National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Parallel CFD & Optimization Unit

Deep Neural Networks and their Differentiation for use in Gradient-Based Aerodynamic Shape Optimization

Diploma Thesis

Konstantina Kovani

Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2023

Acknowledgements

I would first like to take this opportunity to express my sincere gratitude to my supervisor, Professor Kyriakos C. Giannakoglou. I am deeply grateful for his invaluable support and guidance throughout my Diploma Thesis, and for giving me the opportunity to be a member of the PCOpt/NTUA team for almost 2 years. His scientific expertise, problem-solving approach to challenging topics, and his commendable skill in conveying his knowledge to his students have been truly inspiring for me, during all the years of my studies. I consider myself very fortunate that I was able to work under his diligent supervision, and his mentoring in academic matters and beyond has helped me grow as a prospective engineer and as a person.

Secondly, I would like to express my profound appreciation to all the members of the PCOpt/NTUA team for their enthusiastic support. I am wholeheartedly thankful to PhD Candidate Marina Kontou and Dr. Varvara Asouti, whose guidance and assistance have been invaluable and truly inspiring. I feel deeply grateful for welcoming me into your team, providing academic advice and consultation during my studies, and for always sharing chocolates with me.

Lastly, I would like to thank my parents and my sister, Eleni, for always supporting me and believing in me throughout my studies. I am thankful to my friends for accompanying me along the journey and creating unforgettable memories.

National Technical University of Athens
School of Mechanical Engineering
Fluids Section
Parallel CFD & Optimization Unit

Deep Neural Networks and their Differentiation for use in Gradient-Based Aerodynamic Shape Optimization

Diploma Thesis

Konstantina Kovani

Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2023

Abstract

The objective of this Diploma Thesis is the implementation of differentiated Deep Neural Networks (DNNs), within a gradient-based optimization method in fluid mechanics, for predicting both the objective function values and its gradient, and therefore decreasing the overall computational cost of the optimization.

In the proposed method, DNNs, after being trained on a set of patterns for which the objective function values are available, are used to replace both the code simulating the fluid flow and its adjoint solver computing gradients w.r.t. the design variables in problems governed by partial differential equations. The derivatives of the responses of the trained DNNs with respect to its inputs (which are the design variables of the optimization problem) are computed using automatic differentiation in reverse accumulation mode. Parametric studies on the DNNs hyperparameters are conducted, regarding the accuracy in both their predictions and gradients. Prior to successfully and efficiently supporting the optimization loop, gradients are verified against finite differences as well as the adjoint method.

The proposed DNN-driven shape optimization method is presented in two variants. The first (standard), involves DNNs trained only on the objective function values. The second, involves DNNs trained on both the objective function values and its sensitivity derivatives (gradient-assisted training), computed using the adjoint method. Two implementations of the latter are presented: The first, is based on the Sobolev Training of DNNs while the second is a new concept, based on the principles of the polynomial Hermite interpolation. All variants are demonstrated in CFD applications. The standard variant is used for the shape optimization of two isolated airfoils (inviscid and turbulent flow) and an S-bend duct (laminar flow). The Sobolev and Hermite variants are demonstrated in the turbulent flow case. The efficiency

of the proposed optimization in all its variants is compared with an adjoint-based optimization.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης

Βαθιά Νευρωνικά Δίκτυα και η Διαφόρισή τους για Χρήση στην Αιτιοκρατική Βελτιστοποίηση Αεροδυναμικών Μορφών

Διπλωματική Εργασία

Κωνσταντίνα Κοβάνη

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2023

Περίληψη

Στόχος αυτής της διπλωματικής εργασίας είναι η υλοποίηση της διαφόρισης των Βαθιών Νευρωνικών Δικτύων (ΒΝΔ), και η χρήση τους σε μία αιτιοκρατική μέθοδο βελτιστοποίησης στο πεδίο της μηχανικής ρευστών, για την πρόβλεψη τόσο των τιμών της συνάρτησης-στόχου όσο και των παραγώγων της και, συνεπώς, τη μείωση του συνολικού υπολογιστικού κόστους της βελτιστοποίησης.

Στην προτεινόμενη μέθοδο, τα ΒΝΔ, αφού εκπαιδευτούν σε ένα σύνολο μοτίβων-δειγμάτων για τα οποία είναι διαθέσιμες οι τιμές της συνάρτησης-στόχου, χρησιμοποιούνται για να αντικαταστήσουν τόσο τον κώδικα που προσομοιώνει τη ροή του ρευστού, όσο και τον υπολογισμό των παραγώγων ως προς τις μεταβλητές σχεδιασμού με τη συζυγή μέθοδο, σε προβλήματα που διέπονται από μερικές διαφορικές εξισώσεις. Οι παράγωγοι των αποκρίσεων των εκπαιδευμένων ΒΝΔ ως προς τις εισόδους τους (οι οποίες είναι οι μεταβλητές σχεδιασμού του προβλήματος βελτιστοποίησης) υπολογίζονται χρησιμοποιώντας αντίστροφη αυτόματη διαφόριση. Επιπλέον, συμπεριλαμβάνονται παραμετρικές μελέτες ως προς τις (υπερ)παραμέτρους των ΒΝΔ, όσον αφορά τόσο την ακρίβεια των προβλέψεών τους, όσο και των παραγώγων τους. Πριν από την επιτυχή και αποτελεσματική υποστήριξη του βρόχου βελτιστοποίησης, οι υπολογιζόμενες παράγωγοι των ΒΝΔ επαληθεύονται έναντι των πεπερασμένων διαφορών, καθώς και της συζυγούς μεθόδου.

Η προτεινόμενη μέθοδος βελτιστοποίησης σχήματος οδηγούμενη από τα ΒΝΔ, παρουσιάζεται με δύο παραλλαγές. Η πρώτη, περιλαμβάνει ΒΝΔ που έχουν εκπαιδευτεί μόνο στις τιμές της συνάρτησης-στόχου, ενώ η δεύτερη, περιλαμβάνει ΒΝΔ που έχουν εκπαιδευτεί τόσο στις τιμές της συνάρτησης-στόχου, όσο και στις τιμές των παραγώγων ευαισθησίας οι οποίες υπολογίζονται με τη συζυγή μέθοδο. Η τελευταία παραλλαγή παρουσιάζεται με δύο υλοποιήσεις: Η πρώτη βασίζεται στην κατά Sobolev εκπαίδευση

των ΒΝΔ, ενώ η δεύτερη αποτελεί μια νέα ιδέα, βασισμένη στις αρχές της πολυωνυμικής παρεμβολής Hermite. Όλες οι παραλλαγές εφαρμόζονται σε προβλήματα Υπολογιστικής Ρευστοδυναμικής. Η πρώτη παραλλαγή χρησιμοποιείται για τη βελτιστοποίηση σχήματος δύο μεμονωμένων αεροτομών (σε ατριβή και τυρβώδη ροή) και ενός αγωγού S-bend (στρωτή ροή). Οι υλοποιήσεις Sobolev και Hermite της δεύτερης παραλλαγής παρουσιάζονται στην περίπτωση της τυρβώδους ροής. Η αποτελεσματικότητα της προτεινόμενης μεθόδου βελτιστοποίησης, σε όλες τις παραλλαγές της, συγκρίνεται με μια βελτιστοποίηση που βασίζεται στη συζυγή μέθοδο.

Contents

Chapter 1

Introduction

1.1 Artificial Intelligence and Machine Learning

Over the past few years, the evolution of Artificial Intelligence (AI) and Machine Learning (ML) has been nothing short of transformative. Advancements in computing power, the availability of large datasets, and breakthroughs in algorithm design have propelled AI and ML to new heights. Deep learning, a subfield of ML that focuses on neural networks with many layers, has led to significant breakthroughs in areas like image recognition, natural language processing, and speech synthesis. This progress has enabled AI systems to achieve human-level performance in tasks such as playing complex games and generating creative content. With the rapid development of AI-driven applications like self-driving cars, virtual assistants, and recommendation systems, we're witnessing a fundamental transformation of industries and society as AI and ML continue to evolve. Let's not forget the global impact of the recent breakthrough, ChatGPT, a language model created by OpenAI designed to understand and generate human-like text based on the received input. With the introduction of such a powerful tool, new possibilities for innovation and exploration in technology have opened up, along with the raise of complex ethical, societal, and regulatory questions.

ML, as a subfield of AI, focuses on developing algorithms and models that enable computers to learn from data and make predictions or decisions without being explicitly programmed [1]. There are several types of ML [2], each with its own approach and characteristics. The main types of ML and the included tasks in each case are presented in Fig. 1.1. These are briefly:

- **Supervised Learning**: In supervised learning, the algorithm learns from la-

beled training data, where the input data is paired with corresponding target values. The goal is to learn a mapping from inputs to outputs, allowing the model to make predictions on new, unseen data. Common tasks include classification (assigning labels to data points) and regression (predicting continuous values).

- **Unsupervised Learning**: In unsupervised learning, the algorithm works with unlabeled data, seeking to find patterns or structures within the data. This includes clustering (grouping similar data points together) and dimensionality reduction (reducing the number of features while preserving important information).

- **Reinforcement Learning**: Reinforcement learning involves an agent that interacts with an environment to learn how to take actions that maximize a cumulative reward over time. The agent learns through trial and error, receiving feedback in the form of rewards or penalties based on its actions. It's often used for tasks like game playing and robotic control.

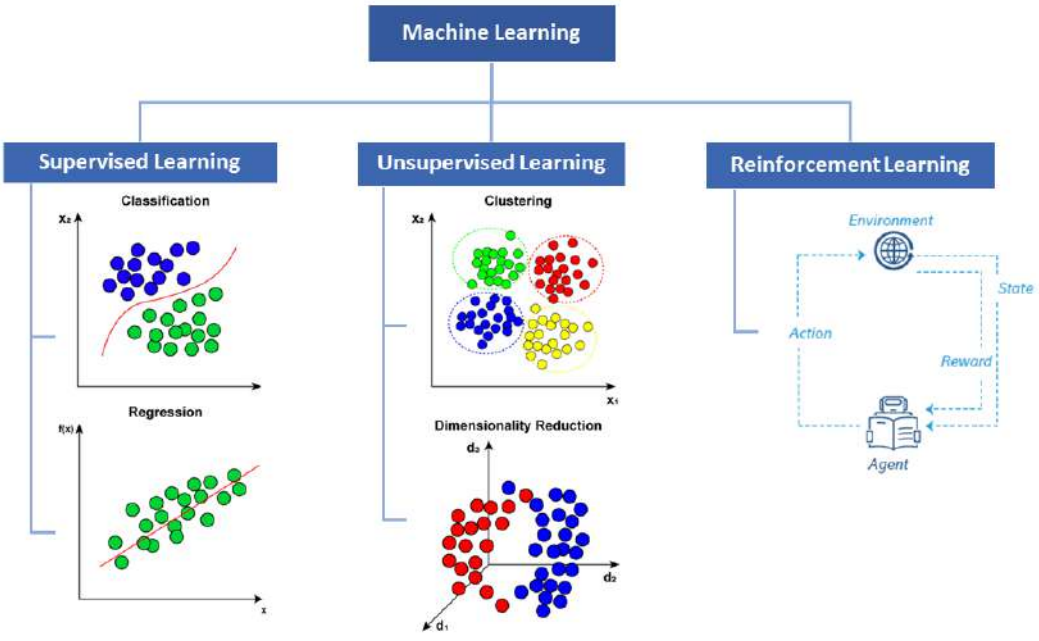


Figure 1.1: *Different Types of ML. Figure from [3].*

1.2 Supervised ML Models for Regression in CFD Applications

In this Diploma Thesis, supervised ML models will be used for regression tasks in the field of Computational Fluid Dynamics (CFD). It was decided to procceed with Deep Neural Networks (DNNs), offspring models of the general Artificial Neural Networks (ANNs) family, that belong in the deep learning subset of ML. Beside ANNs, there are other worth-mentioning ML algorithms that have gained ground in the field of CFD and optimization, with the most commonly used being Random Forests and Support Vector Regression [4, 5], among others. An overview of the models follows, in order to demonstrate their main functionalities. Since, in this work, there is significant interest about the gradients of the models with respect to their inputs, the differentiability of each ML model, along with the capabilities and challenges involved in the computation of the gradient in each case, are presented and compared.

1.2.1 Random Forests

Random Forests [6] is a popular ML algorithm used for both classification and regression tasks, based on the ensemble learning technique. It's designed to improve the performance and robustness of the so-known decision trees by combining the predictions of multiple trees. Ensemble uses two types of methods:

- **Bootstrap Aggregating (Bagging)**: Random Forests employ a technique called bootstrapping, where multiple subsets of the original dataset are created by randomly sampling with replacement. Each subset is known as a "bootstrap sample."
- **Boosting**: It combines weak learners into strong learners by creating sequential models such that the final model has the highest accuracy. A decision tree is constructed for each "bootstrap sample" with a subset of features rather than all the features, that helps introduce diversity in the trees.

Each decision tree is constructed by recursively partitioning the data based on the selected features. The tree continues to grow until a stopping criterion is met, such as reaching a maximum depth, a minimum number of samples in a leaf node, or no further improvements in impurity reduction. Once all the trees are built, they collectively make predictions on new data. For regression problems, all the individual trees' predictions are averaged to obtain the final regression prediction, as shown in Fig. 1.2. This ensemble approach helps to reduce overfitting and improve the generalization performance of the model.

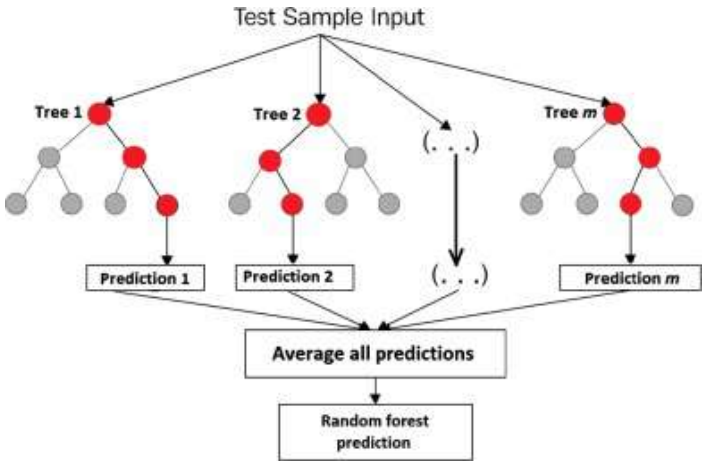


Figure 1.2: *Demonstration of the Random Forest Model. Figure from [7].*

Random Forests have the ability to handle high-dimensional data, detect feature importance, and mitigate overfitting. They are also robust to noisy data and outliers and imune to the curse of dimensionality. However, they may require tuning of hyperparameters, such as the number of trees and the maximum depth of individual trees. Their performance is evidenced in various works in the field of fluid mechanics and aerodynamics, as in [7, 8, 9].

1.2.2 Support Vector Regression

Support Vector Regression (SVR) [10] is a type of ML algorithm used for regression analysis. The goal of SVR is to find a function that approximates the relationship between the input variables and a continuous target variable, while minimizing the prediction error. Unlike Support Vector Machines (SVMs) used for classification tasks, SVR seeks to find a hyperplane that best fits the data points in a continuous space. This is achieved by mapping the input variables to a high-dimensional feature space and finding the hyperplane that maximizes the margin (distance) between the hyperplane and the closest data points, while also minimizing the prediction error. SVR can handle non-linear relationships between the input variables and the target variable by using a kernel function to map the data to a higher-dimensional space. This makes it a powerful tool for regression tasks where there may be complex relationships between the input variables and the target variable. The most important parameters of an SVR are:

- **Kernel**: A kernel helps us find a hyperplane in the higher dimensional space without increasing the computational cost. Kernel functions are used to implicitly perform the transformation into the higher-dimensional feature space without explicitly calculating the transformed features. Common kernels include Linear, Polynomial, Radial Basis Function (RBF), and more.

- **Hyperplane**: The hyperplane, in the transformed feature space, that best approximates the relationship between the input features and the target values.
- **Decision Boundary**: A decision boundary can be thought of as a distance from the original hyperplane. A "margin of error" is defined around the predicted output for each data point, represented by two hyperplanes: an upper margin (ξ-insensitive) and a lower margin (-ξ -insensitive) as shown in Fig. 1.3 .

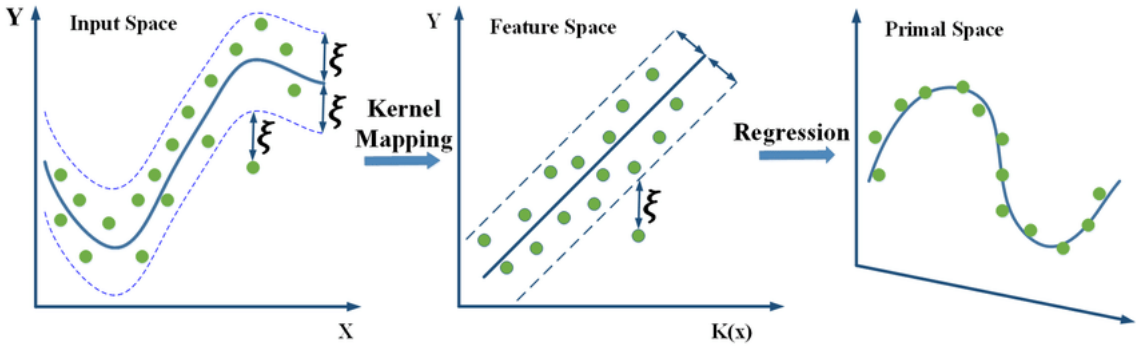


Figure 1.3: *(Left) The initial features are placed on the input space along with the initial hyperplane of the SVR and the defined decision boundary (Middle) After a kernel-mapping procedure, the original features are tranformed and placed in a new space, in which the margin of error will be minimized. (Right) The resulting hyperplane that best fits the input data after the regression procedure is demontrated on the primal feature space. Figure from [11].*

In SVR, the goal is to minimize the margin of error between the predicted output and the true output, while still allowing for a certain degree of error (controlled by hyperparameters). This is achieved by finding the hyperplane that minimizes the sum of the errors within the defined margins while maximizing the margin width. Assuming that the equation of the hyperplane is $Y = w \cdot X + b$ and the equations of the decision boundary are $w \cdot X + b = \xi$ and $w \cdot X + b = -\xi$ for the upper and lower boundary respectively, the objective of the SVR optimization is formulated as a constrained optimization problem expressed by

$$-\xi \leq w \cdot X + b \leq \xi \tag{1.1}$$

SVR seeks to find a balance between fitting the data and preventing overfitting by introducing the margin of error. It also incorporates a regularization term that controls the trade-off between fitting the training data closely and having a simple model that generalizes well to new data. It's important to tune hyperparameters like the regularization parameter and the kernel choice in order to achieve the best performance on a specific dataset. The performance of SVRs is demonstrated in various CFD-related works, as in [12, 13].

1.2.3 Artificial Neural Networks

Artificial Neural Networks (ANNs) [14] are computational models inspired by the structure and function of the human brain. They consist of layers of interconnected nodes, called neurons, organized into input, hidden, and output layers. The configuration of a basic ANN is shown in Fig. 1.4.

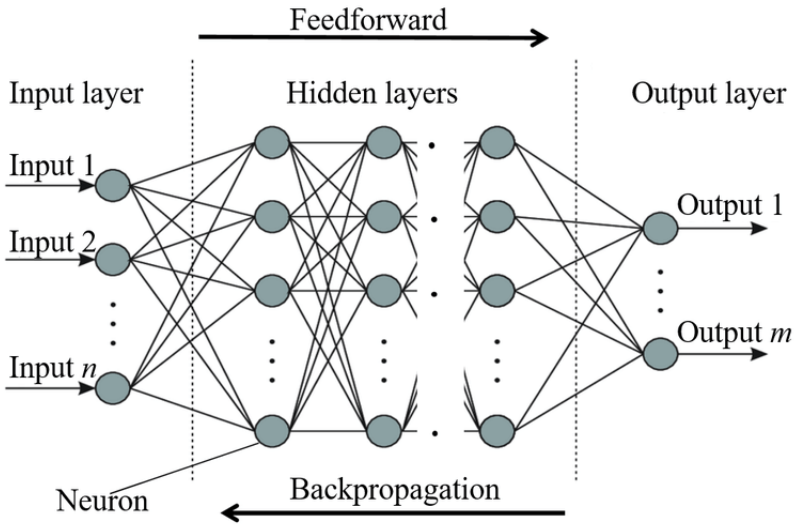


Figure 1.4: *Demonstration of a typical ANN architecture. Figure from [15].*

Neurons process input data through weighted connections and apply activation functions, producing output that is then passed to the next layer. During training, ANNs adjust the weights to minimize the difference between their predictions and actual target values using optimization algorithms. This process involves forward propagation to compute predictions and backward propagation (backpropagation) to update weights based on the calculated gradients. By iteratively fine-tuning weights over multiple epochs, ANNs learn to capture complex patterns and relationships in data, enabling them to make accurate predictions on new, unseen data. ANNs are an extremely powerful tool in CFD applicaitons and the performance of their subclass models, DNNs, will be presented later in this Chapter.

1.2.4 An Overview on the Differentiability of the most Common ML Models used for Regression

Random Forests, Support Vector Regression and ANNs are powerful ML tools, but they have different characteristics when it comes to computing derivatives. Since this work emphasizes in the computation and use of the models' gradients in an optimization process, the differentiability of each model must be considered. The capabilities and challenges when computing the derivatives of the most popular ML

models used in regression are presented in summary

- **Random Forests**: Random Forests are non-differentiable models [16]. Since they are constructed as an ensemble of decision trees, they do not involve an optimization process that requires the computation of gradients. Instead, they rely on a simple, parallel, and non-differentiable procedure for training each decision tree independently. The lack of differentiability means that Random Forests may not perform as well as differentiable models in tasks that require learning intricate patterns and representations from data.

- **Support Vector Regression**: SVRs' differentiability is limited to a subdifferentiability concept, which can affect the overall optimization process. SVRs typically involve loss functions that are not easily differentiable, such as the ϵ-insensitive loss [17]. This makes it challenging to compute derivatives directly from the trained SVR model. Additionally, SVRs are often optimized using techniques like Quadratic Programming (QP) that do not inherently provide the necessary information for differentiating the model and their performance might depend on the choice of the kernel function. Among these techniques, the most common one is to solve the Langragian Dual Problem of the optimization, as in [18]. The problem is transformed into its Lagrangian dual form and Lagrange multipliers (also known as dual variables) are introduced to express the optimization problem as a function of these multipliers. As a result, this method does not require the explicit differentiation of the model.

- **Artificial Neural Networks**: ANNs are highly differentiable models [19] and enable the use of backpropagation to compute gradients (derivatives) analytically. Backpropagation efficiently calculates gradients by propagating errors backward through the network layers using the chain rule of calculus. Modern learning frameworks like TensorFlow and PyTorch provide automatic differentiation capabilities, making it easy to compute accurate derivatives during training and optimization processes. Due to their well-established architectures and optimization techniques, neural networks are designed to handle complex functions and capture intricate relationships within the data. This allows them to compute accurate derivatives for a wide range of functions.

Technically, it is possible to estimate the derivatives of all models using numerical methods, such as finite differences [20]. However, it is not a common or efficient approach due to the complexity and limitations involved. Finite differences would require perturbing input data points and observing the corresponding changes in the predictions, which can introduce additional uncertainty and complexity, especially in cases where the models may not accurately capture the underlying function behavior. In the case of Random Forests, the discrete and piecewise nature of the predictions would make it difficult to define meaningful derivatives. Their predictions can be highly discontinuous, especially at decision boundaries, while finite differences typically assume a continuous function. Also, even if the derivatives for each individual tree were computed, combining these derivatives in a meaningful way

to obtain derivatives for the entire forest would likely be challenging. Similarly, the SVR models produce piecewise linear predictions based on support vectors and their associated coefficients. This piecewise nature of the predictions can make it difficult to compute meaningful derivatives using finite differences, especially at points where the model transitions from one support vector to another.

1.3 DNNs in CFD and Optimization

So far, CFD-based optimization tools for large scale applications usually rely on gradient-based techniques supported by the adjoint method, [21, 22]. The latter computes the gradient of the objective function with respect to (w.r.t.) the design variables at a cost which is independent of their number N. At the same time, DNNs and their integration within simulations are gaining ground due to their ability to handle large volumes of complex data at low computational cost and resources.

For instance, [23] uses conditional variational autoencoders and an integrated generative network for the inverse design of supercritical airfoils. [24] presents a solver based on ML models that predict the required numerical fluxes, in compressible fluid flows, based on high-resolution runs; the solver was fully differentiated using automatic differentiation (AD). A toolkit based on complex-step finite differences for the numerical differentiation of neural networks was proposed in [25], making it computationally lightweight by overcoming the high-order chain rule. In [26], the authors of PCOpt proposed a DNN-based surrogate for the turbulence closure of the Reynolds-Averaged Navier Stokes (RANS) equations; the role of the DNN is to replace the numerical solution of the turbulence and transition models. The DNN-assisted RANS solver was combined with an evolutionary algorithm to optimize the shape of a transonic turbine blade and a car model. ML surrogates were used in aerodynamic shape optimization of transonic airfoils, in [27]. In [28], the performance of ML models used in aerodynamic shape optimization is reviewed, and the efficiency of more advanced models using appropriate geometry parameterization so as to reduce the dimensionality of the design space, is presented. In [29], a numerical methodology based on modal decomposition coupled with the regression analysis for creating reduced-order models of fluid flows is demonstrated. In [30], the efficiency of the adjoint-based optimization is accelerated using DNNs to predict the mapping between the adjoint vector and the local flow variables.

This Diploma Thesis proposes the implementation of differentiated DNNs, within a gradient-based optimization method in fluid mechanics, for predicting both the objective function values and its gradient, in order to reduce the overall cost of the optimization [31].

1.4 Thesis Outline

Following the Introduction, this Thesis is organized as follows:

- **Chapter 2**: A brief introduction in DNNs and how they work. The basic concepts involved, their configuration and hyperparameters, as well as the constituents of the training process are presented. In addition, the technique used to differentiate the DNNs with respect to their inputs is demonstrated, and a study on the parameters that influence the quality of the computed gradients is performed.
- **Chapter 3**: The proposed DNN-driven optimization algorithm is presented and explained, along with the adjoint-based algorithm used for comparison. The in-house softwares and tools involved in each method are presented and the proposed optimization is demontrated in simple function approximation problem.
- **Chapter 4**: The proposed optimization algorithm is demonstrated on the shape optimization of the symmetric NACA0012 isolated airfoil. The flow around the airfoil is inviscid and the DNN-driven optimization is compared with an adjoint-based optimization in terms of efficiency and cost.
- **Chapter 5**: The proposed method is demonstrated on the shape optimization of an S-bend duct with laminar flow. Additional capabilities of the proposed algorithm are explored and the DNN-driven optimization is compared with its adjoint-based counterpart.
- **Chapter 6**: The proposed method is demonstrated on the shape optimization of the low-speed S8052 isolated airfoil. The flow around the airfoil is turbulent and the DNN-driven optimization is compared with an adjoint-based optimization in terms of efficiency and cost. A parametric study on the size of the database used to train the DNNs is performed, in order to demonstrate the sensitivity of both the model's predictions and computed gradients when the database's size increases.
- **Chapter 7**: The incorporation of the DNNs' gradients in the training process is demonstrated and implemented with two variations, based on the Sobolev training method and the Hermite interpolation, respectively. The proposed algorithm is re-newed, in which gradient-assisted trained DNNs are used to drive the descent during the optimization. The re-newed DNN-driven optimization is demonstrated on the S8052 airfoil's shape optimization and compared with the original DNN-Driven algorithm and an adjoint-based optimization, in terms of efficiency and cost.

Chapter 2

Deep Neural Networks

2.1 Neuron Model and Network Architecture

DNNs, as a subset of ANNs, are computational systems that mimic the biological neural systems [14]. The modeling and implementation of a neural network is primarly inspired by the functionality of the basic computational unit of the brain: the neuron. A neuron is considered in its coarse representation, in which all its operations are summarized in the functionality of its two main constituents: the branches (dendrites) and the axon. All neurons receive input signals from their dendrites and are sequently transfered to the main body, where they all get summed. If the final sum is above a certain threshold the neuron can fire, delivering output signals (spikes) along its axon. The axon eventually branches out and connects to dendrites of other neurons via communicative junctions, known as synapses, transfering the output signals to the neighbour neurons. In reality, biological neural networks are significantly complex dynamical systems, in which a vast amount of non-linear computations is performed, and the timing and change rate of operations has a serious effect on the outcome as well. Recently, new models such as Spiking Deep Neural Networks [?] have been developed in order to capture the composition of such complex systems, however their performance is still a topic of research for the neural network community.

The sequence of operations involved in the coarse representation of a neuron motivated the deployment of a mathematical model, focusing on the existance of a computational unit able to collect, process and as well fire signals when defined criteria are met. At the computational model of a neuron signals that travel along the axons interact multiplicatively with the dendrites of other neurons, based on the synaptic strength at that synapse. Synaptic strengths, known as weights, control the influence of signals from one neuron to another and thus, are the parameteres

to be learned. The firing rate of a neuron is modeled by an activation function (f), which represents the frequency of the spikes along the axon. The biological and mathematical representation of a neuron are presented and compared in Fig. 2.1.

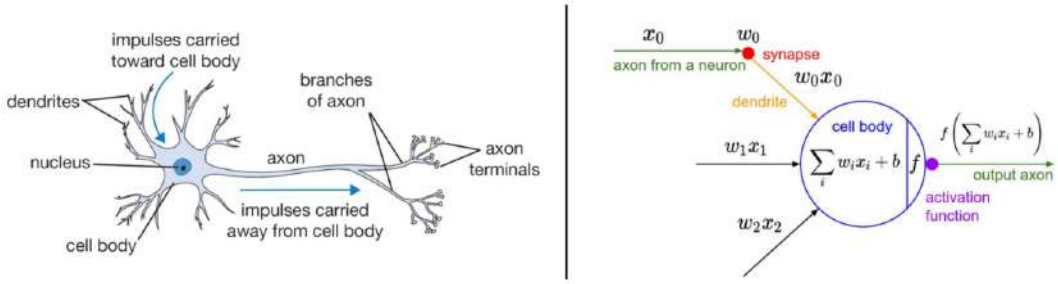


Figure 2.1: *(Left) Representation of a biological neuron and its main constituents. (Right) The mathematical model of a neuron and its core operations. Figure from (https://cs231n.github.io/neural-networks-1/).*

A shown in Fig. 1.4, DNNs are organized into layers, each performing specific functions to process the input data and produce the desired output [32]. Their most significant distinction is their depth, as they can have multiple hidden layers, enabling them to learn hierarchical features and representations. Other neural network architectures might have fewer hidden layers or even just a single hidden layer, making them shallower and less capable of capturing intricate patterns. The arrangement of these layers defines the DNN architecture, that is typically organized as:

- **Input Layer**: The input layer is the initial layer of the network and receives the raw input data. The number of nodes (neurons) in this layer corresponds to the dimensions of the input data.
- **Hidden Layers**: These are the layers between the input and output layers and are responsible for learning increasingly abstract and complex features from the input data. A DNN can have multiple hidden layers, and these layers are where the "deep" aspect of deep learning comes into play. Each hidden layer consists of multiple neurons and, herein, the layers are fully connected, meaning each neuron is connected to every neuron in the previous and subsequent layers.
- **Output Layer**: The output layer produces the final prediction or result of the network. The number of nodes in this layer corresponds to the number of dimensions in the output.

The neurons in each layer are typically followed by an activation function. Activation functions introduce non-linearity to the network, allowing it to learn complex relationships in the data. Each connection between neurons has associated weights and biases, parameters that are learned during the training process.

2.2 Training Process of DNNs

2.2.1 The gradient-based optimization problem

The training process of DNNs is formulated as a gradient-based optimization problem in which the network's parameters (weights and biases) are iteratively updated in order to minimize a defined loss function. An overview of the steps taken during the training algorithm follows:

1. **Parameter Initialization**: The weights and biases of the network are initialized with small random values. Proper initialization is important to prevent the network from getting stuck in local minima during training.

2. **Forward Propagation**: During each iteration (epoch) of training, input data is fed into the network's input layer. The data passes through the hidden layers, and the network computes predictions using the current weights and biases.

3. **Loss Function Calculation**: The output of the network is compared to the actual target values using a loss function, that quantifies the difference between the predicted output and the true values.

4. **BackPropagation**: After calculating the loss, the network performs backpropagation. Backpropagation involves calculating the gradients of the loss with respect to the network's parameters, using the chain rule of calculus. Gradients indicate how much each parameter should be adjusted to decrease the overall loss.

5. **Gradient Descent**: With the gradients calculated, an optimization algorithm (optimizer) is used to update the parameters, by adjusting them in the direction that reduces the loss. Learning rate, which determines the step size of the parameter updates, is a critical hyperparameter in this process.

6. **Parameter Update**: The parameters are updated based on the calculated gradients and the learning rate.

Steps 2 to 6 are repeated for multiple epochs, in which the network processes the entire training dataset. The process continues until a predefined stopping condition is met, usually a maximum number of epochs. To properly monitor the training, the model 's hyperparameters must be carefully tuned. These hyperparameters involve the selection of the appropriate loss function and optimizer, along with the optimal learning rate. Another consideration while training a DNN, is the common phenomenon known as "Overfitting", where a model learns to perform well on the training data but fails to generalize to new data due to capturing noise and random fluctuations [33]. The behavior of an overfitted model is presented in Fig. 2.2.

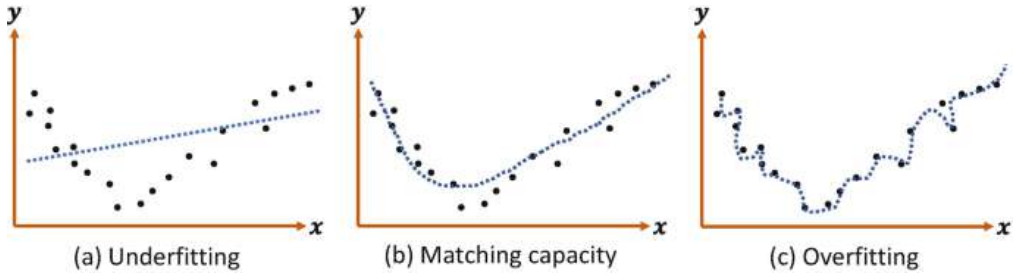


Figure 2.2: *(Left) Behavior of a poor-trained model that results in underfitting of the data. (Middle) Behavior of a better trained DNN that captures more accurately the underlying relations of the data. (Right) Behavior of an overfitted model that fails to generalize on unseen data. Figure from [33]*

The most common way to overcome this issue is to use the Validation technique; During training, a set of data, to be referred as validation set, is set aside and can not be seen by the network. After a pre-defined epoch frequency, the network's performance on the validation set is evaluated, in order to monitor the generalization ability of the DNN and assist in its hyperparameter tuning. As a result, validation prevents overfitting and allows for early stopping if the validation performance starts deteriorating.

2.2.2 Loss Functions for Regression Tasks

The loss function measures the discrepancy between the predicted output of the neural network and the actual target values [34]. It quantifies how well the network is performing on a specific task and the choice of the loss function depends on the nature of the problem being solved. The most commonly used loss functions for regression problems are the Mean Absolute Error (MAE) and Mean Squared Error (MSE), expressed in Eqs. 2.1, 2.2 respectively.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2.1}$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.2}$$

y_i is the target value of each sample indexed with $i = 1, ..., N$. N is the total number of samples and $\hat{y}_i$ is the predicted value from the DNN. Both loss functions behave differently during the optimization; MAE is less sensitive to outliers and treats all errors equally, compared to MSE. This makes it a good choice when the dataset contains noisy data points that might skew the predictions. However, MAE has a discontinuous gradient at zero, which can make the training more challenging. On

the other hand, MSE has a smooth gradient at all points, which aids in faster and more stable optimization using gradient-based methods, while is more sensitive to outliers, as squaring the errors amplifies their effect. The selection between MAE and MSE depends on the characteristics of the dataset and the problem to be solved. In the case of outliers, the more robust MAE loss would be preferable. The MSE loss is more suitable for a non-noisy dataset and for strongly penalizing larger errors.

2.2.3 The Adam Optimizer

The optimizer is the algorithm responsible for updating the weights and biases of the neural network during the training process [35]. There are various optimization algorithms available, each with its own characteristics and advantages. The most common ones are stochastic gradient descent (SGD), Adam, RMSprop, and Adagrad, among others. Each optimizer has different hyperparameters, such as the learning rate, momentum, decay rates, and more. In this work, all DNNs are trained using the Adam optimizer. Adam [36] stands for "Adaptive Moment Estimation" and involves a combination of two gradient descent methodologies:

Momentum: This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace. The weights are updated as follows:

$$w_{t+1} = w_t - am_t \tag{2.3}$$

where w_t, w_{t+1} are the model's weights at the time steps $t, t+1$ respectively, a is the learning rate and,

$$m_t = \beta m_{t-1} + (1-\beta)\left[\frac{\delta L}{\delta w_t}\right] \tag{2.4}$$

$\frac{\delta L}{\delta w_t}$ is the derivative of the loss function (L) with respect to the model's weights at the current time step (t), m_t, m_{t-1} are the aggregates of gradients at time steps $t, t-1$ respectively and β, is a moving average parameter with the default value of 0.9.

Root Mean Square Propagation (RMSprop): In RMSprop, instead of taking the cumulative sum of squared gradients, the 'exponential moving average' is considered as follows:

$$w_{t+1} = w_t - \frac{a}{(v_t+\epsilon)^{\frac{1}{2}}} \cdot \left[\frac{\delta L}{\delta w_t}\right] \tag{2.5}$$

where,

$$v_t = \beta v_{t-1} + (1-\beta)\left[\frac{\delta L}{\delta w_t}\right]^2 \tag{2.6}$$

v_t is the sum of square of past gradients and ϵ is a small positive constant (default value 10^{-8}) used to avoid division by zero when v_t approaches zero.

The formula for computing the model's weights updates results from the combination of Eqs. 2.3 and 2.5, respectively, as:

$$w_{t+1} = w_t - \frac{a}{(v_t + \epsilon)^{\frac{1}{2}}} \cdot m_t \tag{2.7}$$

Combining the advantages of previous models, the Adam optimizer is considered as one of the most versatile optimization algorithms, offering efficient convergence and adaptability to the learning process. While its performance can depend on the hyperparameter settings of the model, it's still the most commonly used optimizer for training neural networks and the most popular in the literature.

2.3 Differentiation of DNNs

2.3.1 Reverse Automatic Differentiation

Reverse Automatic Differentiation (RAD) is the core technique used to compute gradients efficiently in neural networks [37, 38]. During training, it enables the network to learn by adjusting its parameters based on the gradients of the loss function with respect to those parameters. In forward pass, input data travels through the layers of the network, and intermediate values are stored. In the reverse pass, gradients are computed in a top-down manner, starting from the loss function and propagating backwards. Gradients are calculated using the chain rule of calculus, where each layer's contribution to the gradient is the product of the local gradient of the layer's activation function and the upstream gradients. The traces of the forward and backward pass are presented in Fig. 2.3 for a simple model architecture.

While during the training process the models' gradients are computed with respect to their parameteres, herein RAD is used to compute the DNN's gradients with respect to their input variables. As evidenced from Fig. 2.3, the outcome of the full forward pass of a DNN is a result of repeated matrix multiplications, interwoven with the application of the activation functions. Consequently, the computed derivatives resulting from the DNN's differentiation will be highly determined by the model's architecture (width and depth), the - fixed - weights resulting from the training process and, the used activation functions [39, 40, 41].

