Guiding Time-Varying Generative Models with Natural Gradients on Exponential Family Manifold

Song Liu[1] **Leyang Wang**[2] **Yakun Wang**[1]

[1]School of Mathematics, University of Bristol, United Kingdom
[2]Department of Computer Science, University College London, United Kingdom

Abstract

Optimising probabilistic models is a well-studied field in statistics. However, its connection with the training of generative models remains largely under-explored. In this paper, we show that the evolution of time-varying generative models can be projected onto an exponential family manifold, naturally creating a link between the parameters of a generative model and those of a probabilistic model. We then train the generative model by moving its projection on the manifold according to the natural gradient descent scheme. This approach also allows us to efficiently approximate the natural gradient of the KL divergence without relying on MCMC for intractable models. Furthermore, we propose particle versions of the algorithm, which feature closed-form update rules for any parametric model within the exponential family. Through toy and real-world experiments, we validate the effectiveness of the proposed algorithms. The code of the proposed algorithms can be found at `https://github.com/anewgithubname/iNGD`.

1 INTRODUCTION

Modern generative models [Goodfellow et al., 2014, Ho et al., 2020, Song et al., 2021] have become indispensable tools in machine learning, achieving remarkable success in applications [Rombach et al., 2022, Gu et al., 2022, Li et al., 2019a, Tan et al., 2024]. These generative models are neural networks that transform a latent variable into a higher-dimensional sample. They overcome *classic restrictions imposed on probability density models*, such as positivity, normalisation, and encoding of conditional independence via factorisation; thus, they can be designed freely to capture complex, intricate patterns from the high-dimensional data. Although these models produce highly realistic outputs, they can be hard to train. Training them requires massive amounts of data and maintains a delicate balance between “generators” and “discriminators” [Goodfellow et al., 2014, Arjovsky et al., 2017, Wang et al., 2023] or building effective “bridges” between the reference dataset and target dataset [Song et al., 2021, Lipman et al., 2023, Liu et al., 2023, Bortoli et al., 2021] which can be difficult to design.

Parametric probabilistic models, particularly those in the exponential family [Casella and Berger, 2024, Wainwright et al., 2008], play a central role in modern statistical inference, and have well-established theoretical framework and training algorithms. These models, characterised by their sufficient statistics and natural parameters, define a *probability distribution manifold*, the geometric structure of which inspired efficient optimisation methods such as natural gradient descent (NGD) Amari [1998], ensuring stable and efficient parameter estimation [Amari and Nagaoka, 2000]. However, while exponential family models are versatile *in theory*, their use may be limited in practice: the hand-crafted sufficient statistic may fail to capture complex relationships in data; more flexible sufficient statistics (such as neural nets) result in intractable likelihoods. Thus, parameter estimation that requires a likelihood, such as NGD, cannot be easily applied to fit the model.

We aim to unlock the power of modern generative models through the principled training of probabilistic models.

Our work is inspired by two distinct research directions developed in recent years: time-varying generative models [Ho et al., 2020, Song et al., 2021, Liu et al., 2023, Lipman et al., 2023] and Time Score Matching (TSM) [Choi et al., 2022]. Time-varying generative models generate samples progressively, evolving them over time until they match the target distribution. Meanwhile, TSM learns the instantaneous change of a time-varying distribution from data. Recent work demonstrates that TSM can “project” temporal variations of a dataset onto the parameter space of exponential family distributions [Williams et al., 2025].

The core idea of this paper is to evolve a generative model

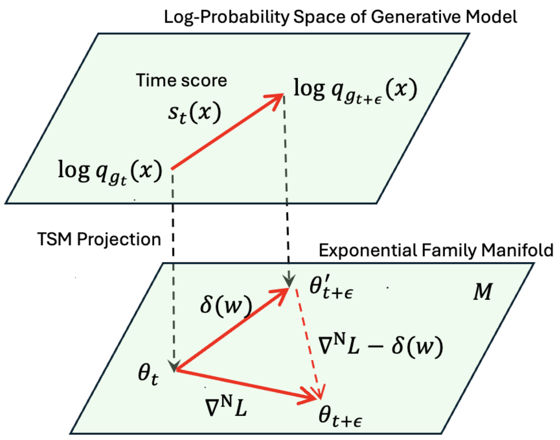

Figure 1: Technical notions illustrated. Matching the projected generative model change $\boldsymbol{\delta}(\boldsymbol{w})$ to the NGD meaning reducing the length of the red dotted line. Symbols are defined in Section 2 and 3.

such that its *projected trajectory* on an exponential family manifold aligns with the trajectory induced by NGD. This way, we get the expressiveness of a modern generative model, and the training efficiency of NGD. The exponential family model acts as a guiding framework for the generative model throughout the training process. An illustration of this idea is in Figure 1. We align the projected changes of the generative model with the NGD update on a parametric manifold (shrinking the length of the red dotted line in Figure 1).

Although our technique applies to all time-varying generative models, we focus on drift-based generative models, where samples are iteratively perturbed by vector-valued functions. We develop two NGD-guided drift-based generative approaches: kernel NGD and neural tangent kernel NGD, both of which admit closed-form sample updates.

2 BACKGROUND

In recent years, it has been recognised that there are many similarities between sampling and optimisation [Wibisono, 2018, Arbel et al., 2019, Chewi, 2024, Cheng et al., 2018, He et al., 2025]. Encouraged by these results, we ask: Given a time-varying generative model g_t, where t is time, can we move its output distribution q_{g_t} toward an optimal distribution q^* using NGD? In literature, generative models are sometimes referred to as implicit models and generative distribution q_{g_t} may not have a parametric density[1], thus optimisation designed for parametric densities cannot be directly applied.

[1]Some generative models, such as normalizing flows [Rezende and Mohamed, 2015], neural ODEs [Chen et al., 2018], admit explicit parametric forms.

Before stating our solutions to this problem, we first introduce a few ideas that are essential for developing the proposed algorithm.

Notations: Vectors ($\boldsymbol{x}$) and matrices ($\boldsymbol{X}$) are denoted in bold. $\dim(\boldsymbol{x})$ denotes the dimension of vector $\boldsymbol{x}$. Random variables (X) are non-bold, capital letters. Euclidean norms are denoted as $\|\cdot\|$, and for elements in a Hilbert space, the Hilbert norm is written as $\|\cdot\|_{\mathcal{H}}$. Expectations and covariances under q are denoted as $\mathbb{E}_q[\cdot]$ and $\mathrm{Cov}_q[\cdot]$, respectively. $\nabla f(\boldsymbol{x}) = [\partial_1 f(\boldsymbol{x}), \partial_2 f(\boldsymbol{x}), \cdots]^\top$. $\nabla f(\boldsymbol{x}_0)$ means the gradient of f evaluated at $\boldsymbol{x}_0$. Suppose $\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^n$, $\nabla \boldsymbol{f}(\boldsymbol{x})$ denotes the Jacobian of $\boldsymbol{f}$ and is a matrix of size $n \times m$.

2.1 TIME-VARYING EXPONENTIAL FAMILY

Definition 2.1. (See, e.g., Section 3.4, Casella and Berger [2024]) A distribution is a member of the **exponential family** with the sufficient statistic T if and only if its density function $q(\boldsymbol{x}; \boldsymbol{\theta})$ can be expressed as:

$$q(\boldsymbol{x}; \boldsymbol{\theta}) := \exp\left(\langle \boldsymbol{\theta}, T(\boldsymbol{x})\rangle - A(\boldsymbol{\theta})\right),$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ is the natural parameter, and $A(\boldsymbol{\theta})$ is the log-normalisation function, defined as:

$$A(\boldsymbol{\theta}) := \log \int \exp\left(\langle \boldsymbol{\theta}, T(\boldsymbol{x})\rangle\right) d\boldsymbol{x}.$$

This family includes many known distributions, such as Gaussian, Gamma and Exponential distributions. We can also choose T to be infinite-dimensional to enable more flexible modelling [Sriperumbudur et al., 2017, Arbel and Gretton, 2018].

Definition 2.2. The parametric manifold of the exponential family distributions with a sufficient statistic T is:

$$\mathcal{M}(T) := \left\{ \boldsymbol{\theta} \in \mathbb{R}^d \;\middle|\; \int q(\boldsymbol{x}; \boldsymbol{\theta})\, d\boldsymbol{x} = 1 \right\},$$

where $q(\boldsymbol{x}; \boldsymbol{\theta})$ depends on T as in Definition 2.1.

Although $\mathcal{M}(T)$ is a parametric manifold, with a slight abuse of the notation, we denote $q_{\boldsymbol{\theta}} \in \mathcal{M}(T)$ to indicate that $q_{\boldsymbol{\theta}}$ is a member of exponential family.

Time-varying exponential family distributions refers to exponential family distributions whose natural parameter is a continuous function of time, i.e., $q_{\boldsymbol{\theta}_t} = q(\boldsymbol{x}; \boldsymbol{\theta}(t))$. Moreover, the time derivative of $\log q_t$ is:

$$\partial_t \log q_{\boldsymbol{\theta}_t} = \langle \partial_t \boldsymbol{\theta}(t), T(\boldsymbol{x})\rangle - \partial_t A(\boldsymbol{\theta}(t)).$$

Equivalently, this can be expressed as the inner product of the natural parameter's rate of change and the centred sufficient statistic (Proposition 3.1 in Williams et al. [2025]):

$$\partial_t \log q_{\boldsymbol{\theta}_t} = \langle \partial_t \boldsymbol{\theta}(t), T(\boldsymbol{x}) - \mathbb{E}_{q_{\boldsymbol{\theta}_t}}[T(\boldsymbol{x})]\rangle. \quad (1)$$

By definition, the time-varying process $\boldsymbol{\theta}(t)$ is a curve on $\mathcal{M}(T)$ and $\partial_t \boldsymbol{\theta}(t)$ is the tangent vector of such curve.

2.2 NATURAL GRADIENT DESCENT

The natural gradient descent is an effective optimisation technique for parametric probabilistic models [Amari, 1998, Amari and Nagaoka, 2000] and has been widely applied in various fields [Bernacchia et al., 2018, Chen et al., 2024b].

