modelling

MDPI

Article

Air Pollution Modelling by Machine Learning Methods

Petra Vidnerová [1,*,†] and Roman Neruda [1,†]

The Czech Academy of Sciences, Institute of Computer Science, 182 07 Prague, Czech Republic; roman@cs.cas.cz
* Correspondence: petra@cs.cas.cz
† These authors contributed equally to this work.

Abstract: Precise environmental modelling of pollutants distributions represents a key factor for addresing the issue of urban air pollution. Nowadays, urban air pollution monitoring is primarily carried out by employing sparse networks of spatially distributed fixed stations. The work in this paper aims at improving the situation by utilizing machine learning models to process the outputs of multi-sensor devices that are small, cheap, albeit less reliable, thus a massive urban deployment of those devices is possible. The main contribution of the paper is the design of a mathematical model providing sensor fusion to extract the information and transform it into the desired pollutant concentrations. Multi-sensor outputs are used as input information for a particular machine learning model trained to produce the CO, NO2, and NOx concentration estimates. Several state-of-the-art machine learning methods, including original algorithms proposed by the authors, are utilized in this study: kernel methods, regularization networks, regularization networks with composite kernels, and deep neural networks. All methods are augmented with a proper hyper-parameter search to achieve the optimal performance for each model. All the methods considered achieved vital results, deep neural networks exhibited the best generalization ability, and regularization networks with product kernels achieved the best fitting of the training set.

Keywords: machine learning; air pollution; sensors; deep neural networks; regularization networks

Citation: Vidnerová, P.; Neruda, R. Air Pollution Modelling by Machine Learning Methods. *Modelling* **2021**, *2*, 659–674. https://doi.org/10.3390/modelling2040035

Academic Editor: Miquel Sànchez-Marrè FiEMSs

Received: 12 October 2021
Accepted: 12 November 2021
Published: 17 November 2021

1. Introduction

Air pollution monitoring is one of the most important emerging environmental issues in the treatment of urban air pollution. Urban atmospheric pollutants are responsible for the respiratory and other illness of urban citizens. Some of the pollutants (e.g., benzene) are known to induce cancers in case of prolonged exposure. Therefore, the precise modelling of pollutants distribution is needed for traffic management and for the definition of mobility plans designed to face these problems. Nowadays, urban air pollution monitoring is primarily carried out employing networks of spatially distributed fixed stations. A limited number of those stations represent a problem in estimating the real distribution of gases and particles in a complex urban environment.

The work in this paper aims at improving the situation by utilizing machine learning models to process outputs of multi-sensor devices that are small and cheap, thus a massive urban deployment of those devices is possible. The outputs of those sensors are less reliable in comparison to the currently used fixed stations, therefore a mathematical model providing sensor fusion is utilized to extract the information and transform it into desired pollutant concentrations. The main idea is that multi-sensor outputs are used as input information for a particular machine learning model which is trained to produce the CO, NO2, and NOx concentrations in a supervised learning scenario. The desired outputs of the model are provided by reliable but expensive measurements of the pollutants. Our results should provide a proof of concept of this approach, and show that with clever processing of sufficient data, it is possible to model and estimate concentrations of desired pollutants even from cheap and unreliable sensors.

Machine learning models have recently been successfully applied to many challenging problems from image recognition and robotics to language translation and DNA sequencing [1,2]. In the majority of these areas they now provide the best solutions available, surpassing the "white-box" analytic solutions. While the lack of explanation abilities of these approaches can be seen as a disadvantage, their efficiency of data modelling and their adaptivity to new problems and datasets represent big benefits. In particular, the area of cheap gas sensors, where the exact analytic descriptions are hard to obtain or differ for different sensor technologies, seems to be a suitable domain for machine learning where expert knowledge is augmented by efficient autonomous algorithms.

The application of machine learning models to the processing of sensor data presents several challenges that have already been identified and tackled by previous research. The majority of related work in this area uses regression approaches to model individual sensor outputs, or in more advanced settings, to provide multi-sensor fusion with relevant data (such as pollutant concentrations) as the output of the model.

The authors of [3] have applied several simple machine learning models for classification and regression tasks to utilize multi-sensor data in the indoor environment for the task of occupancy prediction formulated as supervised learning. Their models include decision trees, Bayesian networks and linear regression. The importance of raw data preprocessing and using multiple inputs (including augmenting data collected by humans) was stressed. The similar indoor scenario is used in paper [4] where Bayesian and neural networks are used for predicting and correcting multi-sensor outputs measuring temperature, humidity, pressure, and so forth. Their results with relatively small datasets and simple models imply the possibility of reconstructing relevant information by multi-sensor fusion.

In the last several years, deep neural networks became a very popular machine learning tool and they were also applied in sensor information processing field. The authors of [5] describe a low-cost multi-sensor hardware device measuring CO2 concentrations. The complete system consists of six sensors and a deep neural network model that demonstrated superior performance compared to linear regression, albeit the computing requirements for training (dozens of hours) are large. A recent work [6] presents a successful application of deep (convolutional) network for multi-sensor data fusion solving the problem of steel element defects. Their results seem to be dependent on clever preprocessing and analysis of the sensor output.

In this paper, we work with datasets that have been already used for analysis and modelling by different machine learning models by De Vito et al. [7,8]. The data consists of dozens of thousands of measurements of concentrations of several gas pollutants obtained from multi-sensor devices recording. As mentioned earlier, the measurements are labelled by reliable conventional air pollution monitoring stations. The technical nature of the data is described in detail in Section 4.

Several machine learning models are compared on the task of prediction of CO, NO2 and NOx concentrations based on the multi-sensor fusion of the above obtained data. The tested methods represent state-of-the-art approaches ranging from kernel methods to deep neural networks. It is important to stress that all methods have been extensively tested for their performance not only by the standard statistical measures and approaches as the cross-validation, but also the optimal choice of hyper-parameters (types of a kernel, size of a model, etc.) has been performed by extensive global search. In the case of regularization networks, the hyper-parameter search is performed by a genetic algorithm, and in the case of deep neural networks, it is performed by an evolution strategy [9].

The obtained results show that all machine learning models we have tested were able to perform the task of modelling the pollutant concentrations from multi-sensor data in a satisfiable way. The overall best model in terms of generalization criteria has been a particular architecture of a deep neural network. On the other hand, the best precision on training data has been achieved by our original architecture of kernel networks with product units.

The paper is organized as follows. In the next section, related work concerning machine learning models is discussed. Section 3 describes the used machine learning algorithms in more detail, namely Section 3.1 derives the regularization network, Section 3.2 introduces the extension of regularization network via composite kernels, while Section 3.3 explains how composite kernels are evolved. Then, Section 3.4 briefly introduces deep neural networks and Section 3.5 describes how architecture of deep neural networks is optimized. Finally, Section 4 describes the dataset. Section 5 contains the results of our experiments and our conclusion can be found in Section 6.

