Engineering Applications of Artificial Intelligence 99 (2021) 104137

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

A deep machine learning algorithm for construction of the Kolmogorov–Arnold representation

A. Polar [a], M. Poluektov [b,*]

[a] Independent Software Consultant, Duluth, GA, USA
[b] International Institute for Nanocomposites Manufacturing, WMG, University of Warwick, Coventry CV4 7AL, UK

ARTICLE INFO

Keywords:
Deep machine learning
Kolmogorov–Arnold representation
Discrete Urysohn operator
Classification trees

ABSTRACT

The Kolmogorov–Arnold representation is a proven adequate replacement of a continuous multivariate function by a hierarchical structure of multiple functions of one variable. The proven existence of such representation inspired many researchers to search for a practical way of its construction, since such model answers the needs of machine learning. This article shows that the Kolmogorov–Arnold representation is not only a composition of functions but also a particular case of a tree of the discrete Urysohn operators. The article introduces new, quick and computationally stable algorithm for constructing of such Urysohn trees. Besides continuous multivariate functions, the suggested algorithm covers the cases with quantised inputs and combination of quantised and continuous inputs. The article also contains multiple results of testing of the suggested algorithm on publicly available datasets, used also by other researchers for benchmarking.

1. Introduction

The Kolmogorov–Arnold representation (Kolmogorov, 1957) of a continuous multivariate function is a decomposition of the function into a structure of inner and outer functions of a single variable. More precisely, function $F : \mathbb{R}^m \to \mathbb{R} \in C\left([0,1]^m\right)$ can be represented as

$$F\left(x_1, x_2, \ldots, x_m\right) = \sum_{k=1}^{2m+1} \Phi^k \left(\sum_{j=1}^{m} f^{kj}\left(x_j\right) \right), \tag{1}$$

where $f^{kj} : [0,1] \to [0,1] \in C[0,1]$ and $\Phi^k : \mathbb{R} \to \mathbb{R} \in C(\mathbb{R})$. The original article (Kolmogorov, 1957) only states the existence of such representation and does not provide a numerical method for its construction.

The further research of this representation can be conventionally divided into generic ways of model reduction (Sprecher, 1965; Lorentz, 1966; Sprecher, 1972; Lorentz et al., 1996) and practical ways of construction of the involved functions after picking one of the existing reduced forms (Igelnik and Parikh, 2003; Coppejans, 2004; Wasserman, 2006; Actor and Knepley, 2017; Actor, 2018). Furthermore, the relation of representation (1) to neural networks (Hecht-Nielsen, 1987; Lippmann, 1987; Kůrková, 1992; Sprecher, 1993; Kuczmann and Iványi, 2002) and specifically to the ReLU networks (Montanelli and Yang, 2020; Schmidt-Hieber, 2020) has been noticed and investigated. The Kolmogorov–Arnold representation also has practical applications, for example, in the field of image processing (Bryant, 2008; Liu, 2015).

This article approaches the construction (or *identification*) of representation (1) from a practical perspective. The aim of this article is to propose an efficient numerical identification algorithm, which is capable to construct the full form of (1), and also to recognise the cases when the reduced form of such representation is sufficient.

A previously unnoticed (to the best knowledge of the authors) property of representation (1) is the connection to the discrete Urysohn operator (Krylov, 1979), which transforms a sequence of scalars into another scalar. More precisely, $U : \mathbb{R}^m \to \mathbb{R}$ is given by

$$U\left(x_1, x_2, \ldots, x_m\right) = \sum_{j=1}^{m} g^j\left(x_j\right), \tag{2}$$

where functions $g^j : \mathbb{R} \to \mathbb{R}$ are functions of one variable. The only formal distinction between the functions of the discrete Urysohn operator and the Kolmogorov–Arnold representation is that g^j are subject to fewer restrictions, e.g. they might have a discontinuity of the first kind. After introduction of auxiliary intermediate parameters ϕ_k, it becomes obvious that the Kolmogorov–Arnold representation is also a *tree* of the discrete Urysohn operators with a single root operator and $(2m+1)$ branch operators:

$$F\left(x_1, x_2, \ldots, x_m\right) = \sum_{k=1}^{2m+1} \Phi^k\left(\phi_k\right), \qquad \phi_k = \sum_{j=1}^{m} f^{kj}\left(x_j\right). \tag{3}$$

The identification algorithm, suggested in this article, for the Kolmogorov–Arnold representation (or a tree of the discrete Urysohn

* Corresponding author.
E-mail address: m.poluektov@warwick.ac.uk (M. Poluektov).

https://doi.org/10.1016/j.engappai.2020.104137
Received 22 June 2020; Received in revised form 14 November 2020; Accepted 15 December 2020
Available online 30 December 2020

operators) is based on recently published research (Poluektov and Polar, 2020) by the same authors on non-parametric identification of an individual discrete Urysohn operator for given input–output data. The novelty of this publication is in advancement of previously-published method (Poluektov and Polar, 2020) to the case when the discrete Urysohn operators are arranged in a chain with unobserved intermediate values, as in Eq. (3). The suggested method uses each Urysohn operator as a single element in the identification and updates all functions of each operator at one step (i.e. the functions are not treated individually), which significantly expedites the entire identification process. The article targets practical aspects rather than theory and is backed up by downloadable and reproducible tests.

Application of representation (1) to modelling of physical or social systems introduces new aspects, not considered in the original work (Kolmogorov, 1957). For a physical system, the data may contain measurement noise, while for a social system, the data may even be stochastic. For example, two individuals with identical demographic parameters (the inputs) can make different decisions regarding a purchase of goods or services (the binary output). When representation (1) is used for modelling of approximate data, it is interpreted as a model of a particular multivariate function, which minimises the error between calculated output F and the real (measured) output for the unseen data.