Definition 2.3. The natural gradient of a loss function $\mathcal{L}(\boldsymbol{\theta})$ is the gradient of the loss function scaled by the inverse Fisher information matrix $\mathcal{F}$.

$$\nabla^N \mathcal{L}(\boldsymbol{\theta}) := \mathcal{F}^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}).$$

Suppose $q_{\boldsymbol{\theta}} \in \mathcal{M}(T)$, the Fisher information matrix is

$$\mathcal{F} = \nabla^2_{\boldsymbol{\theta}} \log q(\boldsymbol{x}; \boldsymbol{\theta}) = \mathrm{Cov}_{q_{\boldsymbol{\theta}}}[T(\boldsymbol{x})],$$

describing the curvature of $\mathcal{M}(T)$ neighbouring $\boldsymbol{\theta}$.

Although methods described in this paper are generic for different $\mathcal{L}$, we focus on the KL divergence in this paper.

Example 1. Suppose $\mathcal{L}$ is the KL divergence from p to q.

$$\mathcal{L} = \mathrm{KL}[p, q] = \mathbb{E}_p[\log p(\boldsymbol{x}) - \log q(\boldsymbol{x})].$$

Suppose q contains parameters $\boldsymbol{\theta}$ and $q_{\boldsymbol{\theta}} \in \mathcal{M}(T)$, then the natural gradient of $\mathcal{L}$ is given by

$$\begin{aligned} \nabla^N \mathrm{KL} &:= -\mathcal{F}^{-1} \mathbb{E}_p[\nabla_{\boldsymbol{\theta}} \log q(\boldsymbol{x}; \boldsymbol{\theta})] \\ &= -\mathrm{Cov}_{q_{\boldsymbol{\theta}}}[T(\boldsymbol{x})]^{-1} \left\{ \mathbb{E}_p[T(\boldsymbol{x})] - \mathbb{E}_{q_{\boldsymbol{\theta}}}[T(\boldsymbol{x})] \right\}. \end{aligned} \tag{2}$$

Except for certain specific choices of T, $\nabla^N \mathrm{KL}$ does not admit a closed-form expression, as neither $\mathbb{E}_{q_{\boldsymbol{\theta}}}[T(\boldsymbol{x})]$ nor $\mathrm{Cov}_{q_{\boldsymbol{\theta}}}[T(\boldsymbol{x})]$ can be expressed in closed form for a general T. If we could sample from $q_{\boldsymbol{\theta}}$, these expectations and covariances could be approximated using Monte Carlo methods. However, generating samples from a complex distribution $q_{\boldsymbol{\theta}}$ itself remains a challenging problem.

2.3 TIME-VARYING GENERATIVE MODEL

In recent years, there has been a growing trend of designing generative models as functions of time, in contrast to the classic generative models where the sample generating mechanism is independent of time. For example, the diffusion generative model [Song et al., 2021] can be interpreted as the solution to a Stochastic Differential Equation at time t, with initial samples drawn from a reference distribution. Similarly, the rectified flow generative model [Liu et al., 2023] can be viewed as the solution to an Ordinary Differential Equation at time t.

Definition 2.4. A **generative model** is a data-generating mechanism defined as $X = \boldsymbol{g}(Z; \boldsymbol{w})$, where Z is a random variable sampled from a latent distribution p_Z. $\boldsymbol{w} \in \mathcal{W}$ are parameters of the generative model $\boldsymbol{g}$. A **time-varying generative model** is defined as $X_t = \boldsymbol{g}(Z, t; \boldsymbol{w})$, i.e., a generative model whose output depends explicitly on t.

2.4 TIME SCORE MATCHING

In applications like time-series analysis, characterising the change of the data generative distribution is an essential task, and recently, a method measures the change of distribution over time was proposed [Choi et al., 2022].

Definition 2.5. Time score is the time derivative of the log density of a time-varying distribution. Given a time-varying distribution q_t, its time score is $s_t := \partial_t(\log q_t)$.

Given a parametric time score model $v(\boldsymbol{x}; t)$ and a time-varying sample $X_t \sim q_t$, the time score can be learned by the **Time Score Matching (TSM)** which minimises the objective: $\int \mathbb{E}\Big[\lambda(t) \left\| s_t(X_t) - v\big(X_t; t\big) \right\|^2\Big] \mathrm{d}t$, where $\lambda(t)$ is a weighting function.

A recent work shows, for the special case where $q_t(\mathbf{x}) \in \mathcal{M}(T)$, the time differential natural parameter $\partial_t \boldsymbol{\theta}$ can directly learned by TSM [Williams et al., 2025] and applied it to learning time-varying graphical models.

3 PROPOSED ALGORITHM: EVOLUTION PROJECTION

3.1 PROBLEM FORMULATION

Suppose we only have access to a target distribution p through samples $Y \sim p$ and a latent variable Z. Our goal is to find a time-varying generative model g_t that progressively approximates the target distribution as $t \to \infty$. Informally speaking, we seek a model such that $Y \overset{d}{\approx} g_\infty(Z)$.

We want to avoid training generative models using GANs or diffusion models, as these models require a large number of samples and are hard to train. Instead, given a loss function $\mathcal{L}(p, q_{\boldsymbol{\theta}_t})$ that measures the difference between p and a time-varying probabilistic model $q_{\boldsymbol{\theta}_t}$, we aim to guide the training of the generative model by minimising the loss $\mathcal{L}$ of the probabilistic model $q_{\boldsymbol{\theta}_t}$ over time.

The key idea of this paper is to align the evolution of the generative model with $\nabla^N \mathcal{L}(\boldsymbol{\theta})$ on the manifold $\mathcal{M}(T)$. Consequently, minimising the loss for $q_{\boldsymbol{\theta}}$ simultaneously drives the generative model toward matching p.

To achieve this, we first need to establish a direct correspondence between the instantaneous change in the generative model and the parametric update on $\mathcal{M}(T)$.

3.2 PROJECTING THE CHANGE

Denote the sample of a time-varying generative model g_t as $X_t \sim q_{g_t}$. We can measure the instantaneous change of the generative model via the its time score $s_t := \partial_t(\log q_{g_t})$.

At a fixed time t_0, we can "project" the time score s_t onto the manifold $\mathcal{M}(T)$ by minimising the squared difference between s_t and the time score of an exponential family distribution, $\partial_t(\log q_{\boldsymbol{\theta}_t}), q_{\boldsymbol{\theta}_t} \in \mathcal{M}(T)$,

$$\int \lambda_{t_0}(t)\mathbb{E}\left[(s_t(X_t) - \partial_t(\log q_{\boldsymbol{\theta}_t})(X_t))^2\right] \mathrm{d}t,$$
$$= \int \lambda_{t_0}(t)\mathbb{E}\left[(s_t(X_t) - \langle \partial_t \boldsymbol{\theta}(t), T(X_t) - \mathbb{E}[T(X_t')]\rangle)^2\right] \mathrm{d}t, \tag{3}$$

where $\lambda_{t_0} = \exp(-(t-t_0)^2/\sigma^2)$ and X_t' is an independent copy of X_t. σ is a hyperparameter fixed in advance. The second line is due to Equation (1). The integration is over the entire real line.

Introducing a linear-in-time model $\boldsymbol{\theta}(t;\boldsymbol{\delta}) = t\boldsymbol{\delta}$, the above objective becomes

$$J(\boldsymbol{\delta}) := \int \lambda_{t_0}(t)\mathbb{E}\left[(s_t(X_t) - \langle \boldsymbol{\delta}, T(X_t) - \mathbb{E}[T(X_t')]\rangle)^2\right] \mathrm{d}t, \tag{4}$$

which depends on the parameter $\boldsymbol{\delta}$. We use linear-in-time model to simplify the optimisation problem. More model choices of $\boldsymbol{\theta}(t)$ could be found in [Williams et al., 2025]. (4) can be viewed as a local regression at the fixed time point t_0: λ is a time smoothing kernel, Equation (4) finds the best score model that approximates $s_t(\boldsymbol{x})$ at t_0.

The minimiser to the above objective is a vector in $\mathbb{R}^d$ that best describes the instantaneous change in the generative model. Moreover, we can show that

Theorem 3.1. *Let $\boldsymbol{\delta}_{t_0} := \operatorname{argmin} J(\boldsymbol{\delta})$, then*

$$\boldsymbol{\delta}_{t_0} = -\left(\int \lambda_{t_0}(t)\mathrm{Cov}[T(X_t)]\mathrm{d}t\right)^{-1} \int \partial_t \lambda_{t_0}(t)\mathbb{E}\left[T(X_t)\right] \mathrm{d}t.$$

The proof can be found in Appendix A. Since $X_t = g(Z, t; \boldsymbol{w})$, we can express $\boldsymbol{\delta}_{t_0}$ as a function of $\boldsymbol{w}$ using the reparameterization trick [Kingma and Welling, 2014].

$$\boldsymbol{\delta}_{t_0}(\boldsymbol{w}) = -C^{-1}\int \partial_t \lambda_{t_0}(t)\mathbb{E}\left[T(g(Z,t;\boldsymbol{w}))\right] \mathrm{d}t,$$
$$C = \int \lambda_{t_0}(t)\mathrm{Cov}[T(g(Z,t;\boldsymbol{w}))]\mathrm{d}t. \tag{5}$$

This expression allows us to approximate $\mathbb{E}[\cdot]$ and $\mathrm{Cov}[\cdot]$ using samples of Z. We illustrate the projection process in Figure 1 where the downward dotted arrow represents the projection by TSM.

Remarks: Different time scores can map to the same projection $\boldsymbol{\delta}_{t_0}$, particularly if the sufficient statistic T is restrictive. However, if T is chosen to make the exponential family is expressive enough, we expect that such information collapse can be avoided. The rigorous proof of this claim is left as a future work. On a positive note, TSM objective (3) involves no normalising terms, consequently, we are free to use any expressive choice of T (e.g. neural networks, or RKHS kernels [Sriperumbudur et al., 2017]).

The hyperparameter σ in Equation (3) will introduce additional biases to the estimation, similar to how a non-zero kernel bandwidth in local regression introduces biases to the estimate. In experiments, we observe that reasonable σ choices (e.g., 0.1) work well. Moreover, in Section 4, we show how to obtain an unbiased estimator for a special type of time-varying generative models.

