Vamplew, P., et al. (2018). "Non-functional regression: A new challenge for neural networks." Neurocomputing 314: 326-335.

Which has been published in final form at:
https://doi.org/10.1016/j.neucom.2018.06.066

Non-Functional Regression: A New Challenge for Neural Networks

Peter Vamplew, Richard Dazeley, Cameron Foale and Tanveer Choudhury

Federation Learning Agents Group, School of Engineering and Information Technology, Federation University Australia, Ballarat, Victoria, Australia

Abstract

This work identifies an important, previously unaddressed issue for regression based on neural networks – learning to accurately approximate problems where the output is not a function of the input (i.e. where the number of outputs required varies across input space). Such non-functional regression problems arise in a number of applications, and can not be adequately handled by existing neural network algorithms. To demonstrate the benefits possible from directly addressing non-functional regression, this paper proposes the first neural algorithm to do so – an extension of the Resource Allocating Network (RAN) which adds additional output neurons to the network structure during training. This new algorithm, called the Resource Allocating Network with Varying Output Cardinality (RANVOC), is demonstrated to be capable of learning to perform non-functional regression, on both artificially constructed data and also on the real-world task of specifying parameter settings for a plasma-spray process. Importantly RANVOC is shown to outperform not just the original RAN algorithm, but also the best possible error rates achievable by any functional form of regression.

Keywords: non-functional relationships, regression, Resource Allocating Network, radial basis functions

1. Introduction

While neural networks offer a flexible and powerful approach to regression (see for example [1, 2, 3]), conventional neural net architectures can not be successfully applied to problems where the output is not a function of the input. However the ability to learn mappings from input to output variable(s) which are non-functional in nature may be required in some applications. As a motivating example, consider the work of Choudhury et al. [4]. Data gathered from experimental evaluation of a plasma spray process was used to train a multi-layer network to predict in-flight particle characteristics of the spray given the values of the power and injection parameters of the device. The mapping from device parameters to in-flight characteristics is functional in nature – given particular parameter settings, a specific set of spray characteristics will be observed (subject to a certain amount of noise). However the reverse mapping is more useful – the user would like to specify the desired spray characteristics and be informed which device settings are required. This mapping may not be in the form of a function, as one set of spray characteristics might in fact be attainable using more than one set of parameter settings. Furthermore these different parameter settings may result in variations in other characteristics of interest to the user but not modelled by the regression system, such as power usage or paint consumption. So in practice a regression system should ideally list all suitable sets of values for the device parameters, or at least a representative sample of these settings, to allow the user to select the configuration which is most appropriate for the task at hand. In addition, certain combinations of spray characteristics may not be achievable under any device settings – it would be desirable for the system to be able to indicate the absence of any valid output when presented with these characteristics as input.

Unfortunately, while standard neural networks can carry out function approximation, they perform inadequately when the regression task requires a non-functional mapping from input to output. For example, a neural net can learn to map an input x to an output $y = x^2$, as for any input value there is a single output value. However the network can not adequately learn the inverse relationship, where given y as an input it returns $x = \sqrt{y}$ as in this case for any value of $y > 0$ there will be two possible values of x. A conventional network trained on a data-set derived from this function, will tend to produce the average of the outputs for the training examples which share the same input. For example, if trained on two cases, one which maps the input $y = 4$ to $x = +2$ and the other which maps $y = 4$ to $x = -2$, the error-reduction process in the training algorithm will learn to output $x = 0$, which is clearly incorrect.

This problem could be avoided by creating the network with two output nodes, and modifying the training data by merging the two cases into a single case with -2 and 2 as the outputs for the input 4. However more generally

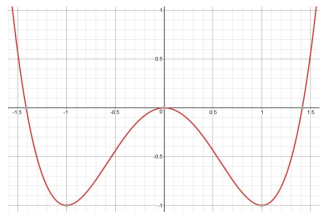

Figure 1: The graph of $y = x^4 - 2x^2$

the number of outputs may depend on the input. For example consider the function in Figure 1. Depending on the value of y there may be zero, two, three or four values for x. Without the ability to explicitly represent a varying number of outputs, no network can adequately represent this inverse mapping from y to x. For real-world data, the number of outputs required at any point in input space may not be known in advance, particularly if the network is learning online, as with streaming data or applications such as reinforcement learning. In addition, there may be regions of input space for which there are no valid output values. Therefore, the ability to adapt the number of outputs dynamically is a requirement for algorithms to operate correctly in this context.

To our knowledge the issue of learning non-functional mappings with varying cardinality has not previously been addressed in the neural network literature. Section 2 of this paper provides a formal definition of the non-functional regression problem. It also presents and discusses the first algorithm designed specifically for non-functional neural regression, the Resource Allocating Network with Variable Output Cardinality (RANVOC). RANVOC is a variant of the Resource Allocating Network (RAN) algorithm [5], designed to demonstrate how an existing neural regression algorithm can be extended to support non-functional regression. Section 3 provides an empirical comparison of the RAN and RANVOC approaches on a suite of artificial benchmark datasets designed to provide insight into the performance of each algorithm under different dataset characteristics. It also compares RANVOC's performance against the theoretical performance limits for any functional regression approach. Section 4 then evaluates the performance of RANVOC on the real-world plasma-spray process dataset. Section 5 provides a summary of the paper along with suggestions for future work.

2. Addressing Non-Functional Regression

2.1. Problem definition

Neural approaches to regression generally assume that the task is to learn an approximate mapping from an input vector I to an output vector O: $I \mapsto O$. This inherently assumes that the output is a function of the input; $O = f(I)$. This paper addresses the more general regression task of learning a mapping from an input I to a (possibly empty) set of outputs $S = \{O_1, .., O_n\}$. It is assumed that the dimensionality of both I and O are fixed (that is, $|O_i| = |O_j|$ for all i and j). However the cardinality of set S may vary depending on the value of I – that is $|S| = f(I)$.

2.2. Designing a neural algorithm for non-functional regression

This section presents the first neural learning algorithm designed specifically for non-functional regression. This algorithm was developed by extending an existing neural regression algorithm to support non-functional regression. This approach has two benefits. First it allows for an empirical comparison of the original and extended algorithm on datasets involving either functional or non-functional relationships between inputs and outputs. By using similar underlying algorithms, any differences observed in performance can clearly be attributed to the modifications made in order to support non-functional regression. The second advantage of this approach is that the methods developed for supporting varying output cardinality may be suitable for use in adapting other neural regression algorithms in the future.