2. Identification of the single Urysohn operator

The identification of a tree of the Urysohn operators is based on the identification of a single operator for the input–output data. This method is published in Poluektov and Polar (2020), where a detailed research of both quantised and continuous discrete Urysohn operators, including ones with multiple vector inputs, is given. It must be mentioned, that Poluektov and Polar (2020) discusses the Urysohn operators from a different perspective — modelling dynamic systems; therefore, the notation there is slightly different to this article. For convenience of the readers, the basic concept is repeated in this section, which represents only a short digest, sufficient for understanding the subsequent sections.

2.1. From linear regression to the Urysohn operator

A remarkable property of the Urysohn operator is that it can be obtained by a generalisation from the standard linear regression, which is shown in this subsection. At the first explanation step, the linear regression model is considered:

$$\hat{z}_i = \sum_{j=1}^{m} w_j x_{j,i}, \tag{4}$$

where $x_{j,i} \in \left[x_{j,\min}, x_{j,\max}\right]$ is the jth input of the ith record (also referred to as "instance" in literature), $\hat{z}_i$ is the calculated model output of the ith record.

The next step is to consider a slightly more general hypothetical model (which is not usually used in practice) just for the illustrative purposes that will become useful later. Thus, model (4) is expanded into another linear model which has two points per given input (left L_j and right R_j) instead of one weight coefficient w_j:

$$\hat{z}_i = \sum_{j=1}^{m} \left(L_j\left(1-p_{j,i}\right) + R_j p_{j,i}\right), \qquad p_{j,i} = \frac{x_{j,i} - x_{j,\min}}{x_{j,\max} - x_{j,\min}}. \tag{5}$$

A large set of input–output records for model (5) forms a system of linear algebraic equations with unknown vector-column

$$V = \begin{bmatrix} L_1 & R_1 & L_2 & R_2 & \dots & L_m & R_m \end{bmatrix}^{\mathrm{T}} \tag{6}$$

and a matrix with rows P_i built out of input values

$$P_i = \begin{bmatrix} 1-p_{1,i} & p_{1,i} & 1-p_{2,i} & p_{2,i} & \dots & 1-p_{m,i} & p_{m,i} \end{bmatrix}. \tag{7}$$

The matrix is always singular independently of the data, but the solution, can be obtained as an approximation by minimising the norm of V, using model Eqs. (5) as constraints. There are different ways of finding the constrained minimum, one of which is the projection descent method (Kaczmarz, 1937; Tewarson, 1969; Faddeev and Faddeeva, 1981; Haykin, 2014). In this method, V is iteratively changed for each input–output record independently. It is convenient to denote V at iteration i as V_i.

In the projection descent method, the initial approximation V_1 is assigned, which is the all-zero vector-column for minimising $|V|$. At each iteration, for each new data record, the following modification is performed:

$$V_{i+1} = V_i + \alpha \frac{z_i - P_i V_i}{\left|P_i\right|^2} P_i^{\mathrm{T}}, \tag{8}$$

where z_i is the real (recorded) output of the ith record and parameter $\alpha \in (0,2)$ provides error filtering and controls the convergence rate. Here, $\left|P_i\right|^2 \in \left[m/2, m\right]$ due to the definition of $p_{j,i}$. Numerator $D_i = z_i - P_i V_i$ in Eq. (8) is the residual. The matter of the projection descent method (Kaczmarz, 1937) is the navigation of point V_i to the solution by projecting it from one hyperplane to another. The magnitude of αD_i controls the distance and the sign of D_i controls the direction. Having the correct sign is crucial, as the wrong sign navigates the projected point away from the solution.

Now, the model based on the discrete Urysohn operator (2) is considered and, for convenience, it is rewritten as

$$\hat{z}_i = \sum_{j=1}^{m} g^j\left(x_{j,i}\right), \tag{9}$$

where $x_{j,i}$ and $\hat{z}_i$ are the inputs and the calculated model output of the ith record, respectively. The next key step is that functions g^j are sought in the class of piecewise-linear functions. The nodal values for each function g^j, i.e. the function values where it changes the slope, are denoted as G_k^j. In this case, the projection descent method is applicable with a few changes. Estimated vector-column V now has a block structure, where all nodal values G_k^j are written in the sequential order:

$$V = [G_1^1 \quad G_2^1 \quad \dots \quad G_{n_1}^1 \quad G_1^2 \quad G_2^2 \quad \dots \quad G_{n_2}^2 \quad \dots \quad G_1^m \quad G_2^m \quad \dots \quad G_{n_m}^m]^{\mathrm{T}},$$

where n_j is the number of the nodes of function g^j. Argument $x_{j,i}$ will always[1] fall into only one linear segment for each function g^j, hence, relative distance $\psi_{j,i}$ between the beginning of the linear segment and $x_{j,i}$ can be introduced. Vector-row P_i also has a corresponding block structure with only one pair of non-zero elements per block, which are $\left(1-\psi_{j,i}\right)$ and $\psi_{j,i}$ for the argument in this block,

$$P_i = \begin{bmatrix} P_i^1 & P_i^2 & \dots & P_i^m \end{bmatrix},$$
$$P_i^j = \begin{bmatrix} \underbrace{0 \quad \dots \quad 0}_{q_{j,i}-1} & 1-\psi_{j,i} & \psi_{j,i} & \underbrace{0 \quad \dots \quad 0}_{n_j-q_{j,i}-1} \end{bmatrix},$$

where $q_{j,i}$ is the number of the linear segment, into which $x_{j,i}$ falls. At this point, it can be seen that previously considered model (5) is a particular case of model (9) with piecewise-linear functions g^j for the case of $n_j = 2$, $\forall j$, i.e. model (5) can be understood as the Urysohn model with functions g^j each consisting of only one linear segment. For multiple linear segments, the matrix of the system becomes sparse, compared to the case of model (5), and the projection descend method converges faster, since the adjacent rows are either orthogonal or near-orthogonal.