3.3 MATCHING TO NGD

From Equation (5), we can see the projection $\boldsymbol{\delta}_{t_0}(\boldsymbol{w})$ creates a connection between the parameter evolution of a generative model and those of a probabilistic model. The key idea of this paper is to align the evolution of the generative model with that of the probabilistic model. In particular, we match $\boldsymbol{\delta}_{t_0}(\boldsymbol{w})$ to the NGD update using the following objective

$$\boldsymbol{w}_{t_0} = \operatorname*{argmin}_{\boldsymbol{w}\in\mathcal{W}} \|\nabla^N \mathcal{L}(\boldsymbol{\theta}_{t_0}) - \boldsymbol{\delta}_{t_0}(\boldsymbol{w})\|^2. \tag{6}$$

In words, we find the generative model parameter $\boldsymbol{w}$ that results in the projected update $\boldsymbol{\delta}_{t_0}(\boldsymbol{w})$ closest to the natural gradient update of the loss function.

In the previous section, we have seen that $\boldsymbol{\delta}_{t_0}(\boldsymbol{w})$ can be approximated using samples Z. Assume that at the starting point, the generator $g(Z, 0; \boldsymbol{w}_0)$ produces an output distribution $q_{g_0} \in \mathcal{M}(T)$, and the trajectory q_{g_t} is precisely traced by the projected updates, we can expect that the samples generated from $g(Z, t; \boldsymbol{w}_t)$ will be close to the samples from $q_{\boldsymbol{\theta}_t}$. Thus, we approximate $\nabla^N \mathcal{L}(\boldsymbol{\theta})$ using samples from $g(Z, t; \boldsymbol{w}_t)$.

After solving (6), instead of taking a natural gradient step, we directly sample from $g(Z, t_0 + \epsilon; \boldsymbol{w}_{t_0})$ using a small $\epsilon > 0$. Since we have already aligned the projected change of the generative model with the natural gradient step, we can expect that the samples generated from $g(Z, t_0 + \epsilon; \boldsymbol{w}_{t_0})$ will be close to the samples from $q(\boldsymbol{x}; \boldsymbol{\theta}_{t_0} + \epsilon\mathcal{F}_{t_0}^{-1}\nabla\mathcal{L}(\boldsymbol{\theta}_{t_0}))$ by actually taking a natural gradient step with step size ϵ. We summarize the entire algorithm in Algorithm 1 and name it implicit NGD (iNGD).

In Figure 2, we show an example of the iNGD and compare it with the actual NGD on a Gaussian manifold $T(\boldsymbol{x}) = [\boldsymbol{x}, \boldsymbol{x}\boldsymbol{x}^\top], \boldsymbol{x} \in \mathbb{R}^2$. In this example, the generative model is an MLP with one hidden layer consisting of 67 neurons. We can see that the samples generated from iNGD accurately retrace the steps of the classic NGD, ultimately producing a set of samples (red dots) that resemble the target distribution (black dotted line).

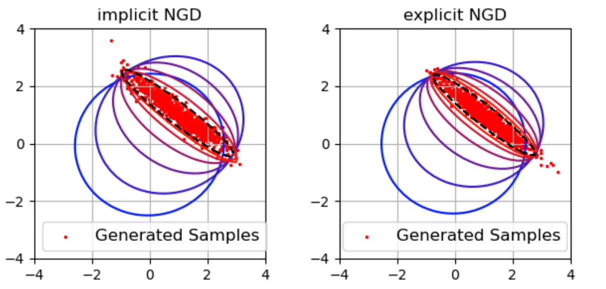

Figure 2: The evolution of parametric distributions under iNGD and NGD on the manifold of Gaussian distributions, i.e., $\mathcal{M}([\boldsymbol{x}, \boldsymbol{x}^\top \boldsymbol{x}])$, starting from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Each ellipse represents the 95% confidence interval of $q_{\boldsymbol{\theta}_t}$. As t increases, the ellipse turns from blue to red. Black dotted line marks the confidence interval the target distribution p. For iNGD, the ellipses are approximated by fitting a Gaussian model to the generated samples.

4 SPECIAL CASE: DRIFT MODEL

One popular class of generative models is the *drift-based generative model*. This model iteratively perturbs samples using a vector-valued function until convergence.

Definition 4.1. At a fixed time t_0, suppose we have samples X_{t_0}, a drift generative model generates samples at a new time point t via the following scheme:

$$\boldsymbol{g}(X_{t_0}, t; \boldsymbol{w}) = X_{t_0} + (t - t_0)\boldsymbol{h}(X_{t_0}; \boldsymbol{w}).$$

This model could be seen as a local linear model and the amount of update $\boldsymbol{h}(X_{t_0}; \boldsymbol{w})$ depends linearly on $t - t_0$. The input of the model is a sample at the time point t_0. To generate samples, we need to draw a batch of samples from an initial distribution q_{g_0} and then successively apply the drift generative model for a sequence of t until convergence. An Euler solver of flow-based generative model is an example of a drift generative model, in which case, $t - t_0$ is the step size of the Euler solver.

We can prove that when using a model introduced in Definition 4.1, Equation (5) has a limiting solution as $\sigma \to 0$, which eliminates the bias caused by the smoothing kernel λ.

Theorem 4.2. *The projection of the time score s_t of a drift generative model has a closed form expression at the limit of $\sigma \to 0$, i.e.,*

$$\begin{aligned}\lim_{\sigma \to 0} \boldsymbol{\delta}_{t_0}(\boldsymbol{w}) &= \mathrm{Cov}[T(X_{t_0})]^{-1}\mathbb{E}\left[\nabla T(X_{t_0})\boldsymbol{h}(X_{t_0}; \boldsymbol{w})\right] \\ &= \mathcal{F}_{t_0}^{-1}\mathbb{E}\left[\nabla T(X_{t_0})\boldsymbol{h}(X_{t_0}; \boldsymbol{w})\right].\end{aligned}$$

The proof can be found in Appendix B. Using this limiting solution, the objective of Equation (6) can be rewritten as:

$$\|\nabla \mathcal{L}(\boldsymbol{\theta}_{t_0}) - \mathbb{E}\left[\nabla^\top T(X_{t_0})\boldsymbol{h}(X_{t_0}; \boldsymbol{w})\right]\|^2_{\mathcal{F}_{t_0}^{-1}}. \quad (7)$$

Algorithm 1 Generative Model Training Guided by the Natural Gradient Descent

```
Require: Target samples $Y \sim p$, latent samples $Z \sim p_Z$,
    step size $\epsilon$, number of iterations $n$
1: $t[0] \leftarrow 0$, initialize $\boldsymbol{w}$.
2: for $i \leftarrow 1$ to $n$ do
3:     Sample $X_{t[i]} \leftarrow g(Z, t[i]; \boldsymbol{w})$
4:     Approximate $\nabla^N \mathcal{L}(\boldsymbol{\theta}_{t[i]})$ with $X_{t[i]}$ and $Y$ using
       Monte Carlo.
5:     Approximate $\boldsymbol{\delta}_{t[i]}(\boldsymbol{w})$ with $X_{t[i]}$ using Monte Carlo.
6:     $\boldsymbol{w} \leftarrow \operatorname{argmin}_{\boldsymbol{w}} \|\nabla^N \mathcal{L}(\boldsymbol{\theta}_{t[i]}) - \boldsymbol{\delta}_{t[i]}(\boldsymbol{w})\|^2$
7:     $t[i+1] \leftarrow t[i] + \epsilon$
8: end for
9: Return: Samples from $g(Z, t[n]; \boldsymbol{w})$
```

Note that the gradient $\nabla \mathcal{L}(\boldsymbol{\theta}_{t_0})$ is an Euclidean gradient. In our experiments, this objective function is more stable and computationally efficient than (6), since it does not require back-propagating through a matrix inversion.

4.1 KERNEL NGD

Now let us consider an example of the drift model where the drift function $\boldsymbol{h}$ is defined as the gradient of an Reproducing Kernel Hilbert Space (RKHS) function.

Example 2 (RKHS Drift Model)**.** A kernel drift function is defined as $\boldsymbol{h}(\boldsymbol{x}; w) := \langle w, \nabla_{\boldsymbol{x}} k(\cdot, \boldsymbol{x})\rangle_{\mathcal{H}}, w \in \mathcal{H}$, where $\mathcal{H}$ is an RKHS with a kernel function k.

To align this generative model with the NGD, we introduce a regularized version of Equation (7)

$$\|\nabla \mathcal{L}(\boldsymbol{\theta}_{t_0}) - \mathbb{E}[\nabla T(X_{t_0})\boldsymbol{h}_w(X_{t_0})]\|^2_{\mathcal{F}_{t_0}^{-1}} + \lambda \|w\|^2_{\mathcal{H}}. \quad (8)$$

Theorem 4.3. *The optimal drift function that minimises Equation* (8) *can be found as*

$$\boldsymbol{h}_{w^*}(\boldsymbol{x}) = \mathbb{E}\left[\nabla T(X_{t_0})\nabla\nabla k(X_{t_0}, \boldsymbol{x})\right]^\top \boldsymbol{\Gamma}^{-1}\nabla \mathcal{L}(\boldsymbol{\theta}_{t_0}) \quad (9)$$

$$\boldsymbol{\Gamma} = \lambda \mathcal{F}_{t_0} + \mathbb{E}\left[\nabla T(X_{t_0})\nabla\nabla k(X_{t_0}, X'_{t_0})\nabla^\top T(X'_{t_0})\right]$$

where X'_{t_0} is an independent copy of X_{t_0} and $\nabla\nabla k(\boldsymbol{x}, \boldsymbol{y})$ is the short hand for $\nabla_{\boldsymbol{x}}\nabla_{\boldsymbol{y}} k(\boldsymbol{x}, \boldsymbol{y})$.

The proof can be found in Appendix C. This result enables the *direct calculation* of particle updates without fitting a generative model first. This inspires us to build a *particle evolution strategy* guided by NGD: First, we sample X_0 from an initial distribution q_0, then we iteratively update each sample X_{t_0} using the formula given by Theorem 4.3 until they converge. This algorithm is summarized in Algorithm 2 and we name it Kernel implicit NGD (KiNG).

