A Comparative Evaluation of Additive Separability Tests for Physics-Informed Machine Learning

Zi-Yu Khoo[1], Jonathan Sze Choong Low[2], and Stéphane Bressan[13]

[1] National University of Singapore. 21 Lower Kent Ridge Rd, Singapore 119077
khoozy@comp.nus.edu.sg, steph@nus.edu.sg
[2] Singapore Institute of Manufacturing Technology, Agency for Science, Technology and Research (A*STAR), Singapore 138634 sclow@simtech.a-star.edu.sg
[3] CNRS@CREATE LTD, 1 Create Way, Singapore 138602

Abstract. Many functions characterising physical systems are additively separable. This is the case, for instance, of mechanical Hamiltonian functions in physics, population growth equations in biology, and consumer preference and utility functions in economics. We consider the scenario in which a surrogate of a function is to be tested for additive separability. The detection that the surrogate is additively separable can be leveraged to improve further learning. Hence, it is beneficial to have the ability to test for such separability in surrogates. The mathematical approach is to test if the mixed partial derivative of the surrogate is zero; or empirically, lower than a threshold. We present and comparatively and empirically evaluate the eight methods to compute the mixed partial derivative of a surrogate function.

Keywords: Second-order bias · inductive bias · symbolic regression

1 Introduction

Many functions characterising physical systems are *additively separable*, such as mechanical Hamiltonian functions in physics [16], population growth equations in biology [13], and consumer preference and utility functions in economics [26].

We consider the scenario in which a machine learning model is learning a surrogate of a function without the information that it is additively separable. Testing the surrogate for the property of additive separability is of interest. Physics-informed neural networks [10] combine the universal function approximation capability of neural networks [9] with the information of the symbolic properties, laws, and constraints of the underlying application domain [15,14,6,23,27]. With information regarding the additive separability of the function, physics-informed neural networks can leverage the additive separability property to improve further learning of the surrogate [11].

The mathematical approach to test for additive separability is to check if the mixed partial derivative of the surrogate is zero. Empirically, the mixed partial derivative of the surrogate should be lower than a small threshold close to zero.

Information regarding the additive separability of a surrogate has seen several usages in physics-informed machine learning applications. In modelling Hamiltonian dynamics, for instance, Gruver et al., in [7] argue that almost all improvement of existing works [2] is due to a second-order bias. This second-order bias results from the additive separability of the modelled Hamiltonian into the system's potential and kinetic energy as functions of position and momentum, respectively. The additive separability allowed the physics-informed neural network to "*avoid[...] artificial complexity from its coordinate system*" (the input variables) and improve its performance [7]. In symbolic regression, Udrescu and Tegmark iteratively test an unknown function for additive separability to divide a symbolic regression problem into two simpler ones that can be tackled separately. This "*guarantee[s] that more accurate symbolic expressions [are] closer to the truth*", improving the performance of the symbolic regression algorithm [24]. Our work to test for additive separability creates the opportunity to leverage additive separability to improve the learning of surrogates as observed in the works of Gruver and Udrescu.

We introduce eight methods to compute the mixed partial derivative of a machine learning model, specifically a multilayer perceptron neural network learning a surrogate of a function. Our surrogate of choice is the multilayer perceptron neural network, although theoretically, any differentiable machine learning model will suffice. The first four of the eight methods compute the mixed partial derivative via finite difference of the multilayer perception neural network. Three methods arise from the different methods to automatically compute mixed partial derivatives using automatic differentiation of the multilayer perceptron neural network. The last method arises from the symbolic differentiation of a multilayer perceptron neural network. We present and comparatively and empirically evaluate the performance of eight methods in computing the mixed partial derivative of the surrogate functions.

The remainder of this paper is structured as follows. Section 2 presents the necessary background for additive separability and multilayer perception neural network surrogates. Section 3 presents the eight methods to compute the mixed partial derivative of a multilayer perception neural network. Section 4 presents and discusses the results of an empirical comparative evaluation of the eight methods. Section 5 concludes the paper.

2   Background and Related Work

2.1   Additive Separability

An additively separable real function $f(\vec{x}, \vec{y}) \in \mathbb{R}$ is of the form $f(\vec{x}, \vec{y}) = g(\vec{x}) + h(\vec{y})$, where $\vec{x} \in \mathbb{R}^n$, $\vec{y} \in \mathbb{R}^m$ are vectors representing disjoint subsets of $\mathbb{R}^{m+n}$ input variables and $g(\vec{x}), f(\vec{y}) \in \mathbb{R}$. x_n and y_m denote elements of $\vec{x}$ and $\vec{y}$.

