Lifted Residual Score Estimation

Tejas Jayashankar [*1] **J. Jon Ryu** [*1] **Xiangxiang Xu** [1] **Gregory Wornell** [1]

Abstract

This paper proposes two new techniques to improve the accuracy of score estimation. The first proposal is a new objective function called the *lifted score estimation objective*, which serves as a replacement for the score matching (SM) objective. Instead of minimizing the expected ℓ_2^2-distance between the learned and true score models, the proposed objective operates in the *lifted space* of the outer-product of a vector-valued function with itself. The distance is defined as the expected squared Frobenius norm of the difference between such matrix-valued objects induced by the learned and true score functions. The second idea is to model and learn the *residual approximation error* of the learned score estimator, given a base score model architecture. We empirically demonstrate that the combination of the two ideas called *lifted residual score estimation* outperforms sliced SM in training VAE and WAE with implicit encoders, and denoising SM in training diffusion models, as evaluated by downstream metrics of sample quality such as the FID score.

1. Introduction

Score estimation, i.e., the task of estimating the gradient of the log density of a data distribution $\mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x})$, is a fundamental task in machine learning and statistics, which has diverse applications in implicit generative modeling (Huszár, 2017; Warde-Farley & Bengio, 2022), training of diffusion generative models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song & Ermon, 2019) and learning unnormalized density models (Hyvärinen, 2005), to name a few. To this end, several parametric and non-parametric score estimators have been proposed in recent years, the most notable of which is the parametric *score matching (SM)* framework proposed by Hyvärinen (2005). Under the hood, these methods are inherently minimizing the expected ℓ_2^2-distance between the true score of the data and the parametric estimate of the score.

Score matching is appealing in theory, as the objective does not require explicit knowledge about the true score function. However, it is computationally cumbersome due to its reliance on higher order derivatives of the estimator. In light of this, several works such as sliced score matching (SSM) (Song et al., 2020) and denoising score matching (DSM) (Vincent, 2011) propose computational workarounds for SM. Another line of research within this area is *score estimation*, typically with non-parametric methods. These methods estimate the score by inverting Stein's identity (Li & Turner, 2018) and/or adopt a kernel estimator of the score (Shi et al., 2018; Zhou et al., 2020).

In this paper, we propose two new ideas which are independently applicable for different scenarios towards better parametric score estimation. The first is a new parametric score estimation technique. Inspired by a matrix-kernel regression view for nonparametric score estimation (Zhou et al., 2020), we estimate the score in the *lifted space* of the outer-product of a vector valued function with itself. Our score estimator, $\mathbf{s}_\theta(\mathbf{x}) \approx \mathbf{s}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x})$, is learned by minimizing the expected squared Frobenius norm between a matrix-valued estimator $\mathbf{s}_\theta(\mathbf{x}_1)\mathbf{s}_\theta(\mathbf{x}_2)^\intercal$ and a matrix-valued target $\mathbf{s}(\mathbf{x}_1)\mathbf{s}(\mathbf{x}_2)^\intercal$. We argue that this construction can be interpreted as defining an optimal kernel in the matrix-valued regression view and that the eigenfunction of this kernel is the score. As we operate in the lifted space, we call our estimation framework *lifted score estimation* (LSE).

We also independently propose another simple technique for improving score estimation, which we call *iterative residual estimation*. In practice, a parametric estimator could be misspecified or limited in its expressivity, e.g., a specific neural network architecture. To compensate for this, we propose a new framework that decomposes the score as $\mathbf{s}(\mathbf{x}) \approx \mathbf{s}_{\theta_1^\star}(\mathbf{x}) + \mathbf{s}_{\theta_2^\star}(\mathbf{x}) + \ldots + \mathbf{s}_{\theta_L^\star}(\mathbf{x})$, where each $\mathbf{s}_{\theta_i^\star}(\mathbf{x})$ for $i > 1$ models residual errors in the approximation.

We empirically demonstrate that implicit generative models trained with LSE outperforms SSM in terms of sample quality measured by FID (Heusel et al., 2017) while diffusion models trained with a noisy extension of LSE outperform

[*]Equal contribution [1]Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge MA, USA. Correspondence to: Tejas Jayashankar <tejasj@mit.edu>, J. Jon Ryu <jongha@mit.edu>.

Accepted by the Structured Probabilistic Inference & Generative Modeling workshop of ICML 2024, Vienna, Austria.

DSM. Furthermore, extending both SSM and DSM with only the residual score estimation framework also outperforms the non-residual baselines in terms of sample quality. We thus propose to combine these two techniques to define the *lifted residual score estimator*. To summarize, our main contributions are as follows.

- **New methods**: We propose a new technique for score estimation called residual LSE which combines two novel methods — score estimation in the lifted space (see §3.1) and iterative residual estimation (see §3.2).
- **Empirical validation**: We demonstrate through experimental results that residual LSE outperforms all existing baselines in terms of FID for the task of implicit generative modeling with VAEs (**46.81** vs. 50.06) and WAEs (**44.36** vs. 47.44) (see §6.1). Our proposed method also outperforms DSM in training diffusion models on the CIFAR-10 dataset, as measured by FID (**6.04** vs. 8.65) (see §6.2).

Paper Organization. This paper is organized as follows. The necessary prerequisites are introduced in §2. §3 motivates and introduces the two central techniques in this paper — *lifted score estimation* and *iterative residual estimation*. We meticulously develop our proposed method for direct score estimation in §4 and extend this to noisy score estimation in §5. A summary of our empirical results is available in §6. Additional details about our experiments, implementation and results can be found in Appendices D, E, F and G. We finally end with a discussion and conclude with §7.

2. Preliminaries

We assume that there exists an underlying distribution with a continuously differentiable density $p(\mathbf{x})$ over $\mathcal{X} \subset \mathbb{R}^D$, where the score function is defined as

$$\mathbf{s}(\mathbf{x}) := \nabla_{\mathbf{x}} \log p(\mathbf{x}) \in \mathbb{R}^D.$$

The goal of score estimation is to construct an estimator $\hat{\mathbf{s}}(\mathbf{x})$ from data $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ drawn from $p(\mathbf{x})$. In this paper, we particularly assume a class of parametric score models $\{\mathbf{s}_\theta(\mathbf{x}) \colon \theta \in \Theta\}$ such as neural networks, and aim to find the best hypothesis from the class as a proxy to the underlying score function $\mathbf{s}(\mathbf{x})$. Our goal is to develop improved parametric score estimation methods, using insights from nonparametric score estimation literature. Hence, in this section, we provide a brief overview on the literature of both parametric and nonparametric score estimation.

2.1. Parametric Score Estimation

Exact Score Matching. Hyvärinen (2005) proposed the *score matching* (SM) objective, which we call *exact* SM to distinguish it from the variants that follow,

$$\begin{aligned}
&\mathcal{L}_{\mathsf{SM}}(\mathbf{s}_\theta) \\
&:= \frac{1}{2}(\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{s}(\mathbf{x}) - \mathbf{s}_\theta(\mathbf{x})\|^2] - \mathbb{E}_{p(\mathbf{x})}[\|\mathbf{s}(\mathbf{x})\|^2]) \\
&= -\mathbb{E}_{p(\mathbf{x})}[\mathbf{s}(\mathbf{x})^\intercal \mathbf{s}_\theta(\mathbf{x})] + \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{s}_\theta(\mathbf{x})\|^2] \qquad (1) \\
&\overset{(a)}{=} \mathbb{E}_{p(\mathbf{x})}[\mathrm{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x}))] + \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{s}_\theta(\mathbf{x})\|^2]. \qquad (2)
\end{aligned}$$

Here, (a) follows by integration by parts, under some mild regularity conditions on the density $p(\mathbf{x})$ and score function $\mathbf{s}(\mathbf{x})$. While the SM framework provides a computable objective function, the computation often becomes demanding due to the trace of the Jacobian $\mathrm{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x}))$.

Sliced Score Matching. As a solution, Song et al. (2020) proposed an equivalent objective

$$\begin{aligned}
&\mathcal{L}_{\mathsf{SSM}}(\mathbf{s}_\theta) \\
&:= \mathbb{E}_{p(\mathbf{x})p(\mathbf{v})}[\mathbf{v}^\intercal \nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x}) \mathbf{v}] + \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{s}_\theta(\mathbf{x})\|^2]. \qquad (3)
\end{aligned}$$

Here, Hutchinson's trick (Hutchinson, 1989) is applied to the first term in (2), $\mathrm{tr}(\nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x})) = \mathbb{E}_{p(\mathbf{v})}[\mathbf{v}^\intercal \nabla_{\mathbf{x}} \mathbf{s}_\theta(\mathbf{x}) \mathbf{v}]$, where $p(\mathbf{v})$ is a noise distribution over $\mathbb{R}^D$ such that $\mathbb{E}_{p(\mathbf{v})}[\mathbf{v}\mathbf{v}^\intercal] = \mathsf{I}_D$. They called this computational trick *slicing* and call the method *sliced SM* (SSM).

Denoising Score Matching. To avoid the computational complexity of derivatives altogether, Vincent (2011) proposed to estimate the score of a noisy version of the underlying distribution. Formally, for a given conditional distribution (or noisy channel) $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ parameterized by a noise parameter σ, let $p_\sigma(\tilde{\mathbf{x}}) := \mathbb{E}_{p(\mathbf{x})}[p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})]$ denote the induced marginal distribution. We can then define $\mathbf{s}_\sigma(\tilde{\mathbf{x}}) := \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}})$ as the score of $p_\sigma(\tilde{\mathbf{x}})$ and $\mathbf{s}_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) := \nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}|\mathbf{x})$ as the conditional score function. Furthermore, let $p_\sigma(\mathbf{x}|\tilde{\mathbf{x}})$ denote the induced conditional from the joint $p(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$. Then, applying (1) on a parametric score model $\tilde{\mathbf{x}} \mapsto \mathbf{s}_{\sigma,\theta}(\tilde{\mathbf{x}})$ and the noisy score $\mathbf{s}_\sigma(\tilde{\mathbf{x}})$, the *denoising SM* (DSM) objective is derived as

$$\begin{aligned}
&\mathcal{L}_{\mathsf{DSM}}(\mathbf{s}_{\sigma,\theta}) \\
&:= -\mathbb{E}_{p_\sigma(\tilde{\mathbf{x}})}[\mathbf{s}_\sigma(\tilde{\mathbf{x}})^\intercal \mathbf{s}_{\sigma,\theta}(\mathbf{x})] + \frac{1}{2}\mathbb{E}_{p_\sigma(\tilde{\mathbf{x}})}[\|\mathbf{s}_{\sigma,\theta}(\tilde{\mathbf{x}})\|^2] \\
&\overset{(b)}{=} -\mathbb{E}_{p(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}[\mathbf{s}_\sigma(\tilde{\mathbf{x}}|\mathbf{x})^\intercal \mathbf{s}_{\sigma,\theta}(\mathbf{x})] + \\
&\qquad \frac{1}{2}\mathbb{E}_{p(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}[\|\mathbf{s}_{\sigma,\theta}(\tilde{\mathbf{x}})\|^2], \qquad (4)
\end{aligned}$$

where we apply the (generalized) Tweedie's formula $\mathbb{E}_{p_\sigma(\mathbf{x}|\tilde{\mathbf{x}})}[\mathbf{s}_\sigma(\tilde{\mathbf{x}}|\mathbf{x})] = \mathbf{s}_\sigma(\tilde{\mathbf{x}})$ in (b) (see Theorem B.1 in Appendix B). When $\mathbf{s}_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ is easy to compute for a given choice of $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ such as a Gaussian, the final objective can be computed without derivatives of the estimator. While

DSM was originally proposed as an approximate solution for direct score estimation, i.e., estimating $\mathbf{s}(\mathbf{x})$, it is not widely used for the same as the denoising parameter is hard to tune. Instead, the DSM has played a key role in the recent development of diffusion models.

In §4, we will develop a new objective function called the *lifted score estimation* objective analogous to these objectives, and we will apply Hutchinson's trick and Tweedie's formula to derive computable form of objectives.

2.2. Nonparametric Score Estimation

Stein Gradient Estimator. Li & Turner (2018) proposed estimating $\mathbf{s}(\mathbf{x})$ using the generalized Stein's identity (Stein, 1981; Gorham & Mackey, 2015) $\mathbb{E}_{p(\mathbf{x})}[\mathbf{h}(\mathbf{x})\mathbf{s}(\mathbf{x})^\intercal + \nabla_{\mathbf{x}}\mathbf{h}(\mathbf{x})] = 0$ for a choice of *test function* $\mathbf{h}\colon \mathcal{X} \to \mathbb{R}^D$ with certain regularity conditions. Note that the identity is essentially equivalent to integration by parts used in SM. Li & Turner (2018) considered an empirical version of the identity and proposed to solve the resulting linear system with a regularizer. They call the resulting estimator the *Stein gradient estimator* (SGE). It should be noted that SGE does not have a principled way for extrapolation, and also the choice of the test function, which governs the quality of the estimator, is highly nontrivial.

Spectral Stein Gradient Estimator. To resolve the latter downsides, Shi et al. (2018) proposed the *spectral Stein gradient estimator* (SSGE), where the idea is to use the stack of the top-L eigenfuctions of a kernel as the test function in the SGE framework. Compared to SGE, SSGE has a principled formula for extrapolation based on the Nyström method and the error was theoretically analyzed.

Nonparametric Score Estimators. Zhou et al. (2020) proposed a unifying framework for nonparametric score estimation methods based on a regularized vector-regression formulation (Baldassarre et al., 2012). For a matrix-valued kernel Γ, let $\mathcal{H}_\Gamma$ denote the reproducing kernel Hilbert space. Then, they defined the score estimator as $\hat{\mathbf{s}}_\lambda := \arg\min_{\mathbf{s}\in\mathcal{H}_\Gamma} \mathbb{E}_{\hat{p}(\mathbf{x})}[\|\mathbf{s}(\mathbf{x}) - \hat{\mathbf{s}}(\mathbf{x})\|_2^2] + \frac{\lambda}{2}\|\mathbf{s}\|_{\mathcal{H}_\Gamma}^2$. Based on its closed-form solution (or its variant with other spectral regularization), Zhou et al. (2020) applied a more general version of Stein's identity (essentially integration by parts), and introduced a general nonparametric estimator that can be computed based on linear system of size $MN \times MN$, for a matrix-valued kernel of size $M \times M$ and data size N. They showed that this estimator subsumes both SGE and SSGE as special cases, when the underlying matrix-valued kernel is essentially scalar-valued. They also demonstrated that the nonparametric score estimator using truly matrix-valued kernels such as curl-free kernels can substantially improve the performance of SSGE.

