Initial Mesh Generation for Solution-Adaptive Methods Using Machine Learning

Vivek Ojha*, Guodong Chen †, and Krzysztof J. Fidkowski‡
Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109, USA

This paper presents a machine-learning approach for determining anisotropic initial meshes in the context of an output-based adaptive solution procedure. Artificial neural networks are used to predict the desired element sizing and anisotropy from flow conditions and geometry parameters. The network serves as a computationally efficient acceleration technique compared to a more common approach of starting with a hand-generated initial mesh and then using adjoint-weighted residual error estimates for element sizing and mesh optimization through error sampling and synthesis (MOESS) for evaluating the element stretching magnitude and direction. The network is trained to provide element sizing, orientation, and stretching in the form of a normalized metric field, as a function of position in a normalized domain. The MOESS-optimized meshes for a variety of steady aerodynamic flows governed by the Reynolds-averaged Navier–Stokes equations in two dimensions provide data for training the multi-layer perceptron network. The network is then deployed and tested by driving mesh adaptation, and the results show similar optimized meshes obtained from the network at much lower computational cost for steady-state cases.

I. Introduction

By virtue of the fast growing computational capacity and highly developed numerical methods, complex aerodynamic simulations are now routinely carried out in aerospace design and analysis. Most of these simulations rely on spatial discretization of the physical domain and appropriate time evolution strategies of the discretized system. The spatial discretization, *i.e.*, the computational mesh, therefore strongly affects both the accuracy and efficiency of the computations. An *a priori* mesh generation procedure based on engineering experience often gives sub-optimal meshes since such a method is mostly agnostic to the solution accuracy or numerical error. A better strategy is the solution-adaptive meshing process, in which the computational mesh is adapted using estimates of the numerical error. In scenarios where the quantities of interest are scalar outputs of the system, the adjoint-based method provides a robust and effective way of quantifying the numerical error and driving the mesh adaptation.[1]

In adjoint-based mesh adaptation, the adjoint variables weight the flow residual to form an estimate of the output error. The adjoint-weighting is applied locally and identifies regions that produce high output errors. Solution-adaptive methods based on adjoint-weighted residual have dramatically improved the accuracy and efficiency of aerodynamic simulations.[2–9] Despite their great success in aerospace applications, the additional computational cost and implementation complexity associated with adjoint-based methods cannot be neglected. First, adjoint-based methods require solving a dual linear system, *i.e.*, the adjoint equation set, which is of the same size as the primal system or even larger when solving on an enriched space. Second, the implementation of adjoint methods often requires the transpose of the residual Jacobian matrix, which is not always available in explicit solvers or Jacobian-free methods. In these circumstances, either the continuous adjoint equations could be derived and directly discretized,[10] or special implementation efforts are required,[11] adding considerable costs and effort in the development. The additional computational costs associated with the adjoint solutions, in addition to the implementation efforts, have largely hindered more

*Ph.D. Candidate, AIAA Student Member
†Postdoctoral Research Fellow, AIAA Member
‡Associate Professor, AIAA Associate Fellow

widespread use of adjoint-based adaptation techniques in practice.

In order to simplify and accelerate the mesh adaptation procedure, several attempts have been explored recently using machine learning techniques, chosen mainly for their non-intrusive nature and fast online evaluations. Manevitz *et al.*[12] used feed-forward neural networks to predict the solution gradients in time-dependent problems, which then provided an indicator to drive mesh adaptation. Zhang *et al.* proposed MeshingNet[13] which takes in a chosen set of parameters to predict the local mesh density, through feed-forward networks with residual connections. Huang *et al.*[14] treated the computational mesh with various element sizes as a gray-scale image such that convolutional neural network architectures can be applied in mesh generation. Chen and Fidkowski[15] presented a convolutional neural network using a physical-reference mapping to avoid direct treatments of physical quantities as images. Most of these studies only focused on the element sizing and largely ignored mesh anisotropy, *i.e.*, element stretching and orientation. However, anisotropic meshes are important in practice for efficiently resolving certain flow features, such as boundary layers, wakes, and shocks in aerodynamic simulations. For optimal anisotropic mesh generation, Fidkowski and Chen[16] designed feed-forward networks to predict the mesh anisotropy information using the flow primal and adjoint features, while still requiring appropriate sizing information from standard adjoint-based error estimates.

In this work, we investigate the idea of using a neural network to predict the optimal computational mesh, including both the element sizing and stretching. We follow the work of Fidkowski and Chen,[16] while taking the element sizing also as a network output. Furthermore, in order to enable fast mesh generation, *a priori* information, such as the geometry and the boundary conditions, are taken as the network inputs instead of the primal and adjoint solutions. The network is trained with optimal anisotropic meshes generated using an adjoint-based method that incorporates the mesh anisotropy through an expensive sampling procedure.[9,17] The goal is to predict the optimal mesh for a given cost only using the *a priori* information. The network is trained on subsonic and transonic aerodynamic simulations over airfoils and is tested on unseen flow conditions. We expect the model to guide optimal mesh generation in practical simulations or to generate favorable initial meshes to accelerate standard adjoint-based adaptation procedures.

The outline for the remainder of this paper is as follows. Section II presents the numerical approach used in this work, a discontinuous finite-element discretization. Section III summarizes output-based error estimation, which is used to drive the adaptation algorithms presented in Section IV. Section V introduces the new machine-learning approach for mesh adaptation. Finally, Section VI presents results of the training and deployment of the neural network, and Section VII concludes with a summary and a discussion of future directions.

II. Equations and Discretization

II.A. Governing Equations

The flow is governed by the Navier–Stokes equations, which in the case of the steady-state conditions assumed in the present work, are given by

$$\nabla \cdot \vec{\mathbf{F}}(\mathbf{u}, \nabla \mathbf{u}) + \mathbf{S}(\mathbf{u}, \nabla \mathbf{u}) = \mathbf{0}, \quad \vec{\mathbf{F}} = \vec{\mathbf{F}}^{i}(\mathbf{u}) - \vec{\mathbf{F}}^{v}(\mathbf{u}, \nabla \mathbf{u}), \tag{1}$$

where $\mathbf{u}(\vec{x}) \in \mathbb{R}^s$ is the conservative state vector, $\vec{x} \in \mathbb{R}^d$ is the spatial coordinate, d is the number of space dimensions, $\vec{\mathbf{F}}^{i}$ and $\vec{\mathbf{F}}^{v}$ are the inviscid and viscous fluxes, respectively, and $\mathbf{S}$ is a source term associated with turbulence modeling closure equations, in this work RANS-SA.[18]

II.B. Discretization

To discretize the differential equation in Eq. 1, a discontinuous Galerkin (DG) finite-element method is used.[19–21] DG, as a finite-element method, approximates the state $\mathbf{u}$ in functional form using linear combinations of basis functions on each element. No continuity constraints are imposed between adjacent elements. Denoting by T_h the set of N_e elements in a non-overlapping tessellation of the domain Ω, the state on element

e, Ω_e, is approximated as

$$\mathbf{u}_h(\vec{x}(\vec{\xi}))\Big|_{\Omega_e} = \sum_{n=1}^{N_p} \mathbf{U}_{en}\phi_{en}(\vec{x}(\vec{\xi})). \tag{2}$$

In this equation, N_p is the number of basis functions per element, $\mathbf{U}_{en}$ is the vector of s coefficients for the n^{th} basis function on element e: $\phi_{en}(\vec{x}(\vec{\xi}))$, and s is the state rank. $\vec{x}$ denotes the global coordinates, and $\vec{\xi}$ denotes the reference-space coordinates in a master element. Formally, $\mathbf{u}_h \in \boldsymbol{\mathcal{V}}_h = [\mathcal{V}_h]^s$, where, if the elements are not curved, $\mathcal{V}_h = \{u \in L_2(\Omega) : u|_{\Omega_e} \in \mathcal{P}^p \ \forall \Omega_e \in T_h\}$, and $\mathcal{P}^p$ denotes polynomials of order p on the element. With the spatial discretization described above, the governing equations can be written in an abbreviated form as

$$\mathbf{R}_h(\mathbf{U}_h) = \mathbf{0}, \tag{3}$$

where $\mathbf{R}$ is the discrete nonlinear spatial residual vector in the space $\boldsymbol{\mathcal{V}_h}$.

III. Output Error Estimation

In a CFD simulation, the numerical error in an output of interest results from discretization errors. To adapt the mesh, discrete adjoint solutions can be used to estimate and localize the output error.[22]

III.A. Adjoint Evaluation

Consider an output that is a function of the discrete state vector,

$$J \equiv J(\mathbf{U}),. \tag{4}$$

The discrete adjoint, $\boldsymbol{\Psi}$, represents the sensitivity of the output to perturbations in the spatial residual, $\mathbf{R}$. To derive the adjoint equations, a Lagrangian is defined as

$$\mathcal{L} = J + \boldsymbol{\Psi}^T\mathbf{R}. \tag{5}$$

Substituting (3) into (5) and requiring stationarity of the Lagrangian with respect to the permissible state variations, $\delta\mathbf{U}$, gives the adjoint differential equation

$$\frac{\partial J}{\partial \mathbf{U}} + \boldsymbol{\Psi}^T\frac{\partial \mathbf{R}}{\partial \mathbf{U}} = \mathbf{0}. \tag{6}$$

III.B. Error Estimation

The adjoint can be used to evaluate the error in the output of interest through the adjoint-weighted residual.[23] Let $\mathbf{U}_H$ be the approximate primal solution obtained from the current approximation space denoted by subscript H and $\boldsymbol{\Psi}_h^T$ be the adjoint in the fine space, denoted by h. The error in the output is defined as

$$\delta J = J_H(\mathbf{U}_H) - J_h(\mathbf{U}_h) \approx \frac{\partial J_h}{\partial \mathbf{U}_h}\delta\mathbf{U}_h \approx -\boldsymbol{\Psi}_h^T\mathbf{R}_h(\mathbf{U}_h^H), \tag{7}$$

where $\delta\mathbf{U}_h = \mathbf{U}_h^H - \mathbf{U}_h$ is the primal error in the state and $\mathbf{U}_h^H$ is the injected solution from space H to h. The exact adjoint, which is unavailable, is approximated in a finer space by increasing the degrees of freedom in the spatial discretization through an order increment.

