Differentiable Compositional Kernel Learning for Gaussian Processes

Shengyang Sun [1 2] **Guodong Zhang** [1 2] **Chaoqi Wang** [1 2] **Wenyuan Zeng** [1 2 3] **Jiaman Li** [1 2] **Roger Grosse** [1 2]

Abstract

The generalization properties of Gaussian processes depend heavily on the choice of kernel, and this choice remains a dark art. We present the Neural Kernel Network (NKN), a flexible family of kernels represented by a neural network. The NKN's architecture is based on the composition rules for kernels, so that each unit of the network corresponds to a valid kernel. It can compactly approximate compositional kernel structures such as those used by the Automatic Statistician (Lloyd et al., 2014), but because the architecture is differentiable, it is end-to-end trainable with gradient-based optimization. We show that the NKN is universal for the class of stationary kernels. Empirically we demonstrate NKN's pattern discovery and extrapolation abilities on several tasks that depend crucially on identifying the underlying structure, including time series and texture extrapolation, as well as Bayesian optimization.

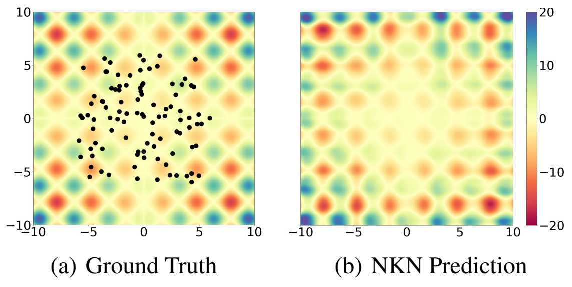


Figure 1. 2-D synthetic data (Left) and extrapolation result using our neural kernel network (Right). The 2-D function is $y = (\cos(2x_1) + \cos(2x_2)) * \sqrt{x_1 x_2}$. Black dots represent 100 training data randomly sampled from $[-6, 6]^2$. This synthetic experiment illustrates NKN's ability to discover and extrapolate patterns.

1. Introduction

Gaussian processes (GPs) are a powerful and widely used class of models due to their nonparametric nature, explicit representation of posterior uncertainty, and ability to flexibly model a variety of structures in data. However, patterns of generalization in GP depend heavily on the choice of kernel function (Rasmussen, 1999); different kernels can impose widely varying modeling assumptions, such as smoothness, linearity, or periodicity. Capturing appropriate kernel structures can be crucial for interpretability and extrapolation (Duvenaud et al., 2013; Wilson & Adams, 2013). Even for experts, choosing GP kernel structures remains a dark art.

GPs' strong dependence on kernel structures has motivated work on automatic kernel learning methods. Sometimes this can be done by imposing a specific kind of structure: e.g., Bach (2009); Duvenaud et al. (2011) learned kernel structures which were additive over subsets of the variables. A more expressive space of kernels is spectral mixtures (Wilson & Adams, 2013; Kom Samo & Roberts, 2015; Remes et al., 2017), which are based on spectral domain summations. For example, spectral mixture (SM) kernels (Wilson & Adams, 2013) approximate all stationary kernels using Gaussian mixture models in the spectral domain. Deep kernel learning (DKL) (Wilson et al., 2016) further boosted the expressiveness by transforming the inputs of spectral mixture base kernel with a deep neural network. However, the expressiveness of DKL still depends heavily on the kernel placed on the output layer.

In another line of work, Duvenaud et al. (2013) defined a context-free grammar of kernel structures based on the composition rules for kernels. Due to its compositionality, this grammar could express combinations of properties such as smoothness, linearity, or periodicity. They performed a greedy search over this grammar to find a kernel struture which matched the input data. Using the learned structures, they were able to produce sensible extrapolations and interpretable decompositions for time series datasets. Lloyd et al. (2014) extended this work to an Automatic Statistician which automatically generated natural language reports. All of these results depended crucially on the compositionality of the underlying space. The drawback was that discrete search over the kernel grammar is very expensive, often requiring hours of computation even for short time series.

[1]Department of Computer Science, University of Toronto, Toronto, ON, CA. [2]Vector Institute. [3]Uber Advanced Technologies Group, Toronto, ON, CA. Correspondence to: Shengyang Sun <ssy@cs.toronto.edu>.

Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

In this paper, we propose the Neural Kernel Network (NKN), a flexible family of kernels represented by a neural network. The network's first layer units represent primitive kernels, including those used by the Automatic Statistician. Subsequent layers are based on the composition rules for kernels, so that each intermediate unit is itself a valid kernel. The NKN can compactly approximate the kernel structures from the Automatic Statistician grammar, but is fully differentiable, so that the kernel structures can be learned with gradient-based optimization. To illustrate the flexibility of our approach, Figure 1 shows the result of fitting an NKN to model a 2-D function; it is able to extrapolate sensibly.

We analyze the NKN's expressive power for various choices of primitive kernels. We show that the NKN can represent nonnegative polynomial functions of its primitive kernels, and from this demonstrate universality for the class of stationary kernels. Our universality result holds even if the width of the network is limited, analogously to Sutskever & Hinton (2008). Interestingly, we find that the network's representations can be made significantly more compact by allowing its units to represent complex-valued kernels, and taking the real component only at the end.

We empirically analyze the NKN's pattern discovery and extrapolation abilities on several tasks that depend crucially on identifying the underlying structure. The NKN produces sensible extrapolations on both 1-D time series datasets and 2-D textures. It outperforms competing approaches on regression benchmarks. In the context of Bayesian optimization, it is able to optimize black-box functions more efficiently than generic smoothness kernels.

