applied sciences

Article

Explanations for Neural Networks by Neural Networks

Sascha Marton *, Stefan Lüdtke and Christian Bartelt

Institute for Enterprise Systems, University of Mannheim, 68131 Mannheim, Germany; luedtke@es.uni-mannheim.de (S.L.); bartelt@es.uni-mannheim.de (C.B.)
* Correspondence: marton@es.uni-mannheim.de

Abstract: Understanding the function learned by a neural network is crucial in many domains, e.g., to detect a model's adaption to concept drift in online learning. Existing global surrogate model approaches generate explanations by maximizing the fidelity between the neural network and a surrogate model on a sample-basis, which can be very time-consuming. Therefore, these approaches are not applicable in scenarios where timely or frequent explanations are required. In this paper, we introduce a real-time approach for generating a symbolic representation of the function learned by a neural network. Our idea is to generate explanations via another neural network (called the *Interpretation Network*, or $\mathcal{I}$-Net), which maps network parameters to a symbolic representation of the network function. We show that the training of an $\mathcal{I}$-Net for a family of functions can be performed up-front and subsequent generation of an explanation only requires querying the $\mathcal{I}$-Net once, which is computationally very efficient and does not require training data. We empirically evaluate our approach for the case of low-order polynomials as explanations, and show that it achieves competitive results for various data and function complexities. To the best of our knowledge, this is the first approach that attempts to *learn* mapping from neural networks to symbolic representations.

Keywords: Explainable AI (XAI); interpretability; explainability; neural networks; machine learning; symbolic representations

Citation: Marton, S.; Lüdtke, S.; Bartelt, C. Explanations for Neural Networks by Neural Networks. *Appl. Sci.* **2022**, *12*, 980. https://doi.org/10.3390/app12030980

Academic Editors: María Paz Sesmero Lorente, Plamen Angelov and Jose Antonio Iglesias Martinez

Received: 8 December 2021
Accepted: 15 January 2022
Published: 18 January 2022

1. Introduction

The ability of artificial neural networks to act as general function approximators has led to impressive results in many application areas. However, the price for this universal applicability is the limited interpretability of the trained model. Overcoming this limitation is a subject of active research in the machine learning community [1]. Popular approaches for explaining the results of neural networks such as LIME [2], SHAP [3], or LRP [4] focus on the impact of different attributes on the predictions of the model for certain examples. While this provides a partial explanation for individual examples, it does not shed a light on the complete network function. Especially when dealing with streaming data, uncovering the network function is very important, e.g., for detecting the adjustment of a model to concept drift or for the identification of catastrophic forgetting.

While there are existing approaches for constructing a compact representation of the network function (such as symbolic metamodeling [5] and symbolic regression [6], for instance), they generate their explanations on a sample-basis. Generating explanations through maximizing the fidelity to the neural network on a sample-basis means that the optimization process for finding a suitable explanation must be performed independently for each model we want to interpret. Since this optimization process is usually very time-consuming, it precludes the application of this method in scenarios where timely explanations are required. Furthermore, they require access to the training data, or at least knowledge of its distribution [7].

In this paper, we present a novel approach that enables the real-time, post-hoc extraction of a symbolic representation of the network function from an already trained neural network λ solely based on its parameters (i.e., weights and biases). The fundamental idea underlying our approach is to use a neural network (called *Interpretation Network*, or $\mathcal{I}$-Net) which maps the parameters of the network λ to a corresponding symbolic representation (e.g., a low-order polynomial). Accordingly, we transfer the task of interpreting neural networks to a machine learning problem, which we solve using neural networks themselves. The $\mathcal{I}$-Net can be trained up-front for a family of functions, so that generating a symbolic expression for a given network λ only requires querying the $\mathcal{I}$-Net once. This way, our approach is applicable when real-time interpretations are required (e.g., in an online learning scenario) or when the training data are either not available anymore or cannot be shared (e.g., personal data). We demonstrate our approach on the example of low-order polynomials, which can approximate a wide range of functions and can efficiently be learned with neural networks, as shown by Andoni et al. [8]. Using low-order polynomials as explanations allows reading of the influence of variables (and interactions of variables) directly from the corresponding coefficients. This makes polynomials interpretable comprehensible functions, allowing interpretability on a modular level. Even if we focus on polynomials in this paper, the approach is general by design and can be applied to arbitrary functions with symbolic representations, e.g., Boolean functions or decision trees.

This paper has four contributions:

1. We introduce the general framework of *Interpretation Networks* ($\mathcal{I}$-Nets) as means of learning the mapping from a neural network to a symbolic representation of the network function;
2. We show how $\mathcal{I}$-Nets can be trained up-front, without requiring access to training data or querying the target network (Section 2);
3. We propose a specific instance of this framework, where the symbolic representations are low-order polynomials (Section 3). Therefore, this instance allows explanations for models where sparse polynomials offer a reasonable explanation;
4. We empirically evaluate our approach against symbolic regression [6], showing that it achieves competitive results without time-consuming optimizations for each specific generated explanation (Section 4).

2. The $\mathcal{I}$-Net Approach

2.1. Explanations for Neural Networks

We start by formalizing the task of generating symbolic representations (explanations) from black-box models, i.e., neural networks. A neural network represents a function $\lambda : X \to Y$. Our goal is to obtain a function $g : X \to Y$, such that g approximates λ, i.e., $\forall \mathbf{x} \in X : \lambda(\mathbf{x}) \approx g(\mathbf{x})$. More specifically, we are interested in functions g that have a simple, *interpretable* representation. There is no universal agreement about what constitutes an interpretable function [9]. A usual requirement is that g should have substantially fewer parameters and interdependencies than λ [2]. In Section 3, we propose to use sparse polynomials as function family of g.

For this paper, it is useful to explicitly distinguish between the functions λ and g, and their *representations* $\theta_\lambda \in \Theta_\lambda$ and $\theta_g \in \Theta_g$. We can think of θ_λ and θ_g as parameters of λ and g, respectively. For example, when λ is a neural network, θ_λ is the vector of all weights and biases of λ, in some systematic order that allows reconstructing the function λ from θ_λ. Similarly, θ_g is the parameter vector for the symbolic model, e.g., coefficients of sparse polynomials, as we will introduce in Section 3.

The process of generating explanations can be formalized as a function $\mathcal{I} : \Theta_\lambda \to \Theta_g$ that maps representations of λ to representations of g. We are interested in functions $\mathcal{I}$ with a high fidelity to the neural network, i.e., where $\forall \mathbf{x} : \lambda(\mathbf{x}) \approx g(\mathbf{x})$. Existing approaches for generating symbolic explanations implement $\mathcal{I}$ via a sample-based procedure. They generate a set of samples $\{\mathbf{x}_i, \lambda(\mathbf{x}_i)\}_{i=1}^{M}$ from λ (where X usually constitutes the data used for the training of λ), and then fit a function g to those samples. This procedure must be repeated for each $\theta_\lambda \in \Theta_\lambda$ independently.

