Gaussian Process Neural Additive Models

Wei Zhang[1], **Brian Barr**[2], **John Paisley**[1]

[1]Columbia University, New York, NY, USA
[2]Capital One, New York, NY, USA
{wz2363, jwp2128}@columbia.edu, brian.barr@capitalone.com

Abstract

Deep neural networks have revolutionized many fields, but their black-box nature also occasionally prevents their wider adoption in fields such as healthcare and finance, where interpretable and explainable models are required. The recent development of Neural Additive Models (NAMs) is a significant step in the direction of interpretable deep learning for tabular datasets. In this paper, we propose a new subclass of NAMs that use a single-layer neural network construction of the Gaussian process via random Fourier features, which we call Gaussian Process Neural Additive Models (GP-NAM). GP-NAMs have the advantage of a convex objective function and number of trainable parameters that grows linearly with feature dimensionality. It suffers no loss in performance compared to deeper NAM approaches because GPs are well-suited for learning complex non-parametric univariate functions. We demonstrate the performance of GP-NAM on several tabular datasets, showing that it achieves comparable or better performance in both classification and regression tasks with a large reduction in the number of parameters.

Introduction

With the rapid evolution of deep neural networks, one major challenge has been interpreting and explaining what they learn. DNNs are still generally considered a black-box model because it is difficult to understand and explain why a specific decision is made. This hinders their uptake in some fields such as healthcare and finance, where explainability is highly desired or even mandated by law. While post-hoc explanations can be given using feature importance or counterfactual methods, they do not provide the inherent level of interpretability contained in the weights of a simple linear model, leading some to call for their total rejection in high-stakes problems (Rudin 2019).

In this paper, we focus on the family of deep models called neural additive models (NAMs) that attempt to unite the flexibility of deep neural networks with the inherent explainability of linear models. Methodologically, the NAM approximation is formulated as

$$\min_{\theta} \ \mathcal{L}_\theta(y, g(x)), \quad g(x) = f_0 + \sum_{i=1}^{d} f_{\theta_i}(x_i) \qquad (1)$$

where $x \in \mathbb{R}^d$ is an input vector with d features, y is the target variable and the penalty between $g(x)$ and y can be, e.g., least squares for regression or the logistic regression penalty for classification. See Figure 1 for an illustration.

The key innovation of NAM methods is that they learn a feature-specific NN shape function. Whereas linear models simply define $f_{\theta_i}(x_i) = \theta_i x_i$, with θ_i determining the impact of x_i on y, the impact of each feature according to a *nonlinear* function $f_{\theta_i}(x_i)$ can be easily understood by inspection of a one-dimensional plot. Prior to the NAM framework, there have been many candidates for shape functions. For example, Hastie and Tibshirani (1990) use the spline function. Lou, Caruana, and Gehrke (2012) use boosted decision trees in a method called explainable boosting machines (EBMs). An alternative type of tree called Neural Oblivious Decision Trees (Popov, Morozov, and Babenko 2019) was proposed as shape function in NODE-GAMs (Chang, Caruana, and Goldenberg 2021). Even polynomial regression fits into this framework.

Recently, Agarwal et al. (2021) proposed using a neural network as the shape function in (1), known as a neural additive model (NAMs). This approach has shown much promise, but comes at the price of potential computational issues. While Radenovic, Dubey, and Mahajan (2022) do reduce computational expense by sharing neural network layers across features, several practical concerns with neural network training remain. Meanwhile, over the past several decades, researchers have explored the relationship between kernel methods and neural networks (Neal 2012). Cho and Saul (2009) introduced a new family of positive-definite kernel functions called multilayer kernels, while Lee et al. (2017) studied infinitely wide neural networks and the Gaussian process, showing the exact equivalence between the two. One of the key findings in Lee et al. (2017) is that Gaussian process predictions typically outperform those of finite-width neural networks.

In this paper, we use this GP/DNN insight to propose a novel family of NAMs that leverage the Gaussian process with RBF kernel as the shape function. To this end, we use the random Fourier feature approximation of the Gaussian process (Rahimi and Recht 2008) and call our framework a Gaussian Process Neural Additive Model (GP-NAM). Unlike other models, the number of parameters in GP-NAM only grows linearly as the input dimension increases, which

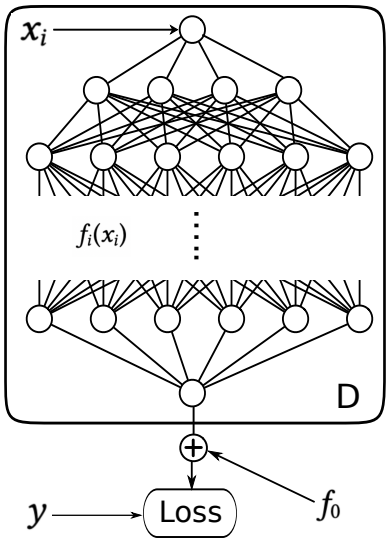

Figure 1: A graphical representation of the neural additive model. x_i is ith feature of an input vector having D dimensions. f_0 is the bias term. y is the response or label. The function $f_i(x_i)$ is the shape function for feature i. The sum $f_0 + \sum_i f_i(x_i)$ is used to predict y.

allows us to train our model quickly, and has a convex objective function which removes dependence on initialization. Furthermore, GP-NAM possesses the same interpretability of related methods since each feature contributes to the output through its own one-dimensional Gaussian process.

