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 optimization in single- and multi-phase flows

Diploma Thesis

Eirini-Sotiria Kefaloukou

Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2025

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Professor Kyriakos Giannakoglou for his trust, guidance and support throughout the implementation of my diploma thesis and in broader academic matters. His ability to convey his knowledge was vital in resolving complex scientific challenges. His expertise and experience have been an inspiration to me during all the years of my studies, and I feel honored to have worked under his supervision.
I would also like to express my sincere gratitude to Dr. Marina Kontou. Her scientific expertise, continuous guidance, and support were decisive during the execution of my diploma thesis. I am grateful for the knowledge she has imparted to me, which has helped me grow as an engineer. I would like to thank PhD candidate Spyros Stalikas for his guidance and continued support.
Finally, I would like to express my gratitude to my family, who, throughout my studies, was by my side with their love and patience.

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 optimization in single- and multi-phase flows

Diploma Thesis

Eirini-Sotiria Kefaloukou

Advisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2025

Abstract

Target of this diploma thesis is the use of Deep Neural Networks (DNNs) in gradient-based Shape Optimization (ShpO), as low-cost surrogates of the primal and adjoint computations, thus reducing the optimization overall cost.
The DNNs are trained on databases containing both the objective function values and their corresponding Sensitivity Derivatives. The networks architecture is inspired by the notion of the Hermite polynomials, since besides the objective function, incorporate gradients in their training process. The computation of gradients is accomplished by differentiating the networks outputs with respect to their inputs, using automatic differentiation in reverse mode. The accuracy of the computed gradients is verified against reference values of the adjoint method. Improving the networks generalization capabilities and reducing the cost of constructing their database is also investigated.
Primary goal of this diploma thesis is the integration of the Hermite-DNNs in the gradient-based ShpO, so as to provide an approximation to the objective function values and derivatives. During the ShpO, the DNN-optimized solutions are re-evaluated on the Computational Fluid Dynamics code. If necessary, this design is incorporated in the database and the networks are re-trained. All implementations are related to ShpO studies in single- and multi-phase flows. Two turbomachinery applications are presented, the first concerns a turbine blade-airfoil (single-phase turbulent flow), and the second a compressor blade-airfoil (single-phase turbulent flow). The proposed gradient-based optimization algorithm is implemented also in the design of an isolated airfoil (single-phase transitional flow) and a hemispherical-cylinder body (two-phase cavitating flow).

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

Βαθέα Νευρωνικά Δίκτυα και η διαφόρισή τους για χρήση σε αιτιοκρατική βελτιστοποίηση μονοφασικών και πολυφασικών ροών

Διπλωματική Εργασία

Ειρήνη-Σωτηρία Κεφαλούκου

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2025

Περίληψη

Στόχος της παρούσας διπλωματικής εργασίας είναι η χρήση Βαθέων Νευρωνικών Δικτύων (ΒΝΔ) σε αιτιοκρατική βελτιστοποίησης μορφής σωμάτων στη μηχανικών των ρευστών, ως χαμηλού-κόστους υποκατάστατα για την επίλυση του πρωτεύοντος και συζυγούς προβλήματος, με στόχο τη μείωση του συνολικού κόστους της βελτιστοποίησης.

Τα ΒΝΔ εκπαιδεύονται σε βάσεις δεδομένων που περιέχουν τόσο τις τιμές της αντικειμενικής συνάρτησης όσο και των αντίστοιχων παραγώγων ευαισθησίας. Η αρχιτεκτονική των δικτύων είναι εμπνευσμένη από τα πολυώνυμα *Hermite* τα οποία εκτός από την αντικειμενική συνάρτηση, συμπεριλαμβάνουν και τις παραγώγους ευαισθησίας στην διαδικασία εκπαίδευσης τους. Η μοντελοποίηση των παραγώγων ευαισθησίας επιτυγχάνεται με τον υπολογισμό των παραγώγων των εξόδων των δικτύων ως προς τις εισόδους τους, χρησιμοποιώντας αντίστροφη αυτόματη διαφόριση. Η ακρίβεια των υπολογιζόμενων παραγώγων ευαισθησίας επικυρώνεται έναντι τιμών αναφοράς της συζυγούς μεθόδου. Επίσης η έρευνα εστιάζει στην βελτίωση των δυνατοτήτων γενίκευσης των ΒΝΔ και στη μείωση του κόστους κατασκευής της βάσης δεδομένων τους.

Πρωταρχικός στόχος είναι η ενσωμάτωση των *Hermite*-ΒΝΔ στην αιτιοκρατική βελτιστοποίηση μορφής, έτσι ώστε να παρέχουν μια προσέγγιση για τις τιμές της αντικειμενικής συνάρτησης και των παραγώγων ευαισθησίας της. Κατά την διάρκεια της βελτιστοποίησης μορφής, οι βελτιστοποιημένες λύσεις από τα ΒΝΔ επαναξιολογούνται από τον κώδικα Υπολογιστικής Ρευστοδυναμικής. Εάν χρειάζεται, η γεωμετρία συμπεριλαμβάνεται στη βάση δεδομένων και τα δίκτυα επανεκπαιδεύονται. Όλες οι υλοποιήσεις αφορούν μελέτες βελτιστοποίησης μορφής σε μονοφασικές και πολυφασικές ροές. Παρουσιάζονται εφαρμογές στροβιλομηχανών, η πρώτη αφορά αεροτομή πτερυγίου στροβίλου (τυρβώδης μονοφασική ροή) και η δεύτερη αεροτομή πτερυγίου συμπιεστή (τυρβώδης μονοφασική ροή). Ο προτεινόμενος αλγόριθμος αιτιοκρατικής βελτιστοποίησης εφαρμόζεται επίσης σε μεμονωμένη αεροτομή (μονοφασική ροή με μετάβαση) και ημισφαιρικό-κύλινδρο σώμα (διφασική ροή με σπηλαίωση).

Contents

Chapter 1

Artificial Intelligence

1.1 Machine Learning

Artificial Intelligence (AI) is an evolving field focused on developing agents that emulate human intelligence. In recent years, it has experienced rapid growth, enabling AI-based systems to perform complex tasks such as perception, learning, problem-solving, and decision-making without human intervention. These systems are applied across a wide range of domains, including healthcare, engineering, and finance. Recent advances have led to the integration of AI in technologies such as robotics, autonomous vehicles, web search engines, virtual assistants, and beyond. ML is a subset of AL that focuses on developing systems capable of performing tasks autonomously. ML algorithms are trained to identify patterns within data and use them to make predictions, classify information, and generalize based on previously learned knowledge. Deep Learning (DL), a specialized subfield of ML, accomplishes these goals through the use of multi-layered Artificial Neural Networks (ANNs) ([1]). The hierarchal relationship is presented in Figure 1.1.

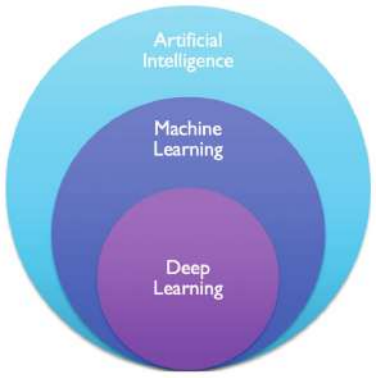


Figure 1.1: *Subfields of AI. Figure from (https://www.researchgate.net/figure/Artificial-intelligence-and-its-subsets_fig1_370470867).*

ML approaches, as illustrated in Figure 1.2, are traditionally divided into three broad categories.

1. **Supervised Learning:** Supervised learning algorithms construct a mathematical model of the labeled training data. Each training pattern consists of input features paired with a corresponding output. Through iterative optimization, these algorithms establish relationships between inputs and the associated outputs, in order to minimize the output prediction error. Supervised learning is applied to two types of problems: classification, where inputs are assigned to binary or multi-class categories, and regression that predicts continuous outputs. Common supervised learning algorithms include regression, random forests, Support Vector Machines (SVM), k-Nearest Neighbors (kNN), and Artificial Neural Networks each with different properties and training configurations, as detailed in ([2]).
2. **Unsupervised Learning:** Unsupervised learning algorithms work with unlabeled data. They identify patterns without explicit guidance, and react on new inputs based on the absence or presence of these patterns. Some applications of unsupervised learning include density estimation of observed data, clustering, where data are categorized based on their similarities, and dimensionality reduction, which transforms the high-dimensional data in low-dimensional, retaining only a subset of the original information ([2]).
3. **Reinforcement Learning:** Reinforcement learning algorithms learn through a trial-and-error process. They interact with the environment, develop through feedback from that and adapt their actions to maximize a reward signal. They are used in application regarding autonomous vehicles, robotics, and game playing.

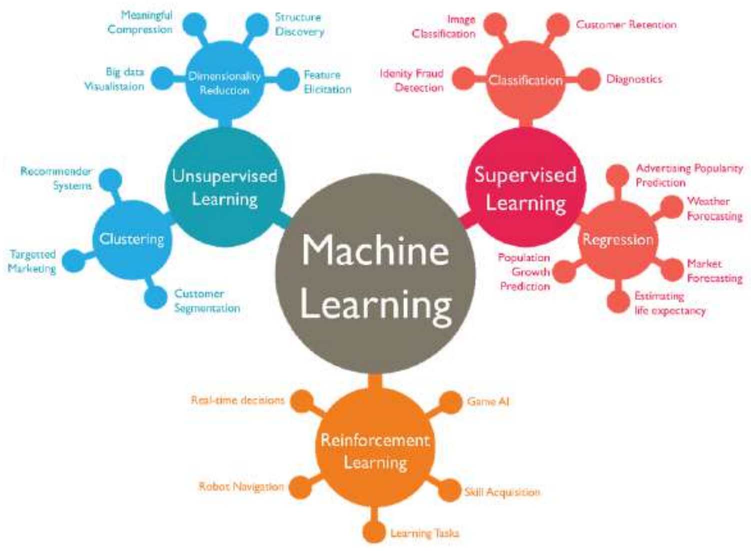


Figure 1.2: *An overview of different types of ML algorithms. Figure from (`https://7wdata.be/visualization/types-of-machine-learning-algorithms-2/`).*

1.2 Artificial Neural Networks

The concept of Artificial Neural Networks (ANNs) ([3]) is inspired by the structure and function of biological neurons in the human brain. A biological neuron comprises three primary components: dendrites, a soma (cell body), and an axon. The dendrites interact with the neighboring neurons, and receive signals from them. The importance of each connection is determined by its synaptic strength, which is learnable and can be adapted. The signals are summed to the soma, and an output is produced. If the output is above a certain threshold the neuron can fire and send an output signal, that travels though its axon and interacts with the dendrites of other neurons. The analogy between biological and artificial neuron is shown in Figure 1.3.

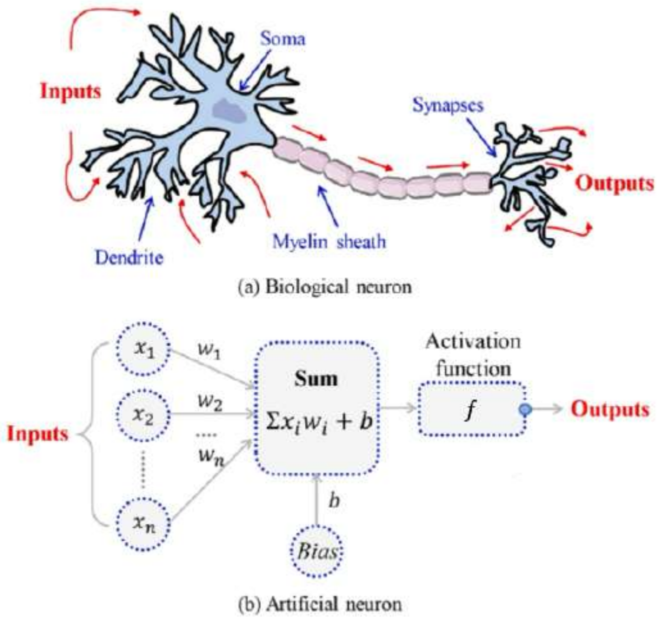


Figure 1.3: *A biological neuron (top) and a neuron of a DNN inspired from that (bottom). Figure from (`https://www.researchgate.net/publication/363833676_Advance_Artificial_Neural_Networks`)*

An artificial neuron has similar structure; it transforms an input vector into a scalar output. It receives an N-dimensional input vector x ϵ $R^N = [x_1, x_2, ..x_N]^T$, and multiplies it with a weight matrix w=$[w_1, w_2, ..w_N]$, often referred to as kernel. This kernel determines the influence of each input feature on the neuron's output. A bias term (b) —a learnable scalar— is typically added, that shifts the activation to the desired direction. The sum passes through a non-linear activation function (f). The activation function decides when the neuron should be activated, and enables the representation of complex, non-linear relationships. Without this non-linearity, the network would be limited to linear mappings, regardless of its depth. The computations performed by a single neuron can be expressed as:

$$\sigma = \sum_{i=1}^{N} x_i w_i \tag{1.1}$$

$$a = f(\sigma + b) \tag{1.2}$$

Figure 1.4 shows the architecture of a feed forward neural network (FFNN). It consists of a series of layers that sequentially process inputs to produce outputs. Each one has a specific purpose:

• **Input Layer** The input layer receives the input patterns; it does not perform any computations, instead it passes them to the proceeding layer. Its dimensionality is determined from the number of features in the input data, with each node corresponding to one feature.

• **Hidden Layers** After the input layer, a series of hidden layers follow, that perform most of the computations. Each hidden layer receives inputs from the preceding layer, processes them, and passes its outputs to the next. Each layer is capable of learning different features. Layers can be either fully connected, also referred to as dense layers, or partially connected. In Fully Connected Neural Networks (FCNNs), each neuron is connected to all neurons in the preceding and succeeding layers. In contrast, Partially Connected Neural Networks (PCNNs [4]) introduce sparsity, by skipping some connections. Each connection maintains a learnable weight, that determines its importance. The number of hidden layers is a user-defined design parameter, typically determined by the complexity of the task. Increasing the depth of the model enhances its ability to learn complex tasks.

• **Output Layer** The output layer aggregates the previous layer's outputs, to form the final predictions. Its dimensionality matches the number of the targets.

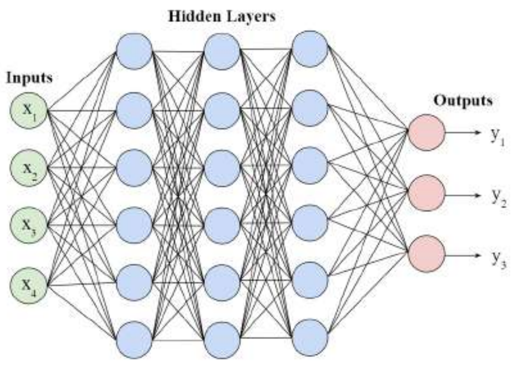


Figure 1.4: *Architecture of a dense network with 4 input units, 3 hidden layers, and 3 output neurons.*

1.3 Machine Learning in Computational Fluid Dynamics and Optimization

Computational Fluid Dynamics (CFD) is a branch of fluid dynamics that employs numerical methods to analyze and solve problems involving fluid flow. Despite their effectiveness, CFD methods come with significant computational cost, influenced by factors such as the chosen algorithms and mesh resolution. The integration of ML techniques in CFD applications has been adopted, due to their ability to learn complex representations and provide cost-efficient surrogates of the traditional solvers. ML applications in CFD can be divided in two main categories: Data-Driven surrogates, and ML-assisted numerical simulations.
Data-Driven surrogates are employed to replace the CFD solutions. For instance, ([5]) uses DNNs to predict the pressure field around a wing, and to substitute the CFD-solver in the Evolutionary Algorithm (EA)-based Shape Optimization (ShpO) of the wing. ([6]) employs Radial Basis Function networks for uncertainty quantification and EA-based ShpO of an isolated airfoil and a wing. The proposed method quantifies the uncertainties regarding the constants of the $\gamma - \tilde{Re}_{\theta t}$ transition model and the roughness of the solid surfaces, and achieved a reduction in the optimization time by orders of magnitude. In ([7]) DNNs trained on high fidelity data obtained with DNS, predict the Reynolds stresses on a turbulent flow in a channel and a stationary passage. ([8]) uses Physics-Informed Neural Networks (PINNs) to solve inverse problems in 3D wake and supersonic flows. PINNs, which are constrained to satisfy the Navier-Stokes equations, approximate the velocity and pressure fields and calculate all required derivatives via automatic differentiation (AD).
ML models can be coupled with the CFD-solver, in order to accelerate the simulations. For example, ([5]) uses DNNs in a conjugate heat transfer problem, dealing with a solid domain in contact with a flow within a duct. In this case, the DNNs replace the solver of the heat conduction equation on the solid domain. In ([9]) DNNs substitute the numerical solution of the turbulence and transition models, by predicting the turbulence viscosity fields. This DNN-assisted RANS solver is applied to the shape optimization of a turbine blade and a car model, resulting in a significant reduction in optimization turnaround time. ([10]) uses DNNs, trained on high-fidelity data generated from simulations on fine grids, to approximate the convection term in the Navier–Stokes equations on coarser meshes. This approach achieves up to a tenfold computational speedup while maintaining solution quality comparable to that of the original high-resolution simulations. An overwiew of the recent advantages of machine learning applications in CFD are presented in ([11], [12]).
Gradient-based optimization algorithms require the computation of objective function's derivatives, which can be obtained through Finite Differences, Direct Differentiation, Complex Variable Method, or the Adjoint method. The computational cost of obtaining the Sensitivity Derivatives (SDs) varies significantly with the chosen approach and can, in some cases, become prohibitively high. When an adjoint

solver is available, the cost of computing SDs becomes independent of the number of design variables and is approximately equal to that of solving the primal problem. In this diploma thesis, attempts for cost reduction are made, by using DNNs as data-driven surrogates of the primal and adjoint solvers. The DNNs, often referred to as Hermite-DNNs, are trained on databases containing the target values and their derivatives. Inputs are the coordinates of the nodes parameterizing the geometry shape, and outputs the target objective values. The SDs are modeled using the derivatives of the networks outputs w.r.t. their inputs, computed via AD. Since the networks are integrated in a gradient-based optimization process, it is essential that their derivatives predictions are as accurate as possible. To achieve this, their configuration is optimized with an EA-based process, seeking to minimize their prediction error.
This DNN-Driven gradient-based optimization was originally introduced in ([13]). In the diploma thesis, its extension to multiphase flow problems is explored. Additionally, attempts are made to enhance the generalization performance of the networks, and various strategies for constructing the training database are examined, with a focus on the computational cost of the database generation.

1. **Chapter 2:** An overview of the training process of the DNNs is provided, highlighting the various hyperparameters that can be tuned during training and the differentiation of the models. Finally, the architecture of the Hermite-DNN is presented in detail.

2. **Chapter 3:** The proposed DNN-driven gradient-based optimization algorithm is presented.

3. **Chapter 4:** The proposed methodology is demonstrated to the ShpO of a turbine blade-airfoil (C3X). The flow is single-phase and turbulent. Different database construction approaches are compared, focusing primarily on their computational demands. Various DNN configurations are evaluated based on their generalization performance. The quality of the DNN-optimized solutions is compared to the solutions obtained with the adjoint-driven ShpO.

4. **Chapter 5:** The proposed DNN-driven algorithm is applied to the ShpO of an isolated airfoil (RG15). The flow around the airfoil is single-phase and transitional. Attempts are made to enhance the networks generalization performance and reduce their database construction cost. The obtained solutions are compared to those of the adjoint-driven optimization, regarding their quality and the computational time required to obtain them.

5. **Chapter 6:** The proposed algorithm is applied to the gradient-based ShpO of a compressor blade-airfoil under single-phase turbulent flow. Its performance is compared to the adjoint-based optimization in terms of computational cost.

6. **Chapter 7:** The generalization of this method to multi-phase flows is investigated. The implementation concerns the ShpO of a hemispherical-cylinder

body. The solutions obtained from the DNN-driven phase are compared to the adjoint-driven ShpO in terms of solution quality.

Chapter 2

Deep Neural Networks

2.1 Training Process

The DNN's training process is formulated as a gradient-based optimization problem. In each iteration, the input data propagate through the network's layers to generate predictions. The discrepancy between these predictions and the target values is quantified using a loss function. The gradients of the loss function w.r.t. each parameter are computed with AD and are subsequently employed to update the model's weights. A complete pass of the entire dataset through the network constitutes one epoch. The hyperparameters that will be tuned during training are discussed below. The section 2.2 presents the architecture of the DNNs employed in this diploma thesis.

2.1.1 Weight Initialization

At the beginning of training, the network's weights are initialized with small random values ([14]), often sampled from a uniform distribution within each layer. As with any other gradient-based optimization algorithm, DNNs are sensitive to the initialization choice. Different initializations lead the optimization algorithm along different trajectories, potentially resulting in models with varying performance.

2.1.2 Loss functions

Loss functions offer a way to asses the model's predictions, by quantifying their discrepancy from the target values. Numerous loss functions have been proposed; the most common ones for regression tasks are Mean Squared Error and Mean Absolute Error, each offering different advantages. The choice between them depends on the nature of the problem and the characteristics of the database, as explained below.

- **Mean Squared Error (MSE)**

  MSE ([15]) measures the average of the squared differences between N predicted values $\hat{S} = [\hat{y_1}, \hat{y_2}, \dots \hat{y_N}]$ and the targets $S = [y_1, y_2, \dots y_N]$.

$$L_{MSE}(\hat{S}, S) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2 \tag{2.1}$$

  The MSE is differentiable at all points, which ensures a smooth and stable training process. By computing squared differences between predicted and target values, MSE amplifies larger errors. This characteristic makes it suitable when the presence of outliers needs to be penalized. In contrast, in scenarios where outliers should have minimal influence, or when the dataset contains noisy samples, the use of MSE might skew predictions and subsequently affect the model's performance.

- **Mean Absolute Error (MAE)**

  MAE ([15]) computes the average of the absolute differences between N predicted values $\hat{S} = [\hat{y_1}, \hat{y_2}, \dots \hat{y_N}]$ and the targets $S = [y_1, y_2, \dots y_N]$.

$$L_{MAE}(\hat{S}, S) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y_i}| \tag{2.2}$$

  The MAE treats all errors equally. It is less sensitive to outliers and prevents them from influencing the training process disproportionately. Consequently, MAE would be more preferable when the dataset contains noisy samples ([16]). It is not differentiable at zero, however such small errors are rarely encountered during training.

2.1.3 Differentiation of DNNs and Backpropagation

After computing the prediction error, the next step involves computing its partial derivatives w.r.t. each of the network's parameters (weights and biases). The forward pass through the network can be expressed as:

$$a^l = b^l + h^{l-1}W^l \tag{2.3}$$

$$h^l = f^l(a^l) \tag{2.4}$$

where f^l, W^l, and b^l denote the activation function, weight matrix, and bias vector of the l-th layer, respectively, where l ϵ [0, ...,L]. For l=0, the tensor h^0 contains the input features of the network. At the final layer l=L, the output tensor h^L corresponds to the network's predictions ($\hat{y}$).

The gradients of the error w.r.t. each parameter are calculated using reverse-mode automatic differentiation (RAD) ([17], [18]). To implement this, all intermediate computations during the forward pass must be recorded. Then, the list of operations in reversed, and the error is propagated backwards, starting from the output and proceeding to the inputs. The back-propagation algorithm is presented below.

1. The gradient of the loss function w.r.t. the predictions for each training pattern is computed:
$$g = \nabla_{\hat{y}} L$$
2. The gradient is propagated backwards, through the activation function, to obtain the gradient w.r.t. the pre-activation values::
$$g \leftarrow \nabla_{a^l} J = g \cdot f'(a^l)$$
3. The gradients w.r.t. the weight and biases of the l-th layer are computed as:
$$\nabla_{b^l} J = g$$
$$\nabla_{W^l} J = h^{(l-1)T} g$$
4. Finally, the gradient is propagated to the previous layer's activations:
$$\nabla_{h^{l-1}} J = g W^{lT}$$

Steps 2-4 are repeated for all the hidden layers. Similarly to the backpropagation of the loss, the gradients of the networks outputs w.r.t. the input variables can be computed. Figure 2.1 presents the computations during the forward and backward pass on a two-layer DNN, including the computation of the gradients w.r.t. the inputs. The accuracy of these gradients, depends on the network's topology (number of neurons and layers), the resulting weights, hence the training configuration, and the activation function's outputs and derivatives.

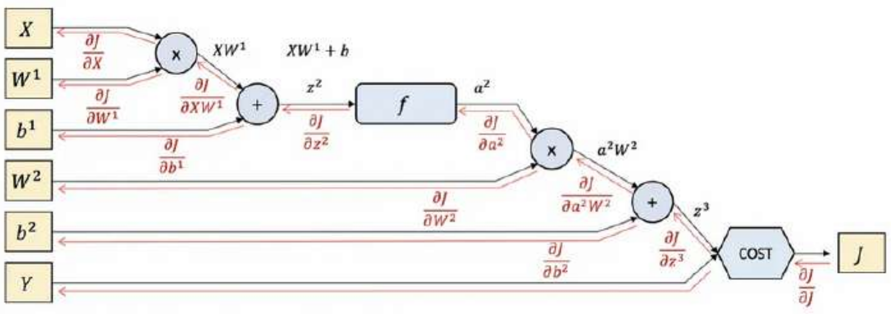


Figure 2.1: *Computational graph of a two-layer DNN. Forward and backward propagations are presented. Figure from (`https://medium.com/data-science/neural-networks-iv-the-graph-approach-cb25590a7f24`).*

2.1.4 Activation Functions

This Section provides an overview of the activation functions considered in this work. The linear activation function returns the input x scaled by a constant c. Since it does not introduce any non-linearity, its use can limit the network's ability to learn complex representations.

$$linear(x) = cx \tag{2.5}$$

The sigmoid activation function ([19]) is defined as:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

It exhibits a smooth, S-shape curve and maps every input between [0,1]. The hyperbolic tangent (tanh) activation function ([19]) is given by:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.7}$$

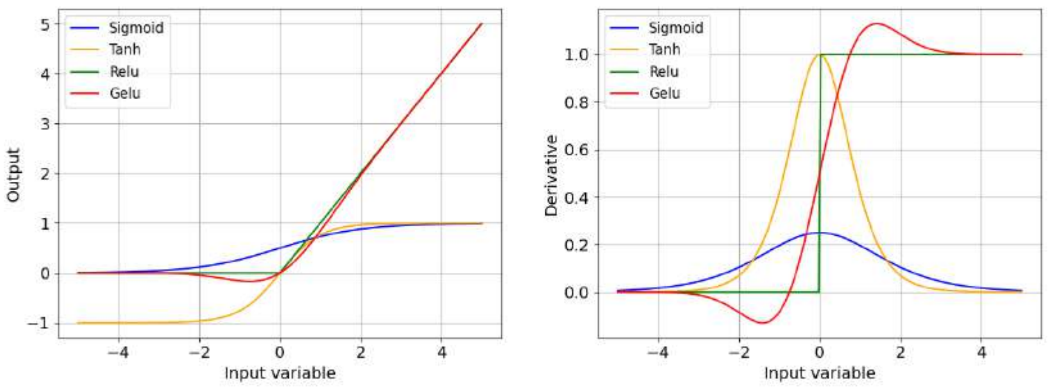
Like the sigmoid, it squashes the inputs, but into the range [-1,1]. They both face a limitation: the gradients for small or large inputs become small, leading to the vanishing gradients phenomenon. Vanishing gradients occur, especially in first layers of deep architectures, when the gradients w.r.t. some parameters are extremely small, resulting in minor updates and slowing down the learning process. The Rectified Linear Unit (ReLU) ([19]) is computed by:

$$ReLU(x) = \max(0, x) \tag{2.8}$$

ReLU is similar to the linear activation function, except that it outputs zero for negative inputs. It has been widely used in deep networks due to its simplicity and improved performance. For positive inputs, the gradients are consistent, which helps mitigate the vanishing gradient problem. However, for negative inputs, the neuron consistently outputs zero and becomes inactive. When a significant fraction of the neurons remain inactive, the network's learning capacity diminishes. This is known as the "dying ReLU" phenomenon. The Gaussian Error Linear Unit (GELU) ([19]) is a non-linear activation function:

$$GELU(x) = x\Phi(x) = \frac{1}{2}x[1 + erf(x/\sqrt{2})] \tag{2.9}$$

GELU is a smooth activation function, that is able to model negative values and gradients. The activation function outputs and their derivatives are illustrated in Figure 2.2.



(a) Activation function outputs (b) Activation derivatives

Figure 2.2: *The activation function outputs (left) and their derivatives (right).*

2.1.5 Optimizers

Optimizers iteratively update the model's weights and biases to minimize the training loss. First-order optimization algorithms, use the first gradients of the loss function to their parameter update rules. Several optimization methods have been developed for training deep neural networks; the algorithms utilized in this study are analyzed below.

I) Adam Optimizer

Adam ([20]), short for Adaptive Momentum Estimation, is a first order optimization algorithm, suitable for training large networks, in terms of data and/or parameters. Adam combines the advantages of two popular optimization methods in deep learning: Momentum and RMSprop.
Momentum-based gradient descent computes an exponential moving average of the previous and current gradients, denoted as m_t, and uses this to update the parameters. Keeping track of past gradients accelerates convergence and helps the algorithm avoid local minima. The update rule is:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.10}$$

$$w_t = w_{t-1} - a m_t \tag{2.11}$$

$\mathrm{g}_t = \frac{\partial L}{\partial w_i}$ represents the gradient of the loss function w.r.t each parameter of the network, β_1 is the momentum factor, α is the learning rate, and w_t, w_{t-1} are the weights of the network at time steps t, t-1 respectively.
RMSprop is an adaptive learning rate optimization algorithm. It maintains an exponentially moving average of the squared gradients, denoted as u_t, to adjust

the learning rate for each parameter individually during training. This adaptive mechanism reduces oscillations and stabilizes the training process.

$$u_t = \beta_2 u_{t-1} + (1 - \beta_2) g_t^2 \tag{2.12}$$

$$w_t = w_{t-1} - \frac{\alpha}{(u_t + \epsilon)^{\frac{1}{2}}} g_t \tag{2.13}$$

g_t^2 is the squared gradient at timestep t, β_2 the decay rate, and ϵ a small constant to avoid division by zero.
The weight update rule for Adam is derived by combining Eq. 2.11 & 2.13. Additionally, since m_t and u_t are initialized as zeros, they are biased towards zero at the beginning of training, herein Adam employs the biased-corrected estimates as follows:

$$w_t = w_{t-1} - \frac{\alpha \hat{mt}}{\hat{u_t}^{\frac{1}{2}} + \epsilon} \tag{2.14}$$

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{2.15}$$

$$\hat{u_t} = \frac{u_t}{1 - \beta_2^t} \tag{2.16}$$

II) AdamW Optimizer

AdamW ([21]) is an extension of the Adam algorithm, that incorporates weight decay in the parameter update rule. Large weights make the network more sensitive towards changes to its inputs and increase the risk of overfitting. Weight decay encourages for smaller, more robust weights, which can enhance the model's generalization performance. Instead of adding a penalty term to the loss function, AdamW decouples the weight decay, so that it doesn't interfere with the adaptive learning rate dynamics. The update rule is:

$$w_t = w_{t-1} - \frac{\alpha \hat{mt}}{\hat{u_t}^{\frac{1}{2}} + \epsilon} - \lambda w_{t-1} \tag{2.17}$$

where λ is the weight decay rate, with default value to 0.004.

II) Adamax Optimizer

Adamax ([20]) is an extension of Adam, that adapts the learning rate using the infinity norm, instead of the average of the squared gradients used in Adam. This approach is effective when dealing with sparse gradient matrices, and parameters with large variations.

$$w_t = w_{t-1} - \alpha \frac{m_t}{(1-\beta_1^t)u_t} \tag{2.18}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t \tag{2.19}$$

$$u_t = max(\beta_2 u_{t-1}, |g_t|) \tag{2.20}$$

2.1.6 Learning Rate

The learning rate of the optimizer is a fundamental hyperparameter in deep learning, as it controls the step size at gradient descent. A high learning rate can cause overshoot of the minimum or even divergence, while a small one may result in slow convergence. The ideal one typically involves a balance between stability and speed, and depends on the specific training configuration and the architecture of the DNN. A common strategy involves starting with a relatively large learning rate, since the weights in this phase are far from the optimal, and gradually decrease it. This allows for rapid initial progress and finer updates as the model approaches a minimum.

2.1.7 Batch size

Batch size refers to the number of samples processed in a single forward and backward pass of the network. It influences both the convergence speed and the model's overall performance. Larger batch sizes reduce the training time per epoch. Usually when dealing with large databases the number of samples that can be processed in each iteration is constrained by the available memory resources. Although larger batch sizes provide more accurate gradient approximations, it has been observed that such training configurations might result in a degradation of the model's generalization performance ([22]). In contrast, small-batch training of DNNs yields solutions with better generalization capabilities. This improvement is attributed to the noise introduced by smaller batches in the gradient estimates, which acts as a form of implicit regularization and helps avoid overfitting. ([23]) highlights that mini-batch training -ranging from 2 to 32- achieves better test accuracy. The optimal batch size depends on factors such as the dataset's size and the complexity of the network.

2.1.8 Generalization capabilities

In the training of DNNs, the objective extends beyond minimizing the training loss; it also involves identifying minimizers that have accurate predictions on unseen inputs. Figure 2.3 illustrates various scenarios that can be encountered during training. Underfitting occurs when the model is too simple to capture the underlying patterns in the data, leading to poor performance on both the training and validation sets. In contrast, overfitting refers to a case in which the model memorizes the training data in detail, but fails to generalize on new inputs.

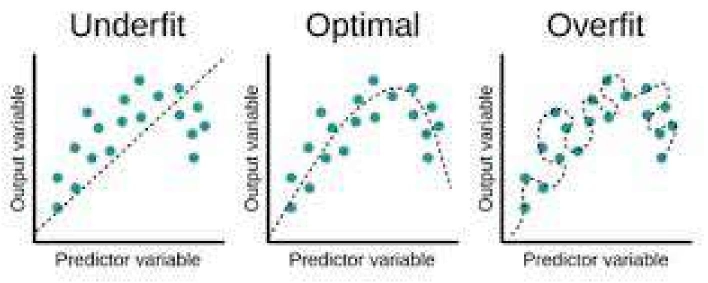


Figure 2.3: *Illustration of underfitting (left), optimal fit (center), and overfitting (right) scenarios in model training. Figure from (`https://evolucionapps.com/understanding-overfitting-and-underfitting-in-machine-learning/`)*

Overfitting occurs when the model is overparameterized relative to the complexity of the patterns it is expected to learn. In such cases, its capacity enables it to memorize noise or less meaningful details. When feasible, increasing the number of training patterns can help address this issue. In this work, cost constraints necessitate the use of small databases. Therefore, the primary focus is on identifying DNN architectures and training configurations that exhibit better generalization performance. Several regularization techniques, that can be imposed during training, have been proposed to prevent overfitting. These include:

1. **Early stopping:** ([24]) To assess generalization during training, the dataset is split into training and validation subsets, following the commonly adopted 80–20 % rule. In the early stages of training, training and validation losses typically decrease together. As training progresses, the validation loss may begin to increase, indicating the onset of overfitting. Early stopping, terminates the training process at this point, in order to preserve the parameters at their optimal values for generalization.

2. **Weight decay:** Weight decay is a widely used regularization technique that penalizes large weights. It suppresses irrelevant components and reduces the tendency to memorize noise from the training features ([25]). Weight decay can be expressed in various forms, including adding a penalty term to the loss function (L1, L2 regularization) or incorporating decay directly into the parameter update rule ([26], [21]).

2.1.9 Pruning

DNNs are frequently over-parameterized, with only a subset of their weights significantly influencing predictions. Pruning techniques aim to remove redundant parameters —such as individual weights, neurons in dense layers, or filters in convolutional layers— that have minimal impact on the network's overall performance. The primary objective is to reduce the network's size while maintaining its accuracy. Additionally since pruning suppresses noise during training, it can potentially improve the generalization performance. Pruning strategies are typically categorized as unstructured ([27]), where individual weights are removed, and structured ([28]), which involve removing entire architectural components such as neurons, or filters. The DNN architecture prior and after pruning is presented in Figure 2.4. While structured pruning yields greater benefits in terms of model compression and inference speed, it leads to higher degradation in accuracy due to the removal of entire computational blocks. The sparse matrices produced by unstructured pruning can be stored using formats such as Compressed Sparse Row (CSR) or Compressed Sparse Column (CSC), which retain only the non-zero values and their corresponding indices. These representations require only 2a+n+1 entries, where a denotes the number of non-zero elements and n the number of rows (in CSR) or columns (in CSC). A more detailed presentation of compression algorithms that can be applied on pruned networks is provided in ([29]). This diploma thesis focuses on unstructured pruning of dense layers; each connection is evaluated independently, and those that do not satisfy specific criteria are removed.

