF) which describes the evolution of the distribution determined by the SDE. Leveraging on this connection, the CNF framework can be applied to generate data, or to estimate the likelihood of unseen points by employing the instantaneous change of variable.

Despite this equivalence in nature and functionality between

[1]Inria, France. Correspondence to: Etrit Haxholli <etrit.haxholli@inria.fr>.

SDE diffusion processes and CNFs, they differ in the manner they are optimized. CNFs are trained by likelihood maximization, which is sub-optimal for many reasons. One of the most important is that since normalizing flows are designed as a chain of transformations, this entire chain must be kept in memory during training, as by construction it is not possible to optimize a step in the chain independently from the rest. While CNFs mitigate the memory bottleneck via the adjoint method, in order to ensure continuity, all vector fields at all time point have to be learned by a single time-dependent neural network, which negatively affects the flexibility of the transformation. Composing many CNFs together is not feasible due to memory and computational constraints, as the serial nature of the composition, as before, implies that each additional CNF introduces a new set of parameters and prolongs training. Each CNF in this context is referred to as a CNF block.

The aforementioned facts highlight the impossibility of independent learning of vector fields in the CNF approach. In order to speed up training and increase the flexibility of the model, in this work we exploit the inherent property of DPMs that scores at different time points can be optimized independently. Therefore, instead of training a single time-varying U-Net to model the scores of the distributions at all time points, we split the training task by dividing the integration time interval of the SDE into smaller sub-intervals with even lengths. For each sub-interval, we deploy a time-dependent U-Net to model the evolution of scores for the distributions defined in that sub-interval. We refer to this approach as time-varying parallel score matching (TPSM). Each of such U-Nets corresponds to a CNF block in the CNF framework, and their composition provides the entire evolution of distribution determined by the FDP.

The task of each U-Net is simplified with increasing number of such sub-intervals, and in the limit, each time sub-interval becomes a point. Thus, in addition to the strategy above, we train a large number of smaller networks that do not depend on time such that each network learns the score at a single time point of the diffusion process (Figure 1). This approach is named discrete parallel score matching (DPSM).

We test our Parallel Score Matching (PSM) approaches on 2D data, as well as image datasets such as CIFAR-10, CelebA 64x64 and ImageNet 64x64. The results demonstrate that our strategy not only improves the log-likelihood results but also speeds up the training process up to 50 times.

2. Background and Related Work

2.1. Normalizing Flows (NFs)

Normalizing Flows: A Normalizing Flow (Tabak & Vanden-Eijnden, 2010; Tabak & Turner, 2013; Rezende & Mohamed, 2015; Dinh et al., 2014) is a transformation defined as a sequence of diffeomorphisms that converts a base probability distribution (e.g., a standard normal) into another distribution by warping the domain on which they are defined. Let $\boldsymbol{Z}$ be a random variable, and define $\boldsymbol{X} = g(\boldsymbol{Z})$, where g is a diffeomorphism with inverse h. If we denote their probability density functions by $f_{\boldsymbol{Z}}$ and $f_{\boldsymbol{X}}$, the change of variable theorem states that:

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = f_{\boldsymbol{Z}}(\boldsymbol{z})|\frac{d\boldsymbol{z}}{d\boldsymbol{x}}| = f_{\boldsymbol{Z}}(h(\boldsymbol{x}))|\frac{dh(\boldsymbol{x})}{d\boldsymbol{x}}|. \quad (1)$$

The objective is the optimization of parameters of h to maximize the likelihood of sampled data points $\boldsymbol{x}$. After training, one can input any test point $\boldsymbol{x}$ on the RHS of Equation 1, and calculate its likelihood. For the generative task, being able to easily recover g from h is essential, as the generated point $\boldsymbol{x}_g$ will take form $\boldsymbol{x}_g$=$g(\boldsymbol{z}_s)$, where $\boldsymbol{z}_s$ is a sampled point from the base distribution $f_{\boldsymbol{Z}}$.
For increased modeling flexibility, we can use a chain (flow) of transformations, $\boldsymbol{z}_i = g_i(\boldsymbol{z}_{i-1}), i \in [n]$. The interested reader can find a more in-depth review of normalizing flows in (Kobyzev et al., 2021) and (Papamakarios et al., 2021).

Neural ODE Flows: Neural ODEs (Chen et al., 2018) are continuous generalizations of residual networks:

$$\boldsymbol{x}_{t_{i+1}} = \boldsymbol{x}_{t_i} + \epsilon f(\boldsymbol{x}_{t_i}, t_i, \boldsymbol{\theta})$$

$$\rightarrow \boldsymbol{x}(t) = \boldsymbol{x}(0) + \int_0^t f(\boldsymbol{x}(\tau), \tau, \boldsymbol{\theta})d\tau, \text{ as } \epsilon \rightarrow 0, \quad (2)$$

where for a network with finite weights, injectivity is guaranteed by the Picard–Lindelöf theorem. (Chen et al., 2018) derive the expression for the instantaneous change of variable, which enables one to train continuous normalizing flows and perform likelihood estimation:

$$\log p(\boldsymbol{x}(0)) = \log p(\boldsymbol{x}(T)) + \int_0^T tr\frac{\partial f(\boldsymbol{x}(t), t, \boldsymbol{\theta})}{\partial \boldsymbol{x}(t)}dt, \quad (3)$$

where $\boldsymbol{x}(0)$ represents a sample from the data. Such models are better known as Continuous Normalizing Flows.

Invertible Residual Flows (IRFs): These models (Behrmann et al., 2019) are similar to the discretized version of Continuous Normalizing Flows:

$$\boldsymbol{x}_{t_{i+1}} = \boldsymbol{x}_{t_i} + f(\boldsymbol{x}_{t_i}, \boldsymbol{\theta_{t_i}})$$

It can be noticed that in this case the output of the residual block is not scaled before being added to the input, and furthermore, each block is comprised of a different set of trainable parameters, instead of allowing the behaviour of the model to evolve via a time input. As injectivity cannot be guaranteed via the Picard–Lindelöf theorem, bijectivity has to be enforced by imposing the contraction condition $Lip(f(\boldsymbol{x}_{t_i}, \boldsymbol{\theta_{t_i}})) < 1$, where $Lip(f(\boldsymbol{x}_{t_i}, \boldsymbol{\theta_{t_i}}))$ is the Lipschitz constant of $f(\boldsymbol{x}_{t_i}, \boldsymbol{\theta_{t_i}})$.

