Applied Soft Computing 133 (2023) 109930

Contents lists available at ScienceDirect

Applied Soft Computing

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

Pareto Optimal Prediction Intervals with Hypernetworks

Antonio Alcántara *, Inés M. Galván, Ricardo Aler

Universidad Carlos III de Madrid, Avenida Universidad, 30, 28911, Leganés (Madrid), Spain

ARTICLE INFO

Article history:
Received 14 December 2021
Received in revised form 11 October 2022
Accepted 7 December 2022
Available online 13 December 2022

MSC:
68T05

Keywords:
Direct prediction intervals estimation
Hypernetworks
Multi-objective optimization
Probabilistic forecasting
Deep neural networks

ABSTRACT

As the relevance of probabilistic forecasting grows, the need of estimating multiple high-quality prediction intervals (PI) also increases. In the current state of the art, most deep neural network gradient descent-based methods take into account interval width and coverage into a single loss function, focusing on a unique nominal coverage target, and adding additional parameters to control the coverage–width trade-off. The Pareto Optimal Prediction Interval Hypernetwork (POPI-HN) approach developed in this work has been derived to treat this coverage–width trade-off as a multi-objective problem, obtaining a complete set of Pareto Optimal solutions (Pareto front). POPI-HN are able to be trained through gradient descent with no need to add extra parameters to control the width–coverage trade-off of PIs. Once the Pareto set has been obtained, users can extract the PI with the required coverage. Comparative results with recently introduced Quality-Driven loss show similar behavior in coverage while improving interval width for the majority of the studied domains, making POPI-HN a competing alternative for estimating uncertainty in regression tasks where PIs with multiple coverages are needed.

1. Introduction

In general, probabilistic forecasting estimates a probability for every possible event that can happen, whereas standard point prediction assigns a single value (typically the average, as in linear least squares). This assignment of probabilities makes probabilistic forecasting more valuable than point forecasting in some research fields. For example, in electricity price forecasting [1], astrophysics for world mass estimation [2], probabilistic photovoltaic generation estimation [3] or wind energy resources [4].

However, when dealing with regression tasks (the target is a continuous value), it is not possible to assign non-zero probabilities to single points. For that reason, methods based on the estimation of Prediction Intervals (PIs) are becoming popular as a way to estimate uncertainty in regression tasks. A PI is formed by two values: a lower and an upper bound, which contain the conditional dependent variable of interest with a certain probability.

There are two main ways of estimating PIs. First, models can output specific values that are later used to construct PIs. For example, several methods in the machine learning literature have been extended to the estimation of conditional quantiles, from where lower and upper bounds for PIs can be computed. Some of the most widely used include Quantile Regression Forests [5], Gradient Boosting Quantile Regression [6], or Natural Gradient Boosting [7]. All these methods are based on optimizing the quantile loss [8] and can be used to estimate a set of quantiles, which are then used to obtain the PIs. On the other hand, models can be constructed to directly estimate the PI's lower and upper bounds, therefore, models output those two bounds. Regarding this approach, Neural Networks (NNs) are the most employed methods, as they can have two outputs that represent the lower and upper bounds of the PIs. Furthermore, NNs for direct estimation usually optimize losses directly related to the quality of the PI. For example, in [9], the Coverage Width-based Criterion (CWC) loss is defined, whereas, in [10], a loss combining the PI coverage and sharpness is used.

Knowing the importance of probabilistic forecasting in all kinds of fields, it might be interesting to investigate the use of Deep Neural Networks (DNNs) as the prediction method for probabilistic regression tasks, due to the good performance and flexibility that DNNs show in many fields, from image classification [11] to bot detection [12] or non-parametric regression [13].

DNNs have been used to estimate PIs employing the prior quantile estimation, making use of the quantile loss, and building statistically centered PIs with their corresponding quantiles [14–18]. Neural networks have been also employed for direct estimation of PIs. For example, in [19–21], one hidden layer neural network is used to estimate the lower and upper bounds of PIs. DNNs have also been considered for this purpose [22]. Nevertheless, in these works, optimization was based on evolutionary algorithms, which are computationally expensive. A more efficient approach is to use gradient descent, the widely used method

* Corresponding author.
E-mail addresses: antalcan@est-econ.uc3m.es (A. Alcántara), igalvan@inf.uc3m.es (I.M. Galván), aler@inf.uc3m.es (R. Aler).

https://doi.org/10.1016/j.asoc.2022.109930

for optimizing DNN loss. However, the use of gradient descent makes the loss function play a key role in the process.

One of the initially used loss functions for PI estimation with neural networks is the Coverage Width-based Criterion (CWC), also known as Lower Upper Bound Estimation method (LUBE), a loss function built as a multiplicative function of width and coverage terms [9], to deal with the trade-off between both (PIs with high coverage must be wider and the other way around). This loss has been employed in a wide range of fields, from short-term wind power forecasting [23,24], to streamflow discharges [25] or cyberattack detection [26].

Neural networks that employ CWC are usually trained by evolutionary algorithms and not gradient descent. The structure of CWC can cause problems with gradient descent optimization: a minimum can be found with trivial PIs of zero width due to the multiplicative structure of the loss [27]. Recently, in [27], a Quality-Driven loss was derived to be efficiently optimized by gradient descent. In this case, the loss has two aggregated coverage and width terms, instead of multiplicative ones. It adds the sample size to the loss for giving confidence about the coverage, and a penalty parameter λ to control the trade-off between width and coverage. Pearce's own results in [27] show how models trained with Quality-Driven loss outperform those trained with CWC (LUBE) in PI quality for some synthetic datasets. More recently, in a probabilistic estimation setting for wind power [28,29], the Quality-Driven loss is used as a training loss for neural networks, which also outperforms the classical CWC in PI width while maintaining the coverage goal. Besides, in these last two works, recurrent structures like RNN, LSTM, or NNres are employed as the goal is to address predictions on time series problems.

Recent works have derived the Quality-Driven loss from different perspectives. For example, [30,31] modified the loss to also include point estimation in addition to the probabilistic one. In [32], changes in the Quality-Driven loss structure were made to help the training process when the batch-size is small, while in [33] a softened version of the loss is presented to estimate multiple PIs.

Despite the extensive use of the Quality-driven loss in the literature, the loss formulations has some issues. First, the addition of parameters in the function to control the coverage importance in the coverage–width trade-off of the PIs requires the parameter to be adjusted. Also, the losses are originally designed to estimate a single PI for a given target coverage. Thus, if multiple PIs (with multiple coverages) with a single model are needed, we need to optimize through the sum or average of losses of PIs, which could lead to possible imbalances in the performance of the obtained PIs. Finally, if the goal is to obtain multiple PIs at once, we need to decide in advance how many PIs are required and for which nominal coverages.

For these reasons, in this work, we study a method based on Hypernetworks [34] that formulates PI estimation as a multi-objective problem between the two PI objectives (width and coverage). The solution to a multi-objective problem is not a single value but a set of Pareto Optimal solutions (the Pareto front). This allows obtaining optimal PIs for every possible nominal coverage with a single model, by looking for the solution in the Pareto front. The main difference between this method based on Hypernetworks and the current state-of-the-art is treating PI estimation as a multi-objective problem instead of a single-objective that aggregates PI metrics. We believe that the multi-objective approach is more appropriate for direct PI estimation, as both coverage and width are in conflict. For instance, coverage increases when the PI is wide, but when the PI is narrow, coverage decreases.

Hypernetworks have recently been used as a method that can handle multi-objective problems [35], with gradient descent as the optimization method. This multi-objective solution has been applied to several problems, such as Multi-task regression, multi-MNIST (image classification), or pixel-wise classification and regression [35]. However, it has not received much attention for PI estimation yet, only on specific fields [36].

The presented approach, Pareto Optimal Prediction Intervals Hypernetworks (POPI-HN), is based on the employment of two DNNs: a small one (the hypernetwork), which takes a vector of preferences for coverage–width trade-off generates parameters (weights), and the target DNN, which making use of the generated weights and the independent variables of the problem, delivers a PI estimation dependent on the preference vector.

The main contributions of POPI-HN are the following:

1. It obtains optimal-width PIs for all nominal coverages employing a multi-objective approach.
2. This is achieved efficiently by using gradient descent on DNNs.
3. Unlike single-objective methods, no penalty parameter for the coverage–width trade-off needs to be adjusted because solutions for all coverages are obtained with a single model.
4. There is no need to specify in advance the number of PIs to be obtained (unlike single-objective methods).

POPI-HN performance has been validated on eight different open-access datasets for regression tasks that use PI quality metrics. POPI-HN PIs have been compared with those generated by a single-objective model: Quality-Driven Deep Neural Networks (QD-DNN). This model employs the Quality-Driven loss and outputs the lower and upper bounds for several predefined PIs.

The structure of this article is as follows. Section 2 will mathematically describe PIs and their two main properties of interest: coverage and width. The Quality-Driven loss will be introduced with its implementation in DNNs for multiple PI estimation (QD-DNN) in Section 3. Section 4 will describe in detail the POPI-HN methodology, from training with preference vectors to PI estimation and Pareto front construction. To test the performance of POPI-HN, Section 5 presents a comparison between multiple data sets and sought PIs between POPI-HN and QD-DNN. Finally, Section 6 draws the main conclusions of this work.

2. Prediction intervals

Probabilistic forecasting is nowadays getting attention for being able to estimate uncertainty in regression tasks. One of the main focuses of probabilistic forecasting is the estimation of PIs. A PI is made up of an upper and lower bound p^{upp} and p^{low} respectively, which contains the dependent variable y with a certain probability.

A model g that produces a PI for the target variable y taking the independent variables $\mathbf{X}$ is described in Eq. (1), where the theoretical probability of the PI containing the conditional target variable is named *PINC* (Prediction Interval Nominal Coverage). The complementary α of *PINC* ($PINC = 1 - \alpha$) can also be used.

$$g(\mathbf{X}) = \left[\hat{p}^{low}, \hat{p}^{upp}\right], \text{ such that } P(\hat{p}^{low} \leq y | \mathbf{X} \leq \hat{p}^{upp}) = PINC = 1 - \alpha \quad (1)$$

In real-world applications, one of the main objectives is to get the actual coverage to be at least equal to the nominal one. This can be measured with the Prediction Interval Coverage Probability (*PICP*, Eq. (2)). This metric allows us to compute the coverage relative to a set of n samples or instances $S = (\boldsymbol{X}, \mathbf{y}) = \{(\boldsymbol{X}_i, y_i)\}_{i=1}^{i=n}$ out of which, the PIs computed by g can be obtained:

$\hat{\boldsymbol{PI}} = \{g(\boldsymbol{X}_i)\}_{i=1}^{i=n} = \{[\hat{p}_i^{low}, \hat{p}_i^{upp}]\}_{i=1}^{i=n}$. In that case, *PICP* can be computed using Eq. (2), where $\mathbb{1}_{[\hat{p}_i^{low}, \hat{p}_i^{upp}]}(y_i)$ is the indicator function that returns 1 when y_i is within the PI bounds, 0 otherwise.

$$PICP(\hat{\boldsymbol{PI}}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{[p_i^{low}, p_i^{upp}]}(y_i) \tag{2}$$

Once the coverage objective is achieved, the quality of the PI can be determined by its width. Thus, if two models build PIs with the same coverage, the one with narrower intervals should be preferred. This metric is measured with the Average Interval Width (*AIW*, Eq. (3)).

$$AIW(\hat{\boldsymbol{PI}}) = \frac{1}{n}\sum_{i=1}^{n}(p_i^{upp} - p_i^{low}) \tag{3}$$

The simplest and most common width metric is the one described in Eq. (3). However, there are possible modifications that can be made to the *AIW* metric. For example, it can be considered that only those PIs that contain the dependent variable should be used to compute the *AIW*. If $c = \sum_{i=1}^{n} \mathbb{1}_{[p_i^{low}, p_i^{upp}]}(y_i)$ is the sum of captured points, the *AIW* for captured points ($AIW_{capt.}$) is formulated as in Eq. (4).

$$AIW_{capt.}(\hat{\boldsymbol{PI}}, \mathbf{y}) = \frac{1}{c}\sum_{i=1}^{n}\left(p_i^{upp} - p_i^{low}\right)\mathbb{1}_{[p_i^{low}, p_i^{upp}]}(y_i) \tag{4}$$

Furthermore, in some cases it is convenient to normalize the width to the maximum possible range $y_{max} - y_{min}$ of the dependent variable in the set of samples, as defined in Eq. (5).

$$AIW_{norm.}(\hat{\boldsymbol{PI}}) = \frac{1}{n(y_{max} - y_{min})}\sum_{i=1}^{n}(p_i^{upp} - p_i^{low}) \tag{5}$$

3. Single-objective method: Quality-Driven deep neural networks

In this section, we introduce the Quality-Driven loss in order to estimate multiple PIs. As stated in Section 1, the Quality-Driven loss [27] was recently introduced as a work intended to improve CWC, adding flexibility and creating a loss structure that can be used to train deep neural networks with gradient descent. This loss can be used within a single-objective approach as it aggregates the width and coverage of the PIs into a single value. The Quality-Driven loss is presented in Eq. (6).

$$Loss_{QD,\alpha}(\hat{\boldsymbol{PI}}, \mathbf{y}) = AIW_{capt.}(\hat{\boldsymbol{PI}}) + \frac{\lambda n}{\alpha(1-\alpha)}\max(0, (1-\alpha) - PICP(\hat{\boldsymbol{PI}}, \mathbf{y}))^2 \tag{6}$$

where $AIW_{capt.}$ is the one defined in Eq. (4), λ is a control parameter, n the number of instances in the training set and $(1-\alpha) = PINC$ represents the sought coverage. As can be seen, the Quality-Driven loss is composed of two adding terms: one related to the width of the PIs and one penalty weighted by λ for the cases when the actual coverage (*PINC*) is below expected.