2. Background

2.1. Gaussian Process Regression

A Gaussian process (GP) defines a distribution $p(f)$ over functions $\mathcal{X} \to \mathcal{R}$ for some domain $\mathcal{X}$. For any finite set $\{\mathbf{x}_1, ..., \mathbf{x}_n\} \subset \mathcal{X}$, the function values $\mathbf{f} = (f(\mathbf{x}_1), f(\mathbf{x}_2), ..., f(\mathbf{x}_n))$ have a multivariate Gaussian distribution. Gaussian processes are parameterized by a mean function $\mu(\cdot)$ and a covariance function or kernel function $k(\cdot, \cdot)$. The marginal distribution of function values is given by

$$\mathbf{f} \sim \mathcal{N}(\mu, \mathbf{K}_{XX}), \tag{1}$$

where $\mathbf{K}_{XX}$ denotes the matrix of $k(\mathbf{x}_i, \mathbf{x}_j)$ for all (i, j).

Assume we are given a set of training input-output pairs, $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n = (\mathbf{X}, \mathbf{y})$, and each target y_n is generated from the corresponding $f(\mathbf{x}_n)$ by adding independent Gaussian noise; i.e.,

$$y_n = f(\mathbf{x}_n) + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, \sigma^2) \tag{2}$$

As the prior on f is a Gaussian process and the likelihood is Gaussian, the posterior on f is also Gaussian. We can use this to make predictions $p(y_*|\mathbf{x}_*, \mathcal{D})$ in closed form:

$$\begin{aligned} &p(y_*|\mathbf{x}_*, \mathcal{D}) = \mathcal{N}(\mu_*, \sigma_*^2) \\ &\mu_* = \mathbf{K}_{*X}(\mathbf{K}_{XX} + \sigma^2\mathbf{I})^{-1}\mathbf{y} \\ &\sigma_*^2 = \mathbf{K}_{**} - \mathbf{K}_{*X}(\mathbf{K}_{XX} + \sigma^2\mathbf{I})^{-1}\mathbf{K}_{X*} + \sigma^2 \end{aligned} \tag{3}$$

Here we assume zero mean function for f. Most GP kernels have several hyperparameters θ which can be optimized jointly with σ to maximize the log marginal likelihood,

$$\mathcal{L}(\theta) = \ln p(\mathbf{y}|\mathbf{0}, \mathbf{K}_{XX} + \sigma^2\mathbf{I}) \tag{4}$$

2.2. Bochner's Theorem

Gaussian Processes depend on specifying a kernel function $k(x, x')$, which acts as a similarity measure between inputs.

Definition 1. *Let $\mathcal{X}$ be a set, and k be a conjugate symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ is a positive definite kernel if $\forall \boldsymbol{x_1}, \cdots, \boldsymbol{x_n} \in \mathcal{X}$ and $\forall \boldsymbol{c_1}, \cdots, \boldsymbol{c_n} \in \mathbb{C}$,*

$$\sum_{i,j=1}^{n} \boldsymbol{c_i}\overline{\boldsymbol{c_j}}k(\boldsymbol{x_i}, \boldsymbol{x_j}) \geq 0, \tag{5}$$

where the bar denotes the complex conjugate. Bochner's Theorem (Bochner, 1959) establishes a bijection between complex-valued stationary kernels and positive finite measures using Fourier transform, thus providing an approach to analyze stationary kernels in the spectral domain (Wilson & Adams, 2013; Kom Samo & Roberts, 2015).

Theorem 1. *(Bochner) A complex-valued function k on $\mathbb{R}^d$ is the covariance function of a weakly stationary mean square continuous complex-valued random process on $\mathbb{R}^d$ if and only if it can be represented as*

$$k(\boldsymbol{\tau}) = \int_{\mathbb{R}^P} \exp(2\pi i \boldsymbol{w}^\top \boldsymbol{\tau}) \psi(\mathrm{d}\boldsymbol{w}) \tag{6}$$

where ψ is a positive and finite measure. If ψ has a density $S(\boldsymbol{w})$, then S is called the spectral density or power spectrum of k. S and k are Fourier duals.

2.3. Automatic Statistician

For compositional kernel learning, the Automatic Statistician (Lloyd et al., 2014; Duvenaud et al., 2013) used a compositional space of kernels defined as sums and products of a small number of primitive kernels. The primitive kernels included:

- radial basis functions, corresponding to smooth functions. $\mathrm{RBF}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2l^2})$
- periodic. $\mathrm{PER}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp(-\frac{2\sin^2(\pi\|\mathbf{x}-\mathbf{x}'\|/p)}{l^2})$
- linear kernel. $\mathrm{LIN}(\mathbf{x}, \mathbf{x}') = \sigma^2 \mathbf{x}^\top \mathbf{x}'$

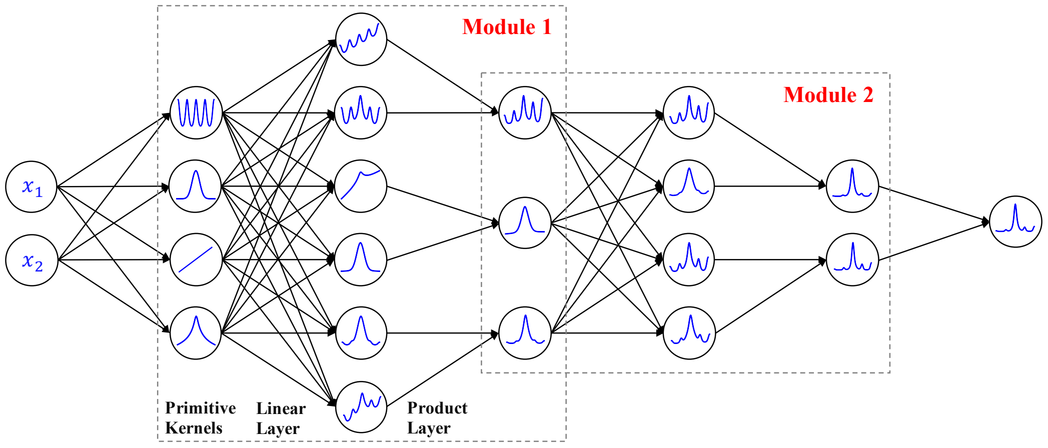


Figure 2. Neural Kernel Network: each module consists of a **Linear** layer and a **Product** layer. NKN is based on compositional rules for kernels, thus every individual unit itself represents a kernel.