2.2. Diffusion Probabilistic Models (DPMs)

Denoising Diffusion Models: A forward diffusion process or diffusion process (Sohl-Dickstein et al., 2015) is a fixed Markov chain that gradually adds Gaussian noise to the data according to a schedule $\{b_t|t \in [n]\}$:

$$q(\boldsymbol{x_t}|\boldsymbol{x_{t-1}}) := \mathcal{N}(\boldsymbol{x_{t-1}}\sqrt{1-b_t}, b_t\boldsymbol{I}). \tag{4}$$

Such a process transforms the data distribution into a standard multivariate normal distribution. If we denote $a_t = 1 - b_t$ and $\bar{a}_t = \prod_{s=1}^{t} a_s$, then we can write:

$$q(\boldsymbol{x_t}|\boldsymbol{x_0}) = \mathcal{N}(\boldsymbol{x_0}\sqrt{\bar{a}_t}, (1-\bar{a}_t)\boldsymbol{I}). \tag{5}$$

The reverse process conditioned on the initial sample is also described by a chain of Gaussian distributions:

$$q(\boldsymbol{x_{t-1}}|\boldsymbol{x_t}, \boldsymbol{x_0}) = \mathcal{N}(\mu(\boldsymbol{x_t}, \boldsymbol{x_0}), \boldsymbol{\Sigma}(t)), \tag{6}$$

where

$$\mu(\boldsymbol{x_t}, \boldsymbol{x_0}) = \mu(\boldsymbol{x_t}, \boldsymbol{x_0}(\boldsymbol{x_t}, \boldsymbol{\varepsilon})) = \frac{1}{\sqrt{a_t}}\big(\boldsymbol{x_t} - \frac{b_t}{\sqrt{1-\bar{a}_t}}\boldsymbol{\varepsilon}\big),$$

$$\boldsymbol{\Sigma}(t) = b_t\boldsymbol{I}.$$

The goal is to approximate this reverse process via

$$p(\boldsymbol{x_{t-1}}|\boldsymbol{x_t}) := \mathcal{N}(\mu_{\boldsymbol{\theta}}(\boldsymbol{x_t}, t), \boldsymbol{\Sigma}(t)) \tag{7}$$

that converts the standard multivariate normal distribution into the data distribution. To this end, for each step t, the KL divergence between $q(\boldsymbol{x_{t-1}}|\boldsymbol{x_t}, \boldsymbol{x_0})$ and $p(\boldsymbol{x_{t-1}}|\boldsymbol{x_t})$ is minimized, which amounts to minimizing

$$\mathbb{E}_{\boldsymbol{x_0}, \boldsymbol{x_t}}||\mu_{\boldsymbol{\theta}}(\boldsymbol{x_t}, t) - \mu(\boldsymbol{x_0}, \boldsymbol{x_t})||^2, \tag{8}$$

or equivalently

$$\mathbb{E}_{\boldsymbol{x_0}, \boldsymbol{\varepsilon}}||\boldsymbol{\varepsilon}_\theta(\boldsymbol{x_0}\sqrt{\bar{a}_t} + \sqrt{(1-\bar{a}_t)}\boldsymbol{\varepsilon}, t) - \boldsymbol{\varepsilon}||^2, \tag{9}$$

for $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, data samples $\boldsymbol{x_0}$, and a neural network $\boldsymbol{\varepsilon}_\theta(\boldsymbol{x_t}, t)$ (Ho et al., 2020).

2.3. SDE Diffusion Models and their CNF Representation

For an $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, the forward diffusion process defined in Equation 4 can be written as

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1}\sqrt{1-b_t} + \sqrt{b_t}\boldsymbol{\varepsilon}. \tag{10}$$

As derived in (Song et al., 2021), the continuous counterpart of this process takes the form

$$d\boldsymbol{x}(t) = -\frac{1}{2}b(t)\boldsymbol{x}(t)dt + \sqrt{b(t)}d\boldsymbol{w}. \tag{11}$$

In this case Equation 5 becomes

$$q(\boldsymbol{x_t}|\boldsymbol{x_0}) = \mathcal{N}(\boldsymbol{x_0}\mu_t, \sigma_t^2\boldsymbol{I}), \tag{12}$$

where $\mu_t = e^{-\frac{1}{2}\int_0^t b(s)ds}$ and $\sigma_t = \sqrt{(1 - e^{-\int_0^t b(s)ds})}$.

As shown through the Fokker-Plack equation, the evolution of the probability density function of the data, as dictated by the FDP, is identical to the evolution dictated by the following ODE transformation:

$$d\boldsymbol{x}(t) = -\frac{1}{2}b(t)[\boldsymbol{x}(t) + \nabla_{\boldsymbol{x}(t)}\log p_t(\boldsymbol{x}(t))]dt = f_t(\boldsymbol{x}(t))dt. \tag{13}$$

Knowing f_t allows us to perform data generation and likelihood estimation, through the framework of continuous normalizing flows. It can be observed that the only unknown in f_t, is the score $\nabla_{\boldsymbol{x}(t)}\log p_t(\boldsymbol{x}(t))$. This quantity can be modelled using a neural network $\boldsymbol{s}_\theta(x(t), t)$ trained either through sliced score matching (Song et al., 2019):

$$min\ \mathbb{E}_{x_t \sim p(x_t)}[\frac{1}{2}||\boldsymbol{s}_\theta(x_t, t)||^2 + div(\boldsymbol{s}_\theta(x_t, t))] \tag{14}$$

or the equivalent MSE denoising loss (Ho et al., 2020; Kingma et al., 2021):

$$min\ \mathbb{E}_{\boldsymbol{x_0}, \boldsymbol{\varepsilon}}||\boldsymbol{s}_\theta(\boldsymbol{x_0}\mu_t + \sigma_t\boldsymbol{\varepsilon}, t) - (-\frac{\boldsymbol{\varepsilon}}{\sigma_t})||^2. \tag{15}$$