The necessary and sufficient condition for a function to be additively separable is that the mixed partial derivative of the function equals zero. The mixed partial derivative is the second derivative of the function. The first derivative is

taken with respect to an element in either $\vec{x}$ or $\vec{y}$, and the second derivative is taken with respect to an element in either $\vec{y}$ or $\vec{x}$. This condition is shown in Equation 1, where the mixed partial derivative is found with respect to x_n, an element in $\vec{x}$ first, then y_m, an element in $\vec{y}$.

$$\frac{\partial^2 f(\vec{x}, \vec{y})}{\partial x_n \partial y_m} = \frac{\partial}{\partial y_m}\left(\frac{\partial(g(\vec{x}) + h(\vec{y}))}{\partial x_n}\right) = \frac{\partial}{\partial y_m}\left(\frac{\partial g(\vec{x})}{\partial x_n}\right) = 0 \tag{1}$$

Finite difference methods refer to those that obtain a numerical solution for partial derivatives by replacing the derivatives with their appropriate numerical differentiation formulae. In general, a finite difference approximation of the value of some derivative of a scalar function $f(x)$ at a point in its domain relies on a suitable combination of sampled function values at its nearby points [17].

Starting with the first-order derivative, the simplest finite difference approximation for a multivariate function $f(\vec{x}, \vec{y})$ is the ordinary difference quotient shown in Equation 2 where the function $f(\cdot)$ is sampled at $f(\vec{x}, \vec{y})|_{x_n = x_N + h}$ and $f(\vec{x}, \vec{y})|_{x_n = x_N}$ and all other elements are kept constant. x_N is a scalar sample of the element x_n in $\vec{x}$, and h is the scalar distance between the two samples of x_n. Indeed, if $f(\cdot)$ is differentiable at x_N then $\frac{\partial f(\vec{x}, \vec{y})}{\partial x}|_{x=x_N}$ is by definition, the limit, as $h \to 0$ of the finite difference quotients [17].

$$\frac{\partial f(\vec{x}, \vec{y})}{\partial \vec{x}}|_{x_n = x_N} = \frac{f(\vec{x}, \vec{y})|_{x_n = x_N + h} - f(\vec{x}, \vec{y})|_{x_n = x_N}}{h} \tag{2}$$

The finite difference for a multivariate function is analogous to partial derivatives in several variables. The finite difference analogue of Equation 1 is shown in Equation 3. The components of the numerator of Equation 3 are defined in Equations 4 to 7. x_N and y_M are scalar samples of elements x_n and y_m respectively, and h and k are scalar. All other elements are kept constant.

$$\frac{\partial^2 f(\vec{x}, \vec{y})}{\partial x_n \partial y_m}\bigg|_{\substack{x_n = x_N \\ y_m = y_M}}$$
$$= \frac{f(x_N + h, y_M + k) - f(x_N + h, y_M) - f(x_N, y_M + k) + f(x_N, y_M)}{h \times k} \tag{3}$$

$$f(x_N + h, y_M + k) = f(\vec{x}, \vec{y})|_{x_n = x_N + h, y_m = y_M + k} \tag{4}$$
$$f(x_N + h, y_M) = f(\vec{x}, \vec{y})|_{x_n = x_N + h, y_m = y_M} \tag{5}$$
$$f(x_N, y_M + k) = f(\vec{x}, \vec{y})|_{x_n = x_N, y_m = y_M + k} \tag{6}$$
$$f(x_N, y_M) = f(\vec{x}, \vec{y})|_{x_n = x_N, y_m = y_M} \tag{7}$$

Seminal works that test for additive separability of a function make use of the finite difference of the function. Udrescu et al. [24] and Bellenot [1] state that for a function to be additively separable, for every pair of samples (x_a, y_a) and (x_b, y_b) where x is to be additively separable from y, the difference between the two pairwise sums of the values of the function at diagonally opposite corners of

the rectangle $\langle (x_a, y_a), (x_b, y_a), (x_b, y_b), (x_a, y_b) \rangle$ equals zero [1,24]. This is shown for a bivariate function $f(x, y)$ in Equation 8.

$$\forall x_a, x_b \in x \quad \forall y_a, y_b \in y$$
$$(f(x_a, y_a) + f(x_b, y_b)) - (f(x_a, y_b) + f(x_b, y_a)) = 0 \qquad (8)$$

Equation 8 and Equation 3 are equivalent in the case where Equation 3 describes a bivariate function with the inputs x_N and y_M, after substituting $x_N = x_a$, $x_N + h = x_b$, $y_M = y_a$, $y_M + k = y_b$, and $h = k = 1$.

Equation 8 can be generalised to test multivariate functions for additive separability. A multivariate function $f(\vec{x}, \vec{y})$ is additive separable if Equation 8 holds, and all elements of $\vec{x}$ and $\vec{y}$ except x_N and y_M are kept constant.

The set of additively separable functions is closed under operations including addition, multiplication by constants, partial derivatives, and integrals with respect to either of the disjoint subsets of input variables [1]. Therefore a test for additive separability only has to decompose a function into two components and can be applied repeatedly to a function that is multiply additively separable.

Several other tests for additive separability have been proposed in the literature, in particular in economics. These tests make assumptions that limit their applicability to specific families of functions. The seminal work of Leontief [12] proposed a test for additive separability for functions with three or more variables and non-zero first derivatives. Gorman [5], Varian [25], Diewert and Parkan [3] and Fleissig and Whitney [4] developed tests for additive separability that were specific to concave and monotonic functions. Polisson, Quah and Renou [21] and Polisson [20] developed tests for additive separability that assume non-satiation or a positive correlation between the input variables and output of a function.

