A Machine Learning Approach to Unresolved-Scale Modeling for Burgers' Equation

Master of Science Thesis

by Michel Robijns

To obtain the degree Master of Science at Delft University of Technology,
defended publicly on Monday April 15, 2019 at 1:00 PM.

Thesis committee:	Dr. S. J. Hulshoff	TU Delft, supervisor
	Dr. R. P. Dwight	TU Delft
	Dr. B. Chen	TU Delft

Contents

1 Introduction

The numerical simulation of the Navier-Stokes equations is a challenging endeavor because small-scale motions have a profound effect on the large-scale flow behavior and cannot simply be ignored. In spite of the exponential increase in computational power, the computational cost of resolving all scales of motion in a direct numerical simulation (DNS) will remain prohibitive for the foreseeable future [1]. In contrast to DNS, large eddy simulation (LES) is a method where only the large scales of motion are resolved. In LES, the flow solution $\boldsymbol{u}$ is decomposed as $\boldsymbol{u} = \bar{\boldsymbol{u}} + \boldsymbol{u'}$ where $\bar{\boldsymbol{u}}$ denotes the large-scale solution and $\boldsymbol{u'}$ denotes the unresolved-scale solution. Rather than resolving all small-scale phenomena, the effects of $\boldsymbol{u'}$ on $\bar{\boldsymbol{u}}$ are represented by a so-called unresolved-scale model.

This thesis is part of a greater effort to apply machine learning to the development of flexible and universal unresolved-scale models. The novelty in the current work is training a neural network to directly predict the unresolved-scale terms without a priori assumptions on the underlying functional relationship. Within the variational multiscale framework [2], the computation of $\bar{u}$ does not require pointwise knowledge of $\boldsymbol{u'}$; instead, the model need only predict integral values of $\boldsymbol{u'}$, this is advantageous because $\boldsymbol{u'}$ is typically a highly erratic, chaotic signal.

Dutch physicist Jan Burgers first suggested using the equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{1.1}$$

as a one-dimensional model of turbulence where ν is a diffusion coefficient [3]. Burgers' equation can be used as a model of the Navier-Stokes equations because it reproduces several features that are expected of turbulence, such as the formation and decay of weak shocks in a compressible fluid and viscous dissipation at small length scales [4]. The contribution of this thesis is an implementation and validation of a neural-network-based unresolved-scale model for Burgers' equation, which paves the way for future application to the Navier-Stokes equations.

1.1 The Big Picture

There is a clear consensus that LES of high Reynolds-number wall-bounded flows can only be performed by resolving the outer layer only [5, 6, 7, 8]. LES therefore requires a so-called wall model that represents the effects of the unresolved motions in the inner layer on the large-scale flow. Numerous wall models have been proposed since the conception of LES in the 1960's, but to date, none of these models have proven to be universally applicable.

Three recent developments have created new opportunities for the development of wall models: (1) the exponential increase in computational power, (2) the rise of machine learning, and (3)

newfound universalities in wall-bounded flows. The exponential increase of computational power has enabled DNS of wall-bounded flows at $\mathrm{Re}_\tau = O(10^3)$ and it is estimated that $\mathrm{Re}_\tau = O(10^4)$ will be achievable around the year 2020 [9]. While problems of engineering interest will remain out of reach for DNS, the capabilities of DNS are sufficient to capture near-wall behavior. So-called "very large scale motions" or VLSMs control this near-wall behavior which therefore has a modelable character [8]. It is hypothesized that neural networks can devise wall models by learning from the interactions between VLSMs and the near-wall behavior. The intention is to resolve VLSMs in an LES to provide inputs to a neural network.

There are estimates that a universal wall model can reduce the computational cost of simulating wall-bounded flows by 99% [6]. If these estimates are true, then the time consumption and the computational cost of the numerical simulation of turbulent flows could be reduced to a fraction of what it is today. Such developments would likely have a strong impact on industries that rely on computational fluid dynamics (CFD). The aerodynamic design process of aircraft, motor vehicles, wind turbines, etc. would be quicker and far less costly. It is also plausible that the performance and efficiency of transportation systems would increase considerably if engineers could focus their design efforts on producing laminar flow. It is estimated that "up to half of the fuel burned by a modern airliner during flight is used to overcome drag due to turbulent boundary layers" [7], so there certainly is vast potential for improvement.

1.2 Data-Driven Turbulence Modeling

The adoption of machine learning as a tool for turbulence modeling is steadily increasing. This data-driven approach can be used for developing new turbulence models as well as calibrating parameters of existing models. Duraisamy et al. [10] suggest a basic six-step procedure for the development of neural-network-based models:

1. Collect a set of flow solutions from a truth model (e.g., a DNS).
2. Extract a set of inputs and corresponding outputs from the flow solutions.
3. Construct a training set from the inputs and outputs of step 2.
4. Select a neural network architecture.
5. Train the neural network on the training set to establish a functional relationship between the inputs and outputs.
6. Embed the resulting model into a simulation and validate its predictions.

Tracey et al. [11] present a proof-of-concept of the above six-step procedure in which solutions from the Spalart-Allmaras turbulence model represent the truth model. The authors conclude their work with a number of useful recommendations:

- The treatment of data and scaling requires much care. Inputs and outputs that have compact data ranges are ideal because they discourage overfitting.
- All candidate turbulence models must be evaluated within a CFD solver. The authors find that the "testing error on individual data points is often a poor predictor of full flow solution quality".

The recommendations of Tracey et al. are followed in this thesis, particularly the last recommendation of testing the neural networks within an actual simulation.

1.3 Literature Review

Among the first attempts to use machine learning for turbulence modeling is the work of Milano and Koumoutsakos [12] published in 2002. Milano and Koumoutsakos train a neural network to predict velocity fields in the viscous sublayer based on wall-only information. The authors conclude that it is difficult to base models on wall-only information because pressure and shear stress at the wall are influenced by large-scale flow structures. Any attempt at the development of wall models should therefore incorporate large-scale flow information in addition to wall-only information. The work of Milano and Koumoutsakos predates the discovery of VLSMs and it is now known that there exist interactions that can be used to devise wall models [7].

Sarghini et al. [13] employ a neural network as unresolved-scale model in an LES. The authors target a computational cost reduction rather than finding a new or improved functional form. To this end, a neural network is trained to predict the turbulent viscosity coefficient c_s that is part of Bardina's scale similar unresolved-scale model [14]. Sarghini et al. show that their model yields a computational time saving of about 20% without compromising the accuracy of Bardina's model. The neural network is reliable at Reynolds numbers that correspond to the Reynolds numbers encountered in training, but switching to higher Reynolds numbers requires retraining the neural network. The authors therefore conclude that their model is not generalizing to new situations. Overfitting certain properties of the training examples prevents a model from generalizing because the network has memorized the training examples rather than learning a fundamental underlying relationship. Generalization and diagnosing overfitting are therefore high priorities in the current work.

In applications where the concentration of a chemical species is of interest, a scalar equation must be solved simultaneously with the governing flow equations. Vollant et al. [15] developed a neural-network-based unresolved-scale model for LES of such a passive scalar. The authors conclude that their procedure "leads to an accurate SGS model, as shown by comparison with classic SGS models in an a posteriori test". Vollant et al. deliberately choose a new problem in the a posteriori test and are therefore confident that their model has the ability to generalize. The idea of choosing a previously unseen test problem inspires the approach of this thesis.

In the work of Gamahara and Hattori [16], a neural network is used to establish a functional relationship between $\bar{\boldsymbol{u}}$ and the pointwise unresolved-scale stress tensor in LES. The authors conclude that a neural network can establish a functional relationship between $\bar{\boldsymbol{u}}$ and the unresolved-scale stress tensor, but with the caveats that: (1) the neural network only performs as well as the Smagorinsky model, and (2) training is only successful when the filter size is small. The results of Gamahara and Hattori show that developing a pointwise predictive model for $\boldsymbol{u}'$ is difficult. Modeling integral quantities in a variational multiscale framework is more likely to be successful as statistics of $\boldsymbol{u}'$ are easier to predict than $\boldsymbol{u}'$ itself.

1.4 Residual-Based Models

The variational multiscale method (VMM) by Hughes [2] is a method where the solution $\boldsymbol{u}$ is decomposed into a large-scale component $\bar{\boldsymbol{u}}$ and an unresolved-scale component $\boldsymbol{u}'$ by means of a projection onto a space of finite-element basis functions. Hughes [2] proceeds to show that $\boldsymbol{u}'$

can be expressed in terms of an element-wise Green's function:

$$\boldsymbol{u}'(\boldsymbol{x}) = -\int_{\Omega} \mathcal{G}(\boldsymbol{x}, \boldsymbol{y}) \mathcal{R}(\bar{\boldsymbol{u}})\, d\Omega \tag{1.2}$$

where $\mathcal{R}(\bar{\boldsymbol{u}})$ is the large-scale residual. Based on (1.2), the unresolved-scales $\boldsymbol{u}'$ are said to be driven by the large-scale residual. In other words,

$$\boldsymbol{u}' = f(\bar{\boldsymbol{u}}, \mathcal{R}(\bar{\boldsymbol{u}})) \tag{1.3}$$

Models that build onto this result are referred to as residual-based models and the idea of residual-based models can be used for the development of unresolved-scale models that are more powerful than the eddy viscosity model initially used by Hughes et al. [17]. A convenient approximation of the functional f in (1.3) is obtained by assuming that:

1. If $\mathcal{R}(\bar{\boldsymbol{u}}) = 0$, then $\boldsymbol{u}' = 0$.
2. If $\mathcal{R}(\bar{\boldsymbol{u}})$ is small, then $\boldsymbol{u}'$ is small.

Hughes [2] shows that these assumptions are physically reasonable and lead to the idea of computing stabilization operators, denoted $\boldsymbol{\tau}$. The unresolved-scales can then be approximated as the product of $\boldsymbol{\tau}$ and the large-scale residual, i.e.,

$$\boldsymbol{u}' \approx -\boldsymbol{\tau} \mathcal{R}(\bar{\boldsymbol{u}}) \tag{1.4}$$

Equation (1.4) is one of the simplest residual-based models. A more sophisticated model that takes the dynamics of the unresolved scales into account is

$$\boldsymbol{u}'_t + \boldsymbol{\tau}^{-1} \boldsymbol{u}' = -\mathcal{R}(\bar{\boldsymbol{u}}) \tag{1.5}$$

There exist several other residual-based models and the work of Oberai and Hughes [18] contains a comprehensive overview. The operator $\boldsymbol{\tau}$ can be viewed as a volume-average Green's function and cannot be derived analytically for all problems. Shakib et al. [19] propose a simple algebraic form for τ in the one-dimensional Burgers' equation given by

$$\tau = \sqrt{\gamma_1 \left(\frac{\bar{u}}{h}\right)^2 + \gamma_2 \left(\frac{\nu}{h^2}\right)^2} \tag{1.6}$$

where $\gamma_1 = 4$, $\gamma_2 = 144$, and h is the element size. Equation (1.6) is based on the assumption that $u' = 0$ at the nodes which implies a nodal projection. Results obtained by (1.6) can therefore not directly be compared to other methods based on the more relevant $\mathcal{L}_2$-projector. Nevertheless, (1.6) serves as a baseline and reference for the development of neural-network-based models.

1.5 Previous Work at TU Delft

An overview of previous work at TU Delft is presented here to establish a context for the current work. Two strategies for the development of neural-network-based models can be distinguished:

1. A constrained approach where a neural network is integrated as a component of an existing unresolved-scale model.

2. An unconstrained approach where a neural network represents the unresolved-scale model in its entirety and is not constrained by enforcing a certain functional form.

The previous work follows the former approach whereas the current work follows the latter. The two above approaches are opposing viewpoints and hybrid models in between these two extremes are also conceivable.

Rather than using $\gamma_1 = 4$ and $\gamma_2 = 144$ in (1.6), Durieux [20] trained a neural network to predict γ_1 and γ_2 depending on local conditions. This method retains the residual-based formulation and therefore results in a robust subgrid-scale model. Beekman [21] expanded on the work of Durieux by evaluating the robustness and suitability of the method at various conditions. The work of Beekman unveiled divergence issues of unknown origin when the problem is randomly forced. Kurian [22] retained the idea of training a neural network to predict γ_1 and γ_2, but investigated the effectiveness of other formulations such as

$$u' = -\tau \mathcal{P}^{\perp}(\mathcal{R}(\bar{u})) \tag{1.7}$$

and

$$u'_t + \tau^{-1} u' = -\tau \mathcal{P}^{\perp}(\mathcal{R}(\bar{u})) \tag{1.8}$$

where $\mathcal{P}^{\perp}$ is an orthogonal projector. Kurian confirmed that (1.6) is insufficient in randomly forced problems and found that using the dynamic model in (1.8) yields a significant improvement. These results show that the unresolved-scales u' are not necessarily quasi-steady and u'_t must therefore be taken into account.

In summary, the results of Durieux, Beekman, and Kurian show that neural networks can improve the algebraic model in (1.6). Retaining the residual-based formulation results in robust and reliable models. However, a major deficiency of the formulation in (1.6) is that there are many cases where no γ_1 and γ_2 exist such that (1.6) yields the correct value of τ. In such cases, there is an error inherent to the rigid algebraic formulation and the advantage of a more liberal approach is obvious. Furthermore, the ability of the models to generalize to new situations has not yet been demonstrated because the models were not tested in previously unseen problems.

1.6 Motivations for the Current Work

The current work places emphasis on two points:

- Following the unconstrained approach where a neural network represents the unresolved-scale model in its entirety and is not limited by a rigid formulation such as (1.6).
- Asserting that the neural network can generalize to new situations by evaluating the performance of the neural networks on previously unseen test problems.

The viewpoint in the current work therefore stands in contrast to the viewpoint in the work of Durieux, Beekman, and Kurian. Nevertheless, the contribution of this thesis complements the previous work on the restricted approach and hybrid models are conceivable in future projects. Furthermore, clearly demonstrating the property of generalization to new situations is a valuable contribution that motivates continued research on neural-network-based models in future projects.

Rather than using the rigid algebraic formulation in (1.6), a neural network could be trained to predict τ directly. The unresolved scales u' then result from

$$u' = -\tau \mathcal{R}(\bar{u}) \tag{1.9}$$

or another residual-based scheme such as (1.7) or (1.8). However, these schemes still retain a rather rigid functional relationship. In actuality, (1.9) merely represents a first-order perturbation approximation and a format such as

$$u' = -\tau_1 \mathcal{R}(\bar{u}) - \tau_2^2 \mathcal{R}(\bar{u}) - \tau_3^3 \mathcal{R}(\bar{u}) - \tau_4^4 \mathcal{R}(\bar{u}) - \dots \tag{1.10}$$

is also consistent [23]. Instead of resorting to formulations like (1.9), a neural network can be trained to directly predict the effects of u' as a function of large-scale features. There are no restrictions on the functional relationship between u' and the input variables when a neural network is trained to directly predict the effects of u'.

Direct, pointwise predictions of u' are problematic, as shown by Gamahara and Hattori [16]. Fortunately, the variational multiscale framework only requires knowledge of integral values of u' for the computation of $\bar{u}$. The unresolved-scale terms in a variational multiscale formulation of Burgers' equation consist of the element-wise integral values given by

$$\underbrace{\int_{x_l}^{x_r} w u_t' dx}_{u_t'\text{-term}} - \underbrace{\int_{x_l}^{x_r} w_x \left(\bar{u}u' + \frac{1}{2}u'^2 \right) dx}_{u'\text{-term}} + \underbrace{\frac{1}{\mathrm{Re}} \int_{x_l}^{x_r} w_x u_x' dx}_{u_x'\text{-term}}, \qquad \forall w \in \mathcal{W} \tag{1.11}$$

where x_l and x_r are respectively the left and right boundaries of an element and w are weighting functions in a set $\mathcal{W}$. A derivation and detailed description of (1.11) is given in Appendix A.

Intuitively, it should be far easier to predict the integral values in (1.11) than u' itself. This alone is a major advantage of the approach in this thesis. The fact that there exists a successful algebraic model for τ (and by extension, for u') suggests that there is a fundamental underlying relationship between u' and large-scale parameters. Thus, it should in principle be possible to train a neural network to predict the unresolved-scale terms in (1.11). A similar claim can be made for the Navier-Stokes equations; as mentioned in Section 1.1, the fact that the large-scale fluctuations in the inner layer are strongly correlated to the large-scale velocity signal in the log layer suggests that there exists a mechanism through which VLSMs modulate the near-wall small-scale fluctuations. Thus, a successful implementation of the framework proposed here paves the way for future implementations as wall models for the Navier-Stokes equations.

1.7 Research Questions

The motivations outlined in the preceding section lead to the following research questions:

Can a neural network establish a functional relationship between large-scale input features and the integral forms of the unresolved-scale terms?

The integral forms of the unresolved-scale terms consist of the u'-, u_x'-, and u_t'-terms in (1.11). The central question in the current work is if these terms can be predicted based on large-scale input features, and if yes, how accurately.

What are appropriate large-scale input features?

A question that naturally follows the previous question is what these large-scale input features should be. Based on the residual-based formulation in (1.3), $\bar{u}$ and $\mathcal{R}(\bar{u})$ are obvious candidates. Furthermore, the input features could be sampled locally or at multiple additional locations such as the adjacent elements.

Can the model be simplified by omitting individual unresolved-scale terms?

Omitting the u_i'-term, for example, would simplify the model considerably. It is therefore worthwhile to investigate potential simplifications prior to training a neural-network-based model of the unresolved-scale terms.

What is a successful neural network architecture?

Finding a successful architecture is an iterative process and any textbook on machine learning will provide guidelines that can assist in this process. Particularly the topology of the neural network is important as it strongly effects the computational complexity of the model.

How many examples are required to train a functional model?

An estimate of the minimum number of examples will provide a guideline for future modeling projects for the Navier-Stokes equations. These estimates are important because running DNSs to create training data is an expensive and time-consuming process.

Can the neural network generalize to new situations?

The objective of machine learning is the development of models that generalize to new situation. The model must be tested in problems that are different from the problems seen in training to assert that it learned an underlying functional relationship and is not merely memorizing examples.

Under what conditions does the model fail to make accurate predictions and what are the implications of these errors?

Neural networks are merely statistical tools and erroneous predictions are therefore unavoidable. It is important to understand when and why these erroneous predictions occur so that the model can be improved by tweaking the machine learning architecture or by adjusting the training examples.

What measures can be taken to improve the resiliency to erroneous predictions?

No amount of tweaking or improving the training examples can fully prevent erroneous predictions. It is therefore necessary to devise stabilization schemes to mitigate the effects of these errors on the solution.

2 Variational Multiscale Large Eddy Simulation of Burgers' Equation

This thesis builds on the foundation of the variational multiscale method (VMM). The content of this chapter is an introduction to the variational multiscale method and a description of the variational multiscale formulation of Burgers' equation. The unresolved-scale terms in the variational form of Burgers' equation are central to this thesis and are therefore discussed in detail. The chapter concludes with a physical interpretation of Burgers' equation by evaluating the energy equation for two different model problems.

