Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF AERONAUTICS, FACULTY OF ENGINEERING

Data-driven hydrodynamic stability analysis of turbulent flows

Author: Iaroslav Kazakov (01334502)

Academic Supervisor: Dr Georgios Rigas

Second Marker: Dr Oliver Buxton

A dissertation submitted in partial fulfillment of the requirements for the degree of MEng Aeronautical Engineering

June 1, 2021

Abstract

This report presents the application of nonlinear Neural Networks (NN) to the turbulent wake behind an axisymmetric body. We set up a particular form of Neural Networks - autoencoder and perform a parametric analysis to find the network structure that yields the best reconstruction of the flow in terms of its mean squared error (MSE). We compare the NN reconstructed flow with the linear Proper Orthogonal Decomposition (POD) method and quantitatively showcase the advantages of the NN. We demonstrate the energetic capacity of both techniques in terms of total kinetic energy and analyse the temporal variables pertinent to each approach outlining the challenges associated with the stochastic nature of the NN and how they affected the results. We apply POD to the NN outputs and observe changes in the decomposed modes as well as the temporal coefficients. Lastly, we suggest improvements and explore the effects of 'disabling' neurons in the NN's latent space on the MSE and the energetic capacity.

Acknowledgements

I want to thank Dr Rigas for his professionalism, intelligence, guidance and approachability. This project was insightful and joyful not only because the topic itself interested me a lot but because my supervisor was the brightest researcher I had ever had an opportunity to collaborate with. I wish him all the success in his career and hope that our paths cross sometime in the future. I would also like to thank Simon Kneer and Branco-Roan Arsenov, whose previous work and ideas helped me tremendously. Lastly, I would like to thank all my family members who supported me throughout this course; it was quite a journey.

Contents

Acknowledgements **iii**

1 Introduction **1**

2 Theory **2**
2.1 Proper Orthogonal Decomposition . . . 2
2.1.1 History . . . 2
2.1.2 Methodology . . . 2
2.1.3 Method of Snapshots . . . 3
2.2 Neural Networks . . . 3
2.2.1 Introduction to Nonlinear Neural Networks . . . 3
2.2.2 History . . . 4
2.2.3 Architecture and Hyperparameters . . . 4
2.2.4 Backpropagation Optimisation . . . 5
2.2.5 Optimisation Methods . . . 5
2.2.6 Activation Functions . . . 5
2.2.7 Learning Rate . . . 6
2.2.8 Epochs and Batch Size . . . 7
2.2.9 Autoencoders . . . 7
2.3 Power Spectral Density . . . 8
2.3.1 Methodology . . . 8
2.4 Similarity Factor . . . 9

3 Data and Implementation **9**
3.1 PIV and Data Collection . . . 9
3.1.1 Axisymmetric Body . . . 9
3.2 Data Preprocessing and Normalisation . . . 10
3.3 Code Implementation . . . 11
3.3.1 Training the NN . . . 11
3.3.2 NN Reconstruction . . . 13
3.3.3 Snapshot POD . . . 14

4 Results and Discussion **15**
4.1 POD Analysis . . . 15
4.1.1 Energy Distribution . . . 15
4.1.2 POD modes . . . 16
4.2 Parametric Analysis . . . 17
4.2.1 Choice of Hidden Layers and Activation Functions . . . 17
4.2.2 Converging and Diverging Depth Hidden Layers . . . 19
4.2.3 Batch Size, Epochs, Learning Rate . . . 21
4.2.4 Choice of NN Architecture . . . 22
4.3 NN Flow Reconstruction . . . 22
4.4 NN and POD Reconstruction . . . 24
4.4.1 Qualitative Difference . . . 24

4.4.2 Mean Squared Error 26
4.5 Energetic Capacity of NNs and POD Modes 27
4.6 Temporal Analysis 28
4.7 POD of the NN Outputs 32

5 Future Work **34**
5.1 POD Analysis 34
5.2 Parametric Analysis 34
5.3 Error, Energy and Similarity 35
5.4 Disabling Latent Neurons 35
5.4.1 4 LN Architecture 35
5.4.2 8 LN Architecture 37

6 Conclusion **38**

References **39**

Appendix **39**

A Appendix **42**
A.1 NN Flow Reconstruction Section 42
A.2 Temporal Analysis Section 43
A.3 POD of the NN Output Section 44
A.4 Disabling Latent Neurons Section 47

1 Introduction

I am an old man now, and when I die and go to heaven there are two matters on which I hope for enlightenment. One is quantum electrodynamics, and the other is the turbulent motion of fluids. And about the former I am rather optimistic.

- Horace Lamb

Turbulence, though discovered as early as in the 1870s and introduced by William Thomson in 1877 [1], is still not fully understood. It is an unsolved mystery that can make a golf ball fly faster [2] and cause an airplane passenger to feel unwell. The problem with solving turbulent flows is the Navier-Stokes (NS) equation, the nonlinear equation that explains fluid motion. Although scientists found various approximations of turbulent flows to obtain an analytical solution to the Navier-Stokes equation, solving it in general form is a grand challenge.

The absence of an analytical solution prompted researchers to come up with statistical, numerical and deterministic approaches. However, all of these methods pose certain limitations. Statistical methods split a flow into two parts: a fluctuation component and a time average, leading to the Reynolds-averaged Navier-Stokes equation (RANS). RANS creates a closure problem constraining the number of geometries for which solutions can be found [3]. Similarly, deterministic approaches rely on various assumptions when solving NS equation, neglecting certain phenomena, such as viscosity, compressibility, 3D effects etc. As per numerical approaches, with computational advantages in microchips and software optimisation, finding a numerical solution to NS became slightly more feasible. Because the NS consists of four terms in Cartesian tensor form, two of which are extremely challenging to account for numerically (namely convective and pressure gradient terms), solving a Direct Numerical Simulation (DNS) for a simple channel flow can take up to six months requiring hundreds of thousands of processor cores [4]. Hence, with limitations presented by all three major approaches, new data-driven methods can be introduced to obtain insights into understanding turbulence.

This thesis focuses on two major data-driven approaches: Proper Orthogonal Decomposition (POD) and nonlinear Neural Networks (NN). Below, we use POD results to benchmark the NN performance and assess which method reconstructs the turbulent flow more accurately. We briefly reconfirm the POD results obtained by Arsenov [5] and create a NN that can reconstruct the turbulent flow from a highly compressed latent space with minimal error. The major difficulty with NN methods is its lack of results interpretability. POD as a method is mathematically understandable, and the modes produced can somewhat be verified across a range of Reynolds numbers using certain experimental set-ups. Nonlinear NNs, on the contrary, are vastly less interpretable because of the backpropagation algorithm, which optimises the architecture using multivariant gradient descent (MGD). The question we answer in this research is not why the NN outputs what it outputs but rather how can we compare the outputs and some elements of the architecture to the POD analysis. We want to

understand the similarities between the flows reconstructed by the NN and the POD outputs and compare the reconstruction error. We also want to get an incentive into the energetic capacity of the NN and observe whether nonlinear activation functions can reconstruct more energy with fewer latent nodes. Finally, we would like to understand whether certain nodes or sets of nodes in the latent space are responsible for certain flow features and observe their temporal behaviour utilising the Welch method. Lastly, we combine the NN outputs with the POD analysis to see the effects of latent space compression on the modes and temporal expansion coefficients.

2 Theory

2.1 Proper Orthogonal Decomposition

2.1.1 History

In the world of fluid dynamics, flows can share distinct features that can be found across various non-dimensional parameters, such as Reynolds number. Studying these physical features can be insightful if one is interested in understanding the origins of certain fluid phenomena. Proper Orthogonal Decomposition, historically known as Principal Component Analysis (PCA), is a linear method invented by Karl Pearson in 1901 [6]. In 1967 Lumley proposed the proper orthogonal decomposition theorem of probability theory to detect coherent spatial patterns in turbulent flows [7]. This linear method was discussed by several researchers in great detail in the past [8], [9], [10] and although well-understood on a mathematical level, the physical interpretability of the coherent structures one observes using is still an ongoing research area.

2.1.2 Methodology

We start with a fluctuation component of the data set that can be represented in a truncated format using orthonormal basis functions and their respective random coefficients. The fluctuation vector is the difference between the vector field and its temporal mean:

$$x(\zeta,t)-\bar{x}(\zeta)=\sum_j a_j\phi_j(\zeta,t) \tag{1}$$

Where a_j and ϕ_j correspond to a random coefficient and an orthonormal vector, respectively. This expression represents the flowfield using an extension to the general form of a Fourier series. To achieve domain independence the spatial (ζ) and temporal domains (t) can be separated and be reexpressed into:

$$x(\zeta,t)-\bar{x}(\zeta)=\sum_j a_j(t)\phi_j(\zeta) \tag{2}$$

Now the series has two separate components, where a_j indicates a temporal value and ζ corresponds to the spatial vector on a grid.

The objective of the POD analysis is to evaluate the optimal basis vectors that can represent a given data best in terms of L2 error. To find the set of orthonormal basis functions ϕ_j one needs to solve the following eigenvector problem:

$$R\phi_j = \lambda_j \phi_j \tag{3}$$

Where λ_j represents eigenvalue and the kinetic energy content of each mode, and $\phi_j \in \mathbf{R}^n$ is the associated orthogonal basis vector. Depending on whether one uses spatial or snapshot method, the covariance matrix can either be $R \in \mathbf{R}^{n\times n}$ or $R \in \mathbf{R}^{m\times m}$. Note, for a snapshot method $\phi_j \in \mathbf{R}^m$. The full derivation of this method can be found in the paper by Taira et al [11].

2.1.3 Method of Snapshots

For the comparative analysis below, only the snapshot method was considered. This method is computationally faster because the dimensionality of a reduced covariance matrix is much smaller if $n << m$ where n and m are temporal and spatial dimensions respectively [12]. The reduced covariance matrix can be found:

$$R_{ij} = \frac{1}{m} x(\zeta, t_i) x(\zeta, t_j) \tag{4}$$

Where m is the number of snapshots, i.e. temporal data observations, and x is the fluctuation component of the data. Fundamentally, the method of Snapshot is built on solving the following eigenvalue problem:

$$R\psi_j = \lambda_j \psi_j \tag{5}$$

Although a different eigenvalue problem is being solved, the nonzero eigenvalues are the same for both covariance matrices [11]. The reduced eigenvector problem can be related to the original orthonormal vectors through:

$$\phi_j = X\psi_j \frac{1}{\sqrt{\lambda_j}} \tag{6}$$

Where $\lambda_j \in \mathbf{R}^n$. Hence, the original orthonormal basis modes can be restored.

2.2 Neural Networks

2.2.1 Introduction to Nonlinear Neural Networks

Neural Networks have become of profound interest to international aerodynamics society. Data-driven approaches became practical tools for modelling nonlinear unsteady aerodynamics of various aircraft configurations where the analytical methods failed to emerge due to large-scale dynamics [13]. For example, Artificial Neural Networks (ANNs) can be used to reduce nonlinear noise effects behind nacelles and imposed vibrations on airframe [14]. Although lacking interpretability of what the data-driven approaches output at each stage, the computational power of Neural Networks had only become more robust in the recent decade [15]. In the sections below, we explain the constituents of a typical NN and their effect on the NN performance.

2.2.2 History

Neural networks were first proposed in 1943 by Warren McCulloch and Walter Pitts [16]. Neural nets, as they were called back then, were a scientific area emerging from both neuroscience and computer science. In 1957, Kolmogorov published a paper concerning the representation of arbitrary continuous functions and their mapping to Neural Networks layers [17], [18]. The full potential only became apparent when Kolmogorov's theorem was extended by Cybenko into the Universal Approximation Theorem (UAT), claiming that any continuous function can be approximated by one hidden layer of Neural Networks [19]. Nonlinear NNs built on this theorem and coupled with further success in microchips, and computational power had fully proven their dominance in many areas.

2.2.3 Architecture and Hyperparameters

A Neural Network is characterised by two elements - its architecture and its hyperparameters. A typical Neural Network architecture consists of an input layer, hidden layers and an output layer. An input of an observable layer is an output of a preceding layer. If each node (a.k.a neuron) in each layer is connected to each neuron from the preceding and succeeding layer - a network is called fully connected. A typical Neural Network with three hidden layers can be seen below:

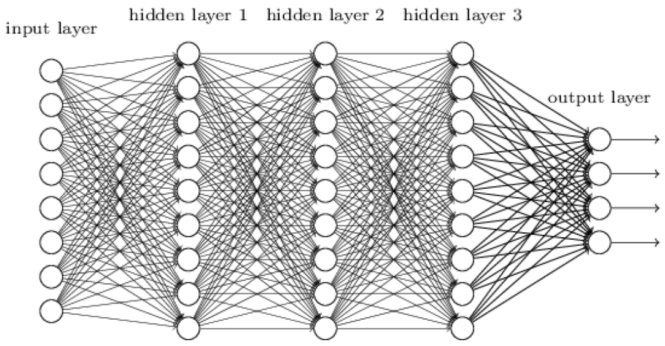


Figure 1: A representation of a Neural Network with one input layer, three hidden layers and one output layer.

The aim is to find the output vector Y:

$$Y = F(X, W, b) \tag{7}$$

Where $X = [x_1, x_2, x_3...]^T$ is the input vector, $W = [W_{11}, W_{12}, W_{13}...; W_{n1}, W_{n2}, W_{n3}...]$ is the weight matrix where each row corresponds to a layer n and $b_n = [b_1, b_2, b_3...]$ is the bias vector at a layer n. Weights and biases are the parameters that are optimised using the backpropagation optimisation technique. Each hidden layer, taking the output from a preceding layer, applies a function (known as activation function F),

multiplies the output by weight and adds bias, passing the signal forward through each layer. Hence, the output Y can be stated as:

$$Y = F({W_N}^T ... (F({W_2}^T (F({W_1}^T X + b_1)) + b_2)) ... + b_N)) \tag{8}$$

The advantage of NNs is that the applied activation function F can be strongly nonlinear. As a result, using UAT, a relatively small number of neurons in a hidden layer can capture significant nonlinear effects pertinent to the model. Thus, the choice of activation functions and hidden layer architecture plays an essential role in the predictability of NNs.

2.2.4 Backpropagation Optimisation

"Training" a NN model means "tuning" weights and biases such that an objective function is minimised. The objective function we use is the L2 loss function between the inputs and outputs:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (Y_i - X_i)^2 \tag{9}$$

Before the optimisation starts, a user initialises the algorithm at a certain point with a particular guess of weights and biases. Backpropagation optimisation works by applying a small perturbation to weight or bias arrays estimating the effects on the output and recording the cost function. This procedure is repeated multiple times until no further perturbation in weights and biases improves the objective function [20].

2.2.5 Optimisation Methods

The choice of an optimisation algorithm for weights orbiases tuning is a grey area as there is not a single best algorithm that works well on all problems. The most popular optimisation algorithms include Stochastic Gradient Descent (SGD), GD with momentum, Adam and Adagrad. For this research set-up, Adam was used as it proved to perform well on aerodynamics datasets in the past [21].

2.2.6 Activation Functions

Activation functions play a significant role in NN reconstruction capacity. The main challenge is that the choice is purely subjective, and little information about one's dataset can be utilised to justify a selection of a given activation function. If the architecture of a NN is trivial, then the choice can be dictated by physical interpretations of a data set, but the NN architecture was rather complicated in this research. A standard set of suitable activation functions include:
1) Tanh(x)- hyperboloic tangent. Used when the symmetry of data can be exploited.

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{10}$$

2) Sigmoid(x) - sigmoid function. Used when the data needs to be bounded between 0 and 1.

$$sigmoid(x) = \frac{1}{e^{-x}+1} \tag{11}$$

3) ReLU(x)- rectified linear unit. Used when the nodes should only contain positive values. Note that this does not mean the entire Network contains only positive values so long as the biases are allowed.

$$ReLU(x) = max(0, x) \tag{12}$$

Graphically these functions are:

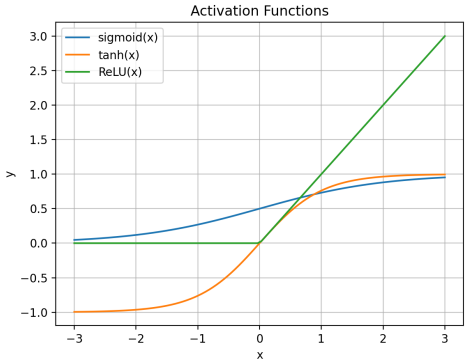


Figure 2: Activation Functions: Sigmoid, Tanh, ReLU.