2.2   Multilayer Perceptron Neural Network Surrogates

Multilayer perceptrons are regression tools [8] that are universal function approximators [9]. They can approximate any function to any degree of accuracy from one finite dimensional space to another.

We consider a multilayer perceptron neural network with one input layer, one hidden layer, and one output layer. The multilayer perception neural network regresses an output, shown in Equation 9. In Equation 9, the neural network has two inputs, x_1 and x_2, and one output, $f(x_1, x_2)$. σ_1 and σ_2 are the activation functions for the input and hidden layer respectively. $F^\intercal$ is the 3 by 2 weight matrix of the input layer, $G^\intercal$ is the 3 by 3 weight matrix of the hidden layer, and $H^\intercal$ is the 1 by 3 weight matrix of the output layer. B is the 1 by 3 bias matrix of the input layer, and C is the 1 by 3 bias matrix of the hidden layer. The elements of $F^\intercal$, $G^\intercal$, $H^\intercal$, B and C are denoted by w, v, n, b and c respectively. Their subscripts indicate the row and column of the element in the matrix. The output of the input layer is denoted as W.

$$f(x_1, x_2) = n_1\sigma_2(v_{11}\sigma_1(W_1) + v_{12}\sigma_1(W_2) + v_{13}\sigma_1(W_3) + c_1)$$
$$+ n_2\sigma_2(v_{21}\sigma_1(W_1) + v_{22}\sigma_1(W_2) + v_{23}\sigma_1(W_3) + c_2)$$
$$+ n_3\sigma_2(v_{31}\sigma_1(W_1) + v_{32}\sigma_1(W_2) + v_{33}\sigma_1(W_3) + c_3) \quad (9)$$

Where the outputs of the input layer are denoted as W and their subscripts enumerate the three outputs of the layer.

$$W_1 = w_{11}x_1 + w_{12}x_2 + b_1$$
$$W_2 = w_{21}x_1 + w_{22}x_2 + b_2$$
$$W_3 = w_{31}x_1 + w_{32}x_2 + b_3 \quad (10)$$

In the context of additively separable functions and their mixed partial derivatives, the derivatives of the output of the multilayer perceptron neural network should be considered. The first derivative of the output of the multilayer perceptron neural network, $f(x_1, x_2)$, with respect to its input, x_1, is computed in Equation 11. σ'_{1,x_1} and σ'_{2,x_1} is the first derivative of the activation functions σ_1 and σ_2 with respect to x_1. The output of the hidden layer is denoted as V.

$$\frac{\partial f(x_1, x_2)}{\partial x_1}$$
$$= n_1\sigma'_{2,x_1}(V_1) \times (v_{11}w_{11}\sigma'_{1,x_1}(W_1) + v_{12}w_{21}\sigma'_{1,x_1}(W_2) + v_{13}w_{31}\sigma'_{1,x_1}(W_3))$$
$$+ n_2\sigma'_{2,x_1}(V_2) \times (v_{21}w_{11}\sigma'_{1,x_1}(W_1) + v_{22}w_{21}\sigma'_{1,x_1}(W_2) + v_{23}w_{31}\sigma'_{1,x_1}(W_3))$$
$$+ n_3\sigma'_{2,x_1}(V_3) \times (v_{31}w_{11}\sigma'_{1,x_1}(W_1) + v_{32}w_{21}\sigma'_{1,x_1}(W_2) + v_{33}w_{31}\sigma'_{1,x_1}(W_3))$$
$$(11)$$

Where

$$V_1 = v_{11}\sigma_1(W_1) + v_{12}\sigma_1(W_2) + v_{13}\sigma_1(W_3) + c_1$$
$$V_2 = v_{21}\sigma_1(W_1) + v_{22}\sigma_1(W_2) + v_{23}\sigma_1(W_3) + c_2$$
$$V_3 = v_{31}\sigma_1(W_1) + v_{32}\sigma_1(W_2) + v_{33}\sigma_1(W_3) + c_3$$

The mixed partial derivative of the multilayer perceptron neural network is shown in Equation 12 where σ'_{1,x_1}, σ'_{1,x_2}, σ'_{2,x_1} and σ'_{2,x_2} are the first derivatives for the activation functions for the input and hidden layer with respect to x_1 and x_2 respectively, and σ''_1 and σ''_2 are the second derivatives for the activation functions for the input and hidden layer with respect to both x_1 and x_2 respectively.

$$\frac{\partial^2 f(x_1, x_2)}{\partial x_1 \partial x_2}$$
$$= n_1\sigma''_2(V_1) \times (VW_{1,x_1}) \times (VW_{1,x_2}) + n_1\sigma'_{2,x_1}(V_1) \times (VW''_1)$$
$$= n_2\sigma''_2(V_2) \times (VW_{2,x_1}) \times (VW_{2,x_2}) + n_2\sigma'_{2,x_1}(V_2) \times (VW''_2)$$
$$= n_3\sigma''_2(V_3) \times (VW_{3,x_1}) \times (VW_{3,x_2}) + n_3\sigma'_{2,x_1}(V_3) \times (VW''_3) \quad (12)$$