2. Related Work

Among machine learning methods, kernel methods became very popular in the 1990s. They have been applied to many real-world problems, and they are still considered to be state-of-the-art methods in various domains [10]. In this work, we study the regularization network (RN), a feed-forward neural network with one hidden layer, designed for supervised learning. RN are based on a good theoretical background and their architecture has been mathematically proven to be the solution to the problem of supervised learning formulated as a minimization problem with regularization (see [11–14]).

The paper [15] demonstrates how the kernel function choice significantly influences the RN performance. Therefore, the optimal choice of the kernel function always depends on a task given. Kůrková et al. have studied the theoretical properties of kernel units with variable and fixed widths [16–19].

More complex models, described in detail in Section 3.2, belong to the class of the *multi-kernel* models. Recently, this field has been studied extensively [10,20–22]; however, corresponding algorithms are mostly designed for the support vector machines (SVM).

In addition, the Support Vector Regression (SVR) model is used in this paper. It is an extension of the well-known Support Vector Machine (SVM) for regression problems. Like the SVM model, SVR depends only on a subset of the training data, because the cost function ignores any training data close to the model prediction [23,24].

Finally, the last method considered in this work is the Deep Neural Network (DNN). DNNs have become the state-of-the-art methods in many machine learning application areas, recently. They have been applied to various problems, such as image and speech recognition, natural language processing [1,2].

DNN are neural networks with multiple hidden layers. In this paper, we consider only feed-forward networks. The types of network units typically depend on the particular application. The commonly used units are traditional perceptrons or the rectified linear unit (*ReLU*).

The weights of DNN are trained by algorithms based on stochastic gradient descent. However, the architecture (i.e., a number and sizes of layers, and a type of activation function) has to be set up manually by an expert. The choice of architecture influences significantly the performance of the DNN, but still, it is typically done by a trial and error method [25].

3. Machine Learning Methods

In this section, we will describe the machine learning models used in our experiments in more details, with the emphasis on our original modifications and extensions. The most important part deals with a hyper-parameter search for (composite) kernel and deep neural networks. We have utilized evolutionary computing optimization to provide a global search procedure for the hyper-parameter sweep. This is done to make sure the optimal model (within some reasonable constraints) is found, and thus the performance comparison is as fair as possible.

First, let us overview the models and hyper-parameter search algorithms:

- regularization networks (RN);
  - **evolutional algorithm for RN hyper-parameter search**;
- RN with **composite kernels**;

  - **evolutional algorithm for RN and composite kernel hyper-parameter search**;
- deep neural networks (DNN);
  - **evolution strategies for DNN architecture search**.

The models and algorithms in bold are our original extensions.

3.1. Regularization Networks and Kernels

This section introduces the regularization network architecture used for supervised learning. The supervised learning we reformulate in the context of a function approximation. Given a set of data points $\{(\vec{x}_i, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^N$ (where $\vec{x}_i$ is an input, y_i is a corresponding output, and N is a number of data points) obtained by random sampling of a real function f, the goal of supervised learning is to search for this function. Generally, the problem is ill-posed. Therefore we add *a priori* knowledge about f. It is usually assumed that f is *smooth*, that is, each pair of similar inputs corresponds to a pair of similar outputs. The solution to the problem is found by minimization the functional (1) which contains both the data and the regularization term.

$$H[f] = \frac{1}{N}\sum_{i=1}^{N}(f(\vec{x}_i) - y_i)^2 + \gamma\Phi[f], \tag{1}$$

where Φ is a regularization term and $\gamma > 0$ is *the regularization parameter*. For a wide class of regularization terms, the solution can be represented by a feed-forward neural network with one hidden layer, and linear output units called a *regularization network* (RN) [14,26]. The RN obtained as a solution is unique and it has the form:

$$f(\vec{x}) = \sum_{i=1}^{N} w_i K(\vec{x}_i, \vec{x}), \qquad (N\gamma I + K)\vec{w} = \vec{y}, \tag{2}$$

where I is the identity matrix, K is the matrix $K_{i,j} = K(\vec{x}_i, \vec{x}_j)$, K is an appropriate kernel function, and $\vec{y} = (y_1, \ldots, y_N)$.

For the given γ and the type of kernel function K, the algorithm is simple and efficient, since it is, in fact, a problem of solving a linear system. In our approach, the search for suitable hyper-parameters is performed by a genetic algorithm optimization technique. This will be described in detail in Section 3.3.

3.2. Composite Kernels

The kernel function represents our presumption (called bias in machine learning) about a solution to the given approximation problem. Therefore, it is usually assumed to be chosen by a user. However, this choice has a significant impact on the learning performance and therefore should be done for each task independently.

In theory, it is typically assumed that a kernel is a symmetric and positive-definite function. On the other hand, in practice, various other functions are used. In [27], it was shown that conditionally positive definite kernels possibly outperform classical kernels.

Commonly used kernel functions are: linear $K(\vec{x}, \vec{y}) = \vec{x}^T\vec{y}$, polynomial $K(\vec{x}, \vec{y}) = (\gamma\vec{x}^T\vec{y} + r)^d, \gamma > 0$, Gaussian radial basis function $K(\vec{x}, \vec{y}) = exp(-\gamma||\vec{x} - \vec{y}||^2), \gamma > 0$, sigmoid $K(\vec{x}, \vec{y}) = \tanh(\gamma\vec{x}^T\vec{y} + r)$. Where, γ, d and r are the kernel's parameters.

Recently, several algorithms have been proposed to use composite kernel functions, that is, functions consisting of combinations of basic kernel functions [20]. Data are often multi-modal and then such composite kernels may better reflect the character of data.

In our previous papers [28,29], we have proposed composite kernel functions for RN and, consequently, evolutionary algorithms searching for data-dependent optimal kernel functions have been proposed in [30] for networks with product units, and in [31] for networks with sum units. Based on Aronszajn's breakthrough results [32], we have shown that it is possible to use composite kernels as activation functions in the RNs (cf. [33]). Such composite kernels also often outperform a simple kernel function.

The *product kernel* is a function K:

$$K(\vec{x}_1, \ldots, \vec{x}_k, \vec{y}_1, \ldots, \vec{y}_k) = K_1(\vec{x}_1, \vec{y}_1) \cdots K_k(\vec{x}_k, \vec{y}_k), \tag{3}$$

where $K_1, \ldots, K_k$ are kernel functions defined on $\Omega_1, \ldots, \Omega_k$ ($\Omega_i \subset \mathbb{R}^{d_i}$) respectively, $\vec{x}_i \in \Omega_i$ and $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_k$.

