DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Deep Learning models for turbulent shear flow

PREM ANAND ALATHUR SRINIVASAN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Deep Learning models for turbulent shear flow

PREM ANAND ALATHUR SRINIVASAN

Degree Projects in Scientific Computing (30 ECTS credits)
Degree Programme in Computer Simulations for Science and Engineering (120 credits)
KTH Royal Institute of Technology year 2018
Supervisors at KTH: Ricardo Vinuesa, Philipp Schlatter, Hossein Azizpour
Examiner at KTH: Michael Hanke

TRITA-SCI-GRU 2018:236
MAT-E 2018:44

Royal Institute of Technology
School of Engineering Sciences
KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci

Abstract

Deep neural networks trained with spatio-temporal evolution of a dynamical system may be regarded as an empirical alternative to conventional models using differential equations. In this thesis, such deep learning models are constructed for the problem of turbulent shear flow. However, as a first step, this modeling is restricted to a simplified low-dimensional representation of turbulence physics. The training datasets for the neural networks are obtained from a 9-dimensional model using Fourier modes proposed by Moehlis, Faisst, and Eckhardt [29] for sinusoidal shear flow. These modes were appropriately chosen to capture the turbulent structures in the near-wall region. The time series of the amplitudes of these modes fully describe the evolution of flow. Trained deep learning models are employed to predict these time series based on a short input seed. Two fundamentally different neural network architectures, namely multilayer perceptrons (MLP) and long short-term memory (LSTM) networks are quantitatively compared in this work. The assessment of these architectures is based on (i) the goodness of fit of their predictions to that of the 9-dimensional model, (ii) the ability of the predictions to capture the near-wall turbulence structures, and (iii) the statistical consistency of the predictions with the test data. LSTMs are observed to make predictions with an error that is around 4 orders of magnitude lower than that of the MLP. Furthermore, the flow fields constructed from the LSTM predictions are remarkably accurate in their statistical behavior. In particular, deviations of 0.45% and 2.49% between the true data and the LSTM predictions were obtained for the mean flow and the streamwise velocity fluctuations, respectively.

Sammanfattning

Djupinlärningsmodeller för turbulent skjuvflöde

Djupa neuronät som är tränade med rum-tids utveckling av ett dynamiskt system kan betraktas som ett empiriskt alternativ till konventionella modeller som använder differentialekvationer. I denna avhandling konstruerar vi sådana djupinlärningsmodeller för att modellera en förenklad lågdimensionell representation av turbulensfysiken. Träningsdata för neuronäten erhålls från en 9-dimensionell modell (Moehlis, Faisst och Eckhardt [29]) för olika Fourier-moder i en skärskikt. Dessa moder har ändamålsenligt valts för att avbilda de turbulenta strukturerna i regionen nära väggen. Amplitudernas tidsserier för dessa moder beskriver fullständigt flödesutvecklingen, och tränade djupinlärningsmodeller används för att förutsäga dessa tidsserier baserat på en kort indatasekvens. Två fundamentalt olika neuronätsarkitekturer, nämligen flerlagerperceptroner (MLP) och långa närminnesnätverk (LSTM), jämförs kvantitativt i denna avhandling. Utvärderingen av dessa arkitekturer är baserad på (i) hur väl deras förutsägelser presterar jämfört med den 9-dimensionella modellen, (ii) förutsägelsernas förmåga att avbilda turbulensstrukturerna nära väggar och (iii) den statistiska överensstämmelsen mellan nätvärkets förutsägelser och testdatan. Det visas att LSTM gör förutsägelser med ett fel på ungefär fyra storleksordningar lägre än för MLP. Vidare, är strömningsfälten som är konstruerade från LSTM-förutsägelser anmärkningsvärt noggranna i deras statistiska beteende. I synnerhet uppmättes avvikelser mellan de sanna- och förutsagda värdena för det genomsnittliga flödet till $0,45\%$, och för de strömvisa hastighetsfluktionerna till $2,49\%$.

Acknowledgements

I am fortunate to have been supervised by Ricardo Vinuesa, whose knowledge and enthusiasm propelled me to complete this thesis project. I am grateful to him for his constant guidance, and also for taking the time to proofread this thesis.

I thank my other supervisors, Hossein Azizpour and Philipp Schlatter, who played an active role in shaping this thesis project with their expertise and insightful comments.

I thank the coordinators of the COSSE joint Master program, Michael Hanke and Reinhard Nabben, for taking care of all administrative procedures at KTH and TU Berlin respectively.

I thank all my friends in Berlin and Stockholm for the unforgettable memories. Special thanks to André Scheir Johansson and his family, who overwhelmed me with their love and kindness.

Words are not enough to thank my parents for their immense support and never-ending faith in me.

Contents

Chapter 1

Introduction

A salient feature of deep learning architectures, especially deep neural networks (DNNs), is their ability to learn spatial or temporal relationships between raw data inputs. Hence, DNNs are a promising alternative to model complex dynamical systems whose physics is not accurately known or contain an inherent stochasticity. The chaotic nature of turbulent fluid flows, and the immense computational effort required to accurately simulate them, offer a possibility for DNN models to potentially achieve performance gains over conventional simulation approaches.

Previous efforts in deep learning for fluid dynamics are largely restricted to using DNNs to accelerate specific parts of a fluid simulation [38, 46]. The ability of DNNs to directly model and simulate entire velocity fields remains unexplored. Furthermore, the assessment of different DNN architectures, based on their prediction errors, can potentially define the future role that DNNs may play in modeling fluid flows.

The objective of this thesis is to assess the performance of DNNs on simplified turbulent data. The training datasets are generated using a low-dimensional model based on Fourier modes [29]. Hence, the flow is fully described by the time-series of amplitudes of these modes. Different DNN architectures are used to predict these time-series, and their ability to statistically adhere to the simplified physics is studied.

This chapter introduces the reader to the idea of data-driven methods in fluid simulations, and describes the problem of interest and its significance.

1.1 Deep learning in fluid mechanics

Artificial neural networks are computational frameworks for learning specific tasks solely from examples. They were inspired by their biological counterparts, which are organized as hierarchical layers of neurons as discovered by Nobel laureates Hubel and Wiesel [19]. Since the development of powerful learning algorithms for artificial neural networks, they are being extensively used for tasks such as image and speech recognition. Although in existence for several decades, the recent success of DNNs can be attributed to two critical factors: (i) increased computational power, and (ii) larger amount of data that benefits the training of a deep architecture.

Data-based methods in fluid mechanics have been in the form of eduction techniques such as proper orthogonal decomposition (POD) [18] or dynamic mode decomposition (DMD) [25]. These methods are used to reduce the dimensionality of the flow data using singular value decomposition. Along with Galerkin projection, POD/DMD can be used to develop low-dimensional models that characterize some of the relevant physical phenomena in the flow [5]. These models are computationally cheap to simulate while providing adequate representation of the physics in their modes. Note that, for high-Reynolds-number (Re) flows, POD/DMD require more eigenfunctions for acceptable convergence rates.

Mathematically, performing POD on a dataset is equivalent to training a linear Neural Network to perform identity mapping on the dataset [3]. This methodology was further generalized by using a nonlinear DNN, which provided better prediction capabilities for an additional computational cost [28]. This was successfully implemented to reconstruct near-wall fields in turbulent flow. DNNs are expected to overcome some of the shortcomings of POD, by extracting multi-scale features and understanding translational or rotational invariance. DNNs have demonstrated this dexterity in tasks such as image recognition. However, DNNs have the disadvantage of higher computational costs and the fact they do not generate modes that have a physical interpretation.

There have been limited attempts to use DNNs in the modeling and analysis of flow fields. Ling, Kurzawski, and Templeton [27] proposed using DNNs for Reynolds-Averaged Navier–Stokes (RANS) turbulence models. The aim of their work was to make an improved pre-

diction for the Reynolds-stress anisotropy tensor using high-fidelity simulation data. To that end, the authors construct a specialized DNN architecture with 8-10 hidden layers that embeds rotational invariance. Improvements from the DNN configurations were also demonstrated for different geometries and Reynolds numbers. Zhang and Duraisamy [47] introduced a new term that adjusts the solver performance to RANS models that can be learned directly from data. This term was determined for various DNS, LES or experimental data using a bayesian inversion process, and used to train a neural network. The network reconstructs this adjustment term in a function form, which is queried by the RANS solver whenever a turbulence model is computed. Weatheritt and Sandberg [43] used genetic programming to develop better RANS models by *learning* a nonlinear stress-strain relationship using turbulent duct flow data obtained from Ref. [41]. Their algorithm symbolically regresses and returns a mathematical equation for the Reynolds stress anisotropy tensor that can be easily implemented in a RANS simulation.

Convolutional neural networks (CNN) are a special type of DNNs with localized connections between the layers. These networks have been used to accelerate time-consuming steps in fluid simulations, which has applications in computer graphics and animation for providing realistic smoke and fluid effects. Tompson et al. [38] incorporated CNNs in the standard operator splitting method to solve the incompressible Euler equations. The method has a pressure-projection step, in which a sparse linear system has to be solved in order to get the velocity update for each time-step. This step, which is traditionally solved using preconditioned conjugate gradient (PCG) or Jacobi methods, is formulated as an unsupervised learning problem and solved using CNNs. In such a learning problem, the network does not require any ground truth. Yang, Yang, and Xiao [46] solved the pressure-projection step by formulating it as a supervised learning problem that uses PCG data as ground truth. In a similar work, Hennigh [15] used an encoder/decoder DNN to compress the state of a Lattice-Boltzmann Simulation, and learn the dynamics from this compressed state.

CNNs have also been used to directly predict velocity fields. In Ref. [13], CNNs were used as surrogate models to predict the steady-state velocity field for non-uniform laminar flow around obstacles. The predictions using CNN were found to be two orders of magni-

tude faster than GPU-accelerated Lattice Boltzmann method simulations, and four orders of magnitude faster than the CPU-based solver at a low error cost. Such surrogate models are of considerable value to engineers for design-space exploration and interactive design.

Although deep learning has plenty of potential, its application to prediction of fluid flows has so far been unexplored. It is therefore important to evaluate the performance of DNNs on one of the hardest problems in fluid mechanics research - simulating velocity fields in turbulent flow. This thesis represents the first step in that direction by attempting to simulate near-wall turbulent structures in shear flow.

1.2 Turbulent structures in shear flow

Shear flow is a fundamental class of flows where the flow velocity changes in the direction perpendicular to the flow due to shear stresses. In wall-bounded shear flows, these stresses originate from the solid boundary. An example of such flows is the plane Couette flow, where the fluid between infinite parallel plates is sheared due to the motion of the plates in the same speed but in opposite directions. It is well-established that the nature of the flow depends on the critical value of the Reynolds Number (Re_c), below which the flow is observed to be laminar, where the velocity is completely parallel to the walls and varies only in the wall-normal direction. For plane Couette flow, this value was estimated to be $Re_c \approx 350$ using bifurcation analysis, which coincides with that found through experiments [32]. Above Re_c, the flow may be considered turbulent, and understanding the transition to turbulence has been the subject of several decades of research and the motivation behind the analysis of simplified models of fluid flow.

Mathematically, shear flow can be viewed as a dynamical system $\mathrm{d}u/\mathrm{d}t = f(u, Re)$ with a stable fixed point (the laminar state), and for $Re > Re_c$ the system has a strange attractor, i.e. the turbulent state. Ironically, there is a considerable complexity in constructing such a simplified model from the full Navier–Stokes equations, while predicting the critical Reynolds number and the Re- dependence of turbulence statistics. Due to the linear stability of the laminar state in shear flows for all Re, center manifold or other reduction techniques cannot be applied.

A different approach to develop low-dimensional models is based

on understanding the mechanisms that maintain turbulence. A large body of experimental work, direct numerical simulations, and proper orthogonal decomposition of flow fields suggest the existence of a self-sustaining process in shear flows. Jiménez and Moin [20] isolated this process by studying periodic solutions of Navier–Stokes equations for turbulent channel flow. The self-sustaining near-wall turbulence cycle can be summarized as follows:

1. In the near-wall region, pairs of counter-rotating *streamwise rolls or vortices* lift the slow-moving fluid away from the wall.
2. The lifted-up *streaks* of fluid have a relatively reduced axial velocity, a fact that leads to the velocity profiles shown in Figure 1.1.
3. The spanwise profiles of these streaks contain inflexion points and are inviscidly unstable, which causes breakdown of these streaks (*bursting*). The nonlinear interactions due to bursting leads to the regeneration of streamwise vortices.

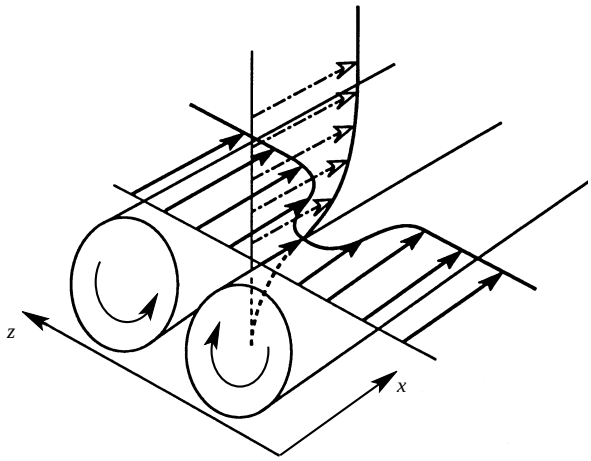


Figure 1.1: Schematic view of the self-sustaining process: x and z are the streamwise and spanwise directions, with the wall on the $xz-$plane. Fluid near the wall is lifted by the counter-rotating vortices, which form the streaks. (Figure extracted from Ref. [33]).

This suggests that the dynamics of shear flow turbulence can in principle be represented by a few modes. Hence, the objective in deriving low-dimensional model is to select the specific modes that describe the self-sustaining cycle with accuracy.

Deep learning can be a valuable supplement to these low-dimensional models. With the development of novel DNN architectures that can effectively learn temporal information and spatial locality, it is worthwhile to see whether deep learning can effectively model shear-flow turbulence by replicating the self-sustaining process. Specifically, we explore the ability of a Deep Learning model to predict future states of a flow field based on previous states, and establish the aspects of the flow that are learned and retained in the predictions.

1.3 Aim and scope

This thesis represents a first step towards modeling turbulent shear flow using deep learning. In this work, the performance of deep learning architectures using simplified turbulent data is assessed. The training datasets are obtained from a low-dimensional model for shear flows described in Ref. [29], and briefly discussed in chapter 2 of this thesis. The model uses nine Fourier modes to describe sinusoidal shear flow in which fluid between two free-slip walls experiences a sinusoidal body force. The modes represent the mean sinusoidal laminar profile along with the turbulent structures and interactions between them. Galerkin projection of the Navier–Stokes equations onto this nine-dimensional Fourier space gives rise to a set of ordinary differential equations (ODEs) that describe the temporal evolution of nine amplitudes.

The derived system of ODEs is highly sensitive to initial conditions. Even a perturbation of 10^{-12} on one of the amplitudes may lead to a different temporal evolution. In this work, we take advantage of this high sensitivity and generate multiple datasets by perturbing an initial condition, as described by Kim [22]. These time series are then used to train deep learning models.