In the computational implementation, matrices and vectors are not built — the linear algebra notations and operations were used only for the explanation. The nodes that must be modified are identified by inputs $x_{j,i}$, the difference between the calculated model output and the actual output is determined and all involved nodal values are modified,

[1] If $x_{j,i}$ falls exactly onto a nodal position, any of the adjacent linear segments can be taken. In either case, one of $\left(1-\psi_{j,i}\right)$ and $\psi_{j,i}$ will be 0, while another will be 1.

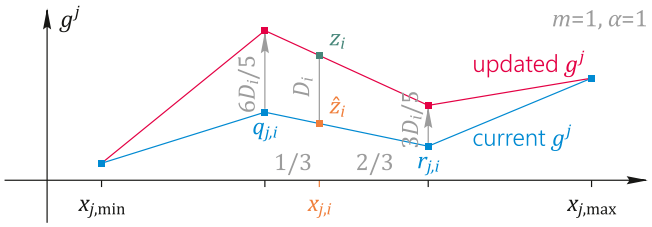

Fig. 1. A schematic illustration of one step of the iterative algorithm for identification of the discrete Urysohn operator. For clarity of the representation, the operator consisting of only one function is taken. The function is split into three linear segments. At the illustrated iteration, the input falls into the second segment and the nodes of that segment are modified according to the relative distances between the input and the nodes.

as schematically illustrated in Fig. 1. The formal algorithm is given below.

2.2. Formal algorithm for the Urysohn operator identification

First, functions g^j must be written in a piecewise-linear form. For each function g^j, input interval $[x_{j,\min}, x_{j,\max}]$ is divided into $(n_j - 1)$ equal intervals; the rescaled inputs and their rounding to the nearest integer values are introduced as

$$b_{j,i} = 1 + (n_j - 1)\,\frac{x_{j,i} - x_{j,\min}}{x_{j,\max} - x_{j,\min}}, \qquad q_{j,i} = \lfloor b_{j,i} \rfloor, \qquad r_{j,i} = \lceil b_{j,i} \rceil, \quad (10)$$

where $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$ are the floor and the ceiling functions, respectively. Now, functions g^j can be written as

$$g^j(x_{j,i}) = (1 - \psi_{j,i})\,G^j[q_{j,i}] + \psi_{j,i} G^j[r_{j,i}], \qquad \psi_{j,i} = b_{j,i} - q_{j,i}, \quad (11)$$

where G^j is the vector-column containing the nodal values of function g^j, with indices shown in $[\cdot]$. Finally, the following norm is introduced:

$$\chi_i = \sum_{j=1}^{m} \left((1 - \psi_{j,i})^2 + {\psi_{j,i}}^2 \right). \quad (12)$$

The algorithm starts with all G^j being the all-zero vector-columns. The algorithm then proceeds iteratively, with one iteration consisting of the following steps:

1. Calculate model output $\hat{z}_i$ based on actual inputs $x_{j,i}$ and the current approximation of model parameters G^j;
2. Calculate difference $D_i = z_i - \hat{z}_i$, where z_i is the real (recorded) output and $\hat{z}_i$ is the calculated model output;
3. Modify parameters G^j, such that $\alpha D_i (1 - \psi_{j,i}) / \chi_i$ is added to parameters $G^j[q_{j,i}]$ and $\alpha D_i \psi_{j,i} / \chi_i$ is added to parameters $G^j[r_{j,i}]$ for each j.

Here, parameter $\alpha \in (0, 2)$ is the noise-reduction parameter. The algorithm is considered to be converged, when D_i becomes sufficiently small for sufficiently large number of iterations consecutively.

It must be noted that it is easy to write a generalisation of the above algorithm for the case when the distances between the neighbouring nodes of function g^j may vary. Furthermore, the model and the algorithm, without any modifications, are also applicable for the quantised inputs. This case can be treated as if all inputs $x_{j,i}$ always fall onto the edges of the linear segments and only one nodal value per function is modified in the algorithm.

3. Identification of an Urysohn tree

As discussed in the introduction, the identification problem for the Urysohn tree consists in finding the unknown functions of the operators for a known set of input–output records. Furthermore, these functions are assumed to be piecewise linear, which means that the identification problem consists in finding the nodal values of the functions (i.e. the function values where the slope changes).

It is convenient to rewrite Eq. (3) and introduce the notation for the ith input–output record in the discrete form:

$$\hat{z}_i = \hat{z}_i(\phi_{k,i}) = \sum_{k=1}^{2m+1} \Phi^k(\phi_{k,i}), \qquad \phi_{k,i} = \sum_{j=1}^{m} f^{kj}(x_{j,i}), \quad (13)$$

where $x_{j,i}$ is the jth input of the ith record, $\hat{z}_i$ is the calculated model output of the ith record. Obviously, if values $\phi_{k,i}$ are known, the identification problem for model (13) reduces to the identification of multiple individual discrete Urysohn operators. Unfortunately, values $\phi_{k,i}$ are not only unobserved, but also exist only as auxiliary mathematical variables.

