Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Multi-objective optimization of Neural Network with Stochastic Directed Search

THESIS

in partial fulfillment of the requirements for the degree of

Master in Computer Science and Engineering

PRESENTS:
Samuel López Ruiz

Advisor:
PhD Katya Rodríguez Vázquez
Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas

Coadvisor:
PhD Carlos Ignacio Hernández Castellanos
Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas

Mexico City, May 2023

Abstract

In engineering, management, and finance it is common to have conflicting objectives that must be jointly optimized. Recently, a promissory approach for modeling these complex problems is through deep neural networks. However, due to their structure, they introduce high-dimensional problems which typically cannot be addressed with current methods. This work develops a gradient-based predictor-corrector method based on directed search to efficiently fine-tune a neural network without retraining the entire network. The multi-objective problem is formulated by dividing the neural network loss function into several objectives and selecting a subset of weights as the decision space. After the neural network is trained with stochastic gradient descent, the proposed method, i.e., *stochastic directed search*, efficiently finds sections of the Pareto front that are of interest to the decision maker. The contributions include the use of Jacobian computation in batches to account for GPU memory limitations, special routines for better approximating boundaries, and an early stopping criterion that limits the search space. The effectiveness of the algorithm is demonstrated by fine-tuning a Temporal Fusion Transformer model that generates multi-horizon quantile forecasts of the S&P 500 Futures financial time series. Nevertheless, our algorithm is applicable to any neural network whose training task can be divided into opposing objectives. Additionally, this work evaluates the hyperparameters' effect on the algorithm performance to provide guidance for the users who want to implement it. Finally, the *stochastic directed search* method is compared with NSGA-II and the results show that it performs competitively while offering several advantages, such as fewer function evaluations, higher hypervolume of Pareto fronts and the ability to solve large problem instances with sizes up to 240,000.

Acknowledgments

I would like to express my sincere gratitude to my thesis supervisor Katya Rodríguez, whose guidance, support, and expertise have been invaluable throughout the research and writing process. Additionally, I am thankful to my professor Carlos Hernández who provided me with insightful feedback, and encouraged me to pursue research in multi-objective optimization topics. I am deeply grateful for their unwavering dedication and commitment to my academic growth.

I would also like to thank my parents, my girlfriend and my aunt Monica López for their support and encouragement. Their love, patience, and encouragement have been an endless source of motivation for me, and I am truly blessed to have them in my life.

Finally, I want to express my heartfelt thanks to Vicente Albiter, who recommended this master's program to me. Without his suggestion, I may never have embarked on this journey, and for that, I am eternally grateful.

Thank you to everyone who has played a role in my academic journey, and I look forward to continuing to learn and grow in the years to come.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Decision-making based on multiple objective functions is ubiquitous in the industry. Conflicting objectives that must be optimized concurrently are common in engineering, finance, and management fields (see Tapia and Coello (2007)). Such decision-making scenarios can be modeled as a multi-objective optimization problem (MOP) to solve them with computational tools. Evaluating a MOP solution involves multiple criteria (objectives) that compete with each other and the trade-off among them invalidates the existence of a single optimal solution for all the objectives. This gives rise to the Pareto set, which consists of solutions with the optimal compromise between different objectives.

This work focuses on the particular set of problems derived from the optimization of neural networks considering multiple objectives. The advancement of neural networks has brought about numerous breakthroughs in various fields, including computer vision, natural language processing, and speech recognition. However, designing and training neural networks is still a challenging task, as it requires a considerable amount of computational resources and technical expertise.

The aim of this thesis is to develop a multi-objective optimization framework for neural networks that addresses these challenges. Specifically, this framework will provide researchers and decision makers with a methodology to fine-tune neural networks effectively and efficiently without the need of retraining the models. Additionally, the framework will provide new insights into the Pareto front of the loss function used to trained the neural network.

The rest of this section includes a brief overview of multi-objective optimization algorithms, the formulation of the neural network optimization problem, related work and the specific contributions of this work.

1.2 Multi-objective optimization

Multi-objective optimization (MOO) methods for finding the Pareto front or a subset of the latter, fall into the following four broad categories:

1. Scalarization methods use parametrization to transform the original multi-objective problem into a classical single-objective optimization problems (see Jahn (1986), Miettinen (1998) Eichfelder (2008), and Marler and Arora (2010)). The most common scalarization method is the

weighted sum, which lacks the capacity of finding non-convex Pareto fronts. This approach has been used repeatedly when formulating multi-task neural networks (see Liao et al. (2016), Wang et al. (2017) Cipolla et al. (2018)). Another popular scalarization method is the Normally Boundary Intersection (NBI) method by Das and Dennis (2006).

2. Population based stochastic methods include evolving strategies, bio-heuristics, and estimation of distribution algorithms, among others. The most prominent and widely used type of bio-inspired algorithms are referred to as multi-objective evolutionary algorithms (MOEAs) (see Deb et al. (2000), Deb (2001), Zitzler et al. (2001), and Khare et al. (2003)). Other common bio-inspired methods include multi-objective particle swarm optimization (MOPSO) and multi-objective ant colony optimization (MOACO) (see Coello Coello and Lechuga (2002), Fieldsend et al. (2002) and Mora et al. (2006), Arteta et al. (2007), respectively). These methods are easy to implement, can find discontinuous Pareto fronts and no gradient information is needed. However, their main drawbacks are their relatively slow convergence and the fact that optimality cannot be guaranteed.

3. Descent direction methods focus on finding only one optimal point and in some cases, the information on the gradient and hessian can be exploited to improve the convergence rate (see Fliege and Svaiter (2000), Schäffler et al. (2002), and Fliege et al. (2009)). These methods are the multi-objective versions of directed search algorithms.

4. Finally, the last category consists of continuation methods proposed by Schütze et al. (2005), Lara et al. (2010), Wang (2012), Martin et al. (2013), and Wang et al. (2019). These methods exploit the fact that given an optimal point and certain conditions, the neighboring Pareto front can be found.

This work focuses on recovering Pareto sets in deep learning problems that can be expressed as MOPs. Specifically, a time series quantile forecasting problem is formulated as bi-objective problem (BOP) and solved using the directed search method, which belongs to the continuation methods family. Improvements are proposed to the directed search method to enhance its performance when solving deep learning problems MOPs. The integration of random data batches in the computation of the Jacobian, give the name to the proposed algorithm: stochastic directed search, which for the rest of this work will be abbreviated as SDS.

1.3 Financial forecasting using deep learning

Identifying and forecasting significant price movements using past observations has been a topic of interest for obvious reasons. The purpose of forecasting models is to extract signals from past observations that will continue in the future. Forecasting in computational finance has been starred by neural networks and evolutionary computation, as expressed by Tsang and Martinez-Jaramillo (2004). However, the use of artificial neural networks for forecasting tasks surged in the '90s since they can learn the non-linearities in time series (see Zhang et al. (1998) and Chakraborty et al. (1992)). In this work, the neural network architecture based on transformers proposed by Lim et al. (2021) is used to generate high-performance quantile predictions.

Quantile prediction forecast models generate lower and upper bound (quantiles) between which the target is expected to lie. The quality assessment of predictions is measured with *quantile coverage* *risk* (QCR), and *quantile estimation risk* (QER). QCR quantifies if the forecast is outside the quantile range, whereas QER measures the meaningfulness of the quantile forecast. Both measures are conflicting in nature, since QCR increases if the forecast is outside the quantile range, and on the contrary, QER increases if the quantile range is too broad.

The conflicting nature of QCR and QER makes it straightforward to formulate the quantile forecast model as a multi-objective problem that can be solved using any of the MOO methods previously mentioned. Establishing a multi-objective framework allows the decision-maker to fine-tune the model without having to retrain the neural network. To the best of our knowledge, the fine-tuning of a quantile forecast model has not been solved using a continuation method. The overview of the proposed methodology is the following:

1. A Temporal Fusion Transformer (TFT) model proposed by Lim et al. (2021) was trained with stochastic gradient descent (SGD) and using a quantile loss function (QL). The dataset for the forecasting task consists of the S&P Futures (ES) and Nasdaq Futures (NQ) with a sampling frequency of 1 hour from January 2015 to June 2021.

2. The optimization task is formulated as a dual bi-objective problem, where the weights of a subset of the TFT model layers are optimized using the novel indicators QCR and QER.

3. Using the weights found in the previous step, the multi-objective optimization is done with the proposed stochastic directed search method.

1.4 Related work

The precursor of quantile forecasts are prediction intervals (PIs), which also define upper and lower bounds but cannot weight between the coverage and meaningfulness of the intervals. On the contraty, quantiles can give more importance to coverage, e.g. (0.1, 0.9) or meaningfulness, e.g. (0.3, 0.7). Prediction intervals are the special case where the quantiles are (0, 1) and the criteria used to measure its quality are the prediction interval coverage probability (PICP) and prediction interval width (PIW), with their normalized versions being PIEE and PINAW.

Several authors have addressed the conflicting nature of PICP /PIEE with PIW/PINAW, by proposing single and multi-objective optimization machine learning problems. Shrivastava et al. (2016) proposed support vector machines (SVM) to determine PIs and formulated the machine learning process as a single-objective problem using scalarization where PICP and PIW are minimized concurrently and the decision space is conformed by SVM cost function and RBF kernel function. Ak et al. (2015) used NN to forecast prediction intervals (PIs) and proposed the well-known MOEA, NSGA-II to find the Pareto front. The optimization objectives used were the PICP and PINAW, and the weights of a three layer NN dense architecture constitute the decision variables. A radial basis function (RBF) neural network was proposed by Zhang et al. (2016) to compute PIs. The centers of the RBF are determined via K-means clustering and the hidden-output weights of the RBF are adjusted using the NSGA-II algorithm to minimize PINAW and maximize PICP. Zhou et al. (2021) proposes a LSTM network to construct PIs in which the internal parameters are optimized with the NSGA-II, INSGA-II, and MOPSO algorithms, and the PIEE and PINAW indicators are used as optimization objectives. López et al. (2022) propose the fine-tuning of a quantile forecasting neural network using NSGA-II and NSGA-III with the objectives QCR and QER. In contrast to global methods as MOEAs, this work implements a local search method to find the Pareto front.

Other multi-objective frameworks have been used to train multi-task learning (MTL) neural networks, in which the objectives are related to the tasks. Sener and Koltun (2018) used the multiple gradient descent algorithms proposed by Désidéri (2012) to optimization a MTL multi-objective problem. Although applied to MTL problems, the methodology by Ma et al. (2020) is the most related to this work. Ma et al. implemented a variant of the directed-search continuation methods proposed by Schütze et al. (2005) and Lara et al. (2010) to train a MTL neural network. Compared with Ma et al. (2020)'s proposal, this work does not require the Hessian-vector products, and the whole Pareto front (PF) is obtained from a single solution belonging (or near) the PF.

1.5 Contribution

In this work, the loss function of the neural network is divided into multiple objectives and solved with the proposed stochastic directed search (SDS) method that incorporates the following contributions:

- The computation of the Jacobian using batches of data.
- A stop criteria to obtain a subset of the Pareto front based on the increment of the sum of the objectives.
- A bisection method for the predictor to get closer to the boundaries in objective space.

These modifications are designed to improve the solutions of MOPs derived from the optimization of neural networks and the results show that the stochastic directed search method can efficiently solve high dimensional problems.

The remainder of this work is organized as follows. Chapter 2 presents a theoretical background overview of multi-objective optimization and describes the predictor-corrector method. Chapter 5 proposes the stochastic directed search continuation method. Chapter 3 formulates the multi-objective optimization problem. Section 6 analyses the impact of hyperparameters on the stochastic directed search performance. Chapter 7 presents the results and discusses our most relevant findings compared with the traditional NSGA-II method. Chapter, Section 8 concludes.

Chapter 2

Background

This section introduces the definition of multi-objective problems as well as the predictor-corrector directed search method.

2.1 Multi-objective problems

A general continuous MOP is defined as:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} F(x) \\ &\text{s.t. } \ g_i(x) \leq 0, \ \ j = 1, \dots, m \\ &\qquad h_j(x) = 0, \ \ j = 1, \dots, p\,, \end{aligned} \tag{2.1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^k$ is the vector of objective functions $f_1, \dots, f_k$, $g : \mathbb{R}^n \to \mathbb{R}^m$ is the vector of inequality constraints and $h : \mathbb{R}^n \to \mathbb{R}^p$ is the vector of equality constraints. The feasible region in decision space is defined as $\mathcal{X} = \{x \in \mathbb{R}^n \mid g(x) \leq 0 \text{ and } h(x) = 0\}$.

Since generally, no solution can minimize all objectives simultaneously, the optimality concept for MOPs is given by the notion of *Pareto dominance.*

Definition 1 *For two given points $x \in \mathcal{X}$ and $y \in \mathcal{X}$, point $x \in \mathbb{R}^n$ dominates $y \in \mathbb{R}^n$ $(x \prec y)$ with respect to F only if $f_i(x) \leq f_i(y)\, \forall\, i = 1, \dots, k$ and $\exists\, i$ s.t. $f_i(x) \neq f_i(y)$.*

Definition 2 *A feasible point $x^* \in \mathcal{X}$ is Pareto efficient if $\nexists\, y \in \mathcal{X}$ s.t. $(y \prec x)$.*

Definition 3 *A feasible point $x^* \in \mathcal{X}$ is weak Pareto efficient if $\nexists\, y \in \mathcal{X}$ s.t. $f_i(y) < f_i(x)\, \forall\, i = 1, \dots, k$.*

The MOO methods are designed to find the Pareto front (or Pareto set), which is defined as:

Definition 4 *The Pareto set of optimal points is defined by $\mathcal{PS} = \{x \in \mathcal{X} \mid \ \nexists\, y \in \mathcal{X} \ : \ y \prec x\}$.*

Definition 5 *The image $F(\mathcal{PS})$ of the Pareto set is called Pareto front $\mathcal{PF}$.*

Local optimal solutions satisfy the Karush-Kuhn-Tucker (Kuhn, Harold, Tucker (1951)) conditions, which are applicable when F is continuous and twice differentiable and under certain conditions, they form a $(k-1)$ dimensional manifold (see Hillermeier (2001)).

With the gradient $\nabla f(x) \in \mathbb{R}^n$ of a multivariable function f defined as:

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{pmatrix} , \tag{2.2}$$

slack variables $\alpha_i \geq 0$, $\mu_i \geq 0$ and $\beta_i \geq 0$ are introduced to define the necessary conditions for Pareto optimality for m equality constraints, p inequality constraints, and k objectives such that:

$$\begin{aligned} \sum_{i=1}^{k} \alpha_i \nabla f_i(x^*) + \sum_{i=1}^{m} \mu_i \nabla h_i(x^*) + \sum_{i=1}^{p} \beta_i \nabla g_i(x^*) = 0 \\ \sum_{i=1}^{k} \alpha_i = 1 \\ h_i(x^*) = 0 \ \forall \ 1, \ldots, m \\ g_i(x^*) \leq 0 \ \forall \ 1, \ldots, p \\ \beta_i g_i(x^*) = 0 \ \forall \ 1, \ldots, p \, . \end{aligned} \tag{2.3}$$

For unconstrained MOPs ($m = p = 0$) and Eqs. 2.3 are reduced to $\sum_{i=1}^{k} \alpha_i \nabla f_i(x^*) = 0$, where the vector α is normal to the tangent surface of the Pareto front at $F(x^*)$.

The critical (non-dominated) points can be defined using the definition of a *descent direction* v.

Definition 6 *A descent direction for function $f : \mathbb{R}^n \to \mathbb{R}$ at a point $x \in \mathbb{R}^n$ is defined by vector v, such that:*

$$\nabla f(x)^T v < 0 \, .$$

Critical points for unconstrained MOPs are thus defined as $x^* \in \mathbb{R}^n$ such that for all possible vectors $v \in \mathbb{R}^n$ there is at least one $i \in \{1, \ldots, k\}$ that satisfies:

$$\nabla f_i(x^*)^T v \geq 0 \, . \tag{2.4}$$

Consequently, to improve one objective, at least another objective must worsen. Similarly, a descent direction v exists for noncritical points.

Additionally, a critical point x^* can be determined using the Jacobian of $F(x^*)$. Specifically, x^* is a critical point of F if

$$\operatorname{rank}(J(x^*)) < k \, , \tag{2.5}$$

where

$$J(x^*) = \begin{pmatrix} \frac{\partial f_1(x^*)}{\partial x_1} & \cdots & \frac{\partial f_1(x^*)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_k(x^*)}{\partial x_1} & \cdots & \frac{\partial f_k(x^*)}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{k \times n} \, . \tag{2.6}$$

Inversely, noncritical points of F exhibit Jacobians with full rank.

2.2 Predictor-directed search corrector method

Continuation methods exploit the optimality conditions to explore the $(k-1)$ dimensional manifold starting from one critical point. Continuation methods are part of the family of local search methods and are susceptible to getting trapped in local optima or missing sections of the Pareto front if the front is not continuous. On the downside, most continuation methods require gradients with few exceptions such as the works of Lara et al. (2010) and Lara et al. (2013). This requirement reduces the universe of applicable problems, but fortunately, computing gradients in neural networks is part of their training process and does not implicate additional effort.

The most common continuation methods are the Zig-zag method proposed by Wang (2012) and the Predictor-Corrector (PC) methods explored by Hillermeier (2001). This work is based on the directed search method, which is a PC algorithm where the corrector is guided by a direction in the objective space. It was originally proposed by Schütze et al. (2005) and further expanded by Schütze et al. (2016), Martín and Schütze (2018), and Wang et al. (2019). The main principle behind the algorithm is to compute a predictor from a non-dominated solution x with a direction as close as possible to the tangent of the continuous front in that point $F(x)$. If the Pareto set is linear, the predictor $F(x+tv)$ will also lie in the front; however, more often than not, correctors are needed because of the nonlinearity of the Pareto set. The correctors seek to find the front starting from the predictor solution and usually several several steps are needed until the criterion for a critical point is met. Fig. 2-1 shows an example of a sequence of predictors and correctors trying to approximate the Pareto front.

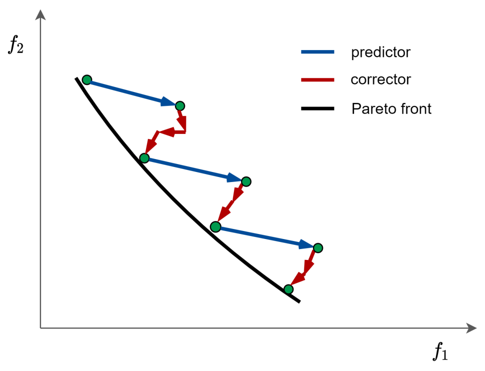


Figure 2-1: Predictor and corrector steps in continuation method.

Hillermeier (2001) proposed a PC method that considered the KKT conditions (Eq. 2.3) and introduced several concepts adopted in recent PC versions. The main drawback of his classical approach was the requirement of computing Hessians. Nonetheless, the PC method presented in this work is an adaptation from the algorithm described by Schütze et al. (2005), which is Hessian-free and applicable to unconstrained MOPs. With the intention of keeping this work replicable and self-contained, this section will explain the complete method and its details.

2.2.1 Predictor

The predictor explores promising directions from an existing Pareto solution. A typical choice is to move in a tangent direction to the curve of interest at point x. Let $x \in \mathbb{R}^n$ be a KKT point and $\alpha \in \mathbb{R}^k$ its associated Lagrange multiplier, such that:

$$\sum_{i=1}^{k} \alpha_i \nabla f_i(x) = J(x)^T \alpha = 0\,.$$

Motivated by the KKT conditions (Eq. 2.3), the following map is obtained:

$$\begin{pmatrix} J(x)^T \alpha \\ (\sum_{i=1}^{k} \alpha_i) - 1 \end{pmatrix} = 0 \,. \tag{2.7}$$

This implies that α is orthogonal to the linearized Pareto front at $F(x)$ and directions d_+, d_- orthogonal to α are promising directions to obtain new predictors p_+, p_- as shown in Fig. 2-2 for $k = 2$.

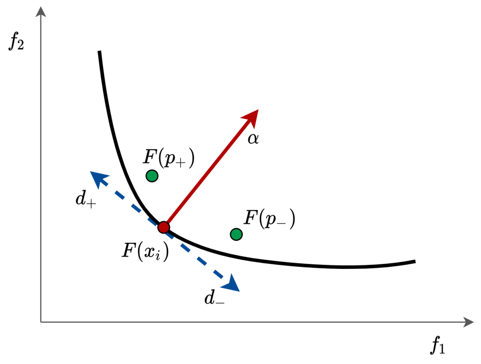


Figure 2-2: Visualization of normal vector α, tangent directions d and predictors $F(p)$.

Schäffler et al. (2002) suggested calculating vector α by solving the following convex problem:

$$\begin{aligned} &\min_{\alpha} \;\; ||J(x)^T \alpha||_2^2 \\ &\text{s.t.} \;\; \alpha_i \geq 0 \forall \;\; 1, \ldots, k, \;\; \sum_{i}^{k} \alpha_i = 1 \,. \end{aligned} \tag{2.8}$$

Since typically the number of objectives k is small compared to the number of variables n, solving the single objective optimization problem takes little time.

Lara et al. (2013) presented the directed search method (DS) for continuous MOPs, which incorporates mapping directions in the image space of F to the domain space. The same theoretical considerations are relevant and applicable in the predictor and corrector steps.

Let $x_0 \in \mathbb{R}^n$ be a point in the solution space with $\text{rank}(J(x_0)) = k$ and vector $d \in \mathbb{R}^k$ the desired search direction in the image space. The task is to find a search direction $v \in \mathbb{R}^n$ in the domain space, such that for $y_0 = x_0 + tv$, it holds that:

$$\lim_{t \to 0} \frac{f_i(y_0) - f_i(x_0)}{t} = \langle \nabla f_i(x_0), v \rangle = d_i, \;\forall \;\; i = 1, \ldots, k \,, \tag{2.9}$$

which can be written in matrix and vector form with the help of the Jacobian of F, as follows:

$$J(x_0)v = d \,. \tag{2.10}$$

The search direction v can be computed by solving the system of linear equations 2.10. Typically for MOPs, the system is under-determined since $n > k$, which means there are multiple solutions for v. One particular solution is:

$$v_+ = J(x_0)^+ d \,, \tag{2.11}$$

where $J^+ \in \mathbb{R}^{n \times k}$ is the pseudo inverse of J defined as $J^+ = J^T (JJ^T)^{-1}$.

Eq. 2.11 presents a method for computing direction v in domain space from a desired direction d in image space for point x_0. The next task is to obtain the directions d from the vector α, which is normal to the linearized Pareto front by doing a QR-factorization of α.

$$\alpha = QR\,, \tag{2.12}$$

where $R = (r_1, 0, \ldots, 0)^T$ with $r_1 \neq 0$ and $Q = (q_1, \ldots, q_k) \in \mathbb{R}^{k \times k}$ is an orthogonal matrix. By Eq. 2.12, $\alpha = r_1 q_1$ and the set of directions d that form orthonormal bases of the tangent space to the Pareto front at $F(x)$ are chosen to be:

$$d_i = q_{i+1} \;\; \forall \;\; i = 1, \ldots, k-1\,, \tag{2.13}$$

where q_i is the ith column vector of Q and live in the image of $F(x)$. From Eqs. 2.11 and 2.13, the predictor directions can be computed as:

$$v_i = J^{+}(x) q_{i+1} \;\; \forall \;\; i = 1, \ldots, k-1\,. \tag{2.14}$$

The predictors' directions are tangent to the Pareto front $\mathcal{PF}$ but not necessarily tangent to the Pareto set $\mathcal{PS}$. The predictors $p \in \mathbb{R}^n$ are expected to lie near the Pareto front to reduce the number or correctors needed to get back to the front. The set of predictors is computed as follows:

$$p_i = x + \lambda t \frac{v_i}{||v_i||}\,, \tag{2.15}$$

where t is the step size, and $\lambda \in \{1, -1\}$ is the direction. Both directions are required to guarantee the Pareto front is explored from all directions, and the selection of the step size t impacts the distance between solutions found and therefore the density of the Pareto front.

To obtain an evenly spaced solution set in the image of F, the following condition is imposed for consecutive solutions x_{i+1} and x_i:

$$||F(x_{i+1}) - F(x_i)|| \sim \tau\,,$$

where $\tau > 0$ is specified by the user. To guarantee equally spaced solutions, Hillermeier (2001) suggested computing the step size as:

$$t = \frac{\tau}{||J(x)v||}\,. \tag{2.16}$$

Alg. 1 shows the procedure to compute the predictors given a point x in the Pareto front. The first step is to decompose the orthogonal vector to the linearized front α into Q and R. For each column vector q of Q, search directions v, step size t, and the predictors p are calculated. For $k > 2$, both directions $\lambda \in \{1, -1\}$ for Eq. 2.15 should be explored; however, for BOPs, using both directions means that one of them leads exactly to where the previous solution was. To avoid unnecessary computations, for BOPs a single direction is used when computing the predictor in line 7 of Alg. 1, where the direction is given by the parameter λ. For $k > 2$, many d directions exist and a complementary orientation method is needed to efficiently explore the Pareto front. Schütze et al. (2005) proposed box-constrained regions to explore similar regions only once and at the same time, obtain equally distanced solutions.