Where VW'_{1,x_1}, VW'_{1,x_2}, VW'_{2,x_1}, VW'_{2,x_2}, VW'_{3,x_1}, VW'_{3,x_2}, VW''_1, VW''_2 and VW''_3 are computed using the chain rule.

$$VW'_{1,x_1} = v_{11} \times w_{11} \times \sigma'_{1,x_1}(W_1) + v_{12} \times w_{21} \times \sigma'_{1,x_1}(W_2) + v_{13} \times w_{31} \times \sigma'_{1,x_1}(W_3)$$
$$VW'_{1,x_2} = v_{11} \times w_{21} \times \sigma'_{1,x_2}(W_1) + v_{12} \times w_{22} \times \sigma'_{1,x_2}(W_2) + v_{13} \times w_{32} \times \sigma'_{1,x_2}(W_3)$$
$$VW''_1 = v_{11} \times w_{11} \times w_{12} \times \sigma''_1(W_1) + v_{12} \times w_{21} \times w_{22} \times \sigma''_1(W_2)$$
$$+ v_{13} \times w_{31} \times w_{32} \times \sigma''_1(W_3)$$

$$VW'_{2,x_1} = v_{21} \times w_{11} \times \sigma'_{1,x_1}(W_1) + v_{22} \times w_{21} \times \sigma'_{1,x_1}(W_2) + v_{23} \times w_{31} \times \sigma'_{1,x_1}(W_3)$$
$$VW'_{2,x_2} = v_{21} \times w_{21} \times \sigma'_{1,x_2}(W_1) + v_{22} \times w_{22} \times \sigma'_{1,x_2}(W_2) + v_{23} \times w_{32} \times \sigma'_{1,x_2}(W_3)$$
$$VW''_2 = v_{21} \times w_{11} \times w_{12} \times \sigma''_1(W_1) + v_{22} \times w_{21} \times w_{22} \times \sigma''_1(W_2)$$
$$+ v_{23} \times w_{31} \times w_{32} \times \sigma''_1(W_3)$$

$$VW'_{3,x_1} = v_{31} \times w_{11} \times \sigma'_{1,x_1}(W_1) + v_{32} \times w_{21} \times \sigma'_{1,x_1}(W_2) + v_{33} \times w_{31} \times \sigma'_{1,x_1}(W_3)$$
$$VW'_{3,x_2} = v_{31} \times w_{21} \times \sigma'_{1,x_2}(W_1) + v_{32} \times w_{22} \times \sigma'_{1,x_2}(W_2) + v_{33} \times w_{32} \times \sigma'_{1,x_2}(W_3)$$
$$VW''_3 = v_{31} \times w_{11} \times w_{12} \times \sigma''_1(W_1) + v_{32} \times w_{21} \times w_{22} \times \sigma''_1(W_2)$$
$$+ v_{33} \times w_{31} \times w_{32} \times \sigma''_1(W_3)$$

3   Methodology

We consider the scenario in which a machine learning model is a surrogate of a function without the information regarding additive separability of the function. The surrogate is a multilayer perceptron neural network that learns a multivariate function $f(\vec{x}, \vec{y})$. We design eight methods to compute the mixed partial derivative of the surrogate, to test the surrogate, and hence the unknown function, for additive separability. This section describes the design of the eight methods.

The first four methods compute the mixed partial derivative via finite difference. Three methods arise from the different methods to automatically compute mixed partial derivatives using automatic differentiation of the surrogate. The last method arises from the symbolic differentiation of the surrogate.

Method 1 computes the mixed partial derivative via Equation 3 by evaluating the surrogate at (x_N, y_M), $(x_N + h, y_M)$, $(x_N, y_M + k)$ and $(x_N + h, y_M + k)$ in Equation 3, and setting $h = k = 1$.

Method 2 computes the mixed partial derivative via Equation 3 by evaluating the surrogate at (x_N, y_M), $(x_N + h, y_M)$, $(x_N, y_M + k)$ and $(x_N + h, y_M + k)$ in Equation 3, and setting h and k to be the distances between x_N and $x_N + h$, y_M and $y_M + k$ respectively.

Method 3 computes the mixed partial derivative via Equation 3 by evaluating the surrogate at (x_N, y_M), $(x_N + h, y_M)$, $(x_N, y_M + k)$ and $(x_N + h, y_M + k)$ in

Equation 3. However, it defines $x_N + h$ and $y_M + k$ to be the median of x_N and y_M respectively. It sets $h = k = 1$. We note that this is the methodology used by Udrescu et al. in their symbolic regression algorithm, AI Feynman [24].

Method 4 computes the mixed partial derivative via Equation 3 by evaluating the surrogate at (x_N, y_M), $(x_N + h, y_M)$, $(x_N, y_M + k)$ and $(x_N + h, y_M + k)$ in Equation 3. However, it defines $x_N + h$ and $y_M + k$ to be the median of x_N and y_M respectively. It sets h and k to be the distances between x_N and $x_N + h$, y_M and $y_M + k$ respectively.