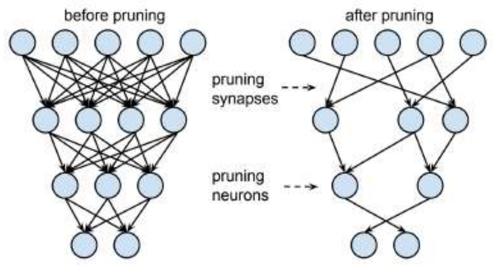
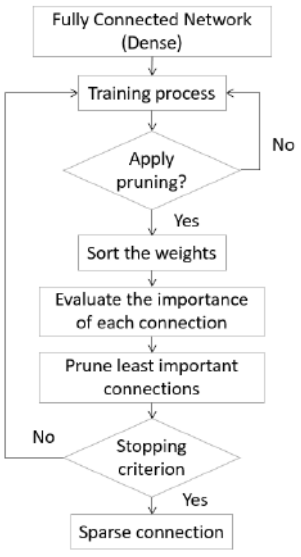
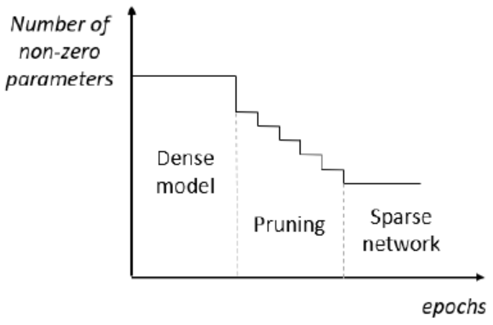


Figure 2.4: *DNN architecture before (left) and after (right) pruning. In unstructured pruning, individual weights are set to zero, creating sparse connections between the layers, while in structured pruning entire neurons are removed. Figure from* (`https://vitalitylearning.medium.com/neural-network-node-pruning-a-keras-implementation-on-mnist-e696f4276e2f`)

Several pruning criteria have been proposed in the literature; commonly used methods prune based on magnitude, or loss change. Magnitude-based pruning considers the weights, filters, etc, with larger values more important for the predictions, while those with absolute values below a threshold contribute less and can be trimmed. ([27]) utilizes a magnitude-based iterative pruning algorithm, applied gradually until the desired prune-ratio is achieved. ([28]) prunes filters based on the sum of the absolute values of their kernel weights. The method presented in ([30]) estimates the importance of each weight by computing a Taylor expansion for the change in the loss function resulting from the removal of the specific connection. A thorough overview of pruning algorithms is presented in ([31]). This work adopts a magnitude-based criterion, introduced iterative during training, as described in ([27]). Sparsity is increased gradually, giving the model the appropriate time to adapt to the fewer parameters. This can achieve better accuracy levels at higher prune-ratios, compared to one-shot pruning methods applied prior or after training ([31]). Training begins with a fully connected network. At specific optimizer steps, as specified by a user-defined schedule, pruning is applied: the weights are sorted according to their absolute values, and those below a threshold are pruned. Training proceeds with the remaining parameters, allowing the network to adapt to its sparse structure. This pruning cycle is repeated until the desired sparsity is achieved, or a user-defined stooping criterion of pruning is satisfied. In the developed code, a binary mask is assigned to each trainable layer of the network, with the same shape as the weight matrix of that layer. The mask contains the value of 0 at the indices of the pruned weights, and the value of 1 at those of active weights. During the forward pass, pruned weights do not contribute to the network's output. The binary masks are applied to the backpropagated gradients, ensuring that pruned weights receive no updates and remain inactive. These masks are updated only when pruning is applied and remain fixed throughout the remaining training process. The pruning algorithm, along with the evolution of the number of remaining parameters in the network, is illustrated in Figure 2.5.


Fully Connected Network (Dense)
Training process
Apply pruning?
No
Yes
Sort the weights
Evaluate the importance of each connection
Prune least important connections
Stopping criterion
No
Yes
Sparse connection

(a) Pruning algorithm



(b) Remaining parameters

Figure 2.5: *The pruning algorithm (left) and the network's parameters at different phases of training (right).*

2.2 Hermite-trained DNNs

2.2.1 Hermite interpolation

Hermite interpolation ([32]) is applied in cases where both the function values and their first derivatives are known at N$_s$ interpolation points. It computes a polynomial, that together with its first derivatives, satisfies the given function and derivative values at each node. This polynomial can be expressed as a linear combination of orthogonal polynomials, called Hermite basis polynomials. Two types of Hermite basis polynomials are defined, $H_j(x)$, where j=1,..,N$_s$, that contributes only to the value of the function for the j-th point, without being involved in satisfying the derivative value and $\overline{H}_j(x)$ that has the exact opposite role. As a result, $H_j(x_i)$ must return the value of 1 when evaluated at the i=j point, where i=1,...,N$_s$ and zero at any other point, while $\overline{H}_j^{'}(x_i)$ must equal 1 at the j-th point and zero otherwise.

$$\begin{aligned} H_j(x_i) = \delta_i^j \quad H_j'(x_i) = 0 \\ \overline{H}_j(x_i) = 0 \quad \overline{H}_j'(x_i) = \delta_i^j \end{aligned} \tag{2.21}$$

The Hermite basis polynomials are defined using the Lagrange basis polynomials (L_j):

$$H_j(x) = (1 - 2L_j^{'}(x_j)(x - x_j))(L_j(x))^2 \tag{2.22}$$

$$\overline{H}_j(x) = (x - x_j)(L_j(x))^2 \tag{2.23}$$

The Hermite interpolation equation is expressed in Eq. 2.24 as a linear combination of the basis polynomials:

$$g(x) = \sum_{j=0}^{N} y_j H_j(x) + \sum_{j=0}^{N} y_j' \overline{H}_j(x) \tag{2.24}$$

2.2.2 Hermite DNN configuration

Eq. 2.24 can be written in matrix form as:

$$
\begin{bmatrix} g(x_{i=1}) \\ g(x_{i=2}) \\ \vdots \\ g(x_{i=N_s}) \end{bmatrix}
=
\begin{bmatrix}
H_{j=1}(x_{i=1}) & H_{j=2}(x_{i=1}) & \dots & H_{j=N_s}(x_{i=1}) \\
H_{j=1}(x_{i=2}) & H_{j=2}(x_{i=2}) & \dots & H_{j=N_s}(x_{i=2}) \\
\vdots & \vdots & \vdots & \vdots \\
H_{j=1}(x_{i=N_s}) & H_{j=2}(x_{i=N_s}) & \dots & H_{j=N_s}(x_{i=N_s})
\end{bmatrix}
\begin{bmatrix} y_{j=1} \\ y_{j=2} \\ \vdots \\ y_{j=N_s} \end{bmatrix}
$$

$$
+
\begin{bmatrix}
\overline{H_{j=1}}(x_{i=1}) & \overline{H_{j=2}}(x_{i=2}) & \dots & \overline{H_{j=N_s}}(x_{i=N_s}) \\
\overline{H_{j=1}}(x_{i=1}) & \overline{H_{j=2}}(x_{i=2}) & \dots & \overline{H_{j=N_s}}(x_{i=N_s}) \\
\vdots & \vdots & \vdots & \vdots \\
\overline{H_{j=1}}(x_{i=1}) & \overline{H_{j=2}}(x_{i=2}) & \dots & \overline{H_{j=N_s}}(x_{i=N_s})
\end{bmatrix}
\begin{bmatrix} y'_{j=1} \\ y'_{j=2} \\ \vdots \\ y'_{j=N_s} \end{bmatrix}
\tag{2.25}
$$

The first term of Eq. 2.25 consists of the product of an $[N_s \times N_s]$ matrix, whose entries are the values of the H_j basis polynomials evaluated at each of the N$_s$ interpolation points, and a vector containing the N$_s$ function values. Similarly, the second term is the product of an $[N_s \times N_s]$ matrix, formed by the values of the $\overline{H_j}$ polynomials at the same points, and a vector containing the N$_s$ derivative values. Based on the properties of the basis polynomials of Eq. 2.21 the H_j, $\overline{H_j}$ matrices equal the identity and zero matrix respectively. The Hermite interpolation method was formulated for a function with one design variable. The generalization to higher dimensions can be complicated. In this diploma thesis an alternative generalization method is implemented, where instead of the explicit computations, the interpolation function (g) is approximated using DNNs ([13]). The network consists of two branches, named B_{func}, B_{grad}. Each branch models the corresponding term of Eq. 2.25 and determines the contribution of the function and derivative values to the total interpolation function. The outputs from each branch are summed up to form the DNN's output with the prediction of g. The DNN's branches architecture is shown in Figure 2.6 and discussed next.
The input data ($\hat{X}$) correspond to the values of the design variables. Both branches share the same input data, hence the architecture of the input layer is identical in each one. Each branch contains a series of hidden layers, whose architecture is user-defined and should not necessarily be the same. In all the cases, the hidden layers consist of fully connected layers, however different types of layers can also be incorporated. Following the hidden layers, an additional layer is introduced, to be referred as basis layer, with number of neurons equal to the number of training patterns, N_s. When the entire database is processed the output of the basis layer forms an $[N_s \times N_s]$ tensor, that serves as surrogate of the basis polynomials.
To compute the first term of 2.25, the H_j matrix is multiplied with the vector of target function values. To achieve this within the B_{func} branch, a fully connected layer is introduced, consisting of a single neuron. This layer is non-trainable, and its weight matrix of size $[N_s \times 1]$ is initialized with the normalized target function values. As a result, the output of this branch ($[N_s \times 1]$) will hopefully act as the

first term of the Hermite interpolation function.
The B_{grad} branch has similar structure as the B_{func} branch, except for the output layer, which consists of N_b neurons—equal to the number of design variables and thus to the dimensionality of the input layer. The output layer's weights are non-trainable and defined by the $[N_s \times N_b]$ matrix with the target derivatives of the normalized outputs with respect to the normalized inputs. Thus, the contribution of each partial derivative will be taken into account, by adding N_b terms to the interpolation function, similar to the second term of Eq. 2.25.
The output layer of the network contains a single neuron, as this diploma thesis considers only one target. This neuron uses a linear activation function. The outputs from the two branches are combined into a single tensor of shape $[N_s \times (N_{\text{b}} + 1)]$. This combined tensor then passes through the output layer, where the contributions of each term are added to produce the predictions of the Hermite interpolation function.

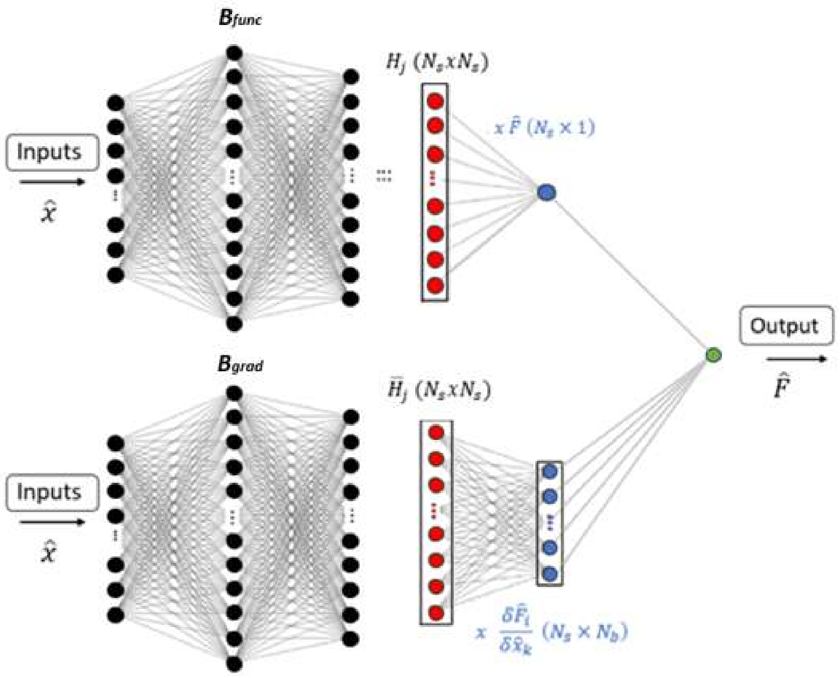


Figure 2.6: *Hermite DNN architecture*

Input (x) and output (y) data of the networks are normalized with the minimum and maximum values encountered in the samples. The normalization process scales the entire dataset to the same range and ensures that each feature contributes equally during learning.

$$x_k = \frac{X_k - X_{k,min}}{X_{k,max} - X_{k,min}} \tag{2.26}$$

$$y = \frac{Y - Y_{\text{min}}}{Y_{\text{max}} - Y_{\text{min}}} \tag{2.27}$$

$X_{k,min}$, $X_{k,max}$ represent the minimum and maximum bounds of the k-th design variable, where k=1,..,N$_b$, X_k is the dimensional value and x_k the normalized one. The derivatives of the normalized outputs w.r.t. the normalized inputs are rescaled using Eq. 2.28:

$$\frac{dy}{dx_k} = \frac{d\left(\frac{Y-Y_{\min}}{Y_{\max}-Y_{\min}}\right)}{d\left(\frac{X_k-X_{k,\min}}{X_{k,\max}-X_{k,\min}}\right)}) = \frac{dY}{dX_k} \cdot \frac{X_{k,\max}-X_{k,\min}}{Y_{\max}-Y_{\min}} \tag{2.28}$$

While in standard training of DNNs, the loss function to be minimized is the error between the DNN's predictions ($\hat{y}$) and the normalized target values, in Hermite training of DNNs the error between the predicted derivatives ($D_x\hat{y}$) and the target ones ($D_x y$) must be included in the loss function, hence N_b terms will be added that measure the error of each partial derivative. Herein, the network will be trained to match the function's values and after being differentiated, to predict the target derivatives. The $Loss_{hermite}$ consists of the (N_b+1) terms of Eq. 2.29, where l$_k$ represents the loss function used in each one.

$$\mathcal{L}_{\text{Hermite}} = w_1\,\ell_0(y,\hat{y}) + \sum_{k=1}^{N_b} w_k\,\ell_k\left(D_{x_k}y, D_{x_k}\hat{y}\right) \tag{2.29}$$

Since no restrictions are applied, the properties of the basis polynomials in Eq. 2.21 are not necessarily satisfied and the contribution of each branch will be determined during training. Although y,$\hat{y}$ range in [0,1], the normalized derivatives ($D_x y$, $D_x\hat{y}$) does not necessarily have the same bounds, since their values are determined from the values of SDs and DNN's input and output. Different scales of function and derivative terms can be ineffective for the learning process of the network. Learning can be balanced by adjusting the weights of each term on $Loss_{hermite}$ accordingly.

Chapter 3

The Proposed DNN-driven Optimization

3.1 The RANS equations

The Reynolds Averaged Navier Stokes (RANS) equations for single phase flows are expressed:

$$R_n^{\mathrm{MF}} = \underbrace{\frac{\partial f_{nk}^{\mathrm{inv}}}{\partial x_k}}_{\text{inviscid}} - \underbrace{\frac{\partial f_{nk}^{\mathrm{vis}}}{\partial x_k}}_{\text{viscous}} = 0 \tag{3.1}$$

The inviscid and viscous fluxes are given by:

$$f_{nk}^{\mathrm{inv}} = \begin{bmatrix} \rho u_k \\ \rho u_1 u_k + p\delta_{1k} \\ \rho u_2 u_k + p\delta_{2k} \\ \rho u_3 u_k + p\delta_{3k} \\ \rho u_k h_t \end{bmatrix} \qquad f_{nk}^{\mathrm{vis}} = \begin{bmatrix} 0 \\ \tau_{1k} \\ \tau_{2k} \\ \tau_{3k} \\ u_l \tau_{lk} + q_k \end{bmatrix} \tag{3.2}$$

where ρ is fluid's density and p the static pressure. The viscous stresses are computed as:

$$\tau_{nk} = (\mu + \mu_t)\left(\frac{\partial u_k}{\partial x_m} + \frac{\partial u_m}{\partial x_k}\right) - \frac{2}{3}\left(\frac{\partial u_l}{\partial x_l}\right)\delta_{km} \tag{3.3}$$

In this diploma thesis, a cavitating flow is also considered, for which the homogeneous two-phase approach is employed. Both phases (liquid and vapor) share the same velocity and pressure fields, and are distinguished from their volume fractions. A volume fraction is defined as the ratio of each phase's volume in a computational cell, to the total volume of that cell. The mixture's density (ρ), and dynamic viscosity (μ) are computed as a weighted-sum of each phase's properties:

$$\rho = f(a_l) = \rho_\ell a_l + \rho_v \alpha_v \tag{3.4}$$

$$\mu = g(a_l) = \mu_\ell a_l + \mu_v \alpha_v \tag{3.5}$$

ρ_l, μ_l stand for the density and dynamic viscosity of the liquid phase, and ρ_v, μ_v for the corresponding ones of the vapor phase. Since the two phases share the same pressure and velocity fields, the RANS are solved for the mixture:

$$R_n^{\mathrm{MF}} = \underbrace{\frac{\partial f_{nk}^{\mathrm{inv}}}{\partial x_k}}_{\text{inviscid}} - \underbrace{\frac{\partial f_{nk}^{\mathrm{vis}}}{\partial x_k}}_{\text{viscous}} - \underbrace{S_n^{\mathrm{cav,MF}}}_{\text{cavitation}} = 0 \tag{3.6}$$

The viscous, inviscid fluxes, and the source terms are given by:

$$\mathbf{f}_{nk}^{\mathrm{inv}} = \begin{bmatrix} u_k \\ \rho u_1 u_k + p\delta_{1k} \\ \rho u_2 u_k + p\delta_{2k} \\ \rho u_3 u_k + p\delta_{3k} \end{bmatrix}, \quad \mathbf{f}_{nk}^{\mathrm{vis}} = \begin{bmatrix} 0 \\ \tau_{1k} \\ \tau_{2k} \\ \tau_{3k} \end{bmatrix}, \quad \mathbf{S}_n^{\mathrm{cav,MF}} = \begin{bmatrix} \left(\frac{1}{\rho_\ell} - \frac{1}{\rho_v}\right)(\dot{m}_{\mathrm{cond}} - \dot{m}_{\mathrm{evap}}) \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.7}$$

The liquid volume fraction is computed by the mass conservation of the liquid phase:

$$R^{\mathrm{VF}} = \frac{\partial a_l u_j}{\partial x_j} - S^{\mathrm{cav,VF}} = 0 \tag{3.8}$$

where

$$S^{\mathrm{cav,VF}} = \frac{1}{\rho_l}(\dot{m}_{cond} - \dot{m}_{evap}) \tag{3.9}$$

$$a_l + a_v = 1 \tag{3.10}$$

The condensation and evaporation mass rates are computed using empirical formulas, specifically the cavitation model proposed by Kunz et al. in ([33]).

Cavitation model	$\dot{m}_{evap}$	$\dot{m}_{cond}$
Kunz	$C_e \rho_v a_l \frac{\max(p_{vap}-p,0)}{0.5\rho_\ell u_{ref}^2 t_{ref}}$	$C_c \rho_v \alpha_v \frac{a_l^2}{t_{ref}}$

Table 3.1: *Evaporation and Condensation mass rates.*

C_e, C_c are empirical coefficients, p_{vap} is the constant vapor pressure, u_{ref}, t_{ref} are the reference velocity and the time scale. The viscous stresses are:

$$\tau_{nk} = (\mu + \mu_t)\left(\frac{\partial u_k}{\partial x_m} + \frac{\partial u_m}{\partial x_k}\right) \tag{3.11}$$

The CFD tool used in this diploma thesis, is the GPU-accelerated solver PUMA developed by PCOpt/NTUA ([34]). PUMA solves compressible and incompressible flows using the vertex-centered finite volume method on unstructured and hybrid meshes. Additionally, PUMA enables surface parameterization via Volumetric Non-Uniform Rational B-Splines (NURBS) ([35]). Several mesh-morphing techniques are available, that can be used as standard morphers or during an optimization process. A variety of turbulence and transition models are implemented in PUMA. In this diploma thesis, the Spalart–Allmaras turbulence model ([36]) is utilized, and when needed, a version of the $\gamma - \tilde{Re}_\theta$ transition model, namely the version proposed by Piotrowski&Zingg (SA-sLM2015) ([37], [38]).

3.2 The Spalart-Allmaras Turbulence Model

The Spalart-Allamaras turbulence model ([36]) solves an additional partial differential equation (PDE), for the turbulence field variable $\tilde{\nu}$. The PDE is presented in Eq. 3.12. The turbulent viscosity (μ_t) is computed using Eq. 3.13. In case of cavitating flow simulations, the equation is solved for the liquid-vapor mixture.

$$R^{\tilde{\nu}} = \frac{\partial(\rho\tilde{\nu}u_k)}{\partial x_k} - \frac{\rho}{\sigma}\left[\frac{\partial}{\partial x_k}\left((\nu+\tilde{\nu})\frac{\partial\tilde{\nu}}{\partial x_k}\right) + C_{b2}\frac{\partial\tilde{\nu}}{\partial x_k}\frac{\partial\tilde{\nu}}{\partial x_k}\right] - P_{\tilde{\nu}} + D_{\tilde{\nu}} = 0 \tag{3.12}$$

$$\mu_t = \tilde{\nu}\rho f_{v1} \tag{3.13}$$

where the production (P) and dissipation (D) terms are:

$$P_{\tilde{\nu}} = \rho c_{b1}(1-f_{t2})\,\tilde{S}\,\tilde{\nu}, \quad D_{\tilde{\nu}} = \rho\left(c_{w1}f_w - \frac{c_{b1}}{\kappa^2}f_{t2}\right)\left(\frac{\tilde{\nu}}{\Delta}\right)^2 \tag{3.14}$$

More information about the constants of the Spalart-Allmaras turbulence model are provided in ([36]).

3.3 The γ–$\tilde{Re}_\theta$ Transition Model

The γ-$\tilde{Re}_\theta$ ([37]) transition model solves two additional PDE, for the transition intermittency (γ), and the transition momentum-thickness Reynolds number ($\tilde{Re}_\theta$):

$$\begin{aligned} R^{\gamma} &= \frac{\partial(\rho u_k\gamma)}{\partial x_k} - \frac{\partial}{\partial x_k}\left[\left(\mu+\frac{\mu_t}{\sigma_f}\right)\frac{\partial\gamma}{\partial x_k}\right] - P_\gamma + D_\gamma = 0 \\ R^{\tilde{Re}_{\theta_t}} &= \frac{\partial(\rho u_k\,\tilde{Re}_{\theta_t})}{\partial x_k} - \frac{\partial}{\partial x_k}\left[\sigma_{\theta t}\,(\mu_t+\mu)\,\frac{\partial\tilde{Re}_{\theta_t}}{\partial x_k}\right] - P_{\theta_t} - D_{SCF} = 0 \end{aligned} \tag{3.15}$$

Piotrowski & Zingg proposed smoother approximations for the source terms, forming the version SA-sLM2015 ([38]). The transition model is coupled with the Spalart-Allmaras turbulence model, by affecting its source terms as:

$$P_{\tilde{\nu}} = \gamma \rho c_{b1} \tilde{S} \tilde{\nu} \tag{3.16}$$

$$D_{\tilde{\nu}} = \rho c_{w1} f_w \left(\frac{\tilde{\nu}}{\Delta} \right)^2 \tag{3.17}$$

More information for the source temrs and the constants of each model are provided in ([37], [38]).

3.4 The Adjoint-Driven Optimization Process

In each optimization cycle in a gradient-based process, the primal, here the RANS equations, is solved. Then, the computation of the derivatives of the objective function w.r.t. the design variables, also referred to as SDs, are required. The adjoint method is used to compute the SDs, either in its continuous form or with consistent discretization schemes ([39], [40]).
The continuous adjoint approach constructs an augmented objective function, which combines the objective sought to be minimized and the integrals, over the flow domain, of the products of the primal residuals and the adjoint variables. Once the primal has converged, and their residuals are close to zero, the augmented equals the objective. The augmented function is then differentiated w.r.t. the design variables. From the differentiation integrals are formed that include derivatives of the flow variables w.r.t. the design variables. To avoid the computation of that terms, the multipliers of these derivatives are set to zero. This forms the field adjoint equations, that are discretized and solved, at a computational cost approximately equivalent to that of solving the primal equations.

3.5 DNNs as surrogates of the CFD-solver

Two DNN-driven optimization strategies are compared in this diploma thesis. The first one is presented in Figure 3.1 and discussed next.

1. **Parameterization:** To construct the database used for training of the DNNs, a set of potential geometries must be generated. This is achieved by sampling the design variable space, to create various combinations of the control points positions. Each combination corresponds to a different geometry. Sampling is implemented with Latin Hypercube Sampling (LHS) ([41]). LHS generates N samples in $[0, 1)^d$, where d is the number of design variables. For each one it places a point in every $[j/N, (j + 1)/N)$ interval at random position, for j=0,..N-1. LHS ensures the samples are representative of the real variability, even at small datasets.

2. **Mesh-adaptation and CFD-evaluation:** A grid displacement method is employed to adapt the mesh to the modified geometry. The geometry is then evaluated using the CFD-solver, and the results are added the database. For each sample, besides the primal evaluation, the SDs are also included in the database. Given the computational cost of these evaluations, the number of samples is kept as small as possible.

3. **DNN training:** DNNs are trained on the constructed database. Inputs to the models are the design variables, and output the objective function prediction.

4. **DNN-driven gradient descent:** The optimization process is initiated from the baseline geometry. During this phase, the expensive primal and adjoint evaluations are replaced by the cost-efficient DNN surrogates.

5. **CFD re-evaluation of the optimized solution:** The solution obtained from the DNN-driven optimization is evaluated on the CFD tool. If the optimized solution satisfies the desired accuracy, the optimization process can be terminated. Otherwise, the re-evaluated geometry is added to the database and steps 2-3 are repeated. The new gradient descent begins from the previously optimized solution.

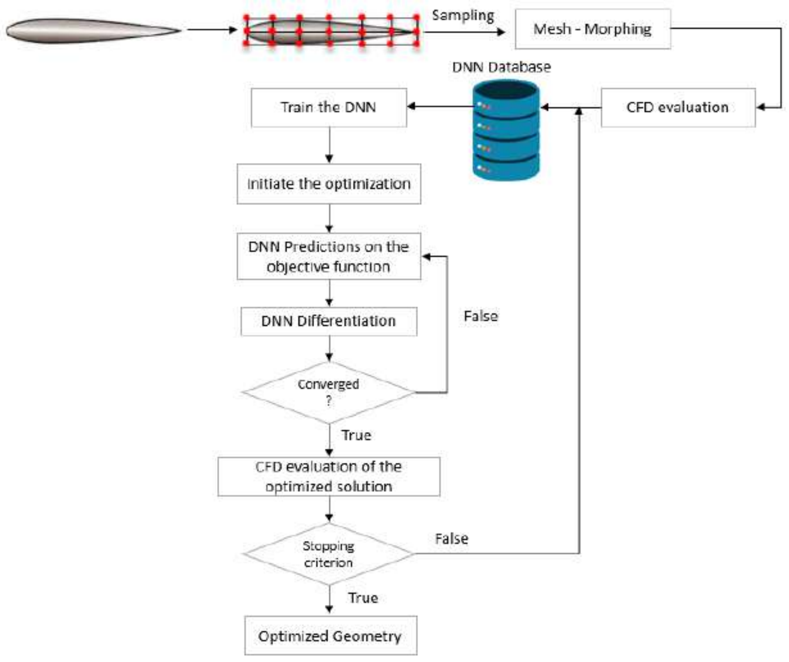


Figure 3.1: *Representation of the first DNN-driven optimization process.*

The second one employs a hybrid adjoint and DNN-driven optimization approach, as shown in Figure 3.2. During the early optimization cycles, the flow fields and SDs are computed using the CFD-solver. Each intermediate geometry is stored, to construct the DNN database. After a few initial cycles, the adjoint-driven optimization is stopped. The optimization process resumes, starting from the last candidate solution obtained by the adjoint-driven phase. In this second phase, the optimization is driven exclusively by DNNs. The DNN optimized geometry is re-evaluated on the CFD tool, to verify its accuracy. Based on this evaluation, the designer may choose to continue the optimization. If so, the new geometry is added to the database, the networks are retrained, and the DNN-driven optimization is repeated using the updated models.

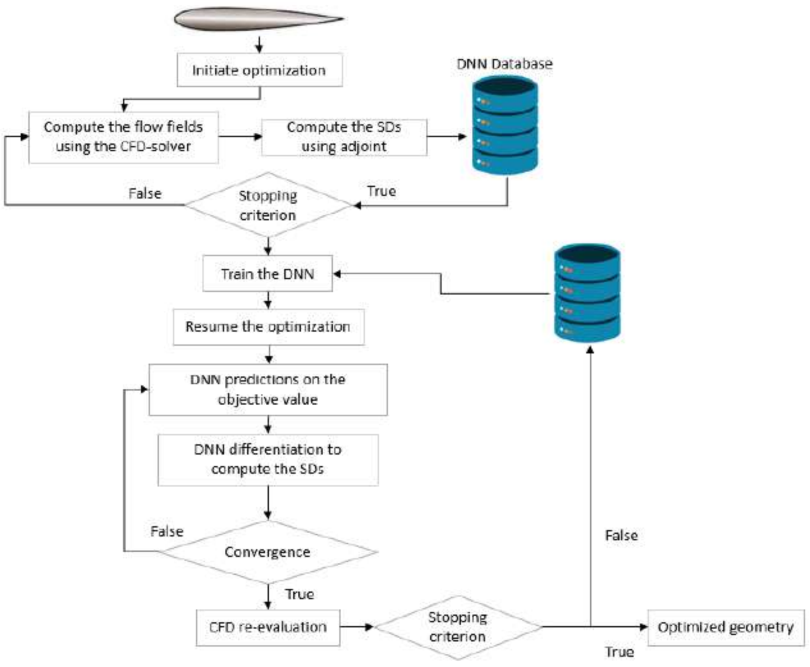


Figure 3.2: *Demonstration of the second DNN-driven optimization.*

Each DNN configuration used in this diploma thesis, is optimized with the in-house evolutionary software, EASY ([42]). EASY is a general purpose, high-fidelity software develop by PCOpt/NTUA. It is based on generalized evolution algorithms and can be used for single- and multi-objective optimizations. Additionally a coupling possibility with low-cost on-lime metamodels is available, to reduce the EA-based optimization turnaround time. The DNN optimization using EASY was introduced in [40]. The enhancement of this setup is also investigated in this diploma thesis, as demonstrated in Chapter 4.

3.6 The L-BFGS Algorithm

In optimization second-order methods utilize the Hessian matrix, which consists of the second-order partial derivatives of the objective function. The Hessian contains information about the curvature of the objective, and enables fast convergence near the optimum. However, constructing it is computationally expensive. Quasi-Newton methods approximate the inverse Hessian matrix, preserving the fast convergence advantages of second-order methods while avoiding the computational cost of computing the Hessian directly. They have the following update rule:

$$x^{k+1} \leftarrow x^k - a_k H_k \nabla f(x) \tag{3.18}$$

where H_k is the approximation of the inverse Hessian matrix at time-step k. Specifically, the BFGS algorithm uses the following formula for updating the H matrix:

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T \tag{3.19}$$

where

$$\begin{aligned} s_k &= x_{k+1} - x_k \\ y_k &= \nabla f(x_{k+1}) - \nabla f(x_k) \\ \\ \rho_k &= 1/y_k^T s_k \\ \\ V_k &= 1 - \rho_k y_k s_k^T \end{aligned}$$

The L-BFGS method ([43]) is used in the shape optimization studies of this work. It is a variant of the BFGS algorithm, that rather than storing the full approximation of the inverse Hessian matrix, retains only the most recent m values of s,y, since these have a greater impact on the approximation. The H_k is constructed by updating a symmetric, positive definite H_0 matrix $\hat{m}$+1 times using the pairs $[s_i, y_i]_{i=k-\hat{m}}^k$, where $\hat{m}$=min(k,m-1):

$$\begin{aligned} H_{k+1} = {} & (V_k^T \dots V_{k-\hat{m}}^T) H_0 (V_{k-\hat{m}} \dots V_k) \\ & + \rho_{k-\hat{m}} (V_k^T \dots V_{k-\hat{m}+1}^T) s_{k-\hat{m}} s_{k-\hat{m}}^T (V_{k-\hat{m}+1} \dots V_k) \\ & + \rho_{k-\hat{m}+1} (V_k^T \dots V_{k-\hat{m}+2}^T) s_{k-\hat{m}+1} s_{k-\hat{m}+1}^T (V_{k-\hat{m}+2} \dots V_k) \\ & \vdots \\ & + \rho_k s_k s_k^T \end{aligned}$$

Chapter 4

Single-Phase Turbulent Flow around a Turbine Blade-Airfoil

4.1 Introduction

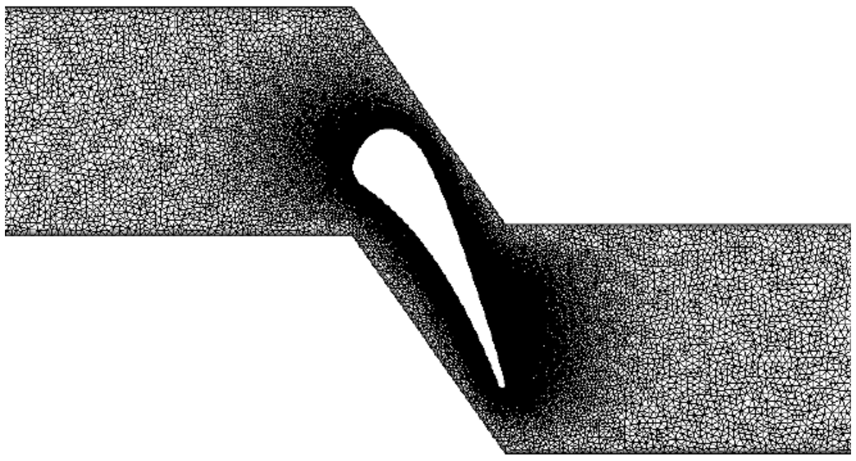
In this Chapter, the aerodynamic ShpO of the C3X turbine blade-airfoil is performed. Aim of the optimization is to minimize the massed-averaged p_t losses of the cascade (Δp_t), while preserving the exit flow angle (a_{exit}) close to its original value. The employed Hermite-DNNs are trained to predict the objective values and, after they are differentiated, their SDs. Once trained, these networks are utilized to drive the gradient-based ShpO of the blade-airfoil, by substituting both the primal and adjoint computations. In the first part of this Chapter, the entire ShpO process is implemented using networks trained on a database generated by the near-random LHS. Various DNN configurations are compared, regarding their generalization capabilities, and the memory requirements for storing them. In the second part an alternative approach is implemented. The ShpO during the initial cycles is driven by adjoint; after a specific number of cycles the CFD evaluations are substituted by the DNNs predictions.

4.2 Flow conditions and parameterization

The C3X is a cooled turbine blade, introduced in [44], and has been widely studied in heat transfer and optimization cases. The flow inlet and outlet conditions are illustrated in Table 4.1. An unstructured computational mesh of approximately 95K nodes is generated. The mesh is shown in Figure 4.1.

Flow Conditions	
Inlet total temperature (K)	808
Inlet total pressure (bar)	2.44
Inlet flow angle (°)	0
Outlet static pressure (bar)	1.43
Working fluid	Air

Table 4.1: *The turbine blade-airfoil case: Flow Conditions.*



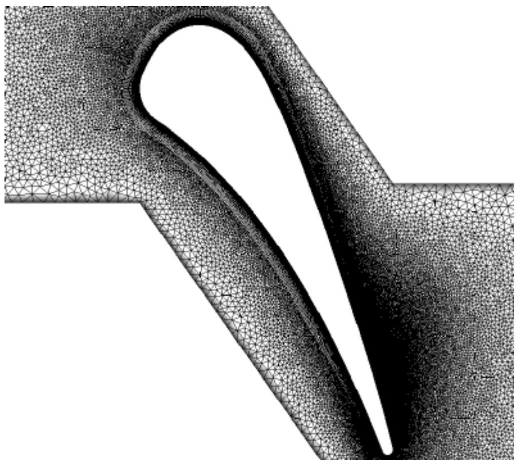


Figure 4.1: *The turbine blade-airfoil case: Computational mesh of the whole domain (top) and the region near the solid boundaries (bottom).*

The airfoil is parameterized using a 6 x 3 NURBS lattice, shown in Figure 4.2, that controls the airfoil's shape. The lattice is equidistant in each direction. The black points remain fixed during the optimization and the red ones are free to move in the cord-wise and pitch-wise direction with a maximum displacement of $0.15d$ around their initial values, where d represents the distance of adjacent control points in each direction. This results to 32 design variables ($\vec{b} \in \mathsf{R}^{32}$). During the optimization process, a grid displacement method is used to adapted the mesh to the design variables displacements. In all cases, the Inverse Distance Weighting (IDW) method is used ([45]). The displacement of each mesh node is the average of the displacement of each CP weighted by the inverse of its distance from the specific node. Nodes near the parameterized geometry are displaced more, while farfield points are not influenced significantly.

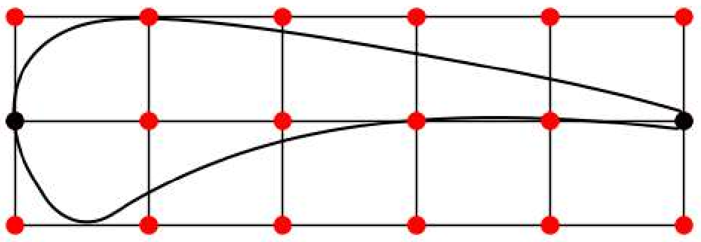


Figure 4.2: *The turbine blade-airfoil case: Blade-airfoil parameterization.*

4.3 DNN Configuration and Training

Two separate DNNs are build for Δp_t and a_{exit}, in order to ensure high accuracy on the SDs predictions. The database to be used for training of the networks is created by modifying the geometry of the baseline airfoil. The design variable space is sampled to create 29 different combinations of the Control Points (CPs) position. Sampling is implemented with LHS and for each sample the primal and adjoint problem is solved, thus Δp_t, a_{exit} and their SDs are computed. These quantities, along with those of the baseline geometry, compose the DNN database (DB_{LHS}). Development, training and differentiation of DNNs is carried out in the Tensorflow framework using Python ([46]). Input to each branch of the DNNs is the [N×32] tensor containing the coordinates of the DB_{LHS} samples, where N is the number of samples the DNN processes at the training or validation step. The outputs of each branch are concatenated to form the output [N×1] tensor, containing the predictions on the target values. The DNNs are then differentiated to produce an [N×32] tensor with the predicted derivatives. To asses generalization during training, the database is split in training and validation patterns, using the commonly adopted 80%-20% rule. As a result, the networks are trained on 24 samples, with a fixed set of 6 samples used for validation.