3. New Ideas: Lifting and Residual Learning

In this section, we introduce and motivate two new ideas to improve the quality of score estimation. In §4 and §5, we explain how these ideas can be implemented in a practical way for direct and noisy score function estimation, respectively.

3.1. Lifted Score Estimation

A New Objective. Existing score matching frameworks (Hyvärinen, 2005; Song et al., 2020; Vincent, 2011) estimate the score by minimizing the *expected ℓ_2^2-error* $\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{s}(\mathbf{x}) - \mathbf{f}_\theta(\mathbf{x})\|_2^2]$ between the true score $\mathbf{s}(\mathbf{x})$ and model score $\mathbf{f}_\theta(\mathbf{x})$. In this paper, we propose a new criterion

$$\mathbb{E}_{p(\mathbf{x}_1)p(\mathbf{x}_2)}[\|\mathbf{s}(\mathbf{x}_1)\mathbf{s}(\mathbf{x}_2)^\intercal - \mathbf{f}_\theta(\mathbf{x}_1)\mathbf{f}_\theta(\mathbf{x}_2)^\intercal\|_F^2]. \quad (5)$$

Rather than the standard squared distance in the Euclidean space, we *lift* the scores to the space of their outer products and consider the squared Frobenius distance in the lifted space; hence, we call it the *lifted score estimation* (LSE) objective. While the score matching framework was originally proposed for training unnormalized parametric models, our goal is to construct a good score estimation procedure, and we thus use the term *estimation* instead of *matching*.

Motivation. This new criterion can be motivated from the matrix-kernel regression view, which was adopted in the nonparametric score estimator (Zhou et al., 2020). From the matrix-valued-kernel regression view, it can be argued that an *optimal* kernel which minimizes the approximation error of (5) is the rank-1 kernel $\Gamma(\mathbf{x}, \mathbf{x}') = \mathbf{s}(\mathbf{x})\mathbf{s}(\mathbf{x}')^\intercal$ (see Appendix A). Notably, the only eigenfunction of this kernel is the score function. Hence, we train a score estimator $\mathbf{f}_\theta(\mathbf{x})$ by finding the *rank-1 approximation* $\mathbf{f}_\theta(\mathbf{x})\mathbf{f}_\theta(\mathbf{x})^\intercal$ of the kernel that minimizes the error measured by the expected squared Frobenius norm in (5).

Sign Ambiguity and Solution. Note that unlike the SM objective, an optimal $\mathbf{f}_\theta(\mathbf{x})$ for (5) would be proportional to the score $\mathbf{s}(\mathbf{x})$, but with a potential sign flip. That is, the LSE objective is invariant to scaling with a $\{\pm 1\}$-valued function $\kappa\colon \mathcal{X} \to \{\pm 1\}$, since for $\tilde{\mathbf{f}}_\theta(\mathbf{x}) := \kappa(\mathbf{x})\mathbf{f}_\theta(\mathbf{x})$, we have

$$\begin{aligned}&\mathbb{E}_{p(\mathbf{x}_1)p(\mathbf{x}_2)}[\|\mathbf{s}(\mathbf{x}_1)\mathbf{s}(\mathbf{x}_2)^\intercal - \tilde{\mathbf{f}}_\theta(\mathbf{x}_1)\tilde{\mathbf{f}}_\theta(\mathbf{x}_2)^\intercal\|_F^2] \\ &= \mathbb{E}_{p(\mathbf{x}_1)p(\mathbf{x}_2)}[\|\mathbf{s}(\mathbf{x}_1)\mathbf{s}(\mathbf{x}_2)^\intercal - \mathbf{f}_\theta(\mathbf{x}_1)\mathbf{f}_\theta(\mathbf{x}_2)^\intercal\|_F^2].\end{aligned}$$

This implies that the sign information is absent for each point $\mathbf{x}$ in the LSE framework. However, in practice, we find that it suffices to keep track of a *single* sign $\kappa \in \{\pm 1\}$ to form a score model, i.e., $\kappa\mathbf{f}_\theta(\mathbf{x}) \approx \mathbf{s}(\mathbf{x})$. An intuition behind the success of this simple heuristic can be given as follows. Suppose that both the underlying score function and a parametric model $\mathbf{f}_\theta(\mathbf{x})$ we assume are *sufficiently smooth*

over $\mathbf{x}$. Then, a sign correction function $\kappa(\mathbf{x})$ must not change the sign too abruptly as $\mathbf{x}$ varies, as otherwise it will violate the smoothness. In practice, we empirically find that using a single-sign estimator $\hat{\kappa}_\theta := \operatorname{sgn}(\mathbb{E}_{p(\mathbf{x})}[\mathbf{s}(\mathbf{x})^\intercal \mathbf{f}_\theta(\mathbf{x})])$ works surprisingly well. Later, we will explain how to estimate the best single sign estimate on the fly for each application scenario.

3.2. Iterative Residual Estimation

In practice, when estimating a score function using a parametric model such as a neural network, there will very likely exist a *residual error* even with the best possible fit under a criterion. This is especially true in modern-day applications, where the underlying distribution is extremely high-dimensional and multimodal, such as distributions over images or text.

In such a realistic scenario (also known as model-misspecified case) , it is natural to consider learning the *residual* of the true score after the estimation procedure. For example, suppose that we found the best score model $\mathbf{s}_{\theta_1^\star}(\mathbf{x})$ under a criterion. Defining $\mathbf{r}^{(1)}(\mathbf{x}) := \mathbf{s}(\mathbf{x}) - \mathbf{s}_{\theta_1^\star}(\mathbf{x})$, we can attempt to find the best model $\mathbf{s}_{\theta_2^\star}(\mathbf{x})$ that models this residual. Repeating this procedure, we essentially learn the score by a decomposition

$$\mathbf{s}(\mathbf{x}) \approx \mathbf{s}_{\theta_1^\star}(\mathbf{x}) + \mathbf{s}_{\theta_2^\star}(\mathbf{x}) + \ldots + \mathbf{s}_{\theta_L^\star}(\mathbf{x}).$$

Such an additive decomposition is especially natural in the score function domain (i.e., gradient of log probability), as it can be understood as a *successive refinement* of the underlying distribution, e.g.,

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) \approx \nabla_{\mathbf{x}} \log p_{\theta_1^\star}(\mathbf{x}) + \ldots + \nabla_{\mathbf{x}} \log p_{\theta_L^\star}(\mathbf{x}).$$

Here, for the sake of motivation, we hypothetically suppose that $\mathbf{s}_{\theta_\ell^\star}(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_{\theta_\ell^\star}(\mathbf{x})$ for some density model $p_\theta(\mathbf{x})$. From a practical point of view, the residual learning idea provides a way to improve the quality of score estimation by systematically stacking a given base parametric model.

Note that this idea does not assume a specific learning criterion in the subroutine, and it can be paired with the standard SM-type objectives such as SSM and DSM. Somewhat surprisingly, in this paper, we empirically show that the residual learning exhibits significant performance boost with lifting, while the other pairings sometimes only exhibit marginal improvements.

4. Direct Score Estimation

Lifted Sliced Score Estimation. As argued above, we consider estimating the score with *lifting*, by minimizing the LSE objective

$$\begin{aligned}
&\mathcal{L}_{\mathsf{LSE}}(\mathbf{s}; \mathbf{f}_\theta) \\
&:= \frac{1}{2}\mathbb{E}_{p(\mathbf{x}_1)p(\mathbf{x}_2)}\Big[\|\mathbf{s}(\mathbf{x}_1)\mathbf{s}(\mathbf{x}_2)^\intercal - \mathbf{f}_\theta(\mathbf{x}_1)\mathbf{f}_\theta(\mathbf{x}_2)^\intercal\|_F^2 \\
&\qquad - \|\mathbf{s}(\mathbf{x}_1)\mathbf{s}(\mathbf{x}_2)^\intercal\|_F^2\Big] \\
&= -(\mathbb{E}_{p(\mathbf{x})}[\mathbf{s}^\intercal(\mathbf{x})\mathbf{f}_\theta(\mathbf{x})])^2 + \frac{1}{2}(\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{f}_\theta(\mathbf{x})\|_2^2])^2. \quad (6)
\end{aligned}$$

Compare this to the original SM loss in (1). Here, the term $\mathbb{E}_{p(\mathbf{x})}[\mathbf{s}^\intercal(\mathbf{x})\mathbf{f}_\theta(\mathbf{x})]$ can be computed using integration by parts $-\mathbb{E}_{p(\mathbf{x})}[\operatorname{tr}(\nabla_{\mathbf{x}}\mathbf{f}_\theta(\mathbf{x}))]$ or with Hutchinson's trick (i.e., slicing) as $-\mathbb{E}_{p(\mathbf{x})p(\mathbf{v})}[\mathbf{v}^\intercal\nabla_{\mathbf{x}}\mathbf{f}_\theta(\mathbf{x})\mathbf{v}]$. For the sake of computational efficiency, we can assume the sliced version as the final objective function by default.

Lifted *Residual* Sliced Score Estimation. We can now extend LSE with residual learning. Let $\mathcal{F} = \{\mathbf{f}_\theta \colon \mathcal{X} \to \mathbb{R}^D | \theta \in \theta\}$ denote a class of parametric functions, e.g., a set of functions induced by a neural network architecture. Defining $\mathbf{r}^{(1)}(\mathbf{x}) := \mathbf{s}(\mathbf{x})$ to be the level-1 residual, we can find the best $\mathbf{f}^{(1)} \in \mathcal{F}$ that fits $\mathbf{r}^{(1)}$, i.e.,

$$\mathbf{f}^{(1)} := \arg\min_{\mathbf{f}\in\mathcal{F}} \mathcal{L}_{\mathsf{LSE}}(\mathbf{r}^{(1)}; \mathbf{f}).$$

Recall that $\mathbf{f}^{(1)}$ approximates $\mathbf{r}^{(1)}$ up to a sign flip. Thus, we first estimate the sign as $\kappa^{(1)} := \operatorname{sgn}(\mathbb{E}_{p(\mathbf{x})}[\mathbf{r}^{(1)}(\mathbf{x})^\intercal\mathbf{f}^{(1)}(\mathbf{x})])$, and then define the approximation error as the level-2 residual $\mathbf{r}^{(2)} := \mathbf{r}^{(1)} - \kappa^{(1)}\mathbf{f}^{(1)}$. Then, we can repeatedly apply this learning procedure by considering the residual as a new object to be estimated. In general, for a given level-ℓ residual $\mathbf{r}^{(\ell)}$ for $\ell \geq 1$, we define

$$\mathbf{f}^{(\ell)} = \arg\min_{\mathbf{f}\in\mathcal{F}} \mathcal{L}_{\mathsf{LSE}}(\mathbf{r}^{(\ell)}; \mathbf{f}), \quad (7)$$

whereby the level-$(\ell+1)$ residual is $\mathbf{r}^{(\ell+1)} := \mathbf{r}^{(\ell)} - \kappa^{(\ell)}\mathbf{f}^{(\ell)} = \mathbf{s} - \sum_{i=1}^{\ell}\kappa^{(i)}\mathbf{f}^{(i)}$, and $\kappa^{(i)} := \operatorname{sgn}(\mathbb{E}_{p(\mathbf{x})}[\mathbf{r}^{(i)}(\mathbf{x})^\intercal\mathbf{f}^{(i)}(\mathbf{x})]$. For each ℓ, the ℓ-th LSE objective can be explicitly written as

$$\begin{aligned}
&\mathcal{L}_{\mathsf{LSE}}(\mathbf{r}^{(\ell)}; \mathbf{f}) \\
&= -(\mathbb{E}_{p(\mathbf{x})}[\mathbf{r}^{(\ell)}(\mathbf{x})^\intercal\mathbf{f}(\mathbf{x})])^2 + \frac{1}{2}(\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{f}(\mathbf{x})\|_2^2])^2 \\
&= -\Big(\mathbb{E}_{p(\mathbf{x})}[\mathbf{s}(\mathbf{x})^\intercal\mathbf{f}(\mathbf{x})] - \sum_{i=1}^{\ell-1}\kappa^{(i)}\mathbb{E}_{p(\mathbf{x})}[\mathbf{f}^{(i)}(\mathbf{x})^\intercal\mathbf{f}(\mathbf{x})]\Big)^2 \\
&\qquad + \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{f}(\mathbf{x})\|_2^2]^2, \quad (8)
\end{aligned}$$

where $\mathbb{E}_{p(\mathbf{x})}[\mathbf{s}(\mathbf{x})^\intercal\mathbf{f}(\mathbf{x})]$ can be computed with integration by parts or slicing. Note that the gradient with respect to $\mathbf{f}$ can be computed in an unbiased manner. Additionally, the sign estimators $\kappa^{(i)}$ can be computed by slicing as well.

Practical Optimization Scheme. In practice, solving each level of the optimization problem in a sequential manner as above will incur a $O(L)$-multiplicative computational overhead. A more efficient approach is to emulate solving the series of the optimization problems simultaneously with minibatch samples; see Algorithm 1. Here, the gradient of each objective is estimated in an unbiased manner with the given minibatch of size B, and $\mathsf{GradOpt}(\mathbf{f}, \hat{\nabla}_{\mathbf{f}})$ denotes any gradient-based minimization algorithm. The idea is to simultaneously update each level-ℓ model $\mathbf{f}^{(\ell)}$ based on the level-ℓ objective, as if the models from earlier levels in the previous iterates were perfect. For the sign estimates, we maintain the soft statistics $\mathbb{E}_{p(\mathbf{x})}[\mathbf{r}^{(i)}(\mathbf{x})^\intercal \mathbf{f}^{(i)}(\mathbf{x})]$ using an exponential moving average, which we denote as $\mathsf{EMA}_\beta(a_{\mathsf{past}}, a_{\mathsf{new}}) := \beta a_{\mathsf{past}} + (1-\beta) a_{\mathsf{new}}$ below. We include an example implementation with PyTorch in Appendix F. Ultimately, we construct the final score model as $\hat{\mathbf{s}}_t^{(\ell)}(\mathbf{x}) := \sum_{i=1}^{\ell} \kappa_t^{(i)} \mathbf{f}_t^{(i)}(\mathbf{x})$.