IV. Mesh Adaptation

The error estimate developed in the previous section can be localized to elemental contributions and provides information about how to adapt the computational mesh. However, as just one scalar quantity per element, the error indicator is not sufficient to provide information about mesh anisotropy. This information comes from a mesh optimization procedure, MOESS,[9,17] which optimizes the computational mesh in order to minimize the output error at a prescribed computational cost. The approach iteratively determines the optimal change in the mesh metric field given a prescribed metric-cost relationship and a sampling-inferred metric-error relationship.

IV.A. Metric-Based Meshing

A Riemannian metric field, $\mathcal{M}(\vec{x})$, is a field of symmetric positive definite (SPD) tensors that can be used to encode information about the desired size and stretching of a computational mesh. At each point in physical space, $\vec{x}$, the metric tensor $\mathcal{M}(\vec{x})$ provides a "yardstick" for measuring the distance from $\vec{x}$ to another point infinitesimally far away, $\vec{x} + \delta\vec{x}$. After choosing a Cartesian coordinate system and basis for physical space, $\mathcal{M}$ can be represented as a $d \times d$ SPD matrix. The set of points at unit metric distance from $\vec{x}$ is an ellipse: eigenvectors of $\mathcal{M}$ give directions along the principal axes, while the length of each axis (stretching) is the inverse square root of the corresponding eigenvalue. The aspect ratio is the ratio of the largest stretching magnitude to the smallest.

A mesh that conforms to a metric field is one in which each edge has the same length, to some tolerance, when measured with the metric. An example of a two-dimensional metric-conforming mesher is the Bi-dimensional Anisotropic Mesh Generator (BAMG),[24] and this is used to obtain the results in the present work. BAMG generates a mesh given a metric field, which is specified at nodes of a background mesh – the current mesh in an adaptive setting. The optimization determines changes to the current, mesh-implied, metric, $\mathcal{M}_0(\vec{x})$. Affine-invariant[25] changes to the metric field are made via a symmetric step matrix, $\mathcal{S} \in \mathbb{R}^{d\times d}$, according to

$$\mathcal{M} = \mathcal{M}_0^{\frac{1}{2}} \exp(\mathcal{S}) \mathcal{M}_0^{\frac{1}{2}}. \tag{8}$$

Note that $\mathcal{S} = 0$ leaves the metric unchanged, while diagonal values in $\mathcal{S}$ of $\pm$ 2 log2 halve/double the metric stretching sizes.

IV.B. Error Convergence Model

The mesh optimization algorithm requires a model for how the error changes as the metric changes. We consider one element, Ω_e, with a current error $\mathcal{E}_{e0}$ and a proposed metric step matrix of $\mathcal{S}_e$. The error on Ω_e following refinement with this step matrix is given by

$$\mathcal{E}_e = \mathcal{E}_{e0} \exp[\mathrm{tr}(\mathcal{R}_e \mathcal{S}_e)], \tag{9}$$

where $\mathcal{R}_e$ is a symmetric rate tensor. The total error over the mesh is the sum of the elemental errors, $\mathcal{E} = \Sigma_{e=1}^{N_e} \mathcal{E}_e$.The rate tensor, $\mathcal{R}_e$, is determined separately for each element through a sampling procedure.[17]

IV.C. Cost Model

To measure the cost of refinement, we use degrees of freedom, DOF, which on each element just depends on the approximation order p, assumed constant over the elements. By (8) and properties of the metric tensor, when the step matrix $\mathcal{S}_e$ is applied to the metric of element e, the area of the element decreases by $\exp[\frac{1}{2}\mathrm{tr}(\mathcal{S}_e)]$. Equivalently, the number of new elements, and hence degrees of freedom, occupying the original area Ω_e increases by this factor. So the elemental cost model is

$$C_e = C_{e0} \exp\left[\frac{1}{2}\mathrm{tr}(\mathcal{S}_e)\right], \tag{10}$$

where C_{e0}=DOF$_{e_0}$ is the current number of degrees of freedom on element e. The total cost over the mesh is the sum of the elemental costs, $C = \Sigma_{e=1}^{N_e} C_e$.

IV.D. Metric Optimization Algorithm

Given a current mesh with its mesh-implied metric, $\mathcal{M}_0(\vec{x})$, elemental error indicators, $\mathcal{E}_{e0}$, and elemental rate tensor estimates, $\mathcal{R}_e$, the goal of the metric optimization algorithm is to determine the step matrix field, $\mathcal{S}(\vec{x})$, that minimizes the error at a fixed cost. The step matrix field is approximated by values at the mesh vertices, $\mathcal{S}_v$, which are arithmetically-averaged to adjacent elements:

$$\mathcal{S}_e = \frac{1}{|V_e|} \sum_{v \in \mathrm{V}_e} \mathcal{S}_v, \tag{11}$$

where V_e is the set of vertices ($|V_e|$is the number of them) adjacent to element e. The optimization problem is to determine $\mathcal{S}_v$ such that the total error $\mathcal{E}$ is minimized at a prescribed total cost $\mathcal{C}$. First-order optimality conditions require derivatives of the error and cost with respect to $\mathcal{S}_v$. We note that the cost only depends on the trace of the step matrix; i.e. the trace-free part of $\mathcal{S}_e$ stretches an element but does not alter its area. We therefore separate the vertex step matrices into trace ($s_v\mathcal{I}$) and trace-free ($\tilde{\mathcal{S}}_v$) parts, with $\mathcal{I}$ the identity tensor,

$$\mathcal{S}_v = s_v\mathcal{I} + \tilde{\mathcal{S}}_v \tag{12}$$

The optimization algorithm is then the same as presented by Yano:[9]

1. Given a mesh, solution, and adjoint, calculate $\mathcal{E}_e$, $\mathcal{C}_e$, $\mathcal{R}_e$ for each element e.
2. Set $\delta s = \delta s_{\max}/n_{\text{step}}$, $\mathcal{S}_v$=0.
3. Begin loop: $i = 1...n_{\text{step}}$
   (a) Calculate $\mathcal{S}_e$ from (11), $\frac{\partial \mathcal{E}_e}{\partial \mathcal{S}_e}$ from (9) and $\frac{\partial \mathcal{C}_e}{\partial \mathcal{S}_e}$ from (10).
   (b) Calculate derivatives of $\mathcal{E}$ and $\mathcal{C}$ with respect to s_v and $\tilde{\mathcal{S}}_v$.
   (c) At each vertex form the ratio, $\lambda_v = \frac{\partial \mathcal{E}/\partial s_v}{\partial \mathcal{C}/\partial s_v}$ and
      - Refine the metric for 30% of the vertices with the largest $|\lambda_v|$:$\mathcal{S}_v = \mathcal{S}_v + \delta s\mathcal{I}$
      - Coarsen the metric for 30% of the vertices with the smallest $|\lambda_v|$:$\mathcal{S}_v = \mathcal{S}_v - \delta s\mathcal{I}$
   (d) Update the trace-free part of $\mathcal{S}_v$ to enforce stationarity with respect to shape changes at fixed area: $\mathcal{S}_v = \mathcal{S}_v + \delta s(\partial\mathcal{E}/\partial\tilde{\mathcal{S}}_v)/(\partial\mathcal{E}/\partial s_v)$.
   (e) Rescale $\mathcal{S}_v \to \mathcal{S}_v + \beta\mathcal{I}$, where β is a global constant calculated from (10) to constrain the total cost to the desired dof value: $\beta = \frac{2}{d}\log\frac{C_{\text{target}}}{C}$, where C_{target} is the target cost.

Note, λ_v is a Lagrange multiplier in the optimization. It is the ratio of the marginal error to marginal cost of a step matrix trace increase (i.e. mesh refinement). The above algorithm iteratively equidistributes λ_v globally so that, at optimum, all elements have the same marginal error to cost ratio. Constant values that work generally well in the above algorithm are $n_{\text{step}} = 20$ and $\delta s_{\max} = 2\log 2$. In practice, the mesh optimization and flow/adjoint solution are performed several times at a given target cost, C_{target}, until the error stops changing. Then the target cost is increased to reduce the error further if desired.

IV.E. Machine-Learning Based Mesh Adaptation

As an alternative to the adjoint-based mesh adaptation process, a data-driven method is developed to improve the efficiency of non-uniform anisotropic mesh generation compared with existing approaches. Optimum mesh generation for CFD simulations requires information about the elemental size as well their anisotropy, in the CFD domain. The anisotropy information becomes more relevant at higher Reynolds and Mach numbers with many directional features such as boundary layers, wakes, and shocks. In MOESS, both the primal and adjoint solutions are combined via sampling of the adjoint-weighted residual to produce the most efficient element size and anisotropy distribution. This information guides the movement of mesh vertices so that each element is as close as possible to its ideal size and shape. As an alternative to determining the ideal size and anisotropy information from MOESS, we present an approach that uses a neural network to determine the metric from relevant features such as the flow conditions and relative position in the computational domain.

