To appear in Behaviourmetrika special issue on "Interpreting Neural Network Models"

Accuracy versus interpretability in flexible modeling: implementing a tradeoff using Gaussian process models

Tony A. Plate
School of Mathematical and Computing Sciences
Victoria University of Wellington, Wellington, New Zealand
Tony.Plate@vuw.ac.nz

Abstract

One of the widely acknowledged drawbacks of flexible statistical models is that the fitted models are often extremely difficult to interpret. However, if flexible models are constrained to be additive the fitted models are much easier to interpret, as each input can be considered independently. The problem with additive models is that they cannot provide an accurate model if the phenomenon being modeled is not additive. This paper shows that a tradeoff between accuracy and additivity can be implemented easily in Gaussian process models, which are a type of flexible model closely related to feedforward neural networks. One can fit a series of Gaussian process models that begins with the completely flexible and are constrained to be progressively more additive, and thus progressively more interpretable. Observations of how the degree of non-additivity and the test error change as the models become more additive give insight into the importance of interactions in a particular model. Fitted models in the series can also be interpreted graphically with a technique for visualizing the effects of inputs in non-additive models that was adapted from plots for generalized additive models. This visualization technique shows the overall effects of different inputs and also shows which inputs are involved in interactions and how strong those interactions are.

1 Introduction

Although getting accurate predictions on new data from a statistical model may be sufficient, it is often desirable and sometimes essential to be able to interpret the fitted model. This is especially true in medical domains, where black-box methods are viewed with great suspicion (Wyatt 1995; Sharp 1995). Flexible statistical models, such as neural networks, bagged decision trees, or Gaussian processes tend to be rather difficult to interpret. When interpretation is important, simpler but more easily interpretable methods such as logistic regression are often used instead. There is a risk with such simpler methods that they may fail to discover an important relationship in the data because they lack the flexibility to model it.

The standard practice in the neural network modeling community, and, to a lesser extent, in the flexible statistical modeling community has been to minimize model complexity only in the service of minimizing prediction error. However, given the importance of interpretability in some domains, it could be useful to have a modeling technique in which we can trade slight decreases in predictive power for increases in interpretability. The primary cause of difficulty of interpretation of flexible models is interactions rather than non-linear effects in individual variables, so the interpretability of a fitted model can be regarded as inversely proportional to the importance of interactions. This paper combines ideas from generalized additive modeling (Hastie and Tibshirani 1990) and from Gaussian process modeling (Williams and Rasmussen 1996) in an attempt to implement and graphically present the results of various different positions in the tradeoff between minimizing prediction error and maximizing interpretability.

Generalized additive models (GAMs) are among the most interpretable general-purpose flexible (i.e., non-linear) statistical models. GAMs are not fully flexible — in a GAM, variables can have non-linear effects but do not interact. The lack of interaction is, in fact, what makes them so easily interpretable: the independent effect of each variable can be shown in a simple plot. The entire set of plots (one for each variable) for an fitted additive model provides a very informative visualization. Plate et al (1997) showed how these plots could be adapted to visualize non-additive models. The resulting "additive effects" plots are quite informative, but cannot present full information about interactions — unrestricted interactions are inherently complex and there is unlikely to ever be any single universally appropriate method for presenting them.

Neal (1996) showed a close relationship between feedforward neural networks and Gaussian process models: as the number of hidden units in a single-hidden-layer feedforward neural network goes to infinity, its properties (in a Bayesian formulation) converge to those of a Gaussian process. Williams and Rasmussen (1996, Neal (1996) showed how Gaussian process models could be applied directly to high-dimensional supervised learning tasks, and showed that the results appeared very competitive with neural network approaches. Rasmussen (1996), in some careful studies, showed that Gaussian process models often out-perform other models such as neural networks and decision trees. The idea presented in this paper is that of installing a "knob" on Gaussian processes that can be used to lower, in steps, the influence of non-additive components of the model. Since only the non-additive components are difficult to interpret, this allows one to tradeoff interpretability against model accuracy.