In order improve the accuracy of the DNNs predictions their optimal configuration should be identified. Herein, the DNN architecture and hyper-parameters are optimized with the in-house evolutionary algorithm software, EASY. For each network, two optimizations of their configuration are carried out, and these will be illustrated below.

4.3.1 First DNN configuration optimization

The optimized hyperparameters in this case include the number of hidden layers, the number of neurons per layer and the selection of activation functions. This setup was originally introduced in ([40]). The number of hidden layers in each branch range from 3 to 10. The number of neurons are expressed as a power of two and can be selected between 2^5 and 2^{12}. All hidden layers share the same activation function, while different activation functions may be assigned to the final layers of each branch. The remaining DNN's hyperparameters are fixed, to user-defined choices. More specifically, the optimizer is Adam with its default learning rate of 0.001. The batch size equals the number of training samples, and the loss function used during training is MSE. The objective, to be minimized by EASY, is the $\mathcal{L}_{\text{Hermite}}$ measured on the entire database (30 samples), after the DNN (to be referred as DNN$_A$) has been trained for the first 200 epochs. Training of each candidate network does not reach convergence, as this would significantly increase the cost of the EA-based process. Instead, the number of epochs is calibrated, to ensure that the resulting predictions are both reliable and representative of the network's performance. The optimized architectures are listed in Tables 4.2 & 4.3.

DNN Δp_t				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	8	128-2048-512-1024-64-256-24-1	Gelu	Tanh
B_{grad}	6	2048-4096-1024-64-24-32	Gelu	Sigmoid

Table 4.2: *The turbine blade-airfoil case: Optimized architecture of Δp_t DNN_A.*

DNN a_{exit}				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	7	32-4096-64-1024-2048-24-1	Tanh	Tanh
B_{grad}	5	256-1024-64-24-32	Tanh	Sigmoid

Table 4.3: *The turbine blade-airfoil case: Optimized architecture of a_{exit} DNN_A.*

Once the optimal architectures are defined, the DNNs are trained until convergence is reached. The loss curves for the Δp_t and a_{exit} models are presented in Figure 4.3, with the function and derivative components of the loss shown separately. The SDs loss corresponds to the sum of the individual loss terms associated with each design variable. Accurately predicting the derivatives, in addition to the target outputs, is a more complex task; thus, a higher loss is generally expected for the SDs terms.

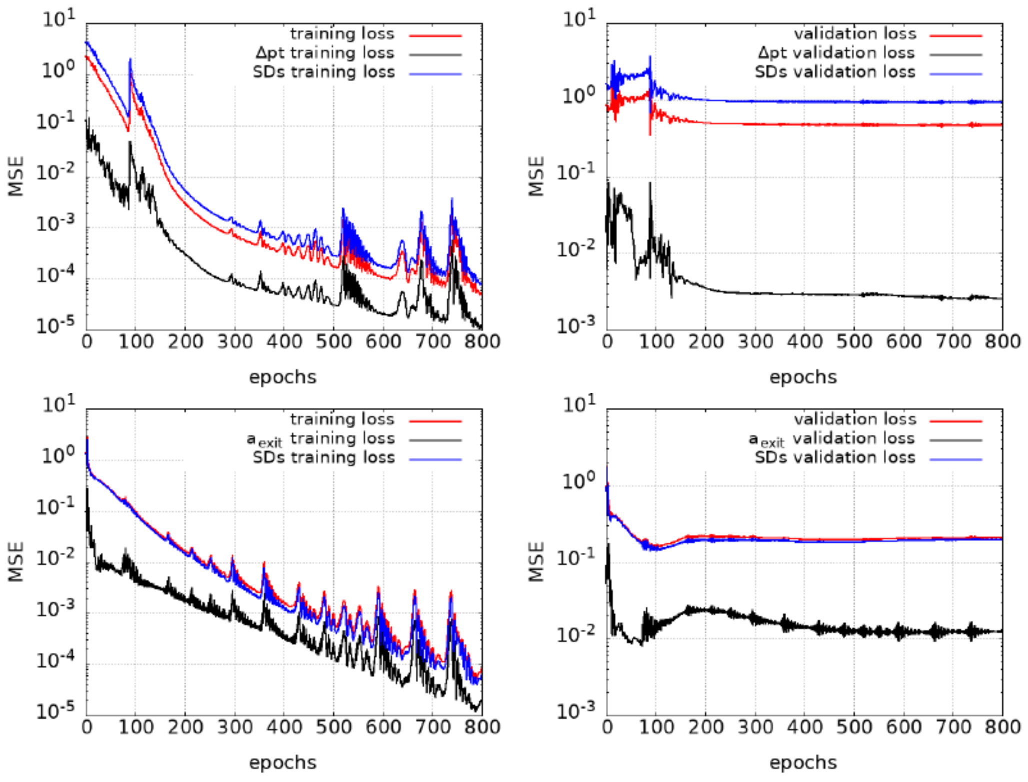


Figure 4.3: *The turbine blade-airfoil case: Training loss convergence (left) and validation loss (right) of Δp_t DNN_A (top) and a_{exit} DNN_A (bottom). Total loss is the weighted sum of the function loss and gradient loss.*

4.3.2 Second DNN configuration optimization

The design variable space of the DNN configuration optimization is expanded to explore a broader range of candidate DNN configurations. Besides the network's architecture, this optimization regards also its training configuration. Primary goal is to identify configurations than generalize better than DNN_A. In addition to the previously optimized hyperparameters, the optimization of the DNN configuration includes the selection of the loss function, the batch size, the training pattern shuffling in batch-processing scenarios, and the choice of optimizer along with its initial learning rate. The loss function selection is between MSE and MAE. The batch size varies from 8 to 24 patterns. The optimizer's choice is among Adam, AdamW, Adamax and Adadelta, with an initial learning rate ranging from 10^{-2} to 10^{-4}. The bounds of the rest design variables of the DNN configuration optimization remain the same as DNN_A. The objective, to be minimized, is the $\mathcal{L}_{\text{Hermite}}$ measured on the entire database (30 samples) after the DNNs (to be referred as DNN_B) are trained for 240 epochs. Since MAE and MSE yield outputs on different scales, at the evaluation step of the models the MSE is used. The optimized configurations are summarized in Tables 4.4 & 4.5

DNN Δp_t				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	12	64-2048-512-4096-1024-32-32-1024-4096-32-24-1	Gelu	Gelu
B_{grad}	7	4096-4096-64-1024-32-24-32	Gelu	Tanh
	Optimizer	**Loss function**	**Batch Size**	**Shuffle (epochs)**
	Adam (lr=0.0007)	MAE	16	15

Table 4.4: *The turbine blade-airfoil case: Optimized architecture of Δp_t DNN_B.*

DNN a_{exit}				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	11	256-1024-1024-64-1024-32-2048-4096-4096-24-1	Gelu	Gelu
B_{grad}	6	512-1024-32-128-24-32	Gelu	Gelu
	Optimizer	**Loss function**	**Batch Size**	**Shuffle (epochs)**
	AdamW (lr=0.002)	MAE	16	20

Table 4.5: *The turbine blade-airfoil case: Optimized architecture of a_{exit} DNN_B.*

Figure 4.4 presents the training and validation loss curves of Δp_t and a_{exit} DNN_B. The comparison of their performance with the corresponding networks of the previous Section is based on the quality of their predictions on the whole database. Regarding the Δp_t DNNs, it can be observed in Figure 4.5 that DNN_A achieves higher accuracy on validation Δp_t predictions. However, in Figure 4.6 DNN_B demonstrates an improvement in SDs of the validation blade; the value of the higher-magnitude derivatives are closer to the reference, and the sign in specific design variables is corrected. The predictions on a_{exit} SDs are depicted in Figure 4.7. DNN_B yields a closer match to the validation SDs, which is desirable since the constraint that will be imposed on a_{exit} during the ShpO is strict.

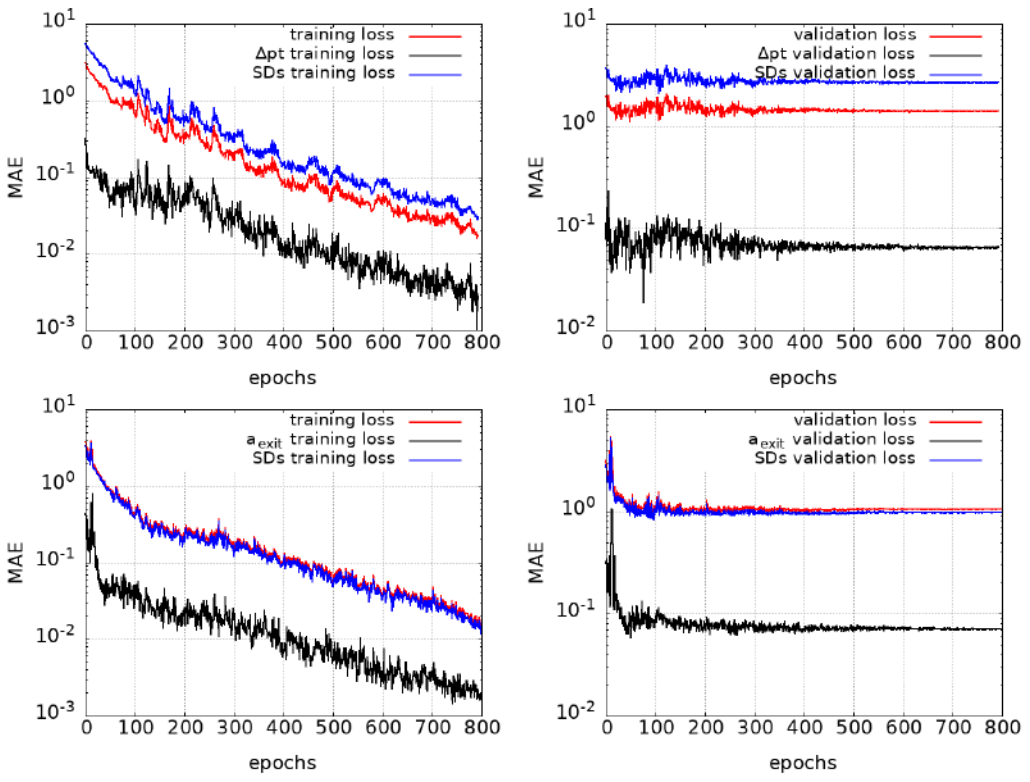


Figure 4.4: *The turbine blade-airfoil case: Top) The training loss convergence (left) and validation loss (right) of Δp_t DNN_B. Bottom) The training (left) and validation (right) loss curves of a_{exit} DNN_B. The convergence of the function and derivative terms is presented separately.*

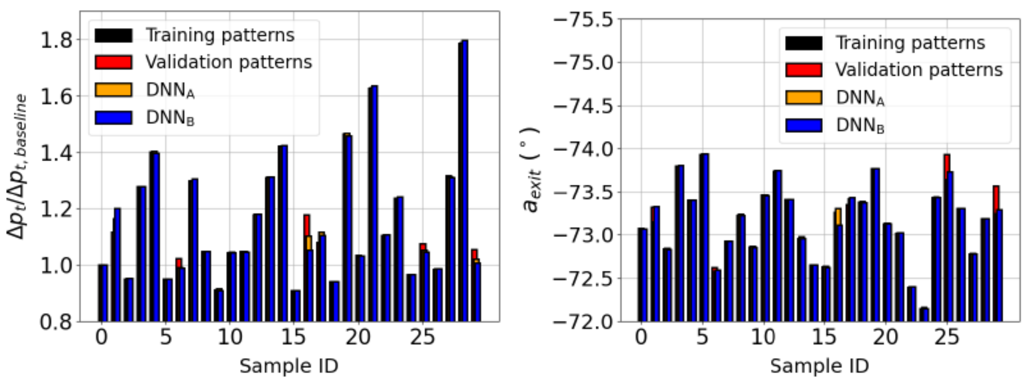


Figure 4.5: *The turbine blade-airfoil case: The Δp_t (right) and a_{exit} (left) values of the whole database computed with CFD and the DNNs predictions.*

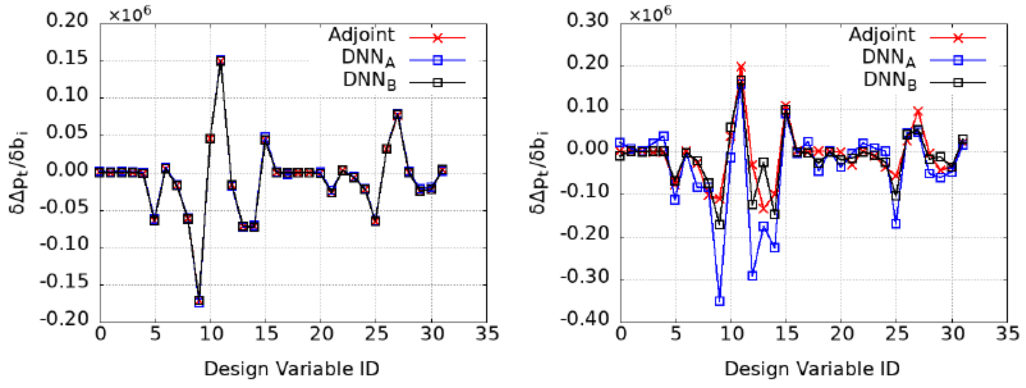


Figure 4.6: *The turbine blade-airfoil case: The Δp_t SDs of the baseline geometry (left) and a validation one (right), against reference values computed with adjoint. Baseline geometry is included in the training database.*

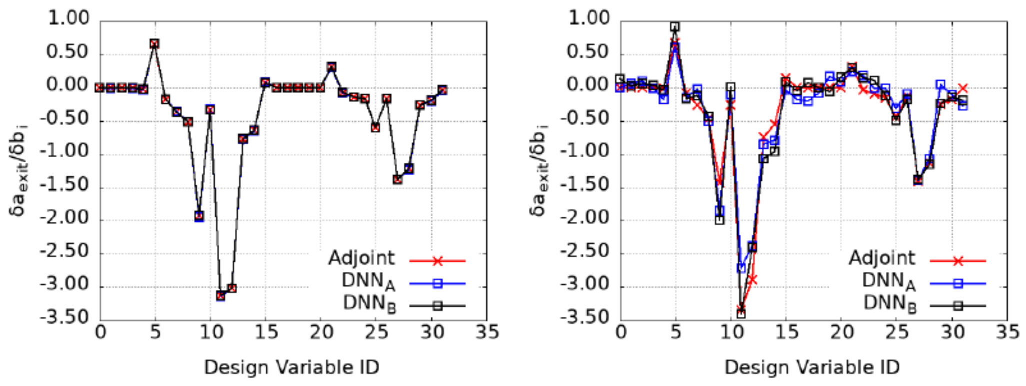


Figure 4.7: *The turbine blade-airfoil case: The baseline geometry (left) and a validation geometry (right) a_{exit} SDs computed with adjoint and the DNNs predictions.*

Expanding the design variable space of the DNN configuration optimization enabled the discovery of better solutions. Well-chosen values for the additional parameters of this Section helped the DNN_B achieve the same training performance as DNN_A, however with improved generalization. Ensuring more reliable predictions beyond the training database is important, since the DNNs will ultimately be used to guide the ShpO of the airfoil. Shuffling the training patterns proved beneficial for both models. By randomly rearranging the training data, the model is exposed to varying subsets in each batch, which helps prevent biases related to the data order. This encourages the learning of more general patterns and reduces the risk of overfitting. Additionally, a smaller batch size compared to that used in the initial configuration of DNN_A, was found to be optimal, confirming the efficiency of mini-batch training of DNNs. The optimal choice of loss function and optimizer's learning rate depends on the specific problem and should be tuned accordingly for each case.

4.3.3 Pruning

This Section explores an additional approach: DNN pruning. Pruning is a widely adopted technique aiming to reduce the complexity of the network by removing its redundant parameters. In this work, pruning is applied throughout the entire network, excluding the final two layers of each branch and the output layer of the network. Only the weight matrices are pruned, leaving the biases untouched, since they are fewer than the weights and their impact on the network's size is smaller. Two different magnitude-based pruning criteria are compared, and their results are demonstrated below. Each one is applied in an iterative manner with a frequency of 100 epochs, starting from around the 300th, until the limit of the 600th epoch is reached. Afterwards a fine-tuning phase follows, until convergence is reached.
The first approach maintains a gradually increasing sparsity, shown in Figure 4.8. During the initial epochs, sparsity increases rapidly, followed by a slower rate as the DNN parameters are progressively reduced. The frequency of applying pruning depends on the training configuration; however, it should not be applied frequently, as this may negatively impact accuracy. The global threshold is computed based on the target sparsity of each epoch. The optimized configurations are presented in Tables 4.6 & 4.7.
The second approach adapts a layer-wise threshold, in contrast to the earlier method that relied on a single global one. Within each layer, the significance of weights is evaluated by computing the mean value (μ) and standard deviation (σ) of their magnitudes. Weights with absolute values below either μ-σ or μ-2σ are considered less significant and can be trimmed. For this approach the optimized configurations of DNN_B are employed.

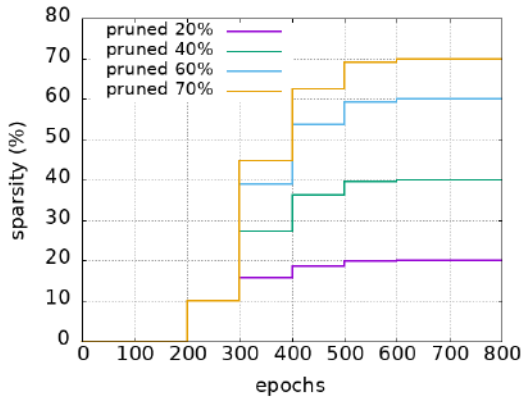


Figure 4.8: *The turbine blade-airfoil case: Different pruning schedules applied to the DNNs. The sparsity at different epochs during training is presented. Sparsity is defined as the ratio of the zero-valued weights to the total number of weights (dense matrix) in the layers targeted for pruning.*

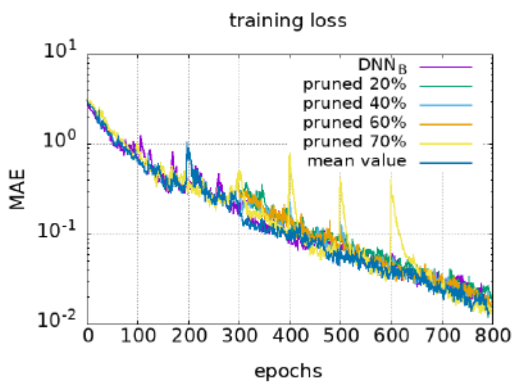
DNN Δp_t				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	11	1024-32-2048-1024-256-256-256-64-32-24-1	Gelu	Tanh
B_{grad}	7	4096-2048-64-512-256-24-32	Gelu	Tanh
	Optimizer	**Loss function**	**Batch Size**	**Shuffle**
	AdamW (lr=0.0014)	MAE	16	No

Table 4.6: *The turbine blade-airfoil case: Optimized architecture of Δp_t pruned DNN_C.*

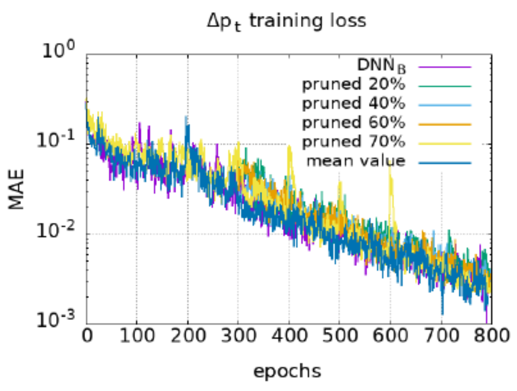
DNN a_{exit}				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	8	1024-32-256-128-512-256-24-1	Gelu	Gelu
B_{grad}	5	512-512-64-24-32	Gelu	Sigmoid
	Optimizer	**Loss function**	**Batch Size**	**Shuffle (epochs)**
	AdamW (lr=0.0027)	MAE	16	10

Table 4.7: *The turbine blade-airfoil case: Optimized architecture of a_{exit} pruned DNN_C.*

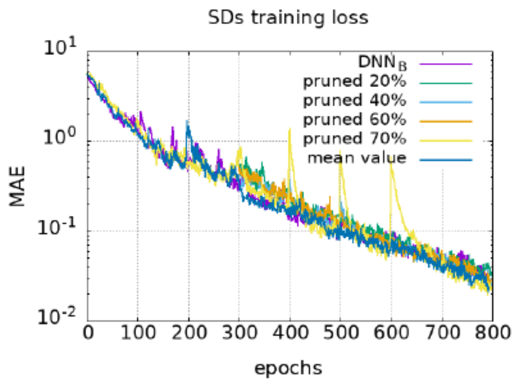
Figures 4.9 & 4.10 present the training and validation loss curves of Δp_t and a_{exit} DNN_C, pruned using each of the predefined criteria. At low sparsity levels, the curves are similar to those of DNN_B. An increase in loss appears at the 70% target sparsity, due to the significant reduction in network's parameters. Notably, the networks can still recover during the fine-tuning epochs. No further improvement is observed on validation predictions, possibly due to the small size of the DB_{LHS}. For the ShpO only the networks with the lowest validation loss are used. These correspond to the models pruned with the second approach, according to the mean value and deviation of each layer's weights. The benefits of pruning on the network's size can be observed by applying a standard compression algorithm: Δp_t DNN_C (84MB) is by 19% lighter than the original dense Δp_t DNN_B, while a_{exit} DNN_C (74MB) by 24%. The benefits of this approach would be more evident in memory-constrained environments, where the same prediction quality would be acquired with a much lighter network. Figure 4.11 presents the sparsity per layer for the two DNN_C; their predictions on the Δp_t, a_{exit} and their SDs are depicted in Figures 4.12 & 4.13.



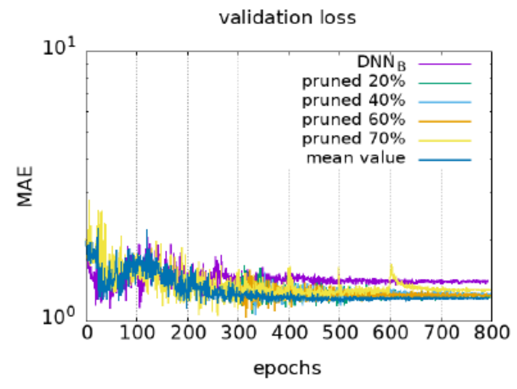
(a) training loss



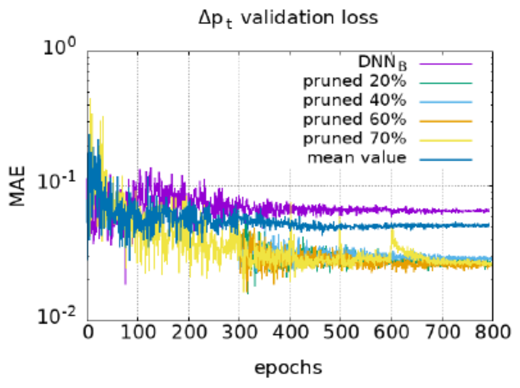
(b) training loss Δp_t



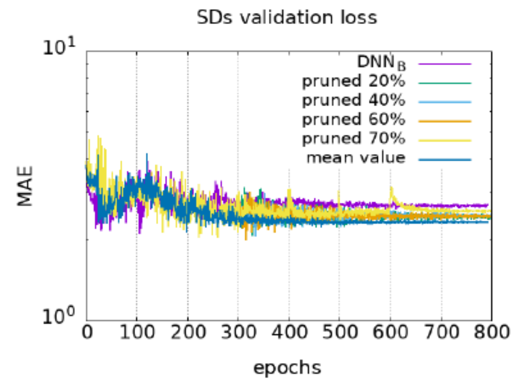
(c) training loss SDs



(d) validation loss

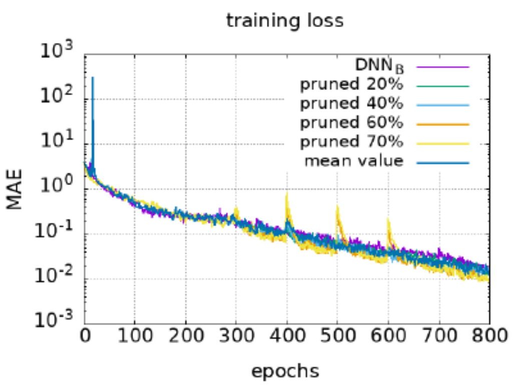


(e) validation loss Δp_t

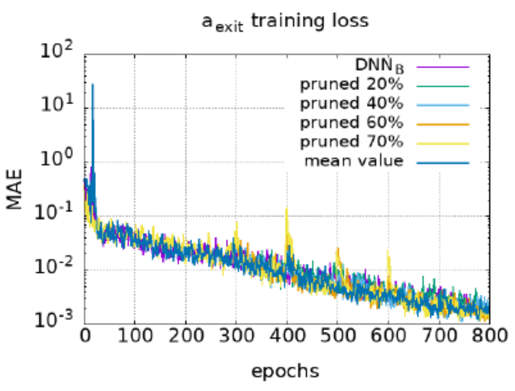


(f) validation loss SDs

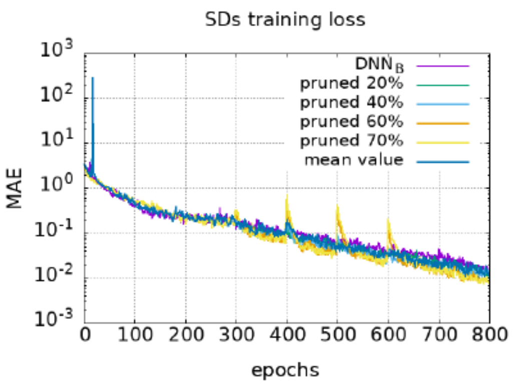
Figure 4.9: *The turbine blade-airfoil case: Training and validation loss curves of Δp_t DNN_C. The convergence of the Δp_t and SDs loss terms is compared using different pruning schedules.*



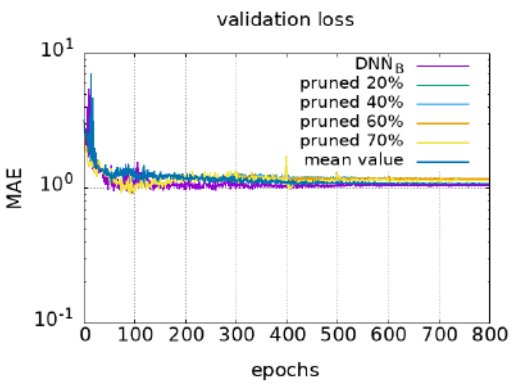
(a) training loss



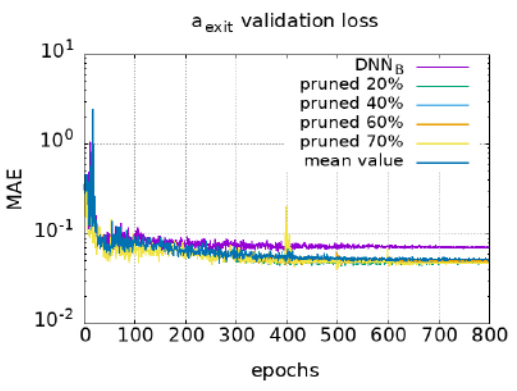
(b) training loss a_{exit}



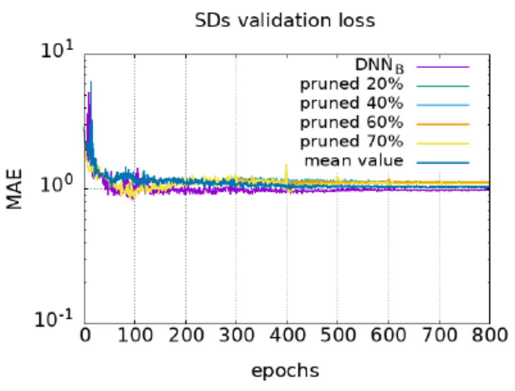
(c) training loss SDs



(d) validation loss

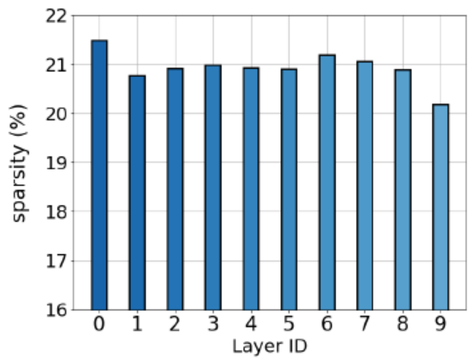


(e) validation loss a_{exit}

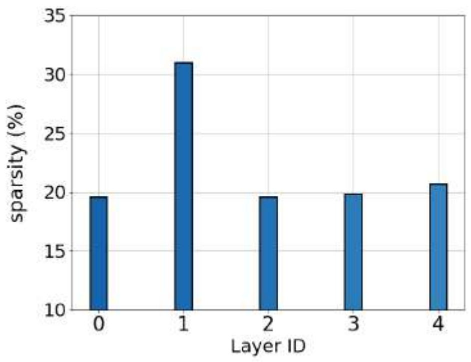


(f) validation loss SDs

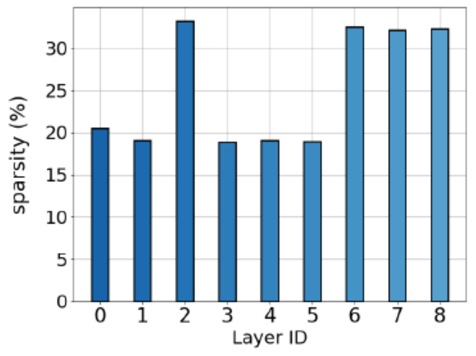
Figure 4.10: *The turbine blade-airfoil case: Training and validation loss curves of a_{exit} DNN_C. The convergence of the a_{exit} and SDs loss terms is also presented separately.*



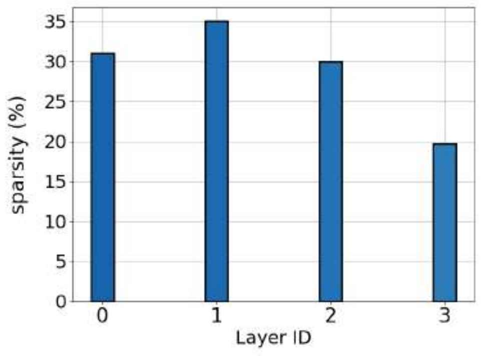
(a) B_{func} branch Δp_t DNN_C



(b) B_{grad} branch Δp_t DNN_C



(c) B_{func} branch a_{exit} DNN_C



(d) B_{grad} branch a_{exit} DNN_C

Figure 4.11: *The turbine blade-airfoil case: Sparsity for each branch of the Δp_t (top) and a_{exit} (bottom) DNN_C. Sparsity is defined as the ratio of the number of pruned weights to the total number of weights in each layer prior to pruning (dense layer).*

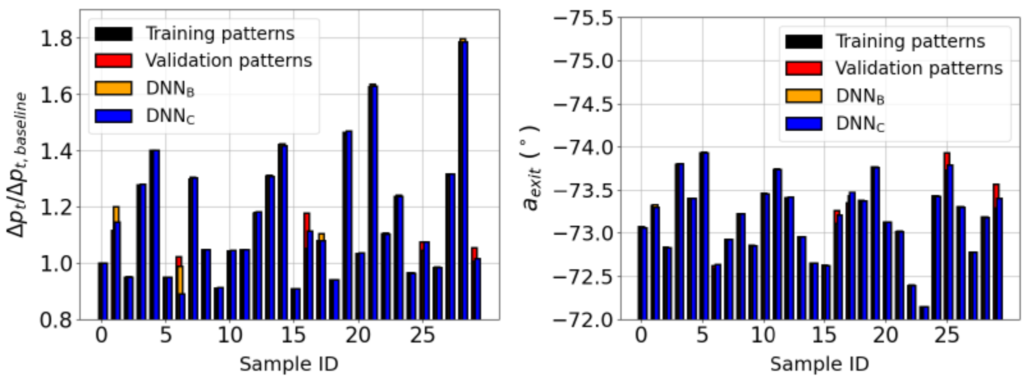


Figure 4.12: *The turbine blade-airfoil case: The target Δp_t (left) and a_{exit} (right) values computed with CFD and the DNNs predictions.*

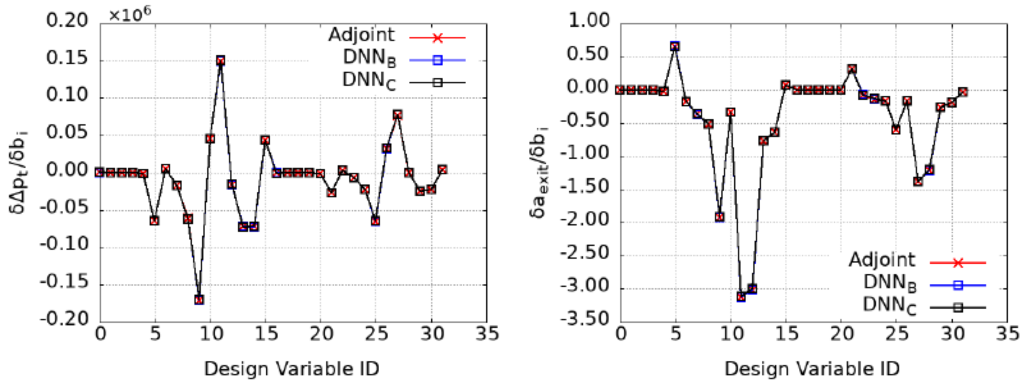


Figure 4.13: *The turbine blade-airfoil case: The baseline geometry SDs computed with adjoint and the DNN_C predictions.*

4.4 ShpO of the turbine blade-airfoil

Each Hermite-DNN is used to drive a distinct optimization processes of the turbine blade-airfoil. Each optimization relies exclusively on the DNNs predictions on the values of the objective function and its SDs. An adjoint-driven optimization is also performed, using PUMA. For a fair comparison, all optimizations are conducted using the L-BFGS algorithm. The convergence of the adjoint-driven process and the solutions obtained from the DNN-driven ones are shown in Figure 4.14. The objective is:

$$F = 10^{-8}\Delta p_t + 10^{-1}(a + 1.275)^2 \tag{4.1}$$

where $a_{\text{target}} = a_{\text{baseline}} = -1.275 rad$.
Each optimization cycle requires 3 Time Units (TU), 1 TU for the primal evaluation, and 2 TUs for the adjoint solution. The DB_{LHS} contains 30 blades, as a result the cost of constructing it amounts to 90 TUs. The optimized solution of each DNN-driven optimization is re-evaluated on the CFD solver. The re-evaluated values along with their SDs are added to the DB_{LHS} and the networks are re-trained. The optimization is repeated from the latest solution using the re-trained networks. Two re-trainings are necessary to achieve an optimized solution similar to the adjoint-driven optimization. Since the cost associated with DNN training is negligible compared to a CFD evaluation, the total cost of the DNN-driven optimization process is 97 TUs. The adjoint-driven optimization requires 37 TUs to converge to its optimized solution.

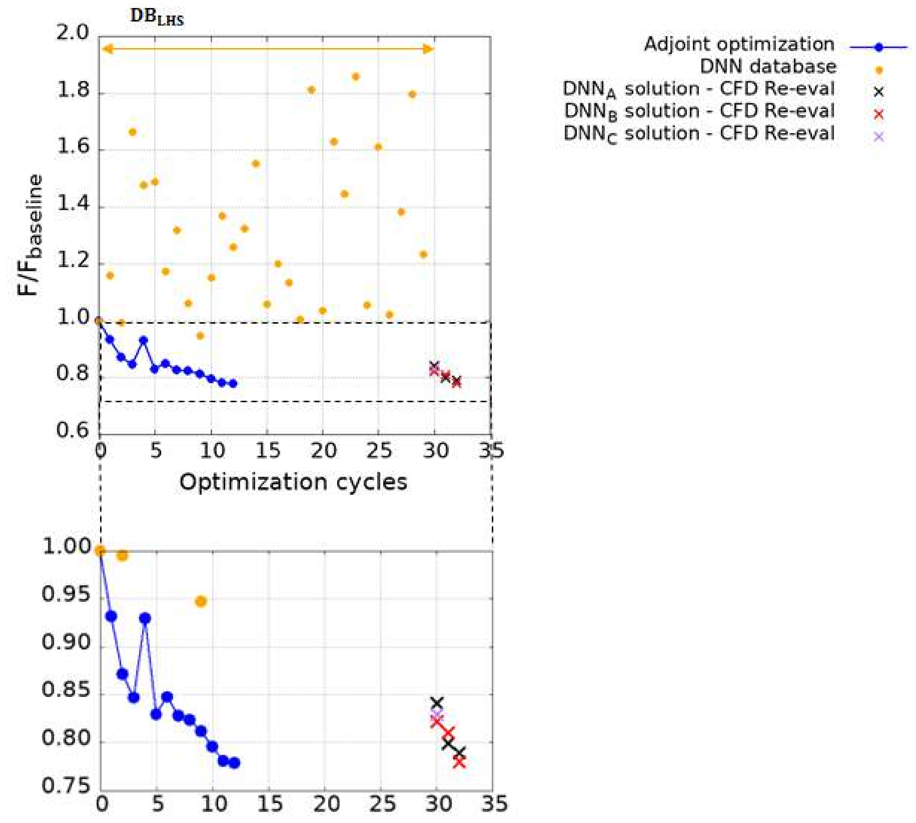


Figure 4.14: *The turbine blade-airfoil case: Convergence of the adjoint-driven optimization, the DB_{LHS} and the DNN-driven optimized solutions, after they are re-evaluated on the CFD tool. For each DNN solution, the cost of constructing the database has been taken into consideration.*

As shown in Figure 4.14 only the DNN_B-driven optimization achieves a similar solution with the adjoint-driven one. Although the cost of constructing the DNN database is higher than that of the adjoint-driven optimization, once the DNN is trained, it can be re-used to drive an optimization process. For instance, when the objective changes or an optimization with relaxed constraint needs to be performed the cost of a DNN-driven optimization is minimal relative to re-starting the entire optimization process.

