5th Symposium on Advances in Approximate Bayesian Inference, 2023 1–14

How to Train Your FALCON: Learning Log-Concave Densities with Energy-Based Neural Networks

Alexander Lin ALIN@SEAS.HARVARD.EDU and **Demba Ba** DEMBA@SEAS.HARVARD.EDU
School of Engineering and Applied Sciences, Harvard University, Boston, MA, USA

Abstract

A classic problem within statistics is *log-concave density (LCD) estimation*, which asks for the best log-concave density that maximizes the probability of observing some input data points. Due to the non-parametric nature of this problem, current algorithms are too computationally demanding to work beyond a few (i.e. ten) dimensions. We introduce a new approach that employs energy-based neural networks to convert the non-parametric problem into a parametric one over the network weights, enabling scalable LCD estimation in high dimensions. By leveraging deep learning infrastructure (e.g. GPUs), our method can learn LCDs up to thousands of times faster than existing approaches, while requiring hundreds of times fewer parameters. We further show that our method learns informative LCDs on a real protein expression dataset with 77 dimensions, which is beyond the capabilities of current LCD estimation algorithms.

1. Introduction

Log-concave densities (LCDs) are fundamental tools for statistical data analysis. A LCD is a continuous distribution $p(\boldsymbol{x}) : \mathbb{R}^D \to \mathbb{R}$ such that its logarithm $\log p(\boldsymbol{x})$ is a concave function. This class encompasses many well-known families, such as the Gaussian, Laplace, exponential, logistic, Chi, Weibull, and Beta distributions (Bagnoli and Bergstrom, 2006). LCDs form a natural, non-parametric generalization of Gaussians, which are arguably the most widely-used densities within all of statistics. Like the Gaussian family, the LCD family is closed under marginalization, convolution, affine transformation, and taking weak limits (Saumard and Wellner, 2014). This has inspired many statistical applications – such as clustering (Cule et al., 2010), regression (Dümbgen et al., 2011), filtering (Henningsson and Åström, 2006), and independent component analysis (Samworth and Yuan, 2012) – to replace Gaussians with the more general class of LCDs for increased modeling flexibility.

Within these applications, a recurring problem is *log-concave density estimation* (Samworth, 2018): given data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^D$, find the maximum likelihood estimator (MLE)

$$p^\star = \arg\max_{p \in \mathcal{P}} \frac{1}{N} \sum_{i=1}^{N} \log p(\boldsymbol{x}_i), \tag{1}$$

where $\mathcal{P}$ is the set of all LCDs. Compared to parametric families such as the set of all Gaussians, the non-parametric nature of $\mathcal{P}$ allows $p^\star$ to flexibly adapt to the shape of the underlying data distribution, without being limited by parametric form. However, this flexibility comes at a cost: in parametric MLE problems, one can find the best estimator by simply optimizing over the relevant parameters (e.g. the mean $\boldsymbol{\mu} \in \mathbb{R}^D$ and the covariance

$\mathbf{\Sigma} \in \mathbb{R}^{D \times D}$ for Gaussians). In contrast, Eq. (1) involves searching over all distributions with a particular geometric constraint, which makes finding $p^\star$ much more challenging.

Current algorithms for LCD estimation have only been applied to low dimensional ($D \leq 10$) problems (Cule et al., 2010; Rathke and Schnörr, 2019). These algorithms directly search over the non-parametric set $\mathcal{P}$ and have heavy computational costs for large D, limiting their applicability to real-world datasets. In this paper, we introduce and study the *function approximator's log-concave (FALCON)* distribution – a new approach to LCD estimation based on deep learning. Our method parameterizes the negative log-density of an LCD using an input-convex neural network (ICNN) (Amos et al., 2017). Given data, we train this network within an energy-based model (EBM) framework (LeCun et al., 2006) to maximize log-likelihood. By turning the non-parametric problem of Eq. (1) into a parametric one over the network's weights, FALCON enables scalable LCD estimation in high dimensions.

Related Work Our work bridges the two disparate literatures on log-concave density estimation and energy-based modeling. Within statistics, several works have studied *LCD estimation* (see reviews by Walther (2009); Saumard and Wellner (2014); Samworth (2018)). Dümbgen and Rufibach (2009) and Liu and Wang (2018) studied the one-dimensional case. Cule et al. (2010) provided an exact algorithm for the multivariate case; other algorithms were later proposed by Koenker and Mizera (2010); Axelrod et al. (2019); Rathke and Schnörr (2019); Chen et al. (2021). To the best of the authors' knowledge, none of these works have applied their methods to datasets beyond a few (e.g. ten) dimensions. In machine learning, *energy-based models* have a rich history of casting maximum likelihood estimation as an energy minimization problem (see reviews by LeCun et al. (2006); Song and Kingma (2021)). Recently, there has been considerable interest in applying EBMs to computer vision (Du and Mordatch, 2019; Suhail et al., 2021), chemistry (Liu et al., 2021), natural language processing (Deng et al., 2020), and reinforcement learning (Haarnoja et al., 2017). These works typically study how well EBMs can generate realistic data after training. Our work has a different focus, illustrating how well EBMs optimize densities compared to exact, non-parametric algorithms; the setting of LCD estimation uniquely enables such comparisons because exact algorithms exist as baselines. These comparisons contribute to understanding the expressive power of EBMs, and to what degree they can be improved.

2. FALCON: Function Approximator's Log-Concave Distribution

FALCON is an energy-based model (EBM) (LeCun et al., 2006), which defines a distribution p_θ over $\mathcal{X} \subseteq \mathbb{R}^D$ by parameterizing its negative log-density (up to an additive constant) using an energy function $E_\theta : \mathcal{X} \to \mathbb{R}$. Specifically, for any $\boldsymbol{x} \in \mathcal{X}$,

$$p_\theta(\boldsymbol{x}) := \frac{\exp(-E_\theta(\boldsymbol{x}))}{Z_\theta}. \tag{2}$$

Here, $Z_\theta := \int_{\tilde{\boldsymbol{x}} \in \mathcal{X}} \exp(-E_\theta(\tilde{\boldsymbol{x}}))d\tilde{\boldsymbol{x}}$ is the normalizing constant and θ are the parameters. In LCD estimation, we want to learn a log-concave p_θ, which means enforcing convexity of E_θ.