Trained models are used for long-term forecasting of time series based on a short input sequence. The models are evaluated based on their consistency with the self-sustaining cycle and also the ability to provide a statistical description of the flow fields. A large part of the thesis is devoted to finding the best architecture for neural networks and their performance for different tuning parameters. A key aspect of this thesis is the comparison between two fundamentally different DNNs namely multilayer perceptrons (MLP) and long short-term

memory (LSTM) networks. An improved predictions are observed when using the latter, since it is specifically designed for time series predictions. This work aims to quantify this improvement.

The outline of this thesis is as follows: In Chapter 2, the low-dimensional model is described and the evaluation metrics for the deep learning models are defined. Chapter 3 deals with finding the best MLP architecture for the problem, and also describes the choice of compilation parameters for the architecture. In Chapter 4, the LSTM architectures are assessed, while highlighting their advantages over MLP. When comparing the performance of both architectures with the ODE model, flow visualizations are added to illustrate that the neural networks reproduce all the near-wall structures. Chapter 5 concludes with an analysis of the general nature of neural network predictions.

Chapter 2

A low-dimensional model for shear flow

The Navier–Stokes equations in wavenumber space can be regarded as an infinite set of nonlinear ordinary differential equations for the Fourier coefficients. Hence, proper orthogonal decomposition of flow fields using Fourier basis functions and subsequent Galerkin projection yields this infinite set of ODEs. A low-dimensional model attempts to capture the dominant features of the flow using a truncated set of a finite number of orthogonal modes.

One of the earliest such models for near-wall turbulence was developed by Aubry et al. [2] using five POD modes. The ability of this model to qualitatively describe the flow behavior led to its further development in the monograph by Holmes et al. [18]. A model exhibiting the self-sustaining process described in Section 1.2 was developed by Waleffe [42] using eight Fourier modes, which was further simplified to four modes for sinusoidal shear flow. In this model, the turbulent state is modeled as a periodic orbit or a strange attractor. However, Eckhardt and Mersmann [8] later showed the fluctuating and transient nature of the turbulent state by projecting the flow evolution equations onto a set of 19 Fourier modes. The dimensionality of this model could be reduced to 9 by removing residual symmetry, without loss of qualitative behavior. The nine-dimensional model described in Ref. [29] is considered in this thesis. The modes of this model represent the mean profile, streamwise vortices, streaks and their instabilities, and the couplings between them. In this chapter, the model is described in detail along with its dynamics.

2.1 Sinusoidal shear flows

Sinusoidal shear flows are flows between infinite parallel free-slip walls with a sinusoidal body force acting on the fluid. This body force causes the laminar flow profile to be sinusoidal as shown in Figure 2.1. The coordinate axes x, y and z represent the streamwise, wall-normal and spanwise directions, respectively. The distance between the walls is d, and the top and bottom walls are at $y = +d/2$ and $y = -d/2$ respectively, with the mid-plane at $y = 0$. The flow is assumed to be periodic in the streamwise and spanwise directions with periods $L_x d/2$ and $L_z d/2$, which define the domain under investigation.

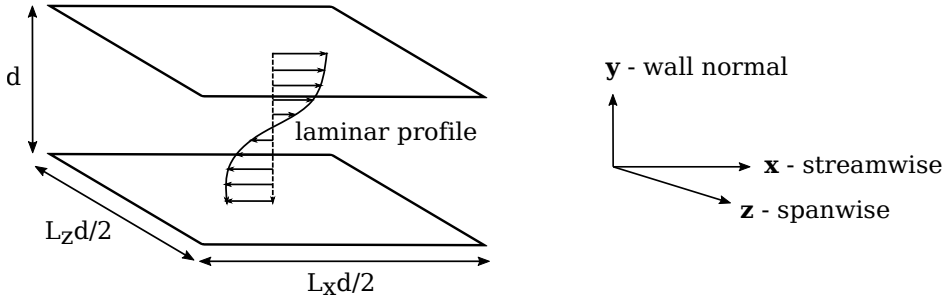


Figure 2.1: The domain for sinusoidal shear flow and its laminar profile.

The non-dimensionalized Navier–Stokes momentum equation for incompressible flow is written as:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla p + \frac{1}{Re}\nabla^2\mathbf{u} + \mathbf{F}(y), \tag{2.1}$$

with the Reynolds Number defined as:

$$Re := \frac{U_0 d}{2\nu}, \tag{2.2}$$

where ν is the fluid kinematic viscosity and U_0 is the characteristic velocity which is taken to be the laminar streamwise velocity at $y = +d/4$. The laminar profile is given by:

$$\mathbf{U}(y) = \sqrt{2}\sin(\pi y/2)\hat{\mathbf{e}}_x. \tag{2.3}$$

which is produced by the non-dimensional body force:

$$\mathbf{F}(y) = \frac{\sqrt{2}\pi^2}{4Re}\sin(\pi y/2)\hat{\mathbf{e}}_x, \tag{2.4}$$

where $\hat{\mathbf{e}}_x$ is the standard basis for $\mathbb{R}^3$ in the x- direction. The upper-case $\mathbf{U}$ denotes the mean velocities while the lower-case $\mathbf{u}$ denotes the instantaneous velocities. Furthermore, since the flow is incompressible, we have:

$$\nabla \cdot \mathbf{u} = 0. \tag{2.5}$$

Due to the free-slip at the walls, the boundary conditions are:

$$v|_{y=\pm d/2} = 0, \qquad \left.\frac{\partial u}{\partial y}\right|_{y=\pm d/2} = \left.\frac{\partial w}{\partial y}\right|_{y=\pm d/2} = 0, \tag{2.6}$$

where u, v and w are the components of $\mathbf{u}$ in the coordinate directions.

2.2 The Fourier modes

In the following, d is set to 2 and hence the domain is denoted as:

$$\Omega = [0, L_x] \times [-1, 1] \times [0, L_z].$$

The nine Fourier modes chosen to describe the flow, as considered in Ref. [29], are listed here. The first mode $\boldsymbol{u}_1$ contains the basic laminar velocity profile, and the ninth mode $\boldsymbol{u}_9$ contains a modification to it due to turbulence. The mode $\boldsymbol{u}_2$ depicts the streak that accounts for the spanwise variation in the streamwise velocity. The streamwise vortices are captured by $\boldsymbol{u}_3$, and the remaining modes are associated with the instabilities in the streaks. Using the quantities $\alpha = 2\pi/L_x$, $\beta = \pi/2$, and $\gamma = 2\pi/L_z$, the modes are defined as follows:

$$\boldsymbol{u}_1 = \begin{pmatrix} \sqrt{2}\sin(\pi y/2) \\ 0 \\ 0 \end{pmatrix}, \tag{2.7}$$

$$\boldsymbol{u}_2 = \begin{pmatrix} \frac{4}{\sqrt{3}}\cos^2(\pi y/2)\cos(\gamma z) \\ 0 \\ 0 \end{pmatrix}, \tag{2.8}$$

$$\boldsymbol{u}_3 = \frac{2}{\sqrt{4\gamma^2 + \pi^2}} \begin{pmatrix} 0 \\ 2\gamma\cos(\pi y/2)\cos(\gamma z) \\ \pi\sin(\pi y/2)\sin(\gamma z) \end{pmatrix}, \tag{2.9}$$

$$\boldsymbol{u}_4 = \begin{pmatrix} 0 \\ 0 \\ \frac{4}{\sqrt{3}} \cos(\alpha x) \cos^2(\pi y/2) \end{pmatrix}, \tag{2.10}$$

$$\boldsymbol{u}_5 = \begin{pmatrix} 0 \\ 0 \\ 2 \sin(\alpha x) \sin(\pi y/2) \end{pmatrix}, \tag{2.11}$$

$$\boldsymbol{u}_6 = \frac{4\sqrt{2}}{\sqrt{3(\alpha^2+\gamma^2)}} \begin{pmatrix} -\gamma \cos(\alpha x) \cos^2(\pi y/2) \sin(\gamma z) \\ 0 \\ \alpha \sin(\alpha x) \cos^2(\pi y/2) \cos(\gamma z) \end{pmatrix}, \tag{2.12}$$

$$\boldsymbol{u}_7 = \frac{2\sqrt{2}}{\sqrt{\alpha^2+\gamma^2}} \begin{pmatrix} \gamma \sin(\alpha x) \sin(\pi y/2) \sin(\gamma z) \\ 0 \\ \alpha \cos(\alpha x) \sin(\pi y/2) \cos(\gamma z) \end{pmatrix}, \tag{2.13}$$

$$\boldsymbol{u}_8 = N_8 \begin{pmatrix} \pi\alpha \sin(\alpha x) \sin(\pi y/2) \sin(\gamma z) \\ 2(\alpha^2+\gamma^2)\alpha \cos(\alpha x) \cos(\pi y/2) \sin(\gamma z) \\ -\pi\gamma \cos(\alpha x) \sin(\pi y/2) \cos(\gamma z) \end{pmatrix}, \tag{2.14}$$

$$\boldsymbol{u}_9 = \begin{pmatrix} \sqrt{2} \sin(3\pi y/2) \\ 0 \\ 0 \end{pmatrix}, \tag{2.15}$$

where

$$N_8 = \frac{2\sqrt{2}}{\sqrt{(\alpha^2+\gamma^2)(4\alpha^2+4\gamma^2+\pi^2)}}.$$

The modes are orthogonal and normalized, and therefore:

$$\int\int\int_\Omega \boldsymbol{u}_n \cdot \boldsymbol{u}_m \, \mathrm{d}^3\boldsymbol{x} = 2L_x L_z \delta_{nm} \tag{2.16}$$

2.3 Modeling equations

The velocity fields in spatial coordinates $(\mathbf{x})$ and time (t) are given by superposition of these nine modes:

$$\boldsymbol{u}(\boldsymbol{x}, t) := \sum_{j=1}^{9} a_j(t) \boldsymbol{u}_j(\boldsymbol{x}). \tag{2.17}$$

Galerkin projection onto this nine-dimensional subspace involves inserting the above ansatz (2.17) in the Navier–Stokes equation (2.1), multiplying by $\boldsymbol{u}_n$, integrating over Ω and finally using the orthogonality condition (2.16). This procedure leads to the following set of ODEs in time:

$$\frac{\mathrm{d}a_1}{\mathrm{d}t} = \frac{\beta^2}{Re} - \frac{\beta^2}{Re}a_1 - \sqrt{\frac{3}{2}}\frac{\beta\gamma}{\kappa_{\alpha\beta\gamma}}a_6a_8 + \sqrt{\frac{3}{2}}\frac{\beta\gamma}{\kappa_{\beta\gamma}}a_2a_3, \tag{2.18}$$

$$\begin{aligned}\frac{\mathrm{d}a_2}{\mathrm{d}t} =& -\left(\frac{4\beta^2}{3}+\gamma^2\right)\frac{a_2}{Re} + \frac{5\sqrt{2}}{3\sqrt{3}}\frac{\gamma^2}{\kappa_{\alpha\gamma}}a_4a_6 - \frac{\gamma^2}{\sqrt{6}\kappa_{\alpha\gamma}}a_5a_7 \\ & - \frac{\alpha\beta\gamma}{\sqrt{6}\kappa_{\alpha\gamma}\kappa_{\alpha\beta\gamma}}a_5a_8 - \sqrt{\frac{3}{2}}\frac{\beta\gamma}{\kappa_{\beta\gamma}}(a_1a_3 + a_3a_9),\end{aligned} \tag{2.19}$$

$$\begin{aligned}\frac{\mathrm{d}a_3}{\mathrm{d}t} =& -\frac{\beta^2+\gamma^2}{Re}a_3 + -\frac{2\alpha\beta\gamma}{\sqrt{6}\kappa_{\alpha\gamma}\kappa_{\beta\gamma}}(a_4a_7 + a_5a_6) \\ & + \frac{\beta^2(3\alpha^2+\gamma^2) - 3\gamma^2(\alpha^2+\gamma^2)}{\sqrt{6}\kappa_{\alpha\gamma}\kappa_{\beta\gamma}\kappa_{\alpha\beta\gamma}}a_4a_8,\end{aligned} \tag{2.20}$$

$$\begin{aligned}\frac{\mathrm{d}a_4}{\mathrm{d}t} =& -\frac{3\alpha^2+4\beta^2}{3Re}a_4 - \frac{\alpha}{\sqrt{6}}(a_1a_5 + a_5a_9) - \frac{10}{3\sqrt{6}}\frac{\alpha^2}{\kappa_{\alpha\gamma}}a_2a_6 \\ & - \sqrt{\frac{3}{2}}\frac{\alpha\beta\gamma}{\kappa_{\alpha\gamma}\kappa_{\beta\gamma}}a_3a_7 - \sqrt{\frac{3}{2}}\frac{\alpha^2\beta^2}{\kappa_{\alpha\gamma}\kappa_{\beta\gamma}\kappa_{\alpha\beta\gamma}}a_3a_8,\end{aligned} \tag{2.21}$$

$$\begin{aligned}\frac{\mathrm{d}a_5}{\mathrm{d}t} =& -\frac{(\alpha^2+\beta^2)}{Re}a_5 + \frac{\alpha}{\sqrt{6}}(a_1a_4 + a_4a_9) + \frac{\alpha^2}{\sqrt{6}\kappa_{\alpha\gamma}}a_2a_7 \\ & - \frac{\alpha\beta\gamma}{\sqrt{6}\kappa_{\alpha\gamma}\kappa_{\alpha\beta\gamma}}a_2a_8 + \frac{2\alpha\beta\gamma}{\sqrt{6}\kappa_{\alpha\gamma}\kappa_{\beta\gamma}}a_3a_6,\end{aligned} \tag{2.22}$$

$$\frac{\mathrm{d}a_6}{\mathrm{d}t} = -\frac{3\alpha^2+4\beta^2+3\gamma^2}{3Re}a_6 + \frac{\alpha}{\sqrt{6}}(a_1a_7 + a_7a_9) + \sqrt{\frac{3}{2}}\frac{\beta\gamma}{\kappa_{\alpha\beta\gamma}}a_1a_8$$

$$\frac{10}{3\sqrt{6}}\frac{\alpha^2-\gamma^2}{\kappa_{\alpha\gamma}}a_2a_4 - 2\sqrt{\frac{2}{3}}\frac{\alpha\beta\gamma}{\kappa_{\alpha\gamma}\kappa_{\beta\gamma}}a_3a_5 + \sqrt{\frac{3}{2}}\frac{\beta\gamma}{\kappa_{\alpha\beta\gamma}}a_8a_9, \tag{2.23}$$

$$\begin{aligned}\frac{\mathrm{d}a_7}{\mathrm{d}t} =& -\frac{\alpha^2+\beta^2+\gamma^2}{Re}a_7 - \frac{\alpha}{\sqrt{6}}(a_1a_6 + a_6a_9)\\ &+ \frac{1}{\sqrt{6}}\frac{\gamma^2-\alpha^2}{\kappa_{\alpha\gamma}}a_2a_5 + \frac{\alpha\beta\gamma}{\sqrt{6}\kappa_{\alpha\gamma}\kappa_{\beta\gamma}}a_3a_4,\end{aligned} \tag{2.24}$$

$$\begin{aligned}\frac{\mathrm{d}a_8}{\mathrm{d}t} =& -\frac{\alpha^2+\beta^2+\gamma^2}{Re}a_8 + \frac{2\alpha\beta\gamma}{\sqrt{6}\kappa_{\alpha\gamma}\kappa_{\alpha\beta\gamma}}a_2a_5\\ &\frac{\gamma^2(3\alpha^2-\beta^2+3\gamma^2)}{\sqrt{6}\kappa_{\alpha\gamma}\kappa_{\beta\gamma}\kappa_{\alpha\beta\gamma}}a_3a_4,\end{aligned} \tag{2.25}$$

$$\frac{\mathrm{d}a_9}{\mathrm{d}t} = -\frac{9\beta^2}{Re}a_9 - \sqrt{\frac{3}{2}}\frac{\beta\gamma}{\kappa_{\alpha\beta\gamma}}a_6a_8 + \sqrt{\frac{3}{2}}\frac{\beta\gamma}{\kappa_{\beta\gamma}}a_2a_3, \tag{2.26}$$

where

$$\kappa_{\alpha\gamma} = \sqrt{\alpha^2+\gamma^2},\ \kappa_{\beta\gamma} = \sqrt{\beta^2+\gamma^2},\ \text{and}\ \kappa_{\alpha\beta\gamma} = \sqrt{\alpha^2+\beta^2+\gamma^2}. \tag{2.27}$$

The model has one stable fixed point for all Re with $a_1 = 1$, and $a_2 = \cdots = a_9 = 0$, which corresponds to the laminar state. However, the existence of other fixed points and periodic orbits and their stability depends on the domain size and the Re of the flow.