2.1 Variational Multiscale Method

The variational multiscale method is a mathematical framework for multiscale problems in physics first published by Hughes [2] in 1995. The VMM is often presented as a framework within the finite element method (FEM); it would be redundant to fully cover the finite element method here, but the terminology and notation adopted from the finite element method warrants a brief introduction.

2.1.1 Finite Element Method

The finite element method can be used to solve partial differential equations (PDEs) by searching for an approximate solution $\boldsymbol{u}(\boldsymbol{x})$ in a function space $\mathcal{V}$,

$$\mathcal{V} = \text{span}\{\boldsymbol{\psi}_1(\boldsymbol{x}), \boldsymbol{\psi}_2(\boldsymbol{x}), \dots, \boldsymbol{\psi}_N(\boldsymbol{x})\} \tag{2.1}$$

where $\{\boldsymbol{\psi}_i\}_{i\in\mathcal{I}_s}$, with $\mathcal{I}_s$ as the index set $\{1, 2, \dots, N\}$, are the so-called basis functions[1]. The approximate solution $u(x)$ can be expressed as a linear combination of $\{\boldsymbol{\psi}_i\}_{i\in\mathcal{I}_s}$:

$$\boldsymbol{u}(\boldsymbol{x}) = \sum_{i\in\mathcal{I}_s} c_i\boldsymbol{\psi}_i(\boldsymbol{x}) \tag{2.2}$$

where $c_i \in \mathbb{R}$ are unknown scalar coefficients. Figure 2.1 shows an arbitrary one-dimensional function $u(x)$ expressed as a linear combination of seven local piecewise-linear basis functions. Notice that the set of basis functions $\mathcal{V}$ in Figure 2.1 inherently satisfies homogeneous boundary conditions, i.e., $u(0) = u(1) = 0$. Nonhomogeneous boundary conditions can be satisfied by special boundary functions or by simply modifying the system matrix. It is common to speak of linear elements as elements associated with piecewise-linear basis functions or hat functions.

[1] Function spaces and basis functions are analogous to vector spaces and basis vectors in linear algebra. Just like a vector space is said to be spanned by its basis vectors, a function space is spanned by its basis functions.

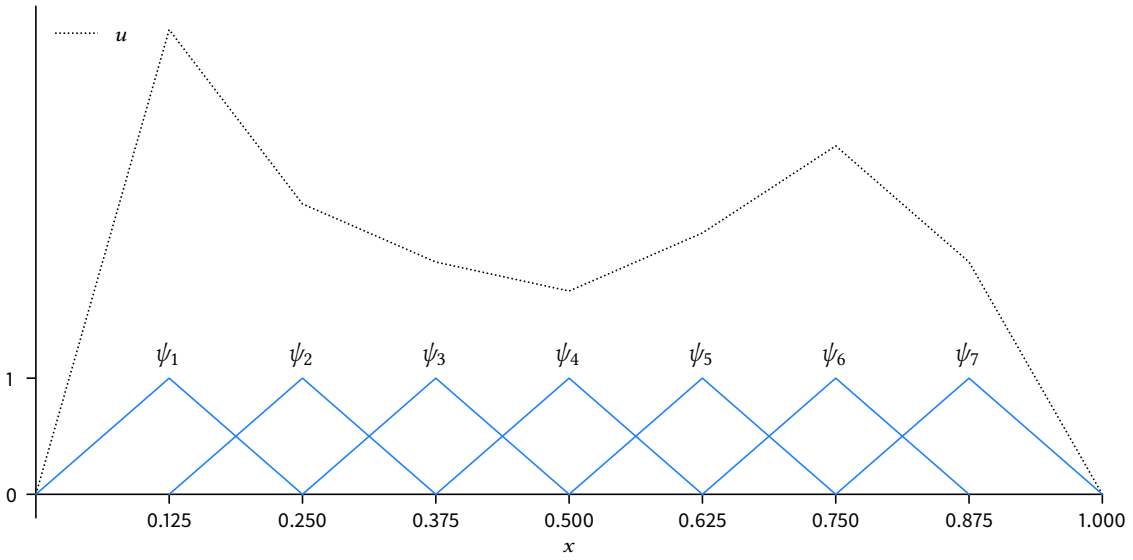

Figure 2.1: An arbitrary function u as a linear combination of seven local piecewise-linear functions, also known as hat functions.

Without going into too much detail, the basic approach of the finite element method can be illustrated with the following general problem:

$$\begin{cases} \mathcal{L}(\boldsymbol{u}) = \boldsymbol{f}, & \text{in } \Omega \\ \boldsymbol{u} = \boldsymbol{g}, & \text{on } \partial\Omega \end{cases} \tag{2.3}$$

where $\mathcal{L}$ is an arbitrary differential operator and $\partial\Omega$ denotes the boundary of the domain Ω. To derive the weak form of (2.3), multiply the PDE by a weighting function $\boldsymbol{w}$ (chosen from a set $\mathcal{W}$), and integrate both sides. The weak form of (2.3) is written as

$$\begin{cases} \text{Find } \boldsymbol{u} \in \mathcal{V} \text{ such that } \forall \boldsymbol{w} \in \mathcal{W}: \\ \displaystyle\int_\Omega \boldsymbol{w}\mathcal{L}(\boldsymbol{u})\, d\Omega = \int_\Omega \boldsymbol{w}\boldsymbol{f}\, d\Omega \end{cases} \tag{2.4}$$

The sets $\mathcal{V}$ and $\mathcal{W}$ can be disjoint, but this thesis uses the Bubnov-Galerkin method where $\mathcal{V} = \mathcal{W}$ [24]. Explicit mathematical definitions and further details such as dealing with boundary conditions can be found in any textbook on the finite element method. In this report, the weighting functions $\boldsymbol{w}$ are piecewise-linear linear functions. Higher-order alternatives exist, but piecewise-linear functions suffice for the purposes of the current work.

2.1.2 Large Eddy Simulation and the Variational Multiscale Method

Hughes et al. first applied the variational multiscale method to LES in the year 2000 [25]. Hughes et al. argue that several shortcomings of the classical LES model, such as the non-commutative filters necessary for wall-bounded flows, are eliminated in the multiscale approach. In the same vein as classical LES, the solution $\boldsymbol{u}$ is decomposed into large-scale and small-scale components,

$$\boldsymbol{u} = \bar{\boldsymbol{u}} + \boldsymbol{u}' \tag{2.5}$$

where $\bar{\boldsymbol{u}}$ is the large-scale solution and $\boldsymbol{u}'$ is the small-scale solution. Hughes et al. explain that the interpretation of $\bar{\boldsymbol{u}}$ and $\boldsymbol{u}'$ is different from classical LES because there exists no simple filter

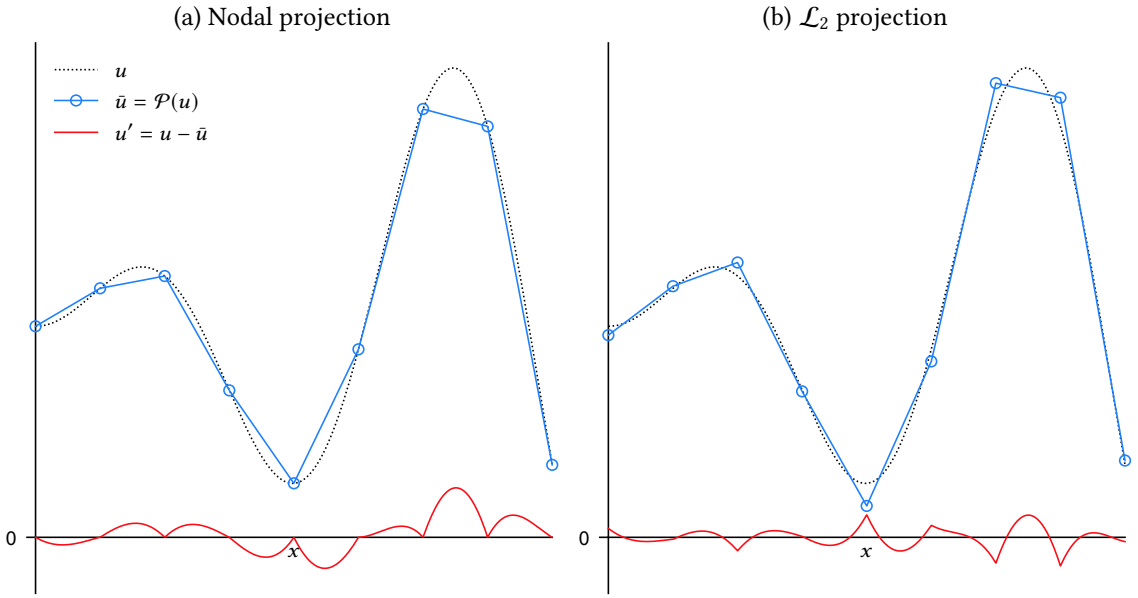

Figure 2.2: Illustration of the projection process with a side-by-side comparison of two common projectors. The $\mathcal{L}_2$ projection provides a good fit in a least squares sense, whereas the nodal projection is exact at the nodes.

to determine $\bar{\boldsymbol{u}}$ from $\boldsymbol{u}$. Instead, $\bar{\boldsymbol{u}}$ is determined from $\boldsymbol{u}$ by a projector $\mathcal{P}$, projecting $\boldsymbol{u}$ onto a space of finite element basis functions, i.e.,

$$\bar{\boldsymbol{u}} = \mathcal{P}(\boldsymbol{u}) \tag{2.6}$$

Figure 2.2 illustrates the projection of an arbitrary function u onto a set of piecewise-linear basis functions. The $\mathcal{L}_2$ projection provides a good fit in a least squares sense and is natural to the Bubnov-Galerkin method, which is based on minimizing the $\mathcal{L}_2$ error. The nodal projector is exact at the nodes, but often produces a greater error within the elements. The small-scale solution $\boldsymbol{u}'$ is the result of subtracting $\bar{\boldsymbol{u}}$ from $\boldsymbol{u}$, i.e.,

$$\boldsymbol{u}' = \boldsymbol{u} - \mathcal{P}(\boldsymbol{u}) \tag{2.7}$$

The goal of multiscale LES is approximating $\boldsymbol{u}'$ such that the solution represents $\bar{\boldsymbol{u}}$ as accurately as possible. The projector is to be chosen by the user, and both the nodal projection and the $\mathcal{L}_2$ projection are common choices. The $\mathcal{L}_2$ projection is used in the remainder of this thesis.

2.1.3 Variational Multiscale Formulation of Burgers' Equation

As mentioned in Chapter 1, Burgers' equation can be used as a model of the Navier-Stokes equations because it reproduces features that are expected of turbulence. The strong form of Burgers' equation reads as follows:

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = f(x,t), & x \in \Omega, t \in (0,T] & \text{(PDE)} \\ u(x,0) = I(x), & x \in \Omega & \text{(initial condition)} \\ u = g(t), & x \in \partial\Omega, t \in (0,T] & \text{(boundary condition)} \end{cases} \tag{2.8}$$

where $\mathrm{Re} = 1/\nu$ is the Reynolds number[2], $f(x,t)$ is a forcing function, $I(x)$ is the initial condition, and $g(t)$ is the boundary value.

To simplify subsequent writing, let

$$(w,u) = \int_\Omega wu \, d\Omega \tag{2.9}$$

This notational convention is common in the context of finite element methods and the variational multiscale method. Using (2.9), the variational multiscale form of Burgers' equation is written as

$$\begin{cases} \text{Find } \bar{u} \in \mathcal{W} \text{ such that } \forall w \in \mathcal{W}: \\ (w, \bar{u}_t) - \frac{1}{2}\left(w_x, \bar{u}^2\right) + \frac{1}{\mathrm{Re}}\left(w_x, \bar{u}_x\right) + \text{unresolved-scale terms} = (w, f) \end{cases} \tag{2.10}$$

where

$$\text{unresolved-scale terms} = (w, u_t') - \left(w_x, \bar{u}u' + \frac{1}{2}u'^2\right) + \frac{1}{\mathrm{Re}}\left(w_x, u_x'\right), \qquad \forall w \in \mathcal{W} \tag{2.11}$$

A step-by-step derivation of (2.10) and (2.11) can be found in Appendix A.

2.2 Unresolved-Scale Terms

As shown in Appendix A, the unresolved-scale terms in the variational multiscale formulation of Burgers' equation are written as

$$\text{unresolved-scale terms} = \underbrace{(w, u_t')}_{u_t'\text{-term}} - \underbrace{\left(w_x, \bar{u}u' + \frac{1}{2}u'^2\right)}_{u'\text{-term}} + \underbrace{\frac{1}{\mathrm{Re}}\left(w_x, u_x'\right)}_{u_x'\text{-term}}, \qquad \forall w \in \mathcal{W} \tag{2.12}$$

For brevity, the individual integral terms are referred to as the u'-, u_t'-, and u_x'-terms.

2.2.1 The u'-Term

The u'-term, a convective term, is given by

$$-\left(w_x, \bar{u}u' + \frac{1}{2}u'^2\right) = -\int_\Omega w_x \left(\bar{u}u' + \frac{1}{2}u'^2\right) dx, \qquad \forall w \in \mathcal{W} \tag{2.13}$$

Since w is linear, its derivatives w_x are constants and can be taken out of the integral, i.e.,

$$-w_x \int_\Omega \left(\bar{u}u' + \frac{1}{2}u'^2\right) dx, \qquad \forall w \in \mathcal{W} \tag{2.14}$$

[2]The Reynolds number in Burgers' equation is defined as

$$\mathrm{Re} \equiv \frac{u}{\nu}$$

Since the characteristic wave speed u is $O(1)$, the Reynolds number is written as $\mathrm{Re} = 1/\nu$.

In the current work, a neural network is trained to predict (2.14) for a given element. Since there are two (non-zero) weighting functions per element, (2.14) is represented by two integral values. Since the derivatives w_x are constants of the same magnitude, but opposite in sign, the two integral values are also of the same magnitude, but opposite in sign. The model need therefore only predict a single integral value; the remaining integral value is obtained by multiplying the prediction by -1.

2.2.2 The u'_x-Term

The u'_x-term, a viscous interaction term, is given by

$$\frac{1}{\mathrm{Re}}\left(w_x, u'_x\right) = \frac{1}{\mathrm{Re}}\int_\Omega w_x u'_x \, dx, \qquad \forall w \in \mathcal{W} \tag{2.15}$$

The constants w_x can be taken out of the integral and since Re is a known constant, it need not be part of the model. Thus, the model need only predict

$$w_x \int_\Omega u'_x \, dx, \qquad \forall w \in \mathcal{W} \tag{2.16}$$

As explained in Section 2.2.1, the derivatives w_x are constants of the same magnitude, but opposite in sign. The model need therefore only predict a single integral value and the remaining integral value is found by flipping the sign.

It is worth mentioning that $(w_x, u'_x) = 0$ if a nodal projection is used. Let x_l and x_r respectively denote the left and right boundaries of an element. From the fundamental theorem of calculus:

$$\int_{x_l}^{x_r} u'_x \, dx = u'(x_l) - u'(x_r) \tag{2.17}$$

Using the nodal projection implies that $u'(x_l) = u'(x_r) = 0$. Therefore,

$$\int_{x_l}^{x_r} u'_x \, dx = 0 \tag{2.18}$$

which yields

$$\frac{1}{\mathrm{Re}}\left(w_x, u'_x\right) = 0, \qquad \forall w \in \mathcal{W} \tag{2.19}$$

This explains why the u'_x-term could be omitted in the work of Durieux, Beekman, and Kurian. This is not the case in the current work because the $\mathcal{L}_2$ projection is used rather than the nodal projection.

2.2.3 The u'_t-Term

The u'_t-term, also known as the temporal term, is given by

$$\left(w, u'_t\right) = \int_\Omega w u'_t \, dx, \qquad \forall w \in \mathcal{W} \tag{2.20}$$

The u'_t-term cannot be simplified by taking w out of the integral. Since there are two weighting functions per element, (2.20) is represented by two integral values and the unresolved-scale model need therefore predict both integral values separately.

2.3 Large-Scale Energy Balance

Burgers' equation obeys the law of conservation of energy, a quantity analogous to the kinetic energy in the Navier-Stokes equations. The large-scale energy is defined as

$$\bar{E} \equiv \int_\Omega \frac{1}{2}\bar{u}^2 dx \tag{2.21}$$

Energy is added to the domain by doing positive work (via the forcing function) and by allowing energy to flow into the domain through its boundaries. Energy is removed from the domain by doing negative work, by letting energy flow out of the domain, and, most importantly, by viscous dissipation.

2.3.1 Evolution Equation of $\bar{E}$

An equation that describes the evolution of the large-scale energy $\bar{E}$ can be derived by substituting $w = \bar{u}$ into the variational form of Burgers' equation. The evolution equation of $\bar{E}$ is written as

$$\frac{d\bar{E}}{dt} = \underbrace{-\frac{1}{2}\int_\Omega \bar{u}\frac{\partial}{\partial x}\left(\bar{u}^2\right) dx}_{\text{advection}} + \underbrace{\frac{1}{\mathrm{Re}}\int_\Omega \bar{u}\frac{\partial^2 \bar{u}}{\partial x^2} dx}_{\text{viscous dissipation}} \underbrace{- \text{unresolved-energy terms}}_{\text{unresolved-scale transfer}} + \underbrace{\int_\Omega \bar{u} f\, dx}_{\text{work done}} \tag{2.22}$$

where

$$\text{unresolved-energy terms} = \int_\Omega \bar{u}\frac{\partial u'}{\partial t} dx + \int_\Omega \bar{u}\frac{\partial}{\partial x}\left(\bar{u}u' + \frac{1}{2}u'u'\right) dx - \frac{1}{\mathrm{Re}}\int_\Omega \bar{u}\frac{\partial^2 u'}{\partial x^2} dx \tag{2.23}$$

A step-by-step derivation of (2.22) and (2.23) can be found in Appendix B.

2.3.2 Model Problems

For technical reasons detailed in Appendix B, (2.22) can (1) only be evaluated globally and (2) only be evaluated for problems with homogeneous or periodic boundary conditions. Nonetheless, evaluating (2.22) for two model problems provides useful physical interpretation of Burgers' equation.

