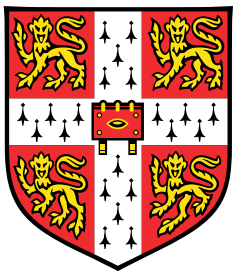
Data-Driven Exploration of Mid-Latitude Weather



I. J. S. Shokar

Supervisors: Prof. P. H. Haynes FRS

Prof. R. R. Kerswell FRS

Centre for Doctoral Training in the Application of Artificial Intelligence to the study of Environmental Risks

University of Cambridge

MRes Research Project Report

Pembroke College July 2021

Declaration

This report is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text and/or bibliography.

I. J. S. Shokar

July 2021

Abstract

In this project we looked at producing data-based reduced order modelling of a fluid dynamical system, that provides an analogue for mid-latitude weather, leveraging the fact that solutions often collapse onto a finite-dimensional manifold. This project used deep learning methods to produce mappings to the lower-dimensional representations of the system and evolve the system in this latent space. Previous research has used such methods on deterministic systems, however this project looked at trying to find this reduced order modelling scheme for a stochastically forced system.

Attempts to make times series predictions of the dynamics opened up questions regarding the time variability of a scholastically forced system. An exploration of how the system responded to changes in initial conditions as well as different realisations of the forcing was conducted to determine the system response when in different states. Here we looked at determining and quantifying the stability of these states in an attempt to understand the fundamental predictability of the system.

Table of contents

Chapter 1

Introduction

1.1 Machine Learning for Climate Science

Climate and weather foresting has developed significantly with the increase in data available [1], however the models of the ocean and atmospheric dynamics, namely Global Circulation Models (GCMs) [2] that are used to produce these predictions are very complex and computationally expensive, resulting in many needing to be run on some of the world's largest supercomputers [3]. Data-driven techniques, such as machine learning provide an excellent opportunity to improve our understanding of these models and hence evaluate their usefulness. Machine learning, broadly describes a array of methods that are well suited to tasks such as pattern recognition, regression, and encapsulating the time-evolution of systems [4]. Deep learning, a branch of machine learning, has many powerful implications, arising from their ability to fit a series of non-linear functions to incoming data in order to capture complex structure [4].

Due to the computational cost of GMCs, not all scales can be simulated, with processes that take place on scales smaller than the spacial resolution of the GCM, as well as fast scale dynamics, having to be approximated, with these approximations known as sub-grid parameterisations. This raises questions over which physical processes are important in maintaining

the fidelity of these models. Machine learning has the potential to be used for predicating future states of weather and climate systems, which can provide both improved forecasting, as well as for the emulation of the physical phenomena that are being parameterised.

High Reynolds number flows such as dynamics in the atmosphere are characterised as turbulent flow and as a result can be described as chaotic systems [5] - a system highly dependent on initial conditions. Subsequently, any incorrect assumptions made during the parameterisation schemes may result in very large errors in predictions. Often, this is accounted for by using an ensemble of predictions, however this simply results in a distribution of incorrect forecasts. Better understanding as well as modelling of these processes may lead to more accurate and reliable forecasts of weather and climate, as well as better quantification of the uncertainty from these models.

1.2 Mid-latitude Idealised Model

Due to the complexity of GCMs, exploring individual components of the model, which will describe the underlying physics, is not possible. As such we will use a simplified 'toy' model, which allows for exploring the fundamental physics of the system in isolation, as well as being able to produce numerical integrations of our model much faster and using orders of magnitude less expensive hardware than that required to run a GCM.

This project considered a simple model, formulated by [6], that provides a useful analogue for week-to-week variations in mid-latitude circulation- the system that describes European weather.

In developing the model a number of simplifications were made, the first of which is that planetary atmospheres and oceans can be considered as shallow fluid layers on a rotating sphere, a consequence of their large horizontal extent compared with their depth. This quasi-two-dimensional nature of planetary scale motions gives rise to dynamics of interest, including the formation of jet streams and cyclones.

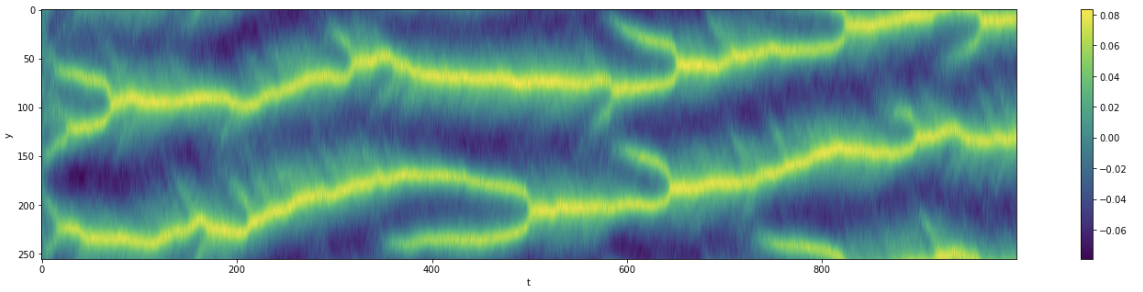


Fig. 1.1 *An example latitude-time plot of the system displaying the values of $\bar{u}$ over y over a period of 1000 observable time steps displaying merging and nucleating behaviour. The colour bar indicates the value of $\bar{u}$.*

This project will focus on the modelling of the atmospheric jets. Understanding how atmospheric jet streams change as concentrations of greenhouse gases increase is very important in understanding implications of climate change for regional weather patterns. Jets, which act like fast-moving rivers, play a fundamental role in the climate system, transporting geophysically-important quantities, such as momentum, heat and tracers, including chemically and thermodynamically important quantities such as ozone and waver vapour [6].

With geophysical dynamics primarily characterised by planetary rotation and stratification, if one is to neglect stratification in the form of a single layer model, the effects of rotation are left to be defined by the Coriolis parameter $f = 2\Omega sin(\theta)$ varies with latitude θ, where Ω is the angular velocity of the rotating body. This can be approximated by a first-order Taylor expansion with a constant latitudinal gradient beta, $\beta = (2\Omega/a)cos(\theta_0)$ close to latitude θ_0 on a rotating body of radius a, reducing the model to a beta-plane approximation [7].

This two-dimensional system displays no intrinsic turbulence due to a lack of dynamical instabilities such as baroclinic instability which inherently posses an eddy-generating mechanism. Turbulence is defined as non-closing bounded orbits in phase space, which cannot exist in a two-dimensional system in the limit $lim_{t\to\infty}$. To generate turbulence in this non-stratified system it must be artificially forced, which is achieved through a stochastic vorticity forcing function, $\xi(\mathbf{x},t)$, which parametersises small-scale eddies. The forcing assumed to be a

random function of both space and time, in which energy is injected homogeneously and isotropically at a constant rate ε.

Energy is predominantly dissipated using linear damping with rate, μ, modelling for example Ekman friction in planetary atmospheres. In order to impose the simplest possible boundary conditions, doubly periodic boundary conditions are imposed on our domain with $(x,y) \in [0, 2\pi L_D]x[0, 2\pi L_D]$ with L_D the length of the domain equal to 256. This leads to the two-dimensional beta-plane vorticity equation being:

$$\frac{\partial \zeta}{\partial t} + J(\psi, \zeta) + \beta \frac{\partial \psi}{\partial x} = \xi - \mu\zeta + \nu_n \nabla^{2n} \zeta \tag{1.1}$$

for a relative vorticity $\zeta = \nabla^2 \psi$, potential vorticity $q = \zeta + f$ velocity field $(u,v) = (\partial_y \psi, \partial_x \psi)$ and J is the determineant of the Jacobian. A more detailed explanation of this can be found in [6].

In this project we only considered the dynamical variability as latitude, y, changes as a function of time, resulting in averaging over x to produce $\bar{u} = \frac{1}{L_D} \sum_{x=0}^{L_D} u_x$, the zonal mean velocity field. This reduces the domain to one-dimention, y, and allows for latitude-time plots to be plotted, an example of which can be seen in figure 1.1.

Within this simple framework, we seek to explore jet stream dynamics and, in particular, variability. Equilibrated jet streams are inherently unstable, and a question arises concerning their time variability. They exhibit a multitude of types of variability, including latitudinal shifts, strength changes and jet mergers [8], [9]. It is anticipated that the stochastic forcing plays a large role in the system variability. By looking at various states of the system we can observe the potential impact the forcing has on the system. In figure 1.1 between $t = 250$ and $t = 300$ the quasi-stable two-jet state can be seen to drift northwards, and this example of a random wandering of the jets is very likely driven by the stochastic forcing analogous to a random walk. An example of a jet nucleation can be seen at $t \approx 370$ at the bottom of the figure, before a merging event can be seen at $t \approx 500$. It cannot be determined how influential

the forcing is on these events, as the change in state could be a consequence of an excitation from one basin of the attractor to another caused by the forcing, however this behaviour also shows similarity to deterministic chaos, with the trajectories in phase space transitioning between unstable solutions [10].

1.3 Manifold Learning

In order to try and learn the dynamics of this system, a method for extracting this information from data is required. With a very number of high degrees of freedom being required to accurately obtain approximate solutions to Partial Differential Equations (PDEs), this is a very computationally expensive task. To overcome this we look to produce a reduced order representation of the system, in which all the key dynamical variables are encapsulated within a latent space encoding of the system. Here we are looking to make an assumption that our system's dynamics lie on an internal manifold embedded in a higher-dimensional space.

Here we propose that state $\mathbf{u}$ lies on some manifold M, $u \in M$, that is embedded in some D dimensional space. We look to find a coordinate mapping onto M to find a corresponding set of coordinates $\mathbf{h}$ that are of D_M dimensions, where $D >> D_M$. This transformation $\mathbf{h} = E(\mathbf{u})$ can be represented using a neural network (NN). In this project undercomplete autoencoder [11], a supervised learning architecture, was chosen as the model to learn this mapping as it acts as an information-filtering bottleneck, processing data down to a lower dimension before mapping back to try and recreate the output despite the bandwidth constraint. This will be discussed in further detail in section 2.1.

There are several methods that are used for dimensional reduction; these include Principle Component Analysis (PCA) and Proper Orthogonal Decomposition (POD). In this project we explored the use of the undercomplete autoencoder [11]. The two aforementioned methods have previously been shown to fail in representing low-order dynamics with a turbulent systems [12]. Given that NNs are combinations of non-linear functions, the autoencoder is

a more appropriate method for dimentionality reduction than those previously mentioned given they are linear, and the system of interest being characterised by a non-linearity.

Other studies, chiefly [13] have looked at extracting governing equations from data. This is very useful for domains, where large volumes are available, which includes the climate sciences, however future predictions made from these generated governing equations can suffer from the same problem as GCMs which due to their complexity, are computationally expensive to evolve in time. The method that is explored in this project, that is encapsulating the dynamics within a neural network structure had a number of benefits over this method as it is able to provide a vast computation speed up when predictions are to be made due to the very fast inference time of neural networks.

Other approaches similar to the method used in this project include using dimentionality reduction to represent turbulent systems include coupling an autoencoder with a Convolutional Neural network (CNN) to find the underlying invariant solutions through a method of latent Fourier analysis [12], as well as using an autoencoder and a corresponding time-evolution network to encapsulate the system on a manifold, exploiting translation symmetry in the representation [14]. Both of these studies looked at the deterministic Kuramoto–Sivashinsky equation on a periodic domain, whereas the system outlined above is stochastically forced, leading to questions over whether the Stochastic Differential Equations (SDEs) that govern the dynamics change this internal manifold and the evolution of dynamics on it.