The method GoodPredictor(x) in line 9 discards a predictor x if the following condition is not met:

$$0 \leq f_i(x) \leq b_i \;\; \forall \;\; i = 1, \ldots, k \,, \tag{2.17}$$

where b_i is the upper bound in the objective space specified by the user. This condition guarantees that the Pareto front will not exceed the given bounds in the objective space. Note that if the condition is not fulfilled, the output set P will remain empty.

Algorithm 1 Predictor Routine.

Input: Point $x \in \mathbb{R}^n$ in $\mathcal{PF}$, Jacobian $J(x)$, constant τ and normal vector α
Output: Set of predictors

```
1:  Routine PREDICTOR($x, \alpha, J(x), \tau, \lambda$)
2:      $q_i \leftarrow$ QRDECOMPOSITION($\alpha$)                    ▷ Eq. 2.12
3:      $P \leftarrow \emptyset$
4:      for $i \in \{1, \ldots, k-1\}$ do
5:          $v_i \leftarrow J^+(x) q_{i+1}$                          ▷ Eq. 2.14
6:          $t \leftarrow \tau / ||J(x) v_i||$                       ▷ Eq. 2.16
7:          $x_p \leftarrow x + \lambda t v_i / ||v_i||$             ▷ Eq. 2.15
8:          if GOODPREDICTOR($F(x_p)$) then
9:              $P \leftarrow P \cup \{x + t v_i / ||v_i||\}$
10:         end if
11:     end for
12:     return $P, t$
13: end Routine
```

2.2.2 Directed search corrector

Given the non-linearity of the Pareto set, the predictor $F(p)$ will not lie in the Pareto front and corrector steps are needed to get back to the front. In this work, the directed search (DS) method presented by Lara et al. (2013) is implemented as the corrector step. The same theoretical principles from the last section are applicable for finding a direction $v_c \in \mathbb{R}^n$ in domain space given a direction d in image space of F. Fig. 2-3 shows point $F(x_i)$ before the predictor step was applied. Under the assumption that $F(p)$ is close to $F(x)$, a promising choice for the direction is $-\alpha$, where α is the orthonormal vector to the linearized Pareto front in $F(x)$.

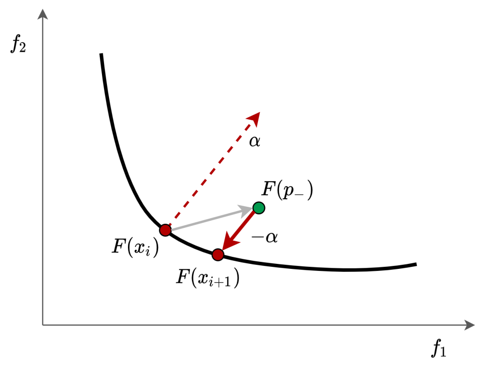


Figure 2-3: Visualization of normal vector α at point x, predictor $F(p)$, and corrector $F(x_{i+1})$.

The corrector can be computed similarly to the predictor, as follows:

$$c_i = p_i + t \frac{v_c}{||v_c||} \,, \tag{2.18}$$

where t is the step size and v_c is the direction in domain space calculated using Eq. 2.11 but with direction $-\alpha$ in image space of F. Unlike the predictor step, several correctors might be needed to get to the Pareto front, therefore to halt the cycle a stopping criterion determines if the corrector lies in or *close enough* to the Pareto front. Two important tasks arise: how to calculate the step size t for the corrector in Eq. 2.18 and which stopping criterion to use. These tasks are key to the performance of the continuation method, affecting the number of correctors needed in each step, which in turn affects the number of function and gradient evaluations needed to approximate the complete Pareto front.

2.2.3 Corrector step size

As stated in Eq. 2.9, a search direction $d \in \mathbb{R}^k$ in the image of F can be mapped to a direction $v \in \mathbb{R}^n$ in domain space with the step $x_{i+1} = x_i + tv$ only if t is sufficiently small. To determine the step size for the corrector of Eq. 2.18, we restrict the maximum angle allowed between the desired direction d in objective space and the corrector direction $\hat{d}$ obtained from the difference $\hat{d} = F(x + tv) - F(x)$. Fig. 2-4 shows a point in objective $F(x)$ the corrector point $F(x+tv)$, the desired direction d, the obtained direction $\hat{d}$ and the angle β between d and $\hat{d}$.

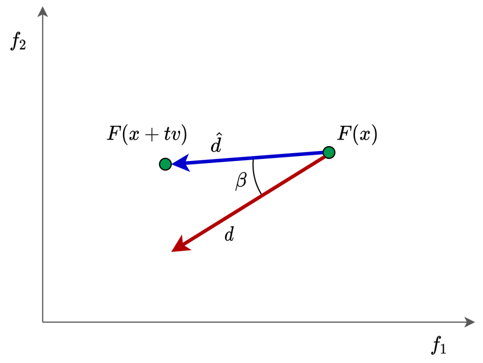


Figure 2-4: Visualization of the angle between the corrector and the desired direction d.

The angle is easily obtained with the formula:

$$\beta = \cos^{-1}\left(\frac{\langle \hat{d}, d\rangle}{||\hat{d}||||d||}\right). \tag{2.19}$$

Larger step sizes t reduce the number of correctors needed, as long as the angle β is within a *reasonable* threshold. A feasible step size given a point x and desired direction d in objective space is defined as:

$$\begin{aligned} \max_{t\in\mathbb{R}} \ & ||\hat{d}|| \\ \text{s.t.} \ & \hat{d} = F(x+tv) - F(x) \\ & \cos^{-1}\left(\frac{\langle \hat{d}, d\rangle}{||\hat{d}||||d||}\right) < \beta_{\max}. \end{aligned} \tag{2.20}$$

Where $\beta_{\max}$ should be defined carefully by the user. Wang et al. (2019) additionally imposes the restriction that $\min(\hat{d}) \geq 0$, which enforces $\hat{d}$ to point to the third quadrant. However, in this work, a more relaxed criterion is used since our gradients are calculated with a subset (batch) of the data and might not always be precise. Therefore, the only condition is that the direction is $B_{\max}$-close to the desired direction in objective space.

Given a point $x \in \mathbb{R}^n$ and an initial step size t_0, Alg. 2 is a simplified version of the bisection search algorithm used to calculate the time step t that satisfies Eqs. 2.20. In practice, a maximum number of iterations (N_a) is also imposed.

Algorithm 2 Next corrector routine.

Input: Point $x \in \mathbb{R}^n$, initial step size t_0, domain direction $v \in \mathbb{R}^n$, search direction $d \in \mathbb{R}^k$, threshold $\beta_{\max}$, and maximum iterations N_a.
Output: Next corrector in domain and objective space: x and $F(x)$.

```
1: Routine NextCorr($x, v, d, t_0, \beta_{\max}, N_a$)
2:     $t \leftarrow t_0$
3:     $n \leftarrow 0$
4:     $\hat{d} \leftarrow F(x + tv) - F(x)$
5:     $\beta \leftarrow \cos^{-1}(\langle \hat{d}, d \rangle)/(||\hat{d}||||d||)$        ▷ Eq. 2.19
6:     while $\beta > \beta_{\max}$ or $n < N_a$ do
7:         $t \leftarrow t/2$
8:         $\hat{d} \leftarrow F(x + tv) - F(x)$
9:         $\beta \leftarrow \cos^{-1}(\langle \hat{d}, d \rangle)/(||\hat{d}||||d||)$
10:        $n \leftarrow n + 1$
11:    end while
12:    return $\hat{d}, F(\hat{d})$
13: end Routine
```

2.2.4 Stopping criteria

By putting the Alg. 2 in a loop, we obtain a sequence of correctors closer to the Pareto front at each iteration and a *stopping criterion* is needed to end the loop when the last corrector is a critical point. The stopping criterion is crucial for the overall method since it determines the compromise between an efficient Pareto front and the computational cost of the corrector step. In this work, three criteria for critical points are explored.

Delta

Pérez (2008) and Wang et al. (2019) proposed the following single-objective optimization problem to compute the search direction v and a scalar variable delta δ that serves as a criterion to determine if x is a critical point:

$$\begin{aligned} \min_{v,\delta} \ & \tfrac{1}{2}||v||_2^2 - \delta \\ \text{s.t.} \ & J(x)v = \delta d \,. \end{aligned}$$

Two cases follow:

1. $v \neq 0 \iff \delta > 0 \iff \exists \ v \in \mathbb{R}^n : v = \frac{1}{\delta} J^{+}(x) d.$
2. $v = 0 \iff \delta = 0$, in which case x is a critical point (see Eq. 2.4).

In practice, a small threshold ϵ is defined to determine if $x \in \mathbb{R}^n$ is a critical point, such that:

$$\delta < \epsilon \iff x \text{ is a critical point.} \tag{2.21}$$

Matrix Rank

As shown in Eq. 2.5, x is a critical point if the *rank* of $J(x)$ is less than k. In practice, the matrix rank is computed with Single Value Decomposition (SVD), and a small threshold ϵ is specified below which SVD values are considered zero, such that:

$$\operatorname{rank}(J(x), \epsilon) < k \iff x \text{ is a critical point.} \tag{2.22}$$

Projection

The final criteria implemented is the norm of the product between the Jacobian and the direction d in the objective space. Intuitively, the projection of d in the linearized Pareto front is being computed and a small projection means that the direction d is parallel to the Jacobian at point x. In practice, a threshold ϵ is specified by the user to determine if x is a critical point, as follows:

$$||J(x)d||_2^2 < \epsilon \iff x \text{ is a critical point.} \tag{2.23}$$

Alg. 3 finds through a series of corrector steps a critical point x^*, when given a point x in domain space.

Algorithm 3 Corrector step.

Input: Starting point $x^* \in \mathbb{R}^n$, search direction d, initial step size t_0, and the maximum number of iterations N_b
Output: Final critical point $x^* \in \mathbb{R}^n$

```
1: Routine Corr($x, d, t_0, N_b, N_a$)
2:     $n \leftarrow 0$
3:     while not IsCritPoint($x, J(x), d$) or $n < N_b$ do
4:         $v \leftarrow J^+(x)d$
5:         $v \leftarrow v/||v||$
6:         $\{x, F(x)\} \leftarrow$ NextCorr($x, v, d, t_0, \beta_{\max}, N_a$)
7:         $n \leftarrow n + 1$
8:     end while
9:     return $x, F(x)$
10: end Routine
```

The subroutine IsCritPoint($x, J(x), d$) in line 3 contains one stopping criteria specified by the user from Eqs. 2.21, 2.22, or 2.23.

2.2.5 Continuation method

The complete method consists of a loop of predictor-corrector steps until the termination criterion is met. Schütze et al. (2005) presented the notion of 'boxes' as a data structure to save the solution set for $k > 2$. However, the MOP in this work is bi-objective and a simplified version adapted for two objectives is used.

The first task is to obtain a critical point x^* given a noncritical point $x \in \mathbb{R}^n$ and the second task is to obtain the Pareto set and Pareto front using that initial critical point x^*. Since $k = 2$, the front exploration can easily be divided into two routines, one for positive predictors and another for negative predictors (see Eq. 2.15).

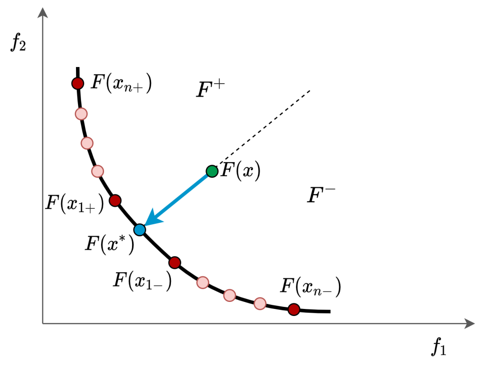


Figure 2-5: A) Get to the Pareto front, B) Explore the Pareto front with two subroutines ($\mathcal{PF}^+$ and $\mathcal{PF}^-$).

Fig. 2-5 shows the initial point x, the critical point x^* found given the initial point, and the two subsets $\mathcal{PF}^+ = \{F(x_{1+}), \ldots, F(x_{n+})\}$ and $\mathcal{PF}^- = \{F(x_{1-}), \ldots, F(x_{n-})\}$, with $n^+ \neq n^-$. The subset $\mathcal{PF}^+$ is obtained using the positive predictors p_{i+} from Eq. 2.15, whereas subset $\mathcal{PF}^-$ comes from the negative predictors p_{i-}. It is clear that the complete front is:

$$\mathcal{PF} = F(x^*) \cup \mathcal{PF}^+ \cup \mathcal{PF}^- \, .$$

Alg. 4 shows the routine involving the predictor and corrector steps, given a starting critical point x^* and a sign to use for the predictor. Although not completely necessary for $k = 2$, a queue method proposed by Schütze et al. (2005) and Wang et al. (2019) is implemented to keep a record of the critical solutions found.

Alg. 5 shows the complete continuation method for BOPs that consists of an initial descent to the Pareto front and the routines that obtain the Pareto fronts $\mathcal{PF}^+$ and $\mathcal{PF}^+$ by changing the sign λ of the predictors.

The subroutine Corrector in line 6 returns x^*, which is a critical point according to the stop criteria used and the initial point for the subroutine ExploreFront. Finally, the sets $\mathcal{X}$ and $\mathcal{F}$ contain the Pareto set and Pareto front, respectively.

Algorithm 4 Explore Pareto front.

Input: Starting critical point $x^* \in \mathbb{R}^n$, sign λ of predictor, desired space τ, number of iterations N_b, N_a

Output: Pareto set $\mathcal{X}$ and Pareto front $\mathcal{F}$

```
1: Routine ExploreFront($x^*, \lambda, N_b, N_a, \tau$)
2:     $M \leftarrow \emptyset$
3:     Enqueue($\mathcal{M}, \{x, F(x), J(x)\}$)
4:     while $\mathcal{M}$ is not empty do
5:         $\{x, F(x), J(x)\} \leftarrow$ Dequeue($\mathcal{M}$)
6:         $\alpha \leftarrow$ ConvexProblem()                         ▷ Eq. 2.8
7:         $\{X_p, t_0\} \leftarrow$ Predictor($x, \alpha, J(x), \tau, \lambda$)
8:         for $x \in X_p$ do
9:             $\{x, F(x)\} \leftarrow$ Corr($x, -\alpha, t_0, N_b, N_a$)
10:            Enqueu($\mathcal{M}, \{x, F(x), J(x)\}$)
11:            $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$
12:            $\mathcal{F} \leftarrow \mathcal{F} \cup \{F(x)\}$
13:        end for
14:    end while
15:    return $\mathcal{X}, \mathcal{F}$
16: end Routine
```

Algorithm 5 Continuation complete method.

Input: Starting point $x \in \mathbb{R}^n$

Output: Pareto set $\mathcal{X}$ and Pareto front $\mathcal{F}$

```
1: Routine Continuation($x$)
2:     Compute $\alpha$ from Problem 2.8
3:     $v \leftarrow J^+(x)a$
4:     $v \leftarrow v/||v||$
5:     $t_0 \leftarrow \tau/||J(x)v||$                              ▷ Eq. 2.16
6:     $\{x^*, F(x)\} \leftarrow$ Corrector($x, -\alpha, t_0$)
7:     $\mathcal{X} \leftarrow \{x^*\}$
8:     $\mathcal{F} \leftarrow \{F(x^*)\}$
9:     for $\lambda \in \{-1, 1\}$ do
10:        $\{\mathcal{X}_i, \mathcal{F}_i\} \leftarrow$ ExploreFront($x^*, \lambda$)
11:        $\mathcal{X} \leftarrow \mathcal{X} \cup \mathcal{X}_i$
12:        $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_i$
13:    end for
14:    return $\mathcal{X}, \mathcal{F}$
15: end Routine
```

2.2.6 MOP Example solved by directed search

This section shows the solution obtained when solving a classic MOP. Consider the bi-objective ($k = 2$) problem with $x \in \mathbb{R}^2$, i.e., $n = 2$:

$$
\begin{aligned}
\min_{x \in \mathbb{R}^2} \; & F(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2)) \\
\text{s.t.} \; & f_1(x_1, x_2) = (x_1 - 1)^4 + (x_2 - 1)^2 \\
& f_2(x_1, x_2) = (x_1 + 1)^4 + (x_2 + 1)^2 \; .
\end{aligned}
$$

Note that the point $x = (0, 0)$ selected as the initial point does not lie in the Pareto set and the upper bounds are $b_1 = b_2 = 7$.

Table 2.1: Continuation method configuration for example.

Parameter	Value	Ref.
Predictor		
Bounds (b_1, b_2)	[7, 7]	Eq. 2.17
Step size (τ)	0.9	Eq. 2.16
Corrector		
Max angle $(\beta_{\max})$	45°	2.20
Crit. point criteria	Delta	Eq. 2.21
Crit. point threshold (ϵ)	1×10^{-4}	Eq. 2.21
Step size max. iter. (N_a)	20	Routine 2
Correctors max. iter. (N_b)	20	Routine 3
General		
Initial point (x)	(0, 0)	Routine 5

Using the hyperparameters from Table 2.1, Figures 2-7 and 2-6 show the Pareto set and Pareto front obtained, respectively. The theoretical efficient boundaries are shown in blue and are labeled as 'Pareto set' or 'Pareto front'. The initial descent to the front (line 6 of Alg. 5) is shown in light blue and is labeled as 'descent'. From Fig 2-7 it is clear that the initial point (shown with a black X) does not lie in the Pareto set.

Several indicators can be used to measure the performance of the continuation method. However, in this section, only the function and Jacobian evaluations are shown in Table 2.2 and divided into three categories: predictor, corrector, and descent steps. The difference between the descent and corrector steps is that descent steps are those applied from the initial solution and the first approximation to the Pareto front, whereas the corrector steps are used in every step of the continuation method. The first column shows the number of steps for each category and the next columns correspond to the function and Jacobian evaluations. As shown in Table 2.2, the predictors (Alg. 1) consume only one function evaluation per step (line 8 of Alg. 1), whereas the correctors use one Jacobian evaluation per step (line 9 of Alg. 4) and several function evaluations to adjust the stepsize in Alg. 2.

Fig. 2-6 shows that the space between the solutions in objective space remains more or less constant because of Eq. 2.16. However, this is not the case in decision space as shown in Fig 2-7. Additionally, it is worth noticing that the extremes of the Pareto front are not close to the defined

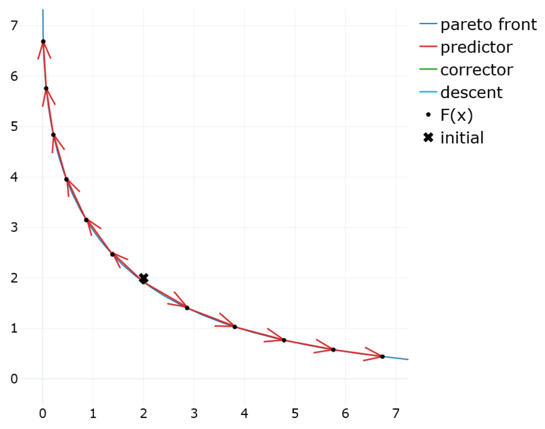


Figure 2-6: Pareto front of example BOP.

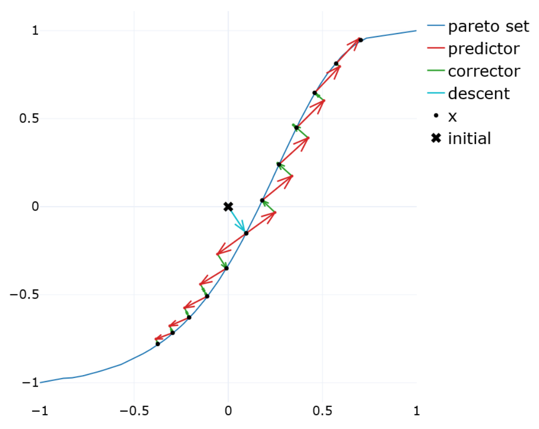


Figure 2-7: Pareto set of example BOP.

Table 2.2: Function and Jacobian evaluations in BOP example.

	steps	F evals	J evals
predictor	13	13	0
corrector	37	74	37
descent	4	10	4
total	52	97	41

boundaries $b_1 = b_2 = 7$. This happens because Eq. 2.17 used in the subroutine GoodPredictor discards the predictor if it is outside the limits and there is no mechanism to get closer to the boundaries.

Chapter 3

Multi-objective neural network problem

This section will present the formulation of the MOP from a neural network (NN) task and the adaptations to the directed search continuation method for efficiently finding its Pareto set and front.

Multi-task networks can easily be transformed into a multi-objective problem, as proposed by Ma et al. (2020) and Sener and Koltun (2018). However, the method used in this work divides the loss function of the neural model into separate objectives, which can be reasoned as the opposite of a scalarization method where multi-objective functions are merged into a single objective using weights. Consequently, the decision space is defined by a subset of the neural network weights and the Pareto front is obtained with the continuation method. The following methodology was implemented:

1. Train the NN with its loss function and a stopping criterion, e.g. overfitting criteria, a fixed number of epochs, etc. The training data subset is used for this step.

2. Select a subset of the weights of the neural network architecture as the starting point for the continuation method.

3. Execute the continuation method and find the *train* Pareto front and set using the training data subset.

4. Finally, the *test* Pareto front is computed from the *train* Pareto set obtained in the previous step but this time using the test data subset.

Since neural networks are prone to overfitting, the *test* Pareto front is a *degraded* version of the *train* Pareto front.

To formulate the MOP, consider the loss function used to train a neural network $\mathcal{L}(\Omega, W)$, where Ω is the domain of training data and W represents its weights. In special cases, the loss function can be expressed as:

$$\mathcal{L}(\Omega, W) = w_1 l_1(\Omega, W) + \cdots + w_k l_k(\Omega, W)\ , \tag{3.1}$$

where l_i are contrary loss functions and w_i is the weight associated with each function. The individual loss functions l_i have the property of decreasing at the expense of increasing another loss function l_j, where $i \neq j$.

The loss function of Eq. 3.1 can be expressed as a MOP where the domain space is a subset of the weights $W_s \subseteq W$ and the objective space is defined by the k individual loss functions, as follows:

$$\min_{W_s} F(x) = (w_1 l_1(\Omega, W_s), \ldots, w_k l_k(\Omega, W_s)) \ , \tag{3.2}$$

Depending on the problem and as shown in Section 7, the weights of only a few layers can be enough to obtain the Pareto front: $|W_s| << |W|$. The starting point for the continuation method of Alg. 5 is defined as:

$$x \in \mathbf{R}^n := W_s \subseteq W_{\text{trained}} \ , \tag{3.3}$$

where W_{trained} is the set of weights obtained after the training of the neural network.

As an example, consider the loss function

$$\mathcal{L}(\Omega, W) = l_1(\Omega, W) + l_2(\Omega, W) \ ,$$

and its respective BOP

$$\min_{W} F(x) = (l_1(\Omega, W), l_2(\Omega, W)) \ ,$$

whose Pareto front is shown in the left subfigure of Fig. 3-1. The right subfigure shows the total loss $\mathcal{L}(\Omega, W)$ in the y-axis and the first loss function in the x-axis. Consequently, each point in the right subfigure is defined by:

$$(x, y) := (l_1, l_1 + l_2) = (l_1, \mathcal{L}) \ ,$$

The initial individual corresponding to a subset of the neural network weights after training (see Eq. 3.3) is shown as a black star in domain space $F(x)$. Note that the initial point is near the local minimum as shown in the right subfigure of Fig. 3-1.

As explained by Borovykh et al. (2019), the challenge when training a neural network lies in finding the optimal tradeoff between the smoothness of the learned function and data overfit. By selecting optimum hyperparameters generalization capability of the network can be improved and the solution found might not be the best local minimum of the region. For these reasons the initial point for the continuation method (shown with the black star) is not always nicely placed in the local minimum of the loss function (see the right subfigure of Fig. 3-1 as an example).