Before presenting the identification steps, additional quantities must be introduced. Assuming that the current approximations for functions Φ^k and f^{kj} are given and the ith data record is considered, the residual and the model error can be introduced as

$$R_i(\phi_{k,i}) = z_i - \hat{z}_i(\phi_{k,i}), \qquad E_i(\phi_{k,i}) = \left| R_i(\phi_{k,i}) \right|,$$

where z_i is the real (recorded) output of the ith record. Next, the increments for auxiliary variables are defined as

$$\Delta\phi_{k,i} = \frac{\mu R_i(\phi_{k,i})}{(2m+1)\,T(\Phi^{k\prime}(\phi_{k,i}))}, \qquad T(\zeta) = \begin{cases} \zeta & \text{if } |\zeta| \geq \delta \\ \delta & \text{if } 0 \leq \zeta < \delta \\ -\delta & \text{if } -\delta < \zeta < 0, \end{cases} \quad (14)$$

where $\mu \in (0, 1]$ is the noise-reduction parameter, function $T(\zeta)$ is introduced to account for the case of zero derivative of Φ^k and δ is a small threshold value.[2] It can be seen that if $\mu = 1$ and if functions Φ^k are linear, with the derivatives greater than δ, then the residual of updated auxiliary variables $R_i(\phi_{k,i} + \Delta\phi_{k,i})$ becomes zero. In general, for piecewise-linear functions Φ^k with the non-zero derivatives and small enough μ, updating the auxiliary variables should decrease the model error[3]:

$$E_i(\phi_{k,i} + \Delta\phi_{k,i}) < E_i(\phi_{k,i}). \quad (15)$$

However, for a prescribed μ, inequality (15) might not hold.

3.1. Record-by-record descent

The basic idea behind the identification algorithm presented here is to try to change auxiliary variables $\phi_{k,i}$ by $\Delta\phi_{k,i}$, verify whether it decreases the modelling error and, if it does, then assume $\phi_{k,i} + \Delta\phi_{k,i}$ to be the new (known) auxiliary variables, based on which all operators are updated. It is useful to remind the utilised terminology — the branch Urysohn operators contain functions f^{kj} and the root Urysohn operator contains functions Φ^k.

The formal steps of algorithm are as follows:

1. Make initial approximations of the Urysohn operators (see Section 3.2);

[2] Here, the definition of $T(\zeta)$ is rather *ad hoc* and is an elementary way of avoiding accidental division by zero. When the derivative of Φ^k is below threshold δ, value $\pm\delta$ is used instead of it to calculate $\Delta\phi_{k,i}$. This means that $\Delta\phi_{k,i}$ might be inaccurate in this case; however, this should not influence significantly the convergence of the proposed algorithm, which will be evident later from step 4 of the algorithm. Alternatively, it is possible to introduce a smooth function

$$T(\zeta) = \frac{\zeta}{(\zeta^2 - 1)\exp(-a\zeta^2) + 1}, \qquad a = \frac{\sqrt{4\delta^4 - 2\delta^2 + 1} - 2\delta^2 + 1}{\delta^2}.$$

This function is approximately ζ for $|\zeta| > \delta$ and is approximately $1/(a\zeta + \zeta)$ for $|\zeta| < \delta$. Thus, when substituted into equation for $\Delta\phi_{k,i}$, division by zero is also avoided.

[3] Except in cases when $(\phi_{k,i} + \Delta\phi_{k,i})$ fall onto the nodal positions of Φ^k, where the sign of its derivative changes from negative to positive.

2. Take one input–output record, calculate auxiliary variables $\phi_{k,i}$, model output $\hat{z}_i$, model error $E_i\left(\phi_{k,i}\right)$ and increments $\Delta\phi_{k,i}$, given the current approximations of the Urysohn operators;
3. Calculate new auxiliary variables $\phi_{k,i}+\Delta\phi_{k,i}$ and calculate model error $E_i\left(\phi_{k,i}+\Delta\phi_{k,i}\right)$ corresponding to the new auxiliary variables;
4. If inequality (15) does not hold, then go to step 2, otherwise proceed;
5. Update all branch Urysohn operators based on the new auxiliary variables obtained at step 3 (see Section 2.2 for the single operator);
6. Change the nodal positions of the root Urysohn operator, if the new auxiliary variables are outside of its domain (see below).
7. Update the root Urysohn operator based on the new auxiliary variables obtained at step 3;
8. Check for convergence, if not converged, then go to step 2.

During the update of the auxiliary variables at step 3, it can happen that some of new auxiliary variables $\phi_{k,i}+\Delta\phi_{k,i}$ are outside of the domain of the functions of the root Urysohn operator. In this case, the nodal positions of corresponding functions Φ^k must be updated (step 6) — new domain limits are calculated and the nodal positions are redistributed to cover the new domain. The function values at the new nodal positions are recalculated using either interpolation (if the new node is inside the old domain) or extrapolation (if the new node is outside the old domain).

The algorithm is specifically presented for the Kolmogorov–Arnold representation, as such representation is sufficient to describe any continuous multivariate function; however, it is easy to construct the generalisation of the algorithm for the case when the tree has arbitrary many layers of the discrete Urysohn operators. In this case, at first, the increments for all auxiliary variables are found from top to bottom (i.e. ones that are closer to the output — first), then all Urysohn operators are updated from bottom to top.

All conducted numerical experiments (see Section 4) confirmed quick convergence; however, at this moment, it is not backed up by a strict theoretical proof.

3.2. Initial approximation and stopping criteria

The identification of a single discrete Urysohn operator converges independently of an initial approximation (Poluektov and Polar, 2020). When all initial values are zeros, it converges to the solution with the minimum norm. The Urysohn tree, however, needs a specific initialisation. The simplest, but still an effective way is to assign random auxiliary variables $\phi_{k,i}$ and then update all branch operators using actual inputs $x_{j,i}$. Afterwards, calculate new auxiliary variables $\phi_{k,i}$ and, based on them, update the root operator for real (recorded) outputs z_i.

In the case if all branch operators are taken to be identical, the identification is much less efficient; thus, it must be avoided. The reason for this is that at each identification step, all branch operators are updated in the same way and, if they all are initially identical, they stay identical, which is the same as having only one addend in the model (i.e. only one Φ^k).