2.2.7 Learning Rate

The learning rate (LR) is a hyperparameter that controls how much to change the gradient magnitude in response to the estimated error each time the model weights are updated [22]. If the LR is too small, the training rate can be lengthy, and the optimisation can get stuck. A value too large can overshoot minima points and never converge, leading to sub-optimal weights. Graphically, the learning rate can be interpreted as the length of the red vector on a 2D plane:

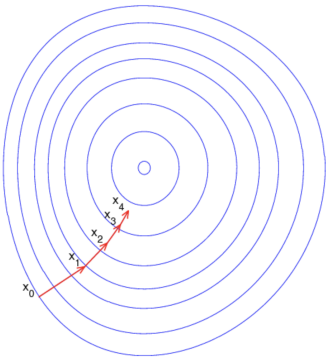


Figure 3: Effects of adjustable learning rate on a 2D plane. The LR is the length of the red vector, changing its value the closer the vector to the local minimum.

In general it is best to have an LR that is not fixed to a particular value and changes on every iteration depending on proximity to the stationary point. This adjustable approach was utilised in the NN architecture below.

2.2.8 Epochs and Batch Size

Epoch number indicates how many times the entire training dataset goes through the Network, i.e. this number refers to one complete cycle through the dataset. The motivation to use epochs is to give the Network a chance to see the previous data to readjust the model parameters so that the model is not biased towards the last few data points during training [23].

Batch size refers to the number of observations used to train a model at one complete pass through the Network. It is advantageous not to use all the available samples in one propagation. Since one trains the Network using fewer samples, the overall training procedure requires less memory. Additionally, the training is faster because, with batches, the Network updates more frequently as opposed to a single sample set. The effects of batching and finding local minima can be demonstrated as follows:

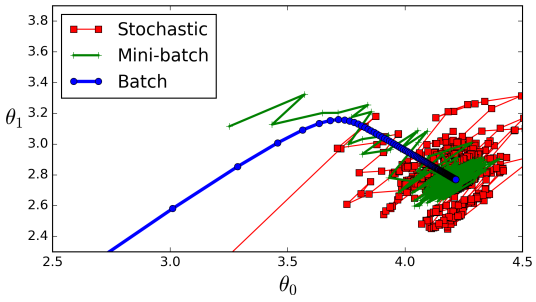


Figure 4: Convergence to the local minimum on a 2D plane. Three types of batch size: Stochastic, Mini-batch, Batch. The size of the batch controls the convergence speed [24].

2.2.9 Autoencoders

In this research, a special type of Neural Networks called autoencoders was used. This architecture consists of three major parts: 1) encoder, connecting inputs with the latent space through hidden layers, 2) a single layer Latent Space, consisting of latent neurons, and 3) decoder, connecting the latent space with the outputs through the number of hidden layers. As per Figure 16, the interpretation of this architecture can be regarded as information compression with the maximum compressibility factor at the latent space (encoder) and subsequent information decompression (decoder). Because of significant compressibility factors, i.e. the ratio of information at the input relative to the information at the latent space, these architectures can ”learn” to filter out the noise and leave out only the ”necessary” information. This architecture couple with a nonlinear activation function is a nonlinear analogy of POD. Based on the number of nodes in the latent space the reconstruction can im-

prove or worsen. Hence, we can regard the latent space as the spatial modes of POD and benchmark the performance of a given NN appropriately [25].

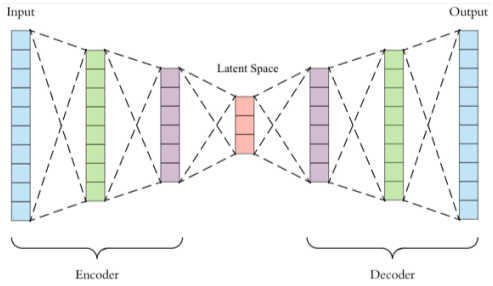


Figure 5: A visual representation of an autoencoding architecture. Three main parts: Encoder, Latent Space, Decoder.

2.3 Power Spectral Density

POD provides insightful information on the spatial characteristics of the flow but gives no information on temporal behaviour. To understand how the temporal coefficients a_j and the value of the latent neurons (LNs) vary in time, Power Spectral Density (PSD) analysis was used.

2.3.1 Methodology

Researchers in [26] utilised the PSD analysis to detect frequencies that contain the largest amount of energy at certain POD modes. This idea motivated the implementation of this technique, encouraging us to compare the energy content at specific frequencies for both POD temporal coefficients and Neural Network latent neurons. This analysis would help us understand whether the two techniques locate PSD peaks at different frequencies. Power Spectral Density is a measure of a signal's power content versus its frequency. It is an extension of the Fourier transform of a signal:

$$PSD = \frac{1}{f_s N}|x|^2 \tag{13}$$

Where f_s is the sampling frequency of Particle Image Velocimetry (PIV) and N is the number of samples. Since the signal x has a discrete nature, the PSD can be expressed:

$$PSD = \frac{1}{f_s N}\left(\frac{1}{\sqrt{N}}\sum_{n=0}^{N-1} x_n e^{-2i\pi kn/N}\right)^2 \tag{14}$$

Because the nature of the signal is stochastic, a direct application of PSD causes the noise to interfere with the "clean" signal. Welch's method can be used to mitigate this issue. Instead of taking the whole domain of a signal, one can divide the data into snippets and define an overlapping value between the snippets. Then by averaging across the PSD of all the snippets with certain overlaps, one can achieve better results in terms of noise cancellation [27].

2.4 Similarity Factor

Qualitative differences can be easily reproduced using graphical packages of a programming language. However, one also needs to compare the results quantitatively in terms of their similarity. For instance, when comparing the results produced by NN and POD, graphical discrepancies can easily be visualised, but these visualisations would not tell the reader the whole story. Therefore, we introduce a vector space model method that allows comparing two vectors of data. For two vectors **x** and **y**, there exists an inner product measuring the cosine of the angle between them:

$$cos(\theta) = \frac{\mathbf{x} \cdot \mathbf{y}}{|\mathbf{x}| \cdot |\mathbf{y}|} \tag{15}$$

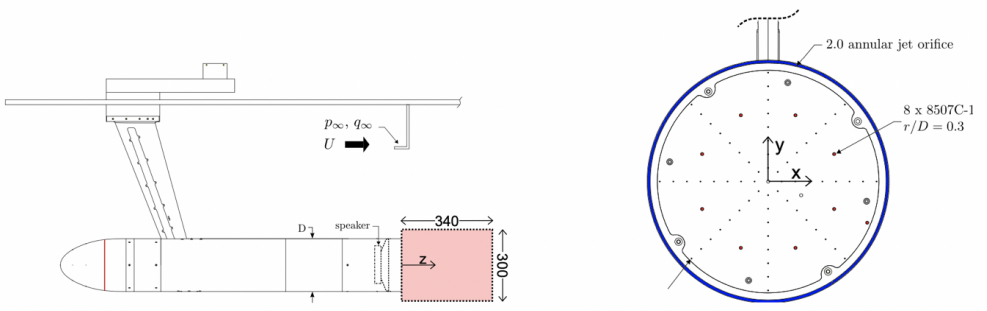
Cosine similarity is a judgment of orientation and not magnitude. Suppose the two vectors have the same orientation. In that case, their cosine similarity is 1; otherwise, two vectors oriented at 90° relative to each other have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.

3 Data and Implementation

3.1 PIV and Data Collection

This section covers the details of the turbulent dataset used for the Modal and Neural Network analyses. The data was collected using Particle Image Velocimetry, a technique allowing to estimate both streamwise and vertical velocities of a flow. The obtained PIV data was collected in the wake of the axisymmetric bluff body with a blunt nose.

3.1.1 Axisymmetric Body



a) Side view b) Rear view

Figure 6: Schematic representation of the axisymmetric body.

Figure 6 shows the axisymmetric body from the side and the rear views. The experiment was run at $U_\infty = 15$ m/s and $Re_D = 2x10^5$. The body itself had a diameter of

$D = 196.5$ mm and the length-to-diameter ratio of $L/D = 6.48$. The red box behind the body corresponds to the grid where the experimental data was collected. The grid was discretised into 164 x 187 evenly spaced mesh with a sampling frequency of 720 Hz.

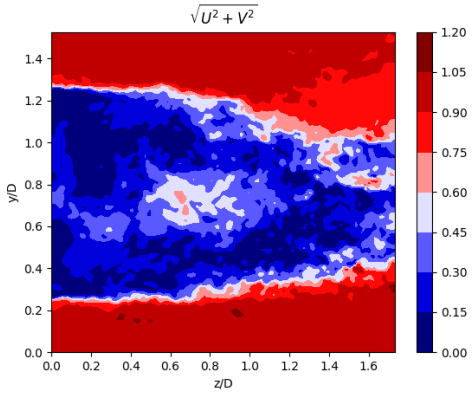


Figure 7: The 100$^{\text{th}}$ snapshot of the total velocity of the turbulent wake past the axisymmetric body.

The experimental data was divided into 13 databases, each one containing 2732 snapshots. Each database contained both streamwise velocity U and vertical velocity V. Thus there were 35516 temporal values of each velocity vector at each grid point. In terms of the spatial part of the array, each velocity component contained 30668 spatial locations on the grid, leading to the total data matrix of 61336 x 35516 size. Figures 7 and 8 demonstrate a 100$^{\text{th}}$ snapshot of the total velocity wake and the mean components of the velocity field, respectively.

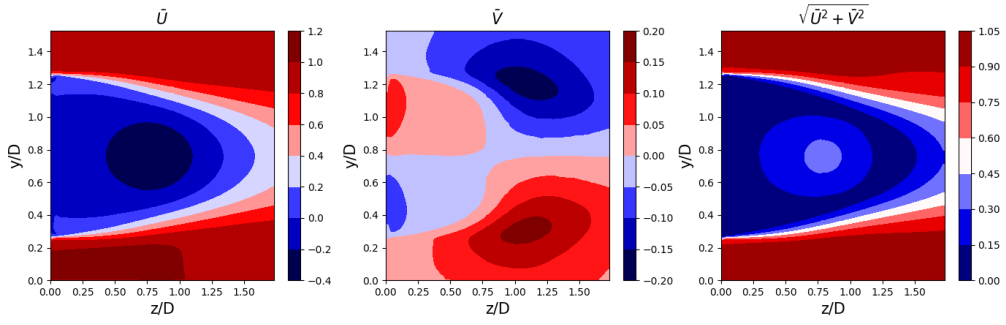


Figure 8: Three mean velocity components of the turbulent wake past the axisymmetric body.

3.2 Data Preprocessing and Normalisation

This report aimed to produce consistent results in terms of bluff body dimensions as well as the speed of the flow. To allow for a comparison between different flows and geometries, the following quantities had to be non-dimensionalised:

Table 1: Non-dimensionalisation of velocity, frequency and boyd length.

Original Quantity	Velocity (m/s)	Frequency (Hz)	Length (m)
Non-dimensionalisation	Freestream U_∞	Strouhal Number (St)	Diameter (D)
Formula	U/U_∞	L/D	$St = fD/U_\infty$

Additionally, since the POD analysis and the NN architectures consider the fluctuation components of the velocity, the time-averaged components $\bar{U}$ and $\bar{V}$ were subtracted from the respective U and V. Hence, both techniques only consider the fluctuating, mean-normalised components of the flowfield.

3.3 Code Implementation

This section describes how the Python 3.7 code works. Three main the pieces of script contained NN Training, NN Reconstruction and Snapshot POD implementation.

3.3.1 Training the NN

It was first necessary to obtain weights and biases that can be used to reconstruct an unknown turbulent flow dataset.
We loaded the libraries and read the data:

```
#-------------------------
#Libraries loading section
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers as ks
from sklearn.model_selection import train_test_split
import pickle as pk
import mat73
#Data reading section
file_name="/home/ik2517/Databases/AxisymmetricBody/PIV-Sync/Data/
   PIV10.mat"
data_dict10 = mat73.loadmat(file_name)
PIV_data10= data_dict10['PIVdata']
U10 = PIV_data10['vz'] #Store the streamwise velocity array
V10 = PIV_data10['vy'] #Store the horizontal velocity array
#-------------------------
```

Listing 1: Data reading section.

Note that it was not always feasible to use all 35516 snapshots during the training stages due to the project's time constraints. Sometimes, the number of training splits was reduced to a single database containing only 2732 snapshots. This change did not significantly affect the accuracy of the results.
Next, we cleaned the data and normalised the vectors.

```
# Data normalisation and cleaning
Uinf = 15
U = np.nan_to_num(U10)/Uinf
V = np.nan_to_num(V10)/Uinf
```

```
U_bar = np.mean(U, axis=2)
V_bar = np.mean(V, axis=2)
U_dash = U-U_bar[:, :, None]
V_dash = V-V_bar[:, :, None]
U_dash = U_dash[:, :, np.newaxis, :]
V_dash = V_dash[:, :, np.newaxis, :]
X = np.concatenate((U_dash, V_dash), axis=2)
training_set=np.transpose(X, (3, 0, 1, 2))
X=training_set
#------------------------
```

Listing 2: Data cleaning and normalisation section.

Having concatenated the two velocity vectors together and transposed the data vector such that the temporal component goes first, the hyperparameters, hidden layers and activation functions were defined.

```
#Hyper parameters and Activation Functions
def relu_activation(x):
    return tf.nn.relu(x) #Define the ReLU function
learning_rate = 0.001
batch_size = 50
epochs = 125
split = 0.3 #The ratio between the test size and sample size
patience = 20 # Number of epochs with no improvement after which
    training will be stopped
LS = 16 #number of latent space neurons
n_hidden = 303 #number of neurons in hidden layer 1
n_hidden_2 = 303 #same but for layer 2
#------------------------
```

Listing 3: Defining hyperparameters and the activation function.

We were then ready to define the layers, feed the data and set up an autoencoding Neural Network model.

```
#------------------------
# Set up the model
X_train, X_test, y_train, y_test = train_test_split(training_set,
    training_set, test_size=split,random_state=1)
#Part: Encoder
input= ks.Input(shape=(X.shape[1], X.shape[2], X.shape[3]))
# Encoder
layer1 = ks.Reshape([X.shape[1] * X.shape[2] * X.shape[3]])(input)
layer2 = ks.Dense(n_hidden, use_bias=True, activation=relu_activation
    )(layer1)
layer3 = ks.Dense(n_hidden_2, use_bias=True, activation=
    relu_activation)(layer2)
encoded = ks.Dense(LS, use_bias=True, activation=None)(layer3)
#Part: Decoder
latent_inputs = ks.Input(shape=(LS))
layer4 = ks.Dense(n_hidden_2, use_bias=True, activation=
    relu_activation)(latent_inputs)
layer5 = ks.Dense(n_hidden, use_bias=True, activation=relu_activation
    )(layer4)
layer6 = ks.Dense(X.shape[1] * X.shape[2] * X.shape[3], use_bias=True
    , activation=None)(layer5)
```

```
decoded = ks.Reshape([X.shape[1], X.shape[2], X.shape[3]])(layer6)
Encoder_model = tf.keras.models.Model(input, encoded)
Decoder_model = tf.keras.models.Model(latent_inputs, decoded)
Autoencoder_model = tf.keras.models.Model(input, Decoder_model(
    Encoder_model(input)))
#------------------------
```

Listing 4: NN model set-up.

We needed to define an optimiser and choose a loss function. It was also advised to create a stopping condition for when no further improvements can be achieved in the objection function. We used ”EarlyStopping” for this reason.

```
#------------------------
# Optimisation Algo and EarlyStopping
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    learning_rate,decay_steps=2000,decay_rate=0.2,staircase=False)
opt = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
Autoencoder_model.compile(optimizer=opt, loss='mse')
early_stop = tf.keras.callbacks.EarlyStopping(monitor='loss',
    patience=patience,  restore_best_weights=True)
history = Autoencoder_model.fit(X_train, y_train,epochs=epochs,
    batch_size=batch_size,shuffle=True,validation_data=(X_test, y_test
    ),callbacks=[early_stop],verbose=0)
Autoencoder_model.save_weights('axisymmetric/models/{}MODES_{}HIDDEN_
    {}HIDDEN_relu'.format(LS, n_hidden, n_hidden_2))
#------------------------
```

Listing 5: Defining the optimisation algorithm and data fitting.

Lastly, we fitted the training set and saved the weights and biases to reconstruct an ”unseen” datasets in the future.

3.3.2 NN Reconstruction

Having trained the model and saved the outputs, we were ready to reconstruct the turbulent field. The training and reconstruction stages had to be separated because the CPU and GPU could get overloaded. This piece of the code is similar to the one where training occurred. The only difference is in the final few lines of the script.

```
#------------------------
# EVERYTHING ELSE IS THE SAME AS BEFORE. Predict the outputs.
Autoencoder_model.compile(optimizer=opt, loss='mse')
Autoencoder_model.load_weights('axisymmetric/models/{}MODES_{}HIDDEN_
    {}HIDDEN_tanh'.format(LS, n_hidden, n_hidden_2))
P = Autoencoder_model.predict(training_set, batch_size=None, verbose
    =0, steps=None, callbacks=None)
U = P[:, :, :, 0]
V = P[:, :, :, 1]
#------------------------
```

Listing 6: Predicting the flow after the model training.