We note that Methods 1 and 2 require a quadratic number of evaluations of the surrogate, while Methods 3 and 4 require a linear number of evaluations of the surrogate.

Method 5 computes the mixed partial derivative via Equation 1. The mixed partial derivative of the surrogate is computed using automatic differentiation, by taking the first derivative of the surrogate with respect to an element in $\vec{x}$, then taking a second derivative of the surrogate with respect to an element in $\vec{y}$.

Method 6 computes the mixed partial derivative via Equation 1. The mixed partial derivative of the surrogate is computed using automatic differentiation, by taking the first derivative of the surrogate with respect to an element in $\vec{y}$, then taking a second derivative of the surrogate with respect to an element in $\vec{x}$.

Method 7 computes the mixed partial derivative via Equation 1. The mixed partial derivative of the surrogate is computed using automatic differentiation, by finding the Hessian of the surrogate with respect to an element in $\vec{x}$ and an element in $\vec{y}$.

Method 8 computes the mixed partial derivative of a surrogate multilayer perceptron neural network symbolically, following Equation 12. Given a surrogate of a function, it creates a second surrogate multilayer perceptron neural network with the same weights and biases. This second surrogate multilayer perception neural network instead models the mixed partial derivative of the unknown function. The new surrogate multilayer perception neural network has layers and activations following Equation 12. The inputs of the second surrogate are the same as the first. The mixed partial derivative of the unknown function is the output of the new surrogate multilayer perceptron neural network.

4   Performance Evaluation

Eight classifiers are created based on the eight methods listed in Section 3. Independently, functions that are either additively or non-additively separable are created, and one surrogate is trained on each function. Each classifier is then given a trained surrogate. The eight classifiers each return a test output. The test outputs are aggregated to compare the eight classifiers. This section presents and discusses the results of the comparison of the eight methods.

4.1   Experimental Setup

Setup of the Additively and Non-Additively Separable Surrogates We create unknown functions that are either additively or non-additively separable. We train one surrogate for each unknown function.

   We create two- and three-variabled unknown functions comprising additive and multiplicative combinations of polynomial, trigonometric, exponential, radical and logarithmic uni-variate sub-functions, shown in Table 1. A total of 3744 additively and non-additively separable unknown functions were created.

Table 1. Twelve sub-functions with input n, a placeholder for variables x, y and z

Sub-functions			
$f(n) = n$	$f(n) = n^2$	$f(n) = (\frac{n}{3})^3$	$f(n) = \frac{1}{n+4}$
$f(n) = \sin(n)$	$f(n) = \cos(n)$	$f(n) = \sin(n)^2$	$f(n) = \cos(n)^2$
$f(n) = \exp(n)$	$f(n) = \log(n+4)$	$f(n) = \sqrt{|n|}$	$f(n) = n^{1/3}$

   One multilayer perceptron neural network surrogate is trained per unknown function. We select 30 data points uniformly at random within the range of $[-3, 3]$ for each input variable of each unknown function[4]. The data is input to the analytical form of the created unknown functions to get function outputs, which we call output data. Tuples of the input and output data are used as training data for a surrogate multilayer perceptron. Each surrogate multilayer perception neural network has two hidden layers of width 26 with softplus activation, mean squared error loss, batch size of 128 and Adam optimizer with learning rate 0.01. 80% of the data is used for training and 20% for validation. All models are trained to convergence using a validation-based dynamic stopping criteria [22] with patience of 500 epochs. All models are trained and evaluated on two GeForce GTX1080 GPUs, with 64 GB of RAM and 12 processors.

Setup of the Eight Classifiers The eight classifiers make use of the trained surrogate for each unknown function, $\hat{f}(\vec{x}, \vec{y})$ to evaluate additive separability. All eight classifiers compute the mixed partial derivative of the surrogate with respect to x_1 and y_1, which are the first elements in $\vec{x}$ and $\vec{y}$ respectively. The values of all other inputs to the surrogates are kept constant for each evaluation of each classifier. The eight classifiers compute the mixed partial derivatives on a test dataset, comprising 30 data points generated uniformly in a grid within the range of $[-3, 3]$ for each input variable of each unknown function.

   Classifier 1 computes the mixed partial derivative via Equation 3 by evaluating the surrogate at $\hat{f}(x_1, y_1)$, $\hat{f}(x_1 + h, y_1)$, $\hat{f}(x_1, y_1 + k)$ and $\hat{f}(x_1 + h, y_1 + k)$. The pair of points (x_1, y_1) and $(x_1 + h, y_1 + k)$ correspond to all pairwise combinations of samples from the test dataset. h and k are set to 1. For a test dataset

---

[4] We note that if data is generated evenly in a grid, Equation 1 (after computing the derivative between consecutive points) and Equation 8 can be evaluated for each unknown function immediately without a surrogate.

comprising n tuples of $(\vec{x}, \vec{y})$, this corresponds to $\frac{n(n-1)}{2}$ combinations of samples. The mixed partial derivative is averaged over all $\frac{n(n-1)}{2}$ combinations of samples for a comparative evaluation.

