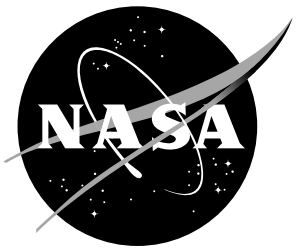
NASA–TM–20220014856



The ANOPP2 Artificial Neural Network Tool (AANNT) Reference Manual

Version 1.5

C. S. Thurman
Langley Research Center, Hampton, Virginia 23681-2199

November 2022

NASA STI Program. . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI Program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI Program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collection of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include organizing and publishing research results, distributing specialized research announcements and feeds, providing information desk and personal search support, and enabling data exchange services.

For more information about the NASA STI Program, see the following:

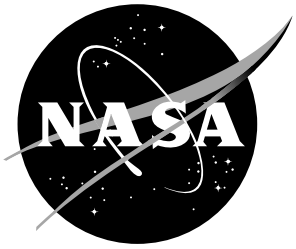
- Access the NASA STI program home page at `http://www.sti.nasa.gov`

- E-mail your question to `help@sti.nasa.gov`

- Phone the NASA STI Information Desk at 757-864-9658

- Write to:
  NASA STI Information Desk
  Mail Stop 148
  NASA Langley Research Center
  Hampton, VA 23681-2199

NASA–TM–20220014856



The ANOPP2 Artificial Neural Network Tool (AANNT) Reference Manual

Version 1.5

C. S. Thurman
Langley Research Center, Hampton, Virginia 23681-2199

National Aeronautics and
Space Administration

Langley Research Center

November 2022

Acknowledgments

This work is funded by NASA's Aeronautic Research Mission Directorate (ARMD), specifically, the Revolutionary Vertical Lift Technology (RVLT) Project in the Advanced Air Vehicles Program (AAVP).

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Abstract

This manual documents version 1.5 of the ANOPP2 Artificial Neural Network Tool (AANNT) developed at the NASA Langley Research Center Aeroacoustics Branch. The AANNT is a suite of standalone software that provide the capabilities of generating optimal artificial neural network prediction models, deploying these prediction models over arbitrary, user-defined data, and conducting sensitivity analyses over these prediction models. The AANNT has been developed for user simplicity and requires only prerequisite Python package installation and user-defined namelist files for execution.

This application programming interface (API) is part of a larger toolkit called the Aircraft NOise Prediction Program 2 (ANOPP2). The goal of ANOPP2 is to provide the ability to independently: (1) assess aircraft system noise; (2) assess aircraft component noise; and (3) evaluate aircraft noise reduction technologies and flight procedures. Additionally, ANOPP2 is designed to provide a capability for understanding the fundamental physics involved in noise generation to support experiments and flight demonstration activities.

As a component of ANOPP2, the AANNT and this document may be included as part of the ANOPP2 distribution, or they may be provided independently of that distribution.

Contents

List of Symbols

English

$\tilde{E}$	error value from an arbitrary cost function
f	general form of a functional relationship
H	number of neurons in the first hidden layer of an artificial neural network
L_1	ridge regression regularization type
L_2	lasso regression regularization type
M	airfoil camber
m	number of connections to a particular artificial neural network neuron
G	location of maximum airfoil camber
N_b	number of rotor blades
s	number of input features
P_I	induced power
P	number of neurons in the second hidden layer of an artificial neural network
Q	number of Profile-Method intervals
R_d^2	coefficient of determination
$R_{q,k}$	mean prediction model output for each interval of the Profile-Method
R	rotor radius
n	number of data samples
$T_{\mathbf{design}}$	target design thrust
t/c	airfoil thickness
$\mathbf{x}_k$	vector of input data points correspondent to the k^{th} input feature
$\mathbf{y}_j$	labeled data correspondent to the j^{th} output
z	output from a neuron in an artificial neural network hidden layer

Greek

α	multiplicative weight
β	boundaries of He initialization distribution
γ	activation function used by an artificial neural network neuron
Ω	rotor rotation rate
θ_0	collective pitch

Subscripts

$x_{\mathbf{max}}$	maximum
$x_{\mathbf{min}}$	minimum
x_q	Profile-Method interval
$x_{\mathbf{tot}}$	total

Superscripts

$\bar{x}$	averaged
$\tilde{x}$	scaled
$\hat{x}$	predicted

List of Figures

List of Tables

List of Examples

Change Log

Version 1.5.0

1. This is the initial instantiation of this reference manual.

1 Overview

This reference manual will discuss the ANOPP2 Artificial Neural Network Tool (AANNT), general Machine Learning (ML) methodology, and how Artificial Neural Networks (ANN) can be used for regression modeling of nonlinear input-output functional relationships. Generally speaking, there are a multitude of different ANN model architectures and hyperparameters that can be used to generate accurate prediction models. These elements of the ANN modeling procedure are typically determined heuristically and are problem specific; varying with the input-output relationship being modeled. In other words, the optimal ANN structure used for one problem may be different from the ANN structure of another. Because of these potential ANN model form variations as well as the common difficulties associated with creating ANNs, a user-friendly suite of tools has been developed by the author and implemented within the NASA second generation Aircraft NOise Prediction Program (ANOPP2) software. The AANNT consists of three main components: Artificial Neural Network Prediction Modeling (ANNPM), Artificial Neural Network Model Deployment (ANNMD), and Artificial Neural Network Sensitivity Analyses (ANNSA), respectively. Example inputs and outputs of the three AANNT components will be provided in this reference manual.

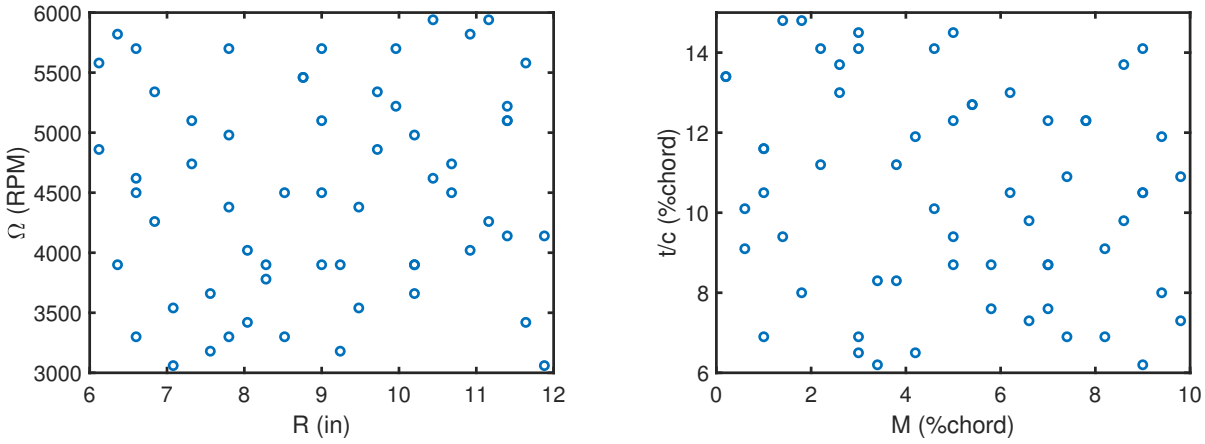
1.1 Prerequisites

To use the AANNT, the user must first supply data describing the relationship being modeled. The format of these data will be discussed in this reference manual, but, in general, these user-supplied data must contain discrete combinatorial data points of the functional inputs and the correspondent outputs, or labeled data. For example, a user may wish to model the relationship between the induced power, P_I, being generated by a rotor relative to various inputs such as the number of rotor blades, N_b, rotor rotation rate, Ω, rotor radius, R, and collective pitch angle, θ_0, similar to what was done by Thurman et al. [2]. The user would first specify the extents of the input feature space or minimum and maximum values of the input features. An example of this is shown in Table 1 where N_b is a discrete (i.e., categorical) input feature and all other input features are continuous between an upper and lower limit.

Table 1: Input feature space. (* indicates categorical input features. All other features are continuous.)

Input Feature	Range
Number of rotor blades (N_b)*	2, 3, 4
Rotor speed (Ω)	3500 RPM – 6000 RPM
Rotor radius (R)	6 in – 8 in
Target design thrust (T_{design})	1.5 lb – 3.0 lb
Airfoil camber (M)	0% – 9%
Location of maximum airfoil camber (G)	20% – 50%
Airfoil thickness (t/c)	6% – 15%
Collective pitch (θ_0)	-5° – $+5^\circ$

The user would then generate a design space consisting of combinations of input feature values within the input feature space. An example design space taken from Thurman [1] is shown in Fig. 1 to illustrate an arbitrary design space projected on a 2-D subdimension of the input feature space.



(a) R vs. Ω. (b) M vs. t/c.

Figure 1: Input data design space projected on a 2-D input feature space. Adapted from Thurman [1].

After the design space development, the user must provide labeled data at each point in the input feature space by experimental or computational methods. For example, the user would run simulations at each discrete data point to predict values of P_I correspondent to each of the input data points. A flowchart illustrating the prerequisites for using AANNT as well as the hierarchy of AANNT components is shown in Fig. 2. The generation of the input feature space and methods for developing the output data for model development are outside the scope of this reference manual, however, the reader is referred to Thurman et al. [2] and Thurman [1] for detailed discussion.

2 Machine Learning

In general, ML involves the development of functional relationships, or prediction models, using prescribed input data points ($\mathbf{x}_k, \mathbf{x}_{k+1}, ..., \mathbf{x}_s$; where s corresponds to the number of input features). The ML prediction model is 'trained' using algorithmic optimization (e.g., stochastic gradient descent) until it adequately 'learns' the underlying relationship. The main goal of any ML procedure is not only to fit the input data points, but to develop prediction models that can generalize to new data, previously unseen by the models, to make accurate predictions anywhere within the prescribed region of experimentation, or input feature space. There are two main types of learning associated with ML: unsupervised learning (i.e., descriptive) and supervised learning (i.e., predictive).

Unsupervised learning entails the development of prediction models where there are no labeled data; in other words, only the prescribed input data points are provided. Since no labeled data are provided during the training procedure, the ML task is to learn patterns, or similarities, in the input data. A common type of unsupervised learning is data clustering, where the prediction model 'clusters' the input data points based upon some learned similarity between the input data [4]. For example, provided the demographic information of shareholders in a particular company, the trained model may predict that people of a certain age group or income status are more likely to hold shares of the company. An example of this unsupervised learning problem is shown in Fig. 3

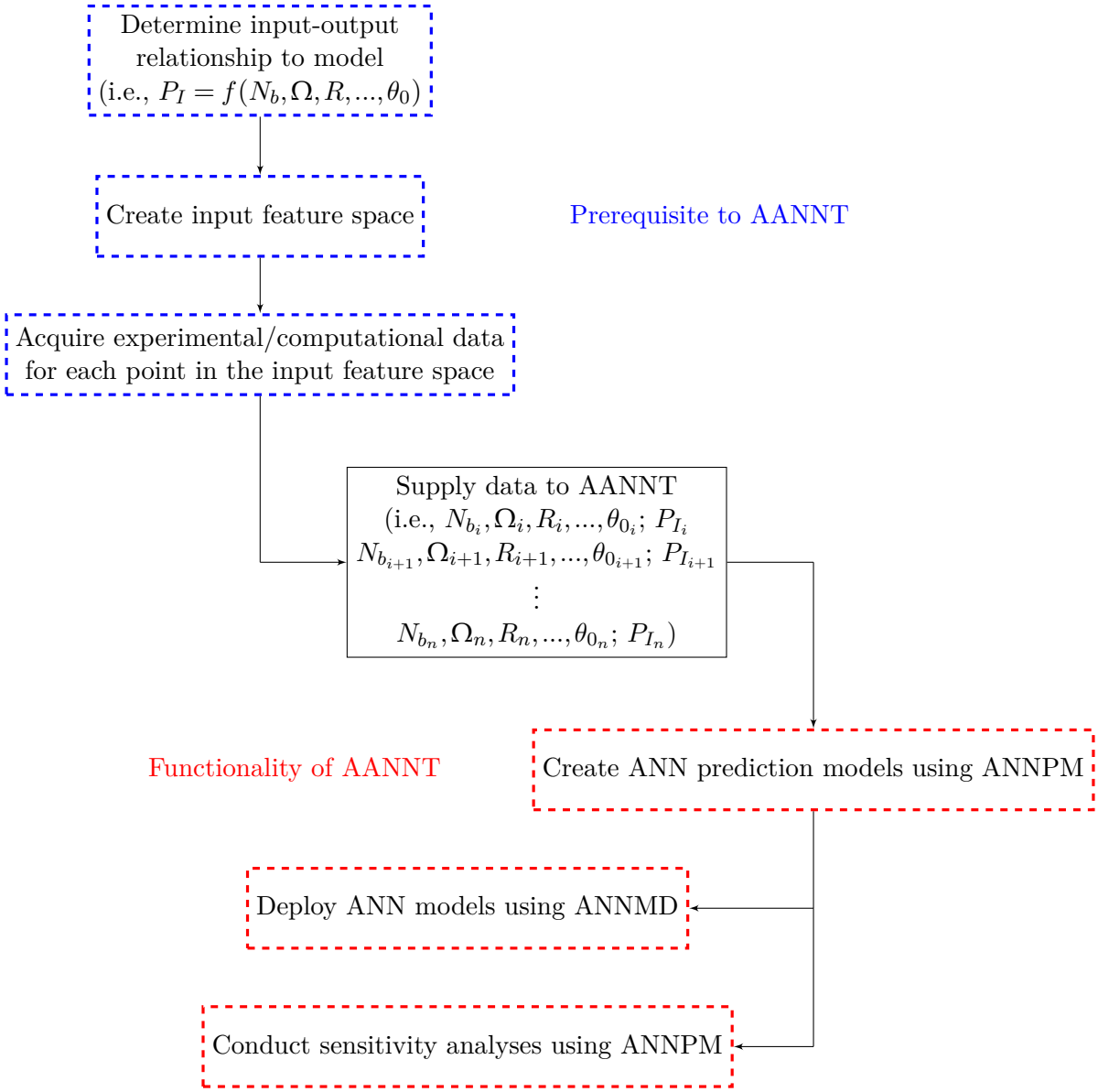


Figure 2: Illustration of the AANNT component hierarchy and the AANNT prerequisites.

where the prediction model has clustered shareholders to a particular demographic group.