The *sum kernel* is a function K: $K(\vec{x}, \vec{y}) = K_1(\vec{x}, \vec{y}) + K_2(\vec{x}, \vec{y})$, where K_1 and K_2 are kernel functions.

It is possible to combine various types of kernels or just two functions of same type but with different parameters, that is, two Gaussians of different widths (note that in this case the Gaussians have the same center).

3.3. Genetic Search for Kernels

In order to find the right kernel function in a data-dependent and autonomous way, a suitable search algorithm is needed. A sound and robust global optimization method called genetic algorithms (GA) [34] was used in our approach. Considered to be an efficient stochastic population-based meta-heuristic, the GA allows to search for various types of kernels, including composite kernels. The GAs work with a population of *individuals* encoding feasible solutions. During the process of search, each individual is evaluated by a *fitness* function value that reflects the quality of the corresponding solution.

Populations in GA evolve so as to approach better individuals. The algorithm starts with individuals generated randomly, and it creates a new population each iteration of the algorithm (generation). In each iteration, the fitness of all individuals is evaluated. Based on the fitness, the individuals are stochastically selected for reproduction and then altered by genetic operators *mutation* and *crossover*.

To find suitable hyper-parameters for RN (described above), our individuals encode the kernel function type, other possible parameters of the kernel function, and the regularization parameter.

For example, a simple kernel function is encoded as $I = \{K, p, \gamma\}$, where K is the type of kernel function, p is a kernel parameter, and γ is a regularization parameter, that is, $I = \{\text{Gaussian}, \text{width} = 0.2, \gamma = 0.001\}$.

The product kernel function can be represented by the individual: $I = \{K_0, p_0, K_1, p_1, i_1, \ldots, i_n, \gamma\}$, where K_0, K_1 are the types of kernel functions, p_0, p_1 are kernel parameters, and $i_1, \ldots, i_n \in \{0, 1\}$ is an index of kernel by which the i-th attribute is processed, i.e $I = \{\text{Gaussian}, 0.73, \text{Inverse\_Multiquadric}, 1.47, [0, 0, 1, 0, 1, 1, 1, 1], \gamma = 0.1\}$.

The genetic operators of mutation and crossover are rather simple and based on standard evolutionary approaches for vectors of floating-point and discrete values. Our mutation operator is a standard biased mutation which performs small random perturbation drawn from normal distribution to a small number of randomly selected numerical values of an individual.

The crossover operator differs slightly for different types of regularization networks: for simple kernels of the same type, we use an arithmetic crossover that modifies the kernel parameter and γ, that is, the new values are generated randomly:

$$\gamma = (1 - r)\gamma_1 + r\gamma_2,$$

where $r \in \langle 0, 1 \rangle$ is a random number, γ_1 and γ_2 are parents' values, and γ is the offspring value.

In the case of composite kernels, the situation is more complicated, and the standard discrete one-point crossover is used—the sub-kernels are interchanged.

A standard and robust *tournament selection* is used as a selection operator.

The fitness value must reflect the quality of the corresponding RN. To estimate the real generalization ability of a network we use a *cross validation error* [35]. Thus, it should be stressed that we are looking for such hyper-parameters that minimize the crossvalidation error.

3.4. Deep Neural Networks

Artificial neural networks (ANNs) are a class of computational models used in machine learning. In general, an ANN is a large collection of simple connected units called artificial neurons. The neurons are typically organized in layers. Networks with more layers between the input and output layer are known as *deep neural networks* (DNN).

A neuron is a computational unit with n real inputs x_i and one real output y. It realizes a function $y = g(\sum_{i=1}^{n} w_i x_i + w_0)$, where g is an activation function, and $w_i \in \mathbb{R}$ are weights. A typical activation function is the sigmoid function $y(z) = \frac{1}{1+e^{-z}}$, but a current popular alternative to sigmoid is the so-called rectified linear unit (*ReLU*): $y(z) = max(0, z)$.

DNN realizes a function $f(W) : \mathbb{R}^N \to \mathbb{R}^M$, where N is the number of input neurons and M is a number of output neurons, W is a matrix of network parameters, the weights. The learning of DNN consists of the optimization of a cost function with respect to weights W. This optimization is typically done by a version of the stochastic gradient descent algorithm. To prevent overfitting, various regularization techniques can be used. The most common method is the so-called *dropout*, which is a very efficient way of performing model averaging by dropping out individual neurons.

The hyper-parameters of DNN (traditionally referred to as an architecture), that is, the number of hidden layers, the number of neurons in individual layers, and so forth, are typically chosen by a user, often by the time-consuming trial and error method.

3.5. Evolution Strategies for DNN Design

In our work, we use evolution strategies to search for the optimal architecture (hyper-parameters) of DNN, while the weights are learned by a conventional gradient based technique.

Evolution strategies (ES) are a kind of evolutionary algorithm. They were designed for real-valued individuals [36]. Similarly to GA, they work with a population of individuals, evolving them by means of selection, crossover and mutation operators. The key operator in ES is the Gaussian mutation:

$$x \leftarrow x + \sigma N(0,1)$$
$$\sigma \leftarrow \sigma(1 + \alpha N(0,1)),$$

where x is a variable under mutation and σ is the corresponding strategy coefficient, $\alpha \in \mathbb{R}$ is a mutation coefficient, $N(0,1)$ stands for normal distribution.

There are two traditional forms for evolution strategies. The $(n+m)$-ES generates new generation by deterministically choosing n best individuals from the set of $(n+m)$ parents and offspring. The (n,m)-ES generates new generation by selecting from m new offspring (typically, $m > n$). The latter approach is considered more robust against local optima premature convergence.

Evolution strategies represent a very successful optimization approach used for solving complex and large problems.

The main idea of our approach to applying ES for DNN architecture search is to restrict the space of feasible architectures as much as possible. Therefore, the architecture specification is simplified. It directly follows the implementation of DNN in the popular Keras library [37], where networks are specified as a list of fully connected layers. A layer is then defined by the number of neurons, the type of an activation function, and the type of regularization (such as dropout).

In our algorithm, the (n,m)-ES is used. Offspring are generated using both mutation and crossover operators. Our individuals represent network topologies and therefore they are not represented by real-valued vectors. So, our operators have to slightly differ from classical ES.

Each individual represents one DNN. It consist of blocks defining the network's layers.

$$I = ([size_i, drop_i, act_i, \sigma_i^{size}, \sigma_1^{drop}]_i)_{i=1}^{H},$$

where H is the number of hidden layers, $size_i$ is the number of neurons in corresponding layer that is dense (fully connected) layer, $drop_i$ is the dropout rate (zero value represents no dropout), $act_i \in \{\text{relu, tanh, sigmoid, hard sigmoid, linear}\}$ is an activation function, and σ_i^{size} and σ_i^{drop} are strategy coefficients corresponding to size and dropout.