Algorithm 1 Iterative Residual Score Estimation

Initialize $\mathbf{f}_0^{(1)}, \dots, \mathbf{f}_0^{(L)} \in \mathcal{F}$.
for $n = 1, \dots, N$ **do**
  Get minibatch samples $\mathcal{D}_n := \{\mathbf{x}_{n1}, \dots, \mathbf{x}_{nB}\}$
  for $\ell = 1, \dots, L$ **do**
    ▷ Update the function at the level-ℓ
    $\mathbf{f}_n^{(\ell)} \leftarrow \mathsf{GradOpt}(\mathbf{f}_{n-1}^{(\ell)}, \hat{\nabla}_{\mathbf{f}_{n-1}^{(\ell)}} \mathcal{L}_{\mathsf{LSE}}(\mathbf{r}_n^{(\ell)}; \mathbf{f}_{n-1}^{(\ell)}))$
    ▷ Define the level-ℓ residual
    $\mathbf{r}_n^{(\ell)} \leftarrow \mathbf{s} - \sum_{i=1}^{\ell-1} \kappa_{n-1}^{(i)} \mathbf{f}_n^{(i)}$
    ▷ Estimate $\mathbb{E}_{p(\mathbf{x})}[\mathbf{r}_n^{(\ell)}(\mathbf{x})^\intercal \mathbf{f}_n^{(\ell)}(\mathbf{x})]$ w/ minibatch
    $a_n^{(\ell)} \leftarrow \frac{1}{B} \sum_{b=1}^{B} \mathbf{r}_n^{(\ell)}(\mathbf{x}_{nb})^\intercal \mathbf{f}_n^{(\ell)}(\mathbf{x}_{nb})$
    ▷ Update the soft statistics
    $A_n^{(\ell)} \leftarrow \mathsf{EMA}_\beta(A_{n-1}^{(\ell)}, a_n^{(\ell)})$
    $\kappa_n^{(\ell)} \leftarrow \mathrm{sgn}(A_n^{(\ell)})$ ▷ Update the sign

Computational Complexity. By efficiently computing the loss using (8) we only deal with D-dimensional intermediate gradients rather than $D \times D$-dimensional intermediate gradients that would otherwise arise out of direct minimization of (6) as illustrated by the PyTorch implementation in Appendix F. Thus, LSE ($L = 1$) bears no extra overhead than SSM gradient computation. However, with the addition of residual models, we incur an $\mathcal{O}(L)$-multiplicative memory overhead. Reducing this overhead through novel model compression and distillation techniques is focus of our future work.

5. Noisy Score Estimation

We now extend the LSE objective function (6) for the marginal score estimation of noisy samples. Let $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ denote a channel with noise standard deviation σ. We are interested in learning the marginal score $\mathbf{s}_\sigma(\tilde{\mathbf{x}}) = \nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}})$ where $p(\tilde{\mathbf{x}}) = \int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x}) p(\mathbf{x}) \mathrm{d}\mathbf{x}$.

Lifted *Denoising* Score Estimation. Consider two such channels with parameters σ_1 and σ_2 respectively. For the ease of exposition, assume $\sigma_1 = \sigma_2 = \sigma$ such that $p(\tilde{\mathbf{x}}_1) = p(\tilde{\mathbf{x}}_2) = p(\tilde{\mathbf{x}})$. In the lifted space, the LSE objective can be used for score estimation,

$$\begin{aligned}
&\mathcal{L}_{\mathsf{LDSE}}(\mathbf{s}_\sigma; \mathbf{f}_\theta) \\
&\overset{(6)}{=} -(\mathbb{E}_{p(\tilde{\mathbf{x}})}[\mathbf{s}_\sigma(\tilde{\mathbf{x}})^\intercal \mathbf{f}_\theta(\tilde{\mathbf{x}})])^2 + \frac{1}{2}(\mathbb{E}_{p(\tilde{\mathbf{x}})}[\|\mathbf{f}_\theta(\tilde{\mathbf{x}})\|^2])^2 \quad (9) \\
&= -(\mathbb{E}_{p(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}[\mathbf{s}_\sigma(\tilde{\mathbf{x}}|\mathbf{x})^\intercal \mathbf{f}_\theta(\tilde{\mathbf{x}})])^2 + \\
&\qquad \frac{1}{2}(\mathbb{E}_{p(\tilde{\mathbf{x}})}[\|\mathbf{f}_\theta(\tilde{\mathbf{x}})\|^2])^2. \quad (10)
\end{aligned}$$

Note that (10) follows from (9) by Tweedie's formula $\mathbf{s}_\sigma(\tilde{\mathbf{x}}) = \mathbb{E}_{p(\mathbf{x}|\tilde{\mathbf{x}})}[\mathbf{s}_\sigma(\tilde{\mathbf{x}}|\mathbf{x})]$ (Theorem B.1), which is the same computational trick exploited in DSM.

Note. Though we introduced the method for score estimation with symmetric channels, all theory developed in this section can be easily extended to asymmetric channels in practice (see Appendix E.3).

Lifted *Residual* Denoising Score Estimation. Again, let $\mathcal{F}$ denote a collection of parametric vector-valued functions. We apply residual learning to improve the score estimate. For each $\ell \geq 1$,

$$\mathbf{r}^{(\ell)}(\tilde{\mathbf{x}}) := \mathbf{s}_\sigma(\tilde{\mathbf{x}}) - \sum_{i=1}^{\ell-1} \kappa_\sigma^{(i)} \mathbf{f}^{(i)}(\tilde{\mathbf{x}}).$$

Here, we explicitly assume a dependence of the sign on the noise level σ, as we will apply the framework for learning noisy scores across multiple noise levels in our denoising diffusion model experiments. Using the residual score estimation framework from §4, we can directly apply the ℓ-th residual LSE objective in (8),

$$\begin{aligned}
&\mathcal{L}_{\mathsf{LDSE}}(\mathbf{r}^{(\ell)}; \mathbf{f}) \\
&:= -\Big(\mathbb{E}_{p(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}[\mathbf{s}_\sigma(\tilde{\mathbf{x}}|\mathbf{x})^\intercal \mathbf{f}(\tilde{\mathbf{x}})] - \\
&\sum_{i=1}^{\ell-1} \kappa_\sigma^{(i)} \mathbb{E}_{p(\tilde{\mathbf{x}})}[\mathbf{f}^{(i)}(\tilde{\mathbf{x}})^\intercal \mathbf{f}(\tilde{\mathbf{x}})]\Big)^2 + \frac{1}{2}\mathbb{E}_{p(\tilde{\mathbf{x}})}[\|\mathbf{f}(\tilde{\mathbf{x}})\|^2], \quad (11)
\end{aligned}$$

where (11) again follows from Tweedie's formula (Theorem B.1).

6. Applications in Training Image Generative Models

In this section we apply our proposed methods for training image generative models: first, LSE for training implicit generative models and then LDSE for training diffusion models. We evaluate all methods in terms of sample quality and compare against various score estimation baselines.

6.1. Training Implicit Generative Models via Direct Latent Score Estimation

6.1.1. Background

Implicit Variational Autoencoder (VAE). An implicit VAE is a latent variable model with a stochastic encoder $\mathbf{z} = \mathbf{g}_\theta(\mathbf{x}, \boldsymbol{\epsilon})$, where $p(\boldsymbol{\epsilon})$ is an auxilliary noise distribution, and a deterministic decoder $p_\psi(\mathbf{x}|\mathbf{z})$. These models are trained by maximizing the evidence lower bound (ELBO) (Kingma & Welling, 2014),

$$\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left[\log p_\psi(\mathbf{x}|\mathbf{z})p(\mathbf{z})\right] + h(q_\theta(\mathbf{z}|\mathbf{x})), \tag{12}$$

where $h(q_\theta(\mathbf{z}|\mathbf{x})) \triangleq \mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}\left[-\log q_\theta(\mathbf{z}|\mathbf{x})\right]$ denotes the differential entropy of the implicit encoder. Li & Turner (2018) show that the gradient of the differential entropy can be computed via an estimate of the conditional score $\nabla_\mathbf{z} \log q_\theta(\mathbf{z}|\mathbf{x})$,

$$\begin{aligned}
&\nabla_\theta h(q_\theta(\mathbf{z}|\mathbf{x})) \\
&= -\nabla_\theta \mathbb{E}_{p(\boldsymbol{\epsilon})}[\log q_\theta(\mathbf{g}_\theta(\mathbf{x}, \boldsymbol{\epsilon})|\mathbf{x})] \\
&= -\mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_\theta \mathbf{g}_\theta(\mathbf{x}, \boldsymbol{\epsilon})^\intercal \nabla_\mathbf{z} \log q_\theta(\mathbf{z}|\mathbf{x})|_{\mathbf{z}=\mathbf{g}_\theta(\mathbf{x},\boldsymbol{\epsilon})}],
\end{aligned} \tag{13}$$

which in practice can be estimated with a conditional score estimator $\hat{\mathbf{s}}_\phi(\mathbf{z}|\mathbf{x}) \approx \nabla_\mathbf{z} \log q_\theta(\mathbf{z}|\mathbf{x})$. Please refer to Appendix D.1 for a more detailed background.

Wasserstein Autoencoder (WAE). For a fixed prior distribution $p(\mathbf{z})$, implicit encoder $\mathbf{z} = \boldsymbol{\mu}_\theta(\mathbf{x}) + \sigma\boldsymbol{\epsilon}$ for $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and deterministic decoder $\mathbf{x} = \mathbf{f}_\psi(\mathbf{z})$, a WAE (Tolstikhin et al., 2018) solves the relaxed optimal transport (OT) problem with a KL regularizer: for a distance function on a metric space c: $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ e.g., the 2-Wasserstein distance $c(x, y) = \|x - y\|^2$, minimize

$$\mathbb{E}_{p(\mathbf{x})q_\theta(\mathbf{z}|\mathbf{x})}\left[c(\mathbf{x}, \mathbf{f}_\psi(\mathbf{z}))\right] - \mathbb{E}_{q_\theta(\mathbf{z})}[\log p(\mathbf{z})] - h(q_\theta(\mathbf{z})). \tag{14}$$

Similar to implicit VAEs, the gradient of the differential entropy can be written as

$$\begin{aligned}
&\nabla_\theta h(q_\theta(\mathbf{z})) \\
&= -\mathbb{E}_{p(\mathbf{x})p(\boldsymbol{\epsilon})}[\nabla_\theta \boldsymbol{\mu}_\theta(\mathbf{x})^\intercal \nabla_\mathbf{z} \log q_\theta(\mathbf{z})|_{\mathbf{z}=\boldsymbol{\mu}_\theta(\mathbf{x})+\sigma\boldsymbol{\epsilon}}],
\end{aligned}$$

which can be approximated with a good score estimator $\hat{\mathbf{s}}_\phi(\mathbf{z}) \approx \nabla_\mathbf{z} \log q_\theta(\mathbf{z})$. For more background on WAEs, please refer to Appendix D.1.

6.1.2. Experiments

We trained implicit VAEs and WAEs with our proposed score estimation method. Particularly, we compared LSE against various baselines in terms of sample quality on the CelebA dataset (Liu et al., 2023) as evaluated by the Fréchet Inception Distance (FID) (Heusel et al., 2017).

Baselines. Our nonparametric baselines include the Stein gradient estimator (Stein) and SSGE. The parametric baselines include SSM and a Gaussian posterior VAE for our implicit VAE experiments. For more details on the baselines we refer to §2. For a fair comparison with residual LSE, we also implemented a residual version of SSM (see Appendix C.1).

Architecture details. We borrowed the implementation for all baselines from the public SSM implementation, and made only a few modifications. Detailed network architecture and implementation details can be found in Appendices D.3 and F. We experimented with $L = 2$ and $L = 3$ for all residual score models.

Training details. We trained all models with a batch size of 128 for 400k iterations on $1\times$ NVIDIA 3090 GPU. All methods compared under a generative model class (i.e., VAE or WAE) utilized the same encoder and decoder architectures. The only difference lies in the score estimation algorithm. Additional training details can be found in Appendix D.4.

Results. We quantified the sample quality by computing the FID on synthesized samples, as reported in Table 1. Also reported is the average value of the ELBO or WAE objective. The best results were obtained with LSE and its residual versions, where $L = 3$ and $L = 2$ outperform all other VAE and WAE models respectively. Surprisingly, we noticed that residual learning paired with lifted score estimation offers significant gains in FID in comparison to residual SSM. In general, even the $L = 1$ version of LSE was on par or outperformed SSM, demonstrating that optimization in the lifted space is empirically beneficial for score estimation. We observed that the residual versions could sometimes lead to a slight degradation in performance and training instability as we continue to increase L beyond a limit, which was $L = 3$ in these experiments. We postulate that this is attributed to the inability of the neural network to learn useful representations if the residuals are too small and noisy. Synthesized samples from different models are included in Appendices G.1 and G.2.

6.2. Training Diffusion Probabilistic Models via Noisy Score Estimation

6.2.1. Background

Diffusion probabilistic models (DPMs) are a class of generative models based on the idea of thermodynamics diffusion (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song & Ermon, 2019). We take the following unified view in our definition of DPMs as inspired by (Kingma & Gao, 2024; Karras et al., 2022). Let $p(\mathbf{x})$ be the data distribution and let $\lambda(t)$ define a log signal to noise ratio (SNR) schedule with distribution

Table 1: FID and ELBO/WAE loss on the test set obtained using different score estimation methods. $L = 1$ denotes the non-residual variant, and the residual variants of SSM and LSE are denoted by the descriptors $L \in \{2, 3\}$, where $L-1$ denotes the number of residual errors that are modeled.