In the next section we review related works. We then review the Gaussian process and random Fourier feature approximation. We then propose our model based on the Gaussian process neural network framework and present an algorithm for learning its parameters. We experiment with several public tabular data sets for regression and classification.

Related Works

Our work is connected to two research directions: feature importance methods and generalized additive models.

Feature importance. Feature importance methods assess the contribution made by each input feature to the output. In linear models this is simply observed by the magnitudes of each learned feature weight (assuming relevant standardization). However, neural networks do not directly provide this information, leading to various post-hoc techniques to analyze the decision boundary. LIME (Ribeiro, Singh, and Guestrin 2016) uses locally linear model approximations around data points to do this, which has consistency issues in the explanations given. SHAP (Lundberg and Lee 2017) is another linear surrogate model-based approach. Integrated gradients (IG) (Sundararajan, Taly, and Yan 2017) and DeepLIFT (Shrikumar, Greenside, and Kundaje 2017) give their explanations by comparing reference points and input points. In contrast to local, data point specific explanations, global attribution methods provide global explanations by clustering vectors pointing from the input data points to the decision boundary (Ibrahim et al. 2019).

Generalized additive models (GAMs). GAMs learn the shape function for each individual feature and approximate a target value through a link function (Wood 2017; Hastie and Tibshirani 1990). There are many candidate shape functions, such as splines (Hastie and Tibshirani 1990), random forests (Lou, Caruana, and Gehrke 2012) and polynomials (Dubey, Radenovic, and Mahajan 2022). Recently, deep neural networks have been employed: Agarwal et al. (2021) construct a 3-layer NN for each input feature independently and introduce a new activation function called ExU for modeling jagged functions. Chang, Caruana, and Goldenberg (2021) use neural oblivious decision trees (Popov, Morozov, and Babenko 2019) as the shape function. Radenovic, Dubey, and Mahajan (2022) introduce a shared 3-layer NN as a basis function that maps each feature onto a vector space. Bouchiat et al. (2023) utilize Bayesian NNs as the shape function and the Laplace approximation to learn its posterior. All these methods are inherently explainable while suffering little loss in prediction performance for many popular tabular data sets. However, they also suffer from scalability issues and usually require regularization techniques such as batch normalization (Ioffe and Szegedy 2015) and dropout (Srivastava et al. 2014) to prevent them from overfitting. A primary contribution of our proposed GP-NAM framework is the avoidance of these issues.

Background: GPs and RFF Linearization

Before discussing its extension to the NAM regression and classification frameworks, we briefly review Gaussian process (GP) regression and the random Fourier feature (RFF) approximation, highlighting its mathematical equivalence to a single-layer neural network. Given data $(x_1, y_1), \ldots, (x_n, y_n)$, where $y \in \mathbb{R}$ and $x \in \mathbb{R}^d$, a GP models this as a function $y(x) : x \to y$ as follows:

Definition 1 (Gaussian process)**.** *Given a pairwise kernel function $k(x, x')$ between any two points x and x' in $\mathbb{R}^d$, a Gaussian process is defined to be the random function $y(x) \sim GP(0, k(x, x'))$ such that for any n data points, $(y_1, \ldots, y_n)$ is Gaussian distributed with $n \times n$ covariance matrix K_n where $K_n(i, j) = k(x_i, x_j)$.*

In this section, x_i indicates the ith observation in $\mathbb{R}^d$ and not the ith feature. We have defined the mean function of the GP to be 0 and are interested in Gaussian kernels, or radial basis functions (RBF), of the form

$$k(x, x') \equiv \exp\left\{-\frac{1}{2b^2}\|x - x'\|^2\right\}, \tag{2}$$

with parameter $b > 0$. We observe that this kernel function models positive correlations based on proximity in the space of x as defined by the scale parameter b; as two points become farther apart, their correlation reduces to zero. In the NAM framework of this paper, each dimension of x will be modeled using a separate GP.

The Gaussian process arises by integrating a linear Gaussian model. That is, let $\phi(x)$ be a mapping of x into another

space. If we define the linear regression model

$$\begin{aligned} y \,|\, x, w &\sim \mathcal{N}(\phi(x)^\top w, \sigma^2), \\ w &\sim \mathcal{N}(0, I), \end{aligned} \tag{3}$$

then the marginal distribution over n observations is

$$y \mid x \sim \mathcal{N}(0, \sigma^2 I_n + K_n),$$

where the $n \times n$ kernel matrix $K_n(i,j) = \phi(x_i)^\top \phi(x_j)$. Here, y is represented as a noise-added process, but setting $\sigma^2 = 0$ generates the underlying noise-free GP.

The power of Gaussian process theory arises when the space $\phi(x)$ is continuous or unknown. This is particularly useful for the RBF kernel, since $\phi(x)$ has a Gaussian form and $\phi(x)^\top \phi(x')$ becomes an integral over $\mathbb{R}^d$, and therefore w needs to be defined over a continuous space in $\mathbb{R}^d$.