- rational quadratic, corresponding to functions with multiple scale variations. $\mathrm{RQ}(\mathbf{x}, \mathbf{x}') = \sigma^2(1 + \frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\alpha l^2})^{\frac{1}{\alpha}}$
- white noise. $\mathrm{WN}(\mathbf{x}, \mathbf{x}') = \sigma^2 \delta_{\mathbf{x},\mathbf{x}'}$
- constant kernel. $\mathrm{C}(\mathbf{x}, \mathbf{x}') = \sigma^2$

The Automatic Statistician searches over the compositional space based on three search operators.

1. Any subexpression $\mathcal{S}$ can be replaced with $\mathcal{S} + \mathcal{B}$, where $\mathcal{B}$ is any primitive kernel family.
2. Any subexpression $\mathcal{S}$ can be replaced with $\mathcal{S} \times \mathcal{B}$, where $\mathcal{B}$ is any primitive kernel family.
3. Any primitive kernel $\mathcal{B}$ can be replaced with any other primitive kernel family $\mathcal{B}'$.

The search procedure relies on a greedy search: at every stage, it searches over all subexpressions and all possible operators, then chooses the highest scoring combination. To score kernel families, it approximates the marginal likelihood using the Bayesian information criterion (Schwarz et al., 1978) after optimizing to find the maximum-likelihood kernel parameters.

3. Neural Kernel Networks

In this section, we introduce the Neural Kernel Network (NKN), a neural net which computes compositional kernel structures and is end-to-end trainable with gradient-based optimization. The input to the network consists of two vectors $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$, and the output $k(\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}$ (or $\mathbb{C}$) is the kernel value. Our NKN architecture is based on well-known composition rules for kernels:

Lemma 2. *For kernels k_1, k_2*

- *For $\lambda_1, \lambda_2 \in \mathbb{R}^+$, $\lambda_1 k_1 + \lambda_2 k_2$ is a kernel.*
- *The product $k_1 k_2$ is a kernel.*

We design the architecture such that every unit of the network computes a kernel, although some of those kernels may be complex-valued.

3.1. Architecture

The first layer of the NKN consists of a set of primitive kernels. Subsequent layers alternate between linear combinations and products. Since the space of kernels is closed under both operations, each unit in the network represents a kernel. Linear combinations and products can be seen as OR-like and AND-like operations, respectively; this is a common pattern in neural net design (LeCun et al., 1989; Poon & Domingos, 2011). The full architecture is illustrated in Figure 2.

Primitive kernels. The first layer of the network consists of a set of primitive kernel families with simple functional forms. While any kernels can be used here, we use the RBF, PER, LIN, and RQ kernels from the Automatic Statistician (see Section 2.3) because these express important structural motifs for GPs. Each of these kernel families has an associated set of hyperparameters (such as lengthscales or variances), and instantiating the hyperparameters gives a kernel. These hyperparameters are treated as parameters (weights) in this layer of the network, and are optimized with the rest of the network. Note that it may be advantageous to have multiple copies of each primitive kernel so that they can be instantiated with different hyperparameters.

Linear layers. The Linear layer closely resembles a fully connected layer in deep neural networks, with each layer $\mathbf{h}_l = \mathbf{W}_l \mathbf{h}_{l-1}$ representing a nonnegative linear combination of units in the previous layer (i.e. $\mathbf{W}_l$ is a nonnegative matrix). In practice, we use the parameterization

$\mathbf{W}_l = \log(1+\exp(\mathbf{A}_l))$ to enforce the nonnegativity constraint. (Here, $\exp$ is applied elementwise.)

The Linear layer can be seen as a OR-like operation: two points are considered similar if either kernel has a high value, while the Linear layer further controls the balance using trainable weights.

Product layers. The Product layer introduces multiplication, in that each unit is the product of several units in the previous layer. This layer has a fixed connectivity pattern and no trainable parameters. While this fixed structure may appear restrictive, Section 3.3 shows that it does not restrict the expressiveness of the network.

The Product layer can be seen as an AND-like operation: two points are considered similar if both constituent kernels have large values.

Activation functions. Analogously to ordinary neural nets, each layer may also include a nonlinear activation function, so that $\mathbf{h}_l = f(\mathbf{z}_l)$, where $\mathbf{z}_l$, the pre-activations, are the result of a linear combination or product. However, f must be selected with care in order to ensure closure of the kernels. Polynomials with positive coefficients, as well as the exponential function $f(z) = e^z$, fulfill this requirement.

Complex-valued kernels. Allowing units in NKN to represent complex-valued kernels as in Definition 1 and take the real component only at the end, can make the network's representations significantly more compact. As complex-valued kernels also maintain closure under summation and multiplication (Yaglom, 2012), additional modifications are unnecessary. In practice, we can include $\exp(i\boldsymbol{\mu}^\top\boldsymbol{\tau})$ in our primitive kernels.

3.2. Learning

Optimization. All trainable parameters can be grouped into two categories: (1) parameters of primitive kernels, e.g., lengthscale in an RBF kernel; (2) parameters of Linear layers. We jointly learn these parameters by maximizing the marginal likelihood $\mathcal{L}(\boldsymbol{\theta})$. Since the NKN architecture is differentiable, we can jointly fit all parameters using gradient-based optimization.

Computational Cost. NKN introduces small computational overhead. Suppose we have N data points and m connections in the NKN; the computational cost of the forward pass is $O(N^2m)$. Note that a moderately-sized NKN, as we used in our experiments[1], has only tens of parameters, and the main computational bottleneck in training lies in inverting kernel matrix, which is an $O(N^3)$ operation; therefore, NKN incurs only small per-iteration overhead compared to ordinary GP training.