Chapter 2

Methodology & Results

2.1 Autoencoder

To see if the system we are using lies on a lower dimensional manifold, an undercomplete autoencoder was used for non-linear dimentionality reduction. Autoencoders are a form of supervised learning where during the training process the network learns how to reconstruct the inputs using a neural network that contains a bottleneck layer. This bottleneck reduces the number of variables that can represent the input data, which leads to a learned representation of the data where the information is encapsulated using a smaller number of variables. A schematic of the autoencoder architecture can be seen in figure 2.1.

An autoencoder is made up of two parts, an encoder and a corresponding decoder. The encoder represents a transformation $\mathbf{h} = E(\mathbf{u})$ from out input space to the manifold space, where **u** is our input of dimension D, while the decoder **u'** $= D(\mathbf{h})$ represents the inverse, such that:

$$\mathbf{u'} = D(E(\mathbf{u})) \tag{2.1}$$

where **u'** is the reconstructed output **h** represents our latent space embedding, with the size of this vector, determined by the of neurons in that layer in the mode, corresponding to

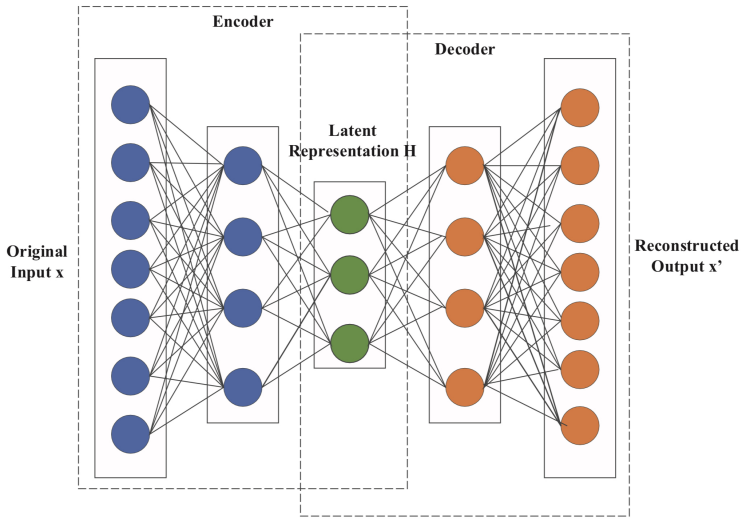


Fig. 2.1 *An example architecture of an autoencoder neural network [15].*

the number of variables that can describe our system in the reduced scheme and thus the dimensionality of this space, D_M.

Network Training

The inputs to the network are a single time-slice of the $\bar{u}$ over y, which is a vector of shape 256. The target outputs of the network are the same as the inputs and the training process attempts to determine these transformation mappings such that the difference between the left-hand-side and the right-hand-side of equation 2.1 is as close to zero as possible. This difference is calculated by a loss function, and the choice here was the Root Mean Squared Error (RMSE) between the LHS and RHS of equation 2.1, defined as:

$$RMSE = \sum_{i=0}^{D} \sqrt{(\bar{u}_i{}' - \bar{u}_i)^2} \tag{2.2}$$

where $\bar{u}_i{}' = D(E(\bar{u}_i))$. Each neuron is connected to every neuron in the following layer, with each connection scaled by a coefficient, or weight, w, and shifted by a bias, b. At each neuron an activation function, σ, is applied. This allows for the stacking of non-linear functions

Layer	Number of Neurons in the Layer	Activation Function
Input	256	Linear
Encoder 1	256	Sigmoid
Encoder 2	256	LeakyReLU
Embedded Layer	5	Linear
Decoder 1	256	LeakyReLU
Decoder 2	256	Sigmoid
Output	256	Linear

Table 2.1 *Network structure for the Autoenocder NN with the embedded layer containing 5 layers.*

where the the output of a neuron, y_k is described by the following summation:

$$y_k = \sigma(\sum_i w_{ki} x_i + b) \tag{2.3}$$

for the k^{th} neuron in a layer where $i = 1, 2, ..., n$ where n is the number of neutrons in the previous layer, and x_i their corresponding outputs,

These comprise the parameters of the network that the training process looks to optimise. Just as with feed-froward neural networks, the training process involves calculating the gradients with respect to the parameters of the network via backpropogation, and using gradient descent to update the parameters with the goal of minimising this error by finding the optimal network parameters.

After exploration, discussed further in the appendix, the network structure of the Autoencoder is displayed in table 2.1. The network was trained using 216000 (256,1)-shaped vectors which took place over 200 epochs, where 20% of the training data was held as validation data, while the other 80% was used for training, leaving 1,000 timesteps for evaluation of the network.

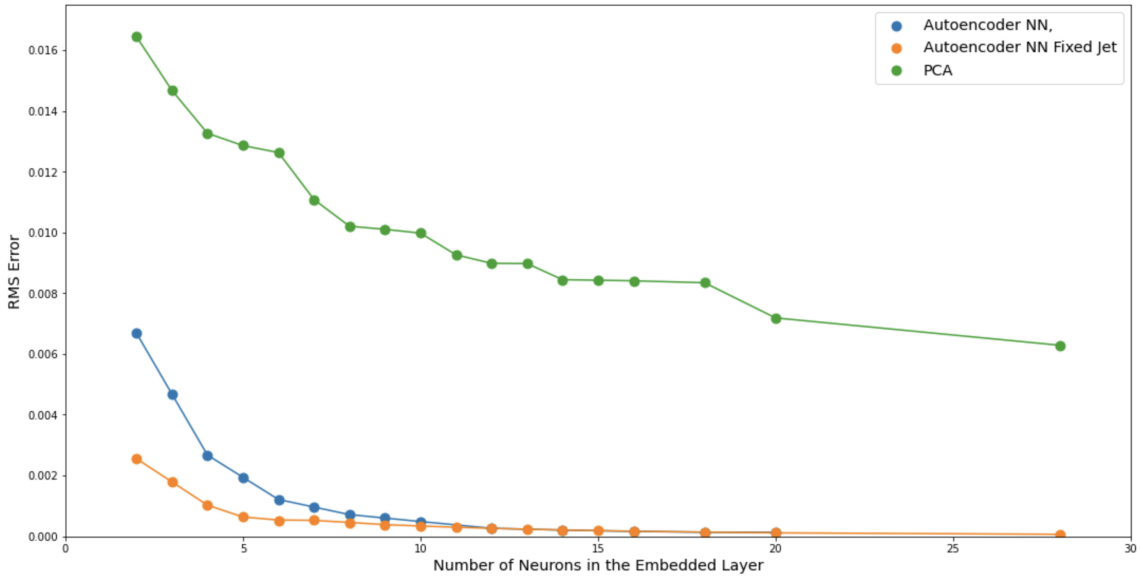


Fig. 2.2 *Plot displaying the RMSE values over a validation set for different dimensions, D_m, of the latent space, determined by the number of neurons in the embedded layer, as a result of training the autoencoder. This was evaluated for when one of the jets was fixed in y (orange), when no shift occurred (blue) and is compared to the results of PCA (green) where the number of basis vectors was the same as the size of the embedded layer.*

Fixing a Jet in Space

As can be seen in figure 1.1 the x-averaged positions of the jets vary in y. While this drifting phenomena is of importance, this study decided to focus on the ability to encapsulate the merging, splitting and relative movement of the jets with respect to one another, as it is expected that the drifting was a result of the random forcing, similar to that of a random walk. As a result, in order to further reduce the degrees of freedom for the autoenocder and to reduce the complexity of the training task, the upper jet of the system was fixed to the same position in y. This is a perfectly valid transformation to make as boundary conditions in y (as well as x) are periodic, and thus this shift corresponds to a shift in the view of the system without altering the system itself, with respect to a fixed time-step. The shift in y is then stored, and reapplied to the output of the network to bring the jet system back to its original position in y. An example figure of this can be found in the appendix, figure A.2.

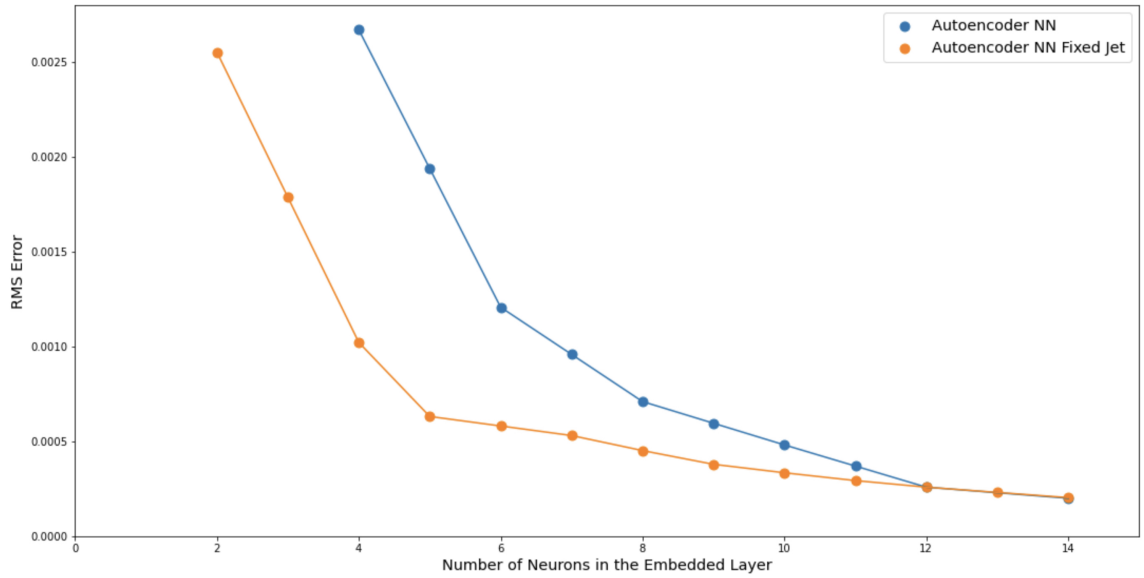


Fig. 2.3 *Plot displaying the same information as figure 2.2, however it shows a subset of the possible D_ms, for clarity, again evaluated for when one of the jets was fixed in y (orange), when no shift occurred (blue).*

Results

Figure 2.2 shows that the mean RMSE over the validation set of data points for different networks, containing a different number of dimensions for the reduced-order model. The figure shows the results for the PCA method as well as the undercomplete autoencoder for the cases where the generated data was unaltered, as well as when one of the jets was fixed in y. It can be seen that both methods perform much better than the PCA method for any number of D_m. Figure 2.3 shows the same data plotted against a subset of the possible D_ms, zoomed in for clarity. In the case where one of the jets was fixed, the network outperforms the case where no shift took place for all D_Ms where D_M>12, to an arbitrary precision. Initially, this indicates that the stochastic process determining the translation of this fixed jet may require an additional six dimensions to be encapsulated fully- however this was not explored further in this study and posses an interesting question for further work.