The novel neural network algorithm developed in this work was required to be able to:

- produce a varying number of outputs depending on the value of the input variables (including the capacity for reporting no output where appropriate)
- learn from the training data the maximum number of outputs required in any region in input space
- perform in a fashion similar to a conventional network if the data presented to it is in fact functional
- be suited to online learning (one intended area of application is multi-objective reinforcement learning [6, 7], where extending methods such as the Pareto set algorithm [8] to more complex problems will require

an online regression algorithm capable of mapping inputs to a varying number of Pareto-optimal output vectors).

As the maximum number of outputs can not be predetermined due to the online learning context, a constructive algorithm is favoured over a fixed network topology [9]. As the number of outputs required varies over input space, the network must also determine which output nodes are relevant for the current input. We anticipate that the relevance of an output node must be able to vary within a constrained range of input space (consider the rapid change required from four to two active outputs as y changes from negative to positive in Figure 1). Therefore a constructive network based on locally responsive units such as radial-basis functions (RBFs) is likely to be more suitable than a network based on more globally responsive units.

The Resource Allocating Network (RAN) proposed by Platt [5] is one of the most widely studied locally-responsive constructive algorithms for online learning. While its efficiency has been surpassed by more recent algorithms such as Huang et al. [10, 11], Vuković and Miljković [12], it is a relatively straightforward algorithm, making it well-suited for this initial demonstration of the techniques required to support variations in output cardinality. The RAN algorithm and its associated notation are presented in Algorithm 1 and Table 1[1]. RAN fits the training data using structural changes to the network (adding new RBF units to its hidden layer) to address large errors, and gradient descent over the numeric parameters of the current structure to address small errors.

2.3. The Resource Allocating Network with Varying Output Cardinality algorithm (RANVOC)

Extending the RAN algorithm to handle non-functional mappings with varying output cardinality requires two major changes. The first alteration is that during training the algorithm must have a means for deciding when it is appropriate to add a new output node to the network, as well as a mechanism for actually adding such a node. The second major change is that because the cardinality of the output set may vary, the algorithm must be able to determine for any given input which output nodes are actually relevant for that input – only the results of these nodes should be included in the output

[1] Some modifications have been made to the notation and presentation of [5] for consistency and clarity of presentation of the RANVOC algorithm later in this paper.

Algorithm 1 Platt's original Resource Allocating Network (RAN) training algorithm

$\delta = \delta_{max}$
$\gamma = T_0$ (from the first input-output pair)
for each presentation of an input-output pair (I, T) **do**
    evaluate hidden nodes $H_j = e^{\|c_j - I\|^2 / {w_j}^2}$
    evaluate output of network $O = \sum_j h_j H_j(I) + \gamma$
    compute error $E = T - O$
    find distance to nearest center $d = min_j \parallel c_j - I \parallel$
    if $\parallel E \parallel \geq \epsilon$ and $d \geq \delta$ **then**
        allocate new unit $c_{new} = I, h_{new} = E$
        if this is the first new unit to be allocated **then**
            width of new unit $= \kappa\delta$
        else
            width of new unit $= \kappa d$
        end if
    else
        perform gradient descent on γ, h_j, c_{jk}
    end if
    if $\delta > \delta_{min}$ **then**
        $\delta = \delta * exp(-1/\tau)$
    end if
end for

Table 1: Notation and parameters for the Resource Allocating Network (RAN)

Notation	
c_j	centre of hidden unit j
w_j	width of hidden unit j
$H_j(I)$	the output of hidden unit j for input pattern I
O	the output of the RAN for input pattern I
h_j	weight from hidden unit j to output unit
γ	offset for output unit
Parameters	
α	learning rate for gradient descent
ϵ	threshold error level for adding a new hidden unit
δ_{max}	maximum width of hidden units
δ_{min}	minimum width of hidden units
τ	decay applied to the width of new hidden units
κ	constant term used in calculating width

set. Figure 2 illustrates the desired behaviour. The example training data has regions of input space where two outputs exist, so RANVOC must add at least one extra output node. Each output node has input regions where it is relevant (indicated by black points). Outside of these regions the node's output can still be evaluated, but will not be included in the output set S.[2]

These two aspects of the algorithm interact during the training phase, and so it may be clearest to first consider how relevance testing is applied on the final network after training, as shown in Algorithm 2. For each output node, the activation of all hidden units connected to that output node is summed, weighted by the $r_{j,k}$ relevance weights, and the output node is regarded as relevant and included in the output set S if this weighted sum exceeds the threshold ρ.

The same approach is also used to distinguish between relevant and irrelevant output nodes during the training phase of RANVOC. This necessitates one small further change from the original RAN algorithm. RAN initialises the network from a first input-output pair by simply setting the offset weights

[2]Note that outside of its region of relevance, an output unit will receive little activation from any hidden nodes, and therefore its output will tend towards a constant level determined by the value of its offset weights γ as shown in the lower half of Figure 2.

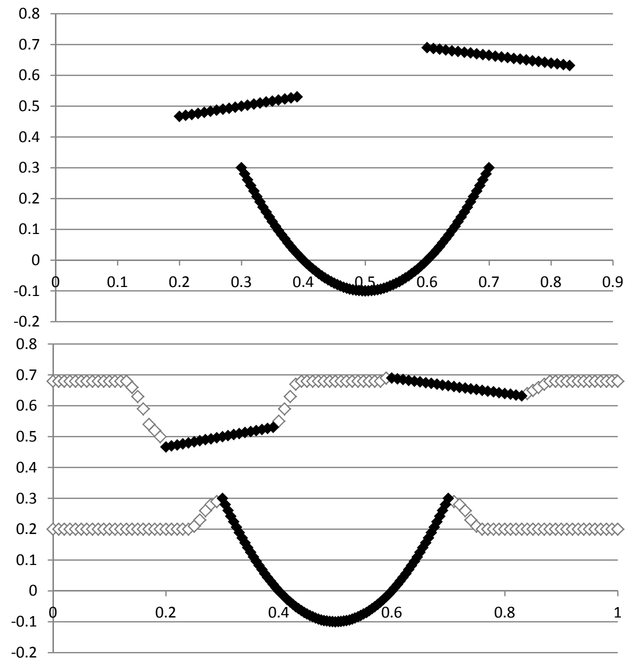

Figure 2: An example dataset requiring non-functional regression (top), and an idealised RANVOC response (bottom). The horizontal axis is the input value, and the vertical axis is the output value. In the RANVOC response dark points indicate the output value of an output unit when it is relevant to the current input, and light nodes indicate the output value when the unit is not relevant.