[1]In our experiments, we found 1 or 2 modules work very well. But it might be advantageous to use more modules in other tasks.

3.3. Universality

In this section, we analyze the expressive power of the NKN, and in particular its ability to approximate arbitrary stationary kernels. Our analysis provides insight into certain design decisions for the NKN: in particular, we show that the NKN can approximate some stationary kernels much more compactly if the units of the network are allowed to take complex values. Furthermore, we show that the fixed structure of the product layers does not limit what the network can represent.

Definition 2. *For kernels $\{k_j\}_{j=1}^n$, a kernel k is positive-weighted polynomial (PWP) of these kernels if $\exists T \in \mathbb{N}$ and $\{w_t, \{p_{tj}\}_{j=1}^n | w_i \in \mathbb{R}^+, p_{tj} \in \mathbb{N}\}_{t=0}^T$, such that*

$$k(x,y) = \sum_{t=1}^T w_t \prod_{j=1}^n k_j^{p_{tj}} \tag{7}$$

holds for all $x, y \in \mathbb{R}$. Its degree is $\max_t \sum_{j=1}^n p_{tj}$.

Composed of summation and multiplication, the NKN naturally forms a positive-weighted polynomial of primitive kernels. Although NKN adopts a fixed multiplication order in the Product layer, the following theorem shows that this fixed architecture doesn't undermine NKN's expressiveness (proof in Appendix D).

Theorem 3. *Given B primitive kernels,*

- *An NKN with width $2B+6$ can represent any PWP of primitive kernels.*
- *An NKN with width 2^{Bp+1} and p Linear-Product modules can represent any PWP with degree no more than 2^p.*

Interestingly, NKNs can sometimes approximate (real-valued) kernels more compactly if the hidden units are allowed to represent complex-valued kernels, and the real part is taken only at the end. In particular, we give an example of a spectral mixture kernel class which can be represented with an NKN with a single complex-valued primitive kernel, but whose real-valued NKN representation requires a primitive kernel for each mixture component (proof in Appendix E).

Example 1. *Define a d-dimensional spectral mixture kernel with $n+1$ components, $k^*(\boldsymbol{\tau}) = \sum_{t=1}^{n+1} \binom{n}{2}^{2t} \cos(4^t \mathbf{1}^\top \boldsymbol{\tau})$. Then $\exists \epsilon > 0$, such that $\forall \{\boldsymbol{\mu_t}\}_{t=1}^n$, and any PWP of $\{\cos(\boldsymbol{\mu_t}^\top \boldsymbol{\tau})\}_{t=1}^n$ denoted as $\bar{k}$,*

$$\max_{\boldsymbol{\tau}\in\mathbb{R}^d} |\bar{k}(\boldsymbol{\tau}) - k^*(\boldsymbol{\tau})| > \epsilon \tag{8}$$

In contrast, k^* can be represented as the real part of a PWP of only one complex-valued primitive kernel $e^{i\mathbf{1}^\top \boldsymbol{\tau}}$,

$$k^*(\boldsymbol{\tau}) = \Re\{\sum_{t=1}^{n+1} \binom{n}{2}^{2t} [e^{i\mathbf{1}^\top \boldsymbol{\tau}}]^{4^t}\} \tag{9}$$

We find that an NKN with small width can approximate any complex-valued stationary kernel, as shown in the following theorem (Proof in Appendix F).

Theorem 4. *For any d-dimensional complex-valued stationary kernel k^* and $\epsilon \in \mathbb{R}^+$, $\exists \{\boldsymbol{\gamma_j}\}_{j=1}^{d}$, $\{\boldsymbol{\mu_j}\}_{j=1}^{2d}$, and an NKN $\bar{k}$ with primitive kernels $\{\exp(-2\pi^2 \|\boldsymbol{\tau} \odot \boldsymbol{\gamma_j}\|^2)\}_{j=1}^{d}$, $\{\exp(i\boldsymbol{\mu_j}^\top \boldsymbol{\tau})\}_{j=1}^{2d}$, and width no more than $6d+6$, such that*

$$\max_{\tau \in \mathbb{R}^d} |\bar{k}(\boldsymbol{\tau}) - k^*(\boldsymbol{\tau})| < \epsilon \tag{10}$$

Beyond approximating stationary kernels, NKN can also capture non-stationary structure by incorporating non-stationary primitive kernels. In Appendix G, we prove that with the proper choice of primitive kernels, NKN can approximate a broad family of non-stationary kernels called generalized spectral kernels (Kom Samo & Roberts, 2015).

4. Related Work

Additive kernels (Duvenaud et al., 2011) are linear combinations of kernels over individual dimensions or groups of dimensions, and are a promising method to combat the curse of dimensionality. While additive kernels need an exponential number of multiplication terms in the input dimension, hierarchical kernel learning (HKL) (Bach, 2009) presents a similar kernel except selecting only a subset to get a polynomial number of terms. However, this subset selection imposes additional optimization difficulty.

Based on Bochner's theorem, there is another a line of work on designing kernels in the spectral domain, including sparse spectrum kernels (SS) (Lázaro-Gredilla et al., 2010); spectral mixture (SM) kernels (Wilson & Adams, 2013); generalized spectral kernels (GSK) (Kom Samo & Roberts, 2015) and generalized spectral mixture (GSM) kernels (Remes et al., 2017). Though these approaches often extrapolate sensibly, capturing complex covariance structure may require a large number of mixture components.

The Automatic Statistician (Duvenaud et al., 2013; Lloyd et al., 2014; Malkomes et al., 2016) used a compositional grammar of kernel structures to analyze datasets and provide natural language reports. In each stage, it considered all production rules and used the one that resulted in the largest log-likelihood improvement. Their model showed good extrapolation for many time series tasks, attributed to the recovery of underlying structure. However, it relied on greedy discrete search over kernel and operator combinations, making it computational expensive, even for small time series datasets.