In the case where one of the jets was fixed, there is a large change in the gradient of the RMSE function at D_M=5, and in the case where no shift took place, this large change

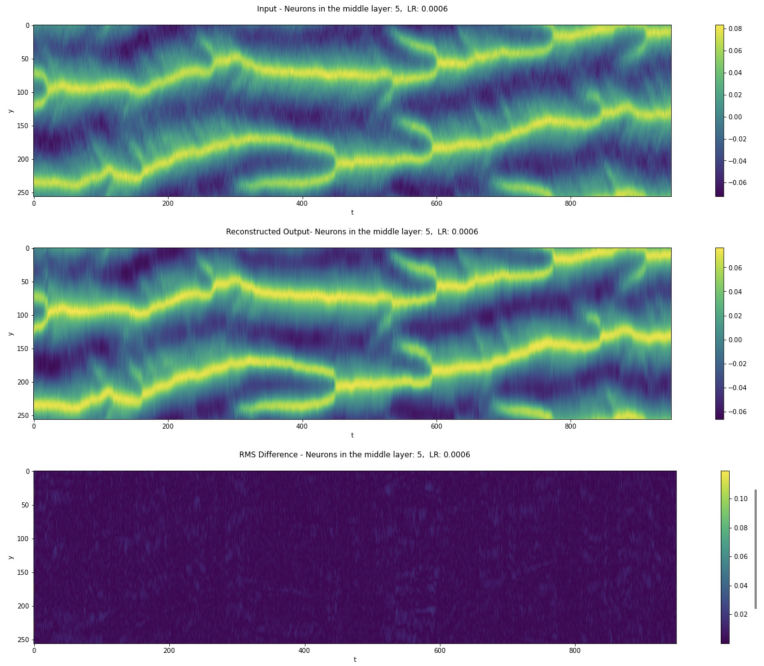


Fig. 2.4 *The top plot shows the input data to the network, over all time steps of this test example, the middle plot shows the reconstructed output from the autoencoder when the number of neurons in the middle layer was 5, corresponding D_M=6 as the network was trained with one of the jets fixed in y. The bottom plot shows the RMSE taken as the difference between the two plots.*

in gradient takes place at D_M=6. Following the conclusions made in [14], this drop on the RMSE coincides with the true D_M for the system. The reason for the increase in this D_M for the case where no-shift occurs is due to an additional piece of information being required for the case where the shift takes place that corresponds to that shift value. This indicates that the true D_M of this system is D_M=6, according to this method. Unlike [14], we do not have the true value of D_M to verify whether this prediction is correct.

In figures 2.4 and 2.5 we see the difference between the original data that is fed to the network for inference and the outputs from the autoencoder, with the jets shifted back to their original position in y, for the cases where the size of the embedded layer are 4 and 5, corresponding to a full D_M=5 and D_M=6 respectively, when factoring the additional shift information. Qualitatively figure 2.5, which used a network with 4 neurons in the embedded layer (D_M=5) is unable to reconstruct the nucleation and merging events as well as figure

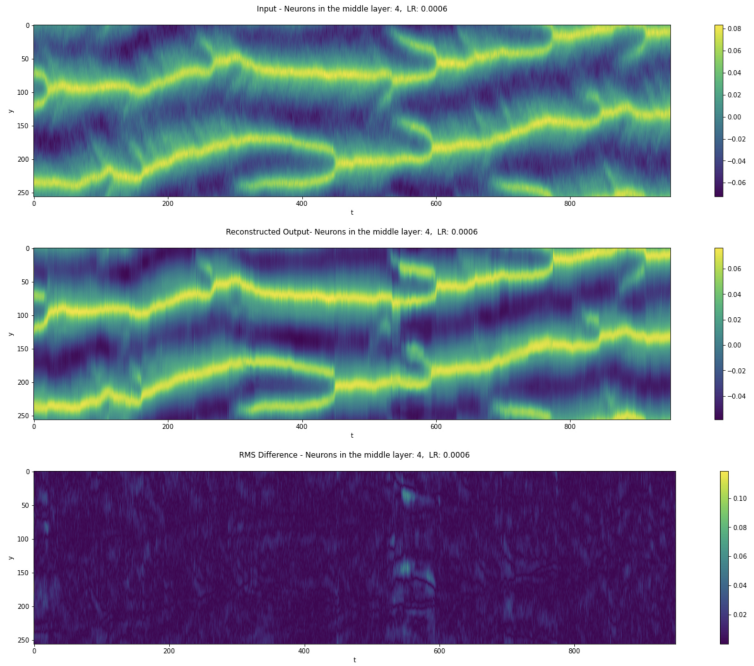


Fig. 2.5 *The top plot shows the input data to the network, over all time steps of this test example, the middle plot shows the reconstructed output from the autoencoder when the number of neurons in the middle layer was 4, corresponding D_M=5 as the network was trained with one of the jets fixed in y. The bottom plot shows the RMSE taken as the difference between the two plots.*

2.4 which used a network with 6 neurons in the embedded layer (D_M=6) and displays a reasonably uniform error over the entire domain. This suggests that, in conjunction with the RMSE curves in figure 2.3, that this the model containing this additional neuron is able to better capture the dynamics of merging events, in particular at $t \approx 550$ a merging event takes place on with both jets, as well as overall an improved modelling of the entire domain, as indicated by the difference plots. It must be noted that the decision to chose which precision of errors are tolerable before considering the best fit with the smallest D_M is very much arbitrary, and this choice is being driven by the results of figure 2.3, however this does show improvements in RMSE as D_M increases and reconstruction plots with higher D_Ms can be found in the appendices. However, this is expected to be expected, as shown in [14], and the additional information being captured by higher D_Ms may, or may not be necessary for determining time evolution of the state on this manifold.

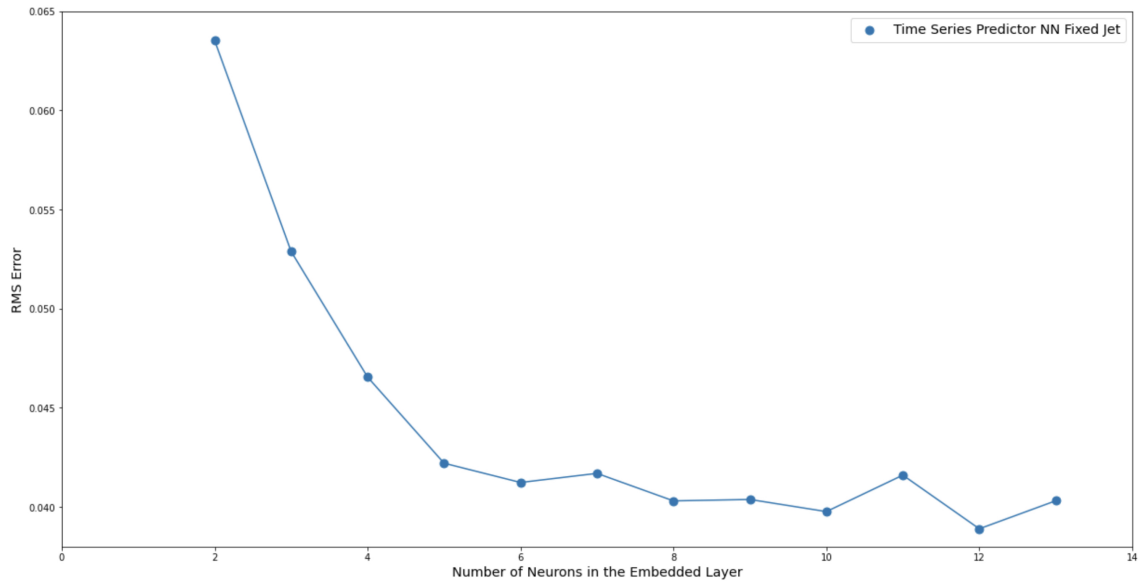


Fig. 2.6 *Plot displaying the RMSE values over a validation set for numbers of neurons in the embedded states of the training data, as a result of training the FFNN to predict the future state of the system. This was evaluated for when one of the jets was fixed.*

2.2 Time Series Evolution

In order to see if the our latent representation of the system was able to capture the dynamics of the system, another NN was used to try and learn the time evolution of the system on this manifold. Initially this was conducted using a simple Feed-Forward Neural Network (FFNN) and a Markovian approach- assuming that all the information required to make a prediction of the state at time $t+1$ is contained within the state at time t, that is that $h(t+1) = F(h(t))$ where F is a function that represents the dynamic evolution of the system. Here the inputs and corresponding targets for the network are the encoded representation of the system, dependent on the chosen size of D_M. For each evaluation all trajectories are evolved from an initial given state of the system, in embedded space this is $h(t=0)$. The network then generates the first prediction for state $h(t=1)$, which then becomes the input for the next prediction. The network was trained using training data which was the embedded state of the system for various D_M, where the one of the jets was again fixed in y, and the RMSE averaged over the validation dataset is show in figure 2.6. As can be seen, a similar change in

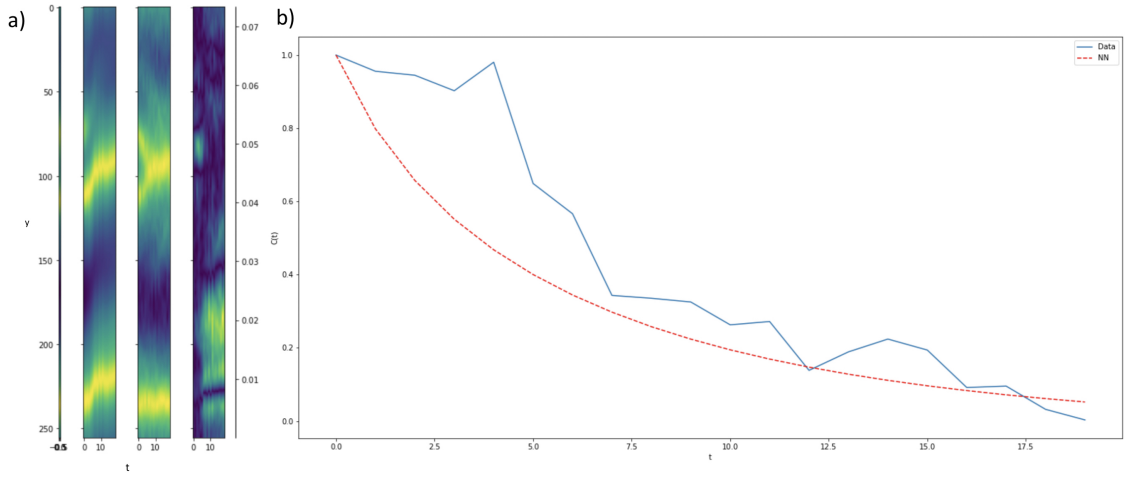


Fig. 2.7 *Plot a) on the left shows the data that the network was shown, centre-left the prediction made by the FFNN predicting the future evolutions of the system. Centre-right shows the truth data, obtained via numerical integration, that the network is being evaluated against. On the right we see the RMSE between the two plots, where yellow indicated a larger difference and purple a small difference. Plot b) Shows the temporal autocorrelation curves for the truth data (blue) and the NN predictions (dashed red). The system predicts 20 time steps ahead from the initial state it was shown. In this plot the system is approaching a merging event*

gradient to figure 2.3 indicates that D_M=6 (5 neurons in the embedded layer plus the shift information) may be the true dimension of the internal manifold of the system, should it lie on one.