Method	VAE		WAE	
	FID↓	ELBO↓	FID↓	WAE↓
Guassian Posterior	52.61	4758	-	-
Stein	91.06	4553	49.82	635
SSGE	95.06	4555	49.72	639
SSM ($L = 1$)	50.06	4678	47.44	482
SSM ($L = 2$)	50.21	4604	47.02	434
SSM ($L = 3$)	49.74	4541	46.27	**352**
LSE ($L = 1$)	50.72	4721	46.46	433
LSE ($L = 2$)	47.68	**4531**	**44.36**	497
LSE ($L = 3$)	**46.81**	4711	45.56	583

$p(\lambda)$ where $t \sim \mathcal{U}(0, 1)$. Under this noise schedule we can define a noisy version of $\mathbf{x}$ at noise level $\lambda(t)$ as

$$\tilde{\mathbf{x}}_t := \alpha_t \mathbf{x} + \sigma_t \boldsymbol{\epsilon} \quad \text{where} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}). \tag{15}$$

The log-SNR is calculated as $\lambda(t) = \log(\alpha_t^2/\sigma_t^2)$ and we can define $p_t(\tilde{\mathbf{x}}_t|\mathbf{x}) := \mathcal{N}(\tilde{\mathbf{x}}_t; \alpha_t \mathbf{x}, \sigma_t \mathbf{I})$. In this paper we will constrain $\alpha_t^2 + \sigma_t^2 = 1$, which is commonly known as the *variance preserving* noise schedule. Due to the 1-1 correspondence between log-SNR and timesteps, we can equivalently express the noisy version of $\mathbf{x}$ at noise level λ as,

$$\tilde{\mathbf{x}}_\lambda := \alpha_\lambda \mathbf{x} + \sigma_\lambda \boldsymbol{\epsilon} \quad \text{where} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}). \tag{16}$$

We will interchange between these two conventions when required. Given noisy samples of data, the diffusion objective can be reduced to a weighted denoising objective,

$$\mathcal{L}_{\mathsf{DPM}}(\boldsymbol{\epsilon}_\theta) = \frac{1}{2}\mathbb{E}_{p(\lambda)p(\mathbf{x})p(\boldsymbol{\epsilon})}\left[\frac{w(\lambda)}{2}\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\tilde{\mathbf{x}}_\lambda; \lambda)\|^2\right], \tag{17}$$

where $w(\lambda)$ is a positive scalar-valued weighting function. Note that for the forward process defined in (15), the conditional score is $\mathbf{s}(\tilde{\mathbf{x}}_\lambda|\mathbf{x}) = -\boldsymbol{\epsilon}/\sigma_\lambda$. Thus, (17) can be interpreted as a weighted DSM loss averaged over multiple noise levels,

$$\mathcal{L}_{\mathsf{DPM}}(\boldsymbol{\epsilon}_\theta) = \frac{1}{2}\mathbb{E}_{p(\lambda)p(\mathbf{x})p(\tilde{\mathbf{x}}_\lambda|\mathbf{x})}\left[w'(\lambda)\left\|\mathbf{s}(\tilde{\mathbf{x}}_\lambda|\mathbf{x}) + \frac{\boldsymbol{\epsilon}_\theta(\tilde{\mathbf{x}}_\lambda; \lambda)}{\sigma_\lambda}\right\|^2\right], \tag{18}$$

where $w'(\lambda) := \sigma_\lambda^2 w(\lambda)$ and the marginal score estimator is $\mathbf{s}_\theta(\tilde{\mathbf{x}}_\lambda; \lambda) := -\boldsymbol{\epsilon}_\theta(\tilde{\mathbf{x}}_\lambda; \lambda)/\sigma_\lambda$. For more details about prior work and additional technical details please refer to Appendix E.1.

Table 2: FID, sFID, and Inception Score (IS) of different score estimation methods. DSM ($L = 2$) refers to our residual DSM baseline.

Method	FID↓	sFID↓	IS↑
DSM ($L = 1$)	8.65	10.92	9.08
DSM ($L = 2$)	6.36	**5.35**	9.08
LDSE ($L = 1$)	7.43	10.96	**9.11**
LDSE ($L = 2$)	**6.04**	5.99	9.08

6.2.2. Experiments

We evaluated our denoising score estimation methods by training the iDDPM architecture (Nichol & Dhariwal, 2021) on the CIFAR-10 dataset (Krizhevsky et al., 2009). We also performed some preliminary experiments on the EDM architecture (Karras et al., 2022) but defer the experimental details and results to Appendix E.

Baselines. We implemented our proposed method on top of the public iDDPM implementation. As we are interested in comparing LDSE against DSM, we utilized the "simple" version of the iDDPM model and did not leverage any additional KL regularizers or learned noise schedules. Additional technical details about the baseline can be found in Appendix E.2.

Architecture details. We used the unconditional CIFAR-10 iDDPM architecture in our experiments. Additional architecture and implementation details can be found in Appendix E.2. Python/PyTorch implementations of LDSE can be found in Appendix F. We also implemented a residual version of DSM (see Appendix C.2) to understand the effect of modeling the residual in the original space versus the lifted space. We experimented with both $L = 1$ and $L = 2$ for LDSE and DSM.

Training details. We trained all iDDPM models with a batch size of 128 for 500k iterations on a single NVIDIA 3090 GPU. We used the default hyperparameters provided by the authors for unconditional CIFAR-10 training across all methods. See Appendix E.3 for the training details.

Evaluation. We generated 50,000 samples with each model and measured the FID, sFID (Nash et al., 2021) and Inception Score (IS) (Salimans et al., 2016). This was repeated thrice to account for stochasticity in the results and the best numbers are reported. We used the DDIM sampler (Song et al., 2021a) with 250 sampling steps as no significant gains in FID were attained for $T > 250$ steps per our experiments and the results in (Nichol & Dhariwal, 2021).

Results. The results of our experiments are shown in Table 2. We observed that LDSE ($L = 1$) slightly outperformed

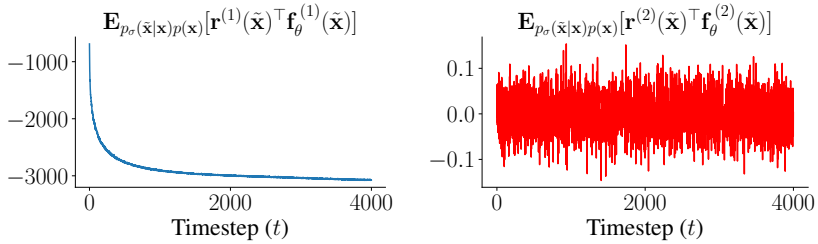


Figure 1: Soft sign calculations with LDSE across the 4000 different timesteps/noise levels. Notice that $\kappa^{(1)}$'s are identically -1 (after applying sgn on the soft signs) but $\kappa^{(2)}$'s differ across noise levels.

the baseline DSM method in terms of FID and IS, thus showing that optimization in the lifted space is a viable alternative to regular DSM. Furthermore, the results validated that iterative residual modeling can significantly improve score estimation in both the unlifted and lifted space. By combining residual learning with lifted estimation, LDSE ($L = 2$) beat all other methods in terms of FID. In Figure 1, we show the soft signs (before applying the sgn funtion) across different noise levels for both the first and second mode in a residual LDSE model. We consistently noticed that $\kappa_t^{(1)}$ is either constantly +1 or -1 for the first mode but that $\kappa_t^{(2)}$ fluctuated across timesteps. This shows that gradients in the lifted space point in the direction of one of the equal minima and leverage the enlarged optimization landscape to improve results. Synthesized samples from different models can be found in Appendix G.3.

In Appendix E.4 we also show that LDSE ($L = 1$) attained comparable results to DSM with the EDM architecture. However, due to the nature of the EDM regression objective (see Appendix E.2), we found that our residual estimators are highly sensitive to the EDM weighting $w(\lambda)$. Extending our setup to alternative diffusion model parametrizations other than DSM/noise estimation is the focus of future work.

7. Concluding Remarks

In this paper, we proposed a new score estimation framework based on two new ideas: lifting and residual learning. We demonstrated that the resulting framework, Lifted Residual Score Estimation (residual LSE or just LSE), achieves impressive downstream task performance for state-of-the-art generative modeling, when we replace the standard SM framework with the proposed one. Empirical results suggest that the LSE framework can be used as a drop-in replacement for SSM or DSM to improve the score estimation performance.

An experienced reader in the field might notice a resemblance between the lifted denoising score estimation and the denoising score matching framework for higher-order derivatives of $\log p_\sigma(\mathbf{x})$ by Meng et al. (2021). Interestingly, it turns out that, if we set $\tilde{\mathbf{x}}_1 = \tilde{\mathbf{x}}_2 = \tilde{\mathbf{x}} \sim p_\sigma(\tilde{\mathbf{x}})$ in the LSE objective (5), it becomes essentially identical to the objective for learning the second-order derivative information in (Meng et al., 2021), as can be seen from the second-order general Tweedie's formula in Theorem B.1. As our approach uses independent distributions $p(\mathbf{x}_1)p(\mathbf{x}_2)$ to define the outer product, we only aim to learn the first-order derivative information (i.e., score function).

References

Ascher, U. M. and Petzold, L. R. *Computer methods for ordinary differential equations and differential-algebraic equations*. SIAM, 1998.

Baldassarre, L., Rosasco, L., Barla, A., and Verri, A. Multi-output learning via spectral filtering. *Mach. Learn.*, 87 (3):259–301, June 2012. ISSN 0885-6125, 1573-0565. doi: 10.1007/s10994-012-5282-y.

De Vito, E., Umanità, V., and Villa, S. An extension of Mercer theorem to matrix-valued measurable kernels. *Appl. Comput. Harmon. Anal.*, 34(3):339–351, 2013.

Gorham, J. and Mackey, L. Measuring sample quality with Stein's method. In *Adv. Neural Inf. Proc. Syst.*, vol. 28, 2015.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Adv. Neural Inf. Proc. Syst.*, vol. 30, 2017.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Adv. Neural Inf. Proc. Syst.*, vol. 33, pp. 6840–6851, 2020.

Huszár, F. Variational inference using implicit distributions. *arXiv preprint arXiv:1702.08235*, 2017.

Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Commun. Stat. - Simul. Comput.*, 18(3):1059–1076, 1989.

Hyvärinen, A. Estimation of non-normalized statistical models by score matching. *J. Mach. Learn. Res.*, 6(4), 2005.

Karras, T., Aittala, M., Aila, T., and Laine, S. Elucidating the design space of diffusion-based generative models. In *Adv. Neural Inf. Proc. Syst.*, vol. 35, pp. 26565–26577, 2022.

Kingma, D. and Gao, R. Understanding diffusion objectives as the ELBO with simple data augmentation. In *Adv. Neural Inf. Proc. Syst.*, vol. 36, 2024.

Kingma, D., Salimans, T., Poole, B., and Ho, J. Variational diffusion models. In *Adv. Neural Inf. Proc. Syst.*, vol. 34, pp. 21696–21707, 2021.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *Int. Conf. Learn. Repr.*, 2014.

Kong, X., Brekelmans, R., and Steeg, G. V. Information-theoretic diffusion. In *Int. Conf. Learn. Repr.*, 2023.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, U. Toronto, 2009.

Li, Y. and Turner, R. E. Gradient estimators for implicit models. In *Int. Conf. Learn. Repr.*, 2018. URL https://openreview.net/forum?id=SJi9WOeRb.

Lipman, Y., Chen, R. T., Ben-Hamu, H., Nickel, M., and Le, M. Flow matching for generative modeling. In *Int. Conf. Learn. Repr.*, 2023.

Liu, X., Gong, C., and Liu, Q. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *Int. Conf. Learn. Repr.*, 2023.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *IEEE Int. Conf. Comp. Vis.*, December 2015.

Lu, C., Zhou, Y., Bao, F., Chen, J., Li, C., and Zhu, J. DPM-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. In *Adv. Neural Inf. Proc. Syst.*, vol. 35, pp. 5775–5787, 2022.

Meng, C., Song, Y., Li, W., and Ermon, S. Estimating high order gradients of the data distribution by denoising. In *Adv. Neural Inf. Proc. Syst.*, vol. 34, pp. 25359–25369, 2021.

Nash, C., Menick, J., Dieleman, S., and Battaglia, P. W. Generating images with sparse representations. In *Proc. Int. Conf. Mach. Learn.*, 2021.

Nichol, A. Q. and Dhariwal, P. Improved denoising diffusion probabilistic models. In *Proc. Int. Conf. Mach. Learn.*, pp. 8162–8171. PMLR, 2021.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. In *Int. Conf. Learn. Repr.*, 2018.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training GANs. In *Adv. Neural Inf. Proc. Syst.*, vol. 29, 2016.

Shi, J., Sun, S., and Zhu, J. A spectral approach to gradient estimation for implicit distributions. In *Proc. Int. Conf. Mach. Learn.*, pp. 4644–4653. PMLR, 2018.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proc. Int. Conf. Mach. Learn.*, pp. 2256–2265. PMLR, 2015.

Song, J., Meng, C., and Ermon, S. Denoising diffusion implicit models. In *Int. Conf. Learn. Repr.*, 2021a.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. In *Adv. Neural Inf. Proc. Syst.*, vol. 32, 2019.

Song, Y., Garg, S., Shi, J., and Ermon, S. Sliced score matching: A scalable approach to density and score estimation. In *Proc. Conf. Uncertainty Artif. Intell.*, pp. 574–584. PMLR, 2020.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *Int. Conf. Learn. Repr.*, 2021b.

Stein, C. M. Estimation of the mean of a multivariate normal distribution. *Ann. Statist.*, pp. 1135–1151, 1981.

Tolstikhin, I., Bousquet, O., Gelly, S., and Schölkopf, B. Wasserstein autoencoders. In *Int. Conf. Learn. Repr.*, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is All You Need. In *Adv. Neural Inf. Proc. Syst.*, vol. 30, 2017.

Vincent, P. A Connection Between Score Matching and Denoising Autoencoders. *Neural Comput.*, 23(7):1661–1674, July 2011. ISSN 0899-7667, 1530-888X. doi: 10.1162/NECO_a_00142.

Warde-Farley, D. and Bengio, Y. Improving Generative Adversarial Networks with Denoising Feature Matching. In *Int. Conf. Learn. Repr.*, 2022.

Zhou, Y., Shi, J., and Zhu, J. Nonparametric Score Estimators. In *Proc. Int. Conf. Mach. Learn.*, pp. 11513–11522. PMLR, 2020.