Algorithm 2 RKHS/Neural Tangent Kernel Implicit Natural Gradient Descent

```
Require: Target samples $Y \sim p$, initial particles $X_0 \sim q_0$,
    step size $\epsilon$, number of iterations $n$
1: $t[0] \leftarrow 0$
2: for $i \leftarrow 1$ to $n$ do
3:     Approximate $\boldsymbol{h}$ with $Y$ and $X_{t[i]}$ using (9) or (11).
4:     $t[i+1] = t[i] + \epsilon$
5:     $X_{t[i+1]} = X_{t[i]} + \epsilon \boldsymbol{h}(X_{t[i]})$
6: end for
7: Return: $X_{t[n]} \sim q_{t_{[n]}}$.
```

In Figure 3, we plot the trajectories of KiNG with different one-dimensional initial and target distributions. We choose $\mathcal{M}(T)$ to be a Gaussian manifold, i.e., $T(x) = [x, x^2]^\top$. In the right plot, the particles do not converge to the bi-modal target distribution, since the movements of our particles are "guided" by the Gaussian manifold, and the best approximation is a Gaussian with a larger standard deviation. This example shows that the particles are indeed guided by the Gaussian manifold throughout the generative process. This phenomenon could be beneficial, if the target is to find the best approximation within a given family or we already have an informative probabilistic model (see Section 5.2).

This behaviour can be changed by replacing the Gaussian manifold with a more expressive manifold. In the left plot of Figure 4, we run similar experiments by letting T be the Radial Basis Functions (RBFs) and we can see that indeed the particles bifurcate and converge to both modes.

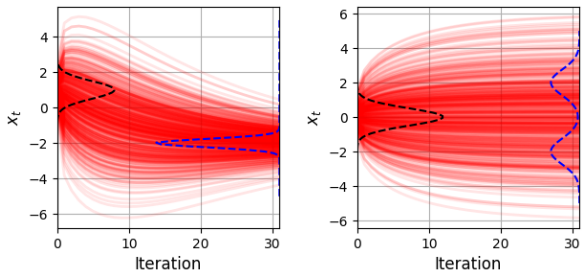

Figure 3: The evolution of particles X_t under KiNG on the manifold of Gaussian distributions. Each red line is a trajectory of a particle in the space-time. The initial distribution q_0 is plotted on the left as black dotted lines, while the target distribution p is plotted on the right as blue dotted lines.

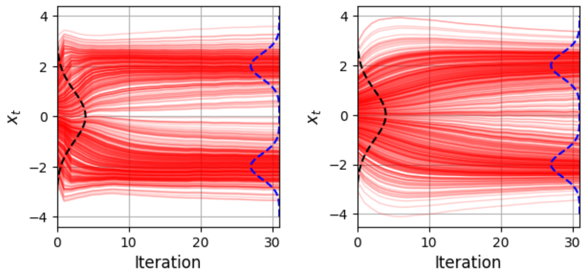

Figure 4: Particle trajectories when using more expressive manifold. $T(\boldsymbol{x}) := [k(x_1, b_1), k(x_1, b_2), \cdots]^\top$, where k is an RBF and b_i are kernel basis randomly chosen from samples of X_{t_0}. Left: KiNG, Right, ntKiNG.

4.2 NEURAL TANGENT KERNEL NGD

KiNG can be generalized to other types of kernels.

Example 3 (Neural Tangent Drift Model)**.** Given a neural network $\boldsymbol{\phi} : \mathbb{R}^{\dim(\boldsymbol{x})} \to \mathbb{R}^{\dim(\boldsymbol{x})}$, a neural tangent drift function is defined as

$$\boldsymbol{h}(\boldsymbol{x}; \boldsymbol{w}) := \nabla_{\boldsymbol{\beta}} \boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{\beta}_0) \boldsymbol{w},$$

where $\boldsymbol{\beta}_0$ are initial weights, $\nabla_{\boldsymbol{\beta}} \boldsymbol{\phi}(\boldsymbol{x}; \boldsymbol{\beta}_0)$ is neural tangent.

Similar to Equation (8), we can solve the following regularized objective to find the update of the particles

$$\|\nabla \mathcal{L}(\boldsymbol{\theta}_{t_0}) - \mathbb{E}[\nabla T(X_{t_0}) \boldsymbol{h}_{\boldsymbol{w}}(X_{t_0})]\|^2_{\mathcal{F}^{-1}_{t_0}} + \lambda \|\boldsymbol{w}\|^2. \quad (10)$$

The optimal drift that minimises Equation (10) can be expressed using the *neural tangent kernel* [Jacot et al., 2018]:

$$\boldsymbol{h}_{\boldsymbol{w}^*}(\boldsymbol{x}) = \mathbb{E}\left[\nabla T(X_{t_0}) \boldsymbol{K}_{\mathrm{NTK}}(X_{t_0}, \boldsymbol{x})\right]^\top \boldsymbol{\Gamma}^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}_{t_0}) \quad (11)$$

$$\boldsymbol{\Gamma} := \lambda \mathcal{F}_{t_0} + \mathbb{E}\left[\nabla T(X_{t_0}) \boldsymbol{K}_{\mathrm{NTK}}(X_{t_0}, X'_{t_0}) \nabla^\top T(X'_{t_0})\right],$$

where $\boldsymbol{K}_{\mathrm{NTK}}$ is the matrix-valued *neural tangent kernel*, defined as $\boldsymbol{K}_{\mathrm{NTK}}(\boldsymbol{x}, \boldsymbol{y}) := \nabla_{\boldsymbol{\beta}} \boldsymbol{\phi}(\boldsymbol{x}) \nabla^\top_{\boldsymbol{\beta}} \boldsymbol{\phi}(\boldsymbol{y})$. This result can be proven using the same technique described in Appendix C. In this paper, we use empirical and a finite-width NTK for simplicity, but NTKs that are infinitely wide can be efficiently computed using off-the-shelf package such as `neural-tangents` [Novak et al., 2020] for a variety of neural network architectures.

The right plot of Figure 4 shows the particle trajectory of a neural tangent KiNG with T as RBF basis. We name this variant of KiNG as ntKiNG.

Remark: Equation (11) requires computing a matrix-valued kernel, which may be computationally demanding if the dimensionality of X_{t_0} is high. However, in experiments, we observe that the formulation works well with a diagonalized scalar kernel, i.e.,

$$[\boldsymbol{K}(X_{t_0}, X'_{t_0})]_{l,m \in \{1 \ldots \dim(\boldsymbol{x})\}} := \begin{cases} k(X_{t_0}, X'_{t_0}), & l = m \\ 0, & l \neq m, \end{cases}$$

where k is any scalar kernel function.

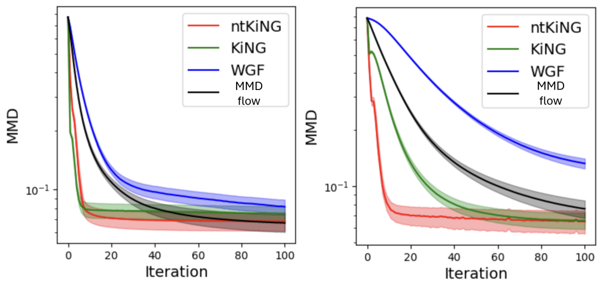

Figure 5: $\mathrm{MMD}[Y, X_t]$ over iterations, the lower the better. Left: 5 dimensions, Right: 20 dimensions. The error bar indicates the standard error.

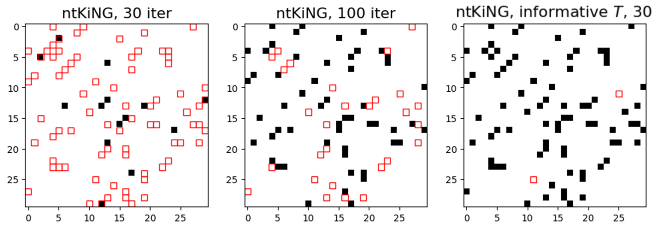

Figure 6: Sparsity pattern of $\mathbf{\Theta}$ recovered by graphical lasso, using samples trained by ntKiNG with different sufficient statistics and number of iterations (red boxes indicate missing edges, the less boxes the better)

5 EXPERIMENTS

In this section, we further validate our method on toy data and real-world data respectively. The summarized details of experiment setups can be found in Appendix F. In all experiments, $\mathcal{L}(p, q_{\boldsymbol{\theta}}) = \mathrm{KL}[p, q_{\boldsymbol{\theta}}]$.

5.1 KINGS VS. REVERSE KL WASSERSTEIN GRADIENT FLOW AND MMD FLOW

In this experiment, we compare KiNG, ntKiNG, with reverse KL Wasserstein Gradient Flow (WGF) [Gao et al., 2019, Liu et al., 2024] and Maximum Mean Discrepancy flow (MMD flow) [Hagemann et al., 2024] on small datasets with different dimensions. See Appendix E for more explanations of these methods.

Since they all minimise different divergences, we measure their performance using MMD [Gretton et al., 2012] between a fresh batch of target samples and X_t.

Let $p = 0.5\mathcal{N}(-2, \boldsymbol{I}) + 0.5\mathcal{N}(2, \boldsymbol{I})$. We draw 100 samples from p as Y, 100 samples from $\mathcal{N}(0, \boldsymbol{I})$ as X_0, and run KiNG, WGF, MMD flow to evolve particles X_t. We plot $\mathrm{MMD}(Y, X_t)$ over iterations. For all methods, we set learning rate to be 1, which is the largest learning rates without causing numerical instability. It can be seen that when dimension is small (5), all methods work relatively well and ntKiNG and KiNG can reduce MMD faster, but when we increase the dimension to 20 the performance gap widens. However, ntKiNG and KiNG still have a commanding lead.

