Learning Deep Kernels for Exponential Family Densities

Li K. Wenliang [*] [1] **Dougal J. Sutherland** [*] [1] **Heiko Strathmann** [1] **Arthur Gretton** [1]

Abstract

The kernel exponential family is a rich class of distributions, which can be fit efficiently and with statistical guarantees by score matching. Being required to choose *a priori* a simple kernel such as the Gaussian, however, limits its practical applicability. We provide a scheme for learning a kernel parameterized by a deep network, which can find complex location-dependent features of the local data geometry. This gives a very rich class of density models, capable of fitting complex structures on moderate-dimensional problems. Compared to deep density models fit via maximum likelihood, our approach provides a complementary set of strengths and tradeoffs: in empirical studies, deep maximum-likelihood models can yield higher likelihoods, while our approach gives better estimates of the gradient of the log density, the *score*, which describes the distribution's shape.

1. Introduction

Density estimation is a foundational problem in statistics and machine learning (**??**), lying at the core of both supervised and unsupervised machine learning problems. Classical techniques such as kernel density estimation, however, struggle to exploit the structure inherent to complex problems, and thus can require unreasonably large sample sizes for adequate fits. For instance, assuming only twice-differentiable densities, the L_2 risk of density estimation with N samples in D dimensions scales as $\mathcal{O}(N^{-4/(4+D)})$ (**?**, Section 6.5).

One promising approach for incorporating this necessary structure is the *kernel exponential family* (**???**). This model allows for any log-density which is suitably smooth under a given kernel, i.e. any function in the corresponding reproducing kernel Hilbert space. Choosing a finite-dimensional kernel recovers any classical exponential family, but when the kernel is sufficiently powerful the class becomes very rich: dense in the family of continuous probability densities on compact domains in KL, TV, Hellinger, and L^r distances (**?**, Corollary 2). The normalization constant is not available in closed form, making fitting by maximum likelihood difficult, but the alternative technique of *score matching* (**?**) allows for practical usage with theoretical convergence guarantees (**?**).

The choice of kernel directly corresponds to a smoothness assumption on the model, allowing one to design a kernel corresponding to prior knowledge about the target density. Yet explicitly deciding upon a kernel to incorporate that knowledge can be complicated. Indeed, previous applications of the kernel exponential family model have exclusively employed simple kernels, such as the Gaussian, with a small number of parameters (e.g. the length scale) chosen by heuristics or cross-validation (e.g. **??**). These kernels are typically *spatially invariant*, corresponding to a uniform smoothness assumption across the domain. Although such kernels are sufficient for consistency in the infinite-sample limit, the induced models can fail in practice on finite datasets, especially if the data takes differently-scaled shapes in different parts of the space. Figure 1 (left) illustrates this problem when fitting a simple mixture of Gaussians. Here there are two "correct" bandwidths, one for the broad mode and one for the narrow mode. A translation-invariant kernel must pick a single one, e.g. an average between the two, and any choice will yield a poor fit on at least part of the density.

In this work, we propose to *learn* the kernel of an exponential family directly from data. We can then achieve far more than simply tuning a length scale, instead learning location-dependent kernels that adapt to the underlying shape and smoothness of the target density. We use kernels of the form

$$k(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{y})), \tag{1}$$

where the deep network $\boldsymbol{\phi}$ extracts features of the input and κ is a simple kernel (e.g. a Gaussian) on those features. These types of kernels have seen success in supervised learning (**??**) and critic functions for implicit generative models (**????**), among other settings. We call the resulting model a deep kernel exponential family (DKEF).

[*]Equal contribution [1]Gatsby Computational Neuroscience Unit, University College London, London, U.K.. Correspondence to: Li K. Wenliang <wenliang2012@gmail.com>, D. J. Sutherland <dougal@gmail.com>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

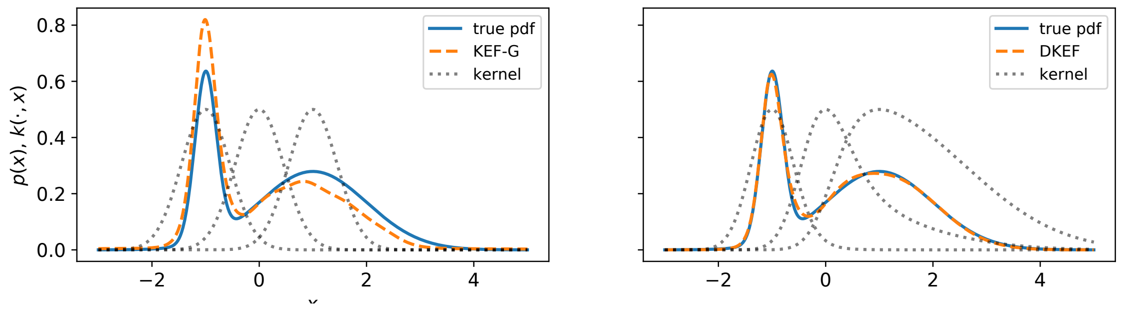

Figure 1: Fitting few samples from a Gaussian mixture, using kernel exponential families. Black dotted lines show $k(-1, \boldsymbol{x})$, $k(0, \boldsymbol{x})$, and $k(1, \boldsymbol{x})$. (Left) Using a location-invariant Gaussian kernel, the sharper component gets too much weight. (Right) A kernel parameterized by a neural network learns length scales that adapt to the density, giving a much better fit.

We can train both kernel parameters (including all the weights of the deep network) and, unusually, even *regularization* parameters directly on the data, in a form of meta-learning. Normally, directly optimizing regularization parameters would always yield 0, since their beneficial effect in preventing overfitting is by definition not seen on the training set. Here, though, we can exploit the closed-form fit of the kernel exponential family to optimize a "held-out" score (Section 3). Figure 1 (right) demonstrates the success of this model on the same mixture of Gaussians; here the learned, location-dependent kernel gives a much better fit.

We compare the results of our new model to recent general-purpose deep density estimators, primarily *autoregressive models* (**???**) and *normalizing flows* (**???**). These models learn deep networks with structures designed to compute normalized densities, and are fit via maximum likelihood. We explore the strengths and limitations of both deep likelihood models and deep kernel exponential families on a variety of datasets, including artificial data designed to illustrate scenarios where certain surprising problems arise, as well as benchmark datasets used previously in the literature. The models fit by maximum likelihood typically give somewhat higher likelihoods, whereas the deep kernel exponential family generally better fits the *shape* of the distribution.

2. Background

Score matching Suppose we observe $\mathcal{D} = \{\boldsymbol{x}_n\}_{n=1}^N$, a set of independent samples $\boldsymbol{x}_n \in \mathbb{R}^D$ from an unknown density $p_0(\boldsymbol{x})$. We posit a class of possible models $\{p_{\boldsymbol{\theta}}\}$, parameterized by $\boldsymbol{\theta}$; our goal is to use the data $\mathcal{D}$ to select some $\hat{\boldsymbol{\theta}}$ such that $p_{\hat{\boldsymbol{\theta}}} \approx p_0$. The standard approach for selecting $\boldsymbol{\theta}$ is maximum likelihood: $\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \prod_{n=1}^N p_{\boldsymbol{\theta}}(\boldsymbol{x}_n)$.