Energy function architecture The *input-convex neural network* (ICNN) (Amos et al., 2017) is a deep learning architecture that guarantees the output is a convex function of the input. In FALCON, we use the ICNN to parameterize E_θ. In theory, the ICNN is flexible

enough to approximate any convex function to arbitrary precision (Chen et al., 2019). For an input $\boldsymbol{x}$, we define $e := E_\theta(\boldsymbol{x})$ by the following cascade of linear and non-linear operations:

$$\begin{aligned} \boldsymbol{z}_1 &= h_1(\boldsymbol{V}_0\boldsymbol{x} + \boldsymbol{b}_0), && && \textit{(input layer)} \\ \boldsymbol{z}_\ell &= h_\ell(\boldsymbol{W}_{\ell-1}\boldsymbol{z}_{\ell-1} + \boldsymbol{V}_{\ell-1}\boldsymbol{x} + \boldsymbol{b}_{\ell-1}), && 2 \le \ell \le L, && \textit{(hidden layers)} \\ e &= \boldsymbol{w}^\top \boldsymbol{z}_L + \boldsymbol{v}^\top \boldsymbol{x} + b, && && \textit{(output layer)} \end{aligned} \tag{3}$$

where $\theta := \{(\boldsymbol{V}_0, \boldsymbol{b}_0), (\boldsymbol{V}_1, \boldsymbol{W}_1, \boldsymbol{b}_1), \ldots, (\boldsymbol{V}_{L-1}, \boldsymbol{W}_{L-1}, \boldsymbol{b}_{L-1}), (\boldsymbol{w}, \boldsymbol{v}, b)\}$ are the parameters of the network and $\{h_1, \ldots, h_L\}$ are the non-linear activations.

To ensure convexity of e in $\boldsymbol{x}$, some restrictions must be placed on the architecture in Eq. (3). Observe that (a) if two functions f and g are convex, then their sum $f + g$ is also convex, and (b) if f is convex and g is convex and increasing, then $g \circ f$ is convex. Thus, we can guarantee convexity in $\boldsymbol{x}$ is maintained at every layer (including the output e) as long as (1) the inter-layer weights $\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots, \boldsymbol{W}_{L-1}, \boldsymbol{w}$ are non-negative and (2) the non-linear activations are convex and increasing functions. The first constraint can be enforced when training the network with (projected) gradient descent, and the second constraint is satisfied by some commonly used activations in deep learning (e.g. ReLU, Softplus).

By composing theoretical results on LCDs and ICNNs, we show that the FALCON family has sufficient expressive power to solve LCD estimation for any finite dataset:

Proposition 1 *Given any finite set of data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^D$, there exists a FALCON distribution parameterized by an ICNN with rectified linear (ReLU) activations that can exactly express the log-concave maximum likelihood estimator p^* of Eq. (1).*

Proof Let $\mathcal{X}$ denote the convex hull of $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$. In Theorems 1 and 2 of Cule et al. (2010), it is shown that (1) for $\boldsymbol{x} \notin \mathcal{X}$, $p^*(\boldsymbol{x}) = 0$, and (2) for $\boldsymbol{x} \in \mathcal{X}$, $-\log p^*(\boldsymbol{x})$ is piece-wise affine, where the pieces come from a finite triangulation of the convex hull. Any function that is piece-wise affine and convex can be equivalently expressed as a maximum over affine functions (Magnani and Boyd, 2009). Theorem 2 of Chen et al. (2019) provides a constructive proof of how one can set the weights of a K-layer ICNN with ReLU activations to express any function that is a maximum over K affine functions, which implies that there exists a FALCON that can exactly express $p^*(\boldsymbol{x})$ on $\mathcal{X}$. ■

3. Model Inference

Training the Network Prop. 1 shows existence of a FALCON density that can solve Eq. (1), but does not say how to find it in practice. We now describe how to train the FALCON parameters θ to optimize log-likelihood. Due to the non-convexity of the objective, we do not find the exact optimum; however, our experiments in Section 4.1 suggest that we can learn LCDs close to the MLE. Given a dataset $\mathcal{S} := \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, the objective is:

$$\max_\theta \frac{1}{N}\sum_{i=1}^N \log p_\theta(\boldsymbol{x}_i) \iff \min_\theta \left[\mathcal{L}(\theta) := \frac{1}{N}\sum_{i=1}^N E_\theta(\boldsymbol{x}_i) + \log Z_\theta\right]. \tag{4}$$

To optimize Eq. (4), we use stochastic gradient descent (SGD). The gradient is

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^N \nabla_\theta E_\theta(\boldsymbol{x}_i) + \nabla_\theta \log Z_\theta = \mathbb{E}_{\boldsymbol{x}_i \sim \mathcal{S}}[\nabla_\theta E_\theta(\boldsymbol{x}_i)] - \mathbb{E}_{\tilde{\boldsymbol{x}} \sim p_\theta}[\nabla_\theta E_\theta(\tilde{\boldsymbol{x}})], \tag{5}$$

where the second equality comes from the log-derivative identity (derivation given in Appendix A.1). Eq. (5) reveals that we can compute an unbiased estimator of the gradient using samples $\tilde{\boldsymbol{x}}$ from p_θ. The full SGD training algorithm is summarized in Appendix B.1.

Sampling from FALCON To obtain $\tilde{\boldsymbol{x}}$, we employ Markov chain Monte Carlo (MCMC), which takes the energy function E_θ and constructs an ergodic Markov chain $\tilde{\boldsymbol{x}}_0 \to \tilde{\boldsymbol{x}}_1 \to \cdots \to \tilde{\boldsymbol{x}}_T \to \cdots$ over $\mathcal{X}$ whose stationary distribution is p_θ. The specific MCMC method we use is the *Metropolis-adjusted Langevin algorithm* (MALA) (Roberts and Tweedie, 1996), which exploits gradient information $\nabla_{\tilde{\boldsymbol{x}}} E_\theta(\tilde{\boldsymbol{x}})$ to efficiently sample from p_θ. We outline T-step MALA in Algorithm B.2.