Supervised learning, on the other hand, involves the use of labeled data. Since this form of learning uses labeled data, the overall goal is for prediction models to learn the mapping between input data, $(\mathbf{x}_k, \mathbf{x}_{k+1}, ..., \mathbf{x}_s)$, and labeled data, $\mathbf{y}_j$, rather than learning a descriptive similarity between the input data points. Using the same input data of demographic information as the previous example, the labeled data, y_1, may hold a binary form of classification describing shareholder status (e.g., $y_1 = 0$ for nonshareholder status and $y_1 = 1$ for shareholder status). Contrary to the previous task of clustering demographic information of shareholders, the supervised learning task is to learn a relationship between the demographic information and shareholder status; that is,

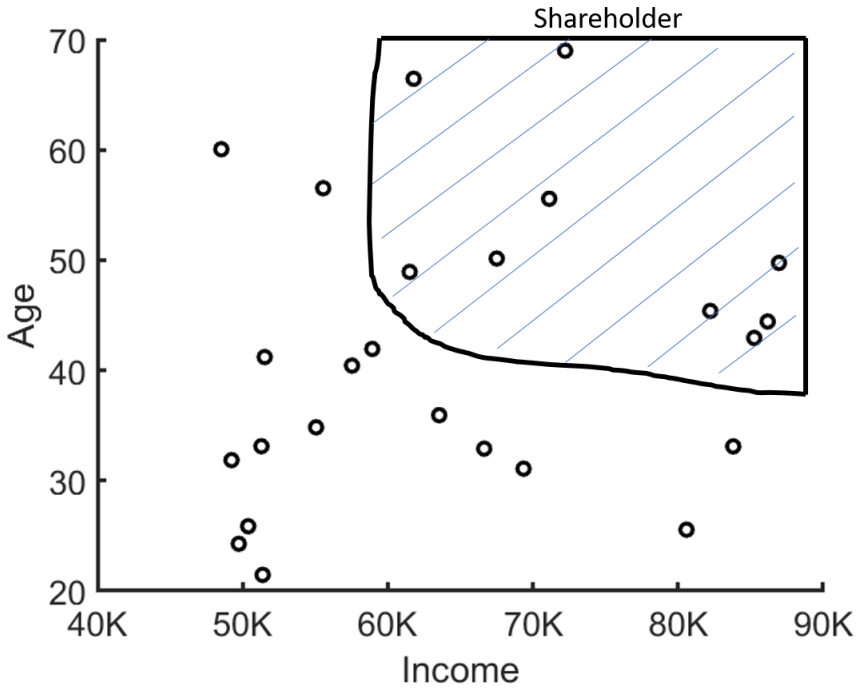


Figure 3: Illustration of an unsupervised clustering problem where shareholder status is determined by age and yearly income.

given $(\mathbf{x}_k, \mathbf{x}_{k+1}, ..., \mathbf{x}_s)$, the model will predict whether a particular individual is a shareholder. The general form of this supervised learning classification problem is:

$$f(\mathbf{x}_k, \mathbf{x}_{k+1}, ..., \mathbf{x}_s) = \hat{y}_1, \tag{1}$$

where $\hat{y}_1$ is the predicted value of y_1. An example of this supervised classification problem is shown in Fig. 4 where the age and income are provided to the ML model, which then uses these inputs to predict shareholder status.

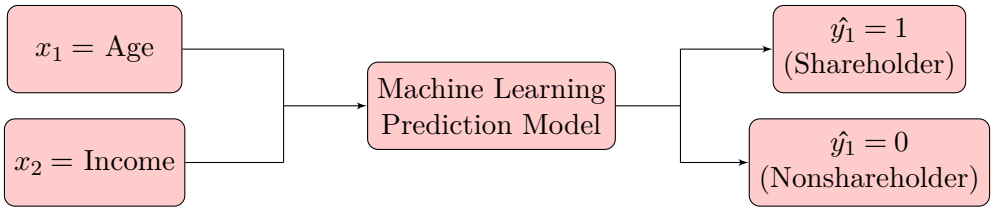


Figure 4: Illustration of a supervised classification problem where shareholder status is predicted by the model using inputted demographic information.

Other forms of this binary classification task may entail the use of multiple labeled data for each input. For example, a shape classification problem may use: x_1 = number of sides and x_2 = angle between sides, where the classification model will have three labeled outputs for circular

($\hat{y}_1 = 1$ for circular or $\hat{y}_1 = 0$ for noncircular), rectangular ($\hat{y}_2 = 1$ for rectangular or $\hat{y}_2 = 0$ for nonrectangular), and triangular ($\hat{y}_3 = 1$ for triangular or $\hat{y}_3 = 0$ for nontriangular). Based upon the predicted outputs, the model will determine what the shape of the object is (e.g., $\hat{y}_1 = 0$, $\hat{y}_2 = 1$, and $\hat{y}_3 = 0$ for rectangular). An example of this supervised multiclass classification problem is shown in Fig. 5 where the number of sides and angle between sides are provided to the ML model, which then uses these inputs to predict the shape.

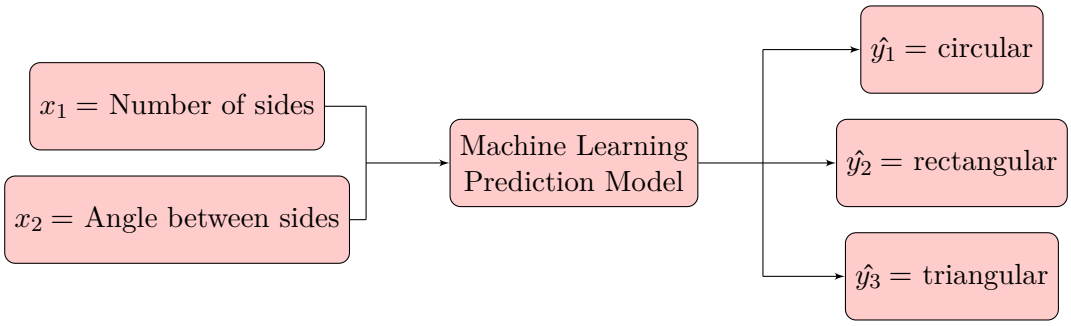


Figure 5: Illustration of a supervised multiclass classification problem where shape is predicted by the model using number of sides and angle between sides.

In addition to the supervised learning classification problem, the other type of supervised learning is regression. Regression also involves the use of labeled data; however, rather than the output consisting of single or multiple categories, the output data for regression are almost exclusively continuous, meaning that the regression problem is one of interpolation [5]. There are many model architectures associated with regression, such as polynomial regression models, Gaussian models (e.g., Kriging [6, 7]), Bayesian interpolation [8, 9], and of particular importance to this documentation, ANNs [10]. A generic example of a supervised regression problem is shown in Fig. 6 where the inputs, $(\mathbf{x}_k, \mathbf{x}_{k+1}, ..., \mathbf{x}_s)$ are provided to the ML model, which predicts a continuous output, $\hat{\mathbf{y}}_1$.

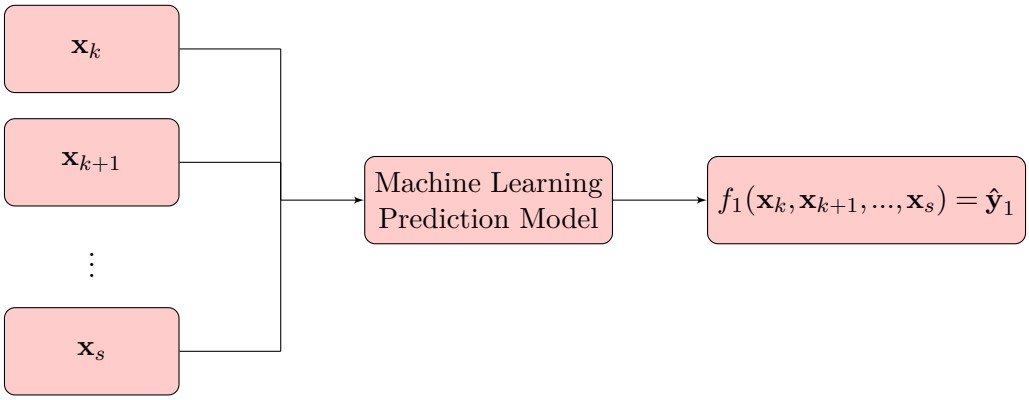


Figure 6: Illustration of a supervised regression problem where a continuous output is predicted by the ML model.

2.1 Artificial Neural Networks for Regression

2.1.1 General Model Form

ANNs were selected as the ML prediction model architecture for this software based upon their successful implementation by Thurman and Somero [11, 12] and their proven application to aeroacoustic problems [1, 2, 13, 14]. The ANN aims to replicate the architecture of the neurons in the human brain, set up in 'hidden' layers as shown in Fig. 7. Each hidden layer consists of a number of activation functions, or neurons, aligned in parallel. All neurons in a particular layer are activated in unison, with different multiplicative weights along the connections between neurons, inputs, and outputs. In general, if an ANN has more than one hidden layer, it is considered a multilayer perceptron (MLP); otherwise, it is a single-layer perceptron (SLP).

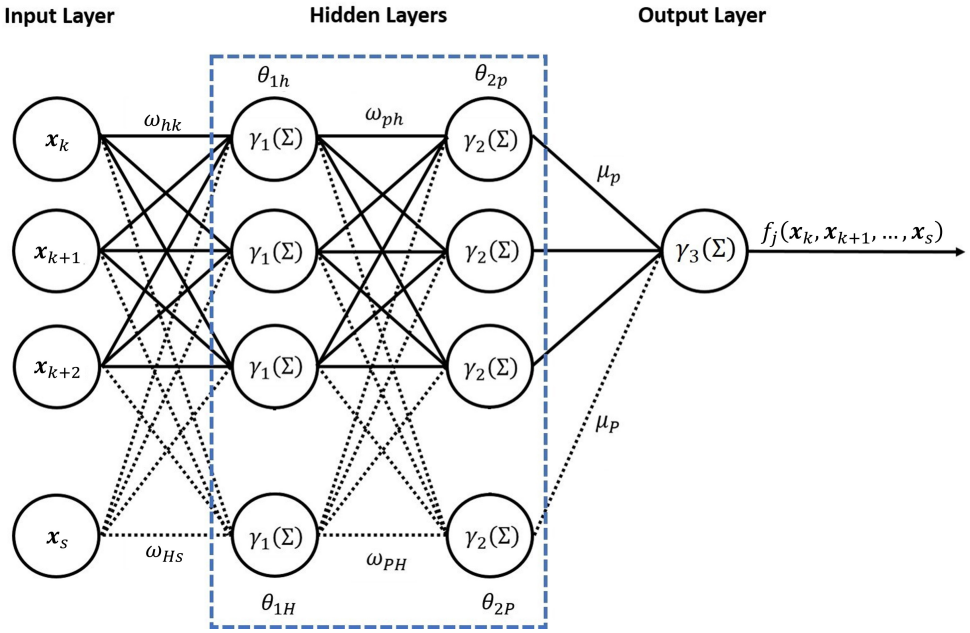


Figure 7: Illustration of a MLP. Adapted from Thurman et al. [2].