The Quality-Driven loss is relative to a single *PINC*. However, if the deep network is required to output PIs for several *PINC* values, the loss can be computed as the sum of Quality-Driven losses for every required coverage. This kind of network for multiple *PINC* values will be called Quality-Driven Deep Neural Networks (QD-DNN) and is used in this work for comparison purposes. More concretely, if the network needs to output PIs for p different nominal coverages $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_p\}$, the Quality-Driven loss for multiple PI estimation is defined in Eq. (7) (note: the α_i are actually the complementary values of the nominal coverage: $\alpha_i = 1 - PINC_i$).

$$Loss_{QD,\boldsymbol{\alpha}}(\hat{\boldsymbol{PI}}, \mathbf{y}) = \sum_{i=1}^{p} Loss_{QD,\alpha_i}(\hat{\boldsymbol{PI}}, \mathbf{y}) \tag{7}$$

The structure of the QD-DNN can be visualized in Fig. 1. As it can be seen, the predictors enter the network, pass through a linear layer, then undergo the non-linear activation, and finally dropout. This process is repeated depending on the depth of the network (number of layers) until reaching the output layer. At the output, $2p$ values are obtained: the lower and upper bound of the PI for each of the p coverages. In Fig. 1, $\hat{PI}_{\alpha_i} = [\hat{p}_{\alpha_i}^{low}, \hat{p}_{\alpha_i}^{upp}]$ represents the ith PI, with required coverage $1-\alpha_i$. For every PI, the Quality-Driven loss (Eq. (6)) is calculated and then, the sum of the losses (Eq. (7)) is used as a final loss for backpropagation.

Estimating multiple PIs (multiple nominal coverages) with a single model by employing QD-DNN has some limitations. First, the number of PIs and the target *PINC* values need to be defined in advance. Secondly, the larger the number of PIs, the more difficult training becomes (compared to using a small set of PIs). Lastly, the λ parameter has to be selected for every coverage and, during training, local minimums can be found where the loss for one PI is improved at the cost of impairing the loss of another one.

Implementing the sum of Quality-Driven losses (Eq. (7)) used in this work for the QD-DNN model follows Algorithm 1. The algorithm needs the training set (*TS*) with the target values $\mathbf{y}_{TS} = \{y_1, y_, \ldots, y_n\}$, the vector $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_p\}$ of sought *PINC* values, and the PIs generated for the training set and each of the α values, represented by $\hat{\boldsymbol{PI}}_{\boldsymbol{\alpha}}$, which is composed of the lower bounds and upper bounds for each α_i: $(\hat{\mathbf{p}}_{\boldsymbol{\alpha}}^{low}, \hat{\mathbf{p}}_{\boldsymbol{\alpha}}^{upp})$. Algorithm 1 also uses some parameters: the softening factor s and the importance of coverage λ.

The algorithm loops over the p different coverages (line 1), computes the Quality-Driven loss for each of them (lines 2–13) and returns the aggregated result (line 14). Lines 2–13 calculate the two main components of the loss: *PICP* and $AIW_{capt.}$, defined in Eqs. (2) and (4), respectively. For the *PICP*, the implementation does not strictly follow Eq. (2), because it was found out by [27] that it was difficult to be optimized. Instead, a sigmoid-soft version is used. This is done in lines 2–7, where $\mathbf{K}_{S,i}$ represents the soft number of captured points (line 7) by the ith PI. Then, soft $PICP_{S,i}$ is computed in line 8 as the mean value of the vector $\mathbf{K}_{S,i}$ (reduce-mean($\mathbf{K}_{S,i}$) means first add all the elements of vector $\mathbf{K}_{S,i}$, then divide by the number of elements). Notice that there are several element-wise vector operations. For instance, $\hat{\mathbf{p}}_{\alpha_i}^{upp} - \mathbf{y}_{TS}$ of line 2 displays the difference between two vectors of size 1 column and n rows each.

The other important element of the loss will be the *AIW* of captured points (Eq. (4)). For this, $\mathbf{K}_{H,i}$ (a vector containing ones and zeros depending on whether the point is captured or not by the PI with coverage α_i) plays an important role (line 4). The computation of $AIW_{capt.}$ defined by [27] is done in line 12. However, we found out that there are some cases, at the beginning of the training process, especially in datasets with a small number of instances, where the sum of values of $\mathbf{K}_{H,i}$ can be zero (i.e. no point is captured). This would lead to a zero divisor in line 12. Our implementation uses line 10 for those cases, so that the *AIW* is assigned the worst possible value (the maximum range of the target variable). This allows the training process to continue without errors even when no points are captured by the PIs. Eventually, some points will be captured and the $AIW_{capt.}$ computation will be able to use line 12. reduce-sum in those lines is just the summation of all the elements of the vector.

Then, the single loss for one PI is computed as in Eq. (6) (line 13). This process will be repeated for all the p different PIs

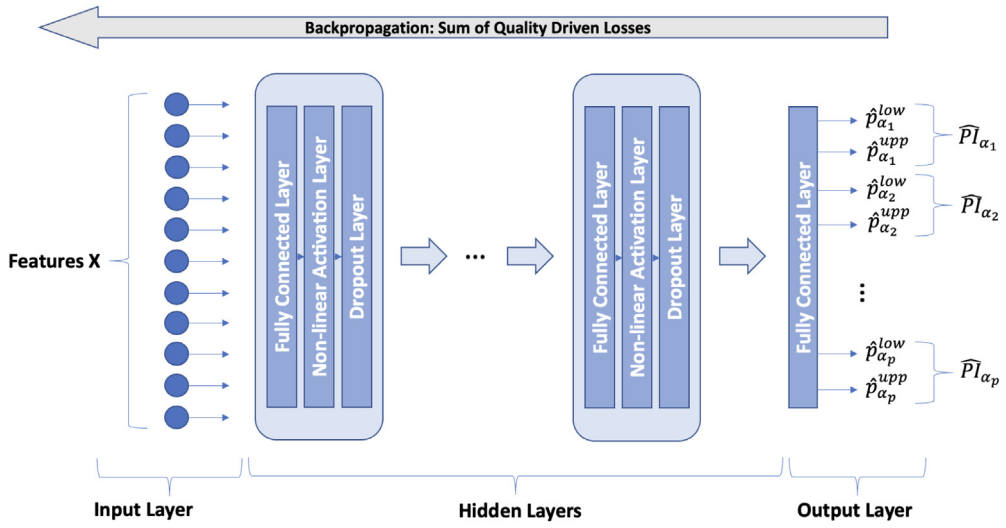

Fig. 1. QD-DNN structure.

generated by QD-DNN, all with different sought *PINC* values. The losses will be aggregated as a sum of losses, producing the final QD loss (Eq. (7)) of line 14. This is the loss that will be used by backpropagation.

Algorithm 1: Implementation of the Sum of Quality-Driven losses for multiple PI estimation.

Data: $TS = (\boldsymbol{X}, \mathbf{y})$: Training Set
p: number of target coverages
$\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, \ldots, \alpha_p\}$: Complementary vector of *PINC* values
$\hat{\boldsymbol{PI}}_{\boldsymbol{\alpha}} = (\hat{\mathbf{p}}^{low}_{\boldsymbol{\alpha}}, \hat{\mathbf{p}}^{upp}_{\boldsymbol{\alpha}})$: PIs generated for the TS
. $\hat{\mathbf{p}}^{low}_{\boldsymbol{\alpha}} = \{\hat{\mathbf{p}}^{low}_{\alpha_i}\}_{i=1}^{i=p}$: Lower bounds of PIs
. $\hat{\mathbf{p}}^{upp}_{\boldsymbol{\alpha}} = \{\hat{\mathbf{p}}^{upp}_{\alpha_i}\}_{i=1}^{i=p}$: Upper bounds of PIs
s: softening factor
λ: parameter of coverage importance
(Note: $\odot$ denotes the element-wise product)
Result: $Loss_{QD,\boldsymbol{\alpha}}$

1 **for** i = *1 TO* p **do**
2 $\quad \mathbf{K}_{HU,i} = max(0, sign(\hat{\mathbf{p}}^{upp}_{\alpha_i} - \mathbf{y}_{TS}))$
3 $\quad \mathbf{K}_{HL,i} = max(0, sign(\mathbf{y}_{TS} - \hat{\mathbf{p}}^{low}_{\alpha_i}))$
4 $\quad \mathbf{K}_{H,i} = \mathbf{K}_{HU,i} \odot \mathbf{K}_{HL,i}$
5 $\quad \mathbf{K}_{SU,i} = sigmoid((\hat{\mathbf{p}}^{upp}_{\alpha_i} - \mathbf{y}_{TS})s)$
6 $\quad \mathbf{K}_{SL,i} = sigmoid((\mathbf{y}_{TS} - \hat{\mathbf{p}}^{low}_{\alpha_i})s)$
7 $\quad \mathbf{K}_{S,i} = \mathbf{K}_{SU,i} \odot \mathbf{K}_{SL,i}$
8 $\quad PICP_{S,i} =$ reduce-mean$(\mathbf{K}_{S,i})$
9 $\quad$ **if** *reduce-sum*$(\mathbf{K}_{H,i}) = 0$ **then**
10 $\quad\quad AIW_{capt.,i} = max(\mathbf{y}_{TS}) - min(\mathbf{y}_{TS})$
11 $\quad$ **else**
12 $\quad\quad AIW_{capt.,i} =$ reduce-sum$((\hat{\mathbf{p}}^{upp}_{\alpha_i} - \hat{\mathbf{p}}^{low}_{\alpha_i}) \odot \mathbf{K}_{H,i})$/reduce-sum$(\mathbf{K}_{H,i})$
13 $\quad Loss_{QD,\alpha_i} = AIW_{capt.,i} + \lambda \frac{n}{\alpha_i(1-\alpha_i)} \max(0, (1-\alpha) - PICP_{S,i}))^2$
14 $Loss_{QD,\boldsymbol{\alpha}} = \sum_{i=1}^{p} Loss_{QD,\alpha_i}$
15 Return$(Loss_{QD,\boldsymbol{\alpha}})$

4. Hypernetworks to solve the coverage–width trade-off in a multi-objective framework

The goal of this article is to develop a method capable of obtaining PIs for all possible *PINC* while dealing with the coverage–width trade-off, taking advantage of the deep neural network structure, and needing as few parameters as possible to facilitate the training process. In this section, we state direct PI estimation as a multi-objective formulation, dealing simultaneously with PIs coverage and width.

The method described in Section 3 was a single-objective method because although Quality-Driven loss considered two goals, coverage and width, both were aggregated into a single quantity, weighted by a trade-off parameter λ. In this section, a multi-objective formulation is proposed where two separate objectives, *AIW* and $\epsilon = 1 - PICP$ of the PIs, are to be minimized simultaneously.