MALA is a particularly suitable choice for FALCON: due to its convex energy function, p_θ is always guaranteed to be log-concave. There is a strong line of theoretical work showing that for *log-concave densities* in particular, the number of iterations T needed for MALA to mix grows sub-linearly with the dimension D (Dwivedi et al., 2018; Chewi et al., 2021; Wu et al., 2022), implying that FALCON can be trained efficiently in high dimensions.

Estimating Log-Likelihood We previously described how to estimate $\nabla_\theta \mathcal{L}(\theta)$, but not how to compute the negative log-likelihood $\mathcal{L}(\theta)$ itself. Computing $\mathcal{L}(\theta)$ is also quite useful: (1) during training, it allows us to assess convergence of FALCON's loss, and (2) after training, it allows us to compare the model fit of FALCON with that of other models. We now introduce an unbiased estimator for the FALCON log-normalizing constant $\log Z_\theta$, which subsequently allows for estimation of $\mathcal{L}(\theta)$. Our approach comes from *path sampling* (Gelman and Meng, 1998; Rischard et al., 2018).

Consider some base distribution $q_0(\boldsymbol{x}) = \exp(-E_0(\boldsymbol{x}))/Z_0$ over the same domain $\mathcal{X}$ for which we know the normalizing constant Z_0 (e.g. the uniform distribution, the Gaussian distribution). We will compute $\log Z_\theta$ by integrating over a continuous path of distributions $q_\lambda(\boldsymbol{x})$ for $\lambda \in [0, 1]$, which starts at $q_0(\boldsymbol{x})$ and ends at $q_1(\boldsymbol{x}) := p_\theta(\boldsymbol{x})$. We define each $q_\lambda(\boldsymbol{x}) := \exp(-E_\lambda(\boldsymbol{x}))/Z_\lambda$, where $E_\lambda(\boldsymbol{x}) := (1-\lambda) \cdot E_0(\boldsymbol{x}) + \lambda \cdot E_\theta(\boldsymbol{x})$ and $Z_\lambda := \int_{\tilde{\boldsymbol{x}} \in \mathcal{X}} \exp(-E_\lambda(\tilde{\boldsymbol{x}})) d\tilde{\boldsymbol{x}}$. Then, it can be shown that (derivation in Appendix A.2)

$$\log Z_\theta = \log Z_0 + \int_0^1 \left(\frac{d}{d\lambda} \log Z_\lambda \right) d\lambda = \log Z_0 + \mathbb{E}_{\lambda \sim \text{Unif}(0,1)} \left[\mathbb{E}_{\tilde{\boldsymbol{x}} \sim q_\lambda} [E_0(\tilde{\boldsymbol{x}}) - E_\theta(\tilde{\boldsymbol{x}})] \right]. \quad (6)$$

Thus, an unbiased estimator of Eq. (6) arises from (1) sampling k values of λ, (2) running an independent MCMC chain for each λ (e.g. using MALA) to collect samples $\tilde{\boldsymbol{x}}$ from q_λ, and (3) averaging the quantity $E_0(\tilde{\boldsymbol{x}}) - E_\theta(\tilde{\boldsymbol{x}})$ over all samples. Note that since $q_1 = p_\theta$ is log-concave, as long as we choose q_0 to be log-concave, then all intermediate distributions q_λ will also be log-concave, allowing for efficient MALA sampling.

4. Experimental Results

4.1. Simulated Data

To evaluate how well FALCON can perform LCD estimation in practice, we benchmark it against two baselines: (a) LogConcDEAD, the seminal algorithm of Cule et al. (2010) which solves Eq. (1) exactly using non-smooth convex optimization, and (b) fmlogcondens, an algorithm proposed by Rathke and Schnörr (2019) that is often faster than LogConcDEAD, but is not guaranteed to find the true MLE. Both packages are implemented in R.

For each dimension D, we generate $N = 512$ data points from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ in $\mathbb{R}^D$ as the input to the LCD estimation algorithms. We use default hyperparameters for LogConcDEAD and fmlogcondens. For FALCON, the network comprises three hidden layers, each with 128 units and ReLU activations. Following Section 3, we train it using SGD for 500 gradient steps and $T = 250$ iterations of MALA per gradient step (see Appendix C.1 for further details). We implement FALCON in PyTorch (Paszke et al., 2019) to leverage automatic differentiation and GPU computation (our code is available at https://github.com/al5250/falcon). Our machine has a 3.5 GHz 16-core Intel i9 CPU and a Nvidia GeForce RTX 3090 GPU.

The results for different dimensions $D = 2, 3, \ldots, 8$, are displayed in Table 1. We make the following observations: (1) The three-layer FALCON learns LCDs with (negative) log-likelihood that has $< 5\%$ error compared to LogConcDEAD and fmlogcondens, (2) with GPU acceleration, FALCON has low time cost across all D, and can be up to 1,900× faster than LogConcDEAD and 120× faster than fmlogcondens, (3) FALCON uses up to 290× fewer parameters than LogConcDEAD and up to 750× fewer parameters than fmlogcondens.

	LogConcDEAD			fmlogcondens			FALCON		
D	**NLL**	**Time**	**Params**	**NLL**	**Time**	**Params**	**NLL**	**Time**	**Params**
2	2.79	0.1	2,268	2.79	0.0	1,338	2.79	2.3	34,307
3	4.05	0.4	13,996	4.05	0.1	13,136	4.14	2.3	34,820
4	5.22	2.0	72,045	5.22	0.4	72,015	5.43	2.3	35,333
5	6.24	4.1	373,122	6.24	1.3	372,978	6.54	2.3	35,846
6	7.08	90.2	1,993,271	7.15	2.5	1,753,010	7.42	2.3	36,359
7	7.83	4,571.5	10,723,640	8.07	16.2	481,936	8.09	2.4	36,872
8	—	—	—	8.69	397.2	28,244,700	8.79	2.4	37,385

Table 1: Comparing the negative log-likelihood, computation time in minutes, and number of parameters of different LCD estimation algorithms. FALCON estimates NLL using Eq. (6); all estimates have ≤ 0.05 standard deviation (see Appendix C.1).