Algorithm 2 RANVOC run-time algorithm

input: input pattern I
output set $S = \emptyset$
for each output node k **do**
    evaluate hidden nodes $H_j = e^{\|c_j - I\|^2 / {w_j}^2}$
    evaluate output of unit $O_k = \sum_j a_{j,k} h_{j,k} H_j(I) + \gamma_k$
    $R_k = \sum_j r_{j,k} H_j(I)$
    if $R_k > \rho$ **then**
        $S = S \cup \{O_k\}$
    end if
end for
output: S

Table 2: Notation and parameters for the Resource Allocating Network with Varying Output Cardinality (RANVOC)

Notation	
J	number of hidden units
K	number of output units
c_j	centre of hidden unit j
w_j	width of hidden unit j
H_j	the output of hidden unit j
O_k	the output of output unit k
R_k	the relevance value of output unit k
$h_{j,k}$	value weight from hidden unit j to output unit k
$a_{j,k}$	binary feature indicating if hidden unit j is connected to output unit k
$r_{j,k}$	relevance weight from hidden unit j to output unit k
γ_k	offset for output unit k
Parameters	
α	learning rate for gradient descent
β	decay rate for relevance weights
ϵ_{low}	threshold error level for adding a new hidden unit
ϵ_{high}	threshold error level for adding a new output unit
δ_{max}	maximum width of hidden units
δ_{min}	minimum width of hidden units
τ	decay applied to the width of new hidden units
κ	constant term used in calculating width of hidden units
ρ	the threshold summed relevance-weighted activation of hidden units required for an output unit to be active for the current input

Algorithm 3 RANVOC training algorithm

$\delta = \delta_{max}$
$\gamma_0 = T_0$ (from the first input-output pair)
$c_0 = I_0; w_0 = \delta; a_{0,0} = 1; r_{0,0} = 1; h_{0,0} = 0$
for each presentation of an input-output pair (I, T) **do**
  active output set $S =$ Algorithm $2(I)$
  $handled = false$
  if $S \neq \emptyset$ **then**
    $k^* = \text{argmin}_k \parallel (T - O_k)] \parallel$ for $k \in S$
    $E = T - O_{k^*}$
    for each $k \in S$ where $k \neq k^*$ **do**
      $j_k = \text{argmin}_j \parallel c_j - I] \parallel$ for $a_{j,k} = 1$
      decrement $r_{j_k,k}$ by β
    end for
    if $\parallel E \parallel < \epsilon_{high}$ **then**
      $j_{k^*} = \text{argmin}_j \parallel c_j - I] \parallel$ for $a_{j,k^*} = 1$
      $d = \parallel c_{j_{k^*}} - I \parallel$
      if $\parallel E \parallel < \epsilon_{low}$ or $d < \delta$ **then**
        $r_{j_{k^*},k^*} = 1$
        perform gradient descent on $\gamma_{k^*}, h_{j,k^*}, c_{j,k^*}$ for all j where $a_{j,k^*} = 1$
      else
        allocate new hidden unit $c_{new} = I, w_{new} = \kappa d$
        $h_{new,k^*} = E, a_{new,k^*} = 1, r_{new,k^*} = 1$
      end if
      $handled = true$
    end if
  end if
  if $\neg handled$ and there is any output node $k \notin S$ **then**
    $k^* = \text{argmin}_k \parallel (T - O_k)] \parallel$ for $k \notin S$
    $E = T - O_{k^*}$
    $j_{k^*} = \text{argmin}_j \parallel c_j - I] \parallel$ for $a_{j,k^*} = 1$
    $d = \parallel c_{j_{k^*}} - I \parallel$
    if $\parallel E \parallel < \epsilon_{low}$ or $d < \delta$ **then**
      $r_{j_{k^*},k^*} = 1$
      perform gradient descent on $\gamma_{k^*}, h_{j,k^*}, c_{j,k^*}$ for all j where $a_{j,k^*} = 1$
    else
      allocate new hidden unit $c_{new} = I, w_{new} = \kappa d$
      $h_{new,k^*} = E, a_{new,k^*} = 1, r_{new,k^*} = 1$
    end if
    $handled = true$
  end if
  if $\neg handled$ **then**
    allocate new hidden unit $c_{new} = I, w_{new} = \kappa\delta$
    allocate new output unit k^*, $\gamma_{k^*} = T$
    $h_{new,k^*} = 0, a_{new,k^*} = 1, r_{new,k^*} = 1$
  end if
  if $\delta > \delta_{min}$ **then**
    $\delta = \delta * exp(-1/\tau)$
  end if
end for

of the output node without adding a hidden unit (line 3 of Algorithm 1). In RANVOC this would result in the output node producing the correct output value for that training point, but not being regarded as relevant. Therefore an initial hidden unit is added to the network to ensure the first output node is regarded as relevant if presented with this same input (lines 2 and 3 of Algorithm 3).

The approach used for deciding whether to add a new output unit uses an error-thresholding mechanism similar to that which RAN uses to determine whether to add a new hidden unit. This is carried out in conjunction with relevance testing in order to minimise the number of output units added to the network. First the algorithm identifies which output units are relevant for the current input, and finds the relevant unit with minimum error with respect to the current target (lines 5-9). If this unit's error is below ϵ_{high}, then it is trained in a fashion akin to that of RAN (lines 14-25). If the error exceeds this threshold then this process is repeated using the output units which are not regarded as relevant (lines 27-39). Only if no existing output units satisfy the ϵ_{high} threshold, does the algorithm add a new output node (lines 41-45).

As well as carrying out the operations required to train the network's output values, Algorithm 3 must also modify the the $r_{j,k}$ weights to ensure the relevance-testing mechanism works. The approach used is to initialise $r_{j,k}$ to 1 when hidden unit j is first connected to output unit k (lines 3, 22, 37 and 44). This ensures that $R_k > \rho$ when it next encounters the same, or very similar, input. However because of the online nature of the training process it is possible that an output which is initially seen as relevant for an input I may later become irrelevant – if the $r_{j,k}$ values remain fixed then output k will always be included in S for this input in the future, harming the network's performance by producing a redundant or inappropriate output. Therefore whenever an output is deemed relevant for the current input, but not selected as the closest match to the current target, the $r_{j,k}$ weight for the most highly activated connected hidden node is decremented (lines 10-13). Over time if an output node is repeatedly in S but not the closest match to the target, its relevance weight will decay until its R_k value no longer exceeds ρ, meaning it will no longer be regarded as relevant. Whenever a unit is identified as the best match for the current target, the $r_{j,k}$ weight is reset to 1, to ensure that the algorithm does not incorrectly ignore genuinely relevant outputs (lines 18 and 33).

Two important features of the RANVOC training algorithm are worth noting. First, if the dataset characteristics are such that the ϵ_{high} threshold is not exceeded then the algorithm is identical to the original RAN algorithm, other than the variation in initial structure noted earlier. Therefore RANVOC should be applicable in a case where it is not known in advance whether the data's input-output mapping is functional or not. Second, the nature of the constructive process ensures that each hidden unit is connected to only a single output node which ensures that the training of one output node can not interfere with the prior learning of the other outputs[3].