Let t be a model that generates PIs conditional to inputs $\mathbf{X}$ as shown in Eq. (8). t is defined by parameters $\boldsymbol{\theta}$, that could be the coefficients of a linear model or the weights of a neural network (as is the case in this work). Eq. (8) describes the situation where a PI is the output of the inputs $\mathbf{X}$ of a single sample. Eq. (9) describes the situation where model t is applied to a set of n instances $S = (\mathbf{X}, \mathbf{y}) = \{(\mathbf{X}_i, y_i)\}_{i=1}^{i=n}$, where $\mathbf{X} = \{\mathbf{X}_i\}_{i=1}^{i=n}$ and $\mathbf{y} = \{y_i\}_{i=1}^{i=n}$. In that case, the output is a set of n PIs ($\hat{\boldsymbol{PI}}$) or equivalently their n lower $\hat{\mathbf{p}}^{low}$ and upper $\hat{\mathbf{p}}^{upp}$ bounds (e.g. $\hat{\mathbf{p}}^{low} = \{\hat{p}_i^{low}\}_{i=1}^{i=n}$).

$$t(\mathbf{X}; \boldsymbol{\theta}) = [\hat{p}^{low}, \hat{p}^{upp}] \tag{8}$$

$$t(\mathbf{X}; \boldsymbol{\theta}) = \hat{\boldsymbol{PI}} = [\hat{\mathbf{p}}^{low}, \hat{\mathbf{p}}^{upp}] \tag{9}$$

Then, the multi-objective optimization problem is defined by Eq. (10).

$$\mathbf{PF} = \underset{\boldsymbol{\theta}}{\text{argmin}} \ \ (AIW(t(\mathbf{X}; \boldsymbol{\theta})), \epsilon(t(\mathbf{X}; \boldsymbol{\theta}), \mathbf{y})) \tag{10}$$

where $\epsilon(\hat{\boldsymbol{PI}}, \mathbf{y}) = 1 - PICP(\hat{\boldsymbol{PI}}, \mathbf{y})$. In the multi-objective case, there does not usually exist a solution that minimizes all objectives simultaneously. Therefore, the optimal solution is not a single one but a set of them, called Pareto optimal solutions, that together constitute the Pareto front (**PF**).

In order to formally define the Pareto front, it is useful to first define when a solution $\boldsymbol{\theta}_1$ dominates another $\boldsymbol{\theta}_2$. In general, this happens when the first solution is better than or equal for all objectives, but it is strictly better for at least one of them. In the case of PIs, where only two objectives are considered (*AIW* and ϵ), dominance is defined by Eq. (11).

$\boldsymbol{\theta}_1$ dominates $\boldsymbol{\theta}_2$ if :

$$\begin{aligned} &AIW_{\boldsymbol{\theta}_1} \leq AIW_{\boldsymbol{\theta}_2} \text{ and } \epsilon_{\boldsymbol{\theta}_1} \leq \epsilon_{\boldsymbol{\theta}_2} \\ &AIW_{\boldsymbol{\theta}_1} < AIW_{\boldsymbol{\theta}_2} \text{ or } \epsilon_{\boldsymbol{\theta}_1} < \epsilon_{\boldsymbol{\theta}_2} \end{aligned} \tag{11}$$

where $(AIW_{\theta_1}, \epsilon_{\theta_1})$ and $(AIW_{\theta_2}, \epsilon_{\theta_2})$ are the two losses or objectives of $t(\mathbf{X}, \boldsymbol{\theta}_1)$ and $t(\mathbf{X}, \boldsymbol{\theta}_2)$, respectively.

A solution $\boldsymbol{\theta}^*$ is called Pareto optimal if it is non-dominated. That is, there is no other solution that dominates it. For a Pareto optimal $\boldsymbol{\theta}^*$ it is not possible to improve coverage (decrease ϵ) without worsening *AIW*, and the other way around. The Pareto front is just the set of Pareto optimal (non-dominated) solutions, as shown in Eq. (12).

$$\mathbf{PF} = \{\boldsymbol{\theta}^* \mid \nexists \boldsymbol{\theta}' \text{ such that } \boldsymbol{\theta}' \text{ dominates } \boldsymbol{\theta}^*\} \tag{12}$$

It is interesting to notice that if the **PF** is obtained, then it is possible to get the model that obtains the best (narrowest) PIs for each possible nominal coverage, which was our initial goal. Let us suppose that we require a model that provides PIs with some coverage *PINC*. Then, we look for the $\boldsymbol{\theta}^*$ model in the **PF** whose ϵ is $1 - PINC$, and let us call it $\boldsymbol{\theta}^*_{PINC}$. This $\boldsymbol{\theta}^*_{PINC}$ will also have some *AIW* value. But given that the interval belongs to the **PF**, it is non-dominated. Thus, there cannot be another model with the same *PINC* but smaller *AIW*.

4.1. Pareto Optimal Prediction Interval Hypernetworks structure

To deal with the coverage–width trade-off, a multi-objective approach has been developed based on hypernetworks (HN). HNs are, in fact, two (deep) neural networks: the hypernetwork and the target (or main) network, and they differ from standard networks as explained below.

Standard (deep) feed-forward neural networks, as the one described in Section 3 (see Fig. 1), are a sequence of layers that contain a set of numerical parameters (the weights). Formally, it can be represented as a function $t(\mathbf{X}; \boldsymbol{\theta})$ with inputs $\mathbf{X}$ and parameters (weights) $\boldsymbol{\theta}$. For standard neural networks, the weights $\boldsymbol{\theta}$ are usually optimized/trained by gradient descent, so that $t(\mathbf{X}; \boldsymbol{\theta})$ produces the right outputs, given the inputs $\mathbf{X}$.

HNs also contain a $t(\mathbf{X}; \boldsymbol{\theta})$ network, called the target or main network, which is also responsible for producing the right outputs for the given inputs to solve the problem at hand. However, the weights $\boldsymbol{\theta}$ are not directly optimized by gradient descent, but are the output of another deep feed-forward network, called the hypernetwork. Formally, it is represented by function $h(\boldsymbol{r}; \boldsymbol{\phi})$ where $\boldsymbol{r}$ are the inputs to the hypernetwork, and $\boldsymbol{\phi}$ are its weights. Mathematically, they work together as $t(\mathbf{X}; h(\boldsymbol{r}; \boldsymbol{\phi}))$. It is the hypernetwork weights $\boldsymbol{\phi}$ that are optimized by gradient descent.

The previous paragraph was a general introduction to hypernetworks, but let us now instantiate the previous HN schema to the problem of generating the Pareto front of PIs. In this case, the inputs $\boldsymbol{r}$ to the hypernetwork $h(\boldsymbol{r}; \boldsymbol{\phi})$ are a vector of preferences (weights) between the two objectives, coverage and width, $\boldsymbol{r} = (r_{AIW}, r_\epsilon)$, such that $r_{AIW} + r_\epsilon = 1$. This vector of preferences determines the properties of coverage and width of the PIs that will be the output of the target network. The general scheme of the approach can be seen in Eq. (13). In short, the hypernetwork $h(\cdot; \boldsymbol{\phi})$ generates a set of weights $\boldsymbol{\theta}^{\boldsymbol{r}}$ for the required properties $\boldsymbol{r}$ of the PI. $\boldsymbol{\theta}^{\boldsymbol{r}}$ are then used as the weights of the target network, which generates the lower and upper bounds of PIs, conditional on inputs $\mathbf{X}$.

$$\begin{aligned} \boldsymbol{\theta}^{\boldsymbol{r}} &= h(\boldsymbol{r}; \boldsymbol{\phi}) \\ \left[\hat{p}^{low_{\boldsymbol{r}}}, \hat{p}^{upp_{\boldsymbol{r}}}\right] &= t(\mathbf{X}; \boldsymbol{\theta}^{\boldsymbol{r}}) \end{aligned} \tag{13}$$

As mentioned above, the preference vector $\boldsymbol{r}$ controls the width and coverage properties of PIs. More specifically, a linear scalarization process is followed, in which the two objectives *AIW* and ϵ are linearly combined with the weights r_{AIW} and r_ϵ, respectively, as shown in Eq. (14).

$$l = r_{AIW} AIW + r_\epsilon \epsilon \tag{14}$$

As will be explained later, the HN is trained so that Eq. (14) is optimized for every different $\boldsymbol{r}$. Thus, the PIs generated by the target network of Eq. (13) are those that are optimal according to the loss function in Eq. (14). In fact, linear scalarization is a commonly used method to generate the points of the Pareto front in multi-objective optimization. In this case, different values of $\boldsymbol{r}$ will generate different points of the Pareto front, which means that the hypernetwork h is trained to learn the Pareto front, and for every $\boldsymbol{r}$, it outputs weights of a different target network, which in turn will generate PIs with the right properties. Therefore, the Pareto front is made up of solutions, each of them being a target network obtained from a different preference vector $\boldsymbol{r_i}$: $\mathbf{PF} = \{t(\cdot; \boldsymbol{\theta}^{\boldsymbol{r_1}}), t(\cdot; \boldsymbol{\theta}^{\boldsymbol{r_2}}), \ldots\}$. This approach will be called Pareto Optimal Prediction Intervals Hypernetworks (POPI-HN) because it generates solutions that are Pareto optimal.

The POPI-HN two-network structure can be observed in Fig. 2. In this case, we aim for a target network of m layers. As we can see, the hypernetwork maps the preference vector $\boldsymbol{r} = (r_{AIW}, r_\epsilon)$ into another space, by means of a multilayer perceptron (MLP) with weights $\boldsymbol{\phi} = \{\phi_1, \ldots, \phi_{m-1}, \phi_m\}$. Thus, the hypernetwork outputs the weights $\boldsymbol{\theta}^{\boldsymbol{r}} = \{\theta_1^{\boldsymbol{r}}, \ldots, \theta_{m-1}^{\boldsymbol{r}}, \theta_m^{\boldsymbol{r}}\}$ of the target network. Finally, the weights $\boldsymbol{\theta}^{\boldsymbol{r}}$ are used to obtain $\hat{PI}^{\boldsymbol{r}} = \left[\hat{p}^{low_{\boldsymbol{r}}}, \hat{p}^{upp_{\boldsymbol{r}}}\right]$ for a dependent variable given a set of features $\mathbf{X}$.

As explained, hypernetworks generate PIs with PICP and *AIW* properties that are optimal according to Eq. (14). However, in general, users need a target network that returns PIs with some particular nominal coverage (or *PINC*). Therefore, to use the HN structure of Fig. 2, it is necessary to compute a preference vector $\boldsymbol{r}$ appropriate for some given *PINC*, called $\boldsymbol{r}_{PINC}$. However, the mapping of *PINC* to $\boldsymbol{r}_{PINC}$ is not straightforward. In POPI-HN, this has been approached by empirically computing the PICP values of target networks obtained from a set of different preference vectors. To avoid overfitting, this set of associations between preference vectors and *PINC* values, called the validation front (VF), is computed using a validation set different from the training set. This allows recovering the right preference vector, given a user-required *PINC* value, by looking for the closest PICP value in the VF.

The method to generate the VF can be seen in Algorithm 2. The VF is constructed by iteratively applying the hypernetwork on v preference vectors $\boldsymbol{r_i} = (\cos(\gamma_i), \sin(\gamma_i))$ for angles γ_i evenly spaced from 0 to $\frac{\pi}{2}$ (lines 3–4). In other words, they are equally spaced two-dimensional unit vectors belonging to the upper right quadrant, where the Pareto front is located, given that the two losses (*AIW* and ϵ) are both positive. The $\boldsymbol{r_i} = (r_{AIW,i}, r_{\epsilon,i})$ vectors are further normalized so that the two preferences add to 1, as required by the hypernetwork (line 5). Then, the weights $\theta^{\boldsymbol{r_i}}$ of the target network that correspond to preference vector $\boldsymbol{r_i}$ are obtained by using the hypernetwork (line 6). Next, the target network with weights $\boldsymbol{\theta}^{\boldsymbol{r_i}}$ is applied to every instance in the validation set ($\boldsymbol{X}_{VS}$) in order to obtain a set of estimated PIs, named $\hat{\boldsymbol{PI}}^{\boldsymbol{r_i}}_{VS}$ (line 7). Then, the *AIW* and *PICP* of each PI in $\hat{\boldsymbol{PI}}^{\boldsymbol{r_i}}_{VS}$ are computed (lines 8–9), represented by $AIW(\hat{\boldsymbol{PI}}^{\boldsymbol{r_i}}_{VS})$ and $PICP(\hat{\boldsymbol{PI}}^{\boldsymbol{r_i}}_{VS}, \mathbf{y}_{VS})$, respectively. Finally, the triplet $(\boldsymbol{r}_i, AIW_i, PICP_i)$ is added to the VF (line 10). This is repeated v times. The result is the validation front $VF = \{(\boldsymbol{r}_1, AIW_1, PICP_1), \ldots, (\boldsymbol{r}_v, AIW_v, PICP_v)\}$, which associates each $\boldsymbol{r}_i$ with its corresponding AIW_i and $PICP_i$.