2.4 Dynamics of the model

A domain with $L_x = 4\pi$ and $L_z = 2\pi$ with $Re = 400$ is considered throughout this thesis. This domain size has been considered in previous studies of shear flow turbulence in plane Couette flow and sinusoidal shear flow, and has been shown to be optimal for the formation of stationary coherent structures [7, 32].

The time series for the nine amplitudes in the model are shown in Figure 2.2. The initial fluctuations correspond to the non-periodic, irregular, transient turbulent state. On analysis, these fluctuations can be attributed to the presence of a high number of unstable periodic orbits for the chosen domain size and Re [30]. The model eventually

decays to the laminar state at approximately $t \approx 3000$. This shows that the fluctuations do not belong to a strange attractor, and instead to a transient state on a chaotic saddle. The amplitude a_1 becomes large near the times $t \approx 750$ and 1300, which indicates an attempt to leave the turbulent state and laminarize. However, the other amplitudes have to be small enough for such an attempt to be successful.

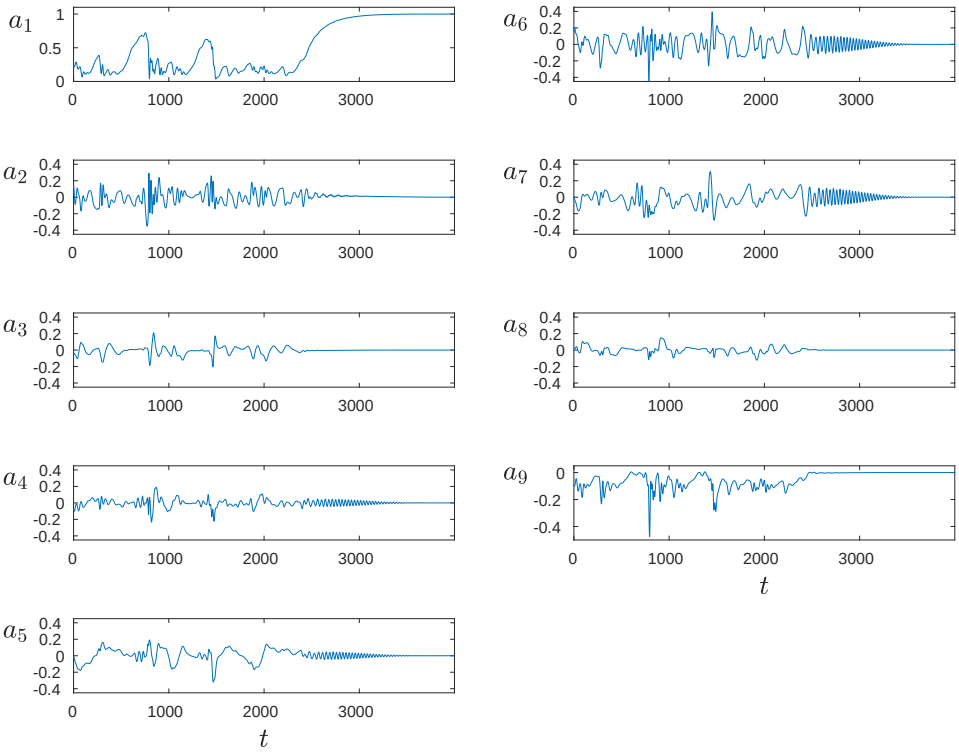


Figure 2.2: Typical chaotic time series for the model coefficients, for $L_x = 4\pi$ and $L_z = 2\pi$ with $Re = 400$.

When the velocity fields are constructed from these amplitudes, the flow clearly shows the self-sustaining process described in Section 1.2. Two slices of the instantaneous flow field at $t = 400$ are shown in Figure 2.3. The upper-right panel shows the downstream vortices, which lift the fluid off the bottom wall, represented by the arrows for the velocity field in the plane. In the same panel, the formation of streaks can be observed by the colors for the downstream component. In the lower-right panel, which shows the mid-plane between the plates $y = 0$, the streak (here, represented by the arrows) goes unstable by the basic sinusoidal instability mode in the model. The instability grows

and breaks up into patches and rearranges with the regions of vertical motion. The vortices reappear due to this phenomenon but in the opposite direction thereby lifting the fluid off the top wall. The self-sustaining process continues in such an alternating fashion until the amplitudes reach the laminar-state fixed point.

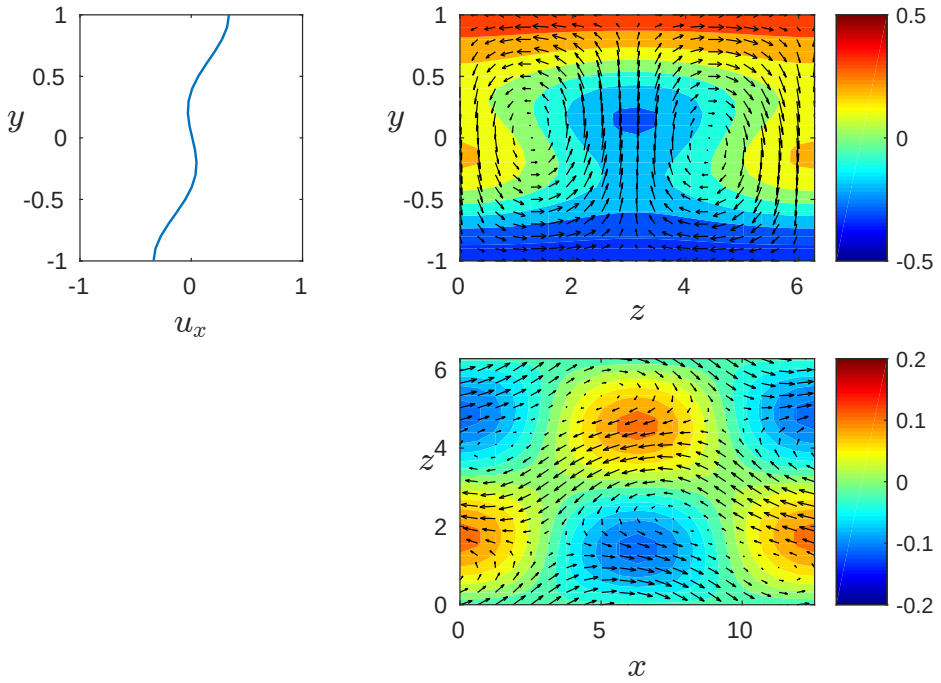


Figure 2.3: Velocity fields constructed from the time series in Figure 2.2 at $t = 400$, for $L_x = 4\pi$ and $L_z = 2\pi$ with $Re = 400$. The left panel shows the mean profile. The upper-right panel shows the streamwise vortices for which the flow is averaged in the downstream direction. The colored contours represent the velocity component perpendicular to the plane, and the vectors represent the components on the plane. The lower-right panel shows the flow in the mid-plane.

2.5 Data generation

The lifetime of the transient turbulent state is highly sensitive to the initial conditions. In fact, the lifetime exhibits a fractal dependence at any resolution as demonstrated in Ref. [29]. To further illustrate this

sensitivity, consider the following initial condition:

$$(a_{10}, a_{20}, a_{30}, a_{50}, a_{60}, a_{70}, a_{80}, a_{90}) = (1, 0.07066, -0.07076, 0, 0, 0, 0, 0) \tag{2.28}$$

for different values of a_{40}. Figure 2.4 shows the time series for amplitude a_1, for three different a_{40} which differ only by 10^{-12}. Although the initial trajectories are similar for up to $t \approx 1000$, they diverge to reach the laminar state at different times. Despite the fact that the system rarely reaches a periodic orbit for the chosen domain size, it does more commonly for a smaller domain of $L_x = 1.75\pi$ and $L_z = 1.2\pi$ for the same Re. In that case, a small perturbation can make the system reach a periodic orbit instead of a fixed point.

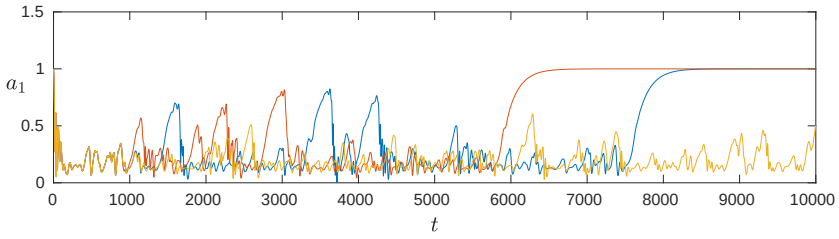


Figure 2.4: Time series of amplitude a_1 for initial conditions (2.28) and three different a_{40} values. Blue: $a_{40} = 0.04$, Red: $a_{40} = 0.04 + 10^{-12}$, Yellow: $a_{40} = 0.04 - 10^{-12}$.

This sensitivity is highly useful in generating datasets for training the neural networks. Since the phenomenon of interest is the non-periodic transient turbulence, time series with such prolonged dynamics are generated and collected. This is done by randomly perturbing a_{40} for the initial conditions (2.28) and generating a time series for up to $t = 4000$. If the generated series exhibits a fixed point or a periodic orbit, it is discarded. By training with such datasets, the neural network can be expected to learn only the chaotic behavior which eliminates any relaxation to stable orbits.

Chapter 3

Multilayer Perceptrons

Predicting the future states of temporal sequences has many real-world applications ranging from forecasting weather to currency exchange rates. In science, the desire to know the future led to the postulation and mathematical formulation of physical laws. In general, there are two types of knowledge that can help predict the behavior of a system. The first is the knowledge of the laws underlying the behavior of the system, which can be expressed in the form of equations. The second is the knowledge of empirical regularities observed in the previous states of the system, while lacking the mechanistic understanding. The aim of neural networks is to acquire the latter, due to the unavailability of the former for most real-world systems.

Multilayer perceptrons (MLPs) are the most basic type of artificial neural networks. They consists of three or more layers of nodes, with each node connected to all nodes in the preceding and succeeding layers. The workhorse of learning with neural networks is the backpropagation algorithm [34], which computes the parameters in each node from the training data. MLPs are frequently used in forecasting time series [24, 36, 44]. However, their major limitation is that they are only able to learn an input-output mapping rendering it *static*. The main objective of this thesis is to quantify this limitation by comparing MLPs with more *dynamic* neural networks.

In this chapter, the conventional way to model time series with an MLP is presented, following a theoretical introduction to MLP architectures. For the chosen problem of predicting the nine Fourier amplitudes, different MLP architectures are systematically investigated while exploring the necessary size of datasets for training.

3.1 Theory

Let X be an input space and Y be an output space. A set of m input-output observations defined as:

$$S = \left\{ (x^{(j)}, y^{(j)}) \;\middle|\; x^{(j)} \in X,\ y^{(j)} \in Y,\ \forall j \in \{1, \ldots, m\} \right\}$$

is called a *training set*, and each tuple (x, y) is called a *training example* or *sample*. The objective of machine learning is to find the mapping $f : X \to Y$ using the training set such that a *loss function* $L(f(x); y)$ is minimized.

For an MLP, the mapping $f(x)$ can be recursively decomposed into functions of weighted sums. The number of recursions is the number of layers in the neural network. The last layer that outputs $z = f(x)$ is called the *output layer*. For an MLP with $l + 1$ layers, the l layers preceding the output layer are called the *hidden layers*. The input x that enters the first hidden layer may theoretically be regarded as the output of an *input layer*. The following indexing is used for the layers:

$$i = \begin{cases} 0 & \text{for the input layer} \\ 1, 2, \ldots l & \text{for the hidden layers} \\ l + 1 & \text{for the output layer} \end{cases}$$

An illustration of an MLP as layers of interconnected nodes is shown in Figure 3.1. The evaluation of $z = f(x)$ is done using the recursive Algorithm 1. If n_i is the number of nodes in the i^{th} layer, for real inputs and outputs (i.e. $X = \mathbb{R}^{n_0}, Y = \mathbb{R}^{n_{l+1}}$), we can further define the dimensions of the variables used in Algorithm 1 as follows:

$$z_i \in \mathbb{R}^{n_i} \quad \forall i \in \{0, 1, \ldots l + 1\},$$
$$h_i \in \mathbb{R}^{n_i},\ W_i \in \mathbb{R}^{n_i \times n_{i-1}},\ b_i \in \mathbb{R}^{n_i} \quad \forall i \in \{1, 2, \ldots l + 1\}.$$

Algorithm 1: Compute the output of an MLP

Input: x
Output: $z = f(x)$
set $z_0 \leftarrow x$
for $i \leftarrow 1$ **to** $l + 1$ **do**
  $h_i \leftarrow W_i z_{i-1} + b_i$
  $z_i \leftarrow g_i(h_i)$
return $z \leftarrow z_{l+1}$

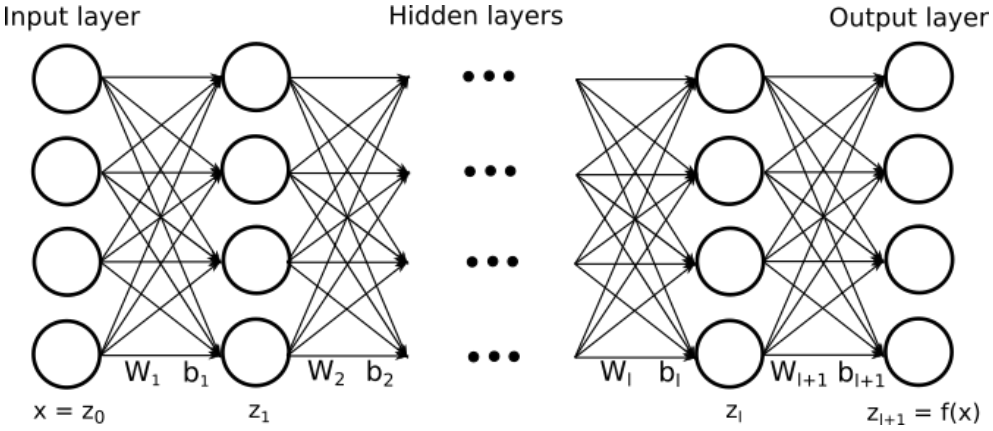


Figure 3.1: The MLP architecture

Here, g_i are the activation functions, which filter the contributing units in each layer. The MLP is fully parametrized by the *weight matrices* W_i and *biases* b_i, which perform the linear transformation from one hidden layer space to the next. The goal of training an MLP is to determine W_i and b_i using training data, for preset activation functions.