The modeling technique advocated in this paper is to fit a series of Gaussian process models progressively constrained to be more and more additive. This additivity knob is intended to serve three purposes: (1) to allow one to see whether interactions are merely gratuitous or are actually necessary to achieve good predictions; (2) to verify observations about the individual effects of variables in a fitted unconstrained model; and (3) to investigate the form of the fitted model and how it changes as the model is forced to be more interpretable. Additive effects plots and diagnostics offer insight into these three issues. The plot of generalization error versus degree of additivity shows the importance of interactions in the fitted model. The plots of the additive effects of individual variables in the series of fitted models show the overall effects of each variable and which variables are involved in interactions.

2 Additive models and additive plots

A large part of the attraction of generalized additive models (GAMs) (Hastie and Tibshirani 1990) is that, although they are non-linear, fitted models are easily interpreted from simple graphical displays.

A GAM can be expressed as

$$g(\bar{\mathbf{x}}) = h(\theta_1(x_1) + \theta_2(x_2) + \cdots + \theta_k(x_k))$$

where $\bar{\mathbf{x}} = (x_1, x_2, \ldots, x_k)$ is the vector of input values and $g(\bar{\mathbf{x}})$ is the predicted output value. The functions θ_1, θ_2 etc can be arbitrary non-linear functions. The function h relates the sum of the θ_i's to the output variable and depends on the distribution of the target variable and the distribution of errors in the target values. For example, for continuous target variables with Gaussian errors of constant variance, the identity function is appropriate for h. For binary valued target variables, the logistic function is appropriate for h: $h(a) = 1/(1 + e^{-a})$, which gives the probability of the output being 1. The possibility of different functions for h is what makes GAMs generalized: appropriate functions exist for a wide range of target error distributions. The way the functions of the inputs combine is what makes GAMs additive. Although the effect of a particular input can be non-linear there are no interactions between inputs: the contribution of a particular input does not depend on the values of other inputs. (However, it is possible to use functions of more than one variable in a component function in a GAM, which allows for pre-specified interactions between particular inputs.)

A GAM can be easily interpreted via plots of the effects of inputs on the additive scale of the model, i.e., plots of θ_i. The θ_i are usually functions of one variable, and hence can be displayed simply.

3 Gaussian process models

Gaussian process models are a well-established modeling technique that have recently received attention as an alternative to feedforward neural networks for high-dimensional nonlinear regression tasks. The recent interest was inspired by Neal (1996) who showed a close relationship between feedforward neural networks and Gaussian process models: as the number of hidden units in a single-hidden-layer feedforward neural network goes to infinity, its properties (in a Bayesian formulation) converge to those of a Gaussian process. Williams and Rasmussen (1996) and Rasmussen (1996) have shown that Gaussian process models can be applied directly to high-dimensional supervised learning tasks, and often out-perform other models such as neural networks and decision trees. Gaussian processes are closely related to a number of well-established techniques, including 'kriging' in geostatistics (Matheron 1963), generalized radial basis functions (Poggio and Girosi 1989), and spline models (Wahba 1990; Wahba, Wang, Gu, Klein, and Klein 1994a).

In the context of interpretation of complex models, the attractiveness of Gaussian process models is that it is simple to fit a Gaussian process model that has both additive and general components (the general components allow interactions). It is also easy to adjust the relative importance of additive and general components.

Only an brief explanation of Gaussian process models will be given here; for a detailed description of how Gaussian process models can be applied to regression and classification tasks, the reader should consult Williams and Rasmussen (1996), Rasmussen (1996), Neal (1997), Mackay (1997), or Gibbs and Mackay (1997).

3.1 Overview of Gaussian process models

The Gaussian process models used in this paper can be seen as a type of nearest neighbor model that adaptively choose which input dimensions (or directions) are important when determining distance.

Consider using a Gaussian process model for a regression task, where for each example we want to predict a single scalar output value based on a number of input values. To begin with, ignore the inputs. Suppose we have N training examples, and we want to make a prediction on one new example. The outputs are assumed to be from a zero-mean $N+1$ dimensional Gaussian distribution, with an $N+1$ by $N+1$ covariance matrix. The vector of outputs can be thought of as a sample from this Gaussian distribution. The predicted distribution for the output for the new example is calculated by conditioning on the other (training) outputs. The mean (or median, or mode) of this distribution can be used as a point value prediction for this output. For example, if there is one training case, and the covariance between the training output and the new output is high, then the new output will have a narrow distribution with a mean close to the training case value. If the covariance is low, the mean will be close to zero and the distribution will be wide.