3. Framework and Relation to Piece-wise Continuous Flows and IRFs

In the case of continuous normalizing flows, since we integrate forward, expression $\boldsymbol{x}(t) = \boldsymbol{x}(0) + \int_0^T f_t dt$ is a function of all vector fields $f_t = f(\boldsymbol{x}(t), \boldsymbol{\theta}, t)$ which are parameterized by the trainable parameters of the network, as well as the time input. Increasing model capacity by using a separate neural network to learn a vector field at each time t is not feasible, as the entire function $\boldsymbol{x}(t_N) = \boldsymbol{x}(\boldsymbol{x}_0, f_{t_1}(\boldsymbol{\theta}_{t_1}), f_{t_2}(\boldsymbol{\theta}_{t_2}), ..., f_{t_N}(\boldsymbol{\theta}_{t_T}))$ would have to be kept on memory, and the transformation would not necessarily be continuous. The memory constraint also appears when multiple CNF blocks are used in a piece-wise continuous flow, as each block $\int_{t_{i-1}}^{t_i} f(\boldsymbol{x}(\tau), \boldsymbol{\theta}_i, \tau)d\tau$ introduces a new set of parameters $\boldsymbol{\theta}_i$. Since these transformations are linked in a chain, they must coexist in memory during the maximum likelihood optimization process. In addition, regarding continuous normalizing flows, the optimization task is challenging as the time dependent network $f(\boldsymbol{x}(t), \boldsymbol{\theta}, t)$ has to generate an evolution process of the data distribution that terminates at a standard normal distribution. Furthermore the loss function is highly complex as not only it contains the end point of the integration path $\boldsymbol{x}(T) = \boldsymbol{x}(0) + \int_0^T f(\boldsymbol{x}(t), \boldsymbol{\theta}, t)dt$, but also the integral of the divergence along that integration path $\int_0^T tr\frac{\partial f(\boldsymbol{x}(t), t, \boldsymbol{\theta})}{\partial \boldsymbol{x}(t)}dt$.

Main bottlenecks of Continuous Normalizing Flows

1) The entire transformation procedure must be kept in memory during optimization, and the loss function is computationally demanding.

2) Due to memory/computational constraints, the number of CNF blocks is limited, reducing the adaptability capabilities of the flow in time.

3) The network must devise an entire process which transforms the data distribution into the standard Gaussian.

On the other hand, diffusion models parameterized by time dependent neural networks, reduce the difficulty of learning the evolution process, as the forward diffusion defines the evolution of the data distribution in time. This simplifies the task of the framework to simply model the already defined vector fields. Unfortunately, if parameterized by a single time varying network, such models still suffer from the reduced flexibility of having a single set of parameters (model) carry the burden of modeling all the vector fields defined by the forward diffusion process. Furthermore, the optimization of a vector field at time t is dependent on the optimization of other vector fields at other time points, which slows down the training process significantly. This standard approach (SA) in DPMs is described in Algorithm 1 for comparison, and is abbreviated as SA-DPM.

Disadvantages of SA in DPMs

1) The adaptability capabilities of the flow with respect to time are reduced, due to the use of a single time-varying network.

2) The optimization of score approximation at any time t_i is still dependent on the optimization of score approximation at any time t_j.

As shown in Appendix A, the scores in a diffusion process evolve continuously, with the given PDE dynamics:

$$\frac{\partial s(\boldsymbol{x},t)}{\partial t} = \frac{1}{2}\nabla_{\boldsymbol{x}} tr \frac{\partial s(\boldsymbol{x},t)}{\partial \boldsymbol{x}} + \frac{1}{2}\frac{\partial s(\boldsymbol{x},t)}{\partial \boldsymbol{x}}(\boldsymbol{x} + s(\boldsymbol{x},t)).$$

Since the distribution at each time t_i and its corresponding score are automatically defined by the distribution of the data, we can learn the scores $\nabla_{\boldsymbol{x}_{t_i}} \log p(\boldsymbol{x}_{t_i}, t_i)$ via each $s_{\boldsymbol{\theta}_i}(\boldsymbol{x}_{t_i})$ at all times t_i in parallel to greatly speed up training. If our models approximate such scores well, then when joined together the modelled transformation will be smooth. This motivates splitting the diffusion process, that is, we split interval $t \in [0,1]$ into $\bigcup_{i=0}^{N-1}[t_i, t_{i+1}]$. In this way, we can still train a time-dependent neural network $s_{\boldsymbol{\theta_i}}(\boldsymbol{x}_t, t)$ to learn the scores $\nabla_{\boldsymbol{x}_t} \log p(\boldsymbol{x}_t, t)$ for time $t \in [t_i, t_{i+1}]$. Such a strategy enables us to reduce the difficulty of the task for each model, increasing modeling capacity, while still maintaining time continuity. Furthermore, since the training can be done in parallel, only one model is loaded in memory during training in each device, and after training it can be saved in a disk and reloaded at test time. The training procedure of this framework is described in Algorithm 2. In this case, each score modeling network $s_{\boldsymbol{\theta_i}}(\boldsymbol{x}_t, t)$ corresponds to a CNF block in the continuous normalizing flow representation.

Extending this approach to its limit, the length of each interval goes to 0, hence we use one model to estimate the score at a given time-point. This case corresponds to an invertible

Algorithm 1 Standard Approach in Diffusion Models

Input: data $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$, batch-size p
Train model $\boldsymbol{s_\theta}(t)$
repeat
  sample τ_1, ..., τ_p from $[0,1]$
  sample $\boldsymbol{x}^0_{\pi(1)}$, ..., $\boldsymbol{x}^0_{\pi(p)}$ from $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$
  generate $\boldsymbol{x}^{\tau_1}_{\pi(1)}$, ..., $\boldsymbol{x}^{\tau_p}_{\pi(p)}$ using Equation 12
  minimize $\Sigma_j ||\boldsymbol{\varepsilon_\theta}(\boldsymbol{x}^{\tau_j}_{\pi(j)}(\boldsymbol{\varepsilon_j}), \tau_j) - (\boldsymbol{\varepsilon_j})||^2$.
until convergence

Algorithm 2 Time-varying Parallel Score Matching (TPSM)

Input: data $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$, batch-size p
Split $[0,1]$ into $\bigcup_{i=0}^{N-1}[t_i, t_{i+1}]$
for $i = 0$ **to** $N-1$, in parallel **do**
  Train model $\boldsymbol{s_{\theta_i}}$
  repeat
    sample τ_1, ..., τ_p from $[t_i, t_{i+1}]$
    sample $\boldsymbol{x}^0_{\pi(1)}$, ..., $\boldsymbol{x}^0_{\pi(p)}$ from $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$
    generate $\boldsymbol{x}^{\tau_1}_{\pi(1)}$, ..., $\boldsymbol{x}^{\tau_p}_{\pi(p)}$ using Equation 12
    minimize $\Sigma_j ||\boldsymbol{\varepsilon}_{\boldsymbol{\theta}_i}(\boldsymbol{x}^{\tau_j}_{\pi(j)}(\boldsymbol{\varepsilon_j}), \tau_j) - (\boldsymbol{\varepsilon_j})||^2$.
  until convergence
end for