Fig. 3 illustrates how the VF can be used to map the required *PINC* into the appropriate preference vector. Fig. 3 is a graphical representation of the $VF = \{(\boldsymbol{r}_i, AIW_i, PICP_i)\}_{i=1}^{i=v}$. Each blue point in the figure corresponds to one element i of the VF with coordinates $(AIW_i, 1 - PICP_i)$. Let us suppose we aim to generate PIs with nominal coverage $PINC = 0.75$. The point in the VF that minimizes $|PICP - 0.75|$ (equivalently $|\epsilon - \alpha| =$

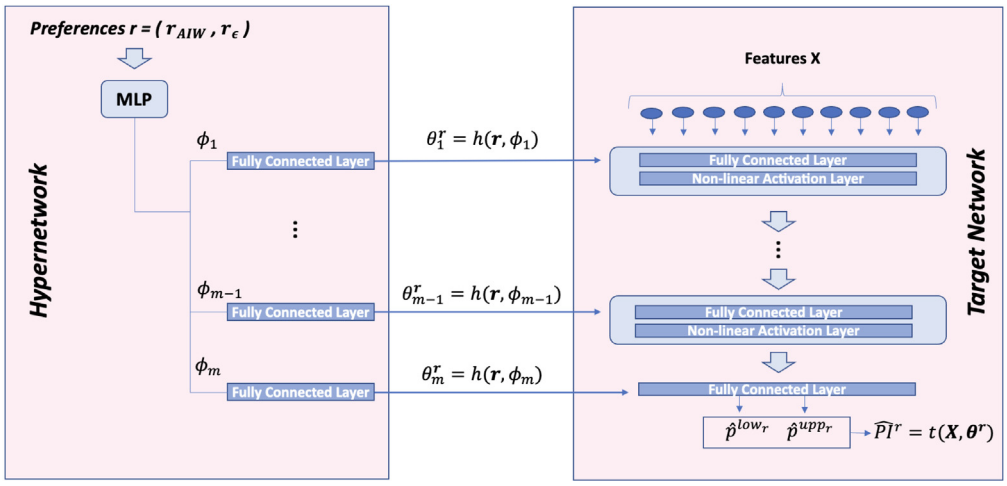

Fig. 2. POPI-HN structure.

Algorithm 2: Generation of the Validation front.

Data: $\boldsymbol{\phi}$: Weights of the hypernetwork
$VS = (\boldsymbol{X}_{VS}, \mathbf{y}_{VS})$: Validation set
v: Number of preference vectors
Result: $VF = \{(\boldsymbol{r}_1, AIW_1, PICP_1), ..., (\boldsymbol{r}_v, AIW_v, PICP_v)\}$: Validation front

1 $VF \leftarrow \{\}$
2 **for** $i = 0$ *TO* $(v\text{-}1)$ **do**
3 $\quad \gamma_i = \frac{\pi}{2} * \frac{i}{v-1}$
4 $\quad (r_{AIW,i}, r_{\epsilon,i}) \leftarrow (cos\gamma_i, sin\gamma_i)$
5 $\quad \boldsymbol{r_i} \leftarrow (\frac{r_{AIW,i}}{r_{AIW,i}+r_{\epsilon,i}}, \frac{r_{\epsilon,i}}{r_{AIW,i}+r_{\epsilon,i}})$
6 $\quad \boldsymbol{\theta}^{\boldsymbol{r_i}} \leftarrow h(\boldsymbol{r_i}; \boldsymbol{\phi})$
7 $\quad \hat{\boldsymbol{PI}}_{VS}^{\boldsymbol{r_i}} \leftarrow t(\boldsymbol{X}_{VS}; \boldsymbol{\theta}^{\boldsymbol{r_i}})$
8 $\quad AIW_i \leftarrow AIW(\hat{\boldsymbol{PI}}_{VS}^{\boldsymbol{r_i}})$
9 $\quad PICP_i \leftarrow PICP(\hat{\boldsymbol{PI}}_{VS}^{\boldsymbol{r_i}}, \mathbf{y}_{VS})$
10 $\quad VF \leftarrow VF \bigcup (\boldsymbol{r}_i, AIW_i, PICP_i)$
11 **end**

$|\epsilon - 0.25|$) is chosen. In the actual VF of Fig. 3, this corresponds with a point with $\epsilon = 1 - PICP = 0.251$ and $AIW = 0.221$, and the corresponding preference vector $\mathbf{r} = (0.679, 0.321)$. Thus, the required target *PINC* 0.75 has been assigned to the preference vector $\mathbf{r} = (0.679, 0.321)$. Now, the appropriate model $t(., h((0.679, 0.321), \boldsymbol{\phi}))$ can be used to generate PIs with the nominal coverage required. It can also be seen that the more points v in the VF, the easier it will be that one of them is close to the target *PINC*.

Algorithm 3 shows how to compute the PI for some *PINC* value ($\hat{\boldsymbol{PI}}_{\boldsymbol{PINC}}$) and some input variables $\boldsymbol{X}$ using the VF and the hypernetwork. First (lines 1–6), the preference vector $\boldsymbol{r}_{PINC}$ that generated a *PICP* value closest to the required *PINC* in the VF is selected. A minor detail is that POPI-HN has a preference/bias for PIs that satisfy the desired coverage, as this is an important objective. This is shown in line 1, so that if there is at least one element e_i in the VF that satisfies the coverage, then the closest element in the VF is obtained from those elements $PICP_k \geq PINC$ (line 2). Otherwise, the closest element will be selected from the remaining elements in the VF (line 4). Next, the hypernetwork h is used together with the selected preference vector to obtain the weights $\boldsymbol{\theta}^{\boldsymbol{r}_{PINC}}$ of the target network (line 7). Finally, in line 8 the target network is applied to the inputs $\boldsymbol{X}$ in order to estimate the PI for the required *PINC* (that is, the lower and upper bounds of the PI).

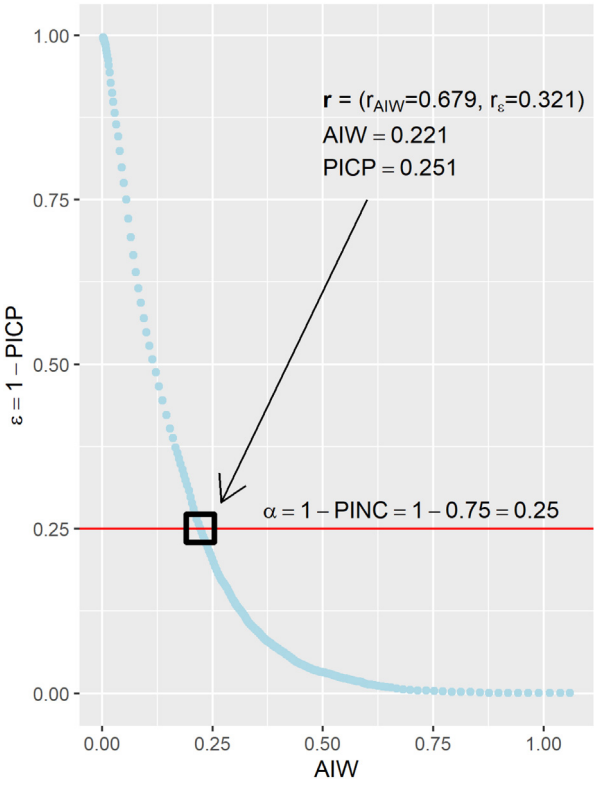

Fig. 3. Using the validation front to map *PINC* = 0.75 into the corresponding preference vector $r = (0.679, 0.321)$.

4.2. Training the Pareto Optimal Prediction Interval network

The hypernetwork weights $\boldsymbol{\phi}$ are the result of a gradient descent training process on the training dataset. Algorithm 4 summarizes the training algorithm. Line 1 initializes the weights $\boldsymbol{\phi}$. Also, although the method loops along a specific maximum number of epochs (*maxEpochs* at line 3), there is an early-stopping mechanism to avoid overfitting [37]. This mechanism is based on the selection of the model corresponding to the epoch where its

Algorithm 3: POPI-HN generation of prediction intervals for some input variables $\boldsymbol{X}$ and some nominal coverage PINC.

Data: $VF = \{(\boldsymbol{r}_i, AIW_i, PICP_i)\}_{i=1}^{i=v}$: Validation front from Alg. 2
PINC: target nominal coverage
$\boldsymbol{X}$: input variables
Result: $\hat{\boldsymbol{PI}}_{\boldsymbol{PINC}} = [\hat{p}^{low_{\boldsymbol{r}_{PINC}}}, \hat{p}^{upp_{\boldsymbol{r}_{PINC}}}]$

1 **if** $\exists e_i \in VF$ *such that* $PICP_i \geq PINC$ **then**
2 $(\boldsymbol{r}_{k_{PINC}}, AIW_{k_{PINC}}, PICP_{k_{PINC}}) = e_{k_{PINC}} \leftarrow \underset{e_k \in VF \text{ subject to } PICP_k \geq PINC}{\text{argmin}} (|PICP_k - PINC|)$
3 **else**
4 $(\boldsymbol{r}_{k_{PINC}}, AIW_{k_{PINC}}, PICP_{k_{PINC}}) = e_{k_{PINC}} \leftarrow \underset{e_k \in VF \text{ subject to } PICP_k < PINC}{\text{argmin}} (|PICP_k - PINC|)$
5 **end**
6 $\boldsymbol{r}_{PINC} \leftarrow \boldsymbol{r}_{k_{PINC}}$
7 $\boldsymbol{\theta}^{\boldsymbol{r}_{PINC}} \leftarrow h(\boldsymbol{r}_{PINC}; \boldsymbol{\phi})$
8 $\hat{\boldsymbol{PI}}_{\boldsymbol{PINC}} = [\hat{p}^{low_{\boldsymbol{r}_{PINC}}}, \hat{p}^{upp_{\boldsymbol{r}_{PINC}}}] \leftarrow t(\boldsymbol{X}; \boldsymbol{\theta}^{\boldsymbol{r}_{PINC}})$
9 Return($\hat{\boldsymbol{PI}}_{\boldsymbol{PINC}}$)

Algorithm 4: Training POPI-HN.

Data: TS = ($\mathbf{X}_{TS}$, $\mathbf{y}_{TS}$): Training Set
VS: Validation Set
Result: $\boldsymbol{\phi}$: Hypernetwork weights

1 $\boldsymbol{\phi}_{best} \leftarrow \boldsymbol{\phi} \leftarrow initializeWeights()$
2 $hv_{best} \leftarrow 0$
3 **for** *epoch = 1 TO maxEpochs* **do**
4 $(r_{AIW}, r_\epsilon) = \boldsymbol{r} \sim \text{Dirichlet}(\beta)$
5 $\boldsymbol{\theta}^{\boldsymbol{r}} \leftarrow h(\boldsymbol{r}; \boldsymbol{\phi})$
6 $\hat{\boldsymbol{PI}}_{\boldsymbol{TS}} \leftarrow t(\mathbf{X}_{TS}; \boldsymbol{\theta}^{\boldsymbol{r}})$
7 $AIW_{\boldsymbol{\theta}^{\boldsymbol{r}}} \leftarrow AIW(\hat{\boldsymbol{PI}}_{\boldsymbol{TS}})$
8 $\epsilon_{\boldsymbol{\theta}^{\boldsymbol{r}}} \leftarrow 1 - PICP_S(\hat{\boldsymbol{PI}}_{\boldsymbol{TS}}, \mathbf{y}_{TS})$
9 $\text{loss} = r_{AIW} AIW_{\boldsymbol{\theta}^{\boldsymbol{r}}} + r_\epsilon \epsilon_{\boldsymbol{\theta}^{\boldsymbol{r}}}$
10 $\boldsymbol{\phi} \leftarrow \text{update}(\text{loss}, \boldsymbol{\phi})$
11 $VF \leftarrow$ Algorithm2($\boldsymbol{\phi}$, VS) # Generate the Validation front
12 $hv' \leftarrow \text{hyper-volume}(VF)$
13 **if** $hv' > hv_{best}$ **then**
14 $hv_{best} \leftarrow hv'$
15 $\boldsymbol{\phi}_{best} \leftarrow \boldsymbol{\phi}$
16 **end**
17 **end**
18 Return($\boldsymbol{\phi}_{best}$)

performance on the VF is best. The quality of the VF is measured by employing the hyper-volume [38]. The best hyper-volume so far, and the associated weights, are stored in variables hv_{best} and $\boldsymbol{\phi}$, respectively. They are initialized in lines 1–2 (0 is the worst hyper-volume value) and they are updated in the training loop every time the performance on the VF is improved (lines 13–16, as explained later).