3.3.3 Snapshot POD

The code below describes how the snapshot modal decomposition was implemented. Note that the normalisation of the dataset is the same for both POD and NNs.

```
#------------------------
#Load the libraries and read the data
import numpy as np
from matplotlib import pyplot as plt
import pickle
from scipy.sparse.linalg import eigs
with open('/home/ik2517/YaroProject/RIGAS_NONLIN/POD/stream_velocity.
    pickle', 'rb') as file:
    U = pickle.load(file) #Normalised streamwise component
with open('/home/ik2517/YaroProject/RIGAS_NONLIN/POD/
    vertical_velocity.pickle', 'rb') as file:
    V = pickle.load(file) #Normalised vertical component
#------------------------
```

Listing 7: Loading the libraries and reading the normalised velocity components.

Then velocity matrices had to be stretched out in a single column vector as in [11]. Using the column vectors, the eigenvector problem can be set up. Notice, the problem is solved in a reduced form, for only the first 200 eigenvalues-eigenvectors are stored in the memory.

```
#------------------------
# Reshape and concatenate the vectors. Then solve EIG problem.
U_dash = U.reshape(164*187, 35516, order='F') # Reorder the matrices
    such that each column is stretched out into a single row
V_dash = V.reshape(164*187, 35516, order='F')
Vel_dash = np.concatenate((U_dash, V_dash), axis=0)
Vel_dash_T = Vel_dash.T
R = np.dot(Vel_dash, Vel_dash_T)/35516
print(R.shape)
eigval, eigvec = eigs(R, 200)
#------------------------
```

Listing 8: Reshaping the matrices and solving eigenvalue problem.

Finally, the temporal coefficient matrix a_j and the associated modes ϕ_j were found:

```
#------------------------
#Finding modes and temporal coefficients
mod = np.dot(eigvec, np.diag((35516*eigval.real)**(-0.5)))
modes = np.dot(Vel_dash, mod) #Modes phi
a = np.dot(Vel_dash_T,modes) #Temporal coefficients
#------------------------
```

Listing 9: Finding the modes.

4 Results and Discussion

4.1 POD Analysis

This subsection presents the results of the Snapshot POD analysis applied to the turbulent flow behind the axisymmetric body. Part of the project included validation and improvement of the analysis performed in [5] since the NN results had to be benchmarked against the POD results. The MATLAB results presented by Arsenov had to be reconfirmed using Python 3.7 with minor differences in computational accuracy due to the specifics of Python algorithms.

4.1.1 Energy Distribution

We started with an understanding of how much energy can be stored in a given mode. We estimated the ratio energy by taking the eigenvalue associated with a particular eigenvector and dividing it by the total kinetic energy of the flow. Following this method, the energy distribution across different modes is given in Figure 9.

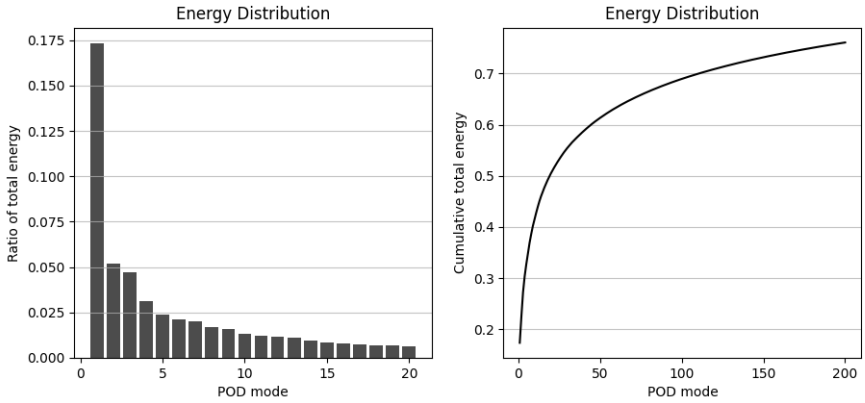


Figure 9: Energy distribution: bar and cumulative line plots of POD modes.

One can observe that the energy content per mode quickly diminishes along the x-axis. For instance, mode 1 contains 17.33%, whereas mode 20 has only 0.64%, so the higher-order modes contain less energy. These results contradict the distributions obtained in [5] because even with 200 modes, the total energy line is less than 0.9. The discrepancy arises due to the method behind the total kinetic energy estimation, whereby Arsenov uses the sum of the 200 eigenvalues instead of taking a more precise sum of the squared velocities. The table below demonstrates how the energy capacity increases as you add modes:

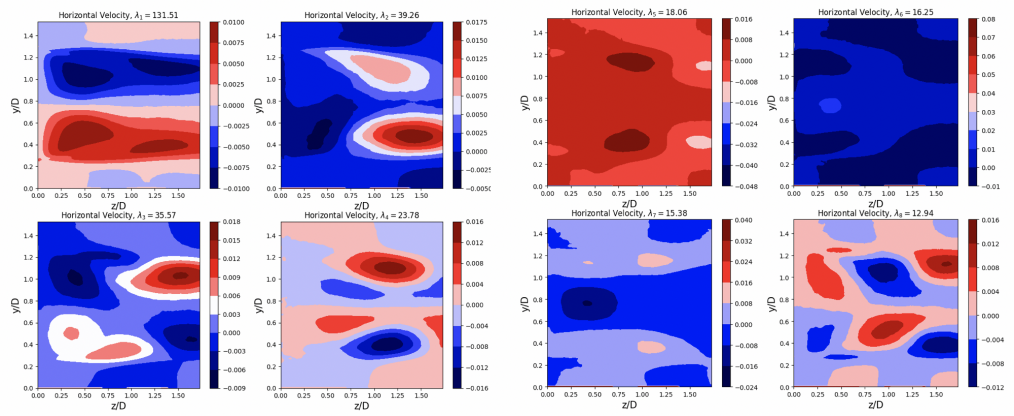
Table 2: Numerical values of energy content of POD modes and the relative improvement in per cents with respect to the previous mode.

Modes	1	2	3	4	5	6	7	8	20
% Change		24	13	11	8.3	4.3	3.8	2.7	31
POD	0.17	0.22	0.27	0.31	0.32	0.34	0.38	0.39	0.51

The % Change explains by what percentage the energy changes with an increase in the number of modes. Note that with only 20 modes, one can restore more than 50% of the total energy and 77% of the total flow energy with 200 modes.

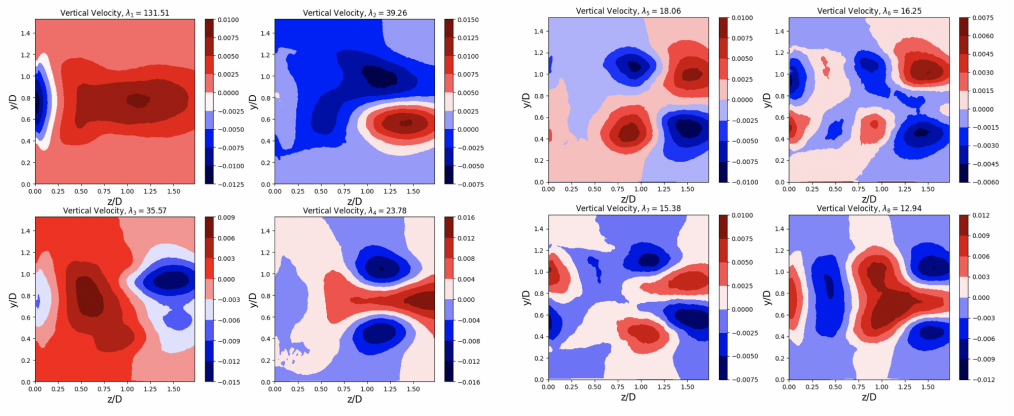
4.1.2 POD modes

Having found the eigenvalues, the modes of the wake behind the axisymmetric body were plotted. We begin with the reproduction of the modes 1-8 of both horizontal and vertical velocity.



a) Modes 1-4 b) Modes 5-8

Figure 10: Horizontal velocity POD modes of the axisymmetric body. Modes: 1-8.



a) Modes 1-4 b) Modes 5-8

Figure 11: Vertical velocity POD modes of the axisymmetric body. Modes: 1-8.

The first horizontal mode is the most energetic one and can be interpreted according to its symmetry along the z-axis. Findings in [26] had shown that a somewhat similar global mode is responsible for breaking the symmetry at a low Reynolds number ($Re \approx 420$). This first bifurcation breaks the symmetry by evolving the field into

the 3D flow. What is noteworthy is the fact that the analysed flow had $Re = 2x10^5$, meaning that this symmetry breaking mode can be shared across different Reynolds numbers.Vortex shedding that can be observed in mode 3 and 5 (Figure 10) are dominant features of the flow. These modes resemble Von Karman street, whereby the regions of positive and negative velocities are being shed into the wake. Similar patterns can be seen in the vertical modes 2 and 3, where the distinct areas of alternating velocities can be spotted. These regimes can be cross-validated against the modes obtained by decomposing vortex shedding flow. Other modes are harder to interpret because their nature is not fully understood.

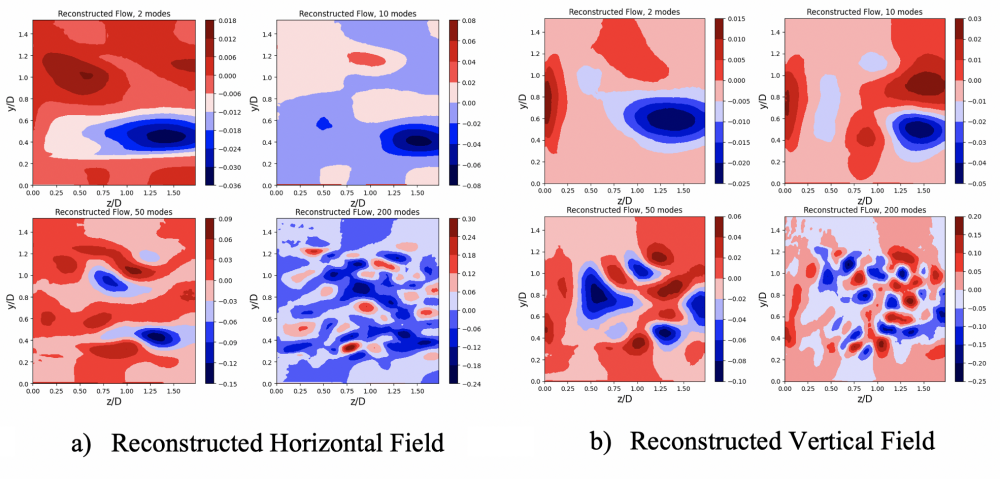


Figure 12: Reconstructed field of vertical and horizontal velocities using different number of Modes at the 100$^{\text{th}}$ snapshot. Number of modes used: 2, 10, 50, 200.

Using the procedure outlined in Section 2, the flow can be reconstructed at a particular snapshot. The more modes one uses, the closer the reconstructed flow matches the original flow.

4.2 Parametric Analysis

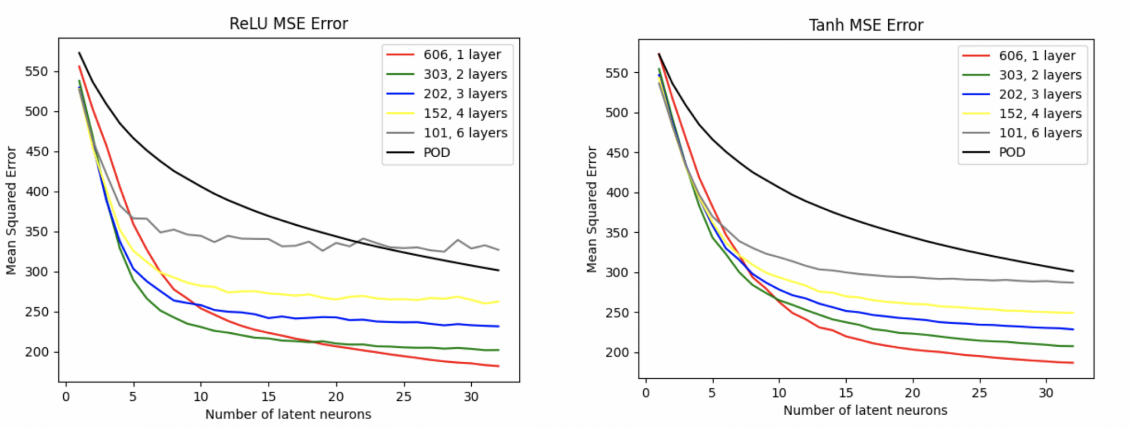
4.2.1 Choice of Hidden Layers and Activation Functions

In this section, we perform a parametric analysis of the Neural Network. We aim to find the best architecture to minimise the mismatch between the outputted NN flow and the original flow. To evaluate the performance of a given autoencoder, we introduce the Mean Squared Error (MSE):

$$MSE = \left(\sum_{n=1}^{N} \left(\sum_{m=1}^{2N_x N_y} (y_m - x_m)^2 \right) \right) * \frac{1}{N} \tag{16}$$

Where $y_m - x_m$ describes the difference between the outputted and input velocity on the grid. N_x and N_y represent the spatial coordinate on x and y axes, respectively. N stands for the temporal snapshot over which the sum of the grid points is averaged.

We started by plotting MSE versus the number of latent Neurons for specific NN architectures. The goal was to observe the effects of the depth and the number of neurons in the latent space on the error. In Figure 13, we compared how the MSE varied across architectures with different activation functions and benchmarked the results against a linear decomposition technique represented by a black line. Note that the first number shown in the legend box indicates the number of the nodes per one hidden layer in both decoding and encoding parts. Besides, note that the total number of nodes stayed constant across the hidden layers. The total number of nodes in the hidden layers was chosen to be proportional to the total number of spatial points in the input layer as advised by NN practitioners [28].



a) ReLU Function MSE b) Tanh Function MSE

Figure 13: MSE versus number of latent neurons for different depth architectures and two activation functions: Tanh and ReLU.

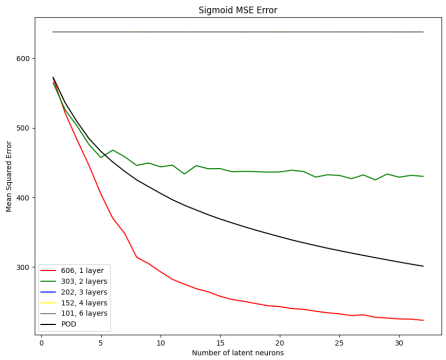


Figure 14: MSE versus number of latent neurons for different depth architectures and one activation functions: Sigmoid.

A larger number of latent neurons in the latent space reduces the MSE asymptotically for all three cases. The depth, i.e. the number of layers used in the encoder and decoder, plays a vital role. For ReLU, an architecture with two hidden layers, i.e. 303 neurons in each, performs better on the region between three neurons and

15 neurons than any other architecture presented in these three plots. An NN architecture with eight neurons yields 234 units of MSE compared to 438 units of MSE produced by POD. Moreover, analogous tanh and sigmoid architectures output 267 and 453 units of MSE, respectively, implying the dominance of the ReLU activation function. To summarise, the best candidates in terms of MSE for a given activation function with eight latent nodes are:

Table 3: Best in terms of MSE constant width deep network architectures with 8 latent nodes for three activation functions: Sigmoid, Tanh, ReLU. NN's MSE is benchmarked relative to the POD MSE.

Activation Function	Sigmoid	Tanh	ReLU
Best Architecture	606, 1 layer	303x303, 2 layers	303x303, 2 layers
MSE	302	267	234
POD % (*)	45.0	64.0	87.2

Eight latent nodes were chosen as a median of the range between one and 16 nodes, a range of POD modes one would typically want to reconstruct flow. (*) indicates the per cent improvement relative to POD. Thus, the ReLU architecture with two hidden layers was used in all computations. This ReLU architecture decreased the MSE error by 87.2% compared to the analogous POD results.