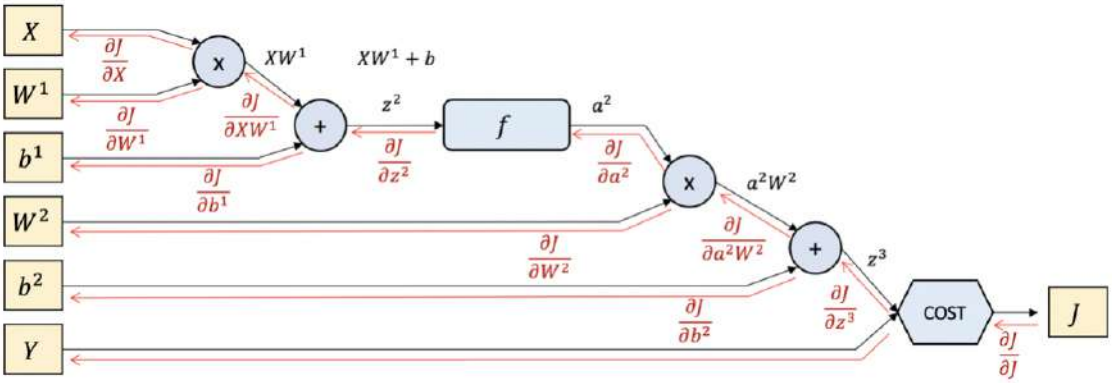


Figure 2.3: *Demonstration of the RAD technique. The traces of the forward pass of the model are shown in black. The traces of the backward pass during the differentiation of the loss function (J) are shown in red. Figure from [42].*

2.3.2 Parameters that Influence the Computed Gradients

Neural Network Architecture: The number of the DNN hidden layers and their neurons (width and depth) will determine the number of weights that will be assigned to each input feature, giving insight into the importance of that feature in making predictions. In networks with more complex structures, computing the gradients with respect to their inputs can be more intricate, due to the interactions and flow of information across different parts of the architecture. Especially in networks with high-dimensional inputs, the gradients can be sensitive to small changes in individual input dimensions, making interpretation and analysis challenging.

Weights Initialization and Non-Unique Solutions: Neural networks are sensitive to their initial weights. This sensitivity is particularly pronounced in deep architectures, where slight changes in the initial weights can lead to different trajectories during optimization, resulting in different solutions. This non-deterministic behavior inherent in the training of DNNs can have notable implications for the computed gradients with respect to their inputs and their stability, causing fluctuations in their directions and magnitudes for the same input across different training runs or solutions.

Weights Magnitude and Sign: The magnitude and sign of the weights in a trained DNN play an important role in shaping its gradients. Larger magnitudes of weights amplify the impact of input changes on the model's output, while both their magnitude and sign determine the strength of the activations across the model's layers. The interplay between weight magnitudes and signs defines the sensitivity of the model to input variations and therefore its gradients. However, the values assigned to the weights after training are such as to properly approximate the output target,

and, in case the training fails to capture the complex interactions and dependencies between inputs and the underlying patterns of the data, the accuracy of the computed gradients will be in question as well.

Activation Functions: Activation functions and their derivatives have a significant influence on the computed gradients of the DNN. The most commonly used activations in the literature are the Rectified Linear Unit (ReLU), the Exponential Linear Unit (ELU), the Gaussian Error Linear Unit (GELU), the tanh and sigmoid. Complementary, the less popular Scaled Exponential Linear Unit (SELU) and the Sigmoid-weighted Linear Unit (swish) are also presented. The activation functions, as expressed from Eqs. 2.8 - 2.14, are demonstrated along with their derivatives in Fig. 2.4, computed on an input variable x in the range of $[-5, 5]$.

$$ReLU = \begin{cases} 0, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases} \tag{2.8}$$

ReLU is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks.

$$ELU = \begin{cases} \alpha(e^x + 1), & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases} \tag{2.9}$$

ELU [43] is an alternate of ReLU. It can output negative values and it slowly becomes smooth until its output is equal to $-\alpha$. The ELU's hyperparameter $\alpha > 0$ controls the value to which an ELU saturates for negative net inputs, diminishing various problems such as vanishing gradients.

$$SELU = s \cdot \begin{cases} \alpha(e^x + 1), & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases} \tag{2.10}$$

SELU [44] resembles ELU and induces self-normalizing properties. The values of α and scale (s) are chosen so that the mean and variance of the inputs are preserved between two consecutive layers, as long as the weights are initialized correctly. The default values are $\alpha = 1.67326324$ and $s = 1.05070098$ respectively.

$$GELU = x \cdot \Phi(x) = \frac{1}{2}x\left[1 + erf\left(\frac{x}{\sqrt{2}}\right)\right] = \frac{1}{2}x\left(1 + tanh\left[\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right]\right) \tag{2.11}$$

$\Phi(x)$ is the standard Gaussian cumulative distribution function. The GELU nonlin-

earity weights inputs by their percentile, rather than gates inputs by their sign as in ReLU [45]. Consequently the GELU can be thought of as a smoother ReLU.

$$tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.12}$$

$$sigmoid = \frac{1}{1 + e^{-x}} \tag{2.13}$$

$$swish = x \cdot sigmoid(\beta x) = \frac{x}{1 + e^{-\beta x}} \tag{2.14}$$

Swish [46] is a self-gating activation function with a constant or trainable parameter $\beta \in [0, 1]$, that allows better tuning of the activation and maximization of the propagated information. This results in smoother gradients and better generalization.

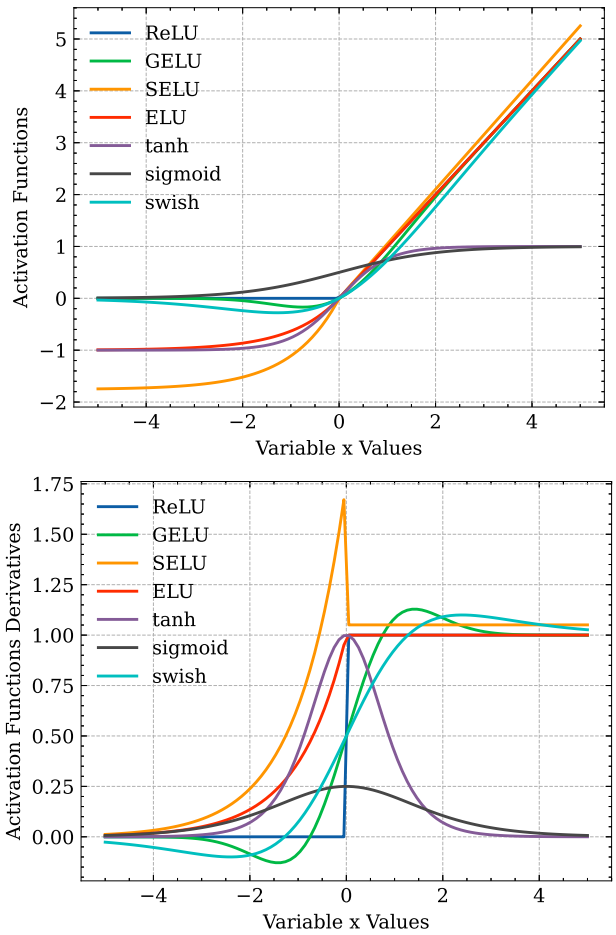


Figure 2.4: *Representation of the most commonly used activation functions in the literature (Top) and their derivatives (Bottom).*

As shown in Fig. 2.4, some activations of the ReLU family, such as ReLU and SELU, are non-smooth activations and its variants introduce non-differentiability at zero. This is known as the "Dying ReLU" phenomenon, which can result in unstable or noisy gradients with respect to the inputs, especially around zero. On the other hand, GELU, ELU and swish are smooth activation functions and have continuous derivatives, providing more stable and interpretable gradients. The derivatives of GELU and swish can also yield negative values, contributing in a more flexible way to the sign of the network's gradients. The sigmoid and tanh activations and their derivatives are also smooth and continuous, however they can saturate for large or small inputs, squashing the input values into a limited range and resulting in the so-known phenomenon of "Vanishing gradients". Vanishing gradients occur when gradients become extremely small, such as to approach zero, resulting in negligible and stalled computed values.

Chapter 3

The Proposed DNN-Driven Gradient-Based Optimization

3.1 Introduction

Herein, the proposed DNN-driven optimization algorithm is presented. The proposed method is compared with the adjoint-based optimization, since the latter is so widely used in CFD-based optimization. The steps of both optimization algorithms are described and presented, as well as the in-house softwares and tools involved in each case. Before going to the CFD-based applications, the proposed method is demonstrated in the minimization of a manifold, bi-variate function, in order to better visualize the DNN-driven descent and the capabilities of the algorithm.

3.2 The Adjoint-Based Optimization Algorithm

The adjoint technique [47] is used in order to compute the sensitivities of an objective function (or constraint), concerning the design variables. It is widely used in CFD to support gradient-based algorithms, as it has the lowest cost of computing derivatives of functions in problems governed by partial differential equations. It can be implemented with two variants, the continuous and the discrete adjoint. In the continuous adjoint method, the gradient of the objective function is computed by solving adjoint equations derived (in the form of PDEs) from the governing fluid flow equations. This involves solving the forward (primal) flow problem, formulating and solving the adjoint equations and then, computing sensitivities with respect to

design variables. The discrete adjoint method handles the discretized flow/primal equations on meshes. It follows a similar process, discretizing equations, solving the forward problem, solving discrete adjoint equations, and finally obtaining sensitivities for optimization. Both methods are crucial for efficiently optimizing designs in CFD, since computing gradients through finite differences would be computationally expensive.

In this work, the continuous adjoint approach is used; A Lagrangian function is formed by adding the objective function to be minimized to the integral (over the flow domain) of the residuals of the flow equations multiplied by the adjoint variables (or Lagrange multipliers). It is evident that objective and Lagrangian functions take on the same value as the flow equations are always satisfied and, thus, their residuals are zero. Therefore, the gradient of the Lagrangian, rather than the objective function, can be computed. This is differentiated w.r.t. to the design variables and terms multiplying the derivatives of flow variables w.r.t. to the design variables are set to zero, leading to the adjoint equations. The adjoint equations are derived in the form of partial differential equations which are, then, discretized and numerically solved using the in-house solver, PUMA [48]. In the adjoint-based optimization, each cycle comprises the numerical solution of the Navier-Stokes equations, that of the adjoint equations and the computation of the sensitivity derivatives (SDs) used to update the design variables vector. Without loss of generality, all updates are computed by steepest descent. The number of optimization cycles to be carried out is determined by a convergence-or-cost related termination criterion.

3.3 DNNs as Surrogates of the Flow and Adjoint CFD Solver in Optimization

Alternatively, this work proposes the replacement of both the flow and the adjoint equations solvers with a trained DNN, which predicts both the objective function value and the SDs. The proposed optimization algorithm is demonstrated in the flow chart of Fig. 3.1. Working with DNNs, the first step is to collect the necessary training data and create the database (to be referred to $\mathrm{DB_{DNN}}$) which the DNN will be trained on. Herein, the $\mathrm{DB_{DNN}}$ is formed by sampling the design space using the Latin Hypercube Sampling (LHS) technique [49], generating the corresponding geometries and evaluating them on the CFD solver. The LHS is effective in case the number of samples must be kept small, and is widely used in DNNs. In this work, reducing the size of the $\mathrm{DB_{DNN}}$ is important as all of its entries should be evaluated on the costly CFD code.

Once the initial $\mathrm{DB_{DNN}}$ resulting from the LHS is available, each round (this term is used to distinguish this loop and the gradient-based descent loop of step 2, in which

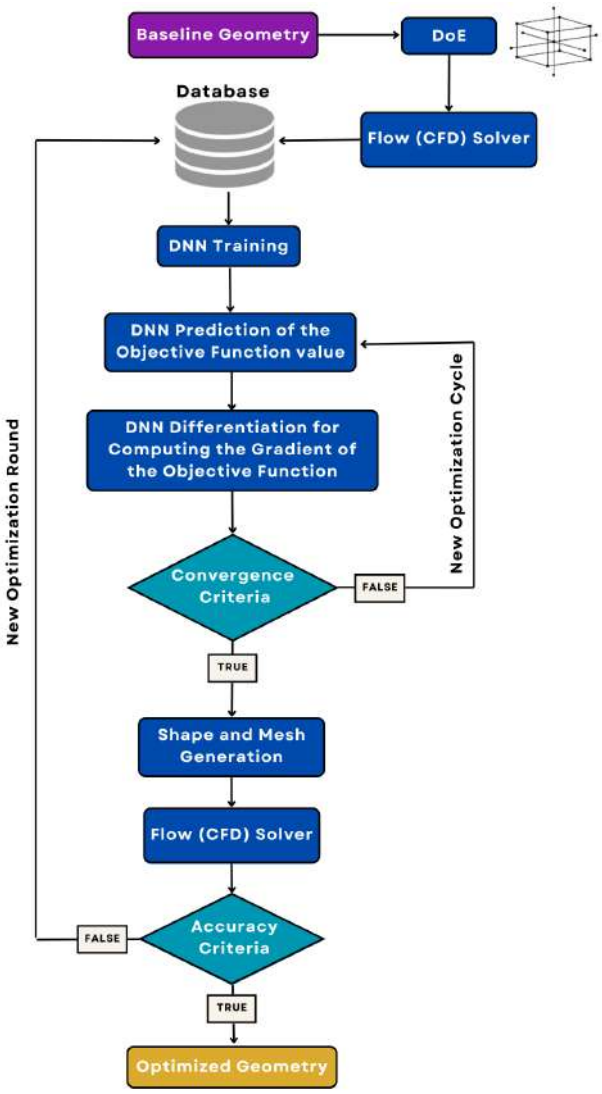


Figure 3.1: *Flow-Chart of the proposed DNN-driven gradient-based optimization algorithm.*

optimization cycles are performed by updating the design vector and the gradient) of the proposed algorithm comprises the following steps:

1. Train the DNN using the data available in the $\mathrm{DB}_{\mathrm{DNN}}$. Herein, the model's configuration is carefully selected in order to achieve high accuracy in both its predicitons and computed gradients. The setup, training and differentiation of the DNNs is carried out in the TensorFlow framework (v2.6.0), [50], using Python.

2. Iteratively optimize (till convergence) by applying gradient-based descent using, exclusively, the DNN-based sensitivities. In the general case, a number of entries selected from the $\mathrm{DB}_{\mathrm{DNN}}$ can be used as starting points (starting

designs) and perform as many runs as the number of starting points.

3. Re-evaluate (all or part of) the "optimized" solution(s) on the CFD tool; the use of quotes ("optimized") makes clear that this is the best solution according to the DNN.

4. Update the DB_{DNN} with all the recently evaluated solutions, if necessary, and repeat all four steps starting from step 1. The termination criterion is related to the DNN prediction accuracy.

In step 1, the DNN is configured differently in each problem. Experience has shown that, the use of a single DNN in all problems is not a viable solution, in CFD-based analysis. Regarding this work, the demonstrated problems involve different physics, hence, different models are deployed for each case. The DNNs hyperparameters result either by a trial-and-error procedure regarding the models's accuracy in both their predictions and computed gradients, or, they are optimized using the in-house evolutionary algorithm software, EASY [51]. The DNNs are configured only once and then can be used to drive optimizations with any user-defined objective function (assuming the geometry parameterization and the flow conditions remain the same). The cost of configuring the DNNs' architecture varies from case to case and, in compare with the cost of solving the flow equations, it can be considered relatively smaller. Especially in the case where solving the flow problem is computationally expensive, i.e. in turbulent flow, the cost of "searching" for the DNNs' configuration is even less important. At all demonstated cases, the cost needed to configure a model's architecture is assumed negligilbe, as a one-time task that once it's finished, provides a DNN model that can be flexibly used in multiple optimizations that involve different objective functions.

In both the adjoint-based and the DNN-driven optimizations, the design variables are not allowed to outpass their upper or lower boundaries, defined when sampling the geometries for constructing the DB_{DNN}. If, after an update (performed in step 2), the design variables' values violate the defined boundaries, these values are set equal to their upper or lower limits, respectively.

3.4 In-House Software and Tools

- **PUMA**: All flow simulations are performed using the in-house GPU-accelerated CFD solver, PUMA, [52, 48] which numerically solves the Navier-Stokes equations for compressible and incompressible fluids; herein the compressible flow variant is used. The flow and their (continuous) adjoint equations are discretized on unstructured/hybrid meshes, using the vertex-centered finite volume technique. The viscous flow equations for compressible fluids are written in the form

$$R_n = \frac{\partial f_{nk}^{\text{inv}}}{\partial x_k} - \frac{\partial f_{nk}^{\text{vis}}}{\partial x_k} = 0 \tag{3.1}$$

where

$f_k^{inv} = [\rho v_k \quad \rho v_k v_1 + p\delta_{1k} \quad \rho v_k v_2 + p\delta_{2k} \quad \rho v_k v_3 + p\delta_{3k} \quad \rho v_k h_t]^T$ are the inviscid and $f_k^{vis} = [0 \quad \tau_{1k} \quad \tau_{2k} \quad \tau_{3k} \quad v_\ell \tau_{\ell k} + q_k]^T$ the viscous fluxes. ρ, p, v_k and h_t stand for the fluid's density, pressure, velocity components, total enthalpy and δ_{km} is the Kronecker symbol, respectively. The viscous stress tensor is given by $\tau_{km} = \mu \left(\frac{\partial v_k}{\partial x_m} + \frac{\partial v_m}{\partial x_k} - \frac{2}{3}\delta_{km} \frac{\partial v_\ell}{\partial x_\ell} \right)$ where μ is the bulk viscosity and q_k the heat flux. All computations are made with second-order accuracy. The inviscid (Euler), laminar and turbulent flow models of simulation are included and, when the latter is selected, PUMA implements a variaty of turbulence models, such as the Spalart-Allmaras model, the standard k-epsilon model and the baseline and SST variants of the k-ω model.

In both the flow and adjoint solvers of PUMA, high parallel efficiency is achieved by the use of Mixed Precision Arithmetics (MPA), [52]. MPA reduces the memory footprint of the code and the memory transactions of the GPU threads with the device memory, without affecting code's accuracy. In particular, the memory demanding computations of the coefficient matrices of the linearized systems is performed with double, though these are stored in single, precision accuracy. The residuals of the equations, determining the accuracy of the simulation, are always computed and stored in double precision.

In addition to the flow and adjoint solvers, PUMA contains a set of shape and mesh morphing (parameterization) techniques based on volumetric Non-Uniform Rational B-Splines (NURBS), [53]. The geometry to be optimized and (part of) the grid are encapsulated within a NURBS lattice. A knot vector and a degree must be defined for each parametric direction. Each time the NURBS lattice points (a.k.a. control points) are displaced, the geometry changes and the CFD grid is adapted to it.

- **EASY**: EASY is a general purpose, high fidelity software for the search of optimal solutions in single-or multi-objective problems. The software has been extensively used in engineering applications and provides users with a high degree of control over the optimization process. Along with a wide range of options on genetic algorithms and evolutionary strategies, the tool also supports the approximation of single and multi-objective functions using ANNs for time consuming problems, and numerous others features, such as optimal selection of the taining patterns, multilevel algorithms, i.e. Hierarchical optimization, as well as the ability to incorporate metamodels when dealing with computationally expensive evaluation tools.

3.5 Demonstration of the proposed DNN-Driven Optimization Algorithm

The proposed optimization method is demonstrated on the minimization of the bivariate function $F(X, Y)$

$$F(X,Y) = cos(2X) + sin(3Y) + sin(X^2) + cos(Y^3), \qquad X, Y \in [-1.6, 1.6] \quad (3.2)$$

As shown in Fig. 3.2, F is symmetric and has two local minima on the design space, located at $(X, Y) = (1.5, 1.5)$ and $(X, Y) = (1.5, -1.5)$ respectively. These points wouldn't be the function's minimas if the (X,Y) domain was expanded beyond the range of $(-1.6, 1.6)$, as verified from the non-zero values of the F derivatives at these specific points.

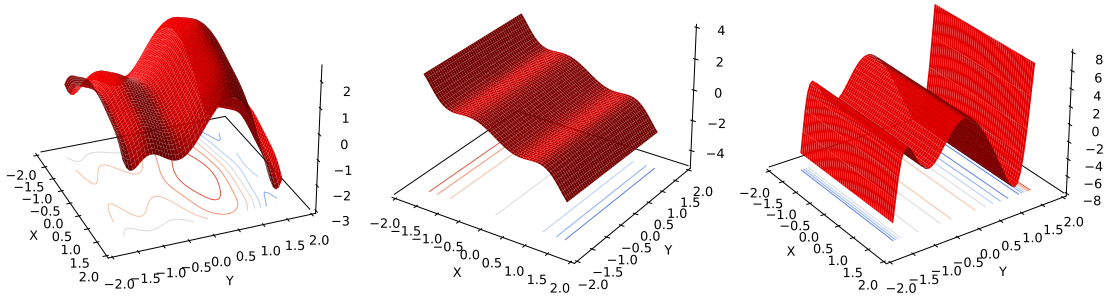


Figure 3.2: *The analytical F surface (Left) and its derivatives w.r.t. X (Middle) and Y (Right) are represented on the X-Y domain.*

First, the design space is sampled using the LHS technique and 25 samples (X-Y combinations) are generated. For each sample the value of F is computed and all data are normalized within the [0, 1] range according to their minimum and maximum values in the samples. Next, a DNN model is trained on the DB_{DNN}. The model's configuration resulted after a trial-and-error procedure on its hyperparameters and consisted of 4 hidden layers with $32-64-32-32$ neurons respectively. This DNN architecture was assessed in terms of accuracy (of both F and its gradient) by combining the most commonly used activation functions (mentioned in the previous Chapter) in all the DNN's layers. The MSE loss is selected as the loss function and Adam as the optimizer with a learning rate of 0.001. After the model was trained, it was evaluated on a test set of X,Y values (Fig. 3.3) and then, it was differentiated w.r.t. X (Fig. 3.4) and Y (Fig. 3.5) respectively in order to compute the two partial derivatives of the DNN approximation function.

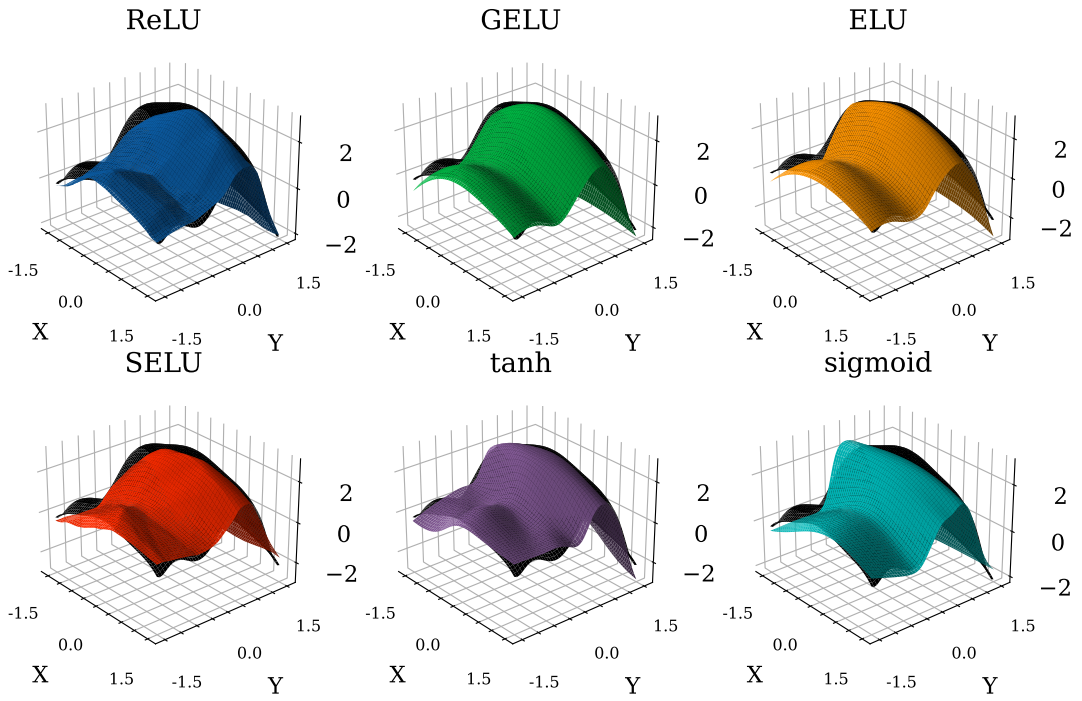


Figure 3.3: *The analytical F function is compared with the predicted surfaces of DNNs, when using diffferent activation functions in their layers.*

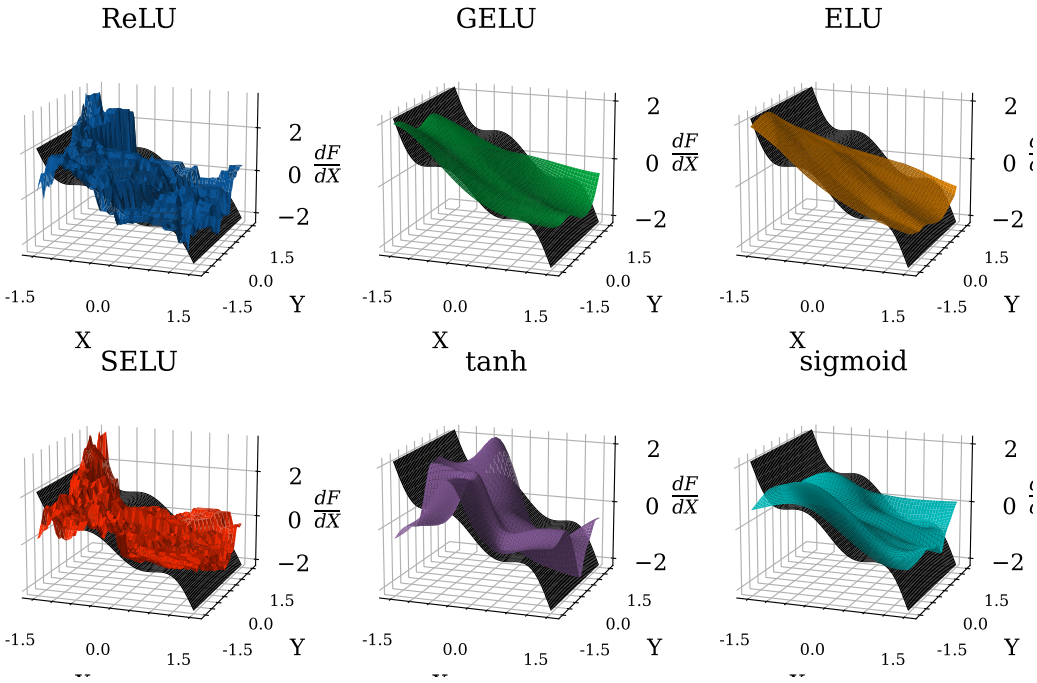


Figure 3.4: *The analytical derivative of F w.r.t. X is compared with the predicted derivatives of DNNs, when using diffferent activation functions in their layers.*

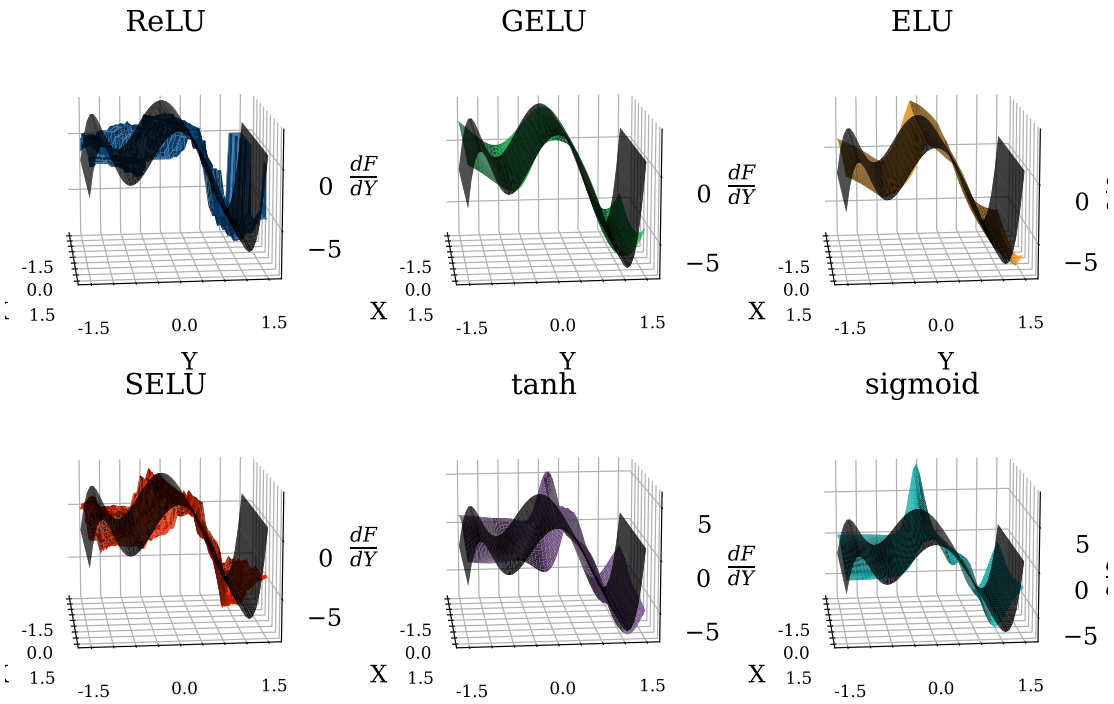


Figure 3.5: *The analytical derivative of F w.r.t. Y is compared with the predicted derivatives of DNNs, when using diffferent activation functions in their layers.*

It can be observed that all DNNs' approximation functions properly match the analytical function. However, after differentiating the DNNs, deviations are observed on both the derivatives. When the ReLU or SELU activations are used on the DNN layers, the computed gradients are coarser than the 'smooth' analytical surfaces, due to the non-continuity of the specific activations. The best gradient accuracy is achieved when using the GELU or ELU activations.

The trained DNNs are used in steepest descent algorithms in order to find the minimum of F in the (X, Y) space. Herein, $(1.0, 1.0)$ is selected as the initialization point and 10 optimization cycles are carried out with a descent step of 0.1. The DNN-driven descents are compared with the analytical descent, in which both F and its derivatives are computed by their explicit mathematical expressions, in (Fig. 3.6). As previously mentioned, at each optimization cycle's update, the design variables (herein the X, Y inputs) can not outpass their upper and lower boundaries in the samples. However, the prerequisite of a succesful DNN-driven optimization, is the capability of the DNNs to extrapolate the output values, herein predicting an output that has a lower F value than the sample with the minimum F value encoutered in the $\mathrm{DB_{DNN}}$. DNNs can struggle with extrapolation, as they might not have learned meaningful patterns and generalize beyond the training data range, and therefore their predictions for out-of-range values might not be accurate or reliable. For a better visualization of the range of the F values that the DNNs were trained on, the

25 generated samples included in the DB_{DNN} are placed in the X-Y domain, in Fig. 3.6.

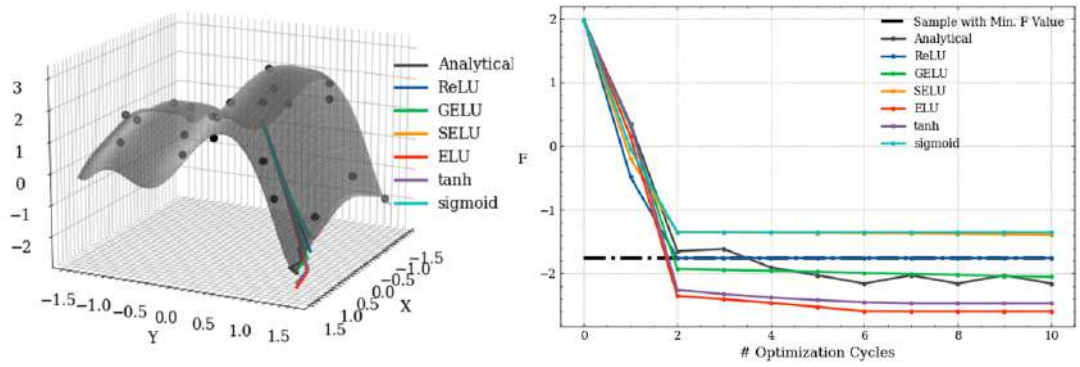


Figure 3.6: *(Left) Representation of the F surface on the X-Y domain, along with the analytical and DNN-driven descents, when using different activation functions, and the 25 samples included in the DB_{DNN}. (Right) Convergence of F at the 10 optimization cycles. The minimum value of F in the samples is drawn with a dotted line, for comparison.*

The steepest descent driven by the DNN trained with the GELU activation results in a solution that matches better the solution of the analytical descent, in comparison with the other activation functions. When the SELU or sigmoid activations are used, the DNN-driven descents result in solutions with a higher F value than the min. F in the samples, and when the ReLU is used, the descent only decreases F till that value. On the contrary, the descents driven by DNNs trained with the GELU, ELU and tanh activations result in solutions with lower values than min. F. One concern pertaining the convergence of F when using the ReLU, SELU and sigmoid activations, is that F remains fixed after it reaches a cetrain threshold (herein, the convergence curve 'flattens' after the 2^{nd} optimization cycle). This behavior is due to the saturation of specific activation functions (that affects the computed gradients) or the incapability to output negative values (that affects the predictions), and can be better understood if the function's convergnce is drawn using the normalized values. The convergence of normalized F, to be referred as F^*, is shown in Fig. 3.7.

Since, now, the min. F^* value corresponds to 0, the models must predict negative outputs in order to further decrease F^*. As shown in Fig. 2.4, ReLU and sigmoid activations are incapable of outputing negative values, and therefore can not predict a lower F^* value than the one included in the DB_{DNN}. In addition, sigmoid and tanh suffer from the 'vanishing saturation' phenomenon, near the regions that their outputs are close to 0 and 1, and -1 and 1, respectively. In these saturated regions, the activation function outputs values that are close to the limits, often lead to gradients that approach or become zero. The 'stalling' of the gradients does not allow the update of the design variables during the optimization, and therefore prevent the decrease of F^*. Despite the behavior of these activations, they can lead

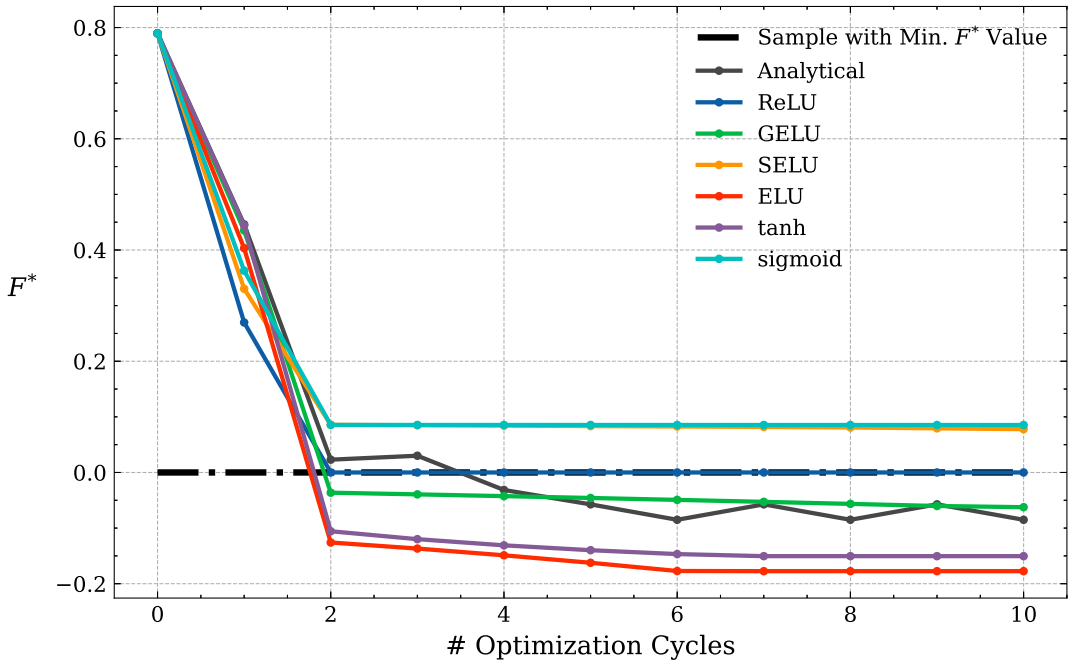


Figure 3.7: *Convergence of F^* at the 10 optimization cycles. The minimum value of F^* in the samples is drawn with a dotted line, for comparison.*

to both accurate predictions and gradients with a performance that varies from case tot case, and therefore they should not be excluded. The occuring limitations can be overcomed by appropriately normalizing the data before training. Normalization is a common practice in deep learning, as it helps in stabilizing and speeding up the training process. It ensures that the input features have a similar scale and it prevents some features from dominating the learning process due to their larger magnitudes. In minimization problems, such as the ones concerned in this work, a solution would be to normalize the training data with a value that is by a percent lower than the minimum encoutered in the samples. In that way, 0 corresponds to a lower value than the minimum in the $\mathrm{DB}_{\mathrm{DNN}}$, allowing activations that output only positive values to predict values outside the range of the training samples. Since the trained DNN will be used in an optimization, the new minimum value used for normalizing the data must be selected according to the best-case-scenario of the improvement expected during the optimization, to prevent the same behavior.

Chapter 4

Problem I: Gradient-Based Optimization of an Isolated Airfoil in Inviscid Flow

4.1 Introduction

Problem I refers to the shape optimization of the NACA0012 (symmetric) airfoil. The flow around NACA0012 is inviscid and the objective is to re-design the airfoil's shape in order to match a user-defined lift coefficient. First, a DNN is assessed and trained to predict the lift coefficient values of the airfoil, when given as input the y-coordinates of the lattice box's control points that parameterize its shape. A parametric study is performed on the model's hyperparameters focusing on the selection of the appropriate activation functions, so as to achieve high accuracy in both the model's predictions and derivatives. Later, a gradient-based optimization is performed fully driven by the DNN, which provides both the objective function's values and its sensitivity derivatives. The DNN-driven descent optimization is compared with an adjoint-based optimization in terms of effectiveness and cost.

4.2 Flow Conditions, Mesh and Shape Parameterization

The flow around the NACA0012 airfoil is inviscid with free-stream Mach number and flow angle equal to $M_\infty = 0.50$ and $\alpha_\infty = 2°$, respectively. An unstructured mesh with $\sim 7.8K$ nodes is used, shown in Fig. 4.1. The farfield boundaries of the computational domain are located about 10 chords away from the airfoil.

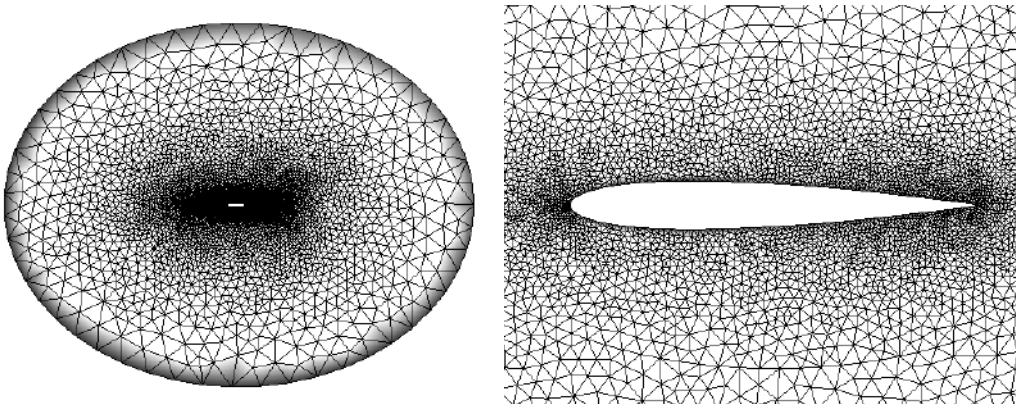


Figure 4.1: *Problem I: (Left) Mesh of the whole computational domain. (Right) Close-up view of the mesh surrounding the airfoil.*

The airfoil shape as well as part of the surrounding mesh are controlled by the 10×7 NURBS lattice of Fig. 6.3. 16 out of the 70 control points are allowed to be displaced in the normal-to-the chord (c), or vertical, direction, resulting to $N = 16$ design variables (and, thus, 16 will be the inputs to the DNN), in total. The design variables ($\vec{b} \in R^N$) are allowed to change within the $\pm 0.05c$ around their initial values, so as to avoid the overlapping of the lattice lines. The control points coinciding with the leading and trailing edge of the airfoil remain fixed.