Many interesting model classes, however, are defined as $p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \tilde{p}_{\boldsymbol{\theta}}(\boldsymbol{x})/Z_{\boldsymbol{\theta}}$, where the normalization constant $Z_{\boldsymbol{\theta}} = \int_{\boldsymbol{x}} \tilde{p}_{\boldsymbol{\theta}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}$ cannot be easily computed. In this setting, an optimization algorithm to estimate $\boldsymbol{\theta}$ by maximum likelihood requires estimating (the derivative of) $Z_{\boldsymbol{\theta}}$ for each candidate $\boldsymbol{\theta}$ considered during optimization. Moreover, the maximum likelihood solution may not even be well-defined when $\boldsymbol{\theta}$ is infinite-dimensional (**??**). The intractability of maximum likelihood led **?** to propose an alternative objective, called *score matching*. Rather than maximizing the likelihood, one minimizes the Fisher divergence $J(p_{\boldsymbol{\theta}} \| p_0)$:

$$\frac{1}{2} \int p_0(\boldsymbol{x}) \| \nabla_{\boldsymbol{x}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) - \nabla_{\boldsymbol{x}} \log p_0(\boldsymbol{x}) \|_2^2 \, \mathrm{d}\boldsymbol{x}. \quad (2)$$

Under mild regularity conditions, this is equal to

$$\int_{\boldsymbol{x}} p_0(\boldsymbol{x}) \sum_{d=1}^D \left[\partial_d^2 \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) + \frac{1}{2} (\partial_d \log p_{\boldsymbol{\theta}}(\boldsymbol{x}))^2 \right] \mathrm{d}\boldsymbol{x}, \quad (3)$$

up to an additive constant depending only on p_0, which can be ignored during training. Here $\partial_d^n f(\boldsymbol{x})$ denotes $\frac{\partial^n}{(\partial y_d)^n} f(\boldsymbol{y})|_{\boldsymbol{y}=\boldsymbol{x}}$. We can estimate (3) with $\hat{J}(p_{\boldsymbol{\theta}}, \mathcal{D})$:

$$\frac{1}{N} \sum_{n=1}^N \sum_{d=1}^D \left[\partial_d^2 \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n) + \frac{1}{2} (\partial_d \log p_{\boldsymbol{\theta}}(\boldsymbol{x}_n))^2 \right]. \quad (4)$$

Notably, (4) does not depend on $Z_{\boldsymbol{\theta}}$, and so we can minimize it to find an unnormalized model $\tilde{p}_{\hat{\boldsymbol{\theta}}}$ for p_0. Score matching is consistent in the well-specified setting (**?**, Theorem 2), and was related to maximum likelihood by **?**, who argues it finds a fit robust to infinitesimal noise.

Unnormalized models $\tilde{p}$ are sufficient for many tasks (**?**), including finding modes, approximating Hamiltonian Monte Carlo on targets without gradients (**?**), and learning discriminative features (**?**). If we require a normalized model, however, we can estimate the normalizing constant once, after estimating $\boldsymbol{\theta}$; this will be far more computationally efficient than estimating it at each step of an iterative maximum likelihood optimization algorithm.

Kernel exponential families The kernel exponential family (**??**) is the class of all densities satisfying a smoothness constraint: $\log \tilde{p}(\boldsymbol{x}) = f(\boldsymbol{x}) + \log q_0(\boldsymbol{x})$, where q_0

is some fixed function and f is any function in the reproducing kernel Hilbert space $\mathcal{H}$ with kernel k. This class is an exponential family with natural parameter f and sufficient statistic $k(\boldsymbol{x}, \cdot)$, due to the reproducing property $f(\boldsymbol{x}) = \langle f, k(\boldsymbol{x}, \cdot)\rangle_{\mathcal{H}}$:

$$\tilde{p}_f(\boldsymbol{x}) = \exp\left(f(\boldsymbol{x})\right) q_0(\boldsymbol{x}) = \exp\left(\langle f, k(\boldsymbol{x}, \cdot)\rangle_{\mathcal{H}}\right) q_0(\boldsymbol{x}).$$

Using a simple finite-dimensional $\mathcal{H}$, we can recover any standard exponential family, e.g. normal, gamma, or Poisson; if $\mathcal{H}$ is sufficiently rich, the family can approximate any continuous distribution with tails like q_0 arbitrarily well (**?**, Example 1 and Corollary 2).

These models do not in general have a closed-form normalizer. For some f and q_0, $\tilde{p}_f$ may not even be normalizable, but if q_0 is e.g. a Gaussian density, typical choices of $\boldsymbol{\phi}$ and κ in (1) guarantee a normalizer exists (Appendix A).

? proved good statistical properties for choosing $f \in \mathcal{H}$ by minimizing a regularized form of (4), $\hat{f} = \arg\min_{f\in\mathcal{H}} \hat{J}(\tilde{p}_f, \mathcal{D}) + \lambda \|f\|^2_{\mathcal{H}}$, but their algorithm has an impractical computational cost of $\mathcal{O}(N^3 D^3)$. This can be alleviated with the Nyström-type "lite" approximation (**??**): select M inducing points $\boldsymbol{z}_m \in \mathbb{R}^D$, and select $f \in \mathcal{H}$ as

$$f^k_{\boldsymbol{\alpha},\boldsymbol{z}}(\boldsymbol{x}) = \sum_{m=1}^{M} \alpha_m k(\boldsymbol{x}, \boldsymbol{z}_m), \quad \tilde{p}^k_{\boldsymbol{\alpha},\boldsymbol{z}} = \tilde{p}_{f^k_{\boldsymbol{\alpha},\boldsymbol{z}}}. \tag{5}$$

As the span of $\{k(\boldsymbol{z}, \cdot)\}_{\boldsymbol{z}\in\mathbb{R}^D}$ is dense in $\mathcal{H}$, this is a natural approximation, similar to classical RBF networks (**?**). The "lite" model often yields excellent empirical results at much lower computational cost than the full estimator. We can regularize (4) in several ways and still find a closed-form solution for $\boldsymbol{\alpha}$. In this work, our loss $\hat{J}(f^k_{\boldsymbol{\alpha},\boldsymbol{z}}, \boldsymbol{\lambda}, \mathcal{D})$ will be

$$\hat{J}(\tilde{p}^k_{\boldsymbol{\alpha},\boldsymbol{z}}, \mathcal{D}) + \frac{\lambda_\alpha}{2}\|\boldsymbol{\alpha}\|^2 + \frac{\lambda_C}{2N}\sum_{n=1}^{N}\sum_{d=1}^{D}\left[\partial_d^2 \log \tilde{p}^k_{\boldsymbol{\alpha},\boldsymbol{z}}(\boldsymbol{x}_n)\right]^2.$$

? used a small λ_α for numerical stability but primarily regularized with $\lambda_H \|f^k_{\boldsymbol{\alpha},\boldsymbol{z}}\|^2_{\mathcal{H}}$. As we change k, however, $\|f\|_{\mathcal{H}}$ changes meaning, and we found empirically that this regularizer tends to harm the fit. The λ_C term was recommended by **?**, encouraging the learned log-density to be smooth without much extra computation; it provides some empirical benefit in our context. Given k, $\boldsymbol{z}$, and $\boldsymbol{\lambda}$, Proposition 3 (Appendix B) shows we can find the optimal $\boldsymbol{\alpha}$ by solving an $M \times M$ linear system in $\mathcal{O}(M^2 N D + M^3)$ time: the $\boldsymbol{\alpha}$ which minimizes $\hat{J}(f^k_{\boldsymbol{\alpha},\boldsymbol{z}}, \boldsymbol{\lambda}, \mathcal{D})$ is

$$\boldsymbol{\alpha}(\boldsymbol{\lambda}, k, \boldsymbol{z}, \mathcal{D}) = -\left(\boldsymbol{G} + \lambda_\alpha \boldsymbol{I} + \lambda_C \boldsymbol{U}\right)^{-1} \boldsymbol{b} \tag{6}$$

$$G_{m,m'} = \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \partial_d k(\boldsymbol{x}_n, \boldsymbol{z}_m)\, \partial_d k(\boldsymbol{x}_n, \boldsymbol{z}_{m'})$$

$$U_{m,m'} = \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \partial_d^2 k(\boldsymbol{x}_n, \boldsymbol{z}_m)\, \partial_d^2 k(\boldsymbol{x}_n, \boldsymbol{z}_{m'})$$

$$b_m = \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \partial_d^2 k(\boldsymbol{x}_n, \boldsymbol{z}_m) + \partial_d \log q_0(\boldsymbol{x}_n)\, \partial_d k(\boldsymbol{x}_n, \boldsymbol{z}_m) + \lambda_C \partial_d^2 \log q_0(\boldsymbol{x}_n)\, \partial_d^2 k(\boldsymbol{x}_n, \boldsymbol{z}_m).$$