Symbolic regression [6] uses genetic programming to find a symbolic expression that maximizes the fidelity of g based on a set of data points. Accordingly, for interpretability purposes, we can apply the algorithm to the predictions of λ, assuming we have query access. Therefore, the functional form of the expressions has to be selected in terms of the allowed operations prior to application. In contrast, symbolic metamodeling [5] does not require predefining the functional form of the expressions. Instead, it uses Meijer G-functions [10] and their parameterization, which allows uncovering infinitely many functional forms (for instance arithmetic, polynomial, algebraic, and analytic expressions) based on a fixed-dimensional parameter space [5]. Furthermore, the corresponding metamodels can be optimized using gradient descent.

While these approaches can lead to concise and accurate explanations in g, they can be computationally very expensive due to the large number of required samples and difficult, time-consuming optimization of the parameters of g. Furthermore, they require the network function λ to start the optimization process which precludes real-time explanations. Specifically, in an online learning situation, it can be necessary to generate explanations in real time, making existing methods become infeasible.

2.2. *Explanations for Neural Networks by Neural Networks*

To overcome this problem, we propose to implement $\mathcal{I}$ as a neural network. This concept is visualized in Figure 1. Thus, we can train $\mathcal{I}$ up-front, so that generating an explanation g only requires querying the $\mathcal{I}$-Net once, which is possible in (close to) real-time.

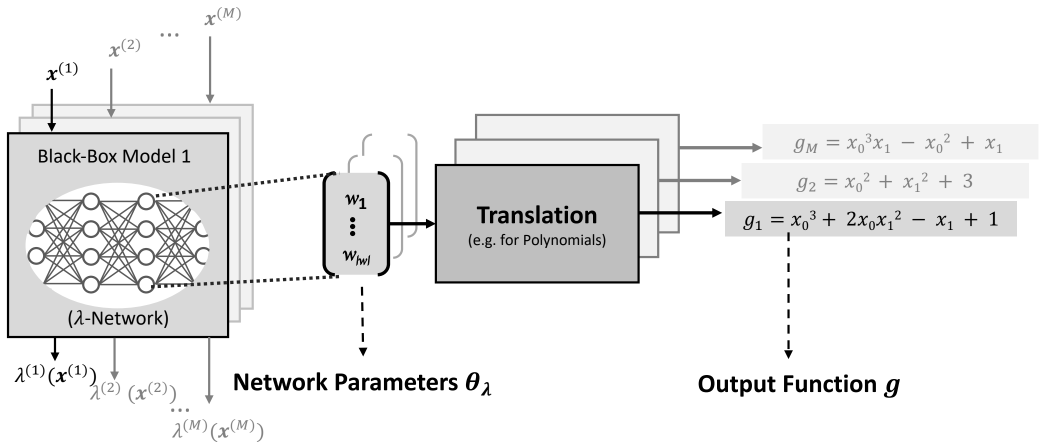

Figure 1. Overview of the $\mathcal{I}$-Net translation approach. The neural network parameters as input are translated into a symbolic representation of a mathematical function.

As described before, our goal is to obtain an $\mathcal{I}$-Net that computes θ_g with $\forall \mathbf{x} : \lambda(\mathbf{x}) \approx g(\mathbf{x})$. More precisely, we can use a distance measure, such as the mean absolute error (MAE) over a set of sample points $\{\mathbf{x}_i\}_{i=1}^M$ to quantify the agreement (the fidelity) between λ and g:

$$\mathrm{MAE}(\theta_\lambda, \theta_g) = \frac{1}{M} \sum_{i=1}^{M} |\lambda(\mathbf{x}^{(i)}) - g(\mathbf{x}^{(i)})|$$

Given a set of network parameters $\Theta_\lambda = \{\theta_\lambda^{(i)}\}_{i=1}^N$, the loss function of the $\mathcal{I}$-Net can be computed as

$$\mathcal{L}_\mathcal{I} = \frac{1}{|\Theta_\lambda|} \sum_{\theta_\lambda \in \Theta_\lambda} \mathrm{MAE}(\theta_\lambda, \mathcal{I}(\theta_\lambda)).$$

Note that training an $\mathcal{I}$-Net does not require any supervision of the desired network output θ_g, but instead we only measure the fidelity between network input λ and network output g. Since both the polynomial function g and the neural network function λ, which are required in the loss, are differentiable, we can ensure an efficient computation.

Here, we focus on generating explanations for fixed function families. That is, each $\mathcal{I}$-Net is learned based on networks λ, which were trained using functions belonging to the same family, e.g., polynomials of fixed dimensionality. For training such an $\mathcal{I}$-Net, we require a set Θ_λ, where each $\theta_\lambda \in \Theta_\lambda$ represents a neural network trained on that function family.

Fortunately, it is not necessary to use real data for training. Instead, it is sufficient to sample a set of functions from the family of interest and train networks λ to approximate these functions. Interestingly, as we will see later, $\mathcal{I}$-Nets trained on such samples can still generate accurate explanations for networks that were *not* trained explicitly on the same function family (see Section 4.2.3).

More specifically, training data for the $\mathcal{I}$-Net can be created in three steps:

1. Sample a set of parameter vectors θ_{g^*};
2. For each θ_{g^*}, generate a set of input-output pairs $\{(x, g^*(x))\}_{i=1}^{M}$;
3. For each θ_{g^*}, train a network λ on $\{(x, g^*(x))\}_{i=1}^{M}$

Here, g^* represents the *target function* of a network λ. We explicitly distinguish between g^* and g, since our goal is finding a function g that approximates what the neural network has actually learned (i.e., the network function) and not the target function g^* (i.e., what the neural network should learn). A specific example of this procedure for the case of sparse polynomials is shown in Algorithm 1.

Note that sampling is only necessary during generation of Θ_λ and in the loss function of the $\mathcal{I}$-Net, but not when generating explanations using the $\mathcal{I}$-Net. This has a number of advantages compared to conventional, sample-based algorithms. First, we can generate explanations in (close to) real-time, because generating explanations only requires querying the $\mathcal{I}$-Net once instead of performing a costly optimization, and the $\mathcal{I}$-Net can be trained up-front, even without knowing the network function λ.

Secondly, the $\mathcal{I}$-Net can utilize the complete function description θ_λ to generate an explanation instead of relying on an incomplete description based on samples.