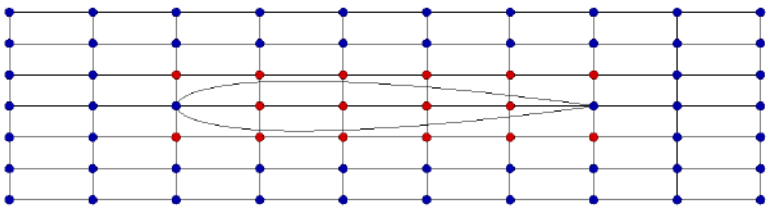


Figure 4.2: *Problem I: NURBS control lattice parameterizing the airfoil contour. Control points in blue are fixed; red ones can be displaced in the normal-to-the-chord direction.*

4.3 DNN Configuration and Training

A DNN model is assessed, so as to predict the lift coefficient (C_L) of the NACA0012 airfoil. C_L is the lift force (L) exerted by the flow on the airfoil normalized by the dynamic pressure coefficient multiplied by the airfoil's chord (c), as $C_L = L/(\frac{1}{2}\rho U_\infty^2 c)$, where U_∞ is the farfield velocity. The LHS technique is used for generating 20 different combinations of the design variables, corresponding to 20 different airfoil shapes. The generated samples are shown in Fig. 4.3. Each sampled geometry is evaluated on the CFD solver and C_L is computed. Thus, the DNN model's input is a $[20\times16]$ tensor with the sampled coordinates of the lattice control points, with the $[20\times1]$ tensor of the corresponding C_L values as output. During the training process, 20% of the generated patterns is splitted for creating the validation set, thus, a set of 15 airfoil geometries is used for training the DNN and a fixed set of 5 geometries is used for validating it.

Figure 4.3: *Problem I: All the generated airfoil shapes contained in the DB_{DNN}.*

4.3.1 Parametric Study on the DNN's hyperparameters

The DNN's configuration derives after a parametric study/trial-and-error procedure for the model's hyperparameters, focusing mainly on the number of the hidden layers, the number of neurons per layer and the activation functions. For this first problem, the selected configuration has four hidden layers, with 32, 32, 64 and 32 neurons, respectively. This DNN architecture was assessed in terms of accuracy (of both C_L and its gradient w.r.t. the design variables) by combining different activation functions. Four DNNs using the ReLU, the GELU, the sigmoid and the tanh activation functions in all hidden layers are trained and compared. All models achieve high accuracy in predicting C_L, however the accuracy of the computed derivatives differs for each model. The results are summarized in Fig. 4.4. The DNN-based SDs for the baseline geometry using GELU are in better agreement with FDs. Small discrepancies are observed in the derivatives w.r.t. some design variables, preserving though the sign of the SDs, in contrast to other activation functions that yield even wrongly signed SDs.

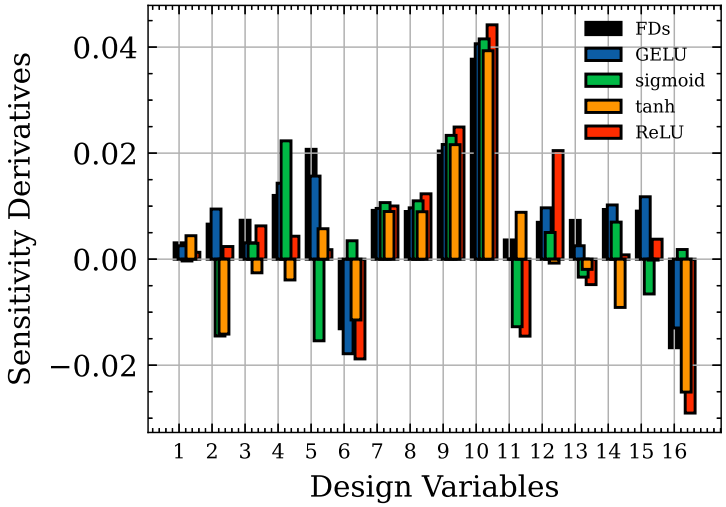


Figure 4.4: *Problem I: SDs of C_L (w.r.t. the design variables) for the baseline geometry computed with FDs (black) and derived from the DNN's differentiation for the different activation functions: ReLU (red), GELU (blue), sigmoid (green), tanh (orange). The SDs computed by the adjoint method of PUMA are omitted as these are practically identical to those computed by FDs.*

4.3.2 DNN Loss Convergence and Accuracy Metrics

After the configuration of the DNN's architecture, the tuning of the rest hyperparameters of the model follows. The Mean Squared Error (MSE) is used as the loss function during training and Adam is selected as the optimizer, with a learning rate of 0.001. In order to configure the optimal number of epochs for training the DNN, a starting number of 1000 epochs is selected. The convergence of both the training and validation losses during the 1000 epochs is presented in Fig. 4.5; approximately 600 epochs are sufficient for achieving an MSE magnitude of around $\sim 10^{-5}$ for both the training and the validation set.

The trained DNN is called to predict the C_L value of each sampled geometry in the $\mathrm{DB_{DNN}}$ and the percentage of the Mean Absolute Error (MAE) between the predictions and the exact C_L (as evaluated on the CFD code) is computed according to 4.1. In Fig. 4.6 the predictions are compared to the exact C_L values and the computed MAE metric for both the training and validation set is shown in a bar-chart. A MAE value of less than 1% is achieved for the training set with the mean value of 0.45%. For the validation set, a mean MAE value of 2.65% is achieved, where

$$MAE\,(\%) = 100 \cdot \left| \frac{C_{L,CFD} - C_{L,DNN}}{C_{L,CFD}} \right| \tag{4.1}$$

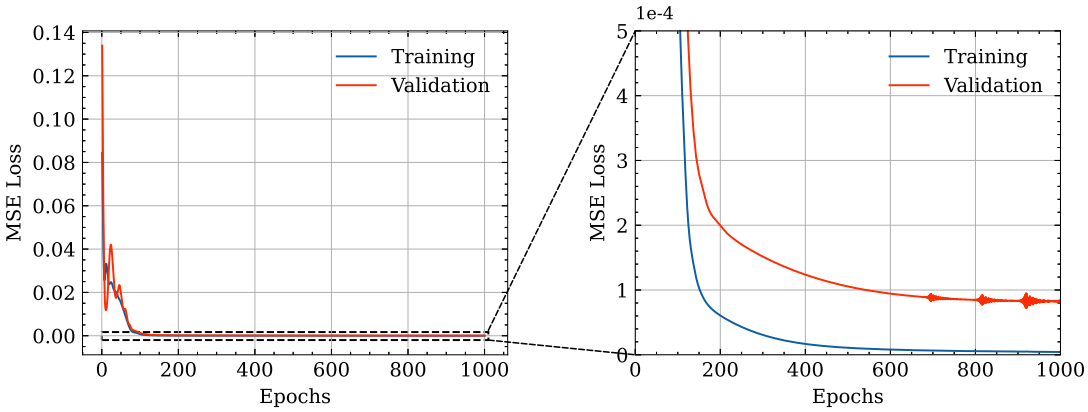


Figure 4.5: *Problem I: (Left) The convergence of the training (red) and validation (blue) losses during the traininng process of the DNN. (Right) View, in scale, of the convergence of the two losses during training. Upon the convergence of the validation loss, noisy fluctuations occur after approximately* $700 - 800$ *epochs, but still of insignificant magnitude.*

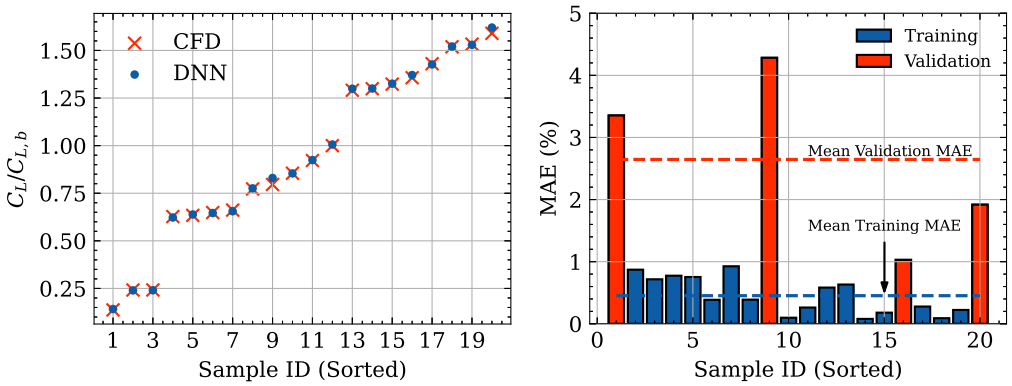


Figure 4.6: *Problem I: (Left) The* $C_L/C_{L,b}$ *ratio values (red) for each sampled geometry in the* DB_{DNN} *(sorted), where* $C_{L,b}$ *is the* C_L *value at the baseline geometry, is compared with the predicted values using the DNN (blue) respectively. (Right) The computed percentage MAE metric for each sample in both the training (blue) and validation (red) sets.*

4.4 The DNN - Driven Optimization Run

The optimization aims at re-designing the initial airfoil so as to match a user-defined lift coefficient value ($C_{L,target}$). The objective function (to be minimized) is

$$F = \frac{1}{2}\left(C_L - C_{L,target}\right)^2 \tag{4.2}$$

where $C_{L,target} = 0.6 \cdot 10^{-2}$, twice as high as the C_L of the baseline profile. Two

runs are carried out; the first run relies exclusively on the DNN using the GELU activation function (as concluded after the previously presented parametric study), while the second one on PUMA and its adjoint solver. Upon convergence of the DNN-based optimization run, the "optimized" solution is re-evaluated on PUMA. This is then added to the $\mathrm{DB_{DNN}}$, the DNN is re-trained, and the optimization is repeated. Three rounds (each of them including re-evaluations of one "optimized" solution per cycle and DNN re-training) were sufficient to reach the optimal solution with a deviation in the C_L values (w.r.t. to the $C_{L,target}$) less than 1%. Given that the cost of a DNN-based optimization as well as that of the DNN training is practically negligible (w.r.t. the cost of a CFD run, even if the less costly inviscid flow model is used), the optimization turnaround time is 23 TUs. This includes the cost to form the $\mathrm{DB_{DNN}}$ (20 TUs) and the three CFD based re-evaluations. On the other hand, the adjoint-based run (with cost of 2 TUs per cycle) needs 32 TUs for reaching the target $C_{L,target}$ value. The convergence histories of the optimization runs are presented in Fig. 4.7.

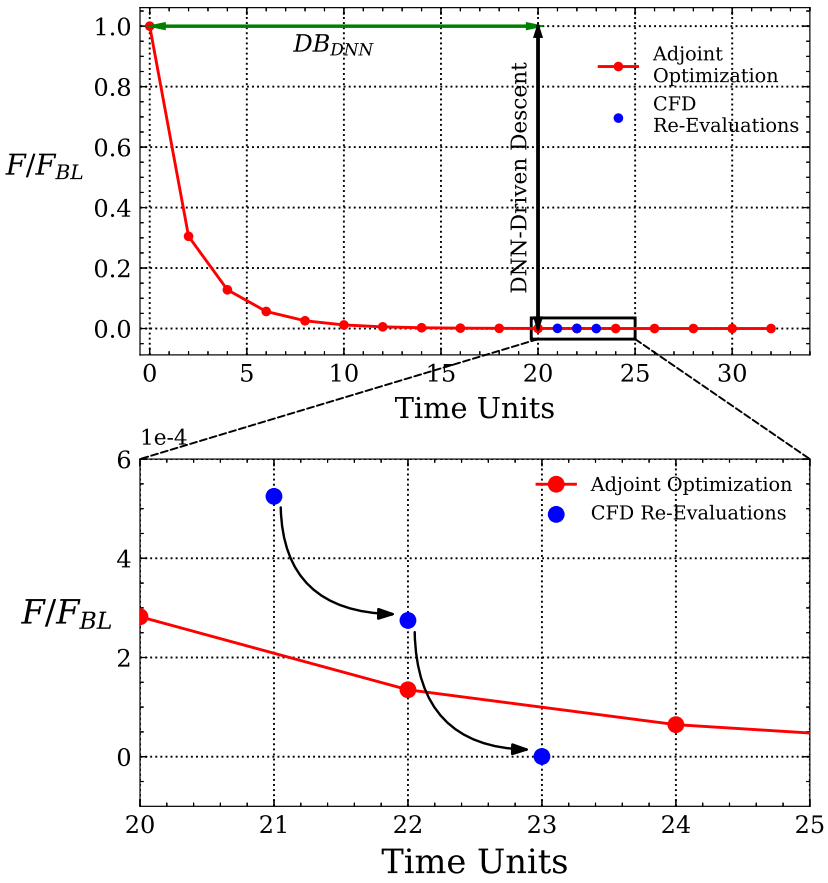


Figure 4.7: *Problem I: (Top) Convergence history of the optimization runs based on the adjoint method (red) and the differentiated DNN (black). Solutions of the DNN-based optimization which are re-evaluated on the CFD tool are shown in filled blue circles. (Bottom) Close-up view of the previous curve between TU* 20 *and* 25*.*

In this problem, the overall cost of the optimization is expected to be small, as one CFD run in a 2D inviscid flow is, in general, computationally inexpensive. The gain on the total cost of the proposed optimization method would be more apparent in 3D cases, as well as turbulent or even unsteady problems. Overall, the DNN-based optimization is by $\sim$31% less expensive than the adjoint-based run and resulted in an even better solution.

4.5 Comparison of the Optimized Geometries

The optimized airfoil shapes that resulted from the two optimization runs are compared in Fig. 4.8; the curvature on both the suction and pressure side of the optimized airfoils was modifyed, in order to increase C_L. The Mach number fields around the optimized airfoils are compared with the baseline airfoil; overall, the flow speed increased (pressure decreased) over the suction side of the optimized airfoils, in order to match the target lift coefficient $C_{L,target}$.

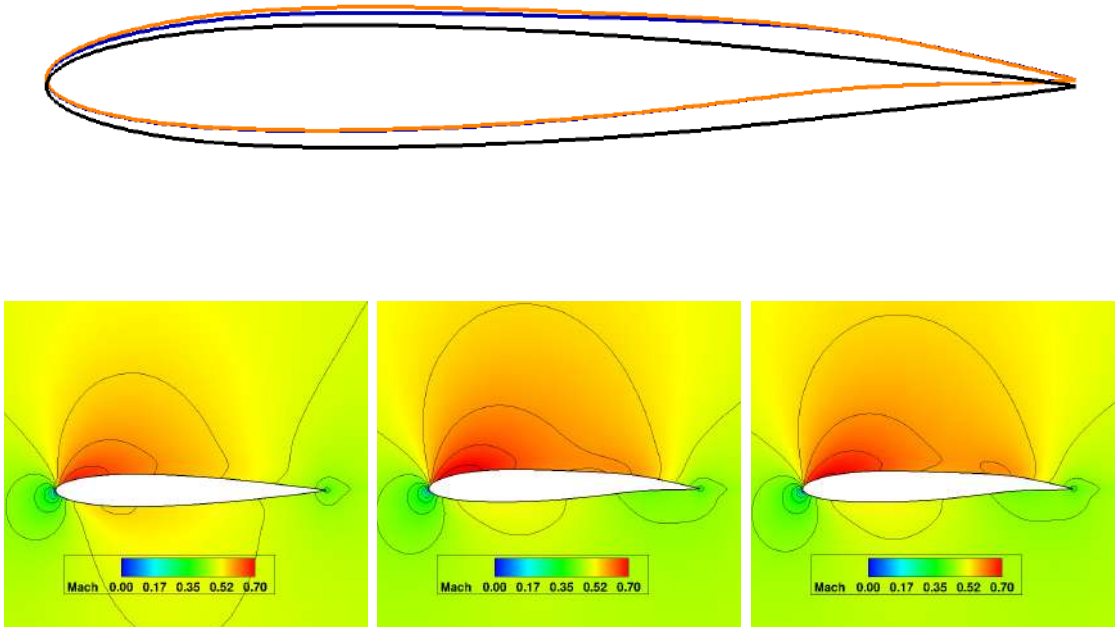


Figure 4.8: *Problem I: (Top) Shape of the baseline (black) and optimized airfoils based on the adjoint method (blue) and the differentiated DNN (red). (Bottom) Mach number fields for the baseline (left) and optimized airfoil resulted from the adjoint method (center) and the differentiated DNN (right).*

Chapter 5

Problem II: Gradient-Based Optimization of an S-Bend Duct with Laminar Flow

5.1 Introduction

Problem II deals with the shape optimization of an S-bend duct in order to minimize the total pressure losses between the duct's inlet and outlet domain. Problem II differs from Problem I since this is an internal (rather than external) aerodynamic case, it has a differernt flow model (viscous flow) and a different objective function. A DNN is assesed and trained on the generated $\mathrm{DB_{DNN}}$ geometries in order to predict the total pressure losses of the duct, when taking as input the y-coordinates of the lattice control points that parameterize its shape. The trained DNN is then used as a surrogate in the gradient-based shape optimization of the baseline duct and the results of the DNN-driven optimization are compared with the outcome of an adjoint-based optimization. Since the DNN-driven descent is of negligible cost, herein a number of individual optimization runs are carried out, starting from each generated sample in the $\mathrm{DB_{DNN}}$. This alternative approach is proposed in order to demonstrate another capability of the DNN-driven algorithm, as a promising way of overcoming the limitation of the gradient-based optimization methods, in which the optimal solutions highly depend on the initialization point.

5.2 Flow Conditions, Mesh and Shape Parameterization

The re-design of the S-bend duct aims at the minimization of the mass-averaged total pressure losses between the inlet (I) and the outlet (O). The objective to minimize is

$$F = \frac{\int\limits_{S_I} p_t \rho v_n dS + \int\limits_{S_O} p_t \rho v_n dS}{\int\limits_{S_I} \rho v_n dS} \tag{5.1}$$

where p_t, v_n are the total pressure and the normal velocity pointing outwards to the CFD domain (this is why in the numerator of Eq. 5.1, the sum, rather than the difference of two integrals appears). F stands for the losses occuring in the flow due to the viscous effects. The flow is laminar with $Re = 1.84 \cdot 10^4$ (Reynolds number based on the duct width) and inlet velocity $U = 20$ m/s. The Mach Number and the total pressure losses fields of the flow for the baseline geometry are shown in Fig. 5.1. The maximum contributions to F are computed near the walls of the S-shaped part of the duct as expected.

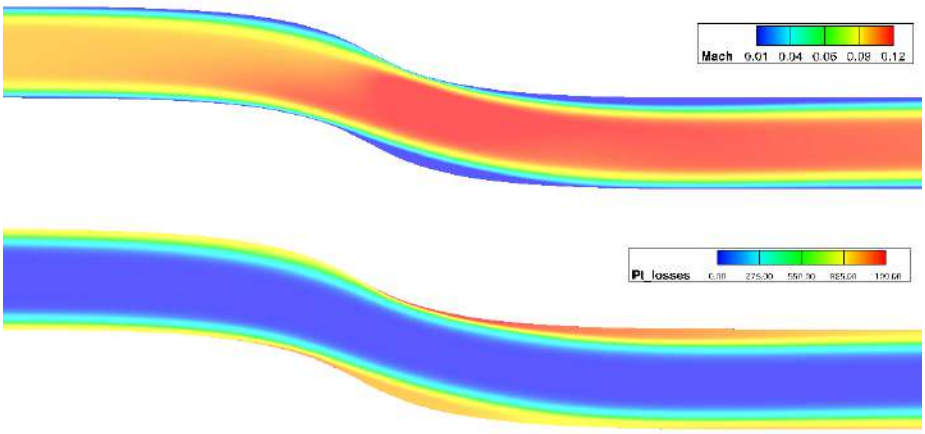


Figure 5.1: *Problem II: (Top) Mach Number Field of the baseline geometry. (Bottom) Total Pressure Losses Fields of the baseline geometry.*

A structured CFD mesh of $\sim 90K$ nodes is generated in order to simulate the flow inside the duct. The duct shape is parameterized using a 8×9 NURBS lattice, Fig. 5.2. 20, (out of the 72) control points are allowed to move in the y direction, yielding $N = 20$ design variables (20 inputs to the DNN) in total. The design variables are

allowed to change within the $\pm 0.05c$ around their initial values, so as to avoid the overlapping of the lattice lines.

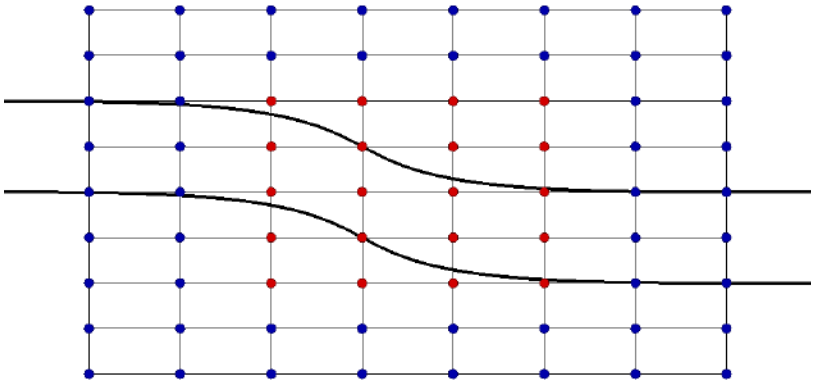


Figure 5.2: *Problem II: Control points of the volumetric NURBS control lattice, parameterizing the S-bend duct. Blue points are kept fixed, whereas red ones can be displaced in the y direction.*

5.3 DNN Configuration and Training

A DB_{DNN} of 50 duct geometries is used to train the DNN and 20% of the generated patterns is splitted to form the (fixed) validation set. In comparison to Problem I, herein a larger DB_{DNN} is constructed, since there are more design variables and viscocity is also on board. The DNN gets the $[50 \times 20]$ tensor of the y coordinates of the control points of all samples as input and computes the $[50 \times 1]$ tensor of the F values (Eq. 5.1). As in Problem I, the model's configuration was decided after comparing various hyperparameter combinations, and is made of 7 layers with $32, 64, 128, 256, 128, 64, 32$ neurons; the GELU activation function is used for all hidden layers and the sigmoid for the output one. Since the sigmoid activation is selected for the output layer, the minimum value for normalizing the training patterns is selected by a percentage lower that the minimum value ecountered in the samples, herein by 10% lower. After configuring the DNN's architecture the rest of its hyperparameters are tuned; The MSE is used as the loss function and Adam as the optimizer with a learning rate of 0.001. Again, the sufficient number of epochs to train the model must be decided, thus, a starting number of 2000 epochs is selected.

As shown in Fig. 5.3 the training loss rapidly converges, however the validation loss remains fixed to a high value throughout the whole training process. This is due to the fact that the DNN model predictions are validated on a fixed set of data that is not adequate for generalization. To overcome this limitation, the technique of cross-validation (or rotation estimation) is used, in which different portions of the DB_{DNN} data are used to validate and train the model on different iterations, during

the training process. Herein, 10 iterations of 200 epochs are carried out, where the DNN predictions in each iteration are validated on a different set of data. The DNN predictions of F when using fixed and shuffled training-validation sets are compared in Fig. 5.4.

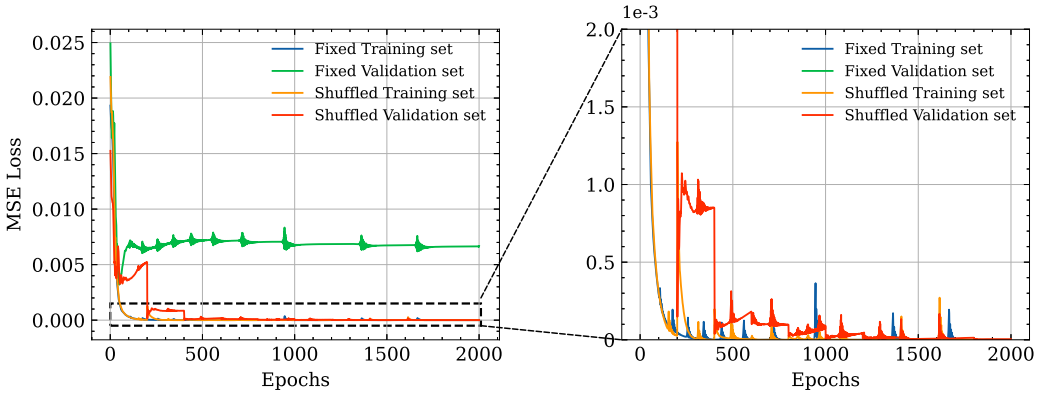


Figure 5.3: *Problem II: (Left) The convergence of the training (blue) and validation (green) loss in the case of fixed training-validation sets is compared with the convergence of the training (orange) and validation (red) loss in the case of shuffled sets. (Right) A scaled view of the losses convergence. In this view, the decrease in both the training and validation losses after each iteration is observed.*

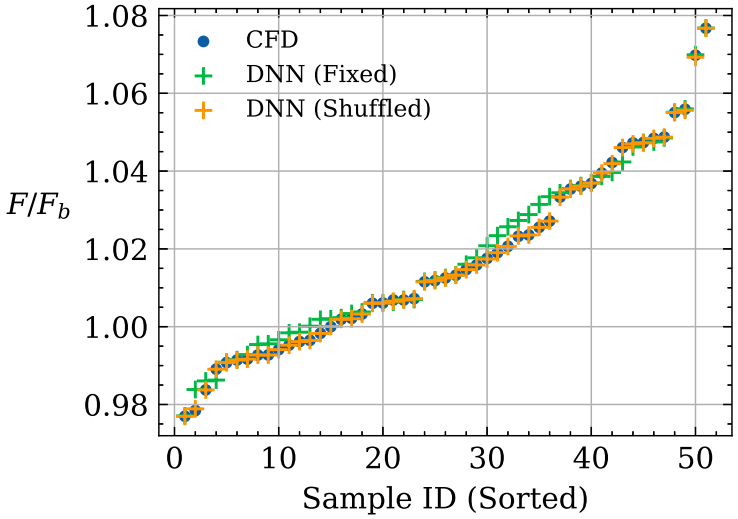


Figure 5.4: *Problem II: The F/F_b ratio values (blue) for each sampled geometry in the DB_{DNN} (sorted), where F_b is computed at the baseline geometry, are compared with the predicted values of the DNN when using fixed (green) and shuffled (orange) training and validation sets respectively. The improvement in the predictions accuracy in the case of cross-validation is observed.*

The percentage MAE metric in both cases is computed and demonstated in Fig. 5.5; the mean MAE value of the validation set in the case of fixed sets is 0.46%, while it

decreases significantly when using the cross-validation technique in the value of 0.007 % . The sensitivity derivatives of F w.r.t. the design variables are computed for both cases and presented in Fig. 5.6. Small discrepancies are observed for both validation methods, however the computed derivatives when using the cross-validation technique match better the reference adjoint ones, verifying again the effectiveness of the method.

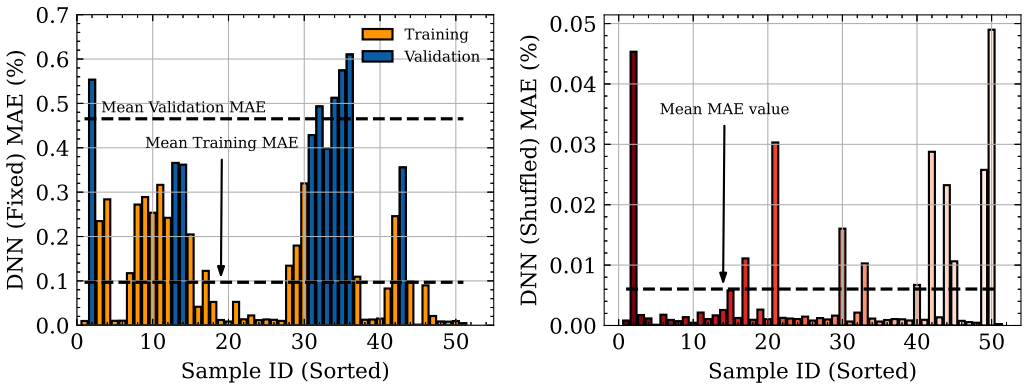


Figure 5.5: *Problem II: (Left) The computed percentage MAE metric for each sample in both the training (orange) and validation (blue) sets in the case that they remain fixed during training. (Right) The percentage MAE metric of the DB_{DNN} samples when the cross-validation technique is used. In this case, the samples used for training and validating the DNN cannot be distinguished as they change after each iteration. In this case, the MAE values are significantly smaller, verifying the effectiveness of shuffling the training and validation data during the training of the model.*

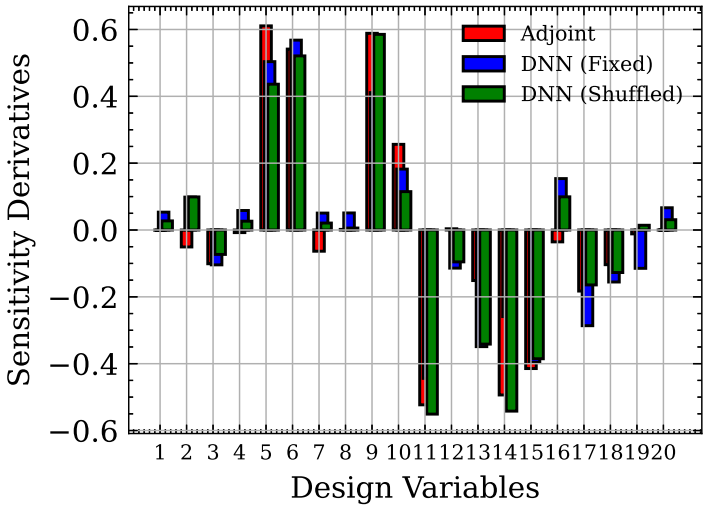


Figure 5.6: *Problem II: SDs of F (w.r.t. the design variables) for the baseline geometry computed with Adjoint (red) and derived from the DNN's differentiation when using fixed (blue) and shuffled (green) training-validation sets.*

5.4 The DNN - Driven Optimization Run

Since the descent phase of the DNN-driven optimization algorithm is of negligible cost, it was decided to perform optimization runs starting from all sampled geometries forming the $\mathrm{DB_{DNN}}$, i.e. 50 runs in total (Fig. 5.7). Though this is not what this work generally proposes, such a decision was made since it allows an exhaustive exploitation of the design space and showcases the appearance of many local minima in this kind of problems.

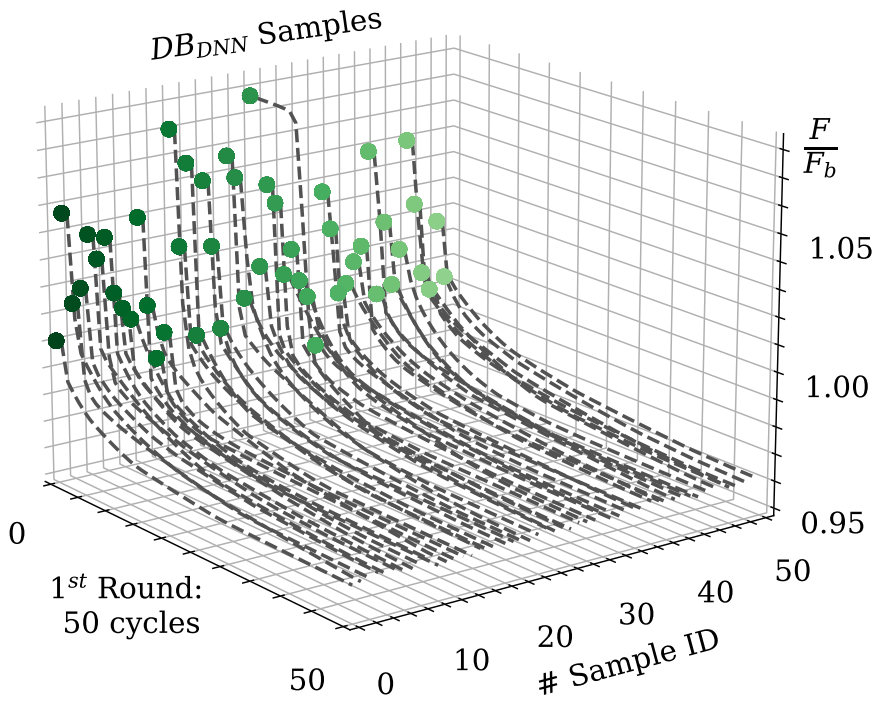


Figure 5.7: *Problem II: Representation of the 50 DNN-Driven optimization runs starting from each sample in the DB_{DNN}. The convergence of F normalized with F_b, is different for each run, resulting in* 50 *different 'optimal' solutions.*

Upon completion of the 50 optimization runs, the designer may decide which of the "optimized" solutions should undergo a CFD-based re-evaluation, at the additional cost of one TU each. Herein, it is decided to re-evaluate 10% of the 50 "optimized" solutions, i.e. the top 5 of them. These are added to the $\mathrm{DB_{DNN}}$, the DNN is re-trained and a second optimization round starts. The outcomes of the 5 DNN-based runs, each of which based on an updated (re-trained) DNN, result in new "optimized" solutions that are re-evaluated on the CFD tool and appended to the $\mathrm{DB_{DNN}}$. Two re-trainings of the DNN proved sufficient to obtain a DNN prediction accuracy less than 0.5%. At the end of this round, only the best among the five "optimized" solutions are re-evaluated, resulting in a reduction in F by 4.6%

compared to the baseline geometry. The overall cost of the DNN-based optimization is 61 TUs, consisting of: 50 TUs to generate the $\mathrm{DB_{DNN}}$, 10 TUs ($=2\times5$) to evaluate the 5 top "optimized" solutions at the end of each cycle and, finally, 1 TU for the evaluation of the final "optimized" geometry on the CFD code. For comparison, an adjoint-based optimization is also performed. The optimization loop results in the same reduction in F compared to the baseline geometry and requires 30 cycles till convergence, at the cost of 60 TUs. The overall DNN-driven optimization and the adjoint based optimization are compared in Fig. 5.8.

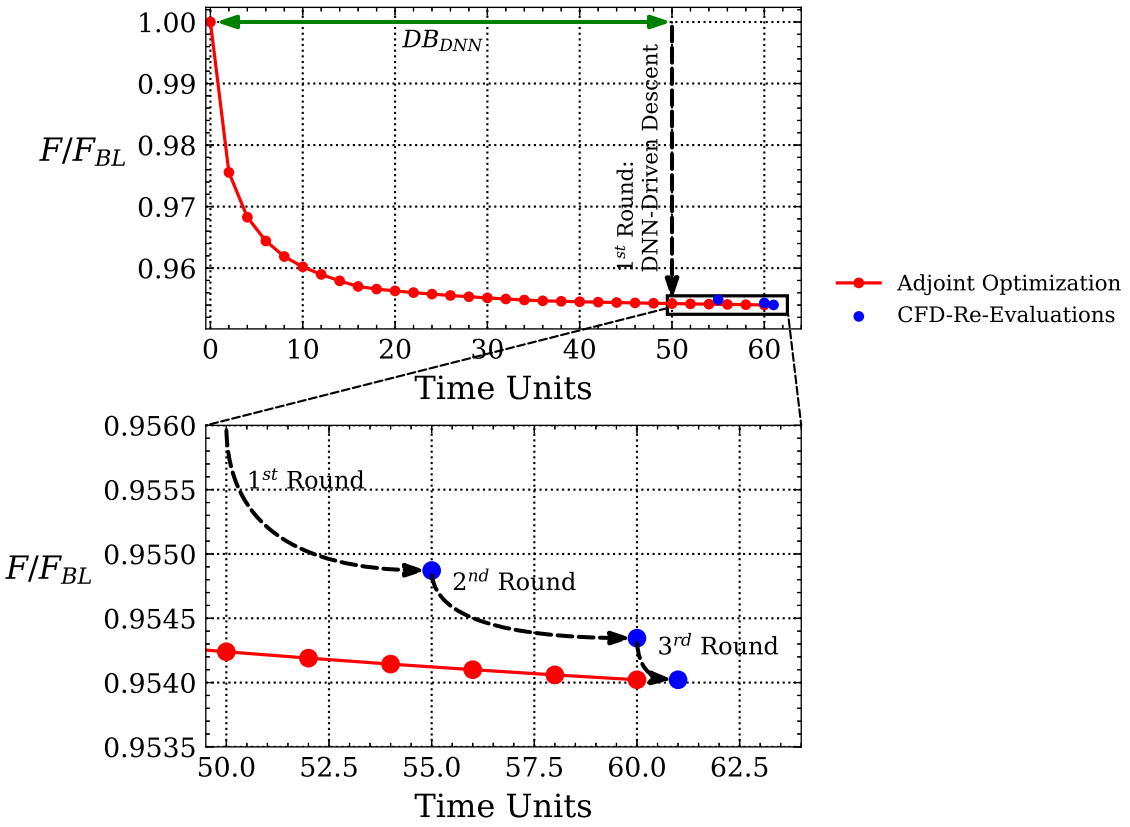


Figure 5.8: *Problem II: (Top) Convergence history of the optimization runs based on the adjoint method (red) and the differentiated DNN (black). Solutions of the DNN-based optimization which are re-evaluated on the CFD tool are shown in filled blue circles. (Bottom) Close-up view of the previous curve, in order to make a clear comparison between the adjoint curve and the three solutions "optimized" by the DNN-driven run and re-evaluated on the CFD code.*

In Problem II, achieving high accuracy on the DNN's predictions and sensitivities was a more difficult task, in comparison to Problem I. Thus, the complexity of the problem (type of the flow, geometry parameterization and selection of the number of design variables) determines the efficancy of the DNN's training and consequently, its performance during the optimization. To improve the accuracy of both the DNN's predictions and sensitivities, a larger training $\mathrm{DB_{DNN}}$ could be

used. Overall, the proposed method had the same performance as the adjoint-based optimization, resulting in the same reduction in F at (almost) the same computational cost. The proposed method also allows multiple optimization runs to be carried out simultaneously, due to the negligible cost of the DNN-driven descent. This capability is promising and useful for two reasons: First, for exploring multiple solutions in problems with many local minima and secondly, for overcoming the limitation of the gradient-based optimization, in which the outcome highly depends on the initialization point.

5.5 Comparison of the Optimized Geometries

The duct shapes optimized using adjoint and the DNN-assisted method are compared with the baseline geometry in Fig. 5.9. The total pressure losses field for the baseline, and the optimized ducts by the two methods are presented in Fig. 5.10. It is clear that the optimization changed the lower side of the duct in order to avoid a small (incipient) separation, shown as a red spot in the baseline geometry. This red spot develops as a narrow red path that reaches the domain exit. The total pressure losses distributions on the inlet and outlet domains of the baseline and optimized ducts are presented in Fig. 5.11. The distributions are computed on the vertical (y) distance at each domain, that ranges from 0 to the duct's diameter (from the duct's lower side in each domain until its upper side), herein $D = 0.2[m]$.