The first model problem, featuring a steady forcing function and homogeneous boundary conditions, is given by

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = 1, & x \in (0,1), t \in (0,2], \mathrm{Re} = 100 \\ u(x,0) = 0, & x \in (0,1) \\ u(0,t) = 0, & t \in (0,2] \\ u(1,t) = 0, & t \in (0,2] \end{cases} \tag{2.24}$$

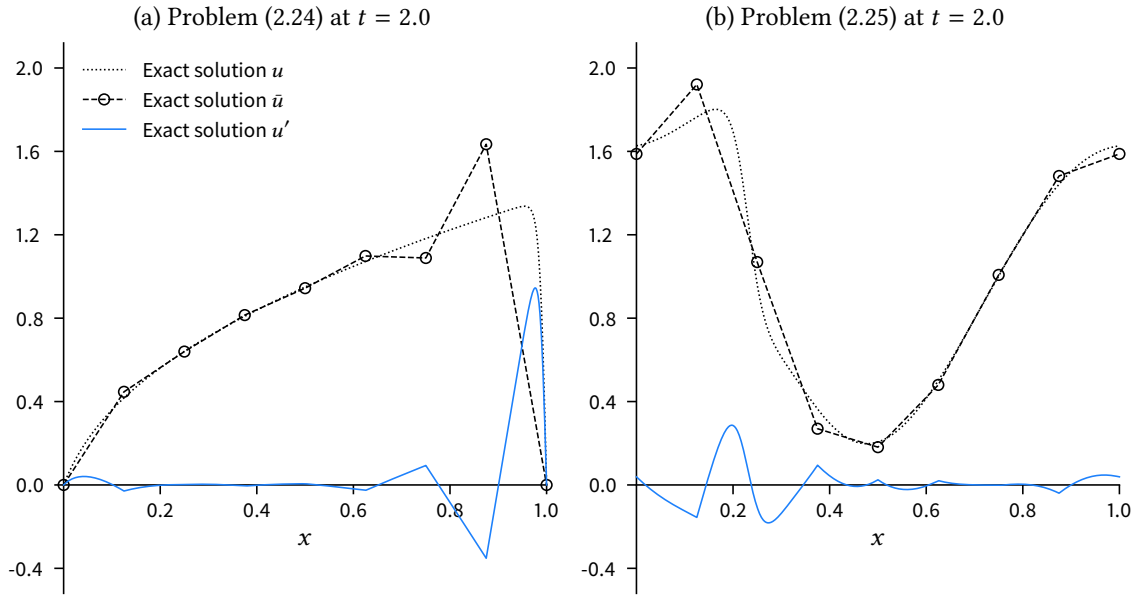

Figure 2.3: Exact solutions to (2.24) and (2.25) at $t = 2.0$. Note that the solution to (2.24) has settled into a steady state whereas the solution to (2.25) has not.

The solution to (2.24) settles into a steady state at $t \approx 1$. The second model problem, featuring a periodic forcing function and periodic boundary conditions, is given by

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = 3\sin(3\pi x)\cos(2\pi t), & x \in (0,1), t \in (0,2], \mathrm{Re} = 100 \\ u(x,0) = 1, & x \in (0,1) \\ u(0,t) = u(1,t), & t \in (0,2] \end{cases} \tag{2.25}$$

Since the forcing function in (2.25) is unsteady, the solution to (2.25) does not settle into a steady state. The exact solutions to (2.24) and (2.25) are given in Figure 2.3.

2.3.3 Evaluating the Evolution Equation

Both model problems are solved by DNS and resulting time series plots of the terms in the energy equation are given in Figure 2.4. The absence of advection is a result of the homogeneous and periodic boundary conditions. Homogeneous boundary conditions isolate the domain (i.e., energy is neither entering nor leaving) whereas periodic boundary conditions ensure that the amount of energy leaving the domain is equal to the amount of energy entering the domain.

Figure 2.4a shows that the unresolved-scales in (2.24) are purely dissipative. The amount of energy transferred to the unresolved-scales is far greater than the viscous dissipation at the large scales; this demonstrates that viscous dissipation occurs predominantly at small scales. Since most energy is dissipated at the unresolved scales, the LES would fail spectacularly in the absence of a good model for the unresolved scales. Figure 2.4b, on the other hand, paints a completely different picture; the unresolved-scale transfer term takes on both negative and positive values. In other words, there can be transfer of energy from unresolved scales to large scales, which is a phenomenon known as backscatter.

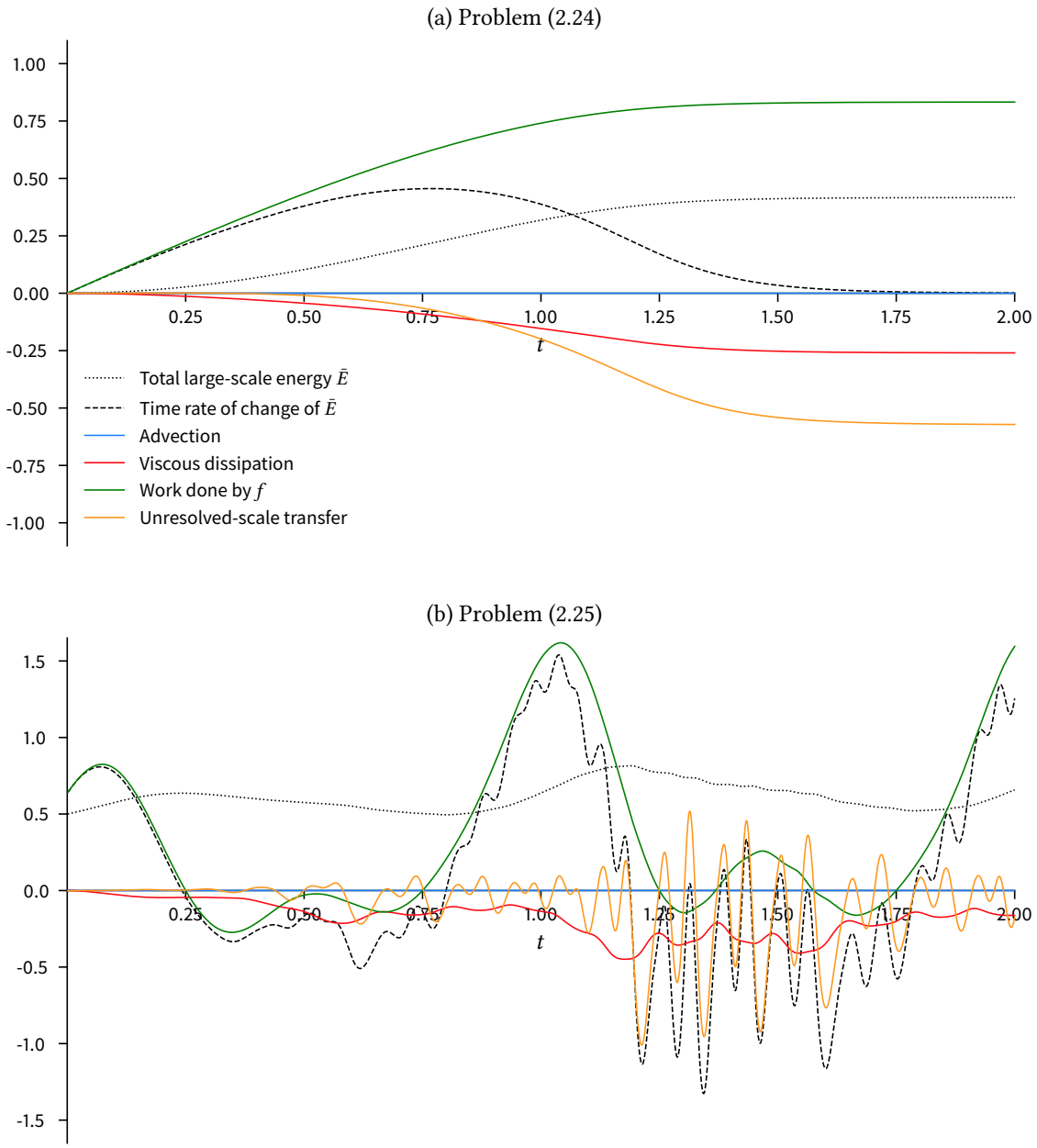

Figure 2.4: Time series plots of the individual terms of the large-scale energy balance evaluated using exact solutions to (2.24) and (2.25).

3 Machine Learning

The conventional approach to programming is solving problems by devising sequences of instructions; algorithms. It is hard to apply this approach to problems like recognizing handwritten digits. Every person's handwriting is different and capturing these variations makes it hard to devise a clear-cut algorithm. Computer scientists have long been puzzled about such problems. It turns out that the solution is surprisingly uncomplicated, but requires a different programming paradigm: machine learning. Machine learning is based on the concept of automatically extracting statistics from data. Machine learning is a particularly powerful tool when the functional relationship between inputs and outputs cannot clearly be defined or is unknown, like in the case of unresolved-scale models.

3.1 Neural Networks

Neural networks are one of many machine learning tools. The principle of neural networks is introduced by investigating the canonical problem of recognizing handwritten digits.

3.1.1 Training Datasets

The MNIST dataset is a set of handwritten digits collected by the United States' National Institute of Standards and Technology. The MNIST dataset is composed of 70 000 different 28×28 pixel greyscale images of handwritten digits. Some samples of the MNIST dataset are shown in Figure 3.1. A single training example is composed of $28 \times 28 = 784$ grayscale values and a corresponding

Figure 3.1: Sample of 6 digits of the MNIST dataset. Published in [26].

output; the digit that the image represents. The grayscale values are called the input features. In other words, the images are encoded as $28 \times 28 = 784$-dimensional vectors, denoted $\boldsymbol{x}$. The outputs are encoded as 10-dimensional vectors $\boldsymbol{y}$. If the image depicts a six, then the sixth element of the output vector is labelled 1, i.e.,

$$\boldsymbol{y} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.1}$$

Training set is the collective term for all training examples. Because the outputs $\boldsymbol{y}$ are manually labeled, this type of machine learning is called supervised learning.

3.1.2 Neurons and Neural Networks

A neural network is composed of so-called neurons. Neurons are computational units that take a number of inputs and produce an output. Figure 3.2 depicts a neuron with three inputs.

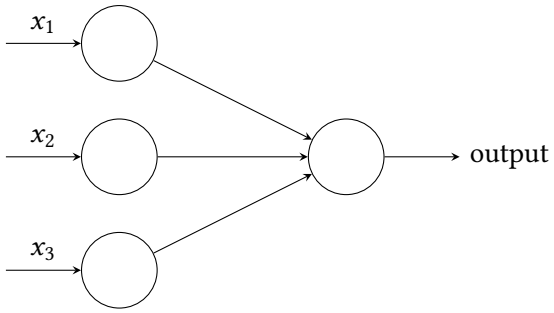

Figure 3.2: Depiction of an individual neuron.

The output of the neuron in figure 3.2 is given by

$$\text{output} = f(x_1 w_1 + x_2 w_2 + x_3 w_3 + b) \tag{3.2}$$

where x_i are inputs, w_i are weights, b is a bias, and f is a nonlinear differentiable function referred to as the activation function. There is a certain weight w_i associated with every input x_i. Equation (3.2) can also be written as

$$\text{output} = f(\boldsymbol{x} \cdot \boldsymbol{w} + b) \tag{3.3}$$

where $\boldsymbol{x}$ and $\boldsymbol{w}$ are respectively the vectors of inputs and weights. The activation function is typically a smoothed version of the Heaviside step function, i.e., $f \to 1$ as $\boldsymbol{x} \cdot \boldsymbol{w} + b \to \infty$ and $f \to 0$ as $\boldsymbol{x} \cdot \boldsymbol{w} + b \to -\infty$.

The neural network for recognizing handwritten digits shown in Figure 3.3 is a collective of interconnected neurons. The input layer consists of 784 units; one unit for every pixel of the image. The particular neural network has a single hidden layer of 15 neurons. The word "hidden" carries no special meaning; it simple means that the layer is neither an input layer nor an output layer. The output layer consists of 10 units in accordance to the output vector in (3.1).

3.1.3 Training a Neural Network

The set of weights and biases are called the parameters of a neural network. The procedure referred to as training automatically tunes the parameters of a neural network such that it predicts the correct result for most, if not all examples in the training set. Training a neural network requires a measure of how well the neural network is performing, the so-called cost function. A commonly used cost function is the mean-squared error (MSE) given by

$$C = \frac{1}{2n} \sum_x ||\boldsymbol{y} - \hat{\boldsymbol{y}})||^2 \tag{3.4}$$

where $\hat{\boldsymbol{y}}$ is the output of the neural network and $\boldsymbol{y}$ is the manually labeled reference output. The performance of the neural network is optimal if $\hat{\boldsymbol{y}} = \boldsymbol{y}$ for all $\boldsymbol{x}$ in the training set, i.e., $C = 0$. Even though (3.4) does not explicitly show it, C is a function of the parameters w_i and b_i. Because

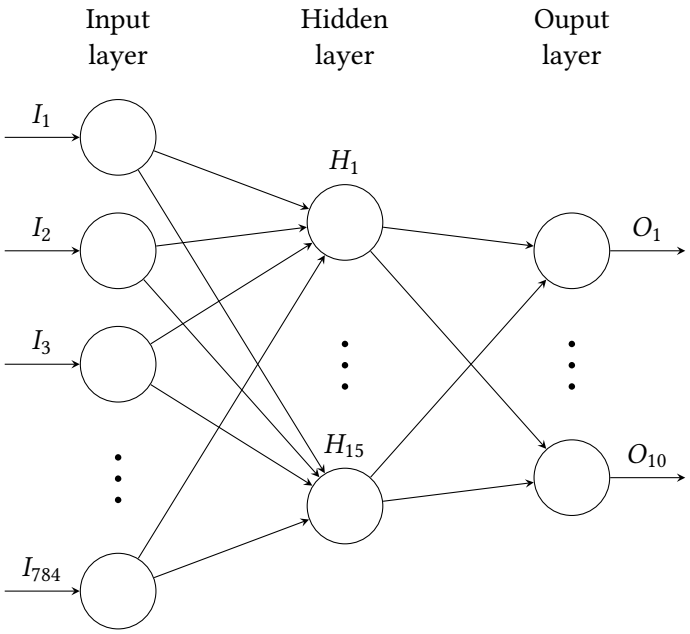

Figure 3.3: The neural network for recognizing handwritten digits.

both the neural network and the cost function are differentiable, the cost can be minimized by updating the parameters as follows:

$$w_i \coloneqq w_i - \alpha \frac{\partial C}{\partial w_i} \tag{3.5}$$

$$b_i \coloneqq b_i - \alpha \frac{\partial C}{\partial b_i} \tag{3.6}$$

where α is the so-called learning rate. Variables like α, the choice of activation function, the number of hidden layers, the number of hidden units per layer, etc., are called hyperparameters. The above minimization procedure of C is the fundamental idea behind the backpropagation algorithm. The backpropagation algorithm essentially solves a minimization problem in a high-dimensional vector space.

3.1.4 Generalization

The purpose of training a neural network is reliably achieving a high accuracy on samples that are not included in the training data set; this is referred to as generalization. To evaluate the ability of a neural network to generalize, the training set is commonly split into a training set, a validation set, and a test set. The training set is used for training and the validation set is used to evaluate the performance of the neural network and to tune the hyperparameters. After training, the neural network is tested a final time on the test set.

A neural network that achieves a high success rate on the training set and a low success rate on the test set is said to be overfitting the training set. The neural network is essentially memorizing the training set and cannot deal with samples not seen in training. A neural network with the ability to generalize would achieve a high success rate on both the training set and the test set. There are a number of design heuristics to ensure that a neural network is able to generalize. Two examples of these heuristics are regularization and dropout.

3.2 Neural-Network-Based Unresolved-Scale Model

In the current work, the inputs and outputs are real-valued numbers. Problems of this type are known as regression problems whereas the example problem in the previous section is called a classification problem. This is the only significant difference with the canonical example of recognizing handwritten digits.

3.2.1 Outputs

The outputs of the neural network are the element-wise integral values of the unresolved-scale terms given by

$$\text{output} = \left\{ \left(w_x, \bar{u}^2 + \frac{1}{2} u' u' \right), \left(w_l, u'_t \right), \left(w_r, u'_t \right), \left(w_x, u'_x \right) \right\} \tag{3.7}$$

where w_l and w_r respectively denote the left and right weighting functions of an element. As explained in Section 2.2, w_x is the gradient of either the left or right weighting function.

3.2.2 Inputs

The fact that Shakib's algebraic model in (1.6) is reasonably accurate suggests that using $\bar{u}$, $\mathcal{R}(\bar{u})$, ν, and h as inputs would at a minimum yield a similarly accurate model. However, the large-scale residual $\mathcal{R}(\bar{u})$ need not be provided explicitly. Since $\mathcal{R}(\bar{u})$ is given by

$$\mathcal{R}(\bar{u}) = \bar{u}_t + \bar{u}\bar{u}_x - \nu \bar{u}_{xx} - f(x) \tag{3.8}$$

the large-scale residual $\mathcal{R}(\bar{u})$ can be provided implicitly via $\bar{u}_t$, $\bar{u}$, and f. Since linear elements are used, the gradient $\bar{u}_x$ is embodied in $\bar{u}$ by providing $\bar{u}$ at the element boundaries. The Laplacian $\bar{u}_{xx}$ is zero for linear elements and need not be given. Thus, an appropriate set of input features should at the minimum contain $\bar{u}$, $\bar{u}_t$, f, ν (or Re), and h. The neural network can then learn an internal representation of $\mathcal{R}(\bar{u})$ and $\bar{u}_x$ if necessary.

3.2.3 The Local Stencil

Sampling the input features $\bar{u}$, $\bar{u}_t$, and f at the element boundaries is referred to as the "local stencil". In other words, the local stencil is given by

$$\text{input} = \left\{ \bar{u}_i, \bar{u}_{i+1}, \left(\frac{\partial \bar{u}}{\partial t} \right)_i, \left(\frac{\partial \bar{u}}{\partial t} \right)_{i+1}, f_i, f_{i+1}, \text{Re}, h \right\} \tag{3.9}$$

where i and $i+1$ respectively denote the left and right element boundaries. An illustration of the local stencil is given in Figure 3.4a.

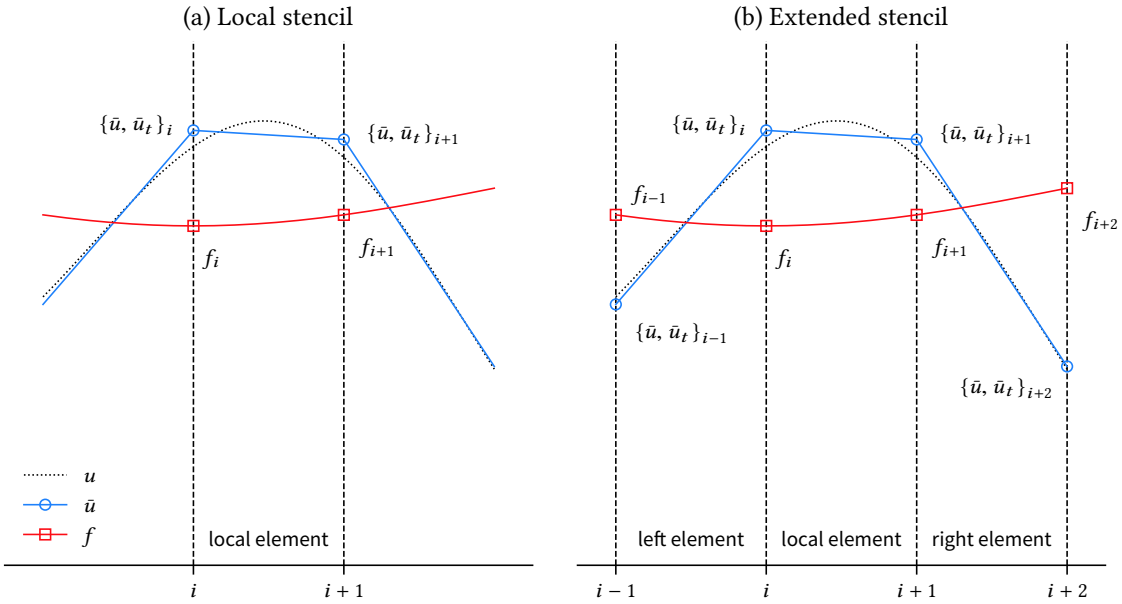

Figure 3.4: Illustration and comparison of the local stencil and the extended stencil.