3.1.1 Training an MLP

Training is done by the backpropagation algorithm, which minimizes the loss function by using gradient descent [21] to adjust the weights and biases in each layer. Since this procedure requires computing the gradient of the loss function, the activation functions g_i used in all layers must be differentiable and continuous. The updates for the weights and biases in the algorithm depend on the choice of loss function. A standard choice is the mean-squared-error (mse) which is defined as:

$$L(f(x); y) = \frac{1}{2m} \sum_{j=1}^{m} ||y^{(j)} - f(x^{(j)})||^2. \tag{3.1}$$

This is the squared deviation of the network output from the expected output for each training example averaged over the training set. For each training example, if the squared error in each layer is denoted by $\delta_i^{(j)}$, the loss function can also be written as:

$$L(f(x); y) = \frac{1}{m} \sum_{j=1}^{m} \delta_{l+1}^{(j)}. \tag{3.2}$$

For this loss function, the backpropagation algorithm is outlined in Algorithm 2.

Algorithm 2: The backpropagation algorithm

Data: Training set S
Output: Weights W_i, and biases b_i for $i = \{1, \dots l+1\}$
while $L(f(x); y) >$ tolerance **do**
  for each training example $(x^{(j)}, y^{(j)})$ **do**
    Compute the MLP output using Algorithm 1
    $z^{(j)} \leftarrow f(x^{(j)})$

    Compute the error in the output layer
    $\delta_{l+1}^{(j)} = ||y^{(j)} - z^{(j)}||^2/2$

    Backpropagate the errors through the hidden layers
    for $i \leftarrow l$ **to** 1 **do**
      $\delta_i^{(j)} \leftarrow \left(W_{i+1}^T \delta_{i+1}^{(j)}\right) \circ g_i'(h_i^{(j)})$

    Update the weights and biases using gradient descent
    for $i \leftarrow 1$ **to** $l+1$ **do**
      $W_i \leftarrow W_i - \alpha\, \delta_i^{(j)}\, z_{i-1}^{(j)T}$
      $b_i \leftarrow b_i - \alpha\, \delta_i^{(j)}$

In the algorithm, the symbol $\circ$ represents the *Hadamard product* or element-wise multiplication. The parameter α used in the last step of the algorithm is called the *learning rate*. It can be manually set to a constant value or allowed to decay over time, since a too-high learning rate can prevent convergence. For all training runs in this thesis, a variant of the gradient descent algorithm called adaptive moment estimation (Adam) [23] is used. Adam is an adaptive learning rate method and is known to outperform standard gradient descent, which has a fixed learning rate.

3.1.2 Overfitting and validation sets

Since training a neural network is essentially fitting a mathematical function to data, it inherently has a risk of overfitting. An overfitted network is one that has achieved very low losses, and performs well on the training set. However, it performs poorly on new data.

To overcome this problem, it is common praxis while training neural networks to set aside a small portion of the training set as a *validation set* which will not be used during training. In training, one complete pass through the training set is called an *epoch*. At the end of each epoch, the model is evaluated using the validation set and a validation loss is calculated using the same loss function in Equation (3.1). Figure 3.2 (right) shows the typical behavior of an overfitted model by the change in the validation loss over several epochs. The default method to stop overfitting is called *early stopping*, whereby the training is stopped when the validation loss starts increasing.

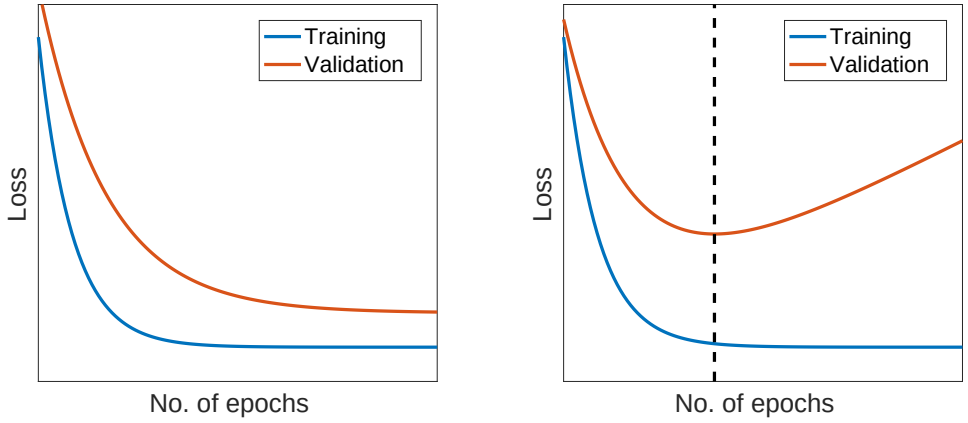


Figure 3.2: Training and validation loss for normal fit (left) and overfit (right). The dotted line in right panel marks early stopping, after which the model begins to overfit.

3.2 MLP modeling for time series

The conventional way to model time series with an MLP is to use a finite number of past samples as inputs. For instance, an input of $x = [a(t-1)\ a(t-2)\ \dots\ a(t-p)]^T$ is used to predict $a(t)$. The parameter p is called the *prediction order*, which is the length of the sequence used as inputs. Essentially, an MLP attempts to model temporal data by giving it a spatial representation. For the chosen problem of predicting the nine Fourier amplitudes, we have an input dimension of $n_0 = 9 \cdot p$ and an output dimension of $n_{l+1} = 9$.

Since the data obtained from the ODE model is in the form of time series, it has to be processed into training samples that can be fed

into the MLP. If each time series contains T time points, training with n_s time series implies a training set of $n_s(T-p)$ samples. The final 20% of this training set is used for validation, and therefore not employed during training. Since all generated time series have a constant $T = 4000$, training with 25 time series implies a training set of $78,000$ examples, and a validation set of $19,500$ examples.

For testing, the initial sequence of length p from a test series is fed to the neural network and sequentially predicted for up to $t = T$. The newly predicted series is compared with the test series using a set of evaluation metrics defined in the upcoming section.

3.2.1 Evaluation metrics

A straightforward metric of choice is the mean absolute error in the predictions, since all amplitudes are bound in the interval $[-1, 1]$. This is also a measure of how closely the MLP fits the ODE model. The absolute error can be defined as:

$$\epsilon_{t,k} = \frac{1}{T-p} \int_p^T |a_{k,test}(t) - a_{k,predicted}(t)| \, \mathrm{d}t \qquad \forall k \in \{1, 2, \dots 9\} \tag{3.3}$$

The predicted time series also have to be statistically consistent with the dynamics of fluid flow. Since shear flow between parallel plates can be assumed to be statistically stationary and predominantly one-dimensional, the mean velocities and velocity fluctuations change only in the wall-normal direction. The mean streamwise component of velocity is defined as follows:

$$U(y) = \frac{1}{L_x L_z T} \int_0^T \int_0^{L_z} \int_0^{L_x} u(x, y, z, t) \, \mathrm{d}x\mathrm{d}z\mathrm{d}t \tag{3.4}$$

The other two components of mean velocity, *i.e.* $V(y)$ and $W(y)$ can be defined likewise. The so-called Reynolds stresses can be defined for each pair of velocity components as:

$$\overline{u_i u_j}(y) = \frac{1}{L_x L_z T} \iiint u_i(x, y, z, t) u_j(x, y, z, t) \, \mathrm{d}x\mathrm{d}z\mathrm{d}t - U_i(y) U_j(y), \tag{3.5}$$

for $(u_i, u_j) \in \{u, v, w\} \times \{u, v, w\}$. Note that x and y in Equations (3.4) and (3.5) represent the coordinate directions and not the neural network inputs and outputs. Henceforth, for clarity the (y) is dropped in

the mean quantities such as $U(y)$. The profiles of all mean quantities obtained from a sample time series are shown in Figure 3.3. Since the dynamics of the flow is chiefly governed by the streamwise velocity component and the Reynolds stresses, error metrics based on them are defined as:

$$\epsilon_U = \frac{1}{2}\int_{-1}^{1} \frac{|U_{test} - U_{predicted}|}{\max(U_{test})}\,\mathrm{d}y, \tag{3.6}$$

$$\epsilon_{\overline{u^2}} = \frac{1}{2}\int_{-1}^{1} \frac{|\overline{u^2}_{test} - \overline{u^2}_{predicted}|}{\max(\overline{u^2}_{test})}\,\mathrm{d}y \tag{3.7}$$

These errors are absolute deviations of the predicted mean profiles from the true mean profiles, scaled to the maximum value in the corresponding true profile.

3.2.2 Test set

Profiles of the mean quantities can vary for different time series. This is illustrated in Figure 3.4 (left) for $\overline{u^2}$ obtained from 50 different time series. This variance can be attributed to the insufficiency of the length of each series $(T = 4000)$. Hence, it is better to evaluate MLP predictions using a selected number of time series called the *test set*. For consistent evaluation of different neural network architectures and parameters, this test set is maintained constant. In order to find the optimal number of test series required for convergence of the flow statistics, the profiles obtained from multiple time series are ensemble-averaged. Figure 3.4 (right) shows the ensemble-averaged $\overline{u^2}$ profiles for 200, 500 and 1000 series. Since the difference in using more than 500 series is negligible, this number is chosen as the optimal number of series in the test set. The errors $\epsilon_{t,k}$, ϵ_U and $\epsilon_{\overline{u^2}}$ evaluated on the ensemble-averaged profile are denoted by $E_{t,k}$, E_U and $E_{\overline{u^2}}$ respectively.

3.3 Predictions using MLP

The first hurdle encountered in training deep learning models is the choice of architecture, *i.e.* the number of hidden layers and units to be used. Since the chosen problem is vastly different from typical learning tasks in the field of computer science, it is not straight-forward to choose a base architecture for initial predictions. In this section, a methodical approach is followed to arrive at a base architecture. This

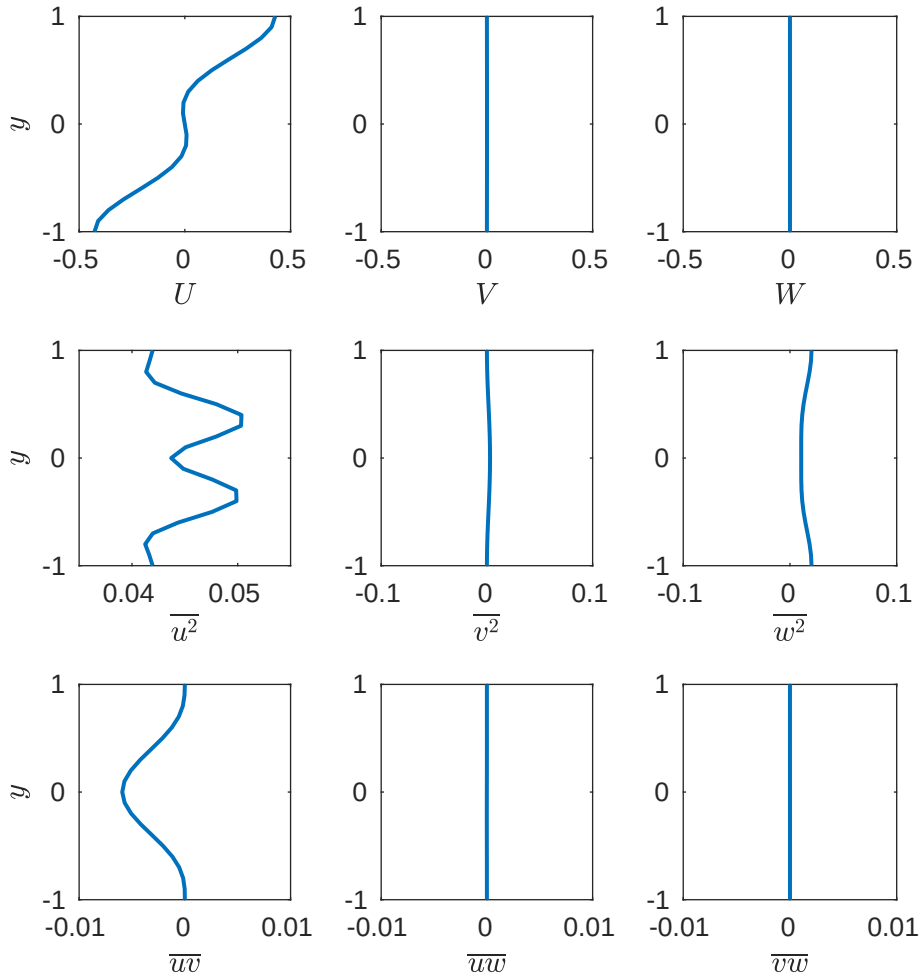


Figure 3.3: The profiles of mean velocities and pairwise Reynolds stresses for a sample time series.

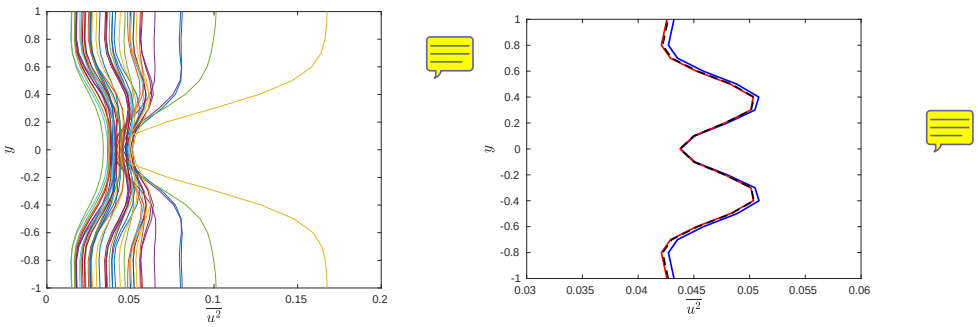


Figure 3.4: Left: The $\overline{u^2}$ profiles for 50 different time series. Right: Ensemble-averaged $\overline{u^2}$ profiles using 200 (blue), 500 (red), and 1000 (black dashed) time series.

architecture is trained and used to make the first predictions. These predictions are then analyzed in order to suggest improvements to the MLP model.