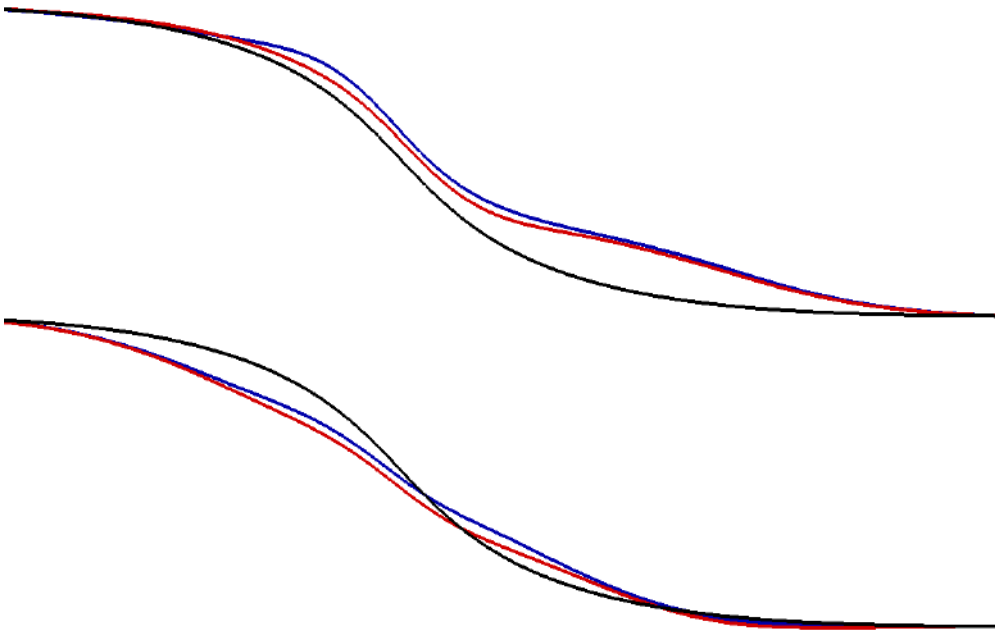


Figure 5.9: *Problem II: Shape of the baseline (black), the adjoint-based (red) and the DNN-based (blue) optimized ducts. Axes not in scale (x:y=1:2).*

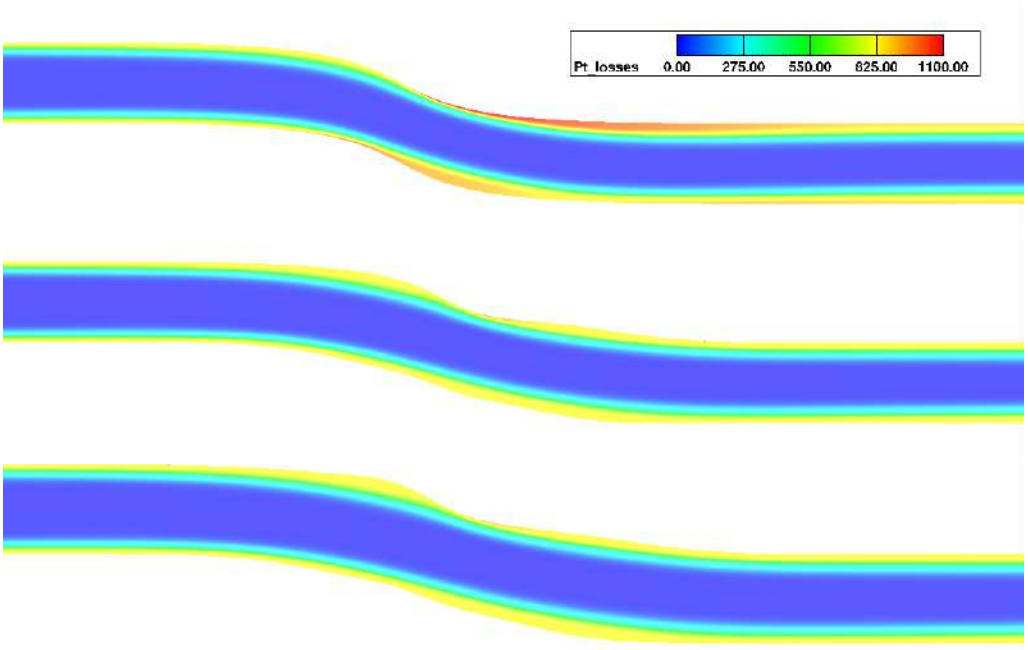


Figure 5.10: *Problem II: Total pressure losses for the baseline (top) , the optimized by the adjoint method (center) and the optimized by the proposed DNN-driven method (bottom) geometries.*

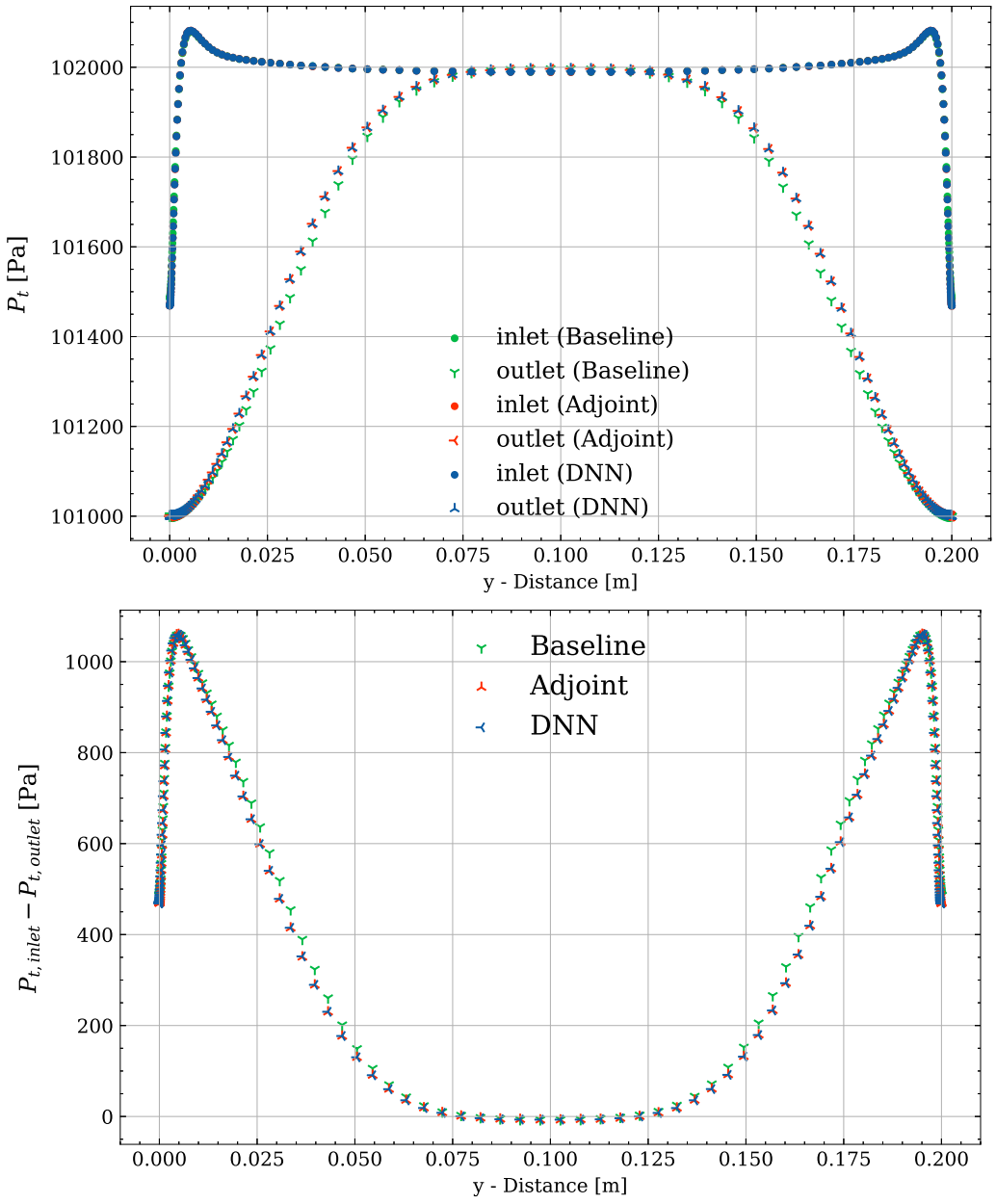


Figure 5.11: *Problem II: (Top) Total pressure distribution at the duct's inlet and outlet domains for the baseline and the optimized geometries using the adjoint and the proposed DNN-driven methods. (Bottom) Total pressure difference between the inlet and outlet distributions for the same duct geometries.*

Chapter 6

Problem III: Turbulent Flow Around an Airfoil

6.1 Introduction

The application of the proposed algorithm on the shape optimization of a the Low-Speed airfoil S8052 is presented as Problem III. The flow is turbulent and the optimization's goal is to re-design the initial airfoil's shape in order to minimize the drag coefficient while matching a, user-defined, target lift coefficient. When performing the optimization, the two objectives are expressed as two individual terms in a common objective function. Two distinguished DNN models are trained on the generated patterns in the DB_{DNN}, in order to predict the values of the two coefficients respectively. Both models take as input the y-coordinates of the lattice box's control points that parameterize the airfoil's shape. Once trained, they are used to drive the airfoil's shape optimization, where each model provides the predictions and sensitivity derivatives of the lift and drag coefficient. As in the previous problems, the effectiveness and outcome of the DNN-driven optimization are compared with the outcome of an adjoint-based optimization.

6.2 Flow Conditions, Mesh and Shape Parameterization

The flow around the S8052 low-speed airfoil is turbulent with $Re = 5 \cdot 10^5$ (Reynolds number based on the airfoil chord). The free-stream Mach number and flow angle are equal to $M_\infty = 0.021$ and $\alpha_\infty = 10°$, respectively. A C-type mesh with $\sim 3.8K$ nodes is used, shown in Fig. 6.1. The farfield boundaries of the computational domain are located about 10 chords away from the airfoil. The mesh lattice lines around the airfoil start from a distance vertical to the airfoil's chord around $\sim 10^{-6}$ [m] in order to ensure that the boundary layer region is included. For the simulation, turbulence is modeled according to the Spalart - Allmaras turbulence model. The flow conditions of the baseline geometrty are depicted in Fig. 6.2. The turbulence viscocity field can be observed with greater magnitude after the 2/3 of the airfoil's chord approximately.

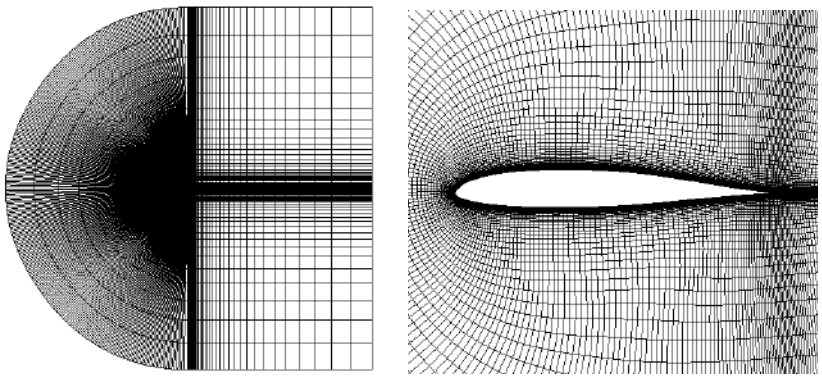


Figure 6.1: *Problem III: (Left) Mesh of the whole computational domain. (Right) Detail of the mesh surrounding the airfoil.*

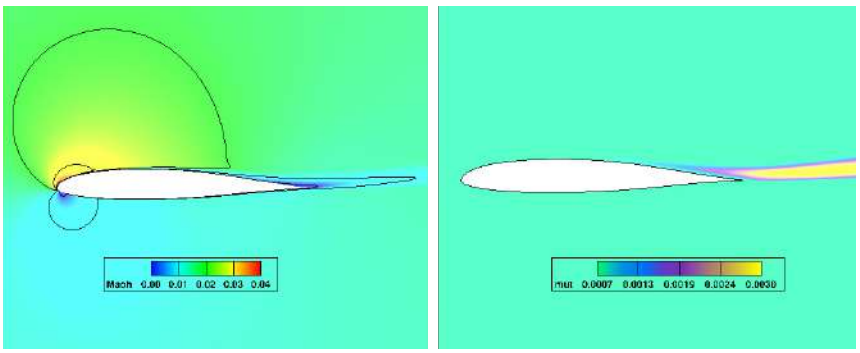


Figure 6.2: *Problem III: (Left) Mach number field for the baseline geometry. (Right) Turbulent viscocity field contour for the baseline airfoil.*

The airfoil's shape as well as part of the sourrounding mesh are controlled by the

10×7 NURBS lattice of Fig. 6.3. 16 out of the 70 control points are allowed to be displaced in the normal-to-the chord direction, resulting to N=16 design variables, in total. The design variables ($\vec{b} \in R^N$) are allowed to change within the $\pm 0.20c$ around their initial values, so as to avoid the overlapping of the lattice lines. The control points coinciding with the leading and trailing edge of the airfoil remain fixed.

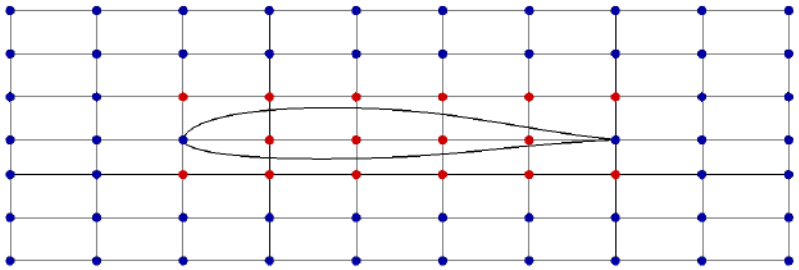


Figure 6.3: *Problem III: NURBS control lattice parameterizing the airfoil contour. Control points in blue are fixed; red ones can be displaced in the normal-to-the-chord direction.*

6.3 DNN Configuration and Training

Herein, two separate models are deployed, one for predicting the lift coefficient (C_L) of the airfoil and one for predicting the drag coefficient (C_D). C_D is defined similarly to C_L, as the drag force (D) exerted by the flow on the airfoil normalized by the dynamic pressure coefficient multiplied by the airfoil's chord (c). Because of the turbulent flow's nature and its significant impact on the drag force, it was decided to isolate the predictions of the two coefficients using two separate models, to be referred as $\mathrm{DNN_{C_L}}$ and $\mathrm{DNN_{C_D}}$ respectively. The separation of the two models will not only allow a better focus on the two target coefficients, but will also improve the accuracy of the computed gradients that are expected to differ noticeably in both scale and nature. A $\mathrm{DB_{DNN}}$ consisting of 40 airfoil geometries is used to train the two models and the cross-validation technique is again selected. Each deployed DNN gets as input the $[50\times16]$ tensor of the $\mathrm{DB_{DNN}}$ samples control points y coordinates and computes the $[50\times1]$ tensor of the C_L or C_D values respectively.

The configurations of the two models were optimized using the in-house evolutionary algorithm software, EASY. Two separate optimizations were carried out for each model regarding the architecture hyperparameters: the number of hidden layers, the number of neurons per hidden layer and the activation functions (hidden layers-output layer). The rest hyperparameters remained fixed during the optimization.

The MSE was selected as the loss function and Adam as the optimizer with a learning rate of 0.001. As in Problem II, a pattern of 10 iterations of 200 epochs was selected for cross-validating the models. The objective of each optimization was the minimization of the baseline geometry's sensitivity derivatives error on the first iteration. The error was measured as the total sum of the absolute differences values between the computed DNN sensitivities and the reference adjoint sensitivities w.r.t each DNN input. The optimized architectures of the DNN_{C_L} and DNN_{C_D} models are presented in Table 6.1.

DNN	Neurons per Hidden Layer	Activations
$\text{DNN}_{\text{C}_\text{L}}$	1024 - 64 - 4096 - 1024 - 128 -128	GELU - ReLU
$\text{DNN}_{\text{C}_\text{D}}$	32 - 32 - 4096 - 256 - 128	GELU - sigmoid

Table 6.1: *Problem III: The optimized configurations of the two models.*

The convergence of the loss function during the training of the two models is shown in Fig. 6.4. After training, the models are called to predict the C_L and C_D coefficients of each sample in the DB_{DNN} and their predictions are depicted on the C_L-C_D space, normalized with the baseline geometry's values respectively (Fig. 6.5). The percentage MAE metric value of each sample's C_L, C_D prediction is computed and presented in Fig. 6.6. As the two models were cross-validated, the training and validation sets dynamically changed after every iteration, hence, a mean MAE value of all samples in the DB_{DNN} representing both the training and validation sets is computed. In both cases, a mean MAE value of around $\sim 0.08\%$. is achieved. The sensitivity derivatives of both C_L and C_D w.r.t. the models inputs are computed for the baseline geometry after the differentiation of the two models and are compared with the reference sensitivities of when using the FDs and the adjoint method in Fig. 6.7. The computed C_L SDs satisfactory match the reference SDs, while in the computed C_D SDs discrepancies and even opposite sign at specific design variables are observed.

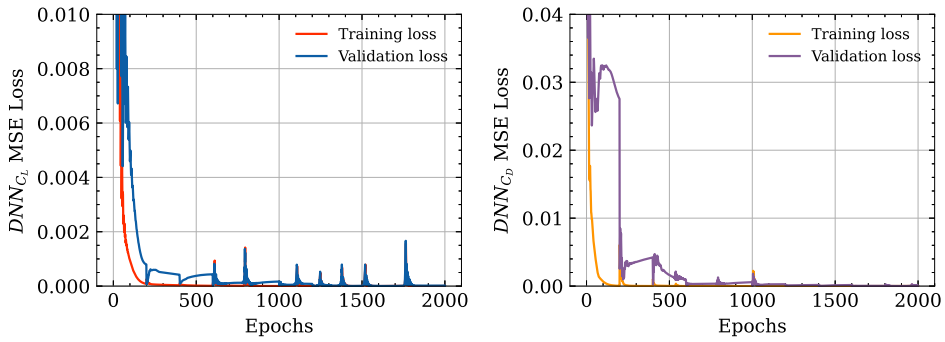


Figure 6.4: *Problem III: (Left) The convergence of the training (red) and validation (blue) loss during the trainng of the DNN_{C_L} model. At specific epochs small spikes can be observed but still of insignificant magnitude. (Right) Convergence of the training (orange) and validation (purple) loss during the trainng of the DNN_{C_D} model.*

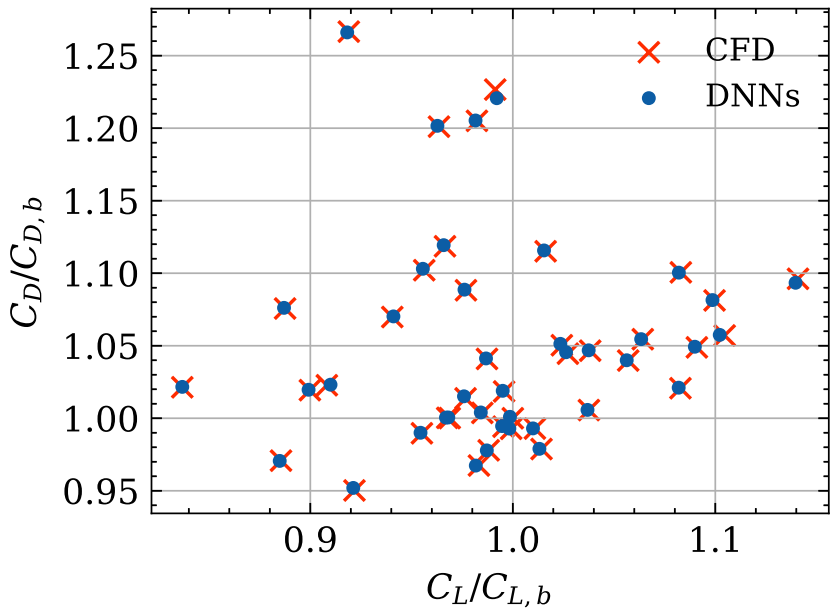


Figure 6.5: *Problem III: The CFD-evaluated values (red) of C_L, C_D for each sampled geometry in the DB_{DNN} are represented on the $C_L - C_D$ space and are compared with the predicted values (blue) of the two DNN models. The target and predicted coefficients are normalized with the baseline geometry's $C_{L,b}, C_{D,b}$ values, respectively.*

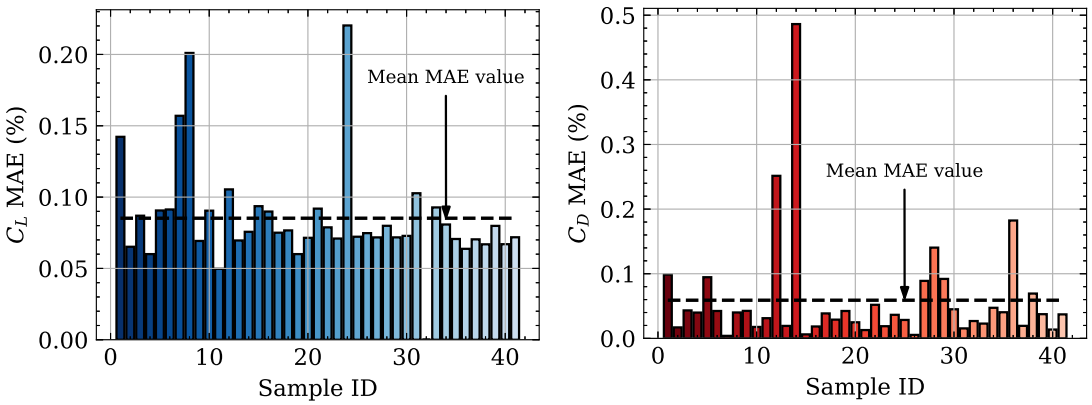


Figure 6.6: *Problem III: Left) The computed percentage MAE metric for the prediction of the C_L coefficient of each sample in the DB_{DNN}. (Right) The computed percentage MAE metric for the prediction of the C_D coefficient of each sample in the DB_{DNN}.*

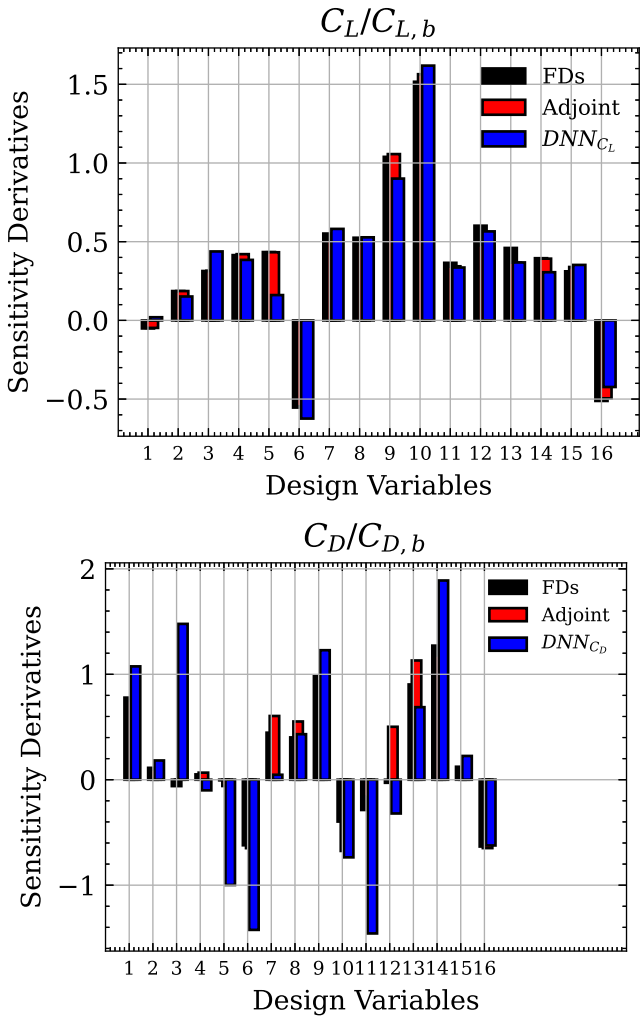


Figure 6.7: *Problem III: The SDs of C_L (Top) and C_D (Bottom) w.r.t. the design variables for the baseline geometry are compared with the computed SDs with FDs (black), adjoint (red) and derived from the DNN_{C_L}'s differentiation (blue).*

6.4 The DNN - Driven Optimization Run

The gradient-based optimization aims at designing a new airfoil in which the C_D is minimized, while the C_L coefficient's value has not decreased over the 10% of the baseline's value. Thus, a $C_{L,target}$ equal to $0.9C_{L,b}$ is set and the objective function (to be minimized) is

$$F = C_D + \frac{1}{2}\left(C_L - C_{L,target}\right)^2 \tag{6.1}$$

Two runs are carried out; the first run relies exclusively on the DNN, while the

second one on PUMA and its adjoint solver. Once the DNN-based optimization run converges, the "optimized" solution is re-evaluated on PUMA. As shown in Fig. 7.17, just one DNN-driven descent followed by one CFD re-evaluation was sufficient to reach the optimal solution. Given that the cost of a DNN-based optimization as well as that of the DNN training is practically negligible, the optimization turnaround time is 41 TUs. This includes the cost to form the $\mathrm{DB_{DNN}}$ (40 TUs) and the cost of the one CFD based re-evaluation. On the other hand, the adjoint-based run needs 50 TUs to converge. In Fig. 6.9, the C_L and C_D values of the DNN-driven and the adjoint-based optimization solutions are placed on the $C_L - C_D$ space along with the $\mathrm{DB_{DNN}}$ samples used to train the two DNN models for comparison.

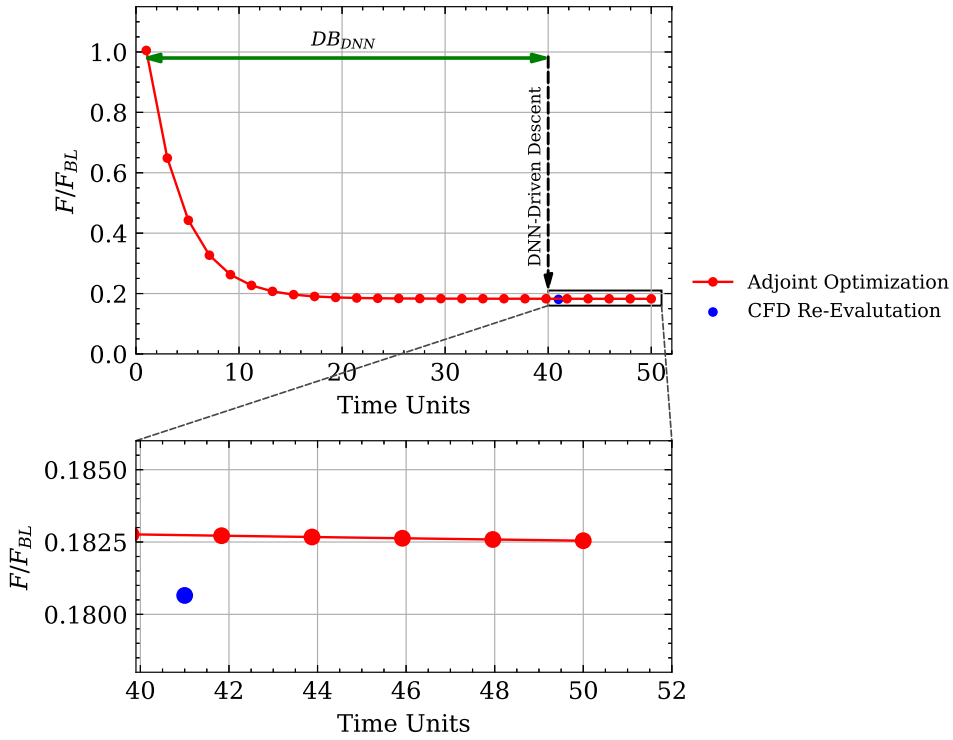


Figure 6.8: *Problem III: (Top) Convergence history of the optimization runs based on the adjoint method (red) and the differentiated DNN (black). The solution of the DNN-based optimization which is re-evaluated on the CFD tool is shown in filled blue circle. (Bottom) Close-up view of the previous curve.*

Even though Problem III deals with turbulent flow, the effectiveness of the proposed optimization method is again evidenced. Herein, the DNN-based optimization is by $\sim$20% less expensive than the adjoint-based run and results to a 4% reduction in C_D. The adjoint-based optimization achieves a 3% reduction in C_D. As observed from Fig. 6.9, the solution of the DNN-driven optimization (after evaluated on the CFD tool) has a lower C_D value than the minimum C_D value of the training samples in the $\mathrm{DB_{DNN}}$, while it matches the $C_{L,target}$ with a deviaiton of 0.1% from the target. The solution resulting from the adjoint-based optimization matches better the $C_{L,target}$

value (deviation of 0.01%), in comparison to the DNN-driven optimization's solution, that decreased F by decreasing the airfoil's C_D value.

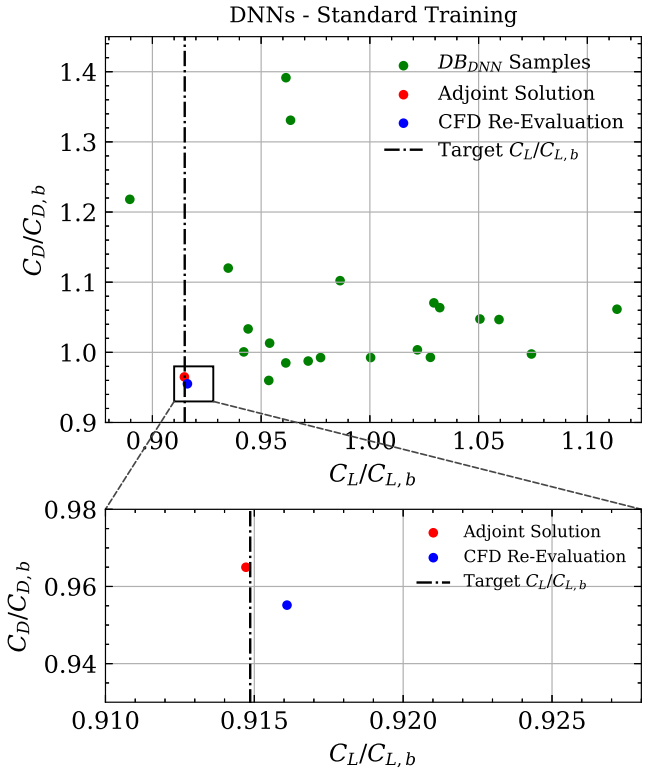


Figure 6.9: *Problem III: (Top) Representation of the DB_{DNN} samples (green), the adjoint-based optimization's solution (red) and the DNN-driven optimization's solution (blue) on the $C_L - C_D$ space. The DNN - driven optimization's solution is placed outside the DB_{DNN} samples area, verifying the effectiveness of the proposed algorithm. (Bottom) Close - up view on the two solutions.*

6.5 Comparison of the Optimized Geometries, Mach Number and Turbulent Viscosity Fields

The optimized airfoil shapes that resulted from the two optimization runs are compared in Fig. 6.10. The Mach number field around the optimized airfoils are compared with the baseline airfoil in Fig. 6.11; overall, the flow speed increased (pressure decreased) over the suction side of the optimized airfoils, in order to match the target lift coefficient $C_{L,target}$. The turbulent viscocity fields of the baseline and optimized geometries are compared in Fig. 6.12.

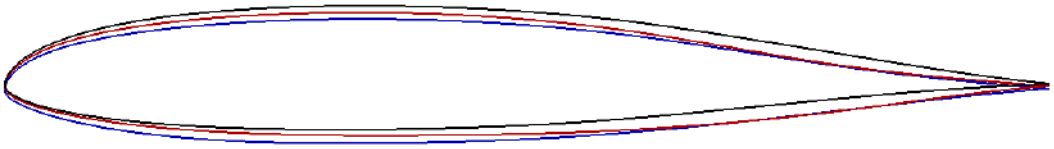


Figure 6.10: *Problem III: Shape of the baseline (black) and optimized airfoils based on the adjoint method (blue) and the differentiated DNN (red).*

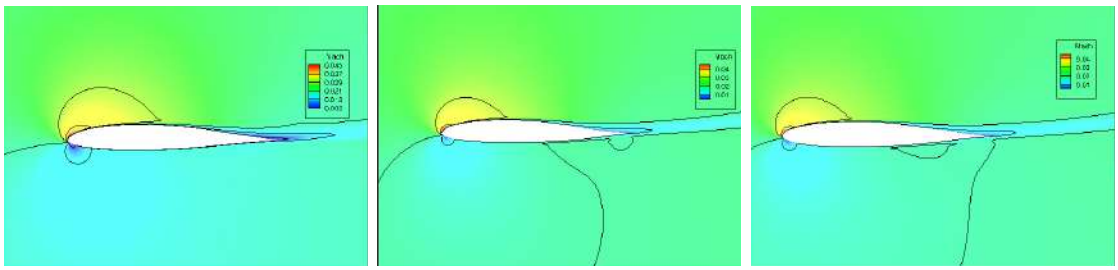


Figure 6.11: *Problem III: Mach number field for the baseline (left) and optimized airfoil resulted from the adjoint method (center) and the differentiated DNN (right)*

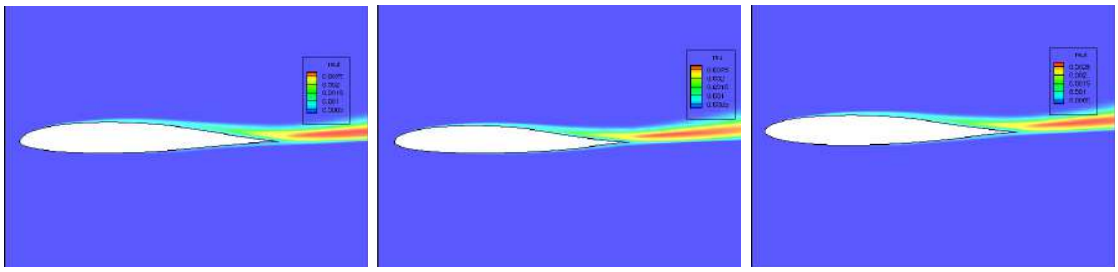


Figure 6.12: *Problem III: Turbulent viscocity field for the baseline (left) and optimized airfoil resulted from the adjoint method (center) and the differentiated DNN (right)*

6.6 Proposals for Improving the DNN Predictions and Gradient Accuracy

At the case of the S8052 airfoil and at all the other demonstrated problems, the $\mathrm{DB_{DNN}}$ was kept as small as possible in order to avoid the increase in the overall optimization cost. However, training the DNNs on a small $\mathrm{DB_{DNN}}$ can lead to several limitations, such as overfitting, generalization difficulty and high variance, mainly due to the data scarcity. A parametric study on the $\mathrm{DB_{DNN}}$ size is conducted in order to show the accuracy changes in both the DNN_{C_L} and DNN_{C_D} predictions and computed gradients. Along with the initial $\mathrm{DB_{DNN}}$ that contained 40 different

airfoil geometries, three additional databases are constructed consisting of 100, 150 and 250 patterns respectively. The DNN_{C_L} and DNN_{C_D} models are trained upon each $\mathrm{DB_{DNN}}$ and then are evaluated on 10 test geometries not included in any of the databases. The computed values of C_L, C_D of one of the 10 samples in the test set are compared with the reference CFD-based values in Table 6.2. Next, the two models are differentated in order to compute the C_L, C_D SDs of the same test geometry. The comparison of the SDs is shown in Fig. 6.13.

	C_L	C_D
CFD evaluation	20.621508	33.581847
	DNN_{C_L}	DNN_{C_D}
$DB_{\mathrm{DNN}} - 40$	20.742136	32.813233
$DB_{\mathrm{DNN}} - 100$	20.656996	33.803564
$DB_{\mathrm{DNN}} - 150$	20.599524	33.636081
$DB_{\mathrm{DNN}} - 250$	20.632137	33.586472

Table 6.2: *Problem III: Table containing the CFD-evaluated C_L, C_D values of the test geometry. The reference values are compared with the predicted values of the DNN_{C_L} and DNN_{C_D} models, when trained on a DB_{DNN} consisting of* 40, 100, 150 *and* 250 *airfoil geometries respectively.*

The accuracy of the predictions is improved when the $\mathrm{DB_{DNN}}$ size increases until a specific number of samples (herein 100 samples, almost 5 times the number of the design variables). Increasing further the size of the $\mathrm{DB_{DNN}}$ doesn't improve the DNNs' accuracy significantly. These observations can be verified for all other samples in the test set. The computed C_L SDs match the reference SDs for all $\mathrm{DB_{DNN}}$ sizes. On the contrary, achieving high accuracy on the C_D SDs is a more difficult task. Deviations are observed from the reference SDs, with gradual improvement as the $\mathrm{DB_{DNN}}$ size increases (again, the improvement is evident until a specific number of entries).

Even though the predictions and the computed gradients of both the DNN_{C_L} and DNN_{C_D} models are more accurate when trained on a larger $\mathrm{DB_{DNN}}$, the optimization can be succesfully driven by the DNNs even when trained on a small $\mathrm{DB_{DNN}}$ (almost 2 times the number of the design variables). As in all gradient-based algorithms used in optimization, it is possible to converge even when the non-exact gradients values are used, as long as their sign is correct. As a result, the challenge in the proposed DNN-driven optimization is to select the $\mathrm{DB_{DNN}}$'s size such that to strike the right balance between the accuracy of the DNN (on both its predictions and its gradient) and the overall cost of the optimization. It is preferable to to keep a small $\mathrm{DB_{DNN}}$ size for training the DNNs, even though discrepancies might occur on their predicitons or gradients, and then improve their accuracy during the optimization process, by performing CFD re-evaluations of the 'optimal' solutions and re-trainings, as many as needed.

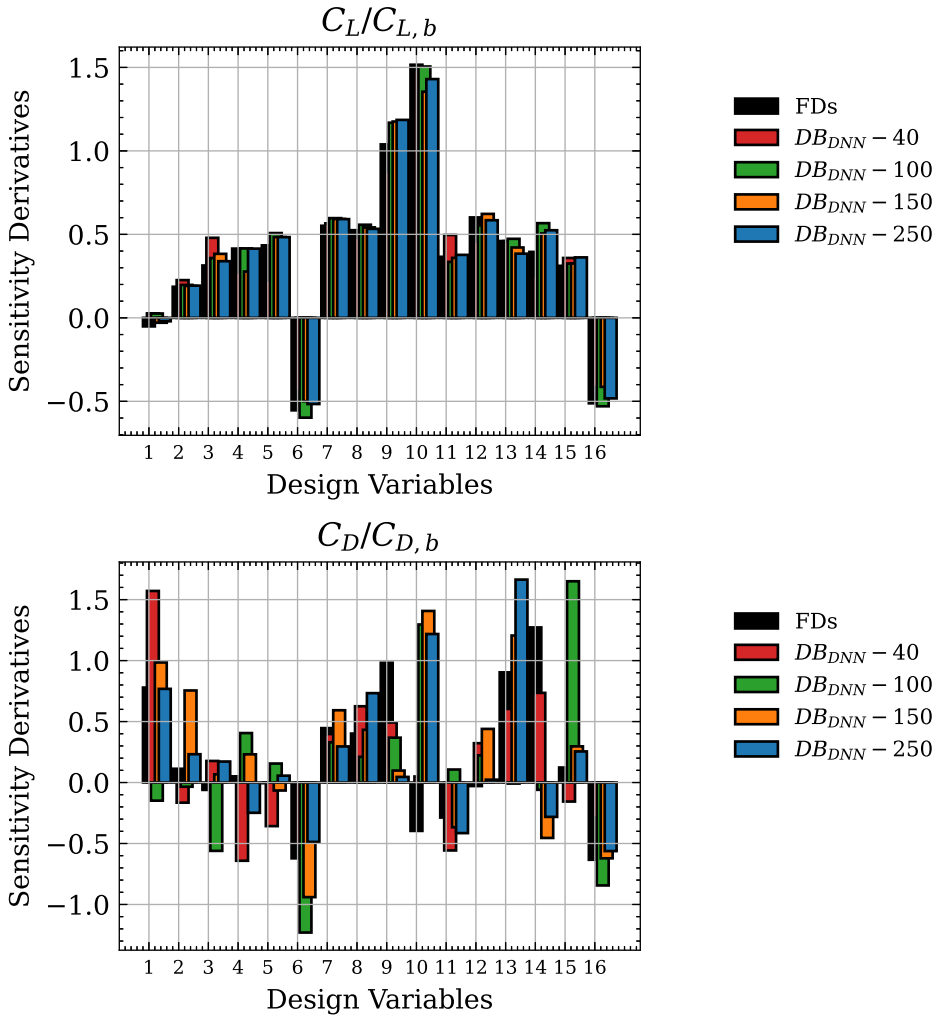


Figure 6.13: *Problem III: The SDs of C_L (Top) and C_D (Bottom) w.r.t. the design variables for the test geometry computed with FDs (black) and resulting from the DNN_{C_L} model's differentiation when trained on a DB_{DNN} consisting of* 40 *(red),* 100 *(green),* 150 *(orange) and* 250 *(blue) sample airfoils respectively.*

Another proposal for improving the accuracy of the DNN predictions and computed gradients is to include more information about the target during training, such as the target values derivatives up to a specific order. That would encode additional information about the target on the model's parameters and possibly lead to accuracy improvement. The potentials of the gradient-assisted training of DNNs is studied and demonstrated in the next Chapter.

Chapter 7

Gradient-Assisted Training of DNNs

7.1 Introduction

In this Chapter, the derivatives of the target functions (usually, the objective function) are implemented in the DNN training process in addition to the target function itself. This method aims at improving the accuracy of the computed gradients, as well as the data-efficiency and generalization capabilities of the learned approximations, by encoding additional information about the target function within the parameters of the neural network. The incorporation of the DNN gradients in the training process is presented in two variants. The first implementation is based on the Sobolev Training method of DNNs. The second, is a new idea that is based on the principles of the Hermite interpolation and thus, is referred as the Hermite Method. Both variants are tested on function-gradient approximation applications in mathematics and later extended to the approximation of the C_L, C_D coefficients of the S8052 airfoil and its sensitivity derivatives as initially presented in Chapter 6. The trained DNNs (with both variants) are used to drive the shape optimization of the same airfoil and the results are compared with the DNN-driven optimization of the standard-trained DNNs and the adjoint-based optimization.