3.2.4 The Extended Stencil

Generally, the interaction between u' and $\bar{u}$ is nonlocal up to a certain correlation length. This correlation length is not necessarily smaller than h. It therefore makes sense to devise an "extended stencil" where information of adjacent elements is included in the input features. The extended stencil is given by

$$\text{input} = \left\{\bar{u}_{i-1}, \bar{u}_i, \bar{u}_{i+1}, \bar{u}_{i+2}, \left(\frac{\partial \bar{u}}{\partial t}\right)_{i-1}, \left(\frac{\partial \bar{u}}{\partial t}\right)_i, \left(\frac{\partial \bar{u}}{\partial t}\right)_{i+1}, \left(\frac{\partial \bar{u}}{\partial t}\right)_{i+2}, f_{i-1}, f_i, f_{i+1}, f_{i+2}, \text{Re}, h\right\} \quad (3.10)$$

An illustration of (3.10) is given in Figure 3.4b. The question whether the extended stencil yields an improvement over the local stencil is answered in Chapter 5.

3.2.5 Additional Input Features

If the elements are distributed over multiple computing nodes, then communication between nodes can result in significant communication overhead. The extended stencil must therefore be limited to adjacent elements only. Nevertheless, the input features can be extended in other ways if the local and extended stencils of the previous sections prove to be insufficient. Since f is a known function, the possibility remains to sample f at other locations in addition to the element boundaries. Furthermore, the time-derivative of f could also be provided. It is also possible to provide to unresolved-scale terms at the previous timestep.

3.2.6 Neural Network Architecture

The current work is implemented with Keras, an open-source neural-network library written in Python [27]. Following the recommendation of François Chollet, the author of Keras, the `rmsprop`

training algorithm is used and the training set is not split into so-called mini-batches. Given the training set and validation set, the training algorithm will minimize the mean-squared error (MSE) on the training set. The mean-absolute error (MAE) on the validation set is used to judge the performance of the model. Minimizing the MSE and not the MAE yields faster convergence. The MAE, on the other hand, is more meaningful as a performance metric [28].

3.2.7 Training Procedure

The training procedure can be summarized as follows:

1. Perform a DNS of a training problem.
2. Compute the integral values of the unresolved-scale terms for all elements and for all timesteps.
3. Find the input values corresponding to the unresolved-scale terms and merge the inputs and outputs into a dataset.
4. Randomly shuffle the dataset.
5. Split the dataset into a training set and a validation set.
6. Normalize both the inputs and outputs.

The normalization step ensures that all inputs and outputs are scaled to approximately the same range centered around zero.

3.3 Integration of the Neural Network in the Simulation

Discretization of the Navier-Stokes equations and Burgers' equation results in a system of nonlinear equations which can be solved using Newton's method. The iterations in Newton's method are typically referred to as corrector passes. There are several ways to integrate a neural-network-based unresolved-scale model into an LES solver. Two possibilities are:

1. Invoking the neural network once before the corrector passes.
2. Invoking the neural network at every corrector pass.

Both options have advantages and disadvantages. With option (1), the prediction of the neural network is inevitably based on the solution at the previous timestep. This has the advantage that the prediction need not be updated during the corrector passes which speeds up convergence. The disadvantage is that the neural network lags behind in terms of information because it cannot use the intermediate solutions available in the corrector passes.

The advantage of option (2) is that the neural network can use the intermediate solutions as inputs to make more accurate predictions. The disadvantage of option (2) is that updating the predictions at every corrector pass slows convergence. Furthermore, the intermediate solutions are unphysical solutions not seen in training, which could be problematic. In the current work, the neural network is invoked at every corrector pass because accuracy is of higher priority than speed of execution. Further details on the integration of the neural network in the LES can be found in Appendix C.

4 Reconstructing $\bar{u}$ with Exact Unresolved-Scale Terms

The unresolved-scale terms in the variational multiscale formulation of Burgers' equation are

$$\text{unresolved-scale terms} = \underbrace{(w, u_t')}_{u_t'\text{-term}} - \underbrace{\left(w_x, \bar{u}u' + \frac{1}{2}u'^2\right)}_{u'\text{-term}} + \underbrace{\frac{1}{\text{Re}}\left(w_x, u_x'\right)}_{u_x'\text{-term}}, \qquad \forall w \in \mathcal{W} \tag{4.1}$$

If there existed a perfect predictive model for these integral terms, then the LES would yield the exact large-scale solution $\bar{u}$ to any problem. However, any real-world model will be imperfect and this chapter answers several questions about the effects of these imperfections on the large-scale solution $\bar{u}$. Specifically:

- Are the individual terms in (4.1) of equal importance?
- Are any of the terms in (4.1) negligible?
- How does the large-scale solution $\bar{u}$ respond to errors in the unresolved-scales?
- How does the significance of the u_t'-term change as Δt_{LES} is increased?

A perfect model for the unresolved-scale terms does not exist, but it can be emulated by inserting exact (precomputed) unresolved-scale terms into an LES. Since the LES should yield the exact large-scale solution $\bar{u}$, this method also serves as a validation of the implementation. Furthermore, rather than inserting exact unresolved-scale terms into the LES, these values can be deliberately changed to observe the effects on the large-scale solution $\bar{u}$.

4.1 Model Problems

Two model problems are used to answer the aforementioned questions about the effects of errors in the unresolved scales. The model problems are solved by DNS with $h = \frac{1}{1024}$ and $\Delta t_{\text{DNS}} = 0.001$. The first model problem, featuring a steady forcing function and Dirichlet boundary conditions, is given by

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\text{Re}}\dfrac{\partial^2 u}{\partial x^2} = 1, & x \in (0,1), t \in (0,2], \text{Re} = 100 \\ u(x,0) = 1, & x \in (0,1) \\ u(0,t) = 1, & t \in (0,2] \\ u(1,t) = 1, & t \in (0,2] \end{cases} \tag{4.2}$$

The solution of (4.2) settles in a steady state at $t \approx 1$.

The second model problem, featuring a periodic forcing function and periodic boundary conditions, is given by

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = f(x,t), & x \in (0,1), t \in (0,2], \mathrm{Re} = 100 \\ u(x,0) = 1 + \sin(2\pi x), & x \in (0,1) \\ u(0,t) = u(1,t), & t \in (0,2] \end{cases} \tag{4.3}$$

where

$$f(x,t) = \sin(\pi x)\sin(\pi t) + \sin(2\pi x)\sin(2\pi t) + \sin(3\pi x)\sin(3\pi t) \tag{4.4}$$

The solution of (4.4) remains unsteady as a result of the unsteady forcing function.

4.2 Solutions sans Unresolved-Scale Model

A comparison between the exact solutions of $\bar{u}$ and the LES solutions of $\bar{u}$ is shown in Figure 4.1. Near $x = 1$ in Figure 4.1a, u abruptly changes from $u \approx 1.5$ to $u = 1$ to satisfy the boundary condition $u(1) = 1$. This abrupt change takes place over a length scale proportional to $\nu = 1/\mathrm{Re}$ and is therefore called a shock (or in this case, a boundary layer). The large-scale solution $\bar{u}$ cannot fully resolve the shock and the associated viscous dissipation. In other words, viscous dissipation acts on a length scale below the mesh size. As a result, the system accumulates excess energy and steep gradients emerge throughout the computational domain to dissipate this excess energy. These oscillations render the LES solution of $\bar{u}$ useless. The oscillations are less pronounced in Figure 4.1b because the gradient of the shock in Figure 4.1b is less steep. Thus, the onset of oscillations is a mesh resolution problem and occurs if the dissipative length scale is smaller than the mesh size h [24]. Since the mesh size h in LESs is greater than the dissipative length scale, the onset of oscillations must be prevented by an unresolved-scale model; an unresolved-scale model prevents the onset of oscillations by modeling the transfer of energy to the unresolved-scales.

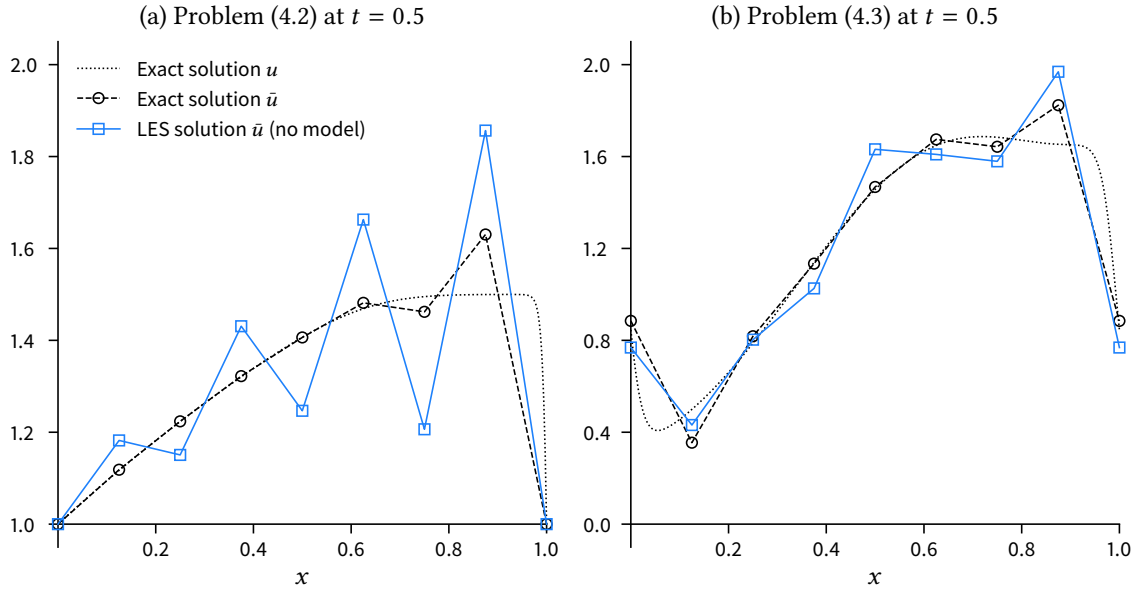

Figure 4.1: Comparison between the exact solutions of $\bar{u}$ and the LES solutions of $\bar{u}$.

4.3 Importance of the Individual Terms

To importance of the individual unresolved-scale terms is examined by starting with an LES sans unresolved-scale model, inserting the exact terms one-by-one, and observing the change in the LES solution. The results of this experiment are shown in Figure 4.2. Inserting only the u'-term dramatically improves the solution compared to the results sans unresolved-scale model in Figure 4.1. Inserting the u'_x-term in addition to the u'-term further improves the solution to a point where it matches the exact solution. As expected from variational multiscale theory, the LES solution matches the exact solution when all exact unresolved-scale terms are inserted. The u'-term clearly is the dominant term, followed by the u'_x-term. These results suggest that the u'_t-term is negligible. In actuality, the significance of the u'_t-term strongly depends on the timestep in the LES. The u'_t-term is expected to be negligible because the unresolved-scale terms are quasi-steady at small timesteps ($\Delta t_{\text{LES}} = \Delta t_{\text{DNS}}$).

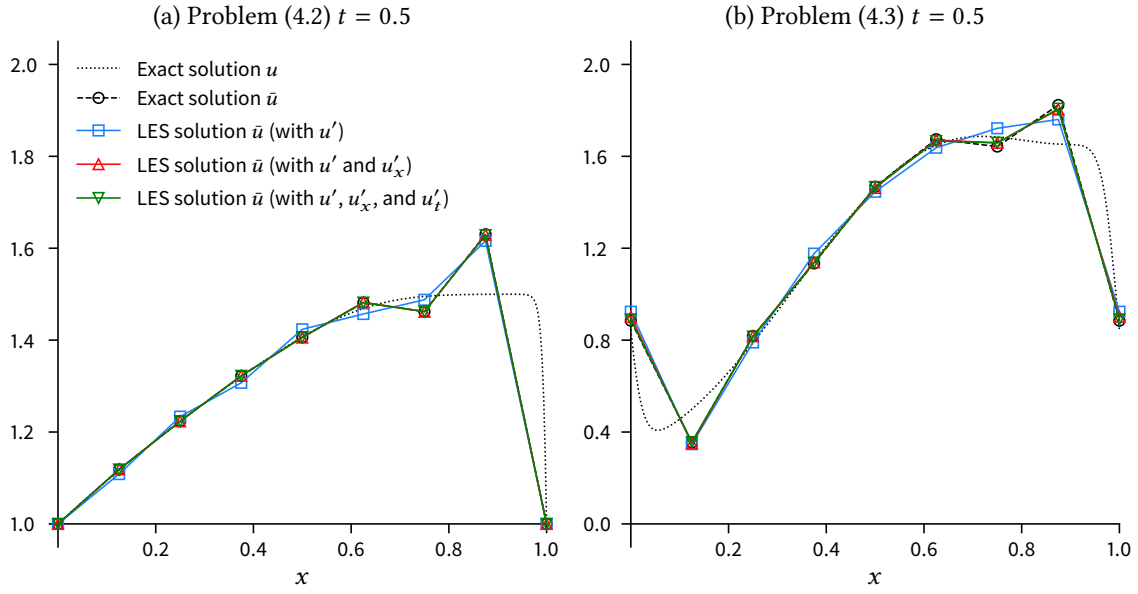

Figure 4.2: LES solutions showing the importance of the individual unresolved-scale terms.

4.4 Significance of the u'_t-Term

The significance of the u'_t-term is examined by omitting the u'_t-term and observing the effects on the LES solution when increasing the timestep. A relevant non-dimensional parameter is the Courant number given by

$$C = \frac{u\Delta t}{h} \tag{4.5}$$

where the characteristic wave speed u is $O(1)$. The results in Figure 4.3 show that the error of the solution increases as the Courant number increases, implying that the u'_t-term is no longer negligible and that the unresolved scales are no longer quasi-steady at larger Courant numbers. This is especially true in highly unsteady problems, as shown in Figure 4.3b. The (w, u'_t)-term represents work done to the system because it closely resembles the term (w, f). The sensitivity to the u'_t-term therefore makes sense because it performs a direct addition or removal of energy.

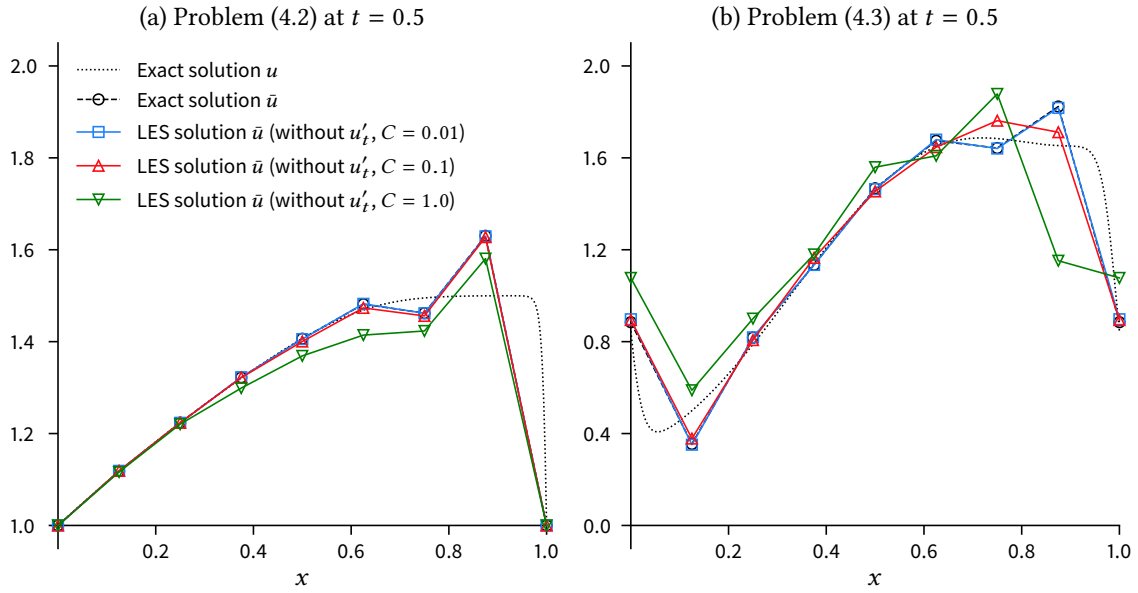

Figure 4.3: LES solutions showing the effects of omitting the u'_t-term at different Courant numbers. The u'_t-term is no longer negligible at higher timesteps.

4.5 Effects of Noisy Predictions

The effects of noisy unresolved-scale-term predictions are examined by adding artificial noise to the integral values before insertion into the LES. Artificial noise is added by multiplying the integral values by a uniformly distributed random number centered around 1. For noise levels of ±10%, the terms are multiplied by a random number in the range $[0.9, 1.1]$. The results of this experiment are shown in Figure 4.4 and show that the large-scale solutions $\bar{u}$ are virtually unaffected by noise levels of ±40%. It must however be noted that this particular method of adding artificial noise preserves the mean value and does therefore, on average, still add or remove the correct amount of energy. This explains why artificial noise has such a small effect on the results.

4.6 Effects of Biased Predictions

The effects of biased unresolved-scale-term predictions are more pronounced than the effects of noisy predictions. To introduce a bias of +10%, the unresolved-scale terms are multiplied by 1.1. The results of this experiment are shown in Figure 4.6 and 4.5 for respectively a positive bias and a negative bias. The effects of biased unresolved-scale-term predictions are significant because a biased predictions result in a shortage or surplus of energy with respect to the exact solution. The difference to the exact solution is most pronounced in Figure 4.5a and 4.6a because the unresolved-scale terms are purely dissipative in this case. In Problem (4.3), the impact on the total energy is less severe because there is local backscatter in addition to energy dissipation. This explains why the effects of biased unresolved-scale-term predictions are less pronounced in Figure 4.5b and 4.6b.