The linear representation in Equation (3) is preferable for scalability to large data sets. For the RBF kernel, since $\phi(x)$ is continuous working in this linear space is impossible. However, approximations can be introduced that seek to construct a finite-dimensional vector $\widehat{\phi}(x)$ such that $\widehat{\phi}(x)^\top \widehat{\phi}(x') \approx \phi(x)^\top \phi(x')$. In this paper we will use the random Fourier feature (RFF) approach, which has the nice property of being mathematically equivalent to a single layer of a fully connected neural network. (We will continue to refer to this approximation as ϕ.) While originally presented for all shift-invariant kernels by Rahimi and Recht (2008), we focus on the RFF approximation to the RBF kernel. In this case, the RFF method approximates Equation (2) using a Monte Carlo integral as follows.

Definition 2 (RFF Approximation)**.** *Let $x \in \mathbb{R}^d$ and define a sample size S. Generate vectors $z_s \sim \mathcal{N}(0, I)$ in $\mathbb{R}^d$ and scalars $c_s \sim Unif(0, 2\pi)$ independently for $s = 1, \ldots, S$. For each x define the vector*

$$\phi(x) = \sqrt{\frac{2}{S}} \left[\cos\left(z_1^\top x/b + c_1\right), \ldots, \cos\left(z_S^\top x/b + c_S\right) \right]^\top$$

Then $\phi(x)^\top \phi(x') \approx \exp\{-\frac{1}{2b^2}\|x - x'\|^2\}$ with equality as sample dimensionality $S \to \infty$.

Using this representation, we can now return to the underlying linear model of the Gaussian process described in Eqs. (3) by learning $w \in \mathbb{R}^S$, which requires that the same sample set $\{(z_s, c_s)\}$ be shared by all data. Since the approximation to the Gaussian kernel holds, the underlying marginal that this linear model approximates is the desired Gaussian process. The values of $\{(z_s, c_s)\}$ are stored for later predictions. Inspection of Definition 2 shows that the function $\phi(x)$ is mathematically equivalent to a single-layer of a neural network in which the weights z and bias c are not learnable and the nonlinearity used is the cosine function.

Gaussian Process Neural Additive Models

We next show how a simple application of the RFF approximation to the GP results in a Gaussian Process Neural Additive Model (GP-NAM) for which few parameters need to be learned. While being mathematically equivalent to a NAM approach, GP-NAM results algorithmically in a direct application of a simple linear model, such as logistic regression or least squares linear regression, applied to a pre-determined feature mapping ϕ. We consider this to be a feature of GP-NAM, since it retains the flexibility and interpretability of other NAM approaches while being fast to learn and avoiding optimization issues with locally optimal solutions because of the convexity of its objective function.

Basic Setup

As mentioned in the introduction, a basic neural additive model is of the form

$$g(x) = f_0 + \sum_{i=1}^{d} f_{\theta_i}(x_i)$$

where x_i is the ith dimension of a vector $x \in \mathbb{R}^d$ and f_{θ_i} is a neural network that maps x_i to its contribution towards the label/response y using parameters θ_i, which are learned from data. For regression, g typically approximates y using the least squares penalty, while for binary classification g is passed through a sigmoid function.

In this paper, we define f to be the random function $f_{\theta_i}(x_i) \sim GP(0, k(x_i, x_i'))$ and use the RFF approximation to learn it. Thus the GP-NAM generative process becomes

$$\begin{aligned} f_{\theta_i}(x_i) &= \phi(x_i)^\top w_i, \\ w_i &\sim \mathcal{N}(0, I), \end{aligned} \tag{4}$$

where $\phi(x_i) \in [-\sqrt{2/S}, \sqrt{2/S}]^S$ is the RFF map:

$$\begin{aligned} \phi(x_i) &= \sqrt{2/S}\big[\cos(z_s x_i/b_i + c_s)\big]_{s=1}^{S}, \\ z_s &\sim \mathcal{N}(0,1), \\ c_s &\sim \text{Uniform}(0, 2\pi). \end{aligned} \tag{5}$$

We let each dimension of x have its own kernel width b_i to account for different scaling of the features.

We observe that Equation (5) is a single-layer neural network, while Equation (4) performs the linear operation of a second layer, which is summed over i to either model y directly for regression, or is passed through a sigmoid to model the binary label y. (A straightforward extension to multiclass problems can be made, but isn't considered in this paper.) This two-step process is also equivalent to a sum over d dimension-specific Gaussian processes evaluated at each dimension's input. The result is a neural additive model where the only learnable parameters are $w_1, \ldots, w_d$, with each $w_i \in \mathbb{R}^S$ and S chosen to provide a good approximation to the GP. Since each GP is one dimensional, we empirically found that $S = 100$ works well.

