Reconstructing Aerodynamic Flows Using Deep Learning

Written: By Michael Ishak
Supervised By: Dr Sean Symon
Date Completed: 03/05/2022
Word Count: 10012

This report is submitted in partial fulfilment of the requirements for the MEng Aeronautics and Astronautics, Faculty of Engineering and the Environment, University of Southampton.

Declaration

I, Michael Ishak, declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

I can confirm that:

1. This work was done wholly or mainly while in candidature for a degree at this University.

2. Where any part of this thesis has previously been submitted for any other qualification at this University or any other institution, this has been clearly stated;

3. Where I have consulted the published work of others, this is always clearly attributed;

4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;

5. I have acknowledged all main sources of help;

6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;

7. None of this work has been published before submission.

Contents

List of Tables

List of Algorithms

List of Figures

Abbreviations

POD	Proper Orthogonal Decomposition
CFD	Computational Fluid Dynamics
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
SVR	Support Vector Regression
CPU	Central Processing Unit
GPU	Graphics Processing Unit
CW	Curds & Whey
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
ADAM	Adaptive Moment Estimation Algorithm
AdaGrad	Adaptive Gradient Algorithm Method
AdaDelta	Adaptive Delta
RMSProp	Root Mean Squared Propagation
PIV	Particle Image Velocimetry
RAM	Random Access Memory
aMSE	Average Mean Squared Error
aRRMSE	Average Relative Root Mean Squared Error
MAE	Mean Absolute Error
R^2	Coefficient of Determination

All symbols are defined in the text when they first appear.

Reconstructing Aerodynamic Flow using Deep Learning

Michael Ishak mi1g19@soton.ac.uk
Department of Aerospace Engineering
University of Southampton
Southampton, United Kingdom

Editor: Dr. Sean Symon

Abstract

Since its advent, artificial intelligence has made major strides, and with the progression in the sub sector of deep learning, improvements in neural networks and modelling have been significant, allowing for artificial intelligence to be a multi-disciplinary tool. It is important to understand the full picture behind data, as such it is necessary to be able to reconstruct a flow field from limited measurements. End-to-end mapping between the raw data from the sensors and the flow field, with minimal processing is used as the test data. In this report, we use an artificial neural network model to perform a super-resolution analysis to reconstruct two-dimensional laminar flow in a Hilbert space. It is assumed that no prior knowledge is known and that the estimation and training methods is completely data driven. We use this model in fluid mechanics with remarkable accuracy, showing that this mathematical architecture is ideal in fields where measured data is often limited.

Keywords: Support Vector Regression (SVR), Convolutional Neural Networks (CNN), Artificial Neural Networks (ANN), Proper Orthogonal Decomposition (POD), sensors, wakes, fluid dynamics, deep learning

1. Introduction

Deep Learning is the science of statistical models and algorithms that computer systems use to undergo tasks without using specific instructions, relying on inference and patterns instead. It is described as "a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy" (IBM Education, 2020). This highlights the importance of deep learning for this project as it allows for a computer to follow a set way of thinking such that it can estimate an accurate output which makes clear the data inputted as shown in Fig. 1. In its essence, deep learning is a computational method based upon statistics and is put in software to reveal patterns in a given dataset.

Figure 1: An illustration showing the aims of this project, to transform a low-resolution image to a high-resolution image by predicting missing data (Minds, 2021)

The continuation and refinement of low-resolution data using deep learning can be critically enabling for applications across the physical sciences as accurate flow estimation can aid in industry practices, such as crafting fuel-efficient auto-motives. This concept also carries weight in the physical sciences as this methodology is used to study blood flow and climate sciences where sensor measurements are limited and the prediction of fluid flow provides utility.

The idea of refining airflow data through deep learning is a very important concept in aerodynamics as it allows for a practitioner of computational fluid dynamics to analyse the numerical data and to elucidate the airflow physics from the image. Furthermore, the significance of this is further highlighted when it is realised that the procedure preceding the use of deep learning was knowledge-driven rather than data driven. Engineers would lean on their own understanding of airflow to predict and estimate what the data should be according to theory which tends to hold random errors such as human errors. This evolution in the industry has meant that machines can compute several equations and create statistical models based on patterns that exist in the data to predict and extrapolate the test data. Not only is the initial prediction of the machine more likely to yield better results, owing to the nature of deep learning, the more experience and data that the statistical model is fed ('training data'), the algorithms automatically improve, improving the output over time.

It is also mentionable that with advancements in technology, more training data become readily available, allowing for improvements in training deep learning models. As such, the mathematical architecture behind reconstructing aerodynamic flows using deep learning is a vital tool that could be used to further our research in many fields other than engineering.

This project's experimental data collected was procured by analysing laminar airflow across a NACA0018 airfoil at an angle of attack of $\alpha = 0°$ (Symon et al., 2019). Camera sensors set in a wind tunnel measure the air velocity around the airfoil in 2 dimensions with a focus on the wake of the airfoil. The purpose of this is to analyse and observe the airflow speed and the induced vortexes. This project aims to use low-resolution fluid data and reconstruct it in a high-resolution flow field using a multi-output regression method. After comparing different methods, SVR (see section 3.1.2.) was chosen because it has been shown to have better predictive performance when compared against statistical and kernel methods (Tsoumakas et al., 2014) (Burnham et al., 1999) (Han et al., 2012) which can be acredited to considering the underlying relationship between targets and features as well as how the targets are related to each other. This, therefore, guarantees a better representation and interpretation of real-world problems and data (Kocev et al., 2009) (Tuia et al., 2011). Another advantage of a multi-target approach is that it can produce a simpler model with a greater computational efficiency (Kocev et al., 2009).

The remainder of this report is organized as follows. In Section 2, prior work in flow field reconstruction are discussed and analysed. In section 3, modern multi-output regression methods are presented as either problem transformation or algorithm adaptation methods. In Section 4, an explanation of the empirical flow field data is given as well as how the data was procured. In Section 4, an insight into the necessary computing concepts is given. In Section 5, details of the methodology is given and the justification for the network and its parameters are outlined thoroughly. An evaluation of the method and the parameters chosen is made in Section 6, highlighting the advantages and disadvantages of this approach and what changes could be made to improve the results. Section 7 looks into the future of flow field reconstruction and a summary of the paper with some conclusions and possible

ideas for future research into flow field reconstruction is given in Section 8.

1.1 Project Aims and Objectives

The structure of this project consists of:

1. An in-depth insight and survey into modern deep learning techniques as well as explaining how they are derived. The advantages and disadvantages of each technique is discussed until a decision for which method to opt for is made

2. A review of the experimental procedure done to procure the aerodynamic flow field data used in this paper and what this means for the deep learning model proposed.

3. Highlighting the relevant software used and the concepts required to be able to recreate the model.

4. A thorough explanation of the method used to reconstruct aerodynamic flow, where reasons for the modelling decisions are given.

5. A comparison between expectations and results. This will be done by discussing the advantages and limitations of the model and how this model could be improved in the future.

6. Conclusions and a discussion into the future of deep learning and flow field reconstruction.

With these objectives in mind, the next section will analyse previous work done to reconstruct flow fields. The papers in this section use a mixture of Proper Orthogonal Decomposition (POD) and deep learning. POD is a non-linear numerical method that reduces the complexity of computer intensive simulations such as computational fluid dynamics (CFD). As such, an argument will be made in favor of using more simple deep learning techniques, to reconstruct flow rather than computationally complex methods.

2. Prior work in Flow Field Reconstruction

Owing to its far-reaching applications, flow field reconstruction is a rich field with research done into it over the past half century. In this section, a brief overview of the most relevant works are provided and are organised into three groups. This includes an insight into feed-forward ANNs that uses CFD and POD input to mimic the target problem, feed-forward ANN to deal with target features directly and feed-forward back propagation CNN.

2.1 Tackling the Target Features directly using ANN Regression

Feed-forward ANNs have been the staple for flow field reconstruction for the last half century (Yu and Hesthaven, 2018b). These ANNs use a schematic similar to that in Fig 2:

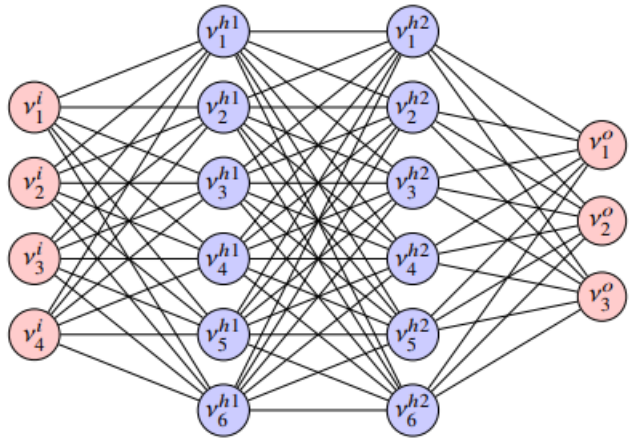


Figure 2: Multi-layer Feed-forward Neural Network (Erichson et al., 2020)

The aim with this method is to learn the relationship that maps the sensor measurements $\mathbf{s} \in \mathbb{R}^p$ to the flow field $\mathbf{x} \in \mathbb{R}^m$ data (Erichson et al., 2020). As a result of the sensor measurements being collected via a sampling process from the flow field, $p \ll m$ and as such, this entire process can be described as

$$\mathbf{s} = \mathbf{H}(\mathbf{X}), \tag{1}$$

where $\mathbf{H} : \mathbb{R}^m \to \mathbb{R}^p$ is an operator for the measurement sensor. Now, to find the flow field estimate, the creation of an inverse model (Li et al., 2020) of the process which produces field $\mathbf{x}$ from measurements $\mathbf{x}$ is described as

$$\mathbf{X} = \mathbf{G}(\mathbf{s}), \tag{2}$$

where $\mathbf{G} : \mathbb{R}^p \to \mathbb{R}^m$ is a non-linear forward operator. Owing to $\mathbf{H}$ being highly non-linear (as is the nature of aerodynamic flow), there is not a direct inversion for the operator $\mathbf{H}$ to obtain operator $\mathbf{G}$.

However, if we use a set of training data $\{x_i, s_i\}$, a function ξ to approximate the forward passing operator $\xi : s \to \hat{x}$ to map a few measurements to the estimated state $\hat{x}$ of the flow,

$$\hat{\mathbf{X}} = \xi\left(s\right), \tag{3}$$

such that any strong deviation from the true output is small (in a Euclidean sense regarding the sensor measurements)

$$\|\xi(s) - \mathbf{G}(\mathbf{s})\|_2^2 < \varepsilon, \tag{4}$$

where ε is a small positive real number. This method is a common practice in deep learning and is increasingly used for flow field reconstruction and prediction. In particular, this method is a traditional method in the application of super-resolution in which this paper aims to recreate for aerodynamic flow.

In this method, an artificial neural network is used to learn the mapping function between the sensor measurements and the flow field. Fig. 2 shows a sketch for the proposed model for the aerodynamic flow reconstruction and the network architecture can be expressed concisely as:

$$s \rightarrow FHL \rightarrow SHL \rightarrow ... \rightarrow OL \rightarrow \hat{x} \tag{5}$$

where FHL stands for the first hidden layer, SHL is the second hidden layer and OL is the output layer.

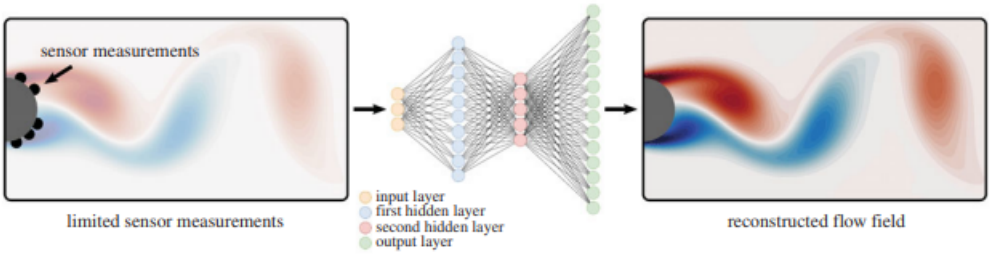


Figure 3: Illustration of end to end mapping of the sensor measurements $\mathbf{s} \in \mathbb{R}^{\text{Sensor No.}}$ to flow field $\hat{\mathbf{x}}^{\text{Outputs No.}}$ (Erichson et al., 2020).

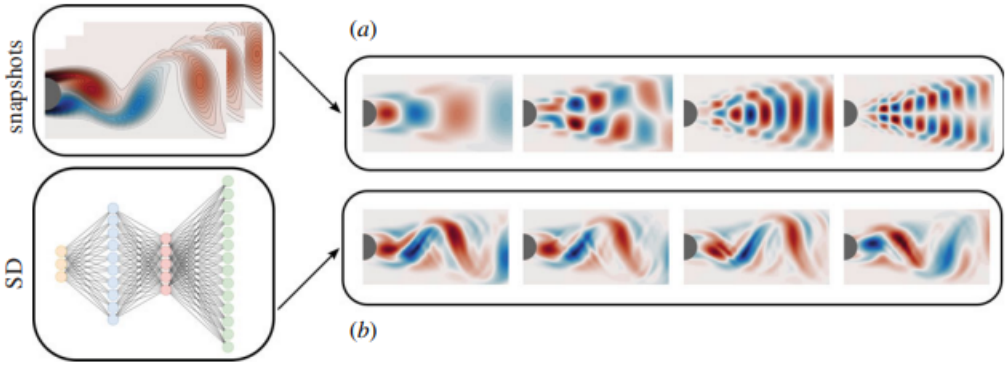


Figure 4: Illustration comparing POD nodes (a) to ANN (SD) nodes (b). By not constraining the nodes to be linear and orthogonal (as is enforced with POD), a more interpretable feature-space can be extracted from the input data as seen in (a) compared to (b). This can be used for the reconstruction of the state-space from limited data (Erichson et al., 2020).

Using a mathematical approach (in this case, SVR as described in Section 3.1.2) rather than computing methods for flow reconstruction has many advantages (Krithika Manohar, 2018). For example, the decoder used by Erichson et al (Erichson et al., 2020) uses a linear dense layer as his last layer and has a supervised learning framework for the low-dimensional euclidean space of the flow field to map the measurements to this same low-dimensional space. This is an advantage as it means that the approximation can be tailored to the sensor measurements and to the state space, preventing observability issues despite these two steps being disconnected. Another advantage is that the method allows for flexibility in how the data is gathered as it means that the data does not have to be linearly related to the state space, unlike many standard methods. Finally, the decoder network produces dynamic low-rank features that are interpretable and accurate.

Fig. 3 shows an example of an algorithm that uses an ANN. The learning is done by the shallow decoder (a term denoting a simple ANN and is represented as SD in Fig. 3 and 4) has elements that resemble the physical quantities (seen in Fig. 4a) which is unlike the POD-based modal approximation method that enforces orthogonality as shown in Fig. 4a (Erichson et al., 2020). This simple approach has high predictive performance as it considers physical properties that the flow has such as the gradient and divergence of the airflow. This also means that the outputs are interpretable, which is desireable.

The main limitation of this approach is standard to other data-driven methods, where the training data set is meant to be as representative as possible of the given system such that it is comprised of samples from the same distribution as the test data; however, a completely representative training data for extrapolating the data is not possible as the test data is empirical, where disturbances in the environment not accounted for can change the output.

2.2 Tackling Mimicked Target Features directly using Regression to improve Costs

Despite the predictive performance of using ANNs to tackle target problems directly, the computational complexity that comes with this method is high, meaning a CPU may struggle computing complex data sets. As such, CFD and POD were used to create multiple input and output variables to address observability issues such as overfitting (Bilbao and Bilbao, 2017) whilst reducing the computational costs.