There have been several attempts (Hinton & Salakhutdinov, 2008; Wilson et al., 2016) to combine neural networks with Gaussian processes. Specifically, they used a fixed kernel structure on top of the hidden representation of a neural network. This is complementary to our work, which focuses on using neural networks to infer the kernel structure itself. Both approaches could potentially be combined.

Instead of represeting kernel parametrically, Oliva et al. (2016) modeled random feature dimension with stick breaking prior and Tobar et al. (2015) generated functions as the convolution between a white noise process and a linear filter drawn from GP. These approaches offer much flexibility but also incur challenges in training.

5. Experiments

We conducted a series of experiments to measure the NKN's predictive ability in several settings: time series, regression benchmarks, and texture images. We focused in particular on extrapolation, since this is a strong test of whether it has uncovered the underlying structure. Furthermore, we tested the NKN on Bayesian Optimization, where model structure and calibrated uncertainty can each enable more efficient exploration. Code is available at `git@github.com:ssydasheng/Neural-Kernel-Network.git`

5.1. Time Series Extrapolation

We first conducted experiments time series datasets to study extrapolation performance. For all of these experiments, as well as the 2-d experiment in Figure 1, we used the same NKN architecture and training setup (Appendix J.1).

We validated the NKN on three time series datasets introduced by Duvenaud et al. (2013): airline passenger volume (Airline), Mauna Loa atmospheric CO_2 concentration (Mauna), and solar irradiance (Solar). Our focus is on extrapolation, since this is a much better test than interpolation for whether the model has learned the underlying structure.

We compared the NKN with the Automatic Statistician (Duvenaud et al., 2013); both methods used RBF, RQ, PER and LIN as the primitive kernels. In addition, because many time series datasets appear to contain a combination of seasonal patterns, long-term trends, and medium-scale variability, we also considered a baseline consisting of sums of PER, LIN, RBF, and Constant kernels, with trainable weights and kernel parameters. We refer to this baseline as "heuristic".

The results for Airline are shown in Figure 3, while the

Table 1. Average test RMSE and log-likelihood for regression benchmarks with random splits.

	Test RMSE				Test log-likelihood			
Dataset	BBB	GP-RBF	GP-SM4	GP-NKN	BBB	GP-RBF	GP-SM4	GP-NKN
Boston	3.171±0.149	2.753±0.137	2.979±0.162	**2.506±0.150**	-2.602±0.031	-2.434±0.069	-2.518±0.107	**-2.394±0.080**
Concrete	5.678±0.087	4.685±0.137	3.730±0.190	**3.688±0.249**	-3.149±0.018	-2.948±0.025	**-2.662±0.053**	-2.842±0.263
Energy	0.565±0.018	0.471±0.013	0.316±0.018	**0.254±0.020**	-1.500±0.006	-0.673±0.035	-0.320±0.089	**-0.213±0.162**
Kin8nm	0.080±0.001	0.068±0.001	**0.061±0.000**	0.067±0.001	1.111±0.007	1.287±0.007	**1.387±0.006**	1.291±0.006
Naval	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	6.143±0.032	9.557±0.001	**9.923±0.000**	9.916±0.000
Pow. Plant	4.023±0.036	3.014±0.068	2.781±0.071	**2.675±0.074**	-2.807±0.010	-2.518±0.020	-2.450±0.022	**-2.406±0.023**
Wine	0.643±0.012	0.597±0.013	0.579±0.012	**0.523±0.011**	-0.977±0.017	0.723±0.067	0.652±0.060	**0.852±0.064**
Yacht	1.174±0.086	0.447±0.083	0.436±0.070	**0.305±0.060**	-2.408±0.007	-0.714±0.449	-0.891±0.523	**-0.116±0.270**

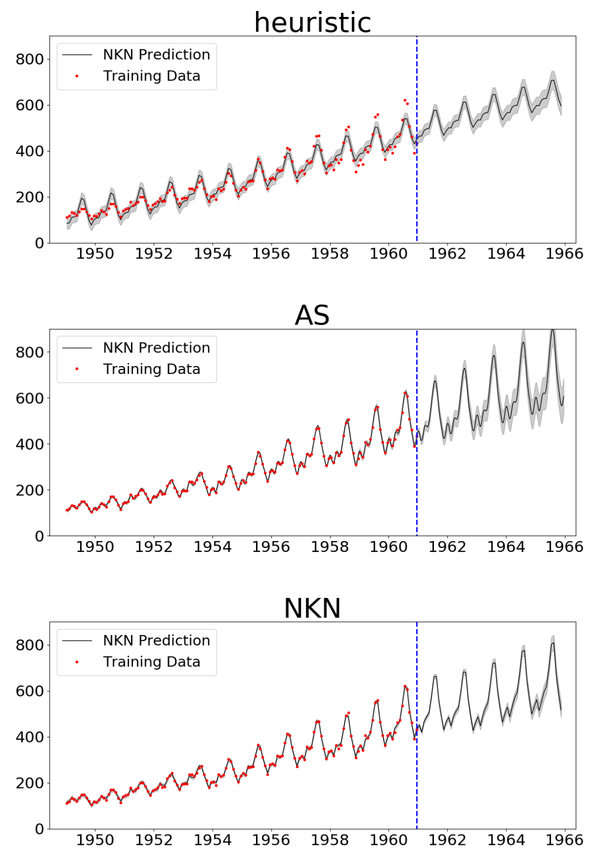


Figure 3. Extrapolation results of NKN on the Airline dataset. “Heuristic” denotes linear combination of RBF, PER, LIN, and Constant kernels. AS represents Automatic Statistician (Duvenaud et al., 2013). The **red circles** are the training points, and the curve after the **blue dashed line** is the extrapolation result. Shaded areas represent 1 standard deviation.

results for Mauna and Solar are shown in Figures 8 and 9 in the Appendix. All three models were able to capture the periodic and increasing patterns. However, the heuristic kernel failed to fit the data points well or capture the increasing amplitude, stemming from its lack of PER*LIN structure. In comparison, both AS and NKN fit the training points perfectly, and generated sensible extrapolations. However, the NKN was far faster to train because it avoided discrete search: for the Airline dataset, the NKN took only 201 seconds, compared with 6147 seconds for AS.