Classifier 2 computes the mixed partial derivative via Equation 3 by evaluating the surrogate at $\hat{f}(x_1, y_1)$, $\hat{f}(x_1 + h, y_1)$, $\hat{f}(x_1, y_1 + k)$ and $\hat{f}(x_1 + h, y_1 + k)$. The pair of points (x_1, y_1) and $(x_1 + h, y_1 + k)$ correspond to all pairwise combinations of samples from the test dataset. h and k are set to be the distances between each pair of samples of (x_1, y_1) and $(x_1 + h, y_1 + k)$. For a test dataset comprising n tuples of $(\vec{x}, \vec{y})$, this corresponds to $\frac{n(n-1)}{2}$ combinations of samples. The mixed partial derivative is averaged over all $\frac{n(n-1)}{2}$ combinations of samples for a comparative evaluation.

Classifier 3 computes the mixed partial derivative via Equation 3 by evaluating the surrogate at $\hat{f}(x_1, y_1)$, $\hat{f}(x_1 + h, y_1)$, $\hat{f}(x_1, y_1 + k)$ and $\hat{f}(x_1 + h, y_1 + k)$. $x_1 + h$ and $y_1 + k$ are the median from all samples of x_1 and y_1 from the test dataset respectively. h and k are set to 1. For a test dataset comprising n tuples of $(\vec{x}, \vec{y})$, the mixed partial derivative is averaged over all n samples for a comparative evaluation.

Classifier 4 computes the mixed partial derivative via Equation 3 by evaluating the surrogate at $\hat{f}(x_1, y_1)$, $\hat{f}(x_1 + h, y_1)$, $\hat{f}(x_1, y_1 + k)$ and $\hat{f}(x_1 + h, y_1 + k)$. $x_1 + h$ and $y_1 + k$ are the median from all samples of x_1 and y_1 from the test dataset respectively. h and k are set to be the distances between each pair of samples of (x_1, y_1) and $(x_1 + h, y_1 + k)$. For a test dataset comprising n tuples of $(\vec{x}, \vec{y})$, the mixed partial derivative is averaged over all n samples for a comparative evaluation.

Classifier 5 computes the mixed partial derivative via Equation 1. The mixed partial derivative of the surrogate, $\hat{f}(x_1, y_1)$, is computed using automatic differentiation via `autograd` in `pytorch` [18,19], at all samples (x_1, y_1) in the test dataset. This is done by taking the first derivative of the surrogate with respect to x_1, then finding the derivative of the surrogate again, but with respect to y_1. For a test dataset comprising n tuples of $(\vec{x}, \vec{y})$, the mixed partial derivative is averaged over all n samples for a comparative evaluation.

Classifier 6 computes the mixed partial derivative via Equation 1. The mixed partial derivative of the surrogate, $\hat{f}(x_1, y_1)$, is computed using automatic differentiation via `autograd` in `pytorch` [18,19], at all samples (x_1, y_1) in the test dataset. This is done by taking the first derivative of the surrogate with respect to y_1, then finding the derivative of the surrogate again, but with respect to x_1. For a test dataset comprising n tuples of $(\vec{x}, \vec{y})$, the mixed partial derivative is averaged over all n samples for a comparative evaluation.

Classifier 7 computes the mixed partial derivative via Equation 1. The mixed partial derivative of the surrogate, $\hat{f}(x_1, y_1)$, is computed using automatic differentiation via `torch.func` in `pytorch` [18,19], at all samples (x_1, y_1) in the test dataset. `torch.func` directly computes the Hessian of the surrogate. For a test dataset comprising n tuples of $(\vec{x}, \vec{y})$, the mixed partial derivative is averaged over all n samples for a comparative evaluation.

Classifier 8 computes the mixed partial derivative of a surrogate multilayer perceptron neural network symbolically, following Equation 12. It models the mixed partial derivative of the unknown function by creating a second surrogate multilayer perceptron neural network with the same architecture. Instead of training the second surrogate multilayer perception neural network, the weights from the surrogate of the unknown function are transferred over. The mixed partial derivative of the surrogate, $\hat{f}(x_1, y_1)$, is computed using the second surrogate multilayer perception neural network at all samples (x_1, y_1) in the test dataset. For a test dataset comprising n tuples of $(\vec{x}, \vec{y})$, the mixed partial derivative is averaged over all n samples for a comparative evaluation.

The activation function of the surrogate multilayer perceptron neural network of the unknown function is the softplus activation function, of the form seen in Equation 13. Therefore, the second surrogate multilayer perception neural network that computes the mixed partial derivative also uses the softplus activation function and its derivatives. The first and second derivatives of the softplus activation function are shown in Equations 14 and 15 respectively.

$$\sigma(x) = \log(\exp(x) + 1) \tag{13}$$

$$\sigma'_x = \frac{\partial \sigma(x)}{\partial x} = \frac{\exp(x)}{\exp(x) + 1} \tag{14}$$

$$\sigma'' = \frac{\partial^2 \sigma(x)}{\partial x^2} = \frac{\exp(x)}{(\exp(x) + 1)^2} \tag{15}$$

All code was implemented in `python`. The code to train the surrogates of the unknown functions, and to implement the eight classifiers, is available at `https://github.com/zykhoo/AdditiveSeparabilityTest.git`.