3.1 Forecasting NN

Forecasting in computational finance has been starred by neural networks and evolutionary computation, as expressed by Tsang and Martinez-Jaramillo (2004). The research of artificial neural networks (ANN) for forecasting surged in the 90's and Zhang et al. (1998) provided a survey of the state of the art neural networks (NN) applications in forecasting with Chakraborty et al. (1992) showing how neural networks are able to learn the non-linearities in time series. Neural networks

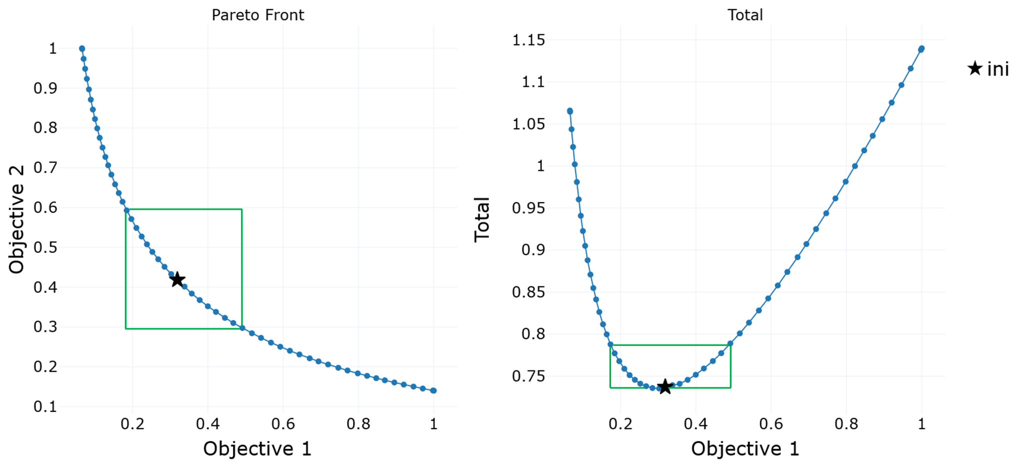


Figure 3-1: Pareto front (left) and the sum of objectives (right) of an example BOP.

have been described as black boxes and some researchers turned to evolutionary computation in an attempt to have more insight into the models. Iba et al. (1994), Kinnear et al. (1994), Neely et al. (1997) and McConaghy et al. (2000) achieved encouraging results using genetic programming in forecasting tasks.

In recent years, several NN architectures have been proposed for time-series forecasting, with fully connected NN, CNN-LSTM, ConvLSTM, and WaveNet models yielding excellent results. Convolutional filters are applied in parallel to separate time series, and thus, correlation structures can be extracted between the multivariate time series. The CNN-LSTM configuration is explored by Xue et al. (2019) and they proposed meta-heuristics to find the optimal network architecture. ConvLSTM also has proven to be useful for time series forecasting and Lee and Kim (2020) used this architecture for stock market forecasting, namely in the S&P500, KOSIPI200 and FTSE100 indexes. On the other hand, long short-term memory (LSTM) networks are state-of-the-art techniques for sequence learning, as handwriting recognition, speech recognition or time series prediction. Fischer and Krauss (2018) proposed a LSTM architecture for predictions using the S&P 500 time series. Salinas et al. (2020) and Rangapuram et al. (2018) explored a recurrent network for probabilistic forecasting. Furthermore, other architectures have also been proposed, e.g. Borovykh et al. (2018) proposed an adaptation of the recent deep convolutional WaveNet architecture for financial time series forecasting. In this work, the authors evaluated what they believed one of the best neural networks configurations for time series forecasting: a Temporal Fusion Transformer (TFT) architecture proposed by Lim et al. (2021).

This novel attention-based architecture generates high performance multi-horizon forecasting and provides interpretable insights of the temporal dynamics. The TFT combines specialized components to learn temporal relationships at different scales and select relevant features for the specific scenario.

Forecasting applications have a variety of data sources with little information about their interactions. The model used in this work considers the following three types of variables as shown in Figure 3-2.

1. Known inputs: they conform the known information about the future (e.g., hour, day of week, upcoming holiday dates, etc.)

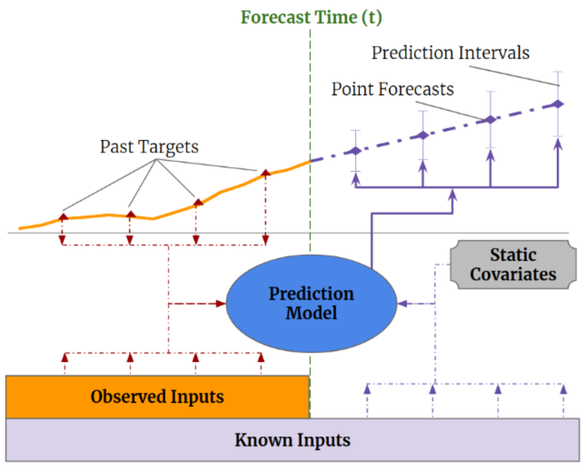


Figure 3-2: Types of inputs and outputs used for multi-horizon forecasting. Source: Lim et al. (2021).

2. Observed inputs: historical time series whose data can be measured at each time step, with no access to future values.

3. Static covariates: general information that may give insight into the application behavior

Most neural networks architectures designed for forecasting tasks are 'black-box' models where forecasts are controlled by complex nonlinear interactions between many neurons. This makes it possible to find complicated relationships between the output and input time-series; however, it is difficult to explain how models arrive at their predictions. In turn, the users might distrust the model and the developers can have trouble debugging it.

Lim et al. (2021) claim that their attention-based DNN architecture for multi-horizon forecasting achieves high performance while enabling new forms of interpretability. Four novel modules are implemented to reduce the common 'black-box' problem of NN:

1. Generation of context vectors for use in other parts of the network

2. Variable selection blocks to minimize the contributions of irrelevant inputs

3. Sequence-to-sequence layer to locally process known and observed inputs

4. Temporal self-attention decoder to learn any long-term dependencies present within the dataset

3.2 Architecture

A brief description of the architecture will be explained in this section and for further understanding, please refer to Lim et al. (2021)'s work.

A visual overview of the high level architecture is shown in Figure 3-3 and the major building blocks of the model are:

1. **Gating mechanisms** to skip over any unused components of the architecture, allowing flexibility to adapt depth and complexity depending on the dataset and scenario.

2. **Variable selection** blocks that select relevant input variables depending on its relevance.

3. **Temporal processing** to learn both long- and short-term temporal relationships from both observed and known time-varying inputs. A sequence-to-sequence layer is employed for local processing, whereas long-term dependencies are captured using a novel interpretable multi-head attention block.

4. **Prediction intervals** via quantile forecasts to determine the range of likely target values at each prediction horizon.

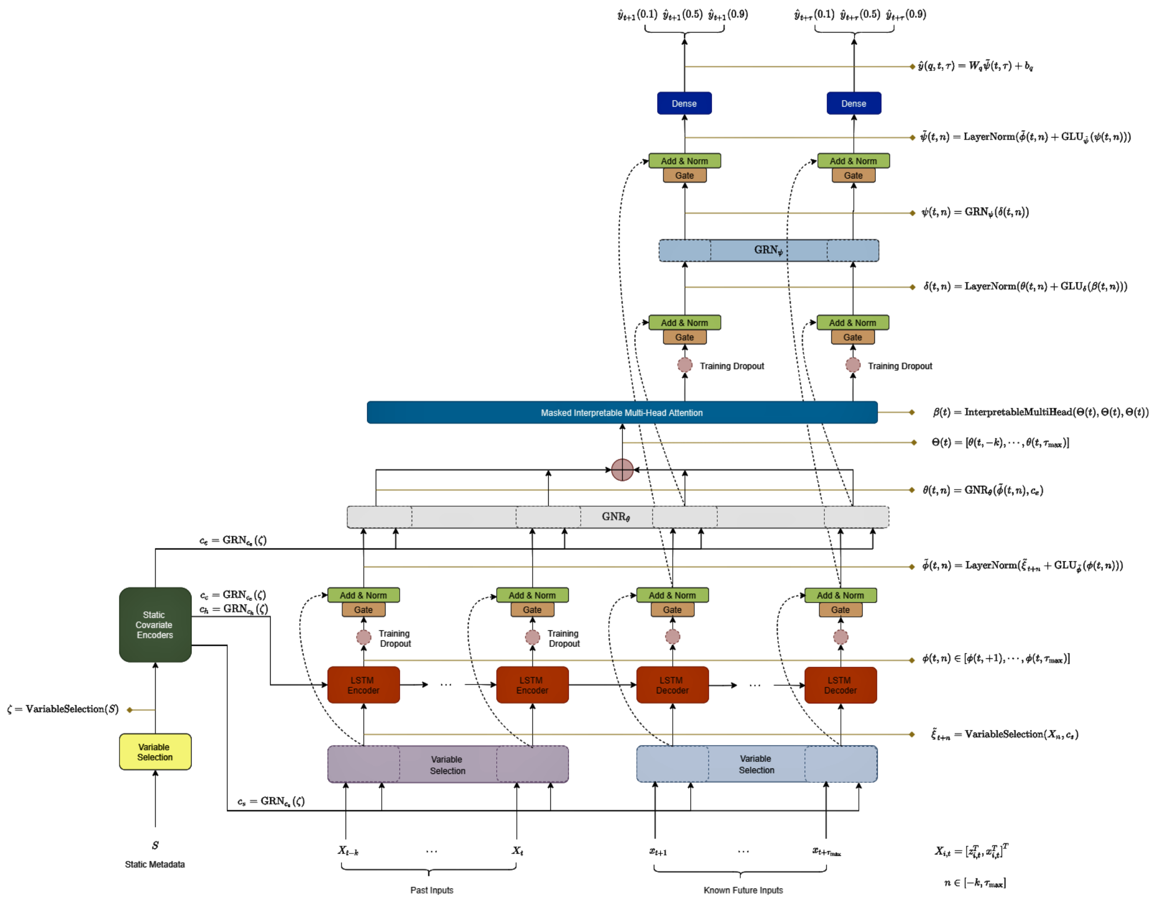
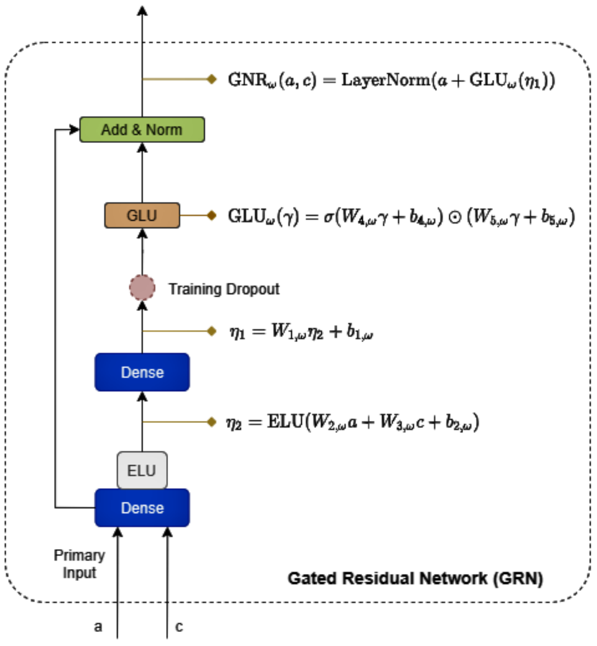


Figure 3-3: Temporal Fusion Transformer high level architecture

The Gated Residual Network (GRN) is proposed as gating mechanisms for the architecture and is shown in Figure 3-4. Its intent is to provide flexibility to suppress parts of the architecture that are not required for a given task.

Another important building block is the variable selection network. Typically, multiple time series are available, but their relevance and specific contribution to the output is unknown. Some time series variables have less predictive content, thus variable selection can greatly help model performance by suppressing the poor-content variables. Furthermore, this block generates entity embeddings for categorical variables and linear transformations for continuous variables transforming



$\mathrm{GNR}_{\omega}(a, c) = \mathrm{LayerNorm}(a + \mathrm{GLU}_{\omega}(\eta_1))$
Add & Norm
GLU
$\mathrm{GLU}_{\omega}(\gamma) = \sigma(W_{4,\omega}\gamma + b_{4,\omega}) \odot (W_{5,\omega}\gamma + b_{5,\omega})$
Training Dropout
$\eta_1 = W_{1,\omega}\eta_2 + b_{1,\omega}$
Dense
$\eta_2 = \mathrm{ELU}(W_{2,\omega}a + W_{3,\omega}c + b_{2,\omega})$
ELU
Dense
Primary Input
Gated Residual Network (GRN)
a
c

Figure 3-4: Gated Residual Network

each input variable into a (d_{model})-dimensional vector. This dimension is used throughout the model in all subsequent layers. Figure 3-5 shows a diagram of the variable selection building block.

After the variable selection network, a LSTM encoder-decoder generates a set of uniform temporal features which serve as inputs into the temporal fusion decoder. The self-attention layer's purpose is to learn long-range dependencies that may be challenging for RNN-based architectures.

The final forecast is obtained using a linear transformation of the output from the temporal fusion decoder:

$$\hat{y}_i(q, t, \tau) = \mathbf{W}_q \tilde{\psi}(t, \tau) + b_q \tag{3.4}$$

where $\mathbf{W}_q \in \mathbb{R}^{1 \times d_{model}}$, $b_q \in \mathbb{R}$ are the weights of the dense layer for the specified quantile q and d_{model} is the hidden state size, common across the TFT model. The model generates prediction intervals on top of point forecasts by the simultaneous prediction of various percentiles (e.g. 10th, 50th and 90th) at each time step t.

The quantile forecast task of the model can be expressed as the following prediction equation:

$$\hat{y}_i(q, t, \tau) = f_q(\tau, y_{i,t-k:t}, z_{i,t-k:t}, x_{i,t-k:t+\tau}, s_i) \tag{3.5}$$

where $\tau \in \{1, \cdots \tau_{\max}\}$, q is the quantile of the τ-step-ahead forecast, y is the target variable, z are the known variables, x are the observed inputs, s the static covariates and $f_q(\cdot)$ is the prediction model. The forecasts are computed simultaneously for the complete forecast horizon $t : t + \tau_{\max}$.

The model was trained by jointly minimizing the quantile loss, aggregated across all quantile outputs:

$$\mathrm{QL}(\Omega, W) = \sum_{y_t=\Omega} \sum_{q \in Q} \sum_{\tau=1}^{\tau_{\max}} \frac{\mathrm{QL}(y_t, \hat{y}(q, t-\tau, \tau), q)}{M\tau_{\max}} \tag{3.6}$$

where Ω is the domain of the training data containing M is the number of samples and W represents

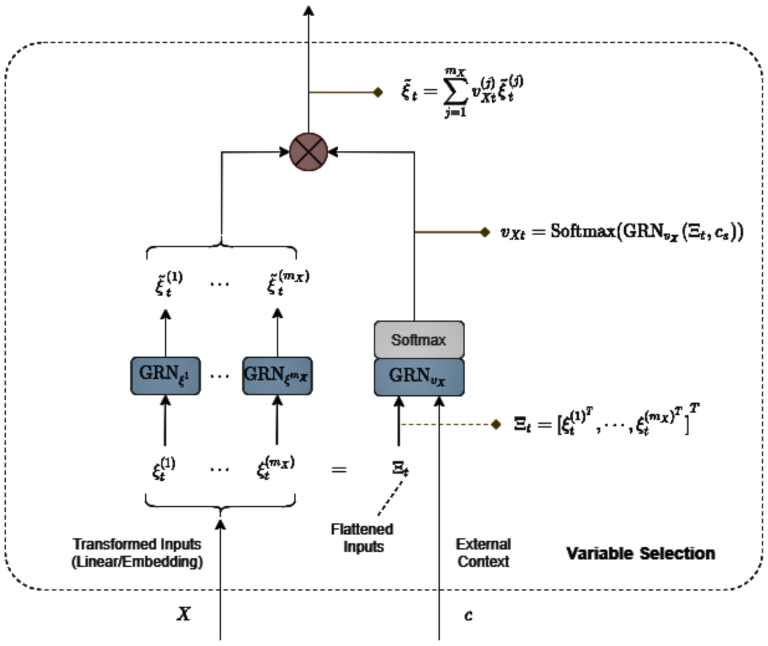

$\tilde{\xi}_t = \sum_{j=1}^{m_X} v_{X_t}^{(j)} \tilde{\xi}_t^{(j)}$
$v_{X_t} = \text{Softmax}(\text{GRN}_{v_X}(\Xi_t, c_s))$
$\tilde{\xi}_t^{(1)}$ ... $\tilde{\xi}_t^{(m_X)}$
Softmax
GRN_{ξ^1} ... $\text{GRN}_{\xi^{m_X}}$
GRN_{v_X}
$\Xi_t = [\xi_t^{(1)^T}, \dots, \xi_t^{(m_X)^T}]^T$
$\xi_t^{(1)}$ ... $\xi_t^{(m_X)}$ = Ξ_t
Transformed Inputs (Linear/Embedding)
Flattened Inputs
External Context
Variable Selection
X
c

Figure 3-5: Variable Selection Network

the model's weights, Q is the set of output quantiles and the quantile loss (QL) for a single forecast is expressed as:

$$\text{QL}(y, \hat{y}, q) = q(y - \hat{y})_+ + (1 - q)(\hat{y} - y)_+ \tag{3.7}$$

where $(\cdot)_+ = \max(0, \cdot)$.

For out-of-sample testing, the model was evaluated using the normalized quantile losses proposed by Lim et al. (2021).

$$q\text{-Risk} = \frac{\sum_{q \in \tilde{\Omega}} \sum_{\tau=1}^{\tau_{\max}} \text{QL}(y_t, \hat{y}(q, t - \tau, \tau), q)}{\sum_{y \in \tilde{\Omega}} \sum_{\tau=1}^{\tau_{\max}} |y_t|} \tag{3.8}$$

where $\tilde{\Omega}$ is the domain of the test data. For convenience, the aggregation and normalization of Equation 3.8 will be expressed as $\Phi(\cdot)$, i.e.,

$$q\text{-Risk} = \Phi(\text{QL}(y_t, \hat{y}(q, t - \tau, \tau), q))$$

3.3 Multi-objective framework

Before adapting the training of the model into a multi-objective problem (MOP), a brief overview of the prediction interval (PI) multi-objective evaluation indicators is presented.

As mentioned in the introduction, the PIs' quality is traditionally measured using variants of the prediction interval coverage probability (PICP) and prediction interval width (PIW). The PICP represents the probability that the true target lies between the lower and upper bound, whereas the PIW simply measures the difference of the estimated upper $y_{i,u}$ and lower bound values $y_{i,l}$.

$$\text{PICP} = \frac{1}{T}\sum_{i=1}^{T} a_i$$

$$\text{where } \; a_i = \begin{cases} 1 & y_i \in [y_{i,l}, y_{i,u}] \\ 0 & y_i \notin [y_{i,l}, y_{i,u}] \end{cases}$$

The coverage probability can be extended to the PI estimation error (PIEE) to quantify the error in case the target lies outside the PI. PIEE provides a measure of the quality of PIs.

$$E_i = \begin{cases} (y_i - y_{i,u}) & \text{if } \; y_i \geq y_{i,u} \\ 0 & \text{if } \; y_{i,l} \leq y_i \leq y_{i,u} \\ (y_{i,l} - y_i) & \text{if } \; y_i \geq y_{i,u} \end{cases}$$

the overall PIEE is computed as:

$$\text{PIEE} = \frac{\sum_{i=1}^{T} E_i}{T \times (y_{\max} - y_{\min})}$$

Additionally, the PIW is expressed as:

$$\text{PIW} = \sum_{i=1}^{T} (y_{i,l} - y_{i,u})$$

A useful modification of the PIW is the prediction interval normalized average width (PINAW), which objectively compares PIs regardless of the magnitudes of the true targets.

$$\text{PINAW} = \frac{\sum_{i=1}^{T} (y_{i,l} - y_{i,u})}{T \times (y_{\max} - y_{\min})}$$

where $y_{\min}$ and $y_{\max}$ represent the minimum and maximum values taken by the target, respectively.

It should be noted that PICP/PIEE and PIW/PINAW are conflicting in nature, as increasing the coverage may come at the cost of increasing the width, and reducing the width may reduce the possibilities of covering the desired targets.

Depending on the forecast application, the decision maker (DM) may need to fine-tune the performance of the quantile prediction model. Suppose the forecast task is used as part of an investing strategy in financial markets. The strategy analyst might need to fine-tune the quantile interval predictions according to his/her specific needs without losing performance. A larger upper quantile range and smaller lower quantile range can reduce the risk in a long strategy and simultaneously show the potential upside. With this motivation in mind, the training of a quantile forecast neuronal network is formulated as a multi-objective problem.

The quantile loss (QL) shown in Equation 3.7 can be divided into *Quantile Coverage Error* (QCE) and *Quantile Estimation Error* (QEE) with the help of an auxiliary variable γ.

$$
\begin{aligned}
\mathrm{QCE}(y,\hat{y},q) &= \gamma q(y-\hat{y})_+ + (1-\gamma)(1-q)(\hat{y}-y)_+ \\
\mathrm{QEE}(y,\hat{y},q) &= (1-\gamma)q(y-\hat{y})_+ + \gamma(1-q)(\hat{y}-y)_+ \\
\text{where } \gamma &= \begin{cases} 1 & \text{if } q \geq 0.5 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
\tag{3.9}
$$

QCE is analogous to the PIEE since it quantifies if the forecast is outside the quantile range, whereas QEE is analogous to the PINAW and measures the meaningfulness of the quantile forecast. The two measures are conflicting in nature, since QCE increases if the forecast is outside the quantile range, and on the contrary, QEE increases if the quantile range is too broad.

Analogous to Equation 3.8, the risk indicators *Quantile Coverage Risk* (QCR) and *Quantile Estimation Risk* (QER) are defined as:

$$
\mathrm{QCR} = \Phi(\mathrm{QCE}(y_t, \hat{y}(q, t-\tau, \tau), q)) \tag{3.10}
$$

$$
\mathrm{QER} = \Phi(\mathrm{QEE}(y_t, \hat{y}(q, t-\tau, \tau), q)) \tag{3.11}
$$

The multi-objective problem is defined y considering only one pair of quantiles: a lower bound quantile, i.e., $q_l < 0.5$ and an upper bound quantile, i.e. $q_u > 0.5$, and their respective errors from Equation 3.9 being: QCE_l, QEE_l, QCE_u, QEE_u.

Following Equations 3.10 and 3.11, four risk indicators are obtained: QCR_l, QER_l, QCR_u, QER_u, by applying the function $\Phi(\cdot)$ which aggregates and normalizes the risk through a given set of data.

This setup gives rise to the formal definition of the dual MOP:

$$
\begin{aligned}
\min_{\mathbf{W}_q, b_q} \; &(\mathrm{QCR}(q, \hat{y}, \mathbf{W}_q, b_q, \tau_{\max}, \tilde{\Omega}), \\
&\mathrm{QER}(q, \hat{y}, \mathbf{W}_q, b_q, \tau_{\max}, \tilde{\Omega})) \\
&\forall \;\; q \in \{q_l, q_u\}
\end{aligned}
\tag{3.12}
$$

subject to

$$
\begin{aligned}
g(W_q, b_q) \leq& 0 \\
-1 \leq& w_q \in \mathbf{W}_q \leq 1 \\
-1 \leq& b_q \leq 1 \; ,
\end{aligned}
$$

where $\tilde{\Omega}$ is the domain of y in the dataset, $\hat{y}$ is the prediction of the model, $\tau_{\max}$ is the number of prediction steps in the future, $\mathbf{W}_q \in \mathbb{R}^{1\times d_{model}}$, $b_q \in \mathbb{R}$ are the weights of the dense layer for the specified quantile q.

The two different MOP formulations that are solved in this work are the following:

1. **Last layer remains unfrozen**: All the model layers' weights are frozen except for the last dense layer. In this case, the lower quantile risks, i.e., QCR_l, QER_l, are not in conflicting nature with the upper quantile risks. In fact, both risks are totally independent.[1] Table 3.1

[1] the practical implementation consists in obtaining the output before the final layer and doing a simple matrix

Table 3.1: Dual bi-objective problem

BOP	Objectives
Lower quantile	QCR_l, QER_l
Upper quantile	QCR_u, QER_u

shows the dual BOP with the objectives involved. The chromosome of individuals has the following structure:

$$\text{ind} = [w_1, w_2, \cdots, w_{d_{\text{model}}}, b]$$

for each of the quantiles $q \in \{q_l, q_u\}$. This means that the size of the individuals is:

$$|\text{ind}| = d_{\text{model}} + 1$$

where $d_{\text{model}} = 386$ and $|ind|$ coincides with the small BOP size of Table 3.2.

2. **Subset of layer remains unfrozen**: Several upstream layers are optimized, and unfrozen shared layers affect both the upper and lower quantile forecasts. However, for the sake of simplicity and evaluation of results, only one quantile[2] is optimized to keep the MOP restricted to two objectives, i.e. QCR and QER.

   The chromosome of individuals, depends on the subset of unfrozen layers (see Eq. 3.3). We propose two BOP cases, one using only the last layer and a medium size problem with the last 5 layers. The individual sizes are shown in Table 3.2.

Table 3.2: BOP sizes.

Name	All weights (W_{trained})	Unfrozen weights (W_s)
small	3,394,193	387
medium	3,394,193	240,515

Finally, Table 3.3 shows the dataset overview and neural network hyperparameters used. A total of 24 input variables were used by the model to predict the 3-period exponential moving average of the S&P futures [3].

3.4 Methodology

In this work, the methodology proposed by Ak et al. (2015), Zhou et al. (2021), and López et al. (2022) is used, where a NN was trained and then the Pareto front was found using a MOO method. The detailed steps of the solution algorithm are provided as follows.

1. **Dataset preparation**. The dataset used for the prediction task consists mainly of the price of the S&P Futures, indicators derived from the price, and additional correlated price time

multiplication between the previous output and the weights to obtain the final predictions.

[2]the lower quantile is optimized

[3]the neural network architecture and dataset can be found in the repository: `https://github.com/samlopezruiz/TimeseriesQuantileForecast`

Table 3.3: Dataset information and TFT configuration.