3. Fitting Deep Kernels

All previous applications of score matching in the kernel exponential family of which we are aware (e.g. **????**) have used kernels of the form $k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{1}{2\sigma^2}\|\boldsymbol{x} - \boldsymbol{y}\|^2\right) + r\left(\boldsymbol{x}^\mathsf{T}\boldsymbol{y} + c\right)^2$, with kernel parameters and regularization weights either fixed a priori or selected via cross-validation. This simple form allows the various kernel derivatives required in (6) to be easily computed by hand, and the small number of parameters makes grid search adequate for model selection. But, as discussed in Section 1, these simple kernels are insufficient for complex datasets. Thus we wish to use a richer class of kernels $\{k_{\boldsymbol{w}}\}$, with a large number of parameters $\boldsymbol{w}$ – in particular, kernels defined by a neural network. This prohibits model selection via simple grid search.

One could attempt to directly minimize $\hat{J}(f^{k_{\boldsymbol{w}}}_{\boldsymbol{\alpha},\boldsymbol{z}}, \boldsymbol{\lambda}, \mathcal{D})$ jointly in the kernel parameters $\boldsymbol{w}$, the model parameters $\boldsymbol{\alpha}$, and perhaps the inducing points $\boldsymbol{z}$. Consider, however, the case where we simply use a Gaussian kernel and $\{\boldsymbol{z}_m\} = \mathcal{D}$. Then we can achieve arbitrarily good values of (3) by taking $\sigma \to 0$, drastically overfitting to the training set $\mathcal{D}$.

We can avoid this problem – and additionally find the best values for the regularization weights $\boldsymbol{\lambda}$ – with a form of meta-learning. We find choices for the kernel and regularization which will give us a good value of $\hat{J}$ on a "validation set" $\mathcal{D}_v$ when fit to a fresh "training set" $\mathcal{D}_t$. Specifically, we take stochastic gradient steps following $\nabla_{\lambda,\boldsymbol{w},\boldsymbol{z}} \hat{J}(\tilde{p}^{k_{\boldsymbol{w}}}_{\boldsymbol{\alpha}(\lambda, k_{\boldsymbol{w}}, \boldsymbol{z}, \mathcal{D}_t),\boldsymbol{z}}, \mathcal{D}_v)$. We can easily do this because we have a differentiable closed-form expression (6) for the fit to $\mathcal{D}_t$, rather than having to e.g. back-propagate through an unrolled iterative optimization procedure. As we used small minibatches in this procedure, for the final fit we use the whole dataset: we first freeze $\boldsymbol{w}$ and $\boldsymbol{z}$ and find the optimal $\boldsymbol{\lambda}$ for the whole training data, then finally fit $\boldsymbol{\alpha}$ with the new $\boldsymbol{\lambda}$. This process is summarized in Algorithm 1.

Computing kernel derivatives Solving for $\boldsymbol{\alpha}$ and computing the loss (4) require matrices of kernel second derivatives, but current deep learning-oriented automatic differentiation systems are not optimized for evaluating tensor-valued higher-order derivatives at once. We therefore implement backpropagation to compute $\boldsymbol{G}$, $\boldsymbol{U}$, and $\boldsymbol{b}$ of (6) as TensorFlow operations (**?**) to obtain the scalar loss $\hat{J}$, and used TensorFlow's automatic differentiation only to optimize $\boldsymbol{w}$,

Algorithm 1: Full training procedure

```
input: Dataset $\mathcal{D}$; initial inducing points $\boldsymbol{z}$, kernel
        parameters $\boldsymbol{w}$, regularization $\boldsymbol{\lambda} = (\lambda_\alpha, \lambda_C)$
Split $\mathcal{D}$ into $\mathcal{D}_1$ and $\mathcal{D}_2$;
Optimize $\boldsymbol{w}$, $\boldsymbol{\lambda}$, $\boldsymbol{z}$, and maybe $q_0$ params:
while $\hat{J}(\tilde{p}^{k_{\boldsymbol{w}}}_{\boldsymbol{\alpha}(\boldsymbol{\lambda}, k_{\boldsymbol{w}}, \boldsymbol{z}, \mathcal{D}_1), \boldsymbol{z}}, \mathcal{D}_2)$ still improving do
    Sample disjoint data subsets $\mathcal{D}_t, \mathcal{D}_v \subset \mathcal{D}_1$;
    $f(\cdot) = \sum_{m=1}^{M} \alpha_m(\boldsymbol{\lambda}, k_{\boldsymbol{w}}, \boldsymbol{z}, \mathcal{D}_t) k_{\boldsymbol{w}}(\boldsymbol{z}_m, \cdot)$;
    $\hat{J} = \frac{1}{|\mathcal{D}_v|} \sum_{n=1}^{|\mathcal{D}_v|} \sum_{d=1}^{D} \left[ \partial_d^2 f(\boldsymbol{x}_n) + \frac{1}{2} (\partial_d f(\boldsymbol{x}_n))^2 \right]$;
    Take SGD step in $\hat{J}$ for $\boldsymbol{w}$, $\boldsymbol{\lambda}$, $\boldsymbol{z}$, maybe $q_0$ params;
end
Optimize $\boldsymbol{\lambda}$ for fitting on larger batches:
while $\hat{J}(\tilde{p}^{k_{\boldsymbol{w}}}_{\boldsymbol{\alpha}(\boldsymbol{\lambda}, k_{\boldsymbol{w}}, \boldsymbol{z}, \mathcal{D}_1), \boldsymbol{z}}, \mathcal{D}_2)$ still improving do
    $f(\cdot) = \sum_{m=1}^{M} \alpha_m(\boldsymbol{\lambda}, k_{\boldsymbol{w}}, \boldsymbol{z}, \mathcal{D}_1) k_{\boldsymbol{w}}(\cdot, \boldsymbol{z}_m)$;
    Sample subset $\mathcal{D}_v \subset D_2$;
    $\hat{J} = \frac{1}{|\mathcal{D}_v|} \sum_{n=1}^{|\mathcal{D}_v|} \sum_{d=1}^{D} \left[ \partial_d^2 f(\boldsymbol{x}_n) + \frac{1}{2} (\partial_d f(\boldsymbol{x}_n))^2 \right]$;
    Take SGD steps in $\hat{J}$ for $\boldsymbol{\lambda}$ only;
end
Finalize $\boldsymbol{\alpha}$ on $\mathcal{D}_1$:
Find $\boldsymbol{\alpha} = \boldsymbol{\alpha}(\boldsymbol{\lambda}, k_{\boldsymbol{w}}, \boldsymbol{z}, \mathcal{D}_1)$;
return: $\log \tilde{p}(\cdot) = \sum_{m=1}^{M} \alpha_m k_{\boldsymbol{w}}(\cdot, \boldsymbol{z}_m) + \log q_0(\cdot)$;
```

$\boldsymbol{z}$, $\boldsymbol{\lambda}$, and q_0 parameters.

Backpropagation to find these second derivatives requires explicitly computing the Hessians of intermediate layers of the network, which becomes quite expensive as the model grows; this limits the size of kernels that our model can use. A more efficient implementation based on Hessian-vector products might be possible in an automatic differentiation system with better support for matrix derivatives.