7.2 Implementation I: The Sobolev Method

7.2.1 Sobolev Training for Deep Neural Networks

The Sobolev training method for deep neural networks was first introduced by Czarnecki [54], as a method to incorporate the target derivatives in addition to the target values while training. The method was inspired by the work of Hornik [55], which proved the universal approximation theorems for neural networks in Sobolev spaces, where distances between functions are defined both in terms of their differences in values and differences in values of their derivatives. A Sobolev Space [56] is a vector space of functions equipped with a norm that is a combination of L^p-norms of functions together with its derivatives up to a given order. Consequently, the loss function to be minimized during the Sobolev training in the case of one target output and one input variable, is defined as the total sum of the norm-differences of the DNN predictions ($\hat{y}_i$) and target values (y_i) in addition to the total sum of the norm-differences of the computed derivatives of the DNN w.r.t. each input ($D_x\hat{y}_i$) and the target derivatives ($D_x y_i$), as expressed in

$$Loss_{sobolev} = Loss_{func} + Loss_{grad} = \sum_{i=1}^{N_s} L_f^p(\hat{y}_i, y_i) + \sum_{i=1}^{N_s} L_g^p(D_x\hat{y}_i, D_x y_i) \qquad (7.1)$$

D_x is the differential operator w.r.t. the variable x and $i = 1, ..., N_s$ where N_s is the total number of samples in the $\mathrm{DB_{DNN}}$. The L_f^p, L_g^p stand for the selected L^p-norms for measuring the error of the predicted function values and the computed gradients respectively: The MAE corresponds to p = 1 and the MSE for p = 2. For multi-output regression problems in which DNNs are used to approximate more than one targets, the $Loss_{func}$ term would contain as many constituent terms as the number of targets. Similarly, in the common case of multiple DNN inputs, the $Loss_{grad}$ constituent contains one term for each partial derivative, measuring the error of the target(s) derivatives w.r.t each input. In addition, an appropriate set of weights could be applied to each term of the $Loss_{sobolev}$, so as to properly tune the overall loss during training and allow for a finer and more qualitative learning from the DNN.

Practically, the difference of a Sobolev-trained DNN (Fig. 7.1) with a standard trained-DNN, pertaining the training process, is that the gradient of the target(s) w.r.t. their input(s) is also computed during training (using external code provided by the user, based on the black-box RAD function of the TensorFlow framework) and is compared with the target/reference gradient values, by adding additional

term(s) at the loss function. Consequently, the DB_{DNN} in the Sobolev case not only contains the target(s) values, but also the values of their first derivative(s) computed at each sample. It is evident that when the Sobolev training method is applied to CFD problems, the computational cost unavoidably increases, due to the expensive computation of the required SDs. Herein, the reference SDs will be computed using the adjoint method, so as to keep the cost as small as possible.

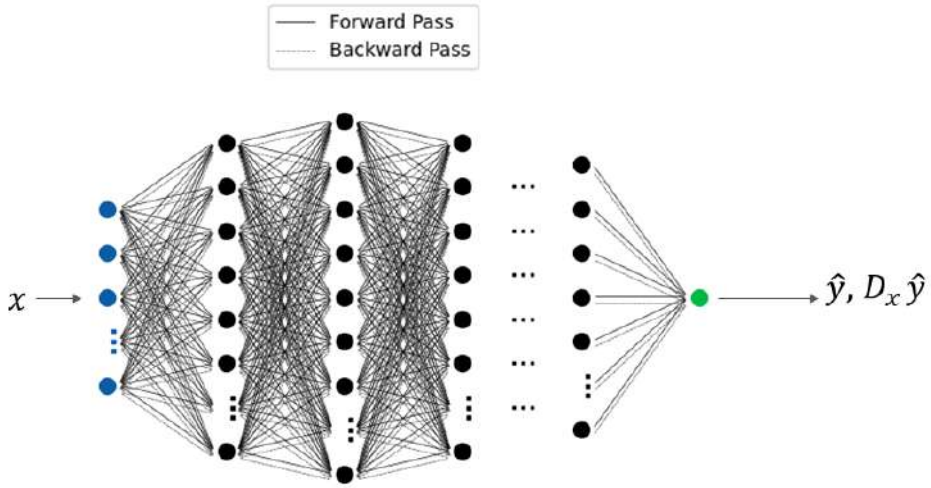


Figure 7.1: *Implementation I : DNN configuration with one input layer (blue) , user-defined hidden layers (black) and one output layer (green). The model's predictions ($\hat{y}$) are computed from the forward pass of the model and the prediction's gradient w.r.t. the x input ($D_x\hat{y}$) is computed using RAD. During training, both the predictions and their gradient are compared with their target values, respectively, using the Sobolev-type loss function.*

The empirical success of the Sobolev training method is demonstrated in a number of works, as in [57], where it was proven that an overparameterized (with more than the necessary parameters to fit the training data accurately) two-layer ReLU neural network trained on the Sobolev loss with gradient flow from random initialization can fit any given function values and any given directional derivatives, under a separation condition on the input data, and in [58], where the Sobolev loss is implemented for training thermodynamic-informed neural networks in order to gain control over the derivatives of the learned functions and derive thermodynamically consistent and interpretable elastoplasticity models that, exhibit excellent learning capacity. In [59], the direct impacts of Sobolev training on neural network surrogate models embedded in optimization problems was examined, where it was shown that sobolev trained surrogate models result in more accurate derivatives (in addition to more accurately predicting outputs), with direct benefits in gradient-based optimization.

7.2.2 Demonstration of the Sobolev Method on the Approximation of a Bi-Variate Function

The implementation of the Sobolev training method is demonstrated on the bivariate function of Chapter 3. To properly incorporate the target derivatives in the training process, it must be ensured that they are in the same scale with the derivatives resulting from the DNN's differentiation. The scale of the computed DNN derivatives depends on the transformations (scaling) applied to the generated DB_{DNN} patterns before training. At all demonstrated cases, both the DNN 's inputs and outputs are normalized within the range of [0, 1] according to their minimum and maximum values in the sampled data, as in equations

$$X^* = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{7.2}$$

$$Y^* = \frac{Y - Y_{min}}{Y_{max} - Y_{min}} \tag{7.3}$$

$$F^* = \frac{F - F_{min}}{F_{max} - F_{min}} \tag{7.4}$$

Since the DNN will be trained on the normalized data, the derivatives of F^* w.r.t. X^* and Y^* respectively will be computed. The relation between the target derivatives on the original scale and the computed DNN derivatives results from the chain rule

$$\frac{dF^*}{dX^*} = \frac{dF^*}{dF}\frac{dF}{dX}\frac{dX}{dX^*} = \frac{X_{max} - X_{min}}{F_{max} - F_{min}} \cdot \frac{dF}{dX} \tag{7.5}$$

$$\frac{dF^*}{dY^*} = \frac{dF^*}{dF}\frac{dF}{dY}\frac{dY}{dY^*} = \frac{Y_{max} - Y_{min}}{F_{max} - F_{min}} \cdot \frac{dF}{dY} \tag{7.6}$$

It is concluded that in order to properly incorporate the derivatives in the training process, each partial derivative must be multiplied with the fraction of the range of the reference input variable with the range of the target output. When the target derivatives are brought to the same scale as the DNN derivatives, they will not necessarily be in the range of [0,1], but will vary within a range that is formed by the ranges of the sampled input and output data respectively. In that case, the two loss terms $Loss_{func}$ and $Loss_{grad}$ of Eq. 7.1 will be in different scale as well. On the TensorFlow-Keras [60] framework implementation, although a common optimizer is used regarding the overall $Loss_{sobolev}$, it is possible to use different loss functions for the $Loss_{func}$ and $Loss_{grad}$ constituents or even to multiply each term with a

different weight during training. In that way, a different penalty is applyed to each constituent and the total loss function is properly tuned, allowing more qualitative learning from the DNNs.

Again, a DNN is assessed (same configuration as in Chapter 3) and trained on the values of both function F and its two partial derivatives $\frac{dF}{dX}$, $\frac{dF}{dY}$ on the $\mathrm{DB}_{\mathrm{DNN}}$. The MSE is used for both loss constituents and Adam as the optimizer with a learning rate of 0.001. Although there is a small scale difference between the target values and the derivatives, no limitations were encoutered during training, thus, no ancilary weights were applied on the two losses. After training, the predictions of F were calculated and the DNN was differentiated w.r.t X and Y respectively. The resulting predictions and derivatives are presented on Fig. 7.2 in compare with their target counterparts, as well as the predictions and computed derivatives of the standard-trained DNN respectively. The Sobolev-trained DNN achieved high accuracy in predicting F, same as in the case of the standard-trained DNN. Improvement was observed on both the computed derivatives, where their surfaces' curvature was properly captured.

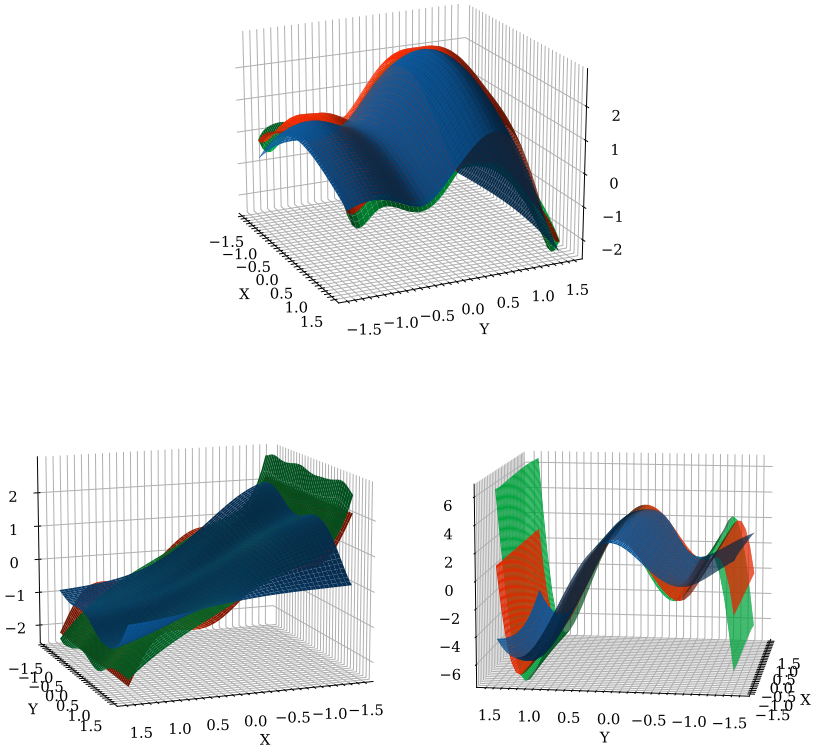


Figure 7.2: *Implementation I : (Top) The F surface predicted by the Sobolev-trained DNN (green) is compared with the predicted surface of the standard-trained DNN (blue) and the analytical surface (red). (Bottom) The same comparison is shown for the derivatives of F w.r.t. X (Left) and Y (Right).*

7.2.3 Demonstration of the Sobolev Method on Problem III

As in Problem III, two separate DNN models are trained according to the Sobolev method in order to predict the S8052 airfoil's C_L, C_D coefficients respectively. The total loss function of Eq. 7.7 consists of $N_b = 16$, in total, gradient loss constituents that are summed up and added to the function loss constituent,

$$Loss_{sobolev} = \sum_{i=1}^{N_s} \left[L_f^p(\hat{y}_i, y_i) + \sum_{j=1}^{N_b} L_g^p(D_{x_j}\hat{y}_i, D_{x_j} y_i) \right] \tag{7.7}$$

A $\mathrm{DB_{DNN}}$ of 25 airfoil geometries is generated using the LHS method in order to train the DNNs. For each sample in the $\mathrm{DB_{DNN}}$ the SDs (w.r.t. the design variables) are computed using the adjoint method. 20% of the sampled airfoils is used to cross-validate the models with the pattern of 8 iterations of 200 epochs each. Since new information about the targets is now included in the training, the previous DNN configurations are no longer guaranteed to be optimal. Two new optimizations were carried out using the EASY software in order to configure the optimal architecture of the two models. The objective of the optimization was to minimize the $Loss_{sobolev}$ on the first iteration. The rest of the models hyperparameters remained fixed during training. The MSE was used as the loss function for all constituents and Adam as the optimizer with a learning rate of 0.001. Again, no ancilary weights were applied on the individual loss terms. The optimized architectures are presented in Table 7.1.

DNN	Neurons per Hidden Layer	Activations
$\mathrm{DNN_{C_L}}$	1024 - 1024 - 2048 - 512 - 64 -4096 - 32 - 2048 -1024	ReLU - GELU
$\mathrm{DNN_{C_D}}$	4096 - 4096 - 64 - 32 - 256 - 256 - 2048 - 64	GELU - GELU

Table 7.1: *Problem III: The optimized configurations of the two models.*

The loss convergence of the two models, again to be referred as DNN_{C_L} and DNN_{C_D} respectively, is presented in Fig. 7.3. Instead of showing the total loss, it was decided to plot the function and gradient terms separately in order to isolate the convergence of each constituent. The function term in this case refers to the coefficients loss and the gradient term to the sensitivities loss. Once trained, the models are called to predict the C_L and C_D coefficients of each sample in the $\mathrm{DB_{DNN}}$ and their predictions are depicted on the C_L-C_D space, normalized with the baseline geometry's values respectively (Fig. 7.4). The percentage MAE metric value of each sample's C_L, C_D prediction is computed and presented in Fig. 7.5; A mean MAE value of around $\sim 0.25\%$ is achieved for the C_L predictions and a value of around $\sim 0.04\%$ for the C_D predictions. The sensitivity derivatives of both C_L and C_D w.r.t. the models inputs are computed for the baseline geometry after the differentiation of the two models and are compared with the sensitivities of the standard-trained DNNs and

their reference FDs and ajoint counterparts in Fig. 7.6; The computed C_L SDs satisfactory match the reference SDs, while the C_D SDs were significantly improved in compare with those of the standard-trained DNN.

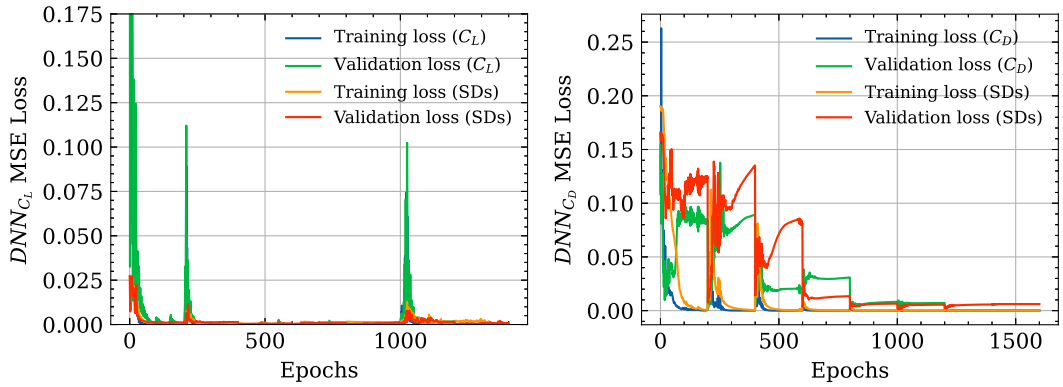


Figure 7.3: *Implementation I : The convergence of the training (blue) and validation (green) function loss terms and the convergence of the training (orange) and validation (red) gradient loss terms during the training of the DNN_{C_L} (Left) and the DNN_{C_D} (Right) models. In both cases, 1600 epochs, in total, were sufficient for all loss functions to fully converge.*

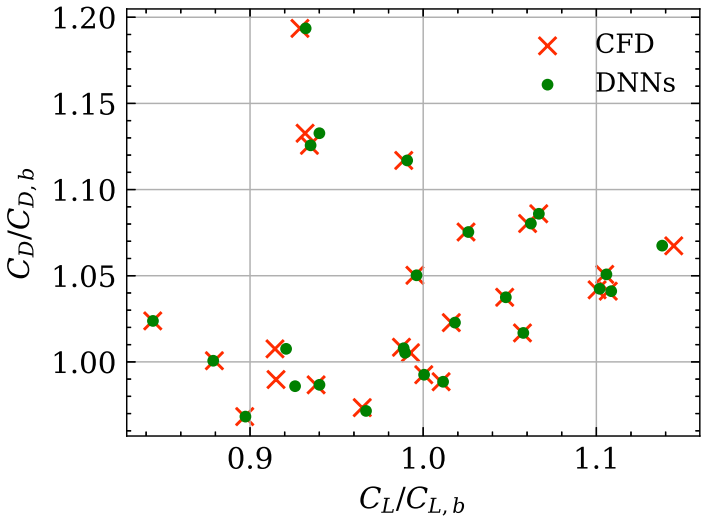


Figure 7.4: *Implementation I : The CFD-evaluated values (red) of C_L, C_D for each sampled geometry in the DB_{DNN} are represented on the $C_L - C_D$ space and are compared with the predicted values (green) of the two DNN models. The target and predicted coefficients are normalized with the baseline geometry's $C_{L,b}, C_{D,b}$ values.*

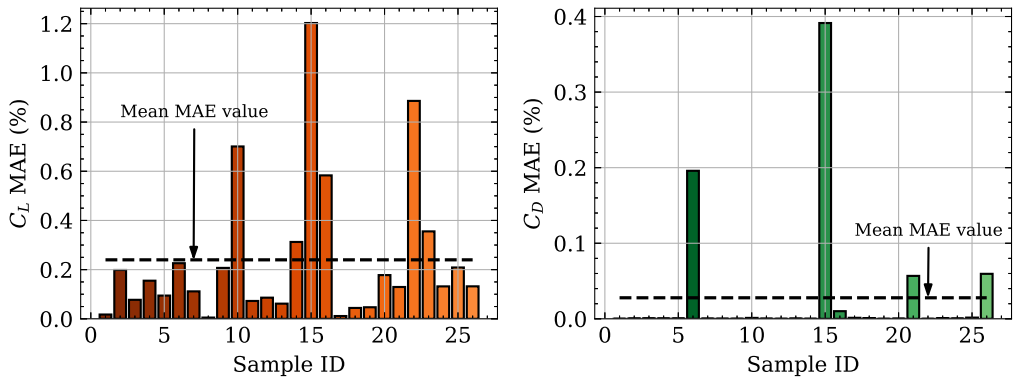


Figure 7.5: *Implementation I : The computed percentage MAE metric for the C_L (Left) and the C_D (Right) predictions of each sample in the DB_{DNN}.*

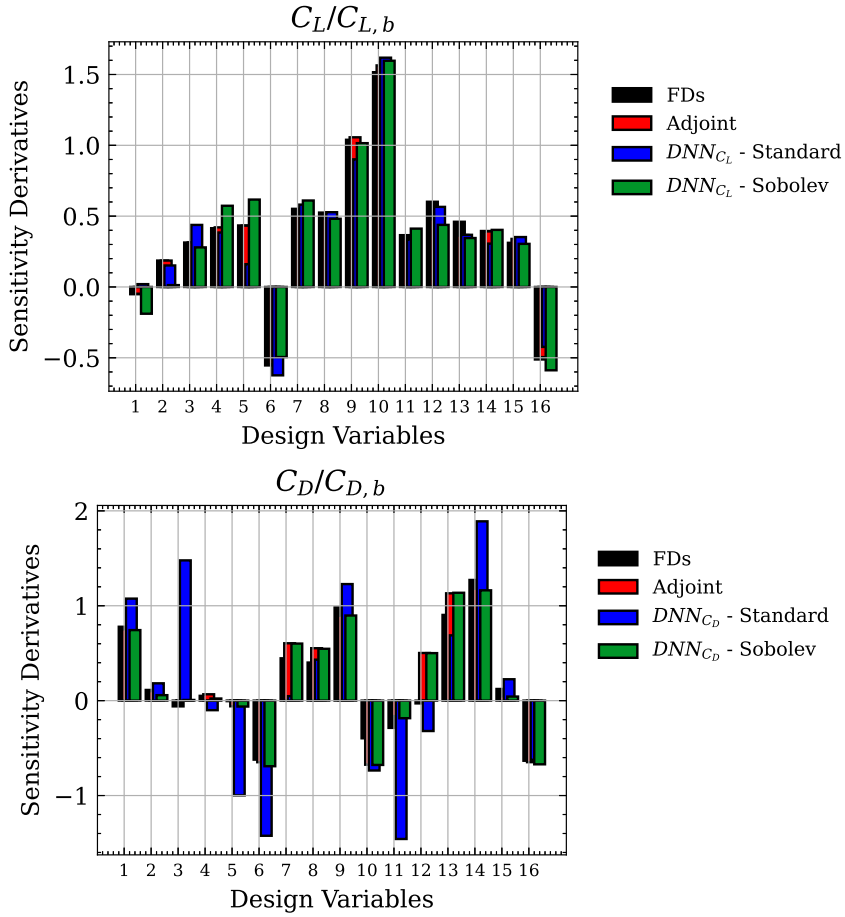


Figure 7.6: *Implementation I : SDs of C_L (Top) and C_D (Bottom) w.r.t. the design variables for the baseline geometry computed with FDs (black), Adjoint (red), the standard-trained DNN (blue) and the Sobolev-trained DNN (green).*

7.3 Implementation II: The Hermite Method

7.3.1 Hemite Interpolation

In Numerical Analysis, the Hermite Interpolation is a method of polynomial interpolation which generalizes the Lagrange interpolation [61]. The original definition refers to problems of one variable with one target output and is used in the case both the target's values and its first derivatives are known for the given sample points. Similarly to the Lagrange interpolation, the total approximation function is formed by a set of basis polynomials in order to pass through the target points and coincide with the given values of its derivatives at these points as well. Herein, two different types of (orthogonal) basis polynomials are defined for each sample, such as to disengage the interpolation of the target's values from the interpolation of the derivative values. The first type of polynomials, to be referred as H_j, where $j = 1, ..., N_s$ contribute only to the interpolation of the target values, while the second type of polynomials, to be referred as $\overline{H}_j$, contribute only to the interpolation of the derivative values. To implement this, the H_j polynomials must return 1 when computed at the x_i sample if $i = j$, where $i = 1, ..., N_s$ and 0 at any other x_i sample, while the $\overline{H}_j$ polynomials must return 0 when computed at the x_i sample. The opposite rule applies to the derivatives of the basis polynomials, H'_j and $\overline{H}'_j$, respectively. The described properties are expressed as

$$\begin{array}{lll} H_j(x_i) = \delta_j^i & , & H'_j(x_i) = 0 \\ \overline{H}_j(x_i) = 0 & , & \overline{H}'_j(x_i) = \delta_j^i \end{array} \tag{7.8}$$

The Lagrange polynomials satisfy the equation $L_j(x_i) = \delta_j^i$ and therefore are used to define the Hermite basis polynomials according to the formulas

$$\begin{array}{c} H_j(x) = (1 - 2L'_j(x_j)(x - x_j))(L_j(x))^2 \\ \overline{H}_j(x) = (x - x_j)(L_j(x))^2 \end{array} \tag{7.9}$$

The final interpolation function is a linear combination of the Hermite basis polynomials multiplyed with the target output or target derivative respectively as

$$g(x) = \sum_{j=1}^{N_s} H_j(x) y_j + \sum_{j=1}^{N_s} \overline{H}_j(x) y'_j \tag{7.10}$$

The Hermite interpolation method can be extended for the interpolation of bi-variate or even multi-variate functions and their derivatives. The basic developed concepts for the implementation of the multivariate Hermite interpolation were described in

[62], while many recent works present modern and intelligent techniques for applying the hermite interpolation on higher dimensions. In [63], new algorithms for Hermite interpolation and evaluation over finite fields of characteristic two were presented, in which the algorithms first reduced the Hermite problems to instances of the standard multipoint interpolation and evaluation problems and then solved them by existing fast algorithms. In [64], the Hermite interpolation was studied on n-dimensional non-equally spaced, rectilinear grids over a field of characteristic zero, given the values of the function at each point of the grid and the partial derivatives up to a maximum degree.

Other works propose the use of ML models in order to overcome the challenges of the Hermite interpolation in high dimensions and on irregular domains. In [65], Radial Basis Functions (RBFs) were applied as a suitable tool to high dimensional Hermite interpolation problem on irregular domains. The available derivatives information was presented using fractional order derivatives instead of integer ones and the optimal recovery conditions for the fractional Hermite interpolant were investigated and presented. In [66], the authors of PCOpt proposed a new variant of Radial Basis Function Networks (RBFNs), with enhanced capacity to approximate any input–output mapping defined by a collection of activation signals and the corresponding responses. The nonlinear mapping from the input to the hidden network units was modified by taking into account approximate values of the directional slopes of the response surface with respect to the free parameters. In [67], an adaptive self-organizing Hermite-polynomial-based neural control system was proposed, were its capability of achievieng favorable control performance in real-time systems was evidenced. The system was composed of a neural controller and a supervisor compensator, designed to eliminate the approximation error between the neural controller and the ideal feedback controller without chattering phenomena. The neural controller consisted of one input, 3 intermediate, and one output layer; The first intermediate layer, named as 'Hermite layer', was a hidden layer in which the Hermite basis polynomials were used as activation functions. The second intermediate layer, that was referred as 'reception layer', consisted of one node in which the summation of all the incoming signals was performed. Finally, the third intermediate layer was a hidden layer with a custom activation function, that was practically resposible for eliminating the error between the neural and the ideal controller.

In this Diploma Thesis, an alternative implementation method of the multivariate Hermite interpolation is proposed, using DNNs as surrogates of the Hermite basis polynomials. This method explores a new way of incorporating the target values derivatives in the training process and its capabilities in achievieng higher accuracy in both the DNN 's predictions and computed derivatives.

7.3.2 DNNs as Surrogates of the Hermite Basis Polynomials

The formula of the total approximation function of Eq. 7.10 can equivalently be re-expressed as the sum of two matrix products as

$$\begin{bmatrix} g(x_{i=1}) \\ g(x_{i=2}) \\ ... \\ g(x_{i=N_s}) \end{bmatrix} = \begin{bmatrix} H_{j=1}(x_{i=1}) & ... & H_{j=1}(x_{i=N_s}) \\ H_{j=2}(x_{i=1}) & ... & H_{j=2}(x_{i=N_s}) \\ ... & & \\ H_{j=N_s}(x_{i=1}) & ... & H_{j=N_s}(x_{j=N_s}) \end{bmatrix} \cdot \begin{bmatrix} Y(x_{i=1}) \\ Y(x_{i=2}) \\ ... \\ Y(x_{i=N_s}) \end{bmatrix} + \begin{bmatrix} \overline{H}_{j=1}(x_{i=1}) & ... & \overline{H}_{j=1}(x_{i=N_s}) \\ \overline{H}_{j=2}(x_{i=1}) & ... & \overline{H}_{j=2}(x_{i=N_s}) \\ ... & & \\ \overline{H}_{j=N_s}(x_{i=1}) & ... & \overline{H}_{j=N_s}(x_{i=N_s}) \end{bmatrix} \cdot \begin{bmatrix} Y'(x_{i=1}) \\ Y'(x_{i=2}) \\ ... \\ Y'(x_{i=N_s}) \end{bmatrix} \tag{7.11}$$

The first matrix contains the values of the N_s, in total, H_j basis polynomials where $j = 1, ..., N_s$, computed at each x_i sample where $i = 1, ..., N_s$ and equals to the identity matrix in order to satisfy the Hermite interpolation's conditions of Eq. 7.8. The second matrix contains the values of the $\overline{H}_j$ basis polynomials computed at each x_i sample and therefore equals to the zero matrix. The two $[N_s \times N_s]$ matrices are multiplied with the $[N_s \times 1]$ vectors containing the target values and the derivative values respectively, and the products of the two multiplications are added to compute the total interpolation values.

At the proposed implementation, the two sets of basis polynomials $H_j, \overline{H}_j$, are modeled by two distinguished DNN branches, to be referred as B_{func} and B_{grad} respectively. In this case, the values of the two matrices of Eq. 7.11 are no longer provided by polynomials with an explicit mathematical expression, but are computed by two separate DNN branches in N_s discrete points. The number of neurons at that layer in each branch is set equal to N_s in order to provide the $[N_s \times N_s]$ tensor when computed at all samples. These layers are to be referred as 'Basis layers'. Next, the output matrices of the two individual Basis layers are multiplied with the vectors containing the target values and the derivative values respectively, and the two multiplication products are again added in order to form the total DNN approximation function. Instead of performing the two multiplications as a post-process procedure, it was decided to add one more layer after the Basis layer of each branch, with fixed, non-trainable weights. The weights of the B_{func}'s output layer are assigned with the values of the target, while the weights of the B_{grad}'s output layer are assigned with the values of the derivatives. In addition, it can be decided whether an activation function will be used on these output layers. The proposed architecture is shown in Fig. 7.7. In summary, the B_{func}, B_{grad} branches consist of the following layers:

- **Input Layer**: Both branches have an input layer with the dimension of the model's inputs.
- **Hidden Layers**: B_{func} and B_{grad} have a user-defined number of hidden layers that is not necessarily equal for the two branches.
- **Basis Layer**: The Basis layer follows the hidden layers. The number of nodes is set equal to the number of samples (N_s) so as to output the $[N_s \times Ns]$ tensor when computed at all samples included in the $\mathrm{DB_{DNN}}$. The resulting kernels, to referred as filters, replace the tensors containing the values of the analytical Hermite basis polynomials $H_j(x)$ and $\overline{H}_j(x)$, respectively, of Eq. **??**.
- **Output Layer**: The output layer of B_{func} has the dimensionality of the targets. Herein, one target is considered. The samples' target values (y_j) are set as the weights of the output layer and remain fixed during the training process. The output layer of B_{grad} has the dimensionality of the input. In case of one variable, the samples' derivatives values (y'_j) are set as the output layer's weights that, similarly, remain fixed during training. The outcomes of the B_{func} and B_{grad} output layers are added in order to form the total DNN interpolation function $g(x)$.

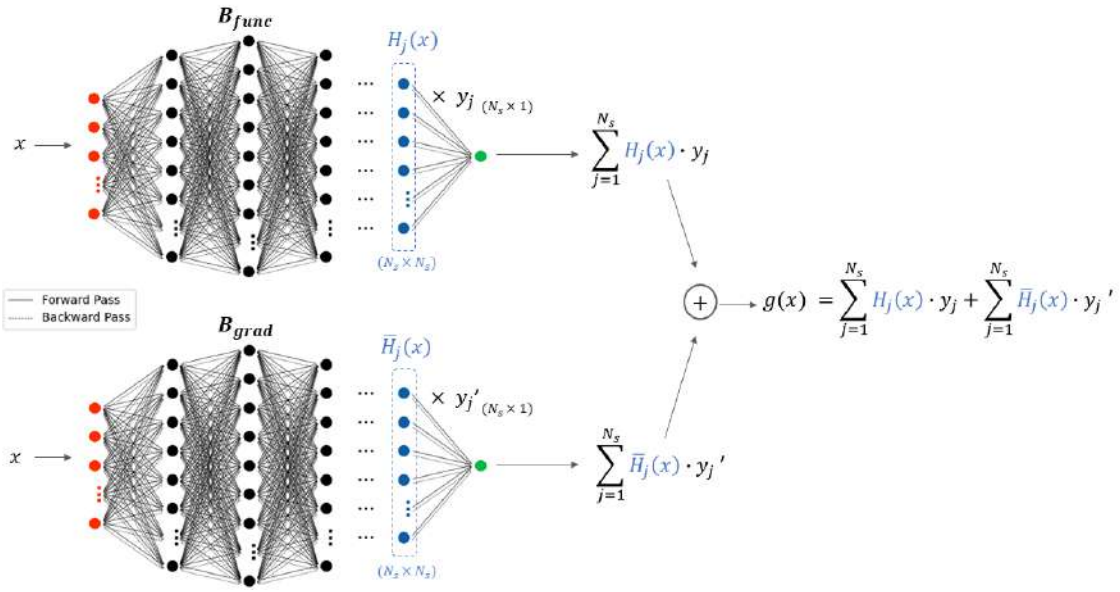


Figure 7.7: *Implementation II: Demonstration of the proposed DNN architecture for one input and one target. The model's input (x) sequentally passes through the input layer (red), hidden layers (black) and Basis layer (blue) of the B_{func} and B_{grad} branches, respectively. A final output layer (green) follows, where the use of an activation function is also possible. The outcomes of the two output layers are added, forming the total DNN interpolation function $g(x)$.*

After configuring the structure of the surrogate model, the next step is to enforce the

Hermite interpolation's conditions. One variation would be to train the proposed DNN such that the outputs of the two Basis layers equal the identity and zero matrix, respectively. In that case, however, no information about the target would pass on the model's parameters. Herein, it was decided to apply the conditions on the total approximation function, by using a Sobolev-type loss function,

$$Loss_{hermite} = Loss_{func} + Loss_{grad} = \sum_{i=1}^{N_s} L_f^p(\hat{g}_i, y_i) + \sum_{i=1}^{N_s} L_g^p(D_x\hat{g}_i, D_x y_i) \quad (7.12)$$

The DNN is trained such that the output approximation function g (even though it consists of two different function-weighted and gradient-weighted terms) to match the target function and, after differentiated, to match the target derivative. Since no restrictions are applied to the output matrices of the Basis layers during training, they will not necessarily be equal to the identity and zero matrices, as in standard Hermite interpolation. Herein, the contribution of each branch will be determined during training. The advantage of this method is that it can easily be extended to higher dimensions. For example, in the case of n (input) variables and one target, the implementation would only require to set the number of neurons on the output layer of B_{grad} branch equal to n, so as to compute the n partial derivatives of the target w.r.t. each input. Again, a Sobolev-type function, that would consist of one function loss term and n constituent gradient loss terms, can be used.

7.3.3 Demonstrarion of the Hermite Method on the Approximation of Uni-Variate and Bi-Variate Functions

The proposed Hermite method is demonstrated in the uni-variate function

$$F_1(x) = sin(3x) + cos(x^3), \qquad x \in [-0.8, 0.8] \quad (7.13)$$

To train the Hermite-structured DNN, 20 samples of x are generated using the LHS method. The configuration of each branch in the surrogate model resulted from a trial-and-error procedure. The B_{func} branch consists of 4 hidden layers with 32-64-32-32 neurons each and the B_{grad} branch consists of 5 hidden layers with 32-64-64-32-32 neurons. In both branches the GELU activation is selected for the hidden layers and it is decided to activate the output layers with the Linear activation. The training patterns are normalized in the range of $[0, 1]$, as in the Sobolev case, and the cross-validation technique is again used to validate the model during training. The MSE loss is used as the loss function for all constituents of the $Loss_{hermite}$ and Adam as the optimizer with a learning rate of 0.001. The predicted values of F_1 from

the trained model as well as the contribution functions of each branch are shown in Fig. 7.8.

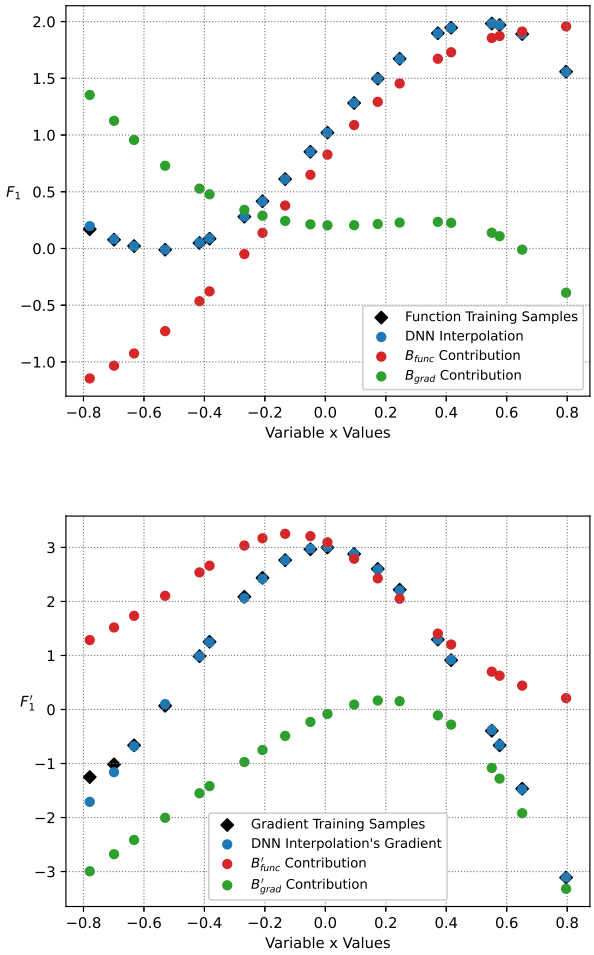


Figure 7.8: *Implementation II : (Left) The computed values of* F_1 *at the* 20 *samples (black) are compared with the predicted values of the surrogate model (blue), resulting from the combination of the* B_{func} *contribution function (red) and the* B_{grad} *contribution function (green). (Right) The derivative of* F_1 *computed at the samples values (black) are compared with the derivative values resulting from the differentiation of the DNNś approximation function (blue). The computed DNN derivative results from the combination of the* B_{func}*'s differentiation (red) and the* B_{grad}*'s differentiation.*

In comparison with the standard Hermite interpolation, herein the two DNN branches contribute equally and complementary to each other, resulting in two different constituent functions that are combined in order to form the total approximation function. To better visualize the contribution of each branch, the filters of the standard and the surrogate interpolation method are compared in Fig. 7.9. The standard H_j and $\overline{H}_j$ filters are point-cloud representations of the identity and zero matrix respectively, as expected. For each sample there is a unique contribution, that is

due to the H_j filter only. At the DNN implementation, two types of contributions are observed. First, both the H_j, $\overline{H}_j$ filters contribute to the total interpolation. Second, there is no longer a disunion between the samples, but the total outcome results from the linear combination of all samples. The value computed at each sample is influenced by a different set of samples values each time, resulting in a 'blended' point-cloud distribution. For comparison, the gradients of the standard and surrogate filters that contribute when computing the gradient of the total interpolation, are presented in Fig. 7.10. Again, the different contribution patterns can be observed between the two implementations, in addition to a scale difference that is due to the use of activation functions in the case of the DNN implementation.