Algorithm 1 Generate training data for $\mathcal{I}$-Net of sparse polynomials (here, $\mathcal{U}(\mathbf{0},\mathbf{1})$ denotes independent samples from a continuous, uniform distribution over the interval $[0,1]$)

```
1:  function GENERATE($M, N, s$)
2:      for $i = 1, \ldots, N$ do
3:          $\theta_{g^*}[1:s] \sim \mathcal{U}(-\mathbf{1},\mathbf{1})$                ▷ Sample $s$ coefficients of target polynomial
4:          $\theta_{g^*}[s+1:2s] \sim \mathrm{Urn}([0,1,\ldots,\binom{n+d}{d}])$        ▷ Sample $s$ monomial indices
5:          for $j = 1, \ldots, M$ do                   ▷ Generate samples from the polynomial
6:              $\mathbf{x}^{(j)} \sim \mathcal{U}(\mathbf{0},\mathbf{1})$
7:              $y^{(j)} = g^*(\mathbf{x}^{(j)})$
8:          Fit neural network $\lambda$ to the training set $\{\mathbf{x}^{(j)}, y^{(j)}\}_{j=1}^{M}$
9:          Add $\lambda$ to training set $\Lambda$
10:     return $\Lambda$
```

3. $\mathcal{I}$-Nets for Sparse Polynomials

When choosing a function family for the generated explanations, a trade-off between performance and interpretability has to be considered. While functions that are too simple might not be able to approximate the network function well, a function that is too complex might not improve the interpretability. Here, we introduce a specific instance of $\mathcal{I}$-Nets, where g is a sparse polynomial. Within this approach, the user can adjust the level of complexity themself by defining the maximum degree d of the polynomial and the number of monomials (i.e., the sparsity) s. As usual, we consider a polynomial to be a multivariate function

$$g(x_1, \ldots, x_n) = \sum_{i=1}^{s} c^{(i)} \prod_{j=1}^{n} x_j^{d^{(i,j)}},$$

where $d^{(i,j)}$ is the exponent of variable j in monomial i, and s is the number of monomials.

With an increasing number of variables and a higher degree of the polynomial, the number of possible monomial terms expands rapidly and is calculated as $\binom{n+d}{d}$, where d is the degree and n the number of variables. Therefore, a cubic, bivariate polynomial has $\binom{2+3}{3} = 10$ possible coefficients, each belonging to one of the following monomials:

0. $x_0^3 x_1^0 = x_0^3$
1. $x_0^2 x_1^1 = x_0^2 x_1$
2. $x_0^2 x_1^0 = x_0^2$
3. $x_0^1 x_1^2 = x_0 x_1^2$
4. $x_0^1 x_1^1 = x_0 x_1$
5. $x_0^1 x_1^0 = x_0$
6. $x_0^0 x_1^3 = x_1^3$
7. $x_0^0 x_1^2 = x_1^2$
8. $x_0^0 x_1^1 = x_1$
9. $x_0^0 x_1^0 = 1$

Accordingly, it is straightforward to represent the bivariate, cubic polynomial $g(x_0, x_1) = 3x_0^3 + x_0^1 - 5x_0 x_1^2 - 3x_1 - 4$ as a vector θ'_g of the coefficients, according to the encoding in Figure 2a. The order of the elements in θ'_g is sorted by the enumeration of monomials shown above. It is easy to see that such a systematic evaluation of monomials exists for all n and d.

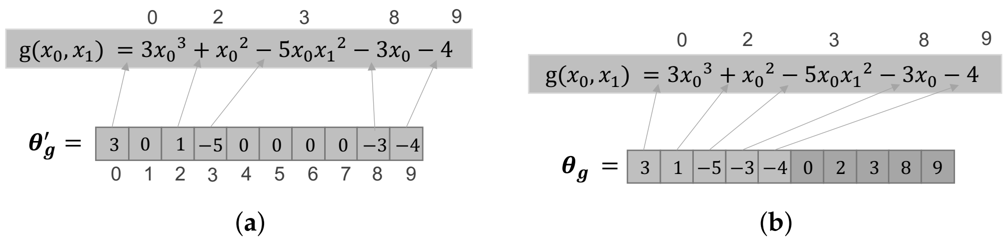
0 2 3 8 9
$g(x_0, x_1) = 3x_0{}^3 + x_0{}^2 - 5x_0x_1{}^2 - 3x_0 - 4$
$\theta'_g =$ 3 0 1 −5 0 0 0 0 −3 −4
0 1 2 3 4 5 6 7 8 9
(a)
0 2 3 8 9
$g(x_0, x_1) = 3x_0{}^3 + x_0{}^2 - 5x_0x_1{}^2 - 3x_0 - 4$
$\theta_g =$ 3 1 −5 −3 −4 0 2 3 8 9
(b)

Figure 2. Polynomial representations: Comparison of dense and sparse polynomial representations for an exemplary polynomial. (**a**) Dense polynomial representation: a value at position i represents the coefficient of the i-th monomial in the systematic enumeration. (**b**) Sparse polynomial representation: the polynomial is represented by pairs of the coefficient and index of the monomial in the systematic enumeration.

In this paper, we are concerned with *sparse* polynomials, where only a small number of monomials have nonzero coefficients. The reason for focusing on sparse polynomials is twofold. First, we are concerned with *explanations* that can be inspected and interpreted by humans. Although there is no universal agreement about what constitutes an interpretable representation [9], we follow Ribeiro et al. [2] who use model complexity as a proxy for interpretability. Thus, we argue that the number of monomials has to be relatively small to serve as a useful and human-understandable representation. Secondly, constraining the number of monomials is a form of regularization, preventing overfitting.

For sparse polynomials, the representation in Figure 2a is very inefficient, especially for an increasing number of variables and degree. For instance, assuming we want to represent a cubic polynomial with 10 variables, we would need a vector θ'_g of length $\binom{10+3}{3} = 286$, even if just a small number of monomials has non-zero coefficients. Additionally, it is difficult for a neural network to predict zero coefficients when using this representation, since linear outputs are not tailored towards predicting a value of 0. This would eventually result in dense polynomials with many small coefficients, which precludes interpretability.

Thus, to efficiently represent and predict sparse polynomials, we define an alternative representation, comprising the coefficient values along with the corresponding monomial indices. This way, the function $g(x_0, x_1) = 3x_0^3 + x_0^1 - 5x_0 x_1^2 - 3x_1 - 4$ can be represented as the vector θ_g, shown in Figure 2b. Within θ_g, the first s entries represent the coefficients and the second s entries represent the corresponding monomial indices in the systematic enumeration of monomials shown previously. The coefficient at position i for $i = \{0, \dots, s\}$ belongs to the monomial index at position $s + i$. Here, s defines the number of monomials with non-zero coefficients, which we also call the *sparsity* of the polynomial in the following.