POD and CFD are both techniques that have been introduced recently into ANNs to improve computational complexity at the cost of predictive performance, allowing for more complicated problems to be solved (Yu and Hesthaven, 2018a). Rather than directly dealing with expensive target problems, this method proposes to simplify the model data using CFD and POD. CFD code is used to generate finite volume target features of which the ANN uses as samples. POD is then used to compress and reduce the data into two. This is done as the size of the vector solutions is proportional to the mesh grid size used in CFD and is usually very large. This would have a very large negative impact on training as more input and output neurons need more hidden neurons, which in turn requires more samples to train the ANN. Therefore, these vectors need to be compressed, which is done using POD (Schmidt and Colonius, 2020). After this, the method continues as a normal ANN model that tackles the target problems directly, using a portion of the simplified solution data as the training set and the validation set. A test set checks the ANN accuracy whilst the validation set helps offset the effects of overfitting.

The main difference between mimicking the target problem and tackling the target problem directly is the type of data used in the model and the objectives of each ANN. Tackling the target problem directly means that this method has better predictive performance compared to mimicking the data because it uses empirical data rather than CFD data.

2.3 Using CNNs to Transform the Target Problem into a High Resolution Image

Another modern deep learning network more commonly used for complex data sets are CNNs. A CNN is used to represent the data as an image in order to identify and learn the spatial features in the data (Fukami et al., 2019).

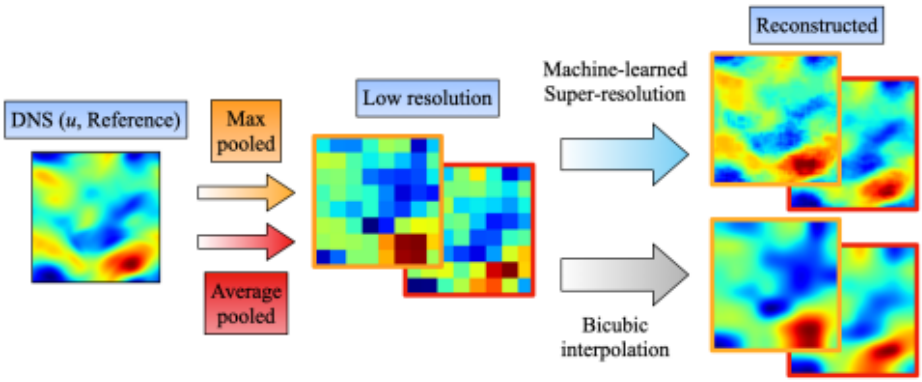


Figure 5: Max-pooling and average-pooling for the super-resolution reconstruction of the target flow field (Yani et al., 2019).

Models that use CNNs for multi-output regression predictions aim to do so using the framework illustrated in Fig. 5. It aims to learn the mapping between two data sets and to create a nonlinear regression function that can recreate and extrapolate the rest of the data. However, rather than relying heavily on pre-determined weights like ANNs, this method uses the data to find spatial features in the output and weights where weights are applied based on how saturated the output features are. Weights are calculated by:

$$\mathbf{w} = argmin_{\mathbf{w}} \left\| y - F\left(\mathbf{x} : \mathbf{w}\right) \right\|_2^2 \tag{6}$$

where $\mathbf{y}$ is the desired high-resolution output, $\mathbf{w}$ is the optimized weight and $F\left(\mathbf{x} : \mathbf{w}\right)$ is the value of the largest target variable.

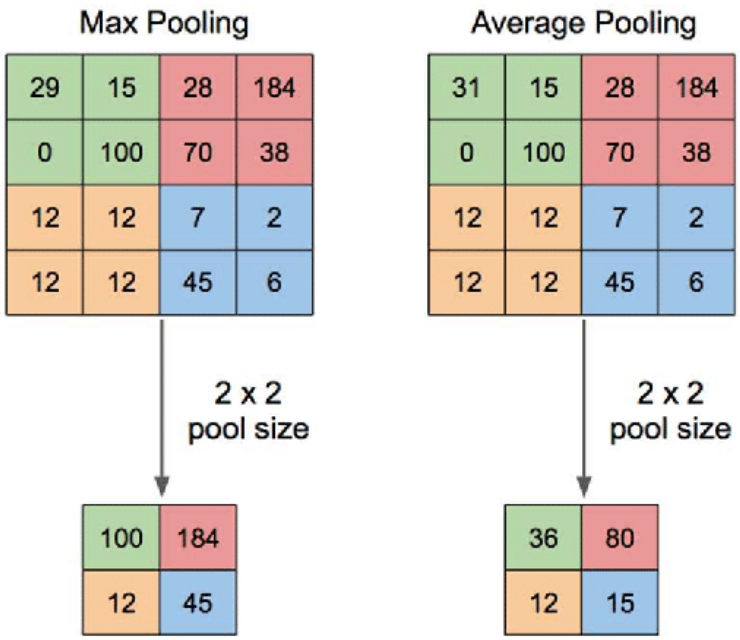


Figure 6: A depiction as to how max-pooling and average-pooling works (Fukami et al., 2019).

For example, in the paper by Fukami *et al.* (Fukami et al., 2019), his reference data was filtered using max-pooling and average-pooling such that the reference data matrix is reduced and has a lower resolution. This is because max-pooling takes the maximum value of each quadrant of the sample data and applies it to a single cell and continues to do this for every section of the sample data. Average-pooling does the same thing but for the mean value of each quadrant rather than the maximum value. This means that the data has reduced in resolution because there is less data in the new matrix. This is illustrated in Fig. 6.

Once the new low-resolution data is calculated, two test data inputs are created, meaning that overfitting is accounted for and is prevented. Another difference with using a CNNs compared to ANNs is how this test data is inputted. The test data is inputted into the model using an iterative method between input variable $q^{(l-1)}$ and output variable $q^{(l)}$ for this particular layer (denoted as l):

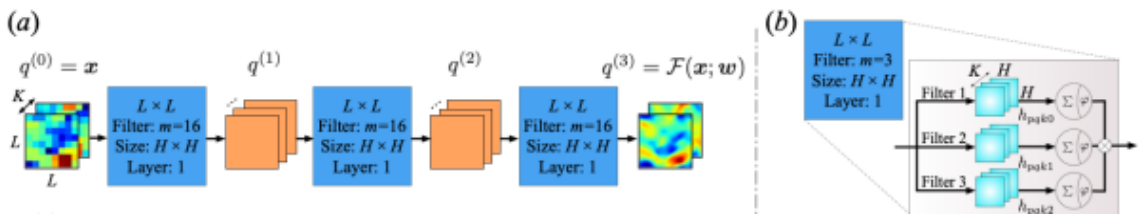


Figure 7: (a) Shows the architecture of the convolutional neural network (CNN) with two dimensional laminar flow and (b) shows the inner working of each CNN for a three filter set-up. (Fukami et al., 2019)

$$q_{ijm}^{(l)} = \phi\left(\sum_{k=0}^{K-1}\sum_{p=0}^{L-1}\sum_{s=0}^{L-1} q_{i+p,j+s,k} h_{pskm}\right), \tag{7}$$

where $q^{(l_max)} = F(\mathbf{x} : \mathbf{w})$ and using Fig. 7 a and b, H is the length of filter h, ϕ is the activation function, in this case the ReLU function (see Section 3.2.3), L is the number of features in each direction and K is the number of images that form the data. Once this is inputted, the data runs through the algorithm and filters to output a high resolution image.

As with ANNs, CNNs have advantages and disadvantages when applied to situation where the outputs are predictions. The main advantage for using CNNS are that they are cheap computationally as it detects spatial features and elements and uses the pattern between them to replicate and produce outputs. CNNs also share weights between kernels, meaning the weights are always optimized and they require less statistical training as it can implicitly detect complex non-linear relationships between dependent and independent variables (Tu, 1996).

The main disadvantage is the inability to be spatially invariant to the test data. Convolutional layers replicate the same kernel weight for each kernel across the entire input data to output a 2D matrix which is essentially an output of a replicated feature detector. By max-pooling this data, any data that is not the maximum value can be changed and the output stays the same, creating an 'invariance of activities', where invariance means that the output stays the same and activity is the output signal of a neuron. As such, max-

pooling loses valuable data and as such require large amounts of data to train the model. CNNs also do not encode relative spatial relationships, meaning that the core math behind the relationship is not learnt (Kondo et al., 2017). This means that aerodynamic properties in the flow are not learnt and are purely mirrored across the data set to predict data.

2.4 Which Methodology to use in this Project

Looking at prior work on flow field reconstruction, for this project, the data used is not complex and a simple model should suffice. Normally this would mean that a CNN would be perfect for this type of flow field, however, there is not enough training data to allow for the CNN to be accurate. As such, an ANN is opted for. The data used in this project has had POD (see Section 4.1) applied to simplify the data and reduce computational complexity. As such, tackling mimicked target problems is what this report will aim to describe.

The next section will look at which algorithm adaptation methods and activation functions will yield the most accurate predictions.

3. Multi-output Regression Methods

3.1 Algorithm Adaptation Methods

In this section, recent, modern multi-output regression methods are outlined as extensions of standard learning algorithms such as statistical methods, support vector machines and kernel methods.

These methods simultaneously predict all the targets using one model whilst being able to consider all internal relationships and dependencies between them. This is advantageous in several ways as it does not require the use of POD to use a low computationally complex model to predict data. Furthermore, by using one multi-target model, it is simpler to interpret when compared to other popular methods, especially when the variables are correlated (Kocev et al., 2009) (Srivastava and Solanky, 2003) (Similä and Tikka, 2007).

3.1.1 Statistical methods

Statistical methods are normally considered the first steps made to predict multiple real-valued outputs simultaneously. It takes advantage of the relationship between variables and uses this information to improve predictive performance.

One of the most popular and efficient ways to solve multi-output regression problems is to use the Curds and Whey (CW) method proposed by Breiman and Friedman (Breiman and Friedman, 1997). Essentially, when given d targets, $\mathbf{y} = (y_1, ..., \hat{y}_d)^T$ with least squares regressions $\hat{\mathbf{y}} = (\hat{y}_1, ..., \hat{y}_d)^T$, a more accurate predictor of $\tilde{y}_i$ of every y_i (Eqn. 8) is found when combining the OLS predcitors (Eqn. 9 and Eqn. 10), rather than the least squares, where $\bar{\mathbf{y}}$ and $\bar{\mathbf{x}}$ are sample means of $\mathbf{y}$ and $\mathbf{x}$ respectively:

$$\tilde{y}_i = \bar{y}_i + \sum_{k=1}^{d} b_{ik}(\hat{y_k} - \bar{y_k}), i \in \{1, ..., d\} \tag{8}$$

$$\tilde{y}_i = \bar{y}_i + \sum_{j=1}^{m} \hat{a}_{ij}(x_j - \bar{x_j}), \tag{9}$$

$$\{\hat{a_{ij}}\}_{j=1}^{m} = arg \min_{\{a_j\}_{j=1}^{m}} \left[\sum_{l=1}^{N} \left(y_i^{(l)} - \bar{y_i} - \sum_{j=1}^{m} a_j \left(x_j^{(l)} - \bar{x_j} \right) \right)^2 \right] \tag{10}$$

In Eqn. 10, $\hat{a}_{ij}$ is the estimated regression coefficient, and b_{ik} is the reduced-rank parameter that transforms the vector-valued OLS predictions ($\hat{\mathbf{y}}$) to the biased estimate ($\tilde{\mathbf{y}}$) which is found using the CW method, form of multi-target shrinking (or reduced-ranking).This maximises the relationship between $\hat{\mathbf{y}}$ and $\tilde{\mathbf{y}}$ as the predictions of the matrix $\mathbf{B} = [b_{ik}] \in \mathbb{R}^{d \times d}$, where $\mathbf{B} = \mathbf{T}^{-1}\mathbf{S}\mathbf{T}$ as $\mathbf{T}$ is a $d \times d$ matrix has rows that are the response coordinates between $\mathbf{y}$ and $\mathbf{x}$ and $\mathbf{S} = diag\,(s_1, ..., s_d)$, a diagonal shrinking matrix. Therefore, to find

$\mathbf{B}$, the CW method starts with transforming ($\mathbf{T}$), then shrinking it (multiplying the matrix by $\mathbf{S}$) then bringing it back to its original form (multiplying by $\mathbf{T}^{-1}$).

Recently, Simil and Tikka (Similä and Tikka, 2007) sought to solve the issue of input selection and shrinkage in multi-output linear regression. They presented a simultaneous variable selection method (L_2SVS), where the weight of an input is measured by a L_2-norm of the regression coefficients of the input values. To solve for the new regression coefficients, $\mathbf{W}$, an estimation of the minimum error sum of squares in relation to a sparsity constraint (how many zeros or values that do no significantly impact a calculation) must be solved:

$$\underset{\mathbf{w}}{min} f(\mathbf{W}) = \frac{1}{2} \|\mathbf{y} - \mathbf{x}\mathbf{W}\|_F^2 \tag{11}$$

in relation to

$$\sum_{j=1}^{m} \|\mathbf{w}_j\|_2 \leq r, \tag{12}$$

where F is a matrix that is the square root of the sum of the absolute squares of its elements (the Frobenius norm), i.e., $\|\mathbf{B}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{d} |b_{i,j}|^2}$. The equation uses $\|\mathbf{w}_j\|_2$ to measure the significance of the jth instance in the model and uses r as a free parameter to control the amount of reduced-ranking applied to the prediction (this is shrinkage). If $r \geq 0$ is large, the most optimal value of $\mathbf{W}$ is equal to the OLS solution, whereas smaller values of r can impose a sparse structure on $\mathbf{W}$, meaning that only some inputs are useful to use in the prediction.

3.1.2 Multi-output support vector regression

To deal with a scenario with a multi-output case, a single-output SVR can be used independently to every output but has some significant disadvantages as it does not take into consideration the possible relationships between the outputs. As such, several methods have been proposed to extend SVR to better manage a multi-output case scenario. To generalise,

minimizing Eqn. 13 needs to be done:

$$\frac{1}{2}\sum_{i=1}^{d}\|\mathbf{w}_i\|^2 + C\sum_{l=1}^{N} L\left(\mathbf{y}^{(l)} - \left(\phi\left(\mathbf{x}^{(l)}\right)^T \mathbf{W} + \mathbf{b}\right)\right), \tag{13}$$

where ϕ is a non-linear transformation to a higher dimensional Hilbert space (this is essentially an infinite dimensional Euclidean space), C is the trade-off parameter between error reduction and regularization. L is an insensitive loss function and the solution is $\mathbf{w}$ and b. The solutions are of the form: $\mathbf{W} = (\mathbf{w}_1, ..., \mathbf{w}_d)$, where it $\mathbf{W}$ is a $m \times d$ matrix and $\mathbf{b} = (b_1, ..., b_d)^T$.

An advantage of this multi-output SVR method is highlighted by Guangcan Liu et al. (Liu et al., 2009), by extending the SVR using the Cokriging (Chilès and Delfiner, 2012) method which exploits the relationships, previously unconsidered, by looking at the proximity in the space of factors and outputs. Through this method and with an appropriate choice of cross-covariances and covariance models, they showed that multi-variate SVR yields better predictions than single-target SVR models.

Recently, Xu et al (Xu et al., 2013). proposed a different way to extend least squares SVR for a multi-output scenario. This is done by finding the weights $\mathbf{W} = (\mathbf{w}_1, ..., \mathbf{w}_d)$ and bias parameters $\mathbf{b} = (b_1, ..., b_d)^T$ that minimizes the objective function:

$$\underset{\mathbf{W}\in\mathbb{R}^{n_h\times d},\mathbf{b}\in\mathbb{R}^d}{min} F(W, A) = \frac{1}{2}trace\left(\mathbf{W}^T\mathbf{W}\right) + \gamma\frac{1}{2}trace\left(\mathbf{A}^T\mathbf{A}\right), \tag{14}$$

$$s.t. \mathbf{Y} = \mathbf{Z}^T\mathbf{W} + repmat\left(\mathbf{b}^T, N, 1\right) + \mathbf{A}, \tag{15}$$

where $\mathbf{Z} = \left(\phi\left(\mathbf{x}^{(1)}\right), ..., \phi\left(\mathbf{x}^{(N)}\right)\right) \in \mathbb{R}^{n_h \times N}, \phi : \mathbb{R}^m \to \mathbb{R}^{n_h}$ is a mapping to a higher Hilbert space, with n_h dimensions. Here, *repmat* is a function that creates a large block matrix which has tiling copies of $\mathbf{b}$ and is defined as a $1 \times d$ matrix $\mathbf{b}$ repmat $\left(b^T, N, 1\right)$. $\mathbf{A} = (\mathbf{a}_1, ..., \mathbf{a}_d) \in \mathbb{R}_+^{N \times d}$ and is a matrix which is made up of slack variables (variables that are added to an inequality to make it into an equality), and $\gamma \in \mathbb{R}^+$ is a positive, real parameter.