V. Neural Network Training

The neural network is trained using meshes adapted for a variety of flow conditions and angles of attack over a NACA 0012 airfoil. All cases are two-dimensional and use the Reynolds-averaged Navier-Stokes equation. The aerodynamic lift output is considered for error estimation and mesh adaptation. Adapted meshes are generated at a chosen target degree-of-freedom cost using MOESS, at a solution approximation order of $p = 2$. The Riemannian metric, described in the previous subsection, stores the ideal size, stretching and orientation of the mesh elements in a single matrix entity and is used to train the network. The metric is a general entity that can be used in any adaptation process, independent of how it is constructed and

what characteristics the user wants to achieve through the adaptation process. As the metric of the adapted mesh is used to train the network, the number/distribution of elements in the initial mesh is irrelevant and any coarse mesh can be used to start the adaptation. In this work, a common initial mesh generated using BAMG,[24] is used for all test cases for simplicity. The number of elements in the final adapted mesh is important as all the adapted meshes are optimized for the the final number of DOFs. Keeping the total DOFs in the optimized mesh the same helps make the training process easier but also restricts the output of the network. In the present study, networks are only trained for a constant total DOF.

The neural network uses the logarithm of the Reynolds number, the angle of attack, and the Mach number as inputs to the parameter network and the normalized centroidal position of each element and the normalized wall distance as inputs to the spatial network to predict the elemental metric of the optimized mesh. The centroidal position vector of each element is normalized by the maximum distance in both dimensions. The wall distance, evaluated by averaging the wall distance of all the nodes of the element, is normalized by the maximum achievable value of the wall distance. To capture the wide range of values in the metric of the elements, the matrix logarithm of the elemental metric is used as the output of the network.

V.A. Neural Network Architecture

A fully connected multi-layer perceptron neural network, as shown in Figure 1 is constructed for training. The neural network consists of two individual networks, which initially process the fluid flow parameters and the spatial position of the elements separately. The parameter and spatial network have two hidden layers, consisting of 10 neurons each, between the input layer and the output layer. The outputs of the parameter and the spatial network are then combined by normalizing them and multiplying them. This serves an an input to the final network which has a single hidden layer of 100 neurons. Within each hidden layer, the input vectors are multiplied by a weight matrix before adding a bias vector. An activation function is then applied to the result to give the output of the hidden layer. The map from the input to the hidden layer involves an entry-wise rectified linear-unit (ReLU) activation function $\sigma(x) = \max(0, x)$, whereas no activation function is used for the output layer calculation. The parameters associated with the network consist of the weights and biases,

$$\mathbf{W}_i \in \mathbb{R}^{n_i \times n_{i-1}}, \quad \mathbf{b}_i \in \mathbb{R}^{n_i}, \tag{13}$$

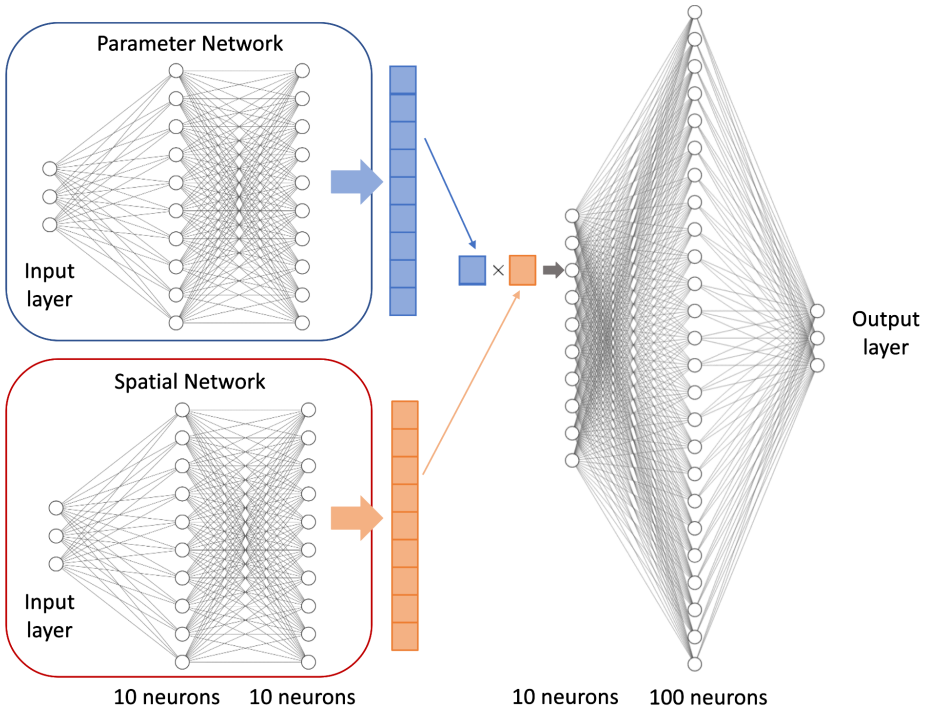


Figure 1: Structure of the artificial neural networks used to predict element anisotropy and sizing.

where n_i is the number of neurons in layer i.

The values of these parameters are determined using an optimization procedure, the adaptive moment (Adam) estimation algorithm in TensorFlow,[26] that minimizes the mean squared error loss function between predicted and actual output layer values. The actual values come from training data, which are obtained from the meshes adapted using MOESS on prototypical cases.

Each element in an adapted mesh serves as a training data point. These data points are split 80%/20% into training and validation categories. The training data are used to drive the optimization, whereas the validation data are used to monitor the loss on untrained data. The training data are broken into mini-batches of size 100 for the optimizer, and the learning rate is set to .005. Prior to training, the weights and biases are initialized randomly from a unit normal distribution. Several tens of thousands of optimization iterations typically lead to a stabilization of the mean-squared error, as shown in Figure 2. Typically, an order of magnitude drop in the loss is observed, without a significant difference between training and test data loss. This indicates that the data are not being over-fitted, which would be difficult to do with the small neural network size presently considered. Furthermore, the results of the training were not found to be overly sensitive to the choices of the mini-batch size, learning rate, or the initialization.

VI. Results

This section presents results obtained from implementing the trained neural network to generate optimized starting meshes for various flow cases. The meshes obtained from the neural network are compared to meshes obtained using MOESS. In all cases, metric-based re-meshing is performed using BAMG.

VI.A. Varying Reynolds number and angle of attack

The first case that is tackled in this study is the optimal mesh generation for a two-dimensional steady-state flow over a NACA 0012 airfoil for a variety of Reynolds numbers and angles of attack at a constant Mach number. Two Mach numbers, one in the subsonic regime, $M = 0.25$, and one in the transonic regime, $M = 0.75$, are used in this case. The airfoil is located in the center of the domain, the boundary of which consists of a square box which spans from $[-100c, 100c]$ in both dimensions. The fluid flow is simulated

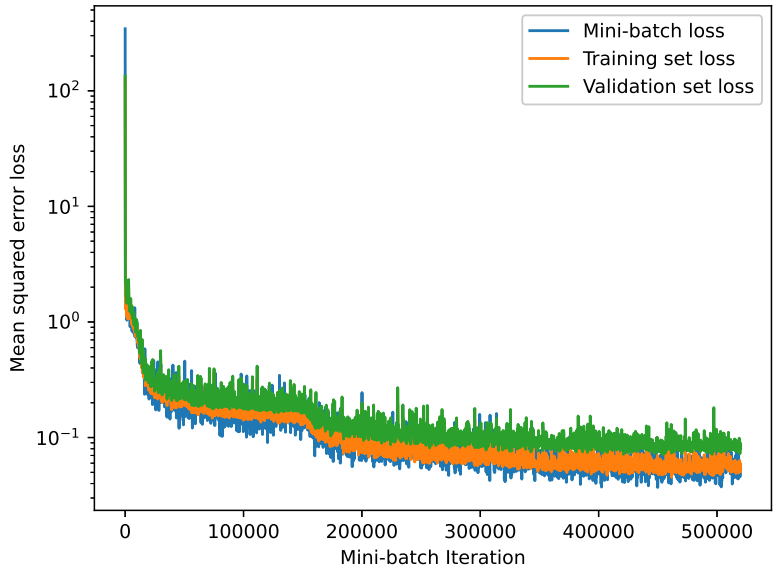
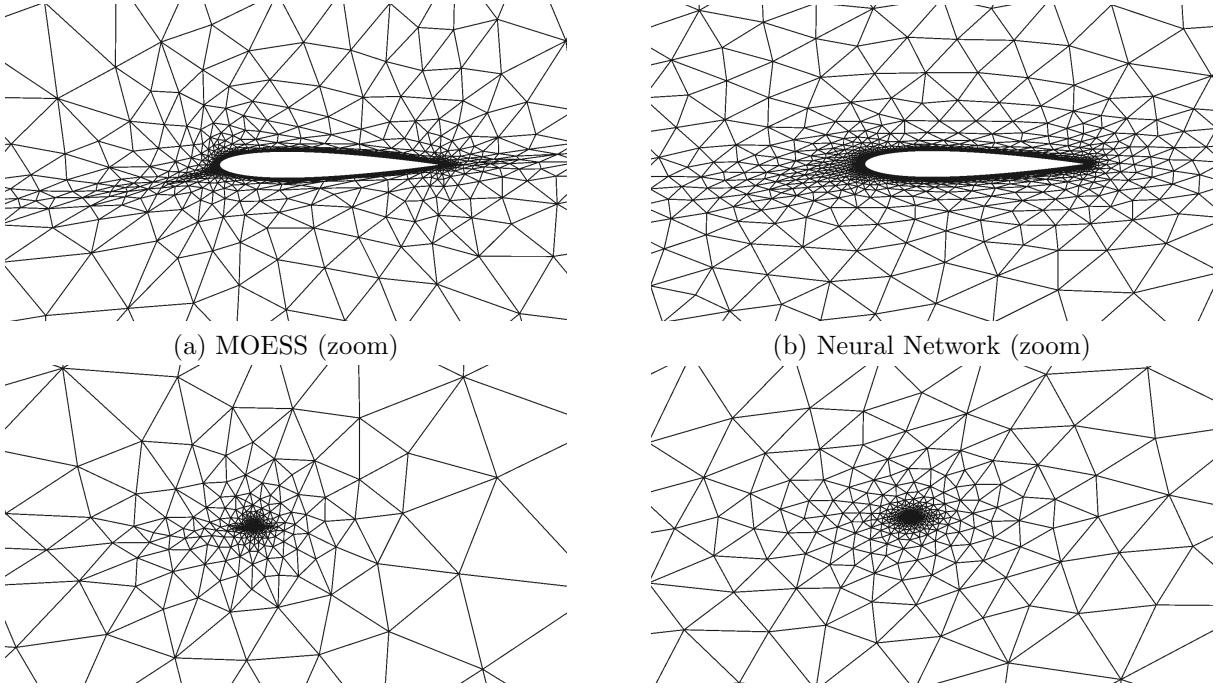


