LEARNING MIXTURE DENSITY VIA NATURAL GRADIENT EXPECTATION MAXIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

We present *natural gradient expectation maximization* (nGEM), a novel approach to learning mixture density networks for conditional density estimation. Mixture density networks are conventionally optimized via the negative log-likelihood (NLL) objective, which suffers from slow convergence and mode collapse. We instead cast mixture density networks as latent-variable models that can be optimized in an expectation maximization framework, and derive tractable factorized natural gradient updates drawing upon theoretical ties between expectation maximization and natural gradient descent. Empirically, we show that nGEM achieves up to $10\times$ faster convergence with almost *zero* extra computational overhead, yet scales well to practical and challenging tasks where NLL otherwise fails for mode collapse.

1 INTRODUCTION

Mixture density networks [5] are neural networks that produce Gaussian mixtures, which are universal approximators of continuous and smooth densities, for conditional density estimation. Specifically, mixture density networks define a conditional distribution $p(\boldsymbol{y}|\boldsymbol{x})$ via a neural network that maps conditions $\boldsymbol{x}$ to the parameters of Gaussian mixture distributions over targets $\boldsymbol{y}$.

Mixture density networks are widely used in multi-valued or distributional regression, such as speech synthesis [8], robotic dynamics [19], trajectory prediction [30], as well as reinforcement learning [2, 14] and flow matching [9]. Mixture density networks are also particularly advantageous for applications that favor fast sampling, analytical density, and interpretable uncertainty quantification.

Despite being accessible in principle, the practical adoption of mixture density networks suffers from optimization challenges. Mixture density networks are usually trained via stochastic gradient descent to minimize the negative log-likelihood loss [5], which is non-convex and contains many poor local minima [10]. Empirical studies [28, 30, 46] also reveal that mixture density networks are susceptible to mode collapse, where Gaussian mixtures effectively degenerate into a single Gaussian component.

Our Contributions In this work, we revisit mixture density network optimization through the lens of latent-variable models and expectation maximization (EM). We identify and build on theoretical connections between EM and natural gradient descent for Gaussian mixtures to construct a tractable and efficient natural gradient update (nGEM), which yields fast and stable convergence by leveraging the information geometry of the underlying probabilistic models.

Empirically, we demonstrate that nGEM effectively mitigates mode collapse and achieves up to $10\times$ faster convergence, while inducing negligible extra computational overhead. We further validate the scalability of nGEM to challenging practical applications, spanning image and regression modeling, simulation-based inference, and distributional reinforcement learning.

2 PRELIMINARIES

In this section, we briefly review the *foundational* concepts: mixture density networks, natural gradient descent, and expectation maximization, upon which the proposed method nGEM is established.

Mixture Density Networks (MDNs) [5] model conditional distributions $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$ using Gaussian mixtures and neural networks. Specifically, MDNs learn a neural network $f(\boldsymbol{x};\boldsymbol{\theta})$ parametrized by $\boldsymbol{\theta}$

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

that maps conditions $\boldsymbol{x}$ to the parameters $\phi(\boldsymbol{x})$ of a Gaussian mixture, where

$$\phi(\boldsymbol{x}) := \{\boldsymbol{\pi}_k(\boldsymbol{x}), \boldsymbol{\mu}_k(\boldsymbol{x}), \boldsymbol{\Sigma}_k(\boldsymbol{x})\}_{k=1}^K \tag{1}$$

is the collection of the weights, means, and covariances of the K Gaussian mixture components.

Given data points $\{\boldsymbol{x}, \boldsymbol{y}\}$, the neural network parameters $\boldsymbol{\theta}$ are typically optimized by minimizing the negative log-likelihood (NLL) loss of the conditional distribution $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$:

$$\mathcal{L}_{\text{NLL}}(\boldsymbol{\theta}) = -\log \sum_{k=1}^{K} \boldsymbol{\pi}_k(\boldsymbol{x}) \mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}_k(\boldsymbol{x}), \boldsymbol{\Sigma}_k(\boldsymbol{x})). \tag{2}$$

Natural Gradient Descent (NGD) [1] is a generalization of the canonical gradient descent (GD) algorithm that accounts for the curvature of the parameter space of probabilistic models $p(\boldsymbol{x}|\boldsymbol{\theta})$. Consider minimizing the objective $J(\boldsymbol{\theta})$ with a learning rate β via gradient descent

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \beta \nabla J(\boldsymbol{\theta}_t). \tag{3}$$

GD performs steepest descent with a Euclidean distance penalty between parameter updates $\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t+1}\|_2^2$. For probabilistic models $p(\boldsymbol{x}|\boldsymbol{\theta})$, however, a more *natural* penalty[1] is the KL divergence $\mathbb{D}_{\text{KL}}(p(\boldsymbol{x}|\boldsymbol{\theta}_t)\|p(\boldsymbol{x}|\boldsymbol{\theta}_{t+1}))$ between the induced distributions, resulting in the NGD update:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \beta F^{-1}(\boldsymbol{\theta}_t) \nabla J(\boldsymbol{\theta}_t), \qquad F(\boldsymbol{\theta}) := -\mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta})}[\nabla_{\boldsymbol{\theta}}^2 \log p(\boldsymbol{x}|\boldsymbol{\theta})], \tag{4}$$

where $F^{-1}(\boldsymbol{\theta}_t)$ is the inverse *Fisher information matrix* (FIM) of $p(\boldsymbol{x}|\boldsymbol{\theta}_t)$. Specifically, the FIM $F(\boldsymbol{\theta})$ under certain conditions [31] can be defined as the negative expected Hessian of the log density.

Intuitively, NGD utilizes second-order derivatives (FIM) to capture the local geometry of the statistical manifold, reprojecting (or *preconditioning*) the gradient updates for faster and more stable optimization. However, a critical downside of NGD lies in the computation and inversion of $F(\boldsymbol{\theta})$, which in many cases are intractable and require expensive approximation [31].

Expectation Maximization (EM) [13] is an algorithm for finding maximum likelihood solutions of probabilistic models with missing data. EM is particularly well-known for its application in Gaussian mixture models, where each observed data point is assumed to be generated by one of the mixture components whose identity is missing.

Let $\boldsymbol{x}$ denote the observed data, $\boldsymbol{z}$ denote the missing data, and $p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$ denote the probabilistic model parametrized by $\boldsymbol{\theta}$. Directly maximization of the *observed-data* log-likelihood $\log p(\boldsymbol{x}|\boldsymbol{\theta}) = \log \int p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) \mathrm{d}\boldsymbol{z}$ is generally difficult due to the integration over $\boldsymbol{z}$. However, note that

$$\begin{aligned} \log p(\boldsymbol{x}|\boldsymbol{\theta}) =& \mathcal{F}(q(\boldsymbol{z}), \boldsymbol{\theta}) + \mathbb{D}_{\text{KL}}(q(\boldsymbol{z})\|p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta})), \quad \text{where} \\ & \mathcal{F}(q(\boldsymbol{z}), \boldsymbol{\theta}) := \mathbb{E}_{q(\boldsymbol{z})}[\log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) - \log q(\boldsymbol{z})] \leq \log p(\boldsymbol{x}|\boldsymbol{\theta}) \end{aligned} \tag{5}$$

is the the *evidence lower bound* (ELBO) of $\log p(\boldsymbol{x}|\boldsymbol{\theta})$ [35], as KL divergence is non-negative.

The EM algorithm maximizes $\log p(\boldsymbol{x}|\boldsymbol{\theta})$ by instead maximizing the lower bound $\mathcal{F}(q(\boldsymbol{z}), \boldsymbol{\theta})$ iteratively. For each iteration t, we alternate between the E-step and the M-step:

- (**E-step**) we maximize $\mathcal{F}(q(\boldsymbol{z}), \boldsymbol{\theta})$ with $\boldsymbol{\theta} = \boldsymbol{\theta}_t$ fixed

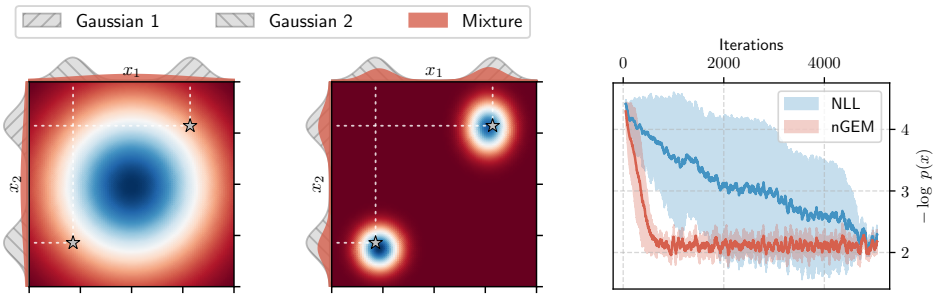
$$q_t(\boldsymbol{z}) = \arg\max_{q(\boldsymbol{z})} \mathcal{F}(q(\boldsymbol{z}), \boldsymbol{\theta}_t) = p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}_t). \tag{6}$$

- (**M-step**) we maximize $\mathcal{F}(q(\boldsymbol{z}), \boldsymbol{\theta})$ with $q(\boldsymbol{z}) = q_t(\boldsymbol{z})$ fixed

$$\boldsymbol{\theta}_{t+1} = \arg\max_{\boldsymbol{\theta}} \mathcal{F}(q_t(\boldsymbol{z}), \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}_t)}[\log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})]. \tag{7}$$

EM guarantees monotonic improvement of $\log p(\boldsymbol{x}|\boldsymbol{\theta})$ until convergence to some *local* maxima [13]. For cases where exact maximization is not tractable, the M-step (7) can be substituted with gradient ascent steps, leading to the generalized EM algorithm [6].

[1] We refer readers to Appendix A for further justification on natural gradient descent.



(a) NLL (b) nGEM (c) Convergence speed with $\beta = 0.01$

Figure 1: Fitting two Gaussians with a Gaussian mixture in $\mathbb{R}^2$. *(a)* Mode collapse with NLL. *(b)* Mode separation with nGEM. Heatmaps denote the learned mixture density, while stars ($\star$) denote the ground truth Gaussian means. Marginal distributions are displayed aside as hashed gray regions for the ground truth Gaussians and red regions for the Gaussian mixture. *(c)* Negative log-likelihoods (↓) of the Gaussian mixture with learning rate $\beta = 0.01$, averaged ($\pm$ std) across 5 random seeds.

3 Natural Gradient Expectation Maximization

While Gaussian mixtures are in theory capable of approximating any continuous and smooth density [6], in practice mixture density networks are found to suffer from slow convergence and mode collapse [28, 30, 46], where the model manages to locate a *subpar mean fit* using one mixture component to cover all or multiple modes in the target distribution (see Figure 1a).

One could instead consider improving the standard NLL objective via natural gradient preconditioning, which accounts for the curvature of the probabilistic parameter space. Unfortunately, this approach is not directly applicable as the FIM of Gaussian mixtures is non-analytical (§B.5).

To address this intractability, we propose natural gradient expectation maximization (nGEM). Building on classic Gaussian mixture models, we establish an EM framework for mixture density networks (§3.1). We show that EM, NGD, and NLL for Gaussian mixtures are tightly connected and in fact *equivalent* under mild assumptions (§3.2). We then exploit such connections to derive a *tractable* natural gradient EM (nGEM) update for learning mixture density networks (§3.3).

3.1 Stochastic Gradient Expectation Maximization

We begin by reinterpreting mixture density networks as deep latent-variable models, with observed features $\boldsymbol{x}$ and targets $\boldsymbol{y}$, latent variables $\boldsymbol{z}$, and network parameters $\boldsymbol{\theta}$. The joint distribution of $\{\boldsymbol{y}, \boldsymbol{z}\}$ conditioned on $\boldsymbol{x}$ and $\boldsymbol{\theta}$ is given by[2]

$$p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta}) = p(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta}) p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}), \tag{8}$$

where $p(\boldsymbol{z}|\boldsymbol{x}) = \text{Cat}(\boldsymbol{\pi}(\boldsymbol{x}))$ is a categorical distribution with $\boldsymbol{\pi}(\boldsymbol{x}) = [\boldsymbol{\pi}_1(\boldsymbol{x}) \ \cdots \ \boldsymbol{\pi}_K(\boldsymbol{x})] \in \Delta^{K-1}$, and $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z} = k) = \mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}_k(\boldsymbol{x}), \boldsymbol{\Sigma}_k(\boldsymbol{x}))$ is a Gaussian distribution.