In the case of the two-layer MLP shown in Fig. 7, the general equations defining the ANN are:

$$z_{1h} = \gamma_1 \left(\sum_{k=1}^{s} \omega_{hk} \mathbf{x}_k + \omega_{h0} \right), \tag{2}$$

$$z_{2p} = \gamma_2 \left(\sum_{h=1}^{H} \omega_{ph} z_{1h} + \omega_{p0} \right), \tag{3}$$

$$f_j(\mathbf{x}_k, \mathbf{x}_{k+1}, ..., \mathbf{x}_s) = \sum_{p=1}^{P} \mu_p z_{2p} + \mu_0 = \hat{\mathbf{y}}_j, \tag{4}$$

where H is the number of neurons in the first hidden layer, and P is the number of neurons in the second hidden layer. In Eqs. 2, 3, and 4, γ is the activation function used by the neurons in a hidden or output layer, z is the output from a neuron in a hidden layer, ω_{hk} are the weights

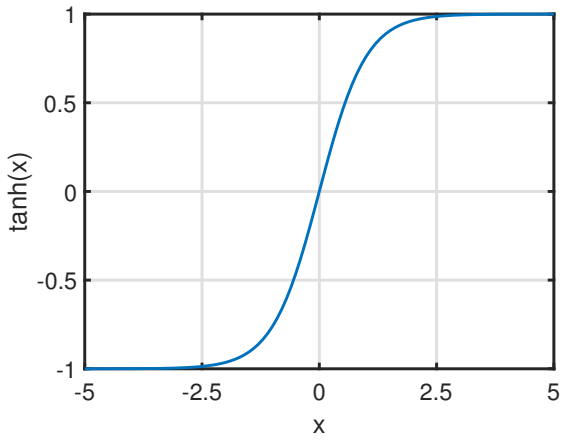
between the input features and the first hidden layer, ω_{ph} are the weights between the first and second hidden layers, μ_p are the weights between the second hidden layer and the output, ω_0 and μ_0 are commonly used bias terms (i.e., intercepts) added to each hidden layer, and f_j is the j^{th} ANN model producing the predicted value, $\hat{\mathbf{y}}_j$.

The activation functions, γ, can be thought of as functional 'mappings' of a neuron's input onto highly nonlinear hyperplanes defined by the structure of γ. These activation functions are necessary to introduce nonlinearity to the ANN, enabling it to effectively model complex, nonlinear input-output relationships. Commonly used activation functions for regression applications are the hyperbolic tangent (tanh), defined by Eq. 5 and shown in Fig. 8a, and continuously differentiable variants of the rectified linear unit (ReLU), such as the exponential linear unit (ELU), which is defined in Eq. 6 and shown in Fig. 8b:

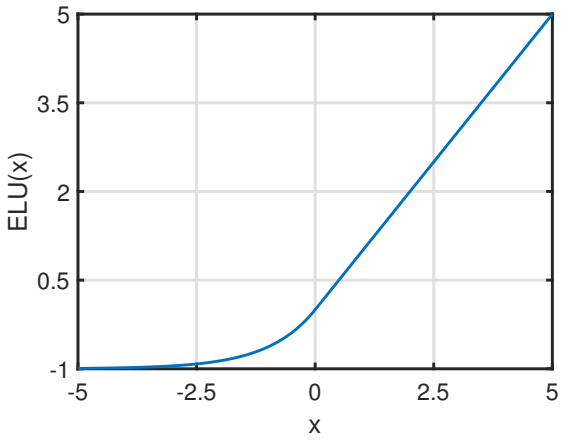
$$\gamma(x) = \tanh(x) = \frac{2}{1+e^{-x}} - 1, \tag{5}$$

$$\gamma(x) = \text{ELU}(x) = \begin{cases} \alpha(e^x - 1) & x < 0 \\ x & x \geq 0, \end{cases} \tag{6}$$

where α is a multiplicative weight that often holds a value of unity.



(a) Hyperbolic tangent (tanh) activation function.



(b) Exponential linear unit (ELU) activation function.

Figure 8: Illustration of commonly used ANN activation functions over a 1-D input feature space, x, where $-5 \leq x \leq +5$.

2.1.2 Model Training

The training procedure for an ANN first involves the random initialization of all weights along the connections between neurons, inputs, and outputs. The input data, $(\mathbf{x}_k, \mathbf{x}_{k+1}, ..., \mathbf{x}_s)$, are then provided to the ANN in a feedforward manner and a predicted value, $\hat{\mathbf{y}}_j$, is produced. This predicted value is then tested against the provided output, or labeled, data associated with the input data,

and a cost function, $\tilde{E}$, is evaluated. This cost function is a numerical description of the error between the labeled output data, $\mathbf{y}_j$, and the predicted output, $\hat{\mathbf{y}}_j$. Commonly used cost functions associated with ANN training are the mean absolute error (MAE) and the mean squared error (MSE) shown in Eqs. 7 and 8, respectively:

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|, \tag{7}$$

$$\text{MSE} = \frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2, \tag{8}$$

where n is the number of experimental data samples. This error calculated by the cost function is then propagated from the output, $\hat{\mathbf{y}}_j$, back through the ANN via the chain rule of differentiation:

$$\frac{\partial\tilde{E}}{\partial\omega_{hk}} = \frac{\partial\tilde{E}}{\partial\hat{\mathbf{y}}_j}\frac{\partial\hat{\mathbf{y}}_j}{\partial z_{2h}}\frac{\partial z_{2h}}{\partial\omega_{ph}}\frac{\partial\omega_{ph}}{\partial z_{1h}}\frac{\partial z_{1h}}{\partial\omega_{hk}}, \tag{9}$$

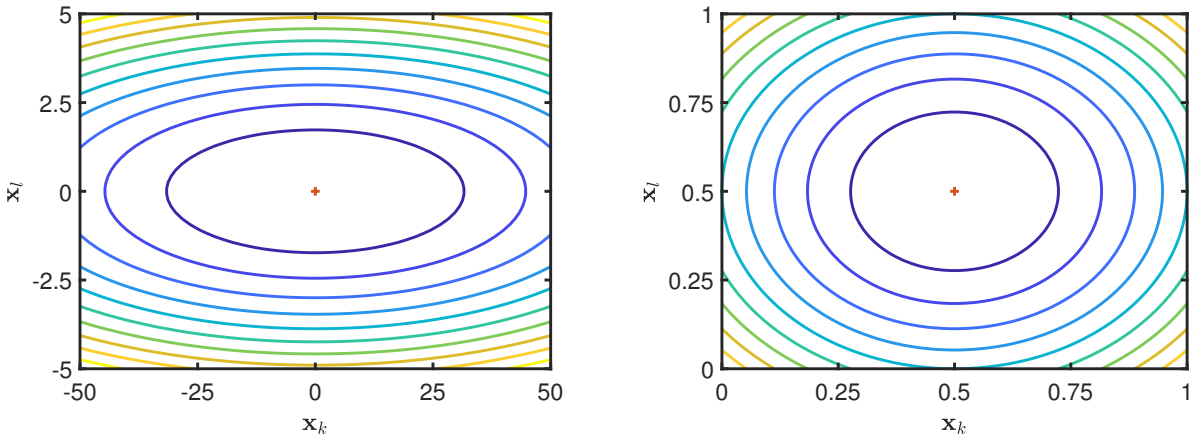
where $\partial\tilde{E}/\omega_{hi}$ is the change in error associated with changing the value of the weight, ω_{hi}, between the input layer and the first hidden layer for the two-layer MLP shown in Fig. 7. This method for propagating the error back through the ANN has been coined *backpropagation* by Rumelhart et al. [15]. The gradients calculated using this backpropagation are typically used with an optimization algorithm (e.g., stochastic gradient descent, adaptive gradient algorithm (Adagrad) [16], RMSProp [17]) to update the weights along each connection in the ANN. One optimization cycle through the training data samples, n, is considered to be one *epoch*.

2.1.3 Model Regularization

Various forms of regularization are often used in the ANN training process to promote optimization convergence. The first type of regularization that is often used involves the rescaling of all input features to a common basis (i.e., $0 \leq \mathbf{x}_k \leq 1$) before they are presented to the ANN [18]. In general, input features may differ by orders of magnitude from each other (i.e., $\mathbf{x}_k << \mathbf{x}_l$), mitigating the optimizer's ability to find a global minimum. This simple data manipulation ensures that the trained weight values along connections of the ANN are of similar magnitudes, thus improving convergence and stabilization of the optimizer. It can be inferred from the unscaled input feature space in Fig. 9a that the gradient in the $\mathbf{x}_l$ direction, $\partial f(\mathbf{x}_k,\mathbf{x}_l)/\partial\mathbf{x}_l$, is much larger than the gradient in the $\mathbf{x}_k$ direction, $\partial f(\mathbf{x}_k,\mathbf{x}_l)/\partial\mathbf{x}_k$, meaning that an optimizer would have more difficulty converging to the global minimum than for the scaled input feature space in Fig. 9b, where the gradients in both directions, $\mathbf{x}_k$ and $\mathbf{x}_l$, are equivalent. A similar rescaling can be applied to the output of each hidden layer to further improve optimization convergence.

Another commonly used form of regularization can be described as penalizing $\tilde{E}$ with L_1 regularization (i.e., ridge regression), L_2 regularization (i.e., lasso regression), or both (i.e., elastic net). With L_1 regularization, the sum of the absolute values of all weight parameters, shown in Eq. 10, is added to $\tilde{E}$ before the backpropagation process:

$$L_1 = \sum_i\sum_k|\omega_{ik}|. \tag{10}$$



(a) Unscaled 2-D input feature space.

(b) Scaled 2-D input feature space.

Figure 9: Comparison between arbitrary unscaled and scaled input feature spaces, $(\mathbf{x}_k, \mathbf{x}_l)$.

Since the general functional form of the absolute value operator behaves like a 'ridge,' this penalization drives learned weight values along connects of the ANN to zero. In contrast, L_2 regularization adds the sum of the squared weight parameters, which can be seen in Eq. 11:

$$L_2 = \sum_i \sum_k \omega_{ik}^2. \tag{11}$$

This penalization reduces the learned weight values toward zero, without necessarily making them zero, since the parabolic functional form of the square operator will 'lasso' about zero. Both the L_1 and L_2 penalizations serve to mitigate the ANN from learning large weight values along each connection, thus deterring the ANN from over- or underfitting the training data.

Dropout can also be used as a form of regularization to prevent overfitting the training data [19]. This dropout method randomly eliminates a certain percentage of the neurons in the hidden layer during each epoch of the optimization procedure, which ensures that the weights associated with any particular neuron are not overtrained. Other forms of regularization include, but are not limited to, various ensemble methods (e.g., K-fold cross-validation [20], boosting [21], bootstrap aggregation [22]) and imposing an early stopping criterion on the training process based upon a satisfactory value of $\tilde{E}$.

3 Artificial Neural Network Prediction Modeling (ANNPM)

3.1 Process Summary

The ANNPM portion of the AANNT deals exclusively with the development of ANN prediction models and leverages ML functionalities available in Google's TensorFlow platform [23] and implemented in Python. This software is structured to operate using a namelist file specified by the user. This namelist allows the user to specify the number, names, and mimimum/maximum values of any continuous input features to be used in the modeling procedure as well as a flag for enabling the rescaling of these input features. The specification of categorical (i.e., discrete) inputs, the number and names of the outputs, and the rescaling of the output data is also available to the user. The ordering of these user-defined namelist inputs associated with the input features should follow the same structure as the training data file supplied by the user and specified in the namelist, with the categorical input features being listed first, followed by the continuous input features, and lastly, the labeled data.

The software then organizes the provided input feature space, and rescales the continuous input features using Eq. 12, if this option is selected by the user:

$$\tilde{\mathbf{x}}_k = \frac{\mathbf{x}_k - \mathbf{x}_{k_{min}}}{\mathbf{x}_{k_{max}} - \mathbf{x}_{k_{min}}}, \tag{12}$$

where the subscript, k, corresponds to the number of input features. In Eq. 12, $\tilde{\mathbf{x}}$ corresponds to the rescaled input feature, $\boldsymbol{x}$, ranging from $0 \leq \tilde{\mathbf{x}} \leq 1$. If categorical input features are specified, ANNPM uses one-hot encoding to generate additional binary input features correspondent to the discrete values of each categorical input feature. For example, since the number of rotor blades, N_b, is a categorical feature with values correspondent to three-, four-, and five-bladed rotors, the one-hot encoding procedure would generate three input features; each correspondent to either a three-, four-, or five-bladed rotor. If a particular input data sample consists of a three-bladed rotor, the column associated with three-bladed rotors would hold a value of one, whereas the columns associated with four- and five-bladed rotors would hold values of zero. An example of one-hot encoding can be seen in Table 2.