Kernel architecture We will choose our kernel $k_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y})$ as a mixture of R Gaussian kernels with length scales σ_r, taking in features of the data extracted by a network $\boldsymbol{\phi}_{\boldsymbol{w}_r}(\cdot)$:

$$\sum_{r=1}^{R} \rho_r \exp\left(-\frac{1}{2\sigma_r^2} \left\| \boldsymbol{\phi}_{\boldsymbol{w}_r}(\boldsymbol{x}) - \boldsymbol{\phi}_{\boldsymbol{w}_r}(\boldsymbol{y}) \right\|^2 \right). \qquad (7)$$

Combining R components makes it easier to account for both short-range and long-range dependencies. We constrain $\rho_r \geq 0$ to ensure a valid kernel, and $\sum_{r=1}^{R} \rho_r = 1$ for simplicity. The networks $\boldsymbol{\phi}_{\boldsymbol{w}}$ are made of L fully connected layers of width W. For $L > 1$, we found that adding a skip connection from data directly to the top layer speeds up learning. A softplus nonlinearity, $\log(1 + \exp(x))$, ensures that the model is twice-differentiable so (3) is well-defined.

3.1. Behavior on Mixtures

One interesting limitation of score matching is the following: suppose that p_0 is composed of two disconnected components, $p_0(\boldsymbol{x}) = \pi p_1(\boldsymbol{x}) + (1 - \pi) p_2(\boldsymbol{x})$ for $\pi \in (0, 1)$ and p_1, p_2 having disjoint, separated support. Then $\nabla \log p_0(x)$ will be $\nabla \log p_1(\boldsymbol{x})$ in the support of p_1, and $\nabla \log p_2(\boldsymbol{x})$ in the support of p_2. Score matching compares $\nabla \log \tilde{p}_{\boldsymbol{\theta}}$ to $\nabla \log p_1$ and $\nabla \log p_2$, but is completely blind to $\tilde{p}_{\boldsymbol{\theta}}$'s relative mass between the two components; it is equally happy with *any* reweighting of the components.

If all modes are connected by regions of positive density, then the log density gradient in between components will determine their relative weight, and indeed score matching is then consistent. But when p_0 is *nearly* zero between two dense components, so that there are no or few samples in between, score matching will generally have insufficient evidence to weight nearly-separate components.

Proposition 4 (Appendix C) studies the the kernel exponential family in this case. For two components that are completely separated according to k, (6) fits each as it would if given only that component, except that the effective λ_α is scaled: smaller components are regularized more.

Appendix C.1 studies a simplified case where the kernel length scale σ is far wider than the component; then the density ratio between components, which should be $\frac{\pi}{1-\pi}$, is approximately $\exp\left(\frac{D}{2\sigma^2 \lambda_\alpha} \left(\pi - \frac{1}{2} \right) \right)$. Depending on the value of $\frac{D}{2\sigma^2 \lambda_\alpha}$, this ratio will often either be quite extreme, or nearly 1. It is unclear, however, how well this result generalizes to other settings.

A heuristic workaround when disjoint components are suspected is as follows: run a clustering algorithm to identify disjoint components, separately fit a model to each cluster, then weight each model according to its sample count. When the components are well-separated, this clustering is straightforward, but it may be difficult in high-dimensional cases when samples are sparse but not fully separated.

3.2. Model Evaluation

In addition to qualitatively evaluating fits, we will evaluate our models with three quantitative criteria. The first is the finite-set Stein discrepancy (FSSD; **?**), a measure of model fit which does not depend on the normalizer $Z_{\boldsymbol{\theta}}$. It examines the fit of the model at J test locations $\boldsymbol{V} = \{\boldsymbol{v}_b\}_{b=1}^{B}$ using an kernel $l(\cdot, \cdot)$, as $\frac{1}{DB} \sum_{b=1}^{B} \| \mathbb{E}_{\boldsymbol{x} \sim p_0} [l(\boldsymbol{x}, \boldsymbol{v}_b) \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} l(\boldsymbol{x}, \boldsymbol{v}_b)] \|^2$. With randomly selected $\boldsymbol{V}$ and some mild assumptions, it is zero if and only if $p = p_0$. We use a Gaussian kernel with bandwidth equal to the median distance between test points,[1]and choose $\boldsymbol{V}$ by adding small Gaussian noise to data points. **?** construct a hypothesis test to test which of p and p' is closer to p_0 in the FSSD. We will report a score – the p-value of this test – which is near 0 when model p is better, near 1 when model p' is better, and around $\frac{1}{2}$ when the two models are equivalent. We emphasize that we are using this as a model comparison score on an interpretable scale, but *not* following a hypothesis testing framework. Another similar performance measure is the kernel Stein discrepancy (KSD) (**?**), where the model's goodness-of-fit is evaluated at all test data points rather than at random test locations. We omit the results as they are essentially identical to that of the FSSD, even across a wide range of kernel bandwidths.

As all our models are twice-differentiable, we also compare the score-matching loss (4) on held-out test data. A lower score-matching loss implies a smaller Fisher divergence between model and data distributions.

Finally, we compare test log-likelihoods, using importance sampling estimates of the normalizing constant $Z_{\boldsymbol{\theta}}$:

$$\hat{Z}_{\boldsymbol{\theta}} = \frac{1}{U}\sum_{u=1}^{U} \mathrm{r}_u \quad \text{where} \quad \mathrm{r}_u := \frac{\tilde{p}_{\boldsymbol{\theta}}(\boldsymbol{y}_u)}{q_0(\boldsymbol{y}_u)}, \ \boldsymbol{y}_u \sim q_0,$$

so $\mathbb{E}\,\hat{Z}_{\boldsymbol{\theta}} = \int \frac{\tilde{p}_{\boldsymbol{\theta}}(\boldsymbol{y}_u)}{q_0(\boldsymbol{y}_u)} q_0(\boldsymbol{y}_u) = Z_{\boldsymbol{\theta}}$. Our log-likelihood estimate is $\log \hat{p}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log \tilde{p}_{\boldsymbol{\theta}}(\boldsymbol{x}) - \log \hat{Z}_{\boldsymbol{\theta}}$. This estimator is consistent, but Jensen's inequality tells us that $\mathbb{E} \log \hat{p}_{\boldsymbol{\theta}}(\boldsymbol{x}) > \log p_{\boldsymbol{\theta}}(\boldsymbol{x})$, so our evaluation will be over-optimistic. Worse, the variance of $\log \hat{Z}_{\boldsymbol{\theta}}$ can be misleadingly small when the bias is still quite large; we observed this in our experiments. We can, though, bound the bias:

Proposition 1. *Suppose that* $a, s \in \mathbb{R}$ *are such that* $\Pr(\mathrm{r}_u \geq a) = 1$ *and* $\Pr(\mathrm{r}_u \leq s) \leq \rho < \frac{1}{2}$. *Define* $t := (s+a)/2$, $\psi(q, Z_{\boldsymbol{\theta}}) := \log\frac{Z}{q} + \frac{q}{Z} - 1$, *and let* $P := \max\left(\psi(a, Z_{\boldsymbol{\theta}}), \psi(t, Z_{\boldsymbol{\theta}})\right)$. *Then*

$$\log Z_{\boldsymbol{\theta}} - \mathbb{E} \log \hat{Z}_{\boldsymbol{\theta}} \leq \frac{\psi\left(t, Z_{\boldsymbol{\theta}}\right)}{\left(Z_{\boldsymbol{\theta}} - t\right)^2} \frac{\mathrm{Var}[\mathrm{r}_u]}{U} + P\left(4\rho(1-\rho)\right)^{\frac{U}{2}}.$$

(Proof in Appendix D.) We can find a, s because we propose from q_0, and thus we can effectively estimate the bound (Appendix D.1). This estimate of the upper bound is itself biased upwards (Proposition 6), so it is likely, though not guaranteed, that the estimate overstates the amount of bias.