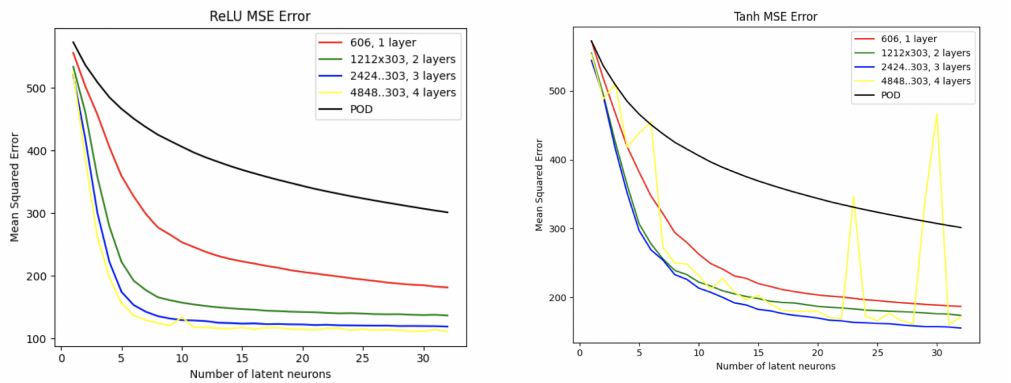
4.2.2 Converging and Diverging Depth Hidden Layers

This section researches deep layer architectures such that the compression decompression of information occurs smoothly, i.e. starting from a wider hidden layer and finishing with the narrower ones when approaching the latent space. The same number of neurons for all hidden layers suffices, but sometimes starting with more neurons at the start and subsequently decreasing them can improve performance [29].In Figure 15 one can see a similar distribution in terms of MSE for a given activation function. The legend box describes four architectures and the POD results that the reader saw earlier. The four architectures follow the same "proportionality" rule, i.e. number of nodes proportional to the input layer width. The architectures displayed in the legend box are:

1. one hidden layer, 606 neurons each.
2. two hidden layers, 1212x303 neurons each.
3. three hidden layers, 2424x1212x303 neurons in each respective hidden layer.
4. four hidden layers, 4848x2424x1212x303 neurons in each respective hidden layer.

More nodes in the latent space result in smaller MSE, converging asymptotically. The best model is ReLU 4848x2424x1212x606 neurons in each respective deep layer. The ReLU MSE resulting from such architecture with eight latent neurons is 120

units which is 256% better than the MSE obtained from eight POD modes. Similar tanh architecture also produced good results yielding 211 MSE units leading to 107% improvement benchmarking against POD results. In convergent and divergent architectures with more neurons, the decrease in MSE occurs more rapidly. The MSE with ReLU activation function plateaus out at around 15 nodes for its best hidden layer architecture, implying that most of the improvements occur within the first 10 latent nodes.



a) ReLU Function MSE (varying width) b) Tanh Function MSE (varying width)

Figure 15: MSE versus number of latent neurons for varying number of neurons deep architectures and two activation functions: Tanh and ReLU.

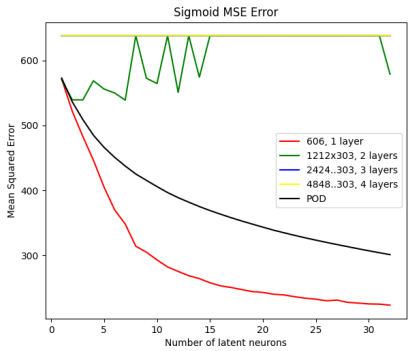


Figure 16: MSE versus number of latent neurons for varying number of neurons deep architectures and one activation functions: Sigmoid.

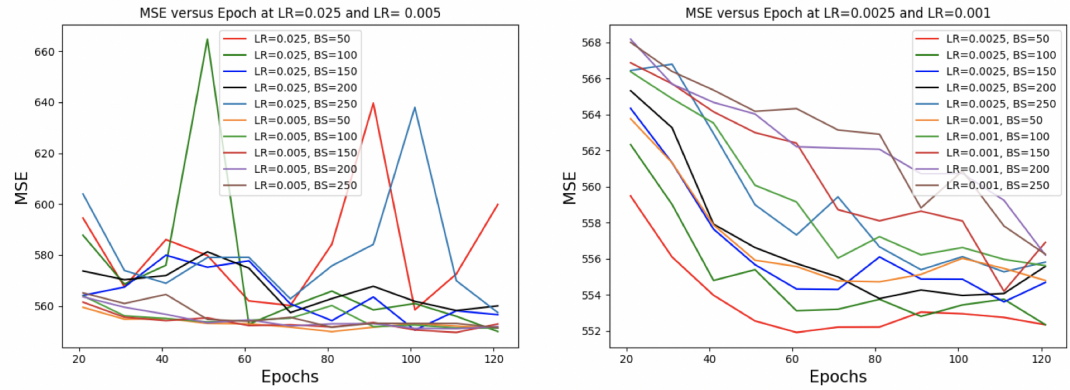
Table 4: Best in terms of MSE varying in number of neurons deep network architectures for three activation functions: Sigmoid, Tanh, ReLU. NN's MSE is benchmarked relative to the POD MSE.

Activation Function	Sigmoid	Tanh	ReLU
Best Architecture	606, 1 layer	2424..303, 3 layers	4848..303, 4 layers
MSE	302	211	120
POD % (**)	45.0	107	256

where (*) has the same meaning as the above. Sigmoid function did not perform well in this case either, diverging from the POD MSE line, as more neurons were used in the hidden layers.

4.2.3 Batch Size, Epochs, Learning Rate

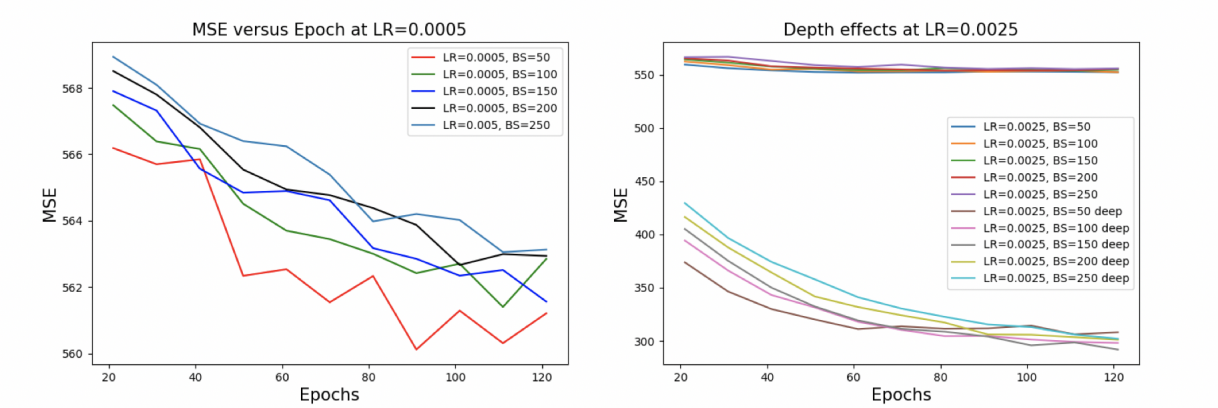
This section describes the effects of batch size (BS), epochs and learning rate on the MSE for a particular NN architecture. Although the outlined parameters were known to affect the training time majorly, the effects on the MSE were unclear. We began by setting up the semi-best NN architecture with a single constant width (606 neurons each) autoencoder with a ReLU activation function. First, the effects of the epochs and batch size were explored. Figures 17 (a) and (b) and Figure 18 (a) demonstrate the effects of an increasing number of epochs, given learning rate and batch size:



a) LR= 0.025 and LR=0.005 b) LR= 0.0025 and LR=0.001

Figure 17: (a) MSE vs epochs for a given batch size for LR=0.025 and LR=0.005, (b) MSE vs epochs for a given batch size for LR=0.0025 and LR=0.001.

An increase in the number of epochs size does not significantly affect the MSE for a learning rate lower than 0.025. The spikes in MSE one can observe for LR = 0.025 could arise because this value of LR is considered to be generally high. A good rule of thumb for sequential Neural Networks is to start with at least LR=0.005 or smaller and the Figures 17 and 18 support this ”empirical” recommendation. For learning rates smaller than 0.025, the payoff in MSE for an increasing epoch number is somewhat distinguishable. MSE decreases with an increase in epochs but not substantially; thus, this dependence is unclear and requires more testing. As per the batch size, the lines are slightly pushed up for an increase in BS, meaning that MSE slightly increases for BS larger than 50.



a) LR= 0.025 and LR=0.005 b) LR= 0.0025 and LR=0.001

Figure 18: (a) MSE vs Epochs for a given batch size for LR=0.0005, (b) Hidden layer depth effects on the MSE for a given batch size for LR=0.0025.

To emphasise the observed insignificance of the three hyperparameters, the depth effect from Figure 18 (b) was demonstrated. By changing the depth from one hidden layer with 606 neurons to two hidden layers with 303x303 neurons each, one is able to decrease the MSE substantially. The effect of the batch size also became more apparent, whereby the trends lines are shifted more heavily with an increasing number of batches. At 20 Epochs and LR=0.0025 with two hidden layers, the reconstruction MSE at BS=50 is 348, with MSE = 462 at BS=250. Hence, BS can affect different architectures differently. Therefore, the research below considers at least two hidden layer architectures with BS=50, LR=0.001 and Epochs=125. With a sub-optimal yet suitable NN depth, the three hyperparameters had proven to minimise the MSE and outputted reasonable training time.

4.2.4 Choice of NN Architecture

Note that the parametric analysis had been explored continuosly throughout the research because of the computational time it took to train the NN. The best ReLU architecture with a varying number of nodes was only discovered later into the project. Most of the research below was done with a 303x303, two hidden layer ReLU architecture which was sub-optimal. However, due to better MSE performance, Neurons Disabling section considered a converging/diverging hidden layer structure with ReLU 1212x303 neurons layers.

4.3 NN Flow Reconstruction

Having intermediately decided on the two hidden layers 303x303 architecture, the reconstructed flow was then produced to visually observe the differences between the actual flow and the Neural Network output. Note that since the latent neurons (LN) number was the parameter of interest, this number was not fixed to a certain value. The results below consider the NN reconstructed flows at a certain snapshot for a varying number of latent neurons.

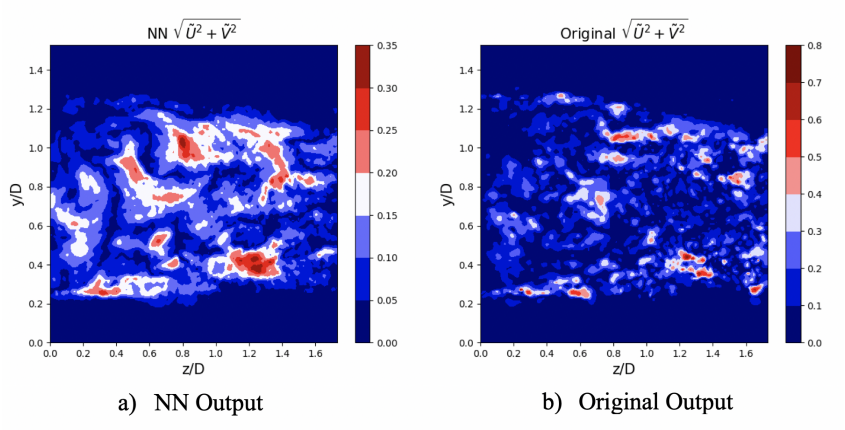


Figure 19: Comparison of the total velocity vector at the 100$^{\text{th}}$ snapshot produced by the 8 LN NN with the original flow.

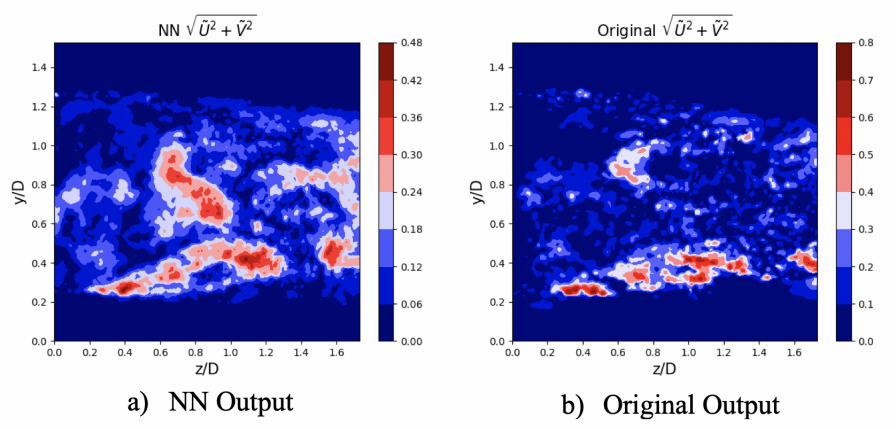


Figure 20: Comparison of the total velocity vector at the 200$^{\text{th}}$ snapshot produced by the 8 LN NN with the original flow.

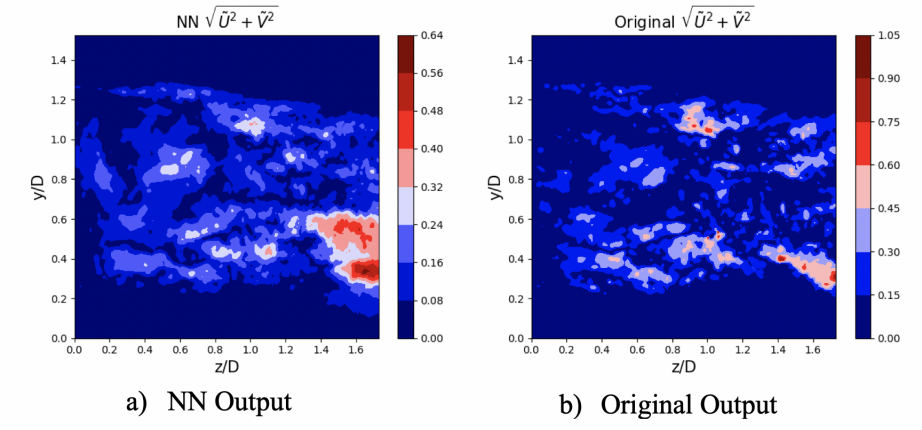


Figure 21: Comparison of the total velocity vector at the 300$^{\text{th}}$ snapshot produced by the 8 LN NN with the original flow.

The agreement between the reconstruction and the original flow is striking, considering that only eight LNs were used in the latent space. Notably, the NN can identify the regions of varying velocity magnitudes with such precision. For instance, the NN can relatively accurately identify the region of higher velocity on the 200th Snapshot at $z/D \in [0.2, 1.4]$ and $y/D \in [0.2, 0.5]$. This similarity is confirmed in the Table 5 below:

Table 5: Similarity factor between the 8LN NN architecture output and the original flow for three snapshots: 100, 200, 300. Average of the similarity factors over all 35516 snapshots.

Snapshot Number/Average	# 100	# 200	# 300	Average (*)
Similarity factor	0.8326	0.8105	0.8015	0.8316

Where (*) indicates the average similarity factor across all the snapshots. Hence the agreement is substantial considering the direction of the overall velocity vector. Note that the similarity factor was estimated based on the total velocity, not the velocity components separately. On a side note, the NN also satisfies the average velocity constraint of U and V By looking at the mean horizontal and vertical components of the velocity from Figure 42 in the Appendix, one can spot that the magnitudes are technically zero.

4.4 NN and POD Reconstruction

This section presents the comparison of the POD and NN reconstructions with the original flow for a particular snapshot. As outlined before, the number of latent nodes and the number of POD modes play pivotal roles in the reconstruction. We compared the reconstruction error as well as the reconstruction itself for a varying number of LNs and modes.

4.4.1 Qualitative Difference

We began by visually analysing the reconstruction of the 101st snapshot of the total velocity $\sqrt{\widetilde{U}^2 + \widetilde{V}^2}$. We analysed four sets of modes and LNs: 2, 8, 16, 32, comparing the reconstructions and the similarities between the flows.

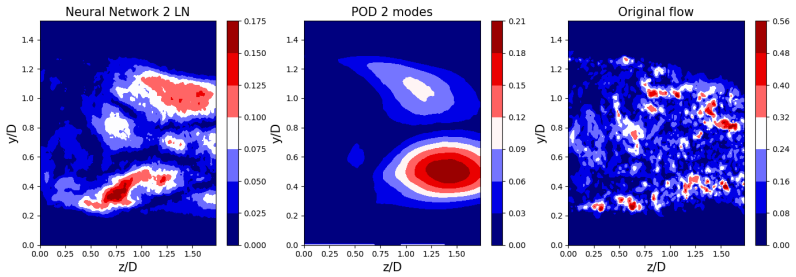


Figure 22: Comparison of 2 POD modes vs 2 latent space neurons reconstructions for $\sqrt{\widetilde{U}^2 + \widetilde{V}^2}$ at 101st snapshot.

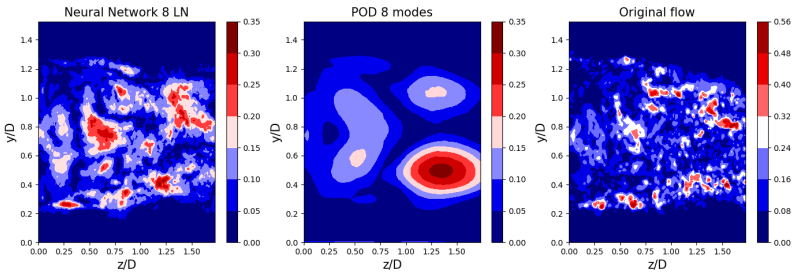


Figure 23: Comparison of 8 POD modes vs 8 latent space neurons reconstructions for $\sqrt{\widetilde{U}^2 + \widetilde{V}^2}$ at 101$^{\text{st}}$ snapshot.