Appendix

A. From Optimal Matrix-Valued Kernels to Lifting

For a matrix-valued kernel $\Gamma(\mathbf{x}, \mathbf{x}') \in \mathbb{R}^{D\times D}$, let $\{\boldsymbol{\phi}_\ell\}_{\ell\geq 1}$ be an orthonormal basis for the set of $\mathbb{R}^D$-valued square-integrable functions over $\mathcal{X}$, i.e., its Mercer expansion (De Vito et al., 2013, Theorem 3.4) is

$$\Gamma(\mathbf{x}, \mathbf{x}') = \sum_{\ell=1}^{\infty} \lambda_\ell \boldsymbol{\phi}_\ell(\mathbf{x})\boldsymbol{\phi}_\ell(\mathbf{x}')^\intercal.$$

Then, the order-L approximation of the target score function $\mathbf{s}(\mathbf{x})$ with respect to the orthonormal basis is

$$\mathbf{s}^{(L)}(\mathbf{x}) := \sum_{\ell=1}^{L} \mathbb{E}_{p(\mathbf{x}')}[\mathbf{s}(\mathbf{x}')^\intercal \boldsymbol{\phi}_\ell(\mathbf{x}')]\boldsymbol{\phi}_\ell(\mathbf{x}),$$

and the approximation error can be written as

$$\begin{aligned}
&\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{s}^{(L)}(\mathbf{x}) - \mathbf{s}(\mathbf{x})\|_2^2] \\
&= \sum_{\ell\geq L+1} \mathbb{E}_{p(\mathbf{x})}[\mathbf{s}(\mathbf{x})^\intercal \boldsymbol{\phi}_\ell(\mathbf{x})]\mathbb{E}_{p(\mathbf{x}')}[\mathbf{s}(\mathbf{x}')^\intercal \boldsymbol{\phi}_\ell(\mathbf{x}')] \\
&= \sum_{\ell\geq L+1} \mathbb{E}_{p(\mathbf{x})p(\mathbf{x}')}[\boldsymbol{\phi}_\ell(\mathbf{x})^\intercal \mathbf{s}(\mathbf{x})\mathbf{s}(\mathbf{x}')^\intercal \boldsymbol{\phi}_\ell(\mathbf{x}')] \\
&= \sum_{\ell\geq L+1} \mathbb{E}_{p(\mathbf{x})p(\mathbf{x}')}[\boldsymbol{\phi}_\ell(\mathbf{x})^\intercal \Gamma^\star(\mathbf{x}, \mathbf{x}')\boldsymbol{\phi}_\ell(\mathbf{x}')],
\end{aligned}$$

where we define the matrix-valued kernel

$$\Gamma^\star(\mathbf{x}, \mathbf{x}') := \mathbf{s}(\mathbf{x})\mathbf{s}(\mathbf{x}')^\intercal \in \mathbb{R}^{D\times D}.$$

Hence, to minimize the approximation error one needs to choose $\Gamma^\star(\mathbf{x}, \mathbf{x}')$ as a choice for $\Gamma(\mathbf{x}, \mathbf{x}')$, so that the eigenbasis is aligned with $\Gamma^\star(\mathbf{x}, \mathbf{x}')$. Since the kernel $\Gamma^\star(\mathbf{x}, \mathbf{x}')$ has rank 1 by definition, we can learn a score model $\mathbf{f}_\theta(\mathbf{x})$ by considering the rank-1 approximation error, i.e.,

$$\mathbb{E}_{p(\mathbf{x})p(\mathbf{x}')}\|\Gamma^\star(\mathbf{x}, \mathbf{x}') - \mathbf{f}_\theta(\mathbf{x})\mathbf{f}_\theta(\mathbf{x}')\|_F^2,$$

which is the lifted score estimation objective.

B. Tweedie's Formula

Theorem B.1 (A general Tweedie's formula)**.** *For $(\mathbf{x}, \tilde{\mathbf{x}}) \sim p(\mathbf{x})p(\tilde{\mathbf{x}}|\mathbf{x})$, we have*

$$\begin{aligned}
\mathbb{E}_{p(\mathbf{x}|\tilde{\mathbf{x}})}[\mathbf{s}(\tilde{\mathbf{x}}|\mathbf{x})] &= \mathbf{s}(\tilde{\mathbf{x}}), \\
\mathbb{E}_{p(\mathbf{x}|\tilde{\mathbf{x}})}[\mathbf{s}(\tilde{\mathbf{x}}|\mathbf{x})\mathbf{s}(\tilde{\mathbf{x}}|\mathbf{x})^\intercal + \mathbf{s}^{(2)}(\tilde{\mathbf{x}}|\mathbf{x})] &= \mathbf{s}(\tilde{\mathbf{x}})\mathbf{s}(\tilde{\mathbf{x}})^\intercal + \mathbf{s}^{(2)}(\tilde{\mathbf{x}}).
\end{aligned}$$

Here, we define $\mathbf{s}^{(2)}(\tilde{\mathbf{x}}) := \nabla_{\tilde{\mathbf{x}}}^2 \log p(\tilde{\mathbf{x}})$ and $\mathbf{s}^{(2)}(\tilde{\mathbf{x}}|\mathbf{x}) := \nabla_{\tilde{\mathbf{x}}}^2 \log p(\tilde{\mathbf{x}}|\mathbf{x})$.

Proof. Consider

$$\begin{aligned}
\mathbf{s}(\tilde{\mathbf{x}}) &= \nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}) \\
&= \frac{\nabla_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}})}{p(\tilde{\mathbf{x}})} \\
&= \int \nabla_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}}|\mathbf{x}) \frac{p(\mathbf{x})}{p(\tilde{\mathbf{x}})} \, \mathrm{d}\mathbf{x}
\end{aligned}$$

$$
\begin{aligned}
&= \int \frac{\nabla_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}}|\mathbf{x})}{p(\tilde{\mathbf{x}}|\mathbf{x})} \frac{p(\mathbf{x})p(\tilde{\mathbf{x}}|\mathbf{x})}{p(\tilde{\mathbf{x}})} \, \mathrm{d}\mathbf{x} \\
&= \int \nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}|\mathbf{x}) p(\mathbf{x}|\tilde{\mathbf{x}}) \, \mathrm{d}\mathbf{x} \\
&= \mathbb{E}_{p(\mathbf{x}|\tilde{\mathbf{x}})}[\nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}|\mathbf{x})] \\
&= \mathbb{E}_{p(\mathbf{x}|\tilde{\mathbf{x}})}[\mathbf{s}(\tilde{\mathbf{x}}|\mathbf{x})].
\end{aligned}
$$

To show the second identity, note that

$$
\begin{aligned}
\mathbf{s}^{(2)}(\tilde{\mathbf{x}}) &= \nabla^2_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}}) \\
&= \frac{\nabla^2_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}})}{p(\tilde{\mathbf{x}})} - \frac{\nabla_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}}) \nabla_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}})^\intercal}{p(\tilde{\mathbf{x}})^2} \\
&= \frac{\nabla^2_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}})}{p(\tilde{\mathbf{x}})} - \mathbf{s}(\tilde{\mathbf{x}}) \mathbf{s}(\tilde{\mathbf{x}})^\intercal. \qquad (19)
\end{aligned}
$$

Applying the same logic from above on the first term, we have

$$
\frac{\nabla^2_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}})}{p(\tilde{\mathbf{x}})} = \mathbb{E}_{p(\mathbf{x}|\tilde{\mathbf{x}})}\Big[\frac{\nabla^2_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}}|\mathbf{x})}{p(\tilde{\mathbf{x}}|\mathbf{x})}\Big].
$$

However, since

$$
\frac{\nabla^2_{\tilde{\mathbf{x}}} p(\tilde{\mathbf{x}}|\mathbf{x})}{p(\tilde{\mathbf{x}}|\mathbf{x})} = \mathbf{s}(\tilde{\mathbf{x}}) \mathbf{s}(\tilde{\mathbf{x}})^\intercal + \mathbf{s}^{(2)}(\tilde{\mathbf{x}}),
$$

rearranging the terms in (19) leads to the desired relation. □

C. Residual Extension for Sliced and Denoising Score Matching

In §4 and §5 we introduced the residual version of LSE and LDSE respectively. We can similarly define the residual score estimation procedure in the original (unlifted) space for both SSM and DSM.

C.1. Residual Sliced Score Matching

Recall the exact SM objective which we redefine as follows,

$$
\mathcal{L}_{\mathsf{SM}}(\mathbf{s}; \mathbf{s}_\theta) = -\mathbb{E}_{p(\mathbf{x})}[\mathbf{s}(\mathbf{x})^\intercal \mathbf{s}_\theta(\mathbf{x})] + \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{s}_\theta(\mathbf{x})\|^2].
$$

We can now extend this objective with residual learning and the slicing trick from SSM (see §2.1). Let $\mathcal{F} = \{\mathbf{f}_\theta \colon \mathcal{X} \to \mathbb{R}^D | \theta \in \theta\}$ denote a class of parametric functions and let $\mathbf{r}^{(1)}(\mathbf{x}) := \mathbf{s}(\mathbf{x})$ be the level-1 residual. We seek to find the best $\mathbf{f}^{(1)} \in \mathcal{F}$ that fits $\mathbf{r}^{(1)}(\mathbf{x})$, i.e.,

$$
\mathbf{f}^{(1)} := \arg\min_{\mathbf{f} \in \mathcal{F}} \mathcal{L}_{\mathsf{SM}}(\mathbf{r}^{(1)}; \mathbf{f}).
$$

Up to this point we have just minimized the standard SM objective. Now, after obtaining $\mathbf{f}^{(1)}$, we can define the approximation error as the level-2 residual $\mathbf{r}^{(2)} := \mathbf{r}^{(1)} - \mathbf{f}^{(1)}$. Then, we can repeatedly apply this learning procedure by considering the residual as a new object to be estimated. In general, for a given level-ℓ residual $\mathbf{r}^{(\ell)}$ for $\ell \geq 1$, we define

$$
\mathbf{f}^{(\ell)} = \arg\min_{\mathbf{f} \in \mathcal{F}} \mathcal{L}_{\mathsf{SM}}(\mathbf{r}^{(\ell)}; \mathbf{f}). \qquad (20)
$$

Expanding out the objective,

$$
\begin{aligned}
\mathcal{L}_{\mathsf{SM}}(\mathbf{r}^{(\ell)}; \mathbf{f}) &= -\mathbb{E}_{p(\mathbf{x})}[\mathbf{r}^{(\ell)}(\mathbf{x})^\intercal \mathbf{f}(\mathbf{x})] + \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{f}(\mathbf{x})\|_2^2] \\
&= -\Big(\mathbb{E}_{p(\mathbf{x})}[\mathbf{s}(\mathbf{x})^\intercal \mathbf{f}(\mathbf{x})] - \sum_{i=1}^{\ell-1} \mathbb{E}_{p(\mathbf{x})}[\mathbf{f}^{(i)}(\mathbf{x})^\intercal \mathbf{f}(\mathbf{x})]\Big) + \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{f}(\mathbf{x})\|_2^2], \qquad (21)
\end{aligned}
$$

where in (21) we use the slicing trick to compute the first term. We call this objective with the slicing trick, the residual sliced score matching estimator and formally define it as,

$$\mathcal{L}_{\mathsf{SSM}}(\mathbf{r}^{(\ell)}; \mathbf{f}) := -\Big(\mathbb{E}_{p(\mathbf{w})p(\mathbf{x})}[\mathbf{w}^\intercal \nabla \mathbf{f}(\mathbf{x})\mathbf{w}] - \sum_{i=1}^{\ell-1} \mathbb{E}_{p(\mathbf{x})}[\mathbf{f}^{(i)}(\mathbf{x})^\intercal \mathbf{f}(\mathbf{x})]\Big) + \frac{1}{2}\mathbb{E}_{p(\mathbf{x})}[\|\mathbf{f}(\mathbf{x})\|_2^2]. \tag{22}$$

Henceforth, we will refer to our SSM extension with $\ell - 1$ residuals as SSM ($L = \ell$). Algorithm 2 describes the overall procedure for residual SSM.

Algorithm 2 Iterative Residual Sliced Score Matching

Initialize $\mathbf{f}_0^{(1)}, \ldots, \mathbf{f}_0^{(L)} \in \mathcal{F}$.
for $n = 1, \ldots, N$ **do**
  Get minibatch $\mathcal{D}_n := \{\mathbf{x}_{n1}, \ldots, \mathbf{x}_{nB}\}$
  for $\ell = 1, \ldots, L$ **do**
    ▷ Update the function at the level-ℓ
    $\mathbf{f}_t^{(\ell)} \leftarrow \mathsf{GradOpt}(\mathbf{f}_{n-1}^{(\ell)}, \hat{\nabla}_{\mathbf{f}_{n-1}^{(\ell)}} \mathcal{L}_{\mathsf{SSM}}(\mathbf{r}_n^{(\ell)}; \mathbf{f}_{n-1}^{(\ell)}))$
    ▷ Define the level-ℓ residual
    $\mathbf{r}_n^{(\ell)} \leftarrow \mathbf{s} - \sum_{i=1}^{\ell-1} \mathbf{f}_n^{(i)}$

C.2. Residual Denoising Score Matching

Let $p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})$ be a noisy channel. DSM learns the marginal score of noisy samples, $\mathbf{s}_\sigma(\tilde{\mathbf{x}}) = \nabla_{\tilde{\mathbf{x}}} \log p(\tilde{\mathbf{x}})$ where $p(\tilde{\mathbf{x}}) = \int p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})p(\mathbf{x})\mathrm{d}\mathbf{x}$. Again, let $\mathcal{F}$ denote a collection of parametric vector-valued functions. We can apply residual learning to improve the DSM score estimate. For each $\ell \geq 1$,

$$\mathbf{r}^{(\ell)}(\tilde{\mathbf{x}}) := \mathbf{s}_\sigma(\tilde{\mathbf{x}}) - \sum_{i=1}^{\ell-1} \mathbf{f}^{(i)}(\tilde{\mathbf{x}}).$$

Using the residual score estimation framework introduced in the previous section, the ℓ-th residual DSM objective is,

$$\mathcal{L}_{\mathsf{DSM}}(\mathbf{r}^{(\ell)}; \mathbf{f}) := -\Big(\mathbb{E}_{p(\mathbf{x})p_\sigma(\tilde{\mathbf{x}}|\mathbf{x})}[\mathbf{s}_\sigma(\tilde{\mathbf{x}}|\mathbf{x})^\intercal \mathbf{f}(\tilde{\mathbf{x}})] - \sum_{i=1}^{\ell-1} \mathbb{E}_{p(\tilde{\mathbf{x}})}[\mathbf{f}^{(i)}(\tilde{\mathbf{x}})^\intercal \mathbf{f}(\tilde{\mathbf{x}})]\Big)^2 + \frac{1}{2}\mathbb{E}_{p(\tilde{\mathbf{x}})}[\|\mathbf{f}(\tilde{\mathbf{x}})\|^2], \tag{23}$$

where in (23) we use the DSM trick arising from Tweedie's formula.