4.2. Real Protein Expression Data

Next, we evaluate FALCON on a real protein expression dataset (Higuera et al., 2015) from the UCI machine learning repository. The dataset measures $D = 77$ proteins across $N = 552$ independent mice samples. Using previous algorithms, it is computationally infeasible to perform LCD estimation on data with this dimensionality. However, we show that with FALCON, we can learn an LCD for this dataset in an efficient manner and furthermore, we can apply the learned LCD to solve statistical problems, such as *missing data imputation*.

We split the dataset in half to form a training set and an evaluation set. Following Section 3, we fit a three-layer FALCON distribution $p_\theta(\boldsymbol{x})$ to the training set. Appendix C.2 provides more details on data processing and model training. For each data point $\boldsymbol{x}_i \in \mathbb{R}^D$ in the evaluation set, we independently mask each of its D measurements with probability $\alpha \in [0, 1]$ to create a missing part $\boldsymbol{x}_i^m \in \mathbb{R}^{M_i}$ and an observed part $\boldsymbol{x}_i^o \in \mathbb{R}^{D-M_i}$. To evaluate how well p_θ captures the true joint density of $\boldsymbol{x}$, we see if it can accurately impute each $\boldsymbol{x}_i^m$ using the conditional distribution $p_\theta(\boldsymbol{x}_i^m | \boldsymbol{x}_i^o)$, where θ was learned on the

training data. Note that $p_\theta(\boldsymbol{x}_i^m|\boldsymbol{x}_i^o)$ is log-concave and we can draw samples from it using MALA (see Appendix C.2). We consider the following two tasks.

Task 1: Point estimate prediction Let $\dot{\boldsymbol{x}}_i^m$ denote the true value of $\boldsymbol{x}_i^m$ from the evaluation set. For FALCON's prediction, we use the point estimate $\hat{\boldsymbol{x}}_i^m := \mathbb{E}_{p_\theta(\boldsymbol{x}_i^m|\boldsymbol{x}_i^o)}[\boldsymbol{x}_i^m]$, which is approximated by a mean over the MALA samples. Over the entire evaluation set, we compute the root mean squared error (RMSE) $(\frac{1}{M}\sum_i \|\dot{\boldsymbol{x}}_i^m - \hat{\boldsymbol{x}}_i^m\|_2^2)^{1/2}$, where $M := \sum_i M_i$ is the total number of missing values. We compare FALCON against other popular algorithms, i.e. mean value imputation (MVI), multivariate Gaussian imputation (MGI), iterative chained equations (ICE), and k-nearest neighbors (KNN) (see Appendix C.2 for descriptions). The results are in Table 2. We observe that FALCON outperforms the other methods, as its non-parametric nature allows it to bend to the shape of the underlying data.

	MVI	**MGI**	**ICE**	**KNN**	**FALCON**
$\alpha = 0.1$	0.970 ±0.014	0.347 ±0.015	0.370 ±0.013	0.409 ±0.018	**0.322 ±0.014**
$\alpha = 0.3$	0.964 ±0.008	0.370 ±0.006	0.473 ±0.009	0.407 ±0.005	**0.339 ±0.005**
$\alpha = 0.5$	0.971 ±0.007	0.407 ±0.008	0.611 ±0.009	0.425 ±0.010	**0.382 ±0.008**

Table 2: RMSE of imputation algorithms for different missing probabilities α on the protein expression dataset. Means and standard deviations are computed over five runs.

Task 2: Uncertainty quantification Beyond point estimates, FALCON can also provide *credible intervals* for the missing values. Given a particular dimension x_{id}^m of the missing data vector $\boldsymbol{x}_i^m$ and some desired confidence level $\gamma \in [0, 1]$, we can construct an interval $[a_{id}, b_{id}]$ such that the trained FALCON distribution believes $p_\theta(a_{id} \leq x_{id}^m \leq b_{id} \mid \boldsymbol{x}_i^o) = \gamma$. In practice, we estimate a_{id} and b_{id} by sorting the MALA samples of $x_{id}^m \mid \boldsymbol{x}_i^o$, collecting the middle γ-percentile samples, and setting a_{id} and b_{id} as the minimum and maximum values of this collection. Of course, the credible interval is only useful if it is *calibrated*, i.e. if the actual empirical probability δ that the true value $\dot{x}_{id}^m$ falls between a_{id} and b_{id} is indeed equal to γ. We therefore define the calibration error as $|\gamma - \delta|$. In Figure 1, we plot FALCON's calibration error for various values of $\gamma \in \{0.1, 0.2, \ldots, 1.0\}$. For comparison, we construct and evaluate similar credible intervals based on multivariate Gaussian imputation. We find that FALCON imputation has better calibration than MGI at all levels of γ. This is due to FALCON's superior flexibility as a density estimator, which we showcase in Appendix C.2.

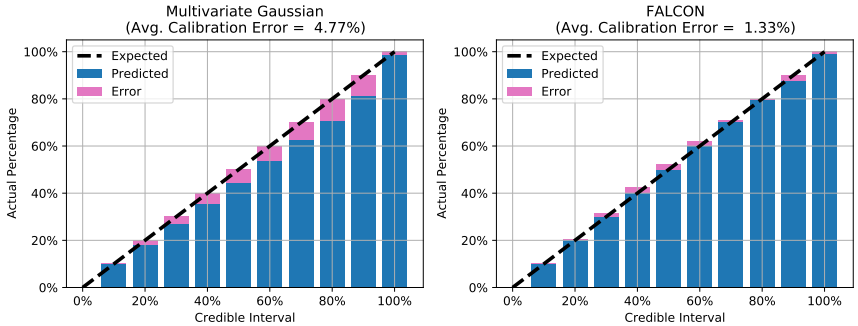

Figure 1: Actual percentage δ vs. credible interval γ for uncertainty quantification.

Acknowledgments

This work was supported by a National Defense Science and Engineering Graduate Fellowship, and the National Science Foundation under Cooperative Agreement PHY-2019786. The authors thank Pierre E. Jacob for helpful discussions in the early stages of this work. The authors also thank the anonymous reviewers for their helpful commentary.

References

Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *International Conference on Machine Learning*, pages 146–155. PMLR, 2017.