3.1.3 Kernel methods

Kernels in deep learning are sets of different types of algorithms that are used for pattern recognition and are largely used to solve non-linear problems using a linear classifier. The way it solves classifications is by moving data values to a higher dimension (Hilbert space) as it means a linear classifier (which is essentially a linear line to separate points in a euclidean space) can use the Hilbert space to set this classifier in a more accurate position. Kernel methods tend to be used as activation functions in SVR models.

Micchelli and Pontil (Micchelli and Pontil, 2005) studied vector-valued learning (vectors that define parametric curves) and analysed the regularized least squares regression from a computer system's point of view. Through this, they generalised a respresentor method to allow for vector-valued learning.

Later, Baldassarre *etal.* (Baldassarre et al., 2012) studied regularized kernel methods for multi-target learning by filtering kernel matrices. They considered ridge regression a special case and different methods such as vector valued approaches as extension of the squared loss function boosting and other iterative methods. They claimed that ridge regression could be used as a low-pass filtering to apply to the kernel matrix and they suggest to use different types of spectral filtering so that matrices are defined in general, minimizing empirical risk.

Additionally, A'lverez et al. (Álvarez et al., 2012) reviewed kernel methods for vector-valued functions and attempted to connect regularization and the Bayesian prospective (a theorem that describes how the conditional probability of an output can be computed using

previous knowledge). By doing so, they provided a large selection of kernel choices to learn divergence-free and curl-free vector fields which is particularly useful at analysing aerodynamic flow.

3.1.4 Summary of the proposed multi-output regression models

Table 1: A summary of the different multi-output regression models.

	Method	**Main Ref.**	**Year Published**
Algorithm adaptation methods	Statistical methods	Izenman van der Merwe and Zidek Breiman and Friedman Similä and Tikka	1975 1980 1997 2007
	Multi-output SVR	Liu et al. Sanchez et al. Vazquez and Walter Xu et al.	2009 2004 2003 2013
	Kernel Methods	Baldassare et al. Micchelli and Pontil A'lverez et al.	2012 2005 2012

Using this literature, this paper hopes to use a suitable form of deep learning to reconstruct aerodynamic flow from a set of experimental data (see Section 4.1). The flow field data was obtained from an experiment carried out by Symon et al. (Symon et al., 2019) (see Section 4.1). This data was mirrored and had proper orthogonal decomposition (POD) implemented on it, meaning that employing the multi-output support vector regression approach is ideal as it uses biases and regressors to make predictions based off of patterns in the data between an input x and output y. Table 1 highlights that there are two multi-output SVR methods, the approach most suitable for this type of deep learning process is the algorithm adaptation method. If we consider the problem transformation method, the idea of using the LS-SVR algorithm means that the problem has been transformed into multiple single-output problems. This means that the computational costs are greater, as it has to run the same procedure N times and it does not take into consideration the relationship between each output. These two disadvantages are quite significant and as such, using

the algorithm adaptation method, which looks at the proximity of the factors and outputs would yield better predictions and not be as costly. The performance is further improved when a co-variance model such as the Sequential model (Chan, 2007) is implemented in the network (see section 5.1.2).

Now, the aim is to discuss the compatibility of different activation functions and optimization methods to this project as well as using this information such that the predictive performance, interpretability and computational cost of the model proposed are as optimal as possible.

3.2 Activation Functions

Activation functions are pivotal to the performance of a neural network model as they determine the output. The purpose of this section is to look at three popular regression activation functions (Linear, Logistic Sigmoid/Tanh Unit Based, and Rectified Linear Activation) to show why the ReLU activation function is ideal for this problem.

3.2.1 Linear Activation Functions

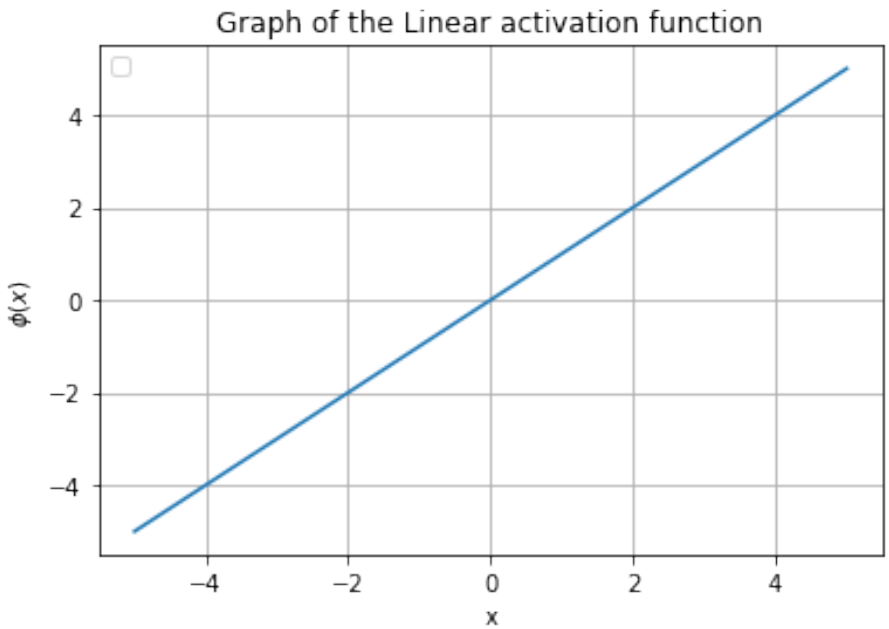


Figure 8: An illustration of the linear function's effect on the relationship between the independent and dependent variable.

A linear activation function is a simple function that yields $c \times x$ with an input of x, where c is a constant. This is illustrated in Fig. 8 for $c = 1$. It is notable that the linear activation function does not add non-linearity into the neural network and as such, the network that uses this activation function outputs the final layer as a linear function despite having multiple layers. As a result, this function should not be used for accurate predictions on convex data.

3.2.2 Logistic Sigmoid/Tanh Unit Based Activation Functions

To introduce the non-linearity that the linear activation function fails to do, the Logistic Sigmoid and Tanh activation functions are both viable options. The Logistic Sigmoid Function can be written as :

$$\text{Logistic Sigmoid(x)} = \frac{1}{1 + e^{-x}}. \tag{16}$$

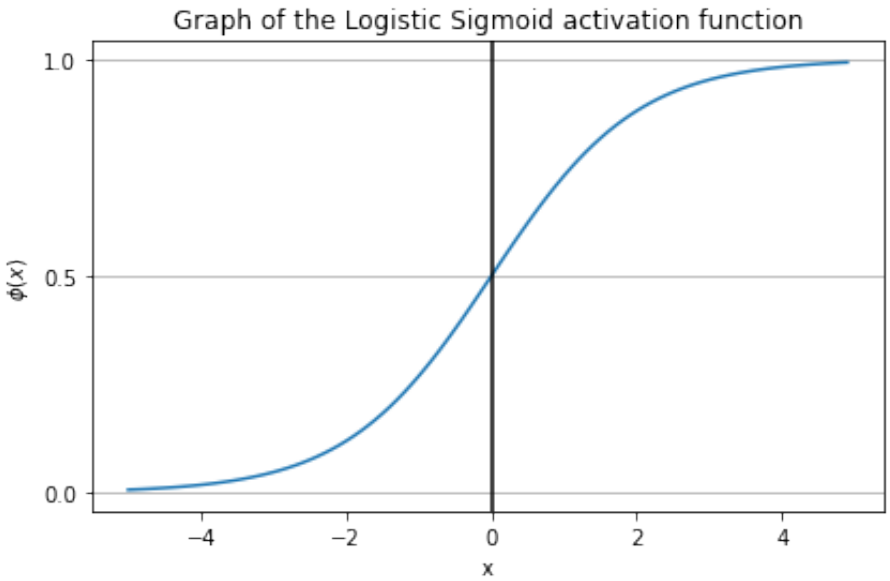


Figure 9: An illustration of the Logistic Sigmoid function's effect on the relationship between the independent and dependent variable.

This function normalises the output values between [0,1] as illustrated in Fig. 9. However, the output is saturated for inputs that are quite close to 1 and those that are close to 0, leading to what is known as the vanishing gradient problem. This term describes a scenario where the gradient of the target functions with respects to a parameter is approximately zero, meaning there are no updates in the parameters during the network training as stochastic gradient descent is used (see section 3.3.1). Another disadvantage to the Logistic Sigmoid activation function is that if the output is not zero-centric, the training will lead to poor convergence.

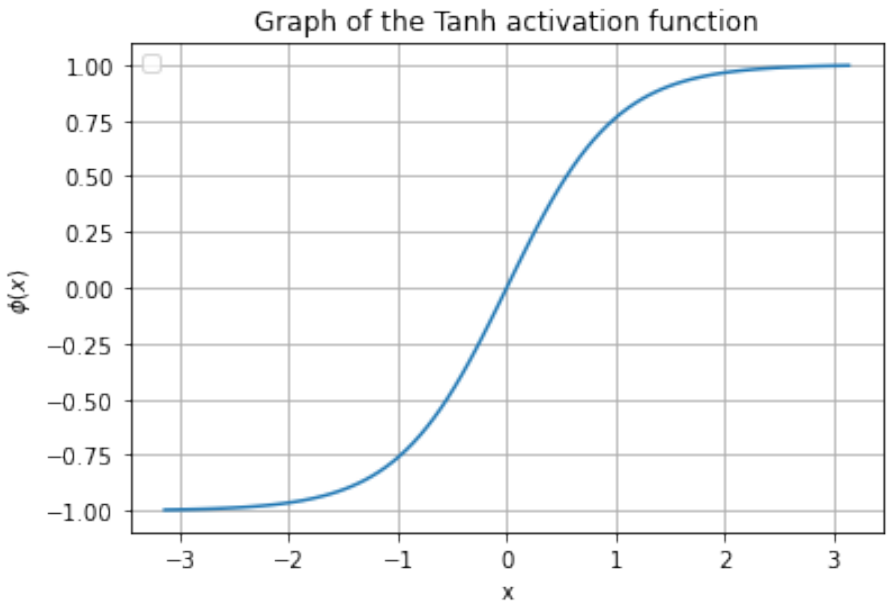


Figure 10: An illustration of the Tanh function's effect on the relationship between the independent and dependent variable.

The Tanh function is similar to the Logistic Sigmoid function as it normalises the function (despite it being between [-1,1]) and is also zero-centric as seen in Fig. 10. This function is given as:

$$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{17}$$

The disadvantages of using the Logistic Sigmoid activation functions (vanishing gradient and computational complexity) also exists with the Tanh function.

3.2.3 The Rectified Linear Unit Activation Functions

The ReLU function (Nair and Hinton, 2010) is a non-linear or piecewsie linear function that affects only the positive input and is the most popular activation function in neural networks.

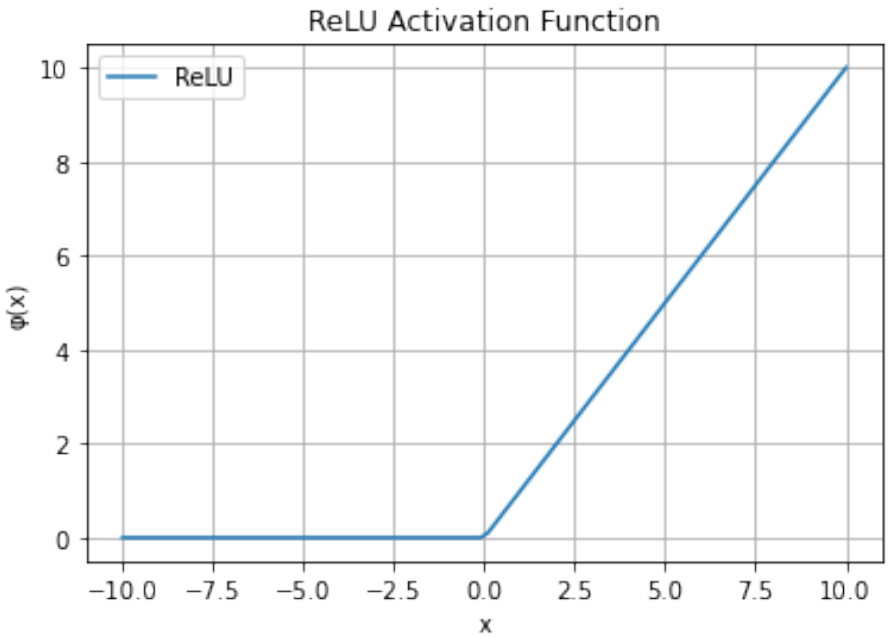


Figure 11: An illustration of the ReLU function's effect on the relationship between the independent and dependent variable.

It is a simple function but yields more accurate results than its predecessors (Logistic Sigmoid and Tanh functions) and is expressed in Eqn. 18:

$$R(x) = max(0, x) \tag{18}$$

Despite what Fig. 11 suggests, the ReLU function is non-linear and is needed to be non-linear as to learn complex relationships from the training data. It only acts as a linear function for positive values and as a non-linear function for the negative values. This function is used as a replacement to the Sigmoid and Tanh functions as it prevents the early layers from being highly saturated at the extremes, averting the prospects of the "vanishing gradient". This function is also more sensitive to smaller changes in the input values.

The advantages of this function are simpler computation (as the derivative stays constant for positive inuts, reducing the time needed for the model to train and minimize errors), representational sparsity (is it representative of outputting a zero value) and linearity (mak-

ing the model easier to optimize and allowing for a smoother flow).

The disadvantages of this function are exploding gradients (when the accumulated gradient causes large differences in the weight updates, causing instability when trying to converge to the global minima which also affects learning) and the "Dying ReLU", the idea that there are "dead neurons" when the neurons are stuck on the negative part of the function. Despite this being quite detrimental to the network, this only happens when the learning rate is too high (for this paper, the learning rate is 1×10^{-8} and so this function is appropriate) or if the negative bias is quite high (which it isn't for the data set used).

3.2.4 Summary of Activation Functions

Table 2: Summary of the Activation Functions (AFs)

AF	Diminishing Gradients?	Limited Non-linearity?	Optimization Difficulty?	Adaptable?	Computationally Efficient?
Linear	No	Yes	Yes	No	No
Sigmoid	Yes	No	Yes	No	Yes
Tanh	Yes	No	Partial	No	Yes
ReLU	Partial	Yes	Partial	No	No

3.3 Optimization Methods

In this paper, only first-order methods are considered as these methods are mainly based on gradient descent. In this section, three representative kernel method algorithms (see Section 3.1.3) are considered, Gradient Descent, Stochastic Gradient Descent and the Adam optimization approach.

3.3.1 Gradient Descent

Gradient Descent is the most common optimization method, and works by updating variables iteratively directly in the opposite direction of the gradients of the target function. These updates gradually converge to the best (optimal) value of the target function and the learning rate λ defines how often the update occurs in each iteration, influencing the

number of iterations needed to reach the optimal value (Ruder, 2016).

Steepest descent algorithm is a widely known algorithm that selects a direction in each iteration to minimise the value of the objective function as quickly as possible. Steepest descent and gradient descent are not the same thing because negative descent in the gradient descent method does not necessarily descend the quickest. Steepest descent is the family tree name that encompases descent techniques in the Euclidean norm such as gradient descent.

This gradient descent method is easy to implement and the solution is the global optimal when the target function is convex. It is noteworthy that the system often converges at a slow rate if the target variable is close to the optimal solution, and as such, more iteration has to be performed.