Next, the method loops for a number of epochs (line 3) where weights $\boldsymbol{\phi}$ are going to be incrementally adjusted. First, a preference vector $\boldsymbol{r}$ is sampled from a Dirichlet distribution of parameter $\beta \in R^2$ (line 4). Then, the hypernetwork is used with the current weights $\boldsymbol{\phi}$ and the preference vector $\boldsymbol{r}$ to obtain the weights $\boldsymbol{\theta}^{\boldsymbol{r}}$ of the target network t (line 5), which is then used to obtain the PIs for each training instance $TS = \{\}_{i=1}^{i=n}$ (line 6). The two losses AIW and ϵ are computed for them (lines 7 and 8). As mentioned above, a linear scalarization process is used to obtain the solutions of the Pareto front. That is, a loss is computed as a weighted aggregation of AIW and ϵ with weights $\boldsymbol{r}$ (line 9), using the loss defined in Eq. (14). This loss is used to update the hypernetwork weights $\boldsymbol{\phi}$ by means of gradient descent (line 10). Finally, a VF is built from the validation set as presented in Algorithm 2 (line 11), whose hyper-volume is compared to the current best hyper-volume (lines 12–13). If it is better, the best hypernetwork weights $\boldsymbol{\phi}$ will be updated (lines 14–15). These weights will be returned by the method (line 18).

For the POPI-HN training process, AIW and $\epsilon = 1 - PICP$ have to be computed. In the case of coverage, a soft version of $PICP$ is used to estimate $\epsilon = 1 - PICP_S$, which will give more stability to the model during training. This soft version improves the training process, especially at the early stage of the training and when dealing with a difficult task where it is possible to have a low coverage at the beginning.

The soft $PICP_S$ is computed in the same way as in Algorithm 1 (lines 2–8). The soft vector of the captured points $\mathbf{K}_S$ is computed as $\mathbf{K}_{SU} \odot \mathbf{K}_{SL} = sigmoid((\mathbf{y}_{TS} - \hat{\mathbf{p}}^{low_{\boldsymbol{r}}})s) \odot sigmoid((\hat{\mathbf{p}}^{upp_{\boldsymbol{r}}} - \mathbf{y}_{TS})s)$, with $\mathbf{y}_{TS}$ as the target vector of values, $\hat{\mathbf{p}}^{low_{\boldsymbol{r}}}$ and $\hat{\mathbf{p}}^{upp_{\boldsymbol{r}}}$ the lower and upper bounds of the PI generated from the preference vector $\boldsymbol{r}$, and s representing a softening factor (a factor of 160 was used in this work). An element-wise product between $\mathbf{K}_{SU}$ and $\mathbf{K}_{SL}$ (which stand for the difference between the upper bound of the PI $\hat{\mathbf{p}}^{upp_{\boldsymbol{r}}}$ and the dependent variable $\mathbf{y}_{TS}$, and between $\mathbf{y}_{TS}$ and the lower bound $\hat{\mathbf{p}}^{low_{\boldsymbol{r}}}$ followed by a softening factor s multiplication and a sigmoid function application) will generate $\mathbf{K}_S$. Finally, the mean of elements across the vector $\mathbf{K}_S$ will result in a soft approximation of the coverage $PICP_S$.

Regarding the width, the value of the AIW will be calculated by normalizing its value with the range of the dependent variable (Eq. (5)). First, a width vector is formed as the difference between the vector of upper values $\hat{\mathbf{p}}^{upp_{\boldsymbol{r}}}$ and lower values $\hat{\mathbf{p}}^{low_{\boldsymbol{r}}}$ of the PIs. Next, the mean of elements across the vector is applied to obtain the AIW (Eq. (3)) and it is divided by the range of the dependent variable: $max(\mathbf{y}) - min(\mathbf{y})$.

With these implementations for the coverage and width, the loss-space will be bounded between zero and one for every metric. This will be useful for the POPI-HN validation process, as the hyper-volume metric needs a reference point to be computed, measuring the quality of the non-dominated front by computing the area with respect to the reference point. With this approach, point (1, 1) can be used as a reference, resulting in homogeneity between different experiments.

Finally, to encourage the reproducibility of the method, the implementation of POPI-HN in Python has made open access available to practitioners in [39].

5. Benchmarking experiments

In this section, the performance of POPI-HN when building several PIs for different target coverages will be evaluated over eight different benchmarking regression datasets. As a baseline, the quality of the PIs will be compared with the ones obtained with QD-DNN models. Therefore, the performance of the multi-objective method (POPI-HN) will be compared with a single-objective method (QD-DNN). Notice that QD-DNN could have an advantage over POPI-HN as it is trained for a specific set of *PINCs*. However, the main focus of POPI-HN is to obtain a complete PF of solutions for all possible *PINC* while having good coverage and width performance.

Table 1
Benchmarking regression datasets.

Dataset	n	D
California[a]	20,640	8
Concrete[b]	1,030	8
Energy[b]	738	8
Kin8nm[c]	8,192	8
Power Plant[b]	9,468	4
Protein[b]	45,730	9
Superconductor[b]	21,263	81
Yacht[b]	308	6

[a]Scikit-learn Datasets: https://scikit-learn.org/stable/datasets/real_world.html.
[b]UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/index.php.
[c]DELVE Repository: https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html.

5.1. Experimental setup

Six different PIs will be built with each regression dataset for comparison purposes: PINC 70%, 75%, 80%, 85%, 90% and 95%. Thus, twelve final values will be obtained in the QD-DNN output layer. However, notice that POPI-HN is capable of generating a PI for every sought *PINC*, therefore the six PIs will be obtained as described in Algorithm 3.

Eight different regression task datasets will be used to compare QD-DNN with POPI-HN. Table 1 summarizes them with information about the number of instances n and the dimension or number of features D. As can be seen, the number of observations in the datasets ranges from a few hundred in Yacht to more than forty-five thousand in Protein. Regarding dimensionality, in most of the domains, it is not large, apart from Superconductor. For each of these eight datasets, the dependent and independent variables have been standardized before the training process.

In each case, it is necessary to select the correct value of the different hyper-parameters, including the number of hidden layers, number of neurons per layer, learning rate, the size of the batch, etc.

For this purpose, the following methodology is applied: data are randomly split into a train and test subsets representing 80% and 20% of the complete dataset, respectively, and a grid search selection is made. Regarding QD-DNN, the best model according to the sum of losses (Eq. (7)) is saved in every epoch. On the other hand, the metric for model selection of POPI-HN is the hyper-volume indicator [38], which in this case is the area of the non-dominated front with respect to a certain reference point. The point (1, 1) (the upper right corner) will be used because, as mentioned above, ϵ and the normalized *AIW* are in the range of 0–1. The larger the hyper-volume, the better, because that is the direction in which the two objectives are optimized. Thus, the model with largest hyper-volume on the VF will be selected.

Regarding the tuning process, a grid search was employed for the hyper-parameter selection. The learning rate was 5×10^{-5} in all cases, using ADAM [40] as the optimizer and Elu [41] as the non-linear activation function. The weight decay regularization technique [42] has also been used. The rest of the selected hyper-parameters can be seen in Table 2.

Notice that for datasets with the smallest number of instances (Concrete, Energy, and Yacht), all training sets are used as a single batch.

When the adjustment process is finished, and QD-DNN and POPI-HN models have their respective hyper-parameters for each dataset, testing is carried out. In this case, datasets are randomly split in an 80/10/10 proportion, for training, validation and test. That is, 80% of the original data are used for training, 10% for validation, and the remaining 10% for test. The validation set is used for early-stopping [37] and hyper-parameter tuning, and the test set for model evaluation.

Models will start training with the respective dataset and corresponding selected hyper-parameters, will be stopped with the best value of Quality-Driven loss or hyper-volume and the unbiased performance of the models will be obtained in the test subset. This procedure will be repeated 20 times, each time with a different random split in the datasets and network initialization.

The form of *PICP* presented in Eq. (2), and the non-normalized, non-captured version of *AIW* (Eq. (3)) will be used to measure the quality of the obtained PIs.

To deal with the possible instability of the models due to the randomized sets and net initialization, the test will be repeated 20 different times. The results will be aggregated (average) across these 20 runs. Tables 3 and 4 display the mean values together with the standard error.

Finally, to verify the reliability of the results, some statistical tests will be applied to the results. To determine that *PICP* is not worse than *PINC* (with a 5% significance level), the t-test [43] will be used. To fulfill the assumptions of the test, normality in the values was also proven by using the Shapiro–Wilk test [44], whereas independence is present due to the randomization of the experiments. Tests were satisfied, not being able to reject the null normality hypothesis in any case for the same significance level.

When comparing the *AIW* values, the non-parametric Mann–Whitney U test [45] has been employed to determine which method produces narrower PIs. This test will tell if one randomly selected value from one group (*AIW* from QD-DNN or POPI-HN) would be smaller or larger than one value from the other group. That is, to check if the differences in *AIW* are significant.

To analyze the variability in the results from the different runs, the non-parametric Levene's test [46] has been used to compare which method produces more variance in their results.

5.2. Results

In this section, the performance of QD-DNN and POPI-HN for the eight different datasets will be presented and discussed. It is important to remember that QD-DNN has been specifically trained for the six PIs, while POPI-HN obtains the complete set of solutions (all possible *PINC* values). The VF has been obtained as shown in Section 4, and the six PIs will be extracted from there (see Algorithm 3). An illustrative example can be found in Fig. 4, where a VF and its corresponding test PIs metrics are shown for a run in each of the eight datasets.

The results will be presented as follows. The two main objectives regarding PI construction will be evaluated: coverage (*PICP*) and width (*AIW*) for the six PIs (i.e. six *PINC* values). Values will be shown as the mean across the 20 different runs, accompanied by their standard error.

5.2.1. Coverage results

PICP values can be seen in Table 3. The average value is shown in bold when the alternative hypothesis ($PICP \leq PINC$) cannot be accepted by the t-test (at 5% confidence level).

As can be seen, both QD-DNN and POPI-HN deliver a good performance in terms of coverage, being able to reach the *PINC* sought in all domains and most of the *PINC* values. Both methods fail in some cases. For example, in the California dataset, QD-DNN is unable to overtake the *PINC* for 90% and 95%. In these same *PINC* values POPI-HN fails for the Concrete domain, and *PINC* 95% with the Kin8nm data.

The case where the *PICP* is farther away from the *PINC* for POPI-HN is in the dataset with a smaller number of instances: Concrete. When the number of instances is big enough, POPI-HN does not fail in coverage. It must be noticed that POPI-HN has a high dependency on the VF, therefore, we can assume that, with a bigger number of observations, the VF will be more representative of the complete dataset.

Table 2
Hyper-parameters selected for each model and dataset.