Discussion

As shown in the previous subsection, a GP additive model can be formulated as an equivalent single-layer neural additive model. We provide an illustration of this GP-NAM framework in Figure 2. In our GP-NAM formulation, only the linear weights of the last layer need to be learned, whereas in the vanilla NAM framework a potentially multilayer neural network is learned for each feature x_i prior

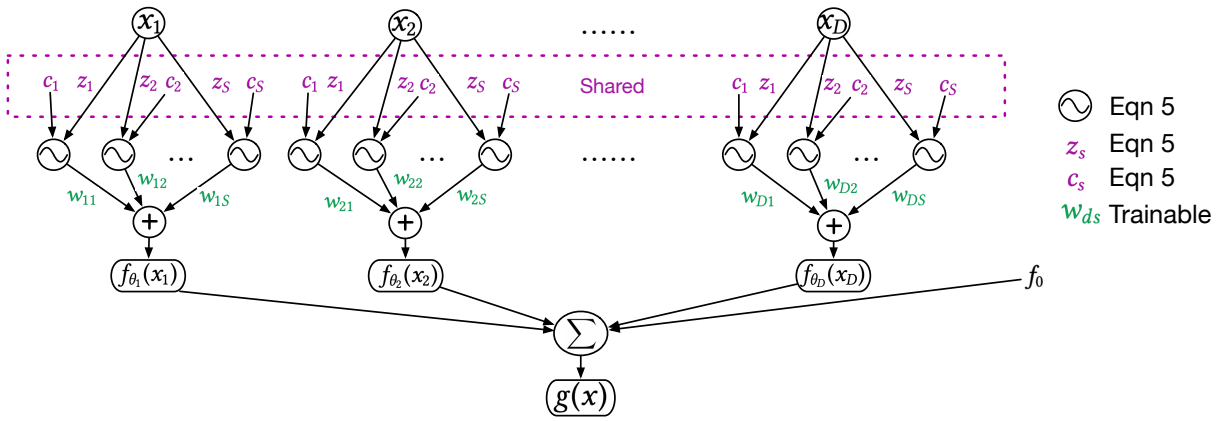

Figure 2: The architecture of GP-NAM. Each x_i represents one feature of a single input vector $x \in \mathbb{R}^d$. Each (z_s, c_s) is shared across shape functions. The GP $f_{\theta_i}(x_i)$ is the shape function for the ith feature. The only trainable parameters are the feature-specific S-dimensional weight vectors $w_1, \ldots, w_D$ that connect the output from the cosine functions to their corresponding GP shape function. The prediction is made by using the sum of the outputs from all the shape functions with the bias term f_0. This is mathematically equivalent to an additive Gaussian process.

to this last linear layer. NBM improves on this by allowing neural network parameters to be shared for each dimension. This leads to some immediate observations.

First, we do not anticipate that GP-NAM will clearly outperform NAM or NBM. This is because, while GP-NAM predefines one layer of parameters z and c, other NAM approaches allow these to be learned along with deeper layers, thereby potentially improving the fit. Therefore, since we share z and c across dimensions of x, GP-NAM can be considered a special case of NBM that is single layer with unlearnable weights and cosine nonlinearity.

However, in the specific case of additive modeling this is not necessarily a downside, and may be an advantage. While deep neural networks can be expected to outperform Gaussian processes on complex, high-dimensional problems, in the one-dimensional setting of additive modeling it is not clear that a neural network on $\mathbb{R}$ is preferable to a Gaussian process. For one-dimensional function approximation problems a GP with RBF kernel is remarkably flexible in the functions it can learn. Though NBM models can be said to contain GP-NAM as a special case – just as both can be said to contain the solution set of the classical linear model $x^\top w$ as a special case – restricting the NBM structure to the GP framework drastically reduces the number of learnable parameters and is convex when y is modeled using least squares or logistic regression. Therefore, learning GP-NAM should be significantly faster and will not suffer from potential local optimal issues arising from the non-convexity of other NAM models.

Finally, we note that extensions to incorporate cross-terms have led to NA^{2}M and NB^{2}M extensions of the form

$$g(x) = f_0 + \sum_{i=1}^{d} f_{\theta_i}(x_i) + \sum_{i'>i} f_{\theta_{ii'}}(x_i, x_{i'}).$$

By letting $\phi(x_i, x_{i'})$ be a two dimensional GP with $z \in \mathbb{R}^2$ as described in the background section, we can extend GP-NAM to GP-NA^{2}M in a similar way.

Algorithm Details

Since GP-NAM is a linear model applied to a predetermined feature mapping ϕ, standard least squares and logistic regression algorithms can be used. We can see the linearity of GP-NAM explicitly by rewriting g as

$$g(x) = w_0 + \sum_{i=1}^{d} \phi(x_i)^\top w_i \tag{6}$$

where $w_0 \in \mathbb{R}$ and the remaining $w \in \mathbb{R}^S$ are the only learnable parameters. In this section, we provide some additional practical details.

First, we note that the Monte Carlo integral of Definition 2 from which we construct ϕ arises from the equality

$$\exp\left\{-\frac{1}{2b^2}\|x-x'\|^2\right\} = \frac{1}{\pi}\int_0^{2\pi}\int_{\mathbb{R}^d} \cos\left(\frac{z^\top x}{b} + c\right)\cos\left(\frac{z^\top x'}{b} + c\right)\mathcal{N}(z|0,I)dzdc \tag{7}$$

as derived by Rahimi and Recht (2008). For multidimensional problems, being able to sample (z_s, c_s) from a joint Gaussian-uniform distribution greatly simplifies the problem. However, since the NAM framework considers each dimension separately, selecting $z_1, \ldots, z_S$ can be done using deterministic grid points calculated from the inverse CDF of the standard univariate normal distribution, providing a slightly improved approximation to the integral. For each dimension, we construct $\{(z_s, c_s)\}$ by pairing a randomly permuted uniform grid of $c \in [0, 2\pi]$ with a grid of $z \in \mathbb{R}$ using the inverse CDF of $\mathcal{N}(0, 1)$.