3.3. Previous Attempts at Deep Score Matching

? used score matching to train a (one-layer) network to output an unnormalized log-density. This approach is essentially a special case of ours: use the kernel $k_{\boldsymbol{w}}(\boldsymbol{x}, \boldsymbol{y}) = \phi_{\boldsymbol{w}}(\boldsymbol{x})\phi_{\boldsymbol{w}}(\boldsymbol{y})$, where $\phi_{\boldsymbol{w}} : \mathbb{R}^D \to \mathbb{R}$. Then the function $f^{k_w}_{\boldsymbol{\alpha},\boldsymbol{z}}(\boldsymbol{x})$ from (5) is

$$\sum_{m=1}^{M} \alpha_m \phi_{\boldsymbol{w}}(\boldsymbol{z}_m)\phi_{\boldsymbol{w}}(\boldsymbol{x}) = \left[\sum_{m=1}^{M} \alpha_m \phi_{\boldsymbol{w}}(\boldsymbol{z}_m)\right] \phi_{\boldsymbol{w}}(\boldsymbol{x}).$$

The scalar in brackets is fit analytically, so $\log p$ is determined almost entirely by the network $\phi_{\boldsymbol{w}}$ plus $\log q_0(\boldsymbol{x})$.

? recently also attempted parameterizing the unnormalizing log-density as a deep network, using an approximation called Parzen score matching. This approximation requires a global constant bandwidth to define the Parzen window size for the loss function, fit to the dataset before learning the model. This is likely only sensible on datasets for which simple fixed-bandwidth kernel density estimation is appropriate; on more complex datasets, the loss may be poorly motivated. It also leads to substantial oversmoothing visible in their results. As they did not provide code for their method, we do not compare to it empirically.

4. Experimental Results

In our experiments, we compare to several alternative methods. The first group are all fit by maximum likelihood, and broadly fall into (at least) one of two categories: *autoregressive models* decompose $p(\mathrm{x}_1, \dots, \mathrm{x}_D) = \prod_{d=1}^{D} p(\mathrm{x}_d | \mathbf{x}_{\leq d})$ and learn a parametric density model for each of these conditionals. *Normalizing flows* instead apply a series of invertible transformations to some simple initial density, say standard normal, and then compute the density of the overall model via the Jacobian of the transformation. We use implementations[2] of the following several models in these categories by **?**:

MADE (**?**) masks the connections of an autoencoder so it is autoregressive. We used two hidden layers, and each conditional a Gaussian. **MADE-MOG** is the same but with each conditional a mixture of 10 Gaussians.

Real NVP (**?**) is a normalizing flow; we use a general-purpose form for non-image datasets.

MAF (**?**). A combination of a normalizing flow and MADE, where the base density is modeled by MADE, with 5 autoregressive layers. **MAF-MOG** instead models the base density by MADE-MOG.

For the models above, we used layers of width 30 for experiments on synthetic data, and 100 for benchmark datasets. Larger values did not improve performance.

KCEF (**?**). Inspired by autoregressive models, the density is modeled by a cascade of kernel conditional exponential

[1]This reasonable choice avoids tuning any parameters. We do not optimize the kernel or the test locations to avoid a situation in which model p is better than p' in some respects but p' better than p in others; instead, we use a simple default mode of comparison.

[2]`github.com/gpapamak/maf`

family distributions, fit by score matching with Gaussian kernels.[3]

DKEF. On synthetic datasets, we consider four variants of our model with one kernel component, $R = 1$. KEF-G refers to the model using a Gaussian kernel with a learned bandwidth. DKEF-G-15 has the kernel (7), with $L = 3$ layers of width $W = 15$. DKEF-G-50 is the same with $W = 50$. To investigate whether the top Gaussian kernel helps performance, we also train DKEF-L-50, whose kernel is $k_\theta(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{\phi}_{\boldsymbol{w}}(\boldsymbol{x}) \cdot \boldsymbol{\phi}_{\boldsymbol{w}}(\boldsymbol{y})$, where $\boldsymbol{\phi}_{\boldsymbol{w}}$ has $W = 50$. To compare with the architecture of **?**, DKEF-L-50-1 has the same architecture as DKEF-L-50 except that we add an extra layer with a single neuron, and use $M = 1$. In all experiments, $q_0(\boldsymbol{x}) = \prod_{d=1}^{D} \exp\left(-|x_d - \mu_d|^{\beta_d}/(2\sigma_d^2)\right)$, with $\beta_d > 1$. On benchmark datasets, we use DKEF-G-30 and KEF-G with three kernel components, $R = 3$. Code for DKEF is at `github.com/kevin-w-li/deep-kexpfam`.

4.1. Behavior on Synthetic Datasets

We first demonstrate the behavior of the models on several two-dimensional synthetic datasets Funnel, Banana, Ring, Square, Cosine, Mixture of Gaussians (MoG) and Mixture of Rings (MoR). Together, they cover a range of geometric complexities and multimodality.

We visualize the fit of various methods by showing the log density function in Figure 2. For each model fit on each distribution, we report the normalized log likelihood (left) and Fisher divergence (right). In general, the kernel score matching methods find cleaner boundaries of the distributions, and our main model KDEF-G-30 produces the lowest Fisher divergence on many of the synthetic datasets while maintaining high likelihoods.

Among the kernel exponential families, DKEF-G outperformed previous versions where ordinary Gaussian kernels were used for either joint (KEF-G) or autoregressive (KCEF) modeling. DKEF-G-50 does not substantially improve over DKEF-G-15; we omit it for space. We can gain additional insights into the model by looking at the shape of the learned kernel, shown by the colored lines; the kernels do indeed adapt to the local geometry.

DKEF-L-50 and DKEF-L-50-1 show good performance when the target density has simple geometries, but had trouble in more complex cases, even with much larger networks than used by DKEF-G-15. It seems that a Gaussian kernel with inducing points provides much stronger representational features than a using linear kernel and/or a single inducing point. A large enough network $\boldsymbol{\phi}_{\boldsymbol{w}}$ would likely be able to perform the task well, but, using currently available software, the second derivatives in the score matching loss limit our ability to use very large networks. A similar phenomenon was observed by **?** in the context of GAN critics, where combining some analytical RKHS optimization with deep networks allowed much smaller networks to work well.

As expected, models fit by DKEFs generally have smaller Fisher divergences than likelihood-based methods. For Funnel and Banana, the true densities are simple transformations of Gaussians, and the normalizing flow models perform relatively well. But on Ring, Square, and Cosine, the shape of the learned distribution by likelihood-based methods exhibits noticeable artifacts, especially at the boundaries. These artifacts, particularly the "breaks" in Ring, may be caused by a normalizing flow's need to be invertible and smooth. The shape learned by DKEF-G-15 is much cleaner.

On multimodal distributions with disjoint components, likelihood-based and score matching-based methods show interesting failure modes. The likelihood-based models often connect separated modes with a "bridge", even for MADE-MOG and MAF-MOG which use explicit mixtures. On the other hand, DKEF is able to find the shapes of the components, but the weighting between modes is unstable. As suggested in Section 3.1, we also fit mixtures of all models (except KCEF) on a partition of MoR found by spectral clustering (**?**); DKEF-G-15 produced an excellent fit.

Another challenge we observed in our experiments is that the estimator of the objective function, $\hat{J}$, tends to be more noisy as the model fit improves. This happens particularly on datasets where there are "sharp" features, such as Square (see Figure 4 in Appendix E.1), where the model's curvature becomes extreme at some points. This can cause higher variance in the gradients of the parameters, and more difficulty in optimization.

4.2. Results on Benchmark Datasets

Following recent work on density estimation (**????**), we trained DKEF and the likelihood-based models on five UCI datasets (**?**); in particular, we used RedWine, WhiteWine, Parkinson, HepMass, and MiniBoone. All performances were measured on held-out test sets. We did not run KCEF due to its computational expense. Appendix E.2 gives further details.