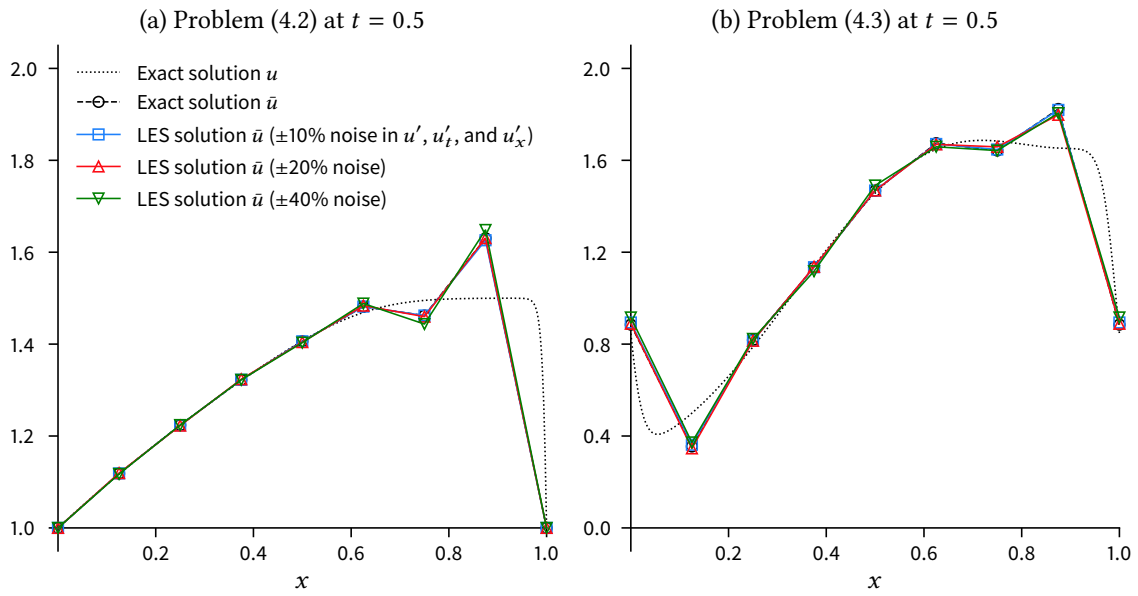

Figure 4.4: LES solutions showing the effects of noise in the unresolved-scale terms.

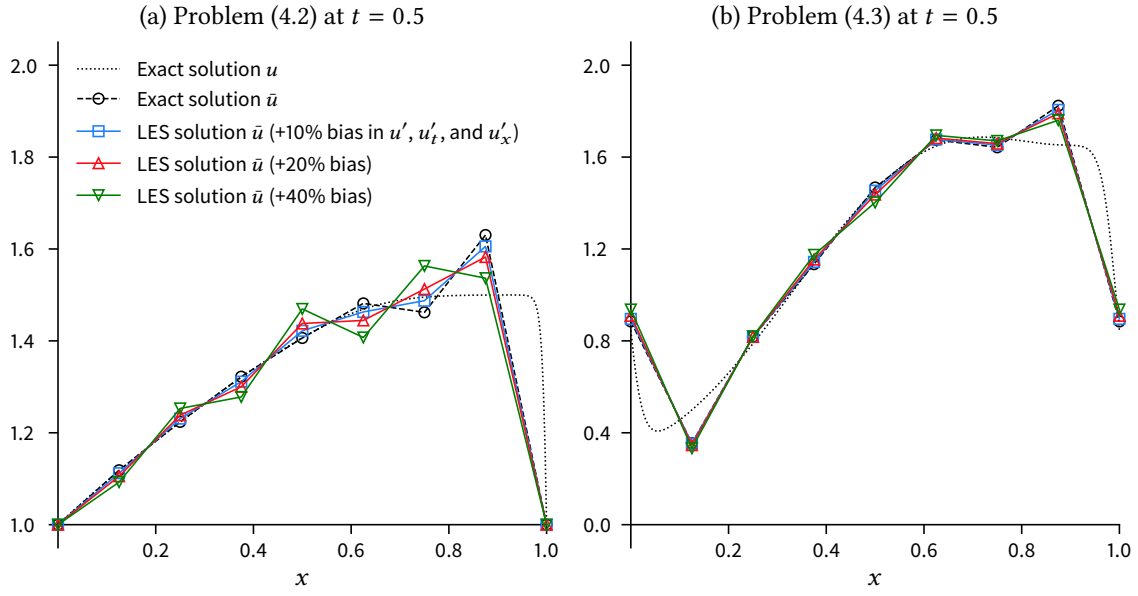

Figure 4.5: LES solutions showing the effect of a positive bias in the unresolved-scale terms.

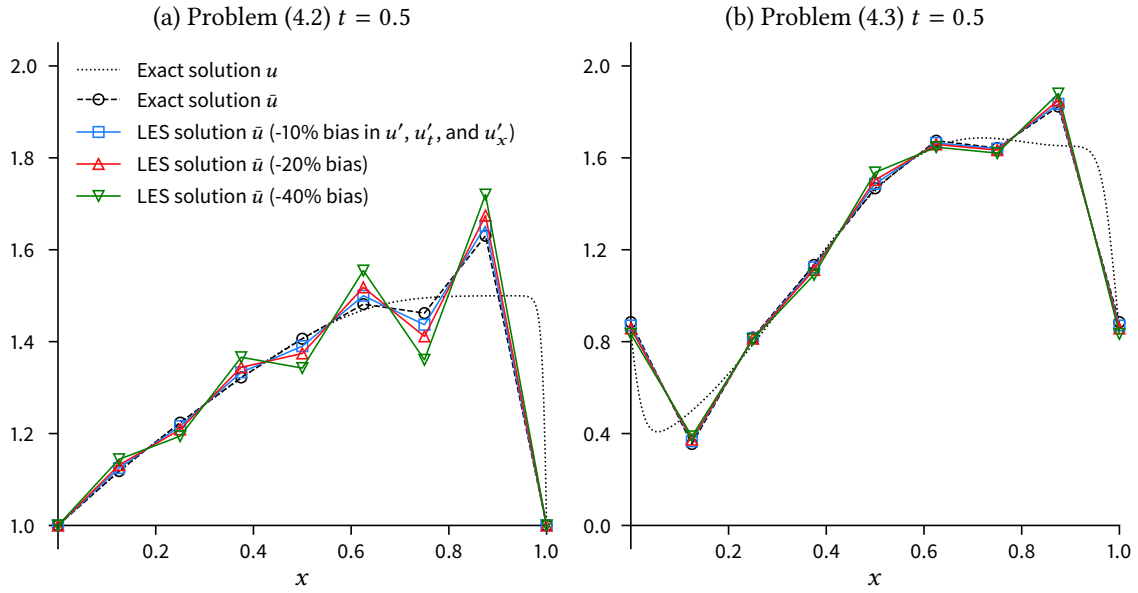

Figure 4.6: LES solutions showing the effect of a negative bias in the unresolved-scale terms.

4.7 Chapter Conclusion

It can be concluded that the individual unresolved-scale terms are not equally important; the u'-term is dominant, followed by the u'_x-term. The significance of the u'_t-term strongly depends on the Courant number. If the Courant number is small (e.g., $C = 0.01$), then the u'_t-term is negligible because the unresolved scales are quasi-steady. If $C = 0.1$, then the u'_t-term is no longer negligible. The sensitivity to the u'_t-term can be explained by recognizing that the u'_t-term represents work done to the system. It therefore performs a direct addition or removal of energy which is not the case for the u'- and u'_x-terms. Generally speaking, none of the unresolved-scale terms are negligible and a neural network must be trained to predict all three terms. The u'_t-term can only be removed from the model if the Courant number is sufficiently small.

The large-scale solution $\bar{u}$ is virtually unaffected by noisy unresolved-scale-term predictions as long as the mean of the predictions remains unchanged. The solution is much more sensitive to biased predictions, especially in problems where the unresolved-scales are purely dissipative, because biased predictions result in a shortage or surplus of energy with respect to the exact solution. The finding that the solution is virtually unaffected by noisy predictions is a strong motivation for the application of neural-network-based models because the predictions of a neural network are inherently noisy.

5 Learning the u'-Term and the u'_x-Term

As mentioned in Chapter 4, a universal model for the unresolved scales would correctly infer the integral form of the unresolved-scale terms regardless of the forcing function, boundary conditions, initial condition, Reynolds number, mesh spacing, and timestep. The purpose of this chapter is training a neural network that satisfies a subset of these requirements. The insights gained from solving a simplified problem are used to solve more general problems in the two subsequent chapters.

The scope of the model is limited by fixing the Reynolds number, mesh spacing, and timestep; specifically, $\mathrm{Re} = 100$, $h_{\mathrm{LES}} = \frac{1}{8}$, and $\Delta t_{\mathrm{LES}} = \Delta t_{\mathrm{DNS}} = 0.001$, which corresponds to a Courant number of 0.01. As demonstrated in Chapter 4, the u'_t-term can be omitted at this Courant number, leaving only the u'- and u'_x-terms to be modeled. Fixing the Reynolds number and mesh spacing removes two variables from the problem and thereby simplifies the complex relationships between the inputs and outputs.

The neural network is trained on a dataset obtained from a training problem. Part of the dataset is set aside as the validation set and used to determine the optimal hyperparameters of the model. Finally, the model is tested in two test problems that are not seen in training. This is done to assert that the model is capable of generalizing to new situations.

5.1 Inputs and Outputs

As discussed in Section 3.2, the set of input features can consist of a local stencil or an extended stencil. The local stencil contains input features local to the element under consideration, i.e.,

$$\text{input} = \left\{ \bar{u}_i, \bar{u}_{i+1}, \left(\frac{\partial \bar{u}}{\partial t}\right)_i, \left(\frac{\partial \bar{u}}{\partial t}\right)_{i+1}, f_i, f_{i+1} \right\} \tag{5.1}$$

whereas the extended stencil contains features of the two adjacent (left and right) elements in addition to the features of the local stencil, i.e.,

$$\text{input} = \left\{ \bar{u}_{i-1}, \bar{u}_i, \bar{u}_{i+1}, \bar{u}_{i+2}, \left(\frac{\partial \bar{u}}{\partial t}\right)_{i-1}, \left(\frac{\partial \bar{u}}{\partial t}\right)_i, \left(\frac{\partial \bar{u}}{\partial t}\right)_{i+1}, \left(\frac{\partial \bar{u}}{\partial t}\right)_{i+2}, f_{i-1}, f_i, f_{i+1}, f_{i+2} \right\} \tag{5.2}$$

Since the u'_t-term is omitted in the current chapter, the outputs are two real-valued numbers given by

$$\text{output} = \left\{ \left(w_x, \bar{u}^2 + \frac{1}{2} u'^2 \right), \left(w_x, u'_x \right) \right\} \tag{5.3}$$

where w_x is the gradient of one of the two weighting functions of the element. As explained in Section 2.2, the outputs corresponding to the remaining weighting function are obtained by flipping the sign of the prediction.

5.2 Training

The training examples must capture all phenomena that are encountered in the future application of the model. The main phenomena in Burgers' equation is the formation of shocks (i.e., large gradients $\frac{\partial u}{\partial x}$) because the dissipation of energy takes place at the length scales of these shocks. The training problem, given by

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = 3\sin(3\pi x)\cos(2\pi t), & x \in (0,1), t \in (0,5], \mathrm{Re} = 100 \\ u(x,0) = 1, & x \in (0,1) \\ u(0,t) = 1 + 0.75\sin(0.4\pi t), & t \in (0,5] \\ u(1,t) = 1 - 0.75\sin(0.4\pi t), & t \in (0,5] \end{cases} \tag{5.4}$$

exhibits a wide variety of shocks as a result of the highly unsteady forcing function and boundary conditions. The forcing function and boundary conditions in (5.4) are deliberately chosen to create a dataset in which all sorts of phenomena are present, including local backscatter.

The exact solution to (5.4) is obtained by DNS using $h_{\mathrm{DNS}} = \frac{1}{1024}$ and $\Delta t_{\mathrm{DNS}} = 0.001$. The training set is derived from the exact solution by evaluating the unresolved-scale terms using $h_{\mathrm{LES}} = \frac{1}{8}$ and $\Delta t_{\mathrm{LES}} = \Delta t_{\mathrm{DNS}} = 0.001$. The dataset consists of 40 000 examples, since the LES-mesh consists of 8 elements and using $\Delta t_{\mathrm{LES}} = 0.001$ for $t \in (0, 5]$ yields a total of 5000 timesteps. As recommended by Ng [29], 20% of the examples are set aside as the validation set. The training set therefore consists of 32 000 examples and the validation set consists of 8000 examples.

5.3 Validation

The most important hyperparameter is the topology of the neural network, i.e., the number of hidden layers and the number of hidden units per hidden layer. The mean absolute errors (MAEs) of 20 arrangements are given in Table 5.1. To put the MAEs into perspective, consider that the outputs are scaled to a range of approximately $[-1, 1]$ as a result of the normalization step. The results in Table 5.1 show that there is a clear benefit to adding hidden layers over adding units to a single hidden layer. For example, in spite of its much lower capacity, model #30 (3474 trainable parameters) yields a lower MAE on the validation set than model #25 (61 442 trainable parameters). Furthermore, the errors on the training set are close to the errors on the validation set. This indicates that none of the models overfit the training data and it is therefore unnecessary to employ techniques to combat overfitting (e.g., regularization and dropout). Comparing the MAEs in Table 5.1a and 5.1b, it is clear that the errors of the models trained on examples using the extended stencil are consistently lower than the errors of the models trained on examples using the local stencil. This undeniably proves that the additional data in the extended stencil contains useful information.

The best performing model in Table 5.1 is neither the model with the most trainable parameters nor the model with the most layers; the best performing model is model #31 with a modest 9282 learnable parameters and 3 hidden layers of 64 units per layer. An overview of the remaining hyperparameters is given in Table 5.2; motivations for selecting these hyperparameters are given in Section 3.2.6.

(a) Local stencil

#	Hidden layers and units	Parameters	Training MAE	Validation MAE
1	{16}	146	0.1549	0.1565
2	{64}	578	0.0875	0.0899
3	{256}	2306	0.0563	0.0560
4	{1024}	9218	0.0383	0.0397
5	{4096}	36 866	0.0378	0.0373
6	{16, 16}	418	0.1048	0.1058
7	{64, 32}	2594	0.0425	0.0434
8	{64, 64}	4738	0.0323	0.0296
9	{16, 16, 16}	690	0.0722	0.0730
10	{64, 32, 16}	3090	0.0327	0.0367
11	{64, 64, 64}	8898	0.0211	0.0214
12	{16, 16, 16, 16}	962	0.0625	0.0646
13	{64, 64, 32, 16}	7250	0.0212	0.0218
14	{64, 64, 64, 64}	13 058	0.0177	0.0194
15	{16, 16, 16, 16, 16}	1234	0.0612	0.0597
16	{64, 64, 64, 32, 16}	11 410	0.0197	0.0207
17	{64, 64, 64, 64, 64}	17 218	0.0186	0.0191
18	{16, 16, 16, 16, 16, 16}	1506	0.0531	0.0562
19	{64, 64, 64, 64, 32, 16}	15 570	0.0189	0.0200
20	{64, 64, 64, 64, 64, 64}	21 378	0.0187	0.0198

(b) Extended stencil

#	Hidden layers and units	Parameters	Training MAE	Validation MAE
21	{16}	242	0.0823	0.0829
22	{64}	962	0.0354	0.0359
23	{256}	3842	0.0218	0.0221
24	{1024}	15 362	0.0166	0.0171
25	{4096}	61 442	0.0163	0.0163
26	{16, 16}	514	0.0484	0.0494
27	{64, 32}	2978	0.0182	0.0184
28	{64, 64}	5122	0.0147	0.0161
29	{16, 16, 16}	786	0.0441	0.0444
30	{64, 32, 16}	3474	0.0174	0.0161
31	{64, 64, 64}	9282	0.0134	0.0132
32	{16, 16, 16, 16}	1058	0.0355	0.0358
33	{64, 64, 32, 16}	7634	0.0145	0.0136
34	{64, 64, 64, 64}	13 442	0.0141	0.0144
35	{16, 16, 16, 16, 16}	1330	0.0324	0.0312
36	{64, 64, 64, 32, 16}	11 794	0.0145	0.0139
37	{64, 64, 64, 64, 64}	17 602	0.0149	0.0145
38	{16, 16, 16, 16, 16, 16}	1602	0.0362	0.0372
39	{64, 64, 64, 64, 32, 16}	15 954	0.0153	0.0140
40	{64, 64, 64, 64, 64, 64}	21 762	0.0162	0.0174

Table 5.1: Comparison of the mean absolute errors (MAEs) of 40 different models.

Dataset	Training examples	32 000
	Validation examples	8000
Model	Type	Densely connected neural network
	Input units	4 (local stencil) or 12 (extended stencil)
	Output units	2
	Hidden layers and units	See Table 5.1
	Trainable parameters	See Table 5.1
	Activation function	`relu`
	Regularization	N/A
	Dropout	N/A
Training algorithm	Optimizer	`rmsprop`
	Loss function	`mse`
	Batch size	N/A
	Epochs	100
Performance	Training MAE	See Table 5.1
	Validation MAE	See Table 5.1

Table 5.2: Summary of the dataset, hyperparameters, and performance of the neural network.

5.3.1 Visualizing the Performance on the Training and Validation Sets

To make a qualitative assessment on how well the models in Table 5.1 match the examples, a time series of predictions is compared to the exact solution. Figure 5.1 shows a time series plot of the predictions of model #1 (i.e., the "local stencil, low capacity model) and the predictions of model #31 (i.e., the "extended stencil, high capacity model"). Notice that model #1 is the worst performing model and model #31 is the best performing model in Table 5.1. For clarity, Figure 5.1 only shows the predictions for the right-most element of the domain (i.e., $x = [0.875, 1]$). Figure 5.1 shows that both the low- and high-capacity model reproduce the behavior of the exact solution. The predictions of the high-capacity model almost perfectly match the ground truth data whereas the predictions of the low-capacity model exhibit more noise. The models are expected to perform well because these examples are part of the training and validation set. The challenge is training a model that performs well in the a posteriori tests.

5.4 Performance Evaluation

The models are tested on two test problems that are substantially different from the problem used to generate the training examples. This is done to assess the ability of the models to generalize to new situations. There are two methods for testing the model on the two test problems:

1. Assembling a test set by DNS of the two test problems and evaluating the neural network on these inputs. The model is tested on individual data points.
2. Running an LES of the test problems where the neural network is invoked at runtime of the LES. The model is tested within an actual LES.

Method (1) allows for a direct comparison between predictions and the exact solution, but the model is not evaluated within an actual LES. In Method (2), the model is evaluated within an actual LES which allows erroneous predictions to affect the further course of the simulation.