The iterative procedure is repeated until the convergence. Given a finite number of records and possibly a larger number of iterations required for the algorithm to converge, from a practical point of view, the user can decide to run the algorithm on the training dataset several times (or passes) and then stop. The number of passes can be decided beforehand. The numerical tests indicate the linear convergence of the algorithm (i.e. the logarithm of the error decreases on average linearly with the number of iterations).

4. Numerical simulations and tests

4.1. Goals, accuracy metrics and overfitting

The numerical simulations of this section pursue multiple goals, primary of which is comparing the proposed approach to the machine-learning results published by other software companies. It must be emphasised that this is not a competition in accuracy, because data modelling is a special type of art, where a model can often be fine-tuned for a considerable time, until a previously published benchmark is beaten. Unfortunately, the fact that a particular method is more accurate for particular dataset than another method cannot be used to make any meaningful conclusions. The goal of the comparison presented here is to make sure that the accuracy is somehow near the level reported by other researchers or companies dealing with data modelling. To facilitate the presentation of the results, a comparative table is presented at the end of the section, Table 1.

The accuracy metrics are computed for the so-called unseen data or data that has not been used in the training process. The reported metrics are the Pearson correlation coefficient between actual z_i and modelled $\hat{z}_i$ outputs and the normalised root mean square error (RMSE), defined as

$$\bar{E}_{\mathrm{RMSE}}=\frac{1}{z_{\max}-z_{\min}}\sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(z_j-\hat{z}_j\right)^2}, \tag{16}$$

where $z_{\min}$ and $z_{\max}$ are the minimum and the maximum values of the output, respectively. The error bounds are reported for each quantity as the 95% confidence interval. In the examples below, to obtain the models, 10 passes of the iterative algorithm has been made on each training dataset (i.e. if the number of records of the training dataset is N, then exactly $10N$ iterations of the algorithm have been performed).

The overfitting for the Kolmogorov–Arnold representation can be easily prevented by a model reduction. According to the model for m input parameters, the representation must have $(2m+1)$ terms with m functions each. As it has been found by testing, such large number of functions may not be required. The number of the linear blocks of each involved function can also be varied and, in some cases, can be even as low as one (i.e. two points per function). Similarly to other commonly-used types of models, the reduced models give higher errors on the training dataset and lower errors on the validation dataset. The trade-off in the accuracy and the model reductions are also demonstrated and explained in the performed tests.

4.2. Multivariate function

The input–output records of physically existing systems may not be used for the assessment of the efficiency of the modelling concept because data may have hidden faults, such as omitted or ignored inputs, significant statistical differences between the training and the validation data, large experimental noise, etc. Therefore, the first test is conducted for generated data — the ideal scenario, for which the method should give a 100%-accurate model.

The simulated system has 5 inputs and 1 output. It is computed by the following formula:

$$z=\frac{\left|\sin\left(x_2\right)^{x_1}-\exp\left(-x_3\right)\right|}{x_4}+x_5\cos\left(x_5\right), \tag{17}$$

which has no physical meaning and is chosen only as a challenging expression. The inputs are generated randomly according to the uniform distribution within the following ranges: $x_1\in[0,0.99]$, $x_2\in[0,1.55]$, $x_3\in[1,1.49]$, $x_4\in[0.4,1.39]$, $x_5\in[0,0.49]$. The output range for Eq. (17) with the given input limits is $z\in[0,2.37]$. The number of records has been taken to be 4000.

Prior to applying the Kolmogorov–Arnold representation, the basic linear regression model, Eq. (4), has been tested. Obviously, Eq. (17) is

Table 1
The summary of the results of the numerical tests.

Model	Multivar. func.	Airfoil	Mushrooms	Dynamic sys.	Bank churn
Linear regression model	$P = 0.88$	–	–	RMSE 3.2%	–
Urysohn operator	$P = 0.93$	–	4/8124 errors	RMSE 1.52%	–
Kolmogorov–Arnold rep.	$P = 0.9935$	$P = 0.9506$	–	RMSE 1.50%	81.0% corr. rec.
Third-party	–	$P = 0.952$	57/8124 errors	–	78.9% corr. rec.

far from being linear and the linear regression has failed, as expected. The entire dataset has been used for the training and the Pearson correlation coefficient of $P = 0.88$ has been observed.

The next test has been the single Urysohn operator, Eq. (9). Again, the entire dataset has been used for the training. The Pearson correlation coefficient of $P = 0.93$ has been observed. The single Urysohn model introduces nonlinearities, therefore, has a higher accuracy than the linear model. However, it cannot reach the ideal accuracy, since each input value in Eq. (9), after being transformed by a function, makes an additive contribution to the output, while Eq. (17) contains inputs also in products.

It should be mentioned that, as discussed in Poluektov and Polar (2020), the number of the nodal points per function in the single Urysohn operator significantly influences the accuracy of the model. However, depending on the data, there can exist an optimal number of points and the increase of the number of points may not necessarily increase the accuracy of the model, but can rather lead to overfitting. To show this, the single Urysohn model with a different number of linear segments (the number of nodes is equal to the number of linear segments plus one) have been fitted to the data. The entire dataset has been split into the training and the validation subsets, 50% of the records each. For the model with 2 linear segments, $P = 0.89$ has been observed on the validation dataset, for 4 linear segments — $P = 0.92$, for 8 linear segments — $P = 0.93$, for 16 linear segments — $P = 0.75$ on the validation dataset and $P = 0.99$ on the training dataset. The results show a clear sign of overfitting on this data for a large number of linear segments, and the best approximation is achieved for the number of segments around 8.

The Kolmogorov–Arnold model has been tested only on the unseen data, using the 10-fold cross-validation method. After the records have been generated, 90% of all records have been used for the training and the remaining 10% for the validation. The test has been repeated 10 times, each time with a different validation segment, such that each record has been used 9 times for the training and 1 time for the validation. For constructing the model, 11 addends in the Kolmogorov–Arnold model (i.e. $(2m + 1)$, where m is the number of the inputs) and 10 nodal points per function have been used.