The *crossover* operator is implemented as a one-point crossover exchanging the whole blocks (layers).

There is a variety of *mutation* operators. Each time mutation is performed, one of them is chosen at random.

- mutateLayer—operates on one randomly selected layer. One of the following actions is performed:
  - changeLayerSize—Gaussian mutation is used, adapting strategy parameters σ^{size}, the final number is rounded (since size has to be an integer);
  - changeDropOut—the dropout rate is mutated by Gaussian mutation adapting strategy parameters σ^{drop};
  - changeActivation—a new activation function is selected randomly from the list of available activations;
- addLayer—new block is generated at random and inserted at random position;
- delLayer—deletes random block.

Note that the ES-like mutation comes in play only when the size of layer or dropout parameter is changed.

Fitness and selection are similar to the GA used for evolution of kernels. The classical cross validation error is used as fitness, and a tournament selection is utilized.

4. Data Set

We have used a real-world dataset [7,8]. It contains measurements of gas multi-sensor MOX array devices recording several gas pollutants concentrations. Data are labelled by a conventional air pollution monitoring station. The frequency of measurements is 1 hour, but data contain many missing values (due to sensor malfunctions). In this paper, we use data from 10 March 2004 to 4 April 2005. Lines with missing values are neglected. There are five sensor measurements as inputs, and we have chosen three output values, representing concentrations of CO, NO2 and NOx gases, as targets.

In the following text, we describe two experiments with different subsets of the data. The first experiment was rather a simple one, it should identify if our models are able to provide reliable results in a simplified scenario. For the first experiment, we have chosen 4 data samples per day for training, and the rest of the data is used for testing. It means we are predicting relatively small intervals between measurements.

The second experiment presents a more realistic scenario. The data were divided into five uniform intervals. In each of the sub-experiments, one of the intervals is chosen for training, while the remaining 80% of the data were left for testing. We have considered three different strategies for how to choose the training part. This task may thus be more difficult, since the testing may also be performed on distant parts of the data, meaning, for example, a different season of the year. Experts in the application area have suggested that, from their experience, the model build for the winter period will probably not work for summer.

Table 1 lists the size of individual datasets. All sets have five inputs and one target. All numbers are normalized to $\langle 0, 1 \rangle$.

Table 1. Overview of datasets' sizes.

Task	Train Set	Test Set	Task	Train Set	Test Set
sparse CO	1224	6120	CO i1-5	1469	5875
sparse NO2	1233	6160	NO2 i1-5	1479	5914
sparse NOx	1233	6163	NOx i1-5	1480	5916

5. Results

5.1. Experimental Setup

Models applied to sensor data include the above described RN, support vector regression (SVR), and DNN.

RN were used with Gaussian, product and sum kernels. Kernels were evolved using the GA described in Section 3.3. The set of available sub-kernels contained Gaussian, Multiquadric, Inverse-Multiquadric, and Sigmoid functions. The population size of GA was set up to 20 and a stop criterion to 300 generations. Elite with 2 individuals was used. Ten-fold cross validation was utilized for fitness.

SVR were trained using the Scikit-learn package [23]. We used SVR with linear, RBF, polynomial and sigmoid kernels [24]. Hyper-parameters were tuned by extensive grid search (10,000 different pairs of regularization parameter and kernel parameter).

Finally, the DNN were trained by RMSProp [37] for 500 epochs. The network architecture was found by the ES evolutionary algorithm described in Section 3.5. The ES was run with $n = 10$ and $m = 30$ for 100 generations. During the fitness evaluation, a five-fold cross validation was used.

Resulting errors are computed on the training and testing sets as mean square error multiplied by the factor of 100. Each experiment was run 10 times, evaluating average errors and their standard deviations.

5.2. Experimental Results

See Tables 2–9 and Figure 1 for results of the first and second experiment, respectively.

The first task contains four training values per day, the rest of measurements is left as a test set. Errors for RN with evolved Gaussian kernels, product and sum kernels are listed in Table 2. Note that product kernels perform the best on training data and under cross validation scenario, while they have results on the test data comparable to other kernels.

Table 3 lists training and testing errors on the first task for RN with product kernels, SVR with linear, RBF, polynomial, and sigmoid kernels, and DNN. Product kernels performed best both in terms of training and testing errors.

Figure 1 shows model performance on five splits on training and testing data. It can be seen that some periods are more suitable for training and general prediction than the others.

The second task is the more difficult one. The meassurements are split into five parts. Each time, one part represents a training data, the rest the test set. So, the predictions are made for seasons that were not included in the training data.

Tables 4 and 5 contain the resulting errors for RN with Gaussian and product kernels, respectively. The situation is the same as in the first experiment. Considering training errors, product kernels achieved lower errors in 10 cases from 15. In case of testing errors, products were winners only in seven cases. However, taking into account minimal values, the product kernels are best in all runs except one in case of training errors and except three in case of test errors. That means it is still possible to improve the search for product kernel. Note that the evolved product kernels are mainly composed of Gaussian and Inverse-Multiquadric functions.

The comparison of RN to SVR on the second task is listed in Tables 6 and 7. In terms of training errors, the RN with product kernels performed best (in nine cases), in terms of testing errors the RN with Gaussian kernels performed best (in seven cases). In general, RN gives better results than SVR.

Table 2. Task 1: Experiment on sparse measurements with RNs. Average errors and standard deviations are listed for networks with Gaussian, Product and Sum kernel. The lowest errors are highlighted.

Cross validation Errors						
	Gaussian Kernel		**Product Kernels**		**Sum Kernels**	
Task	E_{avg}	stddev	E_{avg}	stddev	E_{avg}	stddev
CO	0.152	0.000	**0.148**	0.002	0.152	0.003
NO2	0.429	0.003	**0.407**	0.009	0.434	0.012
NOx	0.227	0.000	**0.207**	0.006	0.229	0.005
Training Errors						
	Gaussian Kernel		**Product Kernels**		**Sum Kernels**	
Task	E_{avg}	stddev	E_{avg}	stddev	E_{avg}	stddev
CO	0.132	0.002	**0.123**	0.005	0.128	0.010
NO2	0.308	0.002	**0.277**	0.025	0.312	0.003
NOx	0.139	0.001	**0.135**	0.011	0.139	0.002
Testing Errors						
	Gaussian Kernel		**Product Kernels**		**Sum Kernels**	
Task	E_{avg}	stddev	E_{avg}	stddev	E_{avg}	stddev
CO	0.136	0.001	**0.134**	0.002	0.138	0.006
NO2	**0.334**	0.002	0.343	0.011	0.338	0.004
NOx	**0.158**	0.001	**0.158**	0.008	0.160	0.005

Table 3. Task 1: Comparison of RNs, SVR, and DNN. Average error values are listed. The lowest errors are highlighted.