3.3.1 Choice of base architecture

The approach to determine a base architecture is to gradually increase the input units and hidden layers until the network accurately fits a single time series for the nine amplitudes. To achieve this, an MLP is trained for multiple epochs with a single dataset, and later used to predict the same dataset. Performing well on training data is a necessary condition to be satisfied by an MLP. For this experiment, the number of units in the hidden layers is set to a constant $n_i = 90$ for $i = \{1, 2, \ldots l\}$, and a hyperbolic tangent is used as the activation function in all layers. A coarse grid search is performed in the space of prediction order p and the number of hidden layers l. Table 3.1 shows the $\epsilon_{t,1}$ values for architectures with different p and l, as these values provide a sufficient measure of the goodness of the fit.

p	$l = 1$	$l = 2$	$l = 3$	$l = 4$	$l = 5$
10	23.34	33.66	25.37	26.01	23.81
50	28.28	20.59	17.66	16.77	22.95
100	18.97	13.58	12.76	17.17	14.42
500	65.21	3.32	6.80	1.43	3.09

Table 3.1: $\epsilon_{t,1}$ values (in %) for various p and l values

Table 3.1 clearly shows that a prediction order of $p = 500$ with at least two hidden layers is required to achieve good fits. The best fit is obtained for $l = 4$ layers, a configuration which is chosen as the base architecture. The predicted time series using this architecture, which has an $\epsilon_{t,1} = 1.43\%$ is shown in Figure 3.5 along with the true time series for all nine amplitudes.

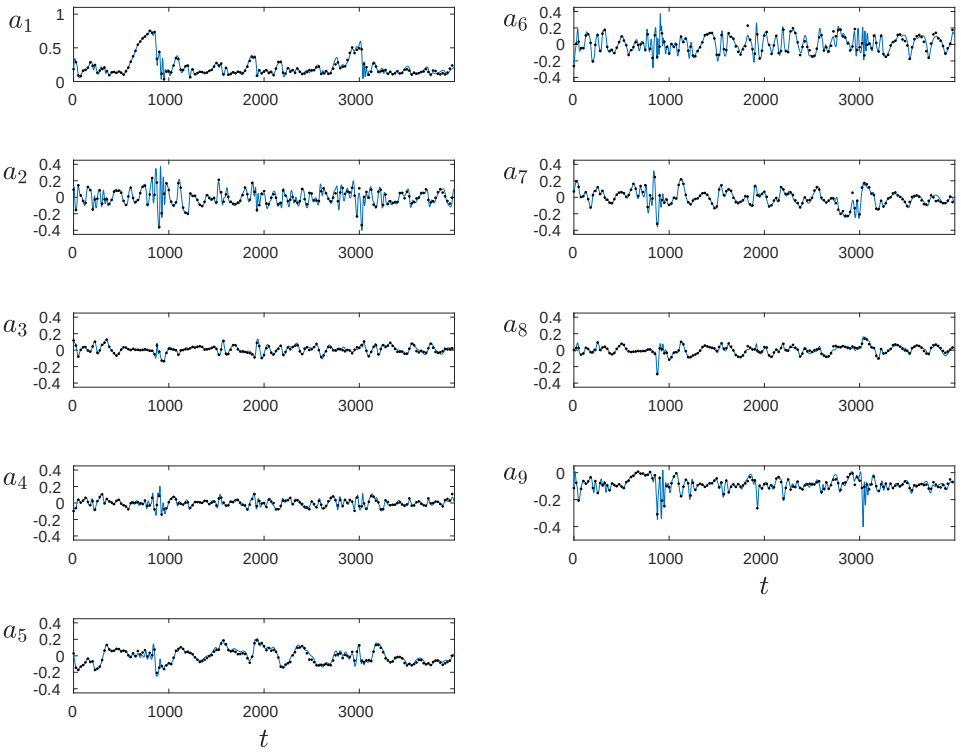


Figure 3.5: True (blue) and predicted (dotted black) time series using $p = 500$ and $l = 4$, which is chosen as the base architecture.

3.3.2 Predictions using the base architecture

The base architecture is defined by the following parameters: $l = 4$, $n_0 = 9 \cdot p = 4500$, $n_1 = 90$, $n_2 = 90$, $n_3 = 90$, $n_4 = 90$, $n_5 = 9$ and $g_i = \tanh, i = \{1, 2, \ldots 5\}$. This MLP is now trained with a larger dataset containing 100 time series and the training and validation (20% of the dataset) loss for each epoch is plotted in Figure 3.6. The plot shows the expected behavior with the training loss generally lower than the validation loss. Furthermore, the MLP model does not overfit as the

validation loss is observed to plateau after several epochs. Therefore, training is stopped when the validation loss does not decrease any further. Due to the noisy nature of these plots, the validation loss will henceforth be reported as the mean over the last ten epochs.

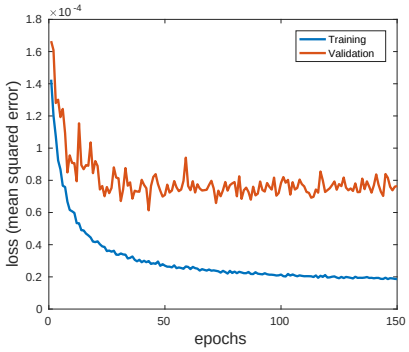


Figure 3.6: Training and validation loss history for the base architecture

The validation loss after training is the error in predicting the next sample in the time series. Although the achieved validation loss of 7.9×10^{-5} suggests an excellent fit, the predicted time series shows otherwise. For a single test sequence, the corresponding predicted sequence is shown in Figure 3.7. The $\epsilon_{t,k}$ errors in this prediction are listed in Table 3.2.

$\epsilon_{t,1}$	$\epsilon_{t,2}$	$\epsilon_{t,3}$	$\epsilon_{t,4}$	$\epsilon_{t,5}$	$\epsilon_{t,6}$	$\epsilon_{t,7}$	$\epsilon_{t,8}$	$\epsilon_{t,9}$
14.22	10.15	5.49	6.23	12.70	12.31	12.35	5.78	6.08

Table 3.2: $\epsilon_{t,k}$ values (in %) for the base architecture predictions.

There is a clear discrepancy between the validation loss during training and the $\epsilon_{t,k}$ values for the predictions. To further analyze this, while testing, each new MLP prediction is used as an initial condition for the ODE model to solve for the next time step. This way, a new time series is generated that contains the corresponding ODE solution for each MLP prediction. This new time series is shown along with the true and predicted time series for amplitude a_1 in Figure 3.8. This figure shows that the reason for the earlier discrepancy is the *propagation of errors* over each successive prediction. The MLP model coincidentally has the same nature as the ODE regarding the sensitivity to the

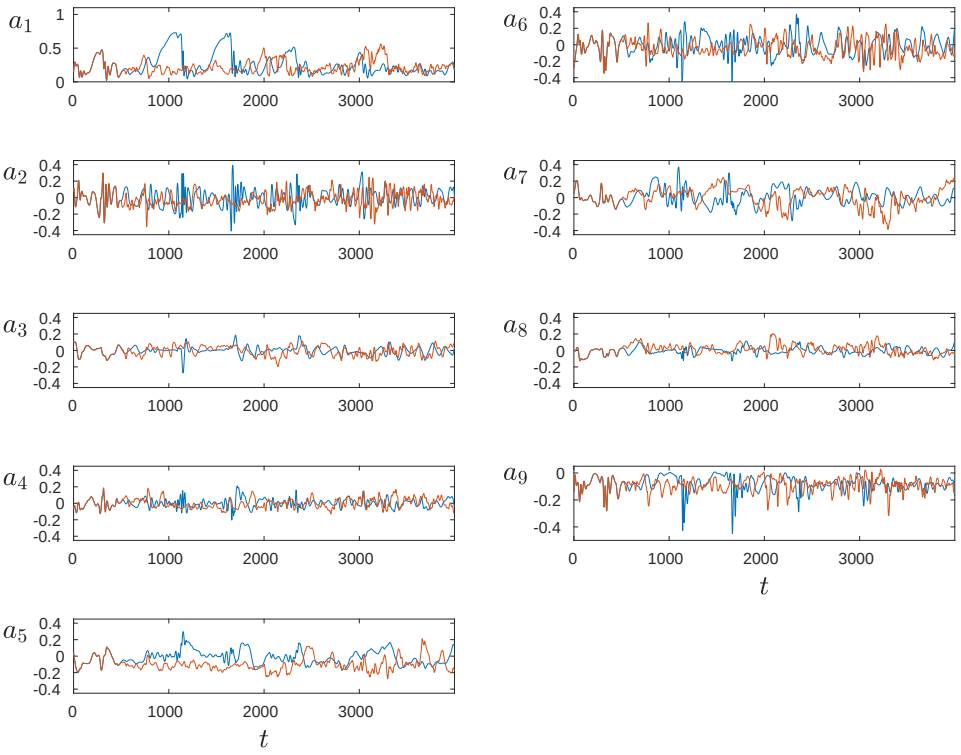


Figure 3.7: Time series predicted (red) using the base architecture based on an input sequence from the test series (blue)

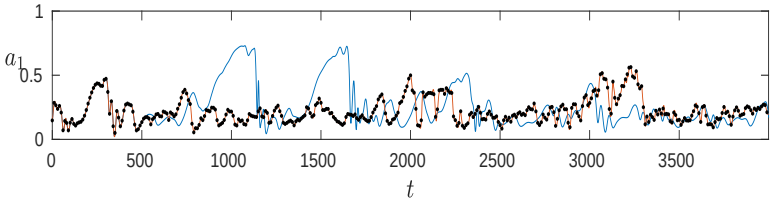


Figure 3.8: Assessment of error propagation during prediction: true (blue), predicted (red) and the ODE-based time series (dotted black).

previous state. A relatively small error of 10^{-5} is amplified to 10^{-1} after 200 successive predictions. It is difficult to infer the order of validation loss required in order to eliminate this discrepancy. Fortunately, fitting the ODE model accurately is only a secondary goal of the project as the primary goal is to simulate the turbulent behavior with the self-sustaining cycle. The velocity fields constructed using the predicted series are shown in Figure 3.9. These fields show similar streaks, vortices and instabilities as the ones observed in Figure 2.3.

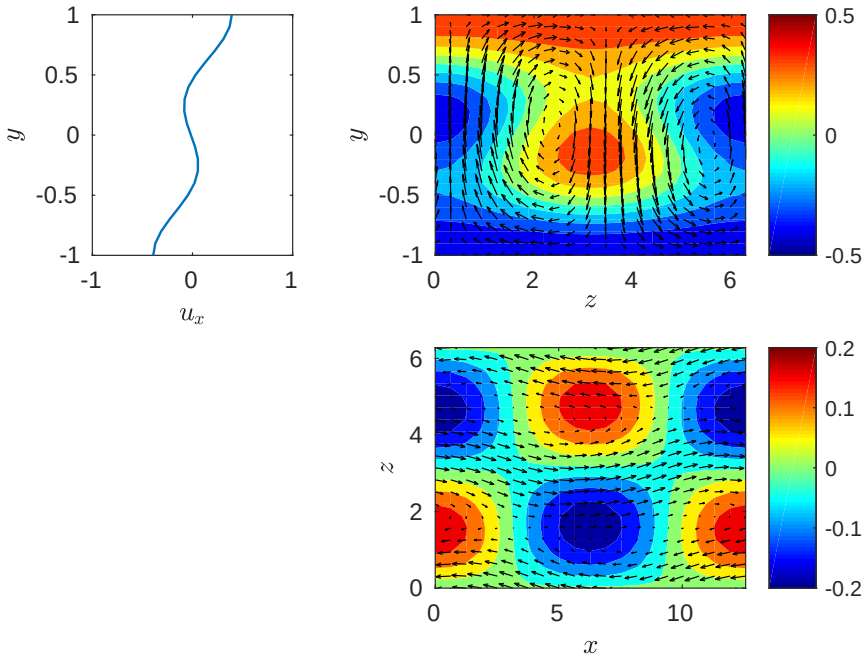


Figure 3.9: Velocity fields constructed from the MLP predictions in Figure 3.7 for $L_x = 4\pi$ and $L_z = 2\pi$ with $Re = 400$. The left panel shows the mean profile. The upper-right panel shows the spanwise flow and the lower-right panel shows the flow in the mid-plane.

The final comparison is based on the profiles of the mean quantities described in Section 3.2.1. The profiles of mean velocities and the Reynolds stresses for both test and predicted sequences is shown in Figure 3.10. These profiles yield $\epsilon_U = 6.21\%$ and $\epsilon_{\overline{u^2}} = 25.81\%$. Hence, despite capturing the self-sustaining process, the MLP model has a significant error in capturing the mean dynamics of the flow. Ensemble-average over 500 test series as described in Section 3.2.2 yields $E_U = 9.36\%$ and $E_{\overline{u^2}} = 35.06\%$. This presents an opportu-

nity for improving the MLP models by exploring different architectures and training with larger datasets. Since the $E_{t,k}$ values may not be improved, and the essential elements of the self-sustaining process appear to have been already captured (Figure 3.9), the room for improvement lies solely in the errors E_U and $E_{\overline{u^2}}$. Henceforth, these two errors will be the key metrics in evaluating all neural network models. However, $E_{t,1}$ (also based on ensemble average of 500 time series) and the validation loss will be provided along with these metrics to show that they do not vary significantly over different models.

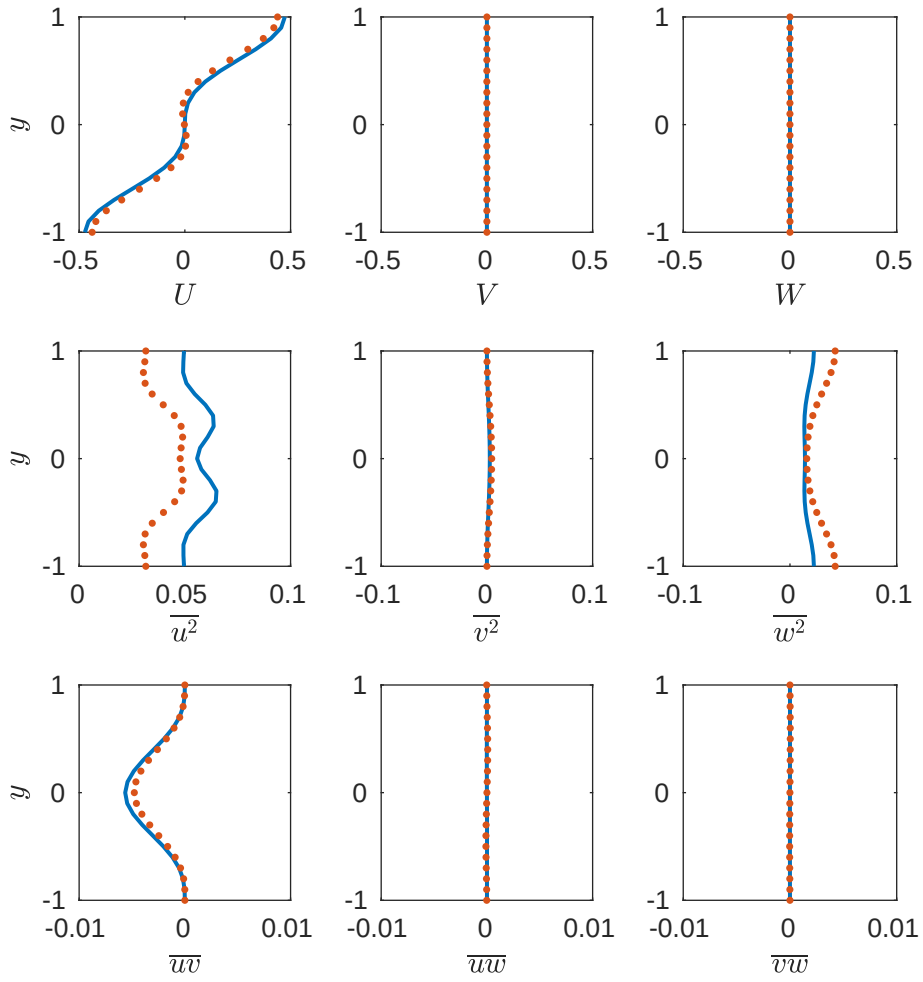


Figure 3.10: The profiles of mean velocities and pairwise Reynolds stresses for the test (blue) and predicted (dotted red) time series.