Table 2: Illustration of categorical input feature one-hot encoding for the number of rotor blades, N_b.

Number of Rotor Blades (N_b)		
$N_b = 3$	$N_b = 4$	$N_b = 5$
1	0	0

Typical ANN prediction modeling involves the use of test data to query the model after each epoch of the training procedure to ensure the ANN will generalize to new data previously unseen by the model. These test data can either be supplied to ANNPM in a separate data file or the user can specify to split the training data by a percentage that will be used for test data. Though it is not recommended to split the training data if Design of Experiments (DoE) is used for input feature space generation, this training/test data split is a commonly used method, so its implementation was made available in ANNPM. These test data are processed identically to the previously mentioned

techniques used for the training data.

After the training and test data have been processed by ANNPM, the architecture of the ANN is then established. The namelist allows the user specification of one or more of the following parameters: the number of hidden layers, the number of neurons in each hidden layer, the activation function to be used for each neuron in the hidden layers, the activation function to be used in the output(s), the cost function to be used in the training procedure, the magnitude of L_1 and L_2 regularization to be used in the training procedure, dropout rate percentage, and the maximum number of epochs in the training procedure. Since TensorFlow is utilized by ANNPM as the basis for ML routines, these user-specified parameters can hold any value that is recognized by TensorFlow with a vast range of selection. Separate prediction models are then internally created by ANNPM for each respective combination of the various user-defined ANN parameters.

The weights for all ANN connections between the inputs, hidden layers, and outputs are initialized using the He method [24] , which initializes each weight based on a random sampling from a uniform distribution between $[-\beta, +\beta]$, with β being defined as:

$$\beta = \sqrt{\frac{6}{m}}, \tag{13}$$

where m is the number of connections to a particular neuron. This He method has shown ANN training convergence superiority over other initialization techniques, especially when used with ReLU activation functions or their variants [24].

The ANN training procedure utilizes the adaptive momentum estimation (ADAM) optimizer [25], which combines the advantages of both the AdaGrad [16] and RMSProp [17] optimizers. The ADAM optimizer is "computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters [25]." This optimizer entails a momentum-based stochastic gradient descent where the optimization learning rate is continuously modified during the training procedure using bias-corrected exponential moving average estimates of the gradient's mean and variance.

The different ANN prediction models generated by ANNPM for each combination of the user-defined parameters are sequentially trained over the processed training data and are evaluated over the specified test data at each epoch using a user-specified evaluation function. An early stopping criterion is also implemented, which stops the training procedure if there is no improvement in the user-specified evaluation function after a default value of 250 training epochs, although the user may specify a different number of epochs in the namelist.

Once all ANN prediction models have been trained, ANNPM evaluates and exports the MAE, the root mean squared error (RMSE), the MSE, the mean absolute scaled error (MASE), the mean absolute percentage error (MAPE), and the coefficient of determination (R_d^2) values calculated over both the training and test data, as well as the model form parameters for each model. The namelist allows the user to specify whether to solely retain the optimal model, in terms of test accuracy, or to retain all trained models. The final ANN prediction model(s) are then saved under a user-specified name in the Hierarchical Data Format (HDF).

3.2 Setup and Execution

ANNPM is standalone Python code, requiring only a user-generated namelist file, training and/or test data, and the installation of the following Python packages:

- sys
- os
- f90nml
- numpy
- pandas
- importlib
- tensorflow
- time

The user is referred to the Anaconda user-guide page: `https://docs.anaconda.com/anaconda/user-guide/tasks/install-packages/` for instruction on package installation. This section of the manual explains the various user-specified inputs to the namelist as well as how to properly execute ANNPM.

3.2.1 User-specified Namelist

The general form of the user-specified ANNPM namelist consists of four sections correspondent to the input data, input feature description, ANN grid search parameters, and output data, respectively. The first section of the ANNPM namelist is the &InputData section, which can be seen in Ex. 1.

Example 1: Illustration of the &InputData section of the user-specified ANNPM namelist.

```
&InputData
! File location/name for the training data.
strTrainingDataFileName = 'training_data.in'

! The type of test data to be used. The default value of 'split' can be used if separate
! test data are unavailable and the user wishes to split the training data into
! training/test data. If the user has separate test data, the 'file' option can be used.
strTestDataType = 'file'

! Percentage of total training data to be used for training if strTestDataType = 'split'.
! The rest of the data (1 - fltDataSplitRatio) will be used for test data. The default
! value is 0.8.
fltDataSplitRatio = 0.8

! File location/name for the testing data. Only needed if strTestDataType = 'file'.
strTestDataFileName = 'test_data.in'
/
```

The data to be used for training the ANN are to be contained in a text file beginning with three rows of text, followed by space-separated columns consisting of categorical input features, continuous input features, and output data, as shown in Ex. 2. This format is identical to that used by the ANOPP2 Design of Experiments Tool (ADoET) and is used here for consistency.

Example 2: Illustration of ANNPM training data format.

```
Latin HyperCube Design (LHD)
Design Points:          40
MaxPro Value:  4.6600233280543 91E-021
2       3849    6.03    2.32    8       43.7    13      -4.6    59.29862941
2       3576    6.15    1.52    0.3     25.3    6.5     -5      23.111179
2       5937    7.76    2.48    8.5     37      15      -4.9    9.473326792
2       5447    7.96    1.56    0.6     44.1    10.5    4.2     85.04665381
2       4744    7.49    1.7     2.9     49.9    6.2     3.6     69.53985482
2       5201    7.59    2.98    8.9     21.4    14.4    -4.4    38.38550923
3       3676    7.86    1.53    8.8     33.2    11.4    -0.6    25.4384549
3       4091    6.04    1.73    8.6     22.3    6.8     -2.6    38.53058865
3       5767    6.4     1.67    4.1     21      9.6     0.4     48.42523567
3       5929    7.39    1.54    7.7     31.3    6.7     5       135.0408063
3       5522    7.16    1.87    0.9     32.7    6       0.7     54.68018672
3       4895    7.92    2.89    5       36.2    13.7    4.1     167.7816834
4       4410    6.13    1.56    8.3     48.3    14.4    3.2     70.75052124
```

The columns, from left to right, correspond to the input features outlined in Table 1 and are ordered first by categorical input features, followed by continuous input features, and lastly, the labeled data. The name of this training data file is specified in the namelist via the input, *strTraining-DataFileName*. Since test data are necessary to validate the ANN prediction model throughout the training procedure, the input, *strTestDataType*, allows two options: 'file' and 'split'. If the user selects 'file', then the test data file, *strTestDataFileName*, must be specified and must follow the previously defined data formatting structure shown in Fig. 2. Otherwise, the user can select the 'split' option, where the training data are randomized, then split between training and test data. The input, *fltDataSplitRatio*, must be specified if using the 'split' option and must hold a float value between zero and one. The *fltDataSplitRatio* value is the ratio of input training data that will be used for training and $(1 - fltDataSplitRatio)$ will be used for test data.

The second section of the ANNPM namelist, &InputFeatureDescription, shown in Ex. 3 deals with the number, name, and minimum/maximum values of the input features.

Example 3: Illustration of the &InputFeatureDescription section of the user-specified ANNPM namelist.

```
&InputFeatureDescription
! Integer value for the number of continuous features.
nContinuousInputs = 7

! Continuous input names.
strContinuousInputNames = 'RPM', 'Radius', 'Target Thrust', 'Camber',
'Camber_loc', 'Thickness', 'Collective'

! Boolean to normalize continuous inputs. The default value of True is recommended.
blnNormalizeContinuousInputs = True

! Lower values of each continuous input.  Only needed
! if blnNormalizeContinuousInputs = True.
fltContinuousInputRangeLow = 3500, 6, 1.5, 0, 20, 6, -5

! Upper values of each continuous input. Only needed
! if blnNormalizeContinuousInputs = True.
fltContinuousInputRangeHigh = 6000, 8, 3, 9, 50, 15, 5

! Integer value for the number of categorical inputs. The default value is 0.
nCategoricalInputs = 1

! Name of the categorical inputs. Only needed if nCategoricalInputs != 0.
strCategoricalInputNames = 'Nb'
/
```

The input, *nContinuousInputs*, is an integer value corresponding to the number of continuous input features, which are then named via the input, *strContinuousInputNames*, in the same order as in the data file specified via *strTrainingDataFileName* in the &InputData section of the ANNPM namelist. The inputs, *fltContinuousInputRangeLow* and *fltContinuousInputRangeHigh*, correspond to the minimum and maximum values of each continuous input feature, respectively. The user can specify, via the boolean input, *blnNormalizeContinuousInputs*, whether ANNPM rescales the continuous input features using Eq. 12. Lastly, the user can specify any number of categorical input features via *nCategoricalInputs* that are named with *strCategoricalInputNames*, again, having the same order as in the data file specified via *strTrainingDataFileName* in the &InputData section of the ANNPM namelist.

The third section of the ANNPM namelist, &GridSearchParameters, deals with the various hyper-parameters to be used during the grid search procedure to find the optimal ANN prediction model. An example of this third namelist section can be seen in Ex. 4.

Example 4: Illustration of the &GridSearchParameters section of the user-specified ANNPM namelist.

```
&GridSearchParameters
! Desired activation functions for grid search. 'gelu' and 'swish' are recommended
! and 'swish' is the default value.
strActivationFunctions = 'tanh', 'elu'

! Desired output activation functions for grid search. The default value of 'linear' is
! recommended.
strOutputActivation = 'relu', 'linear'

! Loss functions for grid search. The default value of 'huber' is recommended.
strLossFunctions = 'mae', 'rsme'

! Percentage of neurons in each hidden layer to be randomly dropped at each epoch during
! training. The default value is 0 or no dropout.
fltDropoutRates = 0.25, 0.50

! Number of hidden layers. The default value is 1.
intHiddenLayers = 1, 2

! Number of neurons or activation functions in each hidden layer. The default
! value is 50.
intNeurons = 100, 250

! L1 regularization values. The default value is 0 or none.
fltL1Regularization = 0.01, 0

! L2 regularization values. The default value is 0 or none.
fltL2Regularization = 0.01, 0.1

! Maximum number of training epochs or passes through the training data. The default
! value is 500.
nMaxEpochs = 1000

! Number of epochs for early stopping in the training procedure. The default value is 100.
intEarlyStopping = 250

! Verbosity used in training. Options are True or False. The default value is True.
blnVerbosity = True

! Test flag for regression testing. Should be set to the default value of False
! for application.
blnTest = False
/
```

One or multiple values can be used with all inputs for the &GridSearchParameters section except for *nMaxEpochs*, which is the integer number of maximum training epochs if the early stopping criterion is not met (typically set to 1000), *intEarlyStopping*, which is the number of epochs over which the early stopping criterion is evaluated, *blnVerbosity*, which is a boolean to specify output verbosity during the training procedure, and *blnTest*, which is a boolean that specifies limited output data for developmental testing purposes only.

The inputs, *strActivationFunctions*, *strOutputActivation*, and *strLossFunctions*, correspond to the activation function used by all hidden layer neurons, the activation function used by the output layer, and the cost function used in the ADAM optimization, respectively. Since Tensorflow is used by ANNPM, values for these two inputs can hold any activation function or loss function recognized by TensorFlow and the user is referred to `https://www.tensorflow.org/api_docs/python/tf/`

`keras/activations` and `https://www.tensorflow.org/api_docs/python/tf/keras/losses` for a comprehensive list of available inputs for activation functions and loss functions, respectively. Based upon user experience and academic literature, variants of ReLU, specifically the Gaussian Error Linear Unit (GELU) and the Swish [26] activation functions, have shown great success for regression problems. For the cost function, MAE or HUBER, which is a combination of both the MAE and MSE cost functions, are recommended for regression problems. If the output data are very small in magnitude (e.g., 0.01), however, the mean squared logarithmic error (MSLE) is recommended.

The inputs, *intHiddenLayers* and *intNeurons*, correspond to the number of hidden layers and the number of neurons in each hidden layer, which define the model architectures to be used in the optimal model grid search. Three forms of regularization can be user-specified in the ANNPM namelist: *fltL1Regularization*, *fltL2Regularization*, and *fltDropoutRates*. These three types of regularization have been discussed in Section 2.1.3 of this reference manual and values of zero can be used for their exclusion.

The last section of the ANNPM namelist, &OutputData, corresponds to the location, number, names, and scale of the models to be outputted by ANNPM. An example of this last section can be seen in Ex. 5.