Making this prediction uses a covariance matrix that has $(N+1)(N+2)/2$ possible degrees of freedom. This may seem like an enormous number of parameters when we only have N training cases. However, this is where the inputs enter the picture. The elements of the covariance matrix are not free to assume arbitrary values — they are derived in a highly parameterized manner from the inputs. Element k, j of the covariance matrix is a function of the distance between the input vectors for the kth and jth examples (here, training and testing examples are treated equally). The distance metric can be parameterized to allow the importance of different dimensions of the input space to vary. For pairs of examples that are close in input space according to the distance metric being used, the corresponding elements of the covariance matrix will be large, which means that we expect their outputs to be close.

For the benefit of the reader not familiar with Gaussian process models, the remainder of this section gives an introduction to the underlying math using terminology and concepts of neural network models. Readers not interested in the mathematical details may like to skip from here directly to Section 4, merely taking the message that we do have a knob with which to control the additivity of a Gaussian process model.

3.2 Neural network modeling

To introduce the terminology, I will describe a multi-layer perceptron (MLP) (aka feedforward neural network). Consider an MLP that maps from an d-dimensional input space to a scalar output. Let $\{(\bar{\mathbf{x}}^{(i)}, t^{(i)})\}_{i=1}^N$ be the *training set* consisting of N input-output pairs $(\bar{\mathbf{x}}^{(i)}, t^{(i)})$. (In general, $\{\cdot\}_{i=a}^b$ denotes an ordered collection of items, indexed from a to b). Let $\mathbf{W}$ be the parameters of the network, and $mlp(\cdot, \mathbf{W})$ be the function computed by the network. Thus, $mlp(\bar{\mathbf{x}}^{(i)}, \mathbf{W})$ is the value computed by the network when presented with the ith d-dimensional input vector $\bar{\mathbf{x}}^{(i)}$. Such a function can be used to model noisy data by proposing that a target output is equal to the output of the network plus some random, zero-mean, noise. In neural network modeling, this noise is commonly assumed to be Gaussian. Let σ^2 be the variance of this noise. Thus, for a network with weight matrix $\mathbf{W}$, an input vector $\bar{\mathbf{x}}^{(i)}$, and a noise level σ, the probability of observing a particular target value $t^{(i)}$ is as follows:

$$\Pr(t^{(i)}|\bar{\mathbf{x}}^{(i)}, \mathbf{W}, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp(\frac{-(t^{(i)} - mlp(\bar{\mathbf{x}}^{(i)}, \mathbf{W}))^2}{2\sigma^2})$$

The probability of observing an entire training set $\{(\bar{\mathbf{x}}^{(i)}, t^{(i)})\}_{i=1}^N$ for a given fitted model and noise level is known as the *likelihood* of the fitted model. It is simply the product of the probabilities of the individual observations:

$$\mathit{Likelihood} = \Pr(\{t^{(i)}\}_{i=1}^N|\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^N, \mathbf{W}, \sigma) = \prod_{i=1}^N \frac{1}{\sigma\sqrt{2\pi}} \exp(\frac{-(t^{(i)} - mlp(\bar{\mathbf{x}}^{(i)}, \mathbf{W}))^2}{2\sigma^2}) \tag{1}$$

The log of this is

$$\begin{aligned} \log \mathit{Likelihood} &= \log \Pr(\{t^{(i)}\}_{i=1}^N|\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^N, \mathbf{W}, \sigma) \\ &= -N\log(\sigma\sqrt{2\pi}) - \frac{1}{2\sigma^2}\sum_{i=1}^N (t^{(i)} - mlp(\bar{\mathbf{x}}^{(i)}, \mathbf{W}))^2, \end{aligned}$$

which, modulo a scaling factor $1/(2\sigma^2)$ and translation by $N\log(\sigma\sqrt{2\pi})$, is the familiar "total error" for a neural network. A common approach to using a neural network is to find the parameters $\mathbf{W}$ that maximize the likelihood of the fitted model (i.e., the probability of the data given the fitted model), which is equivalent to minimizing the total error. This process of finding the optimal set of parameters is known as *fitting* or *training* the network. Note that the noise level σ^2 often need not be computed, but when it is required it can be estimated as the variance of the difference between the network outputs and the target values. This is a *maximum likelihood* approach to model fitting because we attempt to find a single set of weight values that maximizes the likelihood of the fitted model.