3.4 Weight initialization

Since the base MLP architecture has a large number of weights, the training algorithm seeks to find a local or global minimum in a high-dimensional space. The convergence of this search and its rate depends on the values these weights are initialized with. Although randomly choosing these weights based on a uniform distribution is a natural choice, there are a few other options that may perform better.

The results in the previous section were obtained using a random initialization from a uniform distribution in $[-0.5, 0.5]$. Since training a neural network involves a forward pass coupled with backpropagation, it is generally better to prevent alternating enlargement or shrinkage of layer outputs. Hence, initializations based on a suitable normal distributions perform better than that based on uniform distributions. Truncated normal, which bounds the random variable both above and below, is a commonly used distribution for most learning tasks. Glorot and Bengio [9] and He et al. [14] proposed initializers based on the normal distribution that showed better convergence properties than the truncated normal. Base architecture is trained with different initializations for the weights using the same dataset containing 100 time series. These trained models are compared based on their prediction errors listed in Table 3.3. The Glorot initialization [9] is observed to perform better than the other initializations with the lowest E_U and $E_{\overline{u^2}}$ values (recall that these are obtained based on the ensemble-average of 500 predicted time series), and is henceforth used as the default initializer. However, its convergence properties are similar to those of the uniform and truncated normal distributions, as indicated by the validation loss.

Initializer	E_U (%)	$E_{\overline{u^2}}$ (%)	$E_{t,1}$ (%)	Validation Loss
Uniform	9.36	35.06	10.46	7.9×10^{-5}
Truncated normal	9.33	32.52	9.80	8.4×10^{-5}
Glorot [9]	5.88	22.67	11.27	7.8×10^{-5}
He [14]	16.62	46.84	9.45	2.7×10^{-4}

Table 3.3: A comparison of different weight initializers (distributions used for random sampling).

3.5 Activation functions

So far, $\tanh$ has been used as the activation function in all layers of the MLP. However, the necessity of several layers of nonlinearity provided by $\tanh$ is yet to be analyzed. Using *almost* linear activation functions in the hidden layers can significantly reduce the training time while offering similar or better predictions. Activation functions such as the rectified linear unit (ReLU), the Leaky ReLU and the exponential linear unit (eLU) have dominated the field of deep learning in computer vision and speech recognition [26]. These functions and their corresponding graphs are listed in Table 3.4.

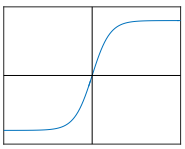
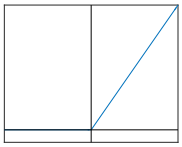
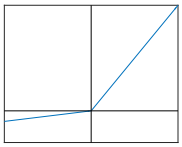
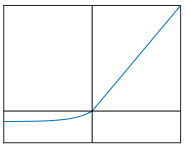
tanh		$f(x) = \tanh(x) = \frac{e^x - e^x}{e^{-x} + e^x}$
ReLu		$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$
Leaky ReLu		$f(\beta, x) = \begin{cases} \beta x & x < 0 \\ x & x \geq 0 \end{cases}$
eLu		$f(\beta, x) = \begin{cases} \beta(e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases}$

Table 3.4: Activation functions and their corresponding graphs and equations.

For the chosen problem, using a linear unit in the output layer fails to bound the prediction error. Hence $\tanh$ is always used in the output layer, since it restricts the amplitudes in the desired interval $[-1, 1]$. Therefore, the linear units listed in Table 3.4 are used only for the hidden layers in the base architecture, and the predicted results are tabulated in Table 3.5. The table confirms the necessity of multiple $\tanh$

layers for this specific problem based on its lower errors.

Activation	E_U (%)	$E_{\overline{u^2}}$ (%)	$E_{t,1}$ (%)	Validation Loss
tanh	5.88	22.67	11.27	7.8×10^{-5}
ReLu	17.64	36.67	10.14	4.3×10^{-5}
Leaky ReLu	20.66	102.92	13.03	4.4×10^{-5}
eLu	12.69	40.32	9.76	4.1×10^{-5}

Table 3.5: A comparison of different activation functions in the hidden layers.

3.6 Architecture dependence on training data size

In this section, the necessary size of the training data is explored for different MLP architectures. These architectures are specifically chosen in order to study the effects of increasing and decreasing the number of hidden layers and units in the base architecture. The studied architectures are listed in Table 3.6. Note that, in the table, the label A3 refers to the base architecture. The architectures A2, A3 and A4 differ in the number of layers, while A1, A3, and A5 differ in the number of units in their 4 hidden layers. The architectures are labeled in the increasing order of parameters contained.

Label	l	$n_i, i = \{1, \dots l\}$
A1	4	45
A2	3	90
A3	4	90
A4	5	90
A5	4	180

Table 3.6: Analyzed MLP architectures and their labels.

The five architectures listed in Table 3.6 are trained with three different sets of data, each containing 25, 100 and 1000 time series. Glorot initialization, and $\tanh$ activation functions are used while training. The prediction errors for all architectures trained with 25, 100 and 1000 datasets are presented in Tables 3.7, 3.8 and 3.9 respectively.

Architecture	E_U (%)	$E_{\overline{u^2}}$ (%)	$E_{t,1}$ (%)	Validation Loss
A1	4.88	23.68	11.27	2.94×10^{-4}
A2	9.11	41.42	8.91	2.61×10^{-4}
A3	16.52	49.01	9.01	2.31×10^{-4}
A4	25.31	67.04	11.43	2.52×10^{-4}
A5	14.93	52.20	9.02	2.80×10^{-4}

Table 3.7: A comparison of MLP architectures trained with 25 datasets.

A quick glance at these tables reveals that the validation losses are similar for all five architectures when trained with identical datasets, and these losses decrease as the size of data is increased. Furthermore, there are no observable trends in the $E_{t,1}$ values, as all architectures generate time series with $8 - 13\%$ error in the first coefficient a_1. In Table 3.7, the observed trend in the E_U and $E_{\overline{u^2}}$ values of A2, A3 and A4 is misleading as the training data is insufficient for all architectures except for A1. The sufficiency of data for A1 in all three training sets is shown by its consistent prediction errors in the three tables.

Architecture	E_U (%)	$E_{\overline{u^2}}$ (%)	$E_{t,1}$ (%)	Validation Loss
A1	1.28	29.26	10.04	7.77×10^{-5}
A2	6.60	21.68	12.01	8.27×10^{-5}
A3	5.88	22.67	11.27	7.8×10^{-5}
A4	4.65	19.15	10.97	8.05×10^{-5}
A5	18.18	59.19	9.65	7.81×10^{-5}

Table 3.8: A comparison of MLP architectures trained with 100 datasets.

When trained with the moderately-sized training data of 100 datasets (Table 3.8), the predictions using A2, A3 and A4 improve considerably. Moreover, a trend can be observed between these architectures, since the errors decrease negligibly as the number of layers are increased. However, this data is still insufficient for A5, which has high prediction errors.

Interesting conclusions may be drawn from Table 3.9, in which all architectures are trained with sufficient data and any existing trends can be clearly observed. The decrease of errors with the increase in number of layers observed in A2, A3 and A4 is more pronounced. Increasing the number of layers from 3 to 5, reduces the $E_{\overline{u^2}}$ error ap-

Architecture	E_U (%)	$E_{\overline{u^2}}$ (%)	$E_{t,1}$ (%)	Validation Loss
A1	1.84	24.91	12.60	3.96×10^{-5}
A2	10.96	36.16	11.76	4.38×10^{-5}
A3	7.00	29.04	9.90	3.90×10^{-5}
A4	3.21	18.61	12.89	3.84×10^{-5}
A5	5.87	27.85	11.88	3.99×10^{-5}

Table 3.9: A comparison of MLP architectures trained with 1000 datasets.

proximately in half. One the other hand, a comparison of A1, A3 and A5 lacks any observable trends. Despite that fact, it is safe to conclude that increasing the number of hidden units from 45 has negligible impact. Among the compared architectures, A4 with an $E_{\overline{u^2}} = 18.61\%$ exhibits the best predictions, suggesting that increasing the number of layers leads to a better performance. This also motivates the use of recurrent neural networks (discussed in the next section), which use a deep layered representation to develop a notion of *time* from the input data.

Chapter 4

Long Short-term Memory

Recurrent neural networks (RNNs) are neural networks specialized for learning from sequential data. These networks have a much closer resemblance to biological neural networks, since they learn not only static information but also the context in which it is presented. This is accomplished by the *internal state* of an RNN, which acts as a memory buffer to retain the information processed previously. RNNs are widely used for tasks involving sequential data such as machine translation, handwriting and speech recognition [12, 35]. RNNs are naturally more suited for time series forecasting than the standard MLP.

In principle, the training process of an RNN is similar to that of an MLP, with slight modifications applied to the loss function and the backpropagation algorithm. However, basic RNNs are generally harder to train than a standard MLP due to the so-called *problem of vanishing gradients* described below. Although this problem exists also for MLPs, it is more pronounced for RNNs. To overcome this problem, several variants of RNN have been developed. A popular variant called the long short-term memory network (LSTM) is employed in this thesis.

In this chapter, the general theory behind a basic RNN is presented along with the motivation for LSTMs. LSTM models are constructed for the chosen problem of predicting the amplitudes of Fourier modes, and evaluated based on the metrics described in Chapter 3.

4.1 RNNs and vanishing gradients

In contrast to a typical MLP, an RNN maps a sequence of data points to another sequence using a recurrence evaluation in time as outlined in Algorithm 3. A standard RNN is a neural network containing a single hidden layer with a feedback loop. The output of the hidden layer in the previous time instance is fed back into the hidden layer along with the current input as illustrated in Figure 4.1. An RNN can be thought of as a very deep MLP with one layer for each instance in the input sequence. This sequential feeding of the input data makes the temporal relationship apparent to the network.

Algorithm 3: Compute the output sequence of an RNN

Input: Sequence $x_1, x_2, \ldots x_p$
Output: Sequence $z_1, z_2, \ldots z_p$
set $h_0 \leftarrow 0$
for $t \leftarrow 1$ **to** p **do**
  $h_t \leftarrow g_h(W_{hx}x_t + W_{hh}h_{t-1} + b_h)$
  $z_t \leftarrow g_z(W_{zh}h_t + b_z)$

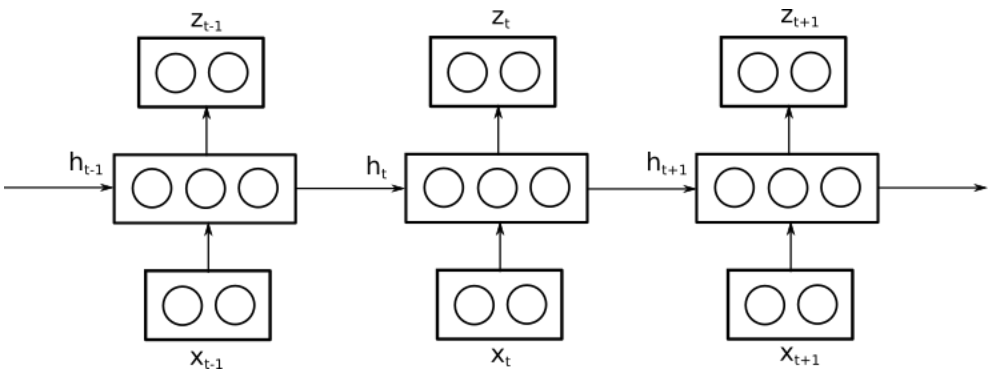


Figure 4.1: The basic RNN architecture.

An RNN is parametrized by the three weight matrices (W_{hx}, W_{hh} and W_{zh}), and the two biases (b_h and b_z). In the Algorithm 3, g_h and g_z are the hidden and output activation functions respectively. The loss function for an RNN for a single training example is the sum over p time steps of the squared error, where p is the prediction order or the

length of the input sequence. The weights and biases are computed by backpropagation through time [45].

RNNs are particularly difficult to train especially for sequences with long-range temporal dependencies. Since the backpropagation algorithm uses the chain rule to compute the weight corrections, small gradients in the later layers curtails the rate of learning of earlier layers. For long sequences, this gradient can become vanishingly small, a fact that prevents the neural network from learning any further. This is called the problem of vanishing gradients and it was first described by Hochreiter [16] and Bengio, Simard, and Frasconi [4]. In 1997, LSTM was proposed by Hochreiter and Schmidhuber [17] as a solution to the problem of vanishing gradients.

4.2 LSTM Networks

The key to an LSTM unit or a *memory cell* is its so-called cell state, denoted for the current time instance as C_t. Information can be added or removed from the cell state using regulating structures called *gates*. The gates are made of a logistic sigmoid function (σ) that acts as a smoothstep from 0 (off) to 1 (on), followed by an element-wise multiplication. The *forget gate* (Equation (4.1)) selects the information to be removed from the cell state. The input gate (Equation (4.2)) allows new information to be added to the cell state, and the output gate (Equation (4.5)) filters the cell state for the output. A visual representation of an LSTM unit is shown in Figure 4.2. A forward pass through an LSTM unit is governed by the following equations:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f), \tag{4.1}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \tag{4.2}$$

$$\widetilde{C}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c), \tag{4.3}$$

$$C_t = f_t \circ C_{t-1} + i_t \circ \widetilde{C}_t, \tag{4.4}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \tag{4.5}$$

$$h_t = o_t \circ \tanh(C_t), \tag{4.6}$$

where the symbol $\circ$ denotes element-wise multiplication. Since these evolution equations are sophisticated, the algorithm for training an LSTM network is highly complex. Fortunately, LSTMs can be trained with relative ease using specialized learning libraries such as Google's TensorFlow [1].