5.2. Regression Benchmarks

5.2.1. Random Training/Test Splits

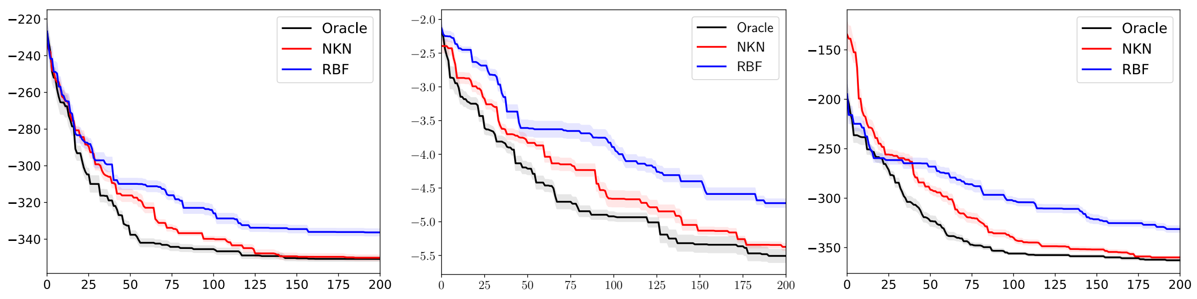
To evaluate the predictive performance of NKN, we first conducted experiments on regression benchmark datasets from the UCI collection (Asuncion & Newman, 2007). Following the settings in Hernández-Lobato & Adams (2015), the datasets were randomly split into training and testing sets, comprising 90% and 10% of the data respectively. This splitting process was repeated 10 times to reduce variability. We compared NKN to RBF and SM (Wilson & Adams, 2013) kernels, and the popular Bayesian neural network method Bayes-by-Backprop (BBB) (Blundell et al., 2015). For the SM kernel, we used 4 mixture components, so we denote it as SM-4. For all experiments, the NKN uses 6 primitive kernels including 2 RQ, 2 RBF, and 2 LIN. The following layers are organized as Linear8-Product4-Linear4-Product2-Linear1.[2] We trained both the variance and d-dimensional lengthscales for all kernels. As a result, for d dimensional inputs, SM-4 has $8d+12$ trainable parameters and NKN has $4d+85$ parameters.

As shown in Table 1, BBB performed worse than the Gaussian processes methods on all datasets. On the other hand, NKN and SM-4 performed consistently better than the standard RBF kernel in terms of both RMSE and log-likelihoods. Moreover, the NKN outperformed the SM-4 kernel on all datasets other than Naval and Kin8nm.

5.2.2. Measuring Extrapolation with PCA splits

Since the experiments just presented used random training/test splits, they can be thought of as measuring interpolation performance. We are also interested in measuring extrapolation performance, since this is a better measure of whether the model has captured the underlying structure. In

[2]The number for each layer represents the output dimension.



(a) Stybtang (b) Michalewicz (c) Stybtang-transform

Figure 4. Bayesian optimization on three tasks. The oracle kernel has the true additive structure of underlying function. Shaded error bars represent 0.2 standard deviations over 10 runs.

Table 2. Test RMSE and log-likelihood for the PCA-split regression benchmarks. N denotes the number of data points.

		Test RMSE			Test log-likelihood		
Dataset	N	GP-RBF	GP-SM	GP-NKN	GP-RBF	GP-SM	GP-NKN
Boston	506	6.390	8.600	**4.658**	-4.063	-4.404	**-3.557**
Concrete	1031	8.531	7.591	**6.242**	-3.246	-3.285	**-3.112**
Energy	768	0.974	**0.447**	0.459	-1.297	**-0.564**	-0.649
Kin8nm	8192	0.093	**0.065**	0.086	0.998	**1.322**	1.057
Naval	11934	0.000	**0.000**	0.000	7.222	**9.037**	6.442
Pow. Plant	9568	4.768	3.931	**3.742**	-3.076	-2.877	**-2.763**
Wine	1599	0.701	0.660	**0.650**	-1.076	-1.002	**-0.972**
Yacht	308	1.190	1.736	**0.528**	-2.896	-2.768	**-0.694**

order to test this, we sorted the data points according to their projection onto the top principal component of the data. The top 1/15 and bottom 1/15 of the data were used as test data, and the remainder was used as training data.

We compared NKN with standard RBF and SM kernels using the same architectural settings as in the previous section. All models were trained for 20,000 iterations. To select the mixture number of SM kernels, we further subdivided the training set into a training and validation set, using the same PCA-splitting method as described above. For each dataset, we trained SM on the sub-training set using $\{1, 2, 3, 4\}$ mixture components and selected the number based on validation error. (We considered up to 4 mixture components in order to roughly match the number of parameters for NKN.) Then we retrained the SM kernel using the combined training and validation sets. The resulting test RMSE and log-likelihood are shown in Table 2.

As seen in Table 2, all three kernels performed significantly worse compared with Table 1, consistent with the intuition that extrapolation is more difficult that interpolation. However, we can see NKN outperformed SM for most of the datasets. In particular, for small datasets (and hence more chance to overfit), NKN performed better than SM by a substantial margin, with the exception of the *Energy* dataset. This demonstrates the NKN was better able to capture the underlying structure, rather than overfitting the training points.