Figure 2: Neural network training loss history, using a 80% training, 20% validation split.

using a RANS solver with the SA turbulence model.[27] For training and testing purposes, 100 optimized meshes are generated for various flow conditions, where $Re \in [10^6, 10^7]$ and angle of attack, $\alpha \in [0, 5]^\circ$, and these meshes are obtained using MOESS. To sample the parameter space, Latin hypercube sampling is used in this work. The data from the various flow cases are randomized and split 80%/20% into training and validation categories. For each training case, starting from a coarse initial mesh with 500 triangular elements, the meshes are subjected to 10 cycles of mesh adaptation with a growth factor of $G_f = 1.5$ and a constraint on the total degree of freedom, DOF=50,000. The total number of degree of freedom after each adaptive iteration is set by the growth factor. The resulting adapted meshes have approximately 8500 triangular elements of approximation order $p = 2$.

A Riemannian metric, generated using the flow and adjoint solution in MOESS or predicted using the neural network, guides the mesh adaptation process. In the neural network, the predicted metric is obtained by taking the matrix exponent of the output obtained from the network. The predicted metric is then provided to BAMG for mesh generation and using the same set of growth factor and number of adaptive iterations as MOESS, the adapted mesh is obtained. Note, in the neural-network approach, no primal or adjoint solutions are needed during the course of adaptation, as the metric only depends on the parameter and spatial network inputs shown in Figure 1, and not on the state or adjoint. To test the prediction capability of the neural network, adapted meshes and output convergence in the adapted meshes are compared between neural network and MOESS for the flow conditions in the validation set. Figure 3 and Figure 4 compare adapted meshes for two flow condition in the validation set at $M = 0.25$. For both of the flow conditions in the subsonic regime, the network is able to reproduce meshes with good detail. The network is able to successfully produce the anisotropic elements above and below the airfoil, thereby, resolving the boundary layer. The element distribution away from the airfoil also looks similar to that of the optimized meshes obtained from MOESS, where large isotropic elements are used.

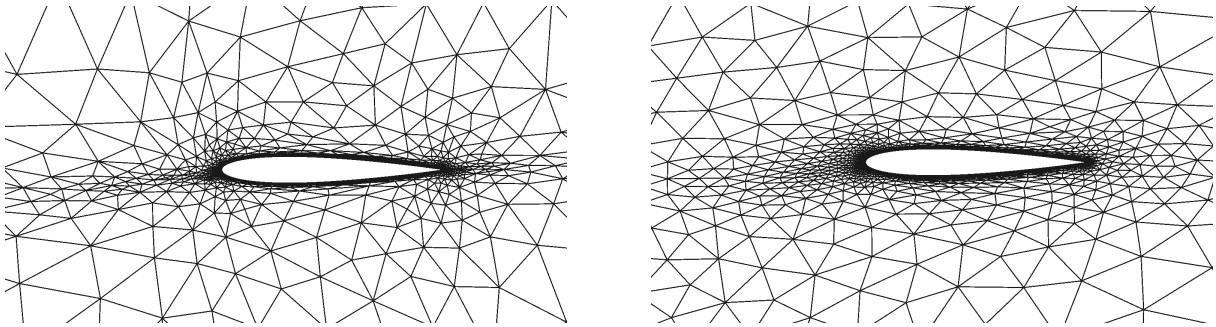
Figure 5 compares the convergence of the lift coefficient for which both the meshes have been adapted. The final mesh obtained after 10 cycles of MOESS serves as the reference mesh. The output convergence in the meshes adapted using the network show, on average, a reduction in the output error after every



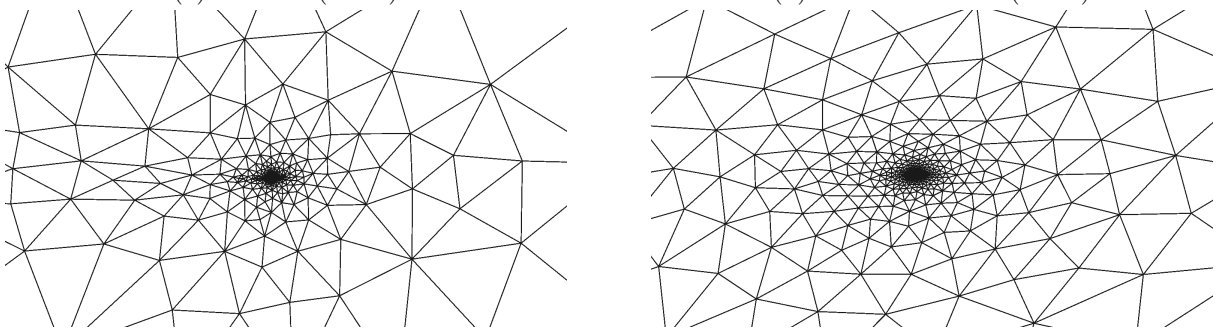
(a) MOESS (zoom) (b) Neural Network (zoom)

(c) MOESS (d) Neural Network

Figure 3: Comparison of element distribution in adapted meshes obtained using MOESS and the neural network for a flow at $M = 0.25$, $\alpha = 4.6^\circ$ and $Re = 3.5 \times 10^6$

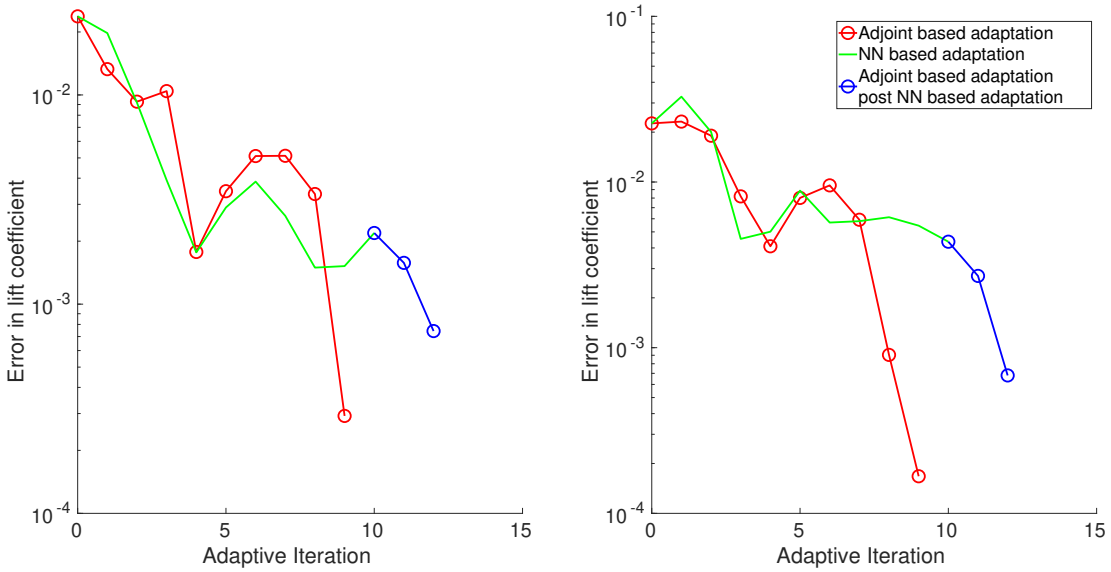


(a) MOESS (zoom) (b) Neural Network (zoom)



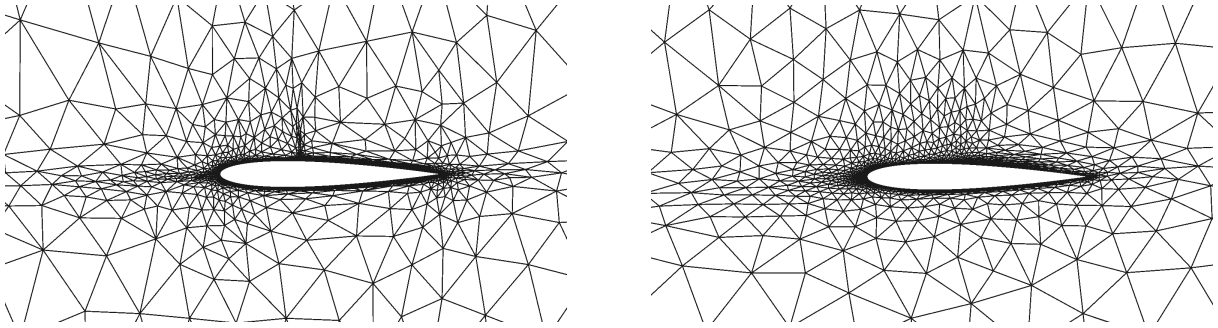
(c) MOESS (d) Neural Network

Figure 4: Comparison of element distribution in adapted meshes obtained using MOESS and the neural network for a flow at $M = 0.25$, $\alpha = 2.65$ and $Re = 4 \times 10^6$.