Figure 3 shows results. In gradient matching as measured by the FSSD, DKEF tends to have the best values. Test set sizes are too small to yield a confident p-value on the Wine datasets, but the model comparison test confidently favors DKEF on datasets with large-enough test sets. The FSSD results agree with KSD, which is omitted. In the score matching loss[4] (3), DKEF is the best on Wine datasets

[3] `github.com/MichaelArbel/KCEF`

[4] The implementation of **?** sometimes produced `NaN` values for the required second derivatives, especially for MADE-MOG. We

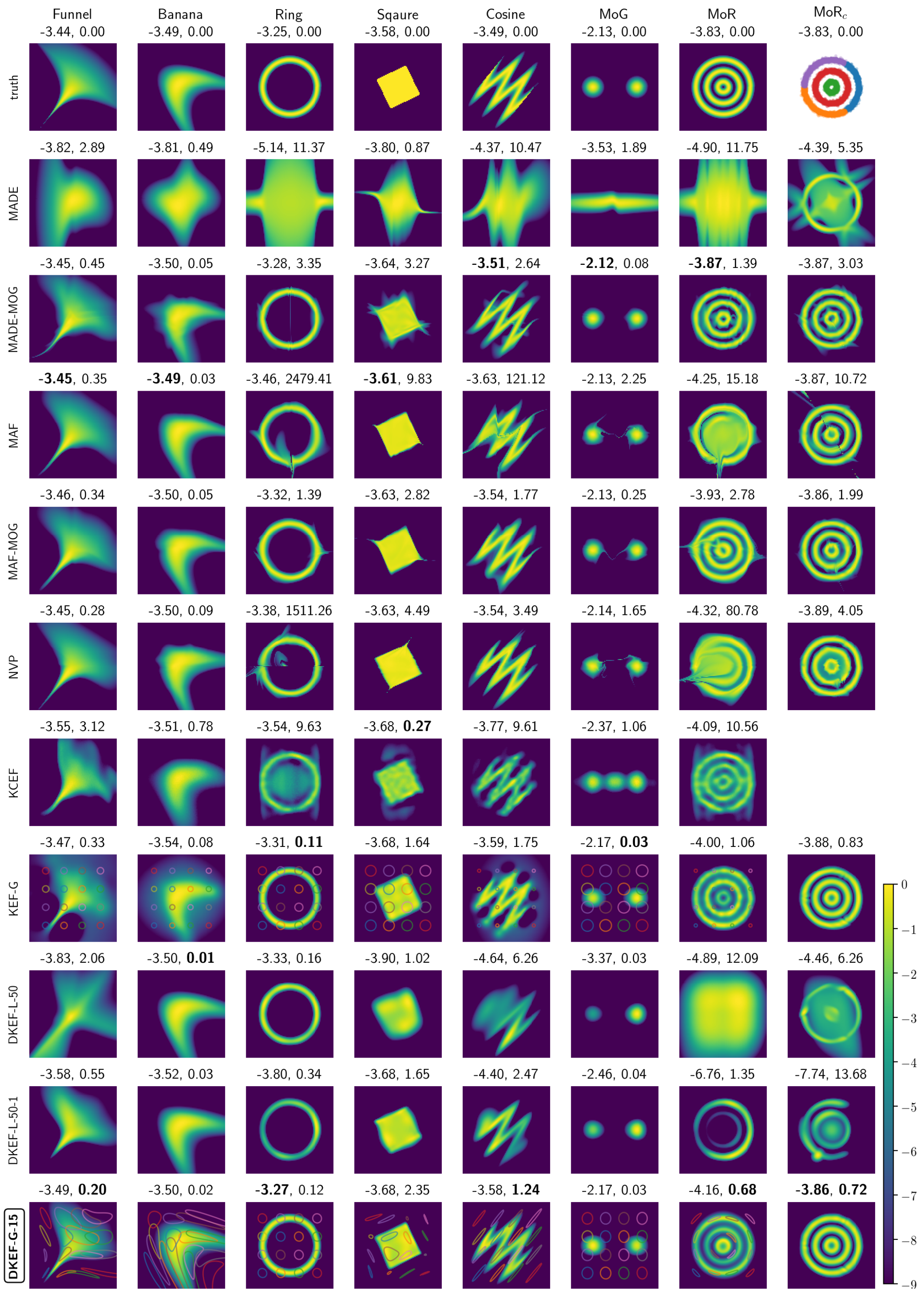

Figure 2: Log densities learned by different models. Our model is DKEF-G-15 at the bottom row. Columns are different synthetic datasets. The rightmost columns shows a mixture of each model (except KCEF) on the same clustering of MoR. We subtracted the maximum from each log density, and clipped the minimum value at -9. Above each panel are shown the average log-likelihoods (left) and Fisher divergence (right) on held-out data points. Bold indicates the best fit. For DKEF-G models, faint colored lines correspond to contours at 0.9 of the kernel evaluated at different locations.

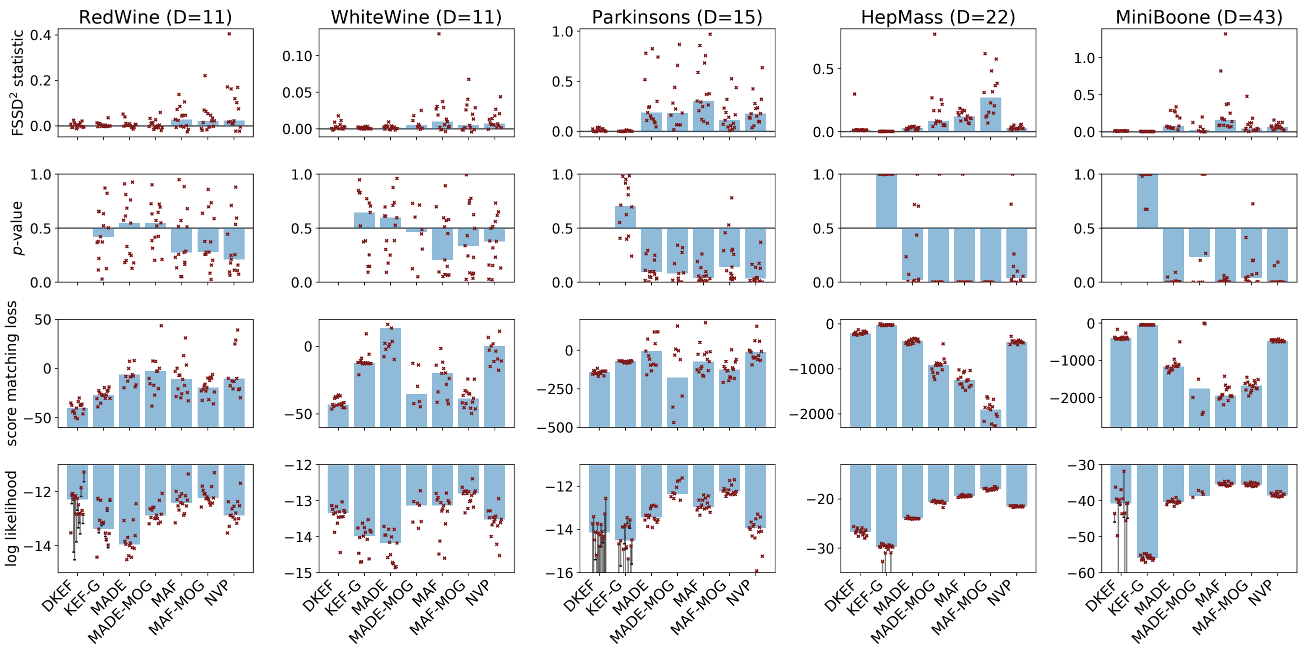

Figure 3: Results on the real datasets; bars show medians, points show each of 15 individual runs, excluding invalid values. (1st row) The estimate of the squared FSSD, a measure of model goodness of fit based on derivatives of the log density; lower is better. (2nd row) The p-value of a test that each model is no better than DKEF in terms of the FSSD; values near 0 indicate that DKEF fits the data significantly better than the other model. (3nd row) Value of the loss (4); lower is better. (4th row) Log-likelihoods; higher is better. DKEF estimates are based on 10^{10} samples for $\hat{Z}_{\boldsymbol{\theta}}$, with vertical lines showing the upper bound on the bias from Proposition 1 (which is often too small to be easily visible).

and most runs on Parkinson, but worse on Hepmass and MiniBoone. FSSD is a somewhat more "global" measure of shape, and is perhaps more weighted towards the bulk of the distribution rather than the tails.[5] In likelihoods, DKEF is comparable to other methods except MADE on Wines but worse on the other, larger, datasets. Note that we trained DKEF while adding Gaussian noise with standard deviation 0.05 to the (whitened) dataset; training without noise improves the score matching loss but harms likelihood, while producing similar results for FSSD.