Example 5: Illustration of the &OutputData section of the user-specified ANNPM namelist.

```
&OutputData
! Location for where to save prediction models. The default value is 'ann_models/'.
strOutputFunctionLocation = 'ann_models/'

! Number of model outputs. The default value is 1.
nModelOutputs = 2

! Name of the output functions. The default value is 'output'.
strOutputFunctionNames = 'Induced_Power', 'Profile_Power'

! Boolean to normalize the outputs. The default value of False is recommended.
blnNormalizeOutput = False

! Lower values of each output. Only needed if blnNormalizeOutput = True.
fltOutputRangeLow = 0, 0

! Upper values of each output. Only needed if blnNormalizeOutput = True.
fltOutputRangeHigh = 200, 200

! Evaluation function for optimal model selection. The default value is 'mae'.
! Options are 'mae', 'rsme', 'mse', 'mape', 'mase', and 'R2'.
strEvalFunction = 'mae'

! Save 'all' models or just the 'opt' model. The default value is 'all'.
strSavedModels = 'all'

! Output summary file name. The default value is 'Model_Summary.out'.
strOutputSummaryFileName = 'Model_Summary.out'
/
```

As the names suggest, *strOutputFunctionLocation*, *nModelOutputs*, and *strOutputFunctionNames* correspond to the directory location, the number of outputs, and the ANN prediction model names used by ANNPM to save the prediction models. The user can specify whether to save only the optimal ANN prediction model ('opt') evaluated over *strEvalFunction* or to save all ANN prediction

models ('all') trained in the grid search procedure using the input, *strSavedModels*. A summary file containing the ANN architecture, as well as the evaluation functions, discussed in Section 3.1, calculated over both the training data and test data is outputted by ANNPM with a file name specified using *strOutputSummaryFileName*.

3.3 Demonstration

To run ANNPM, the user need only run the ANNPM.py script located in the 'Source' directory. This script should be run in the directory containing the user-specified namelist, examples of which can be found in the 'Test/Annpm' directory. A command line argument correspondent to the namelist file name is also necessary to run ANNPM. For example, if the user were to execute ANNPM from the 'Test/Annpm/full_data' directory, the command line execution would take the following form: 'python ../../../Source/ANNPM.py ANNPM.nml'.

Five examples are provided with AANNT, which correspond to the full data case from Ref. [2], the full data case involving output scaling (note that the scaling is only used during the model training/execution; the user inputted training/test data should not be modified and the output data will be rescaled from coded units back to natural units), the full data case with two outputs, a reduced number of inputs case with the exclusion of the categorical input, and a reduced input case with the inclusion of a second, arbitrary categorical input feature. These examples were created to illustrate the implementation of ANNPM with varying numbers and types of input features, as well as the use of 'split' and 'file' data types for *strTestDataType*. In general, these namelist files can be copied and modified by the user to accommodate any regression problem, so long as input data are supplied in the proper format, specified in Ex. 1.

Since the execution of ANNPM is identical for all five example cases, these examples will not be discussed in detail. Only brief highlights of what to expect throughout the ANN prediction modeling procedure will be shown. For example, the user can select *True* for *blnVerbosity* to ensure ANNPM is functioning properly. This will output the cost function value and *MAPE* values calculated over both the training and test data at each epoch. An example of this command line output can be seen in Ex. 6 where *nMaxEpochs* has been set to five for demonstration purposes.

Example 6: Illustration of the ANNPM command line output for $nMaxEpochs = 5$ and $blnVerbosity = True$.

```
Epoch 1/5
142/142 [==========] - 1s 4ms/step - loss: 70.9160 - MAE: 62.3023 - root_mean_squared_error: 74.5974 -
MSE: 5564.7700 - MAPE: 86.3807 - val_loss: 66.3161 - val_MAE: 60.0252 - val_root_mean_squared_error:
70.6326 - val_MSE: 4988.9619 - val_MAPE: 84.3498
Epoch 2/5
142/142 [==========] - 1s 4ms/step - loss: 70.9160 - MAE: 62.3023 - root_mean_squared_error: 74.5974 -
 MSE: 5564.7700 - MAPE: 86.3807 - val_loss: 66.3161 - val_MAE: 60.0252 - val_root_mean_squared_error:
 70.6326 - val_MSE: 4988.9619 - val_MAPE: 84.3498
Epoch 3/5
142/142 [==========] - 0s 3ms/step - loss: 52.0023 - MAE: 40.4462 - root_mean_squared_error: 52.3011 -
 MSE: 2735.4009 - MAPE: 54.3930 - val_loss: 46.7289 - val_MAE: 34.9470 - val_root_mean_squared_error:
 45.3109 - val_MSE: 2053.0798 - val_MAPE: 62.3843
Epoch 4/5
142/142 [==========] - 0s 3ms/step - loss: 44.1299 - MAE: 31.2779 - root_mean_squared_error: 42.6657 -
 MSE: 1820.3657 - MAPE: 46.3757 - val_loss: 37.5834 - val_MAE: 21.4946 - val_root_mean_squared_error:
 30.0063 - val_MSE: 900.3810 - val_MAPE: 47.1466
Epoch 5/5
142/142 [==========] - 0s 3ms/step - loss: 36.8892 - MAE: 21.9505 - root_mean_squared_error: 31.9958 -
MSE: 1023.7319 - MAPE: 33.0207 - val_loss: 30.4601 - val_MAE: 15.6950 - val_root_mean_squared_error:
24.2853 - val_MSE: 589.7770 - val_MAPE: 37.0260

Saving ANN Models
```

Upon successful execution of ANNPM, the user should expect the *strOutputFunctionLocation* to be populated with ANN prediction models in HDF format named according to *strOutputFunctionName* followed by the grid search training iteration number for each particular prediction model. The output file specified by *strOutputSummaryFileName* will be populated with the model architecture and various evaluation metrics of each model trained in the grid search. It should be noted that if more than one output is specified by the user, the evaluation metric used for optimal model identification will be a value averaged over all outputs. An example output file is shown in Ex. 7.

Example 7: Illustration of the output file generated by ANNPM and specified by the namelist input, *strOutputSummaryFileName*.

```
Model 15 Summary:
Hidden Layer Activation Function Type: gelu
Output Layer Activation Function Type: linear
Loss Function Type: huber
Number of Hidden Layers: 2
Number of Neurons per Layer: 250
Dropout Rate: 0.25
Amount of L1 Regularization: 0
Amount of L2 Regularization: 0.1
Model 15 Training MAE: {'Induced_Power': [15.68941879272461]}
Model 15 Training RMSE: {'Induced_Power': [23.833250045776367]}
Model 15 Training MSE: {'Induced_Power': [568.0238037109375]}
Model 15 Training MASE: {'Induced_Power': [0.23031216208322453]}
Model 15 Training MAPE: {'Induced_Power': [24.209136962890625]}
Model 15 Training Rd^2: {'Induced_Power': [0.6660275941094721]}
Model 15 Test MAE: {'Induced_Power': [14.665038108825684]}
Model 15 Test RMSE: {'Induced_Power': [20.594219207763672]}
Model 15 Test MSE: {'Induced_Power': [424.12188720703125]}
Model 15 Test MASE: {'Induced_Power': [0.20937553349770863]}
Model 15 Test MAPE: {'Induced_Power': [29.355205535888672]}
Model 15 Test Rd^2: {'Induced_Power': [0.7155157577343436]}
Model 15 Elapsed Training Time: 4.859565824000001 seconds



Model 16 Summary:
Hidden Layer Activation Function Type: gelu
Output Layer Activation Function Type: swish
Loss Function Type: huber
Number of Hidden Layers: 2
Number of Neurons per Layer: 250
Dropout Rate: 0.25
Amount of L1 Regularization: 0
Amount of L2 Regularization: 0.1
Model 16 Training MAE: {'Induced_Power': [15.691828727722168]}
Model 16 Training RMSE: {'Induced_Power': [24.23138999938965]}
Model 16 Training MSE: {'Induced_Power': [587.1602783203125]}
Model 16 Training MASE: {'Induced_Power': [0.2303475386224769]}
Model 16 Training MAPE: {'Induced_Power': [23.245630264282227]}
Model 16 Training Rd^2: {'Induced_Power': [0.6547762091784133]}
Model 16 Test MAE: {'Induced_Power': [16.153615951538086]}
Model 16 Test RMSE: {'Induced_Power': [22.07985496520996]}
Model 16 Test MSE: {'Induced_Power': [487.5199890136719]}
Model 16 Test MASE: {'Induced_Power': [0.2306282419910611]}
Model 16 Test MAPE: {'Induced_Power': [28.30556869506836]}
Model 16 Test Rd^2: {'Induced_Power': [0.6729908103134172]}
Model 16 Elapsed Training Time: 4.812205824000003 seconds



The best model, based on mase evaluated/averaged over output is model: 10

Total elapsed time to train all models: 382.77096965 seconds
```

4 Artificial Neural Network Model Deployment (ANNMD)

4.1 Process Summary

The ANNMD portion of the AANNT follows a similar structure as ANNPM, however, this tool serves the purpose of evaluating an ANN prediction model over user-specified data and calculating accuracy metrics. Similar to ANNPM, ANNMD is structured to operate using a namelist file specified by the user. This namelist allows the user to specify the number, names, and minimum/maximum values of any continuous input features or outputs used in the modeling procedure performed by ANNPM, as well as a flag for enabling the rescaling of these input features and outputs, similar to the namelist for ANNPM shown in Ex. 1. The categorical input features are specified in the namelist; however, the user must also specify the discrete values of each categorical input feature. In general, any data points specified by the user for model evaluation may not include all discrete values of a categorical input feature, so they must be specified in the namelist to allow for proper one-hot encoding.

Once the input feature descriptions have been identified, the ANNMD namelist allows the user to specify the data over which to deploy the ANN prediction model. These input feature data points can either be prescribed within the namelist, or the user can specify a file containing the evaluation data, which has an identical format to the training and test data used by ANNPM. The evaluation data are then preprocessed by ANNMD identically to how the training/test data were processed by ANNPM with one-hot encoding of categorical input features and continuous input feature scaling, if specified by the user.

The name and location of an ANN prediction model created by ANNPM are specified in the namelist and the model is deployed over the processed evaluation data. If the supplied evaluation data contain labeled data associated with the input feature design points, the user can specify to calculate accuracy metrics over the evaluation data. This is useful for the calculation of additional error metrics such as MAPE, the maximum residual error, and the R_d^2 score,

$$R_d^2 = 1 - \frac{\sum_{j=1}^{n} \left(y_j - \hat{y}_j\right)^2}{\sum_{j=1}^{n} \left(y_j - \bar{y}_n\right)^2}, \tag{14}$$

where $\bar{\mathbf{y}}_n$ is the mean calculated over all evaluation data. In general, the user may typically wish to deploy the ANN prediction model over evaluation data where no labeled data are available; however, if these labeled data are known, such as for test data, this functionality can prove beneficial.

Lastly, ANNMD allows the user to specify whether the predictions calculated over the evaluation data are to be exported in tabular format. This data format specification can be useful if additional postprocessing of the model predictions is necessary.

4.2 Setup and Execution

ANNMD is a standalone Python code, requiring only a user-generated namelist file, evaluation data, prediction models developed by ANNPM, and the installation of the following Python packages:

- sys
- os

- f90nml
- numpy
- pandas
- tensorflow
- scikit-learn

The user is referred to the Anaconda user-guide page: `https://docs.anaconda.com/anaconda/user-guide/tasks/install-packages/` for instruction on package installation. This section of the manual explains the various user-specified inputs to the namelist as well as how to properly execute ANNMD.

4.2.1 User-specified Namelist

The general form of the user-specified ANNMD namelist consists of four sections correspondent to the input data, input feature description, evaluation data description, and output data, respectively. The first section of the ANNMD namelist is the &InputData section, which can be seen in Ex. 8.

Example 8: Illustration of the &InputData section of the user-specified ANNMD namelist.

```
&InputData
! Name of the model directory. The default value is './' or the
! current working directory.
strModelLocation = './'

! Name of the prediction model file generated by ANNPM.
strModelName = 'full_data.h5'

! Define whether the evaluation data will come from a file or will be input
! from namelist. Options are 'file' and 'input'. The default value is 'input'.
strEvalType = 'file'

! Evaluation data file name. Only needed if strEvalType = 'file'.
strEvalFile = 'full_data.in'
/
```

The inputs, *strModelLocation* and *strModelName*, correspond to the directory location and the name of an ANN prediction model created using ANNPM, respectively. The user has two choices when selecting the evaluation data over which to execute a prediction model using ANNMD, which are specified using *strEvalType*. The first selection choice is 'file', which uses data from a text file specified with *strEvalFile* and having a similar format to that discussed in Section 3.2.1. The only difference is that the evaluation data file has one header row, as opposed to three. An example of an input data file can be seen in Ex. 9.