The observed-data log-likelihood can be obtained by marginalizing $p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x})$ over latent variables $\boldsymbol{z}$:

$$\log p(\boldsymbol{y}|\boldsymbol{x}) = \log \sum_{k=1}^{K} \boldsymbol{\pi}_k(\boldsymbol{x}) \mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}_k(\boldsymbol{x}), \boldsymbol{\Sigma}_k(\boldsymbol{x})), \tag{9}$$

which coincides with the negative of $\mathcal{L}_{\text{NLL}}$ (2). Recall that EM aims to maximize the observed-data log-likelihood (equivalently minimizing NLL) by maximizing the lower bound objective $\mathcal{F}(q(\boldsymbol{z}), \boldsymbol{\theta})$ (5). Similarly, we can follow an iterative EM procedure for MDNs. In the **E-step**, we compute (6)

$$q_t(\boldsymbol{z} = k|\boldsymbol{x}) = p(\boldsymbol{z} = k|\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}_t) = \frac{\boldsymbol{\pi}_k(\boldsymbol{x}) \mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}_k(\boldsymbol{x}), \boldsymbol{\Sigma}_k(\boldsymbol{x}))}{\sum_{j=1}^{K} \boldsymbol{\pi}_j(\boldsymbol{x}) \mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}_j(\boldsymbol{x}), \boldsymbol{\Sigma}_j(\boldsymbol{x}))}. \tag{10}$$

[2] We shall omit the dependency on $\boldsymbol{\theta}$ hereafter whenever there is no ambiguity.

For brevity, we let $\boldsymbol{\rho}_k(\boldsymbol{x}) := q_t(\boldsymbol{z} = k|\boldsymbol{x})$, and $\boldsymbol{\rho}(\boldsymbol{x}) = [\boldsymbol{\rho}_1(\boldsymbol{x}) \ \cdots \ \boldsymbol{\rho}_K(\boldsymbol{x})] \in \Delta^{K-1}$ is known as the *responsibilities*, or a "soft" assignment of the data point $\{\boldsymbol{x}, \boldsymbol{y}\}$ to each mixture component.

In the **M-step**, however, exact maximization w.r.t. the parameters $\boldsymbol{\theta}$ is generally intractable due to the complex and non-linear nature of the neural network $f(\boldsymbol{x}; \boldsymbol{\theta})$. Following the generalized EM algorithm, we opt for stochastic gradient descent to maximize the M-step objective (7), leading to the following stochastic gradient EM (sGEM) loss:

$$\mathcal{L}_{\text{sGEM}}(\boldsymbol{\theta}) = -\sum_{k=1}^{K} \lfloor \boldsymbol{\rho}_k(\boldsymbol{x}) \rfloor \log \boldsymbol{\pi}_k(\boldsymbol{x}) \mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}_k(\boldsymbol{x}), \boldsymbol{\Sigma}_k(\boldsymbol{x})), \tag{11}$$

where $\lfloor \cdot \rfloor$ is the *stop-gradient* operator preventing gradient backpropagation, as the responsibilities $\boldsymbol{\rho}(\boldsymbol{x})$ from the E-step are held constant during the M-step as in canonical EM.

3.2 Key Concepts and Connections

Intriguingly, the NLL (2) and sGEM (11) objectives can be shown to be equivalent in gradients. We formalize this insight by proving that $\nabla \mathcal{L}_{\text{NLL}}(\boldsymbol{\theta}) = \nabla \mathcal{L}_{\text{sGEM}}(\boldsymbol{\theta})$ as follows.

Theorem 3.1 (37, 43, §B.1)**.** *Consider a probabilistic model $p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$ with observed data $\boldsymbol{x}$, latent variables $\boldsymbol{z}$, and parameters $\boldsymbol{\theta}$. The observed-data log-likelihood $\log p(\boldsymbol{x}|\boldsymbol{\theta})$ has the same gradients as generalized expectation maximization*

$$\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} = \nabla_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}|\boldsymbol{\theta}_t)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t},$$

where $Q(\boldsymbol{\theta}|\boldsymbol{\theta}_t) := \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}_t)}[\log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})]$ is the M-step objective (7).

Theorem 3.1 states that the observed-data log-likelihood $\log p(\boldsymbol{x}|\boldsymbol{\theta})$ and the M-step objective $Q(\boldsymbol{\theta}|\boldsymbol{\theta}_t)$ have equivalent gradients at $\boldsymbol{\theta} = \boldsymbol{\theta}_t$. We can trivially extend this result to mixture density networks.

Corollary 3.2 (§B.2)**.** *Consider a mixture density network $f(\boldsymbol{x}; \boldsymbol{\theta})$. The* NLL *objective (2) has the same gradients as the* sGEM *objective (11) at $\boldsymbol{\theta} = \boldsymbol{\theta}_t$*

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{NLL}}(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{sGEM}}(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t},$$

where $\boldsymbol{\theta}_t$ is the parameter used for evaluating responsibilities $\boldsymbol{\rho}(\boldsymbol{x})$ (10).

Although NLL and sGEM are equivalent in gradients, we shall see that sGEM unlocks an alternative perspective that allows bypassing the intractability of Gaussian mixture FIM to derive a tractable natural gradient update for MDNs. We start by showing that exact EM is equivalent to NGD.

Theorem 3.3 (26, 38, §B.3)**.** *For EM algorithms with exact E-M steps (6, 7), the update $\boldsymbol{\theta}_t \to \boldsymbol{\theta}_{t+1}$ is equivalent to a natural gradient descent step on the negative log-likelihood objective up to second-order Taylor approximation*

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \hat{F}(\boldsymbol{\theta}_t)^{-1} \nabla_{\boldsymbol{\theta}}(-\log p(\boldsymbol{x}|\boldsymbol{\theta}_t)), \qquad \hat{F}(\boldsymbol{\theta}) := -\mathbb{E}_{p(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta})}[\nabla^2_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})],$$

where the learning rate is one, and $\hat{F}(\boldsymbol{\theta}_t)$ is the complete-data Fisher information matrix at $\boldsymbol{\theta} = \boldsymbol{\theta}_t$.

For mixture density networks $f(\boldsymbol{x}; \boldsymbol{\theta})$, although exact EM updates are not possible, we can emulate such behaviors via natural gradient descent on the sGEM objective (11) using the *complete-data* FIM $\hat{F}(\phi(\boldsymbol{x}))$ w.r.t. the Gaussian mixture parameters $\phi(\boldsymbol{x})$ (1):

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \hat{F}(\phi(\boldsymbol{x}))^{-1} \underbrace{\nabla_{\phi(\boldsymbol{x})} \mathcal{L}_{\text{sGEM}}(\boldsymbol{\theta}_t) \cdot \nabla_{\boldsymbol{\theta}} \phi(\boldsymbol{x})}_{\text{Corollary 3.2 and chain rule}}, \tag{12}$$

Furthermore, we can show that the complete-data FIM of Gaussian mixtures is block-diagonal, enabling independent natural gradient preconditioning for each component distribution.

Theorem 3.4 (§B.4)**.** *Consider a Gaussian mixture $p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$ with observed data $\boldsymbol{x}$, latent variables $\boldsymbol{z}$, and parameters $\boldsymbol{\theta} = \{\boldsymbol{\pi}_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$. The complete-data FIM of the Gaussian mixture $p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$ is block-diagonal with $K + 1$ blocks:*

$$\hat{F}(\boldsymbol{\theta}) = \begin{bmatrix} \boldsymbol{\pi}_1 F_1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \boldsymbol{\pi}_K F_K & \vdots \\ 0 & \cdots & \cdots & F_{\boldsymbol{\pi}} \end{bmatrix},$$

where F_k for $k \in \{1, \ldots, K\}$ is the k-th Gaussian FIM, and $F_{\boldsymbol{\pi}}$ is the categorical FIM.

3.3 Natural Gradient Expectation Maximization

Figure 2 illustrates the theoretical connections among key concepts previously discussed in §3.2. Essentially, we establish an alternative route to connecting negative log-likelihoods and natural gradient descent via expectation maximization: *(a)* we show that NLL and EM are equivalent in gradients (Theorem 3.1), and that *(b)* EM is natural gradient descent with complete-data FIMs (Theorem 3.3).

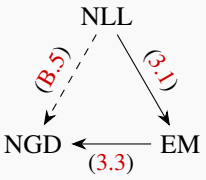


Figure 2: Key concepts and connections in §3.2.

In this section, we proceed to show that one can exploit the block-diagonal structure of the Gaussian mixtures complete-data FIMs (Theorem 3.4) to derive a tractable and factorized natural gradient EM (nGEM) update for mixture density networks.

We start by factorizing the sGEM objective (11) into the following summation of independent terms:

$$\mathcal{L}_{\text{nGEM}}(\boldsymbol{\theta}) = H\left(\lfloor \boldsymbol{\rho}(\boldsymbol{x}) \rfloor, \boldsymbol{\pi}(\boldsymbol{x})\right) - \sum_{k=1}^{K} \lfloor \boldsymbol{\rho}_k(\boldsymbol{x}) \rfloor \log \mathcal{N}(\boldsymbol{y}; \boldsymbol{\mu}_k(\boldsymbol{x}), \boldsymbol{\Sigma}_k(\boldsymbol{x})), \tag{13}$$

which we denote as the nGEM objective. The nGEM objective can be viewed as the sum of $K+1$ terms, where the first term is the cross-entropy between $\boldsymbol{\rho}(\boldsymbol{x})$ and $\boldsymbol{\pi}(\boldsymbol{x})$, and the other K terms are the negative log-likelihoods of Gaussian components weighted by $\boldsymbol{\rho}_k(\boldsymbol{x})$.

Therefore, the gradient of the nGEM objective (13) w.r.t. $\phi(\boldsymbol{x})$ is $\nabla_{\phi(\boldsymbol{x})} := [\nabla_1 \ \cdots \ \nabla_K \ \nabla_{\boldsymbol{\pi}}]$, where $\nabla_{\boldsymbol{\pi}}$ is the gradient of the cross entropy, and ∇_k for $k \in \{1, \ldots, K\}$ is the gradient of the weighted k-th Gaussian negative log-likelihood. Furthermore, due to the block-diagonality of the Gaussian mixture complete-data FIM $\hat{F}(\phi(\boldsymbol{x}))$, the complete-data natural gradient w.r.t. $\phi(\boldsymbol{x})$ is

$$\hat{\nabla}_{\phi(\boldsymbol{x})} := \hat{F}(\phi(\boldsymbol{x}))^{-1} \nabla_{\phi(\boldsymbol{x})} = \begin{bmatrix} (\boldsymbol{\pi}_1 F_1)^{-1} \nabla_1 & \cdots & (\boldsymbol{\pi}_K F_K)^{-1} \nabla_K & F_{\boldsymbol{\pi}}^{-1} \nabla_{\boldsymbol{\pi}} \end{bmatrix}. \tag{14}$$

That is, the complete-data natural gradient $\hat{\nabla}_{\phi(\boldsymbol{x})}$, which is core to the natural gradient EM update (12), can be obtained by preconditioning the regular gradient $\nabla_{\phi(\boldsymbol{x})}$ block-wise for each categorical ($\nabla_{\boldsymbol{\pi}}$) or Gaussian ($\nabla_k$) distribution independently.

- For **Gaussian** distributions, we adopt the common assumption of diagonal covariances[3] in deep learning [25]. We show in §B.6 that for Gaussians parametrized by the mean $\boldsymbol{\mu}$ and (diagonal) standard deviation $\boldsymbol{\sigma}$, the FIM (15) is also diagonal and hence easily invertible.
- For **categorical** distributions parametrized by logits $\boldsymbol{\psi}$ with probabilities $\boldsymbol{\pi} = \operatorname{softmax}(\boldsymbol{\psi})$, we can show that the analytical FIM is (16), which is unfortunately not invertible. Instead we opt for the Moore-Penrose pseudo-inverse of $F(\boldsymbol{\psi})$ with justifications detailed in §B.8.

$$F([\boldsymbol{\mu} \quad \boldsymbol{\sigma}]) = \operatorname{diag}\left(\begin{bmatrix} 1/\boldsymbol{\sigma}^2 & 2/\boldsymbol{\sigma}^2 \end{bmatrix}\right) \tag{15}$$

$$F(\boldsymbol{\psi}) = \operatorname{diag}(\boldsymbol{\pi}) - \boldsymbol{\pi}\boldsymbol{\pi}^\top \tag{16}$$

We now present the complete natural gradient EM (nGEM) algorithm for optimizing mixture density networks. An example implementation in JAX [7] is also made available in Appendix C.

Algorithm 1 Natural Gradient Expectation Maximization

Initialize: training set $\mathcal{D} = \{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N$, mixture density network $f(\boldsymbol{x}; \boldsymbol{\theta}_0)$, learning rate β.

1: **for** each iteration $t \in \{1, \ldots, T\}$ **do**
2: Sample $(\boldsymbol{x}, \boldsymbol{y}) \sim \mathcal{D}$ and compute $\phi(\boldsymbol{x}) = f(\boldsymbol{x}; \boldsymbol{\theta}_t)$ (1)
3: (**E-step**) Compute the responsibilities $\boldsymbol{\rho}(\boldsymbol{x})$ (10)
4: (**M-step**) Compute regular gradients $\nabla_{\phi(\boldsymbol{x})}$ of $\mathcal{L}_{\text{nGEM}}$ (13)
5: Compute the Gaussian (15) and categorical (16) FIMs $F_1, \ldots, F_K, F_{\boldsymbol{\pi}}$
6: Compute the complete-data natural gradients $\hat{\nabla}_{\phi(\boldsymbol{x})}$ (14)
7: Update the mixture density network parameters $\boldsymbol{\theta}$ via backpropagation

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \beta \hat{\nabla}_{\phi(\boldsymbol{x})} \nabla_{\boldsymbol{\theta}} \phi(\boldsymbol{x})$$

Return: optimized mixture density network $f(\boldsymbol{x}; \boldsymbol{\theta}_T)$

[3] We have additionally derived the Gaussian FIM with low-rank covariances in §B.7.