3.3.2 Stochastic Gradient Descent

This method was proposed as a result of the gradient descent approach of not being able to do online updates despite having high computational complexity in each iteration for large data sets (Robbins and Monro, 1951). This method uses one sample randomly to do an update on the gradient after every iteration, rather than directly solving for the exact value of the gradient overall. This is an unbiased estimate of the real gradient and the cost of this algorithm is is independent of the sample numbers and achieves sub-linear convergence speed (Johnson and Zhang, 2013). Stochastic gradient descent reduces the time for update when dealing with a large data set and removes some computational redundancies which are benefits as it means the calculation speeds are significantly quicker. In a highly convex problem, this method can achieve the ideal convergence speed (Blair, 1985). However, it cannot be used for online learning in highly convex problems, otherwise online learning is not an issue (Robbins and Monro, 1951).

The loss function for this method can be written as:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left(y^i - \xi_\theta(s^i)\right)^2 = \frac{1}{N}\sum_{i=1}^{N} cost(\theta, (s^i, y^i)). \tag{19}$$

However, if a random sample is used in stochastic gradient descent, the loss function changes to be:

$$L^*(\theta) = cost(\theta, (s^i, y^i)) = \frac{1}{2}\left(y^i - \xi_\theta(s^i)\right)^2, \tag{20}$$

and the gradient update in stochastic gradient descent uses the sample this random sample i rather than all the samples in every iteration, meaning:

$$\theta' = \theta + \lambda\left(y^i - \xi_\theta(s^i)\right)s^i. \tag{21}$$

However, one problem stochastic gradient descent has is that the direction of the gradient constantly changes (almost oscillates) due to additional noise added by the random sampling in the solution space. This unlike gradient descent as the gradient descent gradient is always in the negative direction and is biased towards the optimal value whereas stochastic gradient descent's variance in gradients is large and unbiased.

3.3.3 Adaptive Moment Estimation

The adaptive moment estimation algorithm (ADAM) is a combination of stochastic gradient descent and the adaptive learning rate method (Duchi et al., 2011). Therefore in this section, an analysis of the adaptive learning rate and the Adam optimizer is done.

The learning rate greatly effects how well the stochastic gradient descent method works, as such, this is a problem because this method uses a manually regulated learning rate and setting an appropriate value is difficult. To combat this, some adaptive methods were introduced which are free of manually regulating a learning rate, were fast to converge and often performed well. These were widely used in deep neural networks to optimize models and the simplest one is the adaptive gradient algorithm method (AdaGrad) (Duchi

et al., 2011). This method changes the learning rate based on the historical gradient in the iterations preceding the current iteration. The formula for this is as follows:

$$\begin{cases} g_t = \frac{\partial L(\theta_t)}{\partial \theta}, \\ V_t = \sqrt{\sum_{i=1}^{t} (g_i)^2 + \epsilon}, \\ \theta_{t+1} = theta_t - \lambda \frac{g_t}{V_t}, \end{cases} \tag{22}$$

where g_t represents the gradient of θ at instance t, V_t is the sum of the historical gradient of θ at instant t and θ_t is the value of θ at instance t.

An advantage of using AdaGrad is that it removes the difficulty of setting the learning rate as the learning rate is solved by summing the histroical gradients, removing the difficulty of setting a representative λ value λ (Ji et al., 2013).

Despite AdaGrad adjusting the learning rate, there are still two main problems. Firstly, the algorithm still requires supervision in setting the global learning rate λ. Secondly, as the time taken for training increases, the summed gradient will become increasingly larger, causing the learning rate to tend to zero, resulting in an unsuccessful parameter update.

To solve this problem, AdaGrad was improved to AdaDelta and RMSProp by summing all the historical gradients but only using the gradients within a certain number of instances from the current instance (Zeiler, 2012), t. It then uses the exponential moving average to solve for the second-order cumulative momentum:

$$V_t = \sqrt{\beta V_{t-1} + 1\left(1-\beta\right)\left(g_t\right)^2}, \tag{23}$$

where β is the exponential decay factor.

Adam is an advanced stochastic gradient descent method (Kingma and Ba, 2014) that use momentum methods an adaptive learning rate (such as that used in AdaGrad) for each parameter and momentum methods. Not only does this method store the exponential

decaying average of V_t like AdaDelta and RMSProp, Adam also stores the exponential deacying average of the previous gradients m_t and is similar to the momentum approach:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\,(g_t)^2 \tag{24}$$

$$V_t = \sqrt{\beta_2 V_{t-1} + 1\,(1 - \beta_2)\,(g_t)^2} \tag{25}$$

where β_1 and β_2 represents the exponential rate of decay, meaning that the final formula for θ is,

$$\theta_{t+1} = m_t - \lambda \frac{\sqrt{1 - \beta_2}}{1 - \beta_1} \cdot \frac{m_t}{V_t + \epsilon}. \tag{26}$$

The values suggested for β_1, β_2 and ϵ are 0.9, 0.999, 1×10^{-8}, respectively. Adam works well with empirical data and tends to perform more favourably compared to other adaptive learning rate algorithms.

3.3.4 Summary of Optimization Methods

Table 3: Summary of the Optimization Methods

Method	**Properties**	**Advantages**	**Disadvantages**
Gradient Descent	Solves for the best value along the gradient descent vector and converges at a linear rate.	Solution is the global optimal when the target function is convex.	Calculation cost is high because the gradients of the total samples are updated after each instance.
Stochastic Gradient Descent	Update parameters are found using small batches of samples that were randomly selected. This approach converges at a sublinear rate.	Calculation time for each iteration is independent of the total number of training samples, meaning it saves on computational complexity.	Difficult to select an appropriate learning rate as using the same learning rate for every parameter is inappropriate. This method can cause the solution to be trapped between two values in some cases.
AdaGrad	The learning rate is constantly adjusted to be compatible with the sum of the squares of the accumulated historical gradients.	In the beginning of training, the learning rate is higher and the learning speed is quicker as the cumulative gradient is smaller.	The cumulative gradient becomes increasingly larger as you train more, causing the learning rate to tend to zero, becoming meaningless. The learning rate also needs supervision.
Adam	Combines adaptive methods with the momentum method to add a bias correction to remove the problem of the learning rate tending to zero.	The process is quite stable and is suitable for most non-convex problems that have large data sets and a high dimensional space.	This method does not always converge, depending on the case.

To summarise, the ADAM approach takes into account curl divergence and gradients (as it uses the momentum method) which is directly useful to the flow field in question which other approaches do not consider. This approach is also computationally less expensive than the AdaGrad method (despite being similar) and has shown to yield accurate results (see section 6.1.8). As such, the ADAM method is the most suitable optimization function.

4. Experimental Procedure

In this section, the steps taken to gather the low-resolution data is explained. It is important to discuss the origin of the experimental data as it serves as an insight into which type of multi-output regression model is most suitable for this data set.

4.1 Experimental Procedure and Procuring the Data Used

The experimental data was gathered by Symon, Sipp and Mckeon (Symon et al., 2019).

A NACA 0018 airfoil at a Reynolds number of $Re = 10250$ and at an angle of attack of $\alpha = 0°$ is measured using a Paritcle Image Velocimetry set-up (PIV). The airfoil was placed vertically, such that it's span is parallel to the test section height, of which is 1.6m in length, 0.46m in width and 0.5m in height at a free-surface water facility. The NACA0018 span measured to be 0.48cm with an aspect ratio of $AR = 4.8$. The laser sheet used was a YLF dual cavity solid-state laser and its center is placed at a vertical height of 2.2m. The set-up used in the experiment used two Phantom Miro 320 cameras, both with resolutions of 1920 $\times$ 1200 pixels and are calibrated at $8.2pxmm^{-1}$, allowing for there to be 50 vectors per chord length as spatial resolution. The seeding particles used were hollow glass spheres with a specific gravity of 1.1 and an average diameter of $11.7\mu m$ and the seeding density was roughly 0.1 particles per square pixel. To ensure the most accurate results could be attained, Symon, Sipp and Mckeon (Symon et al., 2019) calibrated the imagine intensity using white-image subtraction and background-image subtraction to reduce the impact of surface reflections and completed this before each trial.

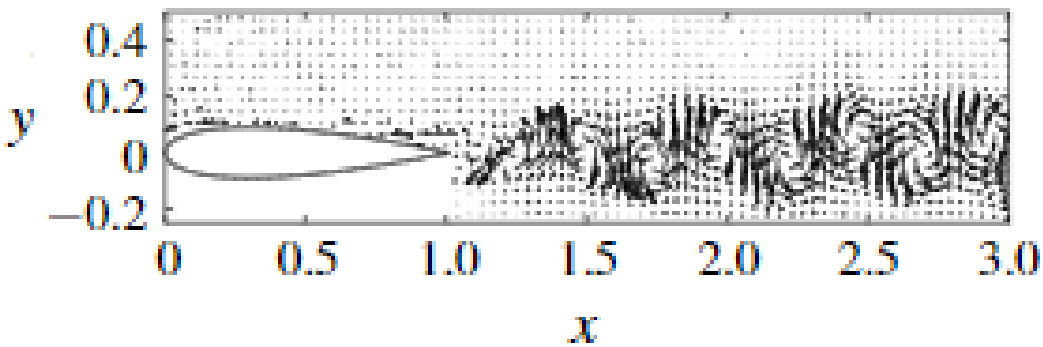


Figure 12: Sample vector plots of the instantaneous velocity of fluid flow for the airfoil placed at an angle of attack of $\alpha = 0\,\text{deg}$ (Symon et al., 2019).

They used LaVision's software, DaVis to compute the velocity vectors by way of fast Fourier transformation for each sequential image. After refining the data, missing vectors were interpolated using non-zero neighbourhood vectors and a median filter was used to detect outliers. A plot of the post-processed data can be seen in Fig. 12 which demonstrates that the velocity vectors below the NACA0018 airfoil cannot be obtained as the laser sheet is blocking this region. However, since the airfoil is symmetric and is placed at an angle of attack of $\alpha = 0\,\text{deg}$ the mean flow can be reflected about the centre line to find the mean flow below the airfoil and in the shadow region.

4.2 What the Outcome of the Data means for our Approach

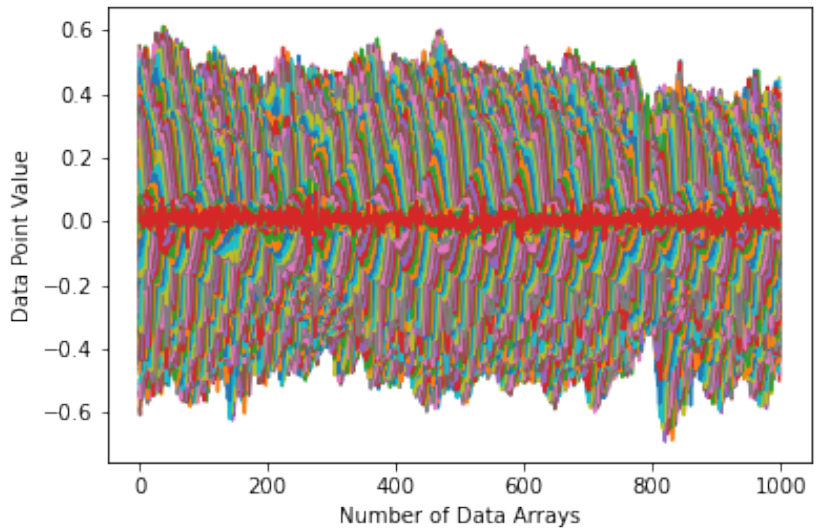


Figure 13: A plot of the data set used as input for the neural network.

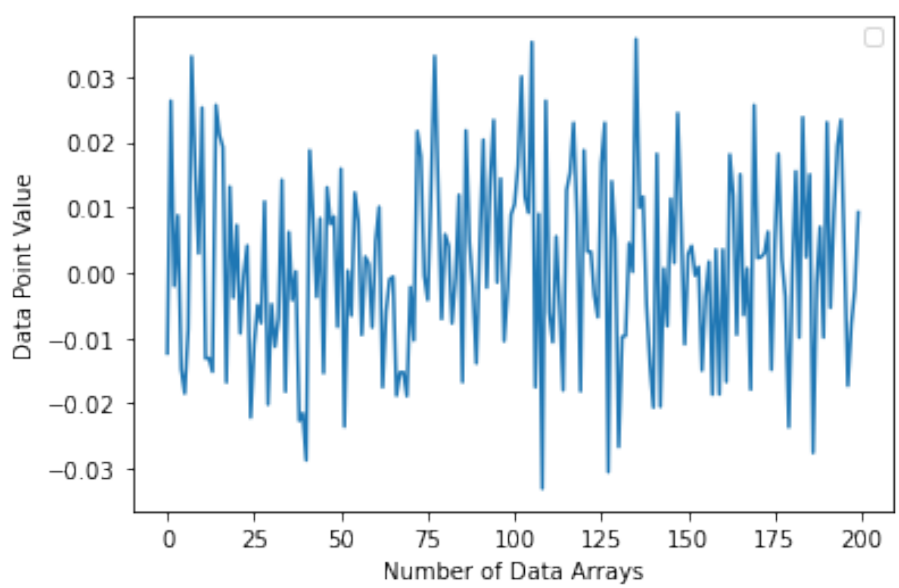


Figure 14: A plot of 200 average test data samples for the neural network.

Fig. 13 shows a plot of the velocity vectors for the first 1000 instances. It is clear to see that the data values follow a cyclical pattern, where the vortexes causing the concentration of velocity values in the wake of the NACA0018 airfoil lie at the peaks and troughs. The raw experimental data has been averaged and plotted using python on Jupyter Notebook (see section 4.2 and 4.1) and assimilates the data shown in Fig. 14, with multiple peaks and troughs for all $204,800$ input values. Symon et al. describes mirroring the data values about the x-axis and using proper orthogonal decomposition to refine the data, resulting in a wave-like pattern with many peaks and troughs at different y-values. Owing to the presence of multiple patterns, it is appropriate to take a pattern-recognition approach which looks at the relationship between each target feature. This means an algorithm adaption method in the form of a multi-output support vector regression is a good approach to implement. As the plots of the data show that the pattern is not geometrically complicated (as seen in Fig. 13 and Fig. 14), a deep neural network consisting of three hidden layers and an output layer could be used to produce a computationally cheap yet significant prediction.

Furthermore, since the data has been symmetrical about the mean reference point, a ReLU activation function (see section 3.2.3) could be used as it is a non-linear function that can learn complicated functions in the positive plane. It is also an appropriate activation function as Fig. 13 shows that the data points are essentially linear and only curve a the peaks and troughs. This means that this activation function is ideal as vanishing gradients (see section 3.2.3) are completely avoided as the gradients are proportional to the node activations and flow well on the active neuron path (Glorot et al., 2011).

5. Methodology

In this section, the objective is to describe the equations for the test case at an angle of attack of $\alpha = 0\,\text{deg}$ and as a result, the mathematics behind the methodology and the reasoning behind this approach to reconstructing aerodynamic flow.

5.1 Framework for the Deep Neural Network

For the model, the TensorFlow deep sequential neural network (Abadi et al., 2016) was the chosen framework for this method. In this subsection, the multi-layer feed-forward network used to reconstruct the flow field is introduced and its training is described.

5.1.1 Framework and Activation Functions

The multi-layer feed-forward network framework is illustrated in Fig. 15 and is formed by three types of layers. There is one input layer, two hidden layers and an output layer, where each layer has one or more nodes. Also, a $feed - forward$ network implies that the information flows from the input to the output and as stated by Cybenko (Cybenko, 1988), this type of feed-forward network can approximate any function because of its three hidden layers.