Example 9: Illustration of ANNMD evaluation data format for *strEvalTpe* = 'file'.

```
# Input Data
2       5000    7.50    2.00    5.00    50.00   11.00   3.00
2       4500    6.50    2.00    5.00    50.00   12.50   5.00
4       3750    6.00    3.00    7.50    45.00   15.00   5.00
```

The columns of the data in this input data file correspond to the input features outlined in Table 1, however, the user can specify any arbitrary value for the input features within the minimum/-

maximum feature ranges. The second selection option for *strEvalType* is 'input', where the user can specify the evaluation data directly in the namelist, via the &EvaluationDataDescription input section. An example is shown in Ex. 10.

Example 10: Illustration of &EvaluationDataDescription for $strEvalTpe =$ 'input'.

```
&EvaluationDataDescription
! Values of Nb for three predictions.
strEvalCategory_Nb = '2', '2', '4'

! Values of RPM for three predictions.
fltEvalContinuous_RPM = 5000, 4500, 3750

! Values of Radius for three predictions.
fltEvalContinuous_Radius = 7.50, 6.50, 6.00

! Values of Target_Thrust for three predictions.
fltEvalContinuous_Target_Thrust = 2.00, 2.00, 3.00

! Values of Camber for three predictions.
fltEvalContinuous_Camber = 5.00, 5.00, 7.50

! Values of Camber_loc for three predictions.
fltEvalContinuous_Camber_loc = 50.00, 50.00, 45.00

! Values of Thickness for three predictions.
fltEvalContinuous_Thickness = 11.00, 12.50, 15.00

! Values of Collective for three predictions.
fltEvalContinuous_Collective = 3.00, 5.00, 5.00
/
```

In general, each categorical input feature has an input in &EvaluationDataDescription beginning with *strEvalCategory_* and followed by the name(s) specified using *strCategoricalInputNames* in the &InputFeatureDescription section. For example, if $strCategoricalInputNames =$ 'Nb', then evaluation data for this categorical input feature can be specified in the &EvaluationDataDescription section using the input, *strEvalCategory_Nb*. Similarly, for continuous input features, the correspondent input in &EvaluationDataDescription begins with *fltEvalContinuous_* and ends with the name specified using *strContinuousInputNames* in the &InputFeatureDescription section.

The &InputFeatureDescription section of the ANNMD namelist is almost identical to the &InputFeatureDescription section of the ANNPM namelist. The only difference being the specification of the categorical input feature values. These values are specified using an input that begins with *strCategoryValues_*, followed by the categorical input feature name specified in *strCategoricalInputNames*. This specification of categorical feature values is necessary to allow for proper one-hot encoding, since the evaluation data may not include all discrete values of a categorical input feature. An example of this section can be seen in Ex. 11.

Example 11: Illustration of the &InputFeatureDescription section of the user-specified ANNMD namelist.

```
&InputFeatureDescription
 ! Integer value for the number of continuous features.
nContinuousInputs = 7

! Continuous input names.
strContinuousInputNames = 'RPM', 'Radius', 'Target Thrust', 'Camber',
'Camber_loc', 'Thickness', 'Collective'

! Boolean to specify whether or not to normalize continuous inputs.
! The default value of True is recommended.
blnNormalizeContinuousInputs = True

! Lower values of each continuous input. Only needed if blnNormalizeContinuousInputs = True.
fltContinuousInputRangeLow = 3500, 6, 1.5, 0, 20, 6, -5

! Upper values of each continuous input. Only needed if blnNormalizeContinuousInputs = True.
fltContinuousInputRangeHigh = 6000, 8, 3, 9, 50, 15, 5

! Number of categorical inputs. The default value is 0.
nCategoricalInputs = 1

! Name of the categorical inputs. Only needed if nCategoricalInputs != 0.
strCategoricalInputNames = 'Nb'

! Category values of the categorical inputs. Only needed if nCategoricalInputs != 0.
strCategoryValues_Nb = '2', '3', '4'

! Number of model outputs. The default value is 1.
nModelOutputs = 1

! Name of the output function(s). The default value is 'output'.
strOutputFunctionNames = 'Induced_Power'

! Boolean to normalize the outputs. The default value of False is recommended.
blnNormalizeOutput = False

! Lower values of each output. Only needed in blnNormalizeOutput = True.
fltOutputRangeLow = 0, 0

! Upper values of each output.  Only needed in blnNormalizeOutput = True.
fltOutputRangeHigh = 200, 200
/
```

The last section of the ANNMD namelist, &OutputData, corresponds to the format and name of the output file generated by ANNMD as well as the option to calculate accuracy metrics over the evaluation data, if output data have been included in the evaluation data file specified using *strEvalFile*. An example of this namelist can be seen in Ex. 12.

Example 12: Illustration of the &OutputData section of the user-specified ANNMD namelist.

```
&OutputData
! Output format type. Options are 'table' or 'none'. The default value is 'table'.
strOutputFormat = 'table'

! Boolean to specify accuracy score calculation. Additional column with labeled data
! is necessary in input data file for this option.
blnAccuracyScore = False


! Output summary file name. The default value is 'Prediction_Results.out'.
strOutputSummaryFileName = 'Prediction_Results.out'
/
```

The name of the output file is specified using *strOutputSummaryFileName*. The format of this output file can be specified using *strOutputFormat*, which supports two options: 'none' and 'table'. Examples of these two options can be seen in Exs. 13 and 14, respectively.

Example 13: Illustration of the output file generated by ANNMD using *strOutputFormat* = 'none'.

```
This file contains the ANN predicted results.

Prediction Number: 1
Nb: 2
RPM: 5000
Radius: 7.5
Target_Thrust: 2.0
Camber: 5
Camber_loc: 50.0
Thickness: 11.0
Collective: 3
Predicted Results: 77.98917

Prediction Number: 2
Nb: 2
RPM: 4500
Radius: 6.5
Target_Thrust: 2.0
Camber: 5
Camber_loc: 50.0
Thickness: 12.5
Collective: 5
Predicted Results: 79.76465

Prediction Number: 3
Nb: 4
RPM: 3750
Radius: 6.0
Target_Thrust: 3.0
Camber: 7.5
Camber_loc: 45.0
Thickness: 15.0
Collective: 5
Predicted Results: 80.289406
```

Example 14: Illustration of the output file generated by ANNMD using *strOutputFormat* = 'table'.

```
This file contains the ANN predicted results.
2       5000    7.5     2.0     5.0     50.0    11.0    3       77.98917
2       4500    6.5     2.0     5.0     50.0    12.5    5       79.76465
4       3750    6.0     3.0     7.5     45.0    15.0    5       80.289406
```

If *blnAccuracyScore* hold a boolean value of *True*, this output file generated by ANNMD will also include calculated values of the MAPE, the R_d^2 score, and the highest residual error. The highest residual error can be used to determine regions of the evaluation feature space where an ANN prediction model performs poorly. This *blnAccuracyScore* can only be used if the evaluation data file specified by *strEvalFile* contains a column of output data. Generally, a user may wish to perform predictions over new data where output data are not known a priori, however, this

included accuracy metric functionality can be very beneficial in the model development process. For example, a user may use this ANNMD functionality to evaluate a prediction model over the test data, determine regions of the input feature space where the model performs poorly, then add additional input feature points in these regions to retrain a prediction model using ANNPM.

4.3 Demonstration

Due to the simplicity of ANNMD, detailed demonstrations of software capability have already been provided in the previous section and will not be discussed in this section. To run ANNMD, the user need only run the ANNMD.py script located in the 'Source' directory. The script should be run in the directory containing the user-specified namelist, examples of which can be found in the 'Test/Annmd' directory. A command line argument correspondent to the namelist file name is also necessary to run ANNMD. For example, if the user were to execute ANNMD from the 'Test/Annmd/full_data' directory, the command line execution would take the following form: 'python ../../../Source/ANNMD.py ANNMD.nml'.

Five examples are provided with AANNT that correspond to the same cases discussed previously in Section 3.3, and use ANN prediction models generated using ANNPM for each of the five cases. In general, these namelist files can be copied and modified by the user to accommodate the execution of any ANN prediction model generated by ANNPM.

5 Artificial Neural Network Sensitivity Analyses (ANNSA)

5.1 Process Summary

Since computer-based experiments are deterministic in nature, there is no statistical basis for quantifying the importance of input features on the output. The inherent error associated with physical experimentation is often used for significance testing (i.e., feature ranking) but in its absence (i.e., deterministic experiments), approaches for sensitivity analyses that are independent of the model form must be utilized. All sensitivity analyses discussed herein are model independent, meaning that they can be performed over any arbitrary prediction model since they leverage different sampling techniques. This is in contrast to model-dependent methods commonly associated with physical experiments, which train a number of prediction models without certain input features to determine the influence of the withheld input feature(s). Because of this model independence, computational cost associated with training additional prediction models is drastically reduced.

The third and final tool in the AANNT, called ANNSA, is a post-processing tool for conducting sensitivity analyses over a trained ANN prediction model created by ANNPM. This tool is structured to operate using a user-specified namelist, which follows the same structure as the namelists for both ANNPM and ANNMD with a user specification of input feature descriptions. In addition to the number, names, and minimum/maximum values of any continuous input features or outputs used in the modeling procedure performed by ANNPM, the precision of each continuous input feature must also be specified, due to each type of sensitivity analysis requiring the generation of new input feature spaces. This specified precision should match the precision of the training data used by ANNPM. The sensitivity analyses in ANNSA allow for categorical input features, however, they must be numeric (e.g., $N_b = 3, 4, 5$). The name and location of an ANN prediction model created by ANNPM are specified in the user namelist, as well as the type of sensitivity analyses wished to be performed. After performing the user-specified sensitivity analyses, ANNSA exports the results to a text file, which can then be interpreted by the user. The following sensitivity analyses are supported by ANNSA: Sobol' Sensitivity Analysis [27], Fourier Amplitude Sensitivity Test (FAST) [28,29], Random Balance Designs Fourier Amplitude Sensitivity Test (RBD-FAST) [30,31], and the Profile-Method [32].

All sensitivity analysis types, except for the Profile-Method, leverage different routines available within the open-source Python library, SALib [33]. These sensitivity analyses utilize different sampling techniques to calculate global-based sensitivity metrics. The sampling techniques are specific to the type of sensitivity analysis being conducted and, in general, use the prediction model output correspondent to each point in the sampling procedure to quantify the partial variances, V_k, V_{kl}, etc., and the total variance, V_{tot}, of a prediction model [34]. The first- and second-order global sensitivity indices , S_k, S_{kl}, for an input feature, k, for example, can then be calculated using:

$$S_k = \frac{V_k}{V_{tot}}, \quad S_{kl} = \frac{V_{kl}}{V_{tot}}, \qquad (k = 1, 2, ..., n; \quad l = 1, 2, ..., n), \tag{15}$$

where:

$$1 = \sum_{k}^{s} S_k + \sum_{k}^{s}\sum_{l}^{s} S_{kl} + \sum_{k}^{s}\sum_{l}^{s}\sum_{u}^{s} S_{klu} + ... + H.O.T. \tag{16}$$

These global sensitivity indices can then be compared to infer which input feature is most significant to the prediction model output (i.e., input feature with the largest sensitivity index).