D. Implicit Generative Model Experiments

D.1. Background

Variational Autoencoder (VAE). Traditional VAEs (Kingma & Welling, 2014) with likelihood distribution, $p_\psi(\mathbf{x}|\mathbf{z})$ and standard normal prior, $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, often parametrize the variational distribution as a Gaussian posterior, $q_\theta(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}_\theta(\mathbf{x}), \sigma_\theta^2(\mathbf{x})\mathbf{I})$, where the mean and covariance are defined by deterministic functions. In practice, for generative modeling, the Gaussian posterior is restrictive in its expressivity and one can instead define an implicit VAE that leverages a parametric stochastic encoder to sample latent variables as $\mathbf{z} = \mathbf{g}_\theta(\mathbf{x}, \boldsymbol{\epsilon})$, where $p(\boldsymbol{\epsilon})$ is an auxilliary noise distribution. The encoder now prescribes a stochastic procedure to generate latent variables by compromising on an explicit posterior model. The overall training objective remains unchanged and these models are still trained by maximizing the evidence lower bound (ELBO) (Kingma & Welling, 2014),

$$\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[\log p_\psi(\mathbf{x}|\mathbf{z})p(\mathbf{z})] + \underbrace{\mathbb{E}_{q_\theta(\mathbf{z}|\mathbf{x})}[-\log q_\theta(\mathbf{z}|\mathbf{x})]}_{\triangleq h(q_\theta(\mathbf{z}|\mathbf{x}))}.$$

Maximization of the ELBO via gradient-based methods involves the computation of the derivative of the differential entropy. Li & Turner (2018) show that this can be readily computed given an estimate of the conditional score and by the fact that the

Generative Model	Name	Configuration
VAE	Implicit Encoder	5×5 conv; m maps; Swish 5×5 conv; $2m$ maps; Swish 5×5 conv; $4m$ maps; Swish 5×5 conv; $8m$ maps; Swish 512 Dense, Swish $D_{\mathbf{z}}$ Dense
WAE	Encoder	concat $[\mathbf{x}$, Swish(Dense($\boldsymbol{\epsilon}$))] along channels 5×5 conv; $2m$ maps; Swish 5×5 conv; $4m$ maps; Swish 5×5 conv; $8m$ maps; Swish 512 Dense, Swish $D_{\mathbf{z}}$ Dense
VAE and WAE	Decoder	Dense, Swish 5×5 conv$^{\intercal}$; $4m$ maps; Swish 5×5 conv$^{\intercal}$; $2m$ maps; Swish 5×5 conv$^{\intercal}$; $1m$ maps; Swish 5×5 conv$^{\intercal}$; c maps; Tanh
VAE	SSM Score ($\mathbf{s}_\theta(\mathbf{z}\|\mathbf{x})$) Res. SSM and LSE Models ($\mathbf{f}_\theta^{(\ell)}(\mathbf{z}\|\mathbf{x})$)	concat $[\mathbf{x}$, Swish(Dense($\mathbf{z}$))] along channels 5×5 conv; m maps; Swish 5×5 conv; $4m$ maps; Swish 5×5 conv; $8m$ maps; Swish 512 Dense, Swish $D_{\mathbf{z}}$ Dense
WAE	SSM Score ($\mathbf{s}_\theta(\mathbf{z})$) Res. SSM and LSE Models ($\mathbf{f}_\theta^{(\ell)}(\mathbf{z})$)	Reshape(Swish(Dense($\mathbf{z}$))]) to 1 channel 5×5 conv; m maps; Swish 5×5 conv; $4m$ maps; Swish 5×5 conv; $8m$ maps; Swish 512 Dense, Swish $D_{\mathbf{z}}$ Dense

Figure 2: Implicit VAE and WAE architectures for CelebA. All convolutions and transposed convolutions use a stride of 2 with appropriate padding dimensions to preserve feature map spatial resolution.

stochasticity is induced by the auxilliary noise,

$$\begin{aligned}\nabla_\theta h(q_\theta(\mathbf{z}|\mathbf{x})) &= -\nabla_\theta \mathbb{E}_{p(\boldsymbol{\epsilon})}[\log q_\theta(\mathbf{g}_\theta(\mathbf{x}, \boldsymbol{\epsilon})|\mathbf{x})] \\ &= -\mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_\theta \mathbf{g}_\theta(\mathbf{x}, \boldsymbol{\epsilon})^\intercal \nabla_{\mathbf{z}} \log q_\theta(\mathbf{z}|\mathbf{x})|_{\mathbf{z}=\mathbf{g}_\theta(\mathbf{x},\boldsymbol{\epsilon})}].\end{aligned}$$

Thus, with a good conditional score estimator $\hat{\mathbf{s}}_\phi(\mathbf{z}|\mathbf{x}) \approx \nabla_{\mathbf{z}} \log q_\theta(\mathbf{z}|\mathbf{x})$, we can approximate the gradient as

$$\nabla_\theta h(q_\theta(\mathbf{z}|\mathbf{x})) \approx -\mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_\theta \mathbf{g}_\theta(\mathbf{x}, \boldsymbol{\epsilon})^\intercal \hat{\mathbf{s}}_\phi(\mathbf{g}_\theta(\mathbf{x}, \boldsymbol{\epsilon})|\mathbf{x})],$$

which in practice can be computed using automatic differentiation techniques.

Wasserstein Autoencoder (WAE). Let $p(\mathbf{z})$ be a fixed prior distribution and $p_\psi(\mathbf{x}|\mathbf{z})$ a likelihood distribution induced by a deterministic decoder $\mathbf{x} = \mathbf{f}_\psi(\mathbf{z})$. A WAE (Tolstikhin et al., 2018) seeks to solve the optimal transport (OT) problem,

$$\inf_{q_\theta : q_\theta(\mathbf{z}) = p(\mathbf{z})} \mathbb{E}_{p(\mathbf{x}) q_\theta(\mathbf{z}|\mathbf{x})} \left[c(\mathbf{x}, \mathbf{f}_\psi(\mathbf{z})) \right], \tag{24}$$

where $c : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a distance function on a metric space, e.g., the 2-Wasserstein distance $c(x, y) = \|x - y\|^2$. While solving the above optimization problem is hard, a relaxed version can be solved with gradient-based methods,

$$\mathbb{E}_{p(\mathbf{x}) q_\theta(\mathbf{z}|\mathbf{x})} \left[c(\mathbf{x}, \mathbf{f}_\psi(\mathbf{z})) \right] + \mathcal{D}(q_\theta(\mathbf{z}), p(\mathbf{z})). \tag{25}$$

Above, the regularizer is an arbitrary divergence that forces the approximate and true posterior to coincide. In our experiments, we choose the KL divergence $D_{KL}(q_\theta(\mathbf{z}) \parallel p(\mathbf{z}))$ and leverage an implicit encoder defined via the re-parametrization

trick (Kingma & Welling, 2014), $\mathbf{z} = \boldsymbol{\mu}_\theta(\mathbf{x}) + \sigma\boldsymbol{\epsilon}$ for $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. As for implicit VAEs, gradient-based optimization of (14) requires computing the entropy term as,

$$\begin{aligned}\nabla_\theta h(q_\theta(\mathbf{z})) &= -\nabla_\theta \mathbb{E}_{q_\theta(\mathbf{z})}[\log q_\theta(\mathbf{z})] \\ &= -\nabla_\theta \mathbb{E}_{p(\mathbf{x})p(\boldsymbol{\epsilon})}[\log q_\theta(\boldsymbol{\mu}_\theta(\mathbf{x}) + \sigma\boldsymbol{\epsilon})] \\ &= -\mathbb{E}_{p(\mathbf{x})p(\boldsymbol{\epsilon})}[\nabla_\theta \boldsymbol{\mu}_\theta(\mathbf{x})^\intercal \nabla_\mathbf{z} \log q_\theta(\mathbf{z})|_{\mathbf{z}=\boldsymbol{\mu}_\theta(\mathbf{x})+\sigma\boldsymbol{\epsilon}}].\end{aligned}$$

With a good conditional score estimator $\hat{\mathbf{s}}_\phi(\mathbf{z}) \approx \nabla_\mathbf{z} \log q_\theta(\mathbf{z})$, we can approximate the gradient as

$$\nabla_\theta h(q_\theta(\mathbf{z})) \approx -\mathbb{E}_{p(\mathbf{x})p(\boldsymbol{\epsilon})}[\nabla_\theta \boldsymbol{\mu}_\theta(\mathbf{x})^\intercal \hat{\mathbf{s}}_\phi(\boldsymbol{\mu}_\theta(\mathbf{x}) + \sigma\boldsymbol{\epsilon})].$$

D.2. Dataset

We used the CelebA dataset (Liu et al., 2015) for our experiments. Following the setup in (Song et al., 2020), we extracted a 140×140 patch from each image and then resized it to 64×64. The images were then mapped to the range $[-1, 1]$ and further dequantized by adding uniform random noise in the interval $[-1/128, 1/128]$ to all pixels.

D.3. Architectures

We reused the architectures of the (implicit) encoder and decoder described in (Song et al., 2020) for all our experiments with the sole modification of replacing all ReLU activations with Swish activations (Ramachandran et al., 2018) as we found this to significantly boost FID results of the various baseline methods. For the sake of completion we summarize the architectures in Table 2. All experiments used a latent dimension size of $D_\mathbf{z} = 32$ and the number of channel maps was set to $m = 64$. As alluded to in §3.1, for training WAEs, we found it sufficient to keep track of a single sign to form the marginal score model for the latent variable $\mathbf{z}$. Surprisingly, in the VAE setting, despite learning the conditional score $\mathbf{s}(\mathbf{z}|\mathbf{x})$, we observed that continuing to keep track of single sign that is independent of $\mathbf{x}$ worked well in practice too.

D.4. Training Details

We trained all models with a batch size of 128 for 400k iterations on a single NVIDIA 3090 GPU. All baselines and proposed methods used the RMSProp optimizer with a learning rate of 0.0001 for both the outer encoder/decoder and score models. Our LSE models use a EMA decay of 0.5 for sign estimation.

D.5. Evaluation

We trained each model three times and evaluated the FID of generated samples across different runs every 10k iterations and reported the best FID for each method across the entire 400k iterations. It should be noted that the FID of generated samples for Stein and SSGE significantly worsened with longer training of the implicit VAE generative model. What is reported is the best FID attained within the earlier stages of training. In contrast, the parametric methods, especially the residual versions, showed a generally monotonically decreasing FID with increasing iteration number.

E. Diffusion Model Experiments

E.1. Background

Prior work. Sohl-Dickstein et al. (2015) first introduced diffusion probabilistic models (DPMs) as deep VAEs (see §D.1) based on the principles of thermodynamic diffusion with a Markov-chain variational posterior that maximizes the ELBO. Several years later, Ho et al. (2020) re-introduced DPMs (DDPMs) with modern neural network architectures and a simplified loss function that set a new state-of-the-art in image generation. Since then, numerous connections to existing literature in statistics, information theory and stochastic differential equations (SDEs) have helped bolster the quality of these models. For example, Song & Ermon (2019) illustrate the equivalence between DDPMs and DSM (see §2.1) at multiple noise levels, thus bridging the areas of diffusion-based models and score-based models. Subsequently, Song et al. (2021b) showed that in continuous time, DPMs can be appropriately interpreted as solving for the reverse of a noising process that evolves as an SDE while Kingma et al. (2021) demonstrated that continuous-time DPMs can interpreted as VAEs and that the variational lower bound is invariant to the noise schedule except for its endpoints, thus bolstering its density estimation capabilities. Following the latter discovery, Kong et al. (2023) show that DPMs can in-fact be used for *exact* likelihood computation by

leveraging techniques from information theory. To further improve DPMs, extensive research has gone into the choice of noise schedules, network architectures and loss functions (Nichol & Dhariwal, 2021; Karras et al., 2022; Kingma & Gao, 2024). Many tangentially discovered frameworks such as rectified flows (Liu et al., 2023) and conditional normalizing flows trained with Gaussian conditional flow matching (Lipman et al., 2023), are also particular instances of (Gaussian) diffusion models with specialized noise schedules and weighted loss functions, as show in (Kingma & Gao, 2024).

Sampling. We can equivalently interpret DPMs as SDEs (Song et al., 2021b) where the forward process in (17) can be expressed as,

$$\mathrm{d}\tilde{\mathbf{x}} = \mathbf{f}(\tilde{\mathbf{x}}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}, \tag{26}$$

where $\mathbf{w}$ is a standard Wiener process and $\tilde{\mathbf{x}}_0 = \mathbf{x}$. The time reversal of this process (i.e., the generative process) is known to follow the reverse SDE,

$$\mathrm{d}\tilde{\mathbf{x}} = \left[\mathbf{f}(\tilde{\mathbf{x}}, t) - g^2(t)\nabla_{\tilde{\mathbf{x}}} \log p_t(\tilde{\mathbf{x}})\right] \mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}. \tag{27}$$

Note that in practice $\nabla_{\tilde{\mathbf{x}}} \log p_t(\tilde{\mathbf{x}})$ would be estimated by the score function $\mathbf{s}_\theta(\tilde{\mathbf{x}}_t; t)$ from DSM (18). Sampling can be simulated through techniques such as annealed Langevin dynamics or ancestral sampling (Song et al., 2021b). While the above reverse SDE is stochastic in nature, there also exists a deterministic process known as the *probability flow ODE* that satisfies the same intermediate marginal distributions,

$$\mathrm{d}\tilde{\mathbf{x}} = \left[\mathbf{f}(\tilde{\mathbf{x}}, t) - \frac{1}{2}g^2(t)\nabla_{\tilde{\mathbf{x}}} \log p_t(\tilde{\mathbf{x}})\right] \mathrm{d}t. \tag{28}$$

The benefit of the ODE formulation is that it can discretized more coarsely and hence sampling can done in fewer timesteps. Furthermore, sampling is possible by plugging in the updates from (28) into black-box ODE solvers, e.g., the Heun 2$^{\text{nd}}$ order solver (Karras et al., 2022). Sampling can be sped even further if (28) can be solved exactly. Lu et al. (2022) show that the exact solution to (28) at timestep t given an initial value at timestep $s < t$ is,

$$\tilde{\mathbf{x}}_t = \frac{\alpha_t}{\alpha_s}\tilde{\mathbf{x}}_s + \alpha_t \int_{\lambda(s)}^{\lambda(t)} e^{-v} \boldsymbol{\epsilon}_\theta \left(\tilde{\mathbf{x}}_{\lambda^{-1}(v)}; \lambda^{-1}(v)\right) \mathrm{d}v. \tag{29}$$

Various samplers can be derived by approximating the exponentially weighted integral in different ways. For example, the widely used DDIM sampler (Song et al., 2021a) is an example of a first-order Taylor expansion of the integral term. At the core of all these algorithms is a score estimator/denoiser, which if learned accurately could improve the quality of samples produced.

E.2. Architectures

We experimented with the iDDPM (Nichol & Dhariwal, 2021) and the EDM (Karras et al., 2022) diffusion model architectures for generative modeling on the CIFAR-10 dataset in our experiments. We implemented our methods on top of the authors' codebase such that their models and each level-ℓ model of the residual LDSE model utilized the same architecture. For example, this means that the baseline model defined by the authors and LDSE ($L = 1$) have the same network backbone and number of parameters. The only change is the objective function itself.