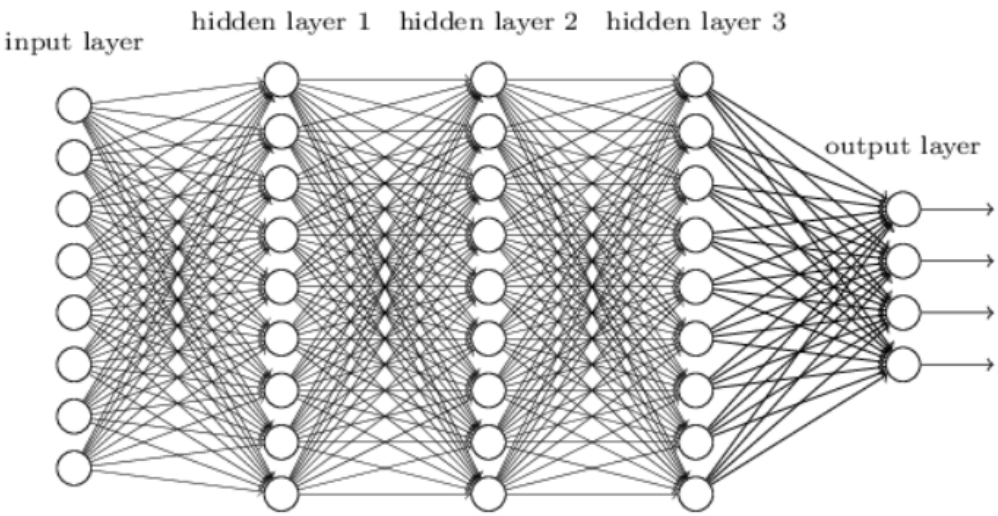


Figure 15: The schematic for the deep neural network used in this method (Goodfellow et al., 2017).

Using Eqn. 3, the idea of weights and biases to define neurons and the framework in Fig. 15 to represent the mapping, we have

$$\begin{cases} X_n^i = X_n, & n = 1, ..., N^i, \\ X_n^{h1} = \vartheta^{h1}\left(\sum_{m=1}^{N^i} W_{n,m}^{h1} Y_m^i + B_n^{h1}\right), & n = 1, ..., N^{h1}, \\ X_n^{h2} = \vartheta^{h2}\left(\sum_{m=1}^{N^{h1}} W_{n,m}^{h2} Y_m^{h1} + B_n^{h2}\right), & n = 1, ..., N^{h2}, \\ X_n^o = \vartheta^{ho}\left(\sum_{m=1}^{N^{h2}} W_{n,m}^o Y_m^{h2} + B_n^o\right), & n = 1, ..., N^o, \\ \hat{X}_n = X_n^o, & n = 1, ..., N^o, \end{cases} \tag{27}$$

where $N^i = 1600$, $N^{h1} = N^{h2} = 3200$, $N_o = 1024$ and are the numbers of dense neurons in each layer. In Fig. 15, W are the weights and B are the biases, where weights are the parameters within the neural network. Weights emphasise how much transformation the test data undergoes within the network's hidden layers and bias is applied when building the model to validate the model when compared to an approved data set. Also, $\vartheta^{h1}(a), \vartheta^{h2}(a), \vartheta^o(a)$ are the activation functions and are necessary for the nonlinear mapping of an artificial neural network.

Despite there being many activation functions proposed in literature (see section 3.2), the activation function used in this method is called the ReLU function (rectified linear unit), for which it's reasoning is explained thoroughly in section 3.2.3.

5.1.2 Sequential Modelling

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1600)              1640000

 dense_1 (Dense)             (None, 3200)              5123200

 dense_2 (Dense)             (None, 3200)              10243200

 dense_3 (Dense)             (None, 1024)              3277824

=================================================================
Total params: 20,284,224
Trainable params: 20,284,224
Non-trainable params: 0
_________________________________________________________________
```

Figure 16: A summary of the sequential deep neural network model used in this project.

A summary of the model used in this paper is shown in Fig. 16. This model denotes the four hidden dense layers in Eqn. 27 as well as the mapping in Eqn. 3. Here, the type of model used is the tensorflow sequential model. This model uses 100 epochs (the number of times the deep learning model is made to learn the data set's pattern for training) and a batch size of 100 (the training data's sample size each time the model randomly selects samples to train with). The sequential model is a model that uses data sequences such as audio clips (or in this case time-series data) and uses the sequence of the previous sample data to learn the sequence and predict the next set of sequences. Therefore, this model is appropriate for the type of experimental data procured. This model can be broken down as follows:

- The model contains three dense (classification) layers (a layer where all the neurons are connected to the neurons of the previous layer, in this case the input layer) and a linar output layer. For example, for the first layer, there are 1600 dense neurons for each element of the input matrix, giving 1,640,000 parameters. The parameters

are broken down as $1600 * 1024$ (inputs $\times$ weights) + 1024 (biases) = $1,640,000$ parameters.

- The same process is completed for the second dense layer, where there are 3200×1600 weights and 3200 biases, giving $5,123,200$ parameters.

- This process is also carried out for the third dense layer as there are 3200×3200 and 3200 biases, giving $10,243,200$ parameters.

- Likewise, for the final dense layer, there are 1024×3200 weights and 1024 biases, giving $3,277,824$ parameters.

- The last layer is linear in Eqn. 27. The other layers have transformations and mappings that change the data. The importance of the last, linear layer is as follows; when a given node is defined by the value of each localized weight, it connects this associated node in the last hidden layer to the nodes of the output layer, allowing for a simpler approach which yields a more accurate result.

It is noteworthy that the parameters and trainable parameters are equal in number here, meaning that all the nodes in these layers can be trained, this means that there are 20,284,224 nodes that will be trained to learn the pattern. A number this high is a good indication as to how successful the model will be, as a higher number indicates that there are more nodes which have been trained to learn the pattern.

5.1.3 Optimization

Algorithm 1: ADAM Optimizer where $\mathbf{Q}$ is the accumulation of weights and biases in the network

Input: $F = \beta_1, \beta_2, \epsilon, \lambda, M^c$ ▷ List of Parameters

Output: $\mathbf{Q}^{k-1}$ ▷ Returned Function

function `ADAM`(F)**:**

  $\mathbf{M}^0 \longleftarrow 0, \mathbf{R}^0 \longleftarrow 0;$

  $\mathbf{E} \longleftarrow 0;$

  $k \longleftarrow 1;$

  while *!StopCriterion(k,*$\mathbf{E}$*)* **do**

    $\mathbf{D}^k \leftarrow \nabla \mathrm{C}^k(\mathbf{Q}^{k-1})$

    $M^k \leftarrow \beta_1 \mathbf{M}^{k-1} + (1-\beta_1)\mathbf{D}^k$

    $\mathbf{R}^k \leftarrow \beta_2 \mathbf{R}^{k-1} + (1-\beta_2)(\mathbf{D}^{\mathbf{k}})^2$

    $\hat{M}^k \leftarrow \mathrm{M}^k/(1-(\beta_1)^k)$

    $\hat{R} \leftarrow \mathbf{R}^k/(1-(\beta_2)^k)$ $\mathrm{Q}^k \leftarrow \mathrm{Q}^{k-1} - \lambda \frac{\hat{M}^k}{\sqrt{\hat{R}^k}+\epsilon}$

    $\mathrm{k} \leftarrow k+1$

  end

  return Q^{k-1}

end function

Input: $G = k, M^c, \mathbf{E}$ ▷ List of Parameters

function `StopCriterion`(G)**:**

  $\mathrm{E}_k \leftarrow ErrorOnValidationSet$

  if $k < M^c + 1$ *or* $E_k \leq E_{k-M^c}$ **then** **return** False.

  else **return** True.

  end function

  function `ErrorOnValidationSet`(G)**:**

  return $\frac{1}{M^{vs}N^o}\sum_{n=1}^{M^{vs}}\sum_{m=1}^{N^o}\left(\tilde{\mathbf{Z}}_m^n - \mathbf{Z}_m^n\right)$

  end function

The ADAM optimizer is used and given in Algorithm 1. It is used to compute the exponential moving averages of the gradient of each instance (denoted as $\mathbf{D_k}$) and its square, where $\mathbf{D_k}$ represents the first and second raw moment estimates of the gradients. The parameters in the Adam optimizer algorithm are all fixed in this paper, where $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$, $\lambda = 0.0001$, where β_1, β_2 are the moving average's exponential decay rate, ϵ is the parameter applied to prevent any division by zero and λ is the learning rate (how often the model updates and learns). The optimizer works by using additional snapshots other than the training set, which is referenced as the validation set and it determines when to stop training. Here, in algorithm 1, the nth snapshot and its respective predicted value in the validation set of M^{vs} snapshots. These values are denoted as $\mathbf{Z^n}$ and $\tilde{\mathbf{Z}}^n$ respectively. The idea is that after each training update k, the squared error E_k is computed on the validation set between the predicted and input values. When $E_k > E_{k-M^c}$, it suggests that the data has begun over-fitting and is developing, this is where the optimizer stops the training. It is important to note, that in all trials, M^c is fixed to 500.

5.2 Summary of the Network

The method can be summarised as follows:

1. Identify that the deep learning network is supervised as it aims to learn the relationship between the input and output columns of the experimental data and that the objective is to be able to accurately extrapolate the data such that the aerodynamic flow field is reconstructed.

2. Use an artificial neural network consisting of a sequential model with three dense layers that uses the ReLU activation function to train the model using 100 epochs with a batch size of 100.

3. Once the model is trained, use the ADAM optimization function to optimize the data output in the output layer.

4. Print the output and save it to a *.csv* file.

5.3 Python and Basic Interfaces needed to test data into the Model

This project uses Python and TensorFlow to code the deep learning algorithm. This subsection will explain the basic python interfaces needed to test data into the framework explained in Section 5.1.

5.3.1 Dataframes

A dataframe is a data format in *pandas* that allows for different datasets to be mapped out to different variables. For example, one such data frame that shows the mapped output velocities from the data used in this project is in Fig. 17, where the indices in this dataframe are on the left, and the velocities are stored in each column.

0	-0.015388	-0.018061	-0.017225	-0.017432	-0.021551	-0.020875
1	-0.020503	-0.022121	-0.022937	-0.025660	-0.026829	-0.020831
2	-0.025807	-0.029693	-0.037537	-0.036433	-0.027930	-0.017400
3	-0.020061	-0.024704	-0.028155	-0.029574	-0.026447	-0.018758
4	-0.024282	-0.031049	-0.027161	-0.016145	-0.018504	-0.021478
..	...	...	...	...	...	...
994	0.049365	0.052633	0.044638	0.034369	0.014549	-0.009631
995	0.022979	0.033276	0.040273	0.030174	0.006497	-0.017069
996	0.037331	0.036149	0.020280	-0.007977	-0.033055	-0.042848
997	0.051583	0.037990	0.009215	-0.023998	-0.046576	-0.044654
998	0.043707	0.035581	0.018074	0.001304	-0.015870	-0.017842

	-0.017741	-0.017191	-0.013313	-0.004646	...	0.036018	0.041966	\
0	-0.017581	-0.012207	0.000321	0.012034	...	0.070037	0.059733	
1	-0.014518	-0.012529	-0.011935	-0.007545	...	0.025258	0.027522	
2	-0.009963	-0.002940	0.004604	0.012945	...	0.051806	0.040222	
3	-0.010899	-0.004631	0.001225	0.011321	...	0.083028	0.079509	
4	-0.009534	0.005584	0.019882	0.028623	...	0.050094	0.026363	
..	...	...	...	...	...	...	...	
994	-0.032471	-0.055545	-0.086681	-0.112242	...	-0.032628	-0.038293	
995	-0.030774	-0.058776	-0.093714	-0.097770	...	-0.041671	-0.037267	
996	-0.053048	-0.094480	-0.123530	-0.123611	...	-0.083608	-0.068848	
997	-0.061457	-0.111810	-0.130886	-0.115299	...	-0.060494	-0.068538	
998	-0.063372	-0.110641	-0.124732	-0.116733	...	-0.056975	-0.043093	

	0.048425	0.037417	0.019049	0.016635	0.025602	0.002764	\
0	0.033965	0.025270	0.017659	0.009105	-0.008453	-0.013596	
1	0.033959	0.027956	0.005576	-0.005063	0.012462	0.067012	
2	0.029230	0.022657	0.008865	0.007025	0.035062	0.047504	
3	0.059730	0.043069	0.035317	0.032384	0.049581	0.038645	
4	0.011483	0.008595	-0.010303	-0.017936	-0.029132	-0.012068	
..	...	...	...	...	...	...	
994	-0.031948	-0.013805	0.000723	0.016496	0.001185	0.001184	
995	-0.033689	-0.016524	0.008832	0.013540	0.004936	0.007320	
996	-0.049264	-0.031441	-0.018129	-0.010475	0.016484	0.013652	
997	-0.053458	-0.022457	-0.008226	0.002557	-0.014883	-0.018723	
998	-0.032770	-0.020635	-0.002309	0.011878	0.018848	0.015331	

Figure 17: An same of the input data as a pandas dataframe, showing indices on the first column and velocities in the rest.

5.3.2 Reading XLSXs into Dataframes and the Corresponding Issues

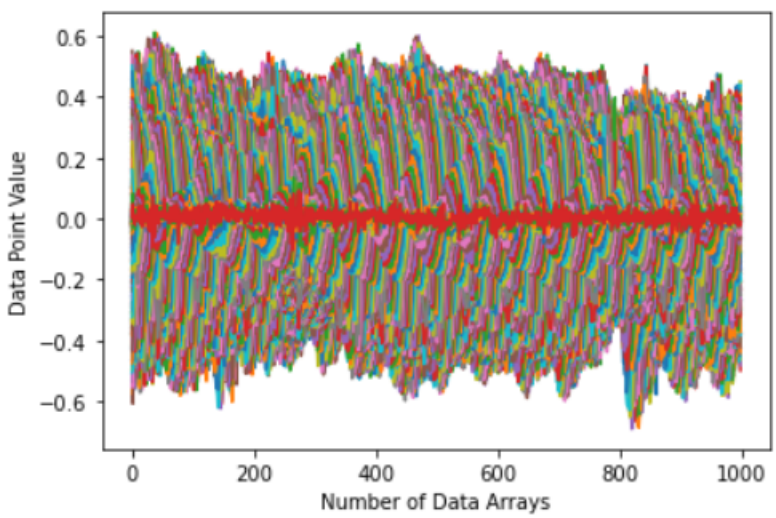
To begin using dataframes, data must first be procured. The experiment outlined in Section 4.1 was provided as an excel file, which can be easily read into a pandas dataframe using function *read_excel* () as shown in Fig. 18.

```
import pandas as pd
dataframeINPUT = pd.read_excel('Test.xlsx', sheet_name = 2)
df = pd.DataFrame(dataframeOUTPUT)

print(df)
```

Figure 18: Illustrating the code used to read data as a pandas dataframe.