	1h-freq S&P
Dataset	
Train samples	27,758
Validation samples	5,917
Test samples	5,917
Network Parameters	
Input steps	48
Output steps	5
Dropout rate	0.3
State size	128
Number of heads	4
Training Parameters	
Minibatch size	64
Learning rate	0.01
Max gradient norm	0.01

series. The dataset is defined from January 2015 to June 2021 with an hourly frequency and was divided into groups of 8 consecutive weeks and each group is then split into 75% train, 15% validation and 15% test data. Figure 3-6 shows an example of the dataset subsets.

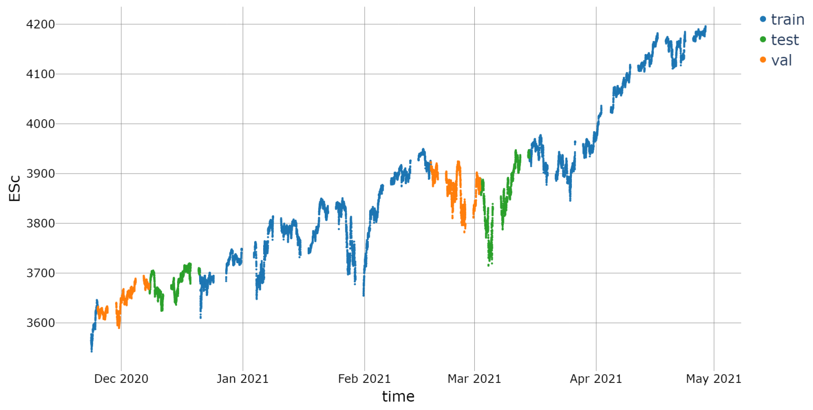


Figure 3-6: 1h freq. ES closing price subsets in color

2. **Training**. The NN model was trained with the original quantile loss function and using the training subset (see López et al. (2022) for more details). The final set of weights is obtained after training the NN model with *stochastic gradient descent* (SGD).

3. **Multi-objective Pareto front**. The multi-objective framework can be viewed as a special case of knowledge transfer. Some weights are frozen and the unfrozen weights form the BOP. In chapter 4, the the Pareto front is obtained using evolutionary multi-objective algorithms (i.e. NSGA-II and NSGA-III), and in chapter 5 the BOP is solved using the proposed stochastic directed search method (SDS).

3.5 Train and test data subsets

It is common practice to train neural networks with a subset of the data and to evaluate its performance with the validation data subset. In MOO for NN tasks, the data subsets play the following roles:

1. The *train* Pareto set and front is obtained using the same data subset used to train the NN.
2. An additional *test* Pareto front is computed from the Pareto set obtained in the previous step, but using the validation data subset.

Chapter 4

MOEA results

This section shows the Pareto fronts obtained when using the NSGA-II and NSGA-III methods. The data preparation and handling were entirely conducted in Python 3.9, relying on the packages numpy, scipy, pandas and tensorflow. The multi-objective algorithms were implemented using the powerful library pymoo explained by Blank and Deb (2020). The neural network models are trained on a Nvidia GeForce RTX 3070 8Gb GPU and the multi-objective framework was executed using an Intel(R) Core(TM) i7-11800H CPU @ 4.6 GHz.

Due to the training stochastic nature, the training was repeated 5 times independently for each pair of lower and upper quantile in the following three configurations:

$$(q_l, q_u) = \{(0.1, 0.9), (0.2, 0.8), (0.3, 0.7)\}$$

Additionally, the mean quantile $q = 0.5$ was always computed, hence, the label given throughout this work to the quantiles configuration is: 'q: 1-5-9', 'q: 2-5-8', 'q: 3-5-7'. Table 4.1 shows the performance obtained, where the first column 'val loss' corresponds to the quantile loss of Equation 3.6 applied to the validation subset, and the second and third column correspond to the quantile risk of Equation 3.8 computed in the test subset.

It is worth noticing that the closer the lower and upper quantiles are to 0.5, the larger the q-risk involved. These results indicate that smaller quantile ranges $(q_u - q_l)$ are a more difficult forecasting task than large quantile ranges, with $q = 0.5$ being the most difficult task. Equation 3.9 is responsible for this behavior since the penalty is less if the prediction is within the quantile range and vice versa.

Table 4.1: Model performance with 5 independent trainings

	val loss	lower q-risk	upper q-risk
q: 1-5-9	0.524 (0.014)	0.548 (0.034)	0.475 (0.014)
q: 2-5-8	0.643 (0.006)	0.764 (0.018)	0.703 (0.007)
q: 3-5-7	0.719 (0.002)	0.893 (0.005)	0.848 (0.002)

4.1 MOEA selection

As explained in section 3, the MOP consists of two bi-objective problems, one for each quantile. But before exploring in detail the results obtained, 20 independent executions were performed on each MOEA to determine the best one for this task. A normalized quantile risk above 1 indicates a bad performance for that indicator, thus, it was considered appropriate to constraint the objective search space to the square limited by

$$\mathrm{QCR}(x) \leq 1 \cap \mathrm{QCE}(x) \leq 1 \tag{4.1}$$

where x is the BOP individual. These restrictions are specified as constraints in the MOP formulation in Equation 3.12. The MOEAs considered in this work are the commonly used *Non Dominated Sorting Genetic Algorithm* versions II and III (NSGA-II and NSGA-III). Table 4.2 shows the parameters used in each independent execution.

Table 4.2: Parameters for MOEA selection

Method	Value
Representation	Real
Initial Population	100
Number of Generations	100
Crossover	SBX: $\eta = 15, p = 0.9$
Mutation	Polynomial: $\eta = 20, p = 1/n_v$
Selection	Tournament (5 ind)
Termination	n generations
Repetitions	20

where n_v is the size of the individual.

The mean hypervolume with its standard deviation obtained in 20 executions for the same model (with frozen layers) and trained with $q_l = 0.1$ and $q_u = 0.9$ are shown in Table 4.3. The reference point $p_r = (10, 10)$ was used to calculate the hypervolume. From Table 4.3, it is clear that NSGA-II obtained a better mean hypervolume for both MOPs (lower and upper quantiles).

Table 4.3: Hypervolume of PF using different MOEAs

	Hv lower quantile	Hv upper quantile
NSGA-II	**97.568** (0.017)	**97.953** (0.012)
NSGA-III	97.530 (0.015)	97.900 (0.021)

To test the statistical significance of NSGA-II over NSGA-III for this task, the Wilcoxon rank (WR) test is applied between the 20 hypervolumes obtained by each algorithm per problem. With a p-value threshold of $p{-}value \leq 0.05$ for the Wilcoxon rank, it can be concluded that the hypervolume obtained with NSGA-II is significantly better than with NSGA-III.

To determine a Condorcet winner, comparisons are done in a pairwise fashion (algorithm X, algorithm Y) for each problem, where victories are granted if the WR p-value is below the specified threshold. A Condorcet winner is determined if an algorithm wins all its individual encounters: (algorithm X, algorithm Y) > (algorithm Y, algorithm X). Considering the two 20 hypervolumes for each quantile, the NSGA-II wins its individual encounters vs NSGA-III and can be considered the Condorcet winner. This can be confirmed with the hypervolume history average during all generations as shown in Figure 4-1.

4.2 Pareto fronts for different quantiles

In this section, the Pareto fronts (PF) of models trained with different quantiles will be analyzed. The results shown in this section were obtained with NSGA-II and the parameters shown in Table 4.2, including the constraints of Equation 4.1.

The first column of Figure 4-2 shows the Pareto front for the lower and upper quantile for three models trained with different quantiles, whereas the second column shows the *quantile total risk*, QTR = QCR + QER, along the quantile coverage risk (QCR) axis. Additionally, the original solution found with the model's training using SGD is shown with a black star.

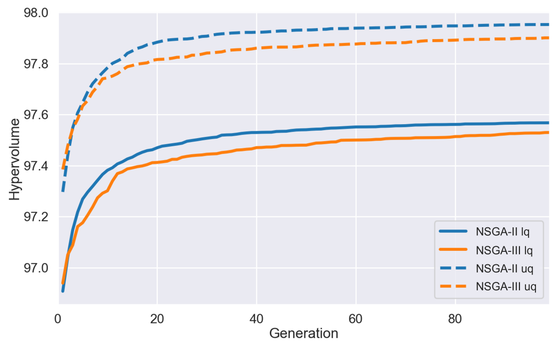


Figure 4-1: Average hypervolume history across the generations.

From the second row of Figure 4-2, it can be observed that the original solution found by the SGD training is located near the local minimum. This behavior is expected since the model is minimizing the quantile loss of Equation 3.6 which coincides with the aggregation of QCR + QER.

It is worth noticing that the geometric positions differ for the three PFs: the PF for 'q: 1-5-9' is closer to the ideal vector $(0, 0)$, whereas the PF for 'q: 3-5-7' is the farthest away. This result does not mean that the model trained via SGD using quantiles 'q: 1-5-9' exhibits better performance than the model trained with quantiles 'q: 3-5-7'. It merely shows that the forecasting task is harder for smaller quantile ranges as was previously discussed in Table 4.1.

Tolerance windows are shown in Figure 4-2 to exemplify the motivation behind finding the PF of an already trained quantile model. By setting an increment threshold in the total error QTR (second row), the decision maker (DM) can modify the extent of these windows which can be considered as the 'sweet spot' where the model is fined tuned. As an example, the increment was defined as 5% and the six windows in the PF are detailed in Table 4.4. The range of the QCR and QER indicators within the tolerance window is large, with the upper extremes doubling sometimes the lower extremes.

4.3 Different quantiles comparison

From the previous section, it is unclear how the PFs of models with different quantiles relate to each other. To compare the PFs under a 'universal' indicator, Equation 3.9 is modified using

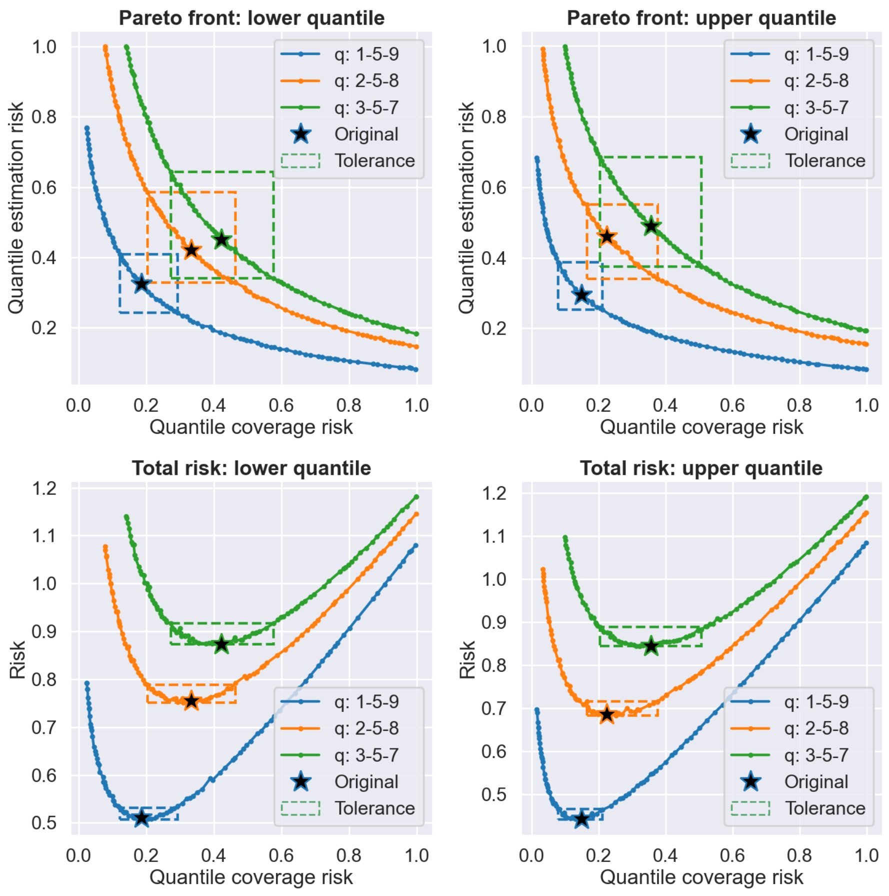


Figure 4-2: Pareto fronts for different quantiles.

equal penalties for over or undershooting predictions as shown in Equation 4.2. Their respective aggregation (see Equations 3.10 and 3.11) is applied to obtain the *equally weighted* risks QCR_{eq} and QER_{eq} respectively.

$$\begin{aligned} \text{QCE}_{eq}(y, \hat{y}, q) &= 0.5\gamma(y - \hat{y})_+ + 0.5(1-\gamma)(\hat{y} - y)_+ \\ \text{QEE}_{eq}(y, \hat{y}, q) &= 0.5(1-\gamma)(y - \hat{y})_+ + 0.5\gamma(\hat{y} - y)_+ \\ \text{where } \gamma &= \begin{cases} 1 & \text{if } q \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{4.2}$$

Figure 4-3 shows the Pareto fronts obtained when measuring the solutions of the Pareto fronts from Figure 4-2 but using the equally weighted indicators QCR_{eq} and QER_{eq}. The first noticeable result is that the three PFs lie in the same global PF, with the 'q: 1-5-9' PF extending for low QCR and the 'q: 3-5-7' extending for low QER.

The tolerance windows and original solutions found during the model's trainings also lie in the global PF and their positions correspond to their original intent. The original solution for model 'q: 1-5-9' reduces QCR_{eq} at the cost of increasing QER_{eq}, whereas the model 'q: 3-5-7' does the opposite and the model 'q: 2-5-8' solution lies between the other two configurations. This behavior

Table 4.4: Tolerance windows with 5% variance in total error

	quantiles	QCR window	QER window
q_l	q: 1-5-9	0.12-0.29, 138%	0.24-0.41, 68%
	q: 2-5-8	0.20-0.46, 127%	0.33-0.58, 78%
	q: 3-5-7	0.27-0.58, 111%	0.34-0.64, 89%
q_u	q: 1-5-9	0.08-0.21, 169%	0.25-0.39, 53%
	q: 2-5-8	0.17-0.38, 128%	0.34-0.55, 62%
	q: 3-5-7	0.20-0.51, 148%	0.38-0.68, 82%

is expected and reflects the original intent of the DM when selecting the quantiles to train the model.

Judging by the shape of the PFs in Figure 4-3, it is interesting to note that a slight decrease in QCR will make the QER increase significantly when the coverage risk is relatively small, likewise, a small decrease in QER will bring a significant increase in QCR when the QER is relatively small. This behavior is also reported by Zhou et al. (2021) with prediction intervals (PIs).

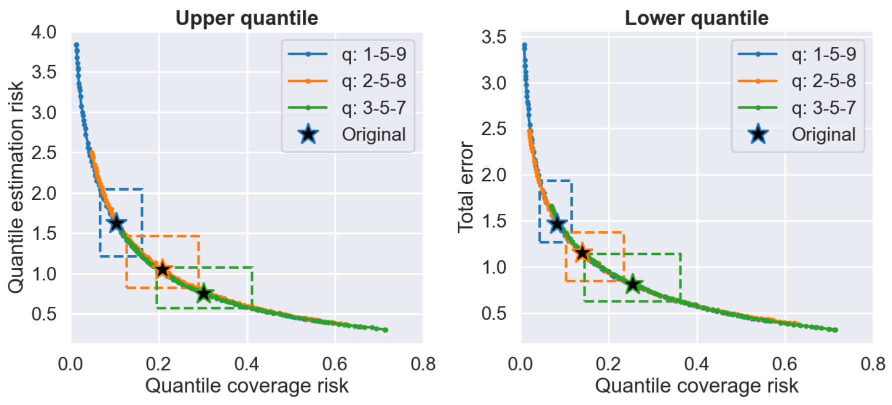


Figure 4-3: Pareto fronts with equally weighted indicators.

4.4 Finding the global Pareto front

The last section raises the question of whether the global Pareto front can be found from any of the models trained with different quantiles. To test this hypothesis, 5 models are trained for each of the three different quantiles' configurations. Due to the stochastic nature of MOEAs, the MOO is done 3 times for each of the previous results. This means that 3 MOO results are obtained for each of the 5 models trained with the same quantile configuration, adding a total of 15 results.

The MOO is done using QCR$_{eq}$ and QER$_{eq}$ as objectives and the constraints are modified to:

$$\text{QCR}_{eq}(x) \leq 1 \cap \text{QCE}_{eq}(x) \leq 2.5 \tag{4.3}$$

Table 4.5 shows the mean and standard deviation obtained for the hypervolume for 15 executions using the reference point $p_r = (10, 10)$. The middle configuration 'q: 2-5-8' seems to achieve a better performance, but to test for statistical significance, the Wilcoxon rank (WR) test is applied between the pair of 15 hypervolumes obtained in each quantile configuration.

Table 4.5: PF Hypervolume for quantiles configuration

	Hv lower quantile	Hv upper quantile
q: 1-5-9	97.235 (0.146)	97.152 (0.129)
q: 2-5-8	**97.287** (0.042)	**97.172** (0.044)
q: 3-5-7	97.215 (0.034)	97.157 (0.039)

The p-values of the Wilcoxon rank test are shown in Table 4.6, where the two values per cell represent each of the two MOPs: lower quantile and upper quantile. With a p-value threshold of $p - value \leq 0.05$, only the performance of the configuration 'q: 2-5-8' is better than 'q: 3-5-7' in the lower quantile MOP. There is no Condorcet winner since no configuration wins its individual encounters against the other two. The result suggests there exists no apparent difference in the global PF obtained from models trained with different quantiles from the multi-objective point of view.

The fact that the PFs are almost identical for the three cases, does not mean predictions are the same since the frozen layers' weights still play an important part in the final forecast. Other indicators could be found to prefer one model over another; however, the risk indicators do not suggest a preference between models trained with different quantiles.

Under the previous conclusions, the tuning of the quantile risk can be done completely *a posteriori* and once the model is trained. Furthermore, the quantile risk tuning can cover the whole range of quantiles configurations, e.g., $q = [0.1, 0.2, \cdots, 0.9]$.

Table 4.6: Wilcoxon ranking test p-values

	q: 1-5-9	q: 2-5-8	q: 3-5-7
q: 1-5-9	-	0.3937	0.1648
	-	0.1648	0.0957
q: 2-5-8	0.6063	-	**0.0000**
	0.8352	-	0.2738
q: 3-5-7	0.8352	1.0000	-
	0.9043	0.7262	-

4.5 Forecasts visualization

This section shows visual examples of how the quantile forecasts can be modified by the DM using the Pareto front. As an example, the PF for the model trained with quantiles 'q: 2-5-8' is used throughout this section. The training and multi-objective optimization are done using the parameters from Tables 3.3 and 4.2 respectively, the constraints of Equation 4.3 are also applied.

Figure 4-4 shows the 1h-freq quantile forecast for the date range: 2019-09-26 12:00 to 2019-09-27 12:00. The forecast on the left is the original solution found by training the model using stochastic descent, whereas the forecast on the right corresponds to the solutions selected by the DM when minimizing QCR for both quantiles (within the tolerance window) as shown in Figure 4-5.

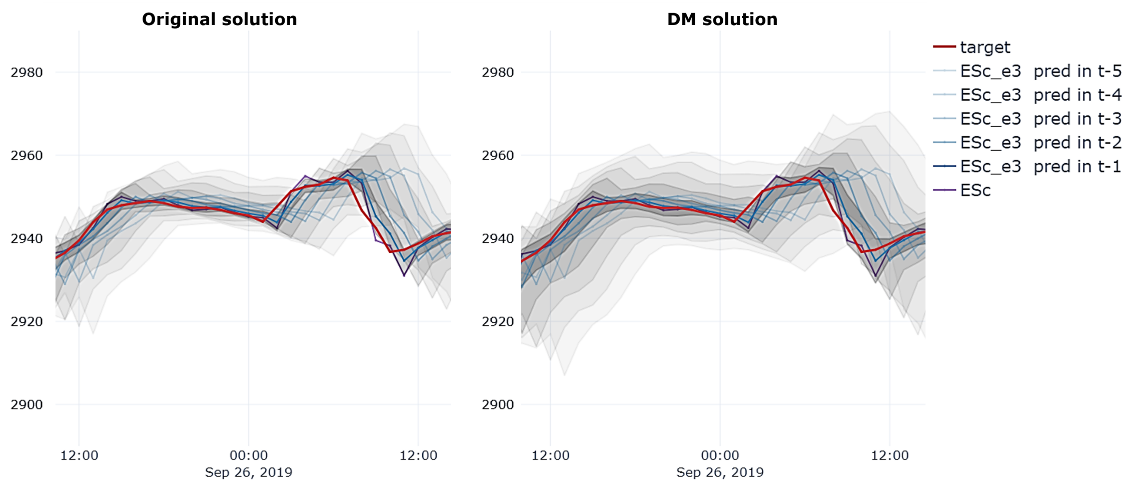


Figure 4-4: Forecasts of original vs. selected solutions.

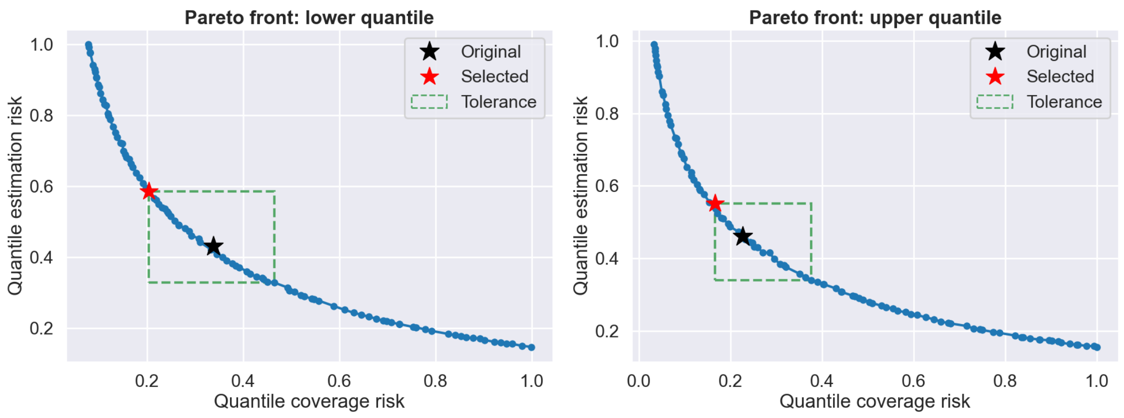


Figure 4-5: Selected solution inside window minimizing QCR.

Chapter 5

Stochastic directed search continuation method.

This work proposes the following contributions to the directed search method adapted for neural networks optimization:

- The computation of the Jacobian using batches of data.
- A stop criteria to obtain a subset of the Pareto front based on an increment of the sum of the objectives.
- A bisection method for the predictor to get closer to the boundaries in objective space.

5.1 Jacobian with batches

For neural networks MOPs, the Jacobian is the gradient of the k objective functions with respect to the weights of the layers (see Eq. 2.6). In practice, the gradients are computed through *automatic differentiation*, which is a technique used in the training of neural networks with backpropagation. Automatic differentiation evaluates derivatives at particular numeric values and does not construct symbolic expressions for derivatives, which greatly simplifies the computation of gradients in complex architectures.

Additionally, when training neural networks, the weights' update is executed using the well-known stochastic gradient descent (SGD) routine. The basic idea behind this technique is to update the parameters with an approximation of the *true gradient*, which is calculated with a mini-batch of the whole dataset. The advantages of SGD include smaller memory requirements and fast convergence because of frequent parameter updates. However, the *noisy gradient* of the mini-batch can point toward other directions which impacts convergence.

The continuation method presented in this work incorporates both techniques for neural network training: automatic differentiation and stochastic mini-batch gradient. No change is needed in the algorithms already presented since only the computation of the Jacobian is altered.

Consider the set of l data batches $\{b_0, \ldots, b_m\}$, where m is defined with the hyperparameter *batch size* (BS) as follows:

$$m = \left\lfloor \frac{N}{\text{BS}} \right\rfloor ,$$

where N is the total number of samples in the train data subset.

Every time the Jacobian $J(x)$ is needed, the gradient with the respect to the weights for the k objective functions is computed using the next batch of data b_{i+1}, as shown in Class structure 6. When $i + 1 > m$, the count is reset to $i = 0$.

Class 6 Jacobian with batches.

```
Routine INITIALIZATION($\{b_0, \ldots, b_m\}$)
    $B \leftarrow \{b_0, \ldots, b_m\}$
    $i \leftarrow 0$
    $m \leftarrow |B|$
end Routine

Routine NOISYJACOBIAN($x$)
    $i \leftarrow (i + 1) \mod m$
    return $J(x, B[i])$
end Routine
```

5.2 Stop criteria based on total increment

When finding the Pareto front where the original loss function of the neural network is divided into k objectives, the decision maker might not be interested in areas where the total loss function $\mathcal{L}$ increases above a given threshold from the initial solution. The green rectangle in Fig. 3-1 shows an example of the area of interest for a decision maker, where the total loss function remains below a 15% increment from the original solution. Under this approach, only a subset of the Pareto front is of interest, and a stopping criterion is proposed to avoid additional and unnecessary computational costs.