For binary classification, the model in (6) can be quickly optimized over w using standard stochastic algorithms for logistic regression. For regularized least squares regression (recalling the Gaussian prior on w) the solution to

Algorithm 1: GP-NAM for regression and classficiation

Require: Data $\{(x, y)\}$, GP width S, kernel widths $b_{1:d}$.
Sample $z_s \sim \mathcal{N}(0, 1)$, $c_s \sim \text{Unif}(0, 2\pi)$, $s = 1 : S$.
    Alternatively, grid using inverse CDF.
Define $\phi(x_i) = \sqrt{2/S}\big[\cos(z_s x_i/b_i + c_s)\big]_{s=1}^{S}$
    and $\phi_n = \text{stack}(1, \phi(x_{1,n}), \ldots, \phi(x_{d,n}))$
Regression: Define $A = \sum_n \phi_n \phi_n^\top$ and $v = \sum_n y_n \phi_n$.
    Solve $(I + A)w = v$ using conjugate gradients
Classification: Solve linear classifier w on $\{(\phi_n, y_n)\}$
Return w

w is closed form, but the dimensionality of the matrix inverse will likely present computational or numerical issues. By stacking $\phi = [1, \phi(x_1), \ldots, \phi(x_d)]$ and $w = [w_0, w_1, \ldots, w_d]$, the classic conjugate gradients algorithm (Nocedal and Wright 2006) is a fast and stable means for solving

$$\Big(I + \sum_{n=1}^{N} \phi_n \phi_n^\top\Big) w = \sum_{n=1}^{N} y_n \phi_n. \tag{8}$$

We therefore do not need to invert the left matrix to solve for w. We summarize these algorithms in Algorithm 1.

Experiments

We experiment using several tabular data sets. A key feature of our model is its reduction in parameters and convex optimization, while still providing competitive performance as a neural additive model. Table 1 and Figure 3 illustrate the magnitude of this reduction for multiple data scenarios. Here, the parameter size of NAM, NBM and GP-NAM are calculated as $|\text{NAM}| = (|\text{NN}| + S)D$, $|\text{NBM}| = |\text{NN}| + SD$ and $|\text{GP-NAM}| = SD$, where S is the basis size, D is the data dimensionality, and $|\text{NN}|$ is the number of parameters in the neural network. For example, Agarwal et al. (2021) and Radenovic, Dubey, and Mahajan (2022) proposed a network of size $|\text{NN}| = 6401$ and $|\text{NN}| = 62,820$, respectively. The reduction in parameters can lead to a significant improvement in training time. For example, for the LCD data set this translates to 5.5 and 3.5 sec/epoch for NAM and NBM on GPU, respectively, and 50 ms/epoch for GP-NAM on our CPU. For reference, NODE-GAM required 250 ms/epoch and EBM required 50 ms/epoch.

Datasets

We perform experiments on several tabular data sets frequently used for additive regression and classification models. This includes **CA Housing**[1], **FICO**[2], for which we follow the processing of Radenovic, Dubey, and Mahajan (2022). We also report performance on **MIMIC-II**[3], **MIMIC-II**[4], **Credit**[5], **Click**[6], **Microsoft**[7], **Year**[8] and **Yahoo**[9], **Churn**[10], **Adult**[11], **Bikeshare**[12] tabular data sets. For these, we follow the processing in Chang, Caruana, and Goldenberg (2021); Popov, Morozov, and Babenko (2019). We also consider our own processing of credit lending data sets **LCD**[13] and **GMSC**[13] More information about these data sets is shown in Table 2.

Model	Bike	CA House	FICO	LCD
NAM	54K	83K	262K	32K
NBM	65K	64K	68K	63K
GP-NAM	801	1201	3901	501

Table 1: Examples of the number of parameters. GP-NAM is a fast, parameter-lite NAM approach with equivalent performance as shown in the quantitative evaluation.

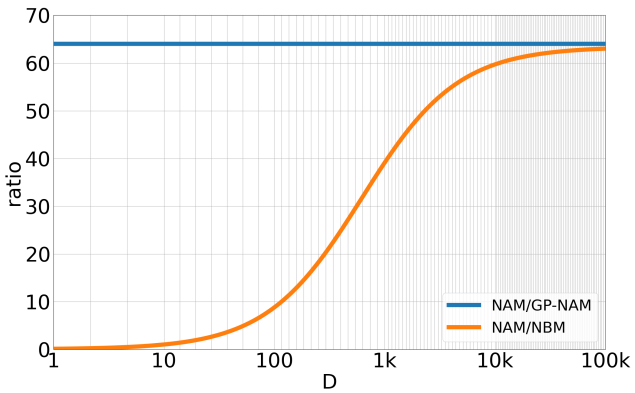

Figure 3: Parameter number ratios $|\text{NAM}|/|\text{NBM}|$ (orange) and $|\text{NAM}|/|\text{GP-NAM}|$ (blue) as a function of data dimensionality. We set $S = 100$ basis functions for all models to give a fair comparison. GP-NAM uses $\sim$60x fewer parameters than NAM regardless of the dimensionality of x, and e.g. $\sim$15x fewer than NBM at $x \in \mathbb{R}^{40}$. We are interested in the tabular data regime where e.g. $D < 500$.