Algorithm 3 Discrete Parallel Score Matching (DPSM)

Input: data $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$, batch-size p
Discretize $[0,1]$ into $\{t_0 = 0, ..., t_i, ..., t_N = 1\}$
for $i = 1$ **to** N, in parallel **do**
  Train model $\boldsymbol{s_{\theta_i}}$
  repeat
    sample $\boldsymbol{x}^0_{\pi(1)}$, ..., $\boldsymbol{x}^0_{\pi(p)}$ from $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$
    generate $\boldsymbol{x}^{t_i}_{\pi(1)}$, ..., $\boldsymbol{x}^{t_i}_{\pi(p)}$ using Equation 12
    minimize $\Sigma_j ||\boldsymbol{\varepsilon}_{\boldsymbol{\theta}_i}(\boldsymbol{x}^{t_i}_{\pi(j)}(\boldsymbol{\varepsilon_j})) - (\boldsymbol{\varepsilon_j})||^2$.
  until convergence
end for

Table 1. A comparison of the properties of Parallel Score Matching (PSM) approaches with other generative diffeomorphism-based frameworks.

METHOD:	CNF	SA-DPM	PSM-DPM
COMPUTAT. DEMANDING LOSS	√	×	×
MEMORY BOTTLENECK	√	×	×
LIMITED VARIABILITY IN TIME	√	√	×
NECESSITY TO DEVISE A DIFFEO -MORPHISM	√	×	×
SCORE OPTIMIZATION INTER DEPENDENCY	√	√	×

residual flow with infinitesimal scaling, as invertibility and differentiability are guaranteed by the fact that the learned transformation approximates the diffusion process.
A description of this training procedure is given in Algorithm 3.

Regarding the time-varying Parallel Score Matching (TPSM) approach, we can generate data as in the original framework by iterating the following integration process:

$$\boldsymbol{x}(t_i) = \boldsymbol{x}(t_{i+1}) + \int_{t_i+1}^{t_i} \big[-\frac{1}{2} b(\tau)\big(\boldsymbol{x}(\tau) + s_{\boldsymbol{\theta}_i}(\boldsymbol{x}_\tau, \tau)\big)\big] d\tau, \tag{16}$$

and perform likelihood estimation via

$$\log p(\boldsymbol{x}(t_0)) = \log p(\boldsymbol{x}(t_N))$$
$$+\Sigma_{i=0}^{N-1} \int_{t_i}^{t_{i+1}} \big[-\frac{1}{2} b(\tau) tr \frac{\partial\big(\boldsymbol{x}(\tau) + s_{\boldsymbol{\theta}_i}(\boldsymbol{x}_\tau, \tau)\big)}{\partial \boldsymbol{x}(\tau)}\big] d\tau, \tag{17}$$

where $\boldsymbol{s}_{\boldsymbol{\theta}_i}(\boldsymbol{x}_t, t) = -\frac{\boldsymbol{\varepsilon}_{\boldsymbol{\theta}_i}(\boldsymbol{x}_t, t)}{\sqrt{1-e^{-\int_0^t b(s)ds}}}$.

Similarly, in the case of Discrete Parallel Score Matching (DPSM), we can easily generate data via back-integrating

$$\boldsymbol{x}(t_i) = \boldsymbol{x}(t_{i+1}) - \epsilon\big[-\frac{1}{2} b(t_{i+1})\big(\boldsymbol{x}(t_{i+1}) + s_{\boldsymbol{\theta}_{t_{i+1}}}(\boldsymbol{x}_{t_{i+1}})\big)\big]$$

Likewise, we can perform likelihood estimation as follows

$$\log p(\boldsymbol{x}(t_0)) = \log p(\boldsymbol{x}(t_N))$$
$$+\Sigma_{i=1}^{N}\big[-\epsilon\frac{1}{2} b(t_i) tr \frac{\partial\big(\boldsymbol{x}(t_i) + s_{\boldsymbol{\theta}_i}(\boldsymbol{x}_{t_i})\big)}{\partial \boldsymbol{x}(t_i)}\big].$$

In addition, in both approaches data samples can be generated by integrating the corresponding reverse SDE (Song et al., 2021).

$$d\boldsymbol{x}(t) = -b(t)[\frac{1}{2}\boldsymbol{x}(t) + \nabla_{\boldsymbol{x}(t)} \log p_t(\boldsymbol{x}(t))]dt + \sqrt{b(t)} d\boldsymbol{w}.$$

A comparison of the properties of Parallel Score Matching (PSM) with other generative diffeomorphism-based frameworks can be found in Table 1.

4. Experiments

We compare the performance between DPMs trained via parallel score matching and those trained through the standard approach (SA-DPM). We perform experiments on 2D data of toy distributions, as well as on standard benchmark datasets such as CIFAR-10, CelebA and ImageNet 64x64.

In the case of 2D toy distribution data, we only compare the TPSM approach against the baseline, that is, SA-DPM. The batch size is 512 in all experiments, and we use a learning rate of 10^{-3}. Each network was trained in a single CPU with equal performance and has the same architecture in both approaches. This network is an MLP with three hidden layers, where the ELU activation function provides the nonlinearity.

On standard image benchmark datasets, in the TPSM approach we use the network introduced in (Ho et al., 2020) which, for the sake of fair comparison, we also use in SA-DPM. For the same reason, in the DPSM approach, we use the even smaller basic U-Net, which originally introduced the U-Net architecture in (Ronneberger et al., 2015). For all models, we use a batch size of 32 on ImageNet and CelebA, and a batch size of 128 for CIFAR-10. In DPSM we use a learning rate of 10^{-3} while in TDSM and the SA-DPM this is reduced to 2×10^{-4}. Each neural network in all three approaches was trained on a single GPU with RTX-2080 level of performance. As in the case of 2D data, we settle for the ELU activation function.

4.1. Toy 2D Datasets

In order to visualise the difference in distribution modeling capabilities between the baseline and the TPSM approach, we first test both frameworks on 2D toy data. The two distributions we experiment on are TY (see upper row of Figure 2) and HG (lower row of Figure 2). The time-varying network used is the same in both approaches and for both distributions. More precisely it is an MLP with the following structure: $4 \to 100 \to 150 \to 100 \to 2$. It should be noted that the input of this model is 4 dimensional, as the variability in time of the model is enabled by having the time component t concatenated to the 2d data input $\boldsymbol{x}(t)$. The output is 2 dimensional, as it predicts the score at point $\boldsymbol{x}(t)$ at time t, that is $[s_1(\boldsymbol{\theta}), s_2(\boldsymbol{\theta})] = \boldsymbol{s}_{\boldsymbol{\theta}} = \boldsymbol{s}_{\boldsymbol{\theta}}(x_1(t), x_2(t), t, t)$.