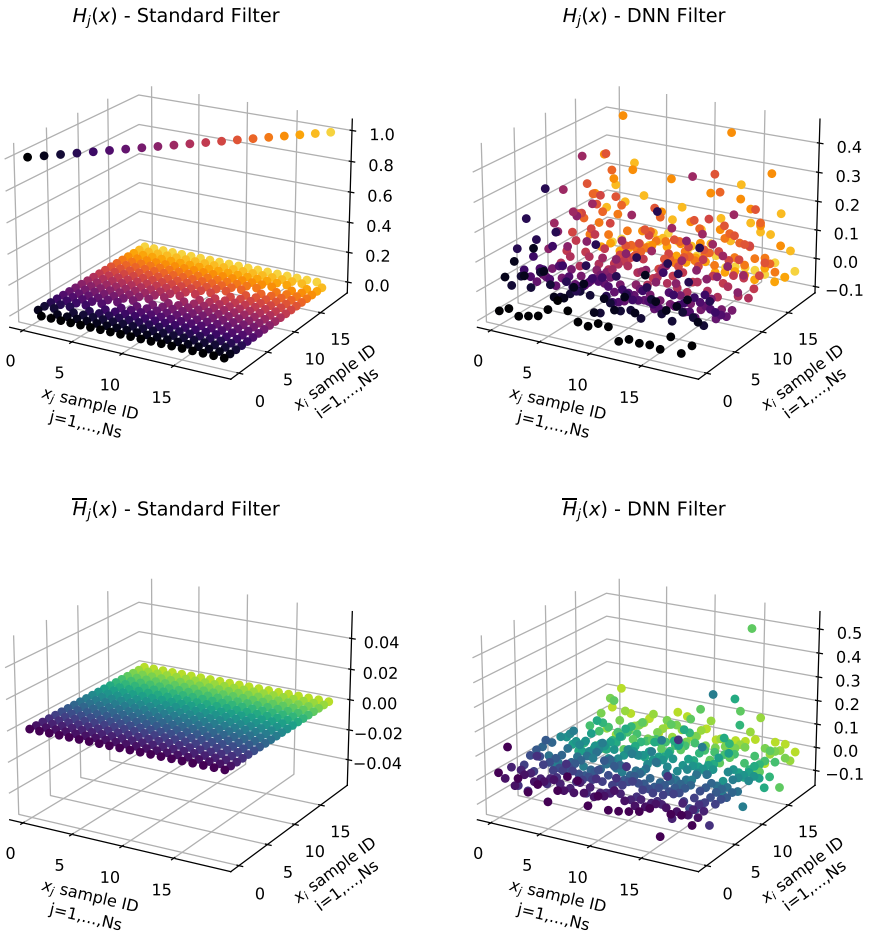


Figure 7.9: *Implementation II : Comparison of the H_j (Top) and $\overline{H}_j$ (Bottom) filters of the standard Hermite interpolation (Left) and the DNN variant (Right).*

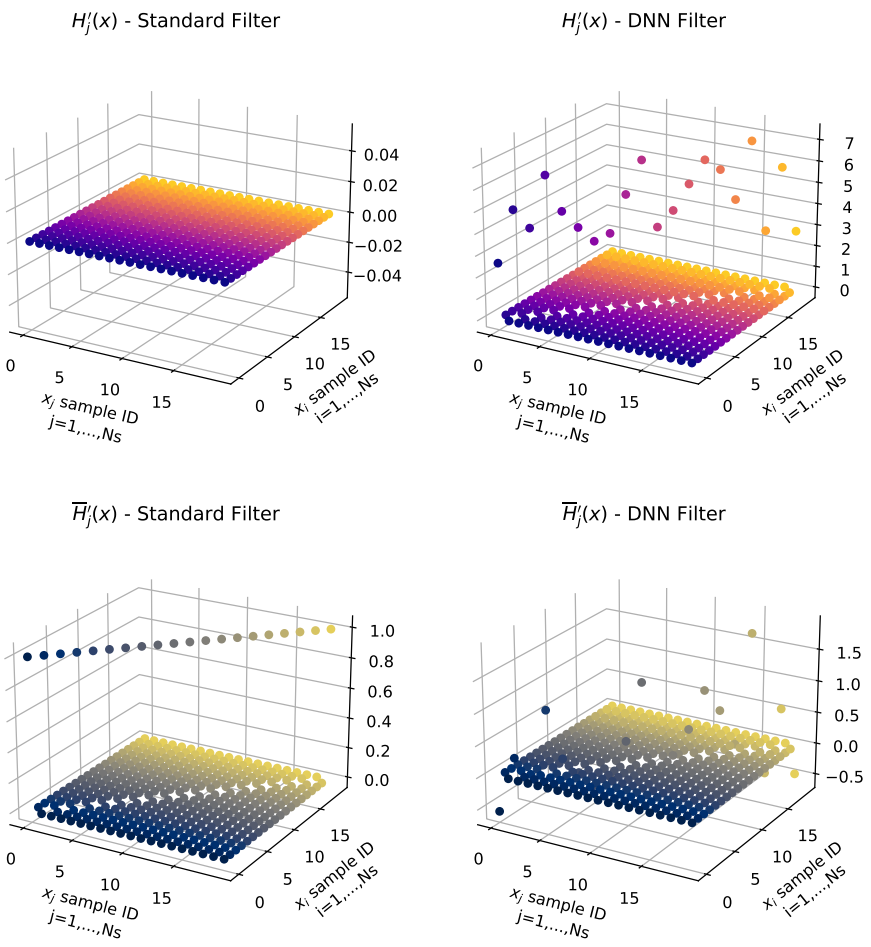


Figure 7.10: *Implementation II : Comparison of the H'_j (Top) and $\overline{H}'_j$ (Bottom) filters of the standard Hermite interpolation (Left) and the DNN variant (Right).*

The Hermite interpolation function is unique, meaning that there is only one possible function that can exactly interpolate a given set of data points and their associated derivatives using the Hermite interpolation method. Herein, the total DNN approximation function highly depends on the model's architecture and mainly, on the used activation functions. To exploit the possible outcomes of the surrogate model, a parametric study is conducted on the previous example, focusing on the activations used on the two output layers of the model; The DNN is re-trained with a different combination of the GELU, tanh, sigmoid and Linear activations on its output layers, and, the case in which no activations are used is also considered. Each training always results in one of the three patterns shown in Fig. 7.11; The two branches can either contribute complementary to each other and have an equivalent influence on the total interpolation, or, the total approximation function can be, almost exclusively, learned by one of the two branches and then the other branch undertakes the role of "filling the gap" whenever it is needed.

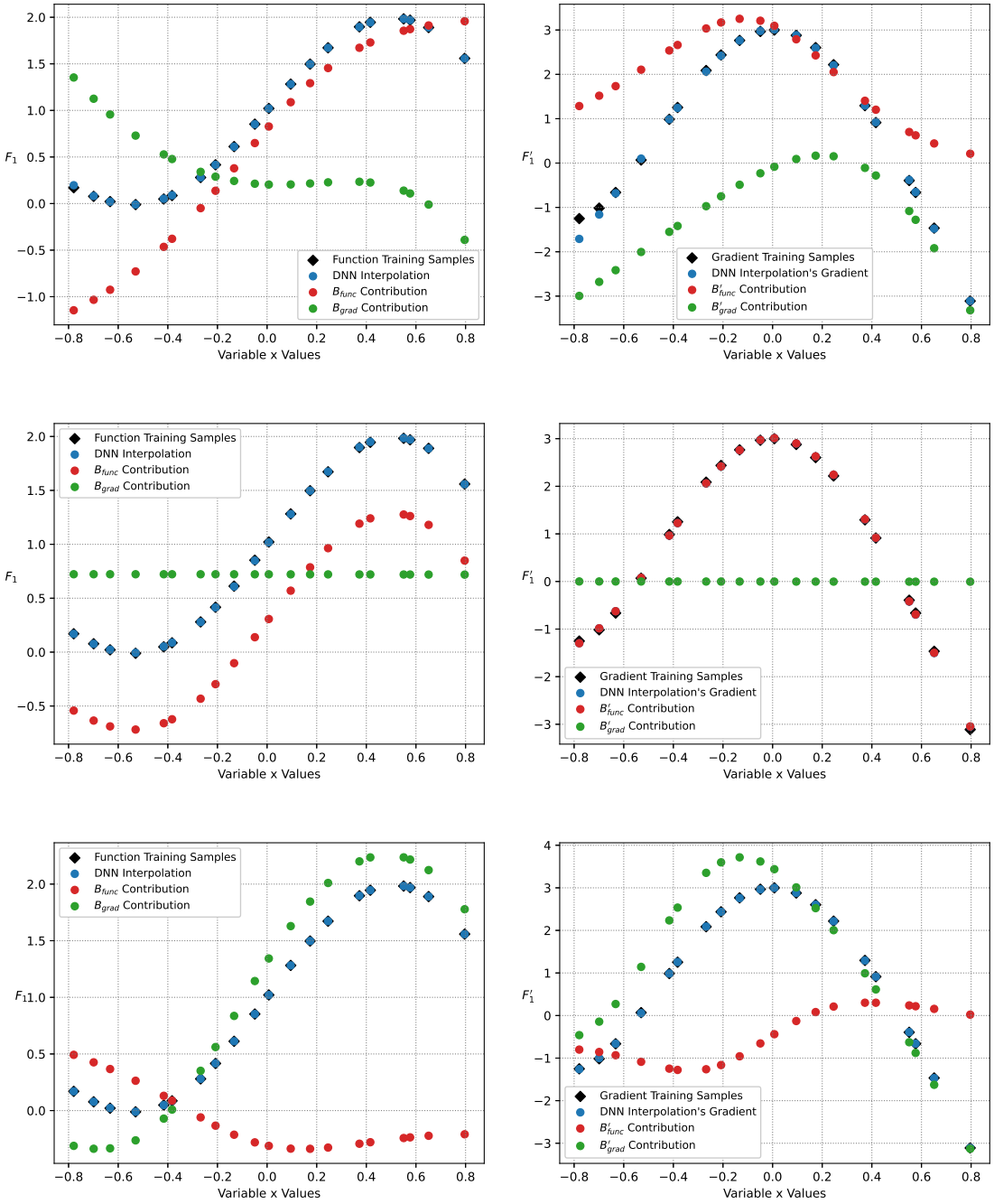


Figure 7.11: *Implementation II : The representations of Fig. 7.8 are re-constructed in the case the tanh or sigmoid (top), the GELU (middle) and Linear or None (Bottom) activations are used on the two output layers of the surrogate model.*

The proposed method is extended to the bi-variate function of Chapter 3. The predicted values of F and its two partial derivatives computed from the trained model as well as the contribution functions of each branch are shown in Fig. 7.12.

The function is learned mainly by the B_{func} branch and the two constituents of B_{grad} correct the curvature and position of the B_{func}'s output. Deviations are observed in the predicted surface near the boundaries of the X-Y domain, that lead to inaccurate derivatives near that region.

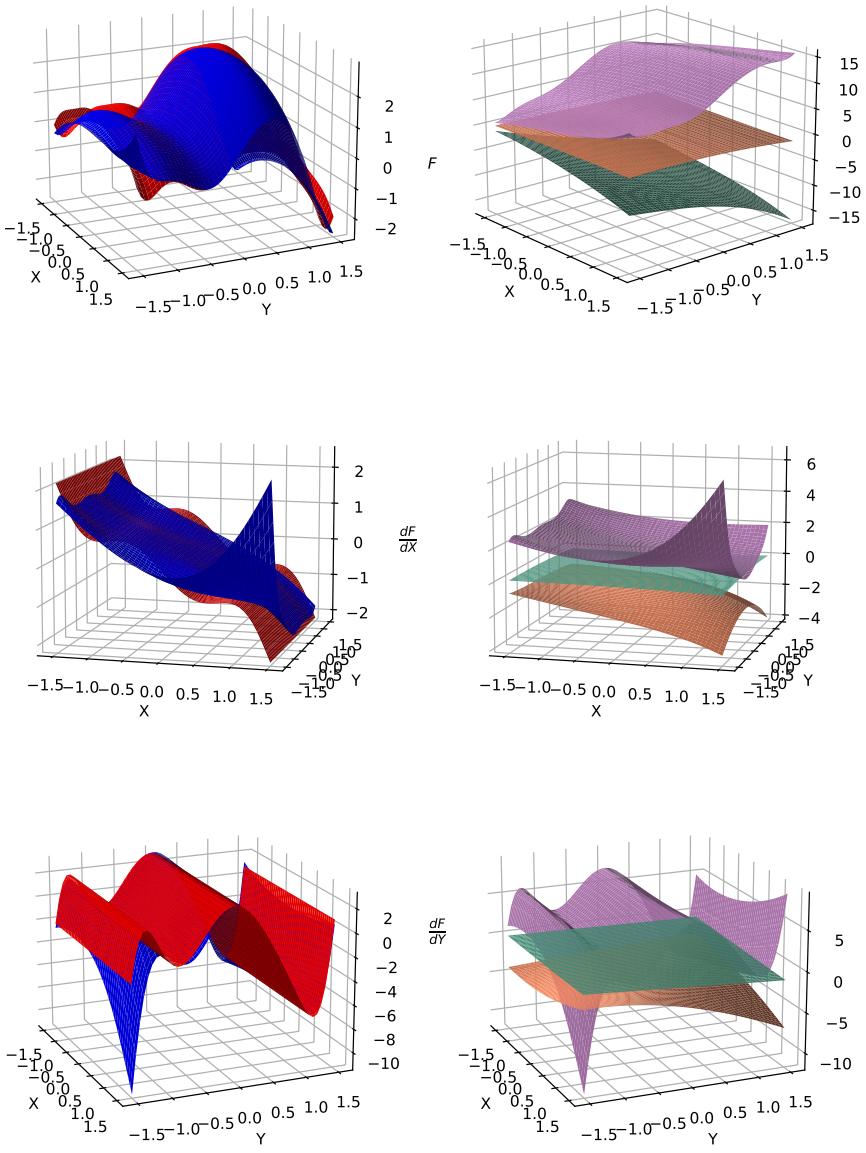


Figure 7.12: *Implementation II : (Top) The target function F (red) is compared with the predicted function (blue) of the surrogate model (left). The predicted surface results from the combination of the B_{func} contribution (purple) and the two contributions of the B_{grad} branch, that refer to the partial derivative of F w.r.t. X (orange) and Y (green) respectively (right). The same representations are depicted for the partial derivative of F w.r.t. X (Middle) and w.r.t. Y (Bottom).*

7.3.4 Demonstration of the Hermite Method on Problem III

In Problem III, two separate DNN models are trained according to the Hermite method in order to predict the S8052 airfoil's C_L, C_D coefficients respectively. The total loss function consists again of $N_b = 16$, in total, gradient loss constituents that are summed up and added to the function loss constituent, as expressed in

$$Loss_{hermite} = \sum_{i=1}^{N_s} \left[L_f^p(\hat{g}_i, y_i) + \sum_{j=1}^{N_b} L_g^p(D_{x_j}\hat{g}_i, D_{x_j}y_i) \right] \tag{7.14}$$

The DB_{DNN} of the 25 airfoil geometries is used to train the DNNs where 20% of the sampled airfoils is used to cross-validate the models with the pattern of 7 iterations of 200 epochs each. Two new optimizations were carried out using the EASY software in order to configure the optimal architecture of the two models. The objective of the optimization was to minimize the $Loss_{hermite}$ on the first cross-validation iteration. The rest of the models hyperparameters remained fixed during training. The MSE is used as the loss function for all constituents and Adam as the optimizer with a learning rate of 0.001. The normalized values of the C_L, C_D coefficients and their sensitivity derivatives are computed within the [0, 1] range for all samples in the DB_{DNN} thus, no ancilary weights are applied on the individual loss terms. The optimized architectures are presented in Table 7.2.

DNN_{C_L}	Neurons per Hidden Layer	Activations
B_{func}	2048 - 4096 -128 - 25	GELU - tanh
B_{grad}	64 - 32 - 4096 - 64 - 32 - 2048 - 128 - 64 - 25	GELU - tanh
DNN_{C_D}	Neurons per Hidden Layer	Activations
B_{func}	4096 - 4096 - 1024 - 32 - 32 - 2048 - 32 - 32 - 25	GELU - GELU
B_{grad}	1024 - 128 - 128 - 32 - 512 - 25	GELU - ReLU

Table 7.2: *Implementation II : Optimal Configurations of the surrogate models.*

The loss convergence of the two models, again to be referred as DNN_{C_L} and DNN_{C_D} respectively, is presented in Fig. 7.13. Instead of showing the total loss, it is decided to plot the function and gradient terms separately in order to isolate the convergence of each constituent. The function term in this case refers to the coefficients loss and the gradient term to the sensitivities loss. Once trained, the models are called to predict the C_L and C_D coefficients of each sample in the DB_{DNN} and its predictions are depicted on the C_L-C_D space, normalized with the baseline geometry's values respectively (Fig. 7.14). The percentage MAE metric value of each sample's C_L, C_D prediction is computed and presented in Fig. 7.15. A mean MAE value of around $\sim 0.3\%$ is achieved for the C_L predictions and a value of around $\sim 0.085\%$ for the C_D

predictions. The SDs of both C_L and C_D w.r.t. the models inputs are computed for the baseline geometry after the differentiation of the two models and are compared with the sensitivities of the standard-trained DNNs, the Sobolev-trained DNNs and their reference FDs and ajoint values in Fig. 7.16. The computed C_L SDs satisfactory match the reference SDs, while deviations are observed on the C_D SDs at specific design variables.

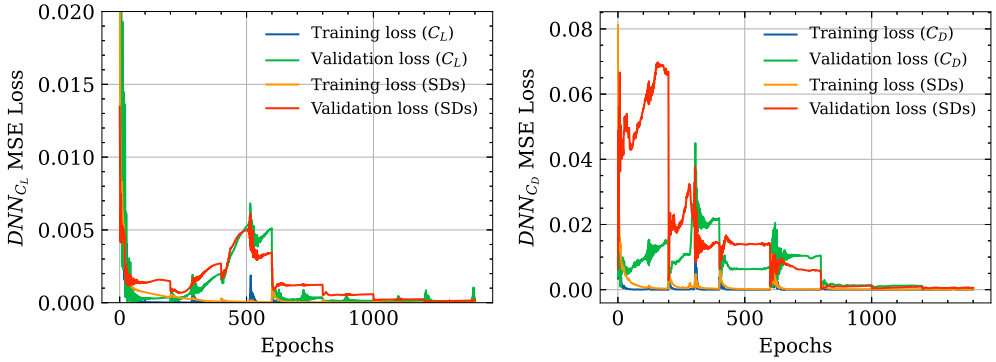


Figure 7.13: *Implementation II : The convergence of the training (blue) and validation (green) function loss terms and the convergence of the training (orange) and validation (red) gradient loss terms during the trainng of the DNN_{C_L} (Left) and the DNN_{C_D} (Right) models.*

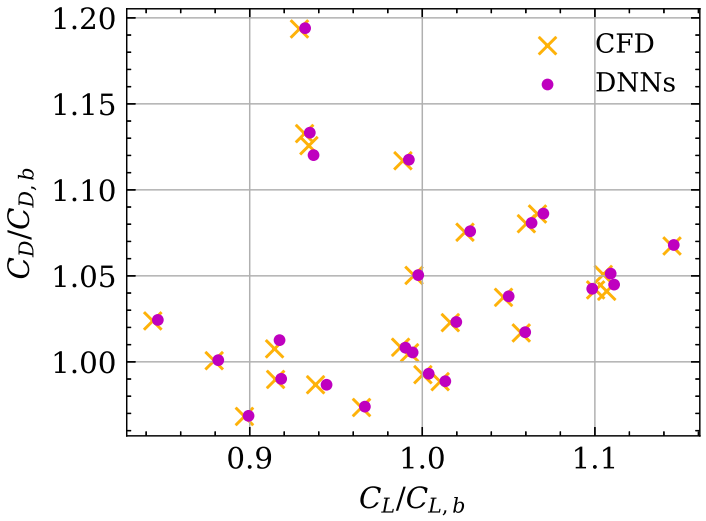


Figure 7.14: *Implementation II : The CFD-evaluated values (yellow) of C_L, C_D for each sampled geometry in the DB_{DNN} are represented on the $C_L - C_D$ space and are compared with the predicted values (purple) of the two DNN models. The target and predicted coefficients are normalized with the baseline values $C_{L,b}, C_{D,b}$.*

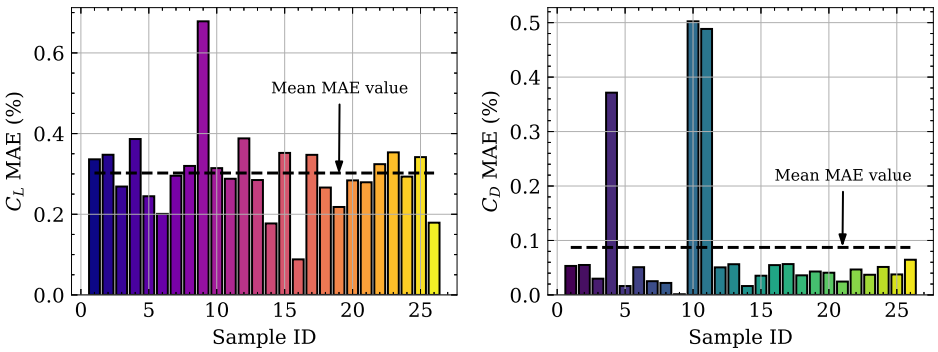


Figure 7.15: *Implementation II : The computed percentage MAE metric for the C_L (Left) and C_D (Right) predictions of each sample in the DB_{DNN}.*

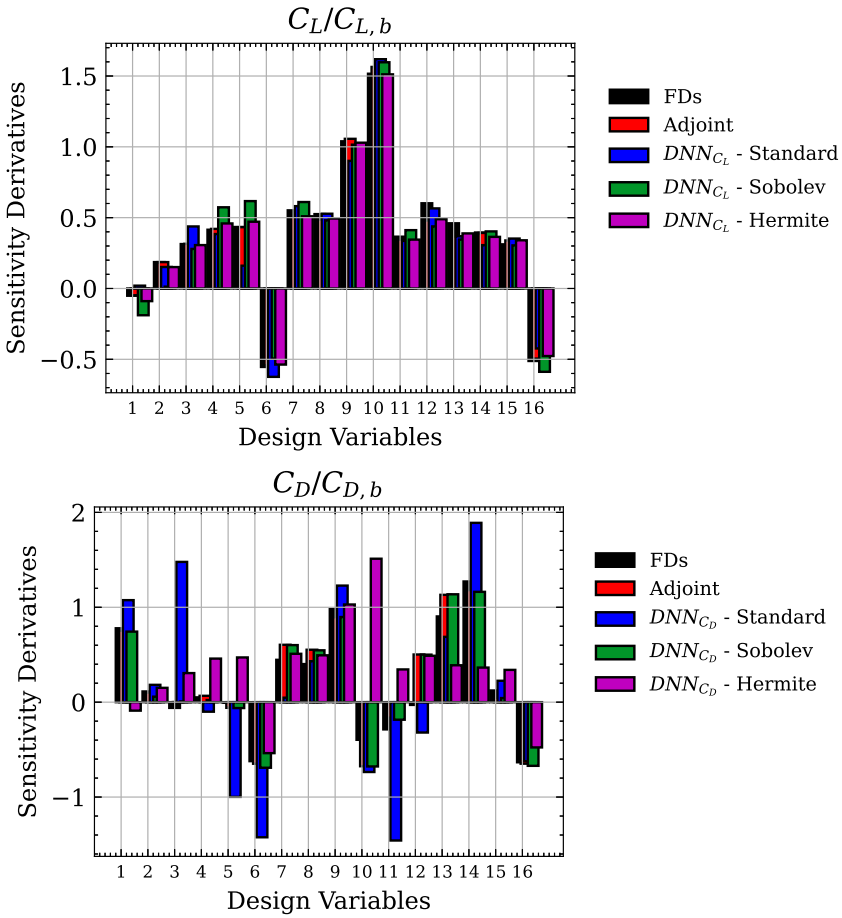


Figure 7.16: *Implementation II : The SDs of C_L (Top) and C_D (Bottom) w.r.t. the design variables for the baseline geometry computed with FDs (black), adjoint (red), the standard-trained DNN (blue), the Sobolev-trained DNN (green) and the Hermite-trained DNN (purple).*

7.4 S8052 Airfoil's Shape Optimization using the Sobolev-trained and Hermite-trained DNNs

The Sobolev-trained and Hermite-trained DNNs are used to drive the S8052 airfoil's shape optimization with the same objective function as in Chapter 6. Two separate DNN-driven descents are carried out driven by the two models. In this case, the cost to form the $\mathrm{DB}_{\mathrm{DNN}}$ is 50 TUs; 25 TUs required for the evaluation of each sampled airfoil on the CFD tool and 25 more TUs for the computation of the SDs of each geometry using the adjoint solver. At both optimization runs, just one DNN - driven descent followed by one CFD re-evaluation is sufficient to reach the optimal solution, resulting in 51 TUs turnaround time. In Fig. 7.17, the two optimization are compared with the optimization of the standard-trained DNN and the adjoint-based optimization.

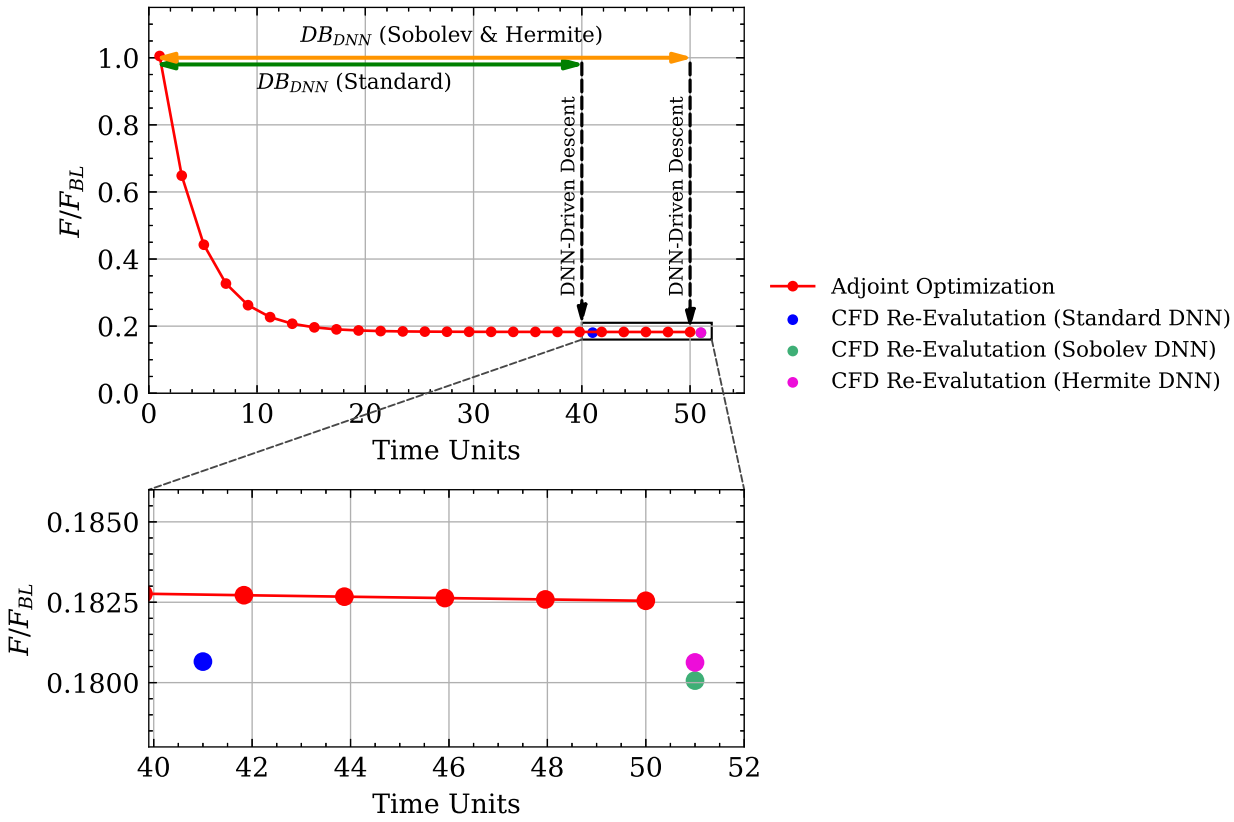


Figure 7.17: *Problem III: (Top) Convergence history of the optimization runs based on the adjoint method (red) and the differentiated DNNs (black). The CFD re-evaluated solutions of the standard-trained (blue), the Sobolev-trained (green) and the Hermite-trained (purple) DNN-driven descents are shown in filled circle points. (Bottom) Close-up view of the previous curve at the area of the solutions for better comparison.*

All DNN-driven optimizations achieve a better solution than the adjoint-based optimization, however, the turnaround time of the Sobolev and Hermite runs is slightly higher, due to the increased cost of constructing the $\mathrm{DB_{DNN}}$. For comparison, in Fig. 6.9, the C_L and C_D values of all the DNN-driven and the adjoint-based optimization solutions are placed on the $C_L - C_D$ space along with the $\mathrm{DB_{DNN}}$ samples used to train the DNN models in each case. The three DNN solutions reduce the C_D value at the optimized geometry (5% reduction with the Sobolev variant and 4% with the Hermite) and result in a small deviation from the $C_{L,target}$ (0.1% with the Sobolev variant and 0.2% with the Hermite). On the contrary, the solution of the adjoint based optimization matches the $C_{L,target}$ for a higher C_D value.

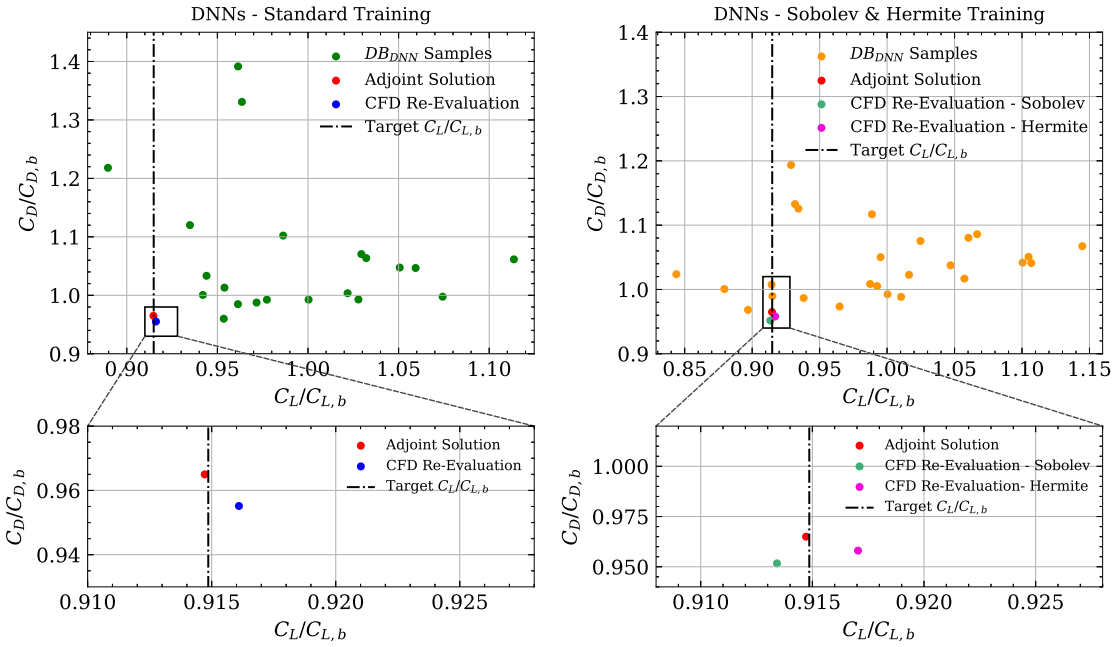


Figure 7.18: *Problem III: (Left) Representation of the* 40 DB_{DNN} *samples (green) used to train the standard DNN, the adjoint-based optimization's solution (red) and the standard-trained DNN-driven optimization's solution (blue) on the* $C_L - C_D$ *space (top). Close - up view on the two solutions (Bottom). (Right) Representation of the* 25 DB_{DNN} *samples (yellow) used to train the Sobolev and Hermite DNNs, the adjoint-based optimization's solution (red) and the Sobolev-trained (green) and Hermite-trained (purple) DNN-driven optimization's solutions on the* $C_L - C_D$ *space (top). Close - up view on the three solutions (Bottom).*

7.5 Comparison of the Adjoint-Based and the DNN-Driven Optimization Runs

The optimized airfoil shapes that resulted from the four optimization runs are compared with the baseline geometry in Fig. 7.19. The pressure coefficient (C_P) and friction coefficient (C_f) of each optimized geometry are presented in Figs. 7.22 and **??**, respectively. For comparison, the C_P and C_f coefficients of the baseline geometry are shown in Fig. 7.21. The Mach number field around the optimized airfoils is depicted in Fig. 7.24 and the turbulent viscocity field in Fig. 7.25.

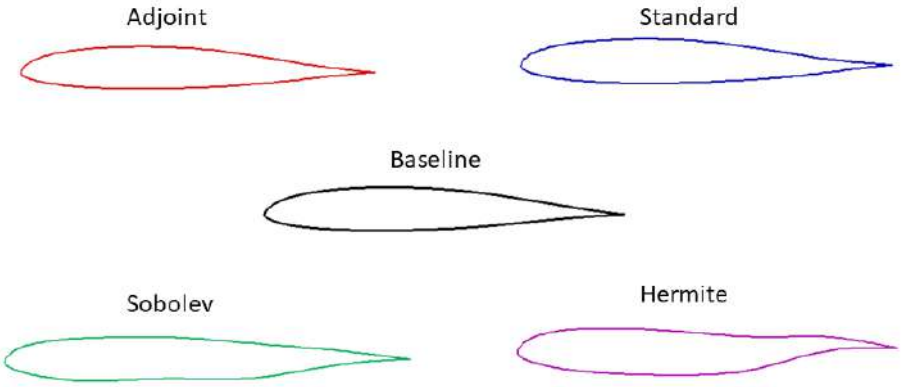


Figure 7.19: *Problem III: The optimized geometries resulting from the adjoint-based (red), the standard-trained (blue), the Sobolev-trained (green) and the Hermite-trained (purple) DNN-driven optimizations are compared with the baseline geometry (black). The objective of the optimization was to reduce the airfoil's C_D, while not decreasing its C_L under the 10% of the $C_{L,b}$ value.*

The four optimizations result in four different airfoils. The optimized geometries of the adjoint-based and standard-trained DNN - driven optimizations are similar; the airfoil's curvature on both the suction and pressure side is properly modifyed so as to match the $C_{L,target}$ and decrease the C_D coefficient. At the optimized airfoil of the Sobolev-trained DNN optimization a flatter suction side is observed in compare to the baseline geometry, in addition to the formation of a cavity on the pressure side. As verifyed in Fig. 7.22, this cavity contributes to the overall lift production, as it increases the pressure difference between the pressure and suction side of the airfoil. The cavity's shape is formed such that the resulting C_L to match the target coefficient's value. On the other hand, the flatter suction side surface decreases the airfoil's C_D, as it allows the flow to remain attached over a longer part of the airfoil. In a flat, or nearly flat, upper surface the turbulent boundary layer remains relatively thin and well-behaved over a larger portion of the airfoil's upper surface and consequently, the turbulent boundary layer experiences lower skin friction drag. This behavior can be verifyed from the decrease of the C_f coefficient

of the optimized airfoil in Fig. 7.23. Similarly to the Sobolev case's solution, a cavity is formed on the pressure side near the trailing edge of the Hermite case's optimized airfoil, contributing to the increase of C_L such as to match the $C_{L,target}$. The surface of the suction side is almost flat until the point where a curved slope is formed, due to the creation of cavity on the pressure side of the airfoil. This slope locally increases the C_D as shown in Fig. 7.23, which is later decreased due to the re-flattening of the suction side's surface until the trailing edge.

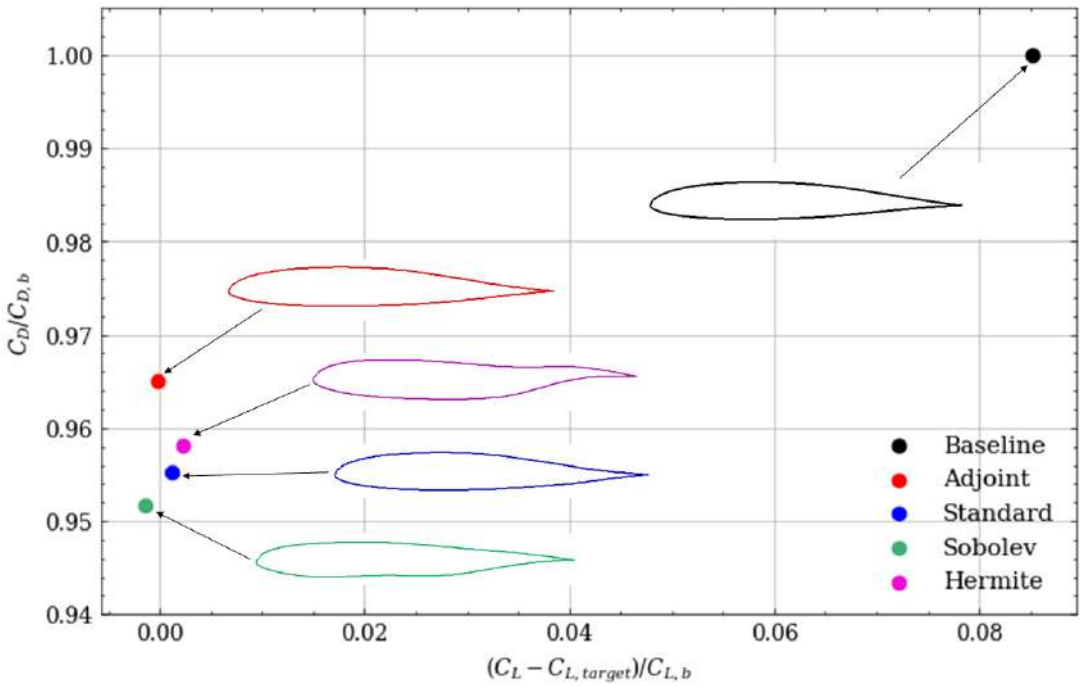


Figure 7.20: *Problem III: The $(C_L - C_{L,target})/C_{L,b}$ and $C_D/C_{D,b}$ values of the optimized geometries resulting from the adjoint-based (red), the standard-trained (blue), the Sobolev-trained (green) and the Hermite-trained (purple) DNN-driven optimizations are compared with the baseline values (black).*

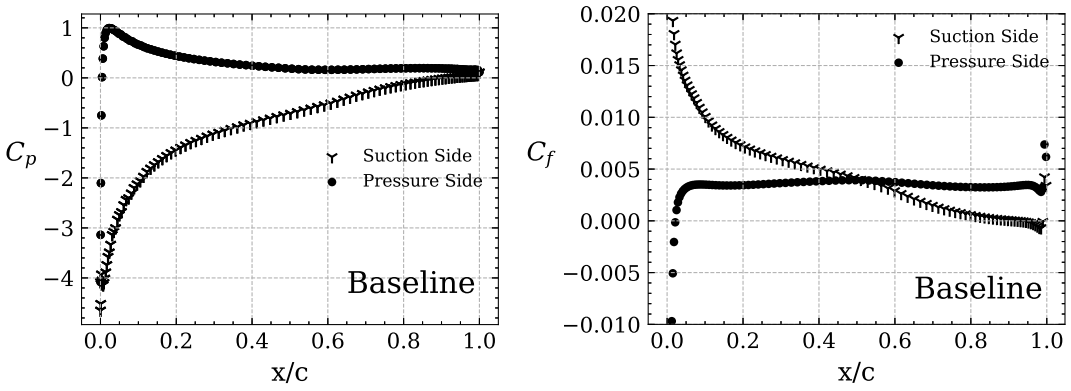


Figure 7.21: *Problem III: (Left) The C_P distribution over the suction and pressure side of the airfoil for the baseline geometry. (Right) The C_f distribution over the suction and pressure side of the airfoil for the baseline geometry.*

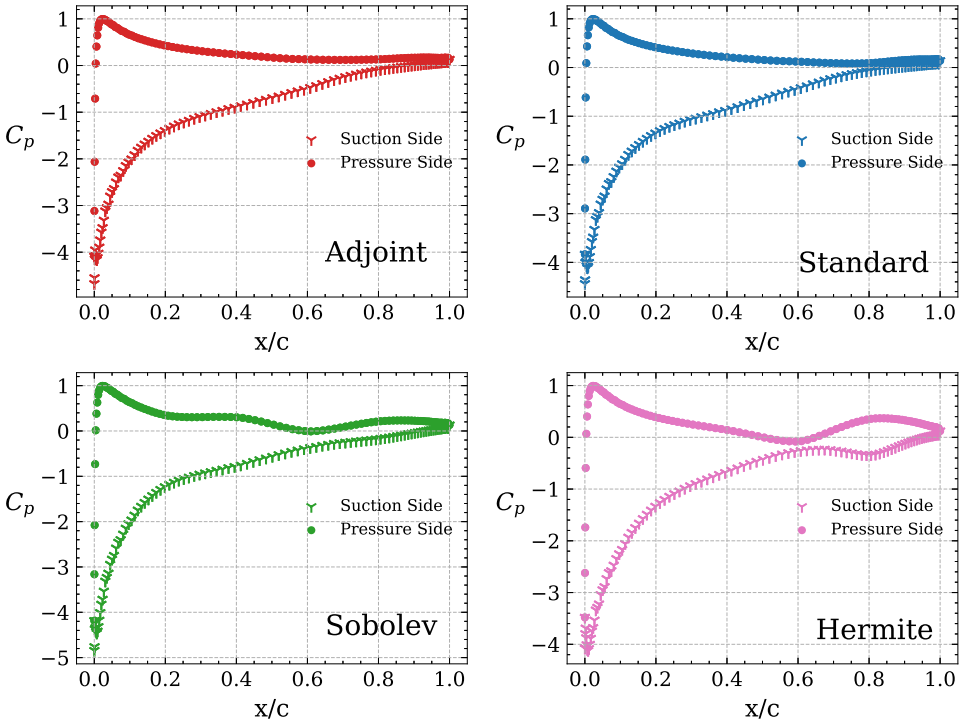


Figure 7.22: *Problem III: (Top) The C_P distribution over the suction and pressure side of the optimized airfoils resulting from the adjoint-based (red) and the standard-trained DNN (blue) optimizations is presented. (Bottom) The C_P distribution over the suction and pressure side of the optimized airfoils resulting from the Sobolev -trained DNN (green) and the Hermite-trained DNN (purple) optimizations is shown.*