Results for models with a fixed Gaussian q_0 were similar (Figure 6, in Appendix E.2).

5. Discussion

Learning deep kernels helps make the kernel exponential family practical for large, complex datasets of moderate dimension. We can exploit the closed-form fit of the $\boldsymbol{\alpha}$ vector to optimize kernel and even regularization parameters using a "held-out" loss, in a particularly convenient instance of meta-learning. We are thus able to find smoothness assumptions that fit our particular data, rather than arbitrarily choosing them a priori.

Computational expense makes score matching with deep kernels difficult to scale to models with large kernel networks, limiting the dimensionality of possible targets. Combining with the kernel conditional exponential family might help alleviate that problem by splitting the model up into several separate but marginally complex components. The kernel exponential family, and score matching in general, also struggles to correctly balance datasets with nearly-disjoint components, but it seems to generally learn density *shapes* better than maximum likelihood-based deep approaches.

Acknowledgments

This work was supported by the Gatsby Charitable Foundation. We thank Heishiro Kanagawa for helpful discussions.

discarded those runs for these plots.

[5]With a kernel approaching a Dirac delta, the FSSD^2 is similar to $\mathrm{KSD}^2 \approx \int p_0(\boldsymbol{x})^2 \left\|\nabla \log p(\boldsymbol{x}) - \nabla \log \tilde{p}_{\boldsymbol{\theta}}(\boldsymbol{x})\right\|^2 \mathrm{d}\boldsymbol{x}$; compare to $J = \frac{1}{2} \int p_0(\boldsymbol{x}) \|\nabla \log p(\boldsymbol{x}) - \nabla \log \tilde{p}_{\boldsymbol{\theta}}(\boldsymbol{x})\|^2 \mathrm{d}\boldsymbol{x}$.

Learning Deep Kernels for Exponential Family Densities: Supplementary material

A. DKEFs can be normalized

Proposition 2. *Consider the kernel $k(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{y}))$, where κ is a kernel such that $\kappa(\boldsymbol{a}, \boldsymbol{a}) \le L_\kappa \|\boldsymbol{a}\|^2 + C_\kappa$ and $\boldsymbol{\phi}$ a function such that $\|\boldsymbol{\phi}(\boldsymbol{x})\| \le L_{\boldsymbol{\phi}} \|\boldsymbol{x}\| + C_{\boldsymbol{\phi}}$. Let $q_0(\boldsymbol{x}) = Q\, r_0(\boldsymbol{V}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}))$, where $Q > 0$ is any scalar and r_0 is a product of independent generalized Gaussian densities, with each $\beta_d > 1$:*

$$r_0(\boldsymbol{z}) = \prod_{d=1}^{D} \frac{\beta_d}{2\,\Gamma\left(\frac{1}{\beta_d}\right)} \exp\left(-|z_d|^{\beta_d}\right).$$

(For example, $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ for strictly positive definite $\boldsymbol{\Sigma}$ could be achieved with $\beta_d = 2$ and $\boldsymbol{V}$ the Cholesky factorization of $\boldsymbol{\Sigma}$.) Then, for any function f in the RKHS $\mathcal{H}$ corresponding to k,

$$\int \exp(f(\boldsymbol{x}))\, q_0(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} < \infty.$$

Proof. First, we have that $f(\boldsymbol{x}) = \langle f, k(\boldsymbol{x}, \cdot)\rangle_{\mathcal{H}} \le \|f\|_{\mathcal{H}} \sqrt{k(\boldsymbol{x}, \boldsymbol{x})}$, and

$$k(\boldsymbol{x}, \boldsymbol{x}) = \kappa(\boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x})) \le L_\kappa \|\boldsymbol{\phi}(\boldsymbol{x})\|^2 + C_\kappa \le L_\kappa (L_{\boldsymbol{\phi}} \|\boldsymbol{x}\|^2 + C_{\boldsymbol{\phi}}) + C_\kappa.$$

Combining these two yields

$$f(\boldsymbol{x}) \le \|f\|_{\mathcal{H}} \sqrt{L_\kappa L_{\boldsymbol{\phi}} \|\boldsymbol{x}\|^2 + L_\kappa C_{\boldsymbol{\phi}} + C_\kappa} \le \|f\|_{\mathcal{H}} \sqrt{L_\kappa L_{\boldsymbol{\phi}}} \|\boldsymbol{x}\| + \|f\|_{\mathcal{H}} \sqrt{L_\kappa C_{\boldsymbol{\phi}} + C_\kappa} \le C_0 + C_1 \|\boldsymbol{x}\|,$$

defining $C_1 := \|f\|_{\mathcal{H}} \sqrt{L_\kappa L_{\boldsymbol{\phi}}}$, $C_0 := \|f\|_{\mathcal{H}} \sqrt{L_\kappa C_{\boldsymbol{\phi}} + C_\kappa}$.

Let $\boldsymbol{z} = \boldsymbol{V}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})$, and let C_r be the normalizing constant of r_0, $C_q := \prod_{d=1}^{D} \frac{\beta_d}{2\alpha_d \Gamma\left(\frac{1}{\beta_d}\right)}$. Then

$$\begin{aligned}
\int \exp(f(\boldsymbol{x}))\, q_0(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} &\le \int \exp\left(C_0 + C_1 \|\boldsymbol{x}\|\right) q_0(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} \\
&= Q \exp(C_0)\, \mathbb{E}_{\boldsymbol{z} \sim r_0}\left[\exp\left(C_1 \|\boldsymbol{V}\boldsymbol{z} + \boldsymbol{\mu}\|\right)\right] \\
&\le Q \exp(C_0 + C_1 \|\boldsymbol{\mu}\|)\, \mathbb{E}_{\boldsymbol{z} \sim r_0}\left[\exp\left(C_1 \|\boldsymbol{V}\| \|\boldsymbol{z}\|\right)\right] \\
&\le Q \exp(C_0 + C_1 \|\boldsymbol{\mu}\|)\, \mathbb{E}_{\boldsymbol{z} \sim r_0}\left[\exp\left(C_1 \|\boldsymbol{V}\| \sum_{d=1}^{D} |z_d|\right)\right] \\
&= Q \exp(C_0 + C_1 \|\boldsymbol{\mu}\|) \prod_{d=1}^{D} \mathbb{E}_{\boldsymbol{z} \sim r_0}\left[\exp\left(C_1 \|\boldsymbol{V}\| |z_d|\right)\right].
\end{aligned}$$

We can now show that each of these expectations is finite: letting $C = C_1 \|\boldsymbol{V}\|$,