3. Evaluation on artificial datasets

As the problem of learning non-functional relationships has not been previously explored in the literature, there are no existing benchmark datasets which can be used to evaluate the RANVOC algorithm. In addition the standard metrics for evaluating regression task performance such as root mean squared error are not directly applicable to data with a varying cardinality of outputs. Therefore this section of the paper will describe the design of the benchmark datasets and experimental methods which have been utilised in this study.

3.1. Benchmark datasets

Six datasets have been developed for this evaluation, differing in the dimensionality of their input and output spaces, and in the extent to which the cardinality of the output varies across the input space. All datasets have inputs and outputs scaled to approximately the same range (0..1) to simplify selection of distance-based parameters, and to facilitate comparison of error values between the different datasets. Each dataset is defined in terms of one or more underlying 'generator' functions which produce (I, O) training pairs. Some datasets have only a single generator (used to assess RANVOC's ability to approximate functional relationships), while others have multiple generators, which are active over different ranges of input space and map to different output ranges. Each dataset was generated by sampling input-output pairs

[3]During development of the RANVOC algorithm we experimented with a variant which supported sharing of hidden units between multiple output nodes. While this produced a saving in storage space for hidden units, it hampered the accuracy of learning as performing gradient descent on the centres of hidden units connected to one output node effected the performance of any other output nodes sharing some of those hidden units.

from each generator. Where an input sampled in this manner also lay within the bounds of another generator(s), the output for that generator was also evaluated and included in the data set. That is, each instance within the dataset consists of an input vector I, along with a set $T = \{O_1, ..., O_n\}$ of one or more target output vectors, where O_1 is the output vector produced by the same generator from which I was sampled, and was used as the target output during training. The additional output vectors $O_2, ..., O_n$ were not used during training, but were used for evaluation purposes as detailed in Section 3.2. For each dataset 100 additional input points were randomly sampled which lay outside of the range of all generators – these will be referred to as 'null points' and are also used during evaluation to assess the ability of the network to correctly indicate that no valid output exists when presented with one of these points as input.

The choice of the number of generators and the degree to which their input and output ranges overlap impacts on the extent to which each dataset can be accurately approximated by functional regression. To quantify this, we propose the *non-functional index* metric (NFI). Consider a dataset D consisting of n data-instances $d = (I, T = \{O_1, ..., O_n\})$. The NFI of each individual data-instance can be calculated as the maximum distance between any pair of output vectors within that instance's set T of output vectors, as shown in Equation 1. For data which is strictly functional, there will be only a single vector in T and so $NFI(d)$ will equal zero. The NFI for the complete dataset D can be calculated as the mean NFI of the individual data-instances, as in Equation 2.

$$NFI(d) = \max_{i,j:1..n} \{\| O_i - O_j \|\} \tag{1}$$

$$NFI(D) = \frac{\sum_{k=1}^{n} NFI(d_k)}{n} \tag{2}$$

The details of the datasets, including their NFI values, are summarised in Table 3. The dataset files used in this study are available for download from researchgate.com/url-anonymised-for-review-purposes.

The first two datasets, Quartic-F and Quartic-NF, are derived from the quartic equation in Figure 1. Quartic-F is based on the mapping $x \mapsto y$ and is defined by a single generator, so all non-null data points have a fixed output cardinality of 1. Quartic-NF is based on the mapping $y \mapsto x$ and so its non-null data points vary in cardinality from 2 to 4.

Table 3: Details of the datasets used in this study

Name	Input dimensions	Output dimensions	Output cardinality	Instances	NFI
Quartic-F	1	1	1	300	0
Quartic-NF	1	1	2-4	300	0.86
Circles	1	1	2-4	300	1.45
Ellipsoid1D-F	1	1	1	300	0
Ellipsoid1D-NF	1	1	1-3	300	0.38
Ellipsoid2D-NF	2	2	1-5	1000	0.40

The dataset Circles is defined by generators representing two concentric circles - the x coordinate of each point on a circle's perimeter is used as an input and the two corresponding y points on the circle as the output. This is an example of a dataset which can not be handled via a functional regression method, as $x \neq f(y)$ and $y \neq f(x)$.

The remaining datasets were all based on generators defined by filled ellipsoids embedded in different dimensionalities of input space. Each ellipsoid maps to an output vector which is defined as a randomly generated combination of linear and quadratic functions of the input variables. Ellipsoid1D-F and Ellipsoid1D-NF uses 1-dimensional input and output space which facilitates visualisation of the performance of the networks when trained on this data. In Ellipsoid1D-F the generators do not overlap in input space, meaning the output is a piecewise non-linear function of the input, with some gaps. For Ellipsoid1D-NF the generators do overlap so the outputs vary in cardinality between 1 and 3. Ellipsoid2D-NF extends this approach to two-dimensional input and output vectors to demonstrate that RANVOC is not restricted to scalar inputs and outputs.

3.2. Experimental methodology and evaluation metrics

Both the RAN and RANVOC algorithms were applied to each data-set using 10-fold crossfold validation. Training was carried out for 100 epochs. The networks were trained using single input-output pairs sampled from the set of active generators. That is to say, the network was never shown more than one output for a specific input instance – we believe this to be an appropriate replication of how these networks would be trained on actual, non-simulated data. After training was completed, the network's performance was evaluated on each data input in both the training and test folds.

The set of outputs produced by the network (possibly empty) was compared against the complete set of output targets in the dataset for that input. In order to allow for potential mismatches between the expected and actual number of outputs, the error metric described in Algorithm 4 was used. By measuring the distance between each target and the closest matching output, and vice-versa, an algorithm is penalised should it produce either too few or too many outputs, or if it produces multiple outputs closely matching one target but none matching another target. In the case where there is one target and one output produced, this measure is equivalent to the root-mean squared error commonly used in evaluating conventional neural systems. To account for situations in which no outputs were produced, the error for each target was set to 1 – as the datasets' targets varied in the range 0..1 this approximated the largest error which could have been measured had an output been produced. This metric will be referred to as distance error in the Results section.

Algorithm 4 An error metric for comparing sets of target and output values

input: set of targets T, set of network outputs S
for each target $t \in T$ **do**
    if S is empty **then**
        $e = e + 1$
    else
        $e = e + \min(\| t - s \|)$ over all $s \in S$
    end if
end for
for each target $s \in S$ **do**
    $e = e + \min(\| t - s \|)$ over all $t \in T$
end for
output: $e/(\| T \| + \| S \|)$

As an additional error metric, the variation in cardinality between the target outputs and the actual outputs produced was measured for each instance shown to the network. The mean of the absolute error in cardinality was measured and reported separately for the training folds, the test folds, and also for the null folds (the data points sampled from regions of input space where no corresponding output values existed). This metric will be referred to as cardinality error in the Results section.