3.3 The Bayesian framework for modeling

In a Bayesian approach, one does not consider just the "most likely" function from the family of functions under consideration (i.e., the most likely set of weights for the neural network.) Rather, one considers all possible functions from some class of functions, weighted by the likelihood and *prior* probability of each function. The prior probability is how probable we think the function is prior to having seen any training data. For example, in neural networks, the common weight-penalty prior makes networks with large weights less probable.

Before writing down the probability of a particular target value in a Bayesian framework, let us generalize the notation to talk about a wider class of functions than neural networks, and also to incorporate the noise into the model. Before, we used $mlp(\cdot, \mathbf{W})$ and σ to define a probability distribution over target values. Now, let $\mathcal{M}$ denote the combined functional and noise model. Thus, where before we wrote the probability of a target value given a particular fitted network specified by $\mathbf{W}$ as $\Pr(t^{(i)}|\bar{\mathbf{x}}^{(i)}, \mathbf{W}, \sigma)$, we now write the probability of a target value given a particular fitted model as $\Pr(t^{(i)}|\bar{\mathbf{x}}^{(i)}, \mathcal{M})$. For some input vector $\bar{\mathbf{x}}^{(i)}$, the mean of this distribution over target values is often used as the point predication for the fitted model (i.e., $\mathcal{E}[t|\bar{\mathbf{x}}^{(i)}, \mathcal{M}]$ corresponds to $mlp(\bar{\mathbf{x}}^{(i)}, \mathbf{W})$).

In a Bayesian framework, we also need to specify a prior over the class of functions under consideration. The parameters which specify the prior are known as *hyperparameters*. Let Ψ be the hyperparameters. The probability of a particular fitted model $\mathcal{M}$ given some hyperparameters Ψ is $\Pr(\mathcal{M}|\Psi)$. In neural networks, a common hyperparameter is the weight penalty factor (aka weight decay parameter), which acts to prefer networks with small weights.

Now we are ready to write down the probability distribution in the Bayesian framework for a single target value. It is the integration over all possible fitted models of the probability of the target value weighted by the prior probability of each fitted model:

$$\Pr(t^{(i)}|\bar{\mathbf{x}}^{(i)}, \Psi) = \int \Pr(t^{(i)}|\bar{\mathbf{x}}^{(i)}, \mathcal{M}) \Pr(\mathcal{M}|\Psi) d\mathcal{M}.$$

Note that $\mathcal{M}$ does not appear on the left-hand side of this equation, because we are integrating over all possible fitted models. However, Ψ does appear, because we are only considering one particular set of hyperparameters. Note that it is possible to integrate over hyperparameters. This requires further hyperparameters to specify a distribution to integrate over and is known as hierarchical Bayesian modeling. This is in fact part of Williams and Rasmussen's (1996) and Neal's (1997) Bayesian approach to Gaussian process modeling. The above equation is sufficient to capture this: the intermediate hyperparameters can be assumed to be subsumed into the model $\mathcal{M}$ and the hyperparameters Ψ are the final, top-level ones.

It is now straightforward to write down the probability of observing the entire set of training data in a Bayesian approach:

$$\Pr(\{t^{(i)}\}_{i=1}^{N}|\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^{N}, \Psi) = \int \Pr(\{t^{(i)}\}_{i=1}^{N}|\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^{N}, \mathcal{M}) \Pr(\mathcal{M}|\Psi) d\mathcal{M}. \quad (2)$$

Note that this is not the likelihood of a particular fitted model $\mathcal{M}$, but can be viewed as the likelihood of a set of hyperparameters. However, if we liked, we could consider this set of hyperparameters to be a specification of a higher-level "model".

The above just gives us a way of calculating the probability of a set of training data. We are often interested in predicting the target at some new point in the input space, call it $\bar{\mathbf{x}}^{(N+1)}$, conditioned on the training data. Using a version of the chain rule ($\Pr(A|B,C) = \Pr(A,B|C)/\Pr(B|C)$), the conditional distribution of $t^{(N+1)}$ can be written as follows:

$$\Pr(t^{(N+1)}|\bar{\mathbf{x}}^{(N+1)}, \{(\bar{\mathbf{x}}^{(i)}, t^{(i)})\}_{i=1}^{N}, \Psi) = \frac{\Pr(\{t^{(i)}\}_{i=1}^{N+1}|\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^{N+1}, \Psi)}{\Pr(\{t^{(i)}\}_{i=1}^{N}|\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^{N}, \Psi)}. \quad (3)$$