Brian Axelrod, Ilias Diakonikolas, Alistair Stewart, Anastasios Sidiropoulos, and Gregory Valiant. A polynomial time algorithm for log-concave maximum likelihood via locally exponential families. *Advances in Neural Information Processing Systems*, 32, 2019.

Mark Bagnoli and Ted Bergstrom. Log-concave probability and its applications. In *Rationality and Equilibrium: A Symposium in Honor of Marcel K. Richter*, pages 217–241. Springer, 2006.

Wenyu Chen, Rahul Mazumder, and Richard J Samworth. A new computational framework for log-concave density estimation. *arXiv preprint arXiv:2105.11387*, 2021.

Yize Chen, Yuanyuan Shi, and Baosen Zhang. Optimal control via neural networks: A convex approach. In *International Conference on Learning Representations*, 2019.

Sinho Chewi, Chen Lu, Kwangjun Ahn, Xiang Cheng, Thibaut Le Gouic, and Philippe Rigollet. Optimal dimension dependence of the metropolis-adjusted langevin algorithm. In *Conference on Learning Theory*, pages 1260–1300. PMLR, 2021.

Madeleine Cule, Richard Samworth, and Michael Stewart. Maximum likelihood estimation of a multi-dimensional log-concave density. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 72(5):545–607, 2010.

Yuntian Deng, Anton Bakhtin, Myle Ott, Arthur Szlam, and Marc'Aurelio Ranzato. Residual energy-based models for text generation. In *International Conference on Learning Representations*, 2020.

Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. *Advances in Neural Information Processing Systems*, 32, 2019.

Lutz Dümbgen and Kaspar Rufibach. Maximum likelihood estimation of a log-concave density and its distribution function: Basic properties and uniform consistency. 2009.

Lutz Dümbgen, Richard Samworth, and Dominic Schuhmacher. Approximation by log-concave distributions, with applications to regression. *The Annals of Statistics*, pages 702–730, 2011.

Raaz Dwivedi, Yuansi Chen, Martin J Wainwright, and Bin Yu. Log-concave sampling: Metropolis-hastings algorithms are fast! In *Conference on learning theory*, pages 793–797. PMLR, 2018.

Andrew Gelman and Xiao-Li Meng. Simulating normalizing constants: From importance sampling to bridge sampling to path sampling. *Statistical science*, pages 163–185, 1998.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*, pages 1352–1361. PMLR, 2017.

Toivo Henningsson and Karl Johan Åström. Log-concave observers. In *Proc. of Mathematical Theory of Networks and Systems*, 2006.

Clara Higuera, Katheleen J Gardiner, and Krzysztof J Cios. Self-organizing feature maps identify proteins critical to learning in a mouse model of down syndrome. *PloS one*, 10 (6):e0129126, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Roger Koenker and Ivan Mizera. Quasi-concave density estimation. *The Annals of Statistics*, pages 2998–3027, 2010.

Yann LeCun, Sumit Chopra, and Raia Hadsell. A tutorial on energy-based learning. 2006.

Katherine J Lee and John B Carlin. Multiple imputation for missing data: fully conditional specification versus multivariate normal imputation. *American journal of epidemiology*, 171(5):624–632, 2010.

Meng Liu, Keqiang Yan, Bora Oztekin, and Shuiwang Ji. Graphebm: Molecular graph generation with energy-based models. In *Energy Based Models Workshop-ICLR 2021*, 2021.

Yu Liu and Yong Wang. A fast algorithm for univariate log-concave density estimation. *Australian & New Zealand Journal of Statistics*, 60(2):258–275, 2018.

Alessandro Magnani and Stephen P Boyd. Convex piecewise-linear fitting. *Optimization and Engineering*, 10:1–17, 2009.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Fabian Rathke and Christoph Schnörr. Fast multivariate log-concave density estimation. *Computational Statistics & Data Analysis*, 140:41–58, 2019.

Maxime Rischard, Pierre E Jacob, and Natesh Pillai. Unbiased estimation of log normalizing constants with applications to bayesian cross-validation. *arXiv preprint arXiv:1810.01382*, 2018.

Gareth O Roberts and Richard L Tweedie. Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli*, pages 341–363, 1996.

Richard J Samworth. Recent progress in log-concave density estimation. *Statistical Science*, 33(4):493–509, 2018.

Richard J Samworth and Ming Yuan. Independent component analysis via nonparametric maximum likelihood estimation. 2012.

Adrien Saumard and Jon A Wellner. Log-concavity and strong log-concavity: a review. *Statistics surveys*, 8:45, 2014.

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

Mohammed Suhail, Abhay Mittal, Behjat Siddiquie, Chris Broaddus, Jayan Eledath, Gerard Medioni, and Leonid Sigal. Energy-based learning for scene graph generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13936–13945, 2021.

Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B Altman. Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525, 2001.

Stef Van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of statistical software*, 45:1–67, 2011.

Guenther Walther. Inference and modeling with log-concave distributions. *Statistical Science*, pages 319–327, 2009.

Keru Wu, Scott Schmidler, and Yuansi Chen. Minimax mixing time of the metropolis-adjusted langevin algorithm for log-concave sampling. *The Journal of Machine Learning Research*, 23(1):12348–12410, 2022.

Appendix A. Derivations

A.1. Log-Derivative Identity (Eq. (5))

This equation boils down to showing $\nabla_\theta \log Z_\theta = -\mathbb{E}_{\tilde{\boldsymbol{x}}\sim p_\theta}[\nabla_\theta E_\theta(\tilde{\boldsymbol{x}})]$. We have

$$\nabla_\theta \log Z_\theta = \frac{1}{Z_\theta} \cdot \nabla_\theta \left[\int_{\tilde{\boldsymbol{x}}\in\mathcal{X}} \exp(-E_\theta(\tilde{\boldsymbol{x}})) d\tilde{\boldsymbol{x}} \right] \tag{7}$$

$$= \frac{1}{Z_\theta} \cdot \left[\int_{\tilde{\boldsymbol{x}}\in\mathcal{X}} \exp(-E_\theta(\tilde{\boldsymbol{x}})) \cdot \nabla_\theta[-E_\theta(\tilde{\boldsymbol{x}})] d\tilde{\boldsymbol{x}} \right] = -\mathbb{E}_{\tilde{\boldsymbol{x}}\sim p_\theta}[\nabla_\theta E_\theta(\tilde{\boldsymbol{x}})]. \tag{8}$$