In what follows we will summarize the network architecture of the iDDPM and EDM models. Though each model trains the noise/score estimator with a different noise schedule and training objective, we can define them under the unified objective in (17):

$$\mathcal{L}_{\mathsf{DPM}}(\boldsymbol{\epsilon}_\theta) = \frac{1}{2}\mathbb{E}_{p(\lambda)p(\mathbf{x})p(\boldsymbol{\epsilon})}\left[w(\lambda)\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\tilde{\mathbf{x}}_\lambda; \lambda)\|^2\right].$$

iDDPM. The iDDPM model utilizes a UNet architecture similar to (Ho et al., 2020) with $4\times$ downsampling/upsampling layers and multi-head attention (Vaswani et al., 2017) at the 16×16 and 8×8 spatial resolution feature map layers. The upsampling stack is a mirror image of the downsampling stack with nearest neighbor upsamplers replacing the convolutional downsampling layers. From highest resolution to lowest resolution, the UNet uses [128, 256, 256, 256] channels respectively with dropout of 0.3.

The iDDPM model uses a cosine scaling schedule,

$$\alpha_t = \cos\left(\frac{\pi}{2} \cdot \frac{t+s}{1+s}\right). \tag{30}$$

Under the variance preserving assumption (i.e., $\alpha_t^2 + \sigma_t^2 = 1$), the log-SNR schedule is,

$$\lambda(t) = -2 \log \left(\tan \left(\frac{\pi}{2} \cdot \frac{t+s}{1+s} \right) \right). \tag{31}$$

and the induced distribution over log-SNRs is,

$$p(\lambda) = -\frac{\mathrm{d}}{\mathrm{d}\lambda} \frac{1+s}{\pi/2} \arctan \left(e^{-\lambda/2} \right) \tag{32}$$

$$= \frac{1+s}{2\pi} \operatorname{sech} \left(\lambda/2 \right). \tag{33}$$

In practice, for training, the interval $t \in [0, 1]$ is discretized into 4000 timesteps for training , $s = 0.008$ and $w(\lambda) = 1$.

EDM. The variance preserving diffusion model with EDM preconditioning utilizes the DDPM++ architecture from (Song et al., 2021b). This is also based on the UNet introduced by Ho et al. (2020) with $4\times$ downsampling/upsampling layers and attention at the 16×16 spatial resolution feature map layer. From highest resolution to lowest resolution, the UNet uses [128, 256, 256, 256] channels respectively with dropout of 0.1.

The EDM model uses a unique noise schedule that is constructed by first defining,

$$\tilde{\lambda} := -\frac{1}{2} \log \lambda \quad \text{and} \quad \tilde{\lambda} \sim \mathcal{N}(-1.2, 1.2^2). \tag{34}$$

It is easy to see that $p(\lambda) \sim \mathcal{N}(2.4, 2.4^2)$. Rather than regressing against the unscaled additive noise as in DSM, EDM regresses against the original sample expressed in the following form,

$$\mathbf{x} = \frac{\tilde{\lambda}_{\text{data}}^2}{\alpha_\lambda (\tilde{\lambda}^2 + \tilde{\lambda}_{\text{data}}^2)} \tilde{\mathbf{x}}_\lambda + \frac{\tilde{\lambda} \cdot \tilde{\lambda}_{\text{data}}}{\sqrt{\tilde{\lambda}^2 + \tilde{\lambda}_{\text{data}}^2}} \mathbf{g}, \tag{35}$$

where $\tilde{\lambda}_{\text{data}} = 0.5$. A neural network $\mathbf{g}_\theta$ is trained to estimate $\mathbf{g}$ by minimizing the objective,

$$\mathcal{L}_{\text{EDM}}(\mathbf{g}_\theta) := \mathbb{E}_{p(\tilde{\lambda}) p(\mathbf{x}) p(\boldsymbol{\epsilon})} \left[\tilde{w}(\tilde{\lambda}) \| \mathbf{g} - \mathbf{g}_\theta \left(\tilde{\mathbf{x}}_\lambda; \lambda \right) \|^2 \right]. \tag{36}$$

Using (34) and (35) we can show that,

$$\mathbf{g} = \frac{\sqrt{e^{-\lambda} + \tilde{\lambda}_{\text{data}}^2}}{e^{-\lambda/2} \tilde{\lambda}_{\text{data}}} \mathbf{x} - \frac{\tilde{\lambda}_{\text{data}}}{\alpha_\lambda e^{-\lambda/2} \sqrt{e^{-\lambda} + \tilde{\lambda}_{\text{data}}^2}} \tilde{\mathbf{x}}_\lambda \tag{37}$$

$$= -\frac{\sqrt{e^{-\lambda} + \tilde{\lambda}_{\text{data}}^2}}{\tilde{\lambda}_{\text{data}}} \boldsymbol{\epsilon} + \frac{e^{-\lambda/2}}{\alpha_\lambda \sqrt{e^{-\lambda} + \tilde{\lambda}_{\text{data}}^2} \tilde{\lambda}_{\text{data}}} \tilde{\mathbf{x}}_\lambda. \tag{38}$$

Therefore, in terms of (17) the EDM objective boils down to the unified diffusion objective with noise distribution $p(\lambda)$ and weighting function,

$$w(\lambda) = \frac{e^{-\lambda} + \tilde{\lambda}_{\text{data}}^2}{\tilde{\lambda}_{\text{data}}^2}. \tag{39}$$

E.3. Training Details

We trained all iDDPM models for 500k iterations on the CIFAR-10 dataset with a batch size of 128. As we are interested in comparing LDSE against DSM, we used the "simple" version of the loss defined in (Nichol & Dhariwal, 2021), and didn't include the KL divergence regularization term or the learned noise variance schedule. We also implemented a residual version (see Appendix C.2). We trained all models with default hyperparameters for unconditional CIFAR-10 used by the iDDPM authors on a single NVIDIA 3090 GPU. All LDSE models use a EMA decay of 0.5 for sign estimation.

We compared the same set of models for our EDM experiments on CIFAR-10. However, unlike iDDPM, rather than using DSM we used the EDM regression objective in (36). All models were trained for 100k iterations with a batch size of 512 distributed across 2 x NVIDIA 3090 GPUs. The baseline EDM model uses the default hyperparameters provided by the authors whereas the residual DSM and LDSE models use a smaller learning rate of 0.0001. All LDSE models use a EMA decay of 0.5 for sign estimation.

LDSE-specific training details. In §5 we introduced the LDSE objective but with symmetric channels. During training of diffusion models, each input is corrupted at a different noise levels and we thus leveraged the asymmetric version of the objective in practice. Furthermore, since we use the variance preserving noise corruption process, the channel is parametrized by the log-SNR λ. Let $p_{\lambda_1}(\tilde{\mathbf{x}}|\mathbf{x})$ and $p_{\lambda_2}(\tilde{\mathbf{x}}|\mathbf{x})$ be two possibly different noisy channels. The asymmetric LDSE objective is,

$$\begin{aligned}\mathcal{L}_{\mathsf{LDSE}}(\mathbf{s}_{\lambda_1}, \mathbf{s}_{\lambda_2}; \mathbf{f}_\theta) \\ = -\mathbb{E}_{p(\mathbf{x})p_{\lambda_1}(\tilde{\mathbf{x}}|\mathbf{x})}[\mathbf{s}_{\lambda_1}(\tilde{\mathbf{x}}|\mathbf{x})^\intercal \mathbf{f}_\theta(\tilde{\mathbf{x}}; \lambda_1)] \cdot \mathbb{E}_{p(\mathbf{x})p_{\lambda_2}(\tilde{\mathbf{x}}|\mathbf{x})}[\mathbf{s}_{\lambda_2}(\tilde{\mathbf{x}}|\mathbf{x})^\intercal \mathbf{f}_\theta(\tilde{\mathbf{x}}; \lambda_2)] \\ + \frac{1}{2}\mathbb{E}_{p_{\lambda_1}(\tilde{\mathbf{x}})}[\|\mathbf{f}_\theta(\tilde{\mathbf{x}}; \lambda_1)\|^2] \cdot \mathbb{E}_{p_{\lambda_2}(\tilde{\mathbf{x}})}[\|\mathbf{f}_\theta(\tilde{\mathbf{x}}; \lambda_2)\|^2].\end{aligned} \tag{40}$$

In practice, given a minibatch of samples, we divided it into a batch of paired samples and replaced all expectations with sample averages. The notable difference is that each sample is corrupted at a different noise levels and the outer product in the lifted space is between two different estimators.

E.4. Additional Results

Here we provide some preliminary results with the EDM architecture. We extended the EDM objective in (36) to the lifted space by leveraging LDSE by weighing the objective by the EDM weights (39), which in principle is just a weighted LDSE loss, with an effective weight of $w(\lambda_1)w(\lambda_2)$, where the weights are defined in (39).

In principle the optimal estimator should be identical to the EDM score estimator and indeed for the $L = 1$ case, our LDSE model is almost comparable with the EDM baseline. However, extending the EDM weighted objective for $L > 1$, resulted in training instabilites and large gradients which we attribute to the scaling terms $w(\lambda)$. This leads to an interesting research question — how do we effectively weigh the lifted score estimation loss and how can we extend the residual variants to regression objectives beyond naïve DSM?

Table 3: FID, sFID and Inception Score (IS) of EDM trained with DSM (baseline) and LDSE ($L = 1$).

Method	FID↓	sFID↓	IS↑
DSM	**2.21**	3.83	**9.59**
LDSE ($L = 1$)	2.25	**3.85**	9.48

E.5. Evaluation

We generated 50,000 samples with each model and computed the FID. We did this three times due to the stochasticity of the sampling scheme and reported the best FID for each model. For the iDDPM models, we used the DDIM sampler (Song et al., 2021a) with 250 sampling steps. This was chosen for computational purposes and was also based on the results reported in (Nichol & Dhariwal, 2021), where only marginal improvements in FID were reported with even larger sampling steps. For the EDM models, we used the default sampling noise schedule and the deterministic Heun 2$^{\text{nd}}$ order sampler (Ascher & Petzold, 1998) with 18 sampling steps or 35 function evaluations per sample.

F. Implementation

In this section we provide a general implementation for the lifted score estimator. This template can be used for both lifted slice score matching and lifted denoising score matching.

```python
class ResidualSignedEstimator(nn.Module):
    """
    The lifted estimator model with sign correction.
    """

    def __init__(
        self,
        basis: nn.Module,
        num_timesteps: int,
        decay: float = 0.5,
    ):
        super().__init__()

        self.basis = basis
        self.decay = decay

        # Create a parameter to store the signs
        self.register_parameter(
            "score_signs",
            nn.Parameter(torch.zeros(num_timesteps, len(basis)), requires_grad=False),
        )

    def modes(
        self,
        input: Tensor,
        **kwargs
    ) -> Tensor:
        """
        Return the basis functions evaluated at the
        input (and timestep.)
        """
        return torch.concat(
            [
                basis_l(input, t, **kwargs).unsqueeze(1) for basis_l in self.basis
            ],
            dim=1,
        )  # (b, l, *)

    @torch.no_grad()
    def get_signs(self, **kwargs) -> Tensor:
        """
        Get the signs of the modes.
        """
        # If diffusion model then index with t
        if "t" in kwargs.keys():
            t = kwargs["t"]
            return torch.sign(self.score_signs[t.long()].data + 1e-7)
        else:
            return torch.sign(self.score_signs.data + 1e-7)

    def update_signs(
        self,
        input: Tensor,
        target: Tensor,
        **kwargs
    ) -> Tuple[Tensor, Tensor]:
        """
        Update the signs of the modes.
        """
        # Compute all the modes (b, l, *)
```

```
        f = self.modes(input, **kwargs)

        if "t" not in kwargs.keys():
            # Then use slicing to compute first term
            w = kwargs["w"]
            jac_f_w = kwargs["jac_f_w"]
            # Calculate <s, f> with Hutchinson
            # trace estimator
            sign_magnitude = torch.einsum("nbd,nbdl->l", w, jac_f_w) / np.prod(jac_f_w.shape[:2])
        else:
            # Update the magnitude of the signs
            with torch.no_grad():
                signs = self.get_signs(t)
                fT_f = torch.triu(
                    torch.einsum("bl...,bL...->blL", f, f) / f.shape[0], diagonal=1
                )
                sign_magnitude = torch.einsum(
                    "b...,bl...->bl", target, f
                    ) - torch.einsum(
                    "bl,blL->bL", signs, fT_f
                )

        return f, sign_magnitude

    def signed_modes(
        self,
        input: .Tensor,
        target: Optional[Tensor] = None,
        update: bool = True,
        **kwargs,
    ) -> Tensor:
        """
        Compute the sign corrected modes.
        """
        # Update the signs using EMA
        if self.training and update:
            assert target is not None, (
                "Target must be provided"
                "for training."
            )
            f, signs = self.update_signs(input, target, t, **kwargs)
            with torch.no_grad():
                if "t" in kwargs.keys():
                    t = kwargs["t"]
                    old_signs = self.score_signs.data[t]
                    new_signs = (
                        self.decay * old_signs
                        + (1 - self.decay) * signs
                    )
                    self.score_signs.data[t] = new_signs
                else:
                    old_signs = self.score_signs
                    new_signs = (
                        self.decay * old_signs +
                        (1 - self.decay) * signs
                    )
                    self.score_signs = new_signs
        else:
            f = self.modes(input, **kwargs)

        signs = self.get_signs(**kwargs)
        f = torch.einsum(
                "bl,bl...->bl...",
                signs.detach(),
                f
```

```
            )

        return f

    def forward(
        self,
        input: Tensor,
        update: bool = False,
        return_modes: bool = False,
        **kwargs,
    ) -> Tensor:
        """
        Compute the estimator.
        """

        f = self.f(input, update=update, **kwargs)
        if return_modes:
            return f
        else:
            return torch.sum(f, dim=1)
```

Listing 1: Implementation of the residual estimator in the lifted space.

Next we provide the implementation for the `target` in the sliced version of LSM using JVP. For diffusion models, the conditional score is just the (scaled) additive noise that we add to the sample.

```
# This computes Jf.w which is needed for
# the Hutchinson trace estimator \E[w^TJfw]
def _batched_fwd_and_jvp(
    _target: torch.Tensor,  # (b, d)
    _cotangent: torch.Tensor,  # (n, b, d)
) -> Tuple[torch.Tensor, torch.Tensor]:
    def _f(_y: torch.Tensor) -> torch.Tensor:
        return self.modes(_y)

    def _jvp(_v: torch.Tensor) -> torch.Tensor:
        return jvp(_f, (_target,), (_v,))

    return vmap(_jvp)(_cotangent)
```

Listing 2: V-Mapped JVP calculation for Hutchinson trace estimator.