```
#DATASET
dataframeINPUT = pd.read_excel('Test.xlsx', sheet_name = 2)
dataframeOUTPUT = pd.read_excel('Test.xlsx', sheet_name = 6)
X, Y = dataframeINPUT.to_numpy(), dataframeOUTPUT.to_numpy()
plt.plot(Y)
plt.xlabel("Number of Data Arrays")
plt.ylabel("Data Point Value")
plt.savefig("Original_Complete_Dataset.png")
plt.show()
xtrain, xtest, ytrain, ytest=train_test_split(X, Y, test_size=1)
print(X.shape)
print(Y.shape)
in_dim = X.shape[1]
out_dim = Y.shape[1]
```




```
(999, 1024)
(999, 1024)
```

Figure 19: Illustrating the code used to solve the tensor problem, and outputting the data.

which reads in an xlsx file corresponding to the sensor measurement outputs into the variable *df*, a *DataFrame* object.

However, an issue arose when doing this. Using Tensorflow means that the test data is required to be in the form of a tensor and not a dataframe, meaning it is incompatible with the tensorflow sequential data.

To solve this problem, the pandas dataframe was converted into a numpy array. Numpy array is another way of transforming dataframes. Although a numpy array is not strictly a tensor array, the only difference between the two is that tensors are supported by accelerator memory components such as GPUs (Song and Mei, 2018). This means that numpy arrays are still be compatible with Tensorflow as long as a CPU is used to process the model. This was completed using the *to_numpy* () function, allowing for the use of the test data from excel as seen in Fig. 19.

6. Results and Discussion

Fig. 20 shows three contour plots, the plot of the predicted values (top), the plot of the input values (middle) and the plot of the true, expected values (bottom). The top plot and the bottom plot are very similar. This shows that this method for flow reconstruction was successful. This conclusion is also supported by Fig. 21 as the average prediction of every row of the data matrix plot is similar in value to the average test value of every row from the true data set.

This section will discuss the most popular numerical evaluation techniques, perform them and discuss the effects of changing different parameters.

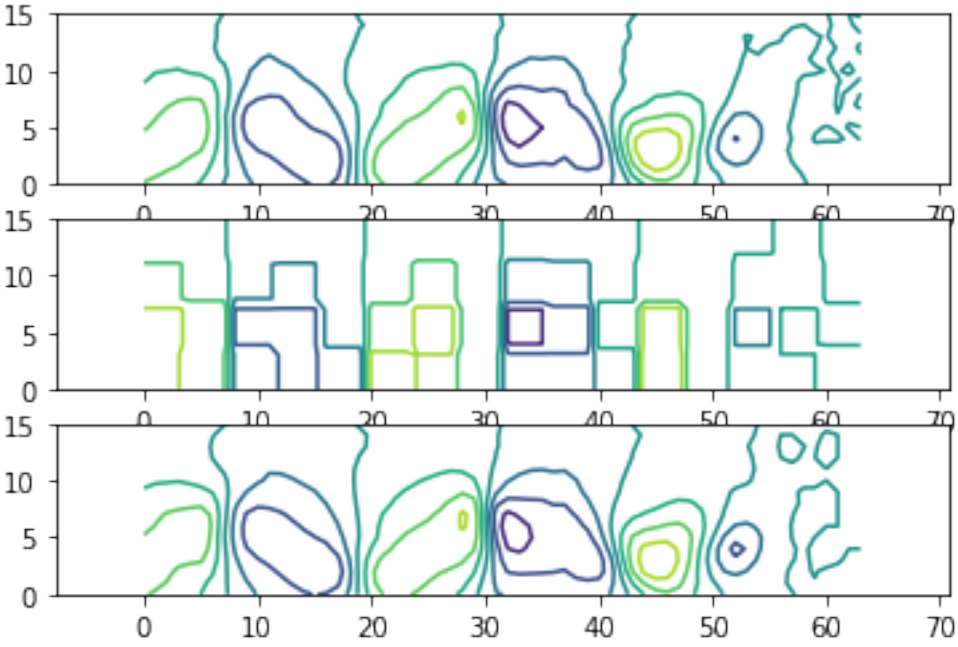


Figure 20: The contour plot used to measure predictive performance, where the top graph shows the predicted data, the middle is the input and the bottom is the validation set that the predicted data is tested against. The predictions and the validation data are similar, indicating good predictive performance.

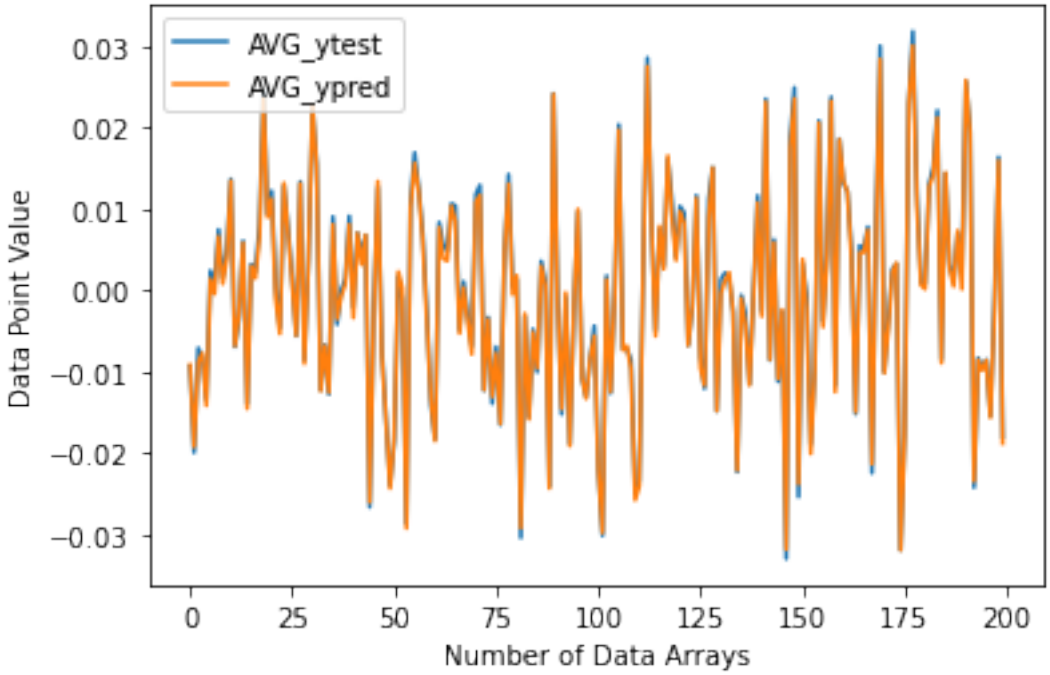


Figure 21: A depiction of the average prediction in every sample compared against every sample of the average validation value.

6.1 Results Evaluation

An evaluation of the model used and how it has attributed to its predictive performance will be discussed. Additonally, an insight into how representable and interpretable the output is of the data is given, and a discussion of the computational costs is made. This will include analysing the effects of a change in epoch number, activation function and optimization type.

6.1.1 Predictive Performance

The Multi-output SVR and kernel methods (or kernel tools, in this case the idea of the ReLU function) are usually designed to yield a good estimation of the mapping, where linearity is not something to be assumed (see sections 3.1.2 and 3.1.3). Using these methods to create a sequential model tends to have the following objectives in mind:

- To speed up computation time.
- To obtain a more sparse representation (such that most of the input and output data are representable).
- To keep a similar error rate as statistical methods (see section 3.1.1.)

However, this method is likened to Liu *etal.*'s method as the main objective is to maximise predictive performance (Chow and Liu, 1968). Therefore, it is to be expected that the predictive performance would be accurate as seen in section 6.1 whilst at the same time, the expected trade-off for this would be higher computational complexity and less interpretable data.

6.1.2 Computational Complexity

Table 4: Comparison between number of epochs used and the time taken to compute.

	Epochs		
	5	100	500
Time Taken (s)	3.34	43.4	216.1
RAM Usage (%)	8	28	63

Although, a high computational complexity is expected using this technique as the method has to solve one very large problem consisting of three smaller problems where the nodes of each problem are interlinked, Tab 4 shows that the computing time is not very long and when compared to the predictive performance, it is a worthwhile trade. Despite this, using an intel i7, 8GB RAM laptop run the code was problematic. While the code ran for epochs of 100 and 500, the RAM usage spiked, slowed the entire system and would occasionally crash when computing 500 epochs.

Tab 4 shows that there is a correlation between more training and higher computational costs which can be attributed to how much RAM is being used to run the code. This is because using a CPU processor on a large set of data is expensive as they excel at running complex algorithms on a small sets of data. As such, an improvement to this model could be to use GPU processing as having more cores mean that they can run simple algorithms to large data sets more efficiently (Buber and Diri, 2018).

6.1.3 Representation and Interpretability

Multi-output algorithm adaptation SVR methods tend to suffer from not being able to provide an accurate description of what links together the output variables and are only interpretable as long as the outputs are not intractable. This can be attributed to the interpretations not being straightforward as the input space is transformed. As such, the improved predictive performance has a trade off with the readability of the data. For this data set, the data produces were tractable, despite this, it is possible that a more complex test data set may mean this method yields uninterpretable results.

6.1.4 Numerical Performance Evaluation Techniques

Here, an introduction to the most popular performance evaluation measures to assess machine-learning models are made. Let $\mathbf{y}^{(l)}$ and $\hat{mathbfy}^{(l)}$ be the actual and predicted vectors for $\mathbf{x}^{(l)}$ for test data size of N_{test}. Also, $\bar{\mathbf{y}}$ and $\bar{\hat{\mathbf{y}}}$ are the average vectors of the actual and predicted outputs. Other than measuring the computing times (Kocev et al., 2009) (Liu et al., 2009) (see section 6.1.2.), these are the most commonly used measures:

- Average mean squared error (aMSE) (Cai F, 2009):

$$aMSE = \frac{\sum_{i=1}^{d} MSE}{d} = \frac{1}{d}\sum_{i=1}^{d}\frac{1}{N_{test}}\sum_{l=1}^{N_{test}}\left(y_i^{(l)} - \hat{y}_i^{(l)}\right)^2$$

- Average relative root mean squared error (aRRMSE) (Tsoumakas et al., 2014):

$$aRRMSE = \frac{\sum_{i=1}^{d} RRMSE}{d} = \frac{1}{d}\sum_{i=1}^{d}\sqrt{\frac{\sum_{l=1}^{N_{test}}\left(y_i^{(l)} - \hat{y}_i^{(l)}\right)^2}{\sum_{l=1}^{N_{test}}\left(y_i^{(l)} - \bar{y}_i\right)^2}}$$

- Mean average error (MAE) (Xu et al., 2013):

$$MAE = \frac{1}{d}\sum_{l=1}^{d}\left|y_i^{(l)}\right|$$

- R^2 - Coefficient of Determination (regression analysis) (Mojjada et al., 2020):

$$R^2 = \frac{\sum_{l=1}^{d}(\hat{y}_i - \bar{y})^2}{\sum_{l=1}^{d}(y_i - \bar{y})^2}$$

Note that the errors are computed as averages over all the separately computed errors for each input. This allows for an assessment of the model performance across multiple outputs, meaning that the use of a normalization vector may be useful to make comparisons easier. It is also worth noting that relative measures such as aRRMSE automatically re-scale the error, meaning no normalization needs to be done.

6.1.5 Effects of changing the training/test splits

A training/test split is how much of the input data is used as training data for the model to learn. For example, a split of 0.2 equates to a 20% of the test data and 80 % of the training data as input, outputting a yield of shape $(200, 1024)$, where the 0.2×1000 dictates the first element of the matrix. A test size of 0.8, yields a matrix of $(800, 1024)$.

Table 5: Effects of Test Size Changes

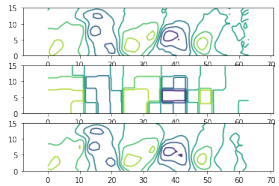
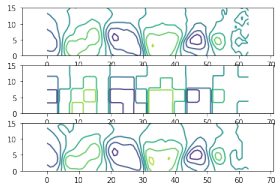
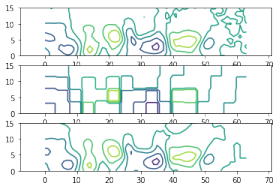
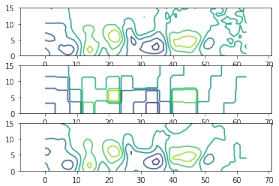
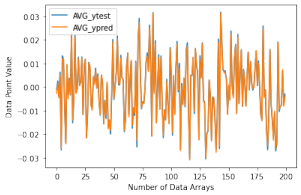
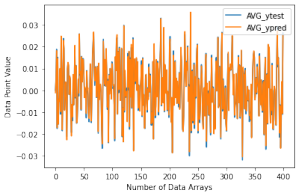
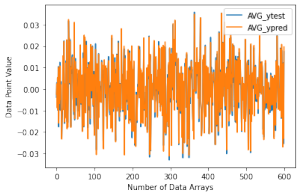
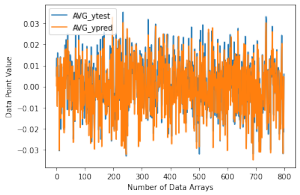
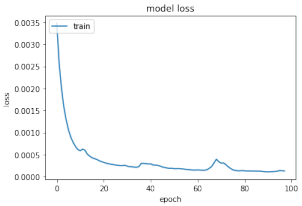
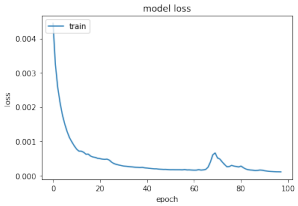
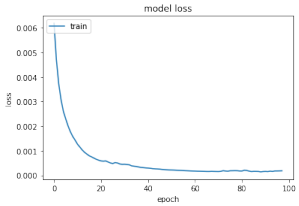
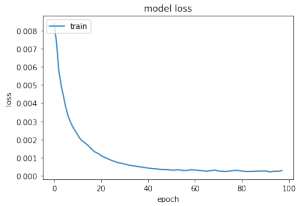
<table>
<tr><th></th><th colspan="4">Test Size</th></tr>
<tr><th></th><th>0.2</th><th>0.4</th><th>0.6</th><th>0.8</th></tr>
<tr><td>aMSE
aRRMSE
MAE
R^2</td><td>0.0150
0.0287
0.162
-1.78×10^{31}</td><td>0.0156
0.0147
0.164
-1.330×10^{31}</td><td>0.0211
0.0105
0.175
-0.981</td><td>0.0354
0.00828
0.180
-0.968</td></tr>
<tr><td>Contour</td>
<td> 
</td>
<td> 
</td>
<td> 
</td>
<td> 
</td></tr>
<tr><td>AVG_Pred vs AVG_Test</td>
<td> 
</td>
<td> 
</td>
<td> 
</td>
<td> 
</td></tr>
<tr><td>Loss</td>
<td> 
</td>
<td> 
</td>
<td> 
</td>
<td> 
</td></tr>
</table>