5.2 GRAPHICAL MODEL RECOVERY WITH INFORMATIVE SUFFICIENT STATISTICS

In this experiment, we showcase how much improvement we can get when using informative sufficient statistics. Exponential family are commonly used to encode graphical models. For example, a Gaussian graphical model is a Gaussian density $p = \mathcal{N}(\mathbf{0}, \mathbf{\Theta}^{-1})$, where the sparsity pattern of $\mathbf{\Theta}$ encodes an undirected graph, describing the interactions of the random variables. One can imagine that if the generated samples approximate p well, we should recover the correct graphical model from these samples. In this experiment, we let p be a 30-dimensional Gaussian graphical model, and draw 200 samples $Y \sim p$, 200 samples $X_0 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and move the X_0 toward p using ntKiNG algorithm. Finally we apply the graphical lasso [Friedman et al., 2007] to estimate graphical models displayed in the left and middle plots of Figure 6. Here $T(\boldsymbol{x}) := [\text{RBF basis}]$.

Since our methods use a probabilistic model to guide its training process, one may wonder if knowing the graphical model structure would improve the performance of the algorithm. To test this, we design a new sufficient statistic $T(\boldsymbol{x}) := [\text{RBF basis}, \forall (i,j) \in \{(i,j) | \Theta_{i,j} \neq 0\}, x_i x_j]$, i.e., we added pairwise potential functions that corresponds to pairwise factors in this graphical model. We ran the ntKiNG again with this new, better informed sufficient statistic for 30 runs. The graphical lasso estimate is shown in the right plot of Figure 6. It can be seen that, when using the informed sufficient statistics, ntKiNG could recover the almost-correct graphical structure in only 30 iterations, while it takes the regular ntKiNG much longer. This suggests, our methods can indeed use a pre-existing probabilistic model to accelerate its generative model training process.

5.3 COVARIATE SHIFT BY DIST. MATCHING

In domain adaptation tasks, samples are drawn from the source distribution p_{XY} and the target distribution q_{XY} where X, Y are covariates and label respectively. The problem is that a classifier trained on source distribution samples may not work on target distribution samples. Covariate shift [Sugiyama et al., 2008, Quiñonero-Candela et al., 2009] refers to a special case where $p_X \neq q_X$ but $p_{Y|X} = q_{Y|X}$. We adopt a "marginal transport" assumption [Courty et al., 2016] that $X \sim q_X$ are generated as $X = \psi(X')$ where $X' \sim p_X$. It means, samples are generated from the source distribution and then "transported" to the target domain. For example, images in the source domain contains photos of objects, while in the target domain, photos contain the same objects but are filtered to reflect certain styles.

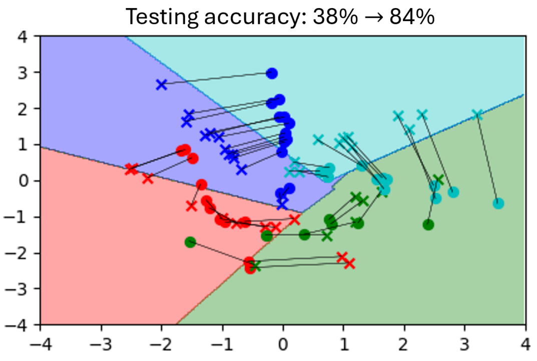

Figure 7: The inverse mapping ψ^{-1} found by ntKiNG. The classification boundary of a source classifier trained on p is depicted in four distinct colors. Target domain samples are marked with $\bullet$, and KiNG transforms these samples to new positions indicated by $\times$. Notably, many samples—especially those in the cyan and blue classes—are transported from the incorrect side of the classification boundary to the correct side and the test accuracy increases as a result.

In the covariate shift setting, we observe joint samples from the source $p_{X,Y}$, but only have target domain covariates from q_X. The goal is to find ψ^{-1}. We find the reverse process by minimising $\mathrm{KL}[p_X, q_t]$ using Algorithm 2, where $X_0 \sim q_0$ are set to be the target domain covariates.

We demonstrate the effectiveness of this algorithm in Figure 7, where the transfer ψ is a clockwise rotation on samples by 45 degrees. An inverse ψ^{-1} is a counter-clockwise rotation and has been correctly identified by ntKiNG.

We further test our algorithm on the Office+Caltech classification dataset [Gong et al., 2012] which is an object recognition dataset with photos collected from four different places: `amazon`, `dslr`, `webcam`, `caltech`. The task is to train a source classifier using one of the places, and test it on samples from another place. We reduce the dimension of the dataset to 50 using PCA and test the performance of ntKiNG against two other particle-based transport methods WGF and MMD flow that also matches q_t with p_X. The performance is measured by the percentage gains compared with directly applying the source classifier to the target samples. The results in Table 1 show that our method achieves the most accuracy gains comparing to WGF and MMD flow. In 10 out of 12 domain adaptations settings, ntKiNG improves the testing accuracy.

5.4 SAMPLE DENOISING WITH PRETRAINED ENERGY-BASED MODEL (EBM)

To further investigate the performance of iNGD on higher-dimensional datasets and more sophisticated sufficient statistics, we conduct two denoising experiments.

We begin by drawing samples $Y \sim q$ and train a deep energy-based model using denoising score matching [Song and Kingma, 2021]. We define the sufficient statistic $T(\boldsymbol{x})$ as the output of the penultimate layer of the EBM, ensuring that $T(\boldsymbol{x})$ captures information about the distribution q. The structure of the EBM can be found in Appendix F.4.

Src. → Tar.	base (%)	ntKiNG	WGF	MMD flow
amz → dslr	69.50	**+13.75**	-0.50	+2.75
amz → web	72.75	**+8.25**	-2.00	+0.75
amz → cal	91.50	-0.75	-5.75	**+0.00**
dslr → amz	86.00	**+1.27**	-6.37	**+1.27**
dslr → web	98.09	**+0.00**	-0.64	**+0.00**
dslr → cal	84.08	**+5.73**	-7.64	+0.64
web → amz	77.97	**+3.39**	-2.71	+1.02
web → dslr	91.19	**+3.39**	-4.07	+1.36
web → cal	76.61	**+3.39**	-3.73	+0.34
cal → amz	82.00	-2.50	-4.50	**-0.25**
cal → dslr	58.25	**+16.50**	+7.00	+3.25
cal → web	65.50	**+10.25**	+1.00	+2.00
Average	**79.45**	**+5.22**	**-2.49**	**+1.09**

Table 1: Comparison of testing accuracy differences (in %) relative to the base classifier without any transfer learning.

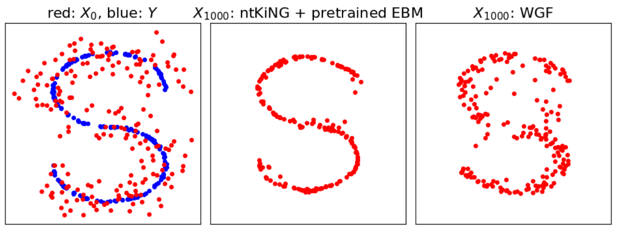

Figure 8: Denoised samples obtained by ntKiNG and WGF. ntKiNG uses pretrained energy-based model as T.

We then construct noisy samples by adding Gaussian noises to Y as X_0 and run ntKiNG and WGF using Y and X_0 as the target and initial samples with 1000 iterations.

We first test the procedure on an S-curve dataset and the result is shown in Figure 8. It can be seen that ntKiNG more effectively recovers the S-shape compared to WGF.

We then perform the experiment on the MNIST dataset (downsampled to 18 by 18), using the same model architecture and estimator to construct T. As we can see from Figure 9, both methods act as decent denoisers, but ntKiNG produces cleaner reconstructions than WGF.

6 RELATED WORKS

Our methods bridge the gap between generative model training/sampling and parametric model optimization. Both domains are extensively studied in the literature.

There has been a trend toward using optimization techniques

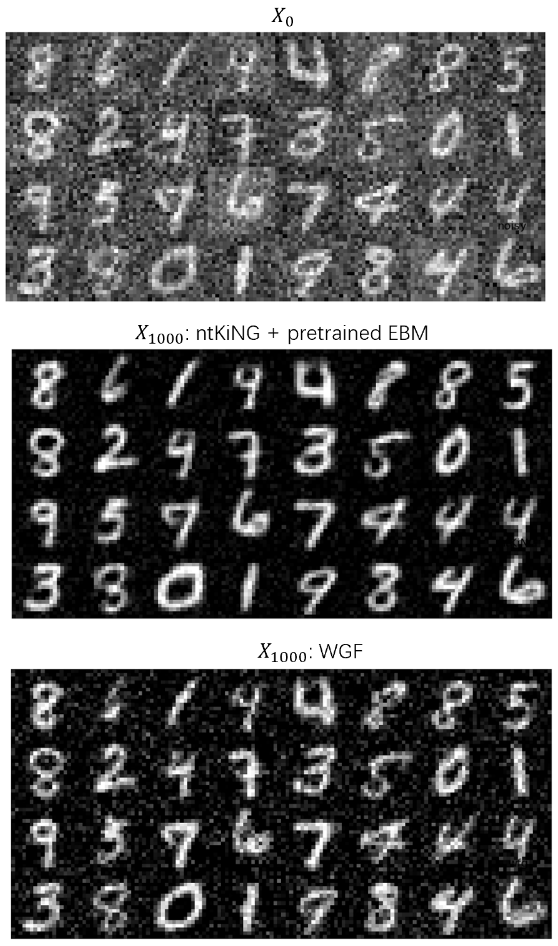

Figure 9: Denoised MNIST images

to sample from unknown distributions. These methods first draw samples from an initial distribution and then move them according to a time-dependent velocity field [Liu, 2017, Chewi et al., 2020, Maurais and Marzouk, 2024]. A typical family of methods is the Wasserstein Gradient Flow [Ambrosio et al., 2008], which has found various applications outside of sampling, such as generative modelling [Gao et al., 2019, Choi et al., 2024] and missing data imputation [Chen et al., 2024a]. Our method falls within this family of algorithms, and we compared two of its variants in our experiments. To the best of our knowledge, none of the existing approaches could leverage a pre-existing probabilistic model to guide the flow of particles. Our framework is also more general: Algorithm 1 works for non-particle based generative models as well (as we see in Section 5.2).