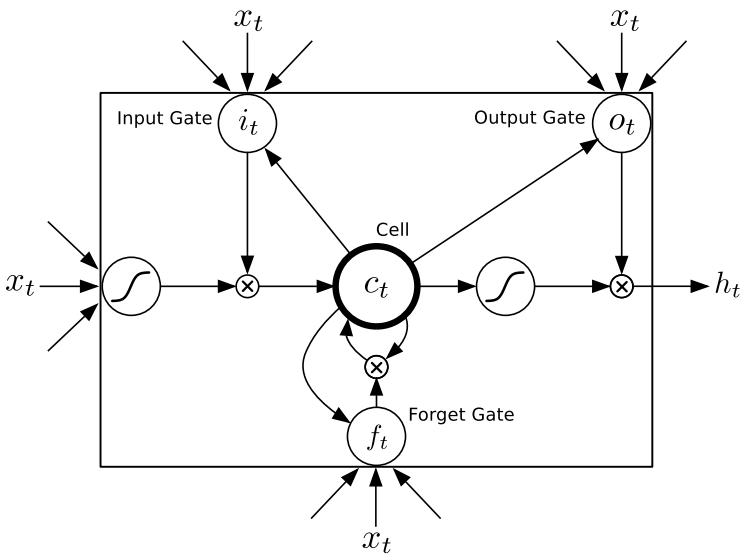


Figure 4.2: A single LSTM cell (Figure extracted from Ref. [11])

4.3 Predictions using LSTM

An LSTM layer comprises of several LSTM units. Since it is not theoretically clear whether stacking multiple layers of LSTMs yields a better performance [10], this study is restricted to a maximum of two LSTM layers. The output of the final LSTM layer is fed to an output layer which is the same as that used in the MLP models in Chapter 3. The output layer has 9 units with a $\tanh$ activation function. Hence, an LSTM architecture is fully defined by the prediction order p, the number of units in the first LSTM layer n_1, and if used, the number of units in the second LSTM layer n_2.

An LSTM architecture defined by $p = 10$ and $n_1 = 90$ is used for the initial investigation. Training this architecture with a dataset containing 100 time series produces the loss plot shown in Figure 4.3. The improvement over MLP models is instantly apparent from the losses,

which are three orders of magnitude lower. Moreover, the prediction order is 50 times lower than that used in the MLP model.

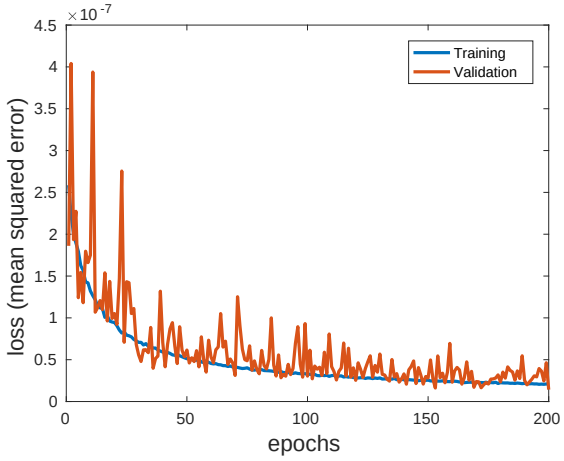


Figure 4.3: Training and validation loss history using LSTM

However, this improvement has no impact on the $\epsilon_{t,k}$ values of the predicted time series using this LSTM model, as shown in Figure 4.4. A similar analysis to that done in Figure 3.8 confirms that this is indeed due to the propagation of errors as shown in Figure 4.5. The validation loss is therefore not sufficiently small to eliminate this propagation. The velocity fields constructed using the predicted series are shown in Figure 4.6. These fields show similar streaks, vortices and instabilities as the ones observed in Figures 2.3 and 3.9.

4.4 Analysis of LSTM architectures

This section contains an analysis of different LSTM architectures and training data sizes in order to improve the ensemble-averaged errors E_U and $E_{\overline{u^2}}$. The value of these errors for the LSTM architecture considered above are shown in Table 4.1. Two other architectures, with one containing a second layer of LSTM units and the other using a longer prediction order, are also considered. The number of units in each LSTM layer is maintained at 90, as changing this parameter is not expected to yield any improvements based on the analysis in Section 3.6. The chosen architectures are trained with 100 datasets, and the

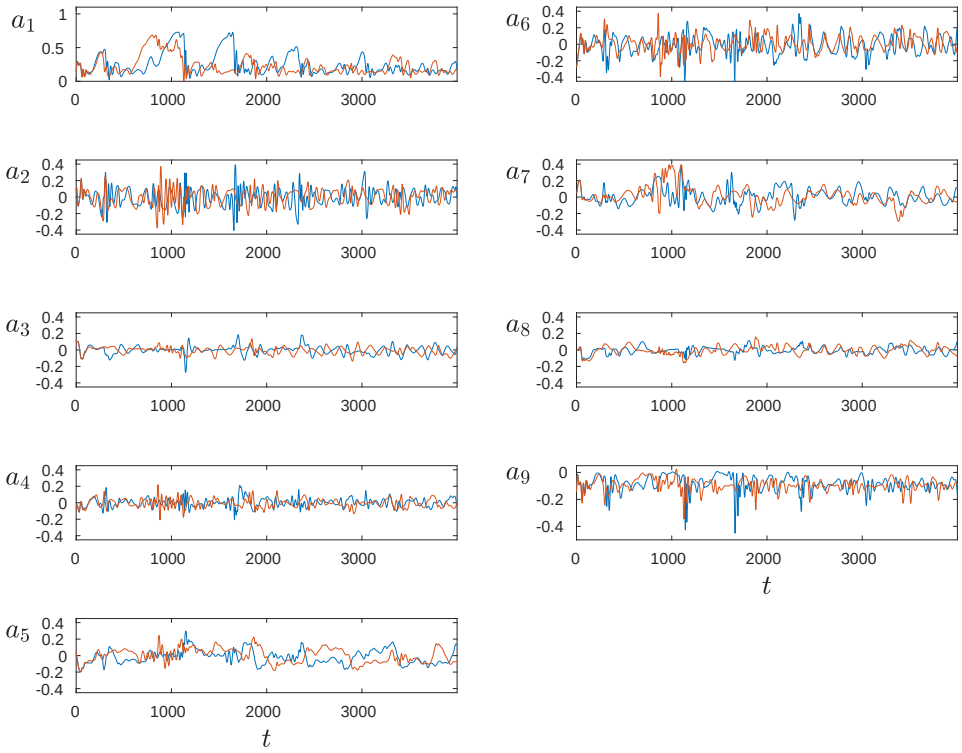


Figure 4.4: Time series predicted (red) using LSTM based on the an input sequence from the same test series (blue) used in Chapter 3.

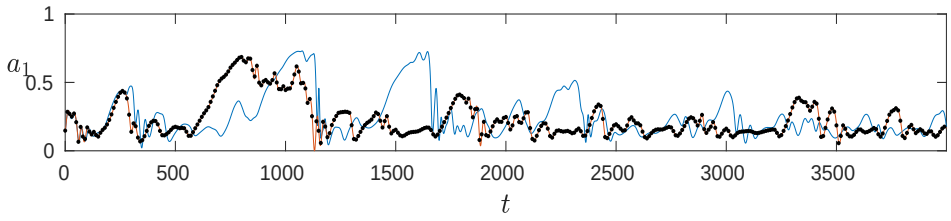


Figure 4.5: Assessment of error propagation for LSTM predictions: true (blue), predicted (red) and the ODE-based time series (dotted black).

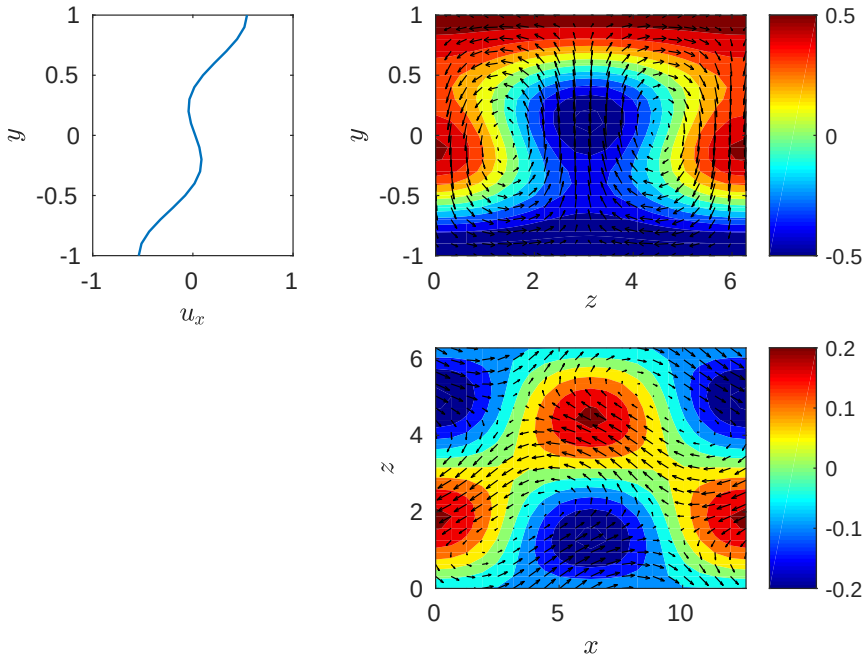


Figure 4.6: Velocity fields constructed from the LSTM predictions in Figure 4.4 for $L_x = 4\pi$ and $L_z = 2\pi$ with $Re = 400$. The left panel shows the mean profile. The upper-right panel shows the spanwise flow and the lower-right panel shows the flow in the mid-plane.

corresponding errors in prediction are presented in Table 4.1. Both of the new architectures predict with improved accuracy, at the cost of longer training time.

p	n_1	n_2	E_U (%)	$E_{\overline{u^2}}$ (%)	$E_{t,1}$ (%)	Validation Loss
10	90	-	2.36	14.73	12.08	2.0×10^{-8}
10	90	90	1.94	6.82	12.19	2.4×10^{-8}
25	90	-	1.72	7.40	11.33	5.2×10^{-8}

Table 4.1: A comparison of different LSTM architectures trained with 100 datasets.

The advantage of using a larger training set is noticeable in Table 4.2. This table contains the results of training an LSTM with $p = 10$ and $n_1 = 90$ using $1,000$ and $10,000$ datasets. The results using $1,000$ training datasets are excellent with a $E_{\overline{u^2}}$ value of 3.44%; this is further improved to 2.49% when using $10,000$ datasets for training. This LSTM model sufficiently captures the dynamics of the simplified turbulent flow, and the mean profiles of the velocities and the Reynolds stresses based on an ensemble average of 500 test and predicted series are shown in Figure 4.7. Note that, the trend in validation loss for increasing data size is similar to that observed previously in Section 3.6. Since the chosen LSTM architecture yields improved predictions when trained with $10,000$ datasets, the other architectures listed in Table 4.1 may be expected to yield superior results for the same training set.

Datasets	E_U (%)	$E_{\overline{u^2}}$ (%)	$E_{t,1}$ (%)	Validation Loss
100	2.36	14.73	12.08	2.0×10^{-8}
1,000	0.83	3.44	12.80	8.5×10^{-9}
10,000	0.45	2.49	13.08	5.2×10^{-9}

Table 4.2: LSTM with $p = 10$ and $n_1 = 90$ trained with different numbers of datasets.

The superiority of LSTMs over MLPs in time series prediction is evident from the following conclusions: (i) LSTM requires a prediction order 50 times lower, and therefore fewer parameters are required for consistent predictions; (ii) the validation losses in LSTM are 4 orders of magnitude lower; and (iii) the prediction of simplified turbulent flow dynamics with LSTM exhibits a remarkable accuracy, with 0.45% error in the mean flow and 2.49% in the streamwise velocity fluctuations.

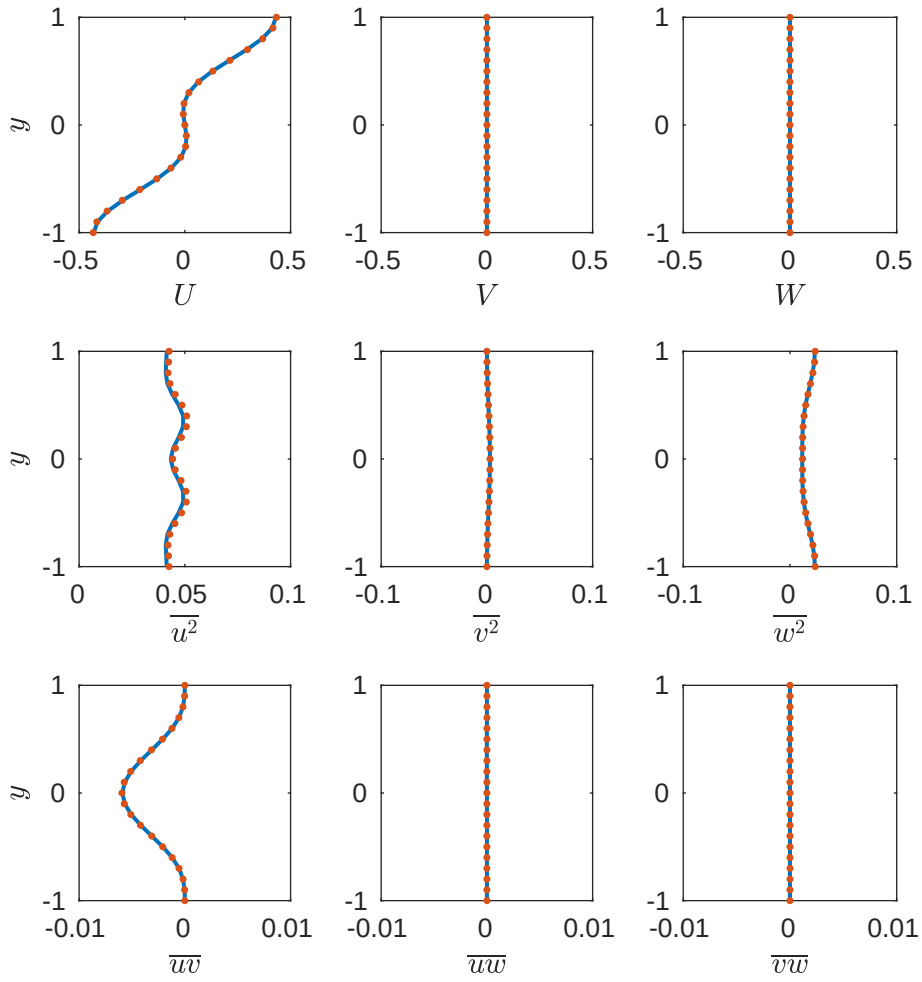


Figure 4.7: The ensemble-averaged profiles of mean velocities and pairwise Reynolds stresses over the test (blue) set and the LSTM predicted (dotted red) set.

Chapter 5

Conclusions and discussion

Neural networks, especially LSTM, are well-known for their ability to "hallucinate", *i.e.* generate arbitrary output sequences regardless of the input sequence provided [11, 37]. This ability can be both an advantage and a disadvantage. On the one hand, it demonstrates the fact that the neural network has learned the important features from the dataset and attempts to reproduce it. On the other hand, it indicates a potential disregard to the input sequence. This disadvantage renders the task of generalizing a neural network for an entire class of problems challenging.

This concluding chapter of the thesis probes into the nature of a neural network model and its ability to hallucinate. To avoid redundancy, only the LSTM model with $p = 10$ and $n_1 = 90$ is considered. This analysis is followed by comments on the future direction of this research, and preliminary results in that direction.

5.1 Analysis of LSTM predictions

To ensure that the predicted time series are not arbitrarily churned, and are consistent with the ODE-generated sequences, the following analysis tools are used: (i) Poincaré maps, which are commonly used to study the state-space of dynamical systems that exhibit regular or chaotic periodic orbits; and (ii) power density spectra, which are used in signal processing to study the distribution of power among the frequency components of a time series.