While the Lyapunov time of the system is unknown due to the stochastic nature of the system we can evaluate the performance of short-time tracking by calculating the temporal autocorrelation, which can be defined as:

$$C(t) = \frac{\langle u(0)u(t)\rangle}{\langle u(0)^2\rangle}. \tag{2.4}$$

Figure 2.7b shows that the temporal autocorrelations between the data (blue) and the predicted outputs (dashed red) do not match for any amount of time. Similarly the curves corresponding to the data and the predicted outputs differ in shape considerably as well. The corresponding figure 2.7a shows that the system state is near a merging event, while figure 2.8a indicates

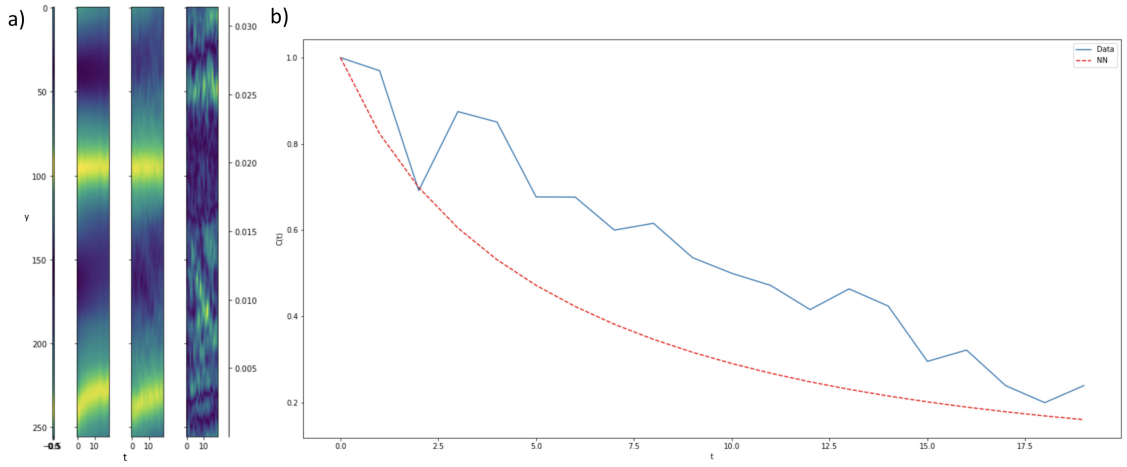


Fig. 2.8 *Plots a) and b) are of the same nature as figure 2.7, however the initial state shown to the network is here a 2-jet system.*

the system is in a 2-jet state, and figure 2.8b shows the temporal autocorrelations between the data and the NN predictions again do not match. The NN predictions at each time step differ from one previous step at near quadratic rate, while the data curves do not display a smooth function, indicating strong influence from the stochastic forcing, even in regimes where the system state does not appear to change. This smoothing from the NN is indicative of a function that is not sufficiently complex to fit to the data- known as underfitting. This can also be seen in figures 2.7a and 2.8a where the difference plots, on the left clearly show that areas of discrepancy between the two plots are often between the two jets, where the NN prediction smooths out these areas. Plots 2.7a and 2.8a show some promise that the network has learned some of the dynamics, indicated by the drift of the bottom jet in 2.8a, however it is clear that a more powerful network is required to encapsulate the complex dynamics of the system.

A more suitable neural network architecture to make predictions on time series data is the Long Short Term Memory Network (LSTM) [16]. A description regarding the nature of LSTM networks can be found in the appendix. The nature of the LSTM allows for a series, or training window, to be shown to the network, rather than the one previous time-step which

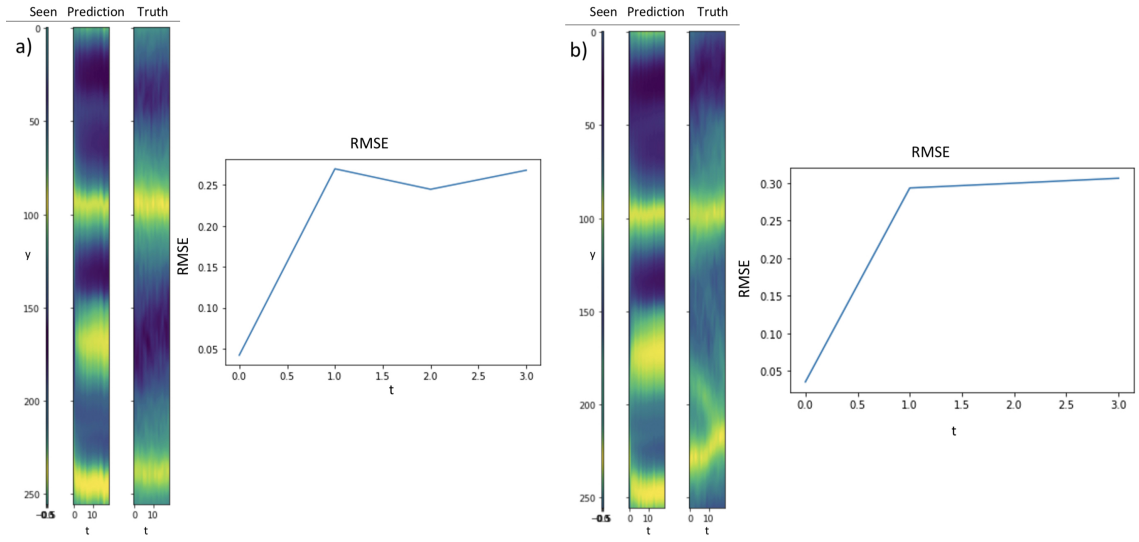


Fig. 2.9 *The plots are of the same nature as figure 2.7a, where in each set of plots on the left shows the data that the network was shown (in this case 1 time step), centre-left the prediction made by the FFNN predicting the future evolutions of the system. Centre-right shows the truth data, obtained via numerical integration, that the network is being evaluated against. On the right we see the RMSE between the two plots averaged for each time step. In this case the network used was the LSTM where one time step was shown to the network. a) Shows a 2-jet system and b) the system is approaching a merging event.*

was used in the previous method. Again the LSTM model was trained on the embedded data where one of the jets was fixed in y. The LSTMs were trained on data where the length of the training data shown to the network were of 1 time step, as previously, as well as 5 time steps.

Figure 2.9 shows the different system states and the LSTM predictions for the network that was trained on a single time step, while figure 2.10 displays this for where the network was trained on a window of 5 time steps of data. Both figures show that the RMS error jumps up immediately to a value of 0.3 after the first time step, before the errors saturate. Contextually this error is an order of magnitude larger than the simple FFNN used for time series prediction, see figure 2.6, and 3 orders of magnitude larger than the error of the autoencoder in figure 2.1. These large errors produce predictions that are sufficiently different to samples from the truth dataset used for training, that the time series predictor is unable to make predictions for subsequent time steps resulting in the prediction of the 3 jet system for

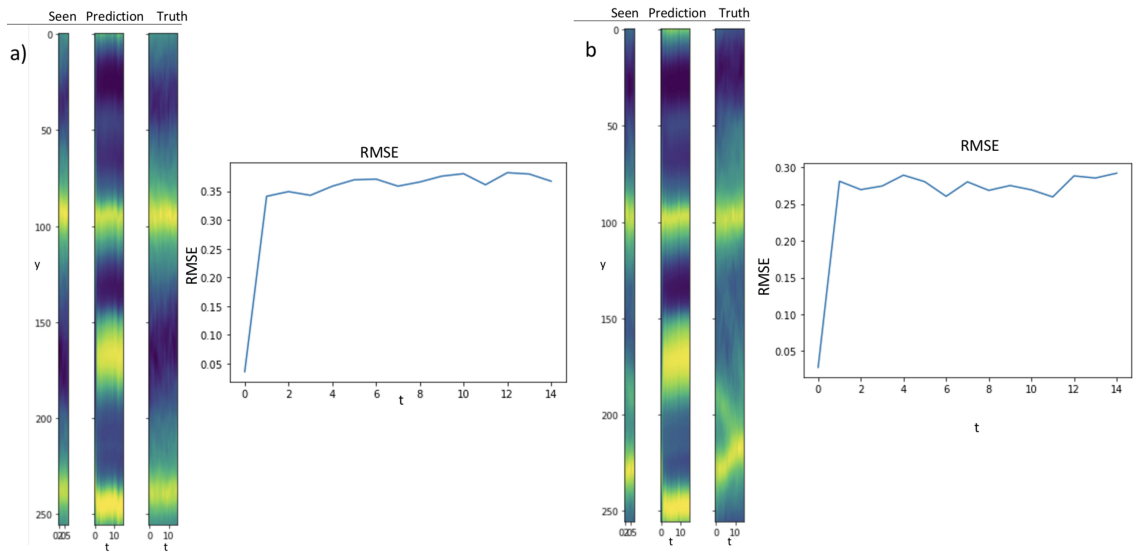


Fig. 2.10 *The plots are of the same in nature as figure 2.9 with the difference being that in this case the LSTM was shown a window of 5 time steps.*

each subsequent prediction, regardless of original state of the system. Another reason for these poor predictions is that the predictions are too different from the distribution of training samples for the decoder network to make accurate mappings. Various different widths and depths of networks, as well as various training window lengths and training data with various dimensions D_M were used to try and improve the training of the network, however nothing proved to be successful. If time permitted, further enquiry into improving this network would have taken place, this is discussed further in the conclusion.

2.3 Predictability

The failure of the LSTM to effectively learn the dynamics of the system, opens up the question of how predictable is this system- that is, how does the system respond to externally induced changes in the conditions at a certain time t. Traditionally this would be done by calculating the Lyapunov exponent of our system, that is rate of separation of infinitesimally close trajectories in phase space [17], to determine the length of time before they diverge.

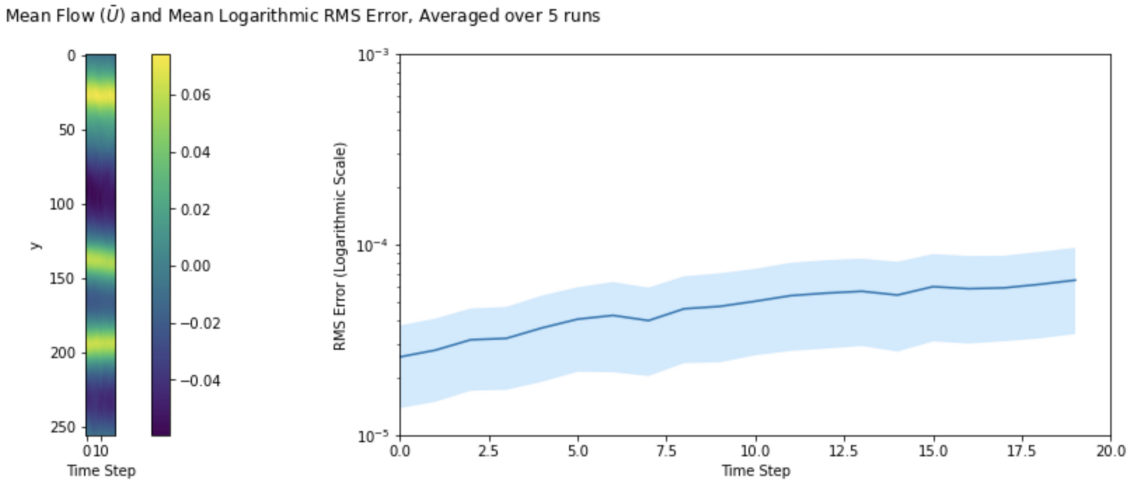


Fig. 2.11 *On the left the plot shows the mean flow averaged over the 5 runs, for 20 time steps. On the right we see the mean difference between runs shown in the solid blue line, with the standard deviation envelope shown in lighter blue. In this case, at time $t = 0$ the runs were given a perturbation; all runs used the same realisation of the forcing, set by using the same random seed.*

However due to the stochastic forcing, this phase space representation to our model is unknown. If looking at two runs with identical realisations of the forcing, one may be able to compare the phase space trajectories of two nearby initial conditions and observe whether the forcing were to push the trajectory into a different basin of the attractor. However, one would not be able to compare such trajectories in phase-space for two different realisations of the forcing, as the forcing very well may dictate the shape of the attractor, producing different dynamical systems, where comparing trajectories would not be meaningful. [18]

As a result of this, understanding how the stochastic forcing influences the system must be conducted in observed coordinate space. Here we use two different systematic changes to alter the system. They are to either perturb the u field at time t, or to change the external forcing at time t, and to observe how the system changes from time $t+1$ onward as a response to either of these two changes.