Based on 5 consecutive executions, the normalised RMSE of $\bar{E}_{\mathrm{RMSE}} = 0.0203 \pm 0.0023$ and the Pearson correlation coefficient of $P = 0.9935 \pm 0.0014$ have been observed. This is the expected performance of the Kolmogorov–Arnold model, approaching the ideal accuracy. The discrepancy can be attributed to a finite number of nodal points per function and a finite number of records used for the identification. Therefore, it can be concluded that the proposed method passes this particular test of modelling the input–output data generated by the non-linear multivariate function.

To study the effects of the model reduction, the number of addends in the Kolmogorov–Arnold model has been varied. For a single addend, the Pearson correlation coefficient of $P = 0.88$ has been observed, for two addends — $P = 0.93$, for three addends — $P = 0.96$. Finally, for 6 addends, P becomes comparable to that of the full model, which means that for this particular example, it is not necessary to use all 11 addends in the Kolmogorov–Arnold model for achieving a high accuracy. This is an illustration of how for many real-life datasets, the model can be reduced and the researchers can vary the complexity by moving from the most simple to the full one. The source code and the data are available online.[4] The code is reusable — the generated data can be replaced by records for a physical system.

4.3. Airfoil self-noise

The next example is a comparison of the proposed approach to the data science and machine-learning platform Neural Designer[5] developed by Artelnics. The chosen dataset is the "Airfoil self-noise", which can be found in UCI Machine Learning Repository (Dua and Graff, 2017). The dataset is experimentally-obtained from a series of aerodynamic and acoustic tests of two- and three-dimensional airfoil blade sections conducted in an anechoic wind tunnel. The details of the Neural Designer modelling techniques can be found in the online help.[6] In this test, the dataset, consisting of 1503 records, is randomly split into 60% training subset, 20% selection subset and 20% testing subset. The model is trained multiple times on the training subset, tested on the unseen selection subset and the result with the best metric is then applied to the testing subset. The company reports that they have achieved the Pearson correlation coefficient of $P = 0.952$.

The modelled system has 5 inputs and 1 output. For constructing the model, 11 addends in the Kolmogorov–Arnold model (i.e. exactly according to Eq. (1)) and 15 nodal points per function have been used. To illustrate the dataset, the first three records are shown below:

800; 0; 0.3048; 71.3; 0.00266337; 126.201

1000; 0; 0.3048; 71.3; 0.00266337; 125.201

1250; 0; 0.3048; 71.3; 0.00266337; 125.951.

The authors followed the same training–testing path as reported by Artelnics — the programme repeated training 10 times, chose the best model according to the metric obtained on the selection subset and applied this model to the testing data. Based on 5 consecutive executions, the Pearson correlation coefficient of $P = 0.9506 \pm 0.0049$ has been observed. This means that for the considered experimental dataset, the accuracy of the proposed modelling and identification technique is at the same level as the accuracy of the machine-learning software Neural Designer. The source code and the data are available online.[7]

4.4. Mushroom classification

The third example is a classification problem. This dataset has been publicly available for a long time and there are many software companies and individuals who tried modelling it. The input consists of 22 mushroom properties and the output is a quality of the mushroom which is ether "edible" or "poisonous". The dataset "Mushroom" consists of 8124 records and can be found in UCI Machine Learning Repository (Dua and Graff, 2017). To illustrate the dataset, the first three records are shown below:

p, x, s, n, t, p, f, c, n, k, e, e, s, s, w, w, p, w, o, p, k, s, u

e, x, s, y, t, a, f, c, b, k, e, c, s, s, w, w, p, w, o, p, n, n, g

e, b, s, w, t, l, f, c, b, n, e, c, s, s, w, w, p, w, o, p, n, n, m .

[4] http://ezcodesample.com/naf/reallife5.html.
[5] https://www.neuraldesigner.com/.
[6] https://www.neuraldesigner.com/learning/examples/airfoil-self-noise-prediction.
[7] http://ezcodesample.com/naf/reallife3.html.

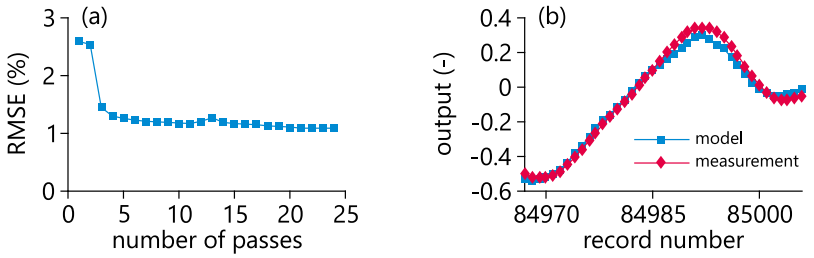

Fig. 2. The convergence of the model depending on the number of passes over the training dataset (a) and a fragment of the output signal, model vs. measurement (b).

The first column contains the output — "edible" or "poisonous", while other columns contain the mushroom properties.

Since the inputs are represented symbolically, some data preprocessing is required. Although the Kolmogorov–Arnold model relies on continuous functions, it can work perfectly with the quantised input data, as discussed in Section 2.2. For the quantised data, e.g. sequential integers, the nodes of the functions are arranged such that a particular input always coincides with a node. The functions then are assumed to be linear between the nodes, as for the non-quantised case. In the implementation, the symbolic inputs were transformed into the sequential integers, for example, the second column of the dataset contains elements of either of the following types: 'b', 'c', 'f', 'k', 's', 'x', which were replaced by integers from 1 to 6. The symbolic outputs 'e' and 'p' were replaced by 1 and −1, respectively.