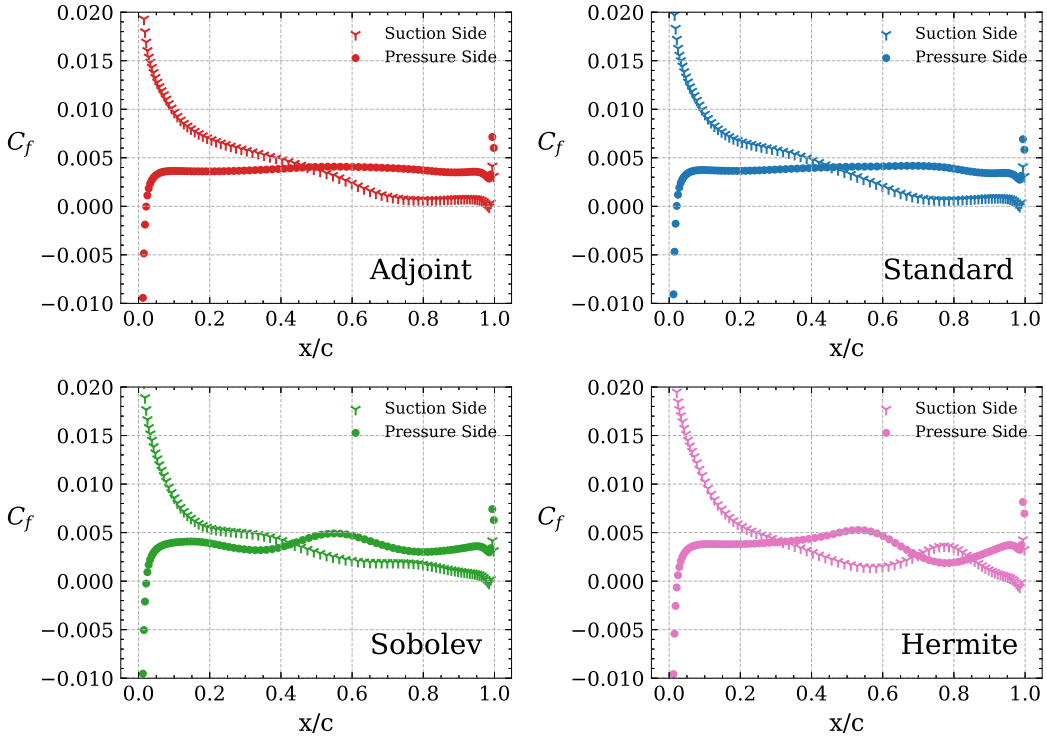


Figure 7.23: *Problem III: (Top) The C_f distribution over the suction and pressure side of the optimized airfoils resulting from the adjoint-based (red) and the standard-trained DNN (blue) optimizations is presented. (Bottom) The C_f distribution over the suction and pressure side of the optimized airfoils resulting from the Sobolev -trained DNN (green) and the Hermite-trained DNN (purple) optimizations is shown.*

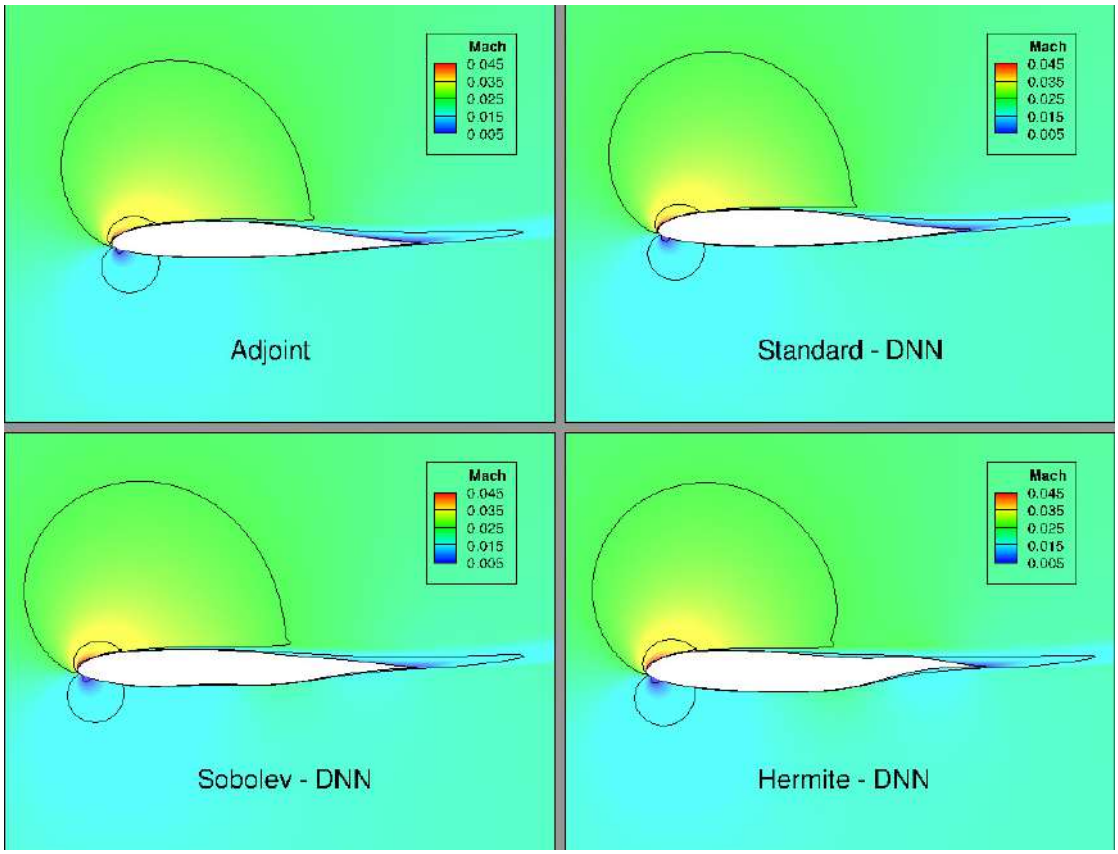


Figure 7.24: *Problem III: (Top) Mach Number field of the optimized airfoils resulting from the adjoint-based (red) and the standard-trained (blue) DNN-driven optimizations. (Bottom) Mach Number field of the optimized airfoils resulting from the Sobolev -trained (green) and the Hermite-trained (purple) DNN-driven optimizations.*

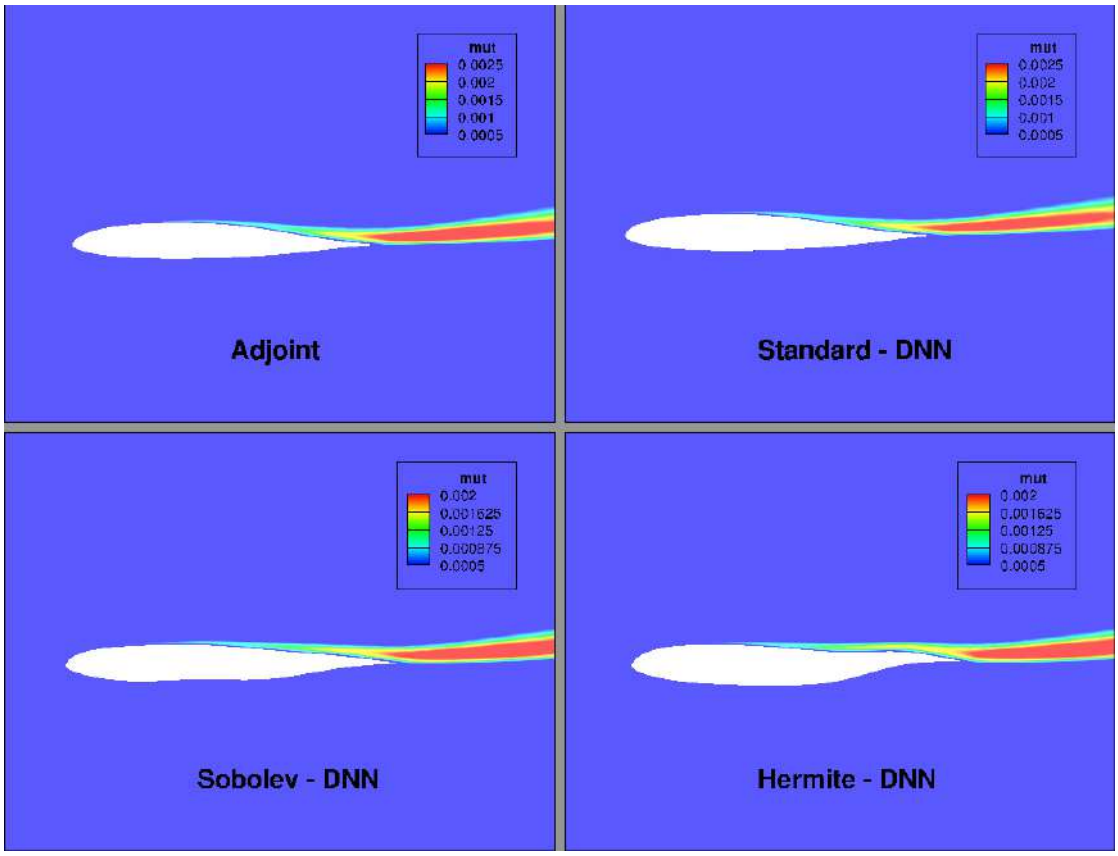


Figure 7.25: *Problem III: (Top) Turbulent viscocity field of the optimized airfoils resulting from the adjoint-based (red) and the standard-trained (blue) DNN-driven optimizations. (Bottom) Turbulent viscocity field of the optimized airfoils resulting from the Sobolev -trained (green) and the Hermite-trained (purple) DNN-driven optimizations.*

Chapter 8

Conclusion

8.1 Overview

In this Diploma Thesis, the implementation of differentiated DNNs, within a gradient-based optimization method in Computational Fluid Dynamics, for predicting the objective function values and their gradients w.r.t the design variables, was demonstrated and assessed. In the newly developed method, DNNs, after being trained on a set of patterns for which the objective function values are available, were used to replace both the code simulating the fluid flow and its adjoint solver computing gradients in CFD problems. To form the training dataset, the baseline geometries were parameterized using NURBS lattices, and the samples were generated using the LHS technique. The size of the database was selected according to the number of the design variables involved in each case, as well as the complexity of the used flow model. Prior to successfully and efficiently supporting the optimization loop, the DNNs' gradients were verified against finite differences and the adjoint method. The proposed DNN-driven shape optimization method was used to optimize the shapes of two isolated airfoils (in inviscid and turbulent flow conditions), as well as a curved duct (with laminar flow conditions). Both the efficiency and efficacy of the programmed software were compared with an adjoint-based optimization.

For the first isolated airfoil case (NACA0012), the objective was to re-design the airfoil's shape, so as to match a target C_L ($C_{L,target} = 0.6 \cdot 10^{-2}$). The flow around the airfoil was inviscid, with a free-stream Mach Number of 0.5 and flow angle of 2^o. A database of 20 airfoil geometries was generated in order to train the DNN. Its configuration resulted from a trial-and-error procedure on its hyperparameters, and a parametric study regarding the accuracy in both its predicitons and gradient was conducted, focusing mainly on the used activation functions. The proposed

optimization, driven by the trained DNN, resulted in a slightly better solution when compared with the adjoint-based optimization, at 31% less computational cost.

For the S-Bend case, the objective was to miminize the total pressure losses between the inlet and outlet. The flow inside the duct was laminar with Reynolds number $R_e = 1.84 \cdot 10^4$. A database of 50 training patterns was generated for training the DNN and its configuration resulted from a trial-and-error procedure on its hyperparameters, the criterion being its accuracy (on both the predictions and the gradient). In this case, the training samples were normalized in the range of $(0, 1)$ using a minimun value which was by 10% lower than the minimum encoutered in the samples, due to the use of the sigmoid activation function on the output layer of the model. In addition, the 'cross-validation' technique was first introduced, that due to its efficacy, was later used in the other isolated airfoil case, too. The proposed optimization resulted in the same reduction in the objective function's value as with the adjoint method (4.6%), at the same computational cost. In addition, a new capability of the proposed algorithm was demonstrated, by performing multiple DNN-driven descents simultaneously, starting from different initialization points.

For the second airfoil case (S8052), the objective was to minimize the C_D, while not decreasing its C_L below 10% of that of the baseline geometry. The flow around the airfoil was turbulent, with Reynolds number $R_e = 5 \cdot 10^5$, a free steam Mach Number of 0.5 and a flow angle of 10^o. A database of 40 airfoils was generated. Herein, two separate DNNs were trained to predict the airfoil's C_L and C_D, respectively, and their configurations were optimized using EASY. The two models were used to drive the airfoil's optimization, that resulted in a better solution than the adjoint-based optimization at 20% less computational cost. The optimized geometry using the proposed method had a by 4% lower C_D value (while matching the target C_L with a 0.1% deviation), in comparison with the adjoint's solution that had by 3% lower C_D value (while matching the target C_L with a 0.01% deviation). In addition, a parametric study regarding the size of the database used to train the models was conducted, in order to study the models' accuracy, in terms of both its response and the gradient of the response.

Next, the use of DNNs that were trained in both the objective function values and its gradient in the proposed optimization was studied. The gradient-assisted training of DNNs was implemented in two variants, the so-named Sobolev and Hermite variants. Both variants of the proposed algorithm were demontrated in the shape optimization of the second isolated airfoil, with the same geometry parameterization, flow conditions and the same objective function. A database of 25 airfoils was generated, including both the objective function's values and its gradients, computed using the adjoint method. Again two separate DNNs were used for predicting the airfoil's C_L and C_D, respectively, and both models' configurations were optimized using EASY. The computed gradients of DNNs were verified against finite differences and the adjoint method. The proposed optimization when using the gradient-assisted trained

DNNs resulted in a better solution than the adjoint-based optimization. The computational cost was the same as the adjoint, and, slightly higher than the cost of the standard DNN-driven method. Specifically, the Sobolev variant's solution had a 5% lower C_D value (while matching the target lift coefficient with a 0.01% deviation), and the Hermite variant's solution had a 4% lower C_D value (while matching the target lift coefficient with a 0.02% deviation).

8.2 Conclusions

By completing the code development and various studies in this Diploma Thesis, the following conclusions are drawn:

- **DNN Architecture and Activation Functions**: The configuration of the DNNs and, in particular, the activation functions play a key role in both the accuracy of the DNN predictions and computed gradients, as well as the efficacy of the proposed optimization algorithm. First, the gradients that result from the DNNs' differentiation highly depend on the continuity and the saturation behavior of the activation functions. After parametric studies conducted in this Diploma Thesis, it was concluded that the use of the GELU activation (exclusively, in all the DNNs' layers, or, only in their hidden layers and then combined with the proper activation function on the output layer) leads to the computation of more accurate gradients, without sploiling the accuracy of the predictions. Next, the DNNs' performance, when used in optimizations, highly depends on the capability of the activations to output negative values, in order to decrease the objective function's value beyond the minimum one encoutered in the samples. For this reason, the training data must properly be normalized. A value that is by a percentage lower than the minimum value in the $\mathrm{DB_{DNN}}$ could be used, according to the used activations. It was concluded that the GELU activation function outperforms, again, the most commonly used activations in the literature.

- **Size of the $\mathbf{DB_{DNN}}$**: The size of the $\mathrm{DB_{DNN}}$ is a trade-off between the DNNs' accuracy (in both their predictions and gradients) and the total cost of the optimization. At the demonstrated CFD problems, the $\mathrm{DB_{DNN}}$'s size was kept as small as possible so as not to increase the computational cost. In some cases, the small size of $\mathrm{DB_{DNN}}$ lead to deviations in both the computed DNN gradients from their reference (adjoint) values, and the DNN-driven descents' outcomes from their re-evaluations on the CFD tool. However, this limitation was overcomed by performing re-trainings of the DNNs on the new evaluated 'optimal' solutions during the optimization. As concluded from the parametric study regarding the $\mathrm{DB_{DNN}}$'s size of Chapter 3, the DNNs' accuracy improves when the $\mathrm{DB_{DNN}}$'s size increases until a specific number of samples. Above that number, the difference in the DNNs accuracy is not that significant. As

a result, it is preferable to to keep a small DB_{DNN} size for training the DNNs, even though discrepancies might occur on their predicitons or gradients, and then improve their accuracy during the optimization process, by performing CFD re-evaluations of the 'optimal' solutions and re-trainings (as many as needed). At all demonstrated CFD problems, this pattern was proven succesful in keeping the overall cost of the optimization small.

- **DNN-driven optimization**: Overall, it was proven that DNNs can sucessfully replace both the flow and adjoint solver in CFD-based optimizations and therefore decrease their turnaround time. Both the efficacy and efficiency of the proposed DNN-driven gradient-based optimization method was evidenced, as it lead to better (with lower objective function value) solutions compared to the widely-used adjoint method, at either the same computational cost with the latter, or by a percentage lower than that. In addition, due to its efficiency, the proposed optimization offers flexible capabilities, such as performing multiple DNN-driven descents simultaneously, starting from different initialization points. This capability is promising in overcoming the limitations of the gradient-based optimization methods, that highly depend on the initialization point, or, even exploitate multiple solutions in case of many local minima. A key advantage of the proposed optimization is that, once the DNNs are trained, they can be used in optimizations with different objective functions (if the CFD mesh, geometry parameterization and flow conditions remain the same). In comparison with other methods, including the adjoint, different optimization runs must be carried out for different objective functions.

- **Gradient-Assisted Training of DNNs**: When the training of DNNs incorporates the targets' derivatives in addition to the target values, it was concluded that the accuracy of the computed DNN gradients increases, without spoiling the accuracy of the predicitons. However, since the computation of the gradients in CFD problems is computationaly expensive, the cost of forming the DB_{DNN} increases and, consequently, the overall cost of the optimization. If an efficient method, such as the adjoint, is used to compute the gradient of the DB_{DNN} samples, as in the demonstrated cases of this Diploma Thesis, the proposed optimization can lead to even better solutions, at the same computational cost with the adjoint method, or slightly higher cost than the DNN-driven optimization when using DNNs that are trained without gradient information. As a result, if an adjoint solver is available, it would be preferable to perform the proposed optimization with DNNs that are trained on both the targets' values and derivatives, still, with a small DB_{DNN} size.

8.3 Future Work Proposals

Based on the implementation of DNNs in gradient-based optimization in CFD, the following future works are proposed:

- First, alternative techniques for implementing the DNNs' gradients in the training process could be studied. Apart from new implementation ideas, the capability of computing high order derivatives of DNNs would be an interesting topic of research, in order to extend the proposed optimization method to other fields, such as robust optimization or to allow the use of a Newton method into the optimization loop.
- Next, the efficacy and capabilities of the proposed DNN-driven optimization could be studied in cases where irregularities occur in the objective function gradients, such as in high-speed aerodynamics (creation of shock waves and boundary layers) or in geometries with discontinuities, as well as in cases where computing the gradients has high memory requirements and/or calls for computational resources, as in unsteady CFD problems.
- Finally, the proposed optimization method could be extended to Multi-Objective Optimization (MOO) problems in CFD. A common objective function could be used, containing as many terms as the number of the optimization's objectives, where each term is multiplied with the appropriate weight. Next, DNNs could be trained to predict each term of the objective function (a common model could be used for all terms, however this increases the possibility of spoiling the models' accuracy in both their predictions and gradients). Due to the negligible cost of the gradient-based DNN-driven descent, many optimization runs could be carried out, where in each run, the weights of each term of the objective function change (according to the designer), in order to prioritize different objectives at each optimization run. The collection of all the DNN-driven descents' solutions, after evaluated on the CFD code, could form a "Paretto Front" and therefore replace the costly evolutionary algorithm softwares that are commonly used for MOO problems in CFD.

Bibliography

[1] Fávero, L., Belfiore, P., and de Freitas Souza, R.: *Chapter 1 - overview of data science, analytics, and machine learning*. In Fávero, L., Belfiore, P., and de Freitas Souza, R. (editors): *Data Science, Analytics and Machine Learning with R*, pages 3–6. Academic Press, 2023, ISBN 978-0-12-824271-1. https://www.sciencedirect.com/science/article/pii/B9780128242711000342.

[2] Sarker, I.: *Machine learning: Algorithms, real-world applications and research directions*. SN Computer Science, 2(3):2661–8907, 2021.

[3] Greiner, R., Berger, D., and Bock, M.: *Artificial Intelligence*, pages 67–108. Springer Fachmedien Wiesbaden, Wiesbaden, 2022, ISBN 978-3-658-38159-2. https://doi.org/10.1007/978-3-658-38159-2_3.

[4] Alsharif, M., Hilary, A., Yahya, K., and Chaudhry, S.: *Machine learning algorithms for smart data analysis in internet of things environment: Taxonomies and research trends*. Symmetry, 12:88, January 2020. 10.3390/sym12010088.

[5] Beck, A. and Kurz, M.: *A perspective on machine learning methods in turbulence modeling*. GAMM-Mitteilungen, 44(1), mar 2021. https://doi.org/10.1002%2Fgamm.202100002, 10.1002/gamm.202100002.

[6] Verikas, A., Gelzinis, A., and Bacauskiene, M.: *Mining data with random forests: A survey and results of new tests*. Pattern Recognit., 44:330–349, 2011. https://api.semanticscholar.org/CorpusID:39661348.

[7] Afzal, A., Aabid, A., Khan, A., Afghan Khan, S., Rajak, U., Nath Verma, T., and Kumar, R.: *Response surface analysis, clustering, and random forest regression of pressure in suddenly expanded high-speed aerodynamic flows*. Aerospace Science and Technology, 107:106318, 2020, ISSN 1270-9638. https://www.sciencedirect.com/science/article/pii/S1270963820310002, https://doi.org/10.1016/j.ast.2020.106318.

[8] Xing, J., Wang, H., Luo, K., Wang, S., Bai, Y., and Fan, J.: *Predictive single-step kinetic model of biomass devolatilization for cfd applications: A comparison study of empirical correlations (ec), artificial neural networks (ann) and random forest (rf)*. Renewable Energy, 136:104–114, 2019,

ISSN 0960-1481. https://www.sciencedirect.com/science/article/pii/S0960148118315350, https://doi.org/10.1016/j.renene.2018.12.088.

[9] Bandi, P., Manelil, N., Maiya, M., Tiwari, S., and Arunvel, T.: *Cfd driven prediction of mean radiant temperature inside an automobile cabin using machine learning.* Thermal Science and Engineering Progress, 37:101619, 2023, ISSN 2451-9049. https://www.sciencedirect.com/science/article/pii/S2451904922004255, https://doi.org/10.1016/j.tsep.2022.101619.

[10] Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., and Lopez, A.: *A comprehensive survey on support vector machine classification: Applications, challenges and trends.* Neurocomputing, 408:189–215, 2020, ISSN 0925-2312. https://www.sciencedirect.com/science/article/pii/S0925231220307153, https://doi.org/10.1016/j.neucom.2019.10.118.

[11] Moradzadeh, A., Mansour, S., Mohammadi-ivatloo, B., and Anvari-Moghaddam, A.: *Performance evaluation of two machine learning techniques in heating and cooling loads forecasting of residential buildings.* Applied Sciences, 10, May 2020. 10.3390/app10113829.

[12] CHEN, S., GAO, Z., ZHU, X., DU, Y., and PANG, C.: *Unstable unsteady aerodynamic modeling based on least squares support vector machines with general excitation.* Chinese Journal of Aeronautics, 33(10):2499–2509, 2020, ISSN 1000-9361. https://www.sciencedirect.com/science/article/pii/S1000936120301734, https://doi.org/10.1016/j.cja.2020.03.009.

[13] Yan, C., Yin, Z., Shen, X., Mi, D., Guo, F., and Long, D.: *Surrogate-based optimization with improved support vector regression for non-circular vent hole on aero-engine turbine disk.* Aerospace Science and Technology, 96:105332, August 2019. 10.1016/j.ast.2019.105332.

[14] Abiodun, O., Jantan, A., Omolara, A., Dada, K., Mohamed, N., and Arshad, H.: *State-of-the-art in artificial neural network applications: A survey.* Heliyon, 4(11):e00938, 2018, ISSN 2405-8440. https://www.sciencedirect.com/science/article/pii/S2405844018332067, https://doi.org/10.1016/j.heliyon.2018.e00938.

[15] Abueidda, D., Koric, S., Al-Rub, R., Parrott, C., James, K., and Sobh, N.: *A deep learning energy method for hyperelasticity and viscoelasticity.* European Journal of Mechanics - A/Solids, 95:104639, 2022, ISSN 0997-7538. https://www.sciencedirect.com/science/article/pii/S0997753822001073, https://doi.org/10.1016/j.euromechsol.2022.104639.

[16] Biau, G. and Scornet, E.: *A random forest guided tour.* TEST, 25, 2016, ISSN 1863-8260. https://doi.org/10.1007/s11749-016-0481-7.

[17] Liang, Z. and Zhang, L.: *Support vector machines with the -insensitive pinball loss function for uncertain data classification.* Neurocomputing, 457:117–127,

2021, ISSN 0925-2312. https://www.sciencedirect.com/science/article/pii/S0925231221009681, https://doi.org/10.1016/j.neucom.2021.06.044.

[18] Cheng, K., Lu, Z., Zhou, Y., Shi, Y., and Wei, Y.: *Global sensitivity analysis using support vector regression.* Applied Mathematical Modelling, 49:587–598, 2017, ISSN 0307-904X. https://www.sciencedirect.com/science/article/pii/S0307904X1730344X, https://doi.org/10.1016/j.apm.2017.05.026.

[19] Ratku, A. and Neumann, D.: *Derivatives of feed-forward neural networks and their application in real-time market risk management.* OR Spectrum, 44, September 2022. 10.1007/s00291-022-00672-1.

[20] Huge, B. and Savine, A.: *Differential machine learning*, 2020.

[21] Elliott, J. and Peraire, J.: *Practical three-dimensional aerodynamic design and optimization using unstructured meshes.* AIAA Journal, 35(9):1479–1485, 1997.

[22] Jameson, A.: *Aerodynamic design via control theory.* Journal of Scientific Computing, 3:233–260, 1988.

[23] Wang, J., Li, R., He, C., Chen, H., Cheng, R., Zhai, C., and Zhang, M.: *An inverse design method for supercritical airfoil based on conditional generative models.* Chinese Journal of Aeronautics, 35(3):62–74, 2022.

[24] Bezgin, D., Buhendwa, A., and Adams, N.: *JAX-Fluids: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows.* Computer Physics Communications, 282:108527, 2023, ISSN 0010-4655.

[25] Shen, S., Shao, T., Zhou, K., Jiang, C., Luo, F., and Yang, Y.: *Hod-net: High-order differentiable deep neural networks and applications.* Proceedings of the AAAI Conference on Artificial Intelligence, 36(8):8249–8258, 2022.

[26] Kontou, M., Asouti, V., and Giannakoglou, K.: *DNN surrogates for turbulence closure in CFD-based shape optimization.* Applied Soft Computing, 134:110013, 2023.

[27] Renganathan, S., Maulik, R., and Ahuja, J.: *Enhanced data efficiency using deep neural networks and gaussian processes for aerodynamic design optimization.* Aerospace Science and Technology, 111:106522, 2021, ISSN 1270-9638.

[28] Li, J., Du, X., and Martins, J.: *Machine learning in aerodynamic shape optimization.* Progress in Aerospace Sciences, 134:100849, 2022.

[29] Lui, H. and Wolf, W.: *Construction of reduced-order models for fluid flows using deep feedforward neural networks.* Journal of Fluid Mechanics, 872:963–994, 2019.

[30] Xu, M., Song, S., Sun, X., Chen, W., and Zhang, W.: *Machine learning for adjoint vector in aerodynamic shape optimization.* Acta Mechanica Sinica, 37(9):1416–1432, 2021.

[31] Kovani, K., Kontou, M., Asouti, V., and Giannakoglou, K.: *Dnn-driven gradient-based shape optimization in fluid mechanics.* In Iliadis, L., Maglogiannis, I., Alonso, S., Jayne, C., and Pimenidis, E. (editors): *Engineering Applications of Neural Networks*, pages 379–390, Cham, 2023. Springer Nature Switzerland, ISBN 978-3-031-34204-2.

[32] Saleem, R., Yuan, B., Kurugollu, F., Anjum, A., and Liu, L.: *Explaining deep neural networks: A survey on the global interpretation methods.* Neurocomputing, 513:165–180, 2022, ISSN 0925-2312. https://www.sciencedirect.com/science/article/pii/S0925231222012218, https://doi.org/10.1016/j.neucom.2022.09.129.

[33] Santos, C. Gonçalves Dos and Papa, J.: *Avoiding overfitting: A survey on regularization methods for convolutional neural networks.* ACM Computing Surveys, 54(10s):1–25, jan 2022. https://doi.org/10.1145%2F3510413, 10.1145/3510413.

[34] Jadon, A., Patil, A., and Jadon, S.: *A comprehensive survey of regression based loss functions for time series forecasting*, 2022.

[35] Shaziya, H.: *A study of the optimization algorithms in deep learning.* March 2020. 10.1109/ICISC44355.2019.9036442.

[36] Jais, I., Ismail, A., and Qamrun, S.: *Adam optimization algorithm for wide and deep neural network.* Knowledge Engineering and Data Science, 2:41, June 2019. 10.17977/um018v2i12019p41-46.

[37] Baydin, A., Pearlmutter, B., Radul, A., and Siskind, J.: *Automatic differentiation in machine learning: a survey*, 2018.

[38] Lee, W., Park, S., and Aiken, A.: *On the correctness of automatic differentiation for neural networks with machine-representable parameters*, 2023.

[39] Shu, H. and Zhu, H.: *Sensitivity analysis of deep neural networks.* January 2019. 10.1609/aaai.v33i01.33014943.

[40] Pizarroso, J., Portela, J., and Muñoz, A.: *bNeuralSens/b: Sensitivity analysis of neural networks.* Journal of Statistical Software, 102(7), 2022. https://doi.org/10.18637%2Fjss.v102.i07, 10.18637/jss.v102.i07.

[41] Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H.: *Understanding neural networks through deep visualization*, 2015.

[42] Sun, Y., Sun, Q., and Qin, K.: *Physics-based deep learning for flow problems.* Energies, 14:7760, November 2021. 10.3390/en14227760.

[43] Clevert, D., Unterthiner, T., and Hochreiter, S.: *Fast and accurate deep network learning by exponential linear units (elus)*, 2016.

[44] Klambauer, G., Unterthiner, T., Mayr, A., and Hochreiter, S.: *Self-normalizing neural networks*, 2017.

[45] Hendrycks, D. and Gimpel, K.: *Gaussian error linear units (gelus)*, 2023.

[46] Ramachandran, P., Zoph, B., and Le, Q.: *Searching for activation functions*, 2017.

[47] Giannakoglou, K. and Papadimitriou, D.: *Adjoint Methods for Shape Optimization*, pages 79–108. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, ISBN 978-3-540-72153-6. https://doi.org/10.1007/978-3-540-72153-6_4.

[48] Trompoukis, X., Tsiakas, K., Asouti, V., and Giannakoglou, K.: *Continuous adjoint-based shape optimization of a turbomachinery stage using a 3D volumetric parameterization*. International Journal for Numerical Methods in Fluids, 2023. 10.1002/fld.5187.

[49] Rakhimov, A., Visser, D., and Komen, E.: *Uncertainty quantification method for cfd applied to the turbulent mixing of two water layers*. Nuclear Engineering and Design, 333:1–15, 2018, ISSN 0029-5493. https://www.sciencedirect.com/science/article/pii/S0029549318303959, https://doi.org/10.1016/j.nucengdes.2018.04.004.

[50] *TensorFlow: Large-Scale Machine Learning on heterogeneous systems*, 2015. https://www.tensorflow.org/, Software available from tensorflow.org.

[51] Giannakoglou, K. The EASY (Evolutionary Algorithms SYstem) software, http://velos0.ltt.mech.ntua.gr/EASY, 2008.

[52] Asouti, V., Trompoukis, X., Kampolis, I., and Giannakoglou, K.: *Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on Graphics Processing Units*. International Journal for Numerical Methods in Fluids, 67(2):232–246, 2011.

[53] Piegl, L. and Tiller, W.: *The NURBS Book (2nd Ed.)*. Springer-Verlag, Berlin, Heidelberg, 1997, ISBN 3540615458.

[54] Czarnecki, W., Osindero, S., Jaderberg, M., Świrszcz, G., and Pascanu, R.: *Sobolev training for neural networks*, 2017.

[55] Hornik, K.: *Approximation capabilities of multilayer feedforward networks*. Neural Networks, 4(2):251–257, 1991, ISSN 0893-6080. https://www.sciencedirect.com/science/article/pii/089360809190009T, https://doi.org/10.1016/0893-6080(91)90009-T.

[56] Diening, L., Harjulehto, P., Hästö, P., and Ruzicka, M.: *Introduction to sobolev spaces*. February 2011. 10.1007/978-3-642-18363-8_8.

[57] Cocola, J. and Hand, P.: *Global convergence of sobolev training for overparameterized neural networks*, 2020.

[58] Vlassis, N. and Sun, W.: *Sobolev training of thermodynamic-informed neural networks for smoothed elasto-plasticity models with level set hardening*, 2020.

[59] Tsay, C.: *Sobolev trained neural network surrogate models for optimization.* Computers Chemical Engineering, 153:107419, 2021, ISSN 0098-1354. https://www.sciencedirect.com/science/article/pii/S0098135421001976, https://doi.org/10.1016/j.compchemeng.2021.107419.

[60] Chollet, F. *et al.*: *Keras*, 2015. https://github.com/fchollet/keras.

[61] http://velos0.ltt.mech.ntua.gr/kgianna/analysis/distr/book_numanal.pdf.

[62] *Multivariate hermite interpolation by algebraic polynomials: A survey.* Journal of Computational and Applied Mathematics, 122(1):167–201, 2000, ISSN 0377-0427. https://www.sciencedirect.com/science/article/pii/S0377042700003678, https://doi.org/10.1016/S0377-0427(00)00367-8, Numerical Analysis in the 20th Century Vol. II: Interpolation and Extrapolation.

[63] Coxon, N.: *Fast hermite interpolation and evaluation over finite fields of characteristic two*, 2018.

[64] Kechriniotis, A., Delibasis, K., Oikonomou, I., and Tsigaridas, G.: *Classical multivariate hermite coordinate interpolation on n-dimensional grids*, 2023.

[65] Esmaeilbeigi, M., Chatrabgoun, O., and Cheraghi, M.: *Fractional hermite interpolation using rbfs in high dimensions over irregular domains with application.* Journal of Computational Physics, 375:1091–1120, 2018, ISSN 0021-9991. https://www.sciencedirect.com/science/article/pii/S0021999118306077, https://doi.org/10.1016/j.jcp.2018.09.013.

[66] Kampolis, I., Karangelos, E., and Giannakoglou, K.: *Gradient-assisted radial basis function networks: Theory and applications.* Applied Mathematical Modelling - APPL MATH MODEL, 28:197–209, February 2004. 10.1016/j.apm.2003.08.002.

[67] Hsu, C.: *Intelligent control of chaotic systems via self-organizing hermite-polynomial-based neural network.* Neurocomputing, 123:197–206, 2014, ISSN 0925-2312. https://www.sciencedirect.com/science/article/pii/S092523121300742X, https://doi.org/10.1016/j.neucom.2013.07.008, Contains Special issue articles: Advances in Pattern Recognition Applications and Methods.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής
& Βελτιστοποίησης

Βαθιά Νευρωνικά Δίκτυα και η Διαφόρισή τους για Χρήση στην Αιτιοκρατική Βελτιστοποίηση Αεροδυναμικών Μορφών

Διπλωματική Εργασία - Εκτενής Περίληψη στην Ελληνική

Κωνσταντίνα Κοβάνη

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2023

Εισαγωγή

Στόχος της Διπλωματικής αυτής Εργασίας, είναι η υλοποίηση της διαφόρισης των Βαθιών Νευρωνικών Δικτύων (ΒΝΔ), και η πρόταση χρήσης τους σε μία αιτιοκρατική μέθοδο βελτιστοποίησης στο πεδίο της μηχανικής ρευστών, τόσο για την πρόβλεψη των τιμών της συνάρτησης-στόχου, όσο και των παραγώγων ευαισθησίας. Η προτεινόμενη μέθοδος παρουσιάζεται με δύο παραλλαγές όσο αφορά την εκπαίδευση των ΒΝΔ, η πρώτη χρησιμοποιώντας δίκτυα εκπαιδευόμενα μόνο στις τιμές της εκάστοτε συνάρτησης-στόχου και η δεύτερη χρησιμοποιώντας δίκτυα εκπαιδευόμενα στις τιμές της συνάρτησης-στόχου και των παραγώγων ευαισθησίας. Η τελευταία παραλλαγή παρουσιάζεται με δύο υλοποιήσεις, βασιζόμενες στη μέθοδο εκπαίδευσης ΒΝΔ Sobolev και στην παρεμβολή Hermite, αντίστοιχα. Η προτεινόμενη μέθοδος οδηγούμενη από τα ΒΝΔ εφαρμόστηκε για τη βελτιστοποίηση του σχήματος δύο μεμονωμένων αεροτομών (με ατριβή και τυρβώδη ροή αντίστοιχα), καθώς και ενός αγωγού S-bend (με στρωτή ροή). Η αποδοτικότητα και το υπολογιστικό κόστος της προτεινόμενης μεθόδου συγκρίθηκαν με τη βελτιστοποίηση με τη συζυγή μέθοδο.

Τεχνητή Νοημοσύνη και Βαθιά Νευρωνικά Δίκτυα

Τα τελευταία χρόνια, η πρόοδος στην Τεχνητή Νοημοσύνη (ΤΝ) και τη Μηχανική Μάθηση (ΜΜ) ήταν ραγδαία και μεταμορφωτική, με όλο και περισσότερα μοντέλα ΜΜ να βρίσκουν εφαρμογή σε τομείς της καθημερινότητας, με ιδιαίτερη έμφαση στα ΒΝΔ. Τα ΒΔΝ [1] είναι υπολογιστικά μοντέλα εμπνευσμένα από τη δομή και τη λειτουργία του ανθρώπινου εγκεφάλου. Αποτελούνται από διασυνδεδεμένους κόμβους, γνωστούς ως νευρώνες, διατεταγμένους σε επίπεδο εισόδου, κρυφά επίπεδα και επίπεδο εξόδου. Το πιο αξιοσημείωτο χαρακτηριστικό τους είναι το βάθος τους, καθώς μπορούν να διαθέτουν πολυάριθμα κρυφά επίπεδα, επιτρέποντάς τους να κατανοήσουν ιεραρχικά χαρακτηριστικά και αναπαραστάσεις των δεδομένων. Οι νευρώνες επεξεργάζονται δεδομένα εισόδου χρησιμοποιώντας σταθμισμένες συνδέσεις (βάρη) και εφαρμόζουν συναρτήσεις ενεργοποίησης, δημιουργώντας μία έξοδο που στη συνέχεια, μεταδίδεται στο επόμενο στρώμα. Κατά τη διάρκεια της εκπαιδευτικής διαδικασίας, τα ΒΝΔ προσαρμόζουν τα βάρη για να ελαχιστοποιήσουν την απόκλιση μεταξύ των προβλέψεών τους και των πραγματικών τιμών-στόχων, χρησιμοποιώντας αλγόριθμους βελτιστοποίησης. Μέσω της επαναληπτικής βελτίωσης των βαρών σε πολλαπλές εποχές εκπαίδευσης, τα ΒΝΔ μαθαίνουν περίπλοκα μοτίβα και σχέσεις μέσα στα δεδομένα, επιτρέποντας έτσι ακριβείς προβλέψεις σε νέα, άγνωστα σε αυτά, δεδομένα. Η ενσωμάτωση των ΒΝΔ σε προσομοιώσεις Υπολογιστικής Ρευστοδυναμικής (ΥΡΔ) είναι ιδιαίτερα διαδεδομένη, λόγω της ικανότητάς τους να χειρίζονται μεγάλους όγκους δεδομένων με χαμηλό υπολογιστικό κόστος και πόρους [2, 3]. Αφού εκπαιδευτούν, μπορούν να επιταχύνουν τη διαδικασία προσομοίωσης αντικαθιστώντας μέρος, ή ολόκληρο, το εργαλείο ΥΡΔ.