Training Errors						
	RN Product	**SVR Linear**	**SVR RBF**	**SVR Poly**	**SVR Sigmoid**	**DNN**
CO	**0.123**	0.229	0.184	0.189	1.496	0.128
NO2	**0.277**	0.475	0.363	0.506	2.036	0.342
NOx	**0.135**	0.531	0.253	1.235	1.989	0.161
Testing Errors						
	RN Product	**SVR Linear**	**SVR RBF**	**SVR Poly**	**SVR Sigmoid**	**DNN**
CO	**0.134**	0.230	0.192	0.192	1.480	0.148
NO2	**0.343**	0.502	0.390	0.532	1.997	0.376
NOx	**0.158**	0.519	0.255	1.196	1.961	0.175

Finally, Tables 8 and 9 show a comparison of RN with product kernels and DNN on the second task. Product kernels performed better in the case of training error (in 12 cases), while DNN are better in the case of testing error (in 10 cases). The DNN showed a better generalization ability, while RN with product kernels are more prone to overfitting.

Since the dataset used is quite small, the evolved DNN had typically only one hidden layer consisting of 70 ReLU units, using dropout rate 0.3.

Table 4. Task 2: Comparison of training errors for RN trained on single epochs. Average values, standard deviations, minimum and maximum values are listed. The lower average and minimum errors are highlighted.

Training Errors								
	Gaussian Kernel				**Product Kernels**			
Task	E_{avg}	stddev	min	max	E_{avg}	stddev	min	max
CO-i1	**0.050**	0.000	0.050	0.050	0.051	0.002	**0.049**	0.055
CO-i2	0.049	0.000	0.049	0.049	**0.046**	0.002	**0.043**	0.050
CO-i3	**0.054**	0.000	**0.053**	0.054	0.056	0.003	0.054	0.065
CO-i4	**0.333**	0.001	0.332	0.334	0.347	0.016	**0.325**	0.378
CO-i5	0.133	0.000	0.132	0.133	**0.097**	0.018	**0.077**	0.142
NO2-i1	**0.096**	0.002	0.093	0.101	0.100	0.015	**0.091**	0.141
NO2-i2	0.133	0.001	0.131	0.134	**0.122**	0.014	**0.105**	0.148
NO2-i3	0.388	0.001	0.384	0.389	**0.314**	0.077	**0.214**	0.434
NO2-i4	0.297	0.002	0.295	0.299	**0.287**	0.012	**0.265**	0.307
NO2-i5	**0.375**	0.001	0.374	0.376	0.389	0.032	**0.330**	0.435
NOx-i1	0.018	0.000	0.018	0.018	**0.017**	0.001	**0.016**	0.020
NOx-i2	0.026	0.000	0.026	0.027	**0.025**	0.002	**0.021**	0.028
NOx-i3	0.156	0.001	0.154	0.158	**0.152**	0.019	**0.121**	0.184
NOx-i4	0.231	0.002	0.229	0.234	**0.230**	0.017	**0.203**	0.258
NOx-i5	0.106	0.023	0.087	0.132	**0.095**	0.011	**0.083**	0.122
	5	1		10	14			

Table 5. Task 2: Comparison of testing errors for RN trained on single epochs. Average values, standard deviations, minimum and maximum values are listed. The lower average and minimum errors are highlighted.

Testing Errors								
	Gaussian Kernel				**Product Kernels**			
Task	E_{avg}	stddev	min	max	E_{avg}	stddev	min	max
CO-i1	**0.210**	0.005	0.205	0.217	0.214	0.020	**0.192**	0.248
CO-i2	1.134	0.007	1.116	1.142	**0.878**	0.088	**0.709**	0.988
CO-i3	0.233	0.009	0.221	0.254	**0.228**	0.019	**0.197**	0.267
CO-i4	**0.326**	0.002	**0.323**	0.329	0.749	0.512	0.433	1.921
CO-i5	**0.296**	0.005	0.287	0.301	0.321	0.050	**0.204**	0.374
NO2-i1	**2.151**	0.052	2.096	2.267	2.263	0.540	**1.189**	2.997
NO2-i2	5.260	0.045	5.161	5.319	**3.928**	1.447	**2.661**	6.874
NO2-i3	**0.718**	0.004	**0.709**	0.721	1.033	0.218	0.764	1.351
NO2-i4	0.735	0.011	0.726	0.757	**0.734**	0.069	**0.669**	0.908
NO2-i5	**0.678**	0.024	**0.655**	0.735	0.913	0.183	0.709	1.302
NOx-i1	2.515	0.015	2.495	2.538	**2.409**	0.159	**2.093**	2.658
NOx-i2	3.113	0.019	3.081	3.139	**2.495**	0.068	**2.416**	2.592
NOx-i3	1.105	0.008	1.088	1.114	**0.956**	0.267	**0.730**	1.689
NOx-i4	**0.952**	0.008	0.941	0.970	1.256	0.520	**0.774**	2.610
NOx-i5	**0.730**	0.102	0.642	0.850	0.748	0.091	**0.544**	0.856
	8		3		7		12	

Table 6. Task 2: Comparison of average training errors for different models: Regularization Networks with Gaussian and Product kernels, SVR with RBF, linear, polynomial and sigmoid kernels. Lowest error for each dataset is highlighted.

Training Errors						
	RN Gaussian	**RN Product**	**SVR Linear**	**SVR RBF**	**SVR Polynomial**	**SVR Sigmoid**
CO-i1	**0.050**	0.051	0.131	0.160	0.120	1.284
CO-i2	0.049	**0.046**	0.122	0.110	0.122	0.835
CO-i3	**0.054**	0.056	0.165	0.141	0.123	1.700
CO-i4	**0.333**	0.347	0.372	0.343	0.352	2.131
CO-i5	0.133	**0.097**	0.186	0.207	0.204	1.215
NO2-i1	**0.096**	0.100	0.316	0.305	0.207	0.337
NO2-i2	0.133	**0.122**	0.366	0.367	0.306	0.386
NO2-i3	0.388	0.314	0.236	**0.199**	0.196	1.842
NO2-i4	0.297	**0.287**	0.360	0.312	0.336	3.253
NO2-i5	0.375	0.389	0.201	**0.171**	0.182	1.656
NOx-i1	0.018	**0.017**	0.175	0.173	0.174	0.899
NOx-i2	0.026	**0.025**	0.270	0.209	0.221	1.400
NOx-i3	0.156	**0.152**	0.471	0.394	0.412	1.513
NOx-i4	0.231	**0.230**	0.421	0.387	0.381	1.839
NOx-i5	0.106	**0.095**	0.493	0.471	0.468	2.411
	4	9	0	2	0	0

Table 7. Task 2: Comparison of average testing errors for different models: Regularization Networks with Gaussian and Product kernels, SVR with RBF, linear, polynomial and sigmoid kernels. Lowest error for each dataset is highlighted.