5.3. Bayesian Optimization

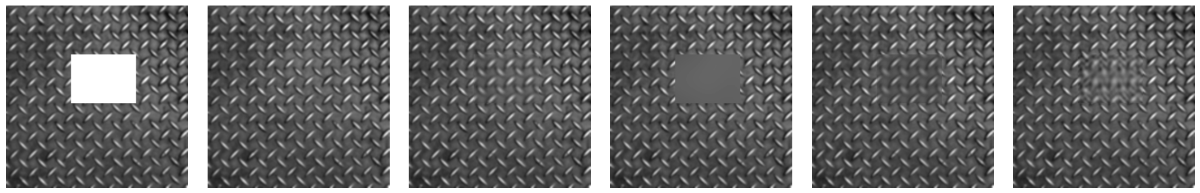
Bayesian optimization (Brochu et al., 2010; Snoek et al., 2012) is a technique for optimization expensive black-box functions which repeatedly queries the function, fits a surrogate function to past queries, and maximizes an acquisition function to choose the next query. It's important to model both the predictive mean (to query points that are likely to perform well) and the predictive variance (to query points that have high uncertainty). Typically, the surrogate functions are estimated using a GP with a simple kernel, such as Matern. But simple kernels lead to inefficient exploration due to the curse of dimensionality, leading various researchers to consider additive kernels (Kandasamy et al., 2015; Gardner et al., 2017; Wang et al., 2017). Since additivity is among the patterns the NKN can learn, we were interested in testing its performance on Bayesian optimization tasks with additive structure. We used Expectated Improvement (EI) to perform BO.

Following the protocol in Kandasamy et al. (2015); Gardner et al. (2017); Wang et al. (2017), we evaluated the performance on three toy function benchmarks with additive structure,

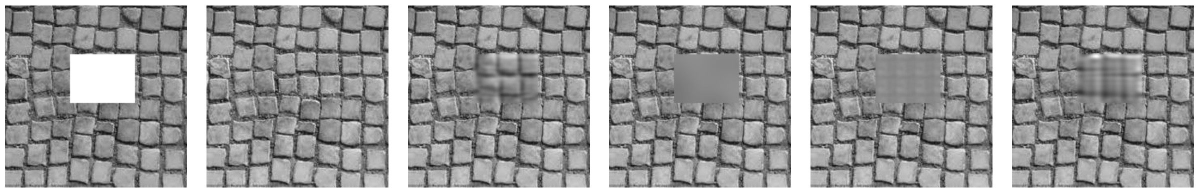
$$f(\mathbf{x}) = \sum_{i=1}^{|P|} f_i(\mathbf{x}[P_i]) \tag{11}$$

The d-dimensional Styblinski-Tang function and Michalewicz function have fully additive structure with independent dimensions. In our experiment, we set $d = 10$ and explored the function over domain $[-4, 4]^d$ for Styblinski-Tang and $[0, \pi]^d$ for Michalewicz. We also experimented with a transformed Styblinski-Tang function, which applies Styblinski-Tang function on partitioned dimension groups.

For modelling additive functions with GP, the kernel can decompose as a summation between additive groups as well. $k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{|P|} k_i(\mathbf{x}[P_i], \mathbf{x}'[P_i])$. We considered an oracle kernel, which was a linear combination of RBF kernels



(a) Observations (b) Ground Truth (c) NKN-4 (d) RBF (e) PER (f) SM-10



(g) Observations (h) Ground Truth (i) NKN-4 (j) RBF (k) PER (l) SM-10

Figure 5. Texture Extrapolation on metal thread plate (top) and paved pattern (bottom).

corresponding to the true additive structure of the function. Both the kernel parameters and the combination coefficients were trained with maximum likelihood. We also tested the standard RBF kernel without additive structure. For the NKN, we used d RBF kernels over individual input dimensions as the primitive kernels. The following layers were arranged as Linear8-Product4-Linear4-Product2-Linear1. Note that, although the primitive kernels corresponded to seperate dimensions, NKN can represent additive structure through these linear combination and product operations. In all cases, we used Expected Improvement as the acquisition function.

As shown in Figure 4, for all three benchmarks, the oracle kernel not only converged faster than RBF kernel, but also found smaller function values by a large margin. In comparsion, we can see that although NKN converged slower than oracle in the beginning, it caught up with oracle eventually and reached the same function value. This suggests that the NKN is able to exploit additivity for Bayesian optimization.

5.4. Texture Extrapolation

Based on Wilson et al. (2014), we evaluated the NKN on texture exploration, a test of the network's ability to learn local correlations as well as complex quasi-periodic patterns. From the original 224×224 images, we removed a 60×80 region, as shown in Figure 5(a). From a regression perspective, this corresponds to 45376 training examples and 4800 test examples, where the inputs are 2-D pixel locations and the outputs are pixel intensities. To scale our algorithms to this setting, we used the approach of Wilson et al. (2014). In particular, we exploited the grid structure of texture images to represent the kernel matrix for the full image as a Kronecker product of kernel matrices along each dimension (Saatçi, 2012). Since some of the grid points are unobserved, we followed the algorithm in Wilson et al. (2014) complete the grid with imaginary observations, and placed infinite measurement noise on these observations.

To reconstruct the missing region, we used an NKN with 4 primitive kernels: LIN, RBF, RQ, and PER. As shown in Figure 5(c), our NKN was able to learn and extrapolate complex image patterns. As baselines, we tested RBF and PER kernels; those results are shown in Figure 5(d) and Figure 5(e). The RBF kernel was unable to extrapolate to the missing region, while the PER kernel was able to extrapolate beyond the training data since the image pattern is almost exactly periodic. We also tested the spectral mixture (SM) kernel, which has previously shown promising results in texture extrapolation. Even with 10 mixture components, its extrapolations were blurrier compared to those of the NKN. The second row shows extrapolations on an irregular paved pattern, which we believe is more difficult. The NKN still provided convincing extrapolation. By contrast, RBF and PER kernels were unable to capture enough information to reconstruct the missing region.