The architecture of the output layer of an $\mathcal{I}$-Net for sparse polynomials is shown in Figure 3. For the coefficient outputs, we use linear output activations. Each index output is represented via $\binom{n+d}{n}$ neurons with softmax activation (i.e., it can be seen as a separate classification task for each index). This way, indices can be constrained to be integers in the range $[0, \binom{n+d}{d})]$, which would not be possible directly when using linear activations.

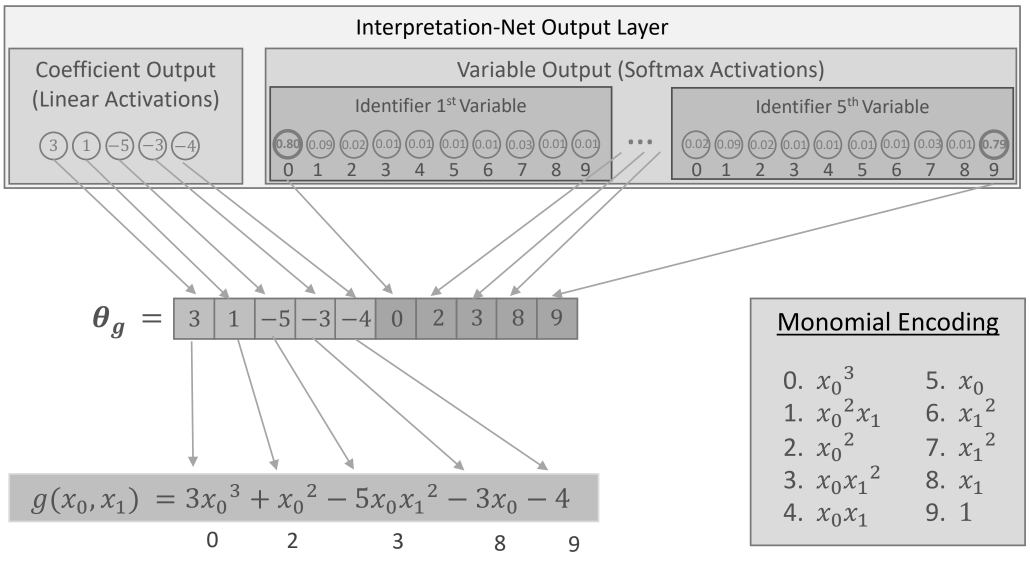

Figure 3. $\mathcal{I}$-Net output layer. The sparse polynomial representation is predicted by the $\mathcal{I}$-Net using different output layers and activations.

We use the sparse polynomial representation as explanations, i.e., as outputs of $\mathcal{I}$-Nets. Note that for a given $\mathcal{I}$-Net, the sparsity parameter s (which defines the dimensionality of θ_λ) is fixed. For an efficient $\mathcal{I}$-Net training and evaluation, we need datasets where polynomial functions can act as reasonable explanations, i.e., datasets where we would choose the function family of polynomials for interpretability purposes in the first place. Therefore, Algorithm 1 shows how reasonable training data (i.e., parameters θ_λ) for an $\mathcal{I}$-Net are generated. First, we sample parameter vectors θ_{g^*} of sparse polynomials as target functions with fixed sparsity s. Thereby, the coefficient values are sampled independently of $\mathcal{U}(-\mathbf{1}, \mathbf{1})$. The position indices in θ_{g^*} are sampled from an urn without replacement with values $[0, 1, \dots, \binom{n+d}{d})]$. Then, for each θ_{g^*}, a number of sample points $\mathbf{x}$ are drawn from $\mathcal{U}(\mathbf{0}, \mathbf{1})$, and the target function values $y = g^*(\mathbf{x})$ are calculated to generate training data. Accordingly, we have N datasets with M samples of dimensionality n comprising random datapoints with function values for a randomly generated polynomial of degree d and sparsity s. Next, a network λ is trained on the sampled data, so that $\forall \mathbf{x} \in X : g^*(\mathbf{x}) \approx \lambda(\mathbf{x})$. The set of parameters of these networks λ can then be used for training an $\mathcal{I}$-Net. The resulting $\mathcal{I}$-Net can generate explanations for sparse polynomials with fixed sparsity s.

4. Evaluation

The goal of the evaluation is to show that $\mathcal{I}$-Nets can produce real-time explanations while still achieving competitive performance compared to symbolic regression using genetic programming [6]. Specifically, within our experiments we investigated the following research questions:

1. What is the runtime of the different methods for generating explanations with respect to the dimensionality of the function (Section 4.2.2)?

2. What is the error of the generated explanations in terms of MAE between original network λ and explanation g (Section 4.2.2)?
3. How does the error of the $\mathcal{I}$-Net explanations change with respect to noise in the training data of λ (Section 4.2.3)?

We did not include symbolic metamodeling [5] in the evaluation, which has only been successfully used for classification and has not achieved reasonable results for regression, as investigated here (We used the official implementation available under: https://github.com/ahmedmalaa/Symbolic-Metamodeling (accessed on 1 December 2021)). Furthermore, we found that the generated explanations are usually very complex and cannot be restricted in complexity. Thus, we argue that they do not serve as human-understandable representations, especially with an increasing complexity of the dataset or an increasing number of variables.

4.1. Experimental Setup

For the evaluation, we trained a set of 100 neural networks λ for each parametrization on randomly generated functions according to Algorithm 1 with the specifications shown in Table 1. This way, the evaluation was performed on datasets where polynomials can act as reasonable explanations with a sufficiently high fidelity. The set of randomly generated polynomials and data points used for the evaluation is distinct from the training set of $\mathcal{I}$-Net, i.e., they are unseeded during the training, thus ensuring a fair comparison. The dataset to train the λ-Nets was split into disjointed train, valid, and test datasets where 2813 samples were used for the training, 937 samples for the validation, and 1250 samples for the performance evaluation.

Table 1. Dataset specifications for λ-Net training.

Parameter	**Dataset Specification**			
	Evaluation			**Visualization**
number_of_variables (n)	5	10	15	1
degree (d)	3	3	3	5
sparsity (s)	5	10	15	3
Noise [a]	$\{0, 0.1, 0.2, 0.3\}$	$\{0, 0.1, 0.2, 0.3\}$	$\{0, 0.1, 0.2, 0.3\}$	0
λ-Net Data Set Size (N)	5000	5000	5000	5000

[a] The noise was injected for each dataset separately by adding a value drawn from $\mathcal{N}(0, \max(f(\mathbf{x_1}), \ldots, f(\mathbf{x_N})) - \min(f(\mathbf{x_1}), \ldots, f(\mathbf{x_N})))$ multiplied by the specified noise level.