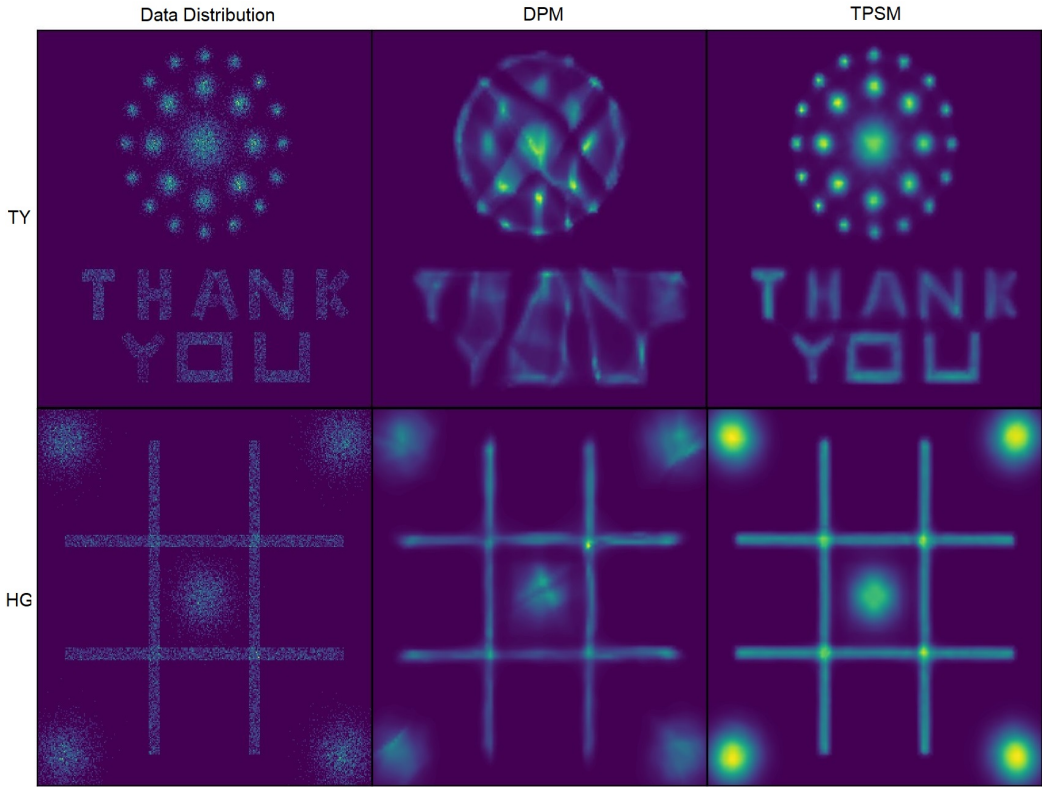

Figure 2. Probability density modeling capabilities of SA (second column) and TPSM (third column) on 2D data of toy distributions.

As previously described, in SA-DPM the model attempts to learn all the scores of the evolution of the distribution dictated by the forward diffusion process:

$$d\boldsymbol{x}(t) = -\frac{1}{2}b(t)[\boldsymbol{x}(t) + \nabla_{\boldsymbol{x}(t)} \log p_t(\boldsymbol{x}(t))]dt;\ t \in [0, 1],$$

where we set $b(t) = 10t$.

On the other hand, in the TPSM approach, we employ 200 such models, where model i is trained to learn the scores of the evolution of the distribution dictated by the same process, however restricted to the time interval $t \in [\frac{i}{200}, \frac{i+1}{200}]$.

Figure 2 shows that TPSM exhibits significantly higher performance in the tasks considered, in comparison to the baseline (SA-DPM), despite the latter having been trained with more than 4x the number of parameter updates. In the first row, the target is the TY distribution, which is a highly complex and challenging 2D distribution to learn. We can notice that the baseline struggles to model such a distribution, as it is not able to separate properly different modalities that comprise the pdf. Naturally, if the size of the model was to be increased significantly, we would expect the SA-DPM to display high performance, but this would increase the already demanding training time. Furthermore, in the more realistic case of high dimensional data, network size would be limited due to memory bottlenecks. Similar differences can be observed in the second row (HG distribution). In Table 2 we show the results of the test loss for both models in both datasets, as well as the difference in traning time. The experiments provided here show that the TPSM approach is characterized by high flexibility of the vector field. TPSM naturally retains the ability to generate samples from the learnt distribution and perform likelihood estimation by employing Equations 16 and 17. In addition, any ODE solver can be used, including adaptive ones, identically as in the case of CNFs. In our experiments we use the Runge-Kutta-4 (RK4) method and we calculate the divergence precisely during validation, as there is no need to use the Hutchinson's trace estimator (Hutchinson, 1990; Grathwohl et al., 2018) due to the low dimensionality of the data.

Table 2. A comparison of the properties of SA-DPM with TPSM (200 blocks, trained in parallel). The results are given in negative log-likelihood (lower is better), and the training time is given in brackets.

METHOD:	SA-DPM	TPSM
TY	0.76 (4H)	0.58 (1H)
HG	1.11 (80 MINS)	1.03 (20 MINS)

4.2. Image Datasets

In this section, we show the performance improvements achieved by applying PSM approaches to image data. To parameterize SA-DPM, we use the time-varying U-Net introducted in (Ho et al., 2020), with the number of channels set to 64. For the TPSM strategy, we model the diffusion process using 10 and 100 blocks, where for each block we employ an instance of the aforementioned U-Net.