6. Conclusion

We proposed the Neural Kernel Network (NKN), a differentiable architecture for compositional kernel learning. Since the architecture is based on the composition rules for kernels, the NKN can compactly approximate the kernel structures from the Automatic Statistician (AS) grammar. But because the architecture is differentiable, the kernel can be learned orders-of-magnitude faster than the AS using gradient-based optimization. We demonstrated the universality of the NKN for the class of stationary kernels, and showed that the network's representations can be made significantly more compact using complex-valued kernels. Empirically, we found the NKN is capable of pattern discovery and extrapolation in both 1-D time series datasets and 2-D textures, and can find and exploit additive structure for Bayesian Optimization.

Acknowledgements

We thank David Duvenaud and Jeongseop Kim for their insightful comments and discussions on this project. SS was supported by a Connaught New Researcher Award and a Connaught Fellowship. GZ was supported by an NSERC Discovery Grant.

References

Asuncion, A. and Newman, D. UCI machine learning repository, 2007.

Bach, F. R. Exploring large feature spaces with hierarchical multiple kernel learning. In *Advances in neural information processing systems*, pp. 105–112, 2009.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Bochner, S. *Lectures on Fourier Integrals: With an Author's Supplement on Monotonic Functions, Stieltjes Integrals and Harmonic Analysis; Translated from the Original German by Morris Tenenbaum and Harry Pollard*. Princeton University Press, 1959.

Brochu, E., Cora, V. M., and De Freitas, N. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

Csiszar, I. and Körner, J. *Information theory: coding theorems for discrete memoryless systems*. Cambridge University Press, 2011.

Duvenaud, D., Lloyd, J. R., Grosse, R., Tenenbaum, J. B., and Ghahramani, Z. Structure discovery in nonparametric regression through compositional kernel search. *arXiv preprint arXiv:1302.4922*, 2013.

Duvenaud, D. K., Nickisch, H., and Rasmussen, C. E. Additive Gaussian processes. In *Advances in neural information processing systems*, pp. 226–234, 2011.

Gardner, J., Guo, C., Weinberger, K., Garnett, R., and Grosse, R. Discovering and exploiting additive structure for Bayesian optimization. In *Artificial Intelligence and Statistics*, pp. 1311–1319, 2017.

Genton, M. G. Classes of kernels for machine learning: a statistics perspective. *Journal of machine learning research*, 2(Dec):299–312, 2001.

Hernández-Lobato, J. M. and Adams, R. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *International Conference on Machine Learning*, pp. 1861–1869, 2015.

Hinton, G. E. and Salakhutdinov, R. R. Using deep belief nets to learn covariance kernels for Gaussian processes. In *Advances in neural information processing systems*, pp. 1249–1256, 2008.

Kakihara, Y. A note on harmonizable and v-bounded processes. *Journal of Multivariate Analysis*, 16(1):140–156, 1985.

Kandasamy, K., Schneider, J., and Póczos, B. High dimensional bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning*, pp. 295–304, 2015.

Kom Samo, Y.-L. and Roberts, S. Generalized spectral kernels. *arXiv preprint arXiv:1506.02236*, 2015.

Lázaro-Gredilla, M., Quiñonero Candela, J., Rasmussen, C. E., and Figueiras-Vidal, A. R. Sparse spectrum Gaussian process regression. *Journal of Machine Learning Research*, 11(Jun):1865–1881, 2010.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Lloyd, J. R., Duvenaud, D. K., Grosse, R. B., Tenenbaum, J. B., and Ghahramani, Z. Automatic construction and natural-language description of nonparametric regression models. 2014.

Malkomes, G., Schaff, C., and Garnett, R. Bayesian optimization for automated model selection. In *Advances in Neural Information Processing Systems*, pp. 2900–2908, 2016.

Oliva, J. B., Dubey, A., Wilson, A. G., Póczos, B., Schneider, J., and Xing, E. P. Bayesian nonparametric kernel-learning. In *Artificial Intelligence and Statistics*, pp. 1078–1086, 2016.

Poon, H. and Domingos, P. Sum-product networks: A new deep architecture. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pp. 689–690. IEEE, 2011.

Rasmussen, C. E. *Evaluation of Gaussian processes and other methods for non-linear regression*. Citeseer, 1999.

Remes, S., Heinonen, M., and Kaski, S. Non-stationary spectral kernels. *arXiv preprint arXiv:1705.08736*, 2017.

Saatçi, Y. *Scalable inference for structured Gaussian process models*. PhD thesis, Citeseer, 2012.

Schwarz, G. et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pp. 2951–2959, 2012.

Sutskever, I. and Hinton, G. E. Deep, narrow sigmoid belief networks are universal approximators. *Neural computation*, 20(11):2629–2636, 2008.

Titsias, M. Variational learning of inducing variables in sparse Gaussian processes. In *Artificial Intelligence and Statistics*, pp. 567–574, 2009.

Tobar, F., Bui, T. D., and Turner, R. E. Learning stationary time series using gaussian processes with nonparametric kernels. In *Advances in Neural Information Processing Systems*, pp. 3501–3509, 2015.

Wang, Z., Li, C., Jegelka, S., and Kohli, P. Batched high-dimensional bayesian optimization via structural kernel learning. *arXiv preprint arXiv:1703.01973*, 2017.

Wiener, N. Tauberian theorems. *Annals of mathematics*, pp. 1–100, 1932.

Wilson, A. and Adams, R. Gaussian process kernels for pattern discovery and extrapolation. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1067–1075, 2013.

Wilson, A. G., Gilboa, E., Nehorai, A., and Cunningham, J. P. Fast kernel learning for multidimensional pattern extrapolation. In *Advances in Neural Information Processing Systems*, pp. 3626–3634, 2014.

Wilson, A. G., Hu, Z., Salakhutdinov, R., and Xing, E. P. Deep kernel learning. In *Artificial Intelligence and Statistics*, pp. 370–378, 2016.

Yaglom, A. M. *Correlation theory of stationary and related random functions: Supplementary notes and references*. Springer Science & Business Media, 2012.