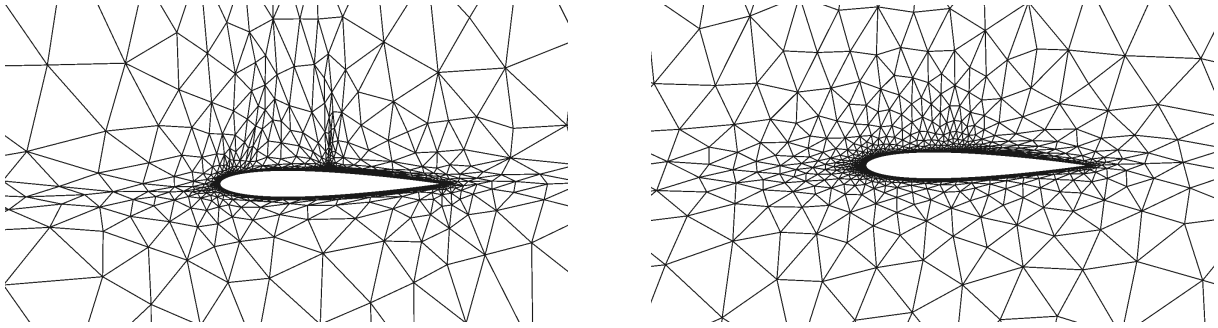
(a) $\alpha = 2.65°$ and $Re = 4 \times 10^6$ (b) $\alpha = 4.6°$ and $Re = 3.5 \times 10^6$

Figure 5: Comparison of output convergence in adapted meshes obtained using MOESS and the neural network for a flow at $M = 0.25$.



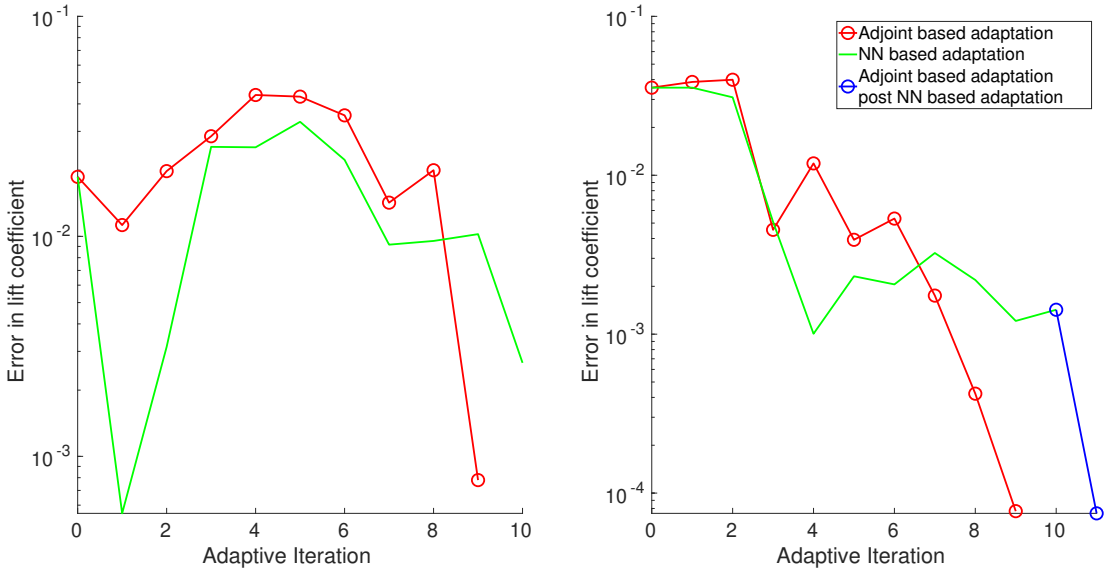
(a) MOESS (zoom) (b) Neural Network (zoom)

Figure 6: Comparison of element distribution in adapted meshes obtained using MOESS and the neural network for a flow at $M = 0.75$, $\alpha = 1.06$ and $Re = 3.5 \times 10^6$.



(a) MOESS (zoom) (b) Neural Network (zoom)

Figure 7: Comparison of element distribution in adapted meshes obtained using MOESS and the neural network for a flow at $M = 0.75$, $\alpha = 2.73$ and $Re = 9.47 \times 10^6$ $\alpha = 0$.



(a) $\alpha = 2.73$ and $Re = 9.47 \times 10^6$ (b) $\alpha = 1.06$ and $Re = 3.5 \times 10^6$

Figure 8: Comparison of output convergence in adapted meshes obtained using MOESS and the neural network for a flow at $M = 0.75$.

adaptation cycle and give comparable accuracy relative to the meshes obtained from MOESS. The network doesn't outperform the meshes obtained using MOESS as the error in training also affects the quality of the output meshes, thereby affecting the output error. The meshes obtained from the network also mirror the distribution and anisotropy in elements and are close to being optimal for the output evaluation , given the constraint on the DOF. In the cases where the meshes obtained from the network are sub-optimal, a few cycles of adjoint based mesh adaptation with the output mesh from the network serving as the starting mesh can be used to further improve the meshes, as shown by the blue curves in Figure 5. The meshes can be improved for the target DOF's by choosing a growth factor, $G_f = 1$ and reshuffling the elements or for a higher growth factor for obtaining optimal meshes for larger DOF's. The number of MOESS adaptive iterations needed to achieve higher accuracy will be fewer than the case when a network is used initially as compared to just using MOESS. This can be observed in the blue curve in Figure 5, where only two adaptive iteration of MOESS are required on the final mesh obtained by the neural network to produce an optimal mesh with similar errors. A growth factor of one is used for the MOESS adaptive shown in the blue curve.

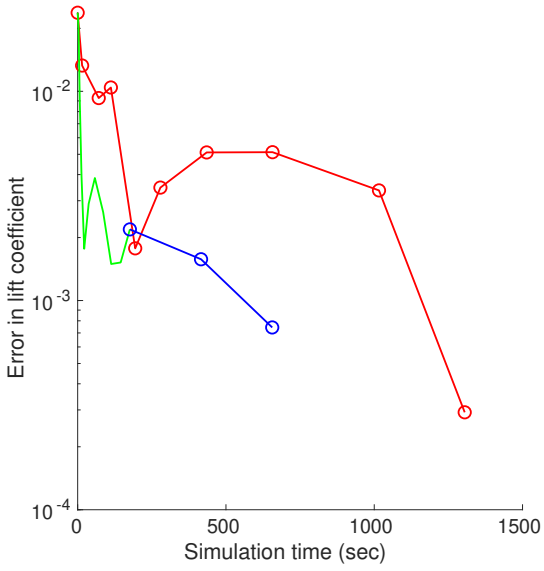
The adapted meshes for the transonic case at $M = 0.75$ show similar results, as shown in Figure 6 and Figure 7. The adapted meshes obtained from the network are able to successfully resolve the boundary layer and stagnation streamline, however, the shock is under resolved by the network. The lack of resolution around the shock compared to the boundary layer results in the network not resolving the shock accurately. The output convergence is unaffected by the under resolution of the shocks as the meshes obtained for the network are able to give comparable accuracy to that obtained form MOESS, as shown in Figure 8. Figure 9 compares the simulation times for the two mesh adaptation strategies for both the subsonic and transonic cases. Mesh adaptation using the network avoids the solution and adjoint solve needed in MOESS resulting in a 10x-15x savings in computational cost. The only time consuming component of the mesh adaption using the neural network is from the mesh generator, BAMG.