For each dataset 20 independent runs of each algorithm were performed, and results were averaged across all folds and all runs. Suitable parameter settings for each algorithm were determined via a small number of test runs, and as far as possible were kept consistent across all datasets. Table 4 summarises these parameter settings.

In addition to comparing the performance of RANVOC to the functional regression performed by RAN, it is also possible to establish bounds on the best performance achievable by any functional regression algorithm. Consider a hypothetical optimal functional regression system which for any data instance produces a single output s which minimises the distance error metric in Algorithm 4. Clearly for functional datasets where T contains only a single target, s will simply equal that target, giving an error of 0. For non-functional datasets, the optimal value of s can be found via a weighted average of the targets, as described in Algorithm 5. Therefore for any given dataset, the lower bound on the distance error for any functional regression system can be established by applying Algorithm 5 to each instance in the data-set, calculating the resulting distance error, and then averaging those errors over all instances. This process was carried out for all of the benchmark datasets, and the results are reported along with the experimental results in the next section.

Algorithm 5 Finding the optimal functional regression output for a given set of targets

```
input: set of targets T = t_1, .., t_n
e_min = ∞
sum = Σ_{j=1}^{n} t_j
for each target t_i ∈ T do
    s_temp = (sum + t_i)/(n+1)
    e = distance error of s_temp using Algorithm 4
    if e < e_min then
        e_min = e
        s = s_temp
    end if
end for
output: s
```

Table 4: Algorithm parameters for each dataset.

Dataset	RAN	RANVOC
All data sets	$\alpha = 0.05$ $\kappa = 0.87$ $epsilon = 0.02$	$\alpha = 0.05$ $\kappa = 0.87$ $epsilon_{low} = 0.02$ $epsilon_{high} = 0.3$ $\rho = 0.8$ $\beta = 0.01$
Quartic-F	$\delta_{max} = 0.2$ $\delta_{min} = 0.01$	$\delta_{max} = 0.05$ $\delta_{min} = 0.01$
Quartic-NF	$\delta_{max} = 0.2$ $\delta_{min} = 0.05$	$\delta_{max} = 0.4$ $\delta_{min} = 0.05$
Circles	$\delta_{max} = 0.4$ $\delta_{min} = 0.2$	$\delta_{max} = 0.2$ $\delta_{min} = 0.02$
Ellipsoid1D-F	$\delta_{max} = 0.05$ $\delta_{min} = 0.01$	$\delta_{max} = 0.1$ $\delta_{min} = 0.01$
Ellipsoid1D-NF	$\delta_{max} = 0.02$ $\delta_{min} = 0.01$	$\delta_{max} = 0.1$ $\delta_{min} = 0.02$
Ellipsoid2D-NF	$\delta_{max} = 0.1$ $\delta_{min} = 0.05$	$\delta_{max} = 0.1$ $\delta_{min} = 0.02$

3.3. Results and discussion

Tables 5 and 6 list the mean performance of each algorithm over 20 crossfold-validated trials on each dataset, for the distance and cardinality error metrics respectively. Looking first at the two datasets based on functional relationships (Quartic-F and Ellipsoid1D-F) it can be seen that RAN fits these datasets extremely accurately, while RANVOC produces a less accurate mapping from input to output. This is to be expected as RANVOC's capacity for producing additional outputs can only harm its performance on problems such as these with fixed output cardinality. The results demonstrate that if a dataset is known to be functional in nature, then the best option is to use an algorithm designed for such data. Nevertheless RANVOC can still perform reasonably if applied to such data.

The results on the remaining, non-functional datasets clearly illustrate the problems with applying a standard regression approach such as RAN to this type of data. From Figure 3 it can be seen that the error produced by the RAN algorithm increases rapidly as the degree of non-functionality of the

Table 5: Mean results of each algorithm on the distance error metric for each dataset over 20 crossfold validated trials.The differences in performance between RAN and RANVOC on each error metric on each dataset have been confirmed as significant at $p \leq 0.01$ using the Wilcoxon Signed Rank Test. Results for the hypothetical Optimal Functional Regression system (OFR) are also shown for comparison

Dataset	Fold	RAN Distance Error	RANVOC Distance Error	OFR Distance Error
Quartic-F	Training	0.00007	0.0012	-
	Test	0.00014	0.0016	0
Quartic-NF	Training	0.1091	0.0013	-
	Test	0.1087	0.0014	0.1015
Circles	Training	0.3724	0.0109	-
	Test	0.3730	0.0113	0.3545
Ellipsoid	Training	0.00004	0.0016	-
	Test	0.00006	0.0012	0
Ellipsoid1D-NF	Training	0.0418	0.0107	-
	Test	0.0420	0.0108	0.0364
Ellipsoid2D-NF	Training	0.0525	0.0145	-
	Test	0.0539	0.0154	0.0426

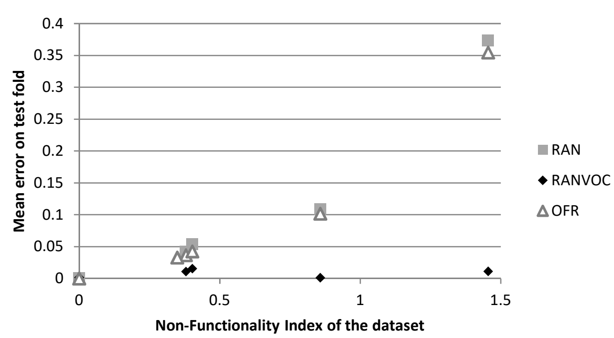

Figure 3: The influence of the NFI of the dataset on the mean test-fold distance error of the RAN and RANVOC algorithms, as well as the hypothetical Optimal Function Regression (OFR).

Table 6: Mean results of each algorithm on the cardinality error metric for each dataset over 20 crossfold validated trials. RAN and the Optimal Functional Regression system have the same cardinality error results, as will any other functional regression, as they produce a single output for each instance. The differences in performance between RANVOC and the functional regression algorithms on each error metric on each dataset have been confirmed as significant at $p \leq 0.01$ using the Wilcoxon Signed Rank Test.