A.2. Path Sampling Identity (Eq. (6))

It follows that

$$\log Z_\theta = \log Z_1 = \log Z_0 + (\log Z_1 - \log Z_0) = \log Z_0 + \int_0^1 \left(\frac{d}{d\lambda} \log Z_\lambda \right) d\lambda \tag{9}$$

$$= \log Z_0 + \int_0^1 \frac{1}{Z_\lambda} \left(\frac{d}{d\lambda} \int_{\tilde{\boldsymbol{x}}\in\mathcal{X}} \exp(-E_\lambda(\tilde{\boldsymbol{x}})) d\tilde{\boldsymbol{x}} \right) d\lambda \tag{10}$$

$$= \log Z_0 + \int_0^1 \frac{1}{Z_\lambda} \left(\int_{\tilde{\boldsymbol{x}}\in\mathcal{X}} \exp(-E_\lambda(\tilde{\boldsymbol{x}})) \cdot \left[\frac{d}{d\lambda}(-E_\lambda(\tilde{\boldsymbol{x}})) \right] d\tilde{\boldsymbol{x}} \right) d\lambda \tag{11}$$

$$= \log Z_0 + \int_0^1 \frac{1}{Z_\lambda} \left(\int_{\tilde{\boldsymbol{x}}\in\mathcal{X}} \exp(-E_\lambda(\tilde{\boldsymbol{x}})) \cdot [E_0(\tilde{\boldsymbol{x}}) - E_\theta(\tilde{\boldsymbol{x}})] d\tilde{\boldsymbol{x}} \right) d\lambda \tag{12}$$

$$= \log Z_0 + \mathbb{E}_{\lambda\sim \mathrm{Unif}(0,1)} \left[\mathbb{E}_{\tilde{\boldsymbol{x}}\sim q_\lambda}[E_0(\tilde{\boldsymbol{x}}) - E_\theta(\tilde{\boldsymbol{x}})] \right]. \tag{13}$$

Appendix B. Algorithms

B.1. Training FALCON with Stochastic Gradient Descent

Algorithm 1: FALCON Training Algorithm

Input: dataset $\mathcal{S} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, number of iterations J, step size η, minibatch size b
Output: trained parameters θ
Initialize $\theta := \{(\boldsymbol{V}_0, \boldsymbol{b}_0), (\boldsymbol{V}_1, \boldsymbol{W}_1, \boldsymbol{b}_1), \ldots, (\boldsymbol{V}_{L-1}, \boldsymbol{W}_{L-1}, \boldsymbol{b}_{L-1}), (\boldsymbol{v}, \boldsymbol{w}, b)\}$
for $j \leftarrow 1$ **to** J **do**
  Sample mini-batch $\mathcal{B} \subseteq \mathcal{S}$, where $|\mathcal{B}| = b$
  $\mathcal{C} \leftarrow \texttt{MCMC}(E_\theta, \text{steps} = T)$ *// Collect MCMC samples from model*
  $g \leftarrow \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x}_i\in\mathcal{B}} \nabla_\theta E_\theta(\boldsymbol{x}_i) - \frac{1}{|\mathcal{C}|} \sum_{\tilde{\boldsymbol{x}}\in\mathcal{C}} \nabla_\theta E_\theta(\tilde{\boldsymbol{x}})$ *// Estimate gradient*
  $\theta \leftarrow \theta - \eta \cdot g$ *// Update parameters*
  for $\ell \leftarrow 1$ **to** $L-1$ **do**
    $\boldsymbol{W}_\ell \leftarrow \mathrm{ReLU}(\boldsymbol{W}_\ell)$ *// Enforce convexity*
  end
  $\boldsymbol{w} \leftarrow \mathrm{ReLU}(\boldsymbol{w})$ *// Enforce convexity*
end

B.2. Drawing Samples from FALCON with MALA

Algorithm 2: Metropolis-Adjusted Langevin Algorithm (MALA)

Input: energy function $E : \mathcal{X} \to \mathbb{R}$, support $\mathcal{X} \subseteq \mathbb{R}^D$, number of steps T, burnin U, initial sample $\tilde{\boldsymbol{x}}_0$, proposal standard deviation σ

Output: set of samples $\mathcal{C}$

Initialize $\mathcal{C} \leftarrow \emptyset$

$\tilde{\boldsymbol{\mu}}_0 \leftarrow \tilde{\boldsymbol{x}}_0 - \frac{1}{2}\sigma^2 \cdot \nabla_{\tilde{\boldsymbol{x}}_0} E(\tilde{\boldsymbol{x}}_0)$ *// Compute mean proposal*

for $t \leftarrow 1$ **to** T **do**

- $\boldsymbol{\varepsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ *// Generate noise*
- $\tilde{\boldsymbol{y}}_t \leftarrow \tilde{\boldsymbol{\mu}}_{t-1} + \sigma \cdot \boldsymbol{\varepsilon}_t$ *// Propose candidate sample*
- $\tilde{\boldsymbol{\nu}}_t \leftarrow \tilde{\boldsymbol{y}}_t - \frac{1}{2}\sigma^2 \cdot \nabla_{\tilde{\boldsymbol{y}}_t} E(\tilde{\boldsymbol{y}}_t)$ *// Compute candidate mean proposal*
- $r \leftarrow \min\left[1, \frac{\exp\left(-E(\tilde{\boldsymbol{y}}_t) - \frac{1}{2\sigma^2}\|\tilde{\boldsymbol{x}}_{t-1} - \tilde{\boldsymbol{\nu}}_t\|_2^2\right)\right)}{\exp\left(-E(\tilde{\boldsymbol{x}}_{t-1}) - \frac{1}{2\sigma^2}\|\tilde{\boldsymbol{y}}_t - \tilde{\boldsymbol{\mu}}_{t-1}\|_2^2\right)\right)}\right]$ *// Compute acceptance prob*
- $\delta \sim \text{Uniform}(0, 1)$
- **if** $\tilde{\boldsymbol{y}}_t \in \mathcal{X}$ *and* $\delta \leq r$ **then**
  - $\tilde{\boldsymbol{x}}_t \leftarrow \tilde{\boldsymbol{y}}_t$ and $\tilde{\boldsymbol{\mu}}_t \leftarrow \tilde{\boldsymbol{\nu}}_t$ *// Accept proposal*
- **else**
  - $\tilde{\boldsymbol{x}}_t \leftarrow \tilde{\boldsymbol{x}}_{t-1}$ and $\tilde{\boldsymbol{\mu}}_t \leftarrow \tilde{\boldsymbol{\mu}}_{t-1}$ *// Reject proposal*
- **end**
- **if** $t \geq U$ **then**
  - $\mathcal{C} \leftarrow \mathcal{C} \cup \{\tilde{\boldsymbol{x}}_t\}$ *// Add sample to final set*
- **end**