Evaluation Metrics Each classifier computes an average mixed partial derivative for each of the 3744 unknown functions. A threshold value is set, and if the average mixed partial derivative of an unknown function falls below or is equal to the threshold, the classifier classifies that unknown function as additively separable. Otherwise, the classifier classifies the unknown function as non-additively separable. This is a binary classification problem.

The metric used to compare the eight methods is the accuracy of the classification, using the threshold that gives no false positives as the optimal threshold. The classification accuracy looks at fractions of correctly assigned positive and negative classifications. As half of the unknown functions are additively separable, and half are non-additively separable, the set of functions is balanced. Furthermore, it is equally important to identify both additively separable and non-additively separable unknown functions. Therefore the classification accuracy is used as a metric. The classification accuracy is computed as the count of true positives and negatives, divided by the count of all classifications. A higher accuracy is better.

4.2   Experimental Results

We report the accuracy for the eight classifiers and their respective thresholds in Tables 2. Additionally, we report the time taken for each classifier to evaluate a surrogate in Table 3

Classifier	1	2	3	4	5	6	7	8
Accuracy	0.8654	0.7647	0.7260	0.6571	0.6343	0.6343	0.6343	0.6343
Threshold	0.0109	0.0030	0.0053	0.0018	0.0050	0.0050	0.0050	0.0050

Table 2. Accuracy and optimal threshold for the eight classifiers.

From Table 2 we observe that generally, all models have high classification accuracy and can be used to classify additively and non-additively separable functions. Classifier 1 performs the best, with an accuracy of 0.8654 at a threshold of 0.0109. Classifiers 2, 3 and 4 also perform well, with accuracies ranging between 0.7647 to 0.6571. Lastly. Classifiers 5, 6, 7 and 8 have an accuracy of 0.6343. It can also be observed from Table 2 that Classifiers 5, 6, 7 and 8 have the same accuracy and thresholds, as they all compute the mixed partial derivative of the neural network through automatic or symbolic differentiation. These mixed partial derivatives are instantaneous or at the limit, and computed at the same samples in the test dataset and, therefore have the same accuracy.

Classifiers 1 through 4 outperform Classifiers 5 through 8. Equation 1 implies that for an additively separable function $f(\vec{x}, \vec{y}) = g(\vec{x}) + h(\vec{y})$, Equations 16 and 17 hold, as the functions $g(\vec{x})$ and $h(\vec{y})$ are independent of $\vec{y}$ and $\vec{x}$ respectively. Therefore, Equations 16 and 17 should also hold even when the change in y_n or x_m is large, or when h and k is large. Classifiers 1 through 4, when computing the mixed partial derivative of a surrogate via finite difference, check that Equations 16 and 17 hold even for large changes in y_n and x_m, therefore outperform Classifiers 5 through 8 that only check to ensure that Equations 16 and 17 hold at the limit.

$$\frac{\partial g(\vec{x})}{\partial y_n} = 0 \tag{16}$$

$$\frac{\partial h(\vec{y})}{\partial x_m} = 0 \tag{17}$$

We make two notes about the observation above. Firstly, Classifiers 5 through 8 compute an instantaneous derivative whose magnitude may depend on the analytical form of the function. A non-additively separable function may have a small instantaneous derivative, and therefore be mistaken as additively separable when using Classifiers 5 through 8. For example, $f(\vec{x}, \vec{y}) = xy$ can have a small instantaneous derivative with respect to x of y and may potentially be classified as additively separable. Classifiers 1 to 4, by computing the partial derivative via finite difference instead of instantaneous derivative, compute the change in $f(\vec{x}, \vec{y})$ and can check if $f(\vec{x}, \vec{y})$ changes greatly when y increases to identify that

it is non-additively separable. Secondly, Classifiers 1 and 2 outperform Classifiers 3 and 4 because they compute the finite difference over larger changes in x_n and y_m. The former takes the distance between any two samples in the test data, while the latter takes the distance between any sample and the median of the test data.

It is for this same reason that Classifiers 1 and 3 outperform Classifiers 2 and 4. The latter normalise Classifiers 1 and 3 by the magnitude of x_n and y_m. The normalisation obscures the effect of computing the finite differences.

Lastly, from Table 3, we observe that generally, Classifiers 5, 6 and 8 are the most time efficient. Classifiers 1 and 2 are time-consuming because they compute n^2 computations. Classifier 7 is also time-consuming because it computes the Hessian of the neural network at each sample in the test dataset. This involves not just computing the mixed partial derivative of the multilayer perceptron neural network, but all other second-order derivatives as well.

Classifier	1	2	3	4	5	6	7	8
Time	48.3195	52.1720	0.0034	0.0032	0.0025	0.0025	136.4385	0.0029

Table 3. Time taken in seconds for the eight classifiers to evaluate a surrogate over a test dataset.