The parameters used for training the λ-Nets are summarized in Table 2. For the λ-Nets we selected the parameters based on the findings of Andoni et al. [8], which prove that neural networks with a single hidden layer are able to learn sparse, cubic polynomials efficiently using SGD with $5 \times s$ neurons. We decided to use Adam instead of SGD due to its faster convergence. As activation function, we selected ReLU, which is defined as $f(x) = \max(0, x)$ due to its computational efficiency and good performance, especially in supervised settings [11]. The parameters for the $\mathcal{I}$-Net training were selected based on coarse to fine tuning to optimize the performance and are summarized in Table 2. Furthermore, the degree and sparsity for the $\mathcal{I}$-Net output layer are set according to the dataset specifications in Table 1.

For symbolic regression, the parameters were tuned similar to the $\mathcal{I}$-Net using coarse to fine grid search. We used the symbolic regression implementation of `gplearn` (Available at https://gplearn.readthedocs.io/en/stable/reference.html#symbolic-regressor (accessed on 1 December 2021)). More details on the hyperparameters and settings of the symbolic regression baseline are described in Appendix A.

Table 2. Parameters used for the λ-Net and $\mathcal{I}$-Net training.

Parameter	**λ-Hyperparameters**	**$\mathcal{I}$-Hyperparameters**
Hidden Layer Neurons [a]	$[5 \times s]$	$[4096, 2048, 1024, 512]$
Hidden Layer Activation	ReLU	ReLU
Batch Size	64	256
Optimizer	Adam	Adam
Learning Rate	0.001	0.0001
Loss Function	MAE	$\mathcal{L}_{\mathcal{I}\text{-}Net}$ (with $M = 500$)
Training Epochs	1,000	500
Early Stopping	Yes	Yes
Number of Training Samples	5,000	45,000

[a] In this context, s stands for the sparsity of θ_g

The λ-Nets used for learning the $\mathcal{I}$-Net were trained on randomly generated functions according to the specification in Table 1. Since the training of 45,000 neural networks for each data-set complexity as well as the application of symbolic regression on 100 neural networks is very time-consuming, all experiments were conducted without repetition and using an arbitrarily selected random seed, ensuring the reproducibility of our results. Our approach was implemented in `Python` using `TensorFlow` [12] and is available at https://doi.org/10.5281/zenodo.5865088 (accessed on 1 December 2021).

4.2. Experimental Results

4.2.1. Inspection of Explanation and Complexity

In the following section, we inspect the explanations generated by $\mathcal{I}$-Nets and compare them with symbolic regression based on their complexity. In addition to the evaluation settings from Table 1, we also consider the case of $n = 1$, to allow for a visual comparison.

In Figure 4, we can see explanations for a selected but arbitrary neural network generated by the $\mathcal{I}$-Net and symbolic regression along with the target function g^*. Additionally, the dotted blue line shows the predictions of the corresponding neural network λ trained on g^*. The corresponding expressions and the fidelity (w.r.t. λ) are shown in Table 3. We can observe that the neural network was able to learn the polynomial target function very accurately, as the functions completely overlap. The $\mathcal{I}$-Net achieved the best performance in this case (MAE = 0.0220) and was also able to approximately capture the functional form. The function generated by symbolic regression comprises just a single term and achieved an MAE of 0.0566. While the error can be considered as close to the $\mathcal{I}$-Net, symbolic regression was not able to capture the functional form in the given interval.

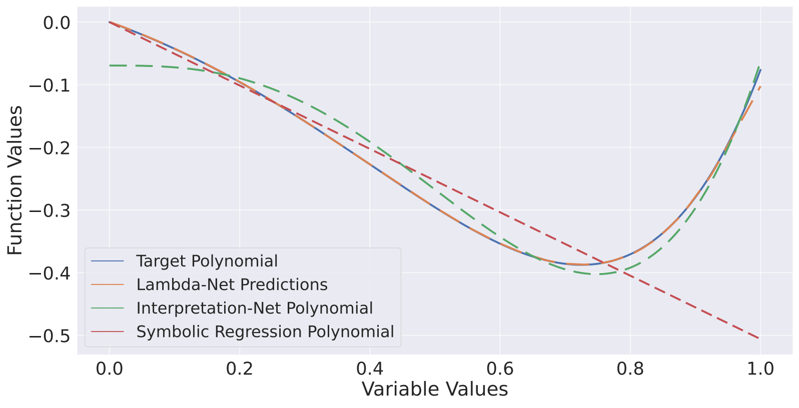

Figure 4. Exemplary visualization of the explanations. This plot visualizes the function generated by the evaluated approaches for $n = 1$ on an arbitrarily selected dataset. The corresponding mathematical expressions and their fidelity can be found in Table 3.

For the multivariate case, a visual comparison of the functions to the model's predictions is not possible. Instead, Table 3 shows the polynomials for arbitrarily selected datasets of each considered complexity. As expected, the complexity of the polynomial generated by the $\mathcal{I}$-Net matches the sparsity defined in Table 1. Accordingly, the complexity of the explanation increases with the complexity of the underlying dataset used for the training of λ. For symbolic regression, complexity of the explanations (w.r.t. number of monomials) does not increase similarly. Instead, especially for $n = 15$, the individual monomials have a higher complexity comprising four different variables, while the maximum degree for the $\mathcal{I}$-Net is three. The fact that the explanation complexity for symbolic regression does not increase substantially can be attributed to the selection of a high `parsimony_coefficient` which punishes large functions during the optimization, preventing overfitting and long runtimes (see Appendix A for details).

Table 3. Examples of generated polynomials.

Number of Variables		**$\mathcal{I}$-Net**	**Symbolic Regression**
	$n = 1$	$3.1812x^4 - 3.1752x^3 - 0.0694$ **(MAE: 0.0220)**	$-0.5060x$ **(MAE: 0.0566)**
	$n = 5$	$-0.0749x_0x_1x_4 + 0.5084x_1x_2 - 0.2185x_1x_4 - 0.432x_2x_4^2 + 0.0939x_3^2x_4$ **(MAE: 0.0068)**	$0.2180x_2^2 + 0.4360x_1x_2 - 0.6540x_4x_2$ **(MAE: 0.0373)**
Function **(Fidelity)**	$n = 10$	$0.1837x_0^2x_3 - 0.2915x_0x_3x_4 - 0.4966x_0x_5x_6 - 0.3588x_1^2 + 0.0340x_1x_3x_8 - 0.1769x_2x_3 + 0.1238x_3^2 - 0.0746x_3x_6x_7 - 0.3337x_6^2 - 0.3666x_6x_8$ **(MAE: 0.1436)**	$-1.0000x_1x_5x_9 - 0.6430x_6$ **(MAE: 0.1371)**
	$n = 15$	$0.0722x_0^3 + 0.6574x_0x_2x_8 + 0.0555x_0x_7x_{13} + 0.0764x_0x_{10} + 0.1436x_1^2x_3 - 0.0359x_2x_3 - 0.1764x_2x_9x_{13} + 0.2182x_2x_{13}x_{14} + 0.5176x_3^2x_{11} + 0.0369x_4 + 0.1653x_6x_7x_{12} - 0.0552x_6x_{12}x_{14} + 0.2252x_7x_8x_9 + 0.2166x_9x_{10}x_{14} + 0.1979x_9x_{11}x_{14}$ **(MAE: 0.1971)**	$1.0000x_3x_8 + 1.0000x_1x_5x_7x_{11} + 1.0000x_1x_5x_7x_{14}$ **(MAE: 0.1498)**