Another trend in generative modelling is “flow matching”, where one aligns the drift function with a pre-constructed flow [Lipman et al., 2023, Liu et al., 2023]. In a similar spirit, our method also aligns the instantaneous change of the generative distribution with a prescribed dynamics (NGD). However, instead of directly matching the velocity field in the sample space, we match the projections of these changes in the parametric space. This approach avoids building arbitrary "bridges" between the reference and target distributions in sample space and instead leverages an effective parametric optimization algorithm to guide the training of the generative model.

In recent years, there has also been efforts to accelerate and approximate NGD using kernel methods, for example, [Arbel et al., 2020, Li et al., 2019b] propose to approximate the natural gradient by optimizing a dual formulation. However, both methods consider optimizing a probabilistic model, rather than a generative model as described in this paper. Performing NGD requires inverting a large matrix. Many research on NGD focuses on approximating the inverse the Fisher Information Matrix [Martens and Grosse, 2015, Grosse and Martens, 2016, George et al., 2018]. Our particle update, e.g., Theorem 4.3 also requires us inverting a matrix with the dimension of the sufficient statistic. It would be an interesting future work to see if these techniques could be adapted to our approach.

7 LIMITATIONS AND FUTURE WORKS

The effectiveness of the guidance heavily relies on the choice of the exponential family manifold, which is determined by the sufficient statistics T. If the guidance is weak, the particles may fail to converge to the target distribution, as demonstrated in Figure 3. In this paper, we show that sophisticated sufficient statistics—such as RBF features or a pretrained EBM—can achieve promising results. Developing theories to better understand the choices of sufficient statistics is an important future work.

In this paper, we only focus on drift-based generative model for its simplicity. An interesting future work is studying the applicability of our methods to other types of generative models (e.g., GAN or diffusion model).

The primary computational bottleneck of our method lies in the inversion of the $\dim(T) \times \dim(T)$ matrix $\mathbf{\Gamma}$ in (11), which become computationally prohibitive if $\dim(T)$ is high. For some choices of T (e.g., RBF), reducing $\dim(T)$ also reduces its expressiveness. Thus, extending iNGD to a high-dimensional T is an urgent future work. Both our method and MMD flow requires computing kernel matrix, which has an n^2 computational complexity. However, there is no matrix inversion involved in MMD flow.

More generally, despite the promising results, the benefits of guiding the generative model in a parametric space remain to be clarified. Developing theories and applications that compare our method with established generative approaches—such as diffusion models—represents an interesting direction for future work.

ACKNOWLEDGEMENTS

We thank four anonymous reviewers and the area chair for their insightful comments. We thank Dr. Sam Power, Dr. Katerina Karoni and Bristol Machine Learning Reading Group for helpful discussions.

References

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191. American Mathematical Soc., 2000.

Luigi Ambrosio, Nicola Gigli, and Giuseppe Savaré. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2008.

M Arbel, A Gretton, W Li, and G Montufar. Kernelized wasserstein natural gradient. In *International Conference on Learning Representations*, 2020.

Michael Arbel and Arthur Gretton. Kernel conditional exponential family. In *International Conference on Artificial Intelligence and Statistics*, 2018.

Michael Arbel, Anna Korba, Adil Salim, and Arthur Gretton. Maximum mean discrepancy gradient flow. *Advances in Neural Information Processing Systems*, 2019.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, 2017.

Alberto Bernacchia, Máté Lengyel, and Guillaume Hennequin. Exact natural gradient in deep linear networks and its application to the nonlinear case. *Advances in Neural Information Processing Systems*, 2018.

Valentin De Bortoli, James Thornton, Jeremy Heng, and Arnaud Doucet. Diffusion schrödinger bridge with applications to score-based generative modeling. In *Advances in Neural Information Processing Systems*, 2021.

George Casella and Roger Berger. *Statistical inference, 2nd Edition*. CRC press, 2024.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

Zhichao Chen, Haoxuan Li, Fangyikang Wang, Odin Zhang, Hu Xu, Xiaoyu Jiang, Zhihuan Song, and Hao Wang. Rethinking the diffusion models for missing data imputation: A gradient flow perspective. In *Advances in Neural Information Processing Systems*, 2024a.

Zhuo Chen, Jacob McCarran, Esteban Vizcaino, Marin Soljacic, and Di Luo. Teng: Time-evolving natural gradient for solving pdes with deep neural nets toward machine precision. In *International Conference on Machine Learning*, 2024b.

Xiang Cheng, Niladri S Chatterji, Peter L Bartlett, and Michael I Jordan. Underdamped langevin mcmc: A non-asymptotic analysis. In *Conference on learning theory*, 2018.

Sinho Chewi. *Log-Concave Sampling*. 2024.

Sinho Chewi, Thibaut Le Gouic, Chen Lu, Tyler Maunu, and Philippe Rigollet. Svgd as a kernelized wasserstein gradient flow of the chi-squared divergence. In *Advances in Neural Information Processing Systems*, 2020.

Jaemoo Choi, Jaewoong Choi, and Myungjoo Kang. Scalable wasserstein gradient flow for generative modeling through unbalanced optimal transport. In *International Conference on Machine Learning*, 2024.

Kristy Choi, Chenlin Meng, Yang Song, and Stefano Ermon. Density ratio estimation via infinitesimal classification. In *International Conference on Artificial Intelligence and Statistics*, 2022.

Nicolas Courty, Rémi Flamary, Devis Tuia, and Alain Rakotomamonjy. Optimal transport for domain adaptation. *IEEE transactions on pattern analysis and machine intelligence*, 39(9):1853–1865, 2016.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Sparse inverse covariance estimation with the graphical lasso. *Biostatistics*, 9(3):432–441, 2007.

Yuan Gao, Yuling Jiao, Yang Wang, Yao Wang, Can Yang, and Shunkang Zhang. Deep generative learning via variational gradient flow. In *International Conference on Machine Learning*, 2019.

Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a kronecker factored eigenbasis. *Advances in Neural Information Processing Systems*, 2018.

Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *Conference on Computer Vision and Pattern Recognition*, 2012.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems*, 2014.

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012.

Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, 2016.

Shuyang Gu, Dong Chen, Jianmin Bao, Fang Wen, Bo Zhang, Dongdong Chen, Lu Yuan, and Baining Guo. Vector quantized diffusion model for text-to-image synthesis. In *Conference on Computer Vision and Pattern Recognition*, 2022.

Paul Hagemann, Johannes Hertrich, Fabian Altekrüger, Robert Beinert, Jannis Chemseddine, and Gabriele Steidl. Posterior sampling based on gradient flows of the MMD with negative distance kernel. In *International Conference on Learning Representations*, 2024.

Jiajun He, Wenlin Chen, Mingtian Zhang, David Barber, and José Miguel Hernández-Lobato. Training neural samplers with reverse diffusive kl divergence. *arXiv preprint arXiv:2410.12456*, 2025.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 2020.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems*, 2018.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, and Ming Liu. Neural speech synthesis with transformer network. In *AAAI Conference on Artificial Intelligence*, 2019a.

Wuchen Li, Alex Tong Lin, and Guido Montúfar. Affine natural proximal learning. In *International Conference on Geometric Science of Information*, 2019b.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Qiang Liu. Stein variational gradient descent as gradient flow. In *Advances in Neural Information Processing Systems*, 2017.

Song Liu, Jiahao Yu, Jack Simons, Mingxuan Yi, and Mark Beaumont. Minimizing f-divergences by interpolating velocity fields. In *International Conference on Machine Learning*, 2024.

Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning*, 2015.

Aimee Maurais and Youssef Marzouk. Sampling in unit time with kernel fisher-rao flow. In *International Conference on Machine Learning*, 2024.

Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *International Conference on Learning Representations*, 2020.

Joaquin Quiñonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. Mit Press, 2009.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, 2015.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Conference on Computer Vision and Pattern Recognition*, 2022.

Yang Song and Diederik P Kingma. How to train your energy-based models. *arXiv preprint arXiv:2101.03288*, 2021.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

Bharath Sriperumbudur, Kenji Fukumizu, Arthur Gretton, Aapo Hyvärinen, and Revant Kumar. Density estimation in infinite dimensional exponential families. *Journal of Machine Learning Research*, 18(57):1–59, 2017.

Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in Neural Information Processing Systems*, 2008.

Xu Tan, Jiawei Chen, Haohe Liu, Jian Cong, Chen Zhang, Yanqing Liu, Xi Wang, Yichong Leng, Yuanhao Yi, Lei He, et al. Naturalspeech: End-to-end text-to-speech synthesis with human-level quality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2): 1–305, 2008.

Zhendong Wang, Huangjie Zheng, Pengcheng He, Weizhu Chen, and Mingyuan Zhou. Diffusion-gan: Training gans with diffusion. In *International Conference on Learning Representations*, 2023.

Max Welling. Kernel ridge regression. `https://web2.qatar.cmu.edu/~gdicaro/10315-Fall19/additional/welling-notes-on-kernel-ridge.pdf`, 2019. Accessed: 2025-02-10.

Andre Wibisono. Sampling as optimization in the space of measures: The langevin dynamics as a composite optimization problem. In *Conference on Learning Theory*, 2018.

Daniel J Williams, Leyang Wang, Qizhen Ying, Song Liu, and Mladen Kolar. High-dimensional differential parameter inference in exponential family using time score matching. *International conference on artificial intelligence and statistics*, 2025.