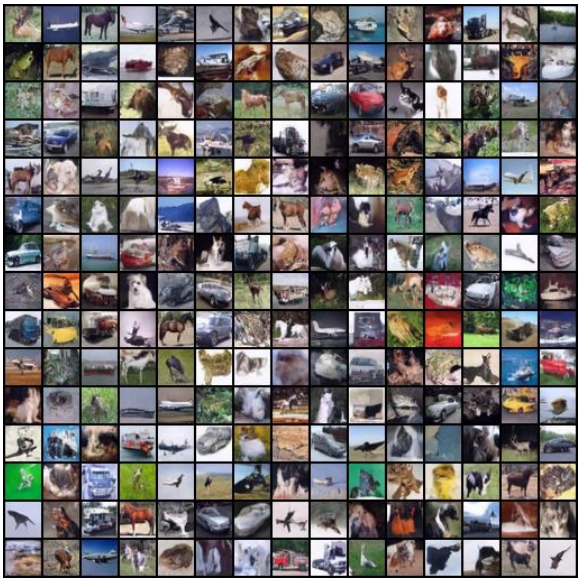

Figure 3. Random non cherry-picked CIFAR-10 samples generated by the DPSM with 1000 training/generative steps.

This approach enables us to reduce the number of parameter updates per block, significantly speeding up the training process. Indeed, for TPSM with 10 blocks (TPSM1) the number of parameter updates is reduced from 800k to 100k on CIFAR-10 and CelebA, while for ImageNet from 2000k to 250k. In addition, we further halve the number of parameter updates in TPSM with 100 blocks (TPSM2), and while this number could be reduced further, in this case we strive to show the improvements in terms of bits/dim (Table 3).

Table 3. Results comparing the performance of SA-DPM and the parallel score-matching approaches, namely TPSM with 10 Block (TPSM1), 100 blocks (TPSM2), and the DPSM with 1000 blocks. We test the models on CIFAR-10, CelebA, and ImageNet (64x64). The results are given in bits/dim (lower is better), and the training time is given in parentheses.

METHOD:	CIFAR-10	CELEBA	IMAGENET
SA-DPM	3.17 (72H)	2.06 (72H)	3.62 (180H)
TPSM1	3.11 (9H)	2.07 (9H)	3.60 (22H)
TPSM2	2.93 (4.5H)	1.90 (4.5H)	3.55 (14H)
DPSM	2.93 (1.5H)	1.94 (2.5H)	3.59 (7H)

Figure 4. Random non cherry-picked CelebA samples generated by the DPSM with 1000 training/generative steps.

The training duration for each approach in each dataset is given in Figure 5. As in the case of 2D data, we set $b(t) = 10t$ in all cases. The RK4 solver (1k steps) was used during density estimation in SA-DPM, TPSM1, and TPSM2. If the Euler method is used, a higher number of integration steps is suggested ($\geq$ 5k), as otherwise sub-optimal results can be obtained, which overestimate the performance of the model. Generated samples from such models can be found in Appendix B. In Appendix C, we provide results in the case that SA-DPM and TPSM2 are trained for the same number of parameter updates.

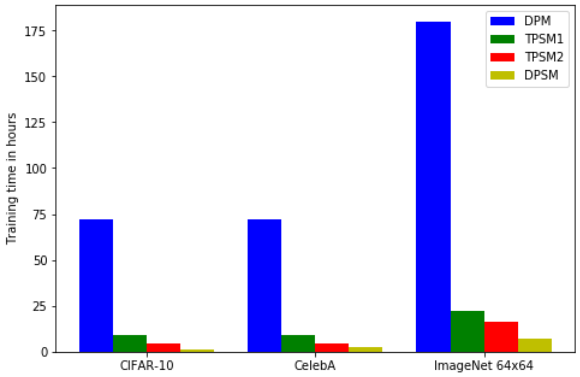

Figure 5. Bar plot showing time differences between PSM approaches and the SA-DPM.

Table 4. Results related to Table 3 comparing number of parameter updates of SA-DPMs and the parallel score matching approaches, namely TPSM with 10 Block, 100 blocks, and the DPSM with 1000 blocks.

METHOD:	CIFAR-10	CELEBA	IMAGENET
SA-DPM	800K	800K	2000K
TPSM 10B	100K	100K	250K
TPSM 100B	50K	50K	150K
DPSM	50K	100K	250K

The model implemented in the DPSM framework differs significantly, as it does not depend on time. In this case, we choose to utilize the most basic U-Net as introduced and implemented in the initial U-Net paper (Ronneberger et al., 2015). The channel architecture is set as follows $(3, a, a*2, a*4, a*8, a*16, a*8, a*4, a*2, a, 3)$.

When a is set to 64 this architecture coincides with the default implementation. The number of steps in the discretization is set to 1000, that is, we use 1000 basic U-Nets, one per step. The parameters $b(t)$, as before, are set to $b(t) = 10t$. The base non time-varying U-Net is about 3.5 times faster than the time-varying U-Net, which combined with the lower number of parameter updates, makes the DPSM training up to 50 times faster than that of the SA-DPM. Furthermore, the U-Net is lighter allowing the usage of batch sizes that are 4x larger than in the time-varying case. A drawback of using DPSM is that for proper precise likelihood estimation, one must use interpolation methods. Since in our experiments we use a thousand models, we simply define the ODE as a piece-wise constant function, that is, in CNF block i we define the field to be constant, as defined by the non time-varying U-Net i which was trained to model score $s_i(\boldsymbol{\theta})$ at time t_i. We use the Euler method with 5k steps for likelihood estimation, and 1k steps for generation, i.e., we do not interpolate in the generation process. Generated samples of CIFAR-10 and CelebA are given in Figures 3 and 4 respectively. The number of parameter updates for each model and dataset is given in Table 4.

5. Limitations and Future Work

5.1. Number of Computing Units

As the premise of the framework is parallel computing, the number of computing units required for the DPSM is large. Indeed, the true parallelization benefits of DPSM can only be achieved through access to computing clusters. On the other hand, the TPSM framework can be utilized by users possessing a single GPU, if the number of blocks is set to be low. Indeed, as shown above, training the 10 CNF blocks in a series takes about the same time as training a single block in the SA-DPM framework, but the performance is increased as shown in Table 3. However, if the number of CNF blocks is increased significantly to the order of the steps in DPSM, then the usage of a cluster is required.

5.2. Likelihood Estimation via DPSM

In DPSM we must interpolate between consequent models in order to perform proper likelihood estimation. Due to the interpolation, the number of integration steps must be set higher than usual slowing down the process.

5.3. Model loading

While the training can be parallelized, the generative and likelihood estimation processes are serial by nature. Hence, during these processes, at time i, the trained model i needs to be loaded from the disk and substitute the previously loaded model $i-1$ in memory. For a large number of models, this slows the generative and likelihood estimation processes. However, in our experiments, since the model that we use in the DPSM framework is lighter, this allows 4x larger batch-sizes in memory for these processes. In our experiments, these two factors cancelled each other and we could generate the same number of pictures per time unit using DPSM, as compared to the baseline SA-DPM.