4.2.2. Runtime and Performance Evaluation

The goal of this experiment was to compare our approach with symbolic regression based on the runtime, as well as the fidelity between the predicted symbolic expression and the corresponding λ-Net.

Figure 5 shows the time required for generating an explanation for a single network. At the smallest setting ($n = 5$), symbolic regression needs approximately 160 s for the interpretation. There is no notable increase in the runtime with an increasing number of variables. This can again be explained by the `parsimony_coefficient`, which restricts the complexity resulting in functions with similar complexity even when the dataset complexity and the number of variables increases (see Section 4.2.1 for a more in-depth evaluation of the function complexity). However, if more complex functions are considered within symbolic regression, the runtime increases significantly, especially for high-dimensional learning problems. Using $\mathcal{I}$-Nets, the training process can be performed up-front, and interpretations can be generated in close to real-time (∼0.003 s), regardless of the number of variables and function complexity.

Figure 6 shows the error of the explanation in terms of MAE between function values of the λ-Net and the corresponding explanation g computed by the $\mathcal{I}$-Net.

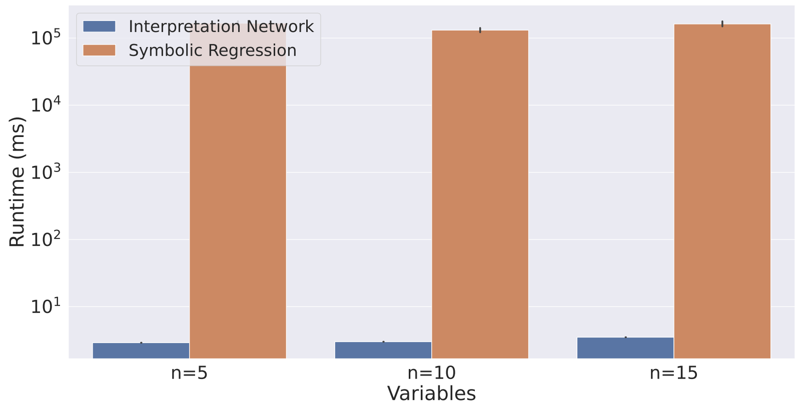

Figure 5. Runtime evaluation. The plot shows a comparison of the average time (in milliseconds) required for explaining a single neural network for symbolic regression and the $\mathcal{I}$-Net on a logarithmic scale.

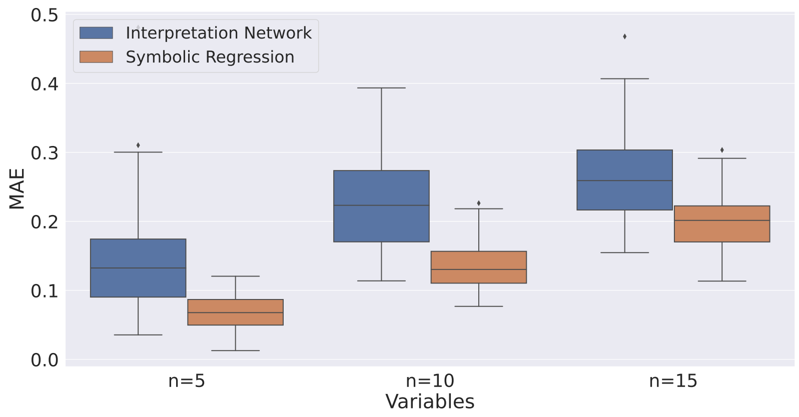

Figure 6. $\mathcal{I}$-Net performance comparison with symbolic regression. This plot shows the performance in terms of the MAE for all evaluated approaches for three different dataset complexities.

When comparing the mean errors of both approaches, we can observe that symbolic regression achieves a higher fidelity to the neural network, especially for $n = 5$ and $n = 10$. For $n = 15$, however, the error difference diminishes. Additionally, we can see that the standard deviation of the error for symbolic regression is low, while $\mathcal{I}$-Nets have a comparatively higher standard deviation. While there are cases in which the performance of $\mathcal{I}$-Nets is superior to symbolic regression, there are also networks where the $\mathcal{I}$-Net is not able to make reasonable predictions, resulting in high errors that lower the mean performance of $\mathcal{I}$-Nets. We suspect that a more sophisticated architecture, a more in-depth hyperparameter optimization, and a larger training data set with higher variety would allow increased the performance of $\mathcal{I}$-Nets, subject to further work.

Furthermore, optimization for symbolic regression was conducted on only 5000 samples, which can be considered to be very low compared to most real-world tasks. However, there was no significant improvement in performance for a larger sample size in our evaluation, which would justify the use of a larger set of samples. This can be traced back to the fact that the underlying network function is close to a polynomial function, which can already be approximated well with a small number of samples. However, this assumption may not hold in other scenarios.

4.2.3. $\mathcal{I}$-Net Performance on Noisy Data

As mentioned before, a key advantage of our approach is that the $\mathcal{I}$-Net can be trained upstream and in a one-time effort based on synthetic data. We only need to ensure that the function family used as $\mathcal{I}$-Net output is able to approximate the neural network we want to interpret sufficiently well. In this experiment, we show that an $\mathcal{I}$-Net trained on a clean, synthetic dataset can be applied to neural networks that have been trained on previously unseen datasets, even if they contain noise and do not explicitly form a polynomial function. Figure 7 shows the performance of the $\mathcal{I}$-Net for different noise levels. Here, the interpretation for the different noise levels is conducted with the same $\mathcal{I}$-Net.