The first method involves applying a small perturbation to the system while the realisation of the forcing, set by a random seed, remains the same. 5 separate training runs are initialised

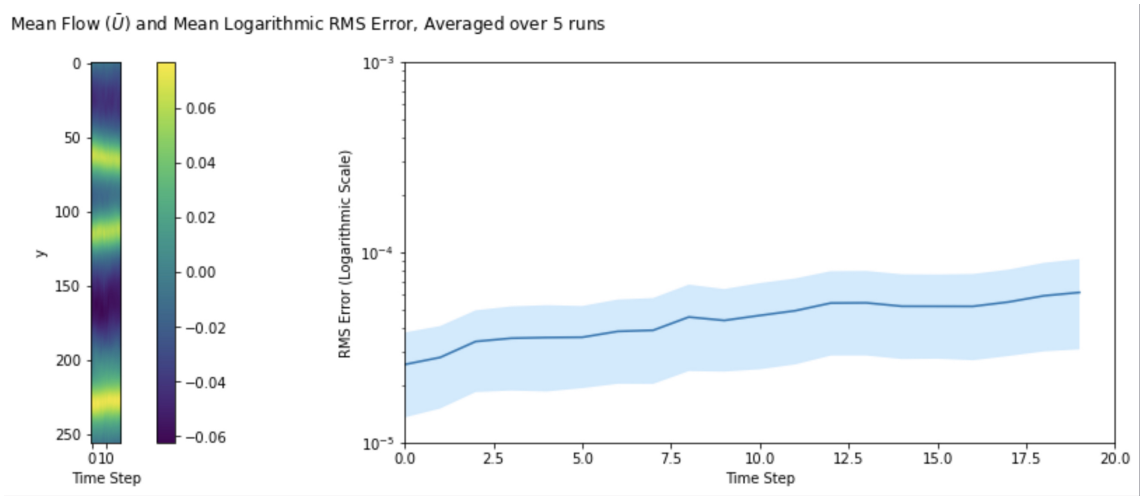


Fig. 2.12 *These plots are of the same nature as figure 2.12, however they were initialised with a different initial state.*

with identical initial conditions and a random seed is used to ensure the forcing is consistent across the runs. At time t a perturbation, of order 10^{-5}, is made to the u field, with each run given a different constant perturbation over all of the *u* field. This order of magnitude was chosen as it was considered sufficiently small to constitute as an appropriate perturbation, but also as this was the order of magnitude of the RMSE from the autoencoder, and would provide a suitable scale to compare the effects of errors on predictability (the error in figure 2.3 shows RMSE of order 10^{-3}, however this was calculated on normalised data, which is of order 10^2 greater in magnitude).

Rather than comparing two trajectories, an ensemble containing five runs results in sum of differences being less sensitive to noise produced by the stochastic forcing. Within our set of five runs, each run was compared to the four others, with the mean of the differences then produced, as well as the standard deviation from the mean. We will describe the divergence of trajectories as the error as these were again computed using the RMSE. These divergence plots can be seen in figure 2.13 where the errors grow exponentially, note the logarithmic axis, which is how a deterministic chaotic system would be expected to evolve, where the divergence would be characterised by a Lyapunov exponent. Here all the runs in the ensemble

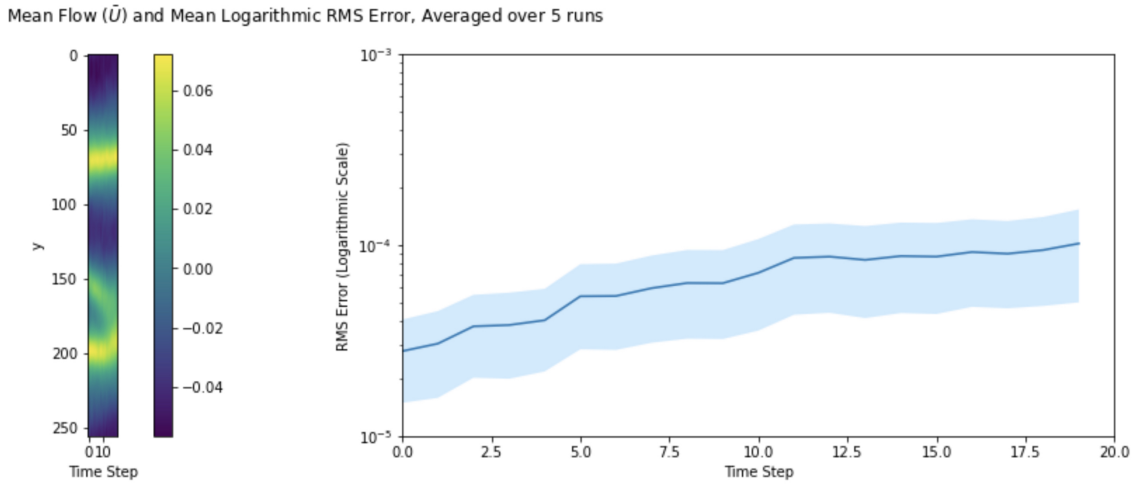


Fig. 2.13 *These plots are of the same nature as figure 2.12 and 2.13, however they were initialised with a different initial state, where a merging event is approaching.*

share the same realisation of the forcing, meaning that this can be neglected when comparing their trajectories, and the comparisons between runs are similar to that of a system without turbulence. Each of figures 2.11, 2.12 2.13, show very similar divergence curves, despite being in very different states that we may define, with figures 2.11 and 2.12 in reasonably stable 3-jet configurations, while figure 2.13 is close to a merging event, again confirming the lack of turbulence without the effects of the forcing.

In order to understand how the forcing on the affects the system, similar to the perturbation of the u-field, a comparison between five runs was made, where the stochastic forcing was reinitialised with a new random seed. Here the error, as can be seen in figures 2.14 and 2.15, the errors grow, with accordance to a square law, immediately after the perturbation is made. This square error is typical of that of Brownian motion. This may explain as to why the neural network could not correctly predict future evolution's accurately- it was unable to correctly model the forcing that drives the system. If given a state of the system, there are several different evolutions that are possible and this is completely dependent on the forcing. The prediction that the model produces, very well may be considered correct under a

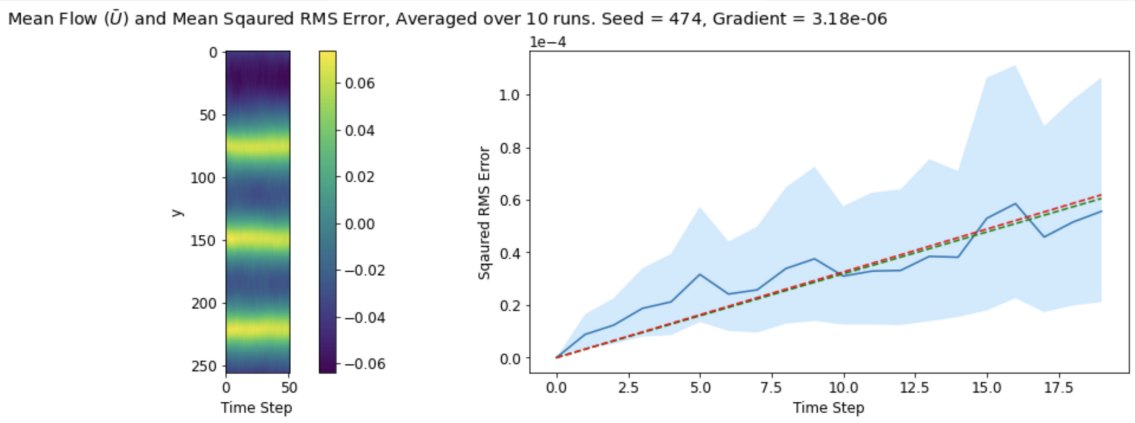


Fig. 2.14 *On the left the plot shows the mean flow averaged over the 5 runs, for 20 time steps. On the right we see the mean difference between runs shown in the solid blue line, with the standard deviation envelope shown in lighter blue. In this case, at time $t = 0$ the runs were initialised with a different random forcing, all other initial conditions remained the same. The green dashed line shows the gradient, determined via regression to represent the diffusion rate, characterised by the diffusion constant. The red dashed line corresponds to the predicted gradient, characterised by the diffusion constant, predicted by the FFNN.*

different realisation of the forcing, however, it does not match the data being used for testing, and thus is shown to perform poorly as a result.

This posses the question of how does the forcing shape the system. Once again there are the two possibilities- that the system distorts the phase-space of the system sufficiently that two different realisations of the forcing produce two different dynamical systems, or that the forcing is significant enough to shift the trajectory to a different regime of the system, an example of this being a significant enough forcing to excite a trajectory out of the well of an attractor. In order to determine this, we looked at whether changing the forcing while the system is in different possible states would change the rate diffusion of trajectories.

One would expect that if the system was in a stable state, such as a long-term three-jet state, as seen in figure 2.14, a perturbation may not be sufficient to shift the system to a new state. If this were the case, it would indicate that the forcing shifted trajectories to different states as opposed to fundamentally shaping the system topology. Following on from this, one would expect a system that was in a more unstable state, that being a jet that is close to a

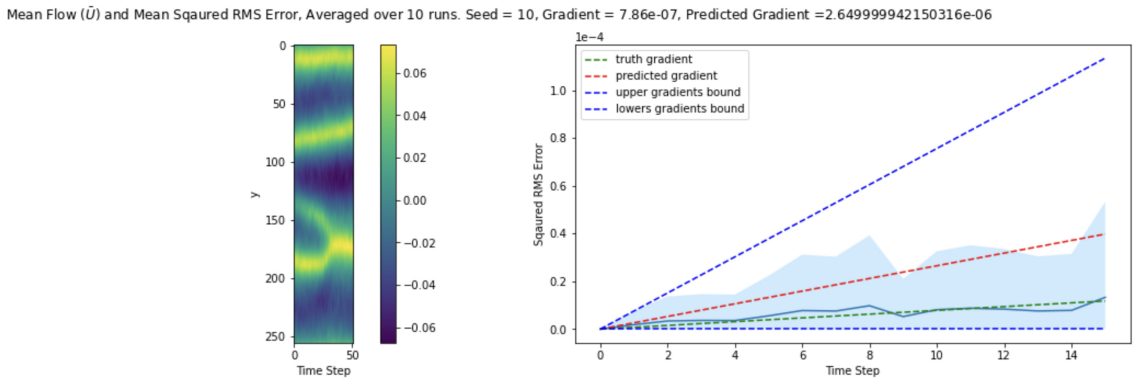


Fig. 2.15 *On the left the plot shows the mean flow averaged over the 5 runs, for 20 time steps, for a diffffernt initial state than that in figure 2.14. On the right we see the mean difference between runs shown in the solid blue line, with the standard deviation envelope shown in lighter blue. In this case, at time $t = 0$ the runs were initialised with a different random forcing, all other initial conditions remained the same. The green dashed line shows the gradient, determined via regression to represent the diffusion rate, characterised by the diffusion constant. The red dashed line corresponds to the predicted gradient, characterised by the diffusion constant, predicted by the FFNN. The blue dashed lines corresponding the the upper and lower bounds of gradients within the training set and are only shown to contextualise the green and red lines.*

merging or nucleating event, such as figure 2.15 where a merging event is about to take place, to be more liable to be significantly influenced by the forcing, and as a result the different trajectories would diverge from each other at a greater rate.