4.5 Reducing DNN database construction cost

The DNN-driven optimization of the previous Section had a high computational cost, due to the higher cost of constructing the DB_{LHS}. In order to mitigate this, the number of training patterns should be reduced. Herein a different approach is explored. A new database is formed, to be referred to as $DB_{adjoint}$, using the initial five solutions of the adjoint-driven optimization. The Δp_t, a_{exit} values of these blades and their corresponding SDs constitute the $DB_{adjoint}$. As previously, Hermite-DNNs are employed for this study, and their configuration is optimized using the first setup introduced in the previous Section. Due to the restricted number of samples, the entire database is utilized during the training process. The optimized DNN (to be referred as DNN_D) architectures are presented in Tables 4.8 & 4.9. Training loss convergence is shown in Figure 4.15, and each network's predictions in Figures 4.16 & 4.17.

DNN Δp_t				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	8	2048-64-512-64-512-128-5-1	Relu	Gelu
B_{grad}	12	512-128-4096-128-32-64-256-1024-32-32-5-32	Relu	Gelu

Table 4.8: *The turbine blade-airfoil case: Optimized architecture of Δp_t DNN_D.*

DNN a_{exit}				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	5	4096-128-64-5-1	Gelu	Relu
B_{grad}	7	256-32-2048-1024-64-5-32	Gelu	Gelu

Table 4.9: *The turbine blade-airfoil case: Optimized architecture of a_{exit} DNN_D.*

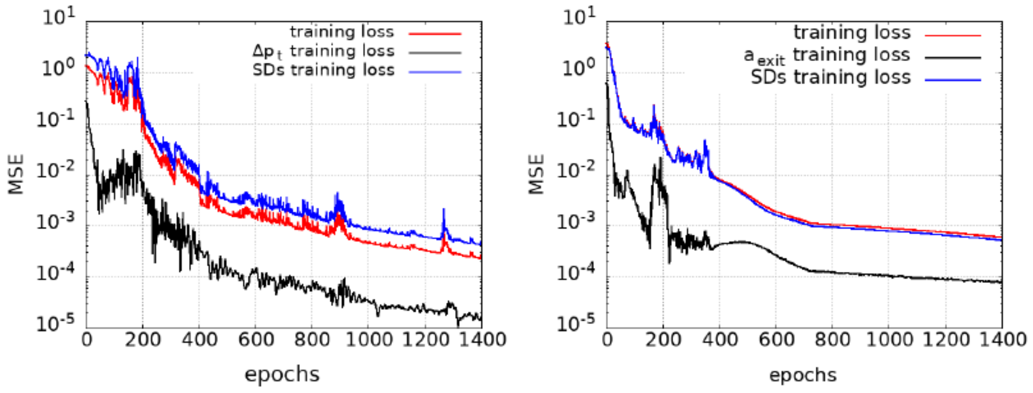


Figure 4.15: *The turbine blade-airfoil case: Training loss of Δp_t (left) and a_{exit} (right) DNN_D. Both models have converged after they are trained for 1400 epochs.*

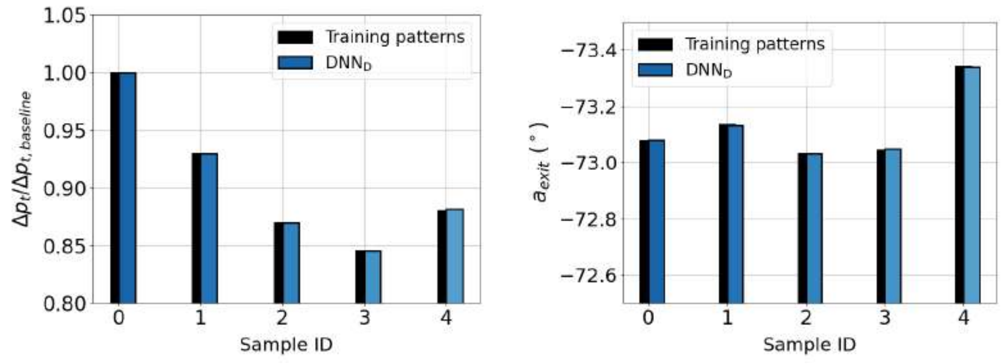


Figure 4.16: *The turbine blade-airfoil case: Training patterns computed with CFD and DNN_D predictions.*

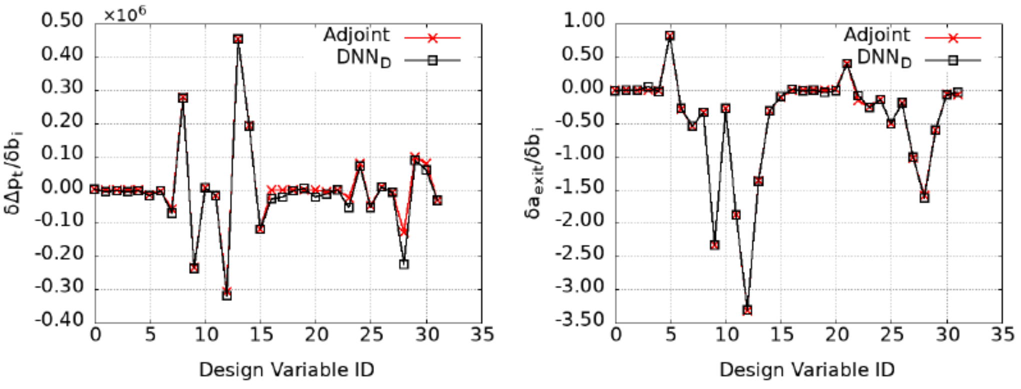


Figure 4.17: *The turbine blade-airfoil case: Δp_t SDs (left) and a_{exit} SDs (right) of the fifth point of the adjoint-driven optimization computed with adjoint and DNN_D predictions.*

The adjoint-initiated ShpO from its fifth point onward is driven exclusively by the DNN_D, as shown in Figure 4.18. The re-evaluation cycles described before, are applied during the DNN-driven phase. Three re-evaluations of intermediate optimized solutions are required to reach a solution of the same quality with the adjoint-driven optimization. The DNN-driven optimization turnaround time amounts to 25 TUs, achieving a 32% cost reduction.
A comparison of the Figures 4.14 & 4.18 reveals that LHS yields a representative sampling of the design variable space, resulting in a database that encompasses a broader variety of blade geometries. However some designs are far from the optimal ones, leading to unnecessary CFD evaluations. In contrast, the initial samples generated through adjoint-driven optimization are concentrated and provide more information in the desired direction. This targeted approach ensures that the networks learn patterns useful for guiding the optimization process, and restricts significantly the database generation turnaround time.

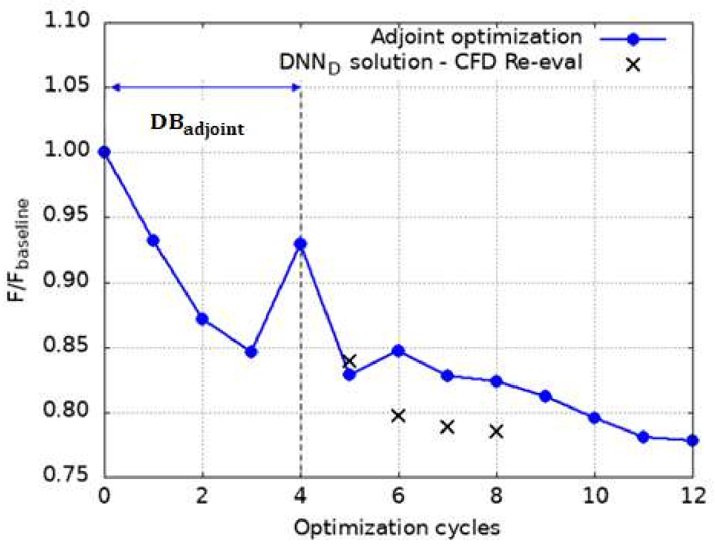


Figure 4.18: *The turbine blade-airfoil case: Convergence of the adjoint-driven optimization and DNN_D solutions, after they are re-evaluated on the CFD code.*

The optimized solutions are presented in Table 4.10 & Figure 4.19. In Figure 4.20, the airfoils of the DB_{LHS} and $DB_{adjoint}$ are presented. The airfoils generated with LHS cover a wider range of potential geometries, while the geometries obtained through the adjoint-driven optimization are closer to the optimized airfoils. The shape of the optimized geometries is compared in Figure 4.21 with the baseline airfoil. The Mach number fields are shown in Figure 4.22.

Comparison of the Optimized Solutions					
	$\frac{F}{F_{\text{baseline}}}$ (%)	Δp_t **(Pa)**	a_{exit} (°)	Δp_t **Reduction (%)**	Δa_{exit} (°)
Baseline	-	4.189×10^3	-73.08	-	-
Adjoint solution	77.83	3.260×10^3	-73.09	22.18	-0.01
DNN$_A$ solution	78.94	3.305×10^3	-73.10	21.11	-0.02
DNN_B **solution**	78.00	3.266×10^3	-73.10	22.03	-0.02
DNN_C **solution**	82.89	3.470×10^3	-73.10	17.16	-0.02
DNN_D **solution**	78.48	3.286×10^3	-73.06	21.56	0.02

Table 4.10: *The turbine blade-airfoil case: Comparison of optimized solutions.*

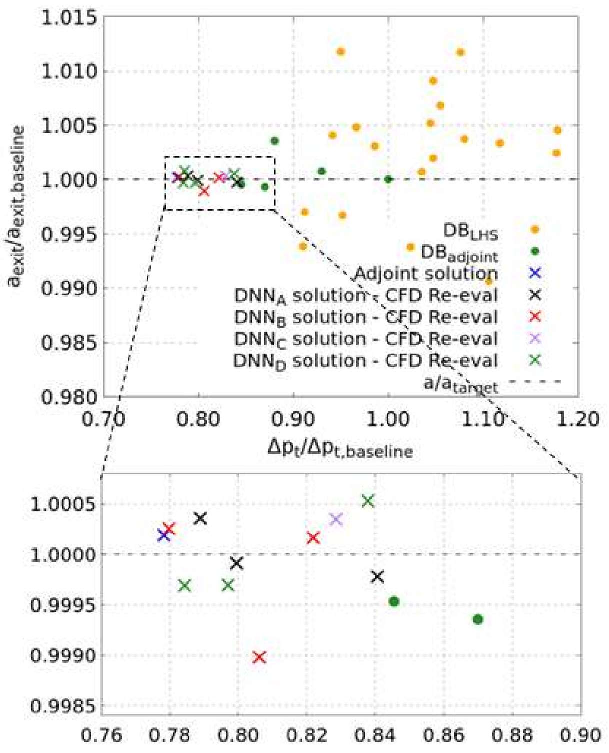


Figure 4.19: *The turbine blade-airfoil case: Solutions obtained from each DNN-driven optimization, after they are re-evaluated on CFD, and adjoint optimal solution. The Δp_t, a_{exit} values are normalized with those of the baseline geometry, respectively.*

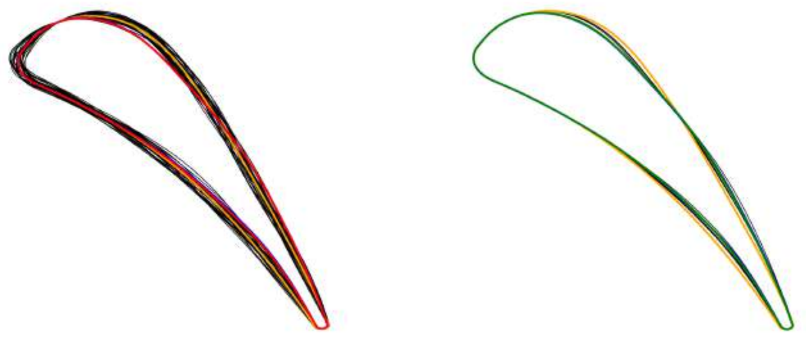


Figure 4.20: *The turbine blade-airfoil case: Left) Geometries generated with LHS (black) are compared to the baseline (orange) and the optimized airfoils obtained with adjoint (blue) and DNN_B (red). Right) The first solutions of the adjoint-driven optimization (black) along with the baseline geometry (orange), the adjoint optimized solution (blue) and DNN_D optimized solution (green).*

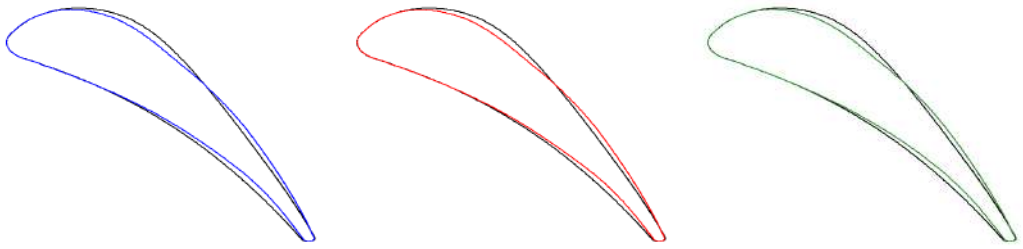


Figure 4.21: *The turbine blade-airfoil case: Optimized geometries resulting from adjoint (blue), DNN_B (red), and DNN_D (green), are compared to the baseline airfoil (black).*

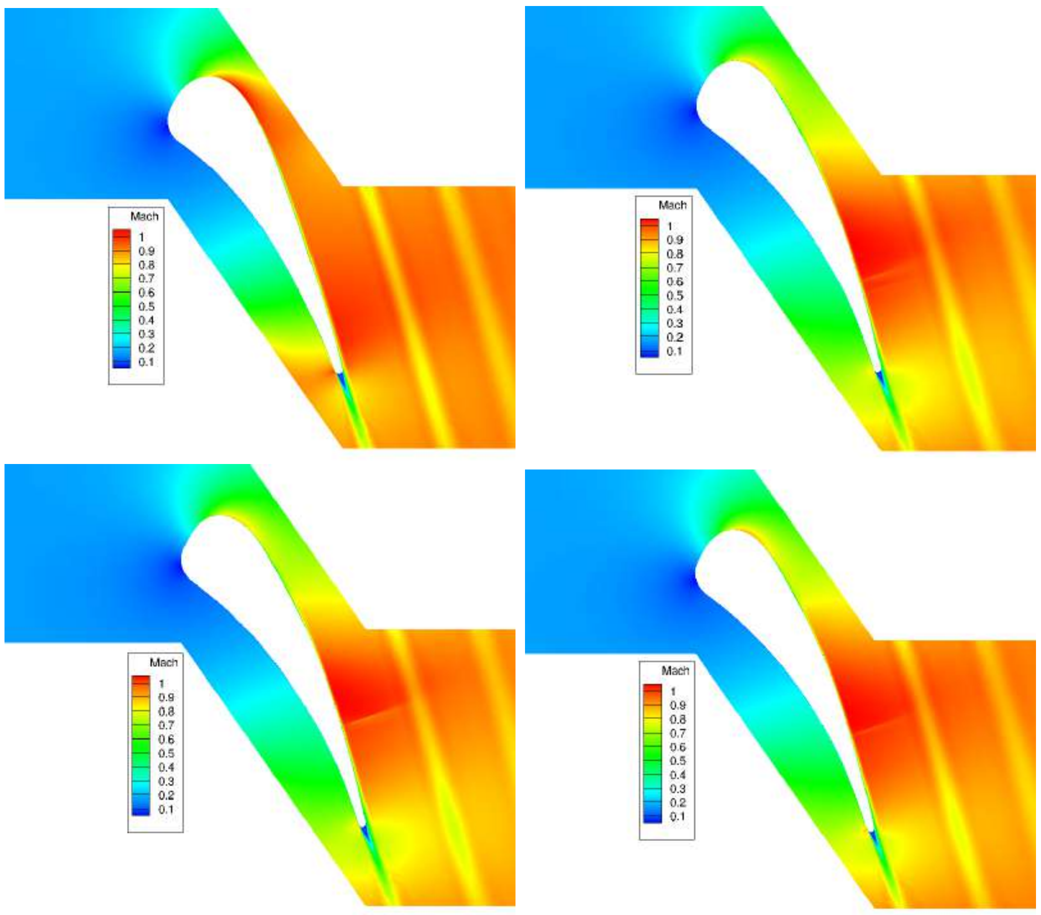


Figure 4.22: *The turbine blade-airfoil case: The Mach number fields for the baseline geometry (top left) and the optimized geometries resulting from adjoint (top right), DNN_B (bottom left) and DNN_D (bottom right).*

Chapter 5

Single-phase Transitional Flow around an Isolated Airfoil

5.1 Introduction

In this Chapter, the aerodynamic ShpO of an isolated airfoil is carried out. Objective of the optimization is the minimization of the drag coefficient (c_d) while maintaining the lift coefficient (c_l) close to the value of the baseline airfoil. The airfoil's polar diagram is first validated against the experimental one. Subsequently, Hermite-trained DNNs are employed to guide the ShpO process. Each DNN-driven optimization relies solely on the predictions generated by the models. Two strategies for constructing the DNN database are examined: the first involves a near-random sampling of the design variable space, and the second utilizes initial solutions obtained from an adjoint-driven optimization. Several DNN configurations are assessed regarding their accuracy, generalization capabilities, and overall performance in the optimization task.

5.2 Flow conditions and parameterization

The RG15 airfoil is a low Reynolds number airfoil originally developed for sailplane applications. Subsequently a family of airfoils derived from the RG15 with varying relative thicknesses was introduced for use in small horizontal axis wind turbines [47]. A C-type mesh, shown in Figure 5.1, is used with approximately 70K nodes. Farfield boundaries are located approximately 100 chords away from the airfoil.
The airfoil's polar diagram is validated against experimental data in Figure 5.2. The available measurements span a range of Reynolds numbers from Re=61.400 to Re=304.200 ([48]). Simulations are conducted at Re=304.200 and free-stream Mach number of 0.01. In the figure, results obtained using only a turbulence

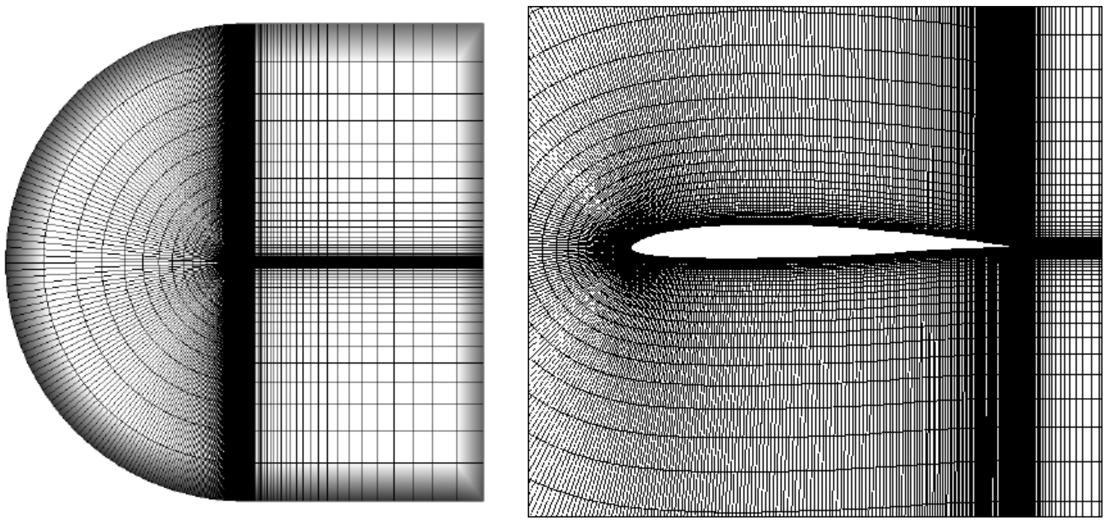


Figure 5.1: *The isolated airfoil case: Computational mesh of the entire domain (left) and close up view (right).*

model are shown with blue markers, while red markers represent simulations that incorporate both turbulence and transition models. Turbulence is modeled using the Spalart–Allmaras model (3.2), and transition with the γ-$\tilde{Re}_\theta$ model (the SA-sLM2015 variant, 3.3). It can be observed that the use of only turbulence model leads to an overestimation of the c_d. It is essential to incorporate a transition model to accurately capture the correct boundary layer behavior.

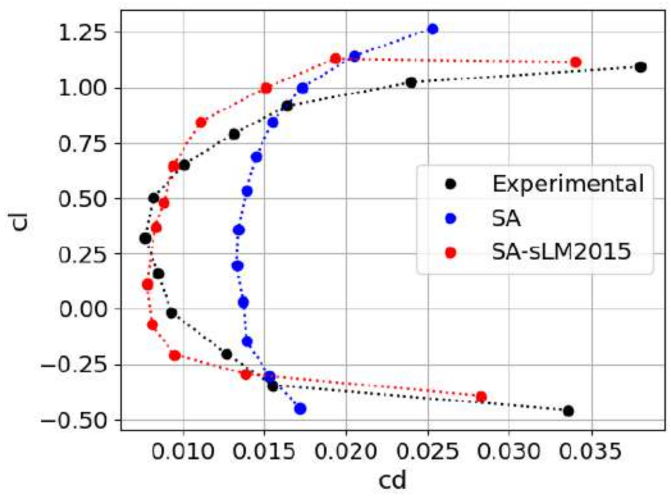


Figure 5.2: *The isolated airfoil case: Airfoil's polar based on experimental data (black), CFD results assuming turbulent flow (blue) and transitional flow (red). Turbulent flow overestimates the cd values.*

For the subsequent analysis, the free-stream velocity is slightly increased, since the variant of PUMA this diploma thesis is based upon solves compressible flows. From now on, transition model will be also included in the modeling. The flow conditions are listed in Table 5.1.

Flow Conditions	
M_∞	0.1
Re	$1.5 \cdot 10^5$
a_∞	2°

Table 5.1: *The isolated airfoil case: Flow conditions.*

The airfoil is parameterized with a 10 x 9 NURBS lattice, shown in Figure 5.3. The black control points remain fixed during the optimization, while the red ones are allowed to move within $\pm 0.4d$ of their initial positions, in order to avoid overlapping. Here d equals the vertical distance between adjacent control points. The CPs are displaced only in the direction normal to the chord, resulting in a total of 28 design variables ($\vec{b} \in R^{28}$).

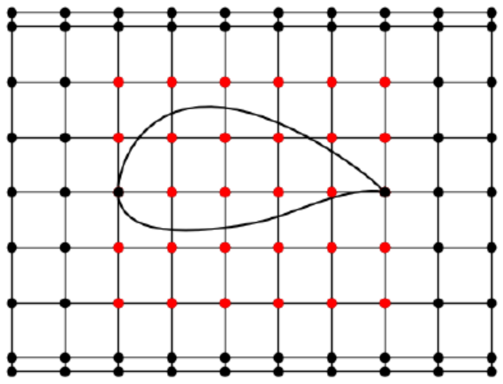


Figure 5.3: *The isolated airfoil case: Airfoil parameterization.*

5.3 DNN Configuration and Training

Two approaches are compared for constructing the database: the first approach uses samples generated via LHS, and the second uses the initial candidate solutions from the adjoint-driven optimization of the airfoil.

5.3.1 Database constructed with LHS

To construct the DNN database, 19 distinct combinations of the design variables are generated using LHS, which correspond to 19 different airfoils. Each airfoil is evaluated using the CFD solver, herein the c_d, c_l values and their SDs are computed. These quantities, along with the baseline geometry added after sampling, form the DNN database (DB_{LHS}). Separate models are employed for each coefficient. Input to each network is the [N×28] tensor containing the y-coordinates of the control points, where N equals the batch size. Output is the [N×1] tensor with the predictions on the coefficients. During training, the models are differentiated to compute the derivatives of their outputs w.r.t. their inputs, that correspond to the SDs for each quantity of interest. A fixed validation set, containing 20% of the DB_{LHS} samples is selected, to evaluate model performance. As in the previous case, two DNN configuration optimizations are conducted, and their outcomes are compared below.

First DNN configuration optimization

The optimized hyperparameters are based on the initial setup introduced in Chapter 4. Objective is the minimization of $\mathcal{L}_{\text{Hermite}}$, after each network has been trained for the first 200 epochs. The MSE is used for each term of the $\mathcal{L}_{\text{Hermite}}$, and the optimizer is Adam with its default learning rate of 0.001. Tables 5.2 & 5.3 summarize the optimized architectures. The training and validation loss convergence is depicted in Figure 5.4.

DNN c_d				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	11	128-1024-64-4096-4096-32 -2048-64-64-16-1	Gelu	Gelu
B_{grad}	9	128-2048-32-32-32-1024-64-16-28	Gelu	Sigmoid

Table 5.2: *The isolated airfoil case: Optimized architecture of c_d DNN_A.*

DNN c_l				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	5	2048-4096-128-16-1	Gelu	Tanh
B_{grad}	11	4096-128-64-256-128-4096-32-1024 -256-16-28	Tanh	Tanh

Table 5.3: *The isolated airfoil case: Optimized architecture of c_l DNN_A.*

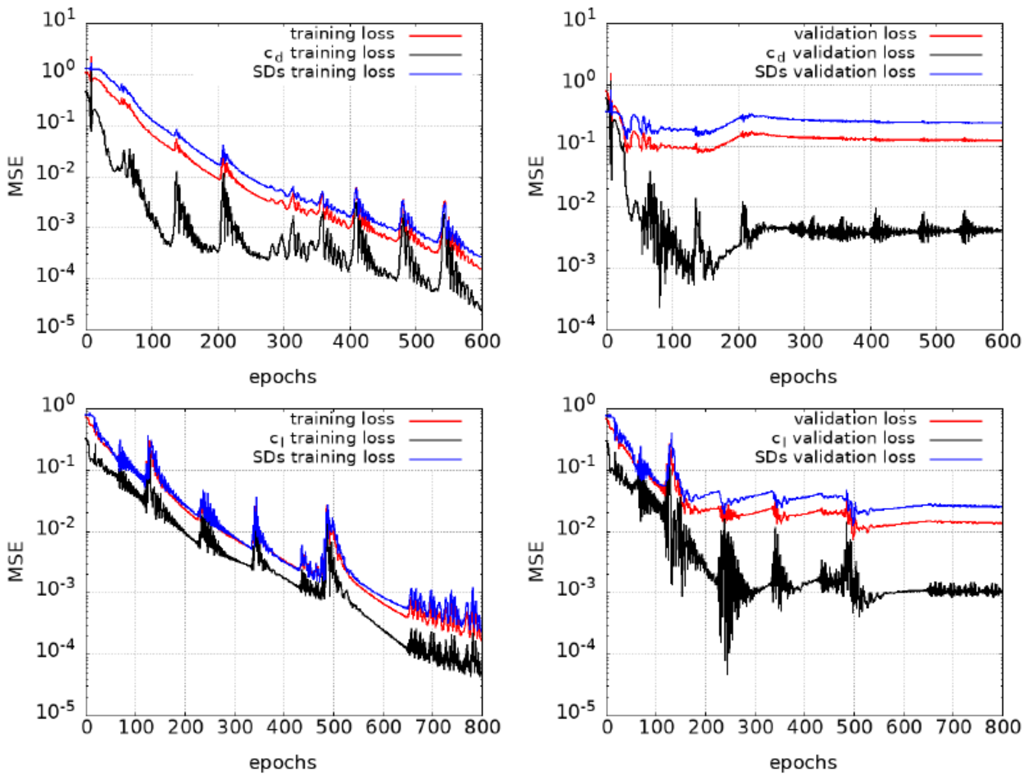


Figure 5.4: *The isolated airfoil case: Left) Training loss convergence of c_d (top) and c_l (bottom) DNN_A. Right) Validation loss of c_d (left) and c_l (right) DNN_A, monitored during training. The total loss is the sum of the function loss and gradient loss.*

Second DNN configuration Optimization

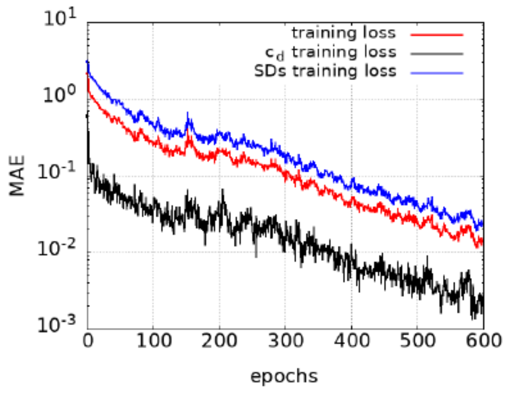
The design variable space of the DNN configuration optimization is expanded to enhance the search for better DNN configurations. For the specific optimization, the second setup introduced in Chapter 4 is used. Objective remains the minimization of $\mathcal{L}_{\text{Hermite}}$, evaluated over the entire database (20 samples) after the first 240 training epochs. The optimized architectures are presented in Tables 5.4 & 5.5. The training loss convergence of the c_d, c_l DNN_B is illustrated in Figure 5.5. As DNN_A and DNN_B for both coefficients are trained using different loss functions, their loss values are not directly comparable. Therefore their predictions on the target outputs are compared in Figure 5.6. An improvement is observed on validation c_l predictions and a slight degradation on validation predictions of c_d DNN_B. However both DNN_B demonstrate improved accuracy in predicting the SDs in Figures 5.7 & 5.8. The validation loss comprises multiple terms, including the function prediction error and loss terms associated with each partial derivative. The superior validation performance of c_d DNN_B is attributed to its enhanced accuracy in approximating these derivatives.

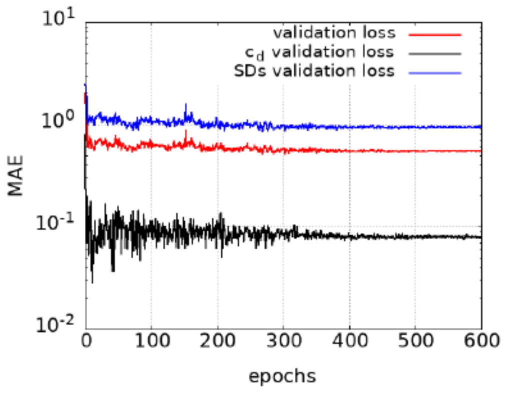
DNN c_d				
	Layers	Neurons	Activation Function	
			Hidden Layers	Final Layer
B_{func}	9	64-2048-64-1024-4096-256-64 -16-1	Tanh	Gelu
B_{grad}	8	1024-2048-2048-256-256-64-16-28	Tanh	Tanh
	Optimizer	**Loss function**	**Batch Size**	**Shuffle**
	Adam (lr=0.0003)	MAE	12	No

Table 5.4: *The isolated airfoil case: Optimized architecture of c_d DNN_B.*

DNN c_l				
	Layers	Neurons	Activation Function	
			Hidden Layers	Final Layer
B_{func}	11	128-4096-64-64-64-2048-128-32-512 -16-1	Gelu	Tanh
B_{grad}	5	512-1024-64-16-28	Tanh	Tanh
	Optimizer	**Loss function**	**Batch Size**	**Shuffle**
	Adamax (lr=0.005)	MAE	12	No

Table 5.5: *The isolated airfoil case: Optimized architecture of c_l DNN_B.*





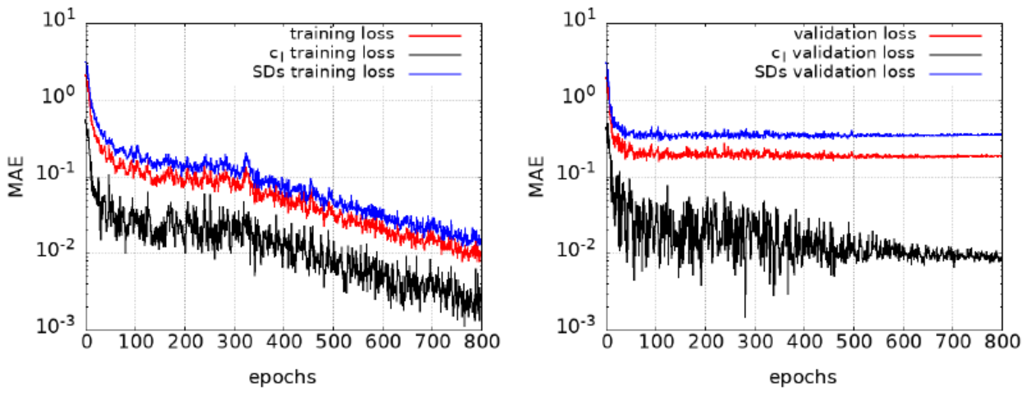


Figure 5.5: *The isolated airfoil case: The training (left) and validation (right) loss curves of c_d DNN_B (top) and c_l DNN_B (bottom).*

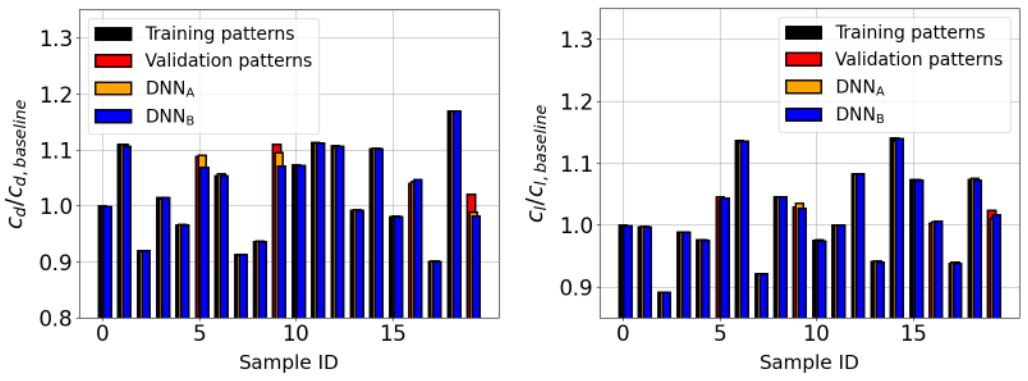


Figure 5.6: *The isolated airfoil case: The target values of training and validation patterns computed with CFD and DNNs predictions. The c_d, c_l values are normalized with the values of the baseline airfoil.*

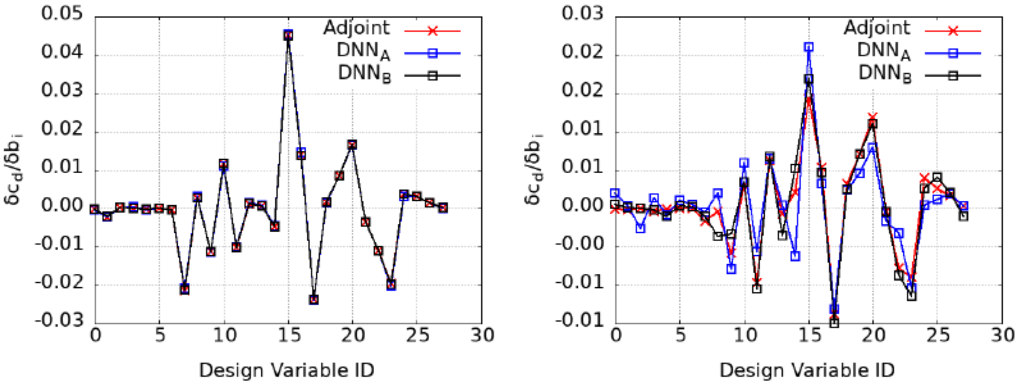


Figure 5.7: *The isolated airfoil case: The c_d SDs for the baseline airfoil (left), which is included in the training database, and a validation airfoil (right).*

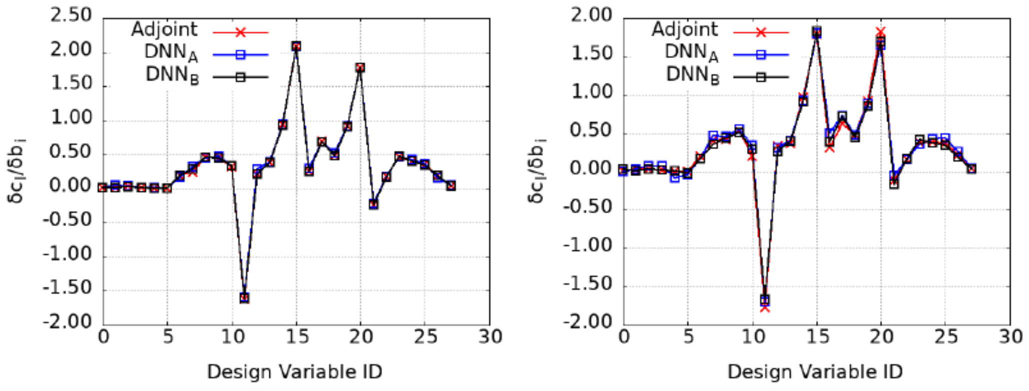
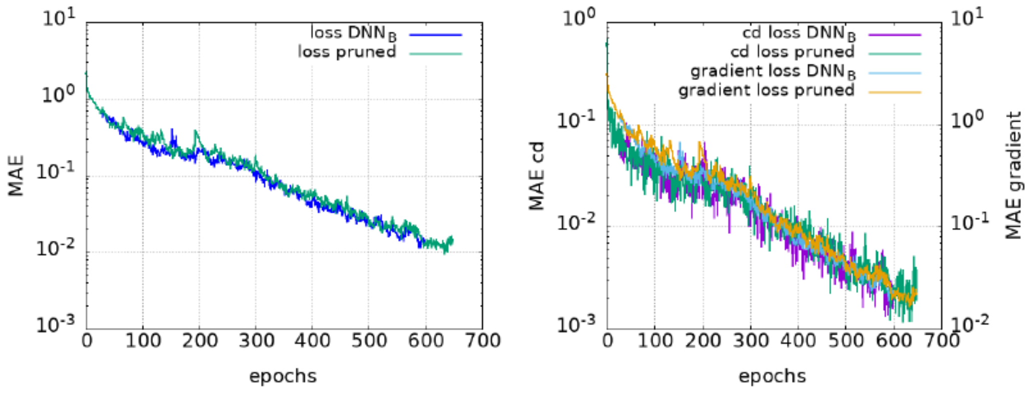


Figure 5.8: *The isolated airfoil case: The c_l SDs for the baseline airfoil (left) and a validation airfoil (right).*

Pruning

In this Section, the impact of pruning is investigated, regarding the network's accuracy and size. The implementation employs the optimized architectures of DNN_B, introduced in the previous subsection. Pruning begins at the 300th epoch and recurs every 100 epochs until the 600th epoch is reached. A magnitude-based pruning criterion is employed, the second one introduced in Chapter 4. The training and validation loss curves for each model are shown in Figures 5.9 & 5.10, alongside those of the initial dense network (DNN_B). The curves corresponding to the pruned networks exhibit similar shape to those of the dense models and ultimately reach the same level of accuracy. This suggests that a significant fraction of the weights have minimal impact on the network's performance and, therefore, can be safely removed.