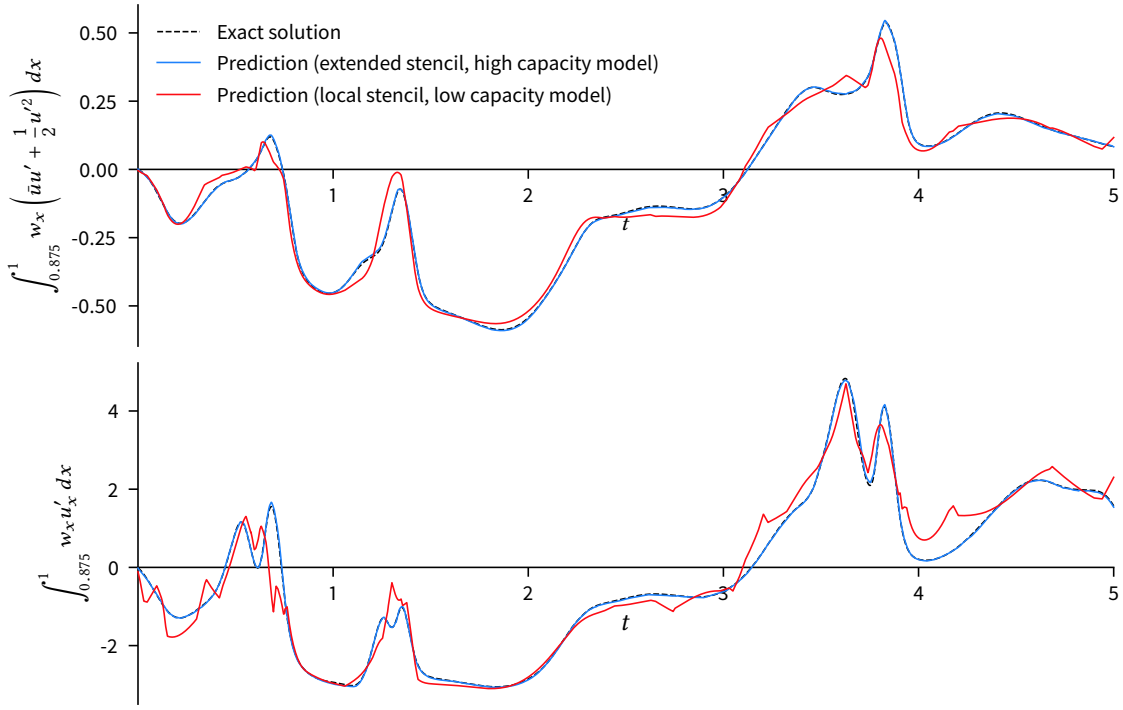

Figure 5.1: Time series of the predicted and exact u'- and u'_x-terms in Problem (5.4) on the right-most element of the domain, i.e., $x \in (0.875, 1)$.

5.4.1 Test Problems

The two test problems are categorically different from the model problem in (5.4). The first test problem, employing a steady forcing function and Dirichlet boundary conditions, is given by

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = 1, & x \in (0,1), t \in (0,2], \mathrm{Re} = 100 \\ u(x,0) = 1, & x \in (0,1) \\ u(0,t) = 1, & t \in (0,2] \\ u(1,t) = 1, & t \in (0,2] \end{cases} \tag{5.5}$$

and the second test problem, employing an unsteady forcing function and periodic boundary conditions, is given by

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = 20(x^3 - x^6)\sin(2\pi t), & x \in (0,1), t \in (0,2], \mathrm{Re} = 100 \\ u(x,0) = 1 + \sin(2\pi x), & x \in (0,1) \\ u(0,t) = u(1,t), & t \in (0,2] \end{cases} \tag{5.6}$$

The forcing functions and boundary conditions in (5.5) and (5.4) clearly have no resemblance to those in the training problem. To perform well on the above two test problems, the model must learn a fundamental underlying relationship between the input features and the unresolved-scale terms.

5.4.2 Performance on Individual Data Points

A comparison of the predictions of the high-capacity model and the low-capacity model is given in Figure 5.2. These results are obtained by running the model on all individual data points; errors do therefore not affect the future evolution of the solution. Figure 5.2 shows that the model is certainly not perfect; the low-capacity model is biased in Figure 5.2a and both models yield a significant error between $t = 0.25$ and $t = 0.5$ in Figure 5.2b. This timeframe overlaps with the timeframe when the shock traverses through the element. The local conditions change dramatically when the shock enters the element through the left boundary and when the shock leaves the element through the right boundary. The model is not fully prepared for a steep moving shock even though the training problem features moving shocks. Nonetheless, Figure 5.2 shows that both models reproduce the overall behavior of the exact solution. This confirms that both models have the ability to generalize to new situations.

5.4.3 Performance at Runtime of an LES

Figure 5.3 shows the solutions to problems (5.5) and (5.4) at two separate timesteps. The LES solutions in Figure 5.3 are obtained by invoking the neural networks at runtime of the LES. Erroneous predictions do therefore affect the future evolution of the solution. Figure 5.3 shows that the LES solution using the high-capacity model is in excellent agreement with the exact solutions whereas the LES solutions using the low-capacity model have a noticeable error, especially near the steep gradient in Figure 5.2b. Nonetheless, even the low-capacity model is a dramatic improvement over the LES solution obtained without an unresolved-scale model. These results show that small errors in the predictions do not necessarily build up over time and cause the solution to diverge from the exact solution.

5.5 Chapter Conclusion

Training a neural network to learn the u'- and u'_x-terms of Burgers' equation has been a success. Setting the hyperparameters as recommended by Chollet results in a fully functional unresolved-scale model, regardless of whether a high-capacity model or a low-capacity model is used. This is a remarkable result because the low capacity model is the worst model in Table 5.1 and the high capacity model is the best model in Table 5.1. This implies that the underlying functional relationship is rather simple; it does not require a large network to learn a functional unresolved-scale model.

As recommended by Duraisamy et al., the models are not only tested on individual data points, but also within an actual LES. Both the low- and high-capacity models successfully pass the a posteriori tests and yield a dramatically improved solution in comparison to the solution without an unresolved-scale model. It can therefore be concluded that both models have successfully learned a functional relationship between the input features and the unresolved-scale terms. The models are not perfect, but errors in the predictions do not significantly change the evolution of $\bar{u}$ compared to the exact solution.

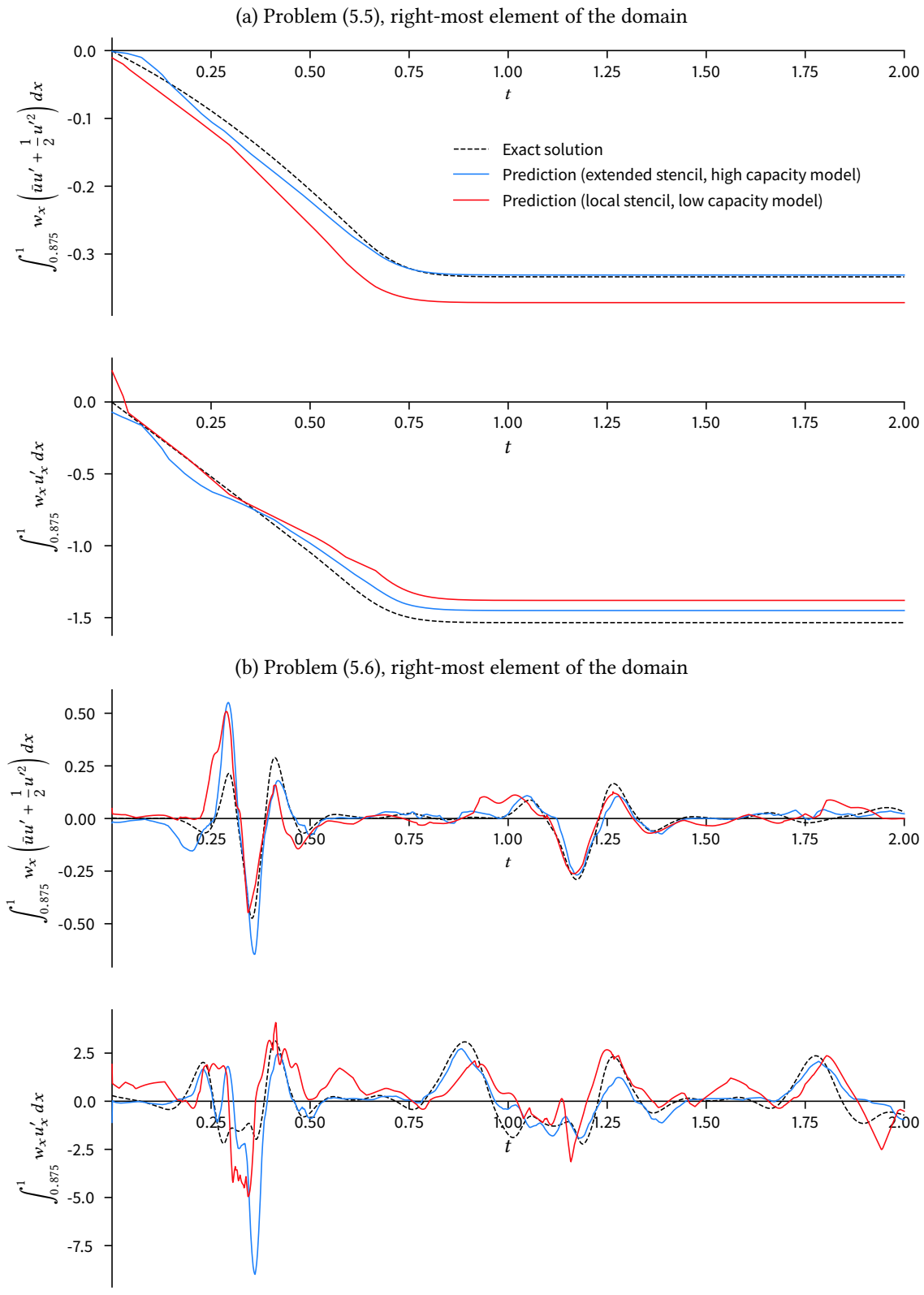

Figure 5.2: Time series of the predicted and exact u'- and u'_x-terms at individual data points on the right-most element of the domain, i.e., $x \in (0.875, 1)$.

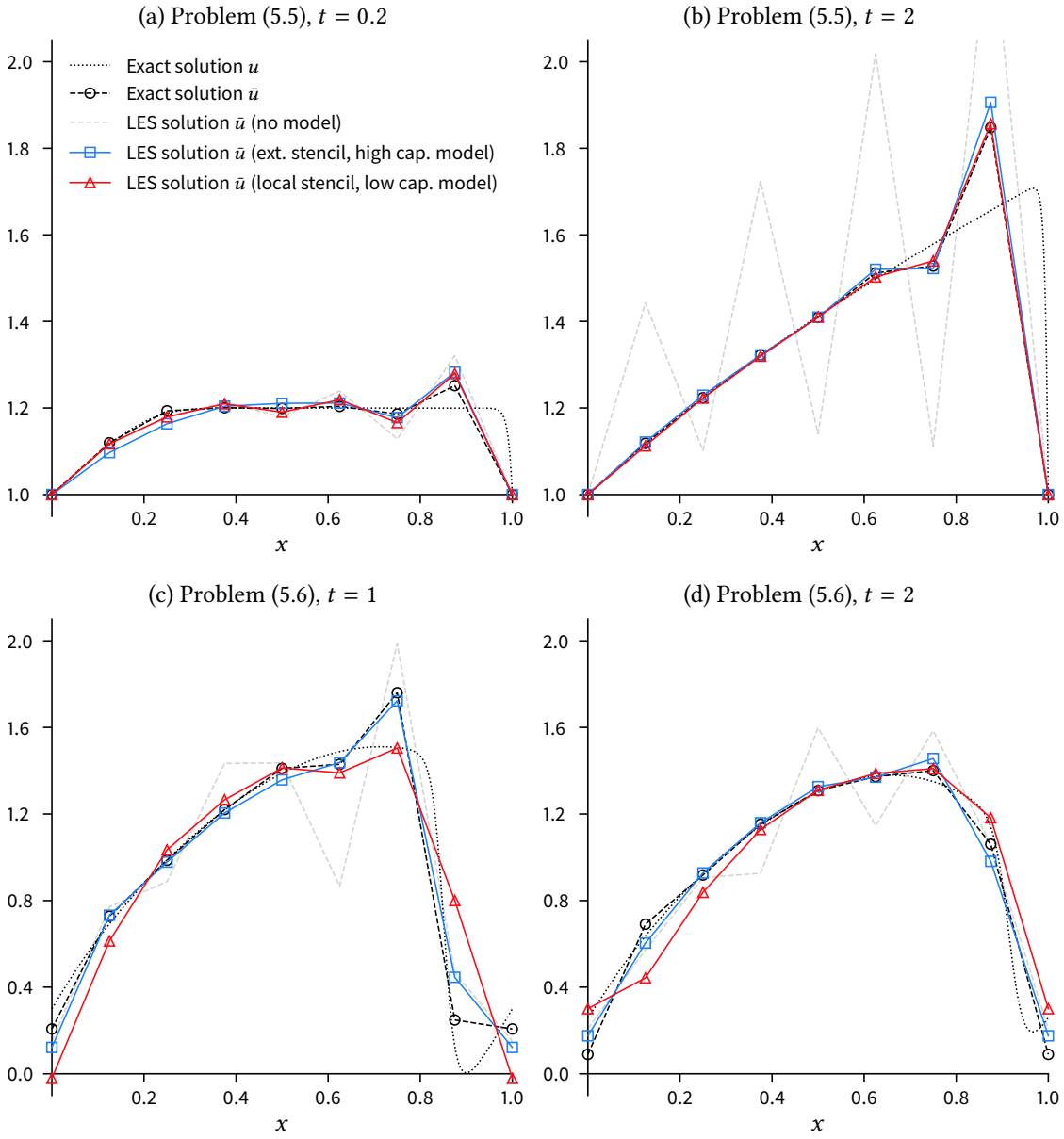

Figure 5.3: LES solutions to Problem (5.5) and (5.6) at separate timesteps obtained by testing the neural network within the simulation.

6 Learning the u_t'-Term

The results of Chapter 4 show that the u_t'-term must be modeled for an accurate reconstruction of $\bar{u}$ when the Courant number is 0.1 or greater. In most LESs of engineering applications, the timestep and associated Courant number are much greater than in DNSs because the time scales of engineering interest (e.g., for the computation of aeroelastic vibrations) are much greater than the small timesteps required for DNSs. Thus, there is a strong motivation to extend the model with the ability of inferring the u_t'-term.

6.1 Inputs and Outputs

Given the success of the extended stencil in Chapter 5, it is unnecessary to continue using the local stencil. There is therefore only one set of input features given by

$$\text{input} = \left\{ \bar{u}_{i-1}, \bar{u}_i, \bar{u}_{i+1}, \bar{u}_{i+2}, \left(\frac{\partial u}{\partial t}\right)_{i-1}, \left(\frac{\partial u}{\partial t}\right)_i, \left(\frac{\partial u}{\partial t}\right)_{i+1}, \left(\frac{\partial u}{\partial t}\right)_{i+2}, f_{i-1}, f_i, f_{i+1}, f_{i+2} \right\} \tag{6.1}$$

Compared to the set of outputs in Chapter 5, the set of outputs in the present chapter is extended by the two u_t'-terms. Thus, the set of outputs is given by

$$\text{output} = \left\{ \left(w_x, \bar{u}^2 + \frac{1}{2} u'u' \right), \left(w_l, u_t' \right), \left(w_r, u_t' \right), \left(w_x, u_x' \right) \right\} \tag{6.2}$$

where w_l and w_r are respectively the left and right weighting functions of a given element. For reasons given in Section 2.2, w_x is the gradient of only one of these two weighting functions.

6.2 Training

A summary of the dataset, hyperparameters, and performance of the neural network is given in Table 6.1. The training problem remains unchanged from Chapter 5, i.e.,

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = 3\sin(3\pi x)\cos(2\pi t), & x \in (0,1), t \in (0,5], \mathrm{Re} = 100 \\ u(x,0) = 1, & x \in (0,1) \\ u(0,t) = 1 + 0.75\sin(0.4\pi t), & t \in (0,5] \\ u(1,t) = 1 - 0.75\sin(0.4\pi t), & t \in (0,5] \end{cases} \tag{6.3}$$

In the same vein as in Chapter 5, the exact solution to (6.3) is obtained by DNS using $h_{\mathrm{DNS}} = \frac{1}{1024}$ and $\Delta t_{\mathrm{DNS}} = 0.001$. The training examples are derived from the DNS results using $h_{\mathrm{LES}} = \frac{1}{8}$ and $\Delta t_{\mathrm{LES}} = 0.01$, which corresponds to a Courant number of 0.1. The number of examples reduces from 40 000 to 4000 because there are 500 distinct LES timesteps rather than 5000 in Chapter 5.

Dataset	Training examples	3200
	Validation examples	800
Model	Type	Densely connected neural network
	Input units	12
	Output units	4
	Hidden layers and units	{64, 64, 64}
	Trainable parameters	9412
	Activation function	`relu`
	Regularization	N/A
	Dropout	N/A
Training algorithm	Optimizer	`rmsprop`
	Loss function	`mse`
	Batch size	N/A
	Epochs	300
Performance	Training MAE	0.0328
	Validation MAE	0.0334

Table 6.1: Summary of the dataset, hyperparameters, and performance of the neural network.

6.3 Validation

As shown in Table 6.1, 20% of the examples are set aside as the validation set. The model achieves a MAE of 0.0328 on the training set and a MAE of 0.0334 on the validation set. To put these numbers in context, consider that the outputs are normalized to approximately the range $[-1, 1]$. The fact that the training MAE is virtually equal to the validation MAE indicates that the model does not overfit the training data. The high-capacity model of Chapter 5, on which the current model is based, achieves training and validation MAEs of respectively 0.0134 and 0.0132. The difference in training and validation MAEs between the current and previous chapters is a result of the greatly reduced number of examples. A comparison of the predicted and exact u_t'-terms in the training problem is given in Figure 6.1. There is an excellent match between the predictions and the exact solutions which implies that a neural network can successfully learn the u_t'-terms based on the inputs in (6.1).

6.4 Performance Evaluation

After training and validation, the model is tested in previously unseen problems to assert its ability to generalize to new situations. To this end, the two test problems from Chapter 5 are used. The first test problem is given by

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = 1, & x \in (0,1), t \in (0,2], \mathrm{Re} = 100 \\ u(x,0) = 1, & x \in (0,1) \\ u(0,t) = 1, & t \in (0,2] \\ u(1,t) = 1, & t \in (0,2] \end{cases} \tag{6.4}$$

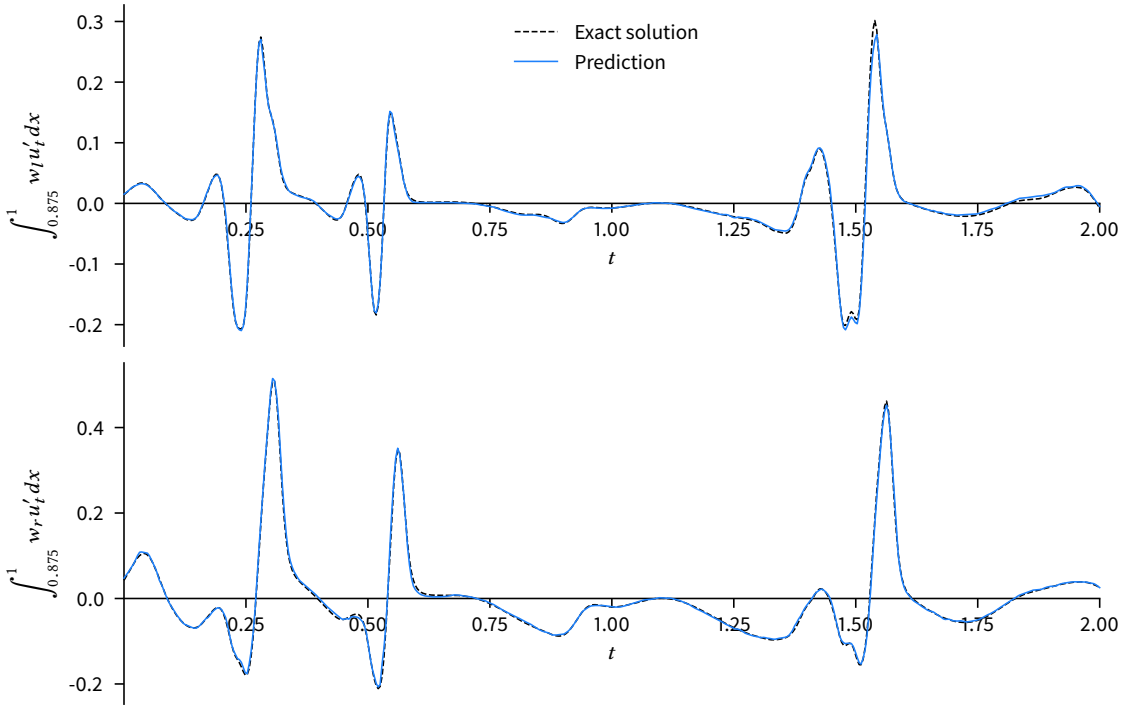

Figure 6.1: Time series of the predicted and exact u_t'-terms on the right-most element of the domain in Problem (6.3), i.e., $x \in (0.875, 1)$.

and the second test problem is given by

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = 20(x^3 - x^6)\sin(2\pi t), & x \in (0,1), t \in (0,2], \mathrm{Re} = 100 \\ u(x,0) = 1 + \sin(2\pi x), & x \in (0,1) \\ u(0,t) = u(1,t), & t \in (0,2] \end{cases} \tag{6.5}$$

As discussed in Chapter 5, there are two methods to test the model: (1) testing on individual data points, and (2) testing within an actual LES.