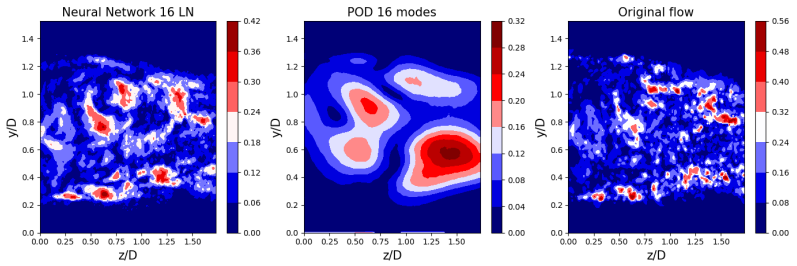


Figure 24: Comparison of 16 POD modes vs 16 latent space neurons reconstructions for $\sqrt{\widetilde{U}^2 + \widetilde{V}^2}$ at 101$^{\text{st}}$ snapshot.

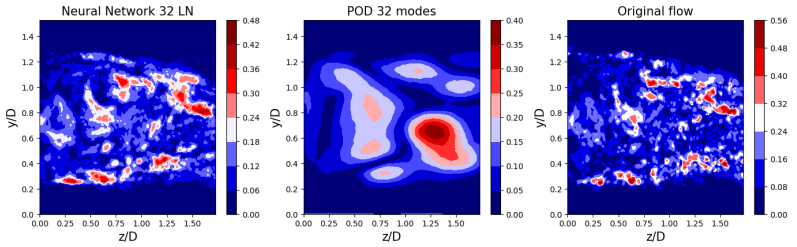


Figure 25: Comparison of 32 POD modes vs 32 latent space neurons reconstructions for $\sqrt{\widetilde{U}^2 + \widetilde{V}^2}$ at 101$^{\text{st}}$ snapshot.

There is a clear trend between the number of latent neurons and modes and the comprehensiveness of the flow. The more nodes one uses, the closer the reconstruction follows the original flow. When looking at the snapshots, it became apparent that the nonlinear Neural Network with an adequate architecture can capture a lot more information about the flow in terms of details. Whilst the 16 POD modes reconstruction still has distinct regions of velocity magnitudes (such as the red vortex at $z/D = 1.5$), the NN with a respective number of nodes outputs a relatively accurate total velocity field. Visually, the slightest discrepancy occurs for a NN with 32 latent modes, whereby the regions of higher velocity (areas in red) are captured utterly precisely. Note that the analogous POD analysis fails to compete against the precision of the NN outputs.

The similarity parameters were then estimated:

Table 6: Similarity factor between the POD reconstruction (for the number of modes) and the original flow at 101st snapshot and the average over all 35516 snapshots. Similarity factor between the NN reconstruction (for the number of LNs) and the original flow at 101st snapshot and the average.

LNs/modes	# 2	# 8	# 16	# 32
POD Similarity # 101st	0.3986	0.6371	0.7222	0.7354
NN Similarity # 101st	0.6938	0.8405	0.8837	0.9520
Average POD Similarity	0.4421	0.6120	0.7191	0.7456
Average NN Similarity	0.6831	0.8316	0.8992	0.9475

The original flow was set as a reference for the similarity. The Average POD similarity factors for a total velocity somewhat agree with the previous research performed by Arsenov [5]. The average similarity between the NN and the original flow is considerably higher than the analogous POD similarity factor for every number of LNs and modes used. 32 LNs has a striking similarity with the original flow across the 35516 Snapshots yielding 0.9475 in similarity whilst the analogous POD factor only hit 0.7456. Thus, the visual observations concluded earlier can be verified by the numbers obtained in Table 6.

4.4.2 Mean Squared Error

The total velocity MSE of the respective NN and POD reconstructions were plotted and summarised in the table below.

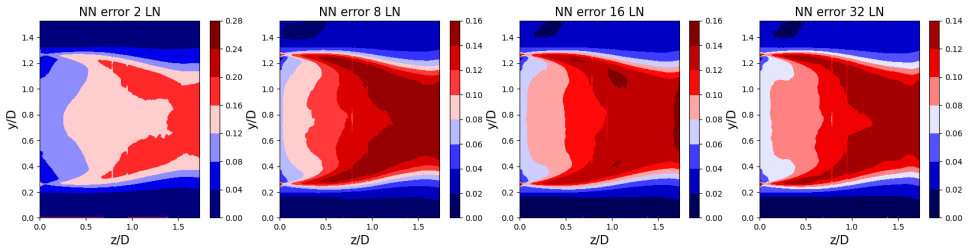


Figure 26: NN $\sqrt{\widetilde{U}^2 + \widetilde{V}^2}$ MSE for 2, 8, 16, 32 latent neurons.

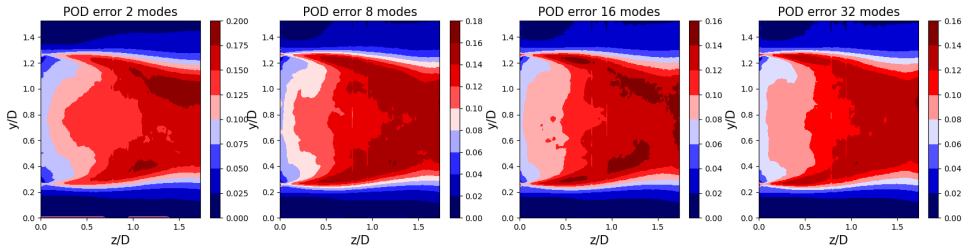


Figure 27: POD $\sqrt{\widetilde{U}^2 + \widetilde{V}^2}$ MSE for 2, 8, 16, 32 modes.

From Figures 26 and 27, the regions of error in the wake was found to be similar in a direction but different in a magnitude. The difference is substantial, especially between eight LN NN and eight modes POD configurations, whereby the Neural Network reduces the MSE by a factor of two. The MSE over the entire grid was summarised in Table 7.

Table 7: MSE value of POD and NN reconstructions for a given number of modes/latent neurons.

Modes/LNs	2	8	16	32
POD MSE	560	419	366	322
NN MSE	532	233	220	213

4.5 Energetic Capacity of NNs and POD Modes

We then embarked on studying the energetic capacity of the POD modes and Neural Networks. The Total Kinetic Energy (TKE) of the original wake behind the axisymmetric body was found to be 758.62 using $U^2 + V^2$ and averaging over the snapshot number. Hence, this value was used in the energy distributions below as the reference. The energy of the POD modes was estimated based on their respective eigenvalues as in [11], whereas the NN energy for a given latent space architecture was found from the TKE of the reconstructed flow. The plots below show the flow energy that can be restored from a particular number of modes and LNs:

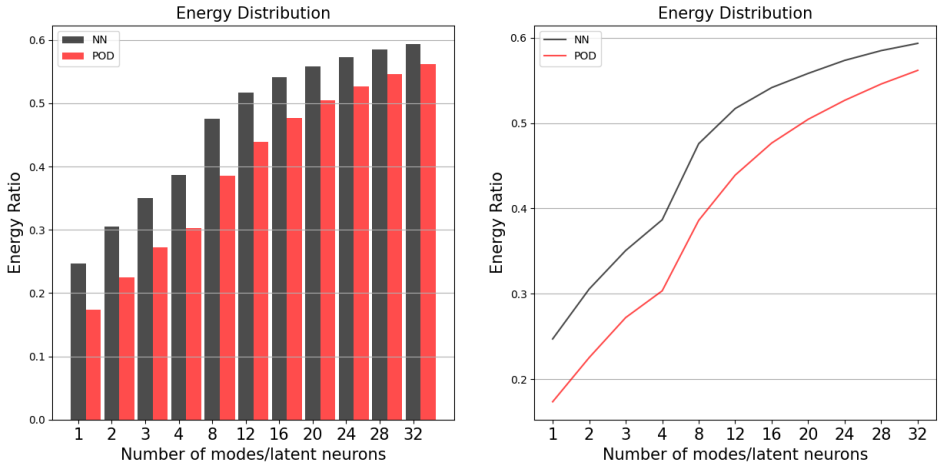


Figure 28: Energy ratio distribution of POD modes and NN latent neurons. Bar plot and line plot.

The plots demonstrate that the given Neural Network is capable of storing more energy. For instance, a NN with one latent neuron can store 25% of the TKE, whereas POD can only store 17% with one POD mode. A NN with ten LNs can retrieve approximately 50% of TKE hence most of the energy can be extracted using only ten

latent nodes. In contrast, over 20 POD modes can only be sufficient to restore more than half of the wake's energy. Interestingly, the energetic gap between the POD modes and the NN LNs decreases with an increase in the respective parameters. For example, the NN with 32 LNs is only 5.4% better than the POD output in terms of their energy content, whereas the same NN with one LN versus one mode POD offers a 47% improvement.

Table 8: Energy content of POD modes and NN reconstructions for the range of modes and LNs. Relative % change in energy content compared to the previous number of modes/LNs used.

Modes/LN	1	2	3	4	8	12	16	20	32
POD	0.17	0.22	0.27	0.3	0.39	0.44	0.48	0.51	0.56
% Change (*) (+)		24	13	11	23	8.3	3.8	3.7	1.7
NN	0.25	0.31	0.35	0.39	0.48	0.52	0.54	0.56	0.59
% Change (+)		29	23	11	30	12.8	9.1	6.3	1.8
% Difference (**)	47	41	30	30	23	18	12.5	9.8	5.4

Where (*) indicates a % change in Energetic Contents of POD modes relative to the previous modes. (**) shows a % difference between the NN and POD performance relative to the POD capacity.

4.6 Temporal Analysis

This section presents findings on the temporal coefficients (TCs) associated with POD modes and NNs. This part began with comparing the behaviour of the temporal variables of POD and the latent neurons in the time domain over a given range of snapshots. The normalised latent neuron values of a two LN Neural Network were compared against the two POD modes temporal coefficients $a_j(t)$.

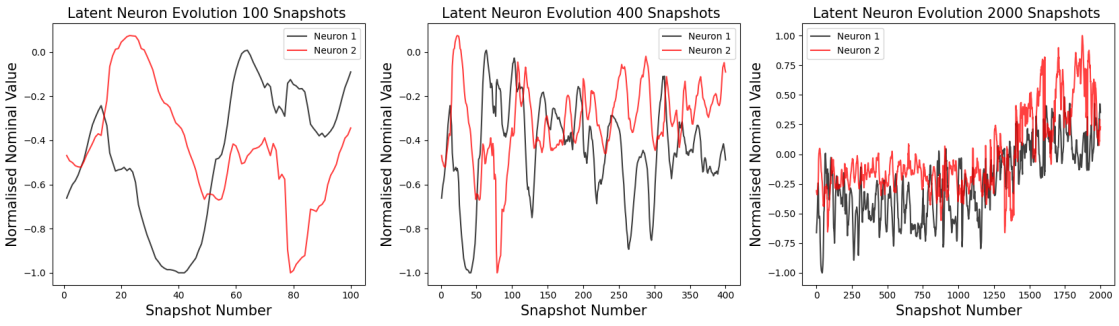


Figure 29: Latent neurons evolution in time for 100, 400 and 2000 snapshots.

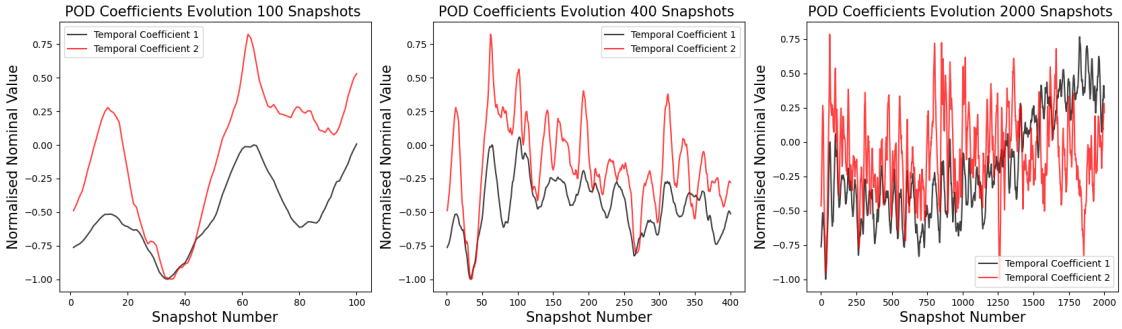


Figure 30: Temporal coefficients evolution in time for 100, 400 and 2000 snapshots.

The fundamental problem with Neural Networks is the stochastic nature of all latent nodes. The nodes one extracts from the latent space do not have an allocated number, and the enumeration of the neurons the reader can observe in the plots below are purely nominal. Unlike POD, where the modes are ordered based on their eigenvalues, i.e. energy content, Neural Network neurons cannot be structured. Hence, it is advisable to interpret the NN results below cumulatively, not comparing an individual neuron PSD with an individual POD temporal coefficient PSD where the ordering makes sense. Figure 29 and 30 demonstrate the evolution of the two LNs and TCs in time for a certain number of snapshots. The nature of the plots is stochastic, and no reasonable periodic function was able to capture the behaviour with an adequate error tolerance. More plots and trajectories can be found in the Appendix. Instead of analysing the time domain, the frequency domain was introduced. We utilised Welch's method to handle noisy signals, such as the TCs and LNs evolution. With 720 Hz as the sampling frequency of PIV, Welch's method took several additional parameters as input:

Table 9: Welch's method selected parameters.

Parameter	Window Size	Overlap Points	FFT Length
Value	512 (Hanning)	256	512

Where FFT length is the length of Fast Fourier Transform. Larger length values provide finer resolution but take longer to compute. Window Size means precisely the size of sampling windows. It reduces the side-lobe level in the spectral density estimate. The Hanning window is a taper, formed by using a weighted cosine. Overlap Points imply how many points between the subsequent blocks can overlap. Higher values afford more influence to the data at the edges, lower - to the data at the centre [30].

As per Figure 31, we began by comparing the results of two LNs NN (same architecture as before) versus the analogous two modes POD.

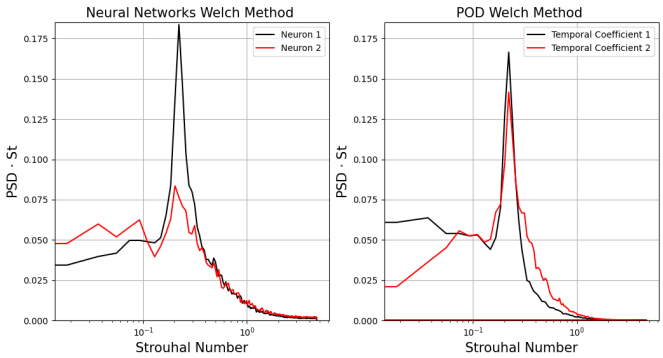


Figure 31: Power Spectral Density of the 2 LNs NN and the 2 TCs POD.

Firstly, by comparing the two figures, it is clear that the energy of the modes and latent neurons is concentrated around $St = 0.22$. Precisely, the peaks were found at the following locations:

Table 10: Observed Strouhal number (St) peaks of the 2 LNs NN and the 2 TCs POD.

LN/TC Number	1	2
NN Strouhal Number (St)	0.221	0.203
POD Strouhal Number (St)	0.221	0.221

PSD of Neuron 2 distribution looks flatter and less distinct in terms of its peaks at $St = 0.203$ compared to the POD results. The energy concentration at $St = 0.22$ was demonstrated to relate to the vortex shedding frequency and is shared across eight POD modes [5]. Due to the latent space limitation, the NN may struggle to capture the vortex shedding regime at $St = 0.22$, thus the performance may improve as more neurons are added.

We proceeded by exploring Welch method outputs for the architecture with four LNs and compared it with the respective TCs from POD:

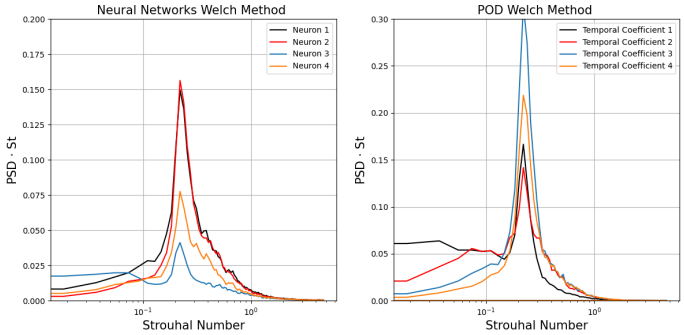


Figure 32: Power Spectral Density of the 4 LNs NN and the 4 TCs POD.

Now, with four modes in the Latent Space, the NN captures all four peaks at the same location as the POD:

Table 11: Observed Strouhal number (St) peaks of the 4 LNs NN and the 4 TCs POD.