4 EXPERIMENTS

In this section, we present experiments designed to answer the following *research questions*: (§4.1) Does nGEM generally yield faster convergence? (§4.2) Does nGEM help mitigate or prevent mode collapse? (§4.3) How does nGEM work with high-dimensional data? (§4.4) How does nGEM perform against baselines on real-world applications? How much computational overhead does nGEM incur?

Synthetic Tasks	(§4.1, §4.2)	Image Modeling	(§4.3)	Regression	(§4.4)
Simulation-based Inference	(§D.3)	Reinforcement Learning	(§D.4)	Hyperparameters	(§D.5)

4.1 MEASURING CONVERGENCE

We start by measuring the convergence rates of nGEM on two synthetic tasks, of which we have access to the ground-truth distributions for evaluation. The first synthetic example is the **Two-Gaussians** dataset, which contains $2n$ training data points equally sampled from two distinct and well-separated Gaussians in $\mathbb{R}^2$, as illustrated in Figures 1a and 1b.

Two-Gaussians is a simple *illustrative* task that does not require a neural network to model conditional distributions. Instead we directly learn the Gaussian mixture parameters using either nGEM or NLL. Figure 3 illustrates how nGEM performs against the standard NLL on this task with different learning rates β by measuring their negative log-likelihoods. We find that nGEM can consistently outperform NLL in terms of convergence rates: *(a)* nGEM can converge within around 500 iterations while NLL requires over 5000 iterations ($\beta = 10^{-2}$), and *(b)* nGEM can reliably converge to the optimal solution whereas NLL catastrophically fails due to mode collapse ($\beta = 10^{-1}$).

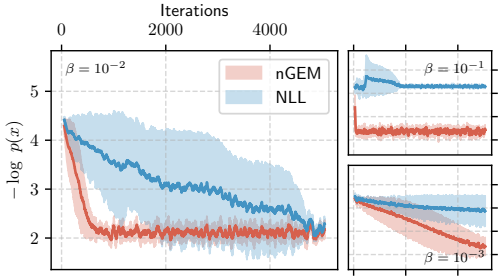


Figure 3: Negative log-likelihoods (↓) of the learned Gaussian mixture on the Two-Gaussians example with different learning rates β. Results averaged ($\pm$ std) across 5 random seeds.

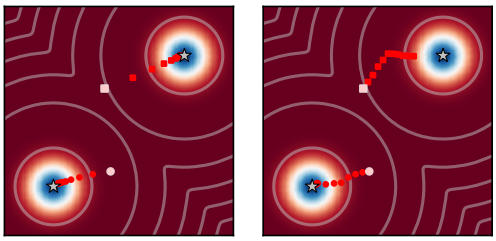


Figure 4: Trajectories of the Gaussian mixture component means. White square/circles are the initial means, red ones are the trajectories during training, and stars ($\star$) are the target means.

Figure 4 visualizes the trajectories of the Gaussian mixture component means during training. We observe that nGEM (*left*) moves along a straight path towards the targets ($\star$), presumably owing to nGEM's information geometry awareness. Furthermore, nGEM takes larger jumps in earlier iterations and automatically switches to smaller steps near convergence. On the other hand, NLL (*right*) moves along a curved path with fixed step sizes, resulting in slower convergence.

The second synthetic example is the **Two-Sinusoids** dataset, which requires modeling a conditional density $p(y|x)$ such that $x \sim \text{Uniform}(0, 4\pi)$ and y is either $\pi \sin(x) + \xi$ or $\pi \sin(x + \pi) + \xi$ with equal chances, where ξ is an additive Gaussian noise sampled from $\mathcal{N}(0, 10^{-1})$. We adopt a standard multi-layer perceptron (MLP) for MDN, and learn the parameters using either nGEM or NLL.

Figure 5 illustrates how nGEM performs against standard NLL on this task with different learning rates β. We find that nGEM still generally outperforms NLL, exhibiting faster and more stable convergence across learning rates. Figure 6 demonstrates that nGEM (*left*) can learn two separate conditional Gaussians to approximate the sine waves, while NLL (*right*) suffers from mode collapse and only uses one high-variance Gaussian to cover both modes.

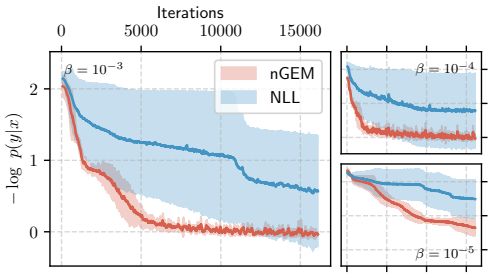


Figure 5: Negative log-likelihoods (↓) of the learned mixture density on the Two-Sinusoids example with different learning rates β. Results averaged ($\pm$ std) across 5 random seeds.

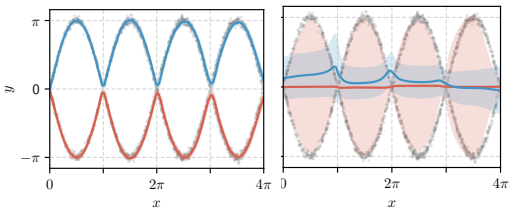


Figure 6: Visualization of the learned mixture density on the Two-Sinusoids example. Red and blue bands are the means ($\pm$ std) of the $K = 2$ Gaussian components conditioned on inputs x.

4.2 Measuring Mode Collapse

For measuring mode collapse, we reuse the synthetic tasks of which we know the optimal mixture weights $\boldsymbol{\pi}^* = [0.5\ 0.5]$ and entropy $H(\boldsymbol{\pi}^*) \approx 0.693$. We assess mode collapse severity by computing the entropy $H(\boldsymbol{\pi})$ of learned mixture weights, as low entropy typically correlates with mode collapse.

Tables 1 and 2 reports the entropy $H(\boldsymbol{\pi})$ of the learned mixture weights on the Two-Gaussians and Two-Sinusoids examples respectively. Note that nGEM consistently learns near-optimal mixture weights $\boldsymbol{\pi}$ without mode collapse, while NLL exhibits catastrophic failure ($\beta = 10^{-1}$ in Table 1) or high instability (high $\pm$ std in Table 2) across different learning rates.

Table 1: Entropy (↑) of the mixture weights $\boldsymbol{\pi}$ on Two-Gaussians with different learning rates β. Results averaged ($\pm$ std) across 5 random seeds.

Method	$\beta = 10^{-1}$	$\beta = 10^{-2}$	$\beta = 10^{-3}$
nGEM	**0.660 ± 0.03**	**0.691 ± 0.00**	**0.692 ± 0.00**
NLL	0.010 ± 0.00	0.687 ± 0.01	0.669 ± 0.02

Table 2: Entropy (↑) of the mixture weights $\boldsymbol{\pi}$ on Two-Sinusoids with different learning rates β. Results averaged ($\pm$ std) across 5 random seeds.

Method	$\beta = 10^{-3}$	$\beta = 10^{-4}$	$\beta = 10^{-5}$
nGEM	**0.693 ± 0.00**	**0.693 ± 0.00**	**0.693 ± 0.00**
NLL	0.416 ± 0.38	0.555 ± 0.31	0.413 ± 0.38

We conduction additional ablation experiments, and find that the *inverse weighting* term $\boldsymbol{\pi}_k^{-1}$ in the complete-data natural gradient (14) is most responsible for mitigating mode collapse. Intuitively, the inverse weighting $\boldsymbol{\pi}_k^{-1}$ amplifies the gradient signals for collapsed components (when $\boldsymbol{\pi}_k \to 0$) to help escaping local minima. We refer the readers to §D.1 for detailed experiment results and analysis.

4.3 Modeling High-Dimensional Data

We proceed to investigate how well nGEM scales to high-dimensional observations. We consider the **inverse MNIST** problem, where one aims to predict the handwritten digit image $\boldsymbol{x} \in \mathbb{R}^{28\times28}$ given its class label $\boldsymbol{y} \in \{0, \ldots, 9\}$. In other words, our model must learn to predict $28 \times 28 = 784$ pixels for each mixture component mean, substantially more challenging than previous toy examples in $\mathbb{R}^2$.

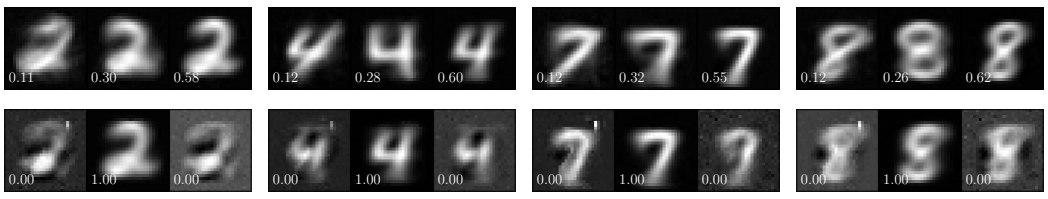


Figure 7: Learned Gaussian mixture ($K = 3$) means $\boldsymbol{\mu}_k$ given class labels $\boldsymbol{y} \in \{2, 4, 7, 8\}$ on the inverse MNIST problem for nGEM (*top*) or NLL (*bottom*). Numbers to the bottom left corner denote the corresponding mixture component weight $\boldsymbol{\pi}_k$. Full visualization results are available in §D.2.

Figure 7 inspects the predicted Gaussian mixture component means given labels. We observe that nGEM (*top*) can use different components to capture subtle style variations and predict diverse sharp images. On the other hand, NLL (*bottom*) suffers from mode collapse ($\boldsymbol{\pi}_k = 1$) and can only predict blurry, visually similar images. We also report the negative log-likelihoods and entropy of the learned mixture density networks on the inverse MNIST problem with $K \in \{3, 10\}$ in Table 6 (§D.2).

4.4 Applications and Baselines

We apply mixture density networks to general **regression modeling**. We conduct baseline experiments on three real-world regression tasks from the standard UCI datasets [21], involving robot kinematics (`kin8nm`), energy efficiency (`energy`), and Boston housing price prediction (`housing`).

For baselines, we consider combining nGEM / NLL with different optimizers, including Adam [24] and curvature-aware preconditioning gradient optimizers: K-FAC [32], Muon [23] and Soap [41]. We also consider NLL with Dirichlet prior regularization [28] for mixture density networks, as well as β-NLL [39], which uses conditional Gaussian distributions for regression, and the canonical mean squared error (MSE). Table 3 reports the the performance of each baseline on all datasets:

- Mixture density networks with nGEM / NLL generally outperform β-NLL and MSE.
- When using the same optimizer, nGEM outperforms NLL in RMSE on all tasks, and also in log-likelihood except `housing` (due to nGEM overfitting on the train set).
- Soap optimizer with nGEM generally performs best in both log-likelihood and RMSE.

In particular, Table 3 also highlights that nGEM is complementary to gradient optimizers [23, 32, 41], and their integration yields additive performance gain surpassing either one in isolation in some cases.

Table 3: Negative log-likelihoods (↓) and RMSE (↓) of 11 baselines and 3 regression tasks. Results are averaged (± std) across 5 random seeds. Note that *(a)* RMSE for Gaussian distributions is computed between the mean and the target, i.e. $\mathrm{RMSE}(\boldsymbol{\mu}, \boldsymbol{y})$, and that *(b)* RMSE for Gaussian mixtures is computed as the minimum RMSE among all Gaussian components, i.e. $\min_k \mathrm{RMSE}(\boldsymbol{\mu}_k, \boldsymbol{y})$.