5.4. Tailored TPSM Models and Powerful Clusters

In this paper, due to computational constrains, we opt to use smaller, older models on all three frameworks (SA-DPM, TPSM, DPSM), as our goal is simply to show that the latter approaches are superior to the first one. In all cases, in the individual blocks (steps) in the TPSM (DPSM) approach, we used models equal or smaller than we did in SA-DPM for fair comparison. Removing these limiters, in the future, it would be beneficial to try to construct bespoke models for the TPSM/DPSM frameworks and see how much these strategies can improve the state of the art. Furthermore, researchers with access to more advanced computing clusters (containing TPU-s with large memories), could test the limits of these approaches.

6. Conclusion

We have presented Parallel Score Matching strategies for training diffusion probabilistic models. We exploited the inherent properties of diffusion models which enable modeling of each score separately. To the best of our knowledge, this property has never been exploited previous works in implementations of DPMS. We showed that learning different groups of scores in parallel via independent neural networks is effective and allows great improvements of training time, while enabling better model flexibility and performance.

References

Behrmann, J., Grathwohl, W., Chen, R. T. Q., Duvenaud, D., and Jacobsen, J.-H. Invertible residual networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 573–582. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/behrmann19a.html.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations, 2018. URL https://arxiv.org/abs/1806.07366.

Dinh, L., Krueger, D., and Bengio, Y. Nice: Non-linear independent components estimation, 2014. URL https://arxiv.org/abs/1410.8516.

Grathwohl, W., Chen, R. T. Q., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models, 2018. URL https://arxiv.org/abs/1810.01367.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *CoRR*, abs/2006.11239, 2020. URL https://arxiv.org/abs/2006.11239.

Hutchinson, M. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation*, 19(2):433–450, 1990. doi: 10.1080/03610919008812866. URL https://doi.org/10.1080/03610919008812866.

Hyvärinen, A. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005. URL http://jmlr.org/papers/v6/hyvarinen05a.html.

Kingma, D. P., Salimans, T., Poole, B., and Ho, J. Variational diffusion models. *CoRR*, abs/2107.00630, 2021. URL https://arxiv.org/abs/2107.00630.

Kobyzev, I., Prince, S. J., and Brubaker, M. A. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3964–3979, nov 2021. doi: 10.1109/tpami.2020.2992934. URL https://doi.org/10.1109%2Ftpami.2020.2992934.

Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021. URL http://jmlr.org/papers/v22/19-1028.html.

Rezende, D. J. and Mohamed, S. Variational inference with normalizing flows, 2015. URL https://arxiv.org/abs/1505.05770.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. *CoRR*, abs/2112.10752, 2021. URL https://arxiv.org/abs/2112.10752.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. URL http://arxiv.org/abs/1505.04597.

Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. *CoRR*, abs/1503.03585, 2015. URL http://arxiv.org/abs/1503.03585.

Song, Y., Garg, S., Shi, J., and Ermon, S. Sliced score matching: A scalable approach to density and score estimation. *CoRR*, abs/1905.07088, 2019. URL http://arxiv.org/abs/1905.07088.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=PxTIG12RRHS.

Tabak, E. G. and Turner, C. V. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013. doi: https://doi.org/10.1002/cpa.21423. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpa.21423.

Tabak, E. G. and Vanden-Eijnden, E. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217 – 233, 2010. doi: cms/1266935020. URL https://doi.org/.

Vincent, P. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011. doi: 10.1162/NECO_a_00142.

A. The Continuous Evolution of the Score During Diffusion

The SDE process of transforming the data distribution to a normal one is the following:

$$d\boldsymbol{x}(t) = -\boldsymbol{x}(t)dt + d\boldsymbol{w} \tag{18}$$

The equivalent process in terms of the evolution of the distribution is given by the following ODE:

$$d\boldsymbol{x}(t) = -\frac{1}{2}\big(\boldsymbol{x}(t) + \nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)\big)dt = f(\boldsymbol{x}(t), t)dt. \tag{19}$$

The instantaneous change of variable formula states that,

$$\frac{d \log p(\boldsymbol{x}(t), t)}{dt} = -tr\frac{\partial f(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}, \tag{20}$$

thus by substituting f:

$$\frac{d \log p(\boldsymbol{x}(t), t)}{dt} = \frac{1}{2}tr\frac{\partial \boldsymbol{x}(t) + \nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)} = \frac{1}{2}D + \frac{1}{2}tr\frac{\partial \nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}. \tag{21}$$

It is easy to notice that,

$$\frac{d\nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)}{dt} = \nabla_{\boldsymbol{x}(t)}\frac{d \log p(\boldsymbol{x}(t), t)}{dt} = \frac{1}{2}\nabla_{\boldsymbol{x}(t)}tr\frac{\partial \nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}. \tag{22}$$

If we denote $s(\boldsymbol{x}(t), t) = \nabla_{\boldsymbol{x}(t)} \log p(\boldsymbol{x}(t), t)$, the last expression becomes:

$$\frac{ds(\boldsymbol{x}(t), t)}{dt} = \frac{1}{2}\nabla_{\boldsymbol{x}(t)}tr\frac{\partial s(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}. \tag{23}$$

Since,

$$\frac{ds(\boldsymbol{x}(t), t)}{dt} = \frac{\partial s(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}\frac{d\boldsymbol{x}(t)}{dt} + \frac{\partial s(\boldsymbol{x}(t), t)}{\partial t} \tag{24}$$

using equation 23, we get:

$$\frac{\partial s(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}\frac{d\boldsymbol{x}(t)}{dt} + \frac{\partial s(\boldsymbol{x}(t), t)}{\partial t} = \frac{1}{2}\nabla_{\boldsymbol{x}(t)}tr\frac{\partial s(\boldsymbol{x}(t), t)}{\partial \boldsymbol{x}(t)}, \tag{25}$$

and therefore

$$\frac{\partial s(\boldsymbol{x}, t)}{\partial t} = \frac{1}{2}\nabla_{\boldsymbol{x}}tr\frac{\partial s(\boldsymbol{x}, t)}{\partial \boldsymbol{x}} - \frac{\partial s(\boldsymbol{x}, t)}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{dt}. \tag{26}$$

We conclude that the score evolves continuously in time as described by the following PDE:

$$\frac{\partial s(\boldsymbol{x}, t)}{\partial t} = \frac{1}{2}\nabla_{\boldsymbol{x}}tr\frac{\partial s(\boldsymbol{x}, t)}{\partial \boldsymbol{x}} + \frac{1}{2}\frac{\partial s(\boldsymbol{x}, t)}{\partial \boldsymbol{x}}(\boldsymbol{x} + s(\boldsymbol{x}, t)). \tag{27}$$

This emphasizes the fact that the scores of all the intermediate distributions defined by the FDP are completely determined by the score of the initial (data) distribution.