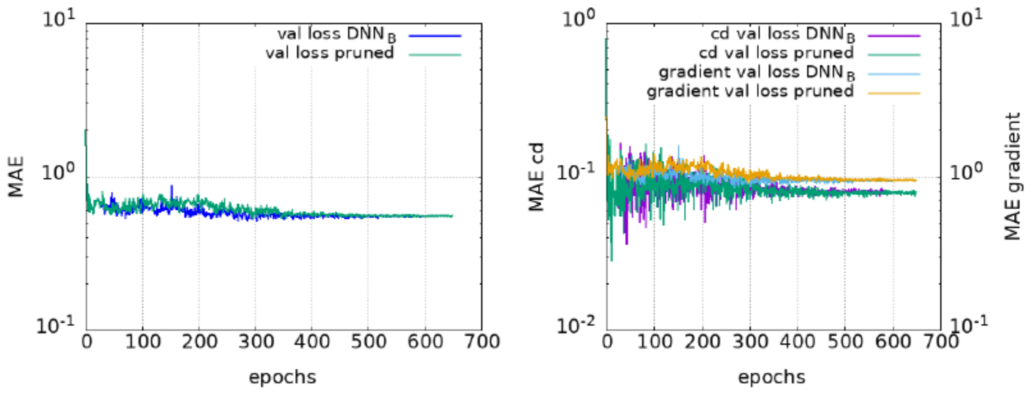


Figure 5.9: *The isolated airfoil case: The training (top) and validation (bottom) loss curves of c_d pruned DNN_C are compared to those of the dense DNN_B. The comparison of each term that make up the total loss is also presented separately (left).*

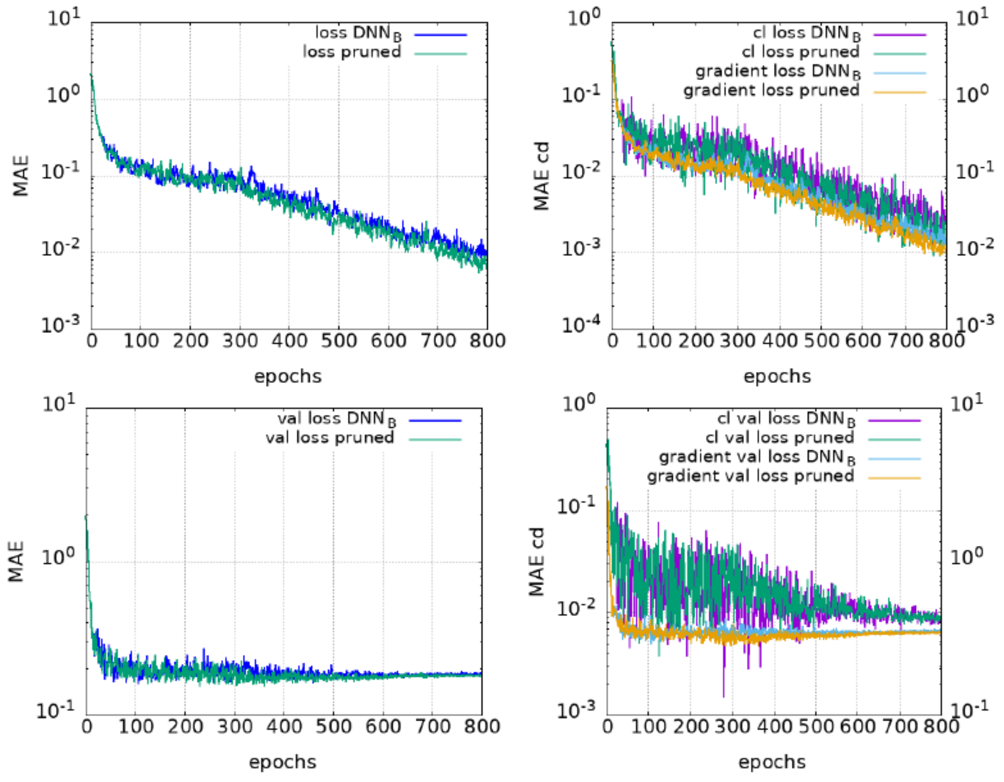
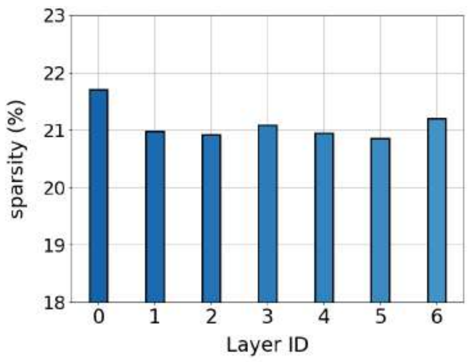
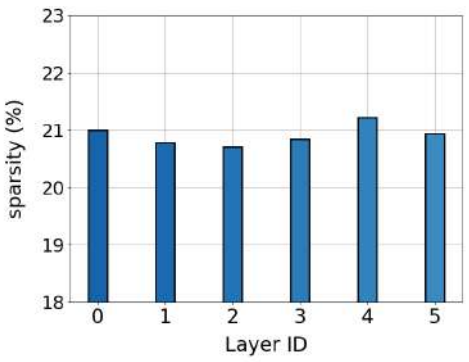
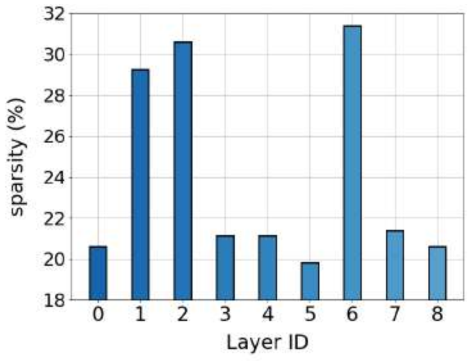


Figure 5.10: *The isolated airfoil case: Training (top) and validation (bottom) loss curves of c_l DNN_C and DNN_B, along with the convergence of each term (left).*

Figure 5.11 shows the sparsity ratio for each layer of the network, defined as the fraction of the number of pruned weights on that layer to the total number of weights in its dense form. The pruned c_d DNN_C is by 15% smaller in size (38MB) compared to the dense DNN_B, and the pruned c_l DNN_C achieves a 24% reduction (5MB). The DNN_C predictions are presented in Figures 5.12 & 5.13.


(a) B_{func} branch c_d DNN_C


(b) B_{grad} branch c_d DNN_C


(c) B_{func} branch c_l DNN_C

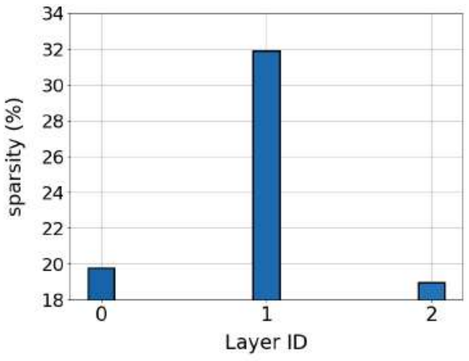
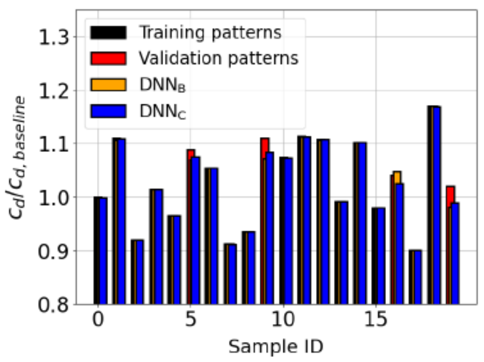

(d) B_{grad} branch c_l DNN_C

Figure 5.11: *The isolated airfoil case: Sparsity for each branch of c_d DNN_C (top) and c_l DNN_C (bottom).*



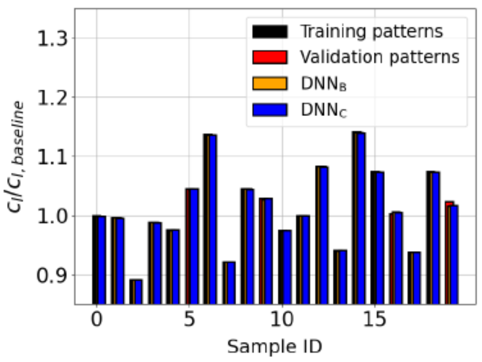


Figure 5.12: *The isolated airfoil case: Comparison of the DNN_B and DNN_C predictions on the c_d (left) and c_l (right) of the airfoils, against the CFD computed values.*

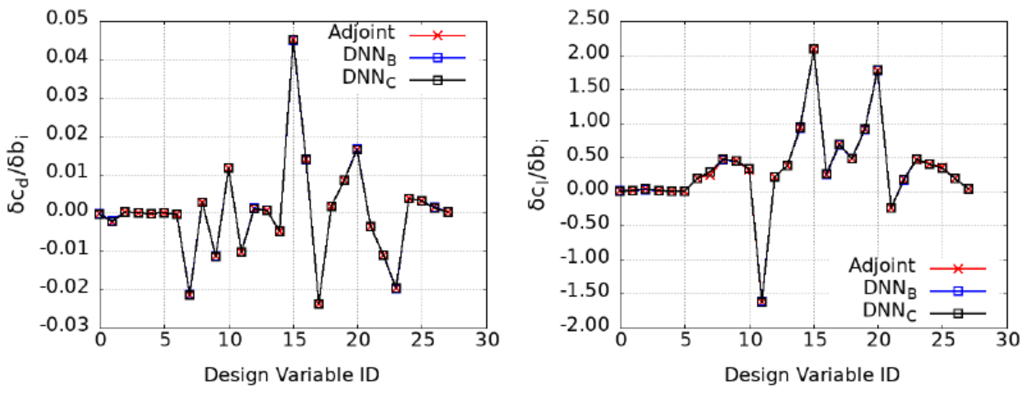


Figure 5.13: *The turbine blade-airfoil case: The c_d (left) and c_l (right) SDs computed with CFD and DNNs predictions.*

5.3.2 Database constructed with adjoint-driven optimization solutions

This approach constructs the DNN database using the initial solutions of the adjoint-driven ShpO. More specifically, the c_d, c_l values and their corresponding SDs, for the six initial airfoils, form the database (DB$_{adjoint}$). As previously, the Hermite-DNNs are employed in this study. Since the c_d DNN_B will be employed in the ShpO to minimize the airfoil's c_d, the minimum c_d value for normalization of its inputs is 5% smaller, than the smallest value encountered in the training patterns. Moreover, since the smallest SDs differ significantly in magnitude from the others, the input bounds for the specific design variables are expanded by increasing the maximum by 5% of its value and decreasing the minimum by 5%. This increases the scale of the corresponding SDs and facilitates a smoother learning. The optimized DNN (to be referred as DNN_D) architectures are presented in Tables 5.6 & 5.7. In Figure 5.14 the convergence of the training process for each model is shown, and their predictions are depicted in Figures 5.15 & 5.16.

DNN c_d				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	5	2048-4096-32-6-1	Gelu	Gelu
B_{func}	7	4096-64-4096-256-128-6-28	Gelu	Sigmoid
	Optimizer	**Loss function**	**Batch Size**	
	AdamW (lr=0.0014)	MSE	6	

Table 5.6: *The isolated airfoil case: Optimized architecture of c_d DNN_D.*

DNN c_l				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	5	128-32-2048-6-1	Gelu	Gelu
B_{grad}	7	128-32-512-32-512-6-32	Gelu	Tanh
	Optimizer	**Loss function**	**Batch Size**	
	AdamW (lr=0.005)	MSE	6	

Table 5.7: *The isolated airfoil case: Optimized architecture of c_l DNN_D.*

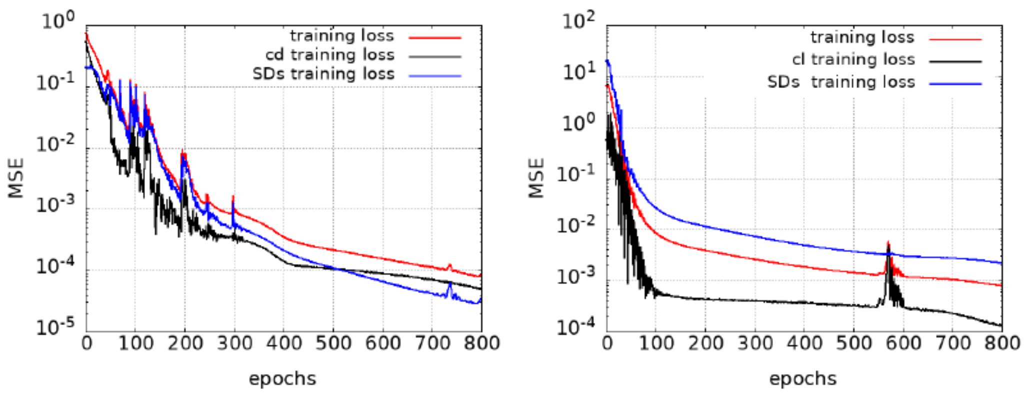


Figure 5.14: *The isolated airfoil case: Training loss of c_d DNN_D (left) and c_l DNN_D (right). Total loss is the weight sum of the coefficients loss and the SDs loss. Since the number of training patterns is small the entire database is used for training, and no separate validation set is provided. After 800 epochs the training loss has converged.*

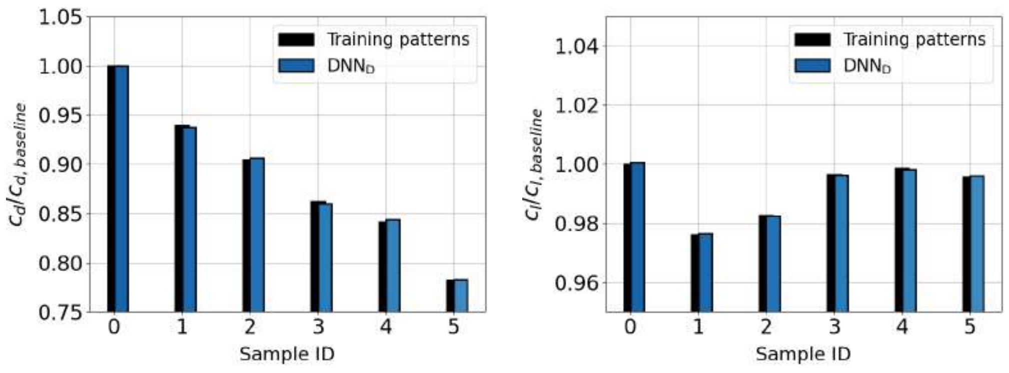


Figure 5.15: *The isolated airfoil case: The c_d (left) and c_l (right) of the airfoils in the $DB_{adjoint}$, computed with CFD and DNNs predictions. The coefficient values are normalized with those of the baseline geometry.*

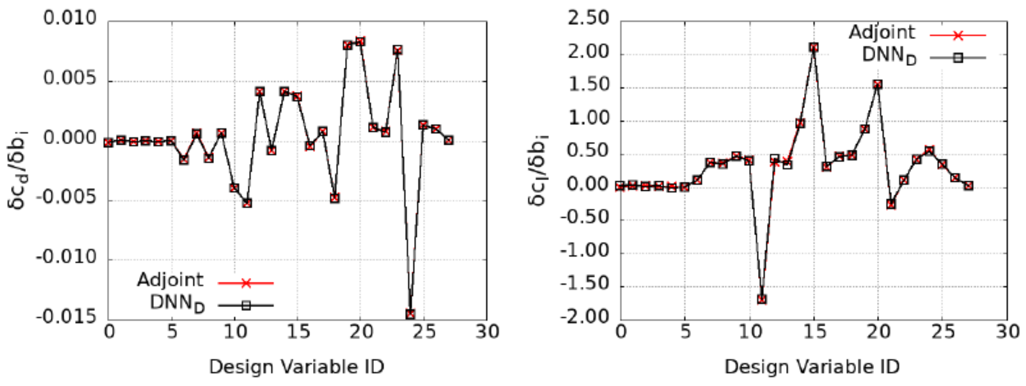


Figure 5.16: *The isolated airfoil case: The c_d SDs (left) and c_l SDs (right) of the sixth point of the adjoint-driven optimization computed with adjoint and DNN_D predictions.*

5.4 Airfoil ShpO

Each of the DNNs of the previous Section are utilized to drive a separate optimization process of the airfoil. The networks trained on the DB_{LHS} dataset initiate the optimization from the baseline airfoil, while the DNN_D, trained on the initial adjoint-driven optimization solutions, begins from the best solution it has seen, which is the sixth sample. Each DNN-driven optimization relies only on the corresponding DNN's predictions. Aim of the optimization is to minimize the airfoil's c_d and keep the c_l close to the value of the baseline airfoil. The objective is:

$$F = 0.1c_d + 0.1(c_l - 0.488)^2 \tag{5.1}$$

where $c_{\text{l,target}} = c_{\text{l,baseline}}$ =4.88×10^{-1}

Each optimization cycle requires 1 TU for the primal solution and 2 TUs for the adjoint one. The adjoint-driven optimization, whose convergence is shown in Figure 5.17 converges in 43 TUs. The optimized solution of the DNN-driven process is re-evaluated on the CFD tool, to verify its quality. If further improvement is desired, the objective and its SDs of the optimized airfoil can be added to the database. The networks are retrained and the optimization process initiates from the latest solution, using the re-trained models. Two re-trainings for the DNNs trained on DB_{LHS} are necessary to acquire solutions of the desired quality. Herein, the optimization turnaround time amounts to 67 TUs. Among them, the DNN_B-driven optimization yields a better solution compared to the adjoint-driven one. In contrast, the DNN_D-driven optimization has a total cost of 22 TUs, making it by 49% faster than the adjoint-driven process and it results in an even better solution. The solutions are presented in Figure 5.18 and Table 5.8.

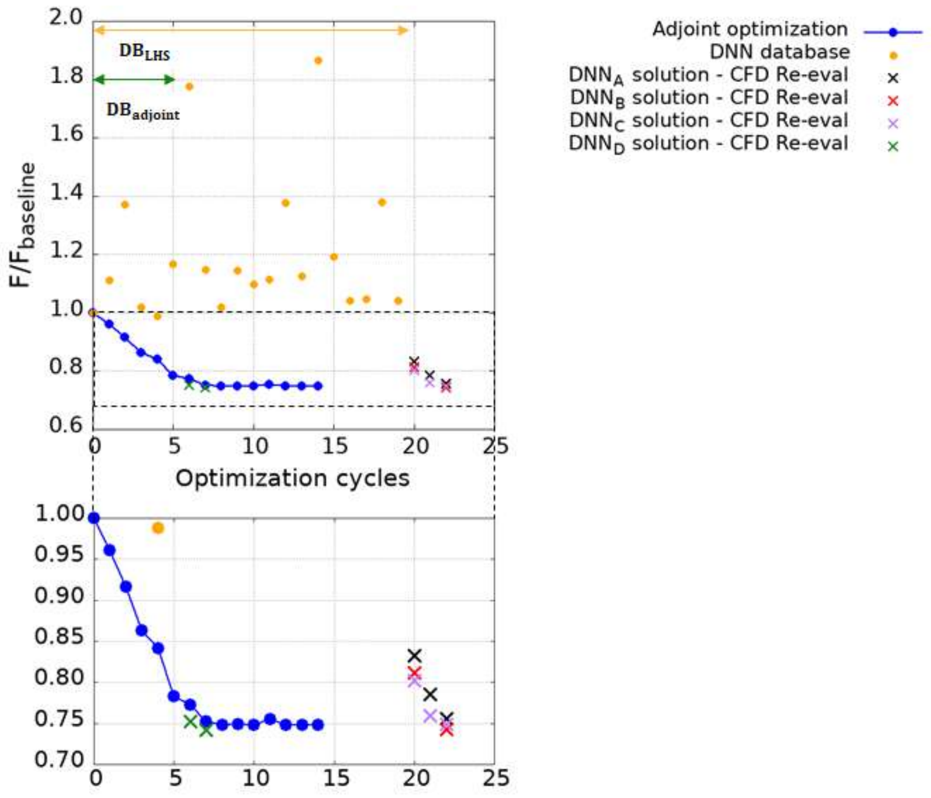


Figure 5.17: *The isolated airfoil case: Convergence of the adjoint-driven optimization and the solutions obtained with each DNN after they are re-evaluated on the CFD code. The cost of constructing the corresponding DNN database has been included.*

Comparison of the Optimized Solutions					
	$\frac{F}{F_{\text{baseline}}}$ (%)	c_d	c_l	c_d **Reduction** (%)	Δc_l (%)
Baseline airfoil	-	6.22×10^{-3}	4.88×10^{-1}	-	-
Adjoint solution	74.78	4.64×10^{-3}	4.86×10^{-1}	25.40	-0.20
DNN_A **solution**	75.61	4.69×10^{-3}	4.86×10^{-1}	24.60	-0.20
DNN_B **solution**	74.26	4.61×10^{-3}	4.86×10^{-1}	25.88	-0.20
DNN_C **solution**	74.88	4.64×10^{-3}	4.85×10^{-1}	25.40	-0.30
DNN_D **solution**	74.19	4.59×10^{-3}	4.84×10^{-1}	26.21	-0.40

Table 5.8: *The isolated airfoil case: Comparison of the optimized solutions.*

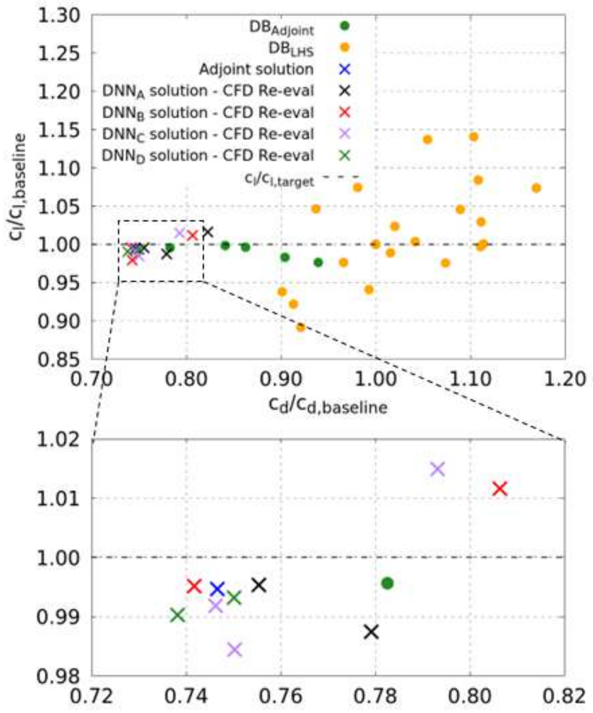


Figure 5.18: *The isolated airfoil case: A comparison is presented between the samples of the DB_{LHS}, $DB_{adjoint}$, and the optimized solutions obtained from the adjoint-driven and DNN-driven optimization processes. All DNN-optimized solutions are re-evaluated on CFD.*

In Figure 5.19, the airfoils of the DB_{LHS} and $DB_{adjoint}$ are presented. The airfoils generated with LHS cover a wider range of potential geometries, while the airfoils of the $DB_{adjoint}$ are closer to the optimized. The shape of the optimized geometries is compared in Figure 5.20 with the baseline airfoil. The shape of the optimized airfoils seem identical. The Mach number and turbulent viscocity fields are presented in Figures 5.21 & 5.22, respectively. In Figure 5.23, the pressure and skin friction coefficient distributions of the optimized airfoils are compared to the baseline. The optimized geometries exhibit an extended laminar region along the suction side, which contributes significantly to drag reduction. To preserve lift, a curvature near the leading edge is formed, that as shown in the c_p plots enhances the pressure difference between the suction and pressure sides.

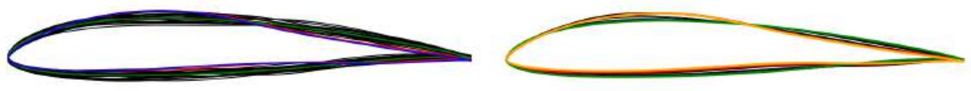


Figure 5.19: *The isolated airfoil case: Left) Geometries generated with LHS (black) are compared to the optimized airfoils obtained with adjoint (red) and DNN_B (blue). Right) The first solutions of the adjoint-driven optimization (black) along with the DNN_D optimized solution (orange) and the baseline airfoil (green).*

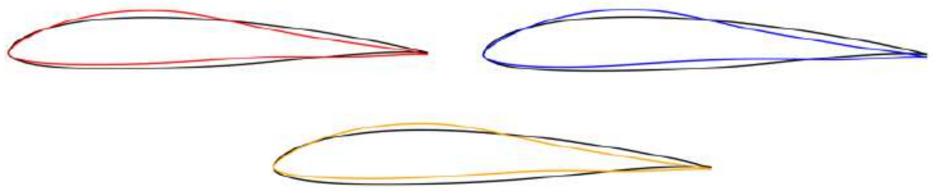


Figure 5.20: *The isolated airfoil case: Comparison of the optimized solution obtained with adjoint (red), DNN_B (blue) and DNN_D (orange) wih the baseline airfoil (black).*

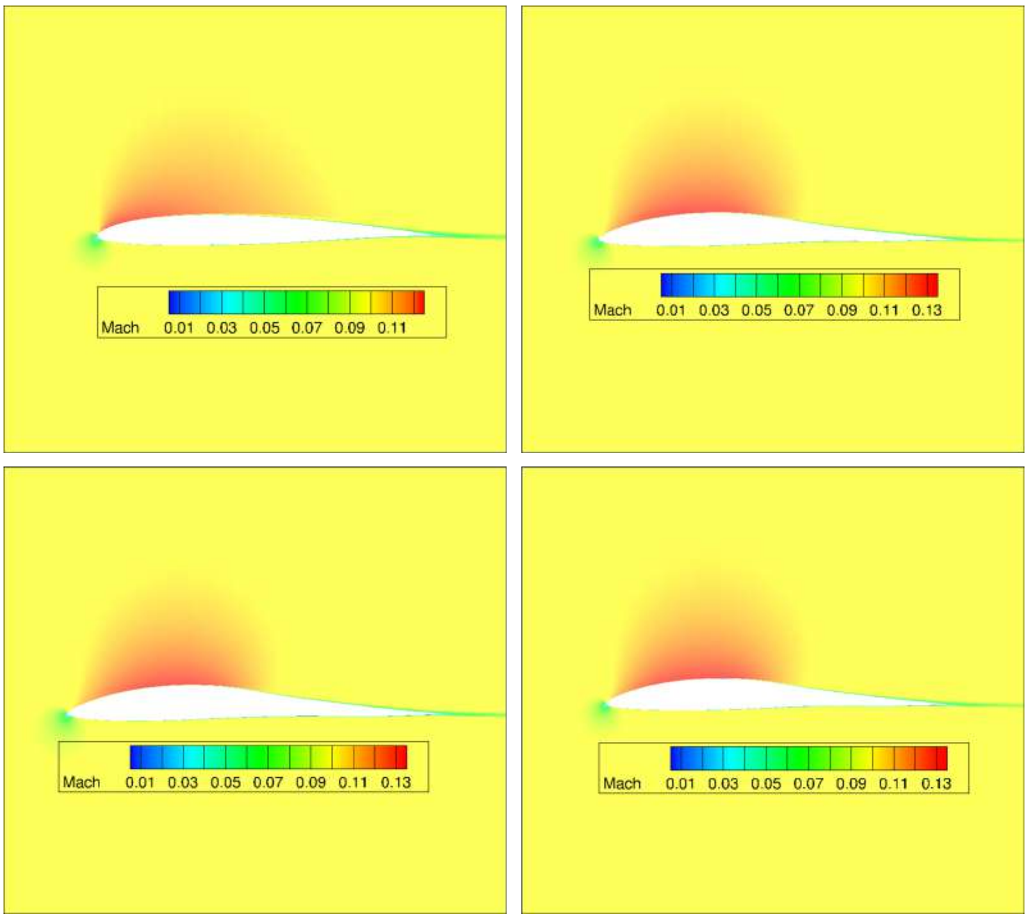


Figure 5.21: *The isolated airfoil case: The Mach number fields for the baseline airfoil (top left), and the 3 airfoils optimized by adjoint (top right), DNN_B (bottom left), and DNN_D (bottom right).*

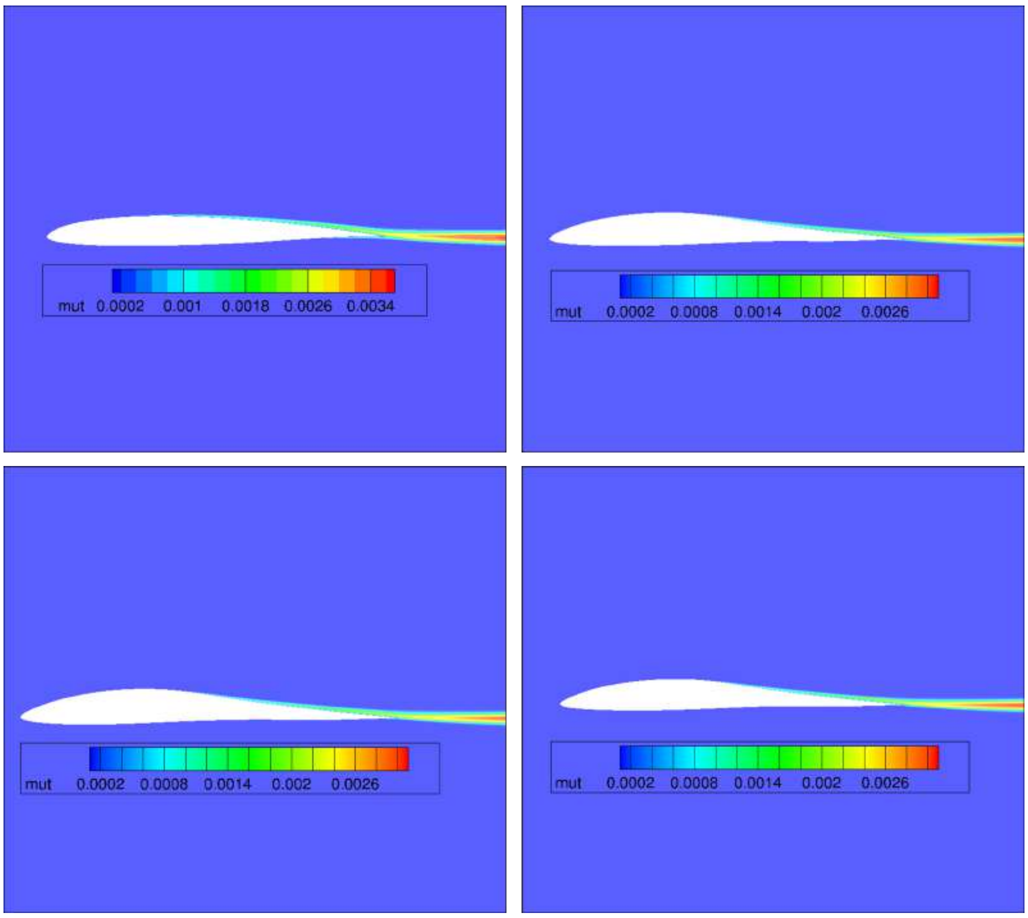


Figure 5.22: *The isolated airfoil case: The turbulent viscosity fields for the baseline airfoil (top left), and the 3 airfoils optimized by adjoint (top right), DNN_B (bottom left), and DNN_D (bottom right).*

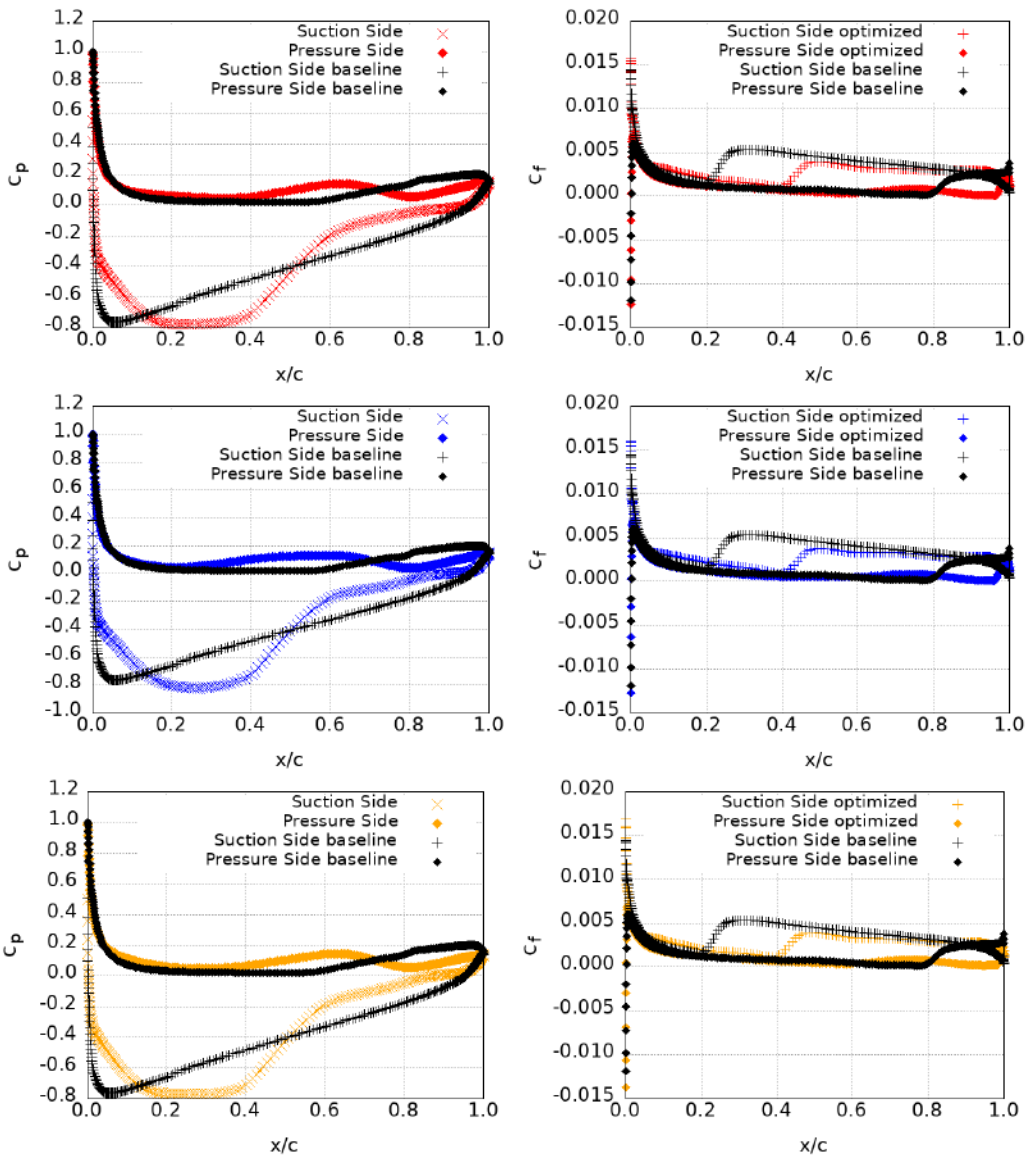


Figure 5.23: *The isolated airfoil case: Left) The pressure coefficient distribution for the baseline airfoil and the optimized ones resulting from adjoint (top), DNN_B (middle), and DNN_D (bottom). Right) The skin friction coefficient distribution for the baseline airfoil and the adjoint-optimized (top), DNN_B-optimized (middle), and DNN_D-optimized (bottom) airfoils.*

Chapter 6

Single-Phase Turbulent Flow around a Compressor Blade-Airfoil

6.1 Introduction

In this Chapter, the aerodynamic ShpO of a 2D low-speed compressor cascade is performed. Aim of the optimization is to minimize the mass-averaged p_t losses (Δp_t) of the cascade, while maintaining the exit flow angle (a_{exit}) close to its original value. The proposed method employs a hybrid adjoint- and DNN-driven optimization approach. The ShpO of the airfoil during the initial cycles is driven by adjoint. The solutions obtained from this initial phase are used to train the Hermite-DNNs. The DNN-driven optimization is then initiated from an intermediate solution of the adjoint-based process, and from that point relies exclusively on the DNNs predictions. This approach is compared to the standard adjoint-driven optimization in terms of cost.