Note that all we need to make predictions is the probability of some set of data. This presages one of the aspects of Gaussian process modeling that can be confusing to newcomers: unlike with neural network models, there is no *simple* functional form giving the prediction of the model at a training point. This is not to say that such a function does not exist or that cannot compute such a prediction; indeed the above equation can be used for this. But, unlike in the maximum likelihood approach to neural network modeling, there is no search for optimal parameters that involves computing predictions of the fitted model at training points. All that is needed in Gaussian process modeling is the ability to compute the probability of some entire set of data given a set of hyperparameters.

Predicting values at new data points in the Bayesian approach (Eqn 3) requires integrating over the space of possible model fits (Eqn 2) (which possibly involves integrating over intermediate level hyperparameters). When the model is a neural networks this is difficult but can be done by making approximations as in MacKay's (1992) Evidence framework, or by integrating numerically using Monte Carlo methods (Neal 1996). One of the advantages of Gaussian process approach to modeling is that it gives an exact analytic form for Eqn 2 and allows one to perform Bayesian analysis using matrix manipulations (at least when there are no intermediate hyperparameters).

3.4 Gaussian process modeling

A *Gaussian process* is a collection of variables $\bar{\mathbf{t}} = (t^{(1)}, \ldots, t^{(n)})$ which have a joint distribution specified by a Gaussian:

$$\Pr(\bar{\mathbf{t}}) = \frac{1}{Z} \exp\left(-\frac{1}{2}(\bar{\mathbf{t}} - \bar{\mu})^T \mathbf{C}^{-1} (\bar{\mathbf{t}} - \bar{\mu})\right),$$

where $\bar{\mu}$ is a vector of means, $\mathbf{C}$ is a covariance matrix (which is symmetric), and Z is a normalizing constant (Z is equal to $(2\pi)^{N/2}\sqrt{\det(\mathbf{C})}$.)

We can use a Gaussian process for modeling by treating target values as the collection of values that are a Gaussian distribution and deriving the covariance matrix $\mathbf{C}$ from the training and testing data data. It is important to note that the entire collection of target values is treated as a *single* sample from a high-dimensional distribution.

The vector of means $\bar{\mu}$ is usually taken to be zero. The matrix $\mathbf{C}$ is usually computed as a parameterized function of the input vectors; it has entries

$$C_{kj} \quad = \quad C(\bar{\mathbf{x}}^{(j)}, \bar{\mathbf{x}}^{(k)}; \Theta). \tag{4}$$

The parameters Θ of this function are customarily referred to as hyperparameters because of their close relationship to the hyperparameters of a neural network. The exact form of $\mathbf{C}$ will be discussed in the next section, but it is usually based on a some sort of distance function so that if the input vectors $\bar{\mathbf{x}}^{(j)}$ and $\bar{\mathbf{x}}^{(k)}$ are close, C_{kj} will be high, making the targets $t^{(j)}$ and $t^{(k)}$ highly correlated. Since $\mathbf{C}$ depends only on the input vectors $\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^N$ and the hyperparameters Θ, we can write the probability of the data given the fitted model as follows (this is the analogue of Eqn 2):

$$\Pr(\{t^{(i)}\}_{i=1}^N | \{\bar{\mathbf{x}}^{(i)}\}_{i=1}^N, \Theta) \quad = \quad \frac{1}{Z} \exp\left(-\frac{1}{2}\bar{\mathbf{t}}_N^T \mathbf{C}^{-1} \bar{\mathbf{t}}_N\right), \tag{5}$$

where $\bar{\mathbf{t}}_N$ is the vector of target values $\{t^{(i)}\}_{i=1}^N$, and $\mathbf{C}$ is a function of Θ and $\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^N$ as before. Note that there is no additional term allowing for noise in the target value; in Gaussian process modeling the priors on the noise and modeling functions are both combined into the covariance matrix.