The Profile-Method, proposed by Lek et al. [32], uses a qualitative sensitivity analysis approach, in contrast to the qualitative approach used by the previously mentioned sensitivity analyses. This method first divides all input features into an equal number of intervals, Q, which, when using a categorical input feature, will be equivalent to the number of discrete values of the categorical input feature. While maintaining the input feature in question at its lowest interval value, the other input features are successively incremented through their intervals and a mean output, $R_{q,k}$, is calculated over the prediction model outputs for each successive incrementation:

$$R_{1,k} = \frac{1}{Q}\sum_{q=1}^{Q} f(x_{1,k}, x_{q,k+1}, x_{q,k+2}, ..., x_{q,s}), \qquad (k = 1, 2, ..., n). \tag{17}$$

The input feature in question is then incremented to its successive interval value, $q = 2$, and the process is repeated to produce $R_{2,k}$. This procedure is carried out over the number of intervals, Q, and is repeated for all input features with the final mean response values being plotted to qualitatively determine which input feature has the largest effect on the response. An illustration of the Profile-Method with two input features is shown in Fig. 10 where it can be inferred that the input feature x_2 has a larger significance than x_1. Since this sensitivity analysis is qualitative, ANNSA uses the maximum and minimum values of $R_{q,k}$ to calculate a quantitative metric, ΔR, for sensitivity comparison between input features:

$$\Delta R_k = R_{q,k_{max}} - R_{q,k_{min}}. \tag{18}$$

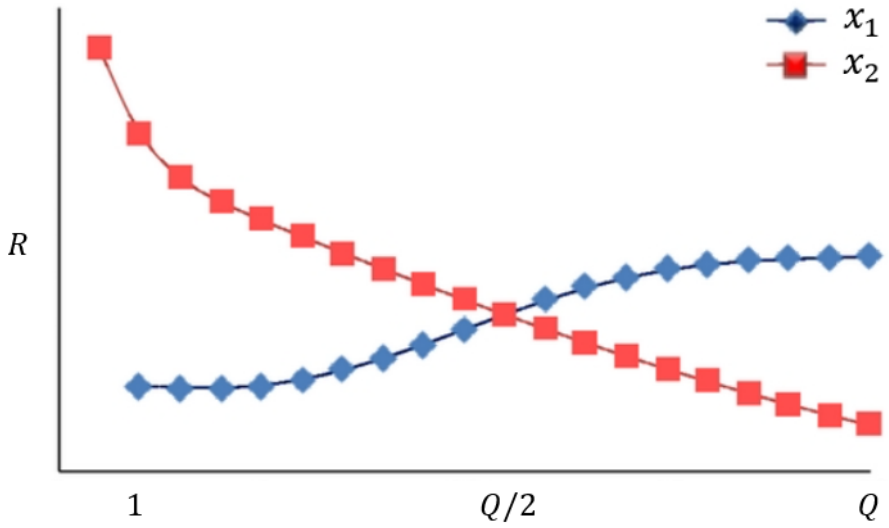


Figure 10: Illustration of the Profile-Method sensitivity analysis. Adapted from Shojaeefard et al. [3].

5.2 Setup and Execution

ANNSA is a standalone Python code, requiring only a user-generated namelist file, evaluation data, prediction models developed by ANNPM, and the installation of the following Python packages:

- sys
- os
- f90nml
- numpy
- pandas
- tensorflow
- SALib

The user is referred to the Anaconda user-guide page: `https://docs.anaconda.com/anaconda/user-guide/tasks/install-packages/` for instruction on package installation. This section of the manual explains the various user-specified inputs to the namelist as well as how to properly execute ANNSA.

5.2.1 User-specified Namelist

The general form of the user-specified ANNSA namelist consists of three sections correspondent to the input data, input feature description, and output data, respectively. The first section of the ANNSA namelist is the &InputData section, which is shown in Ex. 15.

Example 15: Illustration of the &InputData section of the user-specified ANNSA namelist.

```
&InputData
! Name of the model directory. Default value is './' or the current working directory.
strModelLocation = './'

! Name of the model file.
strModelName = 'Induced_Power.h5'
/
```

The &InputData section only contains two inputs, *strModelLocation* and *strModelName*, which correspond to the directory location and name of an ANN prediction model generated by ANNPM, respectively.

Similar to all tools within AANNT, the user-specified namelist also contains a section to describe the input features, &InputFeatureDescription. An example of this section can be seen in Ex. 16. Since ANNSA uses a prediction model generated by ANNPM for sensitivity analysis routines, the data over which sensitivities are calculated must first be manipulated to a form recognizable to the model. This &InputFeatureDescription section follows the exact form as the &InputFeatureDescription from the ANNMD namelist and the various inputs to this section will not be discussed here for brevity. The only difference for the &InputFeatureDescription section in the ANNSA namelist is that it contains the input, *fltContinuousInputPrecision*, which dictates the number of decimals each continuous input feature has. ANNSA internally generates input feature spaces relevant to each type of sensitivity analysis and this precision specification ensures that these input feature

spaces have the same precision as what the ANN prediction models are capable of. Values of *fltContinuousInputPrecision* should match the precision of the input training data used by ANNPM for ANN model development.

Example 16: Illustration of the &InputFeatureDescription section of the user-specified ANNSA namelist.

```
&InputFeatureDescription
! Integer value for the number of continuous features.
nContinuousInputs = 7

! Continuous input names.
strContinuousInputNames = 'RPM', 'Radius', 'Target Thrust', 'Camber',
'Camber_loc', 'Thickness', 'Collective'

! Boolean to normalize continuous inputs. The default value of True is recommended.
blnNormalizeContinuousInputs = True

! Lower values of each continuous input.  Only needed
! if blnNormalizeContinuousInputs = True.
fltContinuousInputRangeLow = 6, 3000, -20, -.33, 0.5, -5, 0, 6

! Upper values of each continuous input. Only needed
! if blnNormalizeContinuousInputs = True.
fltContinuousInputRangeHigh = 12, 6000, 0, 1, 1.5, 10, 10, 15

! Number of decimals for each continuous input.
fltContinuousInputPrecision = 2, 0, 1, 1, 1, 1, 2, 2

! Integer value for the number of categorical inputs. The default value is 0.
nCategoricalInputs = 1

! Name of the categorical input. Only needed if nCategoricalInputs != 0.
strCategoricalInputNames = 'Nb'

! Category values of the categorical input. Only needed if nCategoricalInputs != 0.
strCategoryValues_Nb = '3', '4'

! Number of model outputs. The default value is 1.
nModelOutputs = 1

! Name of the output functions. The default value is 'output'.
strOutputFunctionNames = 'Induced_Power'

! Boolean to normalize the outputs. The default value of False is recommended.
blnNormalizeOutput = False

! Lower values of each output. Only needed if blnNormalizeOutput = True.
fltOutputRangeLow = 0, 0

! Upper values of each output. Only needed if blnNormalizeOutput = True.
fltOutputRangeHigh = 200, 200
/
```

The last section in the ANNSA namelist is the &OutputData section, shown in Ex. 17. This section deals with the types of sensitivity analyses to be conducted, which are specified using *strSensitivityAnalysisType*. Options include 'Sobol', 'FAST', 'RBD-FAST', 'Profile-Method', and 'all'. Multiple types of sensitivity analyses can also be specified at one time (e.g., *strSensitivityAnalysisType* = 'Sobol', 'FAST'). The various supported sensitivity analysis routines have been discussed in Section 5.1, and the user is referred to the references listed in Section 5.1 for details pertaining to each type

of sensitivity analysis supported by ANNSA.

Example 17: Illustration of the &OutputData section of the user-specified ANNSA namelist.

```
&OutputData
! Sensitivity analysis type. The default value is 'Sobol'.
! Options are 'Sobol', 'FAST', 'RBD-FAST', and 'Profile-Method'.
strSensitivityAnalysisType = 'all'

! Test flag to seed the sample spaces for regression testing. Should be set to
! the default value of False for application.
blnTest = False

! Output summary file name. The default value is 'Sensitivity_Results.out'.
strOutputSummaryFileName = 'Sensitivity_Results.out'
/
```

In general, all sensitivity analysis types produce very similar results and the 'Sobol', 'FAST', and 'Profile-Method' sensitivity analysis types are recommended for regression problems, especially those containing a numeric categorical input feature. Since input feature space generation typically involves an iteration procedure, the input, *blnTest*, can be used to set a seed value to ensure the generated input feature spaces are identical with each ANNSA execution. This input is used for development testing, and should maintain a value of *False* for user execution.

Lastly, the input, *strOutputSummaryFile*, can be used to specify the name of the output file produced by ANNSA. This output file contains the first-order and total sensitivities of each input feature for the 'Sobol' and 'FAST' sensitivity analysis types, if specified by the user. Each sensitivity result is a fraction of unity, and larger magnitudes mean that a particular input feature has a higher influence on the ANN prediction model. The total sensitivity results included in this output for the 'Sobol' and 'FAST' sensitivity analysis types include first-, second-, and higher-order sensitivity results, and the difference between the total sensitivity result and the first-order sensitivity analysis result for a particular input feature is equivalent to all second- and subsequent-order sensitivities. Only first-order sensitivity analysis results are supported for the 'RBD-FAST' method. The results from the 'Profile-Method' can be interpreted as first-order sensitivities but are the difference between maximum and minimum response of each input feature at each interval, as discussed in Section 5.1.

5.3 Demonstration

To run ANNSA, the user need only run the ANNSA.py script located in the 'Source' directory. The script should be run in the directory containing the user-specified namelist, examples of which can be found in the 'Test/Annsa' directory. A command line argument correspondent to the namelist file name is also necessary to run ANNSA. For example, if the user were to execute ANNSA from the 'Test/Annsa/full_data' directory, the command line execution would take the following form: 'python ../../../Source/ANNSA.py ANNSA.nml'.

Since ANNSA does not support more than one numerical categorical input feature, only four examples, correspondent to four of the cases from Section 3.3 are provided: the full data case, the full data case with scaled outputs, the full data case with two outputs, and the reduced input case with no categorical input feature. Since the output result file is similar for the two cases, only results and interpretation of results from the full data case will be explained. The output file from

executing ANNSA in the 'Test/Annsa/full_data' directory is shown in Ex. 18.

Example 18: Illustration of the output file generated by ANNSA and specified by the namelist input, *strOutputSummaryFileName*.

```
This file contains the ANN sensitivity results.

Sobol Index Results for Induced_Power:
S1      0.0037266910482248905   0.02819834467838821     0.033822782159579186    0.13706374894524992
0.00333429492378461     0.003036786536123205    0.0002681453512165482   0.6300110979390627
ST      0.02420618478449911     0.04294698205089024     0.055245870852184464    0.24546444105761622
0.006144216325305319    0.005986536321355081    0.0017470788671428794   0.7693515683372419

FAST Results for Induced_Power:
S1      0.01016796714756741     0.018114807848980164    0.033611045558403004    0.16533352438627755
0.0030491680250180992   0.003423958472980088    0.0001681439738338555   0.6634482587460128
ST      0.024784675886210583    0.03674455007389321     0.06073968574276856     0.2534424407726249
0.007762611032796629    0.008568278048312727    0.004530560974836506    0.7840943971634666

RBD-FAST Results for Induced_Power:
S1      0.018829426138984914    0.021174538196349627    0.022634163955036472    0.16234013377403728
0.004330999236934037    0.006923689145822913    -0.0043653127338205075  0.6351246931114976

Profile-Method Results for Induced_Power:
dV      8.688232421875          9.00040054321289        7.340702056884766       32.93979263305664
9.16146469116211        6.071346282958984       3.7981796264648438      57.89155578613281
```

Each of the eight columns of data for each of the sensitivity analysis types correspond to each of the eight input features outlined in Table 1, starting with the categorical input feature, N_b, followed by the categorical input features in the order specified by *strContinuousInputNames* and shown in Ex. 16. As shown in Ex. 18, the last column of data for each of the sensitivity analysis types holds the largest sensitivity value, indicating that the last continuous input feature, θ_0, has the largest effect on the P_I prediction model, followed by the fourth column, correspondent to the third continuous input feature, T_{design}. It should be noted that the ANN prediction models used for this example sensitivity analysis were trained using only five epochs, meaning that these example results are not physically accurate and are provided for demonstration purposes only.

References

1. Thurman, C. S., *Surrogate Modeling and Characterization of Blade-Wake Interaction Noise for Hovering sUAS Rotors using Artificial Neural Networks*, Ph.D. thesis, University of Maryland, College Park, MD, July 2022.

2. Thurman, C. S. and Zawodny, N. S., "Aeroacoustic Characterization of Optimum Hovering Rotors using Artificial Neural Networks," *VFS International 77th Annual Forum & Technology Display, Virtual*, 2021.

3. Shojaeefard, M. H., Akbari, M., Tahani, M., and Farhani, F., "Sensitivity Analysis of the Artificial Neural Network Outputs in Friction Stir Lap Joining of Aluminum and Brass," *Advances in Materials Science and Engineering*, Vol. 2013, 574914, 2013.

4. Alpaydin, E., *Introduction to Machine Learning*, MIT Press, Cambridge, MA, 2004.

5. Fang, K.-T., Li, R., and Sudjianto, A., *Design and Modeling for Computer Experiments*, Chapman and Hall/CRC, New York, NY, 2005.

6. Matheron, G., "Principles of geostatistics," *Economic Geology*, Vol. 58, No. 8, 1963, pp. 1246–1266.

7. Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P., "Design and Analysis of Computer Experiments," *Statistical Science*, Vol. 4, No. 4, 1989, pp. 409–435.

8. Currin, C., Mitchell, T., Morris, M., and Ylvisaker, D., "Bayesian Prediction of Deterministic Functions, with Application to the Design and Analysis of Computer Experiments," *Journal of the American Statistical Association*, Vol. 86, No. 416, 1991, pp. 953–963.

9. Morris, M. D., Mitchell, T. J., and Ylvisaker, D., "Bayesian Design and Analysis of Computer Experiments: Use of Derivatives in Surface Prediction," *Technometrics*, Vol. 35, No. 3, 1993, pp. 243–255.