B. Additional Generated Samples

Below we provide generated samples from all the models that are present in Table 3.

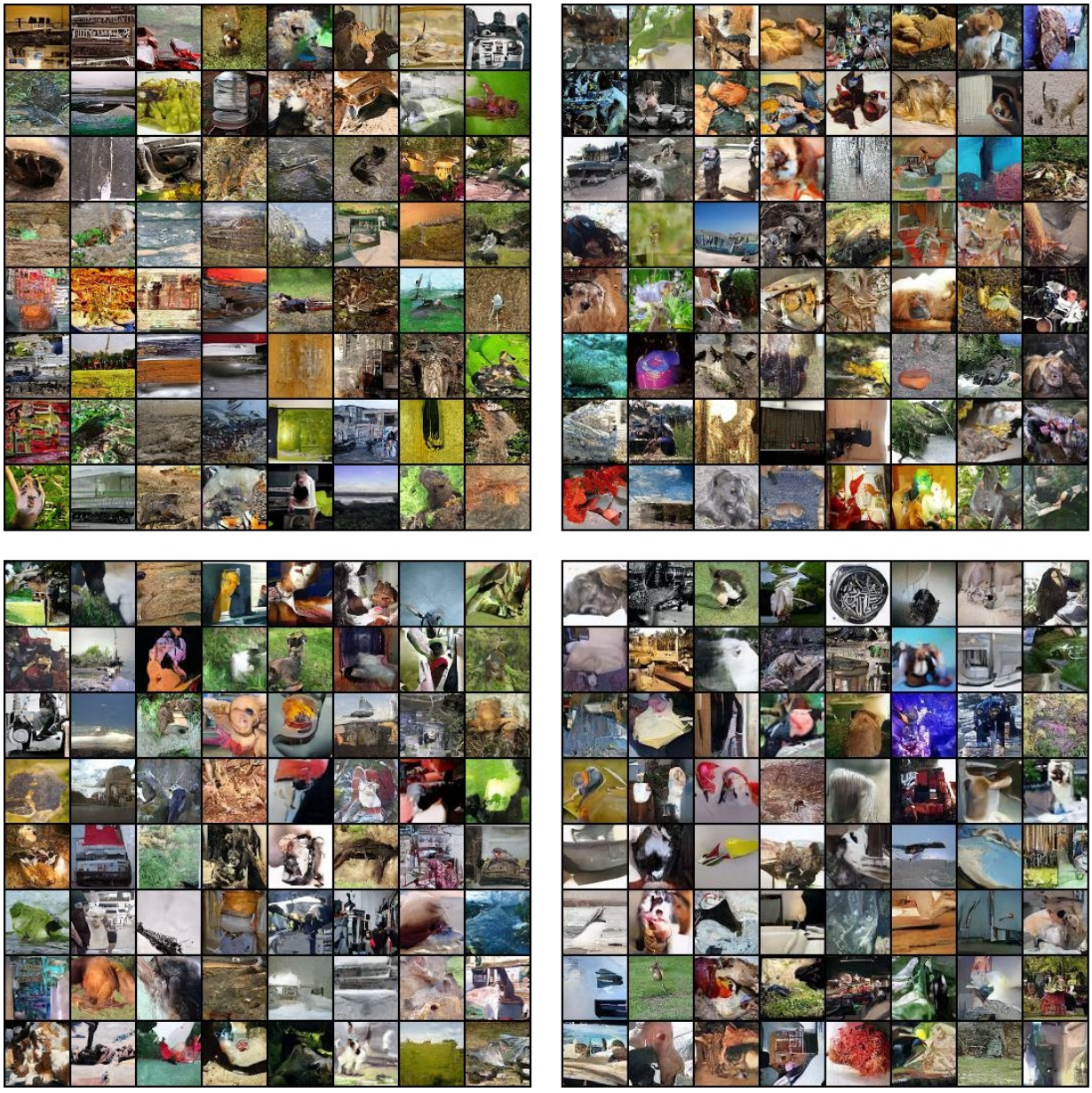

Figure 6. Random non cherry-picked generated ImageNet 64x64 images. Top-Left: SA-DPM, 2000k parameter updates, 1 CNF block, 10k sampling steps via the reverse SDE. Top-Right: TPSM, 250k parameter updates, 10 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Left: TPSM, 150k parameter updates, 100 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Right: DPSM, 250k parameter updates, 1000 CNF blocks, 1k sampling steps via the reverse SDE.

Figure 7. Random non cherry-picked generated CIFAR-10 images. Top-Left: SA-DPM, 800k parameter updates, 1 CNF block, 10k sampling steps via the reverse SDE. Top-Right: TPSM, 100k parameter updates, 10 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Left: TPSM, 50k parameter updates, 100 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Right: DPSM, 50k parameter updates, 1000 CNF blocks, 1k sampling steps via the reverse SDE.

Figure 8. Random non cherry-picked generated CelebA images. Top-Left: SA-DPM, 800k parameter updates, 1 CNF block, 10k sampling steps via the reverse SDE. Top-Right: TPSM, 100k parameter updates, 10 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Left: TPSM, 50k parameter updates, 100 CNF blocks, 10k sampling steps via the reverse SDE. Bottom-Right: DPSM, 100k parameter updates, 1000 CNF blocks, 1k sampling steps via the reverse SDE.

C. Additional Results

Below we give results in the case that SA-DPM and TPSM2 are trained for the same number of parameter updates.

Table 5. Results comparing the performance of SA-DPM and TPSM2 for the same number of parameter updates. We test the models on CIFAR-10, CelebA, and ImageNet (64x64). The results are given in bits/dim (lower is better), the number of parameter updates is given in parentheses, and the training time is given in square brackets.

METHOD:	CIFAR-10	CELEBA	IMAGENET
SA-DPM	3.30 (50K) [4.5H]	2.38 (50K) [4.5H]	3.76 (150K) [14H]
TPSM2	2.93 (50K) [4.5H]	1.90 (50K) [4.5H]	3.55 (150K) [14H]