Baselines

We compare with several state-of-the-art additive models, as well as two black-box methods for reference. All models are implemented with PyTorch and trained using stochastic gradient descent.

Linear. Logistic and linear regression for classification and regression are the most fundamental and interpretable

[1]https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing
[2]https://community.fico.com/s/explainable-machine-learning-challenge
[3]https://archive.physionet.org/mimic2
[4]https://physionet.org/content/mimiciii/
[5]https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
[6]https://www.kaggle.com/c/kddcup2012-track2
[7]https://www.microsoft.com/en-us/research/project/mslr
[8]https://archive.ics.uci.edu/ml/datasets/yearpredictionmsd
[9]https://webscope.sandbox.yahoo.com/catalog.php?datatype=c
[10]https://www.kaggle.com/blastchar/telco-customer-churn
[11]https://archive.ics.uci.edu/dataset/2/adult
[12]https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset
[13] https://github.com/Wei2624/GPNAM

Dataset	#Train	#Val	#Test	#Feat
Churn	4,507	1,127	1,09	20
FICO	7,321	1,046	2,092	39
LCD	10,000	1,000	1,000	5
GMSC	15,000	1,000	1,000	9
MIMIC-II	17,155	2,451	4,902	17
MIMIC-III	17,502	4,376	5,470	57
Adult	20,838	5,210	6,513	14
Credit	199,364	28,481	56,962	30
Click	800,000	100,000	100,000	11
Bikeshare	11,122	2,781	3,476	12
CA Housing	14,447	2,065	4,128	8
Year	370,972	92,743	51,630	90
Yahoo	473,134	71,083	165,660	699
Microsoft	580,539	142,873	241,521	136

Table 2: Statistics from the tabular data sets used in binary classification (top) and regression (bottom) experiments.

models. They provide individual weights for each feature.

NAM (Agarwal et al. 2021). NAM is the first model to use neural networks for generalized additive modeling. We use the authors' implementation.

NODE-GAM (Chang, Caruana, and Goldenberg 2021). NODE-GAM is based on Neural Oblivious Decision Trees (NODE) (Popov, Morozov, and Babenko 2019) where a full decision tree is learned for each feature.

NBM (Radenovic, Dubey, and Mahajan 2022). The Neural Basis Model (NBM) reduces the number of trainable parameters with a shared basis neural network that maps each feature to a predefined number of basis features. These are mapped to shape functions by linear projections.

EBM (Lou et al. 2013). Explainable Boosting Machines gradient boost thousands of shallow tress for each feature and are considered another type of GAM. We use the interpretML library (Nori et al. 2019).

MLP. Multi-layer perceptrons are a black-box model when interpretability is not needed. We use the architecture reported in Radenovic, Dubey, and Mahajan (2022).

XGBoost (Chen and Guestrin 2016). This is another black box baseline. We use the XGBoost library.

Implementation Details

In addition to learning shape functions equivalent to a single-layer neural network, the predetermined hidden layer weights and offset means that back-propagation algorithms are unnecessary. Instead, as outlined in Algorithm 1, we implement a stochastic optimization algorithm for logistic regression to learn a classifier, or solve the classic conjugate gradients problem for regression, both using the stacked feature mappings ϕ. Therefore, the algorithm is faster than those available for deeper NAM models.[14]

For NAM, NODE-GAM and NBM, we use the best parameters provided in Radenovic, Dubey, and Mahajan (2022) or perform a similar hyper-parameter search otherwise. We run Linear, EBMs and XGBoost on CPUs and use the default parameters provided in the libraries.

We follow other recent papers in calculating AUC or Error Rate to evaluate classification performance, and MSE or RMSE to evaluate regression performance on the same training/validation/testing split for each algorithm. We also provide some qualitative evaluation.

Experimental Results and Discussion

Quantitative comparisons. We show our main quantitative results in Table 4. The top half shows results for which higher values are better, and in the bottom half lower values are better. One takeaway from this table is that no NAM approach is clearly best for all problems. GP-NAM occasionally performs best (MIMIC-II, GMSC, Click), or effectively tied for best (Year, CA Housing, Bikeshare, Yahoo, Microsoft). For nearly all of the remaining 6 problems GP-NAM is close to the best along with several other algorithms. Furthermore, for several problems (Credit, Adult, Churn, FICO, LCD, Microsoft) a basic linear model in the features space performs very competitively, indicating a highly linear problem there. For the data sets with obvious improvement (Bikeshare, CA Housing) or moderate improvement (MIMIC-II, MIMIC-III, GMSC, Yahoo), GP-NAM captures the nonlinearity of the problem as well as other NAM approaches. However, as previously highlighted in Table 1, GP-NAM is a parameter-light model with a fast convex optimization algorithm. In Table 3 we also show how GP-NAM can perform well on tabular data in comparison with non-additive models that capture greater complexity in the data, but are harder to interpret.