		Negative Log-Likelihood (↓)			RMSE (↓) $\times 10^1$		
Method		`kin8nm`	`energy`	`housing`	`kin8nm`	`energy`	`housing`
Adam	+ nGEM	-1.73 ± 0.04	-3.29 ± 0.13	-0.39 ± 0.06	0.17 ± 0.01	0.49 ± 0.04	0.81 ± 0.07
	+ NLL	-1.67 ± 0.06	-3.09 ± 0.24	-0.67 ± 0.12	0.33 ± 0.03	0.95 ± 0.12	1.02 ± 0.05
KFAC	+ nGEM	-0.69 ± 0.02	-0.43 ± 0.01	1.15 ± 0.06	0.65 ± 0.03	1.51 ± 0.03	4.43 ± 0.60
	+ NLL	-1.64 ± 0.08	-0.96 ± 0.08	1.24 ± 0.05	0.31 ± 0.02	2.11 ± 0.35	5.11 ± 0.80
Muon	+ nGEM	-1.84 ± 0.05	-3.19 ± 0.08	-0.19 ± 0.02	0.15 ± 0.00	0.53 ± 0.01	1.03 ± 0.11
	+ NLL	-1.80 ± 0.06	-2.68 ± 0.17	-0.24 ± 0.01	0.25 ± 0.00	1.32 ± 0.18	1.51 ± 0.11
Soap	+ nGEM	**-1.96 ± 0.05**	**-4.18 ± 0.10**	-0.48 ± 0.14	**0.11 ± 0.01**	**0.30 ± 0.01**	**0.77 ± 0.09**
	+ NLL	-1.93 ± 0.06	-3.99 ± 0.18	**-0.79 ± 0.11**	0.21 ± 0.01	0.72 ± 0.05	1.00 ± 0.09
Adam	+ Dirichlet	-1.62 ± 0.04	-3.37 ± 0.15	-0.60 ± 0.19	0.37 ± 0.01	0.73 ± 0.03	1.06 ± 0.30
	+ β-NLL	-1.60 ± 0.02	-2.55 ± 0.16	-0.22 ± 0.05	0.40 ± 0.01	0.97 ± 0.04	1.65 ± 0.10
	+ MSE	—	—	—	0.39 ± 0.01	0.95 ± 0.02	1.42 ± 0.04

Apart from regression tasks, we further extend nGEM to simulation-based inference and distributional reinforcement learning. Upon preliminary investigation, mixture density networks with nGEM attain competitive results against established baselines. We defer full results and analysis to §D.3 and §D.4.

Simulation-based inference (SBI) [11] is concerned with Bayesian inference $p(\boldsymbol{\theta}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})$, where $p(\boldsymbol{y}|\boldsymbol{\theta})$ is an expensive or intractable likelihood function and can only be accessed through a black-box simulator. Neural posterior estimation (NPE) learns a neural conditional density estimator to approximate the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y})$ given samples from the prior and simulator.

Distributional reinforcement learning [4, 12] learns a distributional action-value function $Q(s, a)$, which predicts the distribution (instead of expectation) of the discounted future return conditioned on state-action pairs for sequential decision making. Figure 8 shows that mixture density networks w/ nGEM can solve discrete control tasks well.

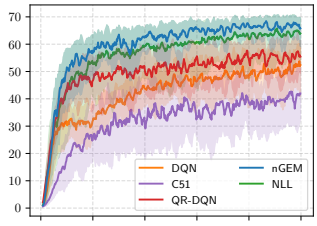


Figure 8: MDN w/ nGEM can solve Gymnax [27] Asterix control tasks.

Finally, Table 4 reports the **wall-clock time** (in seconds) of nGEM / NLL with large $K \in \{20, 100\}$ on the regression datasets. We observe that *(a)* nGEM incurs barely discernible extra runtime overhead over NLL, as nGEM does not require the expensive $O(n^3)$ matrix inversion operations (15, 16), and *(b)* the runtime gap between nGEM and NLL does not increase noticeably with K.

Table 4: Wall-clock time (↓) of nGEM and NLL running for 10^4 gradient updates with Adam optimizer and $K \in \{20, 100\}$ on the regression datasets. Results are averaged ($\pm$ std) across 5 runs.

	kin8nm		energy		housing	
Method	$K = 20$	$K = 100$	$K = 20$	$K = 100$	$K = 20$	$K = 100$
nGEM	10.87 $\pm$ 0.09s	12.04 $\pm$ 0.08s	7.35 $\pm$ 0.06s	8.21 $\pm$ 0.05s	8.09 $\pm$ 0.07s	9.21 $\pm$ 0.05s
NLL	**10.68 $\pm$ 0.04s**	**11.88 $\pm$ 0.06s**	**7.26 $\pm$ 0.05s**	**8.11 $\pm$ 0.08s**	**7.96 $\pm$ 0.09s**	**9.09 $\pm$ 0.11s**

5 Related Work

EM [17] and NGD [32] have both been independently applied to neural networks. However, while the theoretical interplay between EM and NGD is well-established in classical statistics [26, 38], their integration into neural mixture models remains largely unexplored. Our work bridges this gap by being the *first* to explicitly exploit the theoretical synergy to develop a principled optimization framework for mixture density networks, yielding significant empirical gains and insights.

Our theoretical contribution is most related to [39, 44]. β-NLL [39] predicts conditional Gaussians by weighting the Gaussian NLL with a β-exponentiated variance $\boldsymbol{\sigma}^{2\beta}$. Our proposed method nGEM decomposes into β-NLL when $K = \beta = 1$, in which case the inverse Gaussian FIM (15) is equivalent to $\boldsymbol{\sigma}^2$ up to constants. Yan et al. [44] uses Wasserstein-Fisher-Rao gradient flows to learn Gaussian mixtures. However, they *(a)* do not consider mixture density networks, and *(b)* assume simple identity covariances $\boldsymbol{\Sigma} = \mathbf{I}$ while we derive explicit updates for diagonal (§B.6) and low-rank (§B.7) cases.

It is worth noting that nGEM computes and back-propagates the natural gradient in the *output* space, in contrast to prior natural gradient methods [32] that directly operates in the network *weight* space.

Applications Mixture density networks have been applied to speech synthesis [8], sketch generation [18], drone trajectory prediction [30], human pose estimation [28], and flow matching [9]. Mixture density networks are also widely used in reinforcement learning (RL) and robotics, for example as stochastic control policies [14, 45, 46], or distributional value functions [2], or dynamics models [18].

Variational Inference Gaussian mixtures have also been used as surrogates $q(\boldsymbol{z})$ in (natural gradient) variational inference (VI) [3, 29, 34], where one approximates the intractable posterior $p(\boldsymbol{z}|\boldsymbol{x})$ by minimizing $\mathbb{D}_{\mathrm{KL}}(q(\boldsymbol{z})\|p(\boldsymbol{z}|\boldsymbol{x}))$, or equivalently maximizing ELBO (5). However, a key distinction is that in nGEM, the posterior $p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$ is simply the responsibilities (10), which is perfectly tractable and analytical, and our objective is not to estimate the surrogate $q(\boldsymbol{z})$ but the marginal $p(\boldsymbol{y}|\boldsymbol{x})$.

6 Conclusion

In this work, we introduce natural gradient expectation maximization (nGEM) for learning mixture density networks. We derive a tractable natural gradient EM update for mixture density networks building upon theoretical ties between EM and NGD. The proposed method nGEM can effectively stabilize and accelerate convergence, incurs negligible extra computational overhead, while remaining straightforward to implement and accessible for practitioners (see Appendix C).

Our empirical evaluations – spanning from synthetic examples to practical applications – suggest that nGEM is both effective and robust. We expect nGEM to become a *drop-in* replacement of NLL for mixture density networks, unlocking potential in a wide range of conditional density estimation tasks.

Limitations and Future Work Our work focus on mixtures of Gaussians with diagonal covariances. While theoretical results for low-rank covariances are derived (§B.7), they inflict $\mathcal{O}(r^3)$ additional computational cost due to inverting rank-r matrices in practice. Another future direction would be extending nGEM to neural mixtures of general exponential family distributions beyond Gaussians.

AI USE STATEMENT

In this work, we used generative AI for code generation, all of which are manually reviewed or modified for correctness by authors. We also used generative AI for drafting sections, all of which are later substantially edited by authors. Additionally, we used generative AI to aid proving theoretical claims, notably including Theorem 3.3, §B.7, and §B.8. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

[1] Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998. doi: 10.1162/089976698300017746.

[2] Mathis Antonetti, Henrique Donancio, and Florence Forbes. An analysis of distributional reinforcement learning with Gaussian mixtures. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=b4VgI1RTv8.

[3] Oleg Arenz, Philipp Dahlinger, Zihan Ye, Michael Volpp, and Gerhard Neumann. A unified perspective on natural gradient variational inference with Gaussian mixture models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=tLBjsX4tjs.

[4] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 449–458. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/bellemare17a.html.

[5] Christopher Bishop. Mixture density networks, 1994.

[6] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, January 2006.

[7] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

[8] Tim Capes, Paul Coles, Alistair Conkie, Ladan Golipour, Abie Hadjitarkhani, Qiong Hu, Nancy Huddleston, Melvyn Hunt, Jiangchuan Li, Matthias Neeracher, Kishore Prahallad, Tuomo Raitio, Ramya Rasipuram, Greg Townsend, Becci Williamson, David Winarsky, Zhizheng Wu, and Hepeng Zhang. Siri on-device deep learning-guided unit selection text-to-speech system. In *Interspeech 2017*, pages 4011–4015, 2017. doi: 10.21437/Interspeech.2017-1798.

[9] Hansheng Chen, Kai Zhang, Hao Tan, Zexiang Xu, Fujun Luan, Leonidas Guibas, Gordon Wetzstein, and Sai Bi. Gaussian mixture flow matching models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 9783–9802. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/chen25cl.html.

[10] Yudong Chen, Dogyoon Song, Xumei Xi, and Yuqian Zhang. Local minima structures in Gaussian mixture models. *IEEE Transactions on Information Theory*, 70(6):4218–4257, 2024. doi: 10.1109/TIT.2024.3374716.

[11] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020. doi: 10.1073/pnas.1912789117. URL https://www.pnas.org/doi/abs/10.1073/pnas.1912789117.

[12] Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8.

[13] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1):1–22, 1977.

[14] Sheelabhadra Dey and Guni Sharon. Comparing deterministic and soft policy gradients for optimizing Gaussian mixture actors. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=qS9pPu8ODt.

[15] Simon Dirmeier, Antonietta Mira, and Carlo Albert. Simulation-based inference with the Python package sbijax, 2026. URL https://arxiv.org/abs/2409.19435.

[16] David Greenberg, Marcel Nonnenmacher, and Jakob Macke. Automatic posterior transformation for likelihood-free inference. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2404–2414. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/greenberg19a.html.

[17] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Neural expectation maximization. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[18] David Ha and Douglas Eck. A neural representation of sketch drawings. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=Hy6GHpkCW.

[19] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[20] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs), 2023. URL https://arxiv.org/abs/1606.08415.

[21] Jose Miguel Hernandez-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1861–1869, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/hernandez-lobatoc15.html.

[22] Matthew D. Hoffman and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(47):1593–1623, 2014. URL http://jmlr.org/papers/v15/hoffman14a.html.

[23] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://openreview.net/forum?id=8gmWwjFyLj.

[25] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2013. URL https://openreview.net/forum?id=33X9fd2-9FyZd.

[26] Frederik Kunstner, Raunak Kumar, and Mark Schmidt. Homeomorphic-invariance of EM: Non-asymptotic convergence in KL divergence for exponential families via mirror descent. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3295–3303. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/kunstner21a.html.

[27] Robert Tjarko Lange. Gymnax: A JAX-based reinforcement learning environment library, 2022. URL http://github.com/RobertTLange/gymnax.

[28] Chen Li and Gim Hee Lee. Generating multiple hypotheses for 3D human pose estimation with mixture density network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9887–9895, June 2019.

[29] Wu Lin, Mohammad Emtiyaz Khan, and Mark Schmidt. Fast and simple natural-gradient variational inference with mixture of exponential-family approximations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3992–4002. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/lin19b.html.

[30] Osama Makansi, Eddy Ilg, Ozgun Cicek, and Thomas Brox. Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7144–7153, June 2019.

[31] James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020. URL http://jmlr.org/papers/v21/17-678.html.

[32] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2408–2417, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/martens15.html.

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[34] Warren Morningstar, Sharad Vikram, Cusuh Ham, Andrew Gallagher, and Joshua Dillon. Automatic differentiation variational inference with mixtures. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3250–3258. PMLR, 13–15 Apr 2021. URL https://proceedings.mlr.press/v130/morningstar21b.html.

[35] Radford M Neal and Geoffrey E Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

[36] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[37] Ruslan Salakhutdinov, Sam T Roweis, and Zoubin Ghahramani. Optimization with EM and expectation-conjugate-gradient. In *Proceedings of the 20th International Conference on Machine Learning*, pages 672–679, 2003.

[38] Masa-Aki Sato. Online model selection based on the variational Bayes. *Neural computation*, 13(7): 1649–1681, 2001.

[39] Maximilian Seitzer, Arash Tavakoli, Dimitrije Antic, and Georg Martius. On the pitfalls of heteroscedastic uncertainty estimation with probabilistic neural networks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=aPOpXlnV1T.

[40] Edan Toledo. Stoix: Distributed single-agent reinforcement learning end-to-end in jax, April 2024. URL https://github.com/EdanToledo/Stoix.

[41] Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham M. Kakade. SOAP: Improving and stabilizing Shampoo using Adam for language modeling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=IDxZhXrpNf.

[42] Jonas Wildberger, Maximilian Dax, Simon Buchholz, Stephen Green, Jakob H Macke, and Bernhard Schölkopf. Flow matching for scalable simulation-based inference. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 16837–16864. Curran Associates, Inc., 2023.

[43] Weihang Xu, Maryam Fazel, and Simon S. Du. Toward global convergence of gradient EM for over-paramterized Gaussian mixture models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 10770–10800. Curran Associates, Inc., 2024. doi: 10.52202/079017-0344.