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{z} \sim r_0}\left[\exp\left(C |z_d|\right)\right] &= \int_{-\infty}^{\infty} \exp\left(C|z|\right) \cdot \frac{\beta}{2\Gamma(1/\beta)} \exp\left(-|z|^\beta\right) \mathrm{d}z \\
&= 2 \frac{\beta}{2\Gamma(1/\beta)} \int_0^\infty \exp\left(Cz - z^\beta\right) \mathrm{d}z \\
&= 2 \frac{\beta}{2\Gamma(1/\beta)} \left(\int_0^s \exp\left(Cz - z^\beta\right) \mathrm{d}z + \int_s^\infty \exp\left(Cz - z^\beta\right) \mathrm{d}z\right)
\end{aligned}$$

for any $s \in (0, \infty)$. The first integral is clearly finite. Picking $s = (2|C|)^{\frac{1}{\beta - 1}}$, so that $|Cz| < \frac{1}{2} z^\beta$ for $z > s$, gives that

$$\int_s^\infty \exp\left(Cz - z^\beta\right) \mathrm{d}z \le \int_s^\infty \exp\left(-\tfrac{1}{2} z^\beta\right) \mathrm{d}z < \frac{1}{\beta} 2^{\frac{1}{\beta}} \Gamma\left(\frac{1}{\beta}\right) < \infty,$$

so that $\int \exp(f(\boldsymbol{x})) q_0(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} < \infty$ as desired. □

The condition on ϕ holds for any ϕ given by a deep network with Lipschitz activation functions, such as the softplus function we use in this work. The condition on κ also holds for a linear kernel (where $L_\kappa = 1$, $C_\kappa = 0$), any translation-invariant kernel ($L_\kappa = 0$, $C_\kappa = \kappa(0, 0)$), or mixtures thereof. If κ is bounded, the integral is finite for any function ϕ.

The given proof would not hold for a quadratic κ, which has been used previously in the literature; indeed, it is clearly possible for such an f to be unnormalizable.

B. Finding the optimal $\boldsymbol{\alpha}$

We will show a slightly more general result than we need, also allowing for an $\|f\|^2_{\mathcal{H}}$ penalty. This result is related to Lemma 4 of **?**, but is more elementary and specialized to our particular needs while also allowing for more types of regularizers.

Proposition 3. *Consider the loss*

$$\hat{J}(f^k_{\boldsymbol{\alpha},\boldsymbol{z}}, \boldsymbol{\lambda}, \mathcal{D}) = \hat{J}(p^k_{\boldsymbol{\alpha},\boldsymbol{z}}, \mathcal{D}) + \frac{1}{2}\left[\lambda_\alpha \|\boldsymbol{\alpha}\|^2 + \lambda_{\mathcal{H}} \|f^k_{\boldsymbol{\alpha},\boldsymbol{z}}\|^2_{\mathcal{H}} + \lambda_C \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D}\left[\partial_d^2 \log \tilde{p}^k_{\boldsymbol{\alpha},\boldsymbol{z}}(\boldsymbol{x}_n)\right]^2\right]$$

where

$$\hat{J}(p^k_{\boldsymbol{\alpha},\boldsymbol{z}}, \mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D}\left[\partial_d^2 \log \tilde{p}^k_{\boldsymbol{\alpha},\boldsymbol{z}}(\boldsymbol{x}_n) + \frac{1}{2}\left(\partial_d \log \tilde{p}^k_{\boldsymbol{\alpha},\boldsymbol{z}}(\boldsymbol{x}_n)\right)^2\right].$$

For fixed k, $\boldsymbol{z}$, and $\boldsymbol{\lambda}$, as long as $\lambda_{\boldsymbol{\alpha}} > 0$ then the optimal $\boldsymbol{\alpha}$ is

$$\begin{aligned}
\boldsymbol{\alpha}(\boldsymbol{\lambda}, k, \boldsymbol{z}, \mathcal{D}) &= \operatorname*{arg\,min}_{\boldsymbol{\alpha}} \hat{J}(f^k_{\boldsymbol{\alpha},\boldsymbol{z}}, \boldsymbol{\lambda}, \mathcal{D}) = -\left(\boldsymbol{G} + \lambda_\alpha \boldsymbol{I} + \lambda_{\mathcal{H}} \boldsymbol{K} + \lambda_C \boldsymbol{U}\right)^{-1} \boldsymbol{b} \\
G_{m,m'} &= \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \partial_d k(\boldsymbol{x}_n, \boldsymbol{z}_m)\, \partial_d k(\boldsymbol{x}_n, \boldsymbol{z}_{m'}) \\
U_{m,m'} &= \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \partial_d^2 k(\boldsymbol{x}_n, \boldsymbol{z}_m)\, \partial_d^2 k(\boldsymbol{x}_n, \boldsymbol{z}_{m'}) \\
K_{m,m'} &= k(\boldsymbol{z}_m, \boldsymbol{z}_{m'}) \\
b_m &= \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \partial_d^2 k(\boldsymbol{x}_n, \boldsymbol{z}_m) + \partial_d \log q_0(\boldsymbol{x}_n)\, \partial_d k(\boldsymbol{x}_n, \boldsymbol{z}_m) + \lambda_C \partial_d^2 \log q_0(\boldsymbol{x}_n)\, \partial_d^2 k(\boldsymbol{x}_n, \boldsymbol{z}_m).
\end{aligned}$$

Proof. We will show that the loss is quadratic in $\boldsymbol{\alpha}$. Note that

$$\begin{aligned}
\frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \partial_d^2 \log \tilde{p}^k_{\boldsymbol{\alpha},\boldsymbol{z}}(\boldsymbol{x}_n) &= \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D}\left[\sum_{m=1}^{M} \alpha_m \partial_d^2 k(\boldsymbol{x}_n, \boldsymbol{z}_m) + \partial_d^2 \log q_0(\boldsymbol{x}_n)\right] \\
&= \boldsymbol{\alpha}^\mathsf{T}\left[\frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \partial_d^2 k(\boldsymbol{x}_n, \boldsymbol{z}_m)\right]_m + \text{const} \\
\frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \frac{1}{2}\left(\partial_d \log \tilde{p}^k_{\boldsymbol{\alpha},\boldsymbol{z}}(\boldsymbol{x}_n)\right)^2 &= \frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \frac{1}{2}\left(\sum_{m,m'=1}^{M} \alpha_m \alpha_{m'} \partial_d k(\boldsymbol{x}_n, \boldsymbol{z}_m) \partial_d k(\boldsymbol{x}_n, \boldsymbol{z}_{m'})\right. \\
&\qquad \left. + 2\sum_{m=1}^{M} \alpha_m \partial_d \log q_0(\boldsymbol{x}_n) \partial_d k(\boldsymbol{x}_n, \boldsymbol{z}_m) + \left(\partial_d \log q_0(\boldsymbol{x}_n)\right)^2\right) \\
&= \frac{1}{2}\boldsymbol{\alpha}^\mathsf{T} \boldsymbol{G} \boldsymbol{\alpha} + \boldsymbol{\alpha}^\mathsf{T}\left[\frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \partial_d \log q_0(\boldsymbol{x}_n) \partial_d k(\boldsymbol{x}_n, \boldsymbol{z}_m)\right] + \text{const.}
\end{aligned}$$

The λ_C term is of the same form, but with second derivatives:

$$\frac{1}{2N}\sum_{n=1}^{N}\sum_{d=1}^{D}\left(\partial_d^2 \log \tilde{p}^k_{\boldsymbol{\alpha},\boldsymbol{z}}(\boldsymbol{x}_n)\right)^2 = \frac{1}{2}\boldsymbol{\alpha}^\mathsf{T} \boldsymbol{U} \boldsymbol{\alpha} + \boldsymbol{\alpha}^\mathsf{T}\left[\frac{1}{N}\sum_{n=1}^{N}\sum_{d=1}^{D} \partial_d^2 \log q_0(\boldsymbol{x}_n) \partial_d^2 k(\boldsymbol{x}_n, \boldsymbol{z}_m)\right] + \text{const.}$$