Interpretability and stability. The inherent interpretability of a GAM model can be obtained by visualizing the shape function for each feature. In Figure 4 we show the shape functions learned by GP-NAM on the LCD data along with NAM and NODE-GAM, where NAM represents a neural additive model and NODE-GAM a tree-based GAM model.

As is evident, the neural network of NAM does not learn functions with the same smoothness as the GP – indeed, it is not clear how interpretable NAM actually is in this case. NODE-GAM is smoother and analysis can determine which, if either, is more meaningful between it and GP-NAM. We note that for GP-NAM, DTI (debt-to-income) follows a meaningful nonlinear pattern, where defaults are con-

Model	CA House RMSE↓	Bike RMSE↓	FICO AUC↑	LCD AUC↑
Linear	0.7354	145.9	0.7909	0.9459
GP-NAM	0.5586	99.6	0.8043	0.9524
XGBoost	0.4428	50.0	0.7925	0.9567
MLP	0.5041	44.2	0.7936	0.9589

Table 3: Performance comparison on a subset of data sets using complex and non-interpretable XGBoost and MLP. Gives an indication of GP-NAM performance in relation to "upper" and "lower" bounds.

[14]The GP-NAM code for regression and classification can be found at https://github.com/Wei2624/GPNAM

Model	MIMIC-II AUC↑	MIMIC-III AUC↑	Credit AUC↑	GMSC AUC↑	Adult AUC↑	Churn AUC↑	FICO AUC↑	LCD AUC↑
Linear	0.8147	0.7753	0.9770	0.8063	0.9013	0.8345	0.7909	0.9459
EBM	0.8514	0.8090	0.9760	0.8655	0.9277	0.8490	0.7985	0.9519
NAM	0.8539	0.8015	0.9766	0.8548	0.9152	0.8356	0.7993	0.9494
NODE-GAM	0.8320	0.8140	0.9810	0.8215	0.9166	0.8339	0.8063	0.9558
NBM	0.8549	0.8120	0.9829	0.8328	0.9176	0.8389	0.8048	0.9506
GP-NAM	0.8508	0.8159	0.9794	0.8674	0.9167	0.8360	0.8043	0.9524

Model	Bikeshare RMSE↓	Click ERR↓	Microsoft MSE↓	Yahoo MSE↓	Year MSE↓	CA Housing RMSE↓	Model Size Params/feature
Linear	145.9	0.3443	0.8693	0.6765	88.51	0.7354	1
EBM	100.0	0.3338	0.8654	0.6312	85.15	0.5586	10K Stumps
NAM	99.6	0.3317	0.8588	0.6458	85.25	0.5721	S+|NN|
NODE-GAM	100.7	0.3342	0.8533	0.6305	85.09	0.5658	X NODE Tree
NBM	99.4	0.3312	0.8602	0.6384	85.10	0.5638	S+|NN|/D
GP-NAM	99.6	0.3030	0.8588	0.6302	85.10	0.5586	S

Table 4: Quantitative results for several regression and classification tasks. The top half of the plot indicate problems where ↑ is better, and the bottom half where ↓ is better. For several data sets, complex additive models improve significantly over the baseline linear model. Among those, GP-NAM performs at the level of more complex alternatives, and occasionally better.