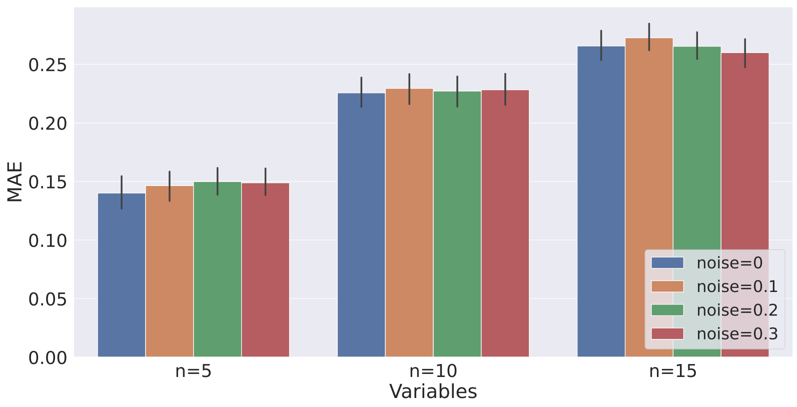

Figure 7. $\mathcal{I}$-Net performance evaluation on noisy data. The plot displays the mean absolute error between the λ-Nets and the functions predicted by the $\mathcal{I}$-Net for different levels of noise and numbers of variables.

As we can see, the performance of the $\mathcal{I}$-Net does not significantly change for different noise levels for all numbers of variables. Accordingly, $\mathcal{I}$-Nets that learned on λ-Nets that were trained on synthetic, clean data are able to make accurate predictions for previously unseen neural networks, even if they are trained on noisy data. Therefore, $\mathcal{I}$-Nets can also be applied to realistic scenarios where noise is inherently contained in the data without requiring additional information or training.

5. Related Approaches

Various methods to interpret black-box models have been proposed in the past decades. Overviews from different perspectives can be found in [1,7,9]. According to the taxonomy of Molnar [7], our approach can be classified as global, post-hoc, and model specific, with an intrinsically interpretable model (i.e., a polynomial function) as interpretation. Therefore, we focus on global interpretation methods with the goal of uncovering the decision-making process of the model based on the impact of the learned parameters (e.g., weights and biases of a neural network) on its features [7]. Global explanations therefore allow us to better understand the relationship and interactions between the features within the learned model. In contrast, local explainability methods such as LIME [13] or SHAP [3] only generate explanations for the models' prediction on a single instance. Therefore, they aim for local fidelity, which means that the explanation only accounts for this specific instance of interest and does not claim to have a high fidelity for further instances. This is fundamentally different from the goal of global interpretability, where we want to find an interpretation that has a high fidelity for the complete model and not only for a specific instance.

There is a wide range of active research in the field of global interpretability. One part of global interpretability research focuses on uncovering the impact of a feature or a set of features on the model's predictions, as in, for instance, partial dependency plots (PDP) [14,15], feature interaction [16–18], or (permutation) feature importance [19]. Another area of global interpretability is concerned with finding data points that are representative for the learned model in order to increase the interpretability, as in, for instance, prototypes and criticisms [7]. While the previously mentioned approaches are similar to $\mathcal{I}$-Nets as global interpretability methods that can be applied post-hoc, they differ significantly in the results of the interpretation method.

As it presents an intrinsically interpretable model as a result of the interpretation, the most relevant work for our paper is work on global surrogate models. Global surrogate models, according to Molnar [7], are defined as an interpretable model trained to approximate the predictions of a black-box model. Thus, interpretability is achieved by inspecting the parameters of the surrogate model. In the literature, surrogate models are considered as model-agnostic and are trained based on the prediction of the model we want to interpret [1,7]. Existing approaches usually differ only in the type of surrogate model that is chosen and the training procedure of the model. Different types of surrogate models have been explored, including mathematical functions [5,20], decision trees [21–23], or rule sets [24,25].

While the result of the interpretation (i.e., an intrinsically interpretable model) of global surrogate models matches our approach, there are also major differences. All mentioned approaches require an optimization process during interpretation, whereas our approach transforms the interpretation task into a machine learning problem that is solved using neural networks up-front. Additionally, existing approaches generate interpretations based on samples from the model to be interpreted, making them model-agnostic. In contrast, the $\mathcal{I}$-Net uses the network parameters of the λ network as the basis for generating explanations; therefore, this approach is model-specific by definition.

In summary, unlike existing approaches, $\mathcal{I}$-Nets enable real-time interpretations of previously unseen and already trained models by a representation of their network function without relying on any data.

6. Conclusions and Future Work

In this paper, we introduced a machine learning approach for the real-time extraction of mathematical functions from already trained neural networks. The presented method relies on the offline training of so-called *Interpretation Networks* ($\mathcal{I}$-Nets) for a certain family of functions. These $\mathcal{I}$-Nets enable the translation of neural network parameters (i.e., the weights and biases) to corresponding, human understandable terms in a well-defined algebraic language. Our approach increases the interpretability of neural networks post-hoc and in real-time by identifying and extracting the respective network function as an interpretable function. We empirically showed that $\mathcal{I}$-Nets can learn to generate polynomial expressions corresponding to given weights and biases of neural networks for different dataset complexities and with different noise levels. We showed that $\mathcal{I}$-Nets can achieve competitive results for the function family of sparse polynomials.

Because of their ability to generate explanations in real-time, $\mathcal{I}$-Nets are especially suited for dynamic environments such as online learning. In this context, they can, for instance, be applied to detect whether a model has adopted a concept drift in the data or to identify catastrophic forgetting by inspecting the symbolic representation of the network function. In addition, the original training data is not required, because our approach directly works on the model parameters, which implicitly contain all relevant information about the network function. Thus, explanations can be generated without exposing confidential training data or when the training data is not available. Furthermore, using the model parameters does not require proper querying. Additionally, $\mathcal{I}$-Nets allow the user to specify the complexity of the explanation based on their needs and therefore can always ensure explanations with a reasonable level of complexity.

In this paper, we focused on lower order polynomials with a moderate number of variables. Using sparse polynomials as explanations, we assume that the model we want to interpret can be represented sufficiently well using this function family. Therefore, the function family of sparse polynomials is not well-suited to explain highly non-linear models, such as, for instance, convolutional neural networks trained on image data. However, the advantage of $\mathcal{I}$-Nets is that the general approach is not limited to a specific function family (i.e., polynomials). Therefore, the transfer of $\mathcal{I}$-Nets to cover further function families is subject to further work.

Appendix A. Parameters of Baseline Model

The parameters of symbolic regression are summarized in Table A1. The value for the `parsimony_coefficient`, which regularizes the complexity during the optimization, was not included in the hyperparameter optimization, since values smaller than 0.001 frequently resulted in functions that were too complex for function value calculation and led to a termination of the optimization process without a significant increase in performance. The set of considered functions for symbolic regression was selected to include only the operations necessary to represent polynomials.