Dataset	Fold	RAN/OFR Cardinality Error	RANVOC Cardinality Error
Quartic-F	Training	0	0.1294
	Test	0	0.1360
	Null	1	0.2067
Quartic-NF	Training	2.907	0.1902
	Test	2.907	0.1995
	Null	1	2.58
Circles	Training	2.06	0.4808
	Test	2.06	0.5052
	Null	1	1.09
Ellipsoid1D-F	Training	0	0.1411
	Test	0	0.1513
	Null	1	0.7987
Ellipsoid1D-NF	Training	0.958	0.3050
	Test	0.958	0.3036
	Null points	1	0.5858
Ellipsoid2D-NF	Training	1.17	0.4476
	Test	1.17	0.4497
	Null points	1	0.3218

dataset increases, whereas RANVOC's performance is much less influenced by the NFI of the dataset. Figures 4 and 5 show that when presented with data with multiple outputs, RAN tends to produce an approximation which bears little resemblance to the original data. RANVOC substantially outperforms RAN in terms of both the distance and cardinality error metrics over the training and test folds. In Figures 4 and 5 it can be seen that RANVOC generally fits the datasets accurately, but that some errors occur at the edge of the regions of relevance for each output unit, particularly when these units have similar output values, suggesting that further work is required to refine the algorithm for training of the relevance weights.

Importantly the results obtained by RANVOC on the non-functional datasets are superior not just to those achieved by RAN, but to the best possible results obtainable by any form of functional regression, as shown in the final column of Table 5 and in Figure 3. This clearly demonstrates the benefits of using a system designed specifically for non-functional regression when the dataset is known to be non-functional.

During execution of these experiments it was observed that RANVOC was considerably more sensitive to the setting of the δ_{min} and δ_{max} parameters which control the width of the hidden unit's activation functions. In particular RAN's performance was largely indifferent to the value of δ_{max}, whereas this parameter had a considerable impact on RANVOC. Specifically the setting of the δ parameters impacted directly on the cardinality of the output set. Overly large values of δ_{max} resulted in RANVOC producing more outputs than required in many regions of input space, whereas small values of δ_{min} resulted in the creation of units which were relevant only to small regions of input space – in some cases this resulted in no outputs being produced for some instances in the test fold.

4. Application to plasma spray processes

The previous set of experiments evaluated and analysed the behaviour of the RAN and RANVOC algorithms on artificially generated datasets. This section examines the application of these methods to a real-world dataset, to demonstrate the potential significance of non-functional regression in practice. Specifically we examine the task of learning a mapping from the desired output characteristics of an atmospheric plasma spray device to the input parameters required to produce those output characteristics. This is the inverse of the mapping previously considered by [4, 13], and this work is based

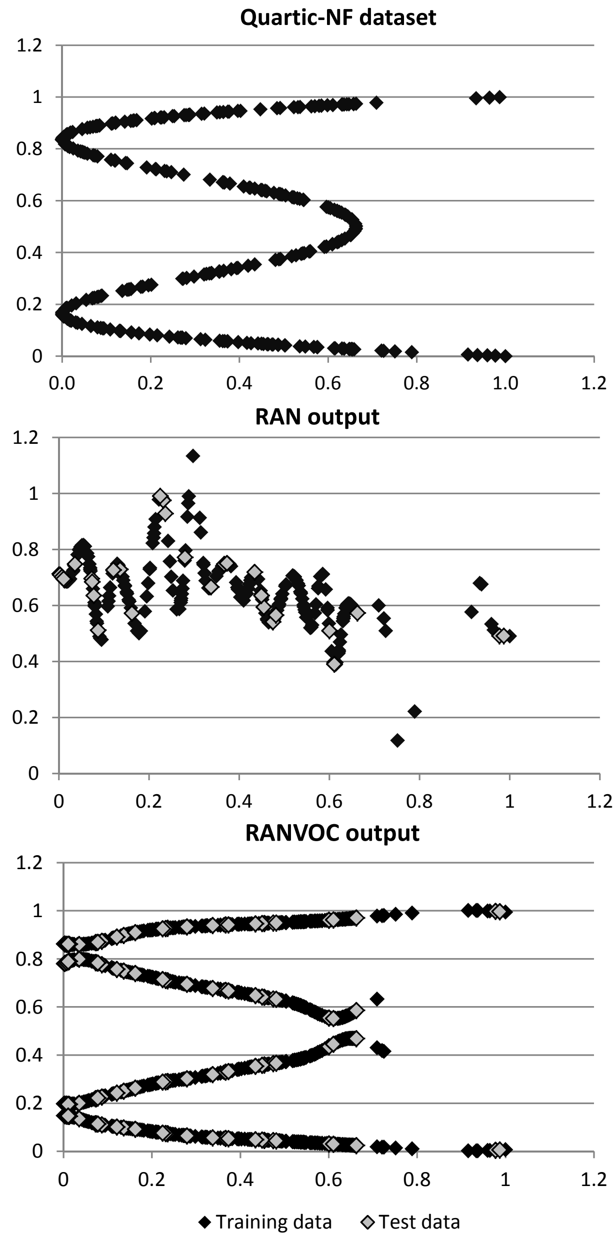

Figure 4: A visualization of the Quartic-NF dataset (top), the output of a randomly selected run of RAN (middle), and of RANVOC (bottom). The horizontal axis is the input value, and the vertical axis is the output value.

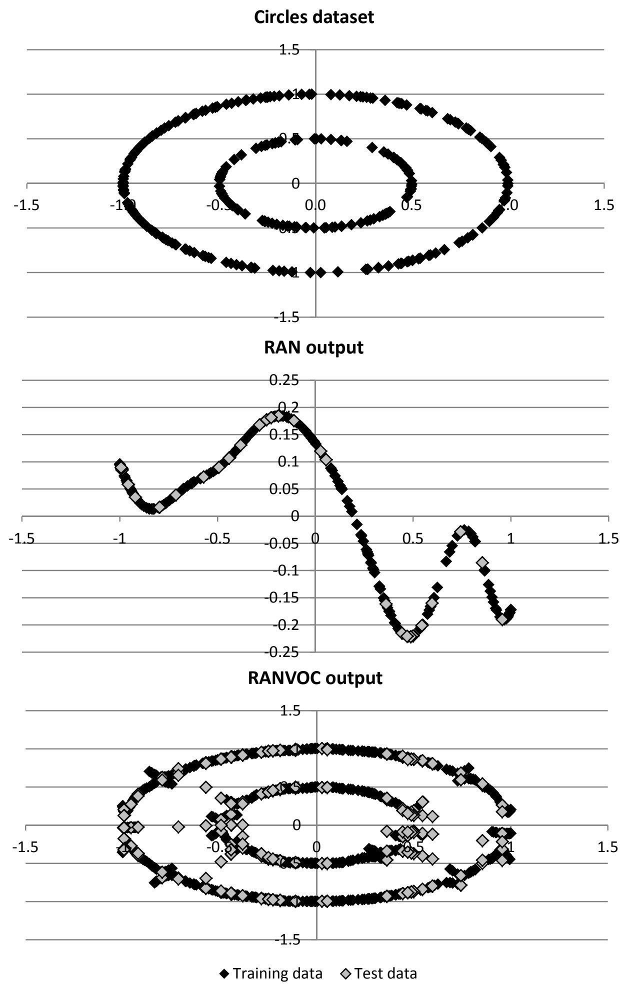

Figure 5: A visualization of the Circles dataset (top), the output of a randomly selected run of RAN (middle), and of RANVOC (bottom). The horizontal axis is the input value, and the vertical axis is the output value.

on the datasets used in those studies, which were originally created by [14].