For a comparison to a third party, the Microsoft ML.NET library[8] has been used. The accuracy for the library using the 10-fold cross-validation method for the classification tree model was 57 errors for the entire dataset of 8124 records. In a single step of the 10-fold cross-validation method, 90% of all records are used for the training and the remaining 10% for the validation. Such step is repeated 10 times, each time with a different validation segment. The execution time for the Microsoft ML.NET library was 63 s.

The same 10-fold cross-validation approach has been used for a single quantised Urysohn operator. For 10 consecutive executions of the programme, the average number of mistakenly identified records was 3.60 ± 0.37. The execution time of the code was 2 s. This result shows that even a single quantised Urysohn operator has outstanding descriptive capabilities and can outperform established techniques, both in accuracy and execution time. The source code and the data are available online.[9]

4.5. Electronic non-linear dynamic system

This test has been conducted for input–output recordings of an electronic non-linear dynamic system, which is assembled on a circuit board. The system is of a Wiener–Hammerstein type and contains a static nonlinearity that is sandwiched between two linear time-invariant blocks. Dataset "Wiener–Hammerstein System (2009)" (Schoukens et al., 2009) has been taken from the website[10] specifically aimed at benchmarking of non-linear dynamic models, where the detailed description of the tested system and the data can be found. Unfortunately, the website deliberately does not provide the accuracy that they have achieved themselves, motivating this by saying that the "benchmark is not intended as a competition" (Schoukens et al., 2009).

Since the considered object is a single-input single-output (SISO) dynamic system, its inputs and outputs are arrays of numbers of size 1 by N. However, an output z_i of a dynamic system depends not only on x_i, but also on a finite fragment of preceding inputs, thus $z_i = z_i\left(x_i, x_{i-1}, x_{i-2}, \ldots, x_{i-m+1}\right)$. Therefore, for the identification of the Kolmogorov–Arnold model, the data has been rearranged accordingly, such that each record represents m inputs and 1 output.

The entire dataset of $N = 188000$ records has been split equally into the training and the validation subsets, according to recommendation of the website. For the considered example, the length of the input fragment that defines the output has been determined to be $m = 35$ elements. In this case, the full Kolmogorov–Arnold representation should have 71 addends. However, only 4 addends appeared to be sufficient and the number of the linear blocks in each function has been taken to be 15. In addition to this, the linear regression model, Eq. (4), and the single Urysohn model, Eq. (9), have been tested as well.

For the Kolmogorov–Arnold representation, the normalised RMSE of $\bar{E}_{\mathrm{RMSE}} = 0.0150 \pm 0.0016$ has been observed, while for the single Urysohn model $\bar{E}_{\mathrm{RMSE}} = 0.0152$ and for the linear model $\bar{E}_{\mathrm{RMSE}} = 0.032$ have been calculated. The difference between the Kolmogorov–Arnold model and the single Urysohn model is insignificant, since when the error is as low as 1.5%, there is almost no room for improvement, especially when the unseen data is of a physical system, with errors in the measurement. Also, the single Urysohn model is a particular case of the Kolmogorov–Arnold model (i.e. one branch and a linear outer function) and it is recommended as a starting point for modelling. Overall, such errors are usually considered to be acceptable for modelling of dynamic systems. The convergence of the model depending on the number of passes over the training dataset and a fragment of the output signal (model vs. measurement) are illustrated in Fig. 2. The source code and the data are available online.[11]

4.6. Bank churn

The final test is a model of the social system, where individuals make decisions regarding a subscription to a bank service. The data are individual features and the output is a human decision. This dataset is taken from the Neural Designer website.[12] The file contains the decisions regarding the bank clients and some of their personal data (provided anonymously), used as inputs. To illustrate the dataset, the first three records are shown below:

15634602; 619; France; Female; 42; 2; 0; 1; 1; 1; 101348.88; 1

15647311; 608; Spain; Female; 41; 1; 83807.86; 1; 0; 1; 112542.58; 0

15619304; 502; France; Female; 42; 8; 159660.8; 3; 1; 0; 113931.57; 1.

The first number is the ID, which is ignored for modelling; the last binary value is YES/NO model output; the other parameters are either quantised, such as the country and the gender, or continuous, such as the balance and the salary. The total number of records is 10 000.

The authors of Neural Designer split the dataset into 60% training subset, 20% selection subset and 20% testing subset. The training is conducted multiple times, the best model is chosen based on the selection subset and is applied to the testing subset. They report the percentage of the correctly modelled records of 78.9% on the testing subset.

[8] https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet.
[9] http://ezcodesample.com/naf/reallife4.html.
[10] http://www.nonlinearbenchmark.org/.
[11] http://ezcodesample.com/naf/reallife1.html.
[12] https://www.neuraldesigner.com/learning/examples/bank-churn.

For the modelling approach proposed in this article, the branch Urysohn operators have been built as having both continuous and quantised inner functions. The number of addends in the Kolmogorov–Arnold model have been taken to be 3, the number of nodes per function has been chosen individually for each parameter between 2 and 6. The test has been conducted by running 10-fold cross-validation (i.e. all predictions are done for the unseen data). For 10 consecutive executions, the average number of the correct predictions has been observed to be 8103.7 ± 71.0, which gives $81.0\% \pm 0.7\%$ accuracy of the model. For this example again, the accuracy of the proposed approach is comparable to that of the Neural Designer software. The source code and the data are available online.[13]

5. Conclusions

In this article, a novel identification method for a tree of the discrete Urysohn operators has been proposed. A particular case of such tree is the Kolmogorov–Arnold representation. The proposed algorithm can be classified as the deep machine-learning algorithm, as it can model human choices and deals with the model consisting of several layers, having unobserved intermediate (hidden) variables.