Dataset	Method	Hidden layers	Neurons per layer	Weight decay	Batch-size	Lambda
California	QD-DNN	4	150	0.	2500	0.05
	POPI-HN	6	200	0.	2500	–
Concrete	QD-DNN	4	200	0.1	Train set	0.05
	POPI-HN	4	100	0.	Train set	–
Energy	QD-DNN	4	150	0.	Train set	0.05
	POPI-HN	4	100	0.	Train set	–
Kin8nm	QD-DNN	5	200	0.1	1500	0.01
	POPI-HN	4	150	0.	1500	–
Power Plant	QD-DNN	3	150	0.1	1500	0.05
	POPI-HN	4	100	0.	1500	–
Protein	QD-DNN	4	150	0.	2500	0.05
	POPI-HN	5	200	0.	2500	–
Superconductor	QD-DNN	4	150	0.	2500	0.05
	POPI-HN	5	150	0.	2500	–
Yacht	QD-DNN	3	150	0.	Train set	0.01
	POPI-HN	4	100	0.	Train set	–

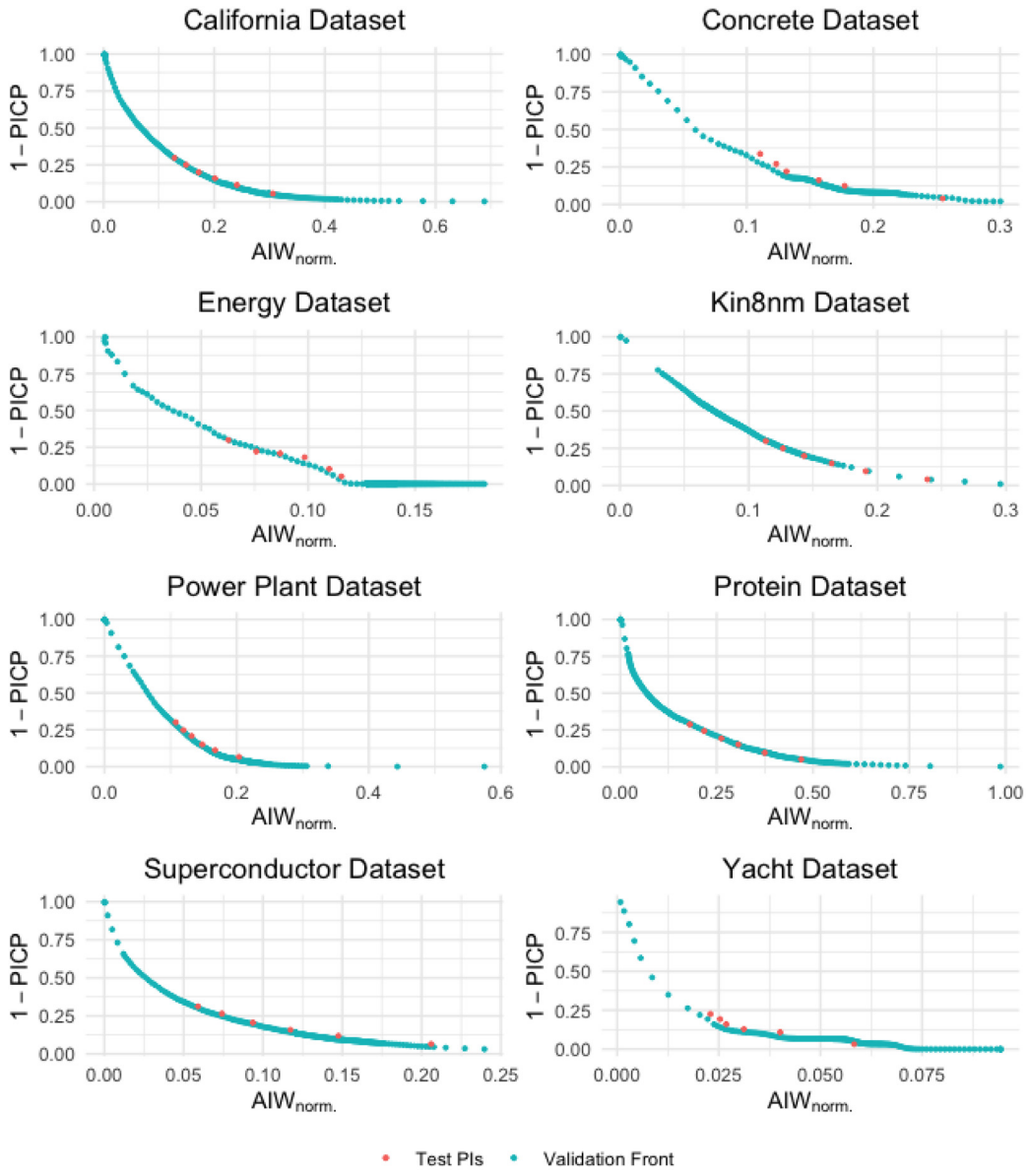

Fig. 4. Example of POPI-HN validation fronts (blue) and Test PIs metrics (red) for each dataset.

Table 3
Mean *PICP* ± one standard error. Values presented in boldface are statistically significant using the t-test.

Dataset	Method	PINC 70	PINC 75	PINC 80	PINC 85	PINC 90	PINC 95
California	QD-DNN	**0.71** ± 0.01	**0.76** ± 0.01	**0.80** ± 0.01	**0.85** ± 0.01	0.89 ± 0.01	0.94 ± 0.01
	POPI-HN	**0.70** ± 0.01	**0.75** ± 0.01	**0.80** ± 0.01	**0.85** ± 0.01	**0.90** ± 0.01	**0.95** ± 0.01
Concrete	QD-DNN	**0.73** ± 0.04	**0.77** ± 0.04	**0.81** ± 0.04	**0.86** ± 0.03	**0.90** ± 0.03	**0.94** ± 0.02
	POPI-HN	**0.69** ± 0.06	**0.74** ± 0.06	**0.80** ± 0.06	**0.84** ± 0.05	0.88 ± 0.05	0.93 ± 0.03
Energy	QD-DNN	**0.91** ± 0.03	**0.94** ± 0.05	**0.96** ± 0.03	**0.98** ± 0.02	**0.99** ± 0.01	**1.00** ± 0.01
	POPI-HN	**0.73** ± 0.07	**0.78** ± 0.06	**0.83** ± 0.05	**0.87** ± 0.06	**0.91** ± 0.05	**0.95** ± 0.03
Kin8nm	QD-DNN	**0.72** ± 0.01	**0.77** ± 0.02	**0.82** ± 0.01	**0.87** ± 0.01	**0.91** ± 0.01	**0.96** ± 0.01
	POPI-HN	**0.70** ± 0.02	**0.74** ± 0.02	**0.80** ± 0.02	**0.85** ± 0.02	**0.90** ± 0.01	0.94 ± 0.01
Power Plant	QD-DNN	**0.72** ± 0.02	**0.77** ± 0.02	**0.81** ± 0.02	**0.86** ± 0.01	**0.91** ± 0.01	**0.95** ± 0.01
	POPI-HN	**0.71** ± 0.02	**0.76** ± 0.02	**0.81** ± 0.02	**0.85** ± 0.01	**0.90** ± 0.01	**0.95** ± 0.01
Protein	QD-DNN	**0.72** ± 0.01	**0.76** ± 0.01	**0.80** ± 0.01	**0.85** ± 0.01	**0.90** ± 0.01	**0.95** ± 0.00
	POPI-HN	**0.70** ± 0.01	**0.75** ± 0.01	**0.80** ± 0.01	**0.85** ± 0.01	**0.90** ± 0.01	**0.95** ± 0.00
Superconductor	QD-DNN	**0.75** ± 0.01	**0.79** ± 0.01	**0.83** ± 0.01	**0.87** ± 0.01	**0.92** ± 0.01	**0.96** ± 0.00
	POPI-HN	**0.71** ± 0.01	**0.76** ± 0.01	**0.81** ± 0.01	**0.85** ± 0.01	**0.90** ± 0.01	**0.95** ± 0.01
Yacht	QD-DNN	**0.86** ± 0.07	**0.87** ± 0.07	**0.90** ± 0.07	**0.92** ± 0.06	**0.95** ± 0.05	**0.98** ± 0.02
	POPI-HN	**0.75** ± 0.09	**0.78** ± 0.09	**0.81** ± 0.05	**0.84** ± 0.05	**0.88** ± 0.06	**0.94** ± 0.05

Table 4
Mean *AIW* ± one standard error. Values presented in boldface are statistically significant using U-test.

Dataset	Method	PINC 70	PINC 75	PINC 80	PINC 85	PINC 90	PINC 95
California	QD-DNN	0.58 ± 0.01	0.65 ± 0.01	**0.74** ± 0.01	**0.86** ± 0.01	**1.02** ± 0.01	**1.30** ± 0.03
	POPI-HN	**0.55** ± 0.02	**0.63** ± 0.02	**0.74** ± 0.03	0.88 ± 0.03	1.07 ± 0.03	1.39 ± 0.05
Concrete	QD-DNN	0.69 ± 0.02	0.77 ± 0.03	0.86 ± 0.02	0.96 ± 0.02	1.10 ± 0.02	1.34 ± 0.03
	POPI-HN	**0.53** ± 0.07	**0.60** ± 0.07	**0.68** ± 0.08	**0.77** ± 0.08	**0.89** ± 0.10	**1.12** ± 0.20
Energy	QD-DNN	0.18 ± 0.01	0.19 ± 0.01	0.22 ± 0.01	0.25 ± 0.01	0.28 ± 0.01	0.34 ± 0.02
	POPI-HN	**0.09** ± 0.01	**0.10** ± 0.01	**0.12** ± 0.01	**0.13** ± 0.01	**0.15** ± 0.02	**0.18** ± 0.02
Kin8nm	QD-DNN	0.80 ± 0.01	0.89 ± 0.01	1.00 ± 0.01	1.12 ± 0.01	1.29 ± 0.02	1.53 ± 0.02
	POPI-HN	**0.51** ± 0.02	**0.56** ± 0.02	**0.63** ± 0.03	**0.71** ± 0.04	**0.81** ± 0.03	**0.95** ± 0.05
Power Plant	QD-DNN	0.51 ± 0.00	0.56 ± 0.00	0.62 ± 0.00	0.69 ± 0.00	0.79 ± 0.01	0.95 ± 0.01
	POPI-HN	**0.42** ± 0.02	**0.46** ± 0.01	**0.52** ± 0.01	**0.58** ± 0.01	**0.67** ± 0.02	**0.80** ± 0.03
Protein	QD-DNN	0.72 ± 0.01	0.83 ± 0.01	0.96 ± 0.01	1.13 ± 0.02	1.35 ± 0.02	**1.73** ± 0.03
	POPI-HN	**0.60** ± 0.02	**0.74** ± 0.02	**0.89** ± 0.02	**1.08** ± 0.02	**1.33** ± 0.03	**1.72** ± 0.05
Superconductor	QD-DNN	0.57 ± 0.02	0.64 ± 0.01	0.73 ± 0.02	0.85 ± 0.02	1.01 ± 0.02	1.25 ± 0.02
	POPI-HN	**0.44** ± 0.02	**0.52** ± 0.02	**0.61** ± 0.02	**0.71** ± 0.02	**0.86** ± 0.03	**1.06** ± 0.03
Yacht	QD-DNN	0.08 ± 0.01	0.09 ± 0.01	0.10 ± 0.01	0.12 ± 0.01	0.14 ± 0.01	0.17 ± 0.02
	POPI-HN	**0.05** ± 0.01	**0.06** ± 0.01	**0.07** ± 0.01	**0.08** ± 0.01	**0.10** ± 0.02	**0.12** ± 0.02

Regarding datasets with a higher number of instances, we can see that the coverage is not under the *PINC* by more than 0.01, for example, with QD-DNN in California and with POPI-HN in Kin8nm datasets.

Generally, both methods perform in a correct way regarding coverage. Also, it can be noticed how the standard error is larger in datasets with a small number of observations, such as Concrete, Energy, or Yacht.

5.2.2. Width results

On the other side, *AIW* results are shown in Table 4. Values are presented in boldface when it is possible to determine through the U-test if there is statistical evidence that *AIW* values produced by POPI-HN are smaller than those of QD-DNN, or the other way around. When it is not possible to reject the null hypothesis that both values are the same, both values are shown in bold.

Results show how POPI-HN produces narrower intervals in all the *PINC* values for the Concrete, Energy, Kin8nm, Power Plant, Protein, Superconductor, and Yacht datasets. In the California dataset, QD-DNN and POPI-HN compete with each other regarding the *AIW* for different *PINC* values. In summary, POPI-HN also shows a good performance regarding the *AIW*.

It might be thought that the direct optimization of the *AIW* by POPI-HN has some performance advantages in a multi-objective context over the optimization done by QD-DNN. Besides, because for multiple PI estimation the QD loss is a sum of losses, there is a possibility that the lowest loss is obtained when improving one PI but worsening another one. In this sense, POPI-HN minimizes its loss for the complete set of solutions.

Table 5
AIW percentual change by using POPI-HN vs. QD-DNN (negative values imply that POPI-HN is better than QD-DNN). Values displayed only if there is significant evidence of *AIW* being different between methods (Table 4).

Dataset	PINC 70	PINC 75	PINC 80	PINC 85	PINC 90	PINC 95
California	**−5%**	**−2%**	–	2%	4%	7%
Concrete	**−23%**	**−23%**	**−20%**	**−20%**	**−19%**	**−16%**
Energy	**−47%**	**−46%**	**−47%**	**−47%**	**−48%**	**−49%**
Kin8nm	**−37%**	**−37%**	**−37%**	**−36%**	**−37%**	**−38%**
Power Plant	**−18%**	**−17%**	**−16%**	**−16%**	**−15%**	**−16%**
Protein	**−16%**	**−11%**	**−7%**	**−4%**	**−2%**	–
Superconductor	**−22%**	**−18%**	**−17%**	**−16%**	**−15%**	**−15%**
Yacht	**−31%**	**−29%**	**−28%**	**−30%**	**−31%**	**−31%**

Finally, to easily check the mean decrease (or increase) of the PI width produced by POPI-HN versus QD-DNN, the percentage change in the *AIW* is stated in Table 5.