Testing Errors						
	RN Gaussian	**RN Product**	**SVR Linear**	**SVR RBF**	**SVR Polynomial**	**SVR Sigmoid**
CO-i1	**0.210**	0.214	0.340	0.280	0.285	1.533
CO-i2	1.134	0.878	0.614	**0.412**	0.621	1.753
CO-i3	0.233	**0.228**	0.314	0.408	0.377	1.427
CO-i4	**0.326**	0.749	1.127	0.692	0.535	1.375
CO-i5	**0.296**	0.321	0.348	0.207	0.198	1.568
NO2-i1	**2.151**	2.263	2.4643	2.404	2.401	2.636
NO2-i2	5.260	3.928	**2.118**	2.250	2.409	2.648
NO2-i3	**0.718**	1.033	1.3083	1.195	1.213	1.984
NO2-i4	0.735	**0.734**	1.978	2.565	1.912	2.531
NO2-i5	**0.678**	0.913	1.0773	1.047	0.967	2.129
NOx-i1	2.515	2.409	**1.062**	1.447	1.202	2.537
NOx-i2	3.113	2.495	**2.162**	1.838	1.387	2.428
NOx-i3	1.105	0.956	**0.594**	0.674	0.665	2.705
NOx-i4	0.952	1.256	0.8646	0.903	**0.778**	2.462
NOx-i5	**0.730**	0.748	1.6328	**0.730**	1.446	2.761
	7	2	4	3	1	0

Table 8. Task 2: Comparison of average training errors of RN with Product kernels and DNN. Average, standard deviation, minimal and maximal values are listed. The lower average value for each dataset is highlighted.

Training Errors								
	Product Kernels				**DNN**			
Task	E_{avg}	stddev	min	max	E_{avg}	stddev	min	max
CO-i1	**0.051**	0.002	0.049	0.055	0.059	0.002	0.055	0.064
CO-i2	**0.046**	0.002	0.043	0.050	0.053	0.003	0.049	0.057
CO-i3	**0.056**	0.003	0.054	0.065	0.070	0.007	0.063	0.085
CO-i4	0.347	0.016	0.325	0.378	**0.343**	0.013	0.330	0.365
CO-i5	**0.097**	0.018	0.077	0.142	0.132	0.005	0.125	0.144
NO2-i1	**0.100**	0.015	0.091	0.141	0.112	0.004	0.107	0.121
NO2-i2	**0.122**	0.014	0.105	0.148	0.186	0.013	0.174	0.214
NO2-i3	**0.314**	0.077	0.214	0.434	0.326	0.009	0.313	0.344
NO2-i4	**0.287**	0.012	0.265	0.307	0.380	0.011	0.368	0.410
NO2-i5	0.389	0.032	0.330	0.435	**0.349**	0.017	0.323	0.379
NOx-i1	**0.017**	0.001	0.016	0.020	0.020	0.001	0.018	0.021
NOx-i2	**0.025**	0.002	0.021	0.028	0.033	0.002	0.030	0.039
NOx-i3	0.152	0.019	0.121	0.184	**0.140**	0.014	0.127	0.166
NOx-i4	**0.230**	0.017	0.203	0.258	0.252	0.015	0.234	0.278
NOx-i5	**0.095**	0.011	0.083	0.122	0.114	0.009	0.102	0.130
	12			3				

Table 9. Task 2: Comparison of average testing errors of RN with Product kernels and DNN. Average, standard deviation, minimal and maximal values are listed. The lower average value for each dataset is highlighted.

Testing Errors								
	Product Kernels				**DNN**			
Task	E_{avg}	stddev	min	max	E_{avg}	stddev	min	max
CO-i1	**0.214**	0.020	0.192	0.248	0.229	0.026	0.195	0.267
CO-i2	0.878	0.088	0.709	0.988	**0.657**	0.024	0.631	0.694
CO-i3	**0.228**	0.019	0.197	0.267	0.256	0.045	0.199	0.349
CO-i4	0.749	0.512	0.433	1.921	**0.526**	0.108	0.308	0.701
CO-i5	0.321	0.050	0.204	0.374	**0.235**	0.025	0.199	0.277
NO2-i1	2.263	0.540	1.189	2.997	**1.506**	0.217	1.132	1.823
NO2-i2	3.928	1.447	2.661	6.874	**1.371**	0.048	1.242	1.415
NO2-i3	1.033	0.218	0.764	1.351	**0.660**	0.078	0.599	0.863
NO2-i4	**0.734**	0.069	0.669	0.908	0.782	0.043	0.711	0.856
NO2-i5	0.913	0.183	0.709	1.302	**0.730**	0.111	0.520	0.905
NOx-i1	2.409	0.159	2.093	2.658	**2.132**	0.086	2.021	2.284
NOx-i2	2.495	0.068	2.416	2.592	**1.599**	0.077	1.444	1.685
NOx-i3	**0.956**	0.267	0.730	1.689	1.339	0.242	1.106	1.955
NOx-i4	**1.256**	0.520	0.774	2.610	1.610	0.164	1.435	2.041
NOx-i5	0.748	0.091	0.544	0.856	**0.622**	0.075	0.521	0.726
	5		10					