To account for the increment in the objectives total, subroutine GOODPREDICTOR of Alg. 1 is complemented with an additional condition. The subroutine is specified as a method of the Class structure 7 that saves the initial solution $F(x_{\text{ini}})$ and the boundaries of Eq. 2.17,

Class 7 Routine to discard predictors.

```
Routine INITIALIZATION($m, F(x_{\text{ini}}), \{b_1, \ldots, b_k\}$)
    $F_{\max} \leftarrow (1+m) \sum_i^k f_i(x_{\text{ini}})$
    $B \leftarrow \{b_1, \ldots, b_k\}$
end Routine
```

```
Routine GOODPREDICTOR($F(x)$)
    for $i \in \{1, \ldots, k\}$ do
        if $f_i(x) > B[i]$ then
            return False
        end if
    end for
    if $\sum_i^k f_i(x) > F_{\max}$ then
        return False
    end if
    return True
end Routine
```

where $\{b_1, \ldots, b_k\}$ are the boundaries for each objective (see Eq. 2.17) and m is the threshold of increment in the total objectives specified by the user and it is measured with respect to the initial solution $F(x_{\text{ini}})$. This routine allows the decision maker to explore only a subset of the Pareto front and save significant computational costs.

5.3 Boundary approximation

As shown in Fig. 2-6, the extremes of the Pareto front are not close to the boundaries $b_1 = b_2 = 7$. This might not be a problem when using a small step size τ; however, for larger step sizes, the Pareto front extremes will fall far away from the boundaries. This happens because Eq. 2.17 used in the subroutine GOODPREDICTOR discards the predictor if it is outside the limits and there is no mechanism to get closer to the boundaries.

Alg. 1 is modified to include a bisection routine to change the predictor step size t and approximate as close as possible to the boundary specified by GOODPREDICTOR. A step size threshold Ψ is defined by the user as a stopping criterion to avoid excessive computation cycles.

Algorithm 8 Predictor with boundary approximation.

Input: Point $x \in \mathbb{R}^n$ in $\mathcal{PF}$, Jacobian $J(x)$, constant τ, normal vector α, and step size threshold Ψ
Output: Set of predictors

```
1:  Routine PREDICTOR($x, \alpha, J(x), \tau, \lambda$)
2:      $q_i \leftarrow$ QRDECOMPOSITION($\alpha$)                ▷ Eq. 2.12
3:      $P \leftarrow \emptyset$
4:      for $i \in \{1, \dots, k-1\}$ do
5:          $v_i \leftarrow J^+(x) q_{i+1}$                        ▷ Eq. 2.14
6:          $t \leftarrow \tau / ||J(x) v_i||$                     ▷ Eq. 2.16
7:          $x_p \leftarrow x + \lambda t v_i / ||v_i||$           ▷ Eq. 2.15
8:          if GOODPREDICTOR($F(x_p)$) then
9:              $P \leftarrow P \cup \{x + t v_i / ||v_i||\}$
10:         else
11:             $t_h \leftarrow t, t_l \leftarrow 0$
12:             while True do
13:                 $t \leftarrow (t_h + t_l)/2$
14:                 $x_p \leftarrow x + \lambda t v_i / ||v_i||$
15:                 if $t_h - t_l < \Psi$ then
16:                     if GOODPREDICTOR($F(x_p)$) then
17:                         $P \leftarrow P \cup \{x + t v_i / ||v_i||\}$
18:                     else
19:                         $P \leftarrow P \cup \{x - t_l v_i / ||v_i||\}$
20:                     end if
21:                     break
22:                 end if
23:                 if GOODPREDICTOR($F(x_p)$) then
24:                     $t_l \leftarrow t$
25:                 else
26:                     $t_h \leftarrow t$
27:                 end if
28:             end while
29:         end if
30:     end for
31:     return $P, t$
32: end Routine
```

Lines 16-19 from Alg. 8 guarantee $F(x_p)$ is within the boundaries once the stopping condition $t_h - t_l < \Psi$ is met.

Chapter 6

Hyperparameter impact on SDS's performance

In this section, the algorithm hyperparameters' impact on the performance is explored. By no means the intent is to optimize the hyperparameters, but instead to shed light on how the hyperparameters affect the dynamics of the algorithm and the Pareto front quality.

Table 6.1: Experiments to evaluate hyperparameter impact.

Hyperparameter	Reference	Range
angle ($\beta_{\max}$)	Eq. 2.19	$[50 - 130]^\circ$
stop criteria	Sec. 2.2.4	[*delta, rank, projection*]
step size (τ)	Eq. 2.16	$[0.005, 0.015, \ldots, 0.035]$
batch size	Sec. 5.1	$[2^7, 2^8, \ldots, 2^{14}]$
model training	Sec. 3.4	$\times 10$ *times*
model size	Table 3.2	[*small, medium*]

To facilitate the interpretation only one hyperparameter is changed per experiment as shown in Table 6.1 and if otherwise specified, the hyperparameters used for the experiments are shown in Table 6.2. The hyperparameters' ranges were chosen to show the impact in the Pareto front obtained, but do not portrait special meaning. Due to the stochastic nature of the method proposed, each of the experiments in Table 6.1 was repeated 10 times with the training data shuffled with a different seed such that statistical tests can be applied to compared experiments.

6.1 Maximum angle experiments

The maximum angle allowed between the desired direction d and the corrector $F(x + tv)$ plays an important role in the number of iterations to find the step size of the corrector and directly impacts the number of function evaluations because of line 8 of Alg. 2. A small $\beta_{\max}$ forces the corrector to point closer to the desired search direction but at the cost of reducing the step size t (line 7 of Alg. 2). In turn, a smaller corrector requires more steps to approximate the Pareto front.

Table 6.2: Continuation method configuration for results.

Parameter	Value	Ref.
Predictor		
Bounds (b_1, b_2)	[1, 1]	Eq. 2.17
Step size (τ)	0.025	Eq. 2.16
Corrector		
Max angle $(\beta_{\max})$	110°	Eq. 2.20
Crit. point criteria	Rank	Eq. 2.22
Crit. point threshold (ϵ)	0.007	Eq. 2.22
Step size max. iter. (N_a)	30	Routine 2
Correctors max. iter. (N_b)	50	Routine 3
General		
Problem size	small	Table 3.2
Batch size	1,024	Section 5.1

6.1.1 Corrector and descent steps varying $\beta_{\max}$

Table 6.3 shows the mean and standard deviation of the corrector and descent number of steps. The descent steps are those required in the initial descent to the Pareto front in line 6 of Alg. 5, whereas the corrector steps are those needed to approximate the Pareto front given the initial point from the previous predictor step.

Table 6.3: Corrector and descent steps in terms of mean and standard deviation in parenthesis when varying $\beta_{\max}$.

$\beta_{\max}$	corrector steps	descent steps
50	17.81 (10.98)	12.0 (10.40)
60	14.21 (10.84)	14.9 (8.78)
70	7.69 (7.71)	5.7 (3.03)
80	4.36 (4.76)	8.9 (8.68)
90	3.63 (4.30)	**4.0** (3.85)
100	3.95 (4.74)	9.7 (8.56)
110	**2.99** (2.86)	8.4 (6.42)
120	3.13 (3.45)	11.6 (7.74)
130	3.30 (3.45)	17.5 (11.32)

Considering that each configuration is executed 10 times, the KruskalWallis H-test is applied to asses if the samples originate from the same distribution. The null hypothesis is that the population median of all of the groups is equal. The test, however, does not identify which samples come from a different distribution. A footnote is added to the box plot to show if the results differ significantly along the configurations.

If the samples come from different distributions, a Wilcoxon rank (WR) test is applied between each pair of experiments to determine if the configuration results are significantly greater/less than another configuration or if no relationship can be established. Table 6.4 shows the Wilcoxon victories computed as follows:

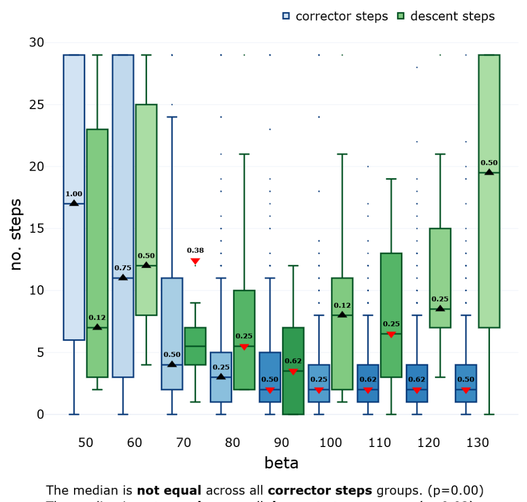
$$M[i,j] = \begin{cases} 1 \text{ if } \mathrm{WR}(R_i, R_j) : \text{p-value} < 0.05 \\ -1 \text{ if } \mathrm{WR}(R_j, R_i) : \text{p-value} < 0.05 \\ 0 \text{ otherwise} \end{cases},$$

where M is the victory matrix, R_i are the results for experiment i, and the Wilcoxon rank considers the alternative that the first input is larger than the second argument. The scope of this section is not to find the best hyperparameter configuration and the victory matrix is only computed to give a sense of how a configuration compares to other results considering statistical significance.

Table 6.4: Wilcoxon rank victories for corrector steps when varying $\beta_{\max}$.

$\beta_{\max}$	50	60	70	80	90	100	110	120	130
50	0	0	0	0	1	0	0	0	0
60	0	0	1	1	1	0	1	0	0
70	0	-1	0	0	0	0	0	-1	-1
80	0	-1	0	0	0	0	0	0	-1
90	-1	-1	0	0	0	-1	0	-1	-1
100	0	0	0	0	1	0	0	0	0
110	0	-1	0	0	0	0	0	0	-1
120	0	0	1	0	1	0	0	0	0
130	0	0	1	1	1	0	1	0	0

Fig. 6-1 shows the box plot for the descent and corrector number of steps as $\beta_{\max}$ varies. The score above the triangles is the sum of victories per configuration, divided by the number of configurations minus one. A black and red triangle indicates a positive or negative aggregate of victories respectively. Scores could be misleading because positive victories cancel out defeats, nevertheless, it shows the total *least* amount of other configurations against which it is greater/less.



The median is **not equal** across all **corrector steps** groups. (p=0.00)
The median is **not equal** across all **descent steps** groups. (p=0.02)

Figure 6-1: Box plot of predictor and corrector steps when varying $\beta_{\max}$.

As shown in Fig. 6-1 and Table 6.3, the configuration $\beta_{\max} = 90°$ requires the least amount of initial descent steps, whereas the configurations $\beta_{\max} = \{110°, 120°\}$ have the best Wilcoxon scores for the corrector steps. For $\beta_{\max} \geq 110°$ it is not completely clear that the threshold relaxation improves the performance since correctors pointing far from the desired direction are allowed. It is worth noticing that the number of corrector steps has a greater impact on performance than the number of initial descent steps.

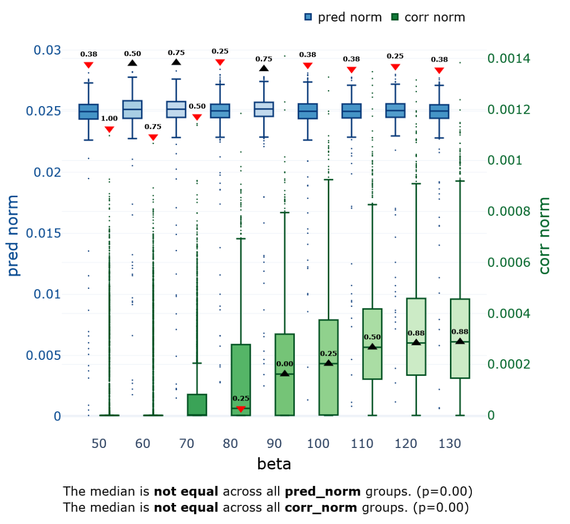
In future analysis, the Wilcoxon Rank victories matrix (Table 6.4) and the means and standard deviations (Table 6.3) will not be shown always, since the most relevant information is shown in the box plot.

6.1.2 Predictor and corrector norms varying $\beta_{\max}$

Fig. 6-2 shows the box plot of the norms ($||\cdot||$) for the predictor (in blue) and the correctors (in green). The corrector norms show a positive correlation with $\beta_{\max}$. A smaller $\beta_{\max}$ reduces the number of iterations of Alg. 3 and therefore the step size for the corrector (line 6 and 7 of Alg. 3). Meanwhile, the predictor norms shows slight changes that are caused only by the initial position of the predictor from the previous corrector step.

6.1.3 Train and validation hypervolume varying $\beta_{\max}$

Fig. 6-2 shows the box plot of the hypervolume (HV) for the training dataset (in blue) and validation dataset (in green). Given its easy interpretation and good properties, the hypervolume indicator is one of the most used quality indicators for the performance evaluation of stochastic multiobjective optimizers. The hypervolume uses a reference point $r \in \mathbb{R}^k$ in objective space to evaluate a solution set by simultaneously taking into account the proximity of the points to the Pareto front, diversity, and spread. For a thorough survey and more information on this indicator, refer to Guerreiro et al. (2021) and Li and Yao (2019), respectively.



The median is **not equal** across all **pred_norm** groups. (p=0.00)
The median is **not equal** across all **corr_norm** groups. (p=0.00)

Figure 6-2: Box plot of predictor and corrector norms when varying $\beta_{\max}$.

Throughout this work, the reference point used to compute the hypervolume is $r = (2, 2)$. Fig. 6-3 shows that, as expected, the hypervolume for the training dataset is better than the hypervolume for the validation dataset. Additionally, the hypervolume for the training dataset presents statistically significant improvement for $\beta_{\max} \geq 100°$, whereas for the validation dataset, $\beta_{\max} \geq 80°$ derives the best performance. For $\beta_{\max}$ values less than 100° the solution is degraded due to the tighter bounds on the corrector direction. This is especially relevant, given that the desired direction α is noisy since it is computed with a batch of the dataset (line 6 of Alg. 4).

6.1.4 General evaluation varying $\beta_{\max}$

The general evaluation framework will be defined in this section. When assessing algorithms, two main evaluation categories are of relevance: (A) solution performance and (B) execution time. The hypervolume with the validation dataset is used as the ultimate performance metric since it is unbiased. To asses execution time, the simplest method is to measure CPU time. However, the CPU is not always precise and behaves stochastically. For that reason, the function and Jacobian evaluations denoted by $E_{F(x)}$ and $E_{J(x)}$ respectively, are counted. The function evaluations are performed by the predictor $E_{P,F(x)}$ and the corrector $E_{C,F(x)}$, whereas the Jacobian evaluation is performed only by the corrector routine (line 3 Alg. 3). In most cases, Jacobians are more computationally intensive than function evaluations. Therefore, the factor ρ_t is proposed as a proxy factor to consolidate Jacobian evaluations into function evaluations. The factor is computed using the mean CPU time needed to compute the function ($t_{F(x)}$) divided by the mean CPU time required to calculate the Jacobian ($t_{J(x)}$). The weighted function evaluations count is defined as:

$$E_w = E_{P,F(x)} + E_{C,F(x)} + E_{J(x)} \times \rho_t \, . \tag{6.1}$$

Table 6.5 shows the count and execution times for the function and Jacobian evaluations. Additionally, the complete continuation method execution time t_{CPU} is shown in the first column.

Since Table 6.5 can be hard to read, Fig. 6-4 shows the scatter plot of the results using coordinates:

$$(x, y) : (\text{Norm. } E_w, \text{Norm. validation HV}) \, .$$

To compare results among all experiments, the performance metrics are normalized as follows: the weighted evaluations E_w are normalized using a maximum of 3,900 and the hypervolume (HV) uses a min-max normalization with a minimum of 3.35 and a maximum of 3.9. Additionally, a 95% covariance confidence ellipse is plotted to show the distribution of the solutions per configuration.

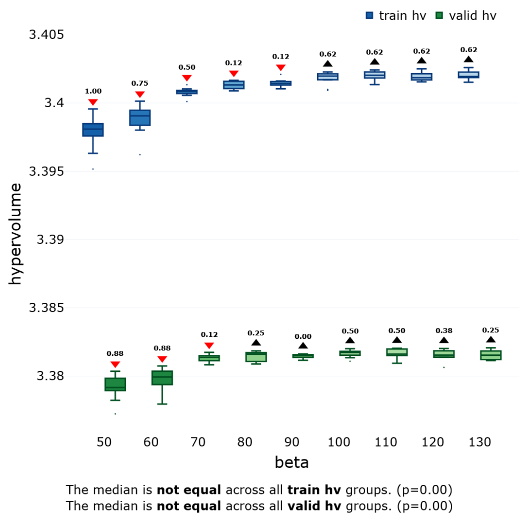


Figure 6-3: Box plot of hypervolume for training and validation datasets when varying $\beta_{\max}$.

Fig. 6-4 shows a visible degrading in function evaluations for $\beta_{\max} \leq 80°$. For $\beta_{\max} \geq 100°$, there are slight improvements in the function evaluations and no improvement in the HV (see also Fig. 6-3). The hyperparameter $\boldsymbol{\beta}_{\max} = \mathbf{100°}$ is chosen for future experiments since it is the smallest $\beta_{\max}$ from which no improvement in the hypervolume is observed. The complete results are shown in Appendix Table A.1.

Table 6.5: Count and execution times for the function and Jacobian evaluations in terms of mean and standard deviation when varying $\beta_{\max}$.

$\beta_{\max}$	t_{CPU} (s)	$t_{F(x)}$ (ms)	$t_{J(x)}$ (ms)	ρ_t	$E_{P,F(x)}$	$E_{C,F(x)}$	$E_{J(x)}$	E_w
50	149.8 (31.4)	50.5 (9.7)	348.4 (45.9)	6.91	75.6 (0.5)	2965.6 (311.5)	83.1 (13.2)	3615.32
60	114.2 (16.0)	45.6 (3.3)	322.8 (9.5)	7.07	75.3 (0.6)	2528.8 (250.2)	66.9 (9.6)	3077.61
70	73.4 (15.0)	45.9 (3.4)	330.3 (31.4)	7.19	75.2 (0.6)	1551.0 (305.5)	37.8 (6.5)	1897.88
80	39.2 (8.3)	**43.9** (1.1)	**307.6** (3.7)	7.00	75.2 (0.4)	731.2 (222.8)	22.9 (4.6)	966.80
90	28.6 (5.8)	44.2 (1.7)	310.9 (7.0)	7.03	**74.8** (0.4)	497.0 (151.5)	20.2 (2.8)	713.75
100	27.7 (7.0)	46.4 (5.6)	354.2 (61.0)	7.63	74.9 (0.3)	368.8 (169.5)	20.8 (4.7)	602.28
110	24.9 (3.6)	56.9 (17.1)	392.2 (93.3)	**6.89**	75.1 (0.3)	218.2 (48.5)	**17.7** (1.9)	415.02
120	**23.2** (2.8)	50.3 (3.0)	354.1 (37.3)	7.04	75.1 (0.3)	190.2 (30.5)	18.1 (2.3)	**392.92**
130	26.0 (3.4)	54.2 (3.3)	388.3 (28.2)	7.16	75.4 (0.8)	**181.4** (49.5)	19.1 (2.2)	393.35

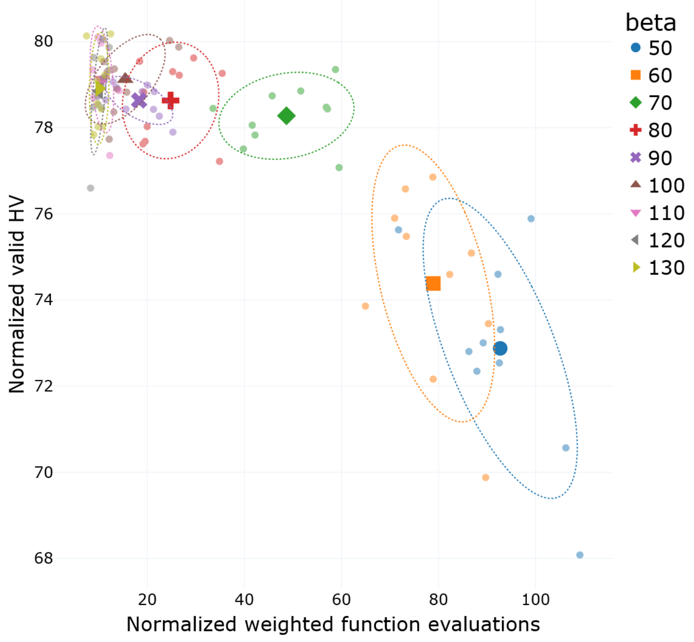


Figure 6-4: Scatter plot of performance metrics when varying $\beta_{\max}$.

6.2 Stopping criteria experiments

According to Table 6.1, the next experiment concerns the stopping corrector criteria explained in Section 2.2.4. The stopping criteria (SC) determines if $x \in \mathcal{R}^n$ is a critical point and halts the corrector step. In practice, these criteria use thresholds to find *almost* critical points. As the results will show, stopping criteria are a key element in the algorithm because it impacts directly in the quality of the solution and the performance of the method. If relaxed criteria are used, the corrector step will halt prematurely, resulting in a degraded Pareto front. On the contrary, strict criteria will overextend the correctors' step, consuming additional computational resources.

6.2.1 Threshold selection for stopping criteria

The *delta* criterion is explained in Section 2.2.4 and consists of a single objective optimization where a small threshold ϵ makes a stricter condition (Eq. 2.21). The second stopping criterion is the *rank* criterion which computes the $\epsilon-$rank of the Jacobian $J(x)$ to determine if the x is a critical point. Likewise, a smaller threshold ϵ accounts for a stricter condition (see Section 2.2.4 and Eq. 2.22). Finally, the *projection* criterion uses the norm of the product between $J(x)$ and the search direction d to determine if x is a critical point. As in the previous two cases, larger ϵ relaxes the stopping condition (see Eq. 2.23).

For the experiments, five thresholds ϵ's were selected for each stopping criterion and since the thresholds ϵ have different meanings, their ranges differ per experiment. The threshold values were carefully selected to start with strict conditions and then cause similar degradation as the threshold increases. Table 6.6 shows the selected threshold values.

Table 6.6: Thresholds ϵ for stopping criteria.

Criteria	*delta*	*rank*	*projection*
strict	5e-05	0.005	0.0003
	0.0001	0.007	0.0006
	0.0002	0.009	0.0009
	0.0005	0.016	0.0012
relaxed	0.0010	0.020	0.0015

6.2.2 Execution time varying stopping criteria

Table 6.7 shows experiment results by varying the stopping criteria (SC) and thresholds ϵ. The experiment nomenclature is **<first letter of criteria>: <threshold value>**, e.g. "d: 0.0001" corresponds to the *delta* criteria with $\epsilon = 0.0001$.

From most experiments, the number of function evaluations is the ideal way to determine the method performance. However, by varying the stopping criteria, the CPU time might not be explained entirely by the function executions. Fig. 6-5 shows the scatter plot between the CPU time and the number of function evaluations (first and second columns of Table 6.7 respectively). The CPU time has a linear relationship with the function evaluations; however, the slope changes for each of the stopping criteria. This finding shows that the single-objective optimization for the *delta* criterion is more than twice as expensive as the *rank* and *projection* criteria. Similarly, the SVD decomposition used to compute the rank of the Jacobian matrix in the *rank* criteria is 25% more expensive than the *projection* criterion.

6.2.3 General evaluation varying the stopping criteria

Fig. 6-6 shows the box plots for the hypervolume along the different experiments varying the stopping criteria and thresholds. The smallest threshold in the *delta* and *rank* criteria yield the best hypervolume. However, no statistical relationship between the two best results can be obtained

Table 6.7: General overview in terms of mean and standard deviation when varying the stopping criteria.

SC	t_{CPU} (s)	E_w	Validation HV
d: 5e-05	125.1 (14.3)	1156.99	**<u>3.383</u>** (2.08e-04)
d: 0.0001	60.3 (7.1)	595.91	3.383 (3.10e-04)
d: 0.0002	35.6 (1.9)	351.84	3.382 (3.98e-04)
d: 0.0005	28.7 (0.4)	304.53	3.378 (3.83e-04)
d: 0.001	25.9 (0.3)	**<u>285.02</u>**	3.373 (5.47e-04)
r: 0.005	72.2 (17.5)	1440.57	3.383 (2.41e-04)
r: 0.007	29.0 (3.5)	555.42	3.383 (3.89e-04)
r: 0.009	21.5 (2.0)	452.91	3.382 (2.38e-04)
r: 0.016	14.1 (0.2)	313.02	3.379 (3.22e-04)
r: 0.020	13.3 (0.1)	303.36	3.378 (3.35e-04)
p: 0.0003	60.6 (13.7)	1412.02	3.382 (6.18e-04)
p: 0.0006	34.7 (11.5)	818.06	3.379 (1.29e-03)
p: 0.0009	17.2 (3.8)	417.68	3.376 (1.73e-03)
p: 0.0012	12.4 (1.0)	292.18	3.369 (1.29e-03)
p: 0.0015	**<u>12.1</u>** (1.7)	297.95	3.364 (2.48e-03)

using the Wilcoxon rank test. Contrastingly, the results for the *projection* criterion are lower, as shown by the box plot and the Wilcoxon score.