Table 6.1.5 shows that for test size values below 0.5, the changes to the output are minimal and quite insensitive as the aMSE and MAE values do not change significantly. However, the relative measure of aRRMSE show that the test size change is significant when compared to its previous state, as an increase in test size implies larger test data. This also means that the algorithm can train on more data, improving its predictive performance. The average measures (aMSE and MAE) also show this pattern after a test size of 0.5 as the change in error between 0.2 and 0.4 is less than that of 0.6 and 0.8. It is also notable that the non-relative measures are increasing in error whilst the aRRMSE measure is decreasing. This suggests that compared to the previous state of the model, error is decreasing, despite the error between the new true validation data and the output is increasing. This is also represented by the contour and predicted vs test plots, where the change between the true/test value and the output value slightly increases as the test size increases.

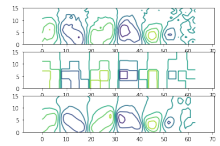
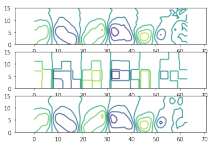
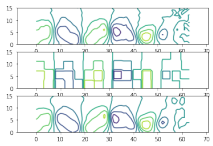
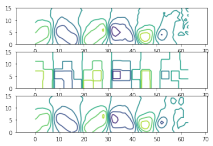
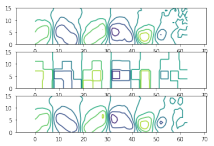
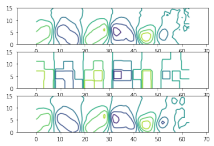
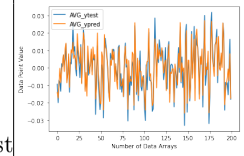
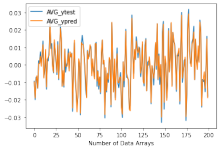
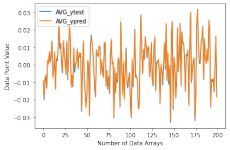
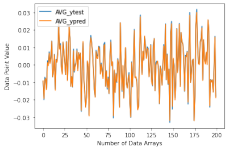
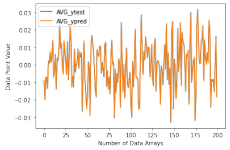
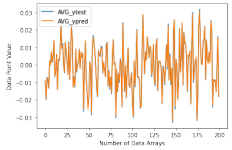
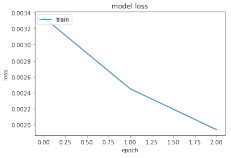
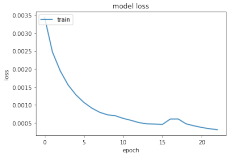
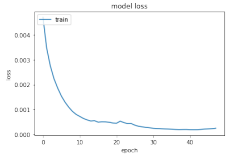
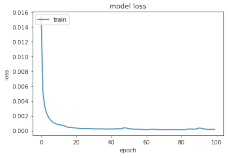
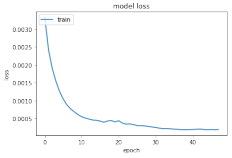
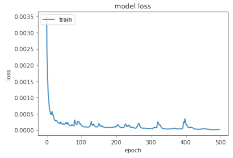
However, these plots are difficult to interpret because of the overfitting that is indicated by the R^2 values. Negative R^2 values imply that there is overfitting, whilst the magnitude is extent of overfitting (Chicco et al., 2021). As such, higher test sizes yielded more interpretable results. Therefore, to improve this model in the future, a test size of 0.8 is ideal as increased training implies better predictive performance and interpretability.

The losses also indicate overfitting for test sizes of 0.2 and 0.4 as the graphs exhibit step-like, reiterative behaviour and it is probable that the data itself is repetitive, meaning shuffling the data is an improvement. However, test sizes 0.6 and 0.8 had curves that were a good fit. This suggests that this issue effects the first 40% of the test data and therefore insignificant to the output when opting for higher test sizes.

It is notable that the computational complexity between test sizes were negligible. In the next section, a discussion on epochs and their effects on performance is made.

6.1.6 Effects of changing the number of epochs on performance

Table 6: Performance Evaluation against Number of Epochs

	Epochs					
	5	25	50	100	200	500
aMSE	0.0694	0.0210	0.0223	0.0152	0.0136	0.0135
aRRMSE	0.0368	0.0302	0.0307	0.0291	0.0290	0.0287
MAE	0.172	0.175	0.172	0.177	0.169	0.168
R^2	-9.75×10^{31}	-2.97×10^{31}	-3.22×10^{30}	-9.5×10^{30}	-1.09×10^{31}	-7.97×10^{30}
Contour						
AVG_Pred vs AVG_Test						
Loss						

In this subsection, the numerical performance evaluation measures are used to evaluate the effectiveness of different epochs to the model.

Table 6.1.6 show that for all normalized (relative) and average measures, the numerical errors decrease as the number of epochs increase. This can be explained by there being more epochs, and therefore more training, improving predictive performance. This is also shown clearly by the contour and predicted vs test plots, where predictions improve with epochs.

The R^2 values indicate that the output regression compared to the true regression is worse than $y = K$, where K is the mean target value for every training data point as they are negative (Chicco et al., 2021). Owing to the magnitude of the R^2, the worse fit can be accredited to over-fitting. These values also have no correlation with each other, meaning the number of epochs do not have an effect on overfitting.

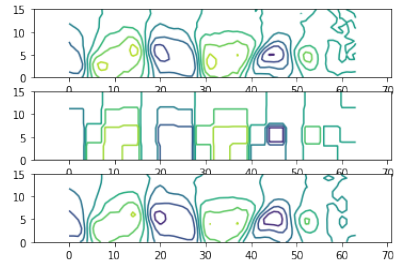
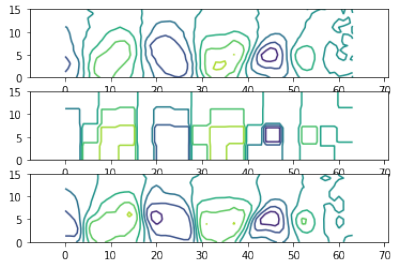
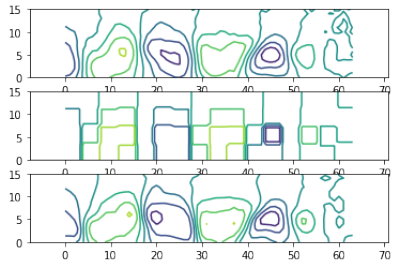
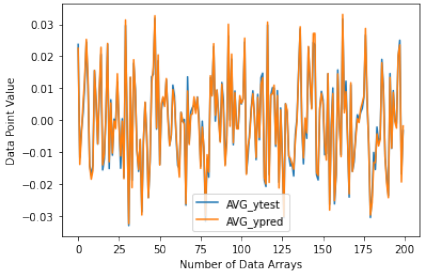
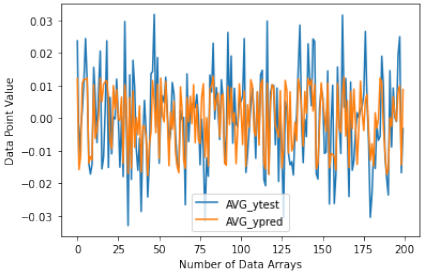
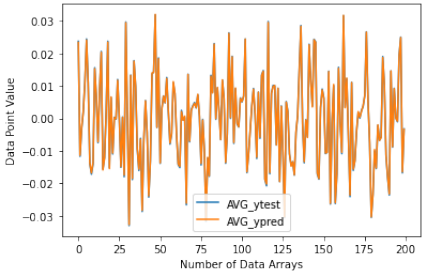
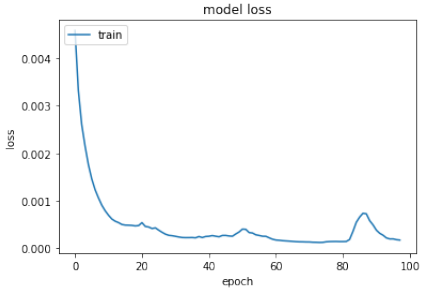
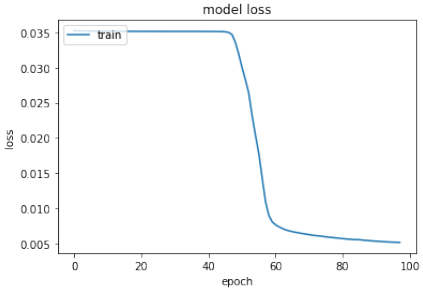
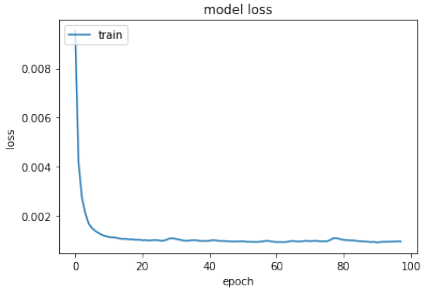
The loss graphs show that for an epoch of 5, the model is underfitted, as the graph has two linear sections rather than a curve (Goodfellow et al., 2017) (Nie et al., 2018), whilst epochs 25 to 200 have a good fit, as the curve is mainly smooth. However, at epoch 500, the curve oscillates before it converges, implying that the model has high overfitting.

Owing to the analysis above, rather than using an epoch of 100 (as the methodology uses), it is best (of the tested epochs) to use an epoch of 200, as it has high predictive performance whilst having little overfitting compared to other epochs. This means that, regardless of the higher predictive performance that comes with increasingly large epochs, using an epoch that induces less overfitting is more desireable, as it is more representative of the true data.

The next section will discuss the effects of different activation functions on the model.

6.1.7 Effects of changing the Activation Function

Table 7: Activation Function Performance Evaluation

	ReLU	Logistic Sigmoid	Tanh
aMSE	0.0152	0.0676	0.0156
aRRMSE	0.00291	0.003	0.00288
MAE	0.177	0.175	0.179
R2	-9.46×10^{30}	-3.19×10^{30}	-3.14×10^{30}
Contour			
AVG_Pred vs AVG_Test			
Loss			

Looking at Table 7, it is clear to see from the figures that the ReLU and Tanh functions yield results with the least error. The loss functions also support this conclusion, where the ReLU and Tanh functions showed a good fit whilst the Sigmoid function suggests that the training data was unrepresentative. This means that the Sigmoid function did not synergise with the training data, further supporting this interpretation of the results (Gupta and Gupta, 2019). Owing to the absence of significant diminishing gradients, the ReLU function would be most suitable as it is the cheaper option (as a result of its limited non-linearity as stated in Table 2). This can be attributed to computations being simpler, meaning that the time taken to train reduces. It could be argued that the Tanh function is better suited as there is no "Dying ReLU", but this is not an issue as the data has been reflected across the neutral axis (see section 4.1).

As such, using the ReLU activation function was the most optimal option compared to other popular options.

6.1.8 Effects of changing the Optimization Method

Table 8: Optimization Performance Evaluation

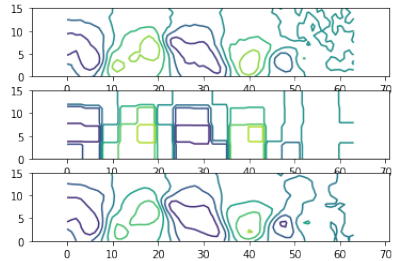
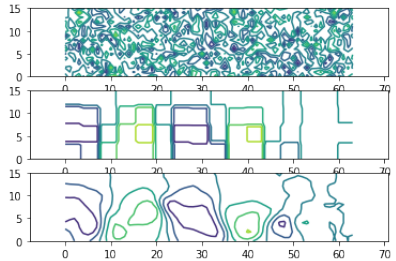
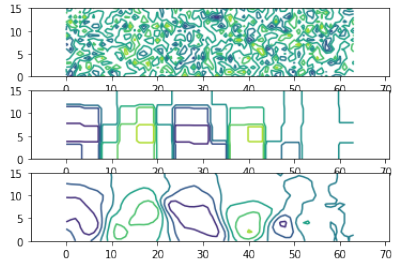
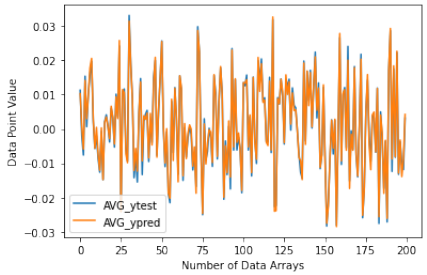
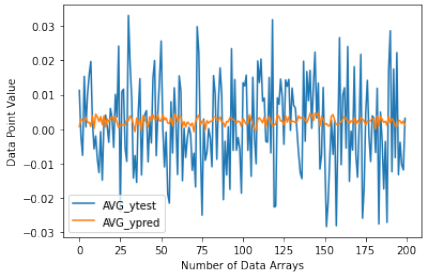
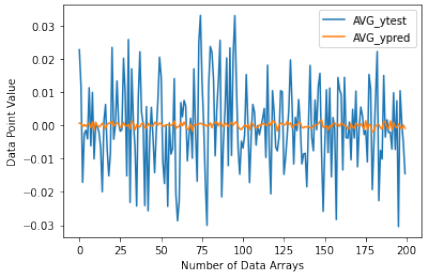
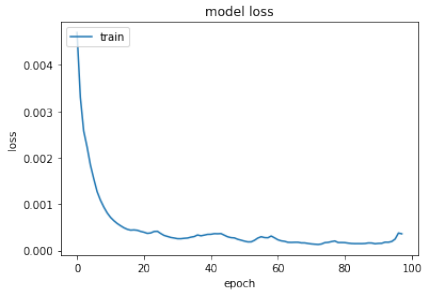
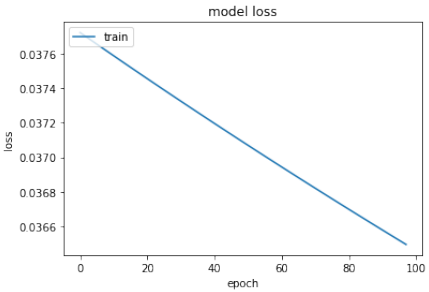
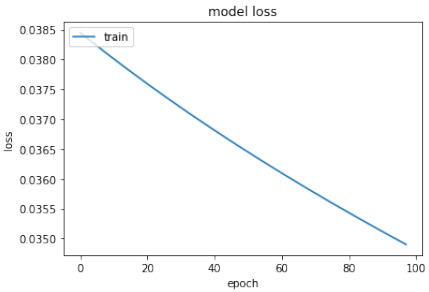
	Optimizers		
	ADAM	AdaGrad	Stochastic Gradient Descent
aMSE	0.0197	0.492	0.425
aRRMSE	0.0315	0.0846	0.0784
MAE	0.175	0.141	0.140
R^2	$-2.20\times10^{+31}$	$-1.32\times10^{+33}$	$0.139\times10^{+32}$
Contour			
AVG_Pred vs AVG_Test			
Loss			

Table 8 demonstrates that the numerical performance evaluation techniques imply that the ADAM optimizer synergises most with the neural network used in this project as the ADAM optimizer function has the lowest error in every measure.

This is further supported by the contour and predictions vs test plots. The contour plots for AdaGrad and SGD show a scrambled prediction, which is unlike their true values. This shows that the predictive performance of the model when using these optimizers is weak. However, the ADAM optimizer plot is very similar to the true value, showing that the ADAM optimizer has the best predictive performance compared to other popular options.

The difference in contour plots can be explained through the loss function graphs. The loss function for AdaGrad and SGD are very similar, and liken very much to a linear plot, indicating that the model was underfit and that it was unable to learn the training data (Chicco et al., 2021). However, the ADAM loss plot shows a curve indicating a good fit, despite illustrating some oscillations which indicate slight overfitting. It is also notable that the ReLU activation function and the ADAM optimizer synergise well as the ReLU activation function continually updates and stabilises the weights that the ADAM optimizer determines (Nair and Hinton, 2010).

For the reasons outlined above, the ADAM optimizer function was the ideal function for this model.

7. Future Work

In this paper, a range of techniques and methods were explored to reconstruct flow fields from Symon $etal.'s$ (Symon et al., 2019) aerodnyamic data set and it is hoped that the attempt to use an artificial neural network can be leveraged for providing some inital areas for future exploration.

There was an attempt to classify the data set using one neural network with three dense layers. Future work could be to attempt using deep neural networks containing convolu-

tional layers and pretrained weights to improve predictive performance as small networks can easily overfit data, whereas larger networks do not tend to have this issue and train more effectively with pretrained weights. Multiple input and output data could also be used to reduce overfitting and computational complexity.

Convolutional neural networks are usually better than a feed-forward network for predictions because CNNs have feature parameter sharing and dimensionality reduction. This means that the number of parameters are reduced, thus computations also decrease.

One such model is shown in Fig. 22:

```
Model: "sequential_38"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_72 (Conv2D)          (None, 996, 1021, 1600)   78400

 conv2d_73 (Conv2D)          (None, 993, 1018, 3200)   81923200

 max_pooling2d_36 (MaxPoolin  (None, 248, 254, 3200)   0
 g2D)

 conv2d_74 (Conv2D)          (None, 245, 251, 3200)    163843200

 conv2d_75 (Conv2D)          (None, 242, 248, 3200)    163843200

 max_pooling2d_37 (MaxPoolin  (None, 60, 62, 3200)     0
 g2D)

 dense_52 (Dense)            (None, 60, 62, 1024)      3277824

 global_max_pooling2d_18 (Gl  (None, 1024)             0
 obalMaxPooling2D)

 dense_53 (Dense)            (None, 1024)              1049600

=================================================================
Total params: 414,015,424
Trainable params: 414,015,424
Non-trainable params: 0
_________________________________________________________________
```

Figure 22: A proposed deep convolutional neural network with three filters, four convolutional layers and two hidden layers for future work.

where multi-layer deep neural networks with convolutional layers help to refine the data such that outliers have no impact. By max-pooling (using more samples with less data that are limited by maximum and minimum values), the problem of over-fitting ceases to exist. This type of network should yield better results as it trains significantly more parameters and neurons, meaning training quality is improved compared to the ANN used.

Finally, future work should look to use the different state-of-the-art multi-output regression approaches outlined in this paper on different aerodynamic datasets. This work should be done on a single node on the Iridis 5 supercomputer (see Appendix A for the specifications), where computational complexity should not pose an issue.

8. Conclusion

In this project, flow field reconstruction and modern multi-output regression techniques are thoroughly surveyed by presenting the details of the popular approaches that have been introduced in literature. Once comparing these techniques, the choice to adopt a sequential SVR framework model was made.

Once this was done, implementing this flow field reconstruction method, based on the use of an artificial neural network began. When given a target problem, a simplified problem is defined using POD to decompose the target problem into velocity values. The data is then split to obtain the training, test and validation samples. The relation between the simplified output values and target inputs is learned using an artificial neural network (ANN) and to obtain the solution of the target feature at a new parameter, the problem and ANN is used to learn the mapping and accurately predict the target feature.

The results showed that the model and the parameters selected proved to be accurate, interpretable and not detrimentally computationally taxing. Changes to the model for the future were made and a suggestion to use convolutional neural network model to improve computational complexity was proposed.

Appendix A.

Iridis 5 related specifications

The iridis 5 supercomputer has two 2.0 GHz Skylake processors (Intel Xeon E5-2670) and 20 cores per proccessor. This means that there are 40 cores per nodes, allowing for iridis to run the deep learning code significantly quicker than an intel i7 8GB RAM laptop.

Appendix B.

Risk Assessment

University of Southampton Health & Safety Risk Assessment

Version: 2.3/2017

Risk Assessment				
Risk Assessment for the activity of	Project of Reconstructing Aerodynamic Flow using Machine Learning		Date	03/11/2021
Unit/Faculty/Directorate	Engineering			
Line Manager/Supervisor	Dr Sean Symon	Signed off		

PART A										
(1) Risk identification			(2) Risk assessment				(3) Risk management			
Hazard	Potential Consequences	Who might be harmed (user; those nearby; those in the vicinity; members of the public)	Inherent			Control measures (use the risk hierarchy)	Residual			Further controls (use the risk hierarchy)
			Likelihood	Impact	Score		Likelihood	Impact	Score	
Not having internet access to HPC.	No access means no machine-learning.	User	4	2	8	**Substitute:** Gaining access to Iridis4 or use computer labs at the university.	2	2	4	Computer labs are closed to students and refused access to Iridis4.
Not yielding an accurate output by using an unideal program for the task.	Can't produce an accurate reconstruction if the program has limitations (e.g. Matlab).	User	3	2	6	**Substitute:** Use more reliable and commonly used programs such as python.	4	1	4	Many python packages to use, opportunity cost in using one package over another in terms of quality and time.

University of Southampton Health & Safety Risk Assessment

Version: 2.3/2017

PART A										
(1) Risk identification			(2) Risk assessment				(3) Risk management			
Hazard	Potential Consequences	Who might be harmed (user; those nearby; those in the vicinity; members of the public)	Inherent			Control measures (use the risk hierarchy)	Residual			Further controls (use the risk hierarchy)
			Likelihood	Impact	Score		Likelihood	Impact	Score	
System damage from overheating by running algorithms.	CPUs in the laptop overheating whilst running the lengthy code.	User	3	1	3	**Eliminate:** Use the Iridis5 supercomputer remotely.	4	1	4	Not having internet access to HPC will halt the entire project.
Eyestrain.	Prolonged use of a computer screen may cause eyes to strain.	User	3	5	15	**Personal Protection:** Scheduling breaks between working periods	1	4	4	Too long a break can mean my Gantt Chart isn't followed.
Corruption in data files when converting Matlab to Python.	Yield an undesired output.	User	3	4	12	**Admin controls:** Convert both Matlab data and Python data into a graph and see if they are identical and to seek advice from my supervisor.	3	2	6	Cannot truly tell if the graphs are identical, as individual coordinates may be slightly different.

University of Southampton Health & Safety Risk Assessment

Version: 2.3/2017

PART B – Action Plan

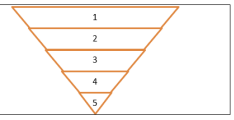
Risk Assessment Action Plan

Part no.	Action to be taken, incl. Cost	By whom	Target date	Review date	Outcome at review date
1	Make the code computationally cheap to run on an average system	User	N/A	N/A	N/A
2	Use python and other machine learning packages	User	N/A	N/A	N/A
3	Use a cooling pad to cool down the system	User	N/A	N/A	N/A
4	Introduce breaks whilst using the computer	User	N/A	N/A	N/A
5	Keep back-up files to OneDrive	User	N/A	N/A	N/A

Responsible manager's signature: Print name: Date:	Responsible manager's signature: Print name: Date

Assessment Guidance

1. Eliminate	Remove the hazard wherever possible which negates the need for further controls	If this is not possible then explain why
2. Substitute	Replace the hazard with one less hazardous	If not possible then explain why
3. Physical controls	Examples: enclosure, fume cupboard, glove box	Likely to still require admin controls as well
4. Admin controls	Examples: training, supervision, signage	
5. Personal protection	Examples: respirators, safety specs, gloves	Last resort as it only protects the individual



LIKELIHOOD	1	2	3	4	5
5	5	10	15	20	25
4	4	8	12	16	20
3	3	6	9	12	15
2	2	4	6	8	10
1	1	2	3	4	5

IMPACT

Risk process

1. Identify the impact and likelihood using the tables above.
2. Identify the risk rating by multiplying the impact by the likelihood using the coloured matrix.
3. If the risk is amber or red – identify control measures to reduce the risk to as low as is reasonably practicable.
4. If the residual risk is green, additional controls are not necessary.
5. If the residual risk is amber the activity can continue but you must identify and implement further controls to reduce the risk to as low as reasonably practicable.
6. If the residual risk is red do not continue with the activity until additional controls have been implemented and the risk is reduced.
7. Control measures should follow the risk hierarchy, where appropriate as per the pyramid above.
8. The cost of implementing control measures can be taken into account but should be proportional to the risk i.e. a control to reduce low risk may not need to be carried out if the cost is high but a control to manage high risk means that even at high cost the control would be necessary.

Impact		Health & Safety
1	Trivial - insignificant	Very minor injuries e.g. slight bruising
2	Minor	Injuries or illness e.g. small cut or abrasion which require basic first aid treatment even in self-administered.
3	Moderate	Injuries or illness e.g. strain or sprain requiring first aid or medical support.
4	Major	Injuries or illness e.g. broken bone requiring medical support >24 hours and time off work >4 weeks.
5	Severe – extremely significant	Fatality or multiple serious injuries or illness requiring hospital admission or significant time off work.

Likelihood	
1	Rare e.g. 1 in 100,000 chance or higher
2	Unlikely e.g. 1 in 10,000 chance or higher
3	Possible e.g. 1 in 1,000 chance or higher
4	Likely e.g. 1 in 100 chance or higher
5	Very Likely e.g. 1 in 10 chance or higher

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016. URL `https://arxiv.org/abs/1603.04467`.

Luca Baldassarre, Lorenzo Rosasco, Annalisa Barla, and Alessandro Verri. Multi-output learning via spectral filtering. 87(3):259–301, jun 2012. ISSN 0885-6125. doi: 10.1007/

s10994-012-5282-y. URL https://doi.org/10.1007/s10994-012-5282-y.

Imanol Bilbao and Javier Bilbao. Overfitting problem and the over-training in the era of data: Particularly for artificial neural networks. In *2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 173–177, 2017. doi: 10.1109/INTELCIS.2017.8260032.

Charles Blair. Problem complexity and method efficiency in optimization (a. s. nemirovsky and d. b. yudin). *SIAM Review*, 27(2):264–265, 1985. doi: 10.1137/1027074. URL https://doi.org/10.1137/1027074.

Leo Breiman and Jerome H. Friedman. Predicting multivariate responses in multiple linear regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 59(1):3–54, 1997. doi: 10.1111/1467-9868.00054.

Ebubekir Buber and Banu Diri. Performance analysis and cpu vs gpu comparison for deep learning. pages 1–6, 10 2018. doi: 10.1109/CEIT.2018.8751930.

Alison J. Burnham, John F. MacGregor, and Román Viveros. Latent variable multivariate regression modeling. *Chemometrics and Intelligent Laboratory Systems*, 48(2):167–180, 1999. ISSN 0169-7439. doi: https://doi.org/10.1016/S0169-7439(99)00018-0. URL https://www.sciencedirect.com/science/article/pii/S0169743999000180.

Cherkassky V Cai F. Ijcnn'09: Proceedings of the 2009 international joint conference on neural networks. IEEE Press, 2009. ISBN 9781424435494. doi: 10.5555/1704555.

Wai Chan. Comparing indirect effects in sem: A sequential model fitting method using covariance-equivalent specifications. *Structural Equation Modeling-a Multidisciplinary Journal - STRUCT EQU MODELING*, 14:326–346, 05 2007. doi: 10.1080/10705510709336749.

Davide Chicco, Matthijs J. Warrens, and Giuseppe Jurman. The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation. *PeerJ Computer Science*, 7:e623, 2021. doi: 10.7717/peerj-cs.623.

Jean-Paul Chilès and Pierre Delfiner. Geostatistics: Modeling spatial uncertainty. *Wiley Series In Probability and Statistics*, 01 2012. doi: 10.1002/9781118136188.

C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467, 1968.

G. Cybenko. *Continuous Valued Neural Networks with Two Hidden Layers are Sufficient.* 1988. URL `https://books.google.co.uk/books?id=EppaHwAACAAJ`.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(null):2121–2159, jul 2011. ISSN 1532-4435.

N. Benjamin Erichson, Lionel Mathelin, Zhewei Yao, Steven L. Brunton, Michael W. Mahoney, and J. Nathan Kutz. Shallow neural networks for fluid flow reconstruction with limited sensors. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2238):20200097, 2020. doi: 10.1098/rspa.2020.0097.

Kai Fukami, Koji Fukagata, and Kunihiko Taira. Super-resolution reconstruction of turbulent flows with machine learning. 11 2019.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL `https://proceedings.mlr.press/v15/glorot11a.html`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT Press, 2017.

Shivani Gupta and Atul Gupta. Dealing with noise problem in machine learning data-sets: A systematic review. *Procedia Computer Science*, 161:466–474, 2019. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2019.11.146. URL `https://www.sciencedirect.com/science/article/pii/S1877050919318575`. The Fifth Information Systems International Conference, 23-24 July 2019, Surabaya, Indonesia.

Zhongyang Han, Ying Liu, Jun Zhao, and Wei Wang. Real time prediction for converter gas tank levels based on multi-output least square support vector regressor. *Control Engineering Practice*, 20(12):1400–1409, 2012. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2012.08.006. URL https://www.sciencedirect.com/science/article/pii/S0967066112001670.

IBM IBM Education. What is machine learning?, 2020. URL https://www.ibm.com/cloud/learn/machine-learning.

Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35 (1):221–231, 2013. doi: 10.1109/TPAMI.2012.59.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper/2013/file/ac1dd209cbcc5e5d1c6e28598e8cbbe8-Paper.pdf.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

Dragi Kocev, Sašo Džeroski, Matt D. White, Graeme R. Newell, and Peter Griffioen. Using single- and multi-target regression trees and ensembles to model a compound index of vegetation condition. *Ecological Modelling*, 220(8):1159–1168, 2009. ISSN 0304-3800. doi: https://doi.org/10.1016/j.ecolmodel.2009.01.037. URL https://www.sciencedirect.com/science/article/pii/S0304380009000775.

Ruho Kondo, Shunsuke Yamakawa, Yumi Masuoka, Shin Tajima, and Ryoji Asahi. Microstructure recognition using convolutional neural networks for prediction of ionic conductivity in ceramics. *Acta Materialia*, 141:29–38, 2017. ISSN 1359-6454. doi: https://doi.org/10.1016/j.actamat.2017.09.004. URL https://www.sciencedirect.com/science/article/pii/S1359645417307383.

J. Nathan Kutz Steven L. Brunton Krithika Manohar, Bingni W. Brunton. Data-driven sparse sensor placement for reconstruction: Demonstrating the benefits of exploiting known patterns. *IEEE Control Systems*, 38(3):63–86, jun 2018. doi: 10.1109/mcs.2018.2810460.

Yunfei () Li, Juntao () Chang, Chen () Kong, and Ziao () Wang. Flow field reconstruction and prediction of the supersonic cascade channel based on a symmetry neural network under complex and variable conditions. *AIP Advances*, 10(6):065116, 2020. doi: 10.1063/5.0008889. URL `https://doi.org/10.1063/5.0008889`.

Guangcan Liu, Zhouchen Lin, and Yong Yu. Multi-output regression on the output manifold. *Pattern Recognition*, 42(11):2737–2743, 2009. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2009.05.001. URL `https://www.sciencedirect.com/science/article/pii/S0031320309001691`.

Charles A. Micchelli and Massimiliano Pontil. On learning vector-valued functions. *Neural Computation*, 17(1):177–204, 2005. doi: 10.1162/0899766052530802.

Beyond Minds. an introduction to super-resolution using deep learning, 2021. URL `https://beyondminds.ai/blog/an-introduction-to-super-resolution-using-deep-learning`.

Ramesh Kumar Mojjada, Arvind Yadav, A.V. Prabhu, and Yuvaraj Natarajan. Machine learning models for covid-19 future forecasting. *Materials Today: Proceedings*, 2020. doi: 10.1016/j.matpr.2020.10.962.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Feiping Nie, Hu Zhanxuan, and Xuelong Li. An investigation for loss functions widely used in machine learning. *Communications in Information and Systems*, 18:37–52, 01 2018. doi: 10.4310/CIS.2018.v18.n1.a2.

Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951. doi: 10.1214/aoms/1177729586. URL https://doi.org/10.1214/aoms/1177729586.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016. URL https://arxiv.org/abs/1609.04747.

Oliver T. Schmidt and Tim Colonius. Guide to spectral proper orthogonal decomposition. *AIAA Journal*, 58(3):1023–1033, 2020. doi: 10.2514/1.J058809. URL https://doi.org/10.2514/1.J058809.

Timo Similä and Jarkko Tikka. Input selection and shrinkage in multiresponse linear regression. *Computational Statistics Data Analysis*, 52(1):406–422, 2007. ISSN 0167-9473. doi: https://doi.org/10.1016/j.csda.2007.01.025. URL https://www.sciencedirect.com/science/article/pii/S0167947307000205.

Yisheng Song and Wei Mei. Structural properties of tensors and complementarity problems. *Journal of Optimization Theory and Applications*, 176(2):289–305, 2018.

M. Srivastava and Tumulesh Solanky. Predicting multivariate response in linear regression model. *Communications in Statistics-simulation and Computation - COMMUN STATIST-SIMULAT COMPUT*, 32:389–409, 01 2003. doi: 10.1081/SAC-120017497.

Sean Symon, Denis Sipp, and Beverley J. McKeon. A tale of two airfoils: resolvent-based modelling of an oscillator versus an amplifier from an experimental mean. *Journal of Fluid Mechanics*, 881, 2019. doi: 10.1017/jfm.2019.747.

Grigorios Tsoumakas, Eleftherios Spyromitros-Xioufis, Aikaterini Vrekou, and Ioannis Vlahavas. Multi-target regression via random linear target combinations. In Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 225–240, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44845-8.

Jack V. Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11): 1225–1231, 1996. doi: 10.1016/s0895-4356(96)00002-9.

Devis Tuia, Jochem Verrelst, Luis Alonso, Fernando Perez-Cruz, and Gustavo Camps-Valls. Multioutput support vector regression for remote sensing biophysical parameter estimation. *IEEE Geoscience and Remote Sensing Letters*, 8(4):804–808, 2011. doi: 10.1109/LGRS.2011.2109934.

Shuo Xu, Xin An, Xiaodong Qiao, Lijun Zhu, and Lin Li. Multi-output least-squares support vector regression machines. *Pattern Recognition Letters*, 34:1078–1084, 07 2013. doi: 10.1016/j.patrec.2013.01.015.

Muhamad Yani, S Irawan, and Casi Setianingsih. Application of transfer learning using convolutional neural network method for early detection of terry's nail. *Journal of Physics: Conference Series*, 1201:012052, 05 2019. doi: 10.1088/1742-6596/1201/1/012052.

Jian Yu and Jan Hesthaven. Flowfield reconstruction method using artificial neural network. *AIAA Journal*, 57:1–17, 11 2018a. doi: 10.2514/1.J057108.

Jian Yu and Jan Hesthaven. Flowfield reconstruction method using artificial neural network. *AIAA Journal*, 57:1–17, 11 2018b. doi: 10.2514/1.J057108.

Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012. URL `https://arxiv.org/abs/1212.5701`.

Mauricio A. Álvarez, Lorenzo Rosasco, and Neil D. Lawrence. Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266, 2012. ISSN 1935-8237. doi: 10.1561/2200000036. URL `http://dx.doi.org/10.1561/2200000036`.