The conditional distribution over the target value $t^{(N+1)}$ for input vector $\bar{\mathbf{x}}^{(N+1)}$ given a training set $\{(\bar{\mathbf{x}}^{(i)}, t^{(i)})\}_{i=1}^N$ and hyperparameters Θ can be derived from Eqns 5 and 3. We use $\bar{\mathbf{t}}_{N+1}$ to denote the vector of target values $\{t^{(i)}\}_{i=1}^{N+1}$, and $\mathbf{C}_N$ and $\mathbf{C}_{N+1}$ to denote the covariance matrices for the sets of input vectors $\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^{N}$ and $\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^{N+1}$ respectively (using the same Θ). Since this conditional density distribution is the ratio of two Gaussian density functions it simplifies to a single Gaussian distribution:

$$\begin{aligned}
\Pr(t^{(N+1)}|\bar{\mathbf{x}}^{(N+1)}, \{(\bar{\mathbf{x}}^{(i)}, t^{(i)})\}_{i=1}^{N}, \Theta) &= \frac{\Pr(\bar{\mathbf{t}}_{N+1}|\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^{N+1}, \Theta)}{\Pr(\bar{\mathbf{t}}_{N}|\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^{N}, \Theta)} \\
&= \frac{Z_N}{Z_{N+1}} \frac{\exp\left(-\frac{1}{2}\bar{\mathbf{t}}_{N+1}^T \mathbf{C}_{N+1}^{-1} \bar{\mathbf{t}}_{N+1}\right)}{\exp\left(-\frac{1}{2}\bar{\mathbf{t}}_{N}^T \mathbf{C}_{N}^{-1} \bar{\mathbf{t}}_{N}\right)} \\
&= \frac{Z_N}{Z_{N+1}} \exp\left[-\frac{1}{2}(\bar{\mathbf{t}}_{N+1}^T \mathbf{C}_{N+1}^{-1} \bar{\mathbf{t}}_{N+1} - \bar{\mathbf{t}}_{N}^T \mathbf{C}_{N}^{-1} \bar{\mathbf{t}}_{N})\right]. \qquad (6)
\end{aligned}$$

Note that the covariance matrix $\mathbf{C}_{N+1}$ is just $\mathbf{C}_N$ with an extra row and column attached:

$$\mathbf{C}_{N+1} = \left[\begin{array}{c|c} \mathbf{C}_N & \bar{\mathbf{k}} \\ \hline \bar{\mathbf{k}}^T & \kappa \end{array}\right],$$

where the vector $\bar{\mathbf{k}}$ and the scalar κ are defined as follows:

$$\begin{aligned}
\bar{\mathbf{k}} &= (C(\bar{\mathbf{x}}^{(1)}, \bar{\mathbf{x}}^{(N+1)}; \Theta), \ldots, C(\bar{\mathbf{x}}^{(N)}, \bar{\mathbf{x}}^{(N+1)}; \Theta)) \\
\kappa &= C(\bar{\mathbf{x}}^{(N+1)}, \bar{\mathbf{x}}^{(N+1)}; \Theta).
\end{aligned}$$

Collecting together the terms in Eqn 6 that are functions of $t^{(N+1)}$, we can express the Gaussian distribution of $t^{(N+1)}$ at $\bar{\mathbf{x}}^{(N+1)}$:

$$\Pr(t^{(N+1)}|\bar{\mathbf{x}}^{(N+1)}, \{(\bar{\mathbf{x}}^{(i)}, t^{(i)})\}_{i=1}^{N}, \Theta) = \frac{1}{Z} \exp\left(\frac{-(t^{(N+1)} - \hat{t}_{N+1})^2}{2\sigma_{\hat{t}_{N+1}}^2}\right), \qquad (7)$$

where $\hat{t}_{N+1} = \bar{\mathbf{k}}^T \mathbf{C}_N^{-1} \bar{\mathbf{t}}_N$ is the mean of the prediction and $\sigma_{\hat{t}_{N+1}}^2 = \kappa - \bar{\mathbf{k}}^T \mathbf{C}_N^{-1} \bar{\mathbf{k}}$ is its variance (the error bars on the prediction are $\pm\sigma_{\hat{t}_{N+1}}$).

Neal's (1996) work on the relationship between Gaussian process models and Bayesian neural networks models concerned predictive models having the form of Eqn 3 in which $\mathcal{M}$ was a neural network. The priors on the networks were such that as the number of hidden units increased, the weight magnitudes were expected to be smaller. This approach to modeling requires an integration over all possible parameters (weight values) for the networks. Neal showed that as the number of hidden units approaches infinity, the properties of such a Bayesian neural network model approach those of a Gaussian process model given by the analytic expression in Eqn 7.