[44] Yuling Yan, Kaizheng Wang, and Philippe Rigollet. Learning Gaussian mixtures using the Wasserstein–Fisher–Rao gradient flow. *The Annals of Statistics*, 52(4):1774 – 1795, 2024. doi: 10.1214/24-AOS2416. URL https://doi.org/10.1214/24-AOS2416.

[45] Hongyi Zhou, Denis Blessing, Ge Li, Onur Celik, Xiaogang Jia, Gerhard Neumann, and Rudolf Lioutikov. Variational distillation of diffusion policies into mixture of experts. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 12739–12766. Curran Associates, Inc., 2024. doi: 10.52202/079017-0405.

[46] You Zhou, Jianfeng Gao, and Tamim Asfour. Movement primitive learning and generalization: Using mixture density networks. *IEEE Robotics & Automation Magazine*, 27(2):22–32, 2020. doi: 10.1109/MRA.2020.2980591.

Appendix

A Natural Gradient Descent

In machine learning and optimization, the gradient descent algorithm for minimizing the objective function $J(\boldsymbol{\theta}) : \mathbb{R}^n \to \mathbb{R}$ is to iteratively update parameters $\boldsymbol{\theta}$ according to

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \beta \nabla J(\boldsymbol{\theta}_t), \tag{17}$$

where β is commonly known as the *learning rate*.

One can show that the above gradient descent update can be derived from minimizing the first-order Taylor expansion of $J(\boldsymbol{\theta})$ at $\boldsymbol{\theta} = \boldsymbol{\theta}_t$ with an additional Euclidean distance penalty

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}} \underbrace{J(\boldsymbol{\theta}_t) + \nabla J(\boldsymbol{\theta}_t)^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t)}_{\text{Taylor expansion}} + (2\beta)^{-1} \underbrace{\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2}_{\text{Penalty}}. \tag{18}$$

Therefore, we say that gradient descent implicitly assumes that the model parameters $\boldsymbol{\theta}$ lives in a *Euclidean space* by using a Euclidean distance penalty to constrain large updates. However, for probabilistic models $p(\boldsymbol{x}|\boldsymbol{\theta})$, the parameter space is generally not Euclidean. Instead, a more natural penalty term would be the Kullback-Leibler (KL) divergence between the induced distributions, resulting in the following optimization objective

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) + \nabla J(\boldsymbol{\theta}_t)^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t) + \beta^{-1} \underbrace{\mathbb{D}_{\mathrm{KL}}(p(\boldsymbol{x}|\boldsymbol{\theta}_t \| p(\boldsymbol{x}|\boldsymbol{\theta}))}_{\text{Penalty}}. \tag{19}$$

Since $\mathbb{D}_{\mathrm{KL}}(p\|q)$ does not have an analytical form generally, we instead consider its second-order Taylor expansion at $\boldsymbol{\theta} = \boldsymbol{\theta}_t$

$$\begin{aligned} \mathbb{D}_{\mathrm{KL}}(p(\boldsymbol{x}|\boldsymbol{\theta}_t \| p(\boldsymbol{x}|\boldsymbol{\theta})) \approx\ & \mathbb{D}_{\mathrm{KL}}(p(\boldsymbol{x}|\boldsymbol{\theta}_t \| p(\boldsymbol{x}|\boldsymbol{\theta}_t)) + \\ & + (\nabla_{\boldsymbol{\theta}} \mathbb{D}_{\mathrm{KL}}(p(\boldsymbol{x}|\boldsymbol{\theta}_t \| p(\boldsymbol{x}|\boldsymbol{\theta}))\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t})^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t) \\ & + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_t)^\top (\nabla^2_{\boldsymbol{\theta}} \mathbb{D}_{\mathrm{KL}}(p(\boldsymbol{x}|\boldsymbol{\theta}_t \| p(\boldsymbol{x}|\boldsymbol{\theta}))\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t})(\boldsymbol{\theta} - \boldsymbol{\theta}_t), \end{aligned} \tag{20}$$

where

1. The first term $\mathbb{D}_{\mathrm{KL}}(p(\boldsymbol{x}|\boldsymbol{\theta}_t \| p(\boldsymbol{x}|\boldsymbol{\theta}_t))$ is trivially *zero*.
2. The second term is also *zero* because the expectation of the score $\nabla \log p(\boldsymbol{x})$ is zero.

$$\nabla_{\boldsymbol{\theta}} \mathbb{D}_{\mathrm{KL}}(p(\boldsymbol{x}|\boldsymbol{\theta}_t \| p(\boldsymbol{x}|\boldsymbol{\theta}))\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} = -\mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta}_t)} \left[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta}_t)\right] = \mathbf{0}. \tag{21}$$

3. The third term can be simplified by noting that the Hessian of the KL divergence is exactly the Fisher information matrix (4).

$$\nabla^2_{\boldsymbol{\theta}} \mathbb{D}_{\mathrm{KL}}(p(\boldsymbol{x}|\boldsymbol{\theta}_t \| p(\boldsymbol{x}|\boldsymbol{\theta}))\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} = -\mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta}_t)} \left[\nabla^2_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta}_t)\right] = F(\boldsymbol{\theta}_t). \tag{22}$$

Therefore, we have $\mathbb{D}_{\mathrm{KL}}(p(\boldsymbol{x}|\boldsymbol{\theta}_t \| p(\boldsymbol{x}|\boldsymbol{\theta})) \approx \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_t)^\top F(\boldsymbol{\theta}_t)(\boldsymbol{\theta} - \boldsymbol{\theta}_t)$. Plugging the approximation back and solving the alternative optimization objective

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) + \nabla J(\boldsymbol{\theta}_t)^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t) + (2\beta)^{-1} \underbrace{(\boldsymbol{\theta} - \boldsymbol{\theta}_t)^\top F(\boldsymbol{\theta}_t)(\boldsymbol{\theta} - \boldsymbol{\theta}_t)}_{\text{Penalty}} \tag{23}$$

lead to the following natural gradient descent (NGD) update

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \beta F^{-1}(\boldsymbol{\theta}_t) \nabla J(\boldsymbol{\theta}_t). \tag{24}$$

Natural gradient descent yields the steepest descent on a Riemannian manifold with Fisher information metrics, which subsumes the standard gradient descent as a special case when $F(\boldsymbol{\theta}) = I$.

B Proofs and Derivations

B.1 Proof of Theorem 3.1

Consider a probabilistic model $p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$ with observed data $\boldsymbol{x}$, latent variables $\boldsymbol{z}$, and parameters $\boldsymbol{\theta}$. We begin by rewriting the gradient of the observed-data log likelihood as follows:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) &= \frac{1}{p(\boldsymbol{x}|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} p(\boldsymbol{x}|\boldsymbol{\theta}) \\
&= \frac{1}{p(\boldsymbol{x}|\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} \int p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) \mathrm{d}\boldsymbol{z} \\
&= \frac{1}{p(\boldsymbol{x}|\boldsymbol{\theta})} \int \nabla_{\boldsymbol{\theta}} p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) \mathrm{d}\boldsymbol{z} \\
&= \frac{1}{p(\boldsymbol{x}|\boldsymbol{\theta})} \int p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) \mathrm{d}\boldsymbol{z} \\
&= \int p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) \mathrm{d}\boldsymbol{z}.
\end{aligned} \tag{25}$$

Note that by definition $Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t) = \int p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta}_t) \log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) \mathrm{d}\boldsymbol{z}$. It is then clear that

$$\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} = \int p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{\theta}_t) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}_t) \mathrm{d}\boldsymbol{z} = \nabla_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}. \tag{26}$$

B.2 Proof of Corollary 3.2

We view the mixture density network $f(\boldsymbol{x}; \boldsymbol{\theta})$ as a latent variable model $p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta})$ with observed features $\boldsymbol{x}$ and targets $\boldsymbol{y}$, latent variables $\boldsymbol{z}$, and network parameters $\boldsymbol{\theta}$.

Following Theorem 3.1 with a trivial substitution of $p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$ with $p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta})$, we can derive that

$$\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta}) = \int p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta}) \mathrm{d}\boldsymbol{z}. \tag{27}$$

Recall that by definition we have

$$\mathcal{L}_{\text{NLL}}(\boldsymbol{\theta}) = -\log p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\theta}), \tag{28}$$

$$\mathcal{L}_{\text{sGEM}}(\boldsymbol{\theta}) = -\int p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}_t) \log p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta}) \mathrm{d}\boldsymbol{z}, \tag{29}$$

where $\boldsymbol{\theta}_t$ is the parameter used for evaluating the responsibilities $\boldsymbol{\rho}^{(\boldsymbol{x})}$ (10).

Therefore, it is clear that

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{NLL}}(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} = -\int p(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}_t) \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta}_t) \mathrm{d}\boldsymbol{z} = \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{sGEM}}(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}. \tag{30}$$

B.3 Proof of Theorem 3.3

Assuming exact E-step (6), the second-order Taylor expansion of the M-step objective $Q(\boldsymbol{\theta}|\boldsymbol{\theta}_t)$ (7) is

$$\begin{aligned}
Q(\boldsymbol{\theta}|\boldsymbol{\theta}_t) \approx\ & Q(\boldsymbol{\theta}_t|\boldsymbol{\theta}_t) \\
& + (\nabla_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t})^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_t) \\
& + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_t)^\top (\nabla^2_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}) (\boldsymbol{\theta} - \boldsymbol{\theta}_t).
\end{aligned} \tag{31}$$

Assuming exact M-step (7), the gradient of $Q(\boldsymbol{\theta}|\boldsymbol{\theta}_t)$ at the maximum $\boldsymbol{\theta}_{t+1}$ must be zero.

$$\nabla_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_{t+1}} = 0. \tag{32}$$

Replacing $Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)$ with the the second-order Taylor approximation (31), we have

$$(\nabla_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}) + (\nabla^2_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t})(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) \approx 0. \tag{33}$$

Note that because

- $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} = \nabla_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$ as shown in Theorem 3.1, and
- $\nabla^2_{\boldsymbol{\theta}} Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_t)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} = \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}_t)}\left[\nabla^2_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}_t)\right]$ by definition,

we can equivalently rewrite the above equation as follows

$$\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta}_t) + \mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}_t)}\left[\nabla^2_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}_t)\right](\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) \approx 0. \tag{34}$$

Taking the expectation on both sides w.r.t. the empirical dataset distribution $p(\boldsymbol{x})$, we have

$$\mathbb{E}_{p(\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta}_t)] + \underbrace{\mathbb{E}_{p(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta}_t)}\left[\nabla^2_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}_t)\right]}_{-\hat{F}(\boldsymbol{\theta}_t)}(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) \approx 0, \tag{35}$$

which can be equivalently rearranged as $\boldsymbol{\theta}_{t+1} \approx \boldsymbol{\theta}_t + \hat{F}(\boldsymbol{\theta}_t)^{-1}\mathbb{E}_{p(\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta}_t)]$.

Finally, when we consider a single data instance $\boldsymbol{x}$ such that $p(\boldsymbol{x})$ is a Dirac delta, we have

$$\boldsymbol{\theta}_{t+1} \approx \boldsymbol{\theta}_t + \hat{F}(\boldsymbol{\theta}_t)^{-1}\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta}_t). \tag{36}$$

Remark. It is worth pointing out that for Gaussian mixtures with $\boldsymbol{\theta} = \{\boldsymbol{\pi}_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$, the second-order Taylor approximation (31) is notably strong. Specifically, $Q(\boldsymbol{\theta}|\boldsymbol{\theta}_t)$ is strictly quadratic in the Gaussian component means $\boldsymbol{\mu}_k$, rendering the approximation in fact *exact*:

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}_t) \propto \sum_{k=1}^K \boldsymbol{\rho}_k \left(\log \boldsymbol{\pi}_k - \frac{1}{2}\log\det(\boldsymbol{\Sigma}_k) - \frac{1}{2}(x - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(x - \boldsymbol{\mu}_k)\right). \tag{37}$$

B.4 Proof of Theorem 3.4

In this section, we show that Gaussian mixtures $p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$ with $\boldsymbol{\theta} = \{\boldsymbol{\pi}_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$ has a block-diagonal $\hat{F}(\boldsymbol{\theta})$ with exactly $K+1$ blocks. We begin by writing down the complete-data log-likelihood

$$\log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) = \sum_{k=1}^K \mathbb{1}_{\{\boldsymbol{z}=k\}}[\log \boldsymbol{\pi}_k + \log \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)], \tag{38}$$

where $\mathbb{1}_{\{\boldsymbol{z}=k\}}$ is the characteristic function that returns 1 when $\boldsymbol{z} = k$ and 0 otherwise.