A PROOF OF THEOREM 3.1

Theorem A.1. *(Theorem 4.1 in [Williams et al., 2025]) Equation* (4) *can be rewritten as the following form*

$$\mathcal{L}(\boldsymbol{\delta}) = \int_0^1 \lambda_{t_0}(t)\mathbb{E}\left[\langle \boldsymbol{\delta}, T(X_t) - \mathbb{E}[T(X_t')]\rangle^2\right] \mathrm{d}t + 2\int_0^1 \partial_t \lambda_{t_0}(t)\mathbb{E}\left[\langle \boldsymbol{\delta}, T(X_t)\rangle\right] \mathrm{d}t + \textit{const.} \tag{12}$$

Notice that Equation (12) is a quadratic minimisation problem. Let $B_t = T(X_t) - \mathbb{E}[T(X_t')]$. Firstly, we find the derivative of the quadratic term. For each t, since $\nabla_{\boldsymbol{\delta}} \left(\langle \boldsymbol{\delta}, B_t\rangle^2\right) = 2\,\langle \boldsymbol{\delta}, B_t\rangle\, B_t$, it follows that

$$\nabla_{\boldsymbol{\delta}} \int_t \lambda_{t_0}(t)\mathbb{E}\big[\,\langle \boldsymbol{\delta}, B_t\rangle^2\big]\,\mathrm{d}t \;=\; 2\,\int_t \lambda_{t_0}(t)\,\mathbb{E}\big[\langle \boldsymbol{\delta}, B_t\rangle\, B_t\big]\,\mathrm{d}t.$$

Since $\mathbb{E}[B_t] = 0$, one has $\mathbb{E}\big[\langle \boldsymbol{\delta}, B_t\rangle\, B_t\big] \;=\; \mathrm{Cov}[B_t]\,\boldsymbol{\delta} \;=\; \mathrm{Cov}[T(X_t)]\,\boldsymbol{\delta}$.
Hence this part becomes

$$2\int_t \lambda_{t_0}(t)\,\mathrm{Cov}[T(X_t)]\,\boldsymbol{\delta}\,\mathrm{d}t.$$

Let us differentiate the linear-in-$\boldsymbol{\delta}$ term. The term $\int_t 2\,\partial_t\lambda_{t_0}(t)\,\mathbb{E}\big[\langle \boldsymbol{\delta},\, T(X_t)\rangle\big]\mathrm{d}t$ is linear in $\boldsymbol{\delta}$. Its gradient w.r.t. $\boldsymbol{\delta}$ is simply

$$2\int_t \partial_t\lambda_{t_0}(t)\,\mathbb{E}[T(X_t)]\,\mathrm{d}t.$$

Putting these together, the gradient of $\mathcal{L}(\boldsymbol{\delta})$ is

$$\nabla_{\boldsymbol{\delta}}\,\mathcal{L}(\boldsymbol{\delta}) \;=\; 2\,\int_t \lambda_{t_0}(t)\,\mathrm{Cov}[T(X_t)]\,\boldsymbol{\delta}\,\mathrm{d}t \;+\; 2\,\int_t \partial_t\lambda_{t_0}(t)\,\mathbb{E}[T(X_t)]\,\mathrm{d}t.$$

To find the minimiser, set this gradient to zero:

$$0 \;=\; 2\,\int_t \lambda_{t_0}(t)\,\mathrm{Cov}[T(X_t)]\,\boldsymbol{\delta}\,\mathrm{d}t \;+\; 2\,\int_t \partial_t\lambda_{t_0}(t)\,\mathbb{E}[T(X_t)]\,\mathrm{d}t.$$

which can be rewritten as

$$\Big(\int_t \lambda_{t_0}(t)\,\mathrm{Cov}[T(X_t)]\,\mathrm{d}t\Big)\,\boldsymbol{\delta} \;=\; -\,\int_t \partial_t\lambda_{t_0}(t)\,\mathbb{E}[T(X_t)]\,\mathrm{d}t.$$

If the matrix $\int_0^1 \lambda_{t_0}(t)\,\mathrm{Cov}[T(X_t)]\,\mathrm{d}t$ is invertible—which is precisely the non-degeneracy (invertibility) of the Fisher information—then we can solve uniquely for $\boldsymbol{\delta}$:

$$\boldsymbol{\delta}_{t_0} \;=\; \Big(\int_t \lambda_{t_0}(t)\,\mathrm{Cov}[T(X_t)]\,\mathrm{d}t\Big)^{-1}\Big(-\int_t \partial_t\lambda_{t_0}(t)\,\mathbb{E}[T(X_t)]\,\mathrm{d}t\Big).$$

B PROOF OF THEOREM 4.2

Recall:

$$\boldsymbol{\delta}_{t_0}(\boldsymbol{w}) = -C^{-1}\int_{-\infty}^{\infty} \partial_t\lambda(t,t_0)\mathbb{E}\left[T(g(Z,t;\boldsymbol{w}))\right]\mathrm{d}t, \quad C = \int_{-\infty}^{\infty} \lambda(t,t_0)\mathrm{Cov}[T(g(Z,t;\boldsymbol{w}))]\mathrm{d}t.$$

The first lemma states a few properties of the Gaussian kernel.

Lemma B.1. $\int_{-\infty}^{\infty} \partial_t\lambda(t,t_0)\mathrm{d}t = 0$. $\int_{-\infty}^{\infty} (t-t_0)\,\partial_t\lambda_\sigma(t,t_0)\,\mathrm{d}t = -1$.

Proof. The first result is due to the Fundamental Theorem of Calculus and the fact that $\lambda(t,t_0) \to 0$ as $|t| \to \infty$. Now, we prove the second statement. Since

$$\lambda_\sigma(t,t_0) \;=\; \frac{1}{\sqrt{2\pi\,\sigma^2}}\,\exp\Big(-\frac{(t-t_0)^2}{2\,\sigma^2}\Big),$$

we have

$$\int_{-\infty}^{\infty} \lambda_\sigma(t, t_0)\,\mathrm{d}t \;=\; 1, \quad \text{and} \quad \partial_t \lambda_\sigma(t, t_0) \;=\; -\frac{(t - t_0)}{\sigma^2}\,\lambda_\sigma(t, t_0).$$

We then have with integration by parts

$$\int_{-\infty}^{\infty} (t - t_0)\,\partial_t \lambda_\sigma(t, t_0)\,\mathrm{d}t \;=\; (t - t_0)\,\lambda_\sigma(t, t_0)\Big|_{-\infty}^{\infty} \;-\; \int_{-\infty}^{\infty} \lambda_\sigma(t, t_0)\,\mathrm{d}t \;=\; 0 \;-\; 1 \;=\; -1. \tag{13}$$

the last equality is due to $\lim_{|t|\to\infty}(t - t_0)\,\lambda_\sigma(t, t_0) = 0$. □

First, we inspect $\int_{-\infty}^{\infty} \partial_t \lambda(t, t_0)\mathbb{E}\big[T\big(g(Z, t; \boldsymbol{w})\big)\big]\mathrm{d}t$. Using the Taylor expansion on $\mathbb{E}\big[T\big(g(Z, t; \boldsymbol{w})\big)\big]$ at t_0, we obtain

$$\mathbb{E}\big[T\big(g(Z, t; \boldsymbol{w})\big)\big] \;=\; \mathbb{E}\big[T\big(X_{t_0}\big)\big] \;+\; (t - t_0)\,\mathbb{E}\big[\nabla T\big(X_{t_0}\big)^\top \boldsymbol{h}\big(X_{t_0}; \boldsymbol{w}\big)\big].$$

Note that we don't have higher order terms as the drift model $g(Z, t; \boldsymbol{w})$ is a linear function of t by definition (see Definition 4.1).

Thus, due to Lemma B.1, we have

$$\int_{-\infty}^{\infty} \partial_t \lambda_\sigma(t, t_0)\,\Big[\mathbb{E}\big(T\big(X_{t_0}\big)\big) \;+\; (t - t_0)\,\mathbb{E}\big(\nabla T\big(X_{t_0}\big)^\top \boldsymbol{h}\big(X_{t_0}; \boldsymbol{w}\big)\big)\Big]\,\mathrm{d}t \;=\; -\,\mathbb{E}\big[\nabla T\big(X_{t_0}\big)^\top \boldsymbol{h}\big(X_{t_0}; \boldsymbol{w}\big)\big]. \tag{14}$$

Now we shift our focus on

$$C \;=\; \int_{-\infty}^{\infty} \lambda_\sigma(t, t_0)\,\mathrm{Cov}\big[T\big(g(Z, t; \boldsymbol{w})\big)\big]\,\mathrm{d}t.$$

As $\sigma \to 0$, $\lambda_\sigma(t, t_0)$ converges to $\delta(t - t_0)$, so $\lim_{\sigma\to 0}\big(\int \lambda(t, t_0) f(t)\mathrm{d}t\big) = f(t_0)$. Hence

$$\lim_{\sigma\to 0} C = \lim_{\sigma\to 0}\int_{-\infty}^{\infty} \lambda_\sigma(t, t_0)\,\mathrm{Cov}\big[T\big(g(Z, t; \boldsymbol{w})\big)\big]\,\mathrm{d}t \;=\; \mathrm{Cov}\big[T(X_{t_0})\big]. \tag{15}$$

Finally, combining Equation (14) and Equation (15) we have the desired result.

C PROOF OF THEOREM 4.3

Proof. First, we introduce Welling's Woodbury identity [Welling, 2019]:

$$(P^{-1} + B^T R^{-1} B)^{-1} B^T R^{-1} = P B^T (B P B^T + R)^{-1}.$$

Recall, that we try to minimise Equation (8)

$$\|\nabla\mathcal{L}(\boldsymbol{\theta}_{t_0}) - \mathbb{E}[\nabla T(X_{t_0})\nabla \boldsymbol{h}_w(X_{t_0})]\|^2_{\mathcal{F}^{-1}_{t_0}} + \lambda\|w\|^2_{\mathcal{H}}. \tag{16}$$

Expanding the first square, up to a constant that does not depend on w, we obtain

$$\mathbb{E}[\nabla T(X_{t_0})\boldsymbol{h}_w(X_{t_0})]^\top \mathcal{F}^{-1}_{t_0}\mathbb{E}[\nabla T(X_{t_0})\boldsymbol{h}_w(X_{t_0})] - 2\nabla^\top\mathcal{L}(\boldsymbol{\theta}_{t_0})\mathcal{F}^{-1}_{t_0}\mathbb{E}[T(X_{t_0})\boldsymbol{h}_w(X_{t_0})] + \lambda\|w\|^2_{\mathcal{H}}. \tag{17}$$