As can be seen, in general, the biggest improvements come with medium coverage intervals, like *PINC* 70% and 75%, and the improvement starts to decline a little for higher coverage intervals like *PINC* 90% and 95%. This might be due to the ease of improvement in the medium part of the Pareto front. When dealing with the highest coverages, more extreme values appear, and the width cannot be minimized in the same way while keeping the coverage level.

To summarize, the improvement in the *AIW* metric is clear by using POPI-HN, in addition to being able to generate a PI for every sought *PINC* with a single trained model. Reductions in the *AIW*

Table 6
Statistical analysis for the variability in the results.

Dataset	Metric	PINC 70	PINC 75	PINC 80	PINC 85	PINC 90	PINC 95
California	PICP	=	=	=	=	=	=
	AIW	–	=	–	–	–	–
Concrete	PICP	=	=	=	=	=	=
	AIW	=	=	=	=	–	–
Energy	PICP	–	=	=	–	–	–
	AIW	=	=	=	=	=	=
Kin8nm	PICP	=	=	=	=	=	=
	AIW	=	=	–	–	–	–
Power Plant	PICP	=	=	=	=	=	=
	AIW	–	–	–	–	–	–
Protein	PICP	=	=	=	=	=	=
	AIW	=	=	=	=	=	–
Superconductor	PICP	=	=	=	=	=	=
	AIW	=	=	=	=	=	=
Yacht	PICP	=	=	=	=	=	-
	AIW	=	=	=	=	=	=

go up to almost 50%, whereas in the few cases where the width increases, it does it by no more than 7%.

5.2.3. Variability results

The last statistical analysis will be focused on the variability of the results. As mentioned before, the non-parametric Levene's test is employed to check whether the results from POPI-HN have a bigger variance than the ones from QD-DNN or vice versa.

Table 6 summarizes this statistical analysis. An equal sign is employed to indicate that the variance from the results is statistically the same. On the other side, a minus sign indicated that the variance of the results from QD-DNN is smaller than the one from POPI-HN.

As we can see, the variance regarding the *PICP* is the same for both POPI-HN and QD-DNN in most of the datasets. Only in datasets with a low number of observations (Energy and Yacht) the variance of the results from POPI-HN is higher in some PINC values. Concerning the *AIW*, POPI-HN tends to throw a higher variability in the results in a bigger number of domains, like California, Kin8nm, or Power Plant.

This behavior may be due to the own nature of the developed methodology, as we select a point in the VF where the coverage is achieved, giving robustness to this metric but allowing more variability in the PI width. From all statistical tests that have been made, we consider that, even if in some cases the variability of the results is higher for POPI-HN, it is compensated by the good average *PICP* and *AIW* performance described in Tables 3 and 4.

5.2.4. POPI-HN hyper-parameter sensitivity

After the performance analyses carried out in the previous sections, we aim now to study the hyper-parameter sensitivity of the proposed POPI-HN in the different employed datasets. The sensitivity study is done with respect to the number of hidden layers and neurons per layer, the main hyper-parameters of POPI-HN.

We make use of the following procedure to obtain the sensitivity results. First, we train POPI-HN for a given set of hyper-parameters (number of layers: 3, 4, 5, and 6; number of hidden neurons per layer: 50, 100, 150, and 200). Periodically, we compute the hyper-volume in the VF, saving the model with the highest one. We use this trained POPI-HN to compute a test front with the test dataset. The computation of this test front is analogous to the computation of the validation front (Algorithm 2), i.e., the trained $h(\cdot;\boldsymbol{\phi})$ is fed with different preference vectors $\boldsymbol{r_i}$ to obtain the target network weights $\boldsymbol{\theta}^{\boldsymbol{r}_i}$. With these weights, PIs are obtained for the test set instances, being able to calculate *PICP* and *AIW*, and generate a test front. Finally, the hyper-volume of the test front is obtained and used to study the sensitivity. This process is repeated for all hyper-parameter configurations.

Test hyper-volume for different hyper-parameter configurations in each dataset are shown in Fig. 5. The color scale is common across all datasets, with the lowest value representing the minimum test hyper-volume in light yellow, and the highest value being the maximum test hyper-volume in dark blue.

Generally, we can see how the hyper-parameter sensitivity is dependent on the dataset employed. For example, in the California, Superconductor, or Yacht datasets, sensitivity on the test hyper-volume is not present regarding the number of neurons or layers.

However, there are other datasets where this sensitivity appears, like Energy, Power Plant, or Protein. In particular, in the Power Plant dataset, the hyper-volume seems to decrease when increasing the number of neurons per layer, whereas in the Protein dataset the hyper-volume tends to increase when increasing the number of hidden layers.

Regardless of the observed sensitivity, the hyper-parameter selection process used in this work (model with the highest hyper-volume in the VF is selected, as explained in Section 5.1), allows finding the best configuration for POPI-HN, obtaining an optimal PI for any desired coverage.

5.2.5. Time performance

We will finish this section by analyzing the running time performance of the proposed POPI-HN. First of all, we make some comments regarding the time complexity of the algorithm.

Notice that POPI-HN is composed of two different NNs: the hypernetwork and the target network. During the training time, the hypernetwork has the same time complexity as a classical DNN, as shown in Algorithm 4. That is, forward and backward (gradient descent) passes are applied across several epochs (one epoch goes through all instances in the training set). The only important additional steps (compared to standard feed-forward networks) are the computation of the target network weights employing the hypernetwork and the computation of hyper-volume required for POPI-HN early-stopping. The former is step 5 of Algorithm 4: $\boldsymbol{\theta}^{\boldsymbol{r}} \leftarrow h(\boldsymbol{r};\boldsymbol{\phi})$ and the later is executed in line 12. Also, gradient descent is applied to the weights of the hypernetwork (not the target network). Similar considerations can be made for prediction/inference (forward pass) with POPI-HN having two extra steps (see Algorithm 3): the $\boldsymbol{\theta}^{\boldsymbol{r}_{PINC}} \leftarrow h(\boldsymbol{r}_{PINC};\boldsymbol{\phi})$ step in the forward pass (line 7) and also the selection of the point in the validation front (lines 1–5 of Algorithm 3).

In any case, actual computing times are going to depend on many factors, such as the network complexity (number of layers and neurons) of the networks, which might be different for QD-DNN and POPI-HN, due to hyper-parameter tuning being carried out individually for each kind of network and each problem. The use of GPUs will also affect actual training and testing times, as many computations can be carried out in parallel. Differences between large and small networks may not be noticed, as they are run in parallel, as far as they fit within GPU memory. Actual training and testing times for the selected models are presented in Table 7. Notice that both POPI-HN and QD-DNN were trained the same amount of epochs in each bench-marking dataset, as explained in Algorithm 4 of Section 4.2. Although the best model selected by early-stopping might correspond to one with lower epochs (the one that obtains the best validation Quality-Driven loss for QD-DNN or hyper-volume for POPI-HN), both are first trained for a maximum number of epochs. All the experiments were carried out in an Intel Xeon Silver 4110 CPU at 2.10 GHz with 125 Gb RAM memory and an NVIDIA GeForce GTX 1080 GPU.

In general, training time differences are minimal between POPI-HN and QD-DNN. When training in the datasets with a

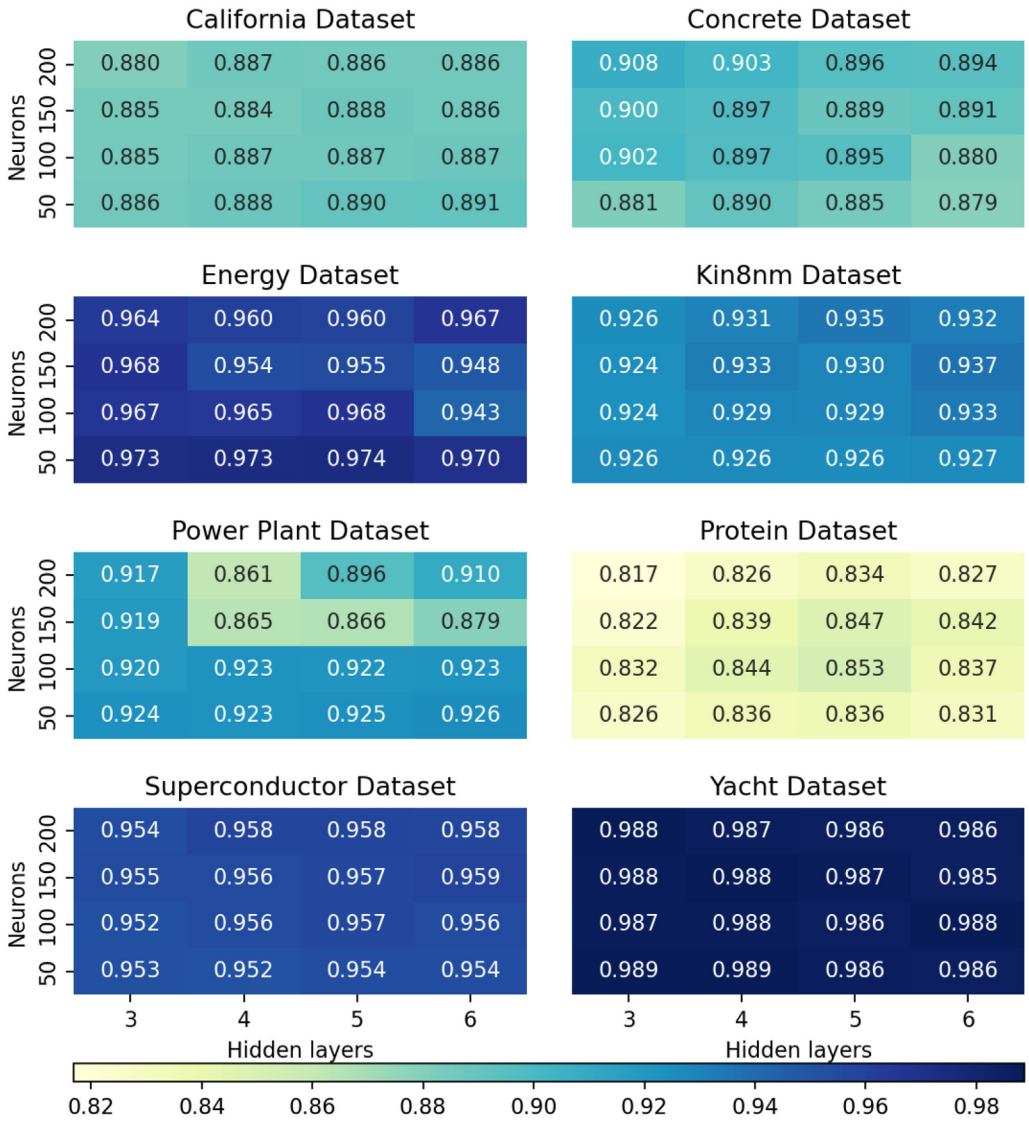

Fig. 5. Test hyper-volume for different hyper-parameter configurations in each dataset.

Table 7
Training and testing time for each final model and dataset.

Dataset	Method	Training time (s)	Testing time (s)
California	QD-DNN	1519.2	0.8
	POPI-HN	2136.6	0.4
Concrete	QD-DNN	418.3	0.3
	POPI-HN	396.6	0.1
Energy	QD-DNN	338.5	0.9
	POPI-HN	370.4	0.1
Kin8nm	QD-DNN	1696.2	0.8
	POPI-HN	1607.1	0.1
Power Plant	QD-DNN	2250.8	0.9
	POPI-HN	1832.9	0.2
Protein	QD-DNN	3721.1	1.5
	POPI-HN	4085.6	1.3
Superconductor	QD-DNN	2674.3	1.0
	POPI-HN	2807.7	0.6
Yacht	QD-DNN	243.6	0.1
	POPI-HN	248.8	0.1

higher number of observations (California, Protein, Superconductor), POPI-HN is slightly slower than QD-DNN. However, this may be due to the hyper-volume computation for saving the best model in a precise epoch, a computational expensive metric with time complexity of $\mathcal{O}(2v + v \log v)$, where 2 represents the number of dimensions in our space (number of objectives to minimize, i.e., *AIW* and $\epsilon = 1 - PICP$), and v is the number of points in the validation front [47].