6.2 Flow conditions and parameterization

The flow around the airfoil is characterized as low-speed and turbulent. A hybrid mesh, consisting of approximately 36K nodes, is used. The mesh, which combines structured and unstructured regions, is shown in Figure 6.1. The flow conditions are summarized in Table 6.1.
The shape of the airfoil and the mesh are controlled by a 9×8 NURBS lattice, equidistant in each direction, shown in Figure 6.2. The red points may move in the chord-wise and pitch-wise direction, within a bound 10% of their distance from adjacent control points in each direction, to ensure smooth deformations in the airfoil's shape. The black points remain fixed during the optimization. This results

in 40 design variables ($\vec{b} \in \mathsf{R}^{40}$).

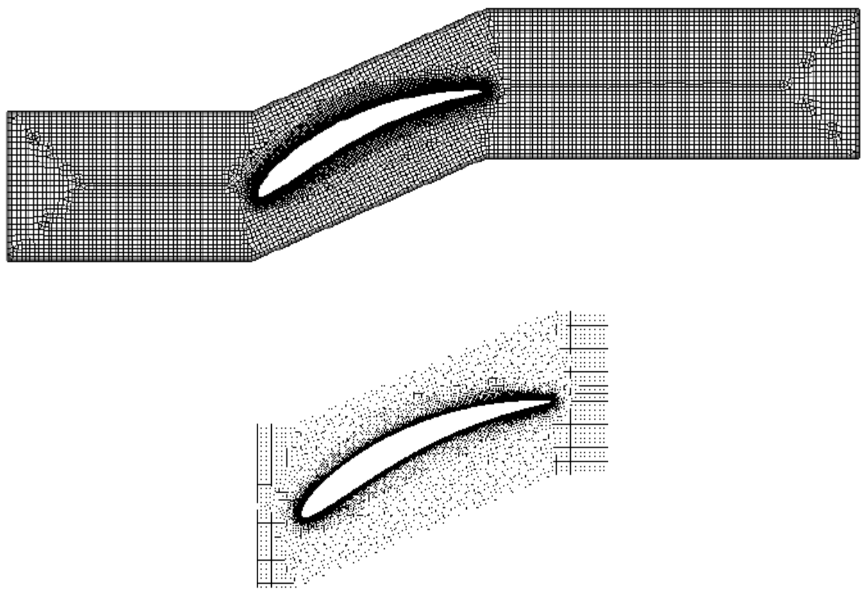


Figure 6.1: *The compressor blade-airfoil case: Computational mesh of the whole domain (top) and the region around the solid boundaries (bottom).*

Flow Conditions	
Inlet total temperature (K)	288
Inlet total pressure (bar)	1.15
Inlet flow angle (°)	44

Table 6.1: *The compressor blade-airfoil case: Flow conditions.*

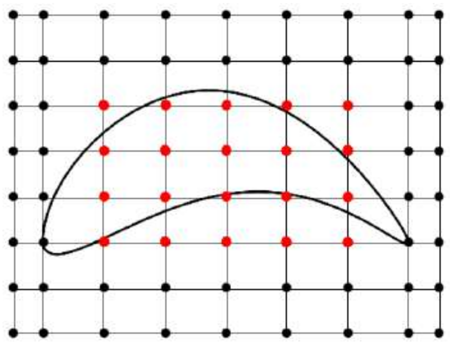


Figure 6.2: *The compressor blade-airfoil case: Blade-airfoil parameterization.*

6.3 DNN Configuration and Training

The database used to train the networks ($DB_{adjoint}$) contains the first five solutions of the adjoint-driven ShpO, shown in Figure 6.8. The objective function to be minimized in the ShpO of the airfoil regards the Δp_t of the cascade, and contains an additional term that measures the deviation of the a_{exit} value from the target one. Each term is analyzed in Section 6.4 and will be provided by a different network. Due to the limited number of samples, all of them are used for training. As a result, input to each branch of the DNNs is the $[5 \times 40]$ tensor containing the coordinates of the active CPs for each sample, and output the $[5 \times 1]$ tensor with the predictions on the Δp_t or a$_{exit}$ values. After differentiation, a $[5 \times 40]$ tensor is computed, containing the SDs predictions for each pattern.

First DNN configuration

The configuration of the DNNs (DNN_A) utilized in this work is presented in Tables 6.2 & 6.3. It is determined by an EA-based optimization, aiming to reduce the $\mathcal{L}_{\text{Hermite}}$ computed using the entire database. This process is based on the second setup described in Chapter 4. The training loss of Δp_t and a_{exit} DNN_A, including both the function and the derivative terms for each network, is illustrated in Figure 6.3.

DNN Δp_t				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	7	4096-512-128-64-2048-5-1	Gelu	Gelu
B_{grad}	9	512-256-64-128-32-256-64-5-40	Gelu	Sigmoid
	Optimizer	**Loss function**	**Batch Size**	
	Adam (lr=0.0027)	MSE	5	

Table 6.2: *The compressor blade-airfoil case: Optimized architecture of* Δp_t DNN_A.

DNN a_{exit}				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	9	128-256-32-256-64-256-32-5-1	Tanh	Gelu
B_{grad}	8	2048-2048-512-32-1024-32-5-40	Tanh	Tanh
	Optimizer	**Loss function**	**Batch Size**	
	Adam (lr=0.0007)	MSE	5	

Table 6.3: *The compressor blade-airfoil case: Optimized architecture of* a$_{exit}$ DNN_A.

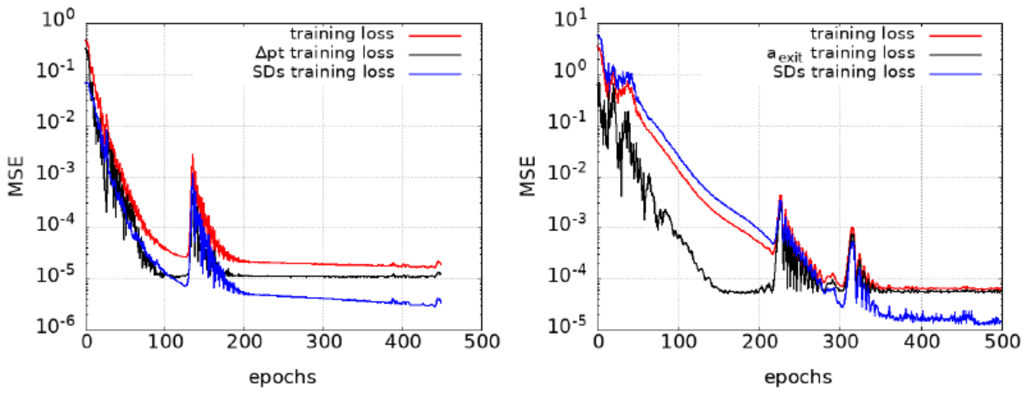
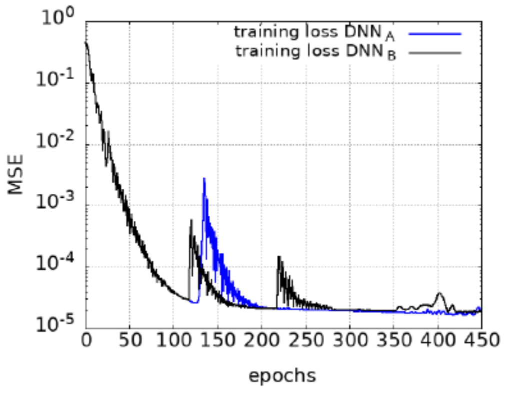


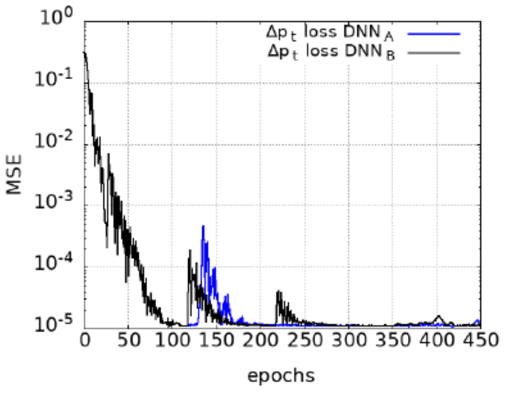
Figure 6.3: *The compressor blade-airfoil case: Training loss convergence of* Δp_t *(left) and* a_{exit} *(right)* DNN_A. *The convergence of each term that make up the total loss is presented separately.*

Pruning

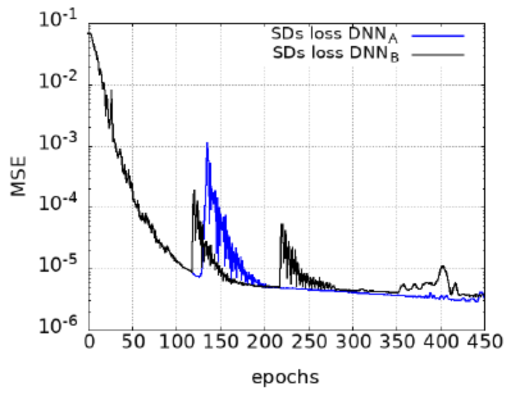
In this Section, the impact of pruning on the training convergence of DNN_A is examined. From now on, the pruned networks will be denoted as DNN_B. The pruning criterion employed, corresponds to the second case analyzed in Chapter 4. The pruning cycles are repeated with a frequency of 100 epochs, 4 times for Δp_t DNN_B and 3 times for a_{exit} DNN_B. The resulting training loss curves are compared to those of the dense network (DNN_A) in Figure 6.4. An increase in loss is observed during the initial pruning epochs, due to the sudden reduction in model's parameters, however, both models recover during subsequent training. The Δp_t DNN_B requires 22% less memory for storage (7.5MB), while the a_{exit} DNN_B achieves a 14% reduction, requiring 17MB. Their predictions on the Δp_t and a_{exit} values are presented in Figure 6.5, normalized by the corresponding values of the baseline airfoil. The SDs predictions for the fifth point of the adjoint-driven optimization are compared in Figure 6.6 with the adjoint reference values. The sparsity of each layer is presented in Figure 6.7.



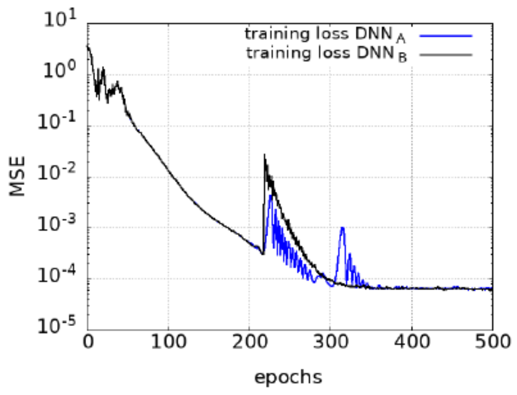
(a) Δp_t DNNs training loss



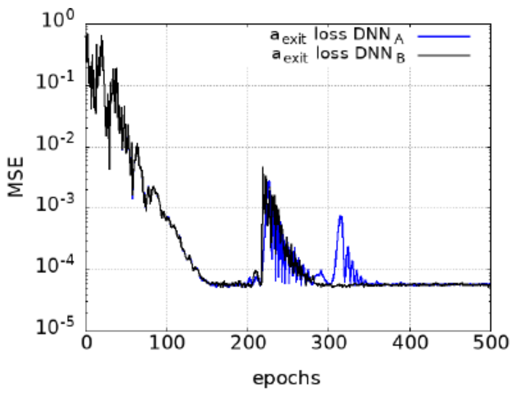
(b) Δp_t DNNs training loss function



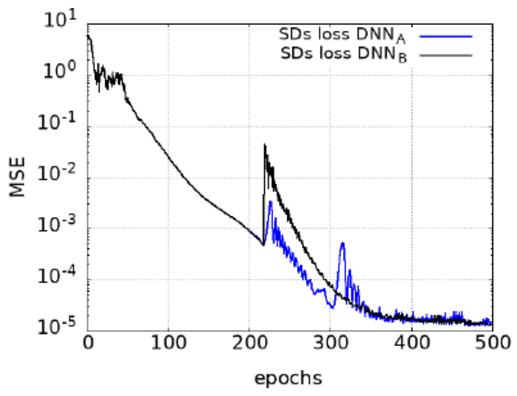
(c) Δp_t DNNs training loss SDs



(d) a_{exit} DNNs training loss



(e) a_{exit} DNNs training loss function



(f) a_{exit} DNNs training loss SDs

Figure 6.4: *The compressor blade-airfoil case: Training loss of Δp_t and a_{exit} DNN_C, compared to the convergence of the dense DNN_A.*

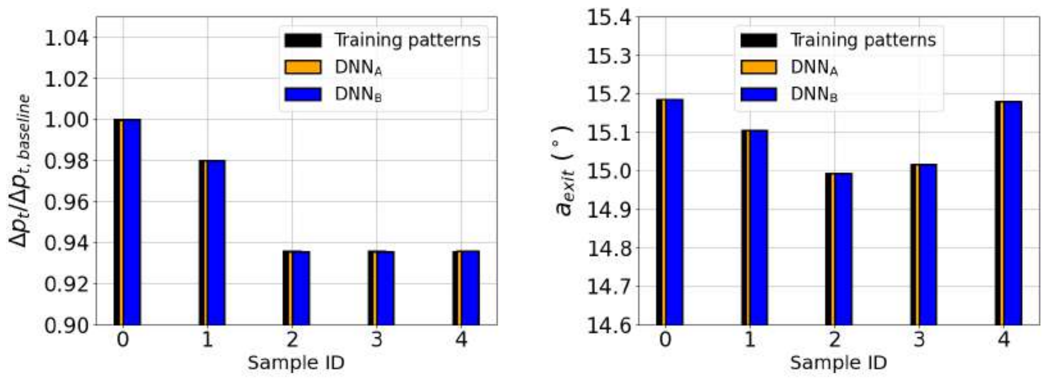


Figure 6.5: *The compressor blade-airfoil case: The training patterns computed with CFD and DNNs predictions. The Δp_t values are normalized with the Δp_t of the baseline geometry. The training database consists of all the patterns.*

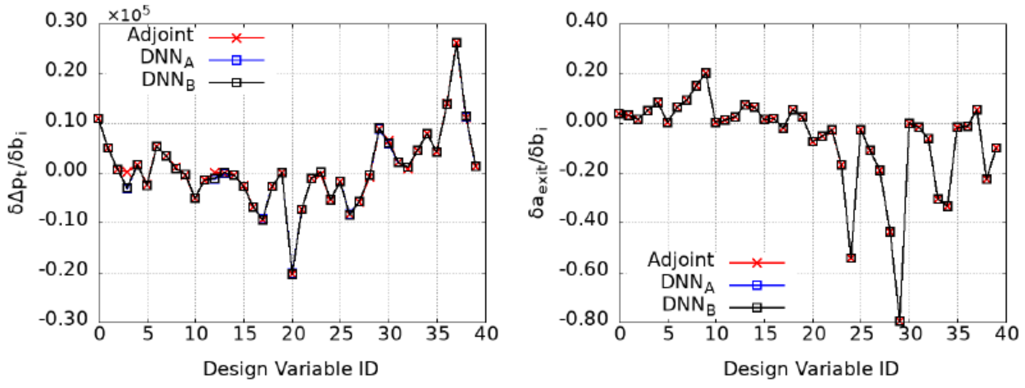


Figure 6.6: *The compressor blade-airfoil case: The Δp_t SDs (left) and a_{exit} SDs (right) of the fifth point of the adjoint-driven optimization computed with adjoint and DNNs predictions.*

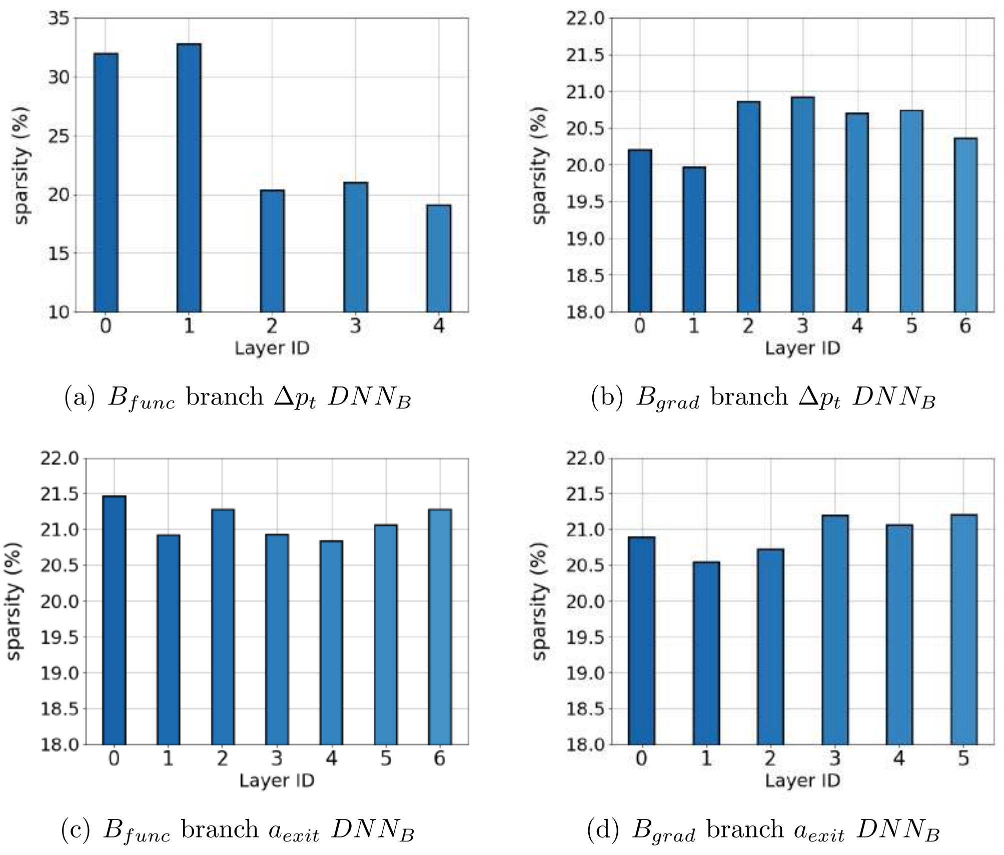


Figure 6.7: *The turbine blade-airfoil case: Sparsity for each branch of the Δp_t (top) and a_{exit} (bottom) DNN_B.*

6.4 ShpO of the compressor blade-airfoil

The Hermite-trained DNNs are employed to drive the optimization process of the blade-airfoil, initiating from the fifth candidate solution of the adjoint-driven optimization. The objective is:

$$F = 10^{-7}\Delta p_t + (a - 0.265)^2 \tag{6.1}$$

where $a_{\text{target}} = a_{\text{baseline}} = 0.265rad$.
The adjoint-driven optimization, whose convergence is shown in Figure 6.8, requires 34 TUs to converge, consisting in each cycle of 1 TU for evaluating each blade and 2TUs for computing the Δp_t and a_{exit} SDs. The DNN-driven optimizations achieve a solution similar to the adjoint-driven one at a total cost of 16 TUs: 15 TUs to construct their database and 1TU for the CFD-evaluation of their optimized solution, making it by 53% faster. The optimized solutions are depicted in the Δp_t, a_{exit} space in Figure 6.9.
Figure 6.10 compares the objective values of the airfoils included in the $DB_{adjoint}$ with those of 20 airfoils generated using LHS. The latter are created by sampling the design variable space to configure 20 distinct combinations of the CPs position, each one forming a different airfoil. Despite the larger database size, the LHS-generated database does not outperform the adjoint-driven approach. In fact, only ~20% of the sampled blades result in objective values better than the baseline. The optimized solutions are summarized in Table 6.4. The shape of the optimized airfoils is compared in Figure 6.11 with the baseline airfoil. The Mach number fields are shown in Figure 6.12.

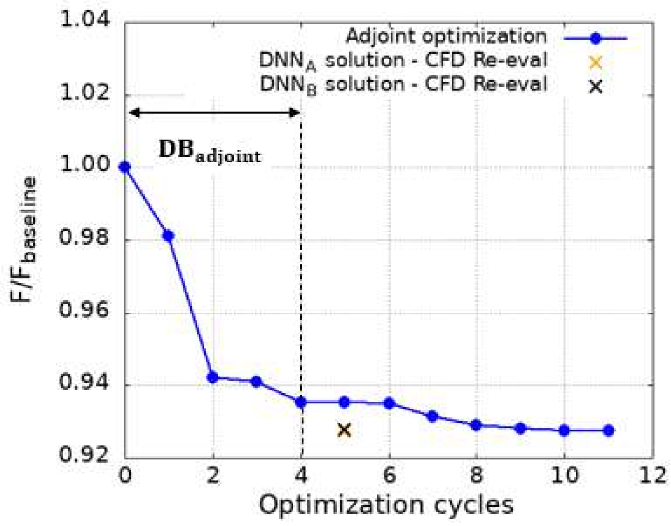


Figure 6.8: *The compressor blade-airfoil case: Convergence of the adjoint-driven optimization and the DNN-driven optimized solutions, after they are re-evaluated on the CFD tool.*

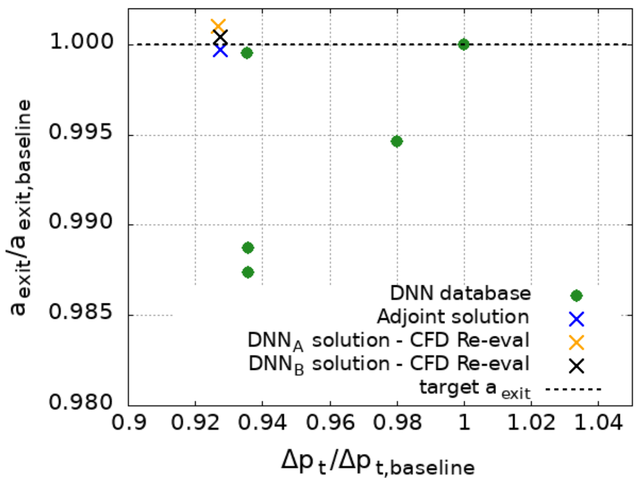


Figure 6.9: *The compressor blade-airfoil case: Solutions of the DNN-driven optimizations after they are re-evaluated on the CFD code, and adjoint optimal solution. Δp_t, a_{exit} values are normalized with those of the baseline airfoil respectively.*

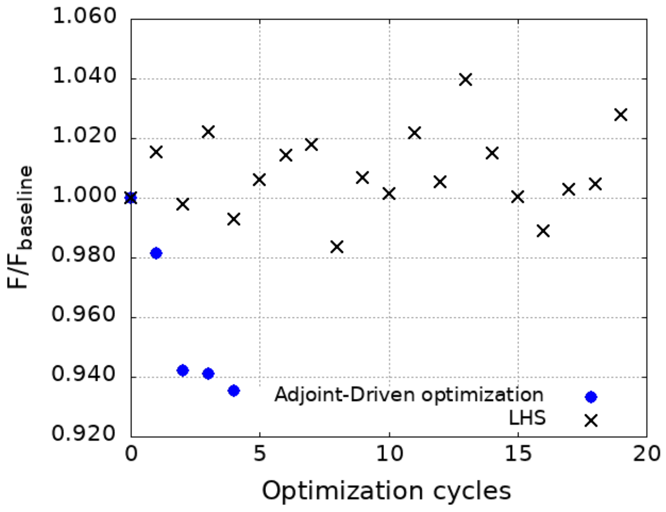


Figure 6.10: *The compressor blade-airfoil case: Initial solutions of the adjoint-driven optimization and samples generated by LHS.*

Comparison of the Optimized Solutions					
	$\frac{F}{F_{\text{baseline}}}$ (%)	Δp_t (Pa)	a_{exit} (°)	Δp_t Reduction (%)	Δa_{exit} (°)
Baseline blade	-	1.721×10^4	15.186	-	-
Adjoint solution	92.75	1.597×10^4	15.181	7.25	-0.005
DNN_A **solution**	92.71	1.595×10^4	15.200	7.30	0.014
DNN_B **solution**	92.77	1.597×10^4	15.190	7.23	0.004

Table 6.4: *The compressor blade-airfoil case: Comparison of optimized solutions.*

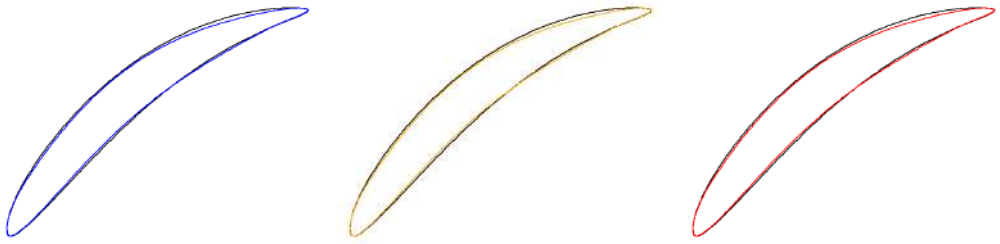


Figure 6.11: *The compressor blade-airfoil case: The baseline (black) is compared to the optimized solutions obtained using adjoint (blue), DNN_A (orange), and DNN_B (red).*

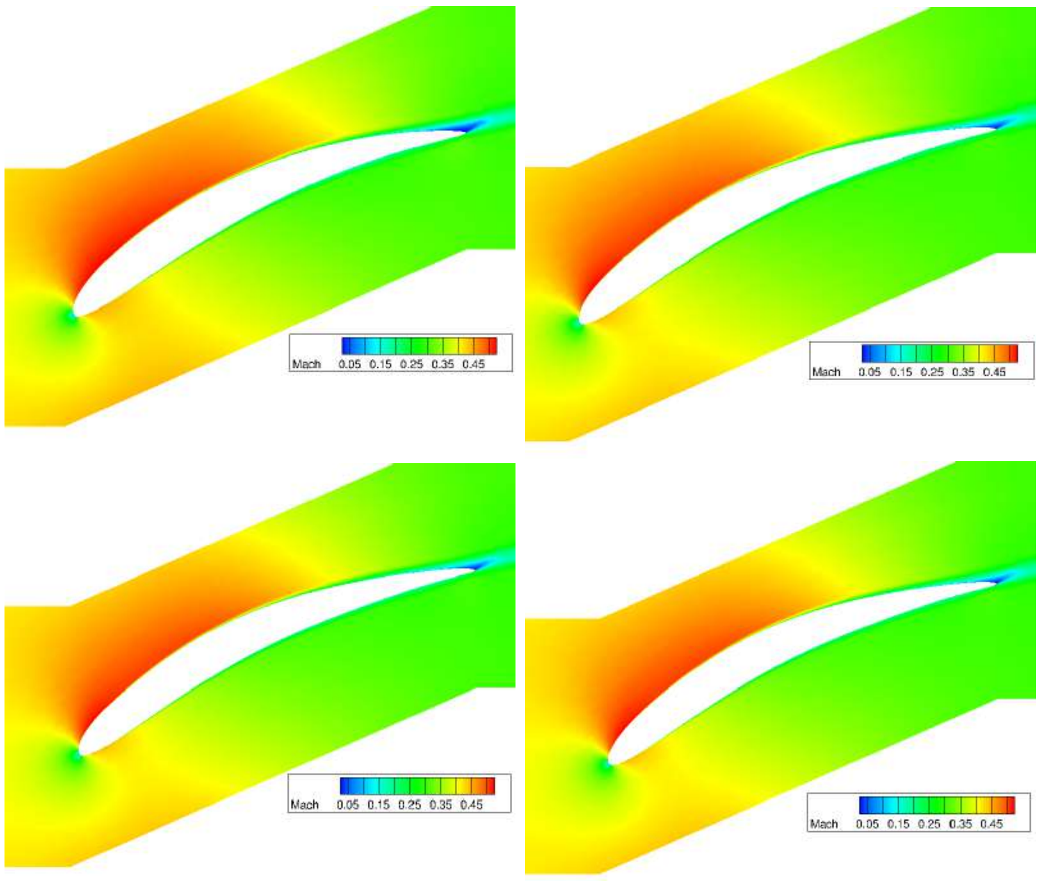


Figure 6.12: *The compressor blade-airfoil case: The Mach number fields for the baseline geometry (top left) and the optimized geometries resulting from adjoint (top right), DNN_A (bottom left) and DNN_B (bottom right).*

Chapter 7

Two-phase Flow around a Hemispherical-Cylinder Body

7.1 Introduction

In this Chapter, the hydrodynamic Shape Optimization of a hemispherical-cylinder body is conducted. Aim of the optimization is the minimization of the cavitation, while maintaining the drag below a certain threshold. This study is of high interest for several navy applications and machines operating underwater, because cavitation affects their performance and might lead to material erosion, performance deterioration and noise.

A two-phase primal flow solver is utilized to capture the generation of vapor bubbles and their influence on the flow around the body. Its adjoint variant is used to compute the SDs, while accounting for the presence of vapor in the water flow. The proposed method employs a hybrid adjoint and DNN-driven optimization approach. The ShpO of the body during the initial optimization cycles is driven by adjoint. The solutions obtained from this initial phase are used to train the Hermite-DNNs, which, as in the previous cases, are trained to predict both the target objective values and their SDs. The DNN-driven optimization is then initiated from an intermediate solution of the adjoint-based process, and from that point relies exclusively on the DNNs predictions.

7.2 Cavitation

Cavitation is a phenomenon in fluid mechanics that occurs when the static pressure of a liquid drops below its vapor pressure, leading to the formation of small vapor-filled cavities within the fluid. ([33]). The cavitation patterns include the three following types; transient isolated bubbles, formed in low-pressure regions and transported by the main flow, attached cavities that develop along the low-pressure surfaces of blades and foils, and vortex cavities, that are formulated within the low-pressure cores of vortices in turbulent wakes. Attached cavities are a common type of cavitation in pumps, that form mainly at the blade edge of the impeller inlet, and hydraulic turbines. The presence of vapor is highly impactful for the flow, causing flow acceleration around the bubble, recirculation regions and alternated forces acting on the body. All these vapor structures are unstable, and when subjected to high-pressure they collapse generating shock waves. These shock waves can produce erosion on the solid surfaces, worsen system's performance, noise and mechanical vibrations.

7.3 Flow Conditions and Parameterization

The geometry consists of a hemispherical head attached to a cylindrical body. The farfield velocity is aligned with the axis of symmetry. In order to scale the cavitation phenomena under different flow conditions, a dimensionless parameter known as cavitation number is introduced:

$$\sigma = \frac{p - p_v}{\frac{1}{2}\rho U^2} \tag{7.1}$$

p stands for an ambient pressure, and p_v for the vaporization pressure at the ambient temperature. The cavitation number describes the probability of a cavitating flow to accur. A small σ value indicates a higher likelihood of cavitation occurring. The flow inlet conditions are presented in Table 7.1.

Flow Conditions	
U_∞ (m/s)	0.144
Re	$2 \cdot 10^5$
σ	0.4
p_v (Pa)	2300

Table 7.1: *The cylindrical hemisphere case: Flow conditions.*

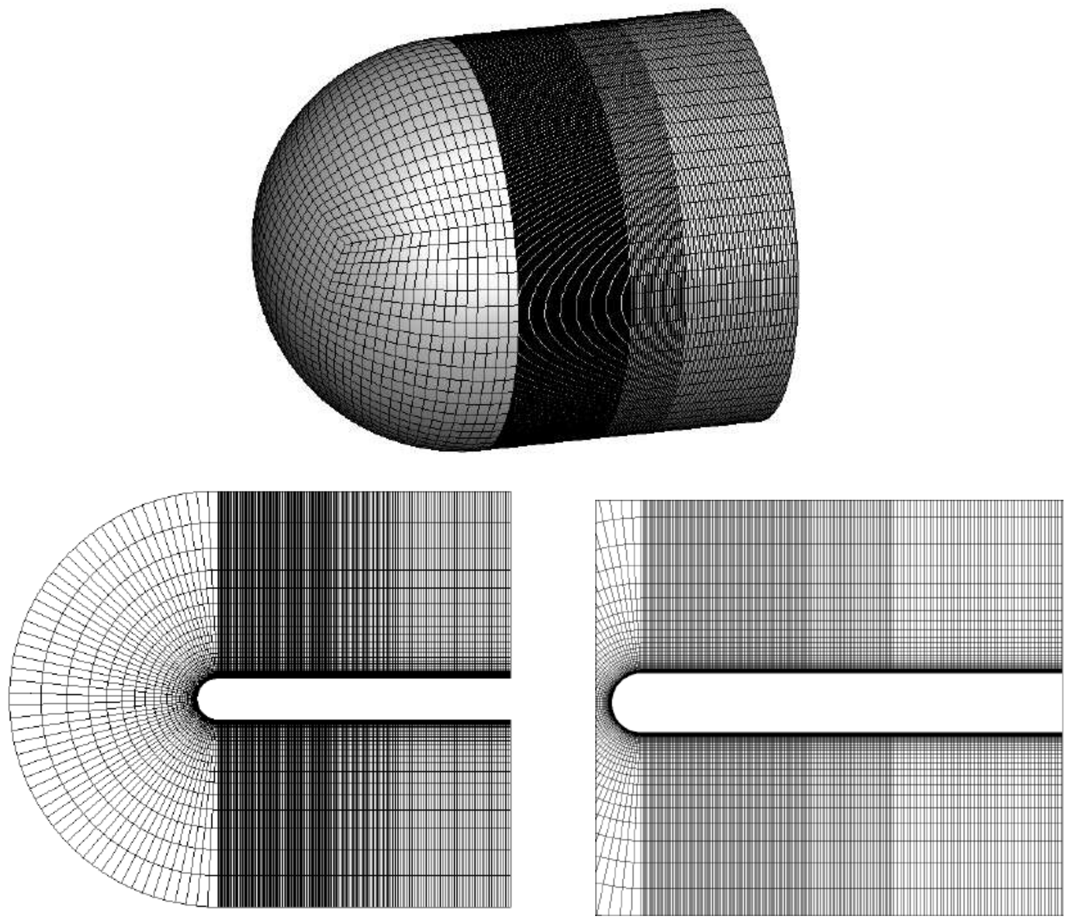


Figure 7.1: *The cylindrical-hemisphere case: Computational mesh of the whole domain (top), a meridional section (bottom left), and near the solid boundaries (bottom right).*

Only the hemispherical part of the body is parameterized. In order to preserve the symmetry of the generated body, the generatrix is parameterized using a Bezier curve with 10 control points. The revolution of the generatrix around the x axis produces the axisymetric 3D body. In Figure 7.2 only the red points may move, with a maximum displacement of 10% their initial values. The control points are allowed to move only vertically, affecting in this way, the local radius of the axisymetric body. This results to 8 design variables ($b \epsilon R^8$). A grid displacement technique is employed to adapt the mesh according to CP displacements. As in the previous cases, the IDW method is utilized.
The flow solver is based an a homogeneous two-phase approach (3.1). Each phase is distinguished through its volume fraction, which corresponds to the percentage of the phase's volume over a computational cell's volume. Turbulence is modeled using the Spalart-Allmaras turbulence model.

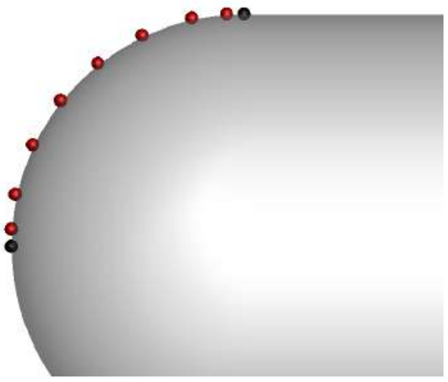


Figure 7.2: *The cylindrical-hemisphere case: The generatrix parameterization.*

As the flow accelerates along the surface of the hemisphere, a decrease in pressure is observed. When the local pressure falls below the vapor pressure of water, phase transition is initiated, leading to the development of a cavitation sheet, as illustrated in Figure 7.3. At some point downstream, the cavitation sheet detaches, as shown in Figure 7.4, forming a vapor cloud around the body. At the end of this region, the cavitation cloud collapses, and the pressure recovers. Due to the rapid recovery, a recirculation zone is also observed.