A Poincaré map for the time series of amplitudes is obtained by representing the intersections of the flow with the hyperplane $a_2 = 0$

on the a_1-a_3 state space. These intersections further have the condition $\dot{a}_2 < 0$ (where $\dot{a}_2$ denotes the time derivative of the coefficient a_2). Since these maps retain the periodic properties of the original system, a lower-dimensional state space is sufficient and convenient for analysis and comparison. The Poincaré maps constructed using the entire test set and the corresponding predicted set are shown in Figure 5.1. These maps are presented in the form of a frequency map, with the darker regions denoting a higher frequency of intersections. The considerable similarity between the maps for test and predicted sequences are further illustrated in Figure 5.2, which shows the contours of the density function of the intersections for the maps in Figure 5.1. This similarity affirms that the chaotic transience is not predicted in an arbitrarily haphazard manner.

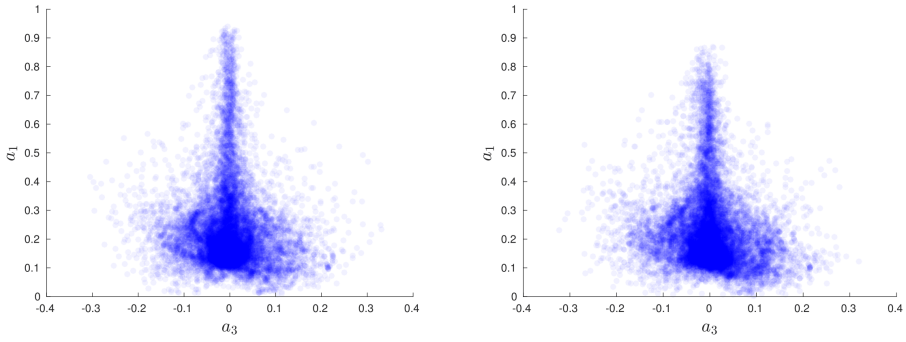


Figure 5.1: Poincaré maps of the time series in the test set (left) and the predicted set (right)

The power-spectral density of a time series is obtained by decomposing the series into a finite number of sine waves of discrete frequencies using fast Fourier transform. The amplitudes of these waves are the *powers* of the corresponding frequencies and a measure of their contribution to the original signal. Figure 5.3 shows the power density spectra of a single predicted time series along with the corresponding test series. It can be observed that the predicted series distributes power into discrete frequencies in a similar way as the test series.

These analyses confirm that the neural network hallucinates in an expected way. However, the sensitivity of the network to input sequences remains to be examined. Figure 5.4 shows the network prediction for a randomly generated input sequence that is not consistent with the ODE model. The erratic behavior of the neural network

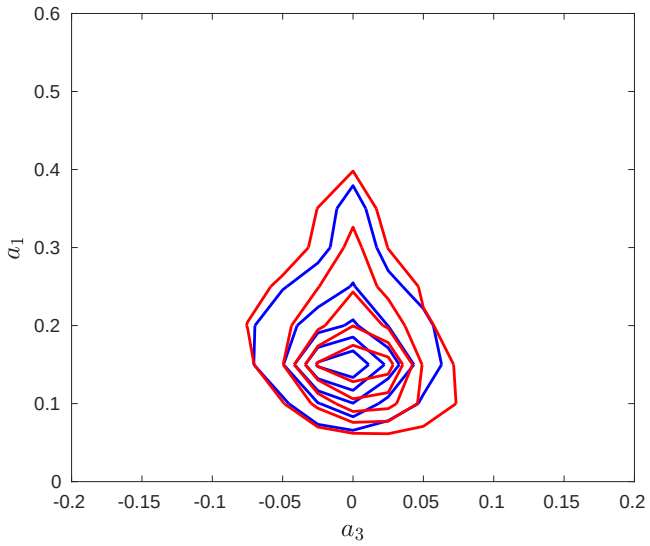


Figure 5.2: Density contours for the test (blue) and predicted (red) set

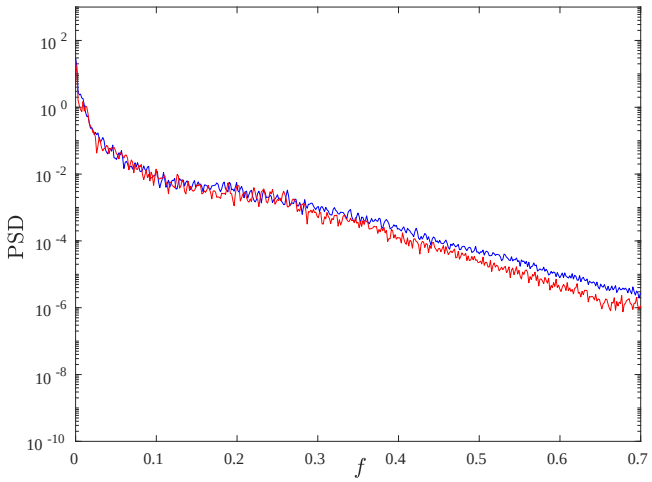


Figure 5.3: Power density spectra of coefficient a_1 from a test (blue) and predicted (red) series

model is an encouraging sign, since it is an indication that the model requires *meaningful* input sequences to perform well.

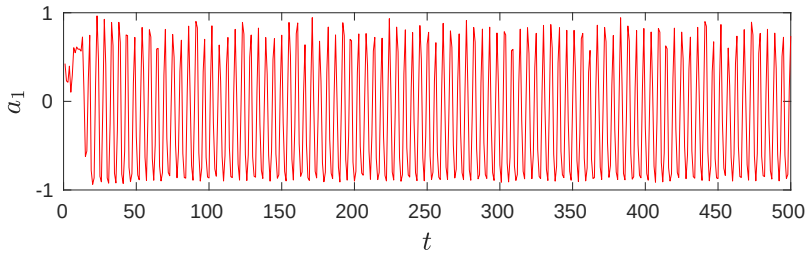


Figure 5.4: LSTM predictions for an inconsistent input sequence

5.2 Future Work

LSTMs have shown promising results in reproducing simplified turbulent flow behavior albeit using only 9 degrees of freedom. However, the ultimate objective is to directly predict entire velocity fields, which may have millions of degrees of freedom. Such predictions would involve exceptionally large networks trained for multiple days with several terabytes of data. Examples of turbulent flow databases which may be used for future training and prediction are found in Refs. [6, 40]. This lofty goal may be attainable with modest computational power by using advanced training strategies. However, with the available time and computational resources, a preliminary step towards this goal is taken. Streamwise velocity fields are extracted as time series for each point in the three-dimensional grid and used as data for training an LSTM. Due to practical constraints, the number of grid points in each coordinate direction was limited to 11. Therefore, each dataset contains $11 \times 11 \times 11$ time series. The prediction order was retained at $p = 10$, while the number of units in a single LSTM layer was increased to $n_1 = 1000$. The predicted mean profiles of U and $\overline{u^2}$ are shown in Figure 5.5. Although the prediction errors are high, the model has learned the sinusoidal nature of mean velocity. In addition, the predictions are symmetric about the midplane.

Deep learning has made rapid strides in video analysis, and has been successful in tasks such as video semantic segmentation, optical flow estimation, and video coloring [31, 39]. Deep 3D convolutional architectures have been trained to make voxel-to-voxel predictions in

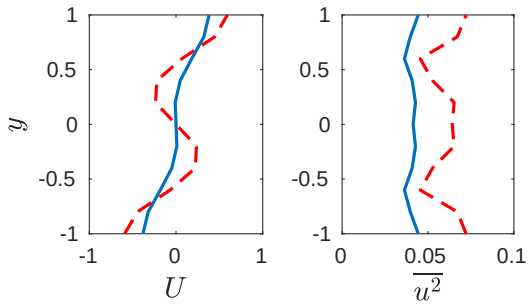


Figure 5.5: True (blue) and predicted (dashed red) mean profiles obtained from predictions using three-dimensional velocity fields

videos. Since velocity fields are essentially voxels representing a value in a regular grid in the three-dimensional space, 3D CNNs may have an important role to play in flow prediction. The rapid growth of deep learning promises exciting times ahead in turbulence modeling.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Nadine Aubry, Philip Holmes, John L Lumley, and Emily Stone. "The dynamics of coherent structures in the wall region of a turbulent boundary layer". In: *Journal of Fluid Mechanics* 192 (1988), pp. 115–173.

[3] Pierre Baldi and Kurt Hornik. "Neural networks and principal component analysis: Learning from examples without local minima". In: *Neural networks* 2.1 (1989), pp. 53–58.

[4] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[5] Peter Benner, Serkan Gugercin, and Karen Willcox. "A survey of projection-based model reduction methods for parametric dynamical systems". In: *SIAM Review* 57.4 (2015), pp. 483–531.

[6] Alexandra Bobke, Ricardo Vinuesa, Ramis Örlü, and Philipp Schlatter. "History effects and near equilibrium in adverse-pressure-gradient turbulent boundary layers". In: *Journal of Fluid Mechanics* 820 (2017), pp. 667–692.

[7] RM Clever and Fritz H Busse. "Tertiary and quaternary solutions for plane Couette flow". In: *Journal of Fluid Mechanics* 344 (1997), pp. 137–153.

[8] Bruno Eckhardt and Alois Mersmann. "Transition to turbulence in a shear flow". In: *Physical Review E* 60.1 (1999), p. 509.

[9] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[10] Yoav Goldberg. "A primer on neural network models for natural language processing". In: *Journal of Artificial Intelligence Research* 57 (2016), pp. 345–420.

[11] Alex Graves. "Generating sequences with recurrent neural networks". In: *arXiv preprint arXiv:1308.0850* (2013).

[12] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. "A novel connectionist system for unconstrained handwriting recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 31.5 (2009), pp. 855–868.

[13] Xiaoxiao Guo, Wei Li, and Francesco Iorio. "Convolutional neural networks for steady flow approximation". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2016, pp. 481–490.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[15] Oliver Hennigh. "Lat-Net: Compressing Lattice Boltzmann Flow Simulations using Deep Neural Networks". In: *arXiv preprint arXiv:1705.09036* (2017).

[16] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: *Diploma, Technische Universität München* 91 (1991), p. 1.

[17] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[18] Philip Holmes, John L Lumley, Gahl Berkooz, and Clarence W Rowley. *Turbulence, coherent structures, dynamical systems and symmetry*. Cambridge university press, 2012.

[19] David H Hubel and Torsten N Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of physiology* 160.1 (1962), pp. 106–154.

[20] Javier Jiménez and Parviz Moin. "The minimal flow unit in near-wall turbulence". In: *Journal of Fluid Mechanics* 225 (1991), pp. 213–240.

[21] Jack Kiefer and Jacob Wolfowitz. "Stochastic estimation of the maximum of a regression function". In: *The Annals of Mathematical Statistics* (1952), pp. 462–466.

[22] Lina Kim. "Transient growth for a sinusoidal shear flow model". MA thesis. University of California, Santa Barbara, 2005.

[23] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[24] Timo Koskela, Mikko Lehtokangas, Jukka Saarinen, and Kimmo Kaski. "Time series prediction with multilayer perceptron, FIR and Elman neural networks". In: *Proceedings of the World Congress on Neural Networks*. Citeseer. 1996, pp. 491–496.

[25] J Nathan Kutz, Steven L Brunton, Bingni W Brunton, and Joshua L Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems*. Vol. 149. SIAM, 2016.

[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), p. 436.

[27] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. "Reynolds averaged turbulence modelling using deep neural networks with embedded invariance". In: *Journal of Fluid Mechanics* 807 (2016), pp. 155–166.

[28] Michele Milano and Petros Koumoutsakos. "Neural network modeling for near wall turbulent flow". In: *Journal of Computational Physics* 182.1 (2002), pp. 1–26.

[29] Jeff Moehlis, Holger Faisst, and Bruno Eckhardt. "A low-dimensional model for turbulent shear flows". In: *New Journal of Physics* 6.1 (2004), p. 56.

[30] Jeff Moehlis, Holger Faisst, and Bruno Eckhardt. "Periodic orbits and chaotic sets in a low-dimensional model for shear flows". In: *SIAM Journal on Applied Dynamical Systems* 4.2 (2005), pp. 352–376.

[31] Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. "V2V-PoseNet: Voxel-to-Voxel Prediction Network for Accurate 3D Hand and Human Pose Estimation from a Single Depth Map". In: *arXiv preprint arXiv:1711.07399* (2017).

[32] M Nagata. "Three-dimensional finite-amplitude solutions in plane Couette flow: bifurcation from infinity". In: *Journal of Fluid Mechanics* 217 (1990), pp. 519–527.

[33] Stephen B Pope. *Turbulent flows*. Cambridge University Press, 2001.

[34] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), p. 533.

[35] Haşim Sak, Andrew Senior, and Françoise Beaufays. "Long short-term memory recurrent neural network architectures for large scale acoustic modeling". In: *15th annual conference of the international speech communication association*. 2014.

[36] Md Shiblee, Prem Kumar Kalra, and B Chandra. "Time series prediction with multilayer perceptron (MLP): a new generalized error based approach". In: *International Conference on Neural Information Processing*. Springer. 2008, pp. 37–44.

[37] Ilya Sutskever, James Martens, and Geoffrey E Hinton. "Generating text with recurrent neural networks". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 1017–1024.

[38] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. "Accelerating eulerian fluid simulation with convolutional networks". In: *arXiv preprint arXiv:1607.03597* (2016).

[39] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. "Deep end2end voxel2voxel prediction". In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2016 IEEE Conference on*. IEEE. 2016, pp. 402–409.

[40] Ricardo Vinuesa, Seyed M Hosseini, Ardeshir Hanifi, Dan S Henningson, and Philipp Schlatter. "Pressure-gradient turbulent boundary layers developing around a wing section". In: *Flow, Turbulence and Combustion* 99.3-4 (2017), pp. 613–641.

[41] Ricardo Vinuesa, Azad Noorani, Adrián Lozano-Durán, George K El Khoury, Philipp Schlatter, Paul F Fischer, and Hassan M Nagib. "Aspect ratio effects in turbulent duct flows studied through direct numerical simulation". In: *Journal of Turbulence* 15.10 (2014), pp. 677–706.

[42] Fabian Waleffe. "On a self-sustaining process in shear flows". In: *Physics of Fluids* 9.4 (1997), pp. 883–900.

[43] J Weatheritt and RD Sandberg. "The development of algebraic stress models using a novel evolutionary algorithm". In: *International Journal of Heat and Fluid Flow* 68 (2017), pp. 298–318.

[44] Andreas S Weigend, Bernardo A Huberman, and David E Rumelhart. "Predicting the future: A connectionist approach". In: *International Journal of Neural Systems* 1.03 (1990), pp. 193–209.

[45] Paul J Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[46] Cheng Yang, Xubo Yang, and Xiangyun Xiao. "Data-driven projection method in fluid simulation". In: *Computer Animation and Virtual Worlds* 27.3-4 (2016), pp. 415–424.

[47] Ze Jia Zhang and Karthikeyan Duraisamy. "Machine learning methods for data-driven turbulence modeling". In: *22nd AIAA Computational Fluid Dynamics Conference*. 2015, p. 2460.

TRITA -SCI-GRU 2018:236