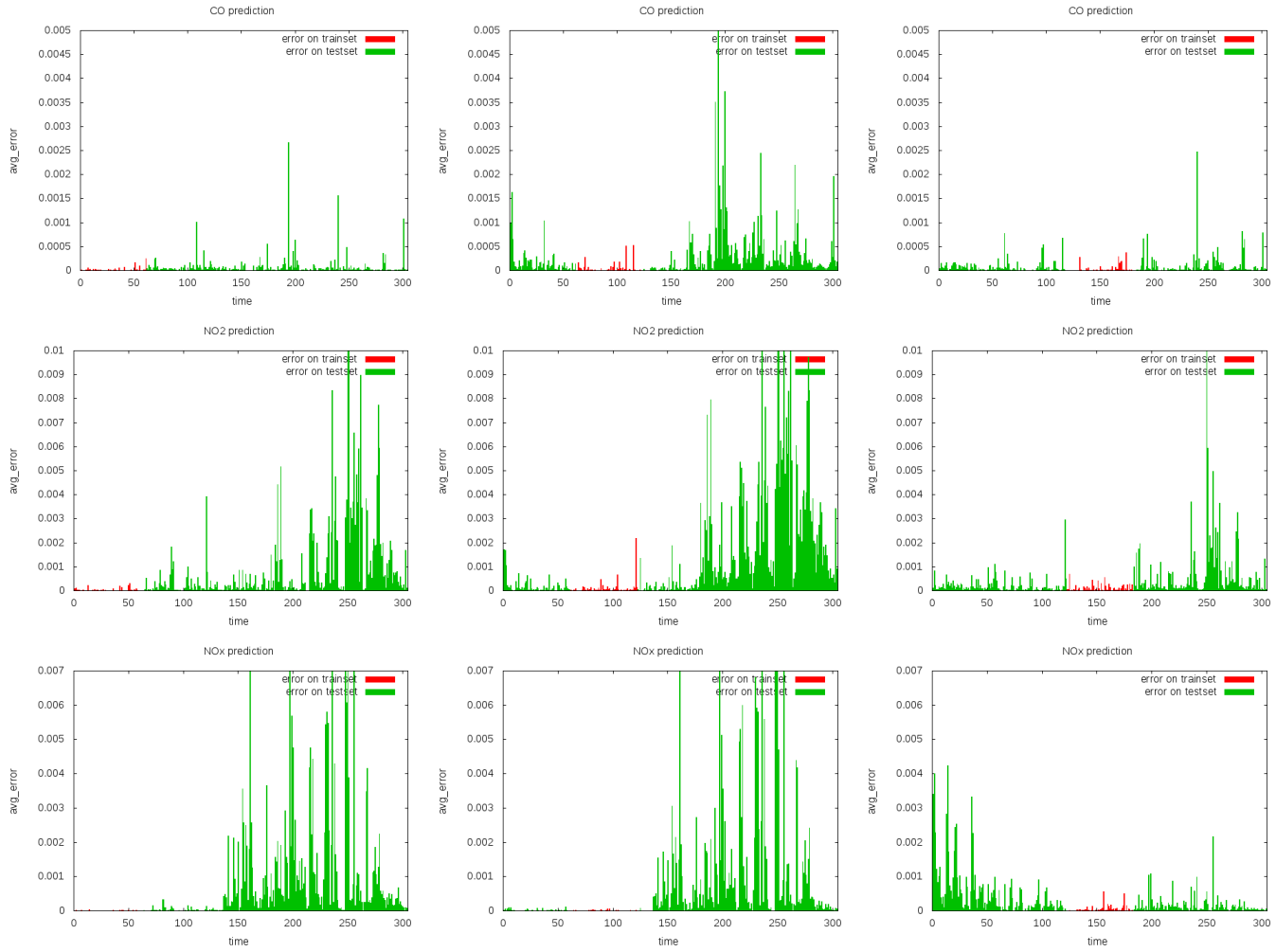

Figure 1. Task 2: Example of product kernel model performance on five splits on training and testing data. The rows correspond to predictions of CO, NO_2, and No_x concentrations. The three figures in one row correspond to three different training/testing splits.

6. Discussion

The goal of this study was to verify the usability of machine learning models in the area of multi-sensor fusion for air pollution modelling. We have identified several machine learning methods that have performed well in related areas of data mining and included several original improvements of those methods from our previous research. The methods proposed by the authors include:

- regularization networks with product and sum kernels;
- evolution of kernels based on a genetic algorithm;
- evolution of DNN architectures based on evolutionary strategies.

Comparisons to the very popular support vector machine regression models with Gaussian, polynomial and sigmoid kernels were also performed.

The model performance has been tested on real datasets described and used by other researchers before. The problem was formulated as a regression task from sensor measurements of air pollution. To ensure the best behaviour of studied methods, the extensive search for optimal hyper-parameters is performed by evolutionary optimization of cross validation error.

The results described above show that all models we have tested provided sound interpretations of the available data, they were able to perform multi-sensor fusion to predict the pollutant concentrations.

The insight into the temporal nature of the data can be obtained from the second experiment which compared results of models trained on 5 data splits. The first general observation is that short-term temporal dependencies were not a problem for the models that were able to reliably predict pollutant concentrations from sensor values. The cross validation results indicate that a well performing predictive model can be built using only 20 percent of the data. The second observation indicates a seasonal aspect of the data that is in correspondence with expert estimates. This is demonstrated by the generally poorer test performance of models trained on the fourth split of the data containing the winter months only. Thus, for the practical deployment of the models, some ensemble technique combining models trained on different seasons would be recommended.

When comparing the relative performances of the models, we can generalize the following outcomes.

The most difficult pollutant to predict is NO2, for which we get the highest errors.

Product kernels, our original model, can outperform standard classical RN models. In comparison to SVR, product kernels have been able to produce better results in the majority of the cases.

DNN performed comparatively to product kernels, achieving better generalizations in certain cases. Namely, on the first task, where values 'in between' are approximated, the product kernels are the winners, benefiting from their local character. On the second task, which is more difficult in general, product kernels performed better in terms of training errors, but in terms of testing errors, DNN provided better results in most cases. It suggests that DNN have better generalization capabilities, while product kernels are more prone to overfitting. Both RN and DNN are vital alternatives for the task of air pollution prediction.

Although our work presented in this paper has been performed on one (relatively representative) dataset, it demonstrates several general issues that are probably relevant to the whole application area. There seems to be a relevant problem with missing data which was not addressed in this work. Since simple mean inputting does not seem to be sufficient, we recommend employing some more sophisticated methods, such as clustering or semi-supervised methods. In general, better data preprocessing and cleaning should improve the results. Next, interesting statistical properties of the data that have been identified, such as relevant differences between summer and winter concentrations, should be studied further in accordance with domain experts, which can probably lead to a creation of several specialized models, or even ensemble methods.

Author Contributions: Both authors contributed to the manuscript equally. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Czech Grant Agency grant 18-23827S and institutional support of the Institute of Computer Science RVO 67985807.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used in this work are also available at UCI ML Repository, see https://archive.ics.uci.edu/ml/datasets/Air+quality (accessed on 1 October 2021).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
2. Lecun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
3. Moraru, A.; Pesko, M.; Porcius, M.; Fortuna, C.; Mladenic, D. Using machine learning on sensor data. In Proceedings of the ITI 2010, 32nd International Conference on Information Technology Interfaces, Dubrovnik, Croatia, 21–24 June 2010; pp. 573–578.