Practitioners could choose to train POPI-HN for a predefined number of epochs without periodically computing the hyper-volume to reduce computation costs. Finally, regarding testing times, both methods are relatively fast, with no large differences among them.

6. Conclusions

In this work, a methodology for estimating Pareto Optimal Prediction Intervals with Hypernetworks has been introduced: POPI-HN. This method can estimate a complete set of Pareto Optimal solutions for the PI coverage–width trade-off in regression problems, that is, a Pareto front, treating the task as a multi-objective problem.

POPI-HN is formed by two deep neural networks: the hypernetwork, which takes a preference vector for the coverage–width trade-off as its input, and it is in charge of generating the weights that the main network will employ. This main network will

output a PI from input features, whose coverage and width are dependent on the preference vector.

One of the main advantages of this method is its flexibility. Sampling random preference vectors and with a proper training process, it is able to obtain the full Pareto front with a single model. That is, for every nominal coverage, the Pareto front contains a model that returns PIs as narrow as possible. On the other side, its ease of use makes POPI-HN an interesting main option to solve probabilistic regression problems. The method does not employ additional parameters or configurations, and no more hyper-parameters than a classical feed-forward neural network are needed to be adjusted.

For comparative purposes, POPI-HN performance was evaluated in eight different benchmarking datasets by obtaining six PIs for six different coverage levels. The obtained PIs were confronted with the ones obtained by Quality Driven Deep Neural Networks. This QD-DNN is also a promising method for the direct estimation of PIs. Its structure allowed easy training by gradient descent and implementation of multiple PI outputs at once. However, QD-DNN shows some disadvantages against POPI-HN: as the method is single-objective, it is needed to decide in advance how many PIs to be obtained, and to adjust an extra parameter of the loss function λ (important to compute the weight of the penalty for PIs that do not satisfy the required coverage).

Results show similar and correct behavior of both methods regarding the coverage objective. However, when we focus on the width objective, POPI-HN outperforms QD-DNN for most *PINC* values in the majority of datasets, achieving a width improvement of up to 49% in some cases. This might be due to the width optimization in the multi-objective framework. Furthermore, time complexity for POPI-HN method is close to QD-DNN.

In summary, the POPI-HN implementation has resulted in a method that not only achieves the required coverage in our experiments but it is also able to improve the PI width obtained by QD-DNN. Furthermore, POPI-HN can obtain PIs for every possible nominal coverage, whereas QD-DNN requires the number of different PIs and their nominal coverages to be predefined. It also requires an additional trade-off λ parameter in order to aggregate the two objectives into a single-objective loss.

The current version of POPI-HN requires splitting the available data into a training set and a validation set. The latter is used to compute the validation front, which is required for mapping nominal coverage values into preference vectors. It would be interesting to carry out future research on alternatives that do not depend on a validation set for this purpose, that for some datasets might be too small. Also, in this work, POPI-HN has been applied to standard feed-forward neural networks, but other architectures appropriated for time series, such as Long Short-Term Memory Neural Networks (LSTM) or temporal transformers might benefit from it. Possible application fields could include, for instance, the energy market to obtain multiple probabilistic forecasts of wind energy, business problems to predict demands at different levels of confidence, or financial markets to model a stock price.

CRediT authorship contribution statement

Antonio Alcántara: Investigation, Methodology, Software, Data curation, Validation, Writing – original draft. **Inés M. Galván:** Conceptualization, Investigation, Methodology, Funding acquisition, Investigation, Supervision, Writing – review. **Ricardo Aler:** Conceptualization, Investigation, Methodology, Software, Funding acquisition, Investigation, Supervision, Writing – review.

Declaration of competing interest

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

Data availability

Github link in the paper

Acknowledgments

This publication is part of the I+D+i project PID2019-10745 5RB-C22, funded by MCIN /AEI/10.13039/501100011033. This work was also supported by the Comunidad de Madrid Excellence Program. Funding for APC: Universidad Carlos III de Madrid (Read & Publish Agreement CRUE-CSIC 2022)

References

[1] J. Nowotarski, R. Weron, Recent advances in electricity price forecasting: A review of probabilistic forecasting, Renew. Sustain. Energy Rev. 81 (2018) 1548–1568.
[2] J. Chen, D. Kipping, Probabilistic forecasting of the masses and radii of other worlds, Astrophys. J. 834 (1) (2016) 17.
[3] C. Wan, J. Lin, Y. Song, Z. Xu, G. Yang, Probabilistic forecasting of photovoltaic generation: An efficient statistical approach, IEEE Trans. Power Syst. 32 (3) (2016) 2471–2472.
[4] D. Kim, J. Hur, Short-term probabilistic forecasting of wind energy resources using the enhanced ensemble method, Energy 157 (2018) 211–226.
[5] N. Meinshausen, G. Ridgeway, Quantile regression forests, J. Mach. Learn. Res. 7 (6) (2006).
[6] M. Landry, T.P. Erlinger, D. Patschke, C. Varrichio, Probabilistic gradient boosting machines for GEFCom2014 wind forecasting, Int. J. Forecast. 32 (3) (2016) 1061–1066.
[7] T. Duan, A. Anand, D.Y. Ding, K.K. Thai, S. Basu, A. Ng, A. Schuler, Ngboost: Natural gradient boosting for probabilistic prediction, in: International Conference on Machine Learning, PMLR, 2020, pp. 2690–2700.
[8] L. Hao, D.Q. Naiman, D.Q. Naiman, Quantile Regression, Vol. 149, Sage, 2007.
[9] A. Khosravi, S. Nahavandi, D. Creighton, A.F. Atiya, Lower upper bound estimation method for construction of neural network-based prediction intervals, IEEE Trans. Neural Netw. 22 (3) (2010) 337–346.
[10] C. Wan, Z. Xu, P. Pinson, Direct interval forecasting of wind power, IEEE Trans. Power Syst. 28 (4) (2013) 4877–4878.
[11] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. 25 (2012) 1097–1105.
[12] S. Kudugunta, E. Ferrara, Deep neural networks for bot detection, Inform. Sci. 467 (2018) 312–322.
[13] J. Schmidt-Hieber, Nonparametric regression using deep neural networks with ReLU activation function, Ann. Statist. 48 (4) (2020) 1875–1897.
[14] W. Zhang, H. Quan, D. Srinivasan, An improved quantile regression neural network for probabilistic load forecasting, IEEE Trans. Smart Grid 10 (4) (2018) 4425–4434.
[15] K. Hatalis, A.J. Lamadrid, K. Scheinberg, S. Kishore, A novel smoothed loss and penalty function for noncrossing composite quantile estimation via deep neural networks, 2019, arXiv preprint arXiv:1909.12122.
[16] D.B. Or, M. Kolomenkin, G. Shabat, Generalized quantile loss for deep neural networks, 2020, arXiv preprint arXiv:2012.14348.
[17] S.J. Moon, J.-J. Jeon, J.S.H. Lee, Y. Kim, Learning multiple quantiles with neural networks, J. Comput. Graph. Statist. (2021) 1–11.
[18] A. Alcántara, I.M. Galván, R. Aler, Deep neural networks for the quantile estimation of regional renewable energy production, Appl. Intell. (2022) 1–36.
[19] H. Quan, D. Srinivasan, A. Khosravi, Particle swarm optimization for construction of neural network-based prediction intervals, Neurocomputing 127 (2014) 172–180.
[20] I.M. Galván, J.M. Valls, A. Cervantes, R. Aler, Multi-objective evolutionary optimization of prediction intervals for solar energy forecasting with neural networks, Inform. Sci. 418 (2017) 363–382.
[21] I.M. Galván, J. Huertas-Tato, F.J. Rodríguez-Benítez, C. Arbizu-Barrena, D. Pozo-Vázquez, R. Aler, Evolutionary-based prediction interval estimation by blending solar radiation forecasting models using meteorological weather types, Appl. Soft Comput. (2021) 107531.
[22] M. Zhou, B. Wang, S. Guo, J. Watada, Multi-objective prediction intervals for wind power forecast based on deep neural networks, Inform. Sci. 550 (2021) 207–220.
[23] H. Quan, D. Srinivasan, A. Khosravi, Short-term load and wind power forecasting using neural network-based prediction intervals, IEEE Trans. Neural Netw. Learn. Syst. 25 (2) (2013) 303–315.

[24] R. Li, Y. Jin, A wind speed interval prediction system based on multi-objective optimization for machine learning method, Appl. Energy 228 (2018) 2207–2220.
[25] R. Taormina, K.-W. Chau, ANN-based interval forecasting of streamflow discharges using the LUBE method and MOFIPS, Eng. Appl. Artif. Intell. 45 (2015) 429–440.
[26] A. Kavousi-Fard, W. Su, T. Jin, A machine-learning-based cyber attack detection model for wireless sensor networks in microgrids, IEEE Trans. Ind. Inform. 17 (1) (2020) 650–658.
[27] T. Pearce, A. Brintrup, M. Zaki, A. Neely, High-quality prediction intervals for deep learning: A distribution-free, ensembled approach, in: International Conference on Machine Learning, PMLR, 2018, pp. 4075–4084.
[28] J. Hu, Y. Lin, J. Tang, J. Zhao, A new wind power interval prediction approach based on reservoir computing and a quality-driven loss function, Appl. Soft Comput. 92 (2020) 106327.
[29] F. Liu, Q. Tao, D. Yang, D. Sidorov, Bidirectional gated recurrent unit-based lower upper bound estimation method for wind power interval prediction, IEEE Trans. Artif. Intell. (2021).
[30] Y. Lai, Y. Shi, Y. Han, Y. Shao, M. Qi, B. Li, Exploring uncertainty in regression neural networks for construction of prediction intervals, Neurocomputing 481 (2022) 249–257.
[31] T.S. Salem, H. Langseth, H. Ramampiaro, Prediction intervals: Split normal mixture from quality-driven deep ensembles, in: Conference on Uncertainty in Artificial Intelligence, PMLR, 2020, pp. 1179–1187.
[32] H. Zhong, L. Xu, An all-batch loss for constructing prediction intervals, Appl. Sci. 11 (4) (2021) 1728.
[33] J. Hu, W. Zhao, J. Tang, Q. Luo, Integrating a softened multi-interval loss function into neural networks for wind power prediction, Appl. Soft Comput. 113 (2021) 108009.
[34] D. Ha, A. Dai, Q.V. Le, Hypernetworks, 2016, arXiv preprint arXiv:1609.09106.
[35] A. Navon, A. Shamsian, G. Chechik, E. Fetaya, Learning the Pareto front with hypernetworks, in: International Conference on Learning Representations, 2021, URL https://openreview.net/forum?id=NjF772F4ZZR.
[36] A. Alcántara, I.M. Galván, R. Aler, Direct estimation of prediction intervals for solar and wind regional energy forecasting with deep neural networks, Eng. Appl. Artif. Intell. 114 (2022) 105128.
[37] Y. Yao, L. Rosasco, A. Caponnetto, On early stopping in gradient descent learning, Constr. Approx. 26 (2) (2007) 289–315.
[38] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms—a comparative case study, in: International Conference on Parallel Problem Solving from Nature, Springer, 1998, pp. 292–301.
[39] A. Alcántara, POPI-HN, 2022, URL https://github.com/antonioalcantaramata/POPI-HN/.
[40] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[41] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), 2015, arXiv preprint arXiv:1511.07289.
[42] A. Krogh, J. Hertz, A simple weight decay can improve generalization, Adv. Neural Inf. Process. Syst. 4 (1991).
[43] T.K. Kim, T test as a parametric statistic, Korean J. Anesthesiol. 68 (6) (2015) 540.
[44] N.M. Razali, Y.B. Wah, et al., Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests, J. Statist. Model. Anal. 2 (1) (2011) 21–33.
[45] P.E. McKnight, J. Najab, Mann-whitney u test, Corsini Encyclopedia Psychol. (2010) 1.
[46] D.W. Nordstokke, B.D. Zumbo, A new nonparametric levene test for equal variances, Psicológica 31 (2) (2010) 401–430.
[47] M. Emmerich, A. Deutz, Time complexity and zeros of the hypervolume indicator gradient field, in: EVOLVE-a Bridge Between Probability, Set Oriented Numerics, and Evolutionary Computation III, Springer, 2014, pp. 169–193.