In order to quantify this, the diffusion constant, defined as α, which is the coefficient of the mean exponential divergence curve approximated using a regression, expresses how quickly trajectories diverge from one another and as a result we use this to describe how stable or unstable the system was at that state. In order to see if the value diffusion constant is a result of the system state, a simple NN was trained to perform a regression task of predicting the diffusion constants given the embedded state at time $t = 0$ where this is defined as the time in which the change in forcing was applied. The NN was a simple FFNN, that attempted to predict this value given one time step of the embedded state. The performance of the network, however was poor. In the case where the state of the flow was stable, which is the most common state type, as shown in figure 1.1, the network is able to accurately predict the

diffusion constant, as seen by the matching gradients in figure 2.14. However in 2.15, where a merging event takes place, the network is unable to correctly predict the diffusion constant, and selects a value close to the mean. Once again, if time permitted, further exploration of this would have taken place, both looking to optimise the network training process, but also to probe at the fundamental predictability of this constant, which in turn describes the predictability of the system.

Chapter 3

Conclusion

In this project, an autoencoder was used encapsulate a stochastically forced system using a reduced order model that mapped the system onto an internal manifold. While the attempt to also encapsulate the dynamics of the system using a neural network did not prove to be successful, it did open up a number of questions regarding the variability of a stochastically driven system and questions. These questions include understanding the influence of the forcing on the system as a function the state that the system is in, and whether this function could be represented by a neural network, as well as the question of whether the system displays any predictability, or whether accurate predictions cannot be made due to the stochastic nature of the system.

Going forward there are several areas of this project that can be explored in much greater depth. Further model exploration as well as the use of other novel techniques, such as echo-state networks [19] may prove far more effective at encapsulating the time-evolution of the system that lies on the internal manifold. When looking at the variability of the system, one could probe the underlying structure of the system's attractor to understand more thoroughly how the forcing shapes the topology of this structure. As well as this, further exploration of the regression task, looking to predict the diffusivity of the trajectories when exposed to different realisations of the forcing, may provide very interesting insights into the influence

of the forcing on the system as well as how stable and predictable the system is in certain states. A clustering task could look to try and group certain states based on learned features, using the embedded representation of the model to attempt to uncover information regarding the latent variables that describe the encoded system. This coupled with further work on the interpretability of the encoded layer would lead to a more robust and explainable machine learning system, as opposed to the black box that these systems can often be seen as.

This project looked to provide the grounding for further exploration into the application of data-driven techniques to better understand weather and climate models, as well looking to encapsulate their behaviour to make accurate predictions at a fraction of the integration cost. Should a model achieve this, it could provide parameterisation schemes for many other phenomena that take place on the sub-grid scales, providing, potentially, both more accurate parameterisations, that may lead to more accurate forecasts, as well as reducing the time to run these GCMs- both outcomes that could significantly improve climate science and weather forecasting. This project also looked at probing a system driven by a stochastic forcing, and with many systems using such to represent parameterised phenomena, understanding better how the forcing shapes these systems, would allow for a more accurate description of the governing dynamics of these systems.

References

[1] Hossein Hassani, Xu Huang, and Emmanuel Silva. Big data and climate change. *Big Data and Cognitive Computing*, 3(1), 2019.

[2] MacCracken M. C. Grotch, S. L. The use of general circulation models to predict regional climatic change. *Journal of Climatet*, 4, 1991.

[3] Huize Wang and Robin Wordsworth. Extremely long convergence times in a 3d GCM simulation of the sub-neptune gliese 1214b. *The Astrophysical Journal*, 891(1):7, feb 2020.

[4] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, Jan 2015.

[5] E. N. Lorenz. Deterministic nonperiodic flow. *Journal of Atmospheric Sciences*, 20, 1963.

[6] Cope. L. The dynamics of geophysical and astrophysical turbulence. *Thesis, DAMTP, University of Cambridge. 2021.*

[7] Frank B. Lipps. A note on the beta-plane approximation. *Tellus*, 1964.

[8] Rintoul S. R. Sokolov, S. Multiple jets of the antarctic circumpolar current south of australia. *Journal of Physical Oceanography*, 37, 2007.

[9] Tim Woollings, Abdel Hannachi, and Brian Hoskins. Variability of the north atlantic eddy-driven jet stream. *Quarterly Journal of the Royal Meteorological Society*, 136(649):856–868, 2010.

[10] James M. Hyman and Basil Nicolaenko. The kuramoto-sivashinsky equation: A bridge between pde's and dynamical systems. *Physica D: Nonlinear Phenomena*, 18(1):113–126, 1986.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

[12] Jacob Page, Michael P. Brenner, and Rich R. Kerswell. Revealing the state space of turbulence using machine learning. *Phys. Rev. Fluids*, 6:034402, Mar 2021.

[13] Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[14] Alec J. Linot and Michael D. Graham. Deep learning to discover and predict dynamics on an inertial manifold. *Physical Review E*, 101(6), Jun 2020.

[15] Hieu Mac, Dung Truong, Lam Nguyen, Hoa Nguyen, Hai Anh Tran, and Duc Tran. Detecting attacks on web applications using autoencoder. In *Proceedings of the Ninth International Symposium on Information and Communication Technology*, SoICT 2018, page 416–421, New York, NY, USA, 2018. Association for Computing Machinery.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[17] Alan Wolf, Jack B. Swift, Harry L. Swinney, and John A. Vastano. Determining lyapunov exponents from a time series. *Physica D: Nonlinear Phenomena*, 16(3):285–317, 1985.

[18] Tanguy Laffargue, Julien Tailleur, and Frédéric van Wijland. Lyapunov exponents of stochastic systems—from micro to macro. *Journal of Statistical Mechanics: Theory and Experiment*, 2016(3), Mar 2016.

[19] Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Phys. Rev. Lett.*, 120:024102, Jan 2018.

[20] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[21] The Mathworks, Inc., Natick, Massachusetts. *MATLAB version 9.3.0.713579 (R2017b)*, 2017.

[22] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms, 2012.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[25] Sajid A. Marhon, Christopher J. F. Cameron, and Stefan C. Kremer. *Recurrent Neural Networks*, pages 29–65. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

Appendix A

A.1 Autoencoder

The following information was not included in the main body of the text as it was considered information that did no hinder one's ability to interpret the results of the study, however they may be useful for those looking to understand the structure of the models used.

All of the Deep Learning models were built using the tensor flow framework [20]. The training data was generated using MATLAB [21], using a set of scripts written by Laura Cope for their thesis [6].

The hyperparameters of a NN are those that describe the structure of the network, these include the activation functions at each layer, the number of neurons in each layer and the gradient descent learning rate. The hyperparameters in model were chosen by the method of Bayesian Hyperparameter Optimisation [22]. As the encoder and the decoder are inverse mappings of each other, the hyperparameter search was constrained to so that the encoder and decoder structures mirrored each other. The method of Bayesian Hyperparameter Optimisation was chosen due to its performance over other methods such as Grid Search or Random search algorithms. The goal of Hyperparameter optimisation is to find the optimal model structure that performs best when applied to a validation set, with the best performance being classified as the lowest value of the objective loss function, in this case that is the RMSE. Due to the computational cost of training, determining the best though searching

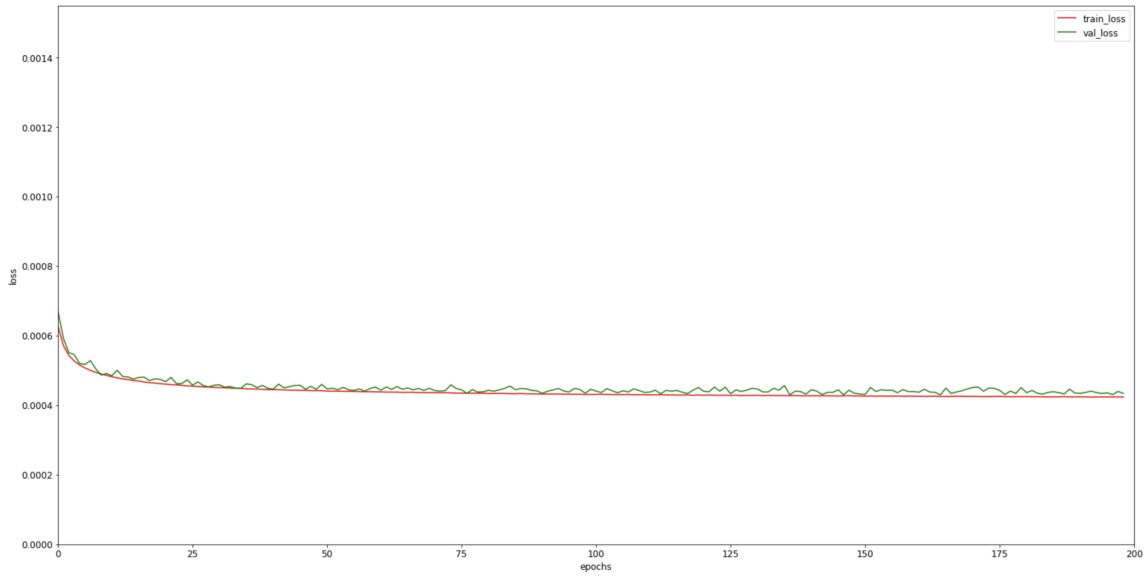


Fig. A.1 *Plot displaying training loss (red) and the validation loss(green) where this loss is RMSE for the training of the Autoencoder with an embedding layer of size 5, and one of the jets was fixed in y to corresponding to* D_M*=6.*

is inefficient often impractical. The optimiser used was the Adam [23], which is uses an adaptive learning rate to converge on the optimal values, with the defined learning rate being the initial rate before the Adam optimiser determines the optimal learning rate for the position on the gradient surface.

The exception to this was the depth of the network, the choice for this was made by completing this process for each of the various options: 3, 5 and 7 hidden layers. Significant improvements in the RMSE were found when increasing the depth of the network from 3 hidden layers to 5, however that there is no significant reduction in RMSE when increasing the depth of the network from 5 to 7 hidden layers. The smaller network of 7 layers was chosen due to it's faster training time due to a smaller number of trainable parameters. The number of trainable parameters for this network is 200,197 when the embedded layer is of size 6.

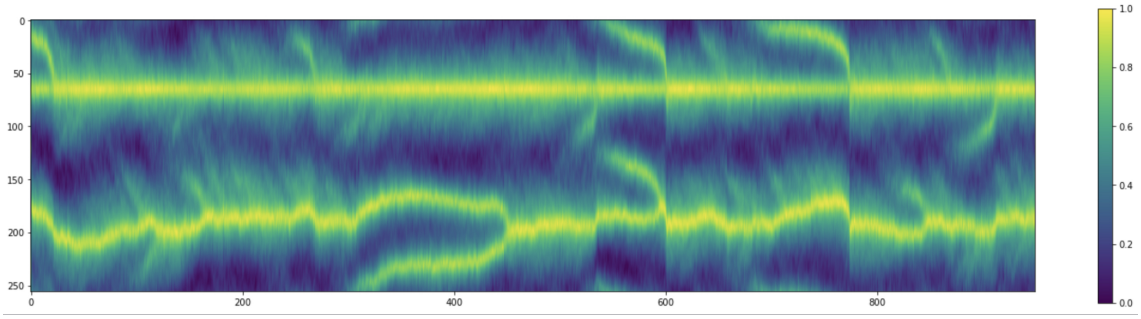


Fig. A.2 *Figure showing the time-latitude plot of the system where one jet has been fixed in y, using the periodic boundary conditions of the domain to do so by shifting the jet in with respect to its postion in the previous time step.*

A.2 Feed Forward Neural Network for Time Evolution

The network used to try and learn the dynamics on the internal manifold is a Feed Forward Neural Network (FFNN). Bayesian Hyperparameter Optimisation was once again used to determine the optimal number of neurons in each layer, the number of layers, the activation functions and the learning rate, the values of which are in table 2.1. In order to prevent outfitting from taking place, a dropout method was applied to each layer [24]. This involves ignoring a random sub-sample of the neurons during the training pass, in this case the proportion of neurons turned off in each layer was 40%. The reason for this is to prevent the network from learning predominantly along one particular path in the network, and as a result performing poorly on unseen data. the optimal parameters, as determined by hyper parameter optimisation, are shown in table A.2The number of trainable parameters for this network is 17,349.