4.1. Data generation and pre-processing

This data was generated by monitoring over an extended period of time the output particle characteristics produced by a plasma spray device using particular input settings. The output characteristics were averaged over time to create a profile of the spray produced using each particular set of input parameters. This process was repeated for 41 different sets of input parameters. Therefore the complete dataset consists of 41 instances where each instance is defined in terms of a vector P of seven input parameters (current, argon plasma gas flow rate, hydrogen plasma gas flow rate, combined flow rate, hydrogen to argon ratio, argon carrier gas flow rate and injector stand-off distance), and a vector C of three output characteristics (particle speed, temperature and diameter). Each attribute was normalised across the complete dataset to the range 0..1.

With just 41 instances this dataset is sparse in nature. [15] note that sparsity of data is often an issue for data derived from monitoring of an industrial process under varying conditions, as production of more comprehensive data may be prohibitively expensive. Their results also indicate that neural networks based on radial basis functions may perform poorly when trained on sparse data. To address this issue, the original plasma spray dataset was augmented by generating additional data instances via the addition of random noise to each of the genuine data-points. This was carried out post-normalization. Uniformly distributed noise in the range ± 0.02 was added to each element of both the P and C vectors. This was carried out 10 times for each of the original data instances, resulting in an expanded dataset containing 410 instances, which is suitable for training a neural network to learn the mapping $C \mapsto P$.

As with the artificial data experiments, the evaluation of the network's performance post-training is complicated by the non-functional nature of this mapping. The evaluation metric defined in Algorithm 4 requires that a set of targets T be available for each evaluation instance. In the case of artificial data this could be created as the true identity of the generators producing the data was known. For real-world data this is not the case, and so to construct a *ground-truth* for use with this evaluation metric, the original plasma spray data instances were clustered on the basis of their C vectors. If two instances had C vectors within 0.1 units of each other, they were merged into a cluster, and it was assumed that when presented with a set of particle characteristics

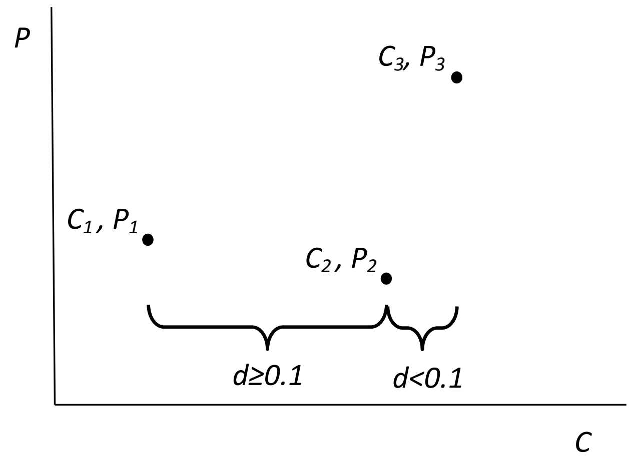

Figure 6: An illustration of the clustering process used to generate target sets T for evaluation of regression performance – note that for simplicity C and P are shown as scalar values, but the process generalises to vectors. For input C_1 the target set $T = \{P_1\}$ while for inputs C_2 and C_3 which lie within 0.1 of each other, $T = \{P_2, P_3\}$.

within that range, the desired behaviour for the network would be to produce the P vector for each of the instances within that cluster. This process is illustrated in Figure 6.

As well as providing a set of target vectors T for each instance in the data-set, this clustering process also provides insight into the underlying nature of the data. The clusters formed via this process contained between one and seven instances. It was observed that the data exhibited mixed levels of non-functionality. Almost 25% of clusters contained only a single instance. However other clusters contained multiple instances, and in some cases these had widely differing spray settings, indicating a non-functional relationship between C and P, which would be expected to pose problems for the RAN approach to regression. The NFI of the plasma-spray dataset is 0.35, which is comparable to that of the Ellipsoid1D-NF and Ellipsoid2D-NF benchmark datasets.

4.2. Experiments, results and discussion

RAN and RANVOC networks were trained on the expanded plasma spray data-set of 410 instances, using 10-fold cross-validation and 20 independently seeded trials of 100 epochs each. The training parameters were $\alpha = 0.05$, $\delta_{max} = 0.3$, $\delta_{min} = 0.15$, $\kappa = 0.87$, $\rho = 0.8$, $\epsilon_{low} = 0.02$, and $\epsilon_{high} = 0.2$.

Table 7: Mean results for RAN and RANVOC over 20 cross-validated trials on the atmospheric plasma spray dataset.The results for the hypothetical Optimal Functional Regression (OFR) system are also shown.

Metric	RAN	RANVOC	OFR
Training fold distance error	0.0479	0.02789	0.0329
Training fold cardinality error	2.44	0.94	2.44
Test fold distance error	0.0488	0.0284	0.0329
Test fold cardinality error	2.44	0.93	2.44

Table 7 summarises the results observed on the plasma spray application. Overall RANVOC produces an improvement in the distance error of around 40% over RAN, on both the training and test folds. In addition RANVOC outperforms RAN by a factor of 2.5 on the cardinality error. A Wilcoxon Sign Ranked Test confirmed that the observed differences in performance were significant at $p \leq 0.01$. RAN's inability to produce more than a single P vector for a given C vector is a major limitation when the data set contains instances with up to seven target vectors. These results reflect the observed variations in performance of the two algorithms on the artificial datasets. As with the artificial datasets, it is possible to apply Algorithm 5 to the plasma spray dataset to calculate the best possible performance of a hypothetical optimal functional regression system – as shown in Table 7 such a system would outperform RAN, but would still be unable to match the results achieved using RANVOC.

Figure 7 provides a more detailed insight into the behaviour of each algorithm, by mapping the test-fold error on each data-instance against the NFI of that instance, for a single representative run of each algorithm. It can be seen that RAN outperforms RANVOC on the functional data-instances with NFI of zero. However as the NFI of the instances increases the error in RAN's output increases rapidly. Meanwhile RANVOC's performance is largely unaffected by the NFI of the data, apart from a few outliers. Overall RAN is far more likely to suggest plasma spray parameters P which deviate substantially from those required to produce the desired spray particle characteristics.