3.5 The covariance matrix for a Gaussian process model

We are now in a position to specify the exact form of the covariance function. The only constraint is that it should produce a non-negative definite covariance matrix for any set of points $\{\bar{\mathbf{x}}^{(i)}\}_{i=1}^{N}$. Williams and Rasmussen (1996), Rasmussen (1996), and Neal (1997) use variations on the following covariance function:

$$C(\bar{\mathbf{x}}^{(k)}, \bar{\mathbf{x}}^{(j)}; \Theta) = c + \sum_{u=1}^{d} \sigma_u^2 x_u^{(k)} x_u^{(j)} + \eta^2 \exp\left(-\sum_{u=1}^{d} \rho_u^2 (x_u^{(k)} - x_u^{(j)})^2\right) + \zeta^2 \delta_{kj}, \qquad (8)$$

where d is the input dimension as before, $\Theta = (c, \{\sigma_u\}_{u=1}^{d}, \eta, \{\rho_u\}_{u=1}^{d}, \zeta)$ is the set of hyperparameters for the Gaussian process, and $\delta_{kj} = 1$ for $k = j$ and 0 otherwise.

The effects of the various hyperparameters can be explained in terms of their effect on the regression function computed by the Gaussian process model. The hyperparameter c (> 0) is a constant part of the covariance, which is the same for any pair of inputs. This adds a constant component to the regression function. The hyperparameters σ_u control the linear part of the covariance and thereby the linear component of the regression function. The hyperparameters η and the ρ_u control the contribution of a distance-based part of the covariance, which adds a nearest-neighbor type smoothing component to the regression function. η controls how much the regression function varies across inputs, i.e., the range of the $t^{(i)}$. A high value of η corresponds to a large range in the output values. ρ_u control the *length scales* on individual input dimensions. When ρ_u is small, the distance between $x_u^{(k)}$ and $x_u^{(j)}$ has little bearing on the covariance, which means that the regression function is relatively unaffected by the uth input dimension. The explanation of the covariance matrix in terms of distances between inputs in Section 3.1 is based on the assumption that this exponential term dominates the covariance. The hyperparameter ζ accounts for noise in the outputs

Predictions from a fitted Gaussian process model can be based on maximum likelihood values for the hyperparameters, i.e., the values of the hyperparameters which maximize Eqn 5. Predictions can also be made in a hierarchical Bayesian framework by specifying a prior distribution over the hyperparameters Θ, and then integrating over Θ using Markov Chain Monte Carlo (MCMC) sampling of points in Θ space (to approximate an integral like that in Eqn 2). The publicly available software described in Neal (1997) performs this MCMC sampling; this software was used for the experiments described in this paper.

3.6 Introducing additive components into the covariance function

A Gaussian process model can easily be made to be additive by changing the covariance function to be an additive function of distances on different input dimensions (Rasmussen 1996; Neal 1997). In fact, a general exponential part can also be included, which gives the model both additive and general smoothing components. The covariance function from Neal (1997), which is the one used in this paper is as follows:

$$C(\bar{\mathbf{x}}^{(k)}, \bar{\mathbf{x}}^{(j)}; \Theta) = c + \eta_0^2 \exp\left(-\sum_{u=1}^{d} \rho_u^2 (x_u^{(k)} - x_u^{(j)})^2\right) + \sum_{u=1}^{d} \eta_u^2 \exp\left(-\phi_u^2 (x_u^{(k)} - x_u^{(j)})^2\right) + \zeta^2 \delta_{kj} \quad (9)$$

where Θ now includes the additional hyperparameters $\{\eta_u\}_{u=1}^d$ and $\{\phi_u\}_{u=1}^d$. Note that the linear term from Eqn 8 (involving σ_u) has been dropped. The first exponential term in this formula, which is the same as above, is the general term that can take account of distance in any direction. The other exponential terms (in the sum) can each only look at distances in directions which line up with axes in the input space.. Thus, if $\eta_0 = 0$, the covariance function is an additive function of the the differences $(x_u^{(k)} - x_u^{(j)})$. Since the predictions of a fitted Gaussian process model are additive with respect to additive terms in the covariance, the predictions of the fitted model will be additive (provided $\eta_0 = 0$). In some cases where the output actually is an additive function on the inputs, the general exponential part is automatically found to have zero importance (i.e, $\eta_0 = 0$).