LN/TC Number	1	2	3	4
NN Strouhal Number (St)	0.221	0.221	0.221	0.221
POD Strouhal Number (St)	0.221	0.221	0.221	0.221

Recalling that the area under the curve is the energy of a particular mode and LN, the NN returned a more spread-out energy distribution. Welch method for POD produced more concentrated regions of PSD. We repeated the same procedure with eight LNs and benchmarked against the eight TCs:

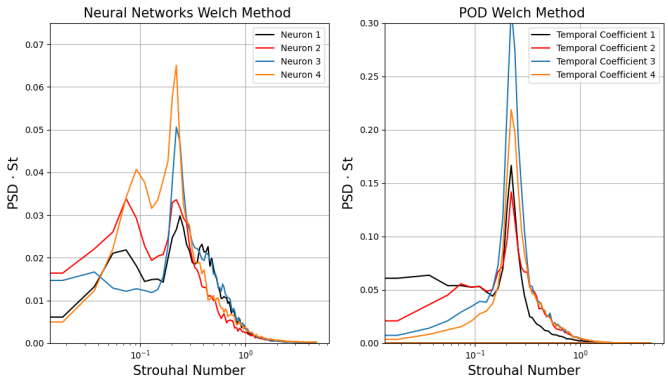


Figure 33: Power Spectral Density of the 8 LNs NN and the 8 TCs POD: First four LNs/modes.

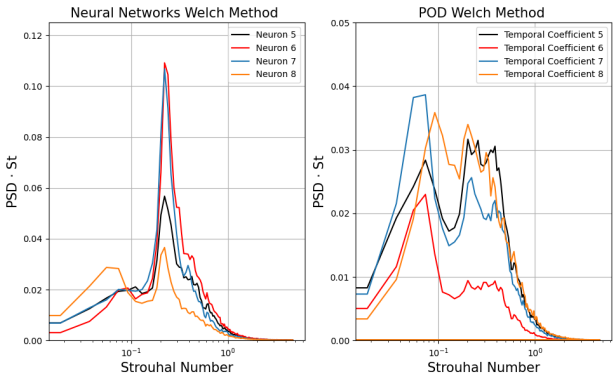


Figure 34: Power Spectral Density of the 8 LNs NN and the 8 TCs POD: Last four LNs/modes.

Figures 33 and 34 demonstrate the difficulty presented by the stochastic nature of the observed frequency peaks. From the PSD of Neuron 3 from Figure 33, it is clear that the enumeration is wrong and, in fact, cannot be allocated correctly based on a standard NumPy matrix indexing. Comparing with the analogous POD results where the modes have a clear energetic structure, the difficulty of analysing the

temporal behaviour of neurons became apparent. Based on the Strouhal number peaks observed with POD, Neurons 5, 6, 7 and 3 should be benchmarked against TCs 1-4. Hence, Neurons 1, 2, 4 and 8 should be compared against TCs 5-8. Note that certain neurons/TCs have two distinct frequency peaks, as demonstrated in the table below:

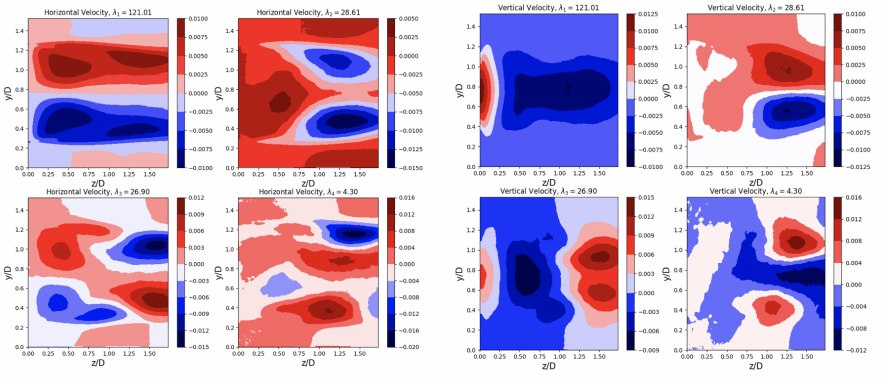
Table 12: Observed Strouhal number (St) peaks of the 8 LNs NN and the 8 TCs POD.

LN/TC Number	1	2	3	4	5	6	7	8
Peak 1 NN St	0.231	0.221	0.221	0.221	0.221	0.221	0.221	0.221
Peak 2 NN St	0.074	0.074	-	0.093	-	-	-	0.074
Peak 1 POD St	0.221	0.221	0.221	0.221	0.213	0.221	0.221	0.221
Peak 2 POD St	-	-	-	-	0.074	0.074	0.074	0.092

The agreement between the two techniques is apparent. Although we do not know which neurons map respective TCs, Neurons 5, 6, 7 and 3, as a set, map TCs 1-4. Respectively, Neurons 1, 2, 4 and 8 map TCs 5-8 and as a set, these groups have strikingly similar frequency peaks. Without any direct link between the NN latent space and the POD temporal expansion coefficients we are able to confirm the energy concentrations at $St = 0.074$ and $St = 0.221$ using both methods.

4.7 POD of the NN Outputs

In this last section, we explore the idea of combining the POD and the NN outputs. The hypothesis we tested was that the NN coupled with the autoencoder architecture acts as a filter of information. The incentive was to reproduce the spatial modes, Welch outputs and energy distributions to observe changes that NN made to the original flow in terms of its POD. We began by producing the spatial POD modes and Welch analysis of 1 LN NN outputs to observe any differences with Figures 10 and 11:



a) Horizontal modes 1-4 obtained from the output of 1 LN NN.

b) Vertical modes 1-4 obtained from the output of 1 LN NN.

Figure 35: Horizontal and vertical POD modes 1-4 of 1 LN NN reconstruction.

Figure 35 demonstrates that 1-4 POD modes of one LN NN output capture the spatial location of the axisymmetric bifurcation mode and the magnitudes relatively well. Therefore, even with a one node-compression in the latent space, the first three modes can be retrieved relatively accurately. The results improve for the four nodes and eight nodes latent space, where more latent neurons led to an improvement in higher-order modes for both horizontal and vertical components. To verify this, please refer to the Appendix. In terms of Welch method and enrgy distributions, the following plots were produced:

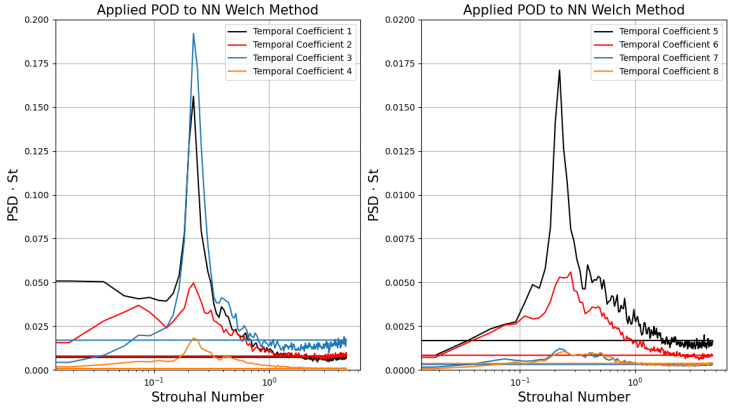
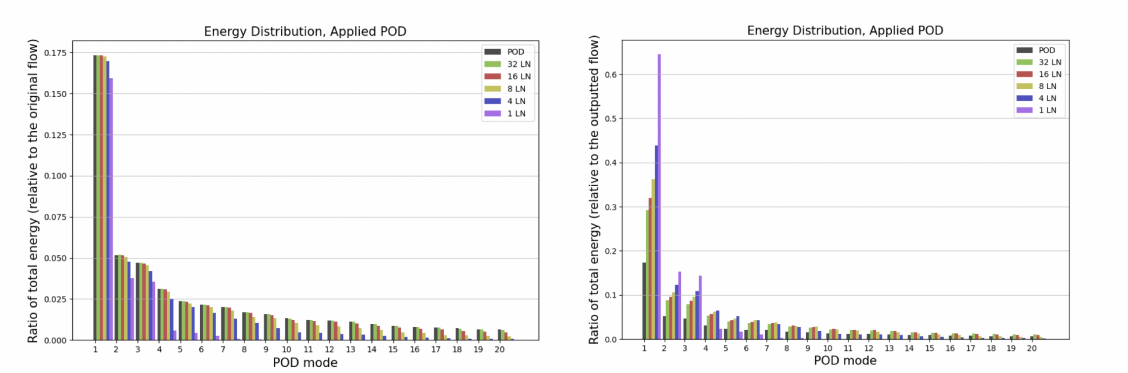


Figure 36: Welch method of POD temporal expansion coefficients applied to 1 LN NN reconstruction. TCs 1-8 are presented and split into two sets.



a) Energy Distribution of POD modes referenced to the original flow.

b) Energy Distribution of POD modes referenced to the NN reconstructed flow.

Figure 37: Effects of latent space neurons on the energy distribution of the POD modes applied to the NN reconstructed flow referenced to a) original flow and b) NN reconstructed flow. The reconstruction was obtained from 1, 4, 8, 16, 32 latent nodes architectures. POD in the legend box indicates the distribution from the POD applied to the original flow.

Interestingly, the energy peaks are attenuated for the modes which were not reproduced well by the application of POD to the NN output. For instance, TC 4 and TCs

7-8 do not have distinguishable peaks. Similarly, the associated spatial modes 4, 7 and 8 could not be produced. The same trend was observed for 4 LN and 8 LN NNs. To observe more Welch results for four LN and eight LN NNs please refer to the Appendix. Lastly, we assessed changes in the energy distribution obtained by feeding the flow dataset through architectures with 1, 4, 8, 16, 32 LNs and applying POD to the reconstructed output.

From Figure 37 a), a smaller number of neurons in LS led to allocating most of their energy to the first few modes. For example, for one LN NN POD application, most energy is allocated to modes 1-3 POD. The analogous architecture with four LNs contains most of its energy in modes 1-9. Interestingly, the NN with only 32 LNs can replicate the energy distribution of the original POD across all 20 modes.

5 Future Work

We believe that this report gave researchers an idea of where the application of Neural Networks to turbulent flows can be insightful. We hope that the reader appreciates the challenges presented by the interpretation of Neural Networks and the limitations of scientific instruments to analyse its results. In the section below, we suggest areas where improvements can be made and introduce a method to explore the effects of ”disabling” latent neurons and propose the metrics that can help understand the effects of individual neurons on the output.

5.1 POD Analysis

Although scientists confirmed specific modes experimentally, understanding the accuracy of the linear technique applied to a highly nonlinear turbulent flow is a challenge that needs to be solved. In [5], Arsenov proposed analysing travelling and standing waves that create coherent structures [31]. He suggested using the Hilbert signal for a complex model decomposition of the wake velocity field. Understanding the wake from the perspective of POD could help NN scientists to adjust their model accordingly and relate the latent space to the modes directly.

5.2 Parametric Analysis

More architectures need to be tested with different activation functions. Compression in the encoder needs to be smoother, occurring over a larger number of neurons. With the tested architectures, the compression factor in the first layer was already significant. Recall that the input layer had more than 30000 neurons, whereas the first hidden layer had only 303 neurons. As per [28], compressing information by 2/3 in every layer is a good rule of thumb, and this needs to be tested. A mix of activation can be introduced, i.e., ReLU in the first layer and tanh in the second layer. Broader ranges of BS, LRs and Epochs can also be considered. Depth can be investigated by adding more layers and keeping the number of neurons constant

for diverging and converging encoders and decoders. All of these effects need to be tested in the MSE sense.

5.3 Error, Energy and Similarity

We looked at these three parameters in isolation, but there can be a link between the three parameters. It is not intuitive to observe higher similarity factors between the flows that have lower MSEs and. However, it does not imply that an increase in MSE by one unit will trigger a decrease in similarity by 0.1 unit. The relation between these parameters can be linear/discontinuous/periodic/asymptotic - we do not know. It would be easier to compare the flows in terms of one parameter and relate to the other two parameters when required.

5.4 Disabling Latent Neurons

We believe that this section should be a good starting point for future researches in this field. This section explores the error and energy content of the reconstructed NN flow when specific neurons in the latent space are set to zero (i.e. disabled). We wanted to explore how numerically "switched on" neurons affect specific parameters. Note that the latent neurons have no digit assigned to them as the neurons are optimised stochastically. In mathematical terms setting the first temporal row of a latent variable to zero makes no difference to nullifying the second or the fourth row. Unlike POD temporal coefficients, where the modes are ordered based on their eigenvalue, i.e. energetic capacity, the NN latent neurons cannot be ordered in the same manner, and this is what we saw earlier when analysing the temporal coefficients using the Welch method. Thus, when estimating the energy and error ratios below, we took averages over some permutation to ensure somewhat precise results.

5.4.1 4 LN Architecture

There are 15 ways of subsequently setting neurons to zero in a four LN architecture. This can trivially be demonstrated: ${}_4C_1 + {}_4C_2 + {}_4C_3 + {}_4C_1 = 4+6+4+1 = 15$. There are four ways of setting one neuron to zero, six ways of setting two neurons to zero and four ways of setting three neurons to zero. Therefore, ideally, one would want to calculate the MSE and energy of the reconstructed flow for all the cases and average over the respective values. Due to time constraints, in Figure 38 below (for a four LN architecture), only two combinations were explored for when one, two and three neurons are "switched off".

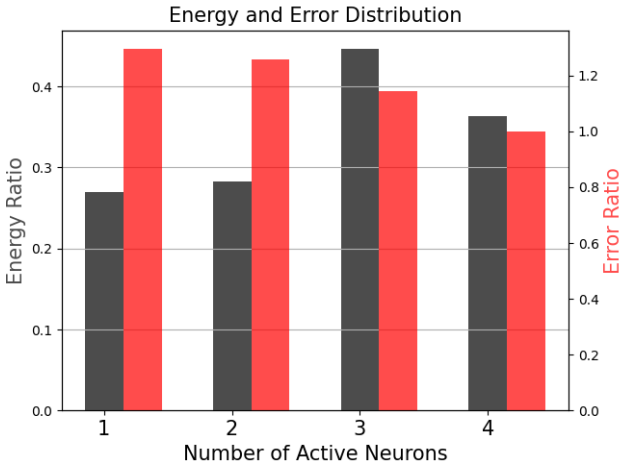


Figure 38: Energy ratio and error ratio versus the number of active neurons for the 4 LNs NN architecture. Error ratio is estimated by evaluating MSE for a given number of activated neurons relative to fully activated latent space MSE. The energy ratio is taken as the outputted TKE divided by the input TKE.

One can see the evolution of the error ratio in red and the energy ratio in black. The behaviour of the energy ratio is different to that of the MSE. One unanticipated result is the overshoot of energy ratio value when three out of four neurons are active. The energetic capacity of NNs with one ”active” neurons dropped to 0.27 and jumped to 0.46 when three active neurons pass the signal forward (compared to 0.37 when all four neurons are ”active”). This trend needs to be reconfirmed by averaging over a complete number of combinations.

Setting specific neurons to zero in the latent space increases the error. With three ”disabled” neurons in the latent space, the error raised by more than 30 per cent. Although the absolute values are not fully accurate, the tendency to diverge from the ”four neurons active” MSE is comprehensible and needs to be reconfirmed taking into account all possible stochastic combinations. Four snapshots in Figure 39 demonstrate how the total MSE ($U^2 + V^2$) changes on the given grid:

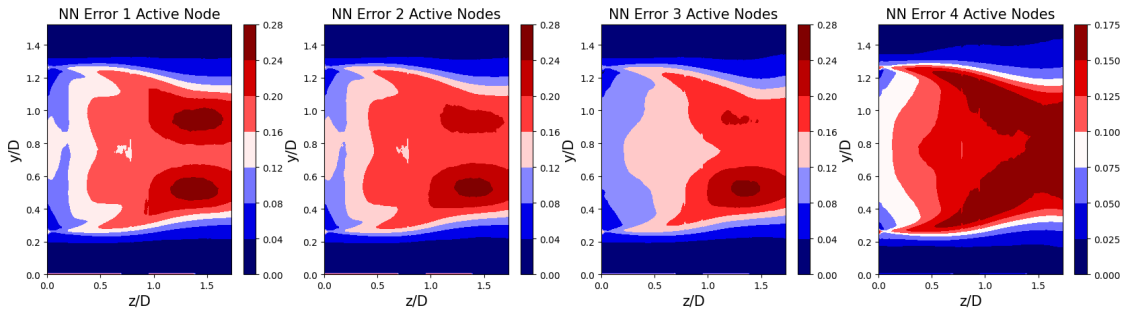


Figure 39: Total MSE for 1, 2, 3, 4 active neurons in the 4 LNs NN architecture.

It is notable from Figure 39 that when the number of active nodes/neurons increases, the region of the largest errors changes. These sketches were produced

in an attempt to understand whether specific neurons can be responsible for certain features. When one out of four neurons is active, two particularly interesting areas at $z/D \in [1.0, 1.5]$ resemble the vortices triggered by the axisymmetric bifurcation mode. As more active neurons are added, this region of uncertainty fades away.