Table A1. Parameter settings for symbolic regression.

Parameter	Value
dataset size	5000
early stopping	10
max_opt_mins	60
metric	MAE
`population_size`	5000
`generations`	100
`parsimony_coefficient`	0.001
`function_set`	('add', 'sub', 'mul')
`init_depth`	$(log_2(s), log_2(s))$

Additionally, early stopping in `gplearn` was implemented based on a specific target score, but it is not possible to terminate the optimization if no improvement is achieved over a specified number of generations. We added this functionality to improve the performance of symbolic regression and ensure comparability. In addition to early stopping, we implemented a mechanism to terminate the optimization and pick the best individual after a specified timeframe (in minutes) to cap the runtime as well as the complexity.

Furthermore, `gplearn` uses syntax trees to represent the functions during optimization. Therefore, we selected the `init_depth` in a way that the maximum length of the functions in the initial population (which is the maximum number of leaf nodes in a syntax tree computed by $2^{\text{max\_depth}}$) matches the sparsity s selected for the $\mathcal{I}$-Net to ensure a fair comparison.

Appendix B. Symbols

Table A2. Abbreviation form.

Symbol	Meaning
λ	neural network function
Λ	set of neural network functions
g	(polynomial) function as output of an explanation method
g^*	(polynomial) function as target of a learning problem
θ	representation of a function
Θ	set of function representations
$\mathcal{I}$	function that maps representations of λ to representations of g
M	number of samples points
N	number of data sets/trained models
n	number of variables
d	degree
s	sparsity
$\mathcal{U}$	uniform distribution
Urn	urn distribution
$\mathcal{N}$	normal distribution

Author Contributions: Conceptualization, S.M. and C.B.; methodology, S.M.; validation, S.M., S.L. and C.B.; data curation, S.M.; writing—original draft preparation, S.M. and C.B.; writing—review and editing, S.M., S.L. and C.B.; visualization, S.M.; supervision, S.L. and C.B.; funding acquisition, C.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The code including data generation used within this paper is available at https://doi.org/10.5281/zenodo.5865088 (accessed on 1 December 2021).

Acknowledgments: This research was supported by the German Federal Ministry for Economic Affairs and Energy (BMWi).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Guidotti, R.; Monreale, A.; Ruggieri, S.; Turini, F.; Giannotti, F.; Pedreschi, D. A survey of methods for explaining black box models. *ACM Comput. Surv. (CSUR)* **2018**, *51*, 1–42. [CrossRef]
2. Ribeiro, M.T.; Singh, S.; Guestrin, C. "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, San Francisco, CA, USA, 13–17 August 2016; pp. 1135–1144.
3. Lundberg, S.M.; Lee, S.I. A unified approach to interpreting model predictions. In Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; pp. 4765–4774.
4. Montavon, G.; Binder, A.; Lapuschkin, S.; Samek, W.; Müller, K.R. Layer-wise relevance propagation: An overview. In *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 193–209.
5. Alaa, A.M.; van der Schaar, M. Demystifying Black-box Models with Symbolic Metamodels. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 11304–11314.
6. Menezes, T.; Roth, C. Symbolic regression of generative network models. *Sci. Rep.* **2014**, *4*, 6284. [CrossRef] [PubMed]
7. Molnar, C. Interpretable Machine Learning. 2020. Available online: https://christophm.github.io/interpretable-ml-book/cite.html (accessed on 1 December 2021).
8. Andoni, A.; Panigrahy, R.; Valiant, G.; Zhang, L. Learning polynomials with neural networks. In Proceedings of the International Conference on Machine Learning, Beijing, China, 21–26 June 2014; pp. 1908–1916.
9. Lipton, Z.C. The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue* **2018**, *16*, 31–57. [CrossRef]
10. Beals, R.; Szmigielski, J. Meijer G-functions: A gentle introduction. *Not. AMS* **2013**, *60*, 866–872. [CrossRef]
11. Glorot, X.; Bordes, A.; Bengio, Y. Deep sparse rectifier neural networks. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Ft. Lauderdale, FL, USA, 11–13 April 2011; pp. 315–323.
12. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online: tensorflow.org (accessed on 1 December 2021).
13. Shrikumar, A.; Greenside, P.; Kundaje, A. Learning important features through propagating activation differences. In Proceedings of the 34th International Conference on Machine Learning—JMLR.org, Sydney, Australia, 6–11 August 2017; Volume 70, pp. 3145–3153.
14. Friedman, J.H. Greedy function approximation: A gradient boosting machine. *Ann. Stat.* **2001**, *29*, 1189–1232. [CrossRef]
15. Zhao, Q.; Hastie, T. Causal interpretations of black-box models. *J. Bus. Econ. Stat.* **2021**, *39*, 272–281. [CrossRef] [PubMed]
16. Friedman, J.H.; Popescu, B.E. Predictive learning via rule ensembles. *Ann. Appl. Stat.* **2008**, *2*, 916–954. [CrossRef]
17. Hooker, G. Discovering additive structure in black box functions. In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, WA, USA, 22–25 August 2004; pp. 575–580.
18. Greenwell, B.M.; Boehmke, B.C.; McCarthy, A.J. A simple and effective model-based variable importance measure. *arXiv* **2018**, arXiv:1805.04755.
19. Fisher, A.; Rudin, C.; Dominici, F. All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously. *J. Mach. Learn. Res.* **2019**, *20*, 1–81.
20. Orzechowski, P.; La Cava, W.; Moore, J.H. Where are we now? A large benchmark study of recent symbolic regression methods. In Proceedings of the Genetic and Evolutionary Computation Conference, Kyoto, Japan, 15–19 July 2018; pp. 1183–1190.
21. Frosst, N.; Hinton, G. Distilling a neural network into a soft decision tree. *arXiv* **2017**, arXiv:1711.09784.
22. Liu, X.; Wang, X.; Matwin, S. Improving the interpretability of deep neural networks with knowledge distillation. In Proceedings of the 2018 IEEE International Conference on Data Mining Workshops (ICDMW), Singapore, 17–20 November 2018; pp. 905–912.
23. Zhang, Q.; Yang, Y.; Ma, H.; Wu, Y.N. Interpreting cnns via decision trees. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 6261–6270.
24. Zilke, J.R.; Loza Mencía, E.; Janssen, F. DeepRED—Rule Extraction from Deep Neural Networks. In *Discovery Science*; Calders, T., Ceci, M., Malerba, D., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 457–473.
25. Zhou, Z.H.; Jiang, Y.; Chen, S.F. Extracting symbolic rules from trained neural network ensembles. *AI Commun.* **2003**, *16*, 3–15.