The suggested method is based on the fundamental properties of the discrete Urysohn operator discovered only recently. Therefore, it is a start of a new technology, which may be improved and upgraded by other researchers. The other methods, to which this technique has been compared, have been evolving for a long time, and companies, which developed the codes, have a long history in this field. Nevertheless, the proposed identification method resulted in models of a comparable accuracy on a large variety of real-life data — a physical object (airfoil blade vibration in a wind tunnel), a biological classification problem (edible/poisonous mushrooms), an electronic dynamic system (a circuit board processing signals in a non-linear way), and a social system (human decisions regarding a subscription to a bank service).

The number of tests, conducted by the authors using publicly available datasets, is higher than provided in this article. All tests showed similar or better results compared to the results reported by other researchers, when different methods have been used.

The authors can point to two major advantages of the Urysohn-tree model. First, it covers a wide range of complexity, starting from the linear regression and including the Kolmogorov–Arnold representation. Furthermore, it allows making a choice in terms of a trade-off between the accuracy and the number of parameters, preventing the overfitting. Second, it may provide a more intuitive model compared to, for example, random forest, neural networks, support vector machine, etc. Each identified function of each Urysohn operator can be shown graphically and, in some cases, the researchers can even make theoretical conclusions regarding an influence of a certain input on the output and can possibly perform a manual tuning. By looking at the functions in the piecewise-linear form, the researchers can make decisions regarding the number of the linear blocks or the number of the branch operators.

Some theoretical aspects are not yet researched, but the algorithm can already be used as is. The name suggested for it is the "Urysohn-tree identification" rather than the "construction of the Kolmogorov–Arnold representation", since the latter representation is a particular tree with certain hierarchy and a number of inner and outer functions, while an Urysohn tree can contain any number of Urysohn objects with at least one sequential connection and at least one unobserved variable, which is the input of one operator and the output of another operator at the same time.

CRediT authorship contribution statement

A. Polar: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Writing - review & editing. **M. Poluektov:** Methodology, Validation, Writing - original draft, Writing - review & editing, Visualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Actor, J., 2018. Computation for the Kolmogorov Superposition Theorem (Master's thesis). Rice University.

Actor, J., Knepley, M.J., 2017. An algorithm for computing Lipschitz inner functions in Kolmogorov's superposition theorem. arXiv:1712.08286.

Bryant, D., 2008. Analysis of Kolmogorov's Superposition Theorem and its Implementation in Applications with Low and High Dimensional Data (PhD thesis). University of Central Florida.

Coppejans, M., 2004. On Kolmogorov's representation of functions of several variables by functions of one variable. J. Econometrics 123 (1), 1–31.

Dua, D., Graff, C., 2017. UCI machine learning repository. http://archive.ics.uci.edu/ml.

Faddeev, D.K., Faddeeva, V.N., 1981. Computational methods of linear algebra. J. Sov. Math. 15 (5), 531–650.

Haykin, S., 2014. Adaptive Filter Theory. Pearson.

Hecht-Nielsen, R., 1987. Kolmogorov's mapping neural network existence theorem. In: Procceedings of the International Conference on Neural Networks, pp. 11–14.

Igelnik, B., Parikh, N., 2003. Kolmogorov's spline network. IEEE Trans. Neural Netw. 14 (4), 725–733.

Kaczmarz, S., 1937. Angenäherte auflösung von systemen linearer gleichungen. Bull. Int. l'Acad. Pol. Sci. Lett. Classe Sci. Math. Nat. Sér. A 35, 355–357.

Kolmogorov, A.N., 1957. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. Dokl. Akad. Nauk SSSR 114 (5), 369–373.

Kůrková, V., 1992. Kolmogorov's theorem and multilayer neural networks. Neural Netw. 5 (3), 501–506.

Krylov, V.V., 1979. Models of discrete systems with infinite-dimensional state space. Autom. Remote Control 40 (5), 694–699.

Kuczmann, M., Iványi, A., 2002. Neural network model of magnetic hysteresis. COMPEL - Int. J. Comput. Math. Electr. Electron. Eng. 21 (3), 364–376.

Lippmann, R.P., 1987. An introduction to computing with neural nets. IEEE ASSP Mag. 4 (2), 4–22.

Liu, X., 2015. Kolmogorov Superposition Theorem and its Applications (PhD thesis). Imperial College London.

Lorentz, G.G., 1966. Approximation of functions. Holt Rinehart and Winston, New York.

Lorentz, G.G., von Golitschek, M., Makovoz, Y., 1996. Constructive Approximation. Springer-Verlag Berlin Heidelberg.

Montanelli, H., Yang, H., 2020. Error bounds for deep ReLU networks using the Kolmogorov-Arnold superposition theorem. Neural Netw. 129, 1–6.

Poluektov, M., Polar, A., 2020. Modelling non-linear control systems using the discrete Urysohn operator. J. Franklin Inst. B 357 (6), 3865–3892.

Schmidt-Hieber, J., 2020. The Kolmogorov-Arnold representation theorem revisited. arXiv:2007.15884.

Schoukens, J., Suykens, J., Ljung, L., 2009. Wiener-Hammerstein benchmark, In: 15th IFAC Symposium on System Identification (SYSID 2009), St. Malo, France.

Sprecher, D.A., 1965. On the structure of continuous functions of several variables. Trans. Amer. Math. Soc. 115, 340–355.

Sprecher, D.A., 1972. An improvement in the superposition theorem of Kolmogorov. J. Math. Anal. Appl. 38 (1), 208–213.

Sprecher, D.A., 1993. A universal mapping for Kolmogorov's superposition theorem. Neural Netw. 6 (8), 1089–1094.

Tewarson, R.P., 1969. Projection methods for solving sparse linear systems. Comput. J. 12 (1), 77–80.

Wasserman, L., 2006. All of Nonparametric Statistics. Springer-Verlag New York.

[13] http://ezcodesample.com/naf/reallife6.html.