6.4.1 Performance on Individual Data Points

Figure 6.2 shows the predictions of the model based on individual data points obtained by DNS of Problem (6.5). The predictions do match the large swings of the exact solution, but there are high localized errors, especially between $t = 0.25$ and $t = 0.50$. The errors in predicting the u'- and u_x'-terms are comparable to the errors in Figure 5.2b. Based on the qualitative findings of Figure 6.2, the model in the current chapter is comparable in performance to the model in Chapter 5, despite the much smaller dataset.

6.4.2 Performance at Runtime of an LES

An LES of Problem (6.4) runs without issue and yields the same results as in Figure 5.3. This is no surprise, as $(w, u_t') \approx 0$ in Problem (6.4) and the model is able to predict this. Difficulties

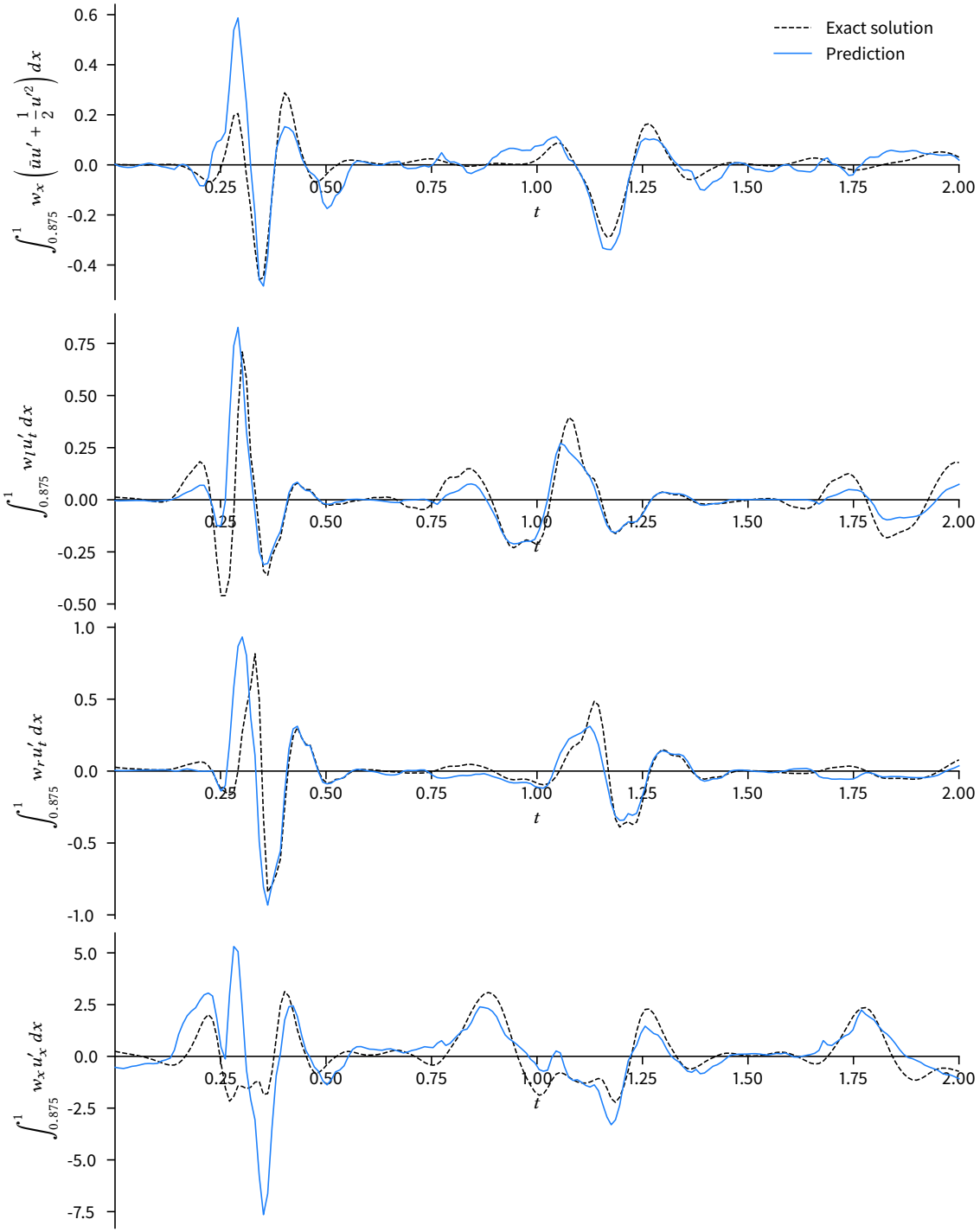

Figure 6.2: Time series of the predicted and exact unresolved-scale terms at individual data points on the right-most element of the domain in Problem (6.5), i.e., $x \in (0.875, 1)$.

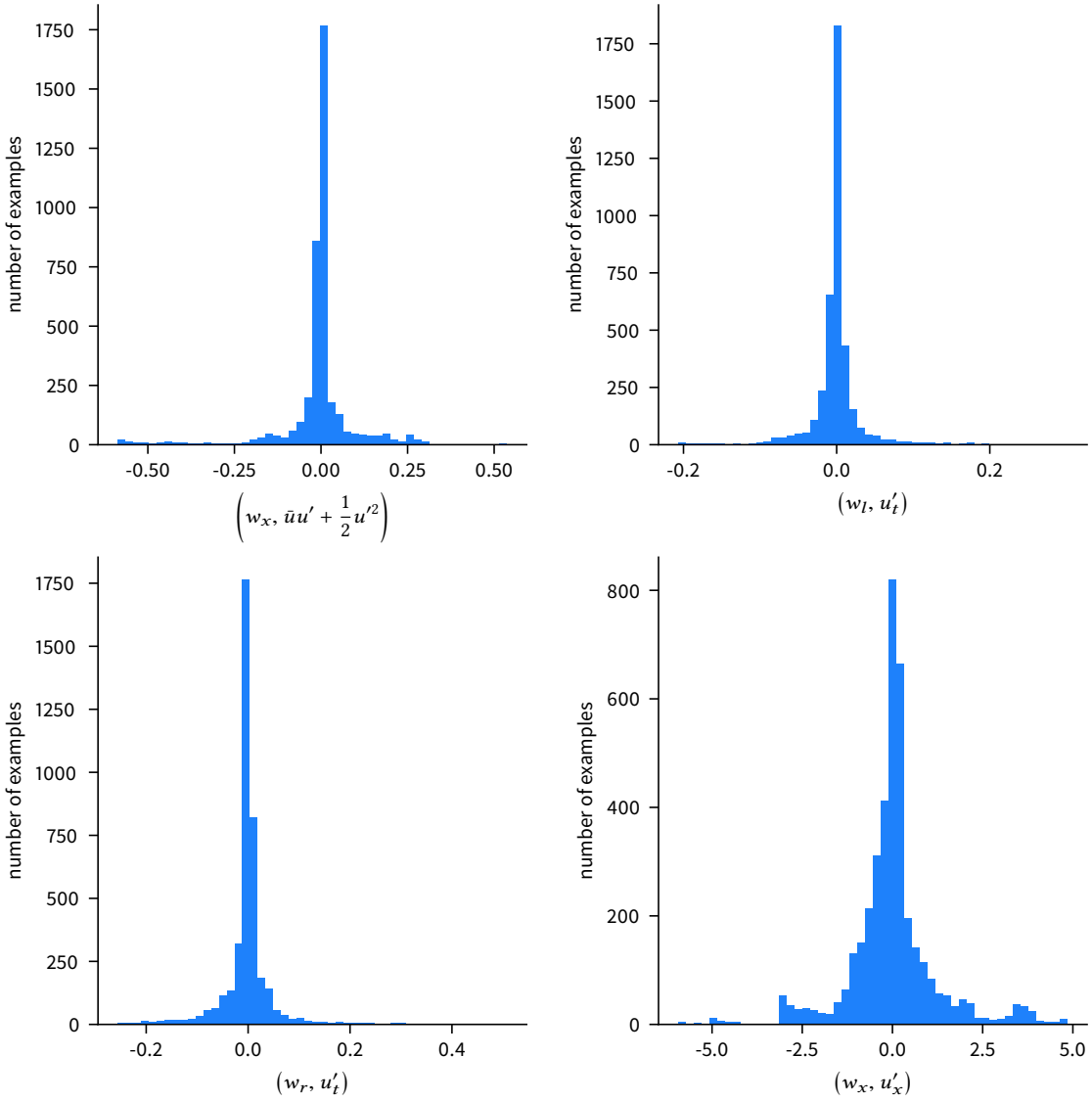

Figure 6.3: Distributions of the unresolved-scale terms in the training set.

emerge when running an LES of Problem (6.5) where $(w, u'_t) \neq 0$. Erroneous predictions cause the solution to grow uncontrollably. The predictions for the (w, u'_t)-term appear to trigger this behavior because the solution is reasonably accurate when the predicted u'_t-terms are omitted. A potential cause is that the intermediate results in the Newton process do not have a physical meaning. Physically meaningless examples are not part of the training set and the neural network model is therefore not trained to handle these inputs.

6.5 Stabilization

Neural networks are statistical tools and erroneous predictions are unavoidable, regardless of how well the neural network is trained. The failure to run an LES of (6.5) shows that a procedure is necessary to handle erroneous predictions. Figure 6.2 shows that the erroneous predictions far exceed the normal range of the unresolved-scale terms. For example, the exact u'_x-term has a

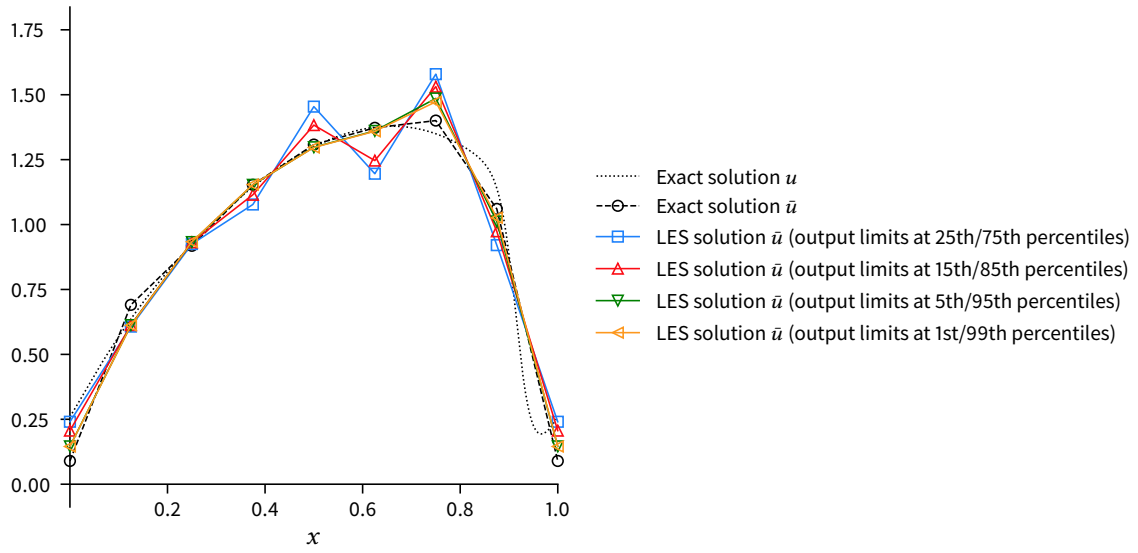

Figure 6.4: Comparison of lower and upper limits to the neural-network outputs.

range of approximately $[-2.5, 2.5]$ whereas the predictions have outliers as low as -7.5 and as high as 5. A rudimentary stabilization scheme could therefore be limiting the unresolved-scale terms to prevent the outliers present in Figure 6.2. Figure 6.3 shows that the unresolved-scale terms in the training set approximately follow a binomial distribution centered around 0. It therefore makes sense to set the limits at certain percentiles of these distributions.

Imposing limits on the neural-network predictions indeed prevents the solution from growing out of control. Figure 6.4 shows that setting the lower and upper limits at respectively the 1st and 99th percentiles yields the closest result to the exact solution. A second try to run LESs of the test problems with limits in place yields the results in Figure 6.5. The results are in excellent agreement with the exact solutions and even slightly superior to the results in Chapter 5.

6.6 Chapter Conclusion

The neural work is able to learn a functional relationship between the input features and the u_t'-terms just as well as it is able to learn a functional relationship between the input features and the other unresolved-scale terms. Compared to Chapter 5, the number of training examples is reduced by a factor of 10 from 32 000 to 3200 and the number of outputs is doubled from 2 to 4. In spite of this, the neural network does not underperform the neural network trained in Chapter 5. This confirms that the number of training examples can safely be reduced to 3200.

Testing the neural network within an LESs leads to problems. In an LES of Problem (6.4), erroneous predictions cause the solution to grow uncontrollably. A rudimentary stabilization scheme where the neural-network predictions are limited prevents the solution from growing uncontrollably. Setting the lower and upper limits to respectively the 1st and 99th percentiles of the training dataset yields results closest to the exact solutions. Using this stabilization scheme, the LES of Problem (6.4) runs without issues. These results show that the model has the ability to generalize even if the Courant number is increased to 0.1 and predictions of the u_t'-terms are necessary.

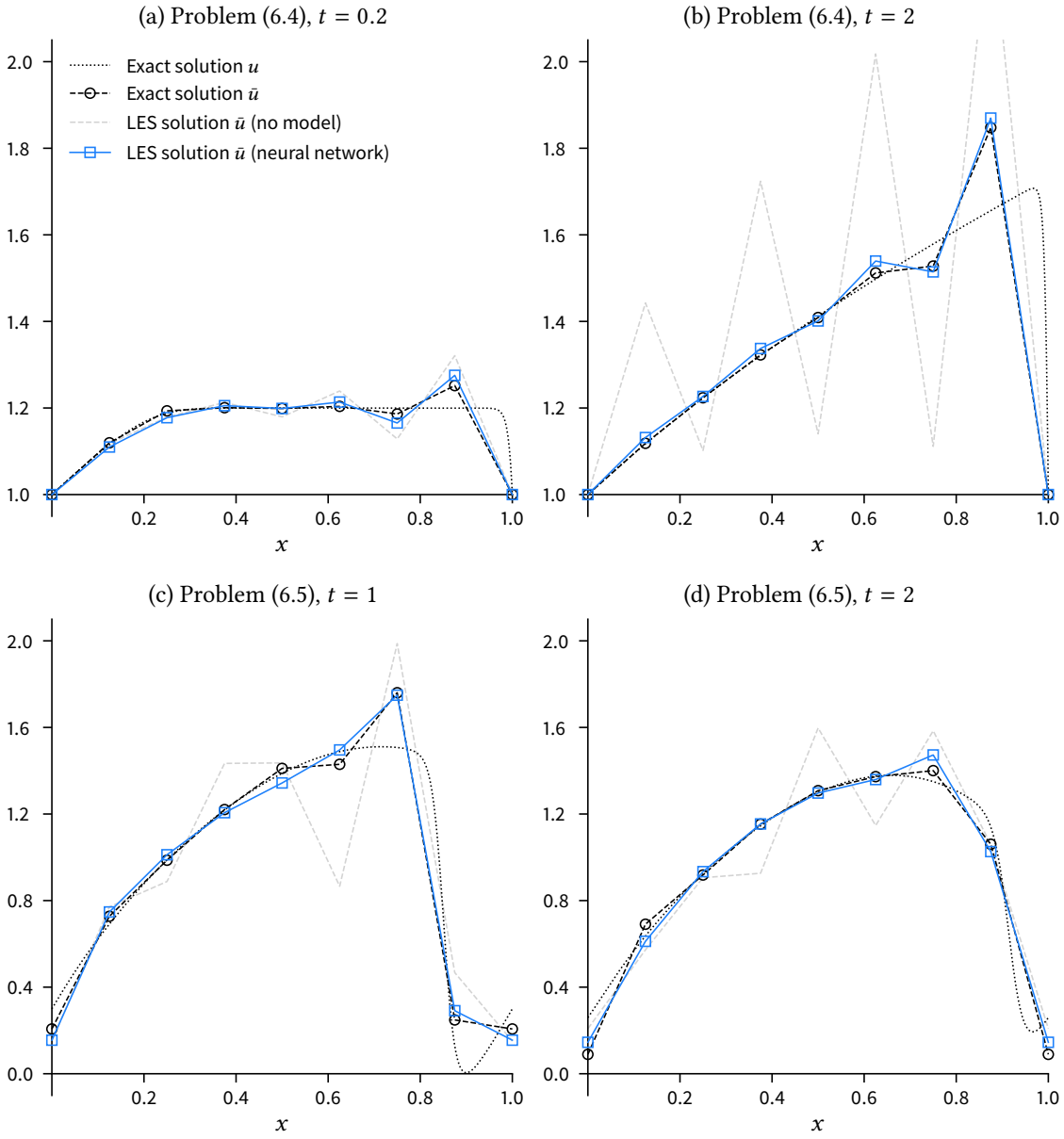

Figure 6.5: LES solutions to Problem (6.4) and (6.5) at separate timesteps obtained by testing the neural network within the simulation.