VI.B. Varying Mach number, Reynolds number and angle of attack

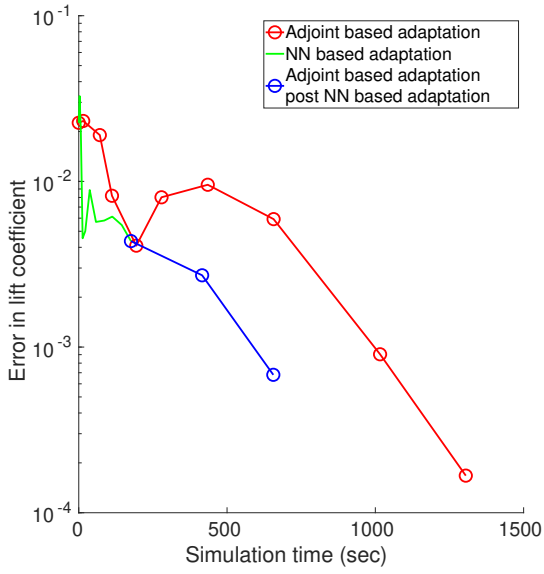
The second case that is tackled in this study is the optimal mesh generation for a two dimensional steady state flow over a NACA 0012 airfoil for a variety of Mach numbers, Reynolds numbers and angles of attack at a constant Mach number. For training and testing purposes, 100 optimized meshes are generated for various flow conditions, where $M \in [0.25, 0.8]$, $Re \in [10^6, 10^7]$ and angle of attack, $\alpha \in [0, 5]^\circ$ are obtained using MOESS. Similar initial meshes and mesh adaptation parameters are used in this case, as described in the first case.

Figure 10 and Figure 11 compare adapted meshes for a transonic and subsonic flow condition respectively, in the validation set. While the adapted meshes from the network perform well in the subsonic regime, they struggle to resolve the shock flow feature, stagnation streamlines and the streamlines exiting from the trailing edge, in the transonic regime. This also results in poor output convergence for the transonic cases as shown in Figure 12. The reason for this poor optimal meshes in the transonic regime is due to the lack of transonic cases used to train the network. Shock features are present in $M \in [0.5, 0.8]$ at high angles of attack, which comprise of only 15% of the total training data set. Secondly, in the case of shocks the ratio of elements used to resolve the shock compared to the boundary layer is low. This leads to less emphasis on the shock position and shock strength by the network. As the flow features targeted by mesh adaptation depend strongly on the Mach number regime, training the networks for subsonic and transonic cases separately can leads to better optimal meshes. Separating the network and training for obtaining optimal meshes only in the transonic regime where $M \in [0.5, 0.8]$ leads to much better output convergence, as shown in Figure 13. This better convergence in the outputs is observed due to better resolution of the stagnation streamlines and the wake but not better shock capturing.

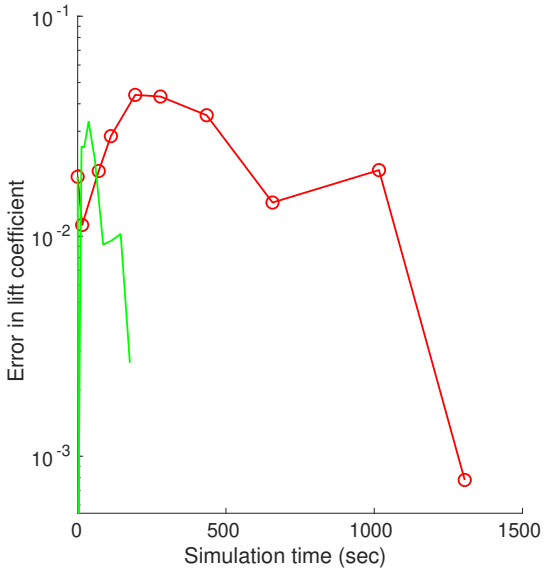
The benefits of the neural network are more pronounced when the mesh adaptation process begins with coarser initial meshes. The lack of convergence in the steady state primal and adjoint solvers due to the coarseness of the mesh affects the mesh adaptation in MOESS, however, the neural network mesh adaptation process is unaffected by the initial mesh.



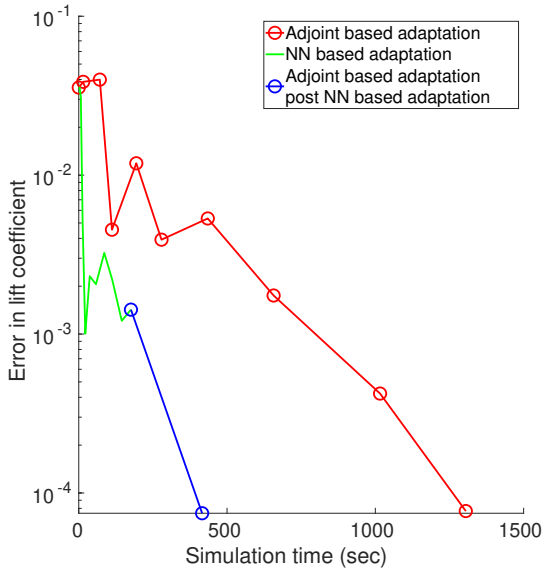
(a) $M = 0.25, \alpha = 2.65$ and $Re = 4 \times 10^6$



(b) $M = 0.25, \alpha = 1.06$ and $Re = 3.5 \times 10^6$

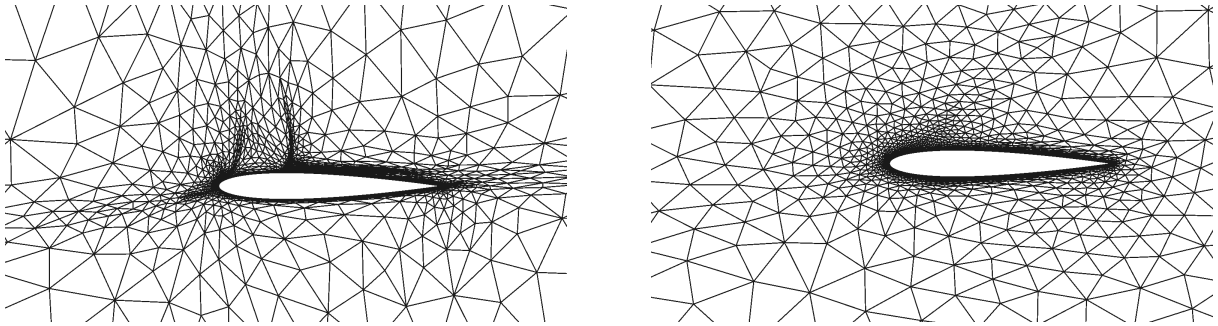


(c) $M = 0.75, \alpha = 2.73$ and $Re = 9.47 \times 10^6$



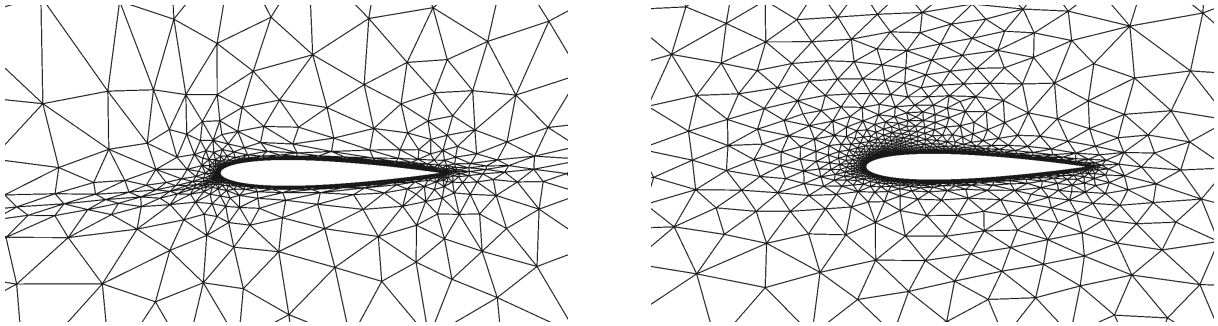
(d) $M = 0.75, \alpha = 1.06$ and $Re = 3.5 \times 10^6$

Figure 9: Comparison of the average time required for adapting a fluid mesh using MOESS and the neural network.



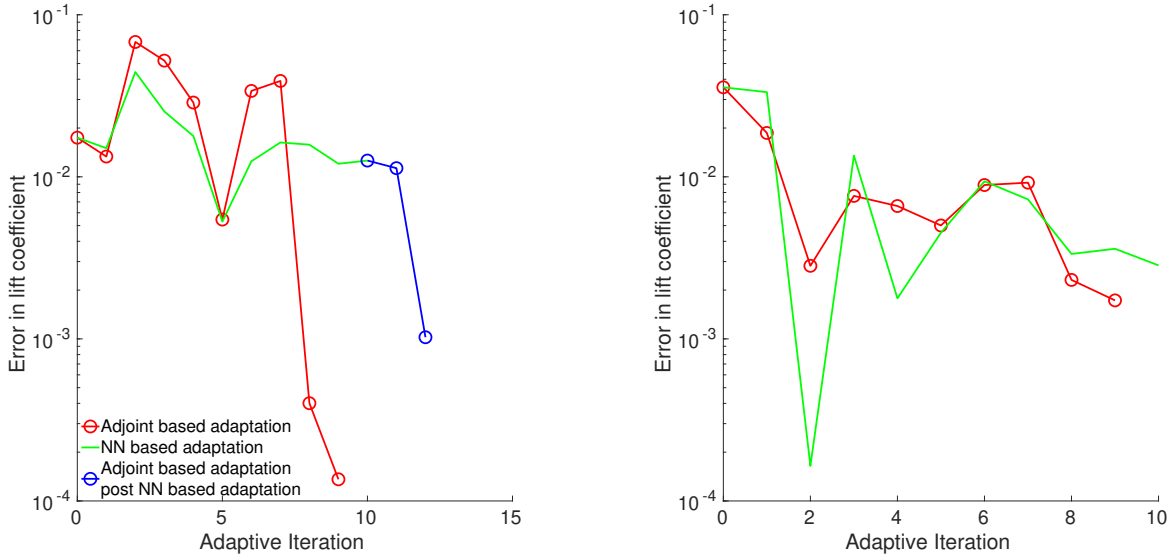
(a) MOESS (zoom) (b) Neural Network (zoom)

Figure 10: Comparison of element distribution in adapted meshes obtained using MOESS and the neural network for a flow at $M = 0.68$, $\alpha = 4.94$ and $Re = 3.05 \times 10^6$.