Note that $\log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$ can be equivalently rewritten as

$$\log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) = \sum_{k=1}^K \mathbb{1}_{\{\boldsymbol{z}=k\}} \log \boldsymbol{\pi}_k + \sum_{k=1}^K \mathbb{1}_{\{\boldsymbol{z}=k\}} \log \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \ell_{\boldsymbol{\pi}} + \sum_{k=1}^K \ell_k, \tag{39}$$

where $\ell_{\boldsymbol{\pi}} := \sum_{k=1}^K \mathbb{1}_{\{\boldsymbol{z}=k\}} \log \boldsymbol{\pi}_k = \log \mathrm{Cat}(\boldsymbol{z}; \boldsymbol{\pi})$ and $\ell_k := \mathbb{1}_{\{\boldsymbol{z}=k\}} \log \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

One can find that $\ell_{\boldsymbol{\pi}}$ is constant w.r.t. $\{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$, and ℓ_k is constant w.r.t. $\boldsymbol{\pi}$ and $\{\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\}_{j\neq k}$. Therefore, the Hessian of the complete-data log-likelihood $\log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})$ is block-diagonal.

$$\nabla^2_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta}) = \begin{bmatrix} \nabla^2_{\boldsymbol{\mu}_1,\boldsymbol{\Sigma}_1}\ell_1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \nabla^2_{\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k}\ell_k & \vdots \\ 0 & \cdots & \cdots & \nabla^2_{\boldsymbol{\pi}}\ell_{\boldsymbol{\pi}} \end{bmatrix}. \tag{40}$$

Finally, note that the complete-data FIM is defined as the negative expected Hessian of the complete-data log-likelihood such that $\hat{F}(\boldsymbol{\theta}) = -\mathbb{E}_{p(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta})}[\nabla^2_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{\theta})]$, and

$$\mathbb{E}[\nabla^2_{\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k}\ell_k] = \mathbb{E}[\mathbb{1}_{\{\boldsymbol{z}=k\}}]\cdot\mathbb{E}[\nabla^2 \log \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)] = \boldsymbol{\pi}_k \cdot -F_k, \tag{41}$$

$$\mathbb{E}[\nabla^2_{\boldsymbol{\pi}}\ell_{\boldsymbol{\pi}}] = \mathbb{E}[\nabla^2 \log \mathrm{Cat}(\boldsymbol{z};\boldsymbol{\pi})] = -F_{\boldsymbol{\pi}}, \tag{42}$$

where F_k for $k \in \{1,\dots,K\}$ is the FIM of the k-th Gaussian $\mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)$ and $F_{\boldsymbol{\pi}}$ is the FIM of the categorical distribution $\mathrm{Cat}(\boldsymbol{z};\boldsymbol{\pi})$.

Therefore, the complete-data FIM $\hat{F}(\boldsymbol{\theta})$ of Gaussian mixtures is given as follows

$$\hat{F}(\boldsymbol{\theta}) = -\mathbb{E}_{p(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta})}[\nabla^2_{\boldsymbol{\theta}} \log p(\boldsymbol{x},\boldsymbol{z}|\boldsymbol{\theta})] = \begin{bmatrix} \boldsymbol{\pi}_1 F_1 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \boldsymbol{\pi}_k F_k & \vdots \\ 0 & \cdots & \cdots & F_{\boldsymbol{\pi}} \end{bmatrix}. \tag{43}$$

B.5 Fisher Information Matrix of Gaussian Mixtures

Consider a Gaussian mixture with parameters $\boldsymbol{\theta} = \{\boldsymbol{\pi}_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$. The log density is

$$\log p(\boldsymbol{x}|\boldsymbol{\theta}) = \log \sum_{k=1}^K \boldsymbol{\pi}_k \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k). \tag{44}$$

The gradient of the log density is

$$\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) = \frac{\sum_{k=1}^K \nabla_{\boldsymbol{\theta}}[\boldsymbol{\pi}_k \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)]}{\sum_{k=1}^K \boldsymbol{\pi}_k \mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}, \tag{45}$$

where the gradient w.r.t. any parameter depends on all other parameters through the denominator.

The interdependence structure of the log density gradient results in highly complex Hessian, which is often practically intractable to evaluate. Furthermore, one needs to integrate over the mixture density $p(\boldsymbol{x}|\boldsymbol{\theta})$ to obtain the FIM, which exacerbates the computational intractability.

B.6 Fisher Information Matrix of Gaussians with Diagonal Covariances

Consider a Gaussian distribution with parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}\}$, with $\boldsymbol{\sigma}$ being the diagonal standard deviation. The log density is

$$\log p(\boldsymbol{x}|\boldsymbol{\theta}) = -\log\det(\mathrm{diag}(\boldsymbol{\sigma})) - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \mathrm{diag}(\boldsymbol{\sigma}^{-2})(\boldsymbol{x}-\boldsymbol{\mu}). \tag{46}$$

The gradient of the log density is

$$\nabla_{\boldsymbol{\mu}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) = \frac{\boldsymbol{x}-\boldsymbol{\mu}}{\boldsymbol{\sigma}^2}, \qquad \nabla_{\boldsymbol{\sigma}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) = \frac{(\boldsymbol{x}-\boldsymbol{\mu})^2}{\boldsymbol{\sigma}^3} - \frac{1}{\boldsymbol{\sigma}}. \tag{47}$$

The Hessian of the log density is

$$\begin{aligned} \nabla_{\boldsymbol{\mu}}\nabla_{\boldsymbol{\mu}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) &= -\frac{1}{\boldsymbol{\sigma}^2}, & \nabla_{\boldsymbol{\sigma}}\nabla_{\boldsymbol{\mu}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) &= -\frac{2(\boldsymbol{x}-\boldsymbol{\mu})}{\boldsymbol{\sigma}^3}, \\ \nabla_{\boldsymbol{\mu}}\nabla_{\boldsymbol{\sigma}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) &= -\frac{2(\boldsymbol{x}-\boldsymbol{\mu})}{\boldsymbol{\sigma}^3}, & \nabla_{\boldsymbol{\sigma}}\nabla_{\boldsymbol{\sigma}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) &= -\frac{3(\boldsymbol{x}-\boldsymbol{\mu})^2}{\boldsymbol{\sigma}^4} + \frac{1}{\boldsymbol{\sigma}^2}. \end{aligned} \tag{48}$$

Note that because $\mathbb{E}[\boldsymbol{x}-\boldsymbol{\mu}] = \mathbf{0}$ and $\mathbb{E}[(\boldsymbol{x}-\boldsymbol{\mu})^2] = \boldsymbol{\sigma}^2$, we have

$$F([\boldsymbol{\mu} \quad \boldsymbol{\sigma}]) = -\mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta})} \begin{bmatrix} \nabla_{\boldsymbol{\mu}}\nabla_{\boldsymbol{\mu}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) & \nabla_{\boldsymbol{\sigma}}\nabla_{\boldsymbol{\mu}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) \\ \nabla_{\boldsymbol{\mu}}\nabla_{\boldsymbol{\sigma}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) & \nabla_{\boldsymbol{\sigma}}\nabla_{\boldsymbol{\sigma}} \log p(\boldsymbol{x}|\boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} 1/\boldsymbol{\sigma}^2 & \mathbf{0} \\ \mathbf{0} & 2/\boldsymbol{\sigma}^2 \end{bmatrix}. \tag{49}$$

B.7 Fisher Information Matrix of Gaussians with Low-Rank Covariances

Consider a Gaussian distribution with parameters $\boldsymbol{\theta} = \{\boldsymbol{\mu}, \boldsymbol{\Lambda}\}$, where $\boldsymbol{\mu} \in \mathbb{R}^d$ is the mean, $\boldsymbol{\Lambda} \in \mathbb{R}^{d\times m}$ is a low-rank matrix with $m \ll d$, and $\boldsymbol{\Sigma} = \boldsymbol{\Lambda}\boldsymbol{\Lambda}^\top + \mathbf{I}$ is the covariance. The log density is

$$\log p(\boldsymbol{x}|\boldsymbol{\theta}) = -\log\det(\boldsymbol{\Sigma}) - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}). \tag{50}$$

For brevity we denote $\boldsymbol{v} := \boldsymbol{x} - \boldsymbol{\mu}$. The gradient of the log density is

$$\begin{aligned}\nabla_{\boldsymbol{\mu}}\log p(\boldsymbol{x}|\boldsymbol{\theta}) &= \boldsymbol{\Sigma}^{-1}\boldsymbol{v},\\ \nabla_{\boldsymbol{\Lambda}}\log p(\boldsymbol{x}|\boldsymbol{\theta}) &= [\boldsymbol{\Sigma}^{-1}\boldsymbol{v}\boldsymbol{v}^\top\boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}]\boldsymbol{\Lambda}.\end{aligned} \tag{51}$$

For brevity we denote $\boldsymbol{\Gamma} := \boldsymbol{\Sigma}^{-1}\boldsymbol{v}\boldsymbol{v}^\top$. The Hessian of the log density is

$$\begin{aligned}\nabla_{\boldsymbol{\mu}}\nabla_{\boldsymbol{\mu}}\log p(\boldsymbol{x}|\boldsymbol{\theta}) &= -\boldsymbol{\Sigma}^{-1},\\ \nabla_{\boldsymbol{\Lambda}}\nabla_{\boldsymbol{\mu}}\log p(\boldsymbol{x}|\boldsymbol{\theta}) &= [\boldsymbol{v}^\top\otimes\mathbf{I}]\cdot\nabla_{\boldsymbol{\Lambda}}\boldsymbol{\Sigma}^{-1},\\ \nabla_{\boldsymbol{\mu}}\nabla_{\boldsymbol{\Lambda}}\log p(\boldsymbol{x}|\boldsymbol{\theta}) &= -\boldsymbol{\Lambda}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{v}\otimes\boldsymbol{\Sigma}^{-1} - \boldsymbol{\Lambda}^\top\boldsymbol{\Sigma}^{-1}\otimes\boldsymbol{\Sigma}^{-1}\boldsymbol{v},\\ \nabla_{\boldsymbol{\Lambda}}\nabla_{\boldsymbol{\Lambda}}\log p(\boldsymbol{x}|\boldsymbol{\theta}) &= [\boldsymbol{\Lambda}^\top\boldsymbol{\Gamma}\otimes\mathbf{I}]\cdot\nabla_{\boldsymbol{\Lambda}}\boldsymbol{\Sigma}^{-1} + [\boldsymbol{\Lambda}^\top\otimes\boldsymbol{\Gamma}]\cdot\nabla_{\boldsymbol{\Lambda}}\boldsymbol{\Sigma}^{-1}\\ &\quad - [\boldsymbol{\Lambda}^\top\otimes\mathbf{I}]\cdot\nabla_{\boldsymbol{\Lambda}}\boldsymbol{\Sigma}^{-1} + \mathbf{I}\otimes[\boldsymbol{\Gamma}\boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}].\end{aligned} \tag{52}$$

where $\otimes$ is the Kronecker product such that for $\mathbf{A}\in\mathbb{R}^{a\times b}$ and $\mathbf{B}\in\mathbb{R}^{c\times d}$ we have $\mathbf{A}\otimes\mathbf{B}\in\mathbb{R}^{ac\times bd}$.

Note that the off-diagonal blocks $\nabla_{\boldsymbol{\Lambda}}\nabla_{\boldsymbol{\mu}}$ and $\nabla_{\boldsymbol{\mu}}\nabla_{\boldsymbol{\Lambda}}$ in the Hessian are linear in $\boldsymbol{v}$ with $\mathbb{E}[\boldsymbol{v}] = \mathbb{E}[\boldsymbol{x}-\boldsymbol{\mu}] = \mathbf{0}$, and that $\mathbb{E}[\boldsymbol{\Gamma}] = \mathbf{I}$ because of $\mathbb{E}[\boldsymbol{v}\boldsymbol{v}^\top] = \boldsymbol{\Sigma}$.