Now we can define the LSE and LDSE objectives with manual gradient computation for implicit and diffusion generative model training respectively.

```
class LiftedScoreEstimationFunction(torch.autograd.Function):
    @staticmethod
    @torch.cuda.amp.custom_fwd
    def forward(
        ctx: torch.autograd.function.FunctionCtx,
        w1: Tensor,  # (n, b, d)
        f1: Tensor,  # (b, d, l)
        jac_f1_w1: Tensor,  # (n, b, d, l)
        w2: Tensor,  # (n, b, d)
        f2: Tensor,  # (b, d, l)
        jac_f2_w2: Tensor,  # (n, b, d, l)
    ) -> Tensor:
        N, B, D, L = jac_f1_w1.shape

        # Hutchinson trace estimator
        w1T_jac_f1_w1 = torch.einsum("nbd,nbdl->l", w1, jac_f1_w1) / N / B
        w2T_jac_f2_w2 = torch.einsum("nbd,nbdl->l", w2, jac_f2_w2) / N / B

        f1T_f1 = torch.einsum("bdl,bdL->lL", f1, f1) / B
        f2T_f2 = torch.einsum("bdl,bdL->lL", f2, f2) / B
```

```
        # Compute the loss
        f1T_f1_sum = torch.sum(torch.tril(f1T_f1, diagonal=-1), dim=-1)
        f2T_f2_sum = torch.sum(torch.tril(f2T_f2, diagonal=-1), dim=-1)
        loss1_unreduced = -(
            w1T_jac_f1_w1 + f1T_f1_sum
        ) * (w2T_jac_f2_w2 + f2T_f2_sum)
        loss1 = torch.sum(loss1_unreduced, dim=-1) / L

        f1T_f1_diagonal = torch.diagonal(f1T_f1)
        f2T_f2_diagonal = torch.diagonal(f2T_f2)
        loss2_unreduced = 0.5 * f1T_f1_diagonal * f2T_f2_diagonal
        loss2 = torch.sum(loss2_unreduced, dim=-1) / L

        loss = loss1 + loss2
        loss_unreduced = loss1_unreduced + loss2_unreduced

        ctx.save_for_backward(
            w1,
            f1,
            w2,
            f2,
            f1T_f1_diagonal,
            f2T_f2_diagonal,
            w1T_jac_f1_w1,
            f1T_f1_sum,
            w2T_jac_f2_w2,
            f2T_f2_sum,
        )

        return loss, loss1, loss2, loss_unreduced

    @staticmethod
    @torch.cuda.amp.custom_bwd
    def backward(
        ctx: torch.autograd.function.FunctionCtx,
        grad_output_loss: torch.Tensor,
        grad_output_loss1: torch.Tensor,
        grad_output_loss2: torch.Tensor,
        grad_loss_unreduced: torch.Tensor,
    ) -> Tuple[torch.Tensor, ...]:
        (
            w1,
            f1,
            w2,
            f2,
            f1T_f1_diagonal,
            f2T_f2_diagonal,
            w1T_jac_f1_w1,
            f1T_f1_sum,
            w2T_jac_f2_w2,
            f2T_f2_sum,
        ) = ctx.saved_tensors

        f1_coeff = w2T_jac_f2_w2 + f2T_f2_sum
        f1_cumsum_roll = torch.cumsum(f1, dim=-1).roll(1, dims=-1)
        f1_cumsum_roll[:, :, 0] = 0
        grad_f1 = -torch.einsum(
            "l,bdl->bdl", f1_coeff, f1_cumsum_roll
        ) + torch.einsum(
            "bdl,l->bdl", f1, f2T_f2_diagonal
        )  # (b, d, l)
        grad_jac_f1_w1 = -f1_coeff * w1[..., None].expand(-1, -1, -1, f1.shape[-1])  # (n, b, d, l)

        f2_coeff = w1T_jac_f1_w1 + f1T_f1_sum
```

```
        f2_cumsum_roll = torch.cumsum(f2, dim=-1).roll(1, dims=-1)
        f2_cumsum_roll[:, :, 0] = 0
        grad_f2 = -torch.einsum(
            "l,bdl->bdl", f2_coeff, f2_cumsum_roll
        ) + torch.einsum(
            "bdl,l->bdl", f2, f1T_f1_diagonal
        )  # (b, d, l)
        grad_jac_f2_w2 = -f2_coeff * w2[..., None].expand(-1, -1, -1, f2.shape[-1])  # (n, b, d, l)

        return (
            None,
            grad_f1 * weight,
            grad_jac_f1_w1 * weight,
            None,
            grad_f2 * weight,
            grad_jac_f2_w2 * weight,
        )


def lifted_score_estimation_loss(
    w: Tensor,  # (n, b, d)
    f: Tensor,  # (b, d, l)
    jac_f_w: Tensor,  # (n, b, d, l)
) -> Tensor:
    # The loss involves computing the squared value
    # of the expectation which can be simulated
    # by dividing the mini-batch into two
    f1, f2 = torch.split(f, f.shape[0] // 2, dim=0)
    jac_f1_w1, jac_f2_w2 = torch.split(jac_f_w, jac_f_w.shape[1] // 2, dim=1)
    w1, w2 = torch.split(w, w.shape[1] // 2, dim=1)

    return LiftedScoreEstimationFunction.apply(
        w1,
        f1,
        jac_f1_w1,
        w2,
        f2,
        jac_f2_w2,
    )
```

Listing 3: LSE loss function.

```
class LiftedEstimationFunction(torch.autograd.Function):
    @staticmethod
    @torch.cuda.amp.custom_fwd
    def forward(
        ctx: torch.autograd.function.FunctionCtx,
        target1: Tensor,  # (b, *)
        f1: Tensor,  # (b, l, *)
        target2: Tensor,  # (b, *)
        f2: Tensor,  # (b, l, *)
    ) -> Tensor:
        # Do shape checking
        B, L, *_ = f1.shape
        ctx.shape = f1.shape

        # Reshape the inputs
        target1 = target1.reshape(B, -1)
        target2 = target2.reshape(B, -1)
        f1 = f1.reshape(B, L, -1)
        f2 = f2.reshape(B, L, -1)

        target1T_f1 = torch.einsum("bd,bld->l", target1, f1) / B
        f1T_f1 = torch.einsum("bld,bLd->lL", f1, f1) / B
        target2T_f2 = torch.einsum("bd,bld->l", target2, f2) / B
```

```
        f2T_f2 = torch.einsum("bld,bLd->lL", f2, f2) / B

        # Compute the loss
        # \sum_{l=1}^L -2 < s_{(l)} | f_l>^2 + <f_l | f_l>^2
        f1T_f1_sum = torch.sum(torch.tril(f1T_f1, diagonal=-1), dim=-1)
        f2T_f2_sum = torch.sum(torch.tril(f2T_f2, diagonal=-1), dim=-1)
        loss1_unreduced = -(
            target1T_f1 - f1T_f1_sum
        ) * (target2T_f2 - f2T_f2_sum)
        loss1 = torch.sum(loss1_unreduced, dim=-1) / L

        f1T_f1_diagonal = torch.diagonal(f1T_f1)
        f2T_f2_diagonal = torch.diagonal(f2T_f2)
        loss2_unreduced = 0.5 * f1T_f1_diagonal * f2T_f2_diagonal
        loss2 = torch.sum(loss2_unreduced, dim=-1) / L

        loss3 = (
            0.5
            * torch.mean(
                torch.einsum(
                    "bd,bd->b", target1, target1
                )
            )
            * torch.mean(
                torch.einsum(
                    "bd,bd->b", target2, target2
                )
            )
        )

        loss = (
            loss1 + loss2 + loss3
        ) / (target1.shape[-1] ** 2)

        ctx.save_for_backward(
            target1,
            target2,
            f1,
            f2,
            f1T_f1_diagonal,
            f2T_f2_diagonal,
            target1T_f1,
            f1T_f1_sum,
            target2T_f2,
            f2T_f2_sum,
        )

        return loss

    @staticmethod
    @torch.cuda.amp.custom_bwd
    def backward(
        ctx: torch.autograd.function.FunctionCtx,
        grad_output_loss: ensor,
    ) -> Tuple[Tensor, ...]:
        del grad_output_loss

        (
            target1,
            target2,
            f1,
            f2,
            f1T_f1_diagonal,
            f2T_f2_diagonal,
            target1T_f1,
```

```
            f1T_f1_sum,
            target2T_f2,
            f2T_f2_sum,
        ) = ctx.saved_tensors

        f1_coeff = target2T_f2 - f2T_f2_sum
        f1_cumsum_roll = torch.cumsum(f1, dim=1).roll(1, dims=1)
        f1_cumsum_roll[:, 0, :] = 0
        grad_f1 = -torch.einsum(
            "l,bld->bld",
            f1_coeff,
            target1.unsqueeze(1) - f1_cumsum_roll
        ) + torch.einsum(
            "bld,l->bld", f1, f2T_f2_diagonal
        )  # (b, l, d)
        f2_coeff = target1T_f1 - f1T_f1_sum
        f2_cumsum_roll = torch.cumsum(f2, dim=1).roll(1, dims=1)
        f2_cumsum_roll[:, 0, :] = 0
        grad_f2 = -torch.einsum(
            "l,bld->bld",
            f2_coeff,
            target2.unsqueeze(1) - f2_cumsum_roll
        ) + torch.einsum(
            "bld,l->bld", f2, f1T_f1_diagonal
        )  # (b, l, d)

        return (
            None,
            (grad_f1 * weight).reshape(*ctx.shape) / (target1.shape[-1] ** 2),
            None,
            (grad_f2 * weight).reshape(*ctx.shape) / (target1.shape[-1] ** 2),
        )


class LiftedEstimationLoss(nn.Module):
    def forward(
        self,
        target: torch.Tensor,  # (b, *)
        f: torch.Tensor,  # (b, l, *)
    ) -> torch.Tensor:
        f1, f2 = torch.split(f, f.shape[0] // 2, dim=0)
        target1, target2 = torch.split(target, target.shape[0] // 2, dim=0)

        return LiftedEstimationFunction.apply(
            target1,
            f1,
            target2,
            f2,
        )
```

Listing 4: LDSE loss function.

G. Samples

G.1. CelebA - VAE

(a) Gaussian Posterior

(b) Stein

(c) Spectral

(d) SSM ($L = 1$)

Figure 3: VAE samples on CelebA.

(e) SSM ($L = 2$)

(f) SSM ($L = 3$)

(g) LSE ($L = 1$)

(h) LSE ($L = 2$)

(i) LSE ($L = 3$)

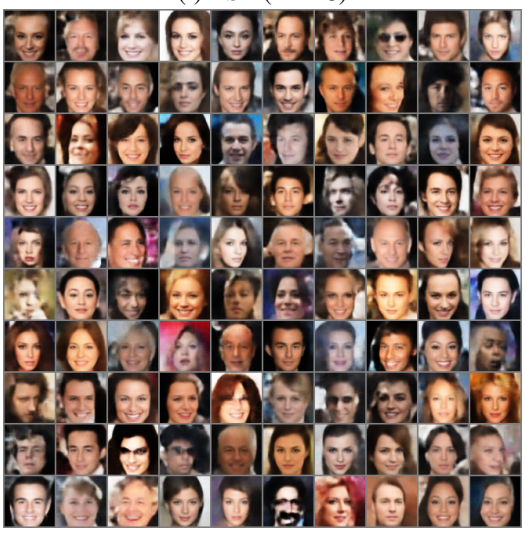


Figure 4: VAE samples on CelebA ctd.

G.2. CelebA - WAE

(a) Stein

(b) Spectral

(c) SSM ($L = 1$)

(d) SSM ($L = 2$)

Figure 5: WAE samples on CelebA.

(e) SSM ($L = 3$)

(f) LSE ($L = 1$)

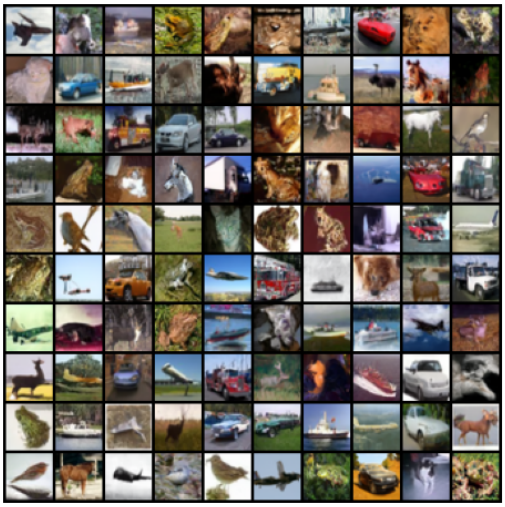
(g) LSE ($L = 2$)

(h) LSE ($L = 3$)

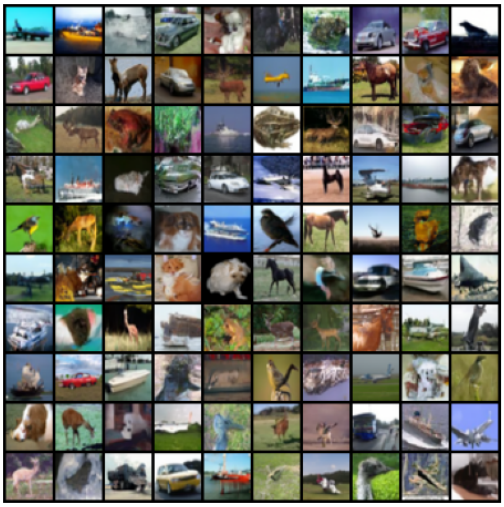
Figure 6: WAE samples on CelebA ctd.

G.3. CIFAR10 - iDDPM

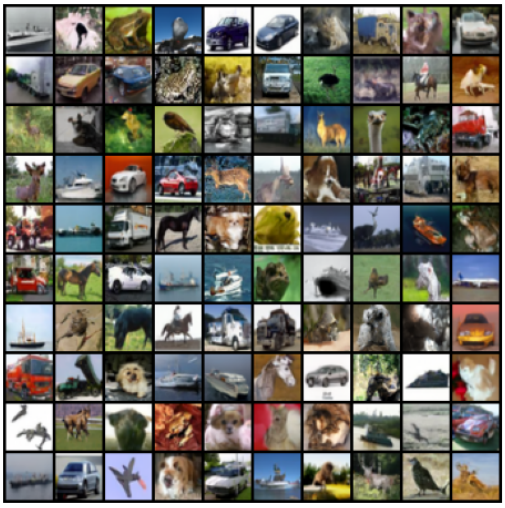
(a) DSM ($L = 1$)



(b) DSM ($L = 2$)



(c) LDSE ($L = 1$)



(d) LDSE ($L = 2$)

Figure 7: iDDPM samples on CIFAR-10