5.4.2 8 LN Architecture

This architecture is more challenging in terms of the available number of combinations for latent neuron disabling. The following number of combinations needs to be calculated: ${}_8C_1 + {}_8C_2 + {}_8C_3 + ... + {}_8C_8 = 8 + 28 + 56 + ... + 8 = 255$. As before, only two cases for each active node case were considered. As a result, the plots below may not be fully representative, yet the trend is similar to the previous architecture.

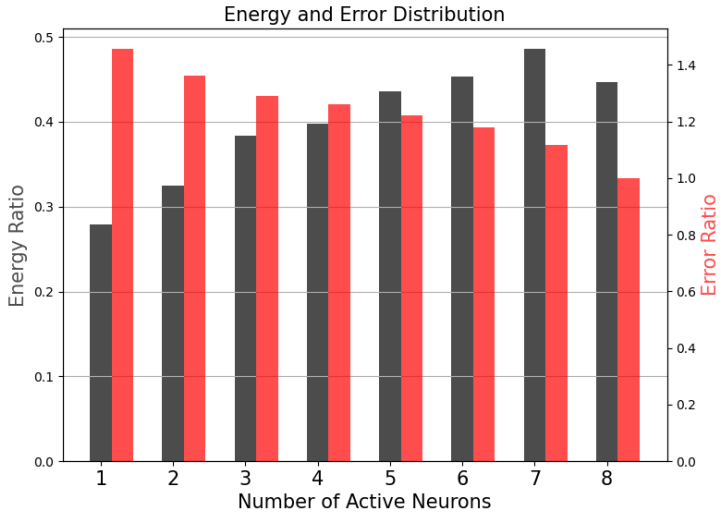


Figure 40: Energy ratio and error ratio versus the number of active neurons for the 8 LNs NN architecture.

For one activated latent neuron, the MSE can increase by 55% relative to the benchmark MSE. The energy ratio has a familiar shape whereby a single active neuron stores 0.28 of the input TKE as opposed to 0.4 when all neurons are activated. The trend may change if averaged over a full number of combinations. Figure 41 presents how the MSE evolves as more neurons are activated.

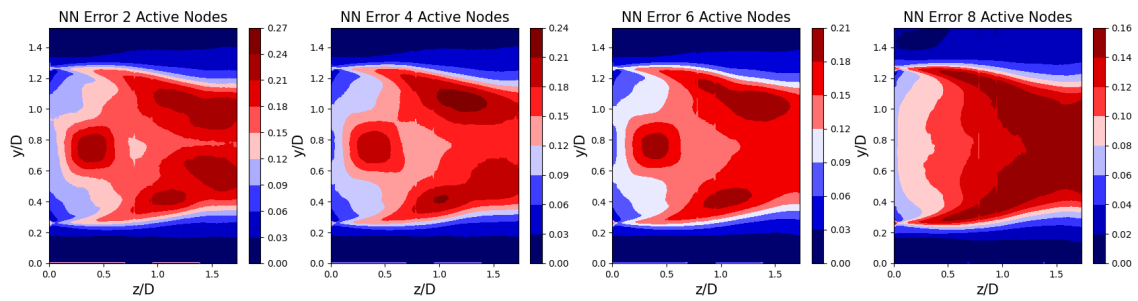


Figure 41: Total MSE for 2, 4, 6, 8 active neurons in the 8 LNs NN architecture.

With two, four and six active nodes, there is an area of mismatch at $z/D \in [0.25, 0.5]$ resembling a vortex. This region disappears when the latent space is fully activated. This error feature may indicate that certain neurons can be responsible for particular coherent structures of the turbulent flow.

6 Conclusion

First, we were able to confirm the spatial POD modes obtained by Arsenov [5] using Python 3.7. The results associated with the energy content of each mode disagreed with previous research and were more accurate in this thesis because the total energy was estimated based on the total kinetic energy (TKE) rather than a partial sum of eigenvalues. As a result, Figure 9 demonstrated lower energy ratio values stored per mode than what was shown in [5].

Next, we were able to investigate the effects of various architecture hyperparameters on the reconstruction MSE. We demonstrated that epochs, batch size and learning rate do not significantly affect the MSE, whereas a Neural Network depth plays a paramount role. As per Figure 18, a larger batch size was shown to shift the MSE vs epochs lines up, hence increasing the reconstruction error. Learning rate values larger than 0.005 were demonstrated to behave unpredictably with an increase in epoch number. MSE oscillated substantially for an LR=0.025, confirming a general rule of thumb that for gradient descent algorithms, LR performs well at around 0.0025. Epochs ranging from 20 to 120 were proven to drive the reconstruction error down with an increase in epochs. Notably, none of these hyperparameters affected the MSE as drastically as the depth of the hidden layers of the encoder and decoder. By fixing the number of neurons and changing a hidden layer with 606 neurons to two 303x303 layers, we were able to decrease the MSE by 90%. Lastly, we found the best hidden-layer architecture that contained four layers with 4848x2424x1212x303 neurons in both encoder and decoder. This autoencoder model performed 256% better in terms of MSE relative to the analogous POD method. Even though this architecture was discovered late into the project, the semi-optimal model with 303x303 neurons in the hidden layer and ReLU activation function, which was used in most of the analysis, demonstrated an 87.2% improvement relative to POD. Hence, we proved that an autoencoder with an adequate architecture outperforms the POD method considerably.

We were able to reconstruct the NN outputs and visually compare them with the original flow. With only eight latent nodes, the directional similarity factor between the original flow and the outputs yielded 0.83 across all 35516 snapshots, comparing with 0.61 in similarity with the POD results. Although the factor does not consider absolute magnitudes as per se, values close to one mean strong agreement in the velocity field direction. Hence, the NN was capable of replicating the flow more accurately.

We then demonstrated that the energy capacity of a given autoencoder is larger for

every increase in latent neuron than the analogous increase in the number of POD modes. An autoencoder with one latent node stores 47% more energy than a single POD mode. The gap between the two method decreases as more neurons/modes are added; 32 latent nodes are only 5.4% better than the analogous POD set-up. This gap shrinkage was only demonstrated for a particular model and compared as a whole latent space without providing insights into individual neuron capacity.

In terms of the temporal analysis, we obtained identical to POD PSD frequency peaks using the latent space variables. Notably, the peaks were slightly different in the case of two latent nodes. However, for latent space with four and eight nodes, the energy peaked at the same location as the analogous temporal expansion coefficients of POD, specifically at $St = 0.221$ and $St = 0.074$. This result is impressive because there physically is no link between the stochastically optimised latent space and the temporal coefficient of POD; the methods are completely different. These findings may indicate that these frequencies are indeed pertinent to the axisymmetric body wake, casting certain doubts aside regarding the applicability of a linear decomposition technique to a highly nonlinear problem.

We introduced the idea of combining POD and NN outputs and demonstrated that several modes could not be reconstructed if the latent space is too small. However, we showcased that one latent node does not imply one correctly reconstructed POD mode, in fact, from a one node reconstruction, we were able to produce at least three POD modes with a high similarity factor and high energy capacity. Similarly, four LN reconstruction could produce at least eight modes accurately. As a result, we were not able to claim whether the direct mapping between the NN outputs and the POD modes exists.

Lastly, we suggested and began exploring the effects of ”disabling” latent neurons. We proposed a framework for the error and energy ratios demonstrating that for a given number of neuron combinations, the error can increase drastically with an increase in deactivated neurons. We also observed that the energy ratio decreases with a decrease in activated neurons. Finally, we drew the reader’s attention to the qualitative discrepancies in error as the number of activated nodes changed, suggesting that certain neurons can be responsible for certain features.

References

[1] Schmitt F.G. Turbulence from 1870 to 1920: The birth of a noun and of a concept. **Comptes Rendus Mécanique**, 450(9):620, 2017.

[2] Cross R. Effects of turbulence on the drag force on a golf ball. **European Journal of Physics**, 37(5):054001, 2016.

[3] Buxton O.R.H. Aircraft aerodynamics ae3-301. 2018-2019.

[4] Lee R. and Moser R.D. Direct numerical simulation of turbulent channel flow up to re 5200. **Journal of Fluid Mechanics**, 774:395–415, 2015.

[5] Arsenov B.R. Pod and spod modal decomposition of turbulent flows. **Imperial College Final Year Project Thesis**, 2020.

[6] Pearson K. On lines and planes of closest fit to systems of points in space. **Philosophical Magazine**, 2(11):559–572, 1901.

[7] Aubrey N. On the hidden beauty of the proper orthogonal decomposition. **Theoretical and Computational Fluid Dynamics**, 2:339–352, 1991.

[8] Peng W., Ma H., and Liu Y. Proper orthogonal decomposition and extended-proper orthogonal decomposition analysis of pressure fluctuations and vortex structures inside a steam turbine control valve. **Eng. Gas Turbines Power**, 141 (4):041035, 2019.

[9] Ostrowski Z., Lecki R.A.B., and Kassab A.J. Advances in application of proper orthogonal decomposition in inverse problems. 2005.

[10] Luo Z. and Chen G. Proper orthogonal decomposition. 2019.

[11] Taira K. and et al. Modal analysis of fluid flows: An overview. **AIAA journal**, 55(12):4020–4023, 2017.

[12] Wang Z., Mcbee B., and Iliescu T. Approximate partitioned method of snapshots for pod. **Journal of Computational and Applied Mathematics**, 307: 374–384, 2016.

[13] Ignatyev D. and Khrabrov A. Experimental study and neural network modeling of aerodynamic characteristics of canard aircraft at high angles of attack. **MDPI**, 2018.

[14] Juna T. and Gang S. An artificial neural network approach for aerodynamic performance retention in airframe noise reduction design of a 3d swept wing model. **Chinese Journal of Aeronautics**, 29(5):1213–1225, 2016.

[15] Jucutan K. Artificial neural network: The brain behind today's smart technology. 2018.

[16] McCulloch W.S. and Pitts W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, 5(1-2):115–133, 1943.

[17] Kolmogorov A.N. On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables. **Doklad Akademii Nauk SSSR**, 108:2(55):179–182, 1956.

[18] Hecht-Nielsen R. Kolmogorov's mapping neural network existence theorem. 1987.

[19] Cybenko G. Approximation by superpositions of a sigmoidal function. **Mathematics of Control, Signals and Systems**, 2:303–314, 1989.

[20] Skorpil V. and Stastny J. Neural networks and back propagation algorithm. 2006.

[21] Wang L. Modal decomposition of fluid flows using autoencoder. **Imperial College Final Year Project Thesis**, 2020.

[22] Brownlee J. Understand the impact of learning rate on neural network performance. 2019.

[23] DeepAI Community. What is an epoch. 2017.

[24] Andrew Ng. Gradient descent optimization stanford lecture notes. 2018.

[25] Agostini L. Exploration and prediction of fluid dynamical systems using autoencoder technology. 2020.

[26] Rigas G., Oxlade A.R., Morgans A.S., and Morrison J.F. Low-dimensional dynamics of a turbulent axisymmetric wake. **Journal of Fluid Mechanics**, 755, 2014.

[27] Solomon O.M. Psd computations using welch's method. **Sandia Report**, 1991.

[28] Gad A. How many hidden layers/neurons to use in artificial neural networks. **Towards Data Science**, 2018.

[29] Shukla L. Designing your neural networks. **Towards Data Science**, 2019.

[30] Welch P. The use of fast fourier transform for the estimation of power spectra: a method based on time averaging over short, modified periodograms. **IEEE Transactions on audio and electroacoustics**, 15(2):70–73, 1967.

[31] Fenny B.F. A complex orthogonal decomposition for wave motion analysis. **Journal of Sound and Vibration**, 310(1-2):77–90, 2008.

A Appendix

A.1 NN Flow Reconstruction Section

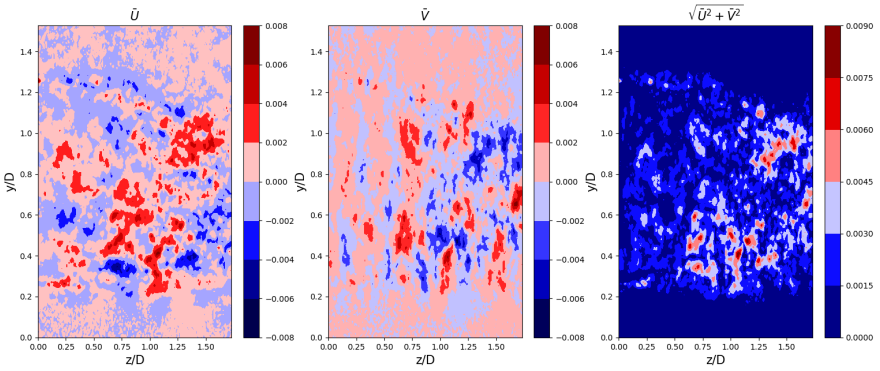


Figure 42: Mean Velocity components of the turbulent wake from the NN outputs.

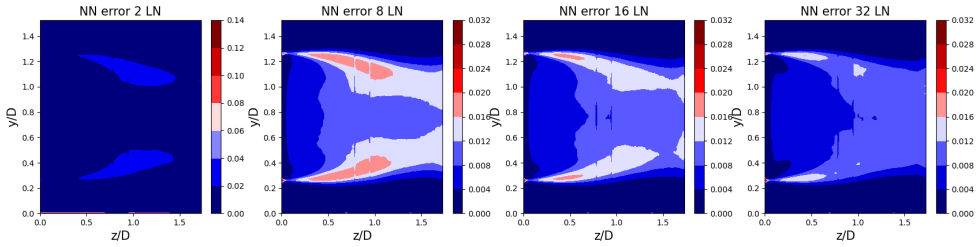


Figure 43: NN $\widetilde{U}$ MSE for 2, 8, 16, 32 LNs.

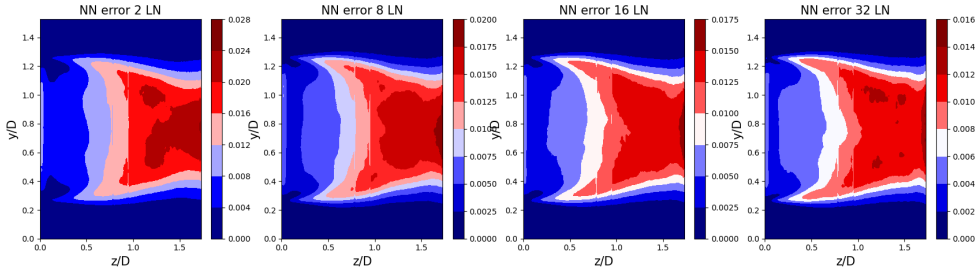


Figure 44: NN $\widetilde{V}$ MSE for 2, 8, 16, 32 LNs.

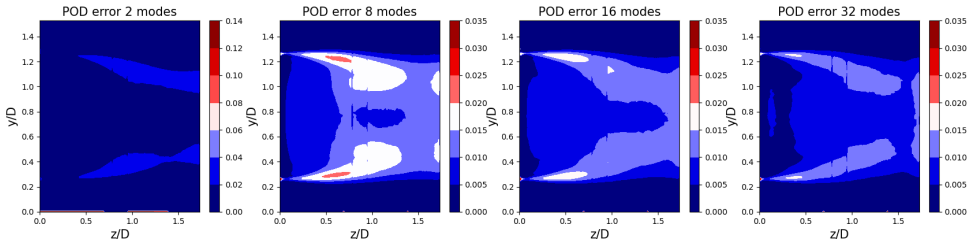


Figure 45: POD $\widetilde{U}$ MSE for 2, 8, 16, 32 modes.

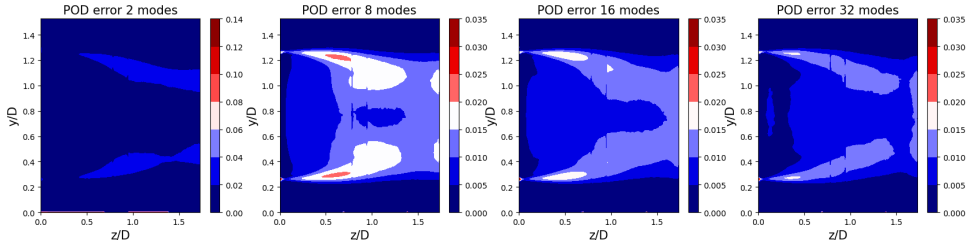


Figure 46: POD $\widetilde{V}$ MSE for 2, 8, 16, 32 modes.