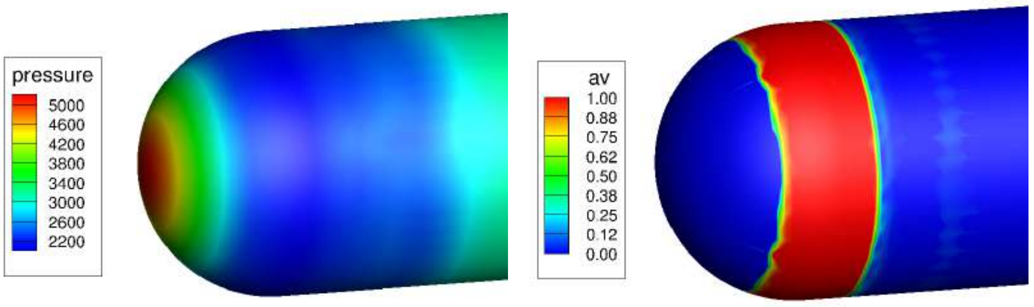


Figure 7.3: *The cylindrical-hemisphere case: Pressure (left) and a_v (right) contours on the solid wall.*

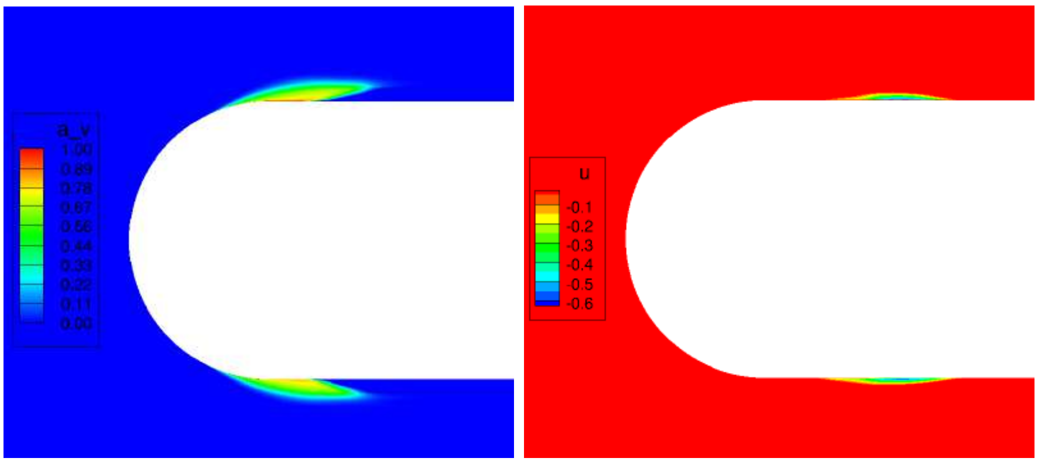


Figure 7.4: *The cylindrical-hemisphere case: The a_v (left) and negative x-velocity components (right) contours in a meridional section.*

7.4 DNN Configuration and Training

The database is constructed using the first eight solutions of the adjoint-driven optimization, along with their corresponding SDs. The objective to be minimized in the SphO of the body is formed by a cavitation and a drag term, that will be analyzed in the following Section. Since the drag and cavitation SDs differ significantly in scale, a separate network is constructed for each term. Input to branch of the model is the $[8 \times 8]$ tensor with the y-coordinates of the CPs. The data propagate through the hidden layers of each branch, whose outputs are combined to form a $[8 \times 1]$ tensor with the predictions on the outputs. During training, each model is differentiated to produce a $[8 \times 8]$ tensor containing the predictions on the target derivatives. Since the networks will be used for minimization of the objective, the minimum bound for normalizing their outputs is set by 20% smaller than the minimum value encountered in the database. Additionally, the minimum and maximum bounds used to normalize the inputs for each design variable are extended by approximately 2% beyond the respective min and max values observed for each variable.
Both the optimal DNN architecture and the training configuration are determined by an EA-based optimization process, conducted with EASY. Objective of the optimization is the minimization of $\mathcal{L}_{\text{Hermite}}$, evaluated over the entire database, after each network has been trained for a small number of epochs. For the EA-based optimization the second setup introduced in Chapter 4 is used. The optimized architectures are presented in Tables 7.2 & 7.3.

DNN cavitation				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	9	2048-2048-1024-256-64-64-256-8-1	Tanh	Tanh
B_{grad}	6	1024-512-256-64-8-8	Tanh	Sigmoid
	Optimizer	**Loss function**	**Batch Size**	
	Adamax (lr=0.005)	MSE	8	

Table 7.2: *The cylindrical-hemisphere case: Optimized architecture of cavitation DNN.*

DNN drag				
	Layers	**Neurons**	**Activation Function**	
			Hidden Layers	**Final Layer**
B_{func}	6	512-2048-64-32-8-1	GELU	Tanh
B_{grad}	7	32-1024-2048-512-512-8-8	GELU	Sigmoid
	Optimizer	**Loss function**	**Batch Size**	
	AdamW (lr=0.005)	MSE	8	

Table 7.3: *The cylindrical-hemisphere case: Optimized architecture of drag DNN.*

The training loss convergence of each DNN, along with the convergence of each term that make up the total loss, are shown in Figure 7.5. The SDs loss is composed of the sum of losses corresponding to each design variable. The DNNs predictions are illustrated in Figures 7.6 & 7.7. It can be observed that the drag DNN minimizes the loss, by minimizing mainly the SDs prediction error. Herein the drag predictions exhibit slight deviations from the target values. However, the network can still be incorporated in the optimization process, as the constraint imposed on drag is not strict, and minor discrepancies in drag predictions are acceptable.

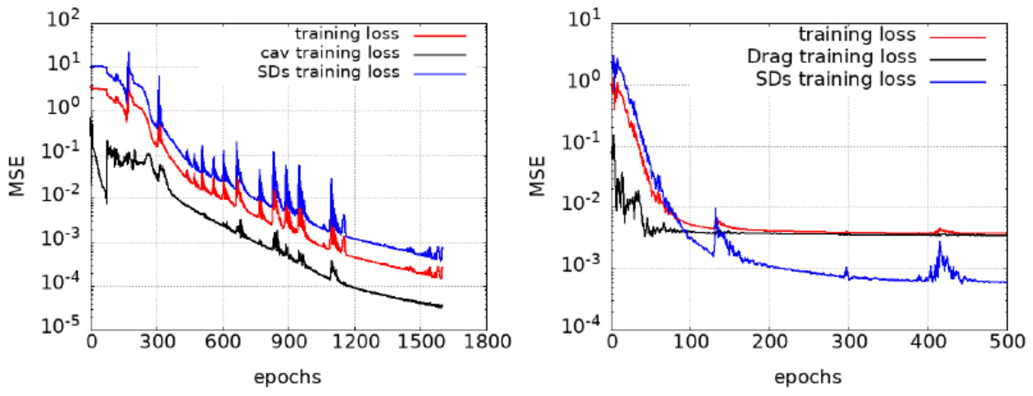


Figure 7.5: *The cylindrical-hemisphere case: Training loss convergence of cavitation and drag DNNs. Total loss is the weighted sum of the function loss and the SDs loss.*

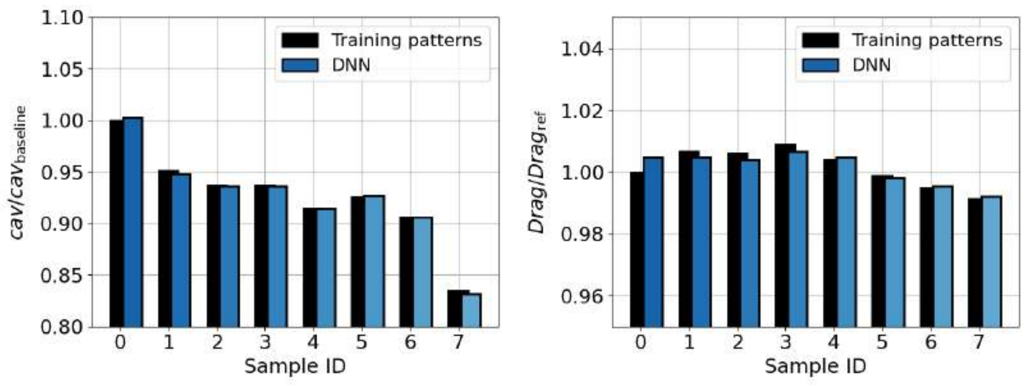


Figure 7.6: *The cylindrical-hemisphere case: The cavitation (left) and drag (right) training patterns computed with CFD and the DNNs predictions. The drag predictions are normalized with the reference drag value introduced in Section 7.5.*

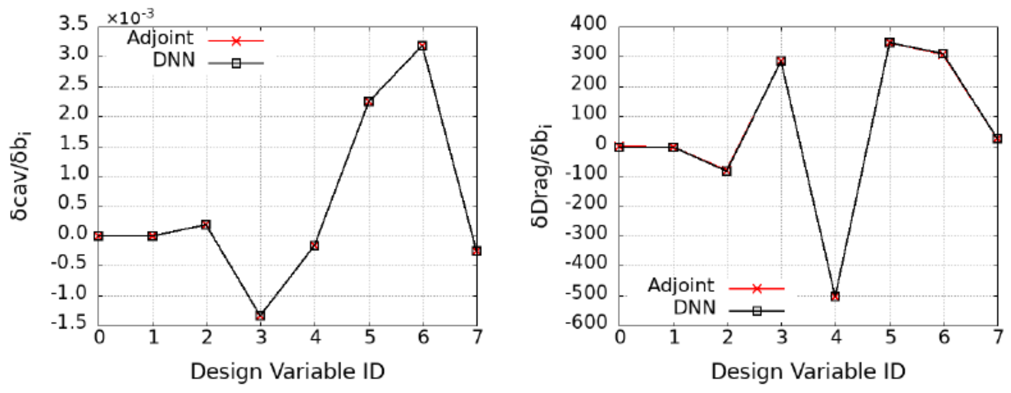


Figure 7.7: *The cylindrical-hemisphere case: The cavitation (left) and drag (right) SDs for the eighth point of the adjoint-driven optimization computed with CFD and DNN predictions.*

7.5 ShpO of the hemispherical-cylinder body

Objective of the optimization is the minimization of the vapor volume fraction (a_v). Additionally, the drag should not exceed a reference value $D_{ref} = 25.86$N, equal to the baseline's drag. The objective is presented in Eq. 7.2.

$$F = \frac{1}{2}\int_{\Omega} a_v^2 dV + \underbrace{\frac{10^{-5}D}{1+e^{-100(D-25.86)}}}_{t_2} \tag{7.2}$$

The second term of the objective function (t_2) is plotted in Figure 7.8. This term adds a penalty if the drag value exceeds the reference one.

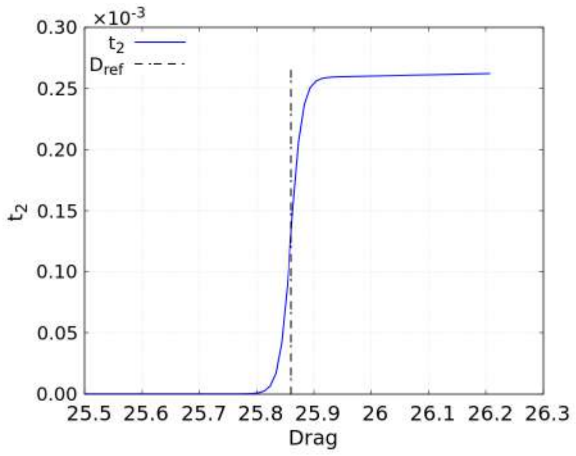


Figure 7.8: *The cylindrical-hemisphere case: Visualization of the second term in the objective function.*

The DNNs are employed to continue the optimization process from the best solution included in their database, hence the eighth bullet in Figure 7.9. The geometry obtained from the DNN-driven optimization is then re-evaluated using the CFD solver. The re-evaluation cycles described in the previous cases, are repeated three times, resulting in a total cost of 34 TUs. The adjoint-driven optimization requires 28 TUs. However, a comparison between the solutions obtained with each process in Figure 7.10 reveals that the DNN-driven process, despite the slightly higher computational cost, yields a better solution. This outcome demonstrates the DNNs ability to extrapolate beyond their training region, therefore verifying the effectiveness of this method.

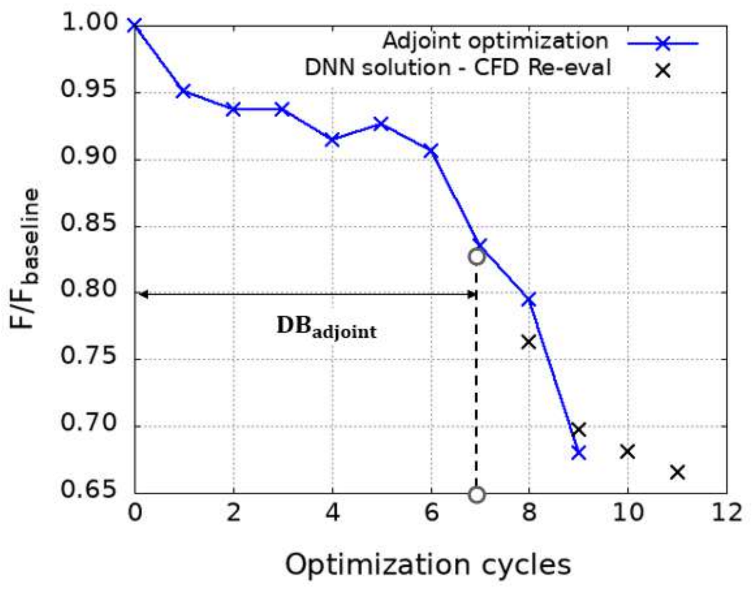


Figure 7.9: *The cylindrical-hemisphere case: Convergence of the adjoint-driven optimization and the DNN solutions after they are re-evaluated on the CFD code.*

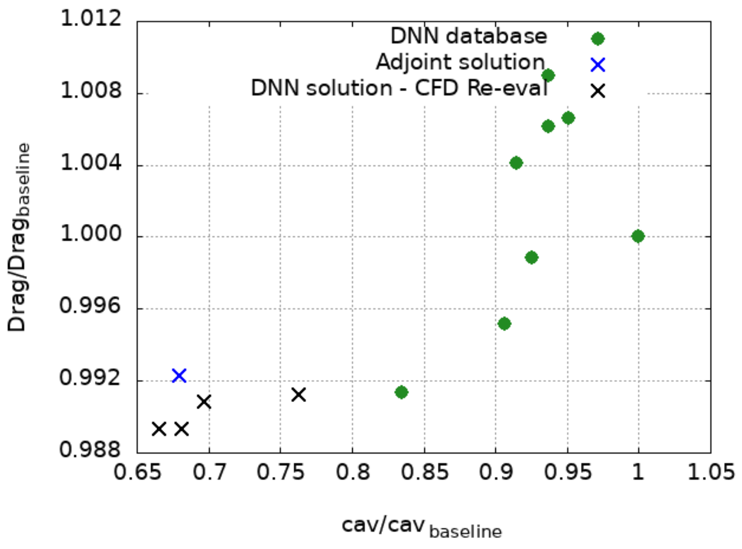


Figure 7.10: *The cylindrical-hemisphere case: A comparison is presented between the database's samples, and the optimized solutions obtained from the adjoint-driven and DNN-driven optimization processes. All DNN optimized solutions are re-evaluated on the CFD code.*

Comparison of the Optimized Solutions					
	$\frac{F}{F_{\text{baseline}}}$ (%)	**cavitation**	*Drag* (N)	**cavitation Reduction (%)**	D/D_{ref}
Baseline airfoil	-	1.18×10^{-5}	25.85	-	-
Adjoint solution	67.9	8.02×10^{-6}	25.66	32.1	0.99
DNN solution	65.7	7.75×10^{-6}	25.25	34.3	0.98

Table 7.4: *The cylindrical-hemisphere case: Comparison of optimized solutions.*

Figure 7.11 compares the a_v contours between the baseline and the DNN-optimized geometry. Vapor bubbles are observed near the inlet region in the optimized geometry, that collapse downstream. Figure 7.12 shows the CPs displacements in the optimized geometry. Figure 7.13 illustrates the local diameter's variation between the optimized and baseline geometries, and the pressure distribution on the optimized one. In the region where phase change initiates in the baseline, the optimized geometry exhibits a reduced curvature. This results in slower pressure reduction, and delays the formation of the cavitation sheet. Overall, the cavitation appears to be spatially confined and less intense in the optimized design.

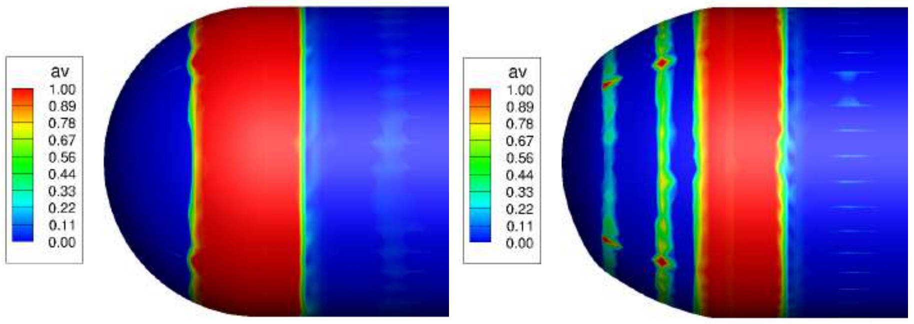


Figure 7.11: *The cylindrical-hemisphere case: The vapor volume fraction contours for the baseline geometry (left) and the DNN-optimized geometry (right).*

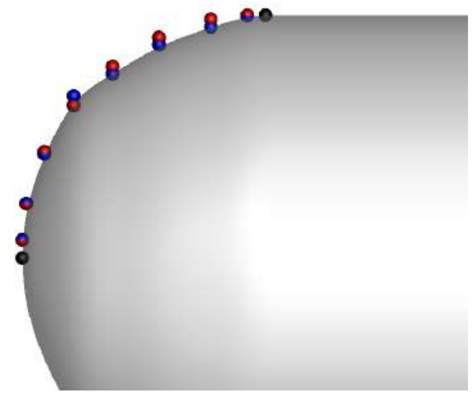


Figure 7.12: *The cylindrical-hemisphere case: The CPs positions for the baseline (red) and the DNN-optimized geometry (blue).*

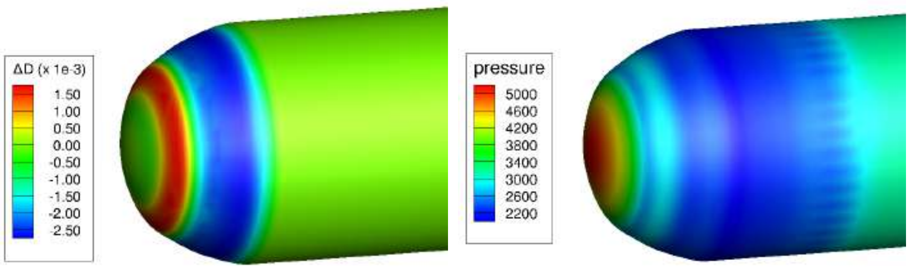


Figure 7.13: *The cylindrical-hemisphere case: The change in local diameter (left) and the pressure distribution on the solid boundaries for the DNN-optimized geometry (right).*

Chapter 8

Conclusion

8.1 Overview - Findings in the Examined Cases

This diploma thesis investigates the implementation of DNNs as data-driven, cost-efficient surrogates for both primal and adjoint evaluations in gradient-based ShpO in fluid dynamics. The DNNs are integrated into the optimization framework through two distinct approaches. In the first, the entire optimization process is driven exclusively by the DNNs. In the second, CFD evaluations in selected optimization cycles are replaced with calls to the low-cost DNN surrogates. In both strategies, the developed DNNs approximate not only the objective function values but also its SDs. The latter are computed via differentiation of the networks outputs with respect to their inputs. The architecture of the DNNs is inspired by the notion of the Hermite polynomials and extends their application to high-dimensional input spaces. The Hermite-DNNs are trained using a gradient-assisted approach, that incorporates the derivatives in the training process. All implementations concern applications of either single- or multi-phase flows.
In the first case the ShpO of a turbine blade-airfoil was carried out. Objective was the minimization of the mass-averaged p_t losses of the cascade, while maintaining the exit flow angle close to the value of the baseline blade. The flow was single-phase and turbulent. The shape of the blade was parameterized using a 6×3 NURBS lattice. The CPs were displaced in the pitch-wise and chord-wise direction, resulting in 32 design variables. In the first approach a database containing 30 blades were generated using LHS. All DNN configurations were optimized using the in-house Evolutionary-Algorithm software, EASY. Two DNN configuration optimizations were implemented; the first one concerned the network's architecture and the second one included also its training configuration. The latter approach demonstrated improved validation accuracy, and during the ShpO of the airfoil achieved a similar solution with the adjoint-driven optimization, with a reduction of approximately 22% on the pt losses of the cascade and a change of 0.02^o in exit flow angle. The pt losses and exit angle networks were pruned resulting in 19% and 24% reduction in the network's storage requirements, respectively. In the second part of this case, DNNs trained on the initial 5 candidate solutions of the adjoint-driven

optimization, substituted the CFD-solver after its first 5 cycles. They achieved a reduction of 32% in the optimization turnaround time.
In the second case, the ShpO of an isolated airfoil was implemented, under single-phase transitional flow, aiming to minimize its c_d and keep the c_l close to the value of the baseline airfoil. The airfoil's polar diagram was validated against the experimental data. The airfoil was parameterized using a 10×9 NURBS lattice, and the control points were displaced in the normal to the chord direction resulting in 28 design variables. Separate models were built for each coefficient. In the first approach, the design variable space was sampled using LHS and a database containing 20 airfoils was generated. The outcomes of two DNN configuration optimizations were compared, the first one involved the network's architectures and the second one was enhanced with additional hyperparameters. An improvement was observed particularly in the SDs validation predictions. In the ShpO the latter network achieved a better solution than the adjoint-driven optimization though in a higher computational cost. The network's optimized solution resulted in a 25.9% reduction in the airfoil's c_d and a change of -0.2% in the c_l, while the adjoint-driven optimization reduced the c_d by 25.4% and maintained the same variation in the c_l. The impact of pruning was explored regarding the network's size. The reductions achieved were 15% and 24% for the c_d and c_l networks respectively. In the second approach a database was formed by the 6 initial solutions of the adjoint-driven optimization, and network's trained on that database resumed the optimization process from its 6th point. A reduction of 49% in the optimization turnaround was achieved. The network resulted in a solution with 26.2% reduced c_d and a change of -0.4% in the c_l. Due to the effectiveness of this method, it was adopted in the last two cases.
The second turbomachinery application concerned the ShpO of a compressor blade-airfoil, so as to minimize its p_t losses and maintain the exit flow close to the baseline. Each term was modeled by a district network. The flow was single-phase and turbulent. The shape of the geometry to be optimized, is parameterized using a 9×10 NURBS lattice. The active CPs were displaced in the chord-wise and pitch-wise direction resulting in a total of 40 design variables. A database containing the initial 5 candidate solutions of the adjoint-driven optimization was constructed. The optimal DNN configuration was determined by an EA-based optimization of its architecture and training configuration. The p_t losses and exit angle networks were also pruned, achieving a 22% and 14% reduction in storage requirements respectively. The DNN-driven optimization achieved a 53% reduction in turnaround time, yielding the same quality in its optimized solution with the adjoint-based process. The optimized solutions showed approximately 7.2% reduced p_t losses and a change of 0.01^o in exit flow angle.
The last case (hemispherical-cylinder body) was a two-phase hydrodynamic flow application around a hemispherical-cylinder body. Objective was the minimization of the cavitation formulation, subject to a constraint that the drag remained below a reference value. Each term is provided by a distinct network. The generatrix was parameterized using a Bezier curve with 10 control points. A total of 8 design variables (the y coordinates of the CPs) were considered in this case. The DNN

database consisted of the first 8 points of the adjoint-driven optimization and the DNN architecture and training configuration was optimized by EASY. The DNN-driven optimization with a slightly higher computational cost, provided a better solution; cavitation was reduced by 34.3% in the DNN-optimized solution, and by 32.1% in the adjoint-optimized solution.

8.2 General Conclusions

It is verified that the DNNs are capable of guiding a gradient-based ShpO. The proposed optimization algorithm (initially driven by adjoint, and then by DNNs), can substitute both primal and adjoint computations, achieving a reduction up to 50% in the optimization turnaround time. In some cases, the DNN-driven optimization yielded solutions with better objective values than the standard adjoint-driven one, however, it should be noted that the availability of an adjoint solver is necessary for the construction of the Hermite-DNNs. Another notable observation is the DNNs ability to extrapolate in logical bounds beyond their training domain, verifying the effectiveness of this method. An additional advantage of the proposed gradient-based optimization algorithm lies in this computational efficiency, since a DNN-driven gradient descent is of negligible cost relative to a CFD evaluation. Whenever the objective is modified or constraints are relaxed, good quality solutions can be obtained without the need to restart the costly CFD-based optimization process. Regarding the construction of the training database, utilizing solutions derived from an adjoint-driven optimization, as it was expected, yielded geometries with improved objective values compared to those generated by LHS and reduced the number of required patterns. Despite the reduced database size, the DNN-driven gradient descent remained feasible and was, in fact, conducted at a substantially lower computational cost.
The Hermite-DNNs delivered reliable predictions not only on the objective function but also on its SDs, which is of significance importance in a gradient-based ShpO. Their prediction accuracy was enhanced by optimizing their configuration using EASY. Expanding its design variable space enabled the discovery of superior models, that demonstrated improved generalization capabilities.
Finally, pruning revealed that the same level of accuracy can be achieved with a network up to 25% smaller, as a significant fraction of the weights have a smaller contribution to the final predictions.

8.3 Future Work Proposals

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

1. Firstly, the implementation of the proposed gradient-based optimization algorithm could be investigated in applications regarding unsteady CFD compu-

tation in single-phase or multi-phase flows. Given the high computational cost of such problems, the integration of DNN surrogates might offer a reduction in the computational demands.

2. Further applications of DNN differentiation could be explored, such as the computation of higher-order derivatives. In particular, DNNs could be used to approximate the Hessian matrix, avoiding the expensive direct computation, thereby providing a cost-efficient approach in case the Netwon's method is selected to drive the optimization.

Bibliography

[1] R. Mukhamediev, Y. Popova, Y. Kuchin, E. Zaitseva, A. Kalimoldayev, A. Symagulov, V. Levashenko, F. Abdoldina, V. Gopejenko, K. Yakunin, E. Muhamedijeva, M. Yelis, "*Review of Artificial Intelligence and Machine Learning Technologies: Classification, Restrictions, Opportunities and Challenges*," *Mathematics*, vol. 10, no. 15, 2022, https://www.mdpi.com/2227-7390/10/15/2552.

[2] S. Shalev-Shwartz, S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 2014.

[3] Y. Bengio, *Learning Deep Architectures for AI*, 2009.

[4] D. Elizondo, E. Fiesler, "*A survey of partially connected neural networks*," *International Journal of Neural Systems*, vol. 8, no. 5-6, pp. 535–558, Oct–Dec 1997.

[5] M. Kontou, D. Kapsoulis, I. Baklagis, X. Trompoukis, K. Giannakoglou, "λ-*DNNs and their implementation in conjugate heat transfer shape optimization*," *Neural Computing and Applications*, vol. 34, January 2022.

[6] V. Asouti, M. Kontou, K. Giannakoglou, "*Radial Basis Function Surrogates for Uncertainty Quantification and Aerodynamic Shape Optimization under Uncertainties*," *Fluids*, vol. 8, no. 11, 2023, https://www.mdpi.com/2311-5521/8/11/292.

[7] Y. Frey Marioni, E.Ortiz, A. Cassinelli, F. Montomoli, P. Adami, R. Vazquez, "*A Machine Learning Approach to Improve Turbulence Modelling from DNS Data Using Neural Networks*," *International Journal of Turbomachinery, Propulsion and Power*, vol. 6, no. 2, 2021, https://www.mdpi.com/2504-186X/6/2/17.

[8] S. Cai, Z. Mao, Z. Wang, M. Yin, G. Karniadakis, "*Physics-informed neural networks (PINNs) for fluid mechanics: A review*," *Acta Mechanica Sinica*, vol. 37, January 2022.

[9] M. Kontou, V. Asouti, K. Giannakoglou, "*DNN surrogates for turbulence closure in CFD-based shape optimization*," *Applied Soft Computing*, vol. 134, p. 110013, February 2023.

[10] D. Kochkov, J. Smith, A. Alieva, Q. Wang, M. Brenner, S. Hoyer, "*Machine learning–accelerated computational fluid dynamics*," *Proceedings of the National Academy of Sciences*, vol. 118, no. 21, p. e2101784118, 2021, https://www.pnas.org/doi/abs/10.1073/pnas.2101784118.

[11] J. Hammond, N. Pepper, F. Montomoli, V. Michelassi, "Machine learning methods in cfd for turbomachinery: A review," *International Journal of Turbomachinery, Propulsion and Power*, vol. 7, no. 2, 2022, https://www.mdpi.com/2504-186X/7/2/16.

[12] H. Wang, Y. Cao, Z. Huang, Y. Liu, P. Hu, X. Luo, Z. Song, W. Zhao, J. Liu, J. Sun, S. Zhang, L. Wei, Y. Wang, T.Wu, Z.H Ma, Y. Sun, "*Recent Advances on Machine Learning for Computational Fluid Dynamics: A Survey*," 2024, https://arxiv.org/abs/2408.12171.

[13] K. Kovani, M. Kontou, V. Asouti, K. Giannakoglou, *DNN-Driven Gradient-Based Shape Optimization in Fluid Mechanics*, June 2023, pp. 379–390.

[14] W. Boulila, M. Driss, M. Al-Sarem, F. Saeed, M. Krichen, "*Weight Initialization Techniques for Deep Learning Algorithms in Remote Sensing: Recent Trends and Future Perspectives*," 2021, https://arxiv.org/abs/2102.07004.

[15] J. Aryan, P. Avinash, J. Shruti, "*A Comprehensive Survey of Regression Based Loss Functions for Time Series Forecasting*," 2022, https://arxiv.org/abs/2211.02989.

[16] J. Qi, J. Du, S.M. Siniscalchi, X. Ma, C.H. Lee, "*On Mean Absolute Error for Deep Neural Network Based Vector-to-Vector Regression*," *IEEE Signal Processing Letters*, vol. 27, p. 1485–1489, 2020, http://dx.doi.org/10.1109/LSP.2020.3016837.

[17] D. Rumelhart, G. Hinton, R. Williams, "*Learning representations by back-propagating errors*," *Nature*, vol. 323, pp. 533–536, 1986, https://api.semanticscholar.org/CorpusID:205001834.

[18] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[19] S. Dubey, S. Singh, B. Chaudhuri, "*Activation functions in deep learning: A comprehensive survey and benchmark*," *Neurocomputing*, vol. 503, pp. 92–108, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231222008426

[20] D.P. Kingma, J. Ba, "*Adam: A Method for Stochastic Optimization*," 2017, https://arxiv.org/abs/1412.6980.

[21] I. Loshchilov, F. Hutter, "*Decoupled Weight Decay Regularization*," 2019, https://arxiv.org/abs/1711.05101.

[22] N. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, P. Tang, "*On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*," 2017, https://arxiv.org/abs/1609.04836.

[23] D. Masters, C. Luschi, "*Revisiting Small Batch Training for Deep Neural Networks*," 2018, https://arxiv.org/abs/1804.07612.

[24] R. Moradi, R. Berangi, B. Minaei, "*A survey of regularization strategies for deep models*," *Artificial Intelligence Review*, vol. 53, August 2020.

[25] A. Krogh, J. Hertz, "*A Simple Weight Decay Can Improve Generalization*," in *Advances in Neural Information Processing Systems*, R. L. J. Moody, S. Hanson, Ed., vol. 4. Morgan-Kaufmann, 1991, https://proceedings.neurips.cc/paper_files/paper/1991/file/8eefcfdf5990e441f0fb6f3fad709e21-Paper.pdf.

[26] G. Zhang, c. Wang, B. Xu, R. Grosse, "*Three Mechanisms of Weight Decay Regularization*," October 2018.

[27] M. Zhu, S. Gupta, "*To prune, or not to prune: exploring the efficacy of pruning for model compression*," 2017, https://arxiv.org/abs/1710.01878.

[28] H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P. Graf, "*Pruning Filters for Efficient ConvNets*," 2017, https://arxiv.org/abs/1608.08710.

[29] S. Han, H. Mao, and W.J. Dally, "*Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*," 2016, https://arxiv.org/abs/1510.00149.

[30] X. Ma, G. Fang, X. Wang, "*LLM-Pruner: On the Structural Pruning of Large Language Models*," 2023, *https://arxiv.org/abs/2305.11627.*

[31] H. Cheng, and M. Zhang, J.Q. Shi, "*A Survey on Deep Neural Network Pruning-Taxonomy, Comparison, Analysis, and Recommendations*," 2024, https://arxiv.org/abs/2308.06767.

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

[33] R. Kunz, D. Boger, D. Stinebring, T. Chyczewski, J. Lindau, H. Gibeling, S. Venkateswaran, T. Govindan, "*A preconditioned Navier–Stokes method for two-phase flows with application to cavitation prediction*," *Computers & Fluids*, vol. 29, no. 8, pp. 849–875, 2000, https://www.sciencedirect.com/science/article/pii/S0045793099000390.

[34] V. Asouti, X. Trompoukis, I. Kampolis, K. Giannakoglou, "Unsteady CFD computations using vertex–centered finite volumes for unstructured grids on Graphics Processing Units," *International Journal for Numerical Methods in Fluids*, vol. 67, no. 2, pp. 232–246, May 2011.

[35] L. Piegl, W. Tiller, *The NURBS book (2nd ed.).* Berlin, Heidelberg: Springer-Verlag, 1997.

[36] P. Spalart and S. Allmaras, "A one-equation turbulence model for aerodynamic flows," AIAA Paper 1992-439, 30th Aerospace Sciences Meeting and Exhibit, Reno, Nevada, USA, January 6–9 1992.

[37] R. Langtry, F. Menter, "*Correlation-Based Transition Modeling for Unstructured Parallelized Computational Fluid Dynamics Codes*," *AIAA Journal*, vol. 47, December 2009.

[38] M. Piotrowski, D. Zingg, "*Smooth Local Correlation-Based Transition Model for the Spalart-Allmaras Turbulence Model*," *AIAA Journal*, October 2020.

[39] K. Giannakoglou, V. Asouti, E. Papoutsis-Kiachagias, N. Galanos, M. Kontou, X. Trompoukis, "*The Think Discrete-Do Continuous Adjoint in Aerodynamic Shape Optimization.*" pp. 223–238, January 2023.

[40] M. Kontou, "The continuous adjoint method with consistent discretization schemes for transitional flows and the use of Deep Neural Networks in shape optimization in fluid mechanics," Ph.D. dissertation, National Technical University of Athens, 2023.

[41] M.Mckay, R. Beckmanm, W. Conover, "*A Comparison of Three Methods for Selecting Vales of Input Variables in the Analysis of Output From a Computer Code*," *Technometrics*, vol. 21, pp. 239–245, May 1979.

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

[43] D. Liu, J. Nocedal, "*On the limited memory BFGS method for large scale optimization*," *Mathematical Programming*, vol. 45, pp. 503–528, 1989, https://api.semanticscholar.org/CorpusID:5681609.

[44] L. Hylton, M. Mihelc, E. Turner, D. Nealy, R. York, "Analytical and Experimental Evaluation of the Heat Transfer Distribution over the Surface of Turbine Vanes," no. NASA-CR-168015, 1983.

[45] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM National Conference*, ser. ACM '68. New York, NY, USA: Association for Computing Machinery, 1968, p. 517–524. [Online]. Available: https://doi.org/10.1145/800186.810616

[46] https://www.tensorflow.org/.

[47] S. Leloudas, "Design and optimization of diffuser-augmented wind turbines," Doctoral Dissertation, School of Production Engineering and Management, Technical University of Crete, Chania, Greece, 2024.

[48] M. Selig, J. Guglielmo, A. Broeren, P. Giguere, *Summary of Low Speed Airfoil Data, Volume 1*, 01 1996.

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

Βαθέα Νευρωνικά Δίκτυα και η διαφόρισή τους για χρήση σε αιτιοκρατική βελτιστοποίηση μονοφασικών και πολυφασικών ροών

Διπλωματική Εργασία

Εκτενής Περίληψη στην Ελληνική

Ειρήνη-Σωτηρία Κεφαλούκου

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2025

Στόχος της διπλωματικής εργασίας είναι η χρήση Βαθέων Νευρωνικών Δικτύων (ΒΝΔ) ως υποκατάστατων του κώδικα Υπολογιστικής Ρευστοδυναμικής (ΥΡΔ). Η προτεινόμενη μέθοδος εφαρμόζεται σε προβλήματα αιτιοκρατικής βελτιστοποίησης μορφής για την πρόβλεψη των τιμών της συνάρτησης-στόχου και των παραγώγων της.