Η Συζυγής Μέθοδος Βελτιστοποίησης

Η συζυγής μέθοδος [4] χρησιμοποιείται για τον υπολογισμό των παραγώγων ευαισθησίας μιας συνάρτησης-στόχου ως προς τις μεταβλητές σχεδιασμού. Χρησιμοποιείται

ευρέως στην ΥΡΔ για την υποστήριξη αιτιοκρατικών αλγορίθμων, καθώς έχει το χαμηλότερο κόστος υπολογισμού παραγώγων συναρτήσεων σε προβλήματα που διέπονται από μερικές διαφορικές εξισώσεις. Σε αυτήν την εργασία χρησιμοποιήθηκε η συνεχής συζυγής μέθοδος. Κάθε κύκλος βελτιστοποίησης περιλαμβάνει την αριθμητική επίλυση των εξισώσεων Navier-Stokes, των συζυγών εξισώσεων και τον υπολογισμό των παραγώγων ευαισθησίας που χρησιμοποιούνται για την ανανέωση του διανύσματος μεταβλητών σχεδιασμού. Οι ανανεώσεις υπολογίζονται με τη μέθοδο της απότομης καθόδου και ο αριθμός των κύκλων βελτιστοποίησης ορίζεται από κριτήρια σύγκλισης ή κόστους. Οι εξισώσεις Navier-Stokes επιλύονται με τον επιλύτη PUMA, [5], του εργαστηρίου για την παραλλαγή της συμπιεστής ροής, και τόσο οι ροικές όσο και συζυγείς εξισώσεις διακριτοποιούνται σε μη δομημένα/υβριδικά πλέγματα, χρησιμοποιώντας την τεχνική των πεπερασμένων όγκων. Οι εξισώσεις συνεκτικής ροής για συμπιεστά ρευστά γράφονται στη μορφή

$$R_n = \frac{\partial f_{nk}^{\text{ιν}}}{\partial x_k} - \frac{\partial f_{nk}^{\text{ις}}}{\partial x_k} = 0 \tag{1}$$

όπου $f_k^{inv} = [\rho v_k \quad \rho v_k v_1 + p\delta_{1k} \quad \rho v_k v_2 + p\delta_{2k} \quad \rho v_k v_3 + p\delta_{3k} \quad \rho v_k h_t]^T$ είναι οι ατριβείς και $f_k^{vis} = [0 \quad \tau_{1k} \quad \tau_{2k} \quad \tau_{3k} \quad v_\ell \tau_{\ell k} + q_k]^T$ είναι οι συνεκτικές ροές. ρ, p, v_k και h_t είναι οι όροι πυκνότητας, πίεσης, ταχύτητας, ολικής ενθαλπίας και δ_{km} είναι το σύμβολο του Kronecker αντίστοιχα. Ο πίνακας συνεκτικών τάσεων δίνεται από $\tau_{km} = \mu \left(\frac{\partial v_k}{\partial x_m} + \frac{\partial v_m}{\partial x_k} - \frac{2}{3}\delta_{km} \frac{\partial v_\ell}{\partial x_\ell} \right)$ όπου μ είναι ο συντελεστής συνεκτικότητας και q_k η ροή θερμότητας. Όλοι οι υπολογισμοί γίνονται με ακρίβεια δεύτερης τάξης. Στον PUMA περιλαμβάνονται τα μοντέλα προσομοίωσης ατριβούς (Euler), στρωτής και τυρβώδους ροής, ενώ στην τελευταία περίπτωση, δίνεται η επιλογή πολυάριθμων μοντέλων τύρβης, όπως το μοντέλο των Spalart-Allmaras, το τυπικό μοντέλο k-ε και, το βασικό και οι SST παραλλαγές του μοντέλου k-ω.

Προτεινόμενος Αλγόριθμος Αιτιοκρατικής Βελτιστοποίησης με ΒΝΔ

Εναλλακτικά, προτείνεται να αντικατασταθούν τόσο ο επιλύτης ροής όσο και των συζυγών εξισώσεων με εκπαιδευμένα ΒΝΔ που προβλέπουν τόσο την τιμή της συνάρτησης-στόχου όσο και των παραγώγων ευαισθησίας. Το πρώτο βήμα είναι η δημιουργία της βάσης δεδομένων με την οποία θα εκπαιδευτούν τα ΒΝΔ. Για τον σκοπό αυτό, οι αρχικές γεωμετρίες του κάθε προβλήματος παραμετροποιούνται χρησιμοποιώντας ογκομετρικές NURBS, [6], (Σχήμα 1) με σημεία ελέγχου (μεταβλητές σχεδιασμού) που μπορούν να μετατοπιστούν, εδώ, κατά την κατακόρυφη διεύθυνση. Η τεχνική δειγματοληψίας Latin Hypercube Sampling,[7], (αποτελεσματική όταν απαιτείται μικρός αριθμός δειγμάτων) χρησιμοποιείται για τη δημιουργία των δειγμάτων, τα οποία στη συνέχεια αξιολογούνται στον PUMA. Το μέγεθος της βάσης δεδομένων επιλέγεται ανάλογα με τον αριθμό των μεταβλητών σχεδιασμού και την πολυπλοκότητα του εκάστοτε μοντέλου ροής. Μετά τη δημιουργία της βάσης δεδομένων, κάθε γύρος (αυτός ο όρος χρησιμοποιείται για να διακρίνει αυτόν τον βρόχο από τον βρόχο απότομης καθόδου σύμφωνα με το βήμα 2 του προτεινόμενου αλγορίθμου) περιλαμβάνει τα ακόλουθα βήματα:

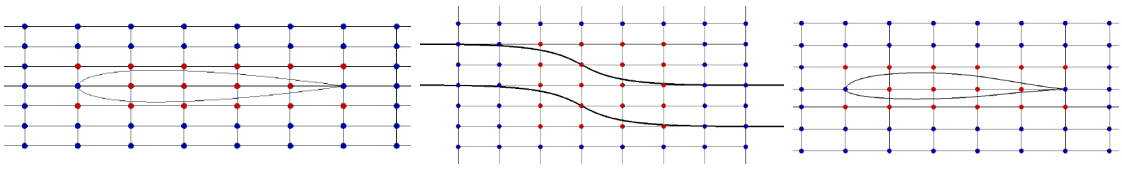
1. Εκπαίδευση του ΒΝΔ με τα δείγματα της βάσης δεδομένων (τα οποία, εδώ, αδιαστατοποιούνται όλα στο (0, 1) σύμφωνα με τις ελάχιστες και μέγιστες τιμές τους στα δείγματα). Η αρχιτεκτονική του μοντέλου πρέπει να επιλεχθεί προσεκτικά, με ιδιαίτερη έμφαση στην επιλογή των συναρτήσεων ενεργοποίησης [8], οι οποίες επιδρούν σημαντικά τόσο στην ακρίβεια (προβλέψεων και παραγώγων) των ΒΝΔ, όσο και στη αποδοτικότητα της προτεινόμενης μεθόδου. Η υλοποίηση πραγματοποιήθηκε στο πλαίσιο του TensorFlow (v2.6.0),[9], με Python.

2. Επαναληπτική βελτιστοποίηση (μέχρι τη σύγκλιση) εφαρμόζοντας απότομη κάθοδο χρησιμοποιώντας, αποκλειστικά, τις παραγώγους που υπολογίζονται από τη διαφόριση του ΒΝΔ, με αντίστροφη αυτόματη διαφόριση, [10]. Σε κάθε κύκλο βελτιστοποίησης, οι μεταβλητές σχεδιασμού δεν επιτρέπεται να ξεπεράσουν τα άνω και κάτω όριά τους τα οποία ορίστηκαν κατά τη δειγματοληψία. Ως ένα 'ακραίο' σενάριο, πολλαπλές καταχωρήσεις που επιλέγονται από τη βάση δεδομένων μπορούν να χρησιμοποιηθούν ως σημεία εκκίνησης (αρχικοποίησης) και να εκτελεστούν τόσες κάθοδοι, όσες και ο αριθμός των σημείων εκκίνησης.

3. Επαναξιολόγηση (όλων ή μερικών) των 'βελτιστοποιημένων' λύσεων με τον κώδικα ΥΡΔ. Η χρήση εισαγωγικών ('βελτιστοποιημένων') καθιστά σαφές ότι αυτή είναι η καλύτερη λύση σύμφωνα με το ΒΝΔ.

4. Ενημέρωση της βάσης δεδομένων με όλες τις πρόσφατα αξιολογημένες λύσεις, εάν είναι απαραίτητο, και επανάληψη των τεσσάρων βημάτων (από το βήμα 1). Το κριτήριο τερματισμού σχετίζεται με την ακρίβεια πρόβλεψης του ΒΝΔ.

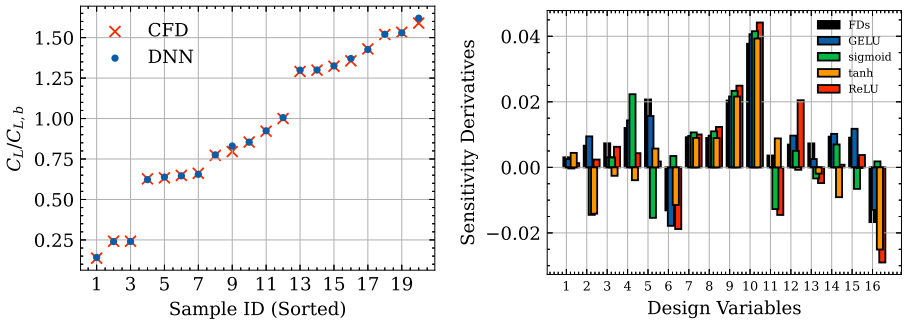
Στο βήμα 1, τα ΒΝΔ διαμορφώνονται διαφορετικά σε κάθε πρόβλημα. Εμπειρικά, η χρήση ενός μόνο ΒΝΔ σε όλα τα προβλήματα δεν ενδείκνυται στην ανάλυση στην ΥΡΔ, αφού διέπονται από διαφορετική φυσική. Οι αρχιτεκτονικές των ΒΝΔ προέκυψαν είτε με μια διαδικασία δοκιμής-και-λάθους σχετικά με την ακρίβεια των μοντέλων τόσο στις προβλέψεις, όσο και στις παραγώγους τους, είτε βελτιστοποιήθηκαν χρησιμοποιώντας το λογισμικό εξελικτικών αλγορίθμων, EASY, [11]. Αφού οριστεί η αρχιτεκτονική τους (και εκπαιδευτούν), τα ΒΝΔ μπορούν να χρησιμοποιηθούν για βελτιστοποιήσεις με οποιαδήποτε συνάρτηση-στόχο που ορίζεται από τον σχεδιαστή (υποθέτοντας ότι οι γεωμετρίες και συνθήκες ροής παραμένουν ίδιες), επομένως το κόστος εύρεσης της αρχιτεκτονικής των ΒΝΔ δεν λαμβάνεται υπόψη.



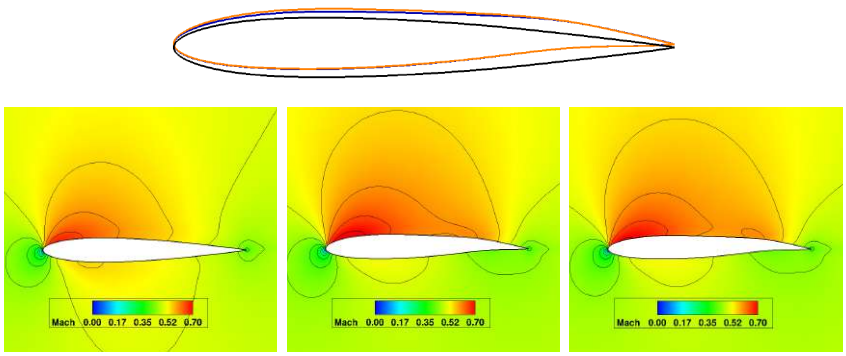
Σχήμα 1: *Πλέγμα παραμετροποίησης (μπλε) και σημεία ελέγχου (κόκκινα) των γεωμετριών της NACA0012 (Αριστερά), του S-bend (Μέση) και της S8052 (Δεξιά).*

Βελτιστοποίηση μίας Μεμονωμένης Αεροτομής (Ατριβής Ροή)

Στο Πρόβλημα Ι, η προτεινόμενη μέθοδος χρησιμοποιήθηκε για τη βελτιστοποίηση του σχήματος της αεροτομής NACA0012, ώστε το επανασχεδιασμένο σχήμα να ταιριάζει με έναν στόχο-τιμή του συντελεστή άνωσης ($C_{L,target} = 0.6 \cdot 10^{-2}$). Η ροή γύρω από την αεροτομή είναι ατριβής (Euler) με γωνία $a = 2^o$ και αριθμό $Mach = 0.5$ για την επ΄ άπειρο ροή. Για την εκπαίδευση του ΒΝΔ, δημιουργήθηκε μια βάση δεδομένων με 20 διαφορετικές γεωμετρίες. Η αρχιτεκτονική του μοντέλου προέκυψε από μια διαδικασία δοκιμής-και-σφάλματος σχετικά με τις (υπερ)παραμέτρους του, ενώ διεξήχθη παραμετρική μελέτη σχετικά με την ακρίβεια τόσο των προβλέψεων του δικτύου και των παραγώγων του, εστιάζοντας κυρίως στις χρησιμοποιούμενες συναρτήσεις ενεργοποίησης (Σχήμα 2). Μεγαλύτερη ακρίβεια επιτυγχάνεται με τη χρήση της συνάρτησης GELU. Η προτεινόμενη βελτιστοποίηση οδήγησε σε καλύτερη λύση (Σχήμα 3) σε σύγκριση με αυτή της συζυγούς μεθόδου, έχοντας κατά 31% μικρότερο υπολογιστικό κόστος. Για την επίτευξη του στόχου, χρειάστηκαν 3 επαν-εκπαιδεύσεις του ΒΝΔ.



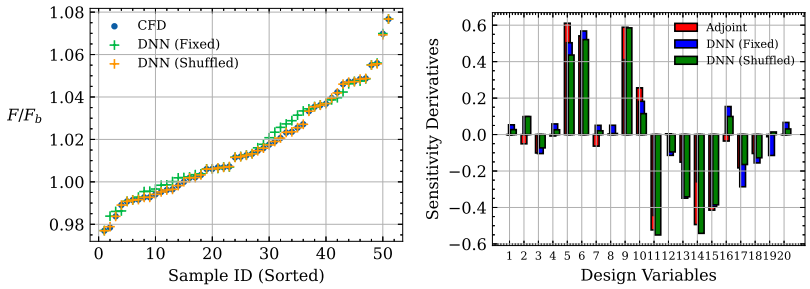
Σχήμα 2: *Πρόβλημα Ι: (Αριστερά) Προβλέψεις του συντελεστή άνωσης των δειγμάτων (ταξινομημένα) σε σύγκριση με τις αξιολογημένες τιμές στον κώδικα ΥΡΔ. (Δεξιά) Οι παράγωγοι του ΒΝΔ όταν έχει εκπαιδευτεί με διαφορετικές συναρτήσεις ενεργοποίησης, σε σύγκριση με τις παραγώγους αναφοράς των πεπερασμένων διαφορών.*



Σχήμα 3: *Πρόβλημα Ι: (Πάνω) Οι βελτιστοποιημένες γεωμετρίες με την προτεινόμενη (μπλε) και τη συζυγή (πορτοκαλί) μέθοδο σε σύγκριση με την αρχική (μαύρη). (Κάτω) Τα πεδία αριθμού Mach για την αρχική γεωμετρία (αριστερά), τη βελτιστοποιημένη με τη χρήση του ΒΝΔ (μέση) και της συζυγούς μεθόδου (δεξιά).*

Βελτιστοποίηση ενός S-bend Αγωγού (Στρωτή Ροή)

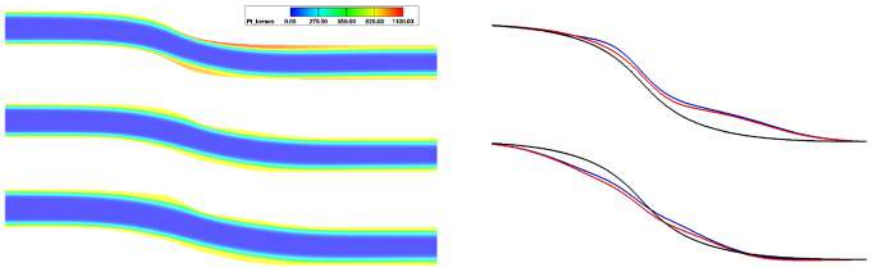
Στην περίπτωση του αγωγού (Πρόβλημα II), ο στόχος είναι η ελαχιστοποίηση των ολικών απωλειών πίεσης μεταξύ εισόδου και εξόδου. Η ροή μέσα στον αγωγό είναι στρωτή με αριθμό Reynolds $R_e = 1.84 \cdot 10^4$. Για την εκπαίδευση του ΒΝΔ δημιουργήθηκαν 50 δείγματα αγωγών και η αρχιτεκτονική του δικτύου προέκυψε από μια διαδικασία δοκιμής-και-σφάλματος με γνώμονα την ακρίβεια τόσο των προβλέψεων όσο και των παραγώγων τους (Σχήμα 4). Εδώ, τα δεδομένα εκπαίδευσης αδιαστατοποιήθηκαν με ελάχιστη τιμή κατά 10% μικρότερη από την ελάχιστη που υπολογίστηκε κατά την αξιολόγηση των δειγμάτων, λόγω της χρήσης της σιγμοειδούς συνάρτησης ενεργοποίησης στο στρώμα εξόδου του ΒΝΔ. Η σιγμοειδής δεν έχει αρνητικό σύνολο τιμών (επομένως δεν επιτρέπει την πρόβλεψη μικρότερης τιμής από την ελάχιστη που υπάρχει στα δείγματα εκπαίδευσης) και χαρακτηρίζεται από το φαινόμενο του 'κορεσμού εξαφάνισης', κατά το οποίο οι παράγωγοι της συνάρτησης εκμηδενίζονται όταν η τιμή της είναι κοντά στο 0 (επομένως αλλοιώνει τις παραγώγους που χρησιμοποιούνται στη βελτιστοποίηση). Επιπλέον, εδώ εισάγεται η τεχνική 'cross-validation' για την επικύρωση των δεδομένων εκπαίδευσης, η οποία, λόγω της αποδοτικότητάς της, θα χρησιμοποιηθεί και στα επόμενα προβλήματα. Η προτεινόμενη βελτιστοποίηση οδήγησε στην ίδια μείωση της συνάρτησης-στόχου (Σχήμα 5) με τη συζυγή μέθοδο $(4, 6\%)$, έχοντας το ίδιο υπολογιστικό κόστος. Για πλήρη σύγκλιση, χρειάστηκαν 3 επαν-εκπαιδεύδεις του ΒΝΔ. Τέλος, λόγω του μηδενικού κόστους της καθόδου που οδηγείται από ΒΝΔ, αναδείχθηκε μία εναλλακτική ικανότητα του προτεινόμενου αλγορίθμου: η πραγματοποίηση πολλαπλών καθόδων, ταυτόχρονα, ξεκινώντας από διαφορετικά σημεία αρχικοποίησης, με προοπτικές βελτίωσης τον αιτιοκρατικών μεθόδων βελτιστοποίησης που εξαρτώνται σημαντικά από τα σημεία εκκίνησης.



Σχήμα 4: *Πρόβλημα II: (Αριστερά) Προβλέψεις των απωλειών ολικής πίεσης των δειγμάτων (ταξινομημένα) σε σύγκριση με τις αξιολογημένες τιμές στον κώδικα ΥΡΔ. (Δεξιά) Οι παράγωγοι του ΒΝΔ σε σύγκριση με τις παραγώγους αναφοράς της συζυγούς μεθόδου.*

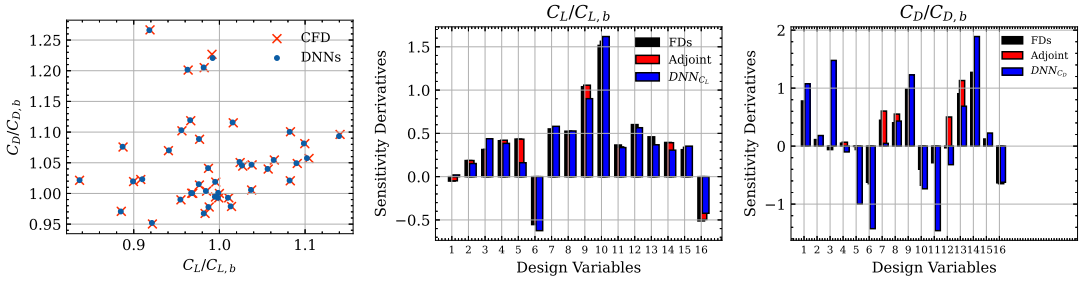
Βελτιστοποίηση μίας Μεμονωμένης Αεροτομής (Τυρβώδης Ροή)

Στην περίπτωση αυτή (Πρόβλημα III), στόχος είναι η ελαχιστοποίηση του συντελεστή οπισθέλκουσας της αρχικής αεροτομής (S8052), χωρίς να μειωθεί ο συντελεστής άνωσης κάτω από το 10% της αρχικής γεωμετρίας. Η ροή γύρω από την αεροτομή είναι τυρβώδης, με αριθμό Reynolds $R_e = 5 \cdot 10^5$, αριθμό Mach της επ' άπειρο ροής 0.021 και γωνία $a = 10^o$. Στο συγκεκριμένο πρόβλημα εκπαιδεύτηκαν δύο ξεχωριστά

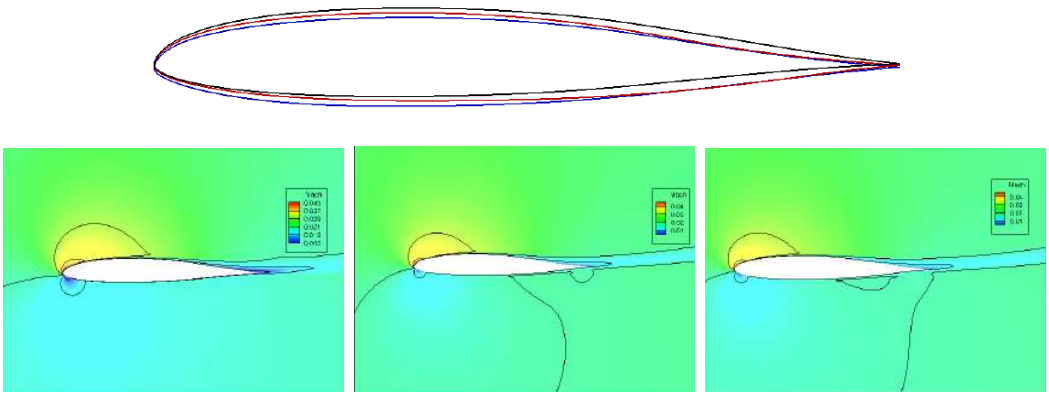


Σχήμα 5: *Πρόβλημα ΙΙ: (Αριστερά) Τα πεδία ολικής πίεσης για την αρχική γεωμετρία (πάνω), τη βελτιστοποιημένη με τη χρήση του ΒΝΔ (μέση) και της συζυγούς μεθόδου (κάτω). (Δεξιά) Οι βελτιστοποιημένες γεωμετρίες με την προτεινόμενη (μπλε) και τη συζυγή (κόκκινη) μέθοδο, σε σύγκριση με την αρχική (μαύρη).*

ΒΝΔ για την πρόβλεψη των συντελεστών άνωσης και οπισθέλκουσας, αντίστοιχα, σε βάση δεδομένων 40 αεροτομών. Οι αρχιτεκτονικές των δικτύων βελτιστοποιήθηκαν χρησιμοποιώντας το λογισμικό EASY, με στόχο την ακρίβεια των υπολογισμένων παραγώγων (Σχήμα 6). Η βελτιστοποίηση της S8052 με την προτεινόμενη μέθοδο οδήγησε σε καλύτερη λύση (Σχήμα 7) από εκείνη της συζυγούς μεθόδου, έχοντας κατά 20% μικρότερο υπολογιστικό κόστος. Εδώ δεν χρειάστηκαν επαν-εκπαιδεύσεις των ΒΝΔ. Η βελτιστοποιημένη γεωμετρία είχε κατά 4% χαμηλότερη τιμή συντελεστή οπισθέλκουσας από την αρχική (ενώ ταίριαζε στην τιμή στόχου συντελεστή άνωσης με απόκλιση $0,1\%$), σε σύγκριση με τη λύση της συζυγούς μεθόδου που είχε κατά 3% χαμηλότερη τιμή συντελεστή οπισθέλκουσας, αλλά με καλυτερη προσέγγιση του στόχου (με απόκλιση 0,01 %). Επιπλέον, πραγματοποιήθηκε παραμετρική μελέτη σχετικά με το μέγεθος της βάσης δεδομένων που χρησιμοποιείται για την εκπαίδευση των ΒΝΔ, προκειμένου να μελετηθεί η επίδραση στην ακρίβεια τόσο των προβλέψεων, όσο και των παραγώγων. Από τη μελέτη έγινε φανερό ότι η ακρίβεια των ΒΝΔ βελτιώνεται όσο το μέγεθος της βάσης δεδομένων αυξάνεται μέχρι ένα συγκεκριμένο αριθμό δειγμάτων, ενώ δεν παρατηρείται σημαντική βελτίωση πάνω από τον αριθμό αυτό.



Σχήμα 6: *Πρόβλημα ΙΙΙ: (Αριστερά) Προβλέψεις των συντελεστών άνωσης και οπισθέλκουσας των δειγμάτων, σε σύγκριση με τις αξιολογημένες τιμές στον κώδικα ΥΡΔ. Οι παράγωγοι των συντελεστών άνωσης (Μέση) και οπισθέλκουσας (Δεξιά) από τα ΒΝΔ σε σύγκριση με τις πεπερασμένες διαφορές και της συζυγούς μεθόδου.*



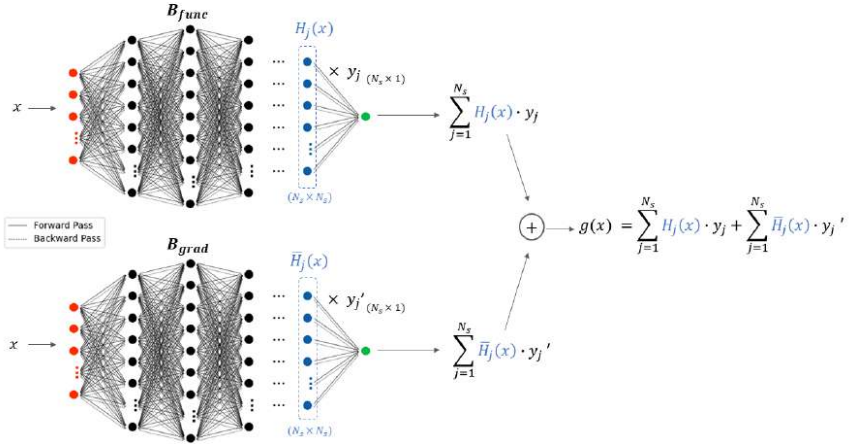
Σχήμα 7: *Πρόβλημα ΙΙΙ: (Πάνω) Οι βελτιστοποιημένες γεωμετρίες με την προτεινόμενη (μπλε) και τη συζυγή (κόκκινη) μέθοδο σε σύγκριση με την αρχική (μαύρη). (Κάτω) Τα πεδία αριθμού Mach για την αρχική γεωμετρία (αριστερά), την βελτιστοποιημένη με τη χρήση του ΒΝΔ (μέση) και της συζυγούς μεθόδου (δεξιά).*

Βελτιστοποίηση μίας Μεμονωμένης Αεροτομής (Τυρβώδης Ροή), με ΒΝΔ Εκπαιδευμένα με τις Υλοποιήσεις Sobolev και Hermite

Υλοποίηση Sobolev: Κατά την υλοποίηση Sobolev, [12], οι παράγωγοι του ΒΝΔ ως προς τις εισόδους του υπολογίζονται κατά τη διάρκεια της εκπαίδευσης, και συγκρίνονται (ταυτόχρονα με τη σύγκριση των προβλέψεων με τις τιμές-στόχους) με τις παραγώγους αναφοράς υπολογισμένες με τη συζυγή μέθοδο. Η σύγκριση γίνεται προσθέτοντας τόσους επιπλέον όρους στη συνάρτηση κόστους (με κατάλληλη στάθμιση), όσοι και οι (μερικές) παράγωγοι ως προς τις μεταβλητές εισόδου.

Υλοποίηση Hermite: Η υλοποίηση Hermite, [13], βασίζεται στην ιδέα της αριθμητικής παρεμβολής Hermite, κατά την οποία ορίζονται δύο οικογένειες ορθογωνίων πολυωνύμων βάσης, έτσι ώστε μόνο η πρώτη να συνεισφέρει στην τελική συνάρτηση παρεμβολής (για την παρεμβολή των τιμών-στόχων των δειγμάτων), ενώ όταν αυτή παραγωγιστεί, να συνεισφέρει μόνο η δεύτερη οικογένεια (για την παρεμβολή των παραγώγων τους). Για την υλοποίηση με χρήση ΒΝΔ, ορίζονται δύο ξεχωριστοί κλάδοι δικτύων, ώστε να μοντελοποιήσουν τις δύο αυτές οικογένειες πολυωνύμων βάσης. Οι δύο κλάδοι αποτελούνται από ένα στρώμα εισόδου, έναν αριθμό κρυφών στρωμάτων (ο αριθμός τους μπορεί να διαφέρει για κάθε κλάδο), ένα στρώμα Βάσης′ και ένα στρώμα εξόδου. Στο στρώμα Βάσης′, ορίζονται τόσοι νευρώνες όσος και ο αριθμός των δειγμάτων εκπαίδευσης, ώστε να προσομοιώσουν το αποτέλεσμα των πολυωνύμων βάσης. Στη συνέχεια, τα (μη-εκπαιδεύσιμα) βάρη του στρώματος εξόδου εξισώνονται με τις τιμές-στόχους των δειγμάτων, στον πρώτο κλάδο, και με τις τιμές των παραγώγων τους, στον δεύτερο. Οι εξόδοι από τους δύο κλάδους προστίθενται, διαμορφώνοντας έτσι την τελική συνάρτηση παρεμβολής κατά αναλογία με αυτή της αναλυτικής παρεμβολής Hermite (Σχήμα 8). Η εκπαίδευση γίνεται κι εδώ με μία συνάρτηση κόστους τύπου Sobolev, μόνο που οι τιμές-στόχοι των δειγμάτων συγκρίνονται με την τελική

έξοδο του προτεινόμενου ΒΔΝ (άθροισμα εξόδων των δύο κλάδων) και οι παράγωγοι αναφοράς, συγκρίνονται με τις παραγώγους της συνάρτησης αυτής. Στην υλοποίηση με χρήση ΒΝΔ, οι δύο κλάδοι συνεισφέρουν ισότιμα και συμπληρωματικά στην τελική παρεμβολή (σε αντίθεση με την αναλυτική μέθοδο) και η κάθε συνεισφορά εξαρτάται σημαντικά από την αρχιτεκτονική του κάθε κλάδου και κυρίως, από τις συναρτήσεις ενεργοποίησης. Ένα πλεονέκτημα της προτεινόμενης υλοποίησης με τα ΒΝΔ είναι ότι μπορεί εύκολα να επεκταθεί για την παρεμβολή πολλών μεταβλητών εισόδου.

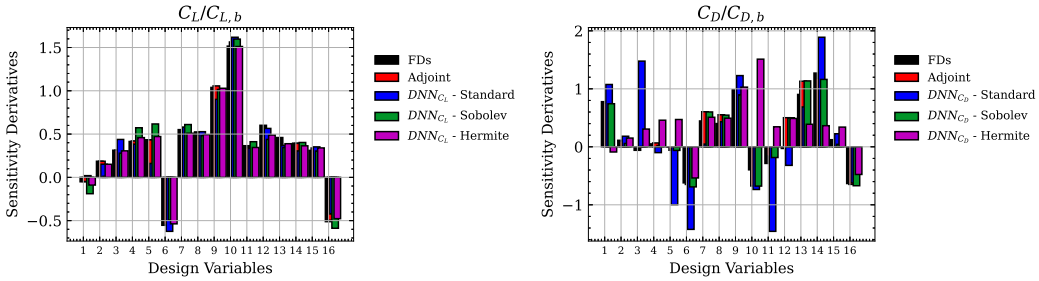


Σχήμα 8: *Προτεινόμενη Αρχιτεκτονική ΒΝΔ για την υλοποίηση Hermite.*

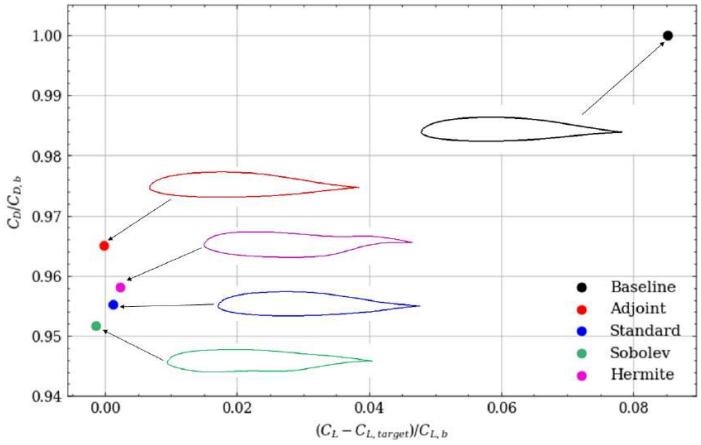
Ο προτεινόμενος αλγόριθμος με τις υλοποιήσεις Sobolev κσι Hermite εφαρμόστηκε στη βελτιστοποίηση της αεροτομής S8052 (ίδιες συνθήκες ροής και ίδια συνάρτηση-στόχου). Χρησιμοποιήθηκαν δύο ΒΝΔ για την πρόβλεψη των συντελεστών άνωσης και οπισθέλκουσας αντίστοιχα,, τα οποία εκπαιδεύτηκαν σε βάση δεδομένων 25 αεροτομών, Οι παράγωγοι ευαισθησίας των δειγμάτων υπολογίστηκαν με τη συζυγή μέθοδο και αρχιτεκτονικές των δικτύων βελτιστοποιήθηκαν χρησιμοποιώντας τον EASY, με γνώμονα την ακρίβεια των προβλέψεων και παραγώγων τους (Σχήμα 9.) Η νέες υλοποιήσεις της μεθόδου οδήγησαν σε καλύτερη λύση σε σύγκριση με τη συζυγή μέθοδο. Ομοίως κι εδώ, δεν χρειάστηκαν επαν-εκπαιδεύσεις των ΒΝΔ. Το κόστος της βελτιστοποίησης ήταν το ίδιο με της συζυγούς μεθόδου και ελαφρώς υψηλότερο από το κόστος της αρχικής υλοποίησης. Η λύση της υλοποίησης Sobolev είχε 5% χαμηλότερη τιμή συντελεστή οπισθέλκουσας (με απόκλιση $0,01\%$ από τον στόχο συντελεστή άνωσης) και η λύση της υλοποίησης Hermite είχε 4% χαμηλότερη τιμή συντελεστή οπισθέλκουσας (με απόκλιση 0,02 % από τον στόχο).

Συμπεράσματα

Από τις μελέτες συμπεραίνεται ότι τα ΒΝΔ μπορούν αποτελεσματικά να αντικαταστήσουν τόσο τον επιλύτη ροής, όσο και των συζυγών εξισώσεων, προβλέποντας την τιμή της συνάρτησης-στόχου και των παραγώγων ευαισθησίας, όταν χρησιμοποιηθούν σε αιτιοκρατικές μεθόδους βελτιστοποίησης στο πεδίο της ΥΡΔ. Αφού εκπαιδευτούν, τα ΒΝΔ μπορούν να χρησιμοποιηθούν ευέλικτα σε πολλές βελτιστοποιήσεις (μίας



Σχήμα 9: *Πρόβλημα ΙΙΙ: Οι παράγωγοι των συντελεστών άνωσης (Αριστερά) και οπισθέλκουσας (Δεξιά) από τα ΒΝΔ εκπαιδευμένα κατά Sobolev και Hermite σε σύγκριση με τις παραγώγους των αρχικών ΒΝΔ , των πεπερασμένων διαφορών και της συζυγούς μεθόδου.*



Σχήμα 10: *Πρόβλημα ΙΙΙ: Οι βελτιστοποιημένες γεωμετρίες από κάθε υλοποίηση του προτεινόμενου αλγορίθμου συγκρίνονται με την βελτιστοποιημένη γεωμετρία από τη συζυγή μέθοδο και την αρχική. Οι λύσεις τοποθετούνται στον χώρο των συντελεστών άνωσης-οπισθέλκοθσας για σύγκριση.*

συγκεκριμένης γεωμετρίας σε συγκεκριμένες συνθήκες), με διαφορετικές συναρτήσεις-στόχους. Η ακρίβεια των παραγώγων των ΒΝΔ όσο και η αποτελεσματικότητα της προτεινόμενης βελτιστοποίησης εξαρτώνται σημαντικά από τις συναρτήσεις ενεργοποίησης, όπου η GELU αναδείχθηκε η πιο κατάλληλη. Η αύξηση του μεγέθους της βάσης δεδομένων βελτιώνει την ακρίβεια των δικτύων, αλλά αυξάνει ταυτόχρονα το υπολογιστικό κόστος της βελτιστοποίησης. Συνεπώς, η εκπαίδευση των ΒΝΔ σε μικρές βάσεις δεδομένων είναι πιο συμφέρουσα, αφού η ακρίβεια της τελικής λύσης μπορεί να βελτιωθεί με επαν-εκπαιδεύσεις των δικτύων κατά τη βελτιστοποίηση. Τέλος, η εισαγωγή των παραγώγων ευαισθησίας στην εκπαίδευση των ΒΝΔ βελτίωσε τόσο την ακρίβεια των παραγώγων τους, όσο και την απόδοση της προτεινόμενης βελτιστοποίησης. Η νέα υλοποίηση προτείνεται έναντι της αρχικής, αν βέβαια η συζυγής μέθοδος για τον υπολογισμό των παραγώγων των δειγμάτων είναι διαθέσιμη.

Βιβλιογραφία

[1] State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938, 2018.

[2] M. Kontou, V. Asouti, and K. Giannakoglou. DNN surrogates for turbulence closure in CFD-based shape optimization. *Applied Soft Computing*, 134:110013, 2023.

[3] K. Kovani, M. Kontou, V. Asouti, and K. Giannakoglou. Dnn-driven gradient-based shape optimization in fluid mechanics. In L. Iliadis, I. Maglogiannis, S. Alonso, C. Jayne, and E. Pimenidis, editors, *Engineering Applications of Neural Networks*, pages 379–390, Cham, 2023. Springer Nature Switzerland.

[4] K. Giannakoglou and D. Papadimitriou. *Adjoint Methods for Shape Optimization*, pages 79–108. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[5] X. Trompoukis, K. Tsiakas, V. Asouti, and K. Giannakoglou. Continuous adjoint-based shape optimization of a turbomachinery stage using a 3D volumetric parameterization. *International Journal for Numerical Methods in Fluids*, 2023.

[6] L. Piegl and W. Tiller. *The NURBS Book (2nd Ed.)*. Springer-Verlag, Berlin, Heidelberg, 1997.

[7] Uncertainty quantification method for cfd applied to the turbulent mixing of two water layers. *Nuclear Engineering and Design*, 333:1–15, 2018.

[8] J. Pizarroso, J. Portela, and A. Muñoz. Neuralsens: Sensitivity analysis of neural networks. *Journal of Statistical Software*, 102(7), 2022.

[9] TensorFlow: Large-Scale Machine Learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[10] A. Baydin, B. Pearlmutter, A. Radul, and J. Siskind. Automatic differentiation in machine learning: a survey, 2018.

[11] K. Giannakoglou. The EASY (Evolutionary Algorithms SYstem) software, http://velos0.ltt.mech.ntua.gr/EASY, 2008.

[12] W. Czarnecki, S. Osindero, M. Jaderberg, G. Świrszcz, and R. Pascanu. Sobolev training for neural networks, 2017.

[13] http://velos0.ltt.mech.ntua.gr/kgianna/analysis/distr/book-numanal.pdf.