A.2 Temporal Analysis Section

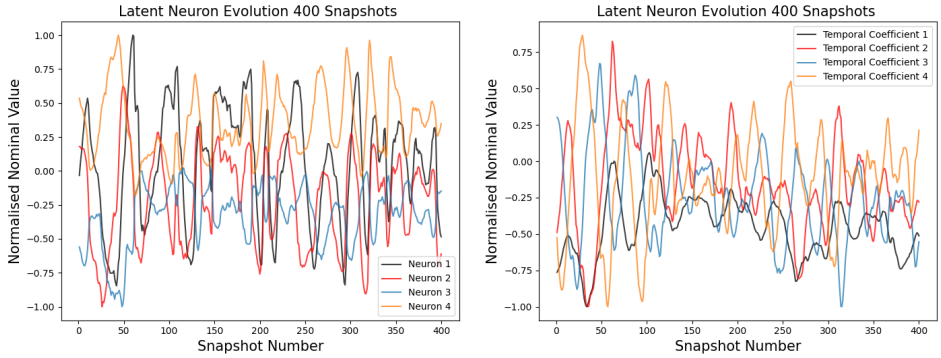


Figure 47: Temporal coefficients and latent neurons evolution for 400 snapshots

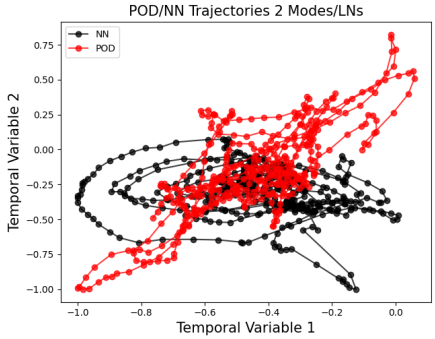


Figure 48: Temporal coefficient trajectories for 2 modes/LNs

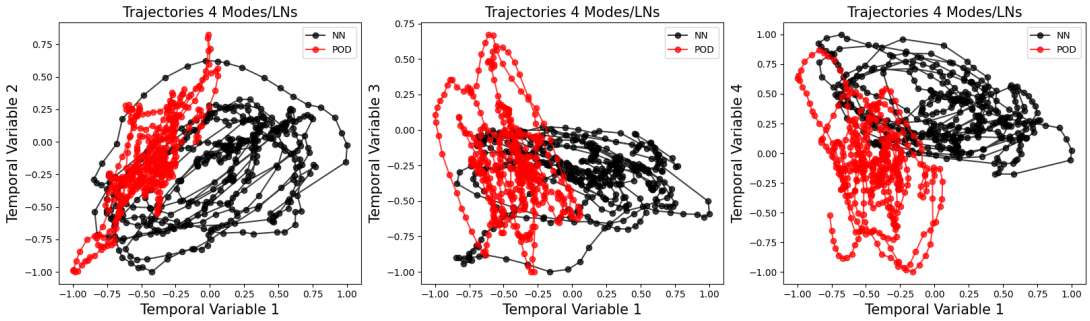


Figure 49: Temporal coefficient trajectories for 4 modes/LNs

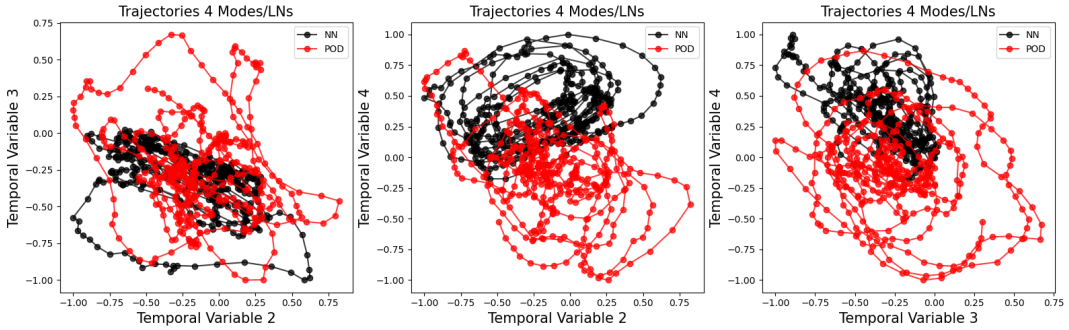
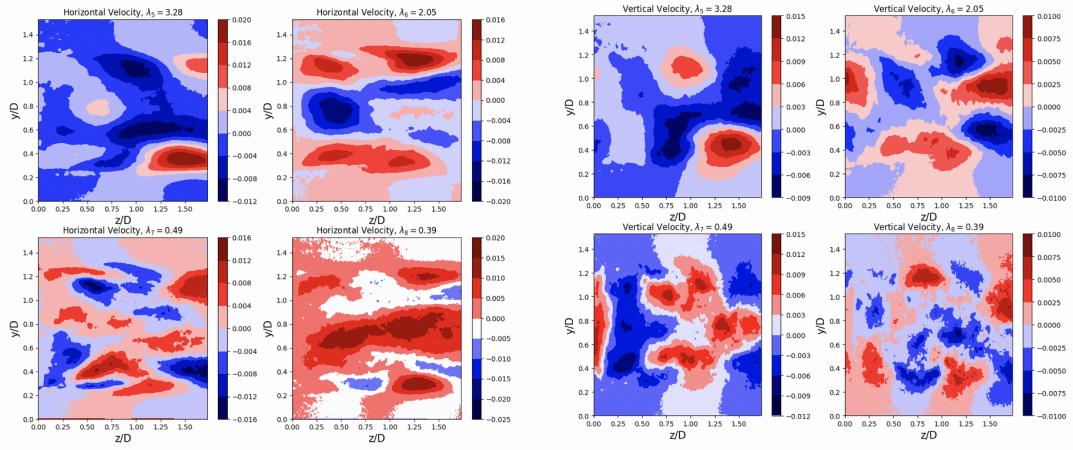


Figure 50: Temporal coefficient trajectories for 4 modes/LNs

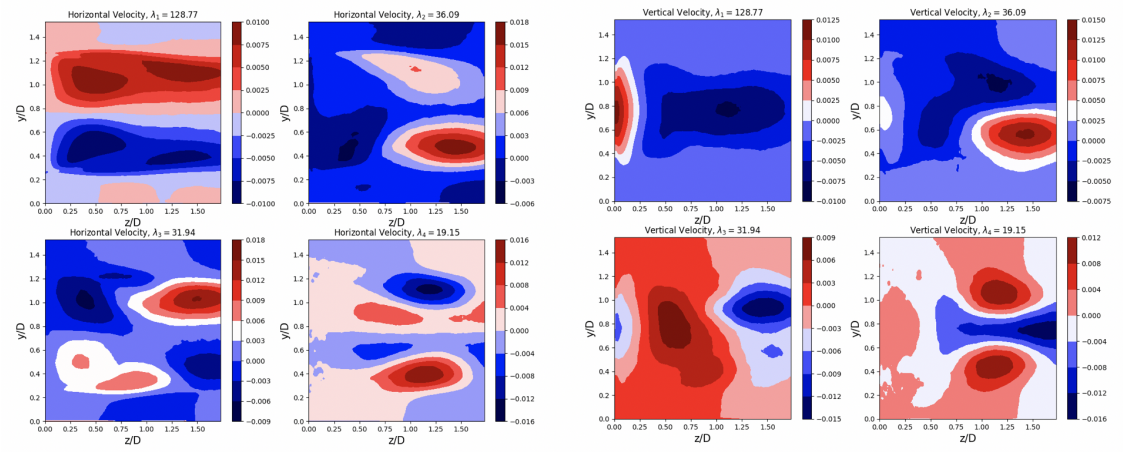
A.3 POD of the NN Output Section



a) Horizontal modes 5-8 obtained from the output of 1 LN NN.

b) Vertical modes 5-8 obtained from the output of 1 LN NN.

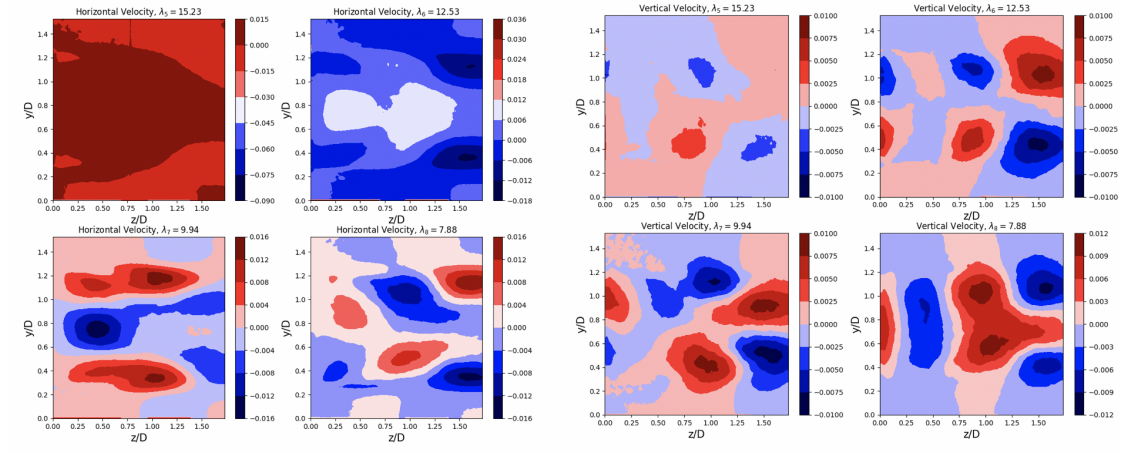
Figure 51: Horizontal and vertical POD modes 5-8 of 1 LN NN reconstruction.



a) Horizontal modes 1-4 obtained from the output of 4 LN NN.

b) Vertical modes 1-4 obtained from the output of 4 LN NN.

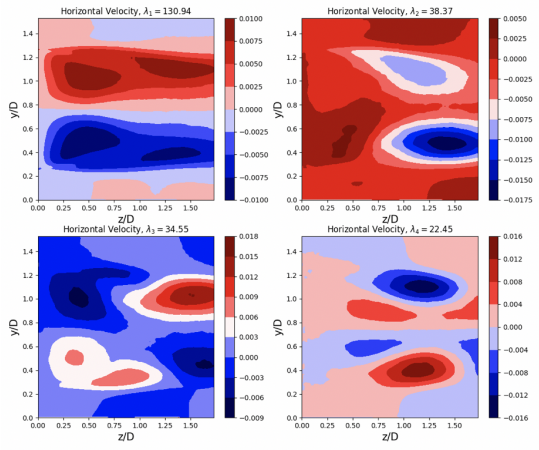
Figure 52: Horizontal and vertical POD modes 1-4 of 4 LN NN reconstruction.



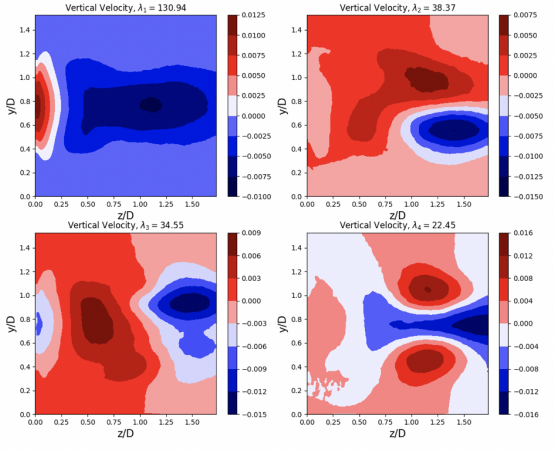
a) Horizontal modes 5-8 obtained from the output of 4 LN NN.

b) Vertical modes 5-8 obtained from the output of 4 LN NN.

Figure 53: Horizontal and vertical POD modes 5-8 of 4 LN NN reconstruction.

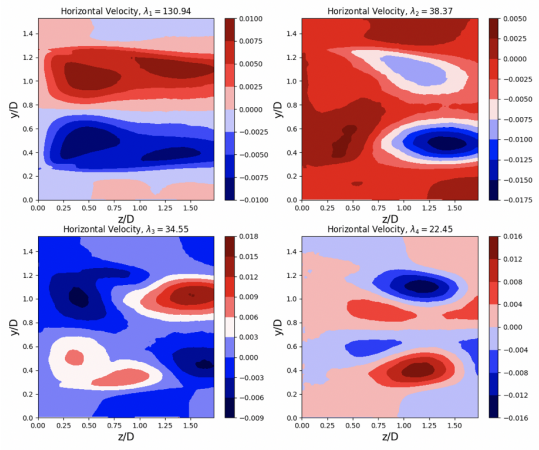


a) Horizontal modes 1-4 obtained from the output of 8 LN NN.

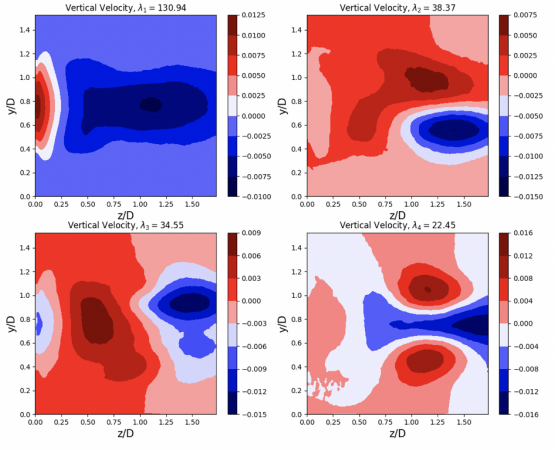


b) Vertical modes 1-4 obtained from the output of 8 LN NN.

Figure 54: Horizontal and vertical POD modes 1-4 of 8 LN NN reconstruction.



a) Horizontal modes 1-4 obtained from the output of 8 LN NN.



b) Vertical modes 1-4 obtained from the output of 8 LN NN.

Figure 55: Horizontal and vertical POD modes 5-8 of 8 LN NN reconstruction.

A.4 Disabling Latent Neurons Section

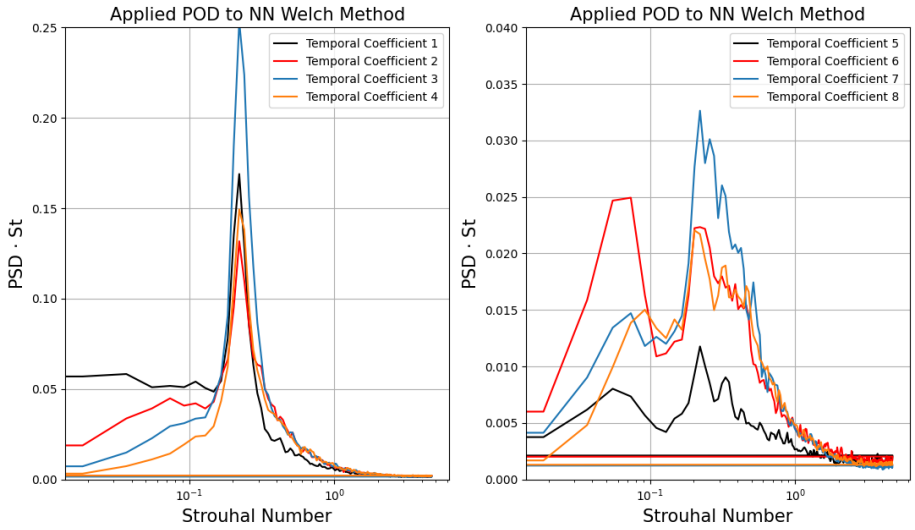


Figure 56: Welch method of POD temporal expansion coefficients applied to 4 LN NN reconstruction.

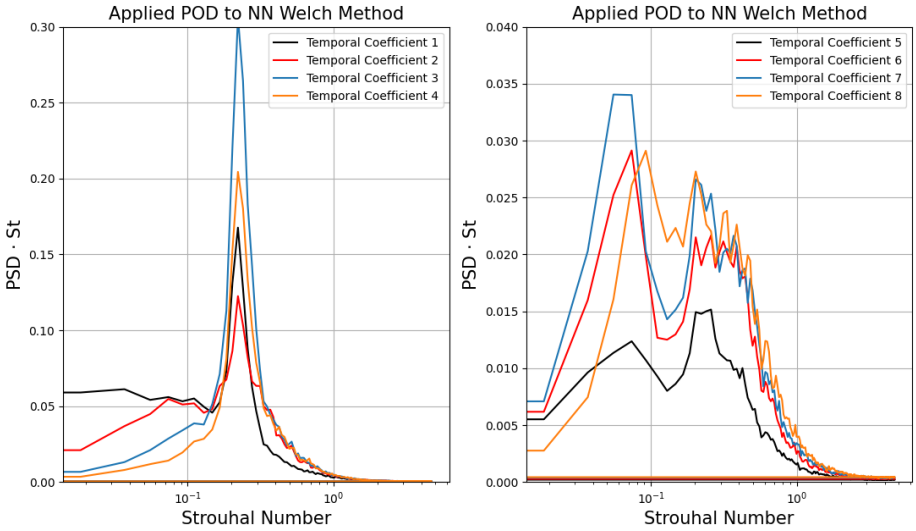


Figure 57: Welch method of POD temporal expansion coefficients applied to 8 LN NN reconstruction.