By definition, $\boldsymbol{h}_w(X_{t_0}) = \langle w, \nabla k(X_{t_0}, \cdot)\rangle$, we obtain a quadratic form with respect to w,

$$\langle w, \mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0}, \cdot)]^\top \mathcal{F}^{-1}_{t_0}\mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0}, \cdot)]w\rangle - \langle w, 2\mathbb{E}[T(X_{t_0})\nabla k(X_{t_0}, \cdot)]^\top \mathcal{F}^{-1}_{t_0}\nabla\mathcal{L}(\boldsymbol{\theta}_{t_0})\rangle + \lambda\|w\|^2_{\mathcal{H}}, \tag{18}$$

where we used $a^\top BCd = (BCd)^\top a = d^\top C^\top B^\top a = \langle d, C^\top B^\top a\rangle$ on the second term and the fact that the inverse of Fisher Information Matrix $\mathcal{F}_{t_0}$ is a symmetric matrix. Differentiating both sides by w and setting the gradient to zero, we obtain the following optimality condition of the least squares:

$$2\mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0}, \cdot)]^\top \mathcal{F}^{-1}_{t_0}\mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0}, \cdot)]w - 2\mathbb{E}[T(X_{t_0})\nabla k(X_{t_0}, \cdot)]^\top \mathcal{F}^{-1}_{t_0}\nabla\mathcal{L}(\boldsymbol{\theta}_{t_0}) + 2\lambda w = 0.$$

Thus, the closed form solution of the optimal solution w^* is

$$\left(\mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0},\cdot)]^\top \underbrace{\mathcal{F}_{t_0}^{-1}}_{R^{-1}} \underbrace{\mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0},\cdot)]}_{B} + \underbrace{\lambda \boldsymbol{I}}_{P^{-1}}\right)^{-1} \mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0},\cdot)]^\top \mathcal{F}_{t_0}^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}_{t_0}). \tag{19}$$

Applying Woodbury's identity, we get:

$$\mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0},\cdot)]^\top \left(\lambda \mathcal{F}_{t_0} + \mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0},\cdot)]\mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0},\cdot)]^\top\right)^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}_{t_0}). \tag{20}$$

Notice the product $\mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0},\cdot)]\mathbb{E}[\nabla T(X_{t_0})\nabla k(X_{t_0},\cdot)]^\top = \mathbb{E}[\nabla T(X_{t_0})\nabla\nabla k(X_{t_0}, X'_{t_0})\nabla^\top T(X'_{t_0})]$, where $\nabla\nabla k(x,y) := \nabla_x \nabla_y k(x,y)$ we obtain the desired result.

□

D STEIN EXPONENTIAL FAMILY

In some applications (e.g. Variational Inference), we do not have samples from the target distribution, instead, we have an unnormalized density. Below, we introduce a special type of exponential family, named Stein exponential family, that allows us to approximate the natural gradient of $\mathrm{KL}[p, q]$.

Definition D.1. A distribution belongs to the Stein exponential family of a target density p and a test function $\boldsymbol{f} \in \mathbb{R}^d$ if and only if it belongs to the exponential family with a sufficient statistic that is $T = S_p\boldsymbol{f}$. Given a test function $\boldsymbol{f} = [f_1, f_2, \dots, f_b]$, a Stein operator $S_p\boldsymbol{f}$ of a probability density p is a vector of functions defined as $S_p\boldsymbol{f} = [S_pf_1, S_pf_2, \dots, S_pf_b]$, where

$$S_pf_i = \partial_i(\log p)\boldsymbol{f} + \partial_i\boldsymbol{f}. \tag{21}$$

An important property of this special type of exponential family is that the expectation of the sufficient statistic is zero under the target distribution, i.e., $\mathbb{E}_p[T(\boldsymbol{x})] = \boldsymbol{0}$. More discussions on the Stein operator could be found in []. Therefore, if $q_{\boldsymbol{\theta}}$ is a Stein exponential family distribution, we can write the natural gradient of $\mathrm{KL}[p, q_{\boldsymbol{\theta}}]$ in Equation (2) as

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \mathcal{F}_t^{-1}\mathbb{E}_{q_{\boldsymbol{\theta}}}\left[S_pf(\boldsymbol{x})\right]. \tag{22}$$

Since the Stein operator only requires the gradient of the log density $\log p$ taken with respect to its input, this special design of exponential family enables us to compute the gradient for $\mathrm{KL}[p, q_t]$ using unnormalized p only.

This parametric model opens up applications such as variational inference and sampling: Given an unnormalized density p, we want to train a generative model to sample from p. We can design a Stein exponential family so that we can approximately minimise $\mathrm{KL}[p, q_t]$ using kernel NGD by pushing particles towards the target distribution.

E REVERSE KL WASSERSTEIN GRADIENT FLOW AND MMD FLOW

The WGF dynamics that minimise $\mathrm{KL}[q_t, p]$ move particles X_t using a simple velocity field:

$$\frac{\mathrm{d}X_t}{\mathrm{d}t} = \nabla\left(\log p\right)(X_t) - \nabla\left(\log q_t\right)(X_t),$$

where the gradient of log density could be easily estimated via kernel density estimation and the MMD flow minimises $\mathrm{MMD}[Y, X_t]$ using the following velocity field:

$$\frac{\mathrm{d}X_t}{\mathrm{d}t} = N\nabla\left(\frac{1}{N}\sum_{i=1}^{n}\|X_t - X_t^{(i)}\| - \frac{1}{M}\sum_{j=1}^{M}\|X_t - Y^{(j)}\|\right).$$

F EXPERIMENT SETUP IN SECTION 5

We summarize the experiments' setup details in this section. For each experiment, we provide details of the dataset and pre-processing procedure, as well as the details of tuning parameters.

F.1 COMPARISON WITH REVERSE KL WASSERSTEIN GRADIENT FLOW AND MMD FLOW

F.1.1 Dataset and Pre-processing

In this experiments, we let $p = 0.5\mathcal{N}(-2, \boldsymbol{I}) + 0.5\mathcal{N}(2, \boldsymbol{I})$. We draw 100 samples from p as the target samples Y, 100 samples from $\mathcal{N}(0, \boldsymbol{I})$ as the initial samples X_0. No further processing is required.

F.1.2 Parameter Tuning

The main tuning parameter are kernel bandwidth and step sizes.

For all methods that uses RBF kernel/basis, we set the bandwidth to be the median pairwise distance of all samples.

For all methods, we use step size 1, as any larger learning rate would result in numerical instability for each method.

For all methods, we run 100 particle updates.

The performance metric MMD uses a Gaussian kernel and the bandwidth is set as the median of pairwise distances of all samples Y and X_t.

F.2 GRAPHICAL MODEL RECOVERY

F.2.1 Dataset and Pre-processing

In this experiment, we let p be a 30-dimensional Gaussian graphical model, and draw 200 samples $Y \sim p$, 200 samples $X_0 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The graphical model $\boldsymbol{\Theta}$ is generated as a random graph, with edge probability 0.05. For each non-zero off-diagonal entry, we set $\Theta_{i,j} = 0.3$. No further processing is required.

F.2.2 Parameter Tuning

For all methods, we use the median of sample pairwise distances as the bandwidth.

For all methods, we set the step size to be 1.

Parameter tuning of Graphical Lasso is handled by `sklearn` internally using 5-fold cross validation and the sparse graph in graphical model is obtained by truncating all values smaller than 0.1. Below are the Python code.

```
# Fit with cross-validation to select alpha
model_cv = GraphicalLassoCV(alphas=10,  # number of alphas or list of alphas
                            cv=5,       # how many folds in cross-validation
                            max_iter=100,
                            tol=1e-4)
model_cv = model_cv.fit(x1_test.cpu().numpy())
Theta_cv = model_cv.precision_
Theta_cv = Theta_cv > 1e-1
```

F.3 EXPERIMENT 2: COVARIATE SHIFT

F.3.1 Dataset and Pre-processing

We validate our method on the dataset Office+Caltech[2], which is a dataset for domain adaptation, consisting of Office 10 and Caltech 10 datasets. It contains the 10 overlapping categories between the Office dataset and Caltech256 dataset [Gong et al., 2012].

[2]`https://github.com/jindongwang/transferlearning/blob/master/data/dataset.md#office+caltech`

The original features are extracted using a DECAF network, and are 4096 dimensional. We apply PCA on the source and target domain to reduce the dimension to 50 with Python code

```
        from sklearn.decomposition import PCA
        pca = PCA(n_components=50)
        pca.fit(X)
        X = pca.transform(X)
        X = X / 100
```

Due to memory space limit, we also randomly pick 200 samples from all target domains as X_0.

F.3.2 Parameter Tuning

For all methods, we set step size to 0.1.

For ntKiNG, we run 100 steps due to reduce the computation cost.

For WGF and MMD flow, we run 1000 steps.

The source classifier is an RBF kernel Support Vector Machines with all hyper-parameters chosen by cross-validation with the following python code:

```
    # Split the data into training and test sets (optional)
    X_train, X_test, y_train, y_test = train_test_split(x, y,
                                 test_size=0.3, random_state=42)

    # Define parameter grid
    param_grid = {
        'C': np.logspace(-3, 3, 5),
        'gamma': np.linspace(.2, 5, 5) * gamma,
        'kernel': ['rbf']
    }

    # Create a SVC classifier
    svc = SVC()

    # Initialize GridSearchCV
    grid_search = GridSearchCV(svc, param_grid, refit=True, verbose=2, cv=5)

    # Fit the model
    grid_search.fit(X_train, y_train)
```

where gamma is the inverse of the median pairwise distances of all inputs.

F.4 EXPERIMENT 3: DENOISING

The deep EBM is constructed using the following pyTorch code:

```
# Define the MLP-based energy-based model
class EnergyBasedModel(nn.Module):
    def __init__(self, input_dim):
        super(EnergyBasedModel, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(input_dim, 1024),
            nn.SiLU(),
            nn.Linear(1024, 1024),
            nn.SiLU(),
            nn.Linear(1024, 211),
            nn.SiLU(),
            nn.Linear(211, 1)
        )

    def forward(self, x, penultimate=False, flattened=False):
        x = x.view(x.size(0), -1)  # Flatten the input
        if not penultimate:
            return self.network(x)
        else:
            for i, layer in enumerate(self.network):
                x = layer(x)
                if i == len(self.network) - 2:
                    break
            return x
```

The model is trained using 10000 samples of Y, with a batch size 777, and adam optimizer with a step size of 0.001. In the S-curve experiment, we add a Gaussian noise to the target sample Y with a standard deviation 0.3, and in the MNIST experiment, we add a Gaussian noise to Y with a standard deviation 0.2.

The noise used in denoising score matching mathces the noise added to Y.