end

Appendix C. Experimental Details

C.1. Simulated Data

Architecture For every dimension D, we construct a FALCON architecture with 3 hidden layers, 128 units per hidden layer, and rectified linear activations. From Theorem 1 and 2 of Cule et al. (2010), we know that the optimal LCD satisfies $p^*(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \notin \mathcal{X}$, where $\mathcal{X}$ is the convex hull of the data. Thus, we precompute $\mathcal{X}$ using the QuickHull algorithm (see https://scipy.github.io/devdocs/reference/generated/scipy.spatial.ConvexHull.html) and only use the ICNN to define $E_\theta(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathcal{X}$ (otherwise, we let $E_\theta(\boldsymbol{x}) = \infty$).

Training We train FALCON for $J = 500$ gradient steps using full batches (i.e. $b = N = 512$) and the Adam optimizer with learning rate 0.001 (Kingma and Ba, 2014). We use gradient clipping with max L_2-norm value of 5. For each MALA step, we run 20 chains in parallel for $T = 250$ steps and a burnin of $U = 100$. Samples are aggregated across all chains to make the FALCON gradient update. The MALA proposal standard deviation is $\sigma = 0.25$. For the first gradient update, we initialize the MALA chain with $\tilde{\boldsymbol{x}}_0 = \mathbf{0}$. For subsequent updates, we initialize each $\tilde{\boldsymbol{x}}_0$ by uniformly sampling over $\tilde{\boldsymbol{x}}_T$ from the different

chains in the previous gradient update; this is an attempt to accelerate mixing of MALA through warm start (see e.g. Dwivedi et al. (2018); Chewi et al. (2021); Wu et al. (2022)).

Since we are using MALA to sample from a distribution p_θ defined over $\mathcal{X}$, we need to be able to check if a proposed point $\tilde{\boldsymbol{y}}$ lies in $\mathcal{X}$ (if not, it is automatically rejected – see Appendix B.2). This check is easy for convex hulls, because any convex hull $\mathcal{X}$ can be defined by the set $\{\tilde{\boldsymbol{y}} | \boldsymbol{A}\tilde{\boldsymbol{y}} \leq \boldsymbol{b}\}$, for some $\boldsymbol{A} \in \mathbb{R}^{P \times D}$ and $\boldsymbol{b} \in \mathbb{R}^P$, where P is the number of faces. The parameters $\boldsymbol{A}, \boldsymbol{b}$ can be easily obtained from the QuickHull algorithm.

NLL Estimation To estimate the FALCON log-normalizing constant $\log Z_\theta$ (Eq. (6)), we use a base distribution $q_0(\boldsymbol{x}) = 1/V$ that is uniform over the convex hull $\mathcal{X}$. Here, V is the volume of the hull, which can be calculated by QuickHull. Note that this is also a log-concave distribution, implying that all intermediate q_λ are also log-concave. We use 20 values of λ, which corresponds to 20 MALA chains and for each chain, we use the same settings as used for FALCON training. For the NLL values in Table 1, we estimate $\log Z$ ten times using the aforementioned process and take the mean. The standard deviation over the ten runs is consistently less than 0.05 for all values of D.

C.2. Real Protein Expression Data

Data Processing We downloaded the data from https://archive.ics.uci.edu/ml/datasets/Mice+Protein+Expression. We removed data points with any missing values, leaving $N = 552$ samples. We randomly split these samples in half to form a training set and an evaluation set. Each dimension of the training set was independently standardized to have zero mean and one standard deviation. The standardization inferred from the training set was then applied to the evaluation set.

FALCON Details We trained a three-layer FALCON on the training set. Each hidden layer had 512 hidden units with ReLU activations. We trained for 250 epochs (i.e. passes through the full training set) with minibatch size $b = 128$, the Adam optimizer, and learning rate 0.001. The MALA sampler ran 10 chains in parallel for $T = 2{,}000$ steps with a burnin of $U = 500$. The proposal standard deviation was $\sigma = 0.1$. Unlike for the simulations, we did not limit p_θ to be non-zero over only the convex hull of the data. This was for two reasons: (1) computing convex hulls in high dimensions (i.e. $D = 77$) is computationally expensive, and (2) it is highly likely that some points in the evaluation set will not lie in the convex hull of the training set. Since we will use the network for missing data imputation, we want it to learn an informative density over all of $\mathcal{X} := \mathbb{R}^D$, and not just the convex hull of the training data.

Conditional Distribution for Missing Data Let $M \leq D$ be the dimension of the missing data $\boldsymbol{x}^m$. For missing data imputation, we want to sample from the distribution $p_\theta(\boldsymbol{x}^m | \boldsymbol{x}^o)$, which can be expressed as

$$p_\theta(\boldsymbol{x}^m | \boldsymbol{x}^o) = \frac{p_\theta(\boldsymbol{x}^m, \boldsymbol{x}^o)}{\int_{\tilde{\boldsymbol{x}}^m \in \mathbb{R}^M} p_\theta(\tilde{\boldsymbol{x}}^m, \boldsymbol{x}^o) d\tilde{\boldsymbol{x}}^m} = \frac{\exp(-E_\theta(\boldsymbol{x}^m, \boldsymbol{x}^o))}{Z_\theta \cdot \int_{\tilde{\boldsymbol{x}}^m \in \mathbb{R}^M} p_\theta(\tilde{\boldsymbol{x}}^m, \boldsymbol{x}^o) d\tilde{\boldsymbol{x}}^m}. \tag{14}$$

Thus, the energy function of $p_\theta(\boldsymbol{x}^m | \boldsymbol{x}^o)$ is $F_{\theta, \boldsymbol{x}^o}(\boldsymbol{x}^m) := E_\theta(\boldsymbol{x}^m, \boldsymbol{x}^o)$. Observe that F is certainly convex in $\boldsymbol{x}^m$, meaning $p_\theta(\boldsymbol{x}^m | \boldsymbol{x}^o)$ is a log-concave density. Given $\boldsymbol{x}^o$ and θ, we can use $F_{\theta, \boldsymbol{x}^o}$ as input to MALA (Appendix B.2) to produce samples $\tilde{\boldsymbol{x}}^m | \boldsymbol{x}^o$. These samples

can then be used to impute the missing data $\boldsymbol{x}^m$ (either through point estimation as in Task 1, or credible intervals as in Task 2).