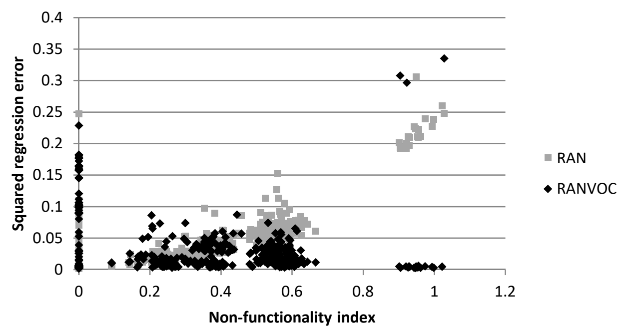

Figure 7: The influence of the NFI of individual data instances on the test-fold error for representative runs of the RAN and RANVOC algorithms

5. Conclusion and future work

This paper has made three main contributions. The first is the identification of non-functional neural regression as an overlooked area of research with important applications in a range of areas of machine learning. The second contribution is the proposal and evaluation of the first neural network algorithm designed specifically for learning non-functional regression tasks, the Resource Allocating Network with Varying Output Cardinality (RANVOC). The final contribution is the establishment of benchmark datasets, an evaluation methodology and metrics suitable for assessing the effectiveness of an algorithm for non-functional regression, and a methodology based on clustering and a Non Functional Index metric to establish whether a datset is better suited to functional or non-functional approaches to regression.

The experimental results have demonstrated that RANVOC offers a substantial improvement in performance over the standard functional approach to regression when applied to data where the input to output mapping is not functional in nature. RANVOC's performance as measured by the distance metric is quite strong across all six artificial datasets examined in this study, regardless of whether the datasets are functional or non-functional. In contrast RAN performs well on the functional datasets, but much less accurately on the non-functional datasets. In particular RAN's output on the Quartic-NF and Circles datasets demonstrates no ability to learn these datasets. These differences in algorithmic behaviour were also evident when the algorithms were applied to a real-world dataset derived from measurements of an atmospheric plasma spray device, with RANVOC providing much improved results in terms of both distance and cardinality error metrics. Importantly

RANVOC has been shown to outperform not just the original RAN algorithm, but also the best possible error rates achievable by any functional form of regression.

The primary limitation exhibited by the RANVOC algorithm is that erroneous values sometimes occur at the borders of the regions for which each output unit is relevant, as evident in Figures 4 and 5. This leads to errors in both the cardinality and distance metrics. A related problem is that the cardinality performance on null points (input examples for which no output should be produced) is sometimes poor.

Future work should investigate two approaches to addressing these limitations. First it was observed that the optimal δ settings for value fitting may not be optimal for relevance testing, as reflected by the fact that the best cardinality results were often achieved at lower values of δ_{max} than were the best distance results. One means to address this may be to maintain two separate sets of hidden units with their own δ parameters – one set of units is used for output value estimation, while the second set is used only for relevance testing. A second, possibly complementary, approach is to make use of null points during training as a means of decaying relevance weights for regions of input space where no output should be produced. The main obstacle to be overcome with such an approach is how to appropriately generate input vectors representing such null points for real datasets for which the underlying generators are not known.

In addition RAN was selected as a suitable choice for initial experimentation with varying output cardinality as its approach to building an RBF network is relatively simple. However RAN's learning capabilities have been improved on by more sophisticated constructive algorithms. For example Huang et al. [11] provides substantial improvements in learning efficiency, while other algorithms such as Huang et al. [10], and Vuković and Miljković [12] provide support for pruning unwanted hidden neurons. The results calculated for the hypothetically optimal functional regression system demonstrate that these more advanced functional regression algorithms can not in themselves match RANVOC's performance on non-functional datasets. However incorporating the varying output cardinality concepts pioneered in RANVOC into these more advanced neural regression systems should allow for the development of more accurate and efficient non-functional regression systems, and this should be a focus of future work.

[1] L. Ciabattoni, G. Ippoliti, S. Longhi, M. Pirro, M. Cavalletti, Solar

irradiation forecasting for PV systems by fully tuned minimal RBF neural networks, in: Neural Nets and Surroundings, Springer, 2013, pp. 289–300.

[2] J.-c. Yin, Z.-j. Zou, F. Xu, Sequential learning radial basis function network for real-time tidal level predictions, Ocean Engineering 57 (2013) 49–55.

[3] D. S. Tok, D.-L. Yu, C. Mathews, D.-Y. Zhao, Q.-M. Zhu, Adaptive structure radial basis function network model for processes with operating region migration, Neurocomputing 155 (2015) 186–193.

[4] T. Choudhury, N. Hosseinzadeh, C. Berndt, Artificial neural network application for predicting in-flight particle characteristics of an atmospheric plasma spray process, Surface and Coatings Technology 205 (2011) 4886–4895.

[5] J. Platt, A resource-allocating network for function interpolation, Neural computation 3 (1991) 213–225.

[6] D. Roijers, P. Vamplew, S. Whiteson, R. Dazeley, A survey of multi-objective sequential decision-making, Journal of Artificial Intelligence Research 48 (2013) 67–113.

[7] M. Drugan, M. Wiering, P. Vamplew, M. Chetty, Special issue on multi-objective reinforcement learning, Neurocomputing 263 (2017).

[8] K. Van Moffaert, A. Nowé, Multi-objective reinforcement learning using sets of Pareto dominating policies, Journal of Machine Learning Research 15 (2014) 3483–3512.

[9] L. Franco, J. M. Jerez, Constructive neural networks, volume 258, Springer, 2009.

[10] G.-B. Huang, P. Saratchandran, N. Sundararajan, A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation, IEEE Transactions on Neural Networks 16 (2005) 57–67.

[11] G.-B. Huang, L. Chen, C. K. Siew, et al., Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Networks 17 (2006) 879–892.

[12] N. Vuković, Z. Miljković, A growing and pruning sequential learning algorithm of hyper basis function neural network for function approximation, Neural Networks 46 (2013) 210–226.

[13] T. Choudhury, C. Berndt, Z. Man, Modular implementation of artificial neural network in predicting in-flight particle characteristics of an atmospheric plasma spray process, Engineering Applications of Artificial Intelligence 45 (2015) 57–70.

[14] S. Guessasma, G. Montavon, P. Gougeon, C. Coddet, Designing expert system using neural computation in view of the control of plasma spray processes, Materials & design 24 (2003) 497–502.

[15] R. Lanouette, J. Thibault, J. L. Valade, Process modeling with neural networks using small experimental datasets, Computers & Chemical Engineering 23 (1999) 1167–1176.