The technique suggested in this paper is to produce a series of fitted models that range from a general function of inputs to a purely additive function of inputs. This can be accomplished as follows: (1) Fit a model with all parameters free (predictions are made based on the MCMC samples from the posterior density); (2) Fit a series of models with decreasing fixed values of η_0. When η_0 is very small, the model will be additive. The proposal of this paper is that insight into the input-output relationship in the data can be gained by visualizing the function computed, and observing the quality of predictions and the degree of non-additivity, as the model is forced to be more and more additive.

4 Visualizing Gaussian process models with additive plots

Plate et al (1997) describe how additive plots (Hastie and Tibshirani 1990) can be adapted for visualizing non-additive models. Consider a general function of d variables, $f(x_1, x_2, \ldots, x_d)$, and the problem of understanding how f varies with the x_i. If $d = 2$ we can plot a surface, but if d is higher, conventional plotting techniques fail us. The problem is that f is not a function of x_i – its value be different depending upon the remaining x_j.

Five possible methods for visualizing the effect of individual inputs on a multivariate function $f(x_1, x_2, \ldots, x_d)$ are described below. For ease of notation, these methods are described with respect to the first input, x_1. They are also illustrated in Figure 1 on the following two 2-d functions from Hwang et al (1994)):

Additive function:

$$g^{(4)}(x_1, x_2) = 1.3356(1.5(1 - x_1) + \mathrm{e}^{2x_1 - 1}\sin(3\pi(x_1 - 0.6)^2) + \mathrm{e}^{3(x_2 - 0.5)}\sin(4\pi(x_2 - 0.9)^2))$$

Complex interaction function:

$$g^{(5)}(x_1, x_2) = 1.9(1.35 + \mathrm{e}^{x_1}\sin(13(x_1 - 0.6)^2)\mathrm{e}^{-x_2}\sin(7x_2))$$

collection of projections (a) Plot 2d projections of the higher-dimensional surface. This can be done by plotting a number of curves of f versus x_1, each with fixed values for the remaining $x_2, \ldots, x_d$. The values observed in the training data are reasonable candidates for these fixed values (taken a vector at a time). This method is unsatisfactory because it not only presents too much information but does not even result in a simple plot for functions that are additive.

collection of translated projections (b) Plot projections with translated to line up at a common reference values, i.e., the curve $(f(x, x_2^{(k)}, \ldots, x_d^{(k)}) - f(b_1, x_2^{(k)}, \ldots, x_d^{(k)})$ vs. x for each training example k. b_1 is a reference (base) value for the first dimension — good choices are zero or the mean of x_1. The translation removes the additive effects of variables other than x_1. These plots show the magnitude of effect that x_1 can have. For fitted models that are additive this method reproduces exactly the plots of the individual component functions. However, for non-additive fitted models there are too many curves, especially when the input space is high-dimensional and there are many training examples. Large segments of the curves may exist in sparsely populated regions of the input space, where the behaviour of the fitted model is neither well-determined by the input data, nor particularly relevant to performance on new data points drawn from the same distribution.

scatter plot of gradients (c) Make a scatter plot of the gradients of f with respect to x_1 at the training data points, i.e., the points $\partial f/\partial x_1 \mid_{\bar{\mathrm{X}}^{(k)}}$ for each example k. While this is less busy, the total magnitude of effect of x_1 is not immediately obvious because the vertical axis is the gradient of f.

scatter plot of additive effects (d) Make a less busy version of (b) by plotting a single point from each curve: the value of f at $x_1 = x_1^{(k)}$ from the curve for example k. This is done by plotting the point $(x_1^{(k)}, f(x_1^{(k)}, x_2^{(k)}, \ldots, x_d^{(k)}) - f(b_1, x_2^{(k)}, \ldots, x_d^{(k)}))$ for each training example k, where b_1 is the reference value as before.

additive effects with slopes (e) Combine the information from (c) and (d) by embellishing the scatter-plot of points (d) with short line segments indicating the partial gradient of f with respect to x_i at each point.

The additive effects plots with slopes (e) (referred to as "additive effects plots" for the remainder of this paper) present the most reasonable amount of information in an interpretable manner. For fitted