A.3 LSTM

LSTMs are a form of recurrent neural network (RNN) [25], these are a group of architectures, that are able to use outputs from previous states as the inputs for the proceeding state, and thus are suitable to data in sequential form and thus time-series data. Used widely for tasks

Layer	Number of Neurons in the Layer	Activation Function
Input	5	Linear
Dense Layer 1 (Dropout)	64 (0.4)	Tanh
Dense Layer 2 (Dropout)	64 (0.4)	Tanh
Dense Layer 3 (Dropout)	64 (0.4)	Tanh
Dense Layer 4 (Dropout)	64 (0.4)	Tanh
Dense Layer 5 (Dropout)	64 (0.4)	Tanh
Dense Layer 6 (Dropout)	64 (0.4)	Tanh
Output	5	Sigmoid

Table A.1 *Network structure for the FFNN for time evolution prediction with a an input containing the initial state embedded into a vector of size 5.*

such as speech recognition, the powerful feature that RNNs posses to retain some contextual information when looking to make a prediction on an input. By storing information about the previous states in the time series, the network is able to make an informed prediction with respect to the most current input from the series. They can be distinguished from traditional feed-forward networks (FFNN) by their inclusion of feedback loops which provide the next computation to be made with the outputs from previous states, giving the network a form of memory, or context in which to make a decision. If one was to unfold the network to visualise the flow of data, these loops would represent the connections between each cell of the network that repeats itself. Each cell receives the corresponding input vector and information from previous cells.

The weights of the network are still evaluated through the process of back propagation. The LSTM is a popular variant of RNNs, in which each cell of the LSTM network containing state cells and gating mechanisms to regulate the flow of information, retaining or forget information from these past outputs. Here the training data can be fed to the network in the form of a sequence, as opposed to a single vector which was the case using the previous networks. The structure of the model con trained two LSTM cells as was determined via hyperparameter optimisation; the first with 128 hidden states and the second 64 hidden states.

Layer	Number of Neurons in the Layer	Activation Function
Input	5	Linear
Dense Layer 1 (Dropout)	64 (0.4)	ReLU
Dense Layer 2 (Dropout)	64 (0.4)	ReLU
Dense Layer 3 (Dropout)	64 (0.4)	ReLU
Dense Layer 4 (Dropout)	64 (0.4)	ReLU
Dense Layer 5 (Dropout)	64 (0.4)	ReLU
Dense Layer 6 (Dropout)	64 (0.4)	ReLU
Dense Layer 7 (Dropout)	64 (0.4)	ReLU
Output	5	Sigmoid

Table A.2 *Network structure for the FFNN for predicting the diffusion rate with a an input containing the initial state embedded into a vector of size 5.*

The number of trainable parameters in the network when the embedded inputs are of shape 5 is 51,269.

A.4 Diffusion Prediction Neural Network

The network used for predicting the diffusion constant, when different realisations of the forcing were used on runs with identical initial conditions was also a simple feed forward network. Following hyper parameter optimisation, the optimal structure of the network was determined as can be seen in A.2. The number of trainable parameters in the network is 17,195.

As was seen in the main body of the report, the model was not successful when looking to predict these values. When using a different dimension for the D_M of the training data, it can be seen in figure A.6 that the training performance was not affected by the size of the training data vectors. This is unusual and not expected and further work would look to understand if this is caused by a flaw in the network, or a fundamental lack of predictability regarding this diffusion constant.

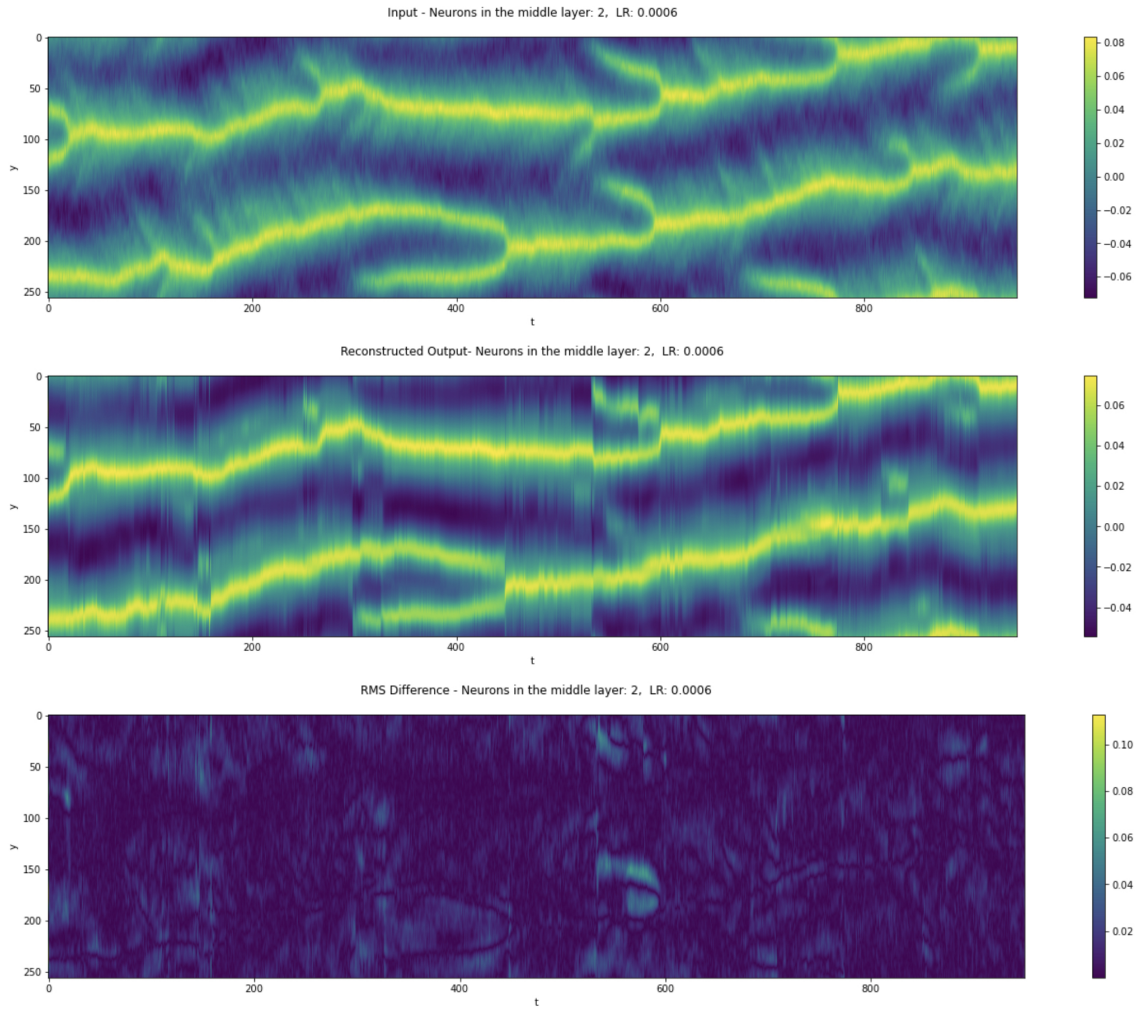


Fig. A.3 *The top plot shows the input data to the network, over all time steps of this test example, the middle plot shows the reconstructed output from the autoencoder when the number of neurons in the middle layer was 2, corresponding D_M=3 as the network was trained with one of the jets fixed in y. The bottom plot shows the RMSE taken as the difference between the two plots.*

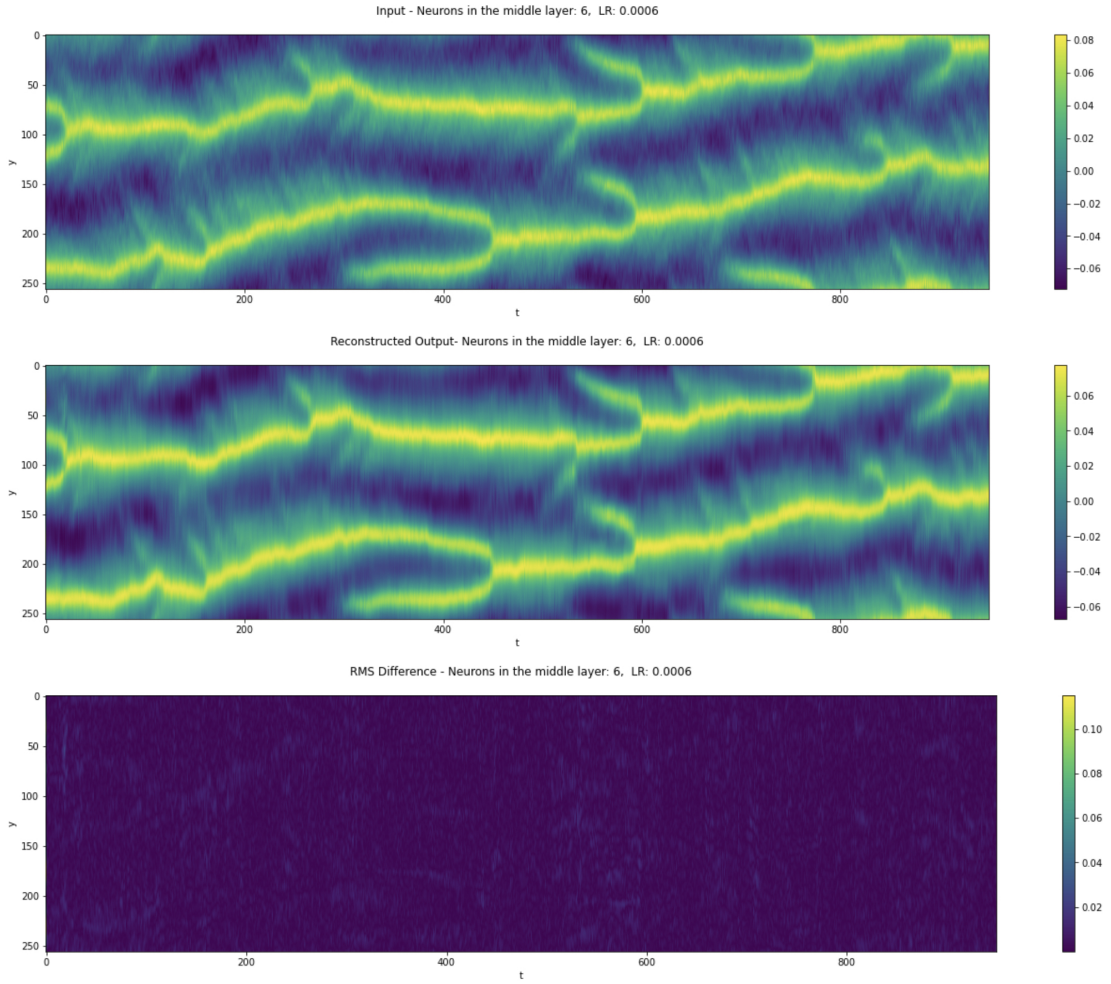


Fig. A.4 *The top plot shows the input data to the network, over all time steps of this test example, the middle plot shows the reconstructed output from the autoencoder when the number of neurons in the middle layer was 6, corresponding D_M=7 as the network was trained with one of the jets fixed in y. The bottom plot shows the RMSE taken as the difference between the two plots.*