Descriptions of Imputation Baselines Here, we briefly summarize the popular imputation algorithms that we compare against FALCON.

Mean value imputation (MVI): This is perhaps the simplest imputation algorithm. For each feature $x_d \in \boldsymbol{x}$, we take a simple mean over its values in the dataset. This mean is used to impute every instance where $x_{(d)}$ is missing from a data point.

Multivariate Gaussian imputation (MGI): This method fits a multivariate Gaussian distribution to the data to capture correlations between all D dimensions (Lee and Carlin, 2010). The learned parameters are the mean $\boldsymbol{\mu} \in \mathbb{R}^D$ and the covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{D \times D}$. Let $\boldsymbol{\mu}^o \in \mathbb{R}^{D-M}$ and $\boldsymbol{\mu}^m \in \mathbb{R}^M$ be the observed and missing dimensions of $\boldsymbol{\mu}$. Similarly, let $\boldsymbol{\Sigma}^{oo} \in \mathbb{R}^{(D-M)\times(D-M)}$ be the covariance of the observed data, $\boldsymbol{\Sigma}^{mm} \in \mathbb{R}^{M \times M}$ be the covariance of the missing data, and $\boldsymbol{\Sigma}^{mo} \in \mathbb{R}^{M \times O}$ be the cross-covariance between missing and observed data. The distribution $p_{\boldsymbol{\mu},\boldsymbol{\Sigma}}(\boldsymbol{x}^m|\boldsymbol{x}^o)$ can be used to infer the missing data. This is also a Gaussian distribution with mean $\boldsymbol{\mu}^{m|o} \in \mathbb{R}^M$ and covariance $\boldsymbol{\Sigma}^{m|o} \in \mathbb{R}^{M \times M}$, given by

$$\boldsymbol{\mu}^{m|o} := \boldsymbol{\mu}^m + \boldsymbol{\Sigma}^{mo}(\boldsymbol{\Sigma}^{oo})^{-1}(\boldsymbol{x}^o - \boldsymbol{\mu}^o), \tag{15}$$

$$\boldsymbol{\Sigma}^{m|o} := \boldsymbol{\Sigma}^{mm} - \boldsymbol{\Sigma}^{mo}(\boldsymbol{\Sigma}^{oo})^{-1}(\boldsymbol{\Sigma}^{mo})^\top. \tag{16}$$

For Task 1 (Table 2), MGI returns the point estimate $\boldsymbol{\mu}^{m|o}$ to impute $\boldsymbol{x}^m$. For Task 2 (Figure 1), MGI constructs γ-credible intervals for each dimension x_d^m by plugging the mean $\mu_d^{m|o}$ and variance $\Sigma_{d,d}^{m|o}$ corresponding to dimension d into the cumulative distribution function of a Gaussian random variable.

Iterative chained equations (ICE): This method iteratively builds a predictive regression model for each missing dimension by using the other dimensions as covariates (Van Buuren and Groothuis-Oudshoorn, 2011). We use the sci-kit learn software (`https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html`).

k-nearest neighbors (KNN): This method uses the observed data $\boldsymbol{x}^o$ to find the closest neighbors and then uses the corresponding dimensions the neighbors to impute the missing data $\boldsymbol{x}^m$ (Troyanskaya et al., 2001). In our experiments, we tune $k \in \{1, 3, 5, 7, 9\}$. We use the sci-kit learn software (`https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html#sklearn.impute.KNNImputer`).

FALCON vs. Multivariate Gaussian for Density Estimation We give some more context for FALCON's superiority in uncertainty quantification over the multivariate Gaussian in Figure 1. Specifically, in Figure 2, we show a couple of examples of how FALCON can flexibly adapt to the shape of the underlying data distribution (similar to non-parametric methods), while the multivariate Gaussian is restricted by its parametric form.

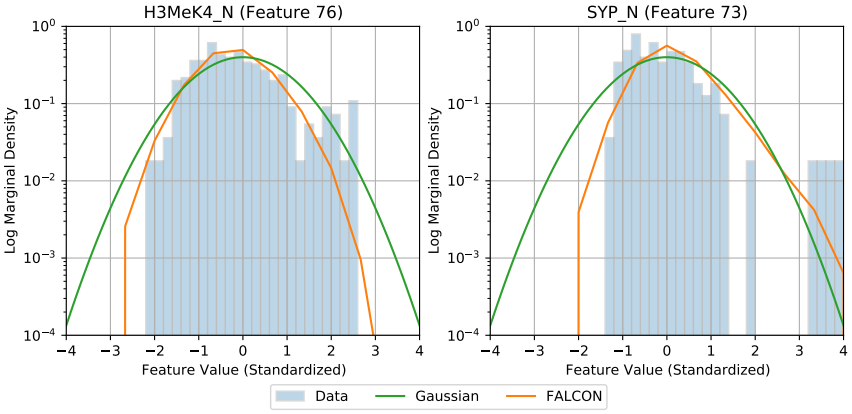

Figure 2: Comparing log-density fits of multivariate Gaussian vs. FALCON for two dimensions of the protein dataset. (Left) FALCON is able to capture the varying curvature of the data; the multivariate Gaussian is forced to have constant curvature (in log-space) due to parametric form. (Right) FALCON is able to capture the asymmetry of the data; the multivariate Gaussian is forced to be symmetric due to parametric form.