Furthermore, we have $\nabla_{\boldsymbol{\Lambda}}\boldsymbol{\Sigma}^{-1} = -(\boldsymbol{\Sigma}^{-1}\otimes\boldsymbol{\Sigma}^{-1})[(\boldsymbol{\Lambda}\otimes\mathbf{I}) + (\mathbf{I}\otimes\boldsymbol{\Lambda})\mathbf{K}]$, where $\mathbf{K}\in\mathbb{R}^{dm\times dm}$ is the commutation matrix such that $\mathbf{K}\mathrm{vec}(\boldsymbol{\Lambda}) = \mathrm{vec}(\boldsymbol{\Lambda}^\top)$. Therefore, the Fisher information matrix is

$$F([\boldsymbol{\mu}\quad\boldsymbol{\Lambda}]) = -\mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\theta})}\begin{bmatrix}\nabla_{\boldsymbol{\mu}}\nabla_{\boldsymbol{\mu}}\log p(\boldsymbol{x}|\boldsymbol{\theta}) & \nabla_{\boldsymbol{\Lambda}}\nabla_{\boldsymbol{\mu}}\log p(\boldsymbol{x}|\boldsymbol{\theta})\\ \nabla_{\boldsymbol{\mu}}\nabla_{\boldsymbol{\Lambda}}\log p(\boldsymbol{x}|\boldsymbol{\theta}) & \nabla_{\boldsymbol{\Lambda}}\nabla_{\boldsymbol{\Lambda}}\log p(\boldsymbol{x}|\boldsymbol{\theta})\end{bmatrix} = \begin{bmatrix}F_{\boldsymbol{\mu}} & 0\\ 0 & F_{\boldsymbol{\Lambda}}\end{bmatrix}, \tag{53}$$

where

$$F_{\boldsymbol{\mu}} = \boldsymbol{\Sigma}^{-1},\quad F_{\boldsymbol{\Lambda}} = -[\boldsymbol{\Lambda}^\top\otimes\mathbf{I}]\cdot\nabla_{\boldsymbol{\Lambda}}\boldsymbol{\Sigma}^{-1} = (\boldsymbol{\Lambda}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\Lambda}\otimes\boldsymbol{\Sigma}^{-1}) + (\boldsymbol{\Lambda}^\top\boldsymbol{\Sigma}^{-1}\otimes\boldsymbol{\Sigma}^{-1}\boldsymbol{\Lambda})\mathbf{K}. \tag{54}$$

Remark. To compute the natural gradient, one would need to invert $F_{\boldsymbol{\mu}}$ and $F_{\boldsymbol{\Lambda}}$. For $F_{\boldsymbol{\mu}}$ we have trivially $F_{\boldsymbol{\mu}}^{-1} = \boldsymbol{\Sigma}$. For $F_{\boldsymbol{\Lambda}}$, however, it is often computationally convenient to assume that

$$F_{\boldsymbol{\Lambda}} \approx \boldsymbol{\Lambda}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\Lambda}\otimes\boldsymbol{\Sigma}^{-1},\qquad F_{\boldsymbol{\Lambda}}^{-1} \approx (\boldsymbol{\Lambda}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\Lambda})^{-1}\otimes\boldsymbol{\Sigma}, \tag{55}$$

and the natural gradient can be approximately simplified as $\hat{\nabla}_{\boldsymbol{\Lambda}} = F_{\boldsymbol{\Lambda}}^{-1}\nabla_{\boldsymbol{\Lambda}} \approx \boldsymbol{\Sigma}\nabla_{\boldsymbol{\Lambda}}(\boldsymbol{\Lambda}^\top\boldsymbol{\Sigma}^{-1}\boldsymbol{\Lambda})^{-1}$. One can further simplify $\boldsymbol{\Sigma}^{-1} = (\boldsymbol{\Lambda}\boldsymbol{\Lambda}^\top + \mathbf{I})^{-1}$ using the Woodbury identity, exploiting the low-rank structure of $\boldsymbol{\Lambda}$ to reduce matrix inversions from $\mathbb{R}^{d\times d}$ to $\mathbb{R}^{m\times m}$.

B.8 Fisher Information Matrix of Categorical Distributions

Consider a categorical distribution parametrized by logits $\boldsymbol{\psi}$ with probabilities $\boldsymbol{\pi} = \mathrm{softmax}(\boldsymbol{\psi}) \in \Delta^{K-1}$. Let $\boldsymbol{x}$ be a one-hot vector describing the event outcome. The log density is

$$\log p(\boldsymbol{x}|\boldsymbol{\psi}) = \boldsymbol{x}^\top\boldsymbol{\psi} - \mathrm{logsumexp}(\boldsymbol{\psi}). \tag{56}$$

The gradient and Hessian of the log-density is respectively

$$\nabla_{\boldsymbol{\psi}}\log p(\boldsymbol{x}|\boldsymbol{\psi}) = \boldsymbol{x} - \boldsymbol{\pi},\qquad \nabla^2_{\boldsymbol{\psi}}\log p(\boldsymbol{x}|\boldsymbol{\psi}) = \boldsymbol{\pi}\boldsymbol{\pi}^\top - \mathrm{diag}(\boldsymbol{\pi}). \tag{57}$$

Therefore, we can derive the FIM

$$F(\boldsymbol{\psi}) = -\mathbb{E}_{p(\boldsymbol{x}|\boldsymbol{\psi})}[\nabla^2_{\boldsymbol{\psi}}\log p(\boldsymbol{x}|\boldsymbol{\psi})] = \mathrm{diag}(\boldsymbol{\pi}) - \boldsymbol{\pi}\boldsymbol{\pi}^\top. \tag{58}$$

Remark. However, note that $F(\boldsymbol{\psi})$ is non-invertible with null space $\mathrm{span}\{\mathbf{1}\}$. In this case, the natural gradient $\hat{\nabla}_{\boldsymbol{\psi}}$ is defined as the solutions to the linear system $F(\boldsymbol{\psi})\hat{\nabla}_{\boldsymbol{\psi}} = \nabla_{\boldsymbol{\psi}}$, where $\nabla_{\boldsymbol{\psi}} := \boldsymbol{\rho} - \boldsymbol{\pi}$.

$$\hat{\nabla}_{\boldsymbol{\psi}} := \mathrm{diag}(\boldsymbol{\pi})^{-1}(\boldsymbol{\rho} - \boldsymbol{\pi}) + c\mathbf{1} = \frac{\boldsymbol{\rho}}{\boldsymbol{\pi}} + c\mathbf{1},. \tag{59}$$

where $c \in \mathbb{R}$ and the division is element-wise.

Since the categorical distribution $p(\boldsymbol{x}|\boldsymbol{\psi})$ is invariant to constant shifts $c\mathbf{1}$ in logits, we have infinitely many possible $\hat{\nabla}_{\boldsymbol{\psi}}$ in the form of $\boldsymbol{\rho}/\boldsymbol{\pi} + c\mathbf{1}$. A natural idea is to impose an additional orthogonality condition $\mathbf{1}^\top \hat{\nabla}_{\boldsymbol{\psi}} = \mathbf{0}$ to avoid redundant updates in the null-space direction, leading to

$$\hat{\nabla}_{\boldsymbol{\psi}} := F(\boldsymbol{\psi})^\dagger \nabla_{\boldsymbol{\psi}} = \left(\mathbf{I} + \frac{1}{K}\mathbf{1}\mathbf{1}^\top\right) \mathrm{diag}(\boldsymbol{\pi})^{-1}(\boldsymbol{\rho} - \boldsymbol{\pi}), \tag{60}$$

where $F(\boldsymbol{\psi})^\dagger$ is the Moore-Penrose pseudo-inverse of $F(\boldsymbol{\psi})$, and $\mathbf{I} + \frac{1}{K}\mathbf{1}\mathbf{1}^\top$ is the centering matrix.

C Implementation

We provide a reference implementation of our approach using the JAX [7] framework.

```
import jax
import jax.numpy as jnp
from jax import Array

Dist = tuple["pi", "mu", "sigma"]

def mdn(x: Array) -> Dist:
    ...

def normal_log_pdf(y: Array, mu: Array, sigma: Array) -> Array:
    ...

@jax.custom_jvp
def precond(dist: Dist) -> Dist:
    return dist

@precond.defjvp
def _(primals: tuple[Dist], tangents: tuple[Dist]) -> tuple[Dist, Dist]:
    dist, grads = *primals, *tangents
    pi, mu, sigma = dist
    g_pi, g_mu, g_sigma = grads
    K, inv_pi = pi.shape[-1], 1.0 / (pi + 1e-5)
    # compute fisher preconditioners
    f_pi = jnp.diag(inv_pi) - (inv_pi / K)
    f_mu = std ** 2
    f_sigma = (std ** 2) / 2.0
    return dist, (f_pi @ g_pi, f_mu * g_mu, f_sigma * g_sigma)

def ngem_loss(x, y):
    pi, mu, sigma = precond(mdn(x))  # (k), (k, d), (k, d)
    # [E-step] compute responsibilities rho
    log_pi = jnp.log(pi)
    log_normal = jax.vmap(normal_log_pdf)(y[jnp.newaxis], mu, sigma)
    log_rho = jax.nn.log_softmax(log_pi + log_normal, axis=-1)
    # [M-step] natural gradient descent
    rho = jax.lax.stop_gradient(jnp.exp(log_rho))
    ratio = jax.lax.stop_gradient(jnp.exp(log_rho - log_pi))
    return -(rho * log_pi + ratio * log_normal).mean()
```

D EXPERIMENTS

D.1 MODE COLLAPSE ABLATIONS

We conduct ablation experiments to determine which parts of nGEM are most effective and responsible for mitigating mode collapse. We test the following learning objective variants: *(a)* NLL, *(b)* nGEM, *(c)* nGEM without inverse Gaussian FIM F_k^{-1}, and *(d)* nGEM without inverse weighting $\boldsymbol{\pi}_k^{-1}$. Please refer to Theorem 3.4 (§B.4, §B.6, §B.8) and Equation (14) for detailed description of the variants.

We conduct the ablations on the Two-Sinusoids example with a mixture density network $f(\boldsymbol{x}; \boldsymbol{\theta})$ specially initialized such that the last linear layer has near-zero weights $\mathbf{W} \sim \mathcal{N}(\mathbf{0}, 10^{-3}\mathbf{I})$ and zero bias $\mathbf{b} = \mathbf{0}$. Consequently, $f(\boldsymbol{x}; \boldsymbol{\theta}) \approx \mathbf{0}$ for all appropriately normalized inputs $\boldsymbol{x}$. This initialization is particularly undesirable as all initial components almost collapse into the same Gaussian, which we refer to as a catastrophic *mode collapse* initialization.

Table 5: Effectiveness of nGEM (and variants) on mode collapse mitigation ($\beta = 10^{-3}$).

Method	Negative Log-Likelihood (↓)	Entropy (↑)
NLL	1.919 ± 0.19	0.000 ± 0.00
nGEM	$\mathbf{0.342 \pm 1.12}$	$\mathbf{0.693 \pm 0.01}$
w/o F_k^{-1}	0.912 ± 0.25	0.467 ± 0.15
w/o $\boldsymbol{\pi}_k^{-1}$	1.909 ± 0.20	0.000 ± 0.00

We find that nGEM can reliably recover from such catastrophic mode collapse initializations and converge optimally for 4/5 times tested. Furthermore, Table 5 reports that *(c)* removing the inverse Gaussian FIM F_k^{-1} only mildly downgrade the convergence rates and quality, while *(d)* removing the inverse weight $1/\boldsymbol{\pi}_k$ will render nGEM completely ineffective for mitigating mode collapse. Intuitively, we can understand the mechanism of nGEM through two complementary views:

- Inverse weighting $\boldsymbol{\pi}_k^{-1}$ that lifts mode collapsed components ($\boldsymbol{\pi}_k \to 0$) out of local minima.
- Inverse Gaussian FIM F_k^{-1} that adaptively schedules the learning rate to accelerate convergence by scaling the gradient signals proportionally to the Gaussian variance $\boldsymbol{\sigma}_k^2$.

D.2 INVERSE MNIST

Figures 9 and 10 illustrates the Gaussian mixture ($K = 3$) component means predicted by mixture density networks, trained using either nGEM or NLL. We observe that nGEM can predict sharp and diverse images, while NLL suffers from mode collapse ($\boldsymbol{\pi}_k = 1$) and can only predicts blurry images.

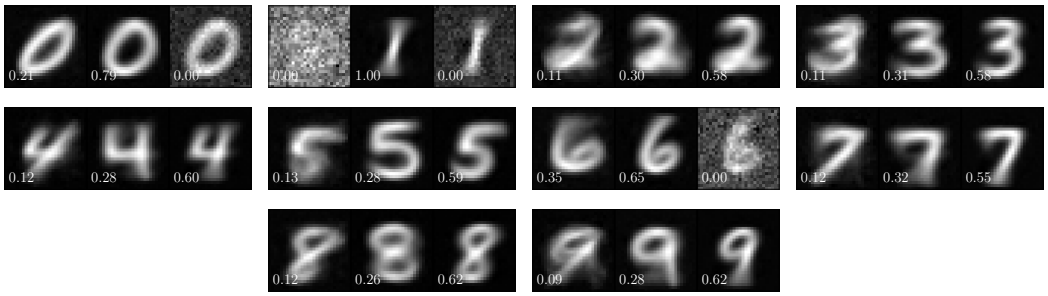


Figure 9: (nGEM) Visualizing the predicted Gaussian mixture ($K = 3$) component means $\boldsymbol{\mu}_k$ given class labels $\{0, 1, \ldots, 9\}$ for the inverse-MNIST example. Numbers to the bottom left corner of each image denote the corresponding mixture component weight $\boldsymbol{\pi}_k$.