4. Smith, M.K.; Castello, C.C.; New, J.R. Machine Learning Techniques Applied to Sensor Data Correction in Building Technologies. In Proceedings of the 2013 12th International Conference on Machine Learning and Applications, Miami, FL, USA, 4–7 December 2013; Volume 1, pp. 305–308. [CrossRef]
5. Ahn, J.; Shin, D.; Kim, K.; Yang, J. Indoor Air Quality Analysis Using Deep Learning with Sensor Data. *Sensors* **2017**, *17*, 2476. [CrossRef] [PubMed]
6. Psuj, G. Multi-Sensor Data Integration Using Deep Learning for Characterization of Defects in Steel Elements. *Sensors* **2018**, *18*, 292. [CrossRef] [PubMed]
7. Vito, S.D.; Massera, E.; Piga, M.; Martinotto, L.; Francia, G.D. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sens. Actuators B Chem.* **2008**, *129*, 750–757. [CrossRef]
8. De Vito, S.; Fattoruso, G.; Pardo, M.; Tortorella, F.; Di Francia, G. Semi-Supervised Learning Techniques in Artificial Olfaction: A Novel Approach to Classification Problems and Drift Counteraction. *Sensors* **2012**, *12*, 3215–3224. [CrossRef]
9. Vidnerová, P.; Neruda, R. Evolution Strategies for Deep Neural Network Models Design. In Proceedings of the ITAT 2017: Information Technologies—Applications and Theory, CEUR Workshop Proceedings, Oravská Lesná, Slovakia, 22–29 September 2017; pp. 159–166.
10. Vert, J.P.; Tsuda, K.; Scholkopf, B. A primer on kernel methods. In *Kernel Methods in Computational Biology*; MIT Press: Cambridge, MA, USA, 2004; pp. 35–70.
11. Cucker, F.; Smale, S. On the Mathematical Foundations of Learning. *Bull. Am. Math. Soc.* **2001**, *39*, 1–49. [CrossRef]
12. Girosi, F. *An Equivalence between Sparse Approximation and Support Vector Machines*; Technical Report; Massachutesetts Institute of Technology: Cambridge, MA, USA, 1997. A.I. Memo No. 1606.
13. Girosi, F.; Jones, M.; Poggio, T. Regularization Theory and Neural Networks Architectures. *Neural Comput.* **1995**, *2*, 219–269. [CrossRef]
14. Poggio, T.; Smale, S. The Mathematics of Learning: Dealing with Data. *Not. AMS* **2003**, *50*, 536–544.
15. Kudová, P. Learning with Kernel Based Regularization Networks. In *Information Technologies—Applications and Theory. Košice*; Prírodovedecká fakulta Univerzity Pavla Jozefa Šafárika: Košice, Slovak Republic, 2005; pp. 83–92.
16. Kainen, P.C.; Kůrková, V.; Sanguineti, M. Complexity of Gaussian radial-basis networks approximating smooth functions. *J. Complex.* **2009**, *25*, 63–74. [CrossRef]
17. Kůrková, V.; Kainen, P.C. Kernel networks with fixed and variable widths. In Proceedings of the ICANNGA 2011, Ljubljana, Slovenia, 14–16 April 2011; LNCS 6593; Springer: Berlin/Heidelberg, Geremany, 2011; pp. 12–21.
18. Kůrková, V. Some comparisons of networks with radial and kernel units. In *Neural Networks: Brain-Inspired Computing and Machine Learning Research*; Springer: Berlin/Heidelberg, Germany, 2012; LNCS 7553; pp. 17–24.
19. Kůrková, V.; Kainen, P.C. Comparing fixed and variable-width Gaussian networks. *Neural Netw.* **2014**, *57*, 23–28. [CrossRef] [PubMed]
20. Gönen, M.; Alpaydin, E. Multiple Kernel Learning Algorithms. *J. Mach. Learn. Res.* **2011**, *12*, 2211–2268.
21. Cortes, C. Invited Talk: Can Learning Kernels Help Performance? In Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, QC, Canada, 14–18 June 2009; ACM: New York, NY, USA, 2009. [CrossRef]
22. Kloft, M.; Blanchard, G. The Local Rademacher Complexity of Lp-Norm Multiple Kernel Learning. In *Advances in Neural Information Processing Systems 24*; Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K., Eds.; Curran Associates, Inc.: New York, NY, USA, 2011; pp. 2438–2446.
23. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
24. Smola, A.; Vapnik, V. Support vector regression machines. *Adv. Neural Inf. Process. Syst.* **1997**, *9*, 155–161.
25. Vidnerova, P.; Neruda, R. Evolving keras architectures for sensor data analysis. In Proceedings of the 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), Sofia, Bulgaria, 6–10 September 2017; pp. 109–112. [CrossRef]
26. Poggio, T.; Girosi, F. *A Theory of Networks for Approximation and Learning*; Technical Report; Massachusetts INST of TECH Cambridge Artificial Intelligence LAB: Cambridge, MA, USA, 1989. A.I. Memo No.1140
27. Boughorbel, S.; Tarel, J.P.; Boujemaa, N. Conditionally Positive Definite Kernels for SVM Based Image Recognition. In Proceedings of the 2005 IEEE International Conference on Multimedia and Expo, Amsterdam, The Netherlands, 6–8 July 2005; pp. 113–116.
28. Vidnerová, R.N.P. Product Multi-kernels for Sensor Data Analysis. In *Artificial Intelligence and Soft Computing*; Rutkowski, L., M., Scherer, R., Tadeusiewicz, R., Zadeh, L., Zurada, J., Eds.; ICAISC 2015; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2015; Volume 9119.
29. Vidnerová, P.; Neruda, R. Sensor Data Air Pollution Prediction by Kernel Models. In Proceedings of the 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, Columbia, 16–19 May 2016; pp. 666–673. [CrossRef]
30. Vidnerová, P.; Neruda, R. *Evolutionary Learning of Regularization Networks with Product Kernel Units*; IEEE: Piscataway, NJ, USA, 2011; pp. 638–643.
31. Vidnerová, P.; Neruda, R. *Evolving Sum and Composite Kernel Functions for Regularization Networks*; Lecture Notes in Computer Science; ICANNGA (1); Dobnikar, A., Lotric, U., Ster, B., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6593, pp. 180–189.
32. Aronszajn, N. Theory of Reproducing Kernels. *Trans. AMS* **1950**, *68*, 337–404. [CrossRef]

33. Kudová, P.; Šámalová, T. Sum and Product Kernel Regularization Networks. *Artificial Intelligence and Soft Computing*; Rutkowski, L., Tadeusiewicz, R., Zadeh, L., Zurada, J., Eds.; Lecture Notes in Artificial Intelligence; Springer: Berlin/Heidelberg, Germany, 2006; pp. 56–65.
34. Mitchell, M. *An Introduction to Genetic Algorithms*; MIT Press: Cambridge, MA, USA, 1996.
35. Stone, M. Cross-validation: A review. *Math. Oper. Stat.* **1978**, *9*, 127–140.
36. Beyer, H.G.; Schwefel, H.P. Evolutionary Strategies: A Comprehensive Introduction. *Nat. Comput.* **2002**, *1*, 3–52. [CrossRef]
37. Chollet, F. Keras. 2015. Available online: https://github.com/fchollet/keras (accessed on 1 October 2021).