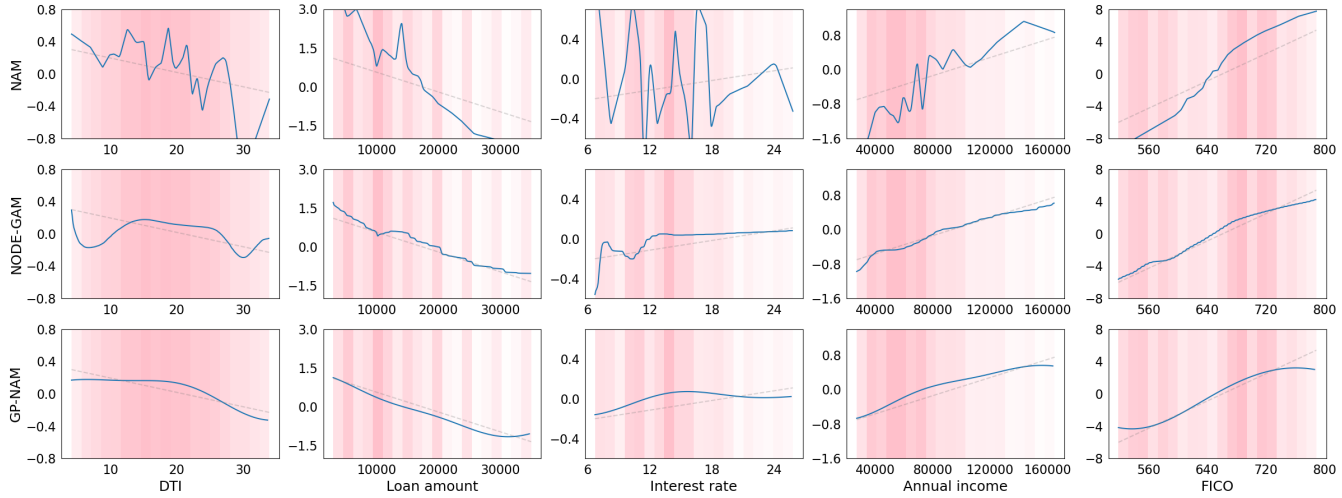

Figure 4: Shape functions of NAM, NODE-GAM and GP-NAM on the LCD data set in the original scales. The density of each feature in the training data is plotted in pink. For reference, logistic regression learned weights indicated by the slope of the light dashed line on each plot. Inspection shows that GP-NAM is in fairly close agreement with linear classification, while still allowing for meaningful nonlinearities to be learned from the data (DTI in particular).

sistently lower probability until a 20% threshold is reached, at which point defaults increase in probability with increasing prior debt. In terms of stability under multiple reruns, since GP-NAM is a convex optimization problem there is no randomness on the learned shape functions. The other algorithms required a run to be chosen.

Conclusion

We have presented a Gaussian Process Neural Additive Model (GP-NAM) for interpretable nonlinear modeling of tabular data. We are motivated by the fact that Gaussian processes are a robust and flexible nonparametric method for univariate function approximation, and thus are as suitable for the generalized additive modeling problem as deeper neural networks. Using the RFF approximation, we demonstrated how GP-NAM is a neural additive model with a single, pre-determined hidden layer and few learnable parameters. The result is an efficient convex optimization problem for regression or classification that performs as well as more complicated, non-convex deep approaches to the problem. Indeed, for low dimensional function approximations, the equivalence of a GP using RFFs with a single layer NN may support a preference for this simpler model over deeper models for certain applications, such as spatio-temporal model averaging (Paisley et al. 2022).

References

Agarwal, R.; Melnick, L.; Frosst, N.; Zhang, X.; Lengerich, B.; Caruana, R.; and Hinton, G. E. 2021. Neural Additive Models: Interpretable machine learning with neural nets. *Advances in Neural Information Processing Systems*.

Bouchiat, K.; Immer, A.; Yèche, H.; Rätsch, G.; and Fortuin, V. 2023. Laplace-Approximated Neural Additive Models: Improving Interpretability with Bayesian Inference. *arXiv preprint arXiv:2305.16905*.

Chang, C.-H.; Caruana, R.; and Goldenberg, A. 2021. Nodegam: Neural generalized additive model for interpretable deep learning. *arXiv preprint arXiv:2106.01613*.

Chen, T.; and Guestrin, C. 2016. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SigKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.

Cho, Y.; and Saul, L. 2009. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems*.

Dubey, A.; Radenovic, F.; and Mahajan, D. 2022. Scalable interpretability via polynomials. *Advances in Neural Information Processing Systems*.

Hastie, T. J.; and Tibshirani, R. J. 1990. *Generalized Additive Models*, volume 43. CRC Press.

Ibrahim, M.; Louie, M.; Modarres, C.; and Paisley, J. 2019. Global explanations of neural networks: Mapping the landscape of predictions. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*.

Ioffe, S.; and Szegedy, C. 2015. Batch Normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*.

Lee, J.; Bahri, Y.; Novak, R.; Schoenholz, S. S.; Pennington, J.; and Sohl-Dickstein, J. 2017. Deep neural networks as Gaussian processes. *arXiv preprint arXiv:1711.00165*.

Lou, Y.; Caruana, R.; and Gehrke, J. 2012. Intelligible models for classification and regression. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Lou, Y.; Caruana, R.; Gehrke, J.; and Hooker, G. 2013. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 623–631.

Lundberg, S. M.; and Lee, S.-I. 2017. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*.

Neal, R. M. 2012. *Bayesian learning for neural networks*. Springer Science & Business Media.

Nocedal, J.; and Wright, S. J. 2006. *Numerical Optimization, Second Edition*. Springer Series in Operations Research.

Nori, H.; Jenkins, S.; Koch, P.; and Caruana, R. 2019. Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*.

Paisley, J.; Rowland, S.; Liu, J. Z.; Coull, B.; and Kioumourtzoglou, M.-A. 2022. Bayesian nonparametric model averaging using scalable Gaussian process representations. In *IEEE International Conference on Big Data*.

Popov, S.; Morozov, S.; and Babenko, A. 2019. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*.

Radenovic, F.; Dubey, A.; and Mahajan, D. 2022. Neural basis models for interpretability. *Advances in Neural Information Processing Systems*.

Rahimi, A.; and Recht, B. 2008. Random Features for Large-Scale Kernel Machines. In *Advances in Neural Information Processing Systems*.

Ribeiro, M. T.; Singh, S.; and Guestrin, C. 2016. "Why should I trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Rudin, C. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5): 206–215.

Shrikumar, A.; Greenside, P.; and Kundaje, A. 2017. Learning important features through propagating activation differences. In *International Conference on Machine Learning*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958.

Sundararajan, M.; Taly, A.; and Yan, Q. 2017. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*.

Wood, S. N. 2017. *Generalized Additive Models: An Introduction with R*. CRC Press.