(a) MOESS (zoom) (b) Neural Network (zoom)

Figure 11: Comparison of element distribution in adapted meshes obtained using MOESS and the neural network for a flow at $M = 0.44$, $\alpha = 4.96$ and $Re = 7.92 \times 10^6$.



(a) $M = 0.68$, $\alpha = 4.94$ and $Re = 3.05 \times 10^6$ (b) $M = 0.44$, $\alpha = 4.96$ and $Re = 7.92 \times 10^6$

Figure 12: Comparison of output convergence in adapted meshes obtained using MOESS and the neural network. The network is trained using meshes adapted for varying Mach number, angle of attack and Reynolds number flows.

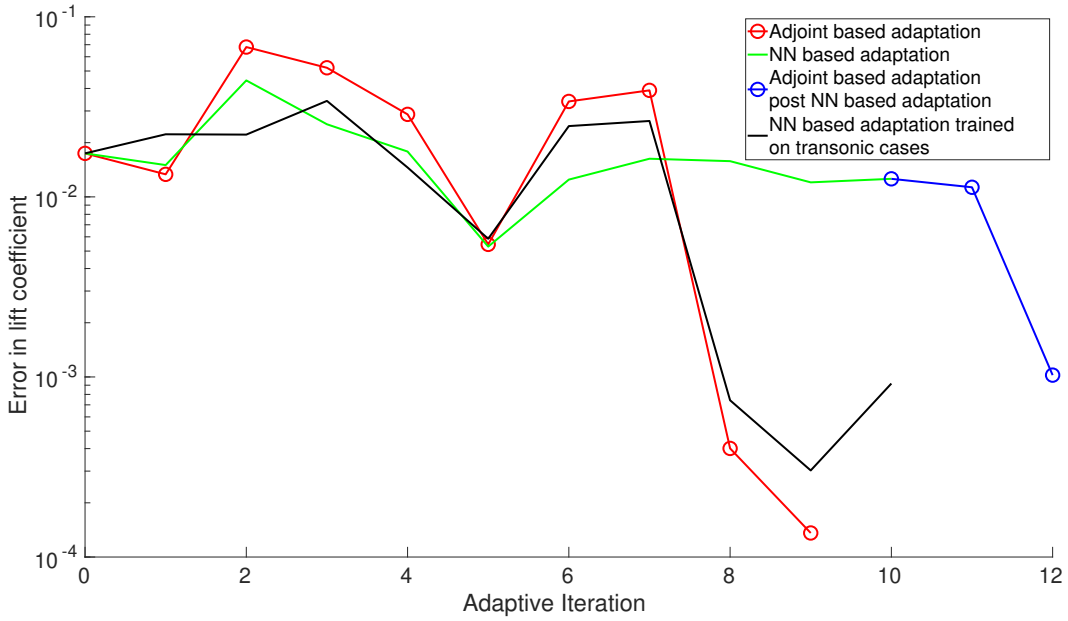
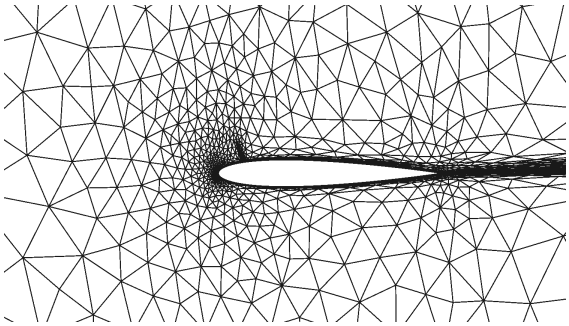


Figure 13: Comparison of output convergence in adapted meshes obtained using MOESS and two neural networks for a flow at $M = 0.68$, $\alpha = 4.94$ and $Re = 3.05 \times 10^6$. The two networks are trained using meshes adapted for varying Mach number, angle of attack and Reynolds number and varying Mach number in the transonic regime, angle of attack and Reynolds number flows.

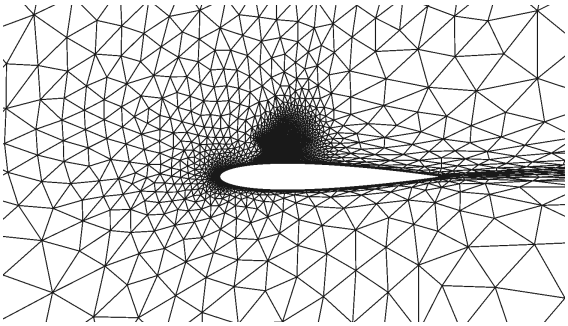
VI.C. Entropy Adjoint - Varying Mach number, Reynolds number and angle of attack

A further test to examine the capability of the network is by training it to reproduce meshes optimized using the entropy adjoint instead of the lift adjoint. The entropy-based adjoint indicator uses entropy variables to drive the mesh adaptation. The areas of the mesh that are targeted by the entropy indicator are those regions that exhibit high net production of spurious entropy. The formulation of the entropy-variable approach can be found in Doetsch et al.[28] Adapting on an output of interest using the entropy variables as adjoints constitutes output-based adaptation. Without the need for a separate system solve to obtain the entropy variables, adapting using entropy variables instead of the output-based adjoint is far less computationally expensive. However, this indicator does not disregard areas of spurious entropy generation that have no effect on a particular engineering output. This may lead to over-refinement, particularly for cases with flow discontinuities. The entropy adjoint resolves the boundary layer, the wake (extending up the boundary of the domain) and also the shock (if present). Due to a greater number of flow features that the entropy adjoint focuses on, it acts as a good litmus test and stresses the network to the maximum. A similar network as previously discussed is used for this test case. For training and testing purposes, 100 optimized meshes are generated for various flow conditions in the transonic regime, where $M \in [0.5, 0.8]$, $Re \in [10^6, 10^7]$ and angle of attack, $\alpha \in [0, 5]^\circ$ are obtained using MOESS. Figure 14 and 15 compare adapted meshes for two flow conditions in the transonic regime, from the validation set. The network is able to accurately resolve the wake and the boundary layer. However, the network struggles in predicting the shock position and strength. For this Mach number range, the shock traverses above the airfoil increasing in strength while moving towards the trailing edge, as the Mach number increases. Instead of refining around the shock location, the network refines the entire region where the shock traverses in an isotropic manner. As the Mach number increases, the refined region in the network adapted meshes moves with along the chord, mimicking the shock traversal.

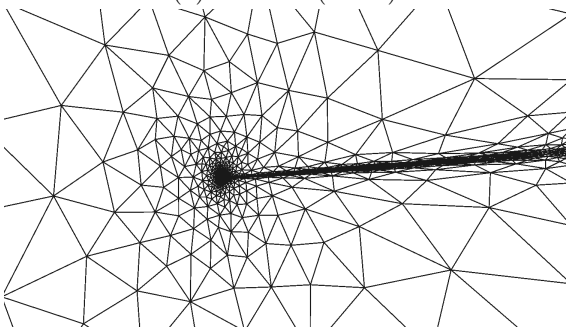
For the first two cases in this study, the lack of training cases containing shocks and less emphasis on the shock features explained the poor resolution of the shock feature in the meshes adapted by the network. However, the entropy adjoint removes both of these pitfalls and the network still struggles to resolve the shock. The number of elements used to resolve the shock, the boundary layer and the wake are comparable in the case of a meshes adapted using the entropy adjoints and 60% of the training dataset has shocks present.



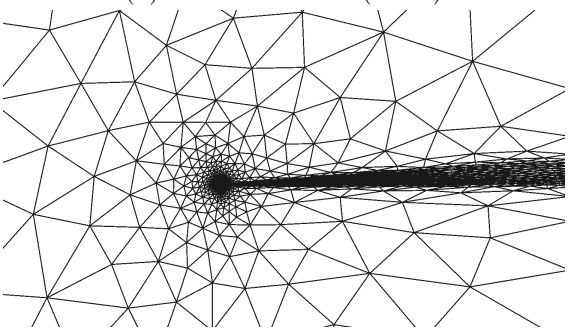
(a) MOESS (zoom)



(b) Neural Network (zoom)

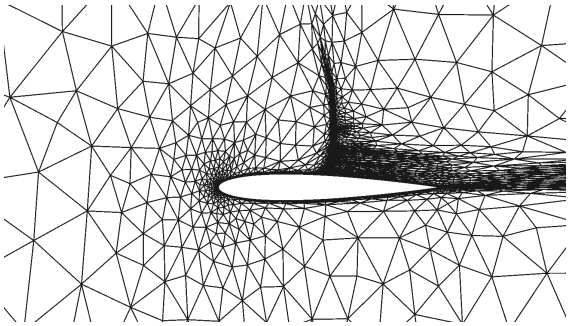


(c) MOESS

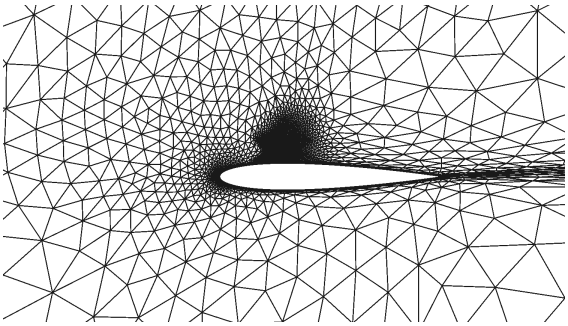


(d) Neural Network

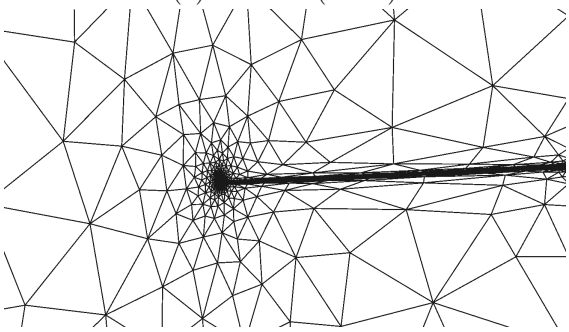
Figure 14: Comparison of element distribution in adapted meshes obtained using MOESS and the neural network for a flow at $M = 0.60$, $\alpha = 4.45$ and $Re = 4.54 \times 10^6$. Entropy adjoint is used to adapt the meshes obtained using MOESS.