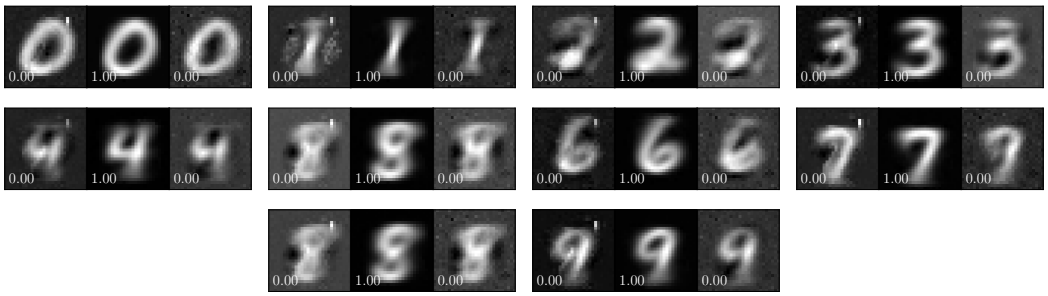


Figure 10: (NLL) Visualizing the predicted Gaussian mixture ($K = 3$) component means $\boldsymbol{\mu}_k$ given class labels $\{0, 1, \ldots, 9\}$ for the inverse-MNIST example. Numbers to the bottom left corner of each image denote the corresponding mixture component weight $\boldsymbol{\pi}_k$.

Table 6 reports the negative log-likelihoods and entropy of the learned mixture density networks on the inverse MNIST problem with $K \in \{3, 10\}$. We find that nGEM consistently outperforms NLL in modeling high-dimensional observations, even when K is considerably overparametrized.

Table 6: Negative log-likelihoods (↓) and entropy (↑) of the learned mixture density on the inverse MNIST problem with $K \in \{3, 10\}$. Results are averaged ($\pm$ std) across 5 random seeds.

	Negative Log-Likelihood (↓)		Entropy (↑)	
Method	$K = 3$	$K = 10$	$K = 3$	$K = 10$
nGEM	**-1529.7 ± 101**	**-1601.4 ± 146**	**0.454 ± 0.15**	**0.688 ± 0.00**
NLL	-1392.8 ± 178	-1470.0 ± 148	0.207 ± 0.22	0.456 ± 0.32

D.3 Simulation-based Inference

For experiments on simulation-based inference, we use the `sbijax` [15] software which provides implementations of common baselines and benchmarks.

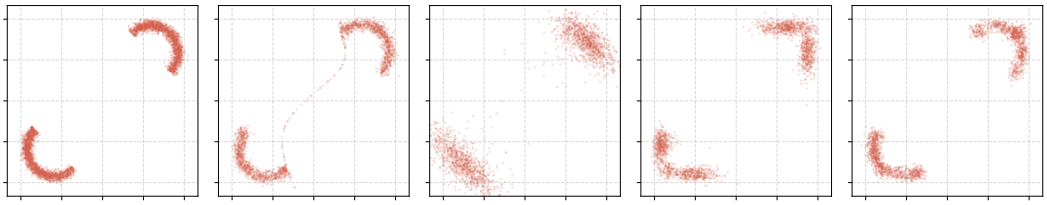
In simulation-based inference, we generally have access to a prior $p(\boldsymbol{\theta})$ and a black-box simulator (likelihood) $p(\boldsymbol{y}|\boldsymbol{\theta})$. We would like to infer the posterior $p(\boldsymbol{\theta}|\boldsymbol{y}) \propto p(\boldsymbol{y}|\boldsymbol{\theta})p(\boldsymbol{\theta})$, but regular Bayesian inference techniques could not be applied as the we do not know the analytical form of the likelihood. Instead, we generate a dataset $\{\boldsymbol{\theta}_n, \boldsymbol{y}_n\}_{n=1}^N$ by sampling $\boldsymbol{\theta}_n \sim p(\boldsymbol{\theta})$ and $\boldsymbol{y}_n \sim p(\boldsymbol{y}|\boldsymbol{\theta}_n)$ using the simulator, and then model the posterior $p(\boldsymbol{\theta}|\boldsymbol{y})$ by fitting a conditional density estimator to the dataset.

For baselines, we consider neural posterior estimation (NPE) methods including automatic posterior transformation (APT) [16] with masked autoregressive flows (MAF) [36], flow matching posterior estimation (FMPE) [42], as well as mixture density networks with NLL / nGEM.

For benchmarks, we consider Two Moons and Hyperboloid models, of which we refer the readers to `sbijax` [15] documentation for detailed model specification. We evaluate the baselines *qualitatively* by visualizing samples from the neural posterior in Figures 11 and 12. Reference posterior samples are generated using the No-U-Turn sampler (NUTS) [22] with $1,000$ warm up steps.

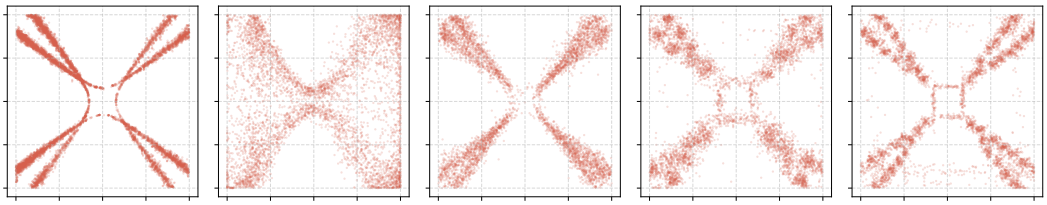
Figure 11 visualizes posterior samples of each baseline for Two Moons. The training dataset contains $N = 20,000$ samples, and Adam optimizer [24] with learning rate $\beta = 10^{-3}$ is used throughout. Note that (b) APT and (c) FMPE are trained for 500 epochs to achieve best results, while (d, e) MDNs are only trained for 50 epochs with $K = 16$ components, yet yield comparable performance. Mixture density networks with nGEM outperform all other baselines in terms of posterior sample quality.

Figure 12 visualizes posterior samples of each baseline for Hyperboloid. The training dataset contains $N = 50,000$ samples, and Adam optimizer [24] with learning rate $\beta = 10^{-3}$ is used throughout. Note that (b) APT and (c) FMPE are trained for 500 epochs, while (d, e) MDNs are only trained for 100 epochs with $K = 64$ components. Mixture density networks with nGEM generally achieve comparable (even superior) performance against other baselines in terms of posterior sample quality, and can clearly recover the complex ring and dual-arm structure in the posterior.



(a) Reference (b) APT (c) FMPE (d) MDN w/ NLL (e) MDN w/ nGEM

Figure 11: Neural posterior over $\boldsymbol{\theta} \in \mathbb{R}^2$ given $\boldsymbol{y} = \mathbf{0}$ for Two Moons.



(a) Reference (b) APT (c) FMPE (d) MDN w/ NLL (e) MDN w/ nGEM

Figure 12: Neural posterior over $\boldsymbol{\theta} \in \mathbb{R}^2$ given $\boldsymbol{y} = 0.618 \cdot \mathbf{1}$ for Hyperboloid.

Table 7 reports the Maximum Mean Discrepancy (MMD) as a quantitative performance measure. We use 10^{-3} bandwidth and 1024 samples from both the reference (NUTS) and model distributions.

Table 7: Maximum Mean Discrepancy ($\downarrow$, $\times 10^3$) of the neural posterior estimation (NPE) methods on the Two Moons and Hyperboloid problems. Results are averaged ($\pm$ std) across 5 random seeds.

Method	APT	FMPE	MDN w/ NLL	MDN w/ nGEM
Two Moons	23.58 ± 7.26	25.96 ± 9.46	18.00 ± 11.61	$\mathbf{16.85 \pm 9.45}$
Hyperboloid	2.95 ± 0.29	$\mathbf{2.85 \pm 0.33}$	2.90 ± 0.29	2.86 ± 0.30

D.4 Distributional Reinforcement Learning

For experiments on distributional reinforcement learning, we use the `stoix` [40] software which implements common RL baselines and provides integration with `gymnax` [27] environments.

For baselines, we consider the standard deep Q-network (DQN) [33], C51 [4], quantile regression (QR) DQN [12], and DQN with Gaussian mixture Q-functions [2]. We use the following distributional *temporal difference* (TD) loss for learning Gaussian mixture Q-functions:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{D}_{\mathrm{KL}}\left(R(s,a) + \gamma \sum_{s',a'} Q_{\boldsymbol{\theta}}(s',a') \,\middle\|\, Q_{\boldsymbol{\theta}}(s,a)\right), \tag{61}$$

where $Q_{\boldsymbol{\theta}}(s,a)$ is a mixture density network parametrized by $\boldsymbol{\theta}$, (s,a) and (s',a') are the current and next state-action pairs with a' selected according to a greedy policy, $R(s,a)$ is the reward function, and $\gamma \in (0,1)$ is the discount factor. We approximate the non-analytical KL divergence between two Gaussian mixtures by instead sampling from the TD target $\lfloor R(s,a) + \gamma \sum_{s',a'} Q_{\boldsymbol{\theta}}(s',a') \rfloor$.

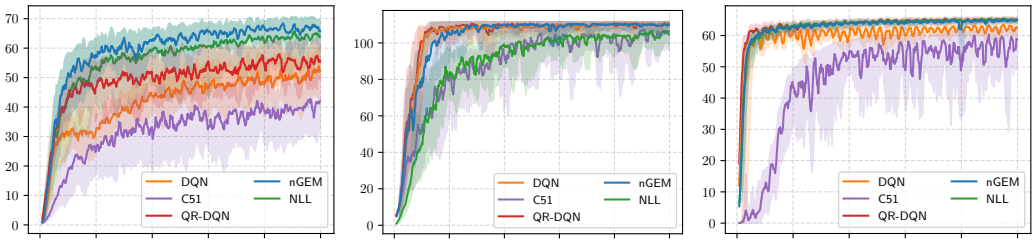
For benchmarks, we consider three discrete control tasks from `gymnax` minimal Atari games including Asterix, Breakout, and Freeway. We evaluate the performance of each baseline in each environment using discounted episode returns (averaged $\pm$ std across 5 random seeds).

Note that for all baselines and benchmarks, we use a budget of 10^8 environment interactions, an MLP with $[512, 512, 512]$ hidden layer sizes, and the Adam optimizer [24] with learning rate $\beta = 5 \times 10^{-4}$.

Table 8: Discounted episode returns (↑) of each baseline in each environment for distributional RL.

Method	Asterix	Breakout	Freeway
DQN	53.97 ± 3.27	**110.81 ± 1.70**	62.58 ± 3.54
C51	40.83 ± 8.59	105.89 ± 7.73	58.87 ± 2.43
QR-DQN	55.12 ± 7.57	109.89 ± 2.00	**65.07 ± 0.66**
GM-DQN + NLL	60.43 ± 4.48	105.55 ± 4.79	64.88 ± 0.34
GM-DQN + nGEM	**64.15 ± 2.13**	109.94 ± 4.33	64.91 ± 0.49

Table 8 reports that *(a)* nGEM consistently outperforms NLL for GM-DQN *(b)* GM-DQN with nGEM achieves comparable (occasionally better) performance against established distributional RL baselines. Figure 13 visualizes the return curves over 10^8 environment steps for each baseline and environment.



(a) Asterix (b) Breakout (c) Freeway

Figure 13: Return curves over environment interactions for distributional RL.

D.5 HYPERPARAMETERS

Tables 9 to 12 reports the hyperparameter settings for all experiments discussed in Section 4, except for the simulation-based inference and distributional reinforcement learning experiments which are discussed in §D.3 and §D.4 above.

Table 9: Hyperparameter settings for Two-Gaussians experiments.

Hyperparameter	**Value**
Number of Components (K)	2
Optimizer	SGD
Learning Rate (β)	$\{10^{-1}, 10^{-2}, 10^{-3}\}$
Epochs	50
Batch Size	1
Dataset Size	200 (2×100)
Random Seeds	$\{1, 2, 3, 4, 5\}$

Table 10: Hyperparameter settings for Two-Sinusoids experiments.

Hyperparameter	**Value**
Number of Components (K)	2
Hidden Layer Sizes	$[128, 128, 128, 128]$
Activation Function	GELU [20]
Optimizer	`Adam`
Learning Rate (β)	$\{10^{-3}, 10^{-4}, 10^{-5}\}$
Epochs	1000
Batch Size	128
Dataset Size	2000 (2×1000)
Random Seeds	$\{1, 2, 3, 4, 5\}$

Table 11: Hyperparameter settings for inverse MNIST experiments.

Hyperparameter	**Value**
Number of Components (K)	$\{3, 10, 20\}$
Hidden Layer Sizes	$[128, 128, 128, 128]$
Activation Function	GELU [20]
Optimizer	`Adam`
Learning Rate (β)	3×10^{-4}
Epochs	100
Batch Size	256
Dataset Size	60000
Random Seeds	$\{1, 2, 3, 4, 5\}$

Table 12: Hyperparameter settings for regression modeling experiments.

Hyperparameter	**Value**
Number of Components (K)	3
Hidden Layer Sizes	$[128, 128]$
Activation Function	GELU [20]
Optimizer	{`Adam`, `KFAC`, `Muon`, `Soap`}
Learning Rate (β)	{`kin8nm`: 10^{-3}, others: 10^{-4}}
Epochs	{`kin8nm`: 250, others: 125}
Batch Size	{`kin8nm`: 128, `energy`: 16, `housing`: 32}
Dataset Size	{`kin8nm`: 8192, `energy`: 768, `housing`: 506}
Random Seeds	$\{1, 2, 3, 4, 5\}$