ΒΝΔ

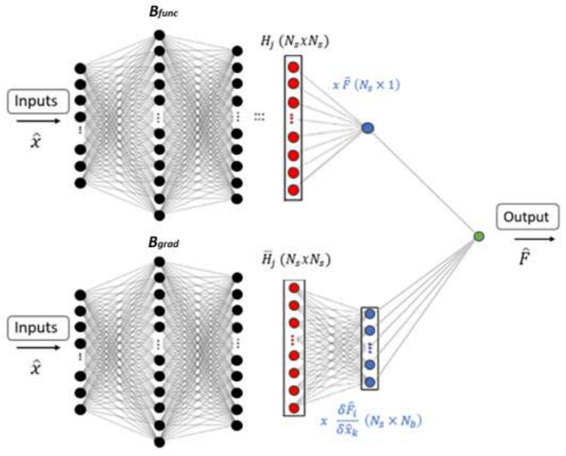
Για την υλοποίηση χρησιμοποιήθηκαν ΒΝΔ τα οποία προσομοιάζουν τη δομή και τις ιδιότητες των πολυωνύμων *Hermite* ([32]). Το πολυώνυμα *Hermite*, μαζί με τις πρώτες τους παραγώγους, ικανοποιούν τη δεδομένη συνάρτηση και τις τιμές παραγώγων της στα δεδομένα σημεία. Θυμίζεται ότι το πολυώνυμο παρεμβολής μπορεί να εκφραστεί ως ένας γραμμικός συνδυασμός ορθογώνιων πολυωνύμων, τα οποία ονομάζονται πολυώνυμα βάσης. Ορίζονται δύο τύποι πολυωνύμων βάσης, το H_j, όπου j=1,..,N$_s$, το οποίο συμβάλλει μόνο στην τιμή της συνάρτησης για το σημείο j, χωρίς να εμπλέκεται στην ικανοποίηση της παραγώγου και το $\overline{H}_j$ που έχει τον ακριβώς αντίθετο ρόλο. Τα πολυώνυμα ορίζονται με βάση τα πολυώνυμα *Langrage*. Η τελική συνάρτηση παρεμβολής είναι

$$g(x) = \sum_{j=0}^{N} y_j H_j(x) + \sum_{j=0}^{N} y'_j \overline{H}_j(x) \tag{8.1}$$

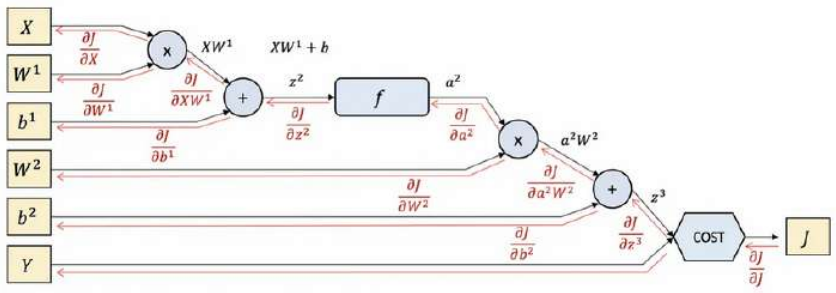
Για τη γενίκευση της μεθόδου παρεμβολής κατά *Hermite* σε υψηλότερες διαστάσεις, προτείνεται η αξιοποίηση ΒΝΔ. Το προτεινόμενο δίκτυο αποτελείται από δύο διακριτούς κλάδους, καθένας εκ των οποίων μοντελοποιεί έναν από τους όρους της προηγούμενης εξίσωσης. Ο πρώτος κλάδος καταλήγει σε δύο επιπλέον επίπεδα. Το πρώτο, γνωστό ως επίπεδο βάσης, περιλαμβάνει αριθμό νευρώνων ίσο με τον αριθμό των δειγμάτων εκπαίδευσης. Το τελικό επίπεδο αποτελείται από έναν μόνο νευρώνα, του οποίου τα βάρη αρχικοποιούνται με τις τιμές των y_i των δειγμάτων εκπαίδευσης και δεν ανανεώνονται κατά την εκπαίδευση του δικτύου. Η έξοδος αυτού του επιπέδου προσεγγίζει τον πρώτο όρο της παραπάνω εξίσωσης. Ο δεύτερος κλάδος παρουσιάζει αντίστοιχη αρχιτεκτονική, με τη διαφορά ότι στο τελικό επίπεδο περιλαμβάνεται αριθμός νευρώνων ίσος με τον αριθμό των μεταβλητών σχεδιασμού του προβλήματος. Με αυτόν τον τρόπο, λαμβάνεται υπόψη η επίδραση κάθε μερικής παραγώγου, μέσω όρων ανάλογων εκείνων του δεύτερου όρου της εξίσωσης. Η αρχιτεκτονική του *Hermite*-ΒΝΔ παρουσιάζεται στο Σχήμα 8.1.

Διαφόριση των ΒΝΔ.

Η εκπαίδευση των ΒΝΔ απαιτεί τον υπολογισμό των παραγώγων της συνάρτησης κόστους ως προς κάθε παράμετρο του μοντέλου. Αυτό επιτυγχάνεται μέσω της αυτόματης διαφόρισης. Το σφάλμα μεταφέρεται από το τελευταίο επίπεδο του δικτύου προς τα πίσω, μέχρι την είσοδο. Αντίστοιχα, μπορούν να υπολογιστούν και οι παράγωγοι της εξόδου του δικτύου ως προς τις εισόδους του, οι οποίες αντιστοιχούν στις παραγώγους ευαισθησίας.. Σε επίπεδο ενός νευρώνα, ο υπολογισμός των παραγώγων βασίζεται στην εφαρμογή του κανόνα της αλυσίδας δύο φορές, όπως παρουσιάζεται στο Σχήμα 8.2



Σχήμα 8.1: *Αρχιτεκτονική δικτύου Hermite.*



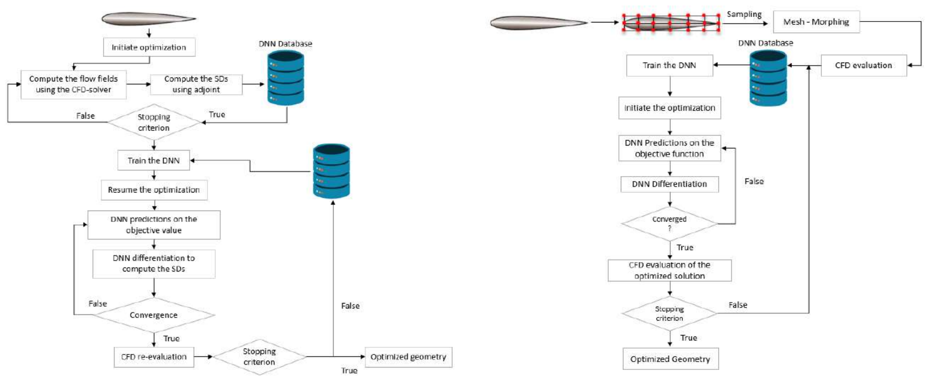
X
W^1
b^1
W^2
b^2
Y
$\frac{\partial J}{\partial X}$
$\frac{\partial J}{\partial W^1}$
$\frac{\partial J}{\partial b^1}$
XW^1
$XW^1 + b$
$\frac{\partial J}{\partial XW^1}$
z^2
$\frac{\partial J}{\partial z^2}$
f
a^2
$\frac{\partial J}{\partial a^2}$
a^2W^2
$\frac{\partial J}{\partial a^2W^2}$
$\frac{\partial J}{\partial W^2}$
z^3
$\frac{\partial J}{\partial z^3}$
$\frac{\partial J}{\partial b^2}$
COST
J
$\frac{\partial J}{\partial J}$

Σχήμα 8.2: *Αλγόριθμος Backpropagation.*

Αποκοπή κλάδων των ΒΝΔ.
Η αποκοπή κλάδων αποτελεί τεχνική μείωσης της πολυπλοκότητας ενός ΒΝΔ, μέσω της αφαίρεσης συνδέσεων (βάρων) που θεωρούνται λιγότερο σημαντικές για την τελική πρόβλεψη. Στο πλαίσιο της εργασίας αυτής, η αφαίρεση επιμέρους βαρών μεταξύ των νευρώνων πραγματοποιείται με κριτήριο το μέγεθος του βάρους και ενσωματώνεται στη διαδικασία εκπαίδευσης.

Ο προτεινόμενος αλγόριθμος αιτιοκρατικής βελτιστοποίησης μορφής
Τα *Hermite*-ΒΝΔ χρησιμοποιούνται για την πρόβλεψη της ροής και τον υπολογισμό των παραγώγων ευαισθησίας, ως υποκατάστατα του λογισμικού ΥΡΔ. Ο υπολογισμός των παραγώγων υλοποιείται με τη συζυγή μέθοδο ([39]). Επ΄ αυτού, αξιολογούνται δύο μέθοδοι, οι οποίες αναλύονται στην συνέχεια.
Στην πρώτη προσέγγιση, για τη δημιουργία της βάσης δεδομένων που θα χρησιμοποιηθεί για την εκπαίδευση των ΒΝΔ, παράγεται ένα σύνολο πιθανών γεωμετριών μέσω δειγματοληψίας του χώρου των μεταβλητών σχεδιασμού. Η δειγματοληψία γίνεται με τη μέθοδο *LHS*, η οποία εξασφαλίζει αντιπροσωπευτικά δείγματα ακόμη και με μικρό αριθμό δεδομένων. Λόγω του υψηλού υπολογιστικού κόστους αξιολόγησης κάθε γεωμετρίας, ο αριθμός των δειγμάτων περιορίζεται. Η αιτιοκρατική βελτιστοποίηση ξεκινά από την αρχική γεωμετρία και βασίζεται αποκλειστικά στις προβλέψεις των ΒΝΔ.
Η δεύτερη προσέγγιση αρχικοποιεί τη βελτιστοποίηση χρησιμοποιώντας τον κώδικα

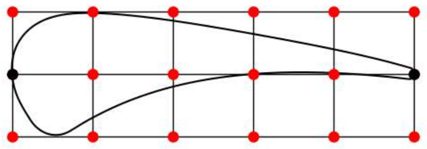
ΥΡΔ. Κάθε ενδιάμεση πιθανή γεωμετρία προστίθεται στη βάση δεδομένων. Μετά από τους πρώτους κύκλους, η βελτιστοποίηση διακόπτεται. Τα ΒΝΔ εκπαιδεύονται πάνω στη νέα βάση δεδομένων και η κάθοδος προχωρά από τη βέλτιστη μέχρι τότε λύση, χρησιμοποιώντας αποκλειστικά τα ΒΝΔ.
Η βελτιστοποιημένη λύση αξιολογείται ξανά με τον κώδικα ΥΡΔ. Αν πληροί τις απαιτήσεις ακρίβειας, η διαδικασία ολοκληρώνεται. Διαφορετικά, η νέα γεωμετρία προστίθεται στη βάση δεδομένων και επαναλαμβάνονται τα βήματα εκπαίδευσης και βελτιστοποίσης, ξεκινώντας από την τελευταία λύση. Η αρχιτεκτονική όλων των ΒΝΔ, βελτιστοποιείται με το λογισμικό εξελικτικών αλγορίθμων, *EASY* ([42]). Οι προτεινόμενοι αλγόριθμοι αιτιοκρατικής βελτιστοποίησης παρουσιάζονται στο Σχήμα 8.3.



Σχήμα 8.3: *Σχηματική αναπαράσταση των προτεινόμενων μεθόδων βελτιστοποίσης.*

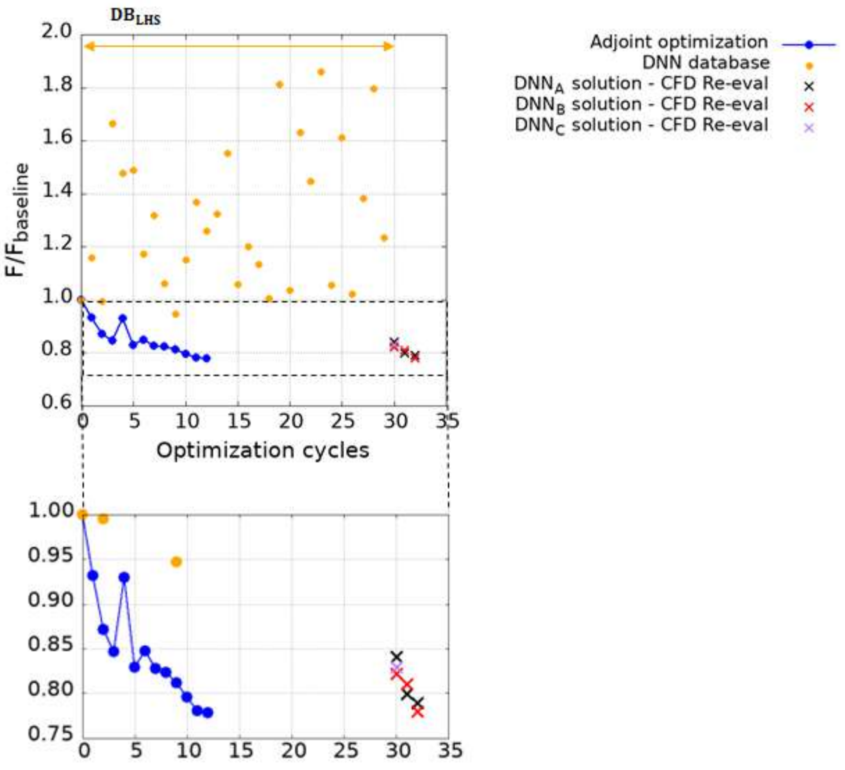
Τυρβώδης μονοφασική ροή γύρω από αεροτομή πτερυγίου στροβίλου

Η προτεινόμενη μέθοδος εφαρμόζεται αρχικά στην πτερύγωση στροβίλου $C3X$. Στόχος της βελτιστοποίησης είναι η ελαχιστοποίηση των απωλειών ολικής πίεσης (Δp_t), διατηρώντας τη γωνία εξόδου της ροής περίπου ίση με της αρχικής γεωμετρίας (a_{exit}). Κάθε ένας όρος μοντελοποιείται με ένα ξεχωριστό ΒΝΔ. Η ροή είναι τυρβώδης με συνθήκες εισόδου p_t=2.44bar, T_t=808K. Για την παραμετροποίηση χρησιμοποιείται ένα 6×3 κουτί παραμετροποίησης *NURBS*. Κάθε σημείο ελέγχου μπορεί να μετατοπισθεί έως 0.15 της απόστασης του από γειτονικά σημεία, και στις δύο διευθύνσεις, καταλήγοντας σε 32 μεταβλητές σχεδιασμού. Η παραμετροποίηση παρουσιάζεται στο Σχήμα 8.4



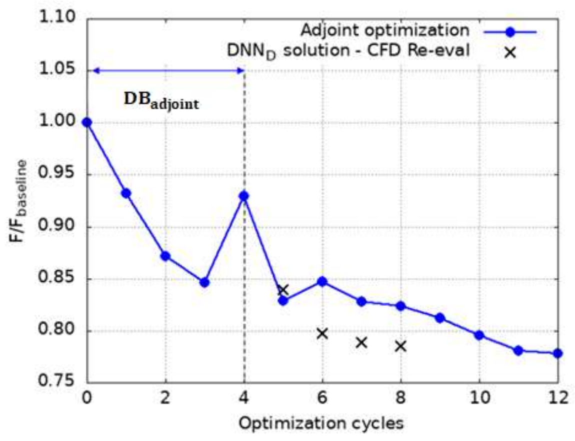
Σχήμα 8.4: *Περίπτωση στροβίλου: Παραμετροποίηση αεροτομής πτερυγίου C3X.*

Στη μελέτη αυτή συγκρίνονται δύο διαφορετικές διαδικασίες κατασκευής της βάσης δεδομένων για την εκπαίδευση των ΒΝΔ. Η πρώτη μέθοδος βασίζεται σε μια σχεδόν τυχαία δειγματοληψία του 32-διάστατου χώρου, χρησιμοποιώντας τη μέθοδο LHS. Εκτελούνται δύο βελτιστοποιήσεις της μορφής των ΒΝΔ. Η πρώτη (ΒΝΔ$_A$) αφορά την αρχιτεκτονική τους και η δεύτερη (ΒΝΔ$_B$) συμπεριλαμβάνει και τη διαδικασία εκπαίδευσης τους. Στη δεύτερη περίπτωση παρατηρείται μικρότερο σφάλμα στα δεδομένα επικύρωσης. Επιπλέον, εφαρμόζεται αποκοπή κλάδων (ΒΝΔ$_C$) με στόχο τη μείωση της πολυπλοκότητας των δικτύων και των απαιτήσεων σε μνήμη για την αποθήκευσή τους. Επιτυγχάνεται μείωση κατα 19% για το Δp_t ΒΝΔ και 24% για το a_{exit} ΒΝΔ. Οι τιμές της αντικειμενικής αποτυπώνονται στο Σχήμα 8.5.

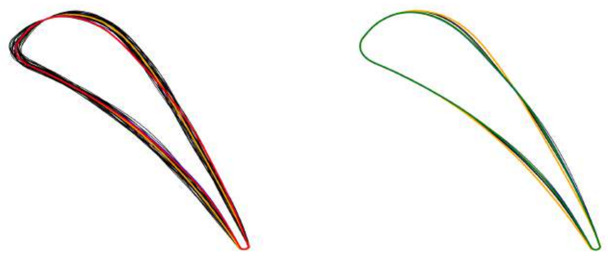


Σχήμα 8.5: *Περίπτωση στροβίλου: Η σύγκλιση της βελτιστοποίησης με τη συζυγή μέθοδο, οι βέλτιστες λύσεις με κάθε δίκτυο αφού επαναξιολογήθηκαν με τον κώδικα ΥΡΔ και η βάση δεδομένων των ΒΝΔ.*

Η δεύτερη μέθοδος εκπαιδεύει τα δίκτυα στα πέντε πρώτα σημεία της βελτιστοποίησης με τη συζυγή μέθοδο. Συνεχίζει την κάθοδο από το πέμπτο σημείο, βασιζόμενη αποκλειστικά στις προβλέψεις των δικτύων (ΒΝΔ$_D$). Λόγω του περιορισμένου αριθμού δειγμάτων, ολόκληρη η βάση δεδομένων χρησιμοποιείται για την εκπαίδευση. Για το ΒΝΔ$_B$ απαιτούνται 2 επανεκπαιδεύσεις προκειμένου να επιτευχθεί λύση συγκρίσιμη με αυτή που προκύπτει με τη συζυγή μέθοδο, ενώ για το ΒΝΔ$_D$ 3 επανεκπαιδεύσεις, όπως φαίνεται στο Σχήμα 8.6. Και στις δύο περιπτώσεις επιτυγχάνεται παρόμοια τιμή στη συνάρτηση-στόχο συγκριτικά με τη συζυγή μέθοδο, με μείωση 22% στο Δp_t και μεταβολή της τάξης των 0.02^o στην a_{exit}. Η χρήση των δικτύων ΒΝΔ$_D$ οδηγεί σε μείωση του υπολογιστικού κόστους, κατά 32%.



Σχήμα 8.6: *Περίπτωση στροβίλου: Η σύγκλιση της βελτιστοποίησης με τη συζυγή μέθοδο, και οι λύσεις του ΒΝΔ$_D$ αφού επαναξιολογήθηκαν με τον κώδικα ΥΡΔ.*

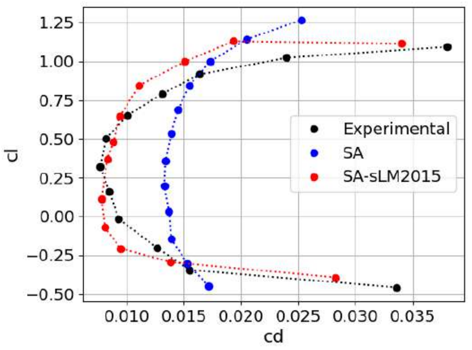


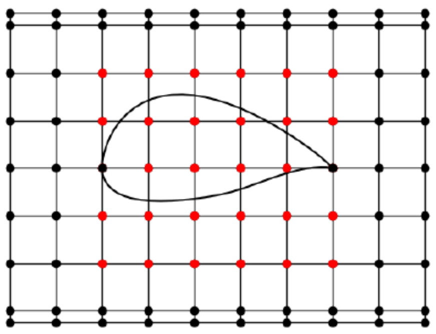
Σχήμα 8.7: *Περίπτωση στροβίλου: Αριστερά) Γεωμετρίες που παράχθηκαν με τη μέθοδο LHS (μαύρο), η βελτιστοποιημένη γεωμετρία με τη συζυγή μέθοδο (μπλε) και το ΒΝΔ$_B$ (κόκκινη). Δεξιά) Πρώτες λύσεις της βελτιστοποίσης με τη συζυγή μέθοδο (μαύρο). Σχεδιάζονται επίσης, η αρχική γεωμετρία (πορτοκαλί) και η βέλτιστη χρησιμοποιώντας το δίκτυο ΒΝΔ$_D$ (πράσινο).*

Μονοφασική ροή με μετάβαση γύρω από μεμονωμένη αεροτομή

Η αεροτομή $RG15$ χρησιμοποιείται σε εφαρμογές χαμηλού αριθμού $Reynolds$, όπως σε πτερύγια ανεμογεννητριών. Οι πολικές της αεροτομής συγκρίνονται με τα πειραματικά δεδομένα στο Σχήμα 8.8, σε αριθμό Re=304.200. Χρησιμοποιήθηκε το μοντέλο τύρβης $Spalart-Allmaras$ και το μοντέλο μετάβασης $SA-sLM2015$. Στόχος είναι η ελαχιστοποίηση του συντελεστή αντίστασης (c_d), διατηρώντας το συντελεστή άνωσης (c_l), περίπου ίσο με της αρχικής αεροτομής. Η μελέτη υλοποιείται σε M=0.1 και Re=$1.5\cdot10^5$. Η παραμετροποίηση πραγματοποιείται χρησιμοποιώντας ένα 10×9 πλέγμα σημείων ελέγχου. Τα σημεία ελέγχου μετατοπίζονται μόνο στην κάθετη στη χορδή διεύθυνση, με μέγιστη μετατόπιση 0.4 της απόστασης από γειτονικά σημεία.

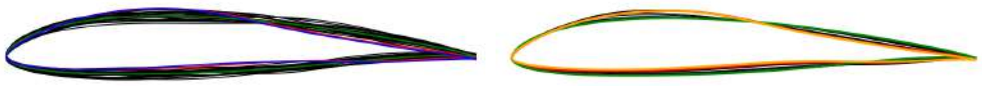
Συγκρίνονται δύο μέθοδοι κατασκευής της βάσης δεδομένων. Η πρώτη υλοποιείται με LHS, παράγοντας 20 δείγματα για τις 28 μεταβλητές σχεδιασμού του προβλήματος,



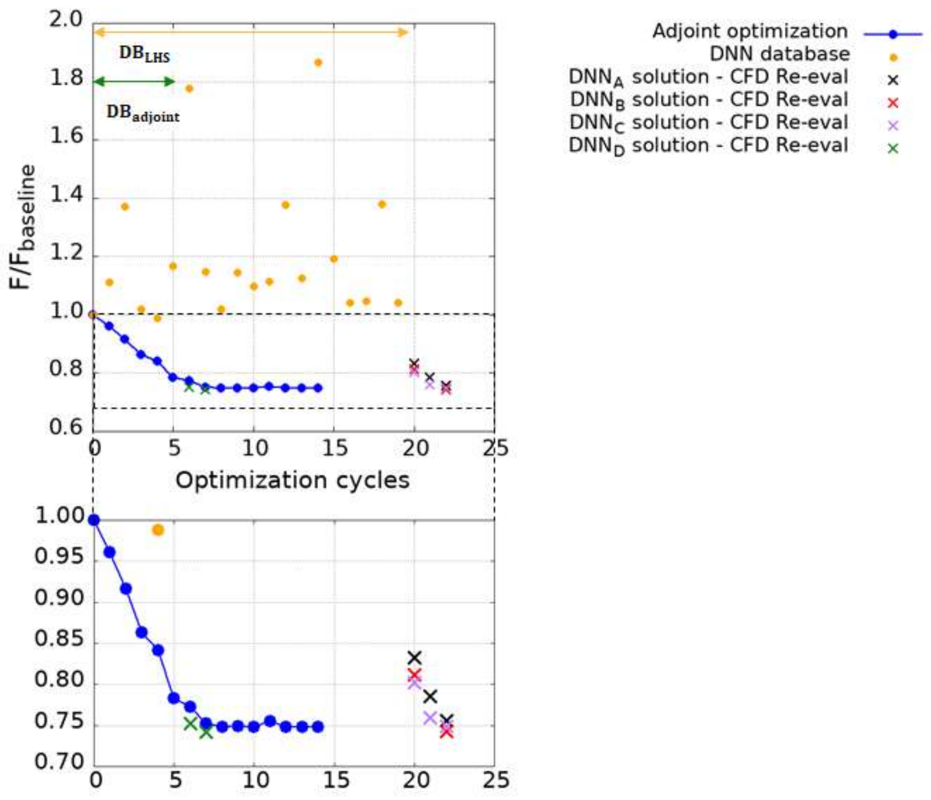


Σχήμα 8.8: *Περίπτωση μεμονωμένης αεροτομής: Πολικές της αεροτομής $RG15$ (α-ριστερά) και η παραμετροποίησης της (δεξιά).*

ενώ η δεύτερη (ΒΝΔ$_D$) χρησιμοποιεί τις 6 πρώτες πιθανές λύσεις της βελτιστοποίησης με τη συζυγή μέθοδο. Αναφορικά με την πρώτη μέθοδο, η βελτιστοποίηση του ΒΝΔ πραγματοποιείται δύο φορές και τα ευρήματα συγκρίνονται. Στο καλύτερο από τα προηγούμενα δίκτυα (ΒΝΔ$_B$) εφαρμόζεται αποκοπή κλάδων (ΒΝΔ$_C$), επιτυγχάνοντας μείωση μεγέθους κατά 15% για το c_d ΒΝΔ και 24% για το c_l ΒΝΔ.
Τα ΒΝΔ$_B$ και ΒΝΔ$_D$ επιτυγχάνουν καλύτερη λύση από τη λύση της βελτιστοποίησης με τη συζυγή μέθοδο, ωστόσο χρησιμοποιώντας τα δίκτυα ΒΝΔ$_D$, υπάρχει μείωση του κόστους κατά 49%. Η βελτιστοποίηση με τη συζυγή μέθοδο αποδίδει μείωση του c_d κατά 25.4% και μεταβολή στο c_l -0.2%, ενώ το ΒΝΔ$_D$ επιτυγχάνει μείωση 26.2% και μεταβολή -0.4%, αντίστοιχα. Οι λύσεις παρουσιάζονται στο Σχήμα 8.10 και οι βελτιστοποιημένες αεροτομές στο Σχήμα 8.9

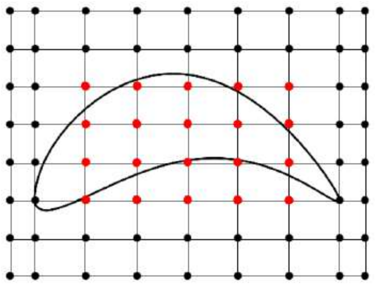


Σχήμα 8.9: *Περίπτωση μεμονωμένης αεροτομής: Αριστερά) Αεροτομές με LHS (μαύρο) και οι γεωμετρίες που παράχθηκαν με τη βελτιστοποίηση με συζυγή μέθοδο (κόκκινο) και το ΒΝΔ$_B$ (μπλε). Δεξιά) Η αρχική γεωμετρία (πράσινο), η βέλτιστη λύση με τα ΒΝΔ$_D$ (πορτοκαλί) και οι αεροτομές που περιλαμβάνονται στη βάση δεδομένων τους (μαύρο).*



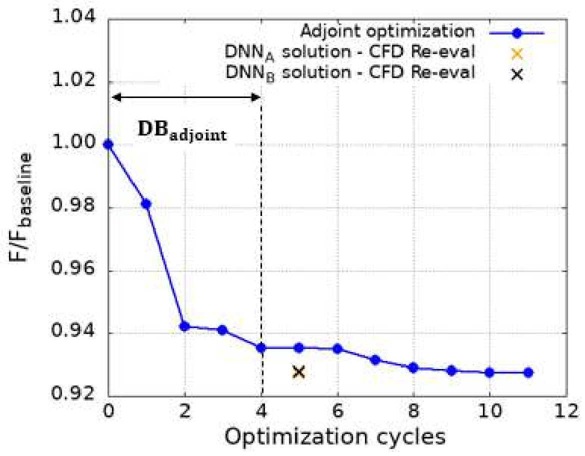
Σχήμα 8.10: *Περίπτωση μεμονωμένης αεροτομής: Οι βάσεις δεδομένων για κάθε δίκτυο, η σύγκλιση της βελτιστοποίησης με τη συζυγή μέθοδο και οι λύσεις που αποκτήθηκαν με κάθε ΒΝΔ.*

Τυρβώδης μονοφασική ροή γύρω από αεροτομή πτερυγίου συμπιεστή
Στόχος της βελτιστοποίησης στη συγκεκριμένη μελέτη είναι η ελαχιστοποίση των απωλειών ολικής πίεσης της πτερύγωσης, διατηρώντας περίπου σταθερή τη γωνία εξόδου της ροής. Η ροή είναι χαμηλής ταχύτητας και τυρβώδης. Οι συνθήκες εισόδου είναι $p_t = 1.15bar$, $T_t = 288K$. Η μορφή της αεροτομής και του πλέγματος ελέγχονται από ένα κουτί παραμετροποίησης $NURBS$ 9×8, όπως φαίνεται στο Σχήμα 8.11. Κάθε σημείο ελέγχου μπορεί να μετατοπιστεί κατά μέγιστο 0.1 της απόστασης από τα γειτονικά του σημεία και στις δύο διευθύνσεις, με αποτέλεσμα ο συνολικός αριθμός των μεταβλητών σχεδιασμού να ανέρχεται σε 40.

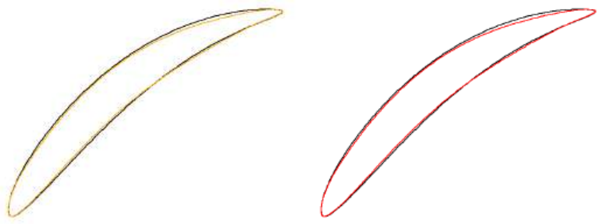


Σχήμα 8.11: *Περίπτωση συμπιεστή: Παραμετροποίηση αεροτομής.*

Σε αυτήν τη μελέτη υλοποιείται η δεύτερη προτεινόμενη μέθοδος αιτιοκρατικής βελτιστοποίησης. Η βελτιστοποίηση στους αρχικούς πέντε κύκλους βασίζεται στη συνεχή συζυγή μέθοδο. Κάθε ενδιάμεση λύση προστίθεται στη βάση δεδομένων και χρησιμοποιείται για να εκπαιδευθούν τα ΒΝΔ (ΒΝΔ$_A$). Η κάθοδος συνεχίζει από την 5η γεωμετρία, χρησιμοποιώντας αποκλειστικά τις προβλέψεις των ΒΝΔ.
Επιπλέον, εξετάζεται η επίδραση της αποκοπής κλάδων στη σύγκλιση και την ακρίβεια των προβλέψεων. Το τελικό ΒΝΔ με αποκοπή κλάδων (ΒΝΔ$_B$) είναι κατά 22% και 14% πιο ελαφρύ από τα αρχικά δίκτυα Δp_t, a_{exit} χωρίς αποκοπή κλάδων, αντίστοιχα. Η προτεινόμενη αιτιοκρατική μέθοδος βελτιστοποίησης είναι κατά 53% γρηγορότερη από τη βελτιστοποίηση με τη συζυγή μέθοδο, καταλήγοντας σε ίδιας ποιότητας λύση, όπως φαίνεται στο Σχήμα 8.12. Η βελτιστοποιημένη αεροτομή παρουσιάζει μείωση απωλειών ολικής πίεσης περίπου 7.2% και μεταβολή στην γωνία εξόδου της τάξης των 0.01^o. Οι βελτιστοποιημένες αεροτομές παρουσιάζονται στο Σχήμα 8.13.



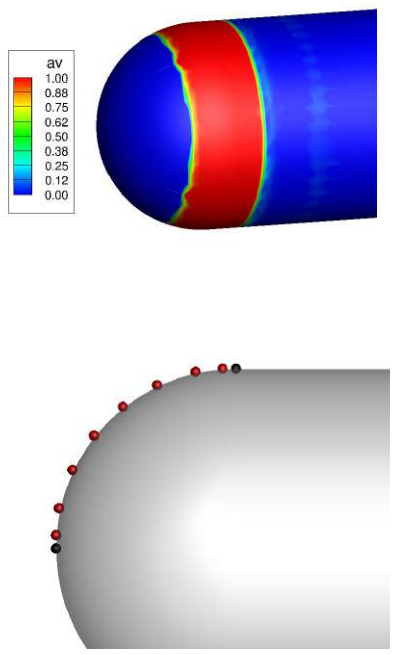
Σχήμα 8.12: *Περίπτωση συμπιεστή: Σύγκλιση της βελτιστοποίησης με τη συζυγή μέθοδο και οι λύσεις με τα ΒΝΔ αφού επαναξιολογήθηκαν με τον κώδικα ΥΡΔ.*



Σχήμα 8.13: *Περίπτωση συμπιεστή: Η βέλτιστη αεροτομή με το ΒΝΔ$_A$ (δεξιά) και με το ΒΝΔ$_B$ (αριστερά).*

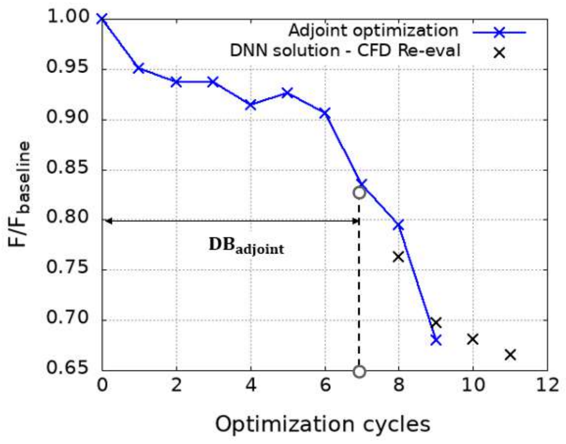
Διφασική ροή γύρω από ημισφαιρικό κυλινδρικό σώμα

Εδώ μελετάται μία διφασική υδροδυναμική ροή, η οποία βασίζεται στην ομογενή προσέγγιση (3.1). Η μελέτη αυτή αφορά τη βελτιστοποίηση ενός ημισφαιρικού κυλινδρικού σώματος με στόχο να μειωθεί η σπηλαίωση και περιορισμό η αντίσταση να μην υπερβεί μία τιμή αναφοράς, ίση με της αρχικής γεωμετρίας. Η σπηλαίωση δημιουργείται όταν τοπικά η στατική πίεση γίνει μικρότερη από την πίεση ατμοποίησης του νερού. Η παρουσία φυσαλίδων ατμού είναι ασταθής, και σε περίπτωση αύξησης πίεσης της ροής, καταρρέουν δημιουργώντας κύματα κρούσης, διάβρωση και κραδασμούς. Στη γεωμετρία που μελετάται, η επιτάχυνση της ροής στην επιφάνεια του σώματος δημιουργεί μία ζώνη σπηλαίωσης, όπως φαίνεται στο Σχήμα 8.14. Μόνο το ημισφαιρικό τμήμα της παραπάνω γεωμετρίας παραμετροποιείται. Για να διατηρηθεί η συμμετρία του, παραμετροποιείται η γενέτειρα του με καμπύλη *Bezier* αποτελούμενη από 10 σημεία ελέγχου. Τα κόκκινα σημεία ελέγχου μετατοπίζονται στην κάθετη διεύθυνση, επηρεάζοντας με αυτό τον τρόπο την τοπική διάμετρο.

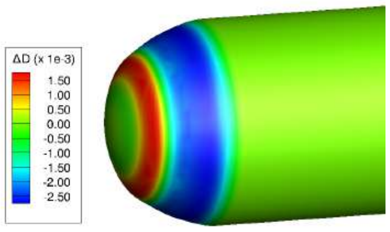


Σχήμα 8.14: *Περίπτωση διαφασικής ροής: Πάνω) Κλάσμα όγκου ατμού γύρω από το στερεό όριο. Κάτω) Παραμετροποίηση του σώματος.*

Σε αυτην τη μελέτη υλοποιείται η δεύτερη προτεινόμενη μέθοδος αιτιοκρατικής βελτιστοποίησης. Στους οκτώ αρχικούς κύκλους η βελτιστοποίηση βασίζεται στη συζυγή μέθοδο. Τα δίκτυα εκπαιδεύονται πάνω στις οκτώ πρώτες γεωμετρίες που παράγονται και συνεχίζουν την κάθοδο από την τελευταία. Κάθε όρος της αντικειμενικής συνάρτησης αποδίδεται από ένα ξεχωριστικό δίκτυο, το οποίο βελτιστοποιείται με τον *EASY*. Έπειτα από 3 επανεκπαιδεύσεις, η προτεινόμενη βελτιστοποίηση, με λίγο μεγαλύτερο κόστος, οδηγεί σε καλύτερη τιμή της συνάρτησης-στόχου, με μείωση της σπηλαίωσης κατά 34.3%, σε σύγκριση με τη συζυγή μέθοδο που αποδίδει μείωση 32.1%. Η σύγκλιση παρουσιάζεται στο Σχήμα 8.15. Η μεταβολή διαμέτρου της βελτιστοποιημένης λύσης που αποκτήθηκε με το δίκτυο παρουσιάζεται στο Σχήμα 8.16.



Σχήμα 8.15: *Περίπτωση διαφασικής ροής: Σύγκλιση της βελτιστοποίησης με τη συζυγή μέθοδο και οι προβλέψεις των ΒΝΔ αφού επαναξιολογήθηκαν με τον κώδικα ΥΡΔ.*



Σχήμα 8.16: *Περίπτωση διαφασικής ροής: Μεταβολή της τοπικής διαμέτρου στη βελτιστοποιημένη γεωμετρία.*

Συμπεράσματα

Τα ΒΝΔ μπορούν να χρησιμοποιηθούν ως υποκατάστατα του κώδικα ΥΡΔ. Ο προτεινόμενος αλγόριθμος οδήγησε σε μείωση κόστους έως και 50%. Σε ορισμένες μελέτες, απέδωσε καλύτερη λύση σε σύγκριση με τη βελτιστοποίηση με τη συζυγή μέθοδο, η διαθεσιμότητα της οποίας όμως είναι απαραίτητη για να υπάρξει η μέθοδος αυτή. Η χρήση των *Hermite*-ΒΝΔ εξασφάλισε ακρίβεια στις υπολογιζόμενες παραγώγους ευαισθησίας, η οποία ενισχύθηκε με τη βελτιστοποίηση επιπλέον παραμέτρων των δικτύων. Η δημιουργία βάσης δεδομένων με υποψήφιες λύσεις της βελτιστοποίησης με τη συζυγή μέθοδο μείωσε το κόστος κατασκευής της και απέδωσε γεωμετρίες με καλύτερη τιμή συνάρτησης-στόχου σε σχέση με τη μέθοδο *LHS*. Τέλος, η αποκοπή κλάδων απέδειξε ότι ένα σημαντικό ποσοστό παραμέτρων δεν είναι σημαντικές για την πρόβλεψη και μπορούν να παραλειφθούν, με στόχο τη χρήση έως και 25% μικρότερου δικτύου.