10. Rumelhart, D. E. and Hinton, G. E. and Williams, R. J., "Learning Internal Representations by Error Propagation," *in* D.E. Rumelhart, J.L. McClelland *Parallel Distributed Processing: Explorations in the Microstructure of Cognition Volume 1: Foundations*, MIT Press, Cambridge, MA, 1986.

11. Thurman, C. S. and Somero, J. R., "Comparison of Meta-Modeling Methodologies through the Statistical-Empirical Prediction Modeling of Hydrodynamic Bodies," *Journal of Ocean Engineering*, Vol. 210, No. 107566, 2020.

12. Thurman, C. S. and Somero, J. R., "Hydrodynamic Characterization of Bodies of Revolution through Statistical-empirical Prediction Modeling using Machine Learning," *Journal of Ship Research*, Vol. 66, June 2022, pp. 182–191.

13. Greenwood, E., "Estimating Helicopter Noise Abatement Information with Machine Learning," *AHS International 74th Annual Forum & Technology Display, Phoenix, AZ*, 2018.

14. Li, S. and Lee, S., "A Machine Learning-Based Fast Prediction of Rotorcraft Broadband Noise," *AIAA AVIATION 2020 Forum, Virtual*, AIAA Paper 2020-2588, June 2020.

15. Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning Representations by Back-propagating Errors," *Nature*, Vol. 323, 1986, pp. 533–536.

16. Duchi, J., Hazan, E., and Singer, Y., "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, Vol. 12, 2011, pp. 2121–2159.

17. Tieleman, T. and Hinton, G., "RMSprop: Divide the Gradient by a Running Average of Its Recent Magnitude," *Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning*, Technical report, 2012.

18. Bishop, C. M., *Neural Networks for Pattern Recognition*, Oxford University Press, New York, NY, 1996.

19. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, Vol. 15, No. 56, 2014, pp. 1929–1958.

20. Bouckaert, R. R., "Choosing between Two Learning Algorithms based on Calibrated Tests," *20th International Conference on Machine Learning, Washington, D.C.*, August 2003.

21. Schapire, R. E., "The Strength of Weak Learnability," *Machine Learning*, Vol. 5, 1990, pp. 197–227.

22. Breiman, L., "Bagging Predictors," *Machine Learning*, Vol. 24, 1996, pp. 123–140.

23. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X., "TensorFlow: Large-scale Machine Learning on Heterogeneous Systems," Available at tensorflow.org.

24. He, K., Zhang, X., Ren, S., and Sun, J., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile*, December 2015.

25. Kingma, D. P. and Lei Ba, J., "Adam: A Method for Stochastic Optimization," *3rd International Conference on Learning Representations, San Diego, CA*, May 2015.

26. Ramachandran, P., Zoph, B., and Le, Q. V., "Searching for Activation Functions," *arXiv*, Vol. 1710, No. 05941, 2017.

27. Sobol', I. M., "Global Sensitivity Indices for Nonlinear Mathematical Models and their Monte Carlo Estimates," *Mathematics and Computers in Simulation, The Second IMACS Seminar on Monte Carlo Methods*, Vol. 55, No. 1–3, 2001, pp. 271–280.

28. Cukier, R. I., Fortuin, C. M., Shuler, K. E., Petschek, A. G., and Schaibly, J. H., "Study of the Sensitivity of Coupled Reaction Systems to Uncertainties in Rate Coefficients. I Theory," *The Journal of Chemical Physics*, Vol. 59, 1973, pp. 3873–3878.

29. Saltelli, A., Tarantola, S., and Chan, K. P.-S., "A Quantitative Model-independent Method for Global Sensitivity Analysis of Model Output," *Technometrics*, Vol. 41, No. 1, 1999, pp. 39–56.

30. McKay, M. D., Beckman, R. J., and Conover, W. J., "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics*, Vol. 21, No. 2, 1979, pp. 239–245.

31. Iman, R. L., Helton, J. C., and Campbell, J. E., "An Approach to Sensitivity Analysis of Computer Models: Part I – Introduction, Input Variable Selection and Preliminary Variable Assessment," *Journal of Quality Technology*, Vol. 13, No. 3, 1981, pp. 174–183.

32. Lek, S., Belaud, A., Baran, P., Dimopoulos, I., and Delacoste, M., "Role of Some Environmental Variables in Trout Abundance Models using Neural Networks," *Aquatic Living Resources*, Vol. 9, No. 1, 1996, pp. 23–29.

33. Herman, J. and Usher, W., "SALib: An open-source Python library for Sensitivity Analysis," *The Journal of Open Source Software*, Vol. 2, No. 9, 2017.

34. Saltelli, A., Annoni, P., Azzini, I., Campolongo, F., Ratto, M., and Tarantoloa, S., "Variance Based Sensitivity Analysis of Model Output. Design and Estimator for the Total Sensitivity Index," *Computer Physics Communications*, Vol. 181, No. 2, 2010, pp. 259–270.

35. Gillian, R. E., "Aircraft Noise Prediction Program User's Manual," NASA TM 84486, 1982.

36. Lopes, L. and Burley, C., "ANOPP2 User's Manual: Version 1.2," NASA TM 2016-219342, 2016.

37. Montgomery, D. C., *Design and Analysis of Experiments*, John Wiley & Sons, 1984.

Acronyms and Abbreviations

AANNT	The ANOPP2 Artificial Neural Network Tool
AAVP	Advanced Air Vehicles Program
ADoET	The ANOPP2 Design of Experiments Tool
ANN	Artificial Neural Network
ANNMD	Artificial Neural Network Model Deployment
ANNPM	Artificial Neural Network Prediction Modeling
ANNSA	Artificial Neural Network Sensitivity Analyses
ANOPP	Aircraft NOise Prediction Program
ANOPP2	Aircraft NOise Prediction Program 2
ARMD	Aeronautics Research Mission Directorate
DoE	Design of Experiments
ELU	Exponential Linear Unit
GELU	Gaussian Error Linear Unit
HDF	Hierarchical Data Format
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MASE	Mean Absolute Scaled Error
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
MSLE	Mean Squared Logarithmic Error
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RVLT	Revolutionary Vertical Lift Technology
SLP	Single-layer Perceptron

Glossary

Aeronautics Research Mission Directorate (ARMD)

NASA aeronautics has made decades of contributions to aviation. Every U.S. commercial aircraft and U.S. air traffic control tower has NASA-developed technology on board that helps improve efficiency and maintain safety. Research conducted by ARMD directly benefits today's air transportation system, the aviation industry, and the passengers and businesses who rely on aviation every day. ARMD scientists, engineers, programmers, test pilots, facilities managers and strategic planners are focused on aviation's future. They design, develop and test advanced technologies that will make aviation much more environmentally friendly, maintain safety in more crowded skies, and ultimately transform the way we fly. (`https://www.nasa.gov/aeroresearch/about-armd`).

Aircraft NOise Prediction Program (ANOPP)

A NASA computer program to predict aircraft component and system noise [35]. ANOPP has been implemented in three ANOPP2 modules.

Aircraft NOise Prediction Program 2 (ANOPP2)

Second generation ANOPP. ANOPP2 employs a mixed-fidelity framework to incorporate methods of differing fidelity allowing semiempirically-based, low-resolution methods, such as those found in ANOPP, to be combined with higher resolution, CFD-based methods required for better understanding of the noise-generating mechanisms [36].

Artificial Neural Network (ANN)

Machine Learning (ML) prediction model architecture, which aims to replicate the neurons in the human brain to develop input-output functional relationships.

Artificial Neural Network Model Deployment (ANNMD)

Functionality of AANNT that deals with ANN model deployment and accuracy metric calculation.

Artificial Neural Network Prediction Modeling (ANNPM)

Functionality of AANNT that deals with ANN prediction modeling.

Artificial Neural Network Sensitivity Analyses (ANNSA)

Functionality of AANNT that deals with ANN sensitivity analyses.

Design of Experiments (DoE)

A process used for planning an experiment so that appropriate data can be collected by statistical methods, resulting in valid and objective conclusions [37].

Exponential Linear Unit (ELU)

A nonzero gradient variant of the ReLU activation function used in regression problems with functional form given in Eq. 6.

Gaussian Error Linear Unit (GELU)

A nonzero gradient variant of the ReLU activation function used in regression problems with functional form given by:

$$\mathrm{GELU}(x) = \frac{x}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right].$$

Machine Learning (ML)

General class of prediction modeling techniques where functional input-output relationships are 'learned' over training data.

Mean Absolute Error (MAE)

A cost function commonly used in regression problems that can be described as the mean of the absolute values of the difference between model predictions and true output data. The functional form of this cost function is expressed in Eq. 7.

Mean Absolute Percentage Error (MAPE)

A metric used for calculating the error between model predictions and true output data. The functional form of this cost function is expressed by:

$$\mathrm{MAPE} = \frac{1}{n}\sum_{j=1}^{n}\left|\frac{y_j - \hat{y}_j}{y_j}\right|.$$

Mean Absolute Scaled Error (MASE)

A cost function commonly used in regression problems that can be described as the mean of the absolute values of the difference between model predictions and true output data divided by the mean value of the true output data. The functional form of this cost function is expressed by:

$$\mathrm{MASE} = \frac{1}{n}\sum_{j=1}^{n}\left|\frac{y_j - \hat{y}_j}{\bar{y}_j}\right|.$$

Mean Squared Error (MSE)

A cost function commonly used in regression problems that can be described as the mean of the squared values of the difference between model predictions and true output data. The functional form of this cost function is expressed in Eq. 8.

Mean Squared Logarithmic Error (MSLE)

A metric used for calculating the error between model predictions and true output data. The functional form of this cost function is expressed by:

$$\mathrm{MSLE} = \frac{1}{n}\sum_{j=1}^{n}\log\left(\frac{y_j + 1}{\hat{y}_j + 1}\right)^2.$$

Multilayer Perceptron (MLP)

ANN architecture that contains two or more hidden layers of neurons.

Rectified Linear Unit (ReLU)

A type of ANN activation function commonly used in regression problems with the following form:

$$\mathrm{ReLU}(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0. \end{cases} \tag{19}$$

Revolutionary Vertical Lift Technology (RVLT) Project

NASA's vision for vertical lift vehicles is to capitalize and improve unique vertical capabilities to greatly benefit the Nation's growing civil flight requirements. (taken from `https://www.nasa.gov/aeroresearch/programs/aavp/rvlt`).

Root Mean Squared Error (RMSE)

A cost function commonly used in regression problems that can be described as the square root of the mean of the squared values of the difference between model predictions and true output data. The functional form of this cost function is expressed by:

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}.$$

Single-layer Perceptron (SLP)

ANN architecture that contains one hidden layer of neurons.

The ANOPP2 Artificial Neural Network Tool (AANNT)

Suite of user-friendly standalone ANN tools within the ANOPP2 software that deals with prediction modeling, model deployment, and sensitivity analyses.

The ANOPP2 Design of Experiments Tool (ADoET)

DoE routine functionality within ANOPP2 that pertains to the development of modern, space-filling DoE regions of experimentation.

Index

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704–0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-11-2022	Technical Memorandum	

4. TITLE AND SUBTITLE

The ANOPP2 Artificial Neural Network Tool (AANNT) Reference Manual

Version 1.5

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Thurman, Christopher S.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

664817.02.07.03.02.01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER

L–XXXXX

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)

NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

NASA–TM–20220014856

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified-Unlimited
Subject Category 00
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

An electronic version can be found at http://ntrs.nasa.gov.

14. ABSTRACT

This manual documents version 1.5 of the ANOPP2 Artificial Neural Network ToolAANNT developed at the NASA Langley Research Center Aeroacoustics Branch. The AANNT is a suite of standalone software that provide the capabilities of generating optimal artificial neural network prediction models, deploying these prediction models over arbitrary, user-defined data, and conducting sensitivity analyses over these prediction models. The AANNT has been developed for user simplicity and requires only prerequisite Python package installation and user-defined namelist files for execution.

This application programming interface (API) is part of a larger toolkit called the Aircraft NOise Prediction Program 2 (ANOPP2). The goal of ANOPP2 is to provide the ability to independently: (1) assess aircraft system noise; (2) assess aircraft component noise; and (3) evaluate aircraft noise reduction technologies and flight procedures. Additionally, ANOPP2 is designed to provide a capability for understanding the fundamental physics involved in noise generation to support experiments and flight demonstration activities.

As a component of ANOPP2, the AANNT and this document may be included as part of the ANOPP2 distribution, or they may be provided independently of that distribution.

15. SUBJECT TERMS

Machine Learning, Prediction Modeling, Artificial Neural Networks

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Information Desk (help@sti.nasa.gov)
U	U	U	UU	47	19b. TELEPHONE NUMBER *(Include area code)* (757) 864-9658

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18