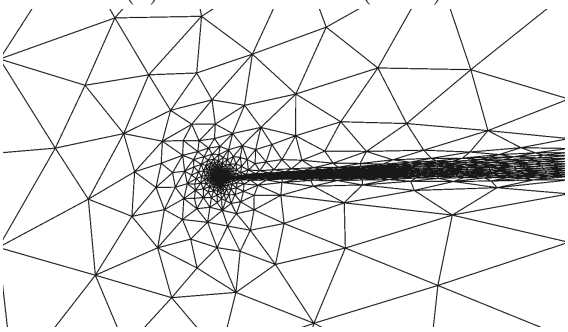
(a) MOESS (zoom)



(b) Neural Network (zoom)



(c) MOESS



(d) Neural Network

Figure 15: Comparison of element distribution in adapted meshes obtained using MOESS and the neural network for a flow at $M = 0.76$, $\alpha = 2.8$ and $Re = 6.67 \times 10^6$. Entropy adjoint is used to adapt the meshes obtained using MOESS.

Resolving the shock flow feature with the flow parameters needs more examination and different network structures are being investigated to resolve this issue. A promising avenue is to use the flow features to firstly predict the shock position and the combining this information with the existing network.

VII. Conclusion

A machine-learning approach for determining the optimal anisotropic initial meshes, in the context of an output-based adaptive solution procedure is developed. Artificial neural networks are used to predict the desired element sizing and anisotropy from flow conditions and relative position in the fluid domain. The benefits of mesh adaptation using a neural network are demonstrated for a steady state flow over a NACA 0012 airfoil, where the network is able to produce optimal starting meshes for a variety of Mach numbers, Reynolds numbers and angles of attack. Meshes optimized using the lift and the entropy adjoint are used to train the network. The meshes generated from the neural network showcase good output convergence as that obtained using adjoint-based mesh adaptation. Mesh adaptation using the machine learning approach produces optimal meshes without solving for the state or the adjoint leading to 10x-15x saving in computational time. The network is able to successfully resolve flow features such as the boundary layer, wake, stagnation streamlines. In the presence of shock features in transonic flow, the network struggles to resolve around the shock location and further investigation is needed. Extension of the machine-learning approach to generate optimal meshes for unsteady simulations is expected to give similar benefits and will be pursued in the future.

References

[1]Fidkowski, K. J. and Darmofal, D. L., "Review of Output-Based Error Estimation and Mesh Adaptation in Computational Fluid Dynamics," *American Institute of Aeronautics and Astronautics Journal*, Vol. 49, No. 4, 2011, pp. 673–694.

[2]Hartmann, R. and Houston, P., "Adaptive Discontinuous Galerkin Finite Element Methods for the Compressible Euler Equations," *Journal of Computational Physics*, Vol. 183, No. 2, December 2002, pp. 508–532.

[3]Venditti, D. A. and Darmofal, D. L., "Grid Adaptation for Functional Outputs: Application to Two-Dimensional Inviscid Flows," *Journal of Computational Physics*, Vol. 176, No. 1, Feb. 2002, pp. 40–69.

[4]Park, M. A., "Adjoint-Based, Three-Dimensional Error Prediction and Grid Adaptation," *AIAA Journal*, Vol. 42, No. 9, September 2004, pp. 1854–1862.

[5]Fidkowski, K. J. and Darmofal, D. L., "A triangular cut-cell adaptive method for high-order discretizations of the compressible Navier–Stokes equations," *Journal of Computational Physics*, Vol. 225, No. 2, August 2007, pp. 1653–1672.

[6]Wang, L. and Mavriplis, D. J., "Adjoint-based h–p adaptive discontinuous Galerkin methods for the 2D compressible Euler equations," *Journal of Computational Physics*, Vol. 228, No. 20, Nov. 2009, pp. 7643–7661.

[7]Loseille, A., Dervieux, A., and Alauzet, F., "Fully anisotropic goal-oriented mesh adaptation for 3D steady Euler equations," *Journal of Computational Physics*, Vol. 229, No. 8, April 2010, pp. 2866–2897.

[8]Nemec, M. and Aftosmis, M., "Adjoint Error Estimation and Adaptive Refinement for Embedded-Boundary Cartesian Meshes," *18th AIAA Computational Fluid Dynamics Conference*, AIAA Paper 2007-4187, June 2007.

[9]Yano, M. et al., *An optimization framework for adaptive higher-order discretizations of partial differential equations on anisotropic simplex meshes*, Ph.D. thesis, Massachusetts Institute of Technology, 2012.

[10]Nadarajah, S. and Jameson, A., "A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization," *38th Aerospace Sciences Meeting and Exhibit*, AIAA Paper 2000-667, January 2000.

[11]Kenway, G. K., Mader, C. A., He, P., and Martins, J. R., "Effective adjoint approaches for computational fluid dynamics," *Progress in Aerospace Sciences*, Vol. 110, Oct. 2019, pp. 100542.

[12]Manevitz, L., Bitar, A., and Givoli, D., "Neural network time series forecasting of finite-element mesh adaptation," *Neurocomputing*, Vol. 63, Jan. 2005, pp. 447–463.

[13]Zhang, Z., Wang, Y., Jimack, P. K., and Wang, H., "MeshingNet: A New Mesh Generation Method Based on Deep Learning," *Lecture Notes in Computer Science*, Springer International Publishing, 2020, pp. 186–198.

[14]Huang, K., Krügener, M., Brown, A., Menhorn, F., Bungartz, H.-J., and Hartmann, D., "Machine Learning-Based Optimal Mesh Generation in Computational Fluid Dynamics," *arXiv preprint arXiv:2102.12923*, 2021.

[15]Chen, G. and Fidkowski, K. J., "Output-Based Adaptive Aerodynamic Simulations Using Convolutional Neural Networks," *Computers & Fluids*, Vol. 223, April 2021.

[16]Fidkowski, K. J. and Chen, G., "Metric-based, goal-oriented mesh adaptation using machine learning," *Journal of Computational Physics*, Vol. 426, Feb. 2021.

[17]Fidkowski, K., "A local sampling approach to anisotropic metric-based mesh optimization," *54th AIAA Aerospace Sciences Meeting*, 2016, p. 0835.

[18]Allmaras, S. R. and Johnson, F. T., "Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model," *Seventh international conference on computational fluid dynamics (ICCFD7)*, Vol. 1902, Big Island, HI, 2012.

[19]Cockburn, B., Karniadakis, G. E., and Shu, C.-W., *Discontinuous Galerkin methods: theory, computation and applications*, Vol. 11, Springer Science & Business Media, 2012.

[20]Reed, W. H. and Hill, T., "Triangular mesh methods for the neutron transport equation," Tech. rep., Los Alamos Scientific Lab., N. Mex.(USA), 1973.

[21]Fidkowski, K. J., Oliver, T. A., Lu, J., and Darmofal, D. L., "p-Multigrid solution of high–order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations," *Journal of Computational Physics*, Vol. 207, 2005, pp. 92–113.

[22]Fidkowski, K. J., "Output-Based Space-Time Mesh Optimization for Unsteady Flows Using Continuous-in-Time Adjoints," *Journal of Computational Physics*, Vol. 341, No. 15, July 2017, pp. 258–277.

[23]Fidkowski, K. J. and Darmofal, D. L., "Review of Output-Based Error Estimation and Mesh Adaptation in Computational Fluid Dynamics," *American Institute of Aeronautics and Astronautics Journal*, Vol. 49, No. 4, 2011, pp. 673–694.

[24]Hecht, F., "BAMG: bidimensional anisotropic mesh generator," *User Guide. INRIA, Rocquencourt*, Vol. 17, 1998.

[25]Pennec, X., Fillard, P., and Ayache, N., "A Riemannian framework for tensor computing," *International Journal of computer vision*, Vol. 66, No. 1, 2006, pp. 41–66.

[26]Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[27]Ceze, M. A. and Fidkowski, K. J., "High-order output-based adaptive simulations of turbulent flow in two mineaviationdimensions," *AIAA Journal*, Vol. 54, No. 9, 2016.

[28]