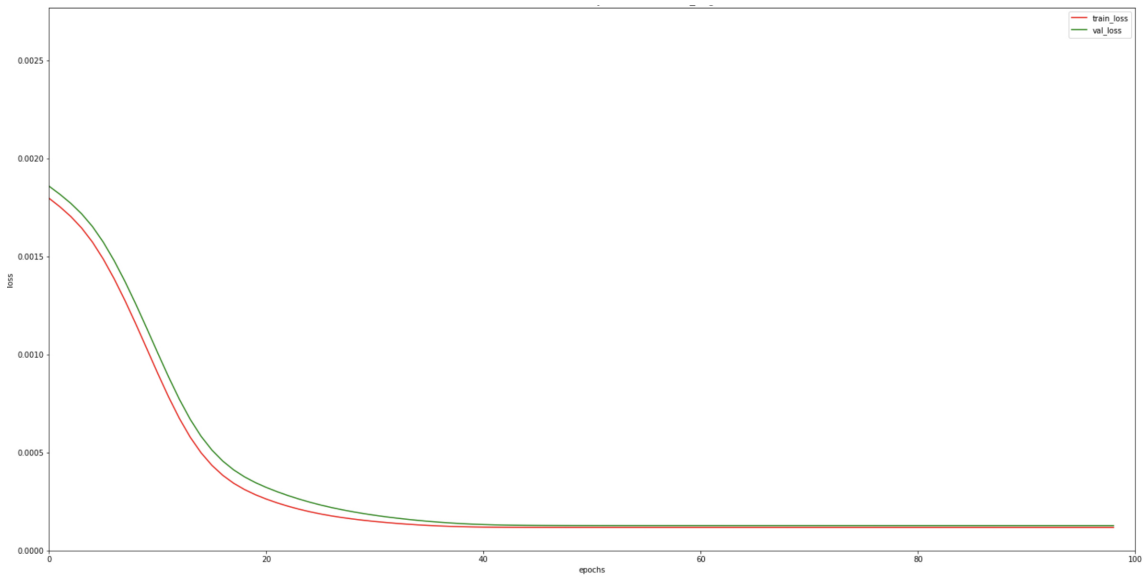


Fig. A.5 *Plot training loss (red) and the validation loss(green) where this loss is RMSE where the LSTM which was shown a window of 5 time steps, with the training data being embedded with size 5, and one of the jets was fixed in y to correpsonding to* D_M*=6.*

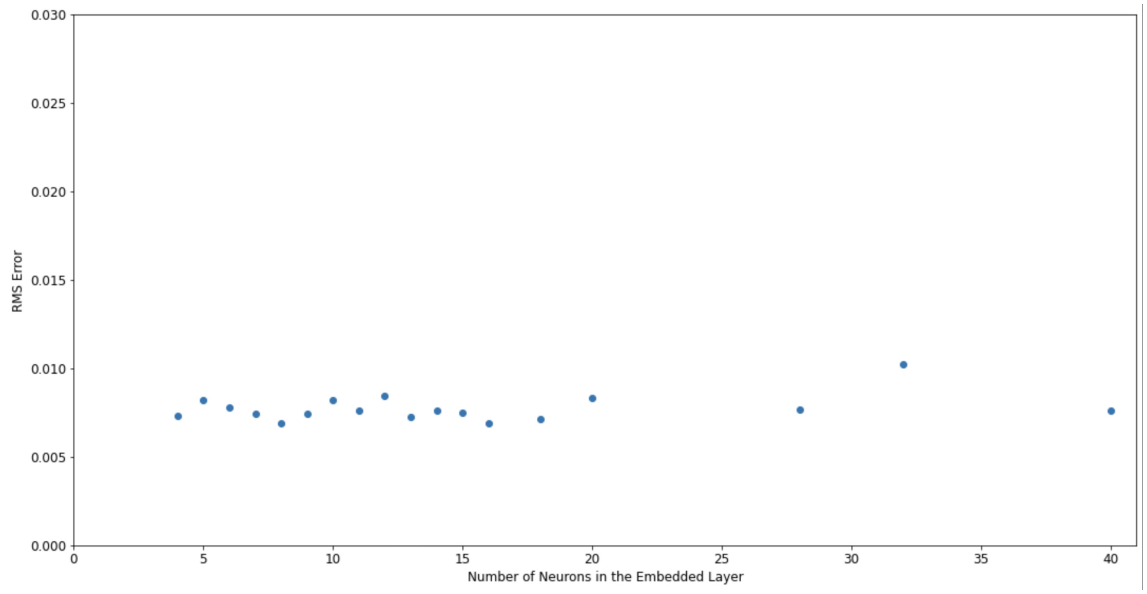


Fig. A.6 *Plot displaying the RMSE values over a validation set for different dimensions,* D_m*, of the latent space, determined by the number of neurons in the embedded layer, as a result of training the FFNN to predict the diffusion rate, where the forcing was altered between runs. This was evaluated for when one of the jets was fixed in y.*

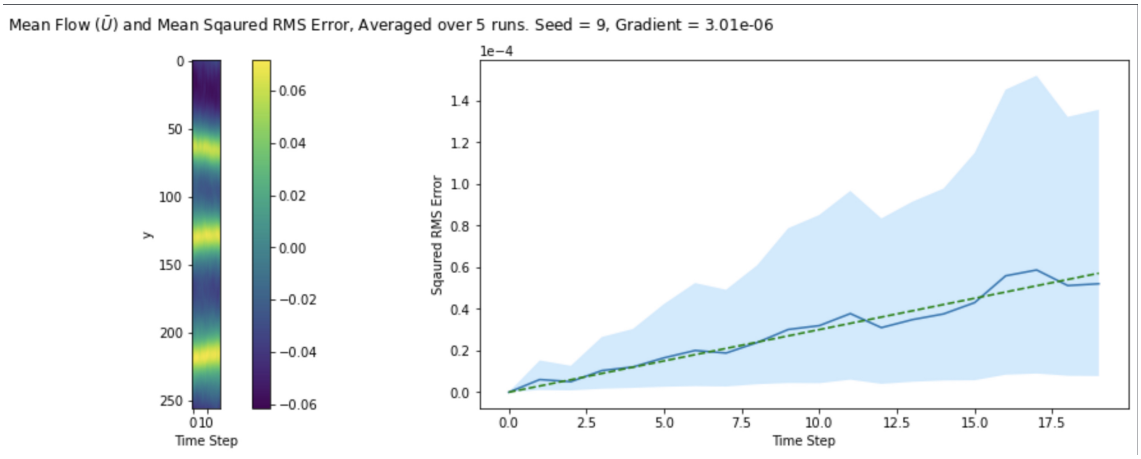


Fig. A.7 *On the left the plot shows the mean flow averaged over the 5 runs, for 20 time steps. On the right we see the mean difference between runs shown in the solid blue line, with the standard deviation envelope shown in lighter blue. In this case, at time $t = 0$ the runs were initialised with a different random forcing, all other initial conditions remained the same. The green dashed line shows the gradient, determined via regression to represent the diffusion rate, characterised by the diffusion constant. The different runs that make up the averages in this figure have been set off with different realisations of the forcing. The initial state for these runs is that of a long-run stable 2-jet.*

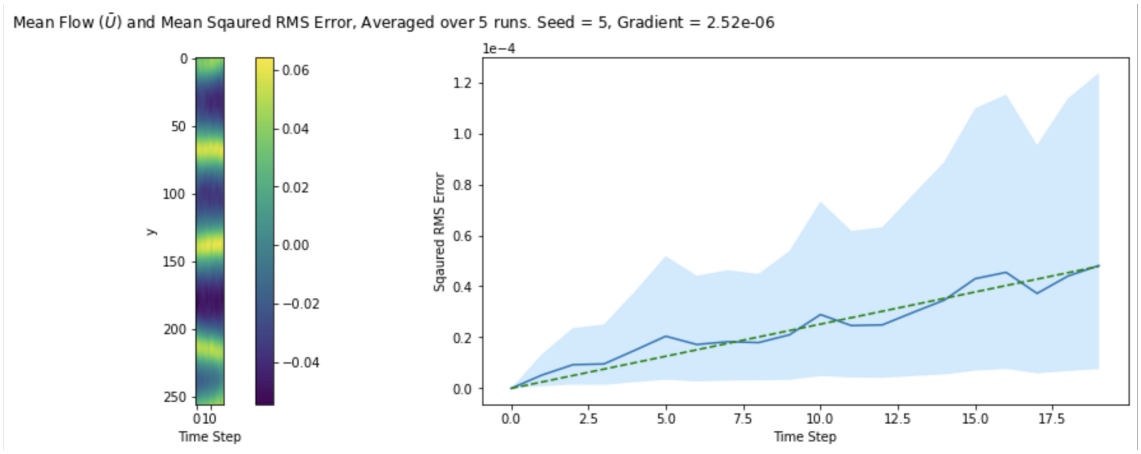


Fig. A.8 *On the left the plot shows the mean flow averaged over the 5 runs, for 20 time steps. On the right we see the mean difference between runs shown in the solid blue line, with the standard deviation envelope shown in lighter blue. In this case, at time $t = 0$ the runs were initialised with a different random forcing, all other initial conditions remained the same. The green dashed line shows the gradient, determined via regression to represent the diffusion rate, characterised by the diffusion constant. The different runs that make up the averages in this figure have been set off with different realisations of the forcing. The initial state for these runs is that of a long-run stable 3-jet.*

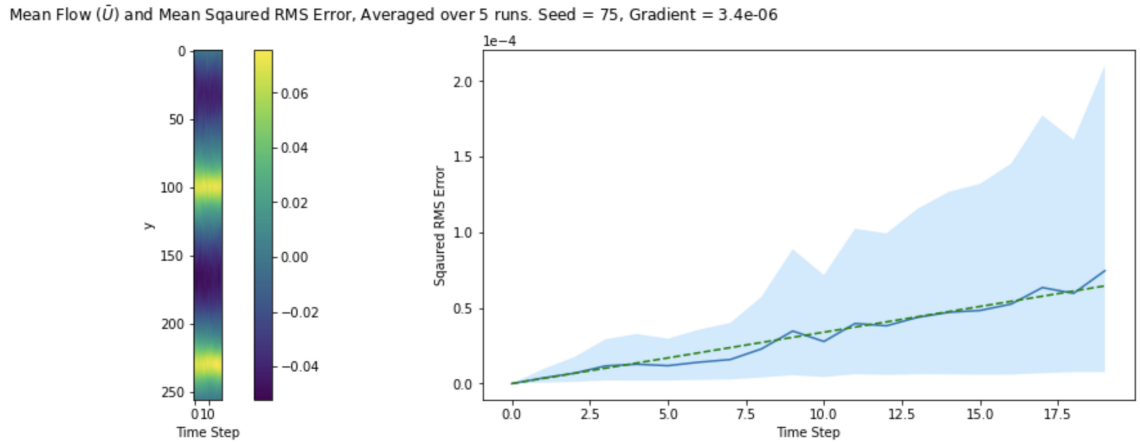


Fig. A.9 *On the left the plot shows the mean flow averaged over the 5 runs, for 20 time steps. On the right we see the mean difference between runs shown in the solid blue line, with the standard deviation envelope shown in lighter blue. In this case, at time* $t = 0$ *the runs were initialised with a different random forcing, all other initial conditions remained the same. The green dashed line shows the gradient, determined via regression to represent the diffusion rate, characterised by the diffusion constant. The different runs that make up the averages in this figure have been set off with different realisations of the forcing. The initial state for these runs is where the system is approaching a merging event .*