Additionally, the Pareto fronts are shown in Fig. 6-7 (for clarity, only the results for the *rank* criteria are shown). As explained in Fig. 3-1, the left subfigure shows the Pareto front, while the right subfigure consists of the total loss function compared to one of the objectives. Although there is almost no difference in the Pareto front, the total loss degrades visibly when the threshold increases.

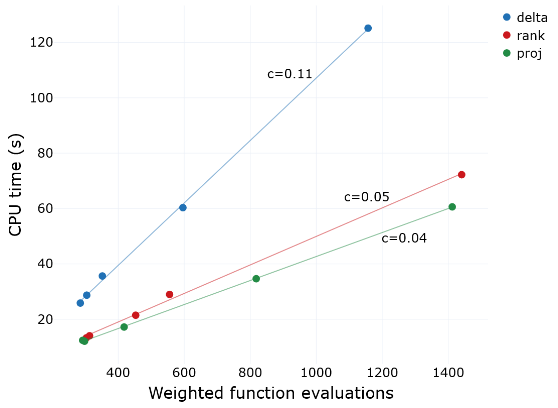


Figure 6-5: Scatter plot of function evaluations vs CPU time when varying the stop criteria.

Finally, Fig. 6-8 shows the general performance overview and can be interpreted as a Pareto front that considers the contrary metrics: solution performance (HV indicator) and algorithm execution time (evaluation counts). The color represents the stopping criteria, whereas the opacity indicates the threshold magnitude. Almost all sets of solutions for the *projection* criterion are dominated by the other two criteria. Consistently with Fig. 6-6, between the *rank* and *delta* solutions, there seems to be no clear winner. However, considering the computational cost of the single-objective optimization for the *delta* criteria, the *rank* criteria seems the more efficient option (see Fig. 6-5). The hyperparameter *rank* criteria with $\boldsymbol{\epsilon = 0.007}$ is chosen for future experiments since the degradation in solution quality is small and also the average function evaluations are less than half than the stricter threshold, i.e. $\epsilon = 0.0005$. The complete numeric results are shown in Appendix Tables A.2 and A.3.

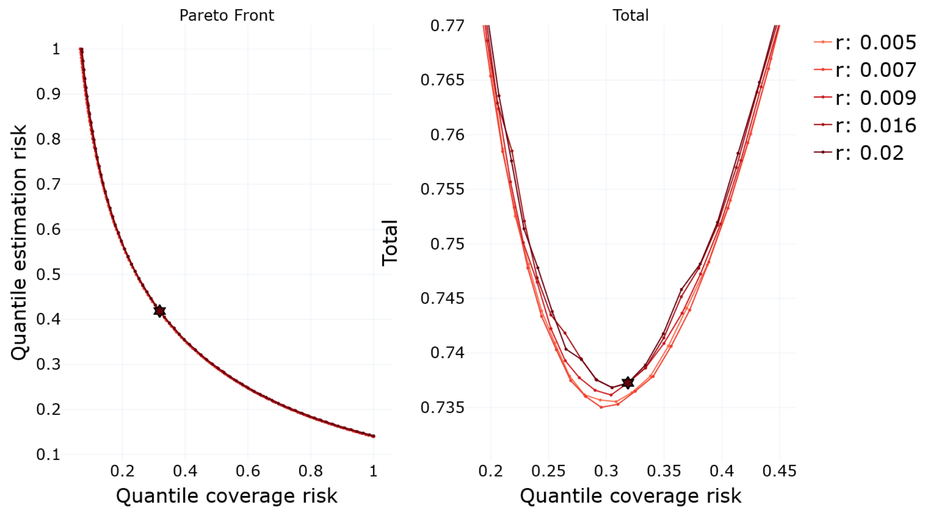


Figure 6-7: Pareto front for *rank*-criterion and different ϵ thresholds.

6.3 Step size parameter experiments

This section explores the impact of the step size parameter τ on the method's performance. For simplicity, this section mentions τ as the step size, although according to Eq. 2.16 the step size t is a function of τ. Not only the predictor size (norm) is a function of τ (see lines 6 and 9 of Alg. 1), but also the corrector size is influenced by the initial step size t_0 computed from the predictor (see lines 7 and 9 of Alg. 4).

The range for the step size parameter experiments was selected to show a visible impact on the algorithm's performance, i.e. $\tau \in [0.005, 0.015, 0.025, 0.035]$.

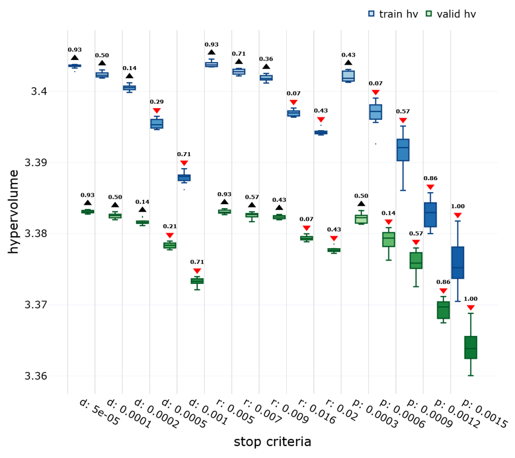


Figure 6-6: Box plot of hypervolume when varying the stop criteria.

As expected and shown in Fig. 6-9, the predictor and corrector norms increase as the step size parameter τ increases. However, the variations are more clear for the predictor than for the corrector. This behavior occurs because the predictor size is a direct function of the step size, whereas the corrector size is initially defined by the step size t_0 but is afterward adjusted with other criteria.

Fig. 6-10 shows the general evaluation when varying the step size parameter. The same Pareto dominance behavior is exhibited when varying τ, since a smaller value improves the quality of the solution and at the same time requires more function evaluations. The improvement in hypervolume when varying the step size is small compared to the function evaluations required and unless there is a need for a high-density Pareto front, larger step size values seem more efficient.

The hyperparameter $\boldsymbol{\tau = 0.025}$ is chosen for future experiments since the degradation in hyper-

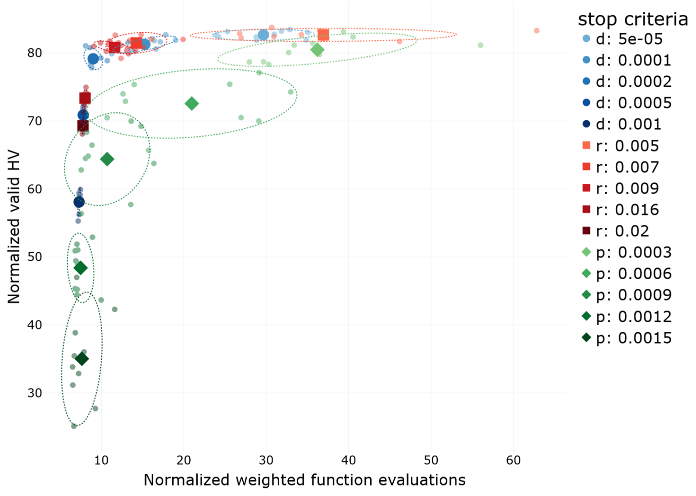


Figure 6-8: Scatter plot of performance metrics when varying the stop criteria.

volume is small and the saving in function evaluations is large. The complete numeric results are shown in Appendix Tables A.4 and A.5.

6.4 Batch size experiments

This section explores the impact of the batch size (BS) on the algorithm's performance. The batch size is used to define the data batches when initializing the *noisy* Jacobian structure shown in Alg. 6. Since GPUs are used to accelerate matrix operations, trying to compute the Jacobian of large matrices can lead to GPU out-of-memory errors. To the best of our knowledge, this work is the first one to introduce Jacobians computed from a subset of data in a Pareto continuation method and one of its objectives is to determine how much the performance of the solution degrades by lowering the batch size. The Jacobian evaluations $E_{J(x)}$ are counted such that the computation of one batch counts as $1/|B|$ evaluations, where $B = \{b_0, b_1, \ldots, b_m\}$ are the dataset batches.

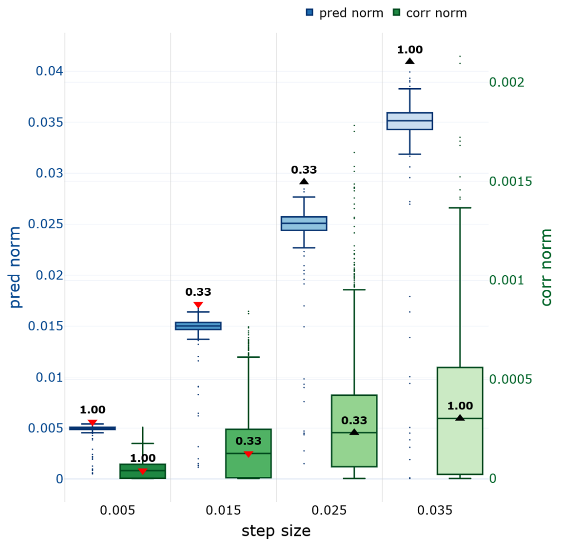


Figure 6-9: Box plot of predictor and corrector norms when varying the step size parameter.

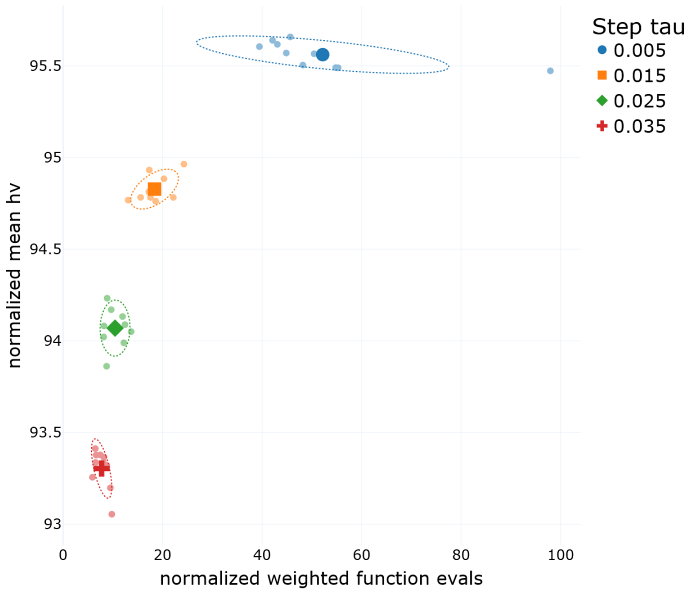


Figure 6-10: Scatter plot of performance metrics when varying the step size parameter.

6.4.1 Execution time varying batch size

Table 6.8 shows the execution time, the weighted function evaluations, and the hypervolume obtained when varying the batch size from 128 to 16,384. The total train dataset consists of 25,522 records, which means a batch size of 16,384 is roughly 64% of the whole dataset. From the table, it is evident that the batch size is inversely related to execution time and function evaluations. This relationship occurs because smaller batch sizes account for noisy Jacobians, which in turn increases the number of corrector steps needed to steer the solutions toward the Pareto front.

Table 6.8: General overview in terms of mean and standard deviation when varying the batch size.

BS	t_{CPU} (s)	E_w	Validation HV
128	401.6 (99.2)	1102.73	3.380 (9.68e-04)
256	200.1 (38.0)	917.61	3.381 (6.82e-04)
512	74.6 (9.1)	580.33	3.382 (2.82e-04)
1,024	25.2 (3.2)	399.63	3.382 (3.06e-04)
2,048	12.6 (1.6)	**337.59**	**3.382** (1.81e-04)
4,096	8.4 (0.8)	363.56	3.382 (2.74e-04)
8,192	7.0 (0.5)	368.80	3.381 (4.66e-04)
16,384	**5.4** (0.3)	363.53	3.382 (1.82e-04)

6.4.2 Corrector steps varying batch size

From Table 6.8, it is visible that for batch sizes greater than 1,024, the function evaluations E_w tend to stabilize at ~360. This behavior is explained because the number of corrector steps depends on the batch size used to compute the Jacobian. Fig. 6-11 shows how the number of corrector steps needed decreases drastically before the batch size of 1,024, from then on, it stabilizes around 3. Nonetheless, the Wilcoxon rank still shows that the reduction in the number of steps is significant as the batch size increases. The number of steps for the initial descent (shown in green) does not show an evident pattern since the total of descent steps is low.

6.4.3 Norms and distance varying batch size

An additional metric introduced in this section is the euclidean distance between the solutions that compose the Pareto front. Fig. 6-12 shows the distance distribution for the Pareto front for the train and validation datasets and although the Wilcoxon score shows that the smallest two batch sizes are significantly larger, the rest of the experiments exhibit similar means. However, the standard deviation decreases as the batch size increases. The cause behind this is that the computation of search directions v depends on the Jacobian at point x (see Eq. 2.11) and small batch sizes translate into noisy search directions and variance in the predictors' norms.

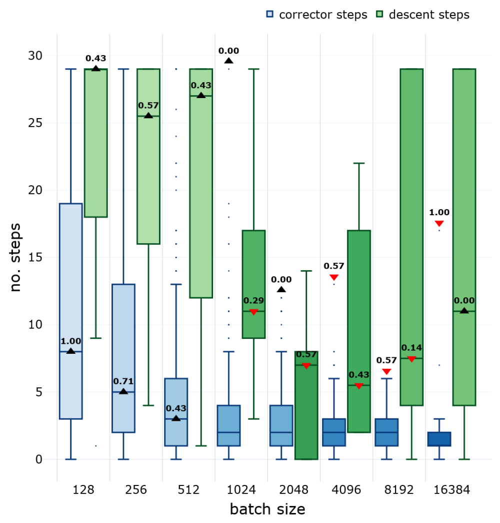


Figure 6-11: Box plot of corrector and descent steps when varying the batch size.

6.4.4 General evaluation varying batch size

Fig. 6-13 shows the general evaluation of the algorithm when varying the batch size. Note that the normalized mean HV presents very small changes, i.e. from 93% to 94.2%. The range in function evaluations is larger than the hypervolume's range, but still small compared to other experiments such as the stopping criteria and the maximum angle $\beta_{\max}$ (Figs. 6-8 and 6-4 respectively). The results show that batch sizes larger than 512 do not degrade significantly the solution quality and only the function evaluations and execution time are impacted.

The hyperparameter $\mathbf{BS} = \mathbf{1,024}$ is chosen for future experiments since the degradation in hypervolume is small, the number of function evaluations is close to the E_w for larger batch sizes and the risk of GPU's memory saturation is small. The complete numeric results are shown in Appendix Tables A.6 and A.7.

6.5 Different training experiment

This section explores the method's performance when the initial solution for the directed stochastic search method varies. For this experiment, the NN model was trained 10 times with the same hyperparameters and dataset, but given the stochastic nature of the training, the initial solution obtained $x \in \mathbf{R}^n := W_s \subset W_{\text{trained}}$ is different for every experiment (see Eq. 3.3). In practice and

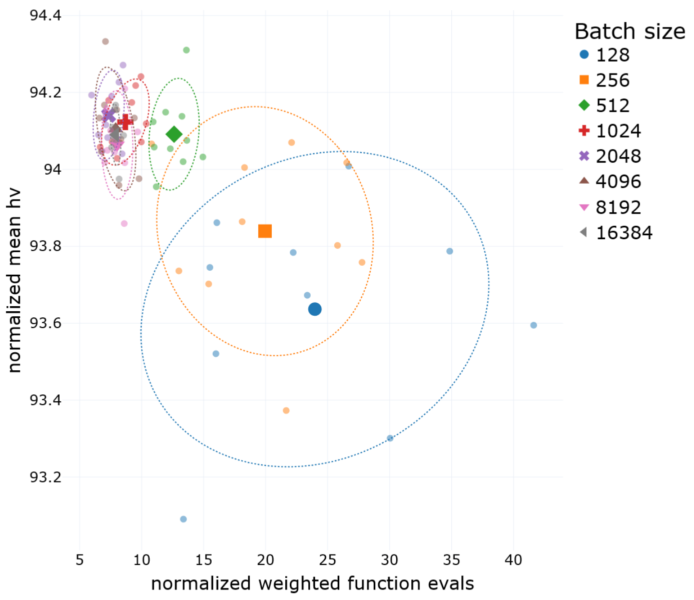


Figure 6-13: Scatter plot of performance metrics when varying the batch size.

to guarantee replicability, a different random seed was selected for each NN training.

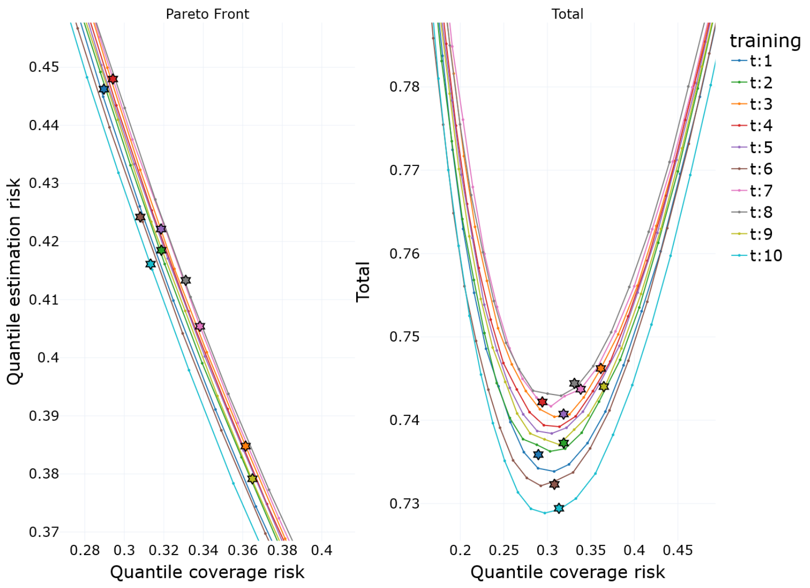


Figure 6-14: Pareto fronts when varying the NN training.

Fig. 6-14 shows a zoomed area of the Pareto fronts for the experiments and their respective initial points. It is evident the Pareto fronts obtained greatly depend on the initial point shown with a star. This behavior is confirmed by the general evaluation shown in Fig. 6-15, where the different solutions for the same training, e.g. t:1, exhibit variance for the function evaluations, but very small variance for the normalized HV. In conclusion, the NN training (i.e., initial point) completely defines the solutions' HV range. Note that the range in the function evaluations is small compared to other experiments.

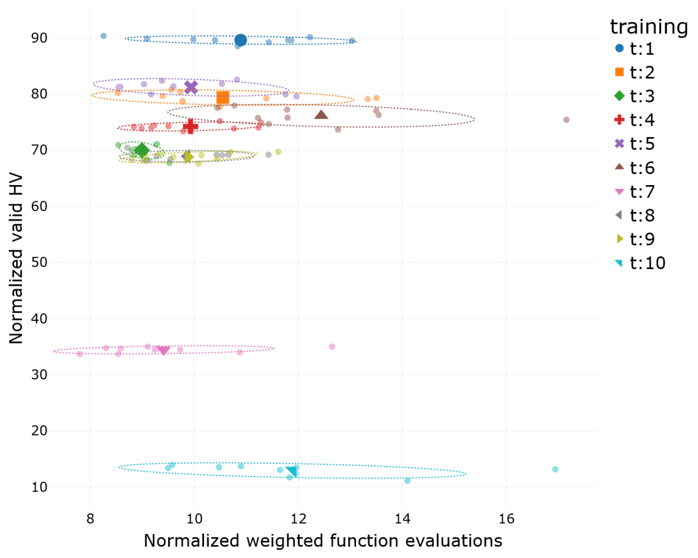


Figure 6-15: Scatter plot of performance metrics when varying the NN training.

The conclusion from this experiment is two-fold. First, the Pareto front obtained with the stochastic continuation method is heavily influenced by the initial solution, which in turn depends on the NN model's training. Secondly, the continuation method is not able to find a global Pareto front, but instead, it obtains local Pareto fronts, which means that the Pareto solutions are local optimum points. The complete numeric results are shown in Appendix Tables A.8 and A.9.

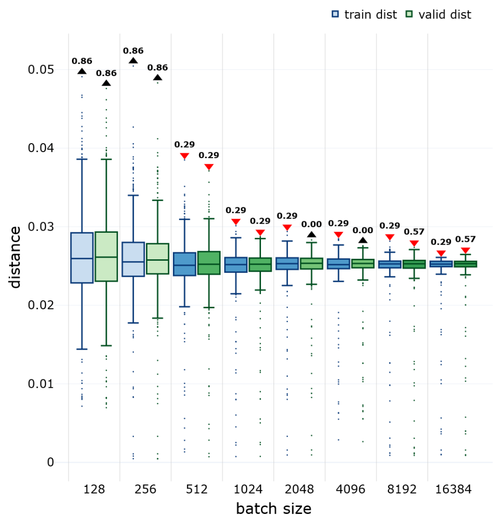


Figure 6-12: Box plot of corrector and predictor distances when varying the batch size.

6.6 Different problem sizes.

In this section, the impact of the multi-objective problem size is explored. The BOP size is dependent on the subset of layers unfrozen. In previous experiments, 387 weights were optimized to find the Pareto front; however, for the medium-size problem more than 240,000 weights are modified (see Table 3.2). Note that the same NN model training is used for both configurations.

As shown in Table 6.9, the execution time increases by a factor of ~ 28, while the function evaluations stay roughly the same. For the hypervolume, a very small degradation is observed for the medium problem size, i.e. 0.0295 %. The Wilcoxon score in Fig. 6-16 shows that no relationship (larger/smaller) can be established for the results obtained in the training dataset; however, for the validation dataset, the solution for the small problem size is statistically significantly better than the medium problem size.

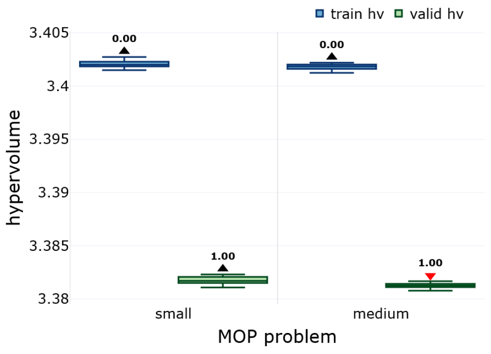


Figure 6-16: Box plot of hypervolume when varying the problem size.

Fig. 6-17 shows the normalized hypervolume vs the normalized function evaluations for the two problem sizes. Compared to other experiments, very small variance is exhibited for both variables.

Table 6.9: General overview in terms of mean and standard deviation when varying the problem size.

Size	t_{CPU} (s)	E_w	Validation HV
small	**<u>20.3</u>** (3.8)	**<u>396.60</u>**	**<u>3.382</u>** (3.46e-04)
medium	559.3 (48.5)	441.36	3.381 (2.45e-04)

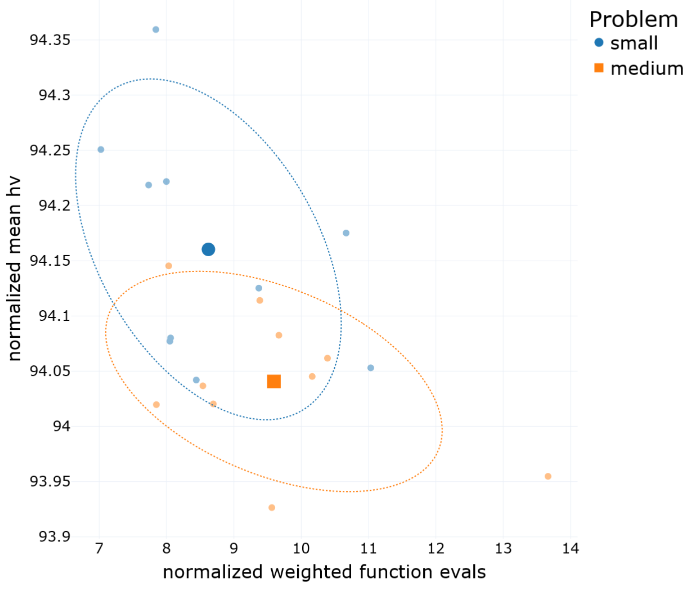


Figure 6-17: Scatter plot of performance metrics when varying the problem size.

In conclusion, correlation exists between the problem size and solution quality. However, the proportion of degradation in the solution is small compared to the increment in weights. The results demonstrate that the method is capable of finding high-quality Pareto fronts even when using large amounts of weights and that only a subset of weights are enough to find the complete Pareto front. The complete numeric results are shown in Appendix Tables A.10 and A.11.

Chapter 7

SDS Results

This section shows the results obtained from the multi-objective optimization of the neural network and using the hyperparameters chosen in the previous section. As explained in the introduction, the neural network used is the Temporal Fusion Transformer and it generates multi-horizon quantile forecasts (see Lim et al. (2021)). The objectives from the MOP are the *quantile coverage risk* (QCR) which quantifies if the forecast is outside the quantile range and *quantile estimation risk* (QER) which measures the meaningfulness of the quantile forecast, and increases if the quantile range is too broad. If more meaningful forecasts are required, the decision maker has to decrease QER at the cost of increasing the QCR, and vice versa.