5   Conclusion

We presented and comparatively and empirically evaluated the performance of eight classifiers for additive separability, to be used to compute the mixed partial derivatives of surrogate functions. Classifier 1 is the most effective, followed by Classifier 2 and Classifier 3. Classifiers 5, 6 and 8 are the most efficient, followed by Classifiers 3 and 4. Classifier 3 is the test of choice given a time constraint.

The surrogate of a function can be tested for additive separability using the methods introduced in this paper. The detection that the surrogate is additively separable can be leveraged to improve further learning. We are now working on the embedding of information regarding additive separability into a multilayer perceptron neural network surrogate that has been detected to be additively separable.

Acknowledgements

References

1. Bellenot, S.F.: Additively separable functions of the form f(x,y)=f(x)+g(y), https://www.math.fsu.edu/~bellenot/class/s05/cal3/proj/project.pdf
2. Chen, R.T.Q., Rubanova, Y., Bettencourt, J., Duvenaud, D.K.: Neural ordinary differential equations. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 31. Curran Associates, Inc. (2018)
3. Diewert, W., Parkan, C.: Tests for the consistency of consumer data. Journal of Econometrics **30**(1), 127–147 (1985)
4. Fleissig, A.R., Whitney, G.A.: Testing additive separability. Economics Letters **96**(2), 215–220 (2007)
5. Gorman, W.M.: Conditions for additive separability. Econometrica **36**(3/4), 605–609 (1968), http://www.jstor.org/stable/1909527
6. Greydanus, S., Dzamba, M., Yosinski, J.: Hamiltonian neural networks. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019)
7. Gruver, N., Finzi, M.A., Stanton, S.D., Wilson, A.G.: Deconstructing the inductive biases of hamiltonian neural networks. In: International Conference on Learning Representations (2022)
8. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer Series in Statistics, Springer New York Inc., New York, NY, USA, second edn. (2008), https://web.stanford.edu/ hastie/Papers/ESLII.pdf
9. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**(5), 359–366 (Jan 1989)
10. Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., Yang, L.: Physics-informed machine learning. Nature Reviews Physics **3**(6), 422–440 (Jun 2021)
11. Khoo, Z.Y., Zhang, D., Bressan, S.: What's next? predicting hamiltonian dynamics from discrete observations of a vector field. In: Database and Expert Systems Applications: 33rd International Conference, DEXA 2022, Vienna, Austria, August 22–24, 2022, Proceedings, Part II. p. 297–302. Springer-Verlag, Berlin, Heidelberg (2022)
12. Leontief, W.: A note on the interrelation of subsets of independent variables of a continuous function with continuous first derivatives. Bulletin of the American Mathematical Society **53**(4), 343 – 350 (1947)
13. Lotka, A.J.: Elements of physical biology. Nature **116**, 461–461 (1925)
14. Lu, Y., Lin, S., Chen, G., Pan, J.: ModLaNets: Learning generalisable dynamics via modularity and physical inductive bias. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvari, C., Niu, G., Sabato, S. (eds.) Proceedings of the 39th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 162, pp. 14384–14397. PMLR (17–23 Jul 2022)
15. McDonald, T., Álvarez, M.: Compositional modeling of nonlinear dynamical systems with ode-based random features. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems. vol. 34, pp. 13809–13819. Curran Associates, Inc. (2021)
16. Meyer, K.R., Hall, G.R.: Hamiltonian Differential Equations and the N-Body Problem, pp. 1–32. Springer New York, New York, NY (1992)

17. Olver, P.: Introduction to Partial Differential Equations. Undergraduate Texts in Mathematics, Springer International Publishing (2013), `https://books.google.com.sg/books?id=aQ8JAgAAQBAJ`
18. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)
19. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc. (2019), `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-li`
20. Polisson, M.: A lattice test for additive separability. IFS Working Papers W18/08, Institute for Fiscal Studies (Mar 2018)
21. Polisson, M., Quah, J.K.H., Renou, L.: Revealed preferences over risk and uncertainty. American Economic Review **110**(6), 1782–1820 (June 2020)
22. Prechelt, L.: Early stopping - but when? In: Orr, G.B., Müller, K.R. (eds.) Neural Networks: Tricks of the Trade. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
23. Shi, R., Mo, Z., Di, X.: Physics-informed deep learning for traffic state estimation: A hybrid paradigm informed by second-order traffic models. Proceedings of the AAAI Conference on Artificial Intelligence **35**(1), 540–547 (May 2021)
24. Udrescu, S.M., Tegmark, M.: AI Feynman: A physics-inspired method for symbolic regression. Science Advances **6**(16) (2020)
25. Varian, H.R.: Non-parametric tests of consumer behaviour. The Review of Economic Studies **50**(1), 99–110 (1983), `http://www.jstor.org/stable/2296957`
26. Varian, H.R.: Microeconomic Analysis. Norton, New York, third edn. (1992)
27. Zhang, G., Yu, Z., Jin, D., Li, Y.: Physics-infused machine learning for crowd simulation. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. p. 2439–2449. KDD '22, Association for Computing Machinery, New York, NY, USA (2022)