The hyperparameters used are shown in Table 6.2 except for the step size parameter, where a smaller parameter $\tau = 5 \times 10^{-3}$ was chosen to obtain a higher-density front. Additionally, a maximum degradation of 5% of the total loss function was imposed (see Section 5.2), since normally the decision-makers are only interested in regions close to the local minimum found in the SGD training.

As explained in Section 3.5, the function and Jacobian evaluations used in the stochastic directed search method are computed with the train data subset. Once the solution set is obtained (where each solution consists of the NN weights), the solution set is evaluated with the validation data subset. Consequently, two Pareto fronts are obtained and Table 7.1 shows their execution time, hypervolume, the euclidean distance between solutions, and their weighted function evaluations E_w. It took only 19.87 s [1] to find the complete Pareto front defined by the boundary restrictions. As shown in Table 7.1, the hypervolume is larger for the train data subset, and it is caused by overfitting. Finally, the function evaluations for the validation subset correspond to the number of solutions in the Pareto front (each solution is evaluated once).

Table 7.1: Results with maximum total loss of 5%.

data subset	t_{CPU} (s)	HV	distance	E_w
train	29.87	3.0139	0.005 (6e-4)	410.76
valid	-	2.9976	0.005 (6e-4)	78

[1] Since the MOP consists of the weights of a neural network, the GPU card is used to enhance the function and Jacobian computation performance. This CPU time was obtained with a GPU Nvidia GeForce RTX 3070 8Gb and CPU Intel Core i7-11800H CPU @ 4.6 GHz.

The left subfigure of Fig. 7-1 shows the Pareto front obtained, whereas the right subfigure shows the total loss $\mathcal{L}(\Omega, W)$ in the y-axis and the first loss function in the x-axis. Consequently, each point in the right subfigure is defined by:

$$(x, y) := (\text{QCR}, \text{QER} + \text{QCR}) = (\text{QCR}, \mathcal{L}) \ .$$

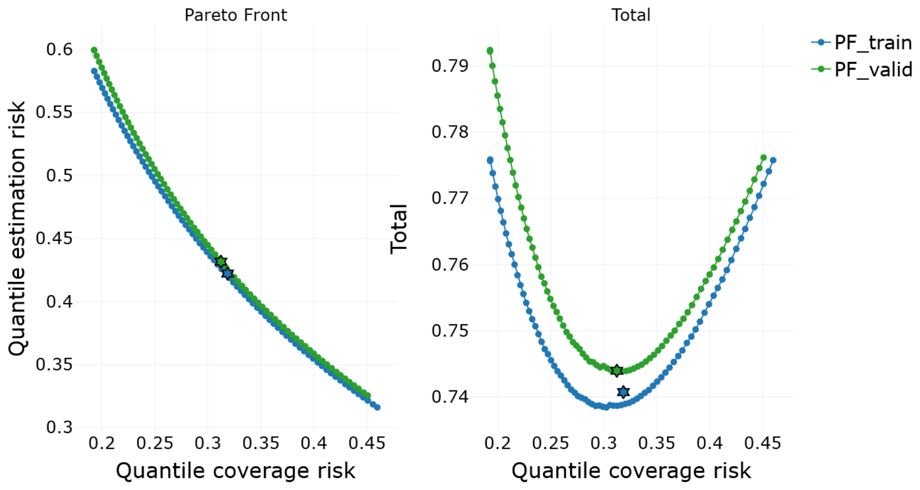


Figure 7-1: Pareto front and total loss for train and test data subsets.

As expected, the Pareto front for the validation subset is a *degraded* version of the Pareto front using the training subset. Additionally, it is evident that the initial solution (shown with a blue star) for the training subset does not lie in its Pareto front, which means that a series of descent steps were needed to first approximate the Pareto front. On the contrary, the initial solution for the validation subset lies in its Pareto front. This behavior can be explained by the early stopping condition used when training the NN. If during 10 epochs no improvement in the validation loss function was observed, the training was stopped to avoid overfitting. The stochastic directed search method was able to improve the initial solution with the training subset, but that did not translate into better performance with the validation subset, which only confirms the fact that the SGD training was effective in finding a local minimum with the validation subset.

From the left subfigure of Fig. 7-1 it can be observed that for the validation subset the QCR range is approximately [0.2, 0.45] and [0.3, 0.6] for the QER. This implies that an increment of only 5% in the total loss can account for variations of $\pm \sim 30\%$ and $\pm \sim 40\%$ for the QCR and QER objectives, respectively. The exact percentages may vary depending on the NN problem, the NN training, and the hyperparameters used; however, the takeaway is that a small increment in the total loss can allow the decision maker to fine-tune the NN using a large tolerance window.

7.1 Comparison with NSGA-II

In this section, the Stochastic Directed Search (SDS) method is compared with the NSGA-II. The comparison is not completely fair, since the SDS is a local search method that uses the information of the gradient, whereas the NSGA-II is a bio-inspired evolutionary algorithm that finds the global Pareto front. Nevertheless, NSGA-II is commonly used in multi-objective optimization and it obtained the best performance when optimizing the same TFT architecture in the work of López et al. (2022). Because of the stochastic nature of both methods, a series of 10 experiments were executed and the results are shown in Table 7.2. For the SDS method, the hyperparameters of Table 6.2 were used, with a step size parameter $\tau = 5\times10^{-3}$ and a maximum degradation of 5% of the total loss function. With the intention of a fair comparison, the NSGA-II search is limited in objective space by the square with its two diagonally opposite points being [(0,0), (0.459, .583)], where (0.459, .583) is the nadir point of the union of the Pareto fronts for the 10 executions of the SDS: $\text{nadir}(\bigcup_{i=1}^{10} \mathcal{PF}_{i,\text{SDS}})$. Additionally, the NSGA-II population size equals the average number of solutions in the Pareto front obtained with the SDS method: $\sum_{i=1}^{10} |\mathcal{PF}_{i,\text{SDS}}|/10 = 78$. Finally, four experiments were performed considering 14, 30, 100, and 200 generations to compare the performance as the generations and function evaluations increase. The 14 generations were chosen to have the same number of function evaluations as the SDS method with a 95% confidence: $675.73[E_w]+178.8[E_w]*1.96 = 1,026.18[E_w]$, which means that $1,026.18[E_w]/78[\text{individuals}] = 13.15$ generations are needed. Regarding the upper bound of generations, a total of 400 generations were chosen to keep the function evaluations under two orders of magnitude from the SDS 675 function evaluation.

Table 7.2: Results for the small problem (Table 3.2) with stochastic directed search (SDS) and NSGA-II methods in terms of mean and standard deviation.

method	t_{CPU} (s)	HV	distance	E_w	generations
SDS	**29.56** (5.05)	**2.9991** (1.46×10^{-3})	4.93×10^{-3} (**1.33×10^{-4}**)	**675.73** (178.8)	-
NSGA-II	55.32 (5.96)	2.9776 (8.98×10^{-3})	6.84×10^{-3} (4.66×10^{-3})	1,092.00 (0.00)	14
NSGA-II	100.58 (2.14)	2.9851 (5.31×10^{-3})	6.03×10^{-3} (4.05×10^{-3})	2,340.00 (0.00)	30
NSGA-II	423.70 (25.08)	2.9890 (7.11×10^{-3})	5.45×10^{-3} (3.56×10^{-3})	7,800.00 (0.00)	100
NSGA-II	766.01 (11.94)	2.9902 (5.68×10^{-3})	5.48×10^{-3} (3.68×10^{-3})	15,600.00 (0.00)	200
NSGA-II	1089.44 (30.27)	2.9907 (6.56×10^{-3})	5.38×10^{-3} (3.56×10^{-3})	23,400.00 (0.00)	300
NSGA-II	1270.37 (64.65)	2.9925 (5.54×10^{-3})	5.36×10^{-3} (3.46×10^{-3})	31,200.00 (0.00)	400

Table 7.3: Results for the medium problem (Table 3.2) with stochastic directed search (SDS) and NSGA-II methods in terms of mean and standard deviation.

method	t_{CPU} (s)	HV	distance	E_w	generations
SDS	**702.90** (64.23)	**3.3825** (3.78×10^{-4})	2.50×10^{-2} (**7.67×10^{-4}**)	**510.36** (85.94)	-
NSGA-II	1,082.65 (123.98)	2.2866 (1.27×10^{-1})	7.91×10^{-2} (1.14×10^{-1})	840.00 (0.00)	14
NSGA-II	1,947.69 (207.23)	2.4823 (1.35×10^{-1})	4.29×10^{-2} (7.43×10^{-2})	1,500.00 (0.00)	25
NSGA-II	3,834.32 (238.92)	2.8391 (1.50×10^{-1})	3.45×10^{-2} (4.07×10^{-2})	3,000.00 (0.00)	50
NSGA-II	5,712.71 (282.95)	3.0939 (5.63×10^{-2})	2.91×10^{-2} (2.22×10^{-2})	4,500.00 (0.00)	75
NSGA-II	7,606.91 (317.35)	3.1689 (3.87×10^{-2})	2.51×10^{-2} (1.54×10^{-2})	6,000.00 (0.00)	100
NSGA-II	9,495.41 (359.73)	3.1888 (2.98×10^{-2})	2.47×10^{-2} (1.68×10^{-2})	7,500.00 (0.00)	125
NSGA-II	11,392.27 (396.44)	3.2107 (3.10×10^{-2})	2.42×10^{-2} (1.44×10^{-2})	9,000.00 (0.00)	150

Table 7.2 shows the execution time, the hypervolume, the distance between the points of the Pareto front, the function evaluations E_w and the number of generations for the NSGA-II. The SDS method exhibits better performance than the NSGA-II no matter the number of generations given to the MOEA. As expected, as the number of generations for the NSGA-II increases, also the hypervolume increase. However, in all cases $n_{\text{gen}} = [30, 100, 200, 300, 400]$, the Wilcoxon ranking test still shows the hypervolume obtained with the SDS is significantly higher with a 95% confidence. It is worth noticing that the standard deviation for the distance between the Pareto front points is one order of magnitude smaller for the SDS method, which indicates that the solutions are more equally distributed compared with the NSGA-II. Overall, the Pareto front obtained with the SDS method exhibits higher quality in terms of the hypervolume, distance standard deviation, and fewer function evaluations, which translates to smaller execution time.

Fig. 7-2 shows the Pareto fronts for the SDS and NSGA-II methods and using 200 generations for the MOEA algorithm. Although the Pareto fronts from the left subfigure look very similar, the right subfigure shows that the NSGA-II solutions are less equally distanced from each other and some solutions are dominated and others are non-dominated with respect to the SDS Pareto front.

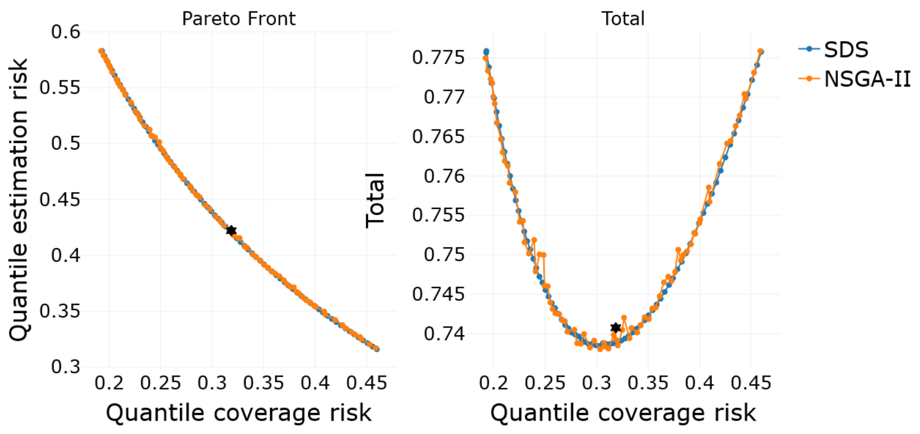


Figure 7-2: Small MOP size Pareto fronts with stochastic directed search (SDS) and NSGA-II methods.

As a final comparison, the medium-size problem was solved with the SDS and NSGA-II methods. Given the high dimensionality in the decision space, the NSGA-II was not able to effectively find the Pareto front with the boundary conditions $f_1(x) \leq 0.45$ and $f_2(x) \leq 0.6$. Therefore, the more relaxed boundary $f_1(x) \leq 1.0$ and $f_2(x) \leq 1.0$ was imposed. Additionally a larger step size $\tau = 2.5 \times 10^{-2}$ was selected for the SDS method. For a fair comparison, the NSGA-II used a population of 60, which was the mean number of solutions in the Pareto front using the SDS method. Given the high execution time, only one experiment was performed for the NSGA-II methods using 150 generations. The additional rows shown in Table 7.3 are snapshots of the execution at the specific generations, i.e., 14, 30, 100, etc. The experiment was executed 10 times using the SDS and NSGA-II methods and the results shown in Table 7.3 show that the SDS is superior in all metrics. With a comparable number of function executions, i.e. NSGA-II at 14 generations, the hypervolume is 32.4% smaller in the NSGA-II, whereas with the maximum of 150 generations, the hypervolume is still 5.07% smaller with the NSGA-II and the number of function evaluations is $\sim$17.63 larger.

To evaluate the impact of the problem size in the NSGA-II performance compared with the SDS method, the results for 100 generations from Tables 7.2 and 7.3 were chosen and compared in Table 7.4 based on the relative change between the SDS and NSGA-II hypervolume and their function evaluations ratio. The comparison shows that for a similar ratio of function evaluations, i.e. ~ 11.6, the relative change in hypervolume with respect to the SDS method is larger for the medium MOP size.

Table 7.4: NSGA-II vs SDS performance with different problem sizes.

MOP size	Table	gen	HV change	E_w ratio
small	7.2	100	-0.33 %	11.54
medium	7.3	100	-6.31 %	11.75

Finally, when optimizing previously trained neural networks, the decision maker is interested in the nearby region from the local minimum found by the SGD training. However, when the number of decision variables is large, finding the Pareto front in the neighboring region of the initial solution is an impossible task for the NSGA-II. The cause behind this is that no steering direction in decision space is available in the NSGA-II and its random crossover and mutation in decision space map to undesired regions in objective space. This justifies the use of a continuation method to find high-quality Pareto fronts and in at the same time computational resources are optimized. The results from Tables 7.2, 7.3, and 7.4 show that the SDS method scales better with the MOP size in comparison with the NGSA-II.

7.2 Hyperparameters *Cheat Sheet*

In this section, an general recommendation concerning the hyperparameters of the SDS is given. Note that this suggestion is based on the results obtained in this work and might not be optimal for other types of problems.

- **Bounds in objective space**. The SDS method allows hard bounds for maximum values in the objective space and also a maximum increase in the total loss function specified as a percentage. A maximum increase in loss function is more intepretable and easy to define without previous knowledge of the Pareto front's shape. Both of these stopping conditions can be applied and it is solely a choice of the decision maker.
- **Step size** The step size parameter (τ) impacts the Pareto front density and should be chosen according to the requirement of solution granularity and the availability of computing resources.
- **Max angle**. The results show that the sweet spot for the maximum angle is $\beta_{\max} \in [90°, 120°]$. Lower values of $\beta_{\max}$ will degrade the Pareto front quality and will require more function evaluations, while higher values allow the corrector to point away from the desired search direction.
- **Critical point criteria**. The results show that the rank stopping criteria yields the best results since it is computationally efficient and guarantees the quality of the solutions obtained.

It is also worth using the delta stopping criteria since its only downside is its computational cost.

- **Critical point threshold**. The optimum threshold (ϵ) range is $\epsilon \in [0.001, 0.010]$, since higher values of ϵ degrade the Pareto front quality and lower values require more function evaluations with no significant improvement in the hypervolume.
- **Batch size**. The sweet spot for the batch size is from 512 to 4,096. Lower batch sizes decrease the hypervolume of the solution while higher batch sizes require more GPU memory and no significant quality improvement is obtained.

Chapter 8

Conclusions and future work

This section summarizes the conclusions derived form this work and some future research paths to improve the proposed stochastic directed search method and its methodology.

8.1 Conclusions

This work shows how a financial forecasting deep learning problem can be expressed as a multi-objective problem to obtain the Pareto front with the proposed *stochastic directed search* method, which leverages the Jacobian information to efficiently find the solution set. The multi-objective framework not only allows the decision maker to fine-tune the model without having to retrain the neural network and thus save computational resources but also gives insight into the shape of the loss function with respect to the objectives.

For this specific task, NSGA-II outperformed NSGA-III in all experiments and by defining an increment threshold of the total quantile risk, an optimum window can be defined for the decision maker. Furthermore, the results show that a small increment in the total risk accounts for a significant range of solutions and justifies further optimizing the deep model after the stochastic gradient descent training. Remarkably, the global Pareto fronts are explored using models trained with different quantiles configurations. The results obtained support the hypothesis that the optimization of the last layer is enough to find solutions across the complete Pareto front, no matter which quantiles were chosen for the model's training.

The SDS method proposed can efficiently find high-quality Pareto fronts using few function evaluations, which in the context of neural networks are expensive. In comparison with NSGA-II, the SDS saves more than 70% of function evaluations and finds a better and more evenly distributed Pareto front. Furthermore, having hard bounds for maximum values in the objective space and a maximum increase in the total loss function specified allows the decision maker to limit the search space without previous knowledge of the Pareto front's shape. Additionally, the decision maker can obtain the desired solution granularity in the Pareto front by changing the get the step size parameter.

This work also evaluated the hyperparameters' effect on the algorithm performance and found their optimum ranges. One example is the corrector maximum angle deviation, where the optimum range was found to be around ninety degrees; where lower values will degrade the Pareto front quality and higher values allow the corrector to point away from the desired search direction. Another finding worth mentioning is that the rank sotpping criteria for the corrector step is the most computationally efficient and simultaneously guarantees the quality of the solutions obtained. Finally, the results show that only small batch sizes decrease the hypervolume of the Pareto front; however, high batch sizes require more GPU memory and no significant quality improvement is obtained.

8.2 Future work

For future work, it would be interesting to address the relationship between the Pareto fronts when using the training and validation data subsets. Early stopping criteria could be implemented in the corrector routine to take into account the improvement with the validation subset. This will avoid the overfitting the corrector step could cause.

This work proposes that the optimization process occurs after the standard training of a neural network using stochastic gradient descent. However, an interesting avenue of research is to couple both process so as to compute *intermediate* Pareto fronts and find a global Pareto front whose solutions do not overfit the training dataset.

Furthermore, it is worth pursuing the creation of heuristics to find optimum values of the SDS method's hyperparameters while taking into account the problem size, solution resolution required, and available computational resources.

Finally, an important next step is to apply this methodology to a broader set of deep neural network problems that contain multiple objectives, e.g. multi-task learning neural networks.

Appendix A

Tables

Table A.1: Steps count, norm, and hypervolume results in terms of mean and standard deviation when varying $\beta_{\max}$.

$\beta_{\max}$	S_{descent}	$S_{\text{corrector}}$	||predictor||	||corrector||	train HV	valid HV
50	12.0 (10.4)	17.8 (11.0)	2.45e-02 (3.20e-03)	3.58e-05 (1.18e-04)	3.398 (1.19e-03)	3.379 (8.81e-04)
60	14.9 (8.8)	14.2 (10.8)	2.47e-02 (2.92e-03)	4.68e-05 (1.33e-04)	3.399 (1.09e-03)	3.380 (8.13e-04)
70	5.7 (3.0)	7.7 (7.7)	2.47e-02 (2.95e-03)	8.21e-05 (1.58e-04)	3.401 (3.04e-04)	3.381 (2.55e-04)
80	8.9 (8.7)	4.4 (4.8)	2.47e-02 (2.80e-03)	1.57e-04 (1.98e-04)	3.401 (2.66e-04)	3.381 (3.42e-04)
90	4.0 (3.8)	3.6 (4.3)	2.48e-02 (2.80e-03)	1.98e-04 (2.05e-04)	3.401 (2.78e-04)	3.381 (1.41e-04)
100	9.7 (8.6)	4.0 (4.7)	2.47e-02 (2.35e-03)	2.42e-04 (2.26e-04)	3.402 (4.49e-04)	3.382 (2.63e-04)
110	8.4 (6.4)	3.0 (2.9)	2.46e-02 (2.78e-03)	3.00e-04 (2.22e-04)	3.402 (3.04e-04)	3.382 (3.13e-04)
120	11.6 (7.7)	3.1 (3.4)	2.46e-02 (2.80e-03)	3.28e-04 (2.29e-04)	3.402 (2.76e-04)	3.382 (3.73e-04)
130	17.5 (11.3)	3.3 (3.5)	2.46e-02 (2.62e-03)	3.29e-04 (2.42e-04)	3.402 (3.37e-04)	3.382 (3.29e-04)

Table A.2: Time execution and evaluations count in terms of mean and standard deviation when varying the stopping criteria.

SC	t_{CPU} (s)	$t_{F(x)}$ (ms)	$t_{J(x)}$ (ms)	ρ_t	$E_{P,F(x)}$	$E_{C,F(x)}$	$E_{J(x)}$	E_w
d: 5e-05	125.1 (14.3)	49.9 (7.4)	337.0 (75.0)	6.76	87.4 (0.8)	764.2 (119.9)	45.2 (6.8)	1156.99
d: 0.0001	60.3 (7.1)	45.3 (1.4)	311.2 (6.3)	6.87	87.7 (0.9)	340.4 (82.8)	24.4 (2.7)	595.91
d: 0.0002	35.6 (1.9)	45.0 (1.7)	308.4 (4.8)	6.85	88.1 (0.5)	156.4 (25.3)	15.7 (0.7)	351.84
d: 0.0005	28.7 (0.4)	45.2 (1.5)	303.6 (2.7)	6.72	88.0 (0.4)	127.6 (4.6)	13.2 (0.2)	304.53
d: 0.001	25.9 (0.3)	44.7 (1.6)	306.1 (3.5)	6.85	87.5 (0.5)	116.2 (2.4)	11.9 (0.2)	285.02
r: 0.005	72.2 (17.5)	45.6 (1.9)	320.8 (11.7)	7.03	87.3 (0.9)	978.8 (319.0)	53.3 (12.1)	1440.57
r: 0.007	29.0 (3.5)	46.4 (3.0)	313.4 (2.5)	6.75	87.9 (0.5)	315.4 (81.9)	22.5 (2.8)	555.42
r: 0.009	21.5 (2.0)	46.4 (1.4)	318.3 (11.7)	6.86	87.9 (0.5)	239.2 (57.9)	18.4 (1.1)	452.91
r: 0.016	14.1 (0.2)	44.4 (1.6)	305.3 (7.3)	6.87	87.6 (0.5)	132.2 (4.6)	13.6 (0.2)	313.02
r: 0.02	13.3 (0.1)	44.1 (1.4)	302.5 (3.8)	6.86	87.7 (0.5)	126.8 (1.6)	13.0 (0.1)	303.36
p: 0.0003	60.6 (13.7)	44.1 (1.7)	296.6 (4.4)	6.73	86.7 (0.5)	986.8 (231.7)	50.3 (10.5)	1412.02
p: 0.0006	34.7 (11.5)	43.9 (1.1)	298.9 (3.5)	6.80	87.4 (0.5)	537.2 (271.1)	28.4 (8.3)	818.06
p: 0.0009	17.2 (3.8)	42.8 (1.3)	295.0 (4.2)	6.89	87.2 (0.7)	221.6 (109.8)	15.8 (3.5)	417.68
p: 0.0012	12.4 (1.0)	43.8 (1.2)	294.8 (3.8)	6.74	87.6 (0.5)	124.4 (35.5)	11.9 (0.7)	292.18
p: 0.0015	12.1 (1.7)	44.1 (1.5)	299.7 (4.9)	6.80	87.3 (0.6)	135.0 (53.5)	11.1 (1.1)	297.95

Table A.3: Steps count, norm, and hypervolume results in terms of mean and standard deviation when varying the stopping criteria.

SC	S_{descent}	$S_{\text{corrector}}$	||predictor||	||corrector||	train HV	valid HV
d: 5e-05	21.5 (9.2)	7.3 (8.2)	1.97e-02 (1.90e-03)	1.32e-04 (1.78e-04)	3.404 (2.75e-04)	3.383 (2.08e-04)
d: 0.0001	7.6 (6.4)	3.5 (4.2)	1.98e-02 (2.19e-03)	2.21e-04 (2.10e-04)	3.402 (3.45e-04)	3.383 (3.10e-04)
d: 0.0002	3.6 (3.1)	1.8 (1.7)	1.97e-02 (2.28e-03)	3.21e-04 (1.77e-04)	3.401 (3.68e-04)	3.382 (3.98e-04)
d: 0.0005	0.4 (0.9)	1.4 (1.5)	1.98e-02 (2.05e-03)	4.13e-04 (1.31e-04)	3.395 (6.61e-04)	3.378 (3.83e-04)
d: 0.001	0.0 (0.0)	1.1 (1.4)	1.99e-02 (2.05e-03)	5.04e-04 (1.33e-04)	3.388 (8.17e-04)	3.373 (5.47e-04)
r: 0.005	25.6 (6.9)	9.2 (8.8)	1.98e-02 (2.06e-03)	1.18e-04 (1.72e-04)	3.404 (3.66e-04)	3.383 (2.41e-04)
r: 0.007	9.3 (6.2)	3.1 (3.3)	1.98e-02 (2.04e-03)	2.11e-04 (1.80e-04)	3.403 (3.54e-04)	3.383 (3.89e-04)
r: 0.009	4.5 (3.5)	2.3 (2.7)	1.98e-02 (1.98e-03)	2.65e-04 (1.80e-04)	3.402 (3.89e-04)	3.382 (2.38e-04)
r: 0.016	0.1 (0.3)	1.4 (1.4)	1.99e-02 (1.76e-03)	4.21e-04 (1.60e-04)	3.397 (3.84e-04)	3.379 (3.22e-04)
r: 0.02	0.0 (0.0)	1.3 (1.3)	1.99e-02 (1.72e-03)	4.74e-04 (1.49e-04)	3.394 (3.69e-04)	3.378 (3.35e-04)
p: 0.0003	0.2 (0.6)	8.6 (10.2)	2.00e-02 (1.73e-03)	1.23e-04 (1.74e-04)	3.402 (6.99e-04)	3.382 (6.18e-04)
p: 0.0006	0.0 (0.0)	4.7 (7.4)	1.99e-02 (2.17e-03)	1.57e-04 (1.85e-04)	3.397 (1.82e-03)	3.379 (1.29e-03)
p: 0.0009	0.0 (0.0)	2.1 (4.0)	1.99e-02 (2.30e-03)	3.27e-04 (2.17e-04)	3.392 (2.57e-03)	3.376 (1.73e-03)
p: 0.0012	0.0 (0.0)	1.1 (1.9)	1.98e-02 (2.36e-03)	5.11e-04 (2.03e-04)	3.383 (1.90e-03)	3.369 (1.29e-03)
p: 0.0015	0.0 (0.0)	1.0 (1.9)	1.99e-02 (2.38e-03)	5.62e-04 (2.16e-04)	3.376 (3.23e-03)	3.364 (2.48e-03)

Table A.4: Time execution and evaluations count in terms of mean and standard deviation when varying the step size.

τ	t_{CPU} (s)	$t_{F(x)}$ (ms)	$t_{J(x)}$ (ms)	ρ_t	$E_{P,F(x)}$	$E_{C,F(x)}$	$E_{J(x)}$	E_w
50	149.8 (31.4)	50.5 (9.7)	348.4 (45.9)	6.91	75.6 (0.5)	2965.6 (311.5)	83.1 (13.2)	3615.32
60	114.2 (16.0)	45.6 (3.3)	322.8 (9.5)	7.07	75.3 (0.6)	2528.8 (250.2)	66.9 (9.6)	3077.61
70	73.4 (15.0)	45.9 (3.4)	330.3 (31.4)	7.19	75.2 (0.6)	1551.0 (305.5)	37.8 (6.5)	1897.88
80	39.2 (8.3)	43.9 (1.1)	307.6 (3.7)	7.00	75.2 (0.4)	731.2 (222.8)	22.9 (4.6)	966.80
90	28.6 (5.8)	44.2 (1.7)	310.9 (7.0)	7.03	74.8 (0.4)	497.0 (151.5)	20.2 (2.8)	713.75
100	27.7 (7.0)	46.4 (5.6)	354.2 (61.0)	7.63	74.9 (0.3)	368.8 (169.5)	20.8 (4.7)	602.28
110	24.9 (3.6)	56.9 (17.1)	392.2 (93.3)	6.89	75.1 (0.3)	218.2 (48.5)	17.7 (1.9)	415.02
120	23.2 (2.8)	50.3 (3.0)	354.1 (37.3)	7.04	75.1 (0.3)	190.2 (30.5)	18.1 (2.3)	392.92
130	26.0 (3.4)	54.2 (3.3)	388.3 (28.2)	7.16	75.4 (0.8)	181.4 (49.5)	19.1 (2.2)	393.35

Table A.5: Steps count, norm, and hypervolume results in terms of mean and standard deviation when varying the step size.

τ	S_{descent}	$S_{\text{corrector}}$	||predictor||	||corrector||	train HV	valid HV
50	12.0 (10.4)	17.8 (11.0)	2.45e-02 (3.20e-03)	3.58e-05 (1.18e-04)	3.398 (1.19e-03)	3.379 (8.81e-04)
60	14.9 (8.8)	14.2 (10.8)	2.47e-02 (2.92e-03)	4.68e-05 (1.33e-04)	3.399 (1.09e-03)	3.380 (8.13e-04)
70	5.7 (3.0)	7.7 (7.7)	2.47e-02 (2.95e-03)	8.21e-05 (1.58e-04)	3.401 (3.04e-04)	3.381 (2.55e-04)
80	8.9 (8.7)	4.4 (4.8)	2.47e-02 (2.80e-03)	1.57e-04 (1.98e-04)	3.401 (2.66e-04)	3.381 (3.42e-04)
90	4.0 (3.8)	3.6 (4.3)	2.48e-02 (2.80e-03)	1.98e-04 (2.05e-04)	3.401 (2.78e-04)	3.381 (1.41e-04)
100	9.7 (8.6)	4.0 (4.7)	2.47e-02 (2.35e-03)	2.42e-04 (2.26e-04)	3.402 (4.49e-04)	3.382 (2.63e-04)
110	8.4 (6.4)	3.0 (2.9)	2.46e-02 (2.78e-03)	3.00e-04 (2.22e-04)	3.402 (3.04e-04)	3.382 (3.13e-04)
120	11.6 (7.7)	3.1 (3.4)	2.46e-02 (2.80e-03)	3.28e-04 (2.29e-04)	3.402 (2.76e-04)	3.382 (3.73e-04)
130	17.5 (11.3)	3.3 (3.5)	2.46e-02 (2.62e-03)	3.29e-04 (2.42e-04)	3.402 (3.37e-04)	3.382 (3.29e-04)

Table A.6: Time execution and evaluations count in terms of mean and standard deviation when varying the batch size.

BS	t_{CPU} (s)	$t_{F(x)}$ (ms)	$t_{J(x)}$ (ms)	ρ_t	$E_{P,F(x)}$	$E_{C,F(x)}$	$E_{J(x)}$	E_w
128	401.6 (99.2)	317.0 (17.9)	2743.0 (215.4)	8.65	72.1 (2.1)	733.0 (371.8)	34.4 (7.6)	1102.73
256	200.1 (38.0)	178.4 (31.1)	1385.5 (151.6)	7.77	72.6 (1.6)	641.2 (250.7)	26.2 (3.1)	917.61
512	74.6 (9.1)	102.4 (11.3)	780.3 (48.0)	7.62	74.5 (0.7)	354.8 (56.6)	19.8 (2.3)	580.33
1024	25.2 (3.2)	51.4 (3.2)	366.1 (15.9)	7.12	74.9 (0.5)	197.0 (43.7)	17.9 (2.1)	399.63
2048	12.6 (1.6)	33.9 (3.1)	210.9 (24.1)	6.23	74.9 (0.3)	164.8 (28.9)	15.7 (1.4)	337.59
4096	8.4 (0.8)	22.0 (3.0)	110.8 (5.2)	5.04	75.1 (0.5)	162.0 (38.5)	25.1 (2.1)	363.56
8192	7.0 (0.5)	15.8 (1.8)	63.9 (4.1)	4.05	75.3 (0.5)	130.2 (32.3)	40.3 (2.6)	368.80
16384	5.4 (0.3)	12.5 (1.6)	32.9 (2.5)	2.64	75.2 (0.4)	100.2 (3.8)	71.3 (3.3)	363.53

Table A.7: Steps count, norm, and hypervolume results in terms of mean and standard deviation when varying the batch size.

BS	S_{descent}	$S_{\text{corrector}}$	||predictor||	||corrector||	train HV	valid HV
128	22.5 (9.6)	12.6 (10.6)	2.47e-02 (3.27e-03)	3.99e-04 (5.66e-04)	3.400 (7.89e-04)	3.380 (9.68e-04)
256	22.0 (8.1)	9.0 (8.9)	2.46e-02 (3.07e-03)	3.28e-04 (4.73e-04)	3.401 (7.98e-04)	3.381 (6.82e-04)
512	21.0 (9.5)	5.0 (5.6)	2.46e-02 (2.86e-03)	3.33e-04 (3.54e-04)	3.402 (3.67e-04)	3.382 (2.82e-04)
1024	12.9 (7.2)	3.1 (3.1)	2.46e-02 (2.81e-03)	2.87e-04 (2.16e-04)	3.402 (2.28e-04)	3.382 (3.06e-04)
2048	5.6 (4.5)	2.7 (2.4)	2.47e-02 (2.56e-03)	2.57e-04 (1.58e-04)	3.402 (3.42e-04)	3.382 (1.81e-04)
4096	8.7 (7.4)	2.2 (1.8)	2.47e-02 (2.68e-03)	2.64e-04 (1.43e-04)	3.402 (3.74e-04)	3.382 (2.74e-04)
8192	12.3 (11.3)	1.9 (1.2)	2.47e-02 (2.99e-03)	2.39e-04 (9.98e-05)	3.402 (2.20e-04)	3.381 (4.66e-04)
16384	13.6 (11.4)	1.5 (1.1)	2.48e-02 (2.78e-03)	2.14e-04 (8.07e-05)	3.402 (2.48e-04)	3.382 (1.82e-04)

Table A.8: Time execution and evaluations count in terms of mean and standard deviation when varying the NN training.

Training	t_{CPU} (s)	$t_{F(x)}$ (ms)	$t_{J(x)}$ (ms)	ρ_t	$E_{P,F(x)}$	$E_{C,F(x)}$	$E_{J(x)}$	E_w
t:1	19.5 (1.9)	44.4 (2.0)	310.9 (3.6)	7.00	74.5 (0.7)	235.4 (48.4)	16.4 (1.2)	424.66
t:2	20.6 (2.3)	43.8 (1.2)	304.1 (4.1)	6.95	75.0 (0.4)	210.8 (49.6)	18.1 (2.1)	411.23
t:3	17.2 (0.5)	45.0 (1.7)	313.9 (3.3)	6.97	75.0 (0.0)	159.6 (8.3)	16.6 (0.4)	350.59
t:4	19.9 (1.9)	48.0 (1.6)	338.0 (6.1)	7.05	74.5 (0.5)	196.4 (27.7)	16.5 (1.1)	387.07
t:5	20.3 (1.7)	49.6 (2.1)	343.5 (5.3)	6.93	75.2 (0.4)	195.2 (40.1)	16.9 (1.2)	387.41
t:6	24.6 (2.8)	45.8 (1.2)	315.2 (5.0)	6.88	74.6 (0.5)	270.0 (64.8)	20.4 (2.2)	485.08
t:7	18.5 (2.2)	45.8 (0.7)	323.8 (7.9)	7.07	74.8 (0.7)	183.2 (45.8)	15.4 (1.5)	366.96
t:8	19.1 (1.8)	45.2 (1.1)	319.7 (4.9)	7.08	74.7 (0.5)	193.2 (27.3)	16.5 (0.9)	384.37
t:9	22.4 (2.1)	48.8 (1.5)	343.0 (4.0)	7.03	75.3 (0.5)	182.4 (24.9)	18.1 (1.3)	385.16
t:10	27.7 (5.4)	50.9 (6.5)	345.8 (30.0)	6.79	75.0 (0.0)	243.8 (61.5)	21.3 (3.5)	463.49

Table A.9: Steps count, norm, and hypervolume results in terms of mean and standard deviation when varying the NN training.

Training	S_{descent}	$S_{\text{corrector}}$	||predictor||	||corrector||	train HV	valid HV
t:1	7.6 (5.9)	2.7 (2.5)	2.47e-02 (2.59e-03)	3.14e-04 (2.35e-04)	3.402 (1.06e-04)	3.386 (1.91e-04)
t:2	10.2 (5.1)	3.1 (3.1)	2.46e-02 (2.79e-03)	3.20e-04 (2.35e-04)	3.402 (3.23e-04)	3.382 (3.46e-04)
t:3	2.0 (2.3)	2.7 (2.1)	2.47e-02 (2.43e-03)	2.66e-04 (1.17e-04)	3.393 (3.10e-04)	3.378 (3.84e-04)
t:4	8.1 (3.1)	2.7 (2.4)	2.47e-02 (2.60e-03)	2.67e-04 (1.62e-04)	3.396 (2.43e-04)	3.380 (2.01e-04)
t:5	6.4 (3.3)	2.8 (2.5)	2.46e-02 (3.23e-03)	2.77e-04 (1.70e-04)	3.397 (2.77e-04)	3.382 (3.93e-04)
t:6	9.4 (7.8)	3.7 (4.2)	2.47e-02 (2.96e-03)	2.84e-04 (2.45e-04)	3.405 (2.19e-04)	3.380 (5.10e-04)
t:7	1.8 (2.2)	2.5 (2.6)	2.47e-02 (3.01e-03)	3.14e-04 (2.16e-04)	3.391 (2.26e-04)	3.364 (1.85e-04)
t:8	4.9 (4.5)	2.7 (2.2)	2.47e-02 (2.71e-03)	2.85e-04 (1.65e-04)	3.390 (3.47e-04)	3.378 (2.58e-04)
t:9	6.9 (5.0)	3.1 (2.3)	2.46e-02 (2.97e-03)	2.91e-04 (1.71e-04)	3.399 (5.60e-04)	3.378 (2.58e-04)
t:10	11.1 (6.9)	3.9 (4.3)	2.47e-02 (2.44e-03)	2.50e-04 (2.02e-04)	3.404 (2.80e-04)	3.355 (3.48e-04)

Table A.10: Time execution and evaluations count in terms of mean and standard deviation when varying the problem size.

Size	t_{CPU} (s)	$t_{F(x)}$ (ms)	$t_{J(x)}$ (ms)	ρ_t	$E_{P,F(x)}$	$E_{C,F(x)}$	$E_{J(x)}$	E_w
small	20.3 (3.8)	45.3 (7.4)	286.6 (24.1)	6.32	75.0 (0.4)	207.4 (46.1)	18.1 (2.1)	396.60
medium	559.3 (48.5)	1121.9 (14.0)	3118.0 (32.1)	2.78	72.9 (0.3)	299.2 (69.4)	24.9 (2.1)	441.36

Table A.11: Steps count, norm, and hypervolume results in terms of mean and standard deviation when varying the problem size.

Size	S_{descent}	$S_{\text{corrector}}$	||predictor||	||corrector||	train HV	valid HV
small	10.2 (5.1)	3.1 (3.1)	2.46e-02 (2.79e-03)	3.20e-04 (2.35e-04)	3.402 (3.23e-04)	3.382 (3.46e-04)
medium	10.5 (7.0)	4.6 (4.0)	2.47e-02 (2.93e-03)	2.03e-04 (1.49e-04)	3.402 (2.68e-04)	3.381 (2.45e-04)

Bibliography

Ak, R., Vitelli, V., and Zio, E. (2015). An Interval-Valued Neural Network Approach for Uncertainty Quantification in Short-Term Wind Speed Prediction. *IEEE Transactions on Neural Networks and Learning Systems*, 26(11):2787–2800.

Arteta, A., Barán, B., and Pinto, D. (2007). Routing and Wavelength Assignment over WDM Optical Networks. A Comparison between MOACOs and Classical approaches. *Proceedings of the 4th international IFIP/ACM Latin American conference on Networking - LANC '07.*

Blank, J. and Deb, K. (2020). Pymoo: Multi-Objective Optimization in Python. *IEEE Access*, 8:89497–89509.

Borovykh, A., Bohte, S., and Oosterlee, C. W. (2018). Dilated convolutional neural networks for time series forecasting. *Journal of Computational Finance*, (March 2017).

Borovykh, A., Oosterlee, C. W., and Bohté, S. M. (2019). Generalization in fully-connected neural networks for time series forecasting. *Journal of Computational Science*, 36:101020.

Chakraborty, K., Mehrotra, K., Mohan, C. K., and Ranka, S. (1992). Forecasting the behavior of multivariate time series using neural networks. *Neural Networks*, 5(6):961–970.

Cipolla, R., Gal, Y., and Kendall, A. (2018). Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 7482–7491.

Coello Coello, C. A. and Lechuga, M. S. (2002). MOPSO: A proposal for multiple objective particle swarm optimization. *Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002*, 2:1051–1056.

Das, I. and Dennis, J. E. (2006). Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. *http://dx.doi.org/10.1137/S1052623496307510*, 8(3):631–657.

Deb, K. (2001). Multi-objective optimization using evolutionary algorithms. page 497.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A Fast Elitist Non-dominated Sorting Genetic Algorithm for Multi-objective Optimization: NSGA-II. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1917:849–858.

Désidéri, J. A. (2012). Multiple-gradient descent algorithm (MGDA) for multiobjective optimization. *Comptes Rendus Mathematique*, 350(5-6):313–318.

Eichfelder, G. (2008). Adaptive Scalarization Methods in Multiobjective Optimization.

Fieldsend, J. E., Fieldsend, J. E., Uk, E. Q., and Singh, S. (2002). A Multi-Objective Algorithm based upon Particle Swarm Optimisation, an Efficient Data Structure and Turbulence.

Fischer, T. and Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669.

Fliege, J., Drummond, L. M., and Svaiter, B. F. (2009). Newton's method for multiobjective optimization. *SIAM Journal on Optimization*, 20(2):602–626.

Fliege, J. and Svaiter, B. F. (2000). Steepest descent methods for multicriteria optimization. *Mathematical Methods of Operations Research*, 51(3):479–494.

Guerreiro, A. P., Fonseca, C. M., and Paquete, L. (2021). The Hypervolume Indicator: Computational Problems and Algorithms. *ACM Computing Surveys*, 54(6).

Hillermeier, C. (2001). Nonlinear Multiobjective Optimization. *Nonlinear Multiobjective Optimization.*

Iba, H., Sato, T., and de Garis, H. (1994). System identification approach to genetic programming. *IEEE Conference on Evolutionary Computation - Proceedings*, 1:401–406.

Jahn, J. (1986). Mathematical vector optimization in partially ordered linear spaces. page 310.

Khare, V., Yao, X., and Deb, K. (2003). Performance Scaling of Multi-objective Evolutionary Algorithms. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2632:376–390.

Kinnear, K. E., Langdon, W. B., Spector, L., Angeline, P. J., and O'Reilly, U.-M. (1994). *Advances in genetic programming.* MIT press.

Kuhn, Harold, Tucker, A. (1951). Nonlinear Programming. In *Proceedings of 2nd Berkeley Symposium on Mathematical Statistics and Probabilistics*, pages 481–491.

Lara, A., Alvarado, S., Salomon, S., Avigad, G., Coello Coello, C. A., and Schütze, O. (2013). The gradient free directed search method as local search within multi-objective evolutionary algorithms. *Advances in Intelligent Systems and Computing*, 175 ADVANC:153–168.

Lara, A., Sanchez, G., Coello, C. A., and Schütze, O. (2010). HCS: A new local search strategy for memetic multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 14(1):112–132.

Lee, S. W. and Kim, H. Y. (2020). Stock market forecasting with super-high dimensional time-series data using ConvLSTM, trend sampling, and specialized data augmentation. *Expert Systems with Applications*, 161:113704.

Li, M. and Yao, X. (2019). Quality Evaluation of Solution Sets in Multiobjective Optimisation. *ACM Computing Surveys (CSUR)*, 52(2).

Liao, Y., Kodagoda, S., Wang, Y., Shi, L., and Liu, Y. (2016). Understand scene categories by objects: A semantic regularized scene classifier using Convolutional Neural Networks. *Proceedings - IEEE International Conference on Robotics and Automation*, 2016-June:2318–2325.

Lim, B., Ark, S., Loeff, N., and Pfister, T. (2021). Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764.

López, S., Henández, C., and Rodríguez, K. (2022). Multi-objective framework for quantile forecasting in financial time series using transformers. *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 395–403.

Ma, P., Du, T., and Matusik, W. (2020). Effcient continuous pareto exploration in multi-Task learning. *37th International Conference on Machine Learning, ICML 2020*, PartF16814:6478–6487.

Marler, R. T. and Arora, J. S. (2010). The weighted sum method for multi-objective optimization: New insights. *Structural and Multidisciplinary Optimization*, 41(6):853–862.

Martín, A. and Schütze, O. (2018). Pareto Tracer: a predictorcorrector method for multi-objective optimization problems. *Engineering Optimization*, 50(3):516–536.

Martin, B., Goldsztejn, A., Granvilliers, L., Martin, B., Goldsztejn, A., Granvilliers, L., Jermann, C., and Continuation, O. (2013). On Continuation Methods for Non-Linear Bi-Objective Optimization : Certified Interval-Based Approach To cite this version : HAL Id : hal-00878729.

McConaghy, T., Leung, H., and Varadan, V. (2000). Functional reconstruction of dynamical systems from time series using genetic programming. In *IECON Proceedings (Industrial Electronics Conference)*, volume 1, pages 2031–2034.

Miettinen, K. (1998). Nonlinear Multiobjective Optimization. 12.

Mora, A. M., Merelo, J. J., Millan, C., Torrecillas, J., and Laredo, J. L. J. (2006). CHAC. A MOACO Algorithm for Computation of Bi-Criteria Military Unit Path in the Battlefield.

Neely, C., Weller, P., and Dittmar, R. (1997). Is Technical Analysis in the Foreign Exchange Market Profitable? A Genetic Programming Approach. *The Journal of Financial and Quantitative Analysis*, 32(4):405.

Pérez, A. M. (2008). Pareto Tracer: Metodo Predictor Corrector para Problemas de Optimizacion Multiobjetivo.

Rangapuram, S. S., Seeger, M., Gasthaus, J., Stella, L., Wang, Y., and Januschowski, T. (2018). Deep state space models for time series forecasting. *Advances in Neural Information Processing Systems*, 2018-Decem(NeurIPS):7785–7794.

Salinas, D., Flunkert, V., Gasthaus, J., and Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191.

Schäffler, S., Schultz, R., and Weinzierl, K. (2002). Stochastic method for the solution of unconstrained vector optimization problems. *Journal of Optimization Theory and Applications*, 114(1):209–222.

Schütze, O., Dell'Aere, A., and Dellnitz, M. (2005). On Continuation Methods for the Numerical Treatment of Multi-Objective Optimization Problems. *Dagstuhl Seminar Proceedings*.

Schütze, O., Martín, A., Lara, A., Alvarado, S., Salinas, E., and Coello, C. A. (2016). The directed search method for multi-objective memetic algorithms. *Computational Optimization and Applications*, 63(2):305–332.

Sener, O. and Koltun, V. (2018). Multi-task learning as multi-objective optimization. *Advances in Neural Information Processing Systems*, 2018-Decem(NeurIPS):527–538.

Shrivastava, N. A., Lohia, K., and Panigrahi, B. K. (2016). A multiobjective framework for wind speed prediction interval forecasts. *Renewable Energy*, 87:903–910.

Tapia, M. G. C. and Coello, C. A. (2007). Applications of multi-objective evolutionary algorithms in economics and finance: A survey. *2007 IEEE Congress on Evolutionary Computation, CEC 2007*, pages 532–539.

Tsang, E. P. and Martinez-Jaramillo, S. (2004). Computational finance. *Computing in Science and Engineering*, 1(6):22–23.

Wang, H. (2012). Zigzag Search for Continuous Multiobjective Optimization. *https://doi.org/10.1287/ijoc.1120.0528*, 25(4):654–665.

Wang, H., Laredo, D., Cuate, O., and Schütze, O. (2019). Enhanced directed search: a continuation method for mixed-integer multi-objective optimization problems. *Annals of Operations Research*, 279(1-2):343–365.

Wang, X., Zhou, Y., Kong, D., Currey, J., Li, D., and Zhou, J. (2017). Unleash the Black Magic in Age: A Multi-Task Deep Neural Network Approach for Cross-Age Face Verification. *Proceedings - 12th IEEE International Conference on Automatic Face and Gesture Recognition, FG 2017 - 1st International Workshop on Adaptive Shot Learning for Gesture Understanding and Production, ASL4GUP 2017, Biometrics in the Wild, Bwild 2017, Heteroge*, pages 596–603.

Xue, N., Triguero, I., Figueredo, G. P., and Landa-Silva, D. (2019). Evolving Deep CNN-LSTMs for Inventory Time Series Prediction. *2019 IEEE Congress on Evolutionary Computation, CEC 2019 - Proceedings*, pages 1517–1524.

Zhang, C., Wei, H., Xie, L., Shen, Y., and Zhang, K. (2016). Direct interval forecasting of wind speed using radial basis function neural networks in a multi-objective optimization framework. *Neurocomputing*, 205:53–63.

Zhang, G., Eddy Patuwo, B., and Y. Hu, M. (1998). Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, 14(1):35–62.

Zhou, M., Wang, B., Guo, S., and Watada, J. (2021). Multi-objective prediction intervals for wind power forecast based on deep neural networks. *Information Sciences*, 550(xxxx):207–220.

Zitzler, E. ., Laumanns, M. ., Thiele, L., Zitzler, E., and Laumanns, M. (2001). SPEA2: Improving the strength pareto evolutionary algorithm. *TIK-Report*, 103.