7 Generalization of Reynolds Number and Element Size

In Chapter 5 and 6, the scope of the model is limited by using a constant Reynolds number and element size (specifically, $\mathrm{Re} = 100$ and $h_{\mathrm{LES}} = \frac{1}{8}$). In the current chapter, a neural network is trained to learn a functional relationship that takes the Reynolds number and element size into account. Thus, the model is almost completely universal; there are no restrictions on the forcing function, boundary conditions, initial condition, Reynolds number, and element size.

7.1 Inputs and Outputs

Compared to Chapter 5, the input features are extended by the Reynolds number and the mesh spacing, i.e.,

$$\text{input} = \left\{ \bar{u}_{i-1}, \bar{u}_i, \bar{u}_{i+1}, \bar{u}_{i+2}, \left(\frac{\partial \bar{u}}{\partial t}\right)_{i-1}, \left(\frac{\partial \bar{u}}{\partial t}\right)_{i}, \left(\frac{\partial \bar{u}}{\partial t}\right)_{i+1}, \left(\frac{\partial \bar{u}}{\partial t}\right)_{i+2}, f_{i-1}, f_i, f_{i+1}, f_{i+2}, \mathrm{Re}, h \right\} \tag{7.1}$$

A relevant non-dimensional number is the Peclet number given by

$$\mathrm{Pe} = \frac{uh}{\nu} \tag{7.2}$$

The cell Peclet number can in principle replace Re and h in (7.1). The output features remain unchanged from Chapter 5 and are given by

$$\text{output} = \left\{ \left(w_x, \bar{u}^2 + \frac{1}{2} u'u' \right), \left(w_x, u'_x \right) \right\} \tag{7.3}$$

7.2 Training

With the exception of the Reynolds number, which is no longer a constant, the training problem remains unchanged from Chapter 5, i.e.,

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = 3\sin(3\pi x)\cos(2\pi t), & x \in (0,1), t \in (0,5] \\ u(x,0) = 1, & x \in (0,1) \\ u(0,t) = 1 + 0.75\sin(0.4\pi t), & t \in (0,5] \\ u(1,t) = 1 - 0.75\sin(0.4\pi t), & t \in (0,5] \end{cases} \tag{7.4}$$

Reynolds number	Mesh spacing	Peclet number	Possible examples	Actual examples
$\mathrm{Re} = 10$	$h_{\mathrm{LES}} = \frac{1}{6}$	$\mathrm{Pe} = 1.67$	30 000	30 000
$\mathrm{Re} = 10$	$h_{\mathrm{LES}} = \frac{1}{8}$	$\mathrm{Pe} = 1.25$	40 000	30 000
$\mathrm{Re} = 10$	$h_{\mathrm{LES}} = \frac{1}{12}$	$\mathrm{Pe} = 0.83$	60 000	30 000
$\mathrm{Re} = 100$	$h_{\mathrm{LES}} = \frac{1}{6}$	$\mathrm{Pe} = 16.7$	30 000	30 000
$\mathrm{Re} = 100$	$h_{\mathrm{LES}} = \frac{1}{8}$	$\mathrm{Pe} = 12.5$	40 000	30 000
$\mathrm{Re} = 100$	$h_{\mathrm{LES}} = \frac{1}{12}$	$\mathrm{Pe} = 8.3$	60 000	30 000
$\mathrm{Re} = 1000$	$h_{\mathrm{LES}} = \frac{1}{6}$	$\mathrm{Pe} = 167$	30 000	30 000
$\mathrm{Re} = 1000$	$h_{\mathrm{LES}} = \frac{1}{8}$	$\mathrm{Pe} = 125$	40 000	30 000
$\mathrm{Re} = 1000$	$h_{\mathrm{LES}} = \frac{1}{12}$	$\mathrm{Pe} = 83$	60 000	30 000

Table 7.1: Overview of the configurations of the training problem.

Exact solutions to (7.4) are obtained by DNSs using $h_{\mathrm{DNS}} = \frac{1}{1024}$, $\Delta t_{\mathrm{DNS}} = 0.001$, and $\mathrm{Re} \in \{10, 100, 1000\}$. The gradients of the shocks do not change strongly when the Reynolds number is increased beyond $\mathrm{Re} = 1000$. Conversely, the necessity of an unresolved-scale model diminishes as the Reynolds number is smaller than $\mathrm{Re} = 10$. The neural network should therefore be able to learn both extremes from examples with $\mathrm{Re} \in \{10, 100, 1000\}$.

The training examples are derived from the exact solutions by evaluating the unresolved-scale terms using $h_{\mathrm{LES}} \in \left\{\frac{1}{6}, \frac{1}{8}, \frac{1}{12}\right\}$ and $\Delta t_{\mathrm{LES}} = \Delta t_{\mathrm{DNS}} = 0.001$, which corresponds to a Courant number of 0.01. As shown in Table 7.1, the number of examples per scenario is limited to 30 000 to avoid a bias towards small mesh spacings in the training set. The combined dataset consists of 270 000 examples. Setting aside 20% of the examples yields a training set consisting of 216 000 examples and a validation set consisting of 54 000 examples. The MAEs on the training and validation sets are respectively 0.0510 and 0.0489 which indicates that the model does not overfit the training set. A detailed validation is omitted because the predictions are on par with Figure 5.1 in Chapter 5. The hyperparameters of the neural network remain unchanged from Chapter 5 and are summarized in Table 7.2.

7.3 Performance Evaluation

The two test problems remain unchanged from Chapter 5. The first test problem is given by

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = 1, & x \in (0,1), t \in (0,2] \\ u(x,0) = 1, & x \in (0,1) \\ u(0,t) = 1, & t \in (0,2] \\ u(1,t) = 1, & t \in (0,2] \end{cases} \tag{7.5}$$

Dataset	Training examples	216 000
	Validation examples	54 000
Model	Type	Densely connected neural network
	Input units	14
	Output units	2
	Hidden layers and units	{64, 64, 64}
	Trainable parameters	9412
	Activation function	`relu`
	Regularization	N/A
	Dropout	N/A
Training algorithm	Optimizer	`rmsprop`
	Loss function	`mse`
	Batch size	N/A
	Epochs	100
Performance	Training MAE	0.0510
	Validation MAE	0.0489

Table 7.2: Summary of the dataset, hyperparameters, and performance of the neural network.

and the second test problem is given by

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = 20(x^3 - x^6)\sin(2\pi t), & x \in (0,1), t \in (0,2] \\ u(x,0) = 1 + \sin(2\pi x), & x \in (0,1) \\ u(0,t) = u(1,t), & t \in (0,2] \end{cases} \tag{7.6}$$

In contrast to Chapter 5 and 6, the Reynolds number and mesh spacing in (7.5) and (7.6) are variables. To assert the model's ability to generalize, (7.5) and (7.6) are solved for $\mathrm{Re} \in \{20, 2000\}$ and $h_{\mathrm{LES}} \in \left\{\frac{1}{7}, \frac{1}{15}\right\}$. These Reynolds numbers and mesh spacings are chosen to test the model at both low and high Reynolds numbers and at both large and small element lengths. Furthermore, $\mathrm{Re} = 2000$ and $h = \frac{1}{15}$ are well outside the scope of the training set to further test the model's ability to generalize.

7.3.1 Performance within an LES

The LES solutions to (7.5) and (7.6) are respectively given in Figure 7.1 and 7.2. The dissipative length scales can be almost completely resolved when $\mathrm{Re} = 20$ and no unresolved-scale model is required. This is confirmed in Figure 7.1a, 7.1b, 7.2a and 7.2b because the LES solutions sans model match the exact solutions. The LES solutions show that the neural network correctly models the absence of unresolved scales when $\mathrm{Re} = 20$. Conversely, an unresolved-scale model is absolutely necessary when $\mathrm{Re} = 2000$. The gradients of the shocks are steep and the dissipative length scales can no longer be resolved. This is confirmed in Figure 7.1c, 7.1d, 7.2c and 7.2d because the LES solutions sans model deviate strongly from the exact solutions. The neural network correctly models the unresolved-scale terms $\mathrm{Re} = 2000$ since the LES solutions are close to the exact solutions.

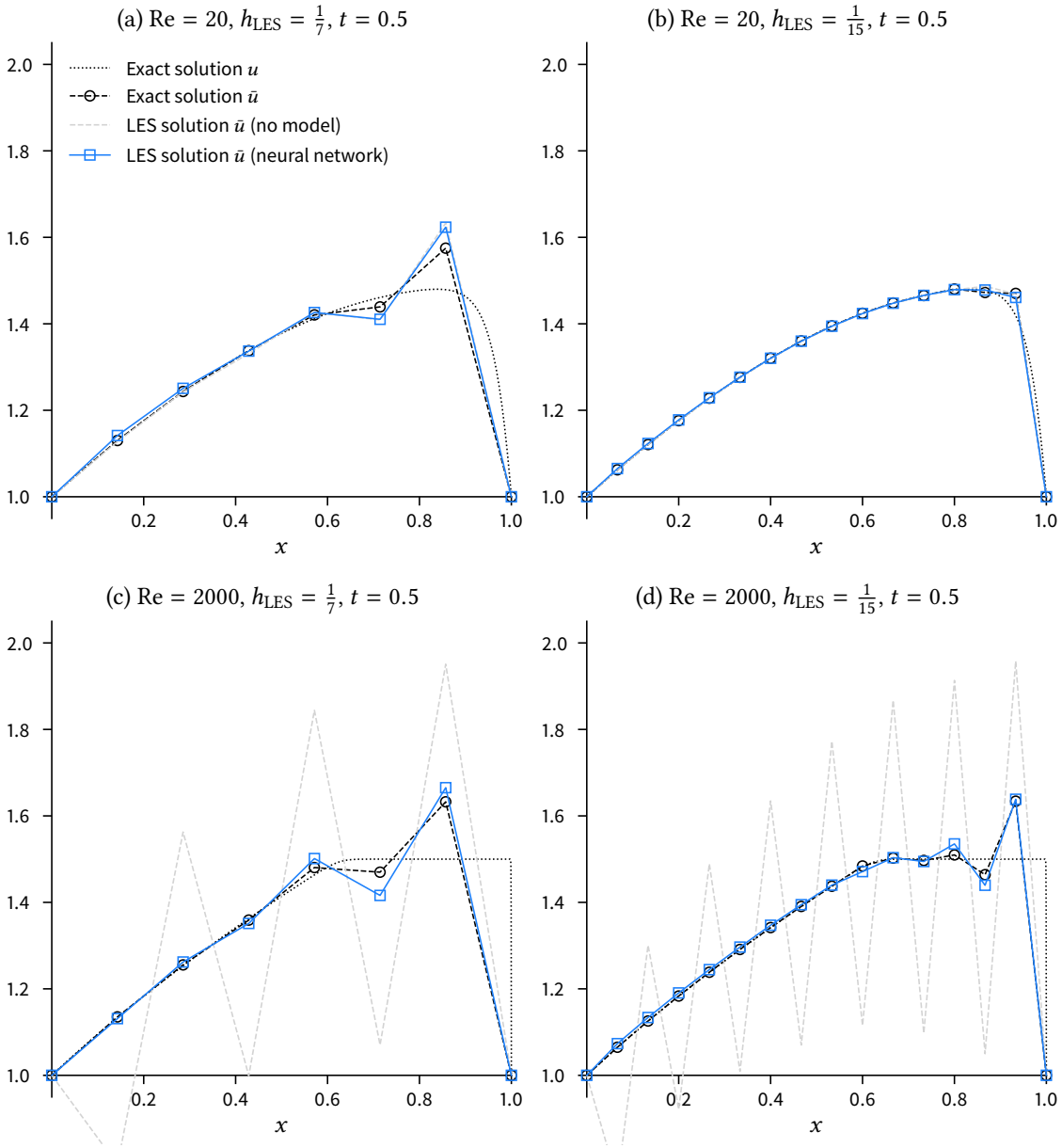

Figure 7.1: LES solutions to Problem (7.5) obtained by testing the neural network within the simulation.

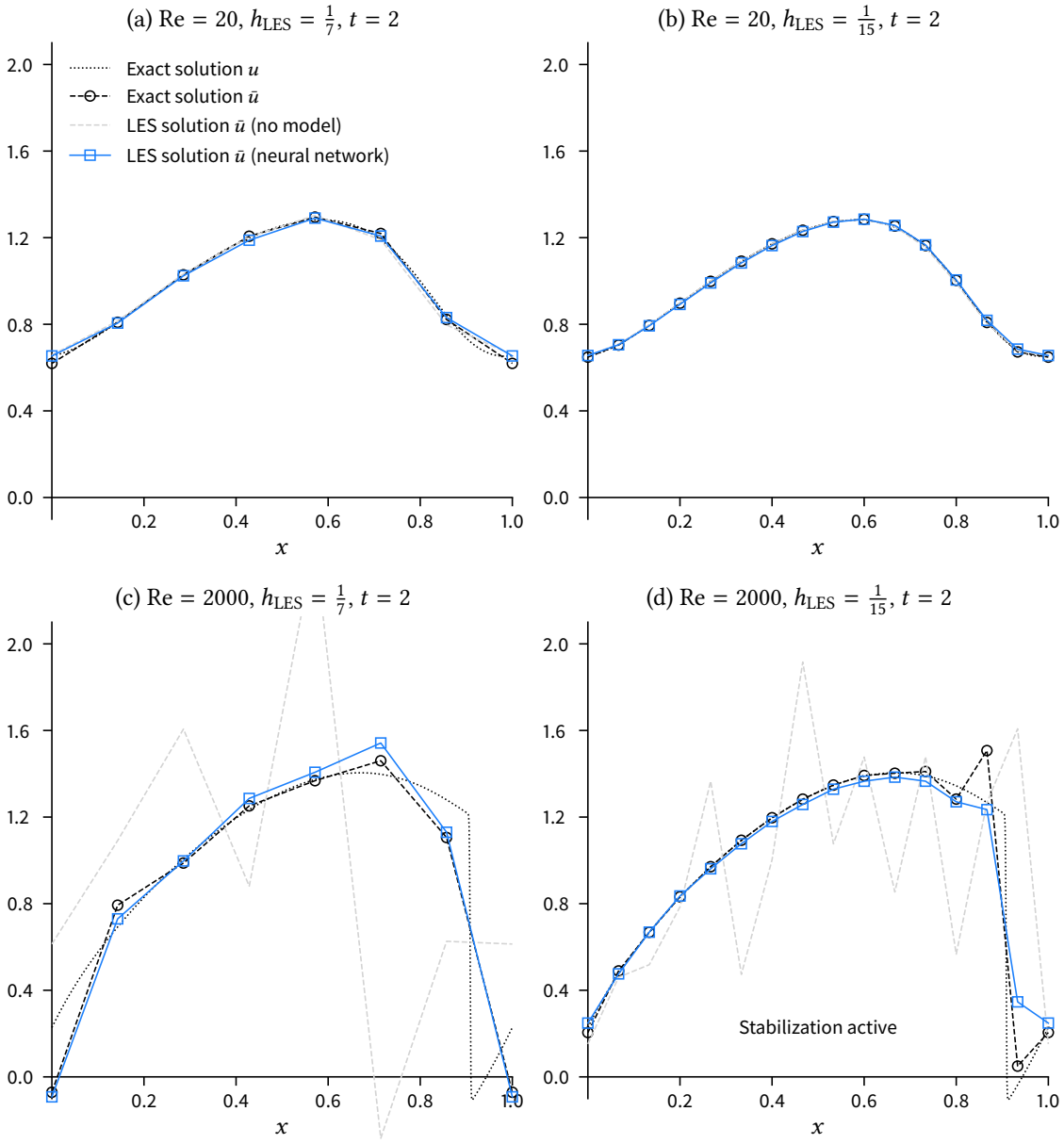

Figure 7.2: LES solutions to Problem (7.6) obtained by testing the neural network within the simulation.

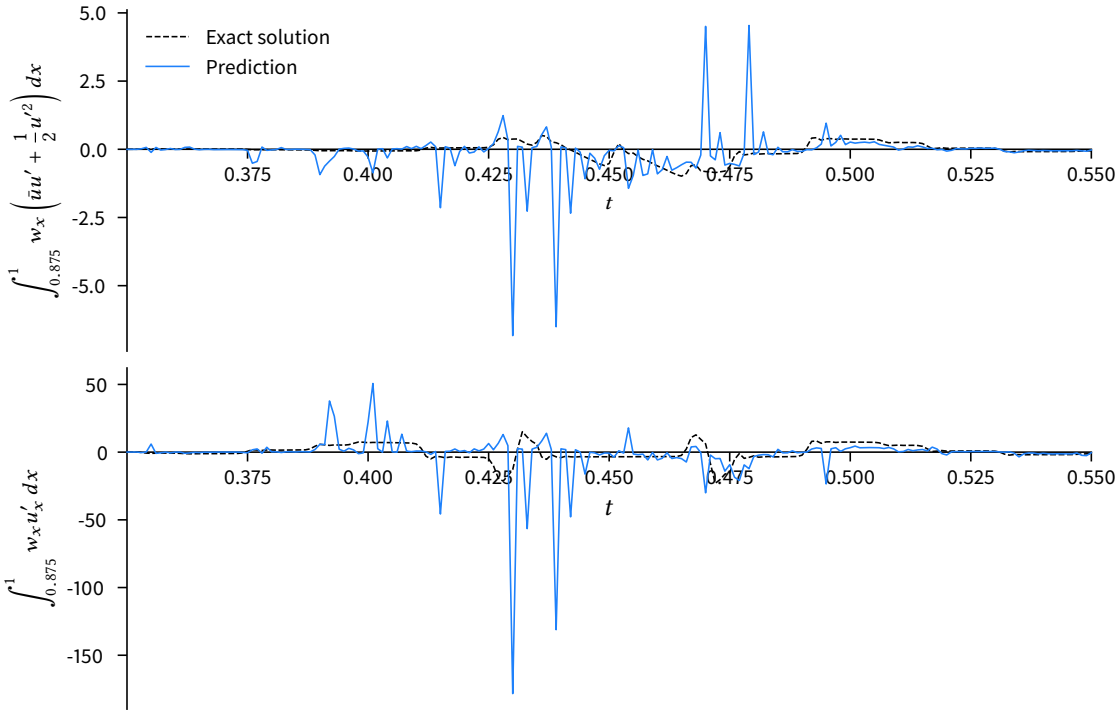

Figure 7.3: Time series of the predicted and exact u'- and u'_x-terms at individual data points on the right-most element of the domain in Problem (7.6), i.e., $x \in (0.875, 1)$.

7.3.2 Stabilization

The model is pushed to its limits in the presence of steep moving shocks. The inputs and outputs change dramatically when a shock enters the domain of an element, particularly when the mesh spacing is small. In Problem (7.6), this results in erroneous predictions that cause uncontrolled growth of the solution in the same vein as in Chapter 6. Figure 7.3 shows the predictions of the model based on individual data points. The predictions have extreme outliers that exceed the exact solution by up to two orders of magnitude. Application of the rudimentary stabilization scheme of Chapter 6 prevents the solution from growing uncontrollably and results in the solution given in Figure 7.2d.

7.4 Chapter Conclusion

The neural network is trained on examples where $\mathrm{Re} \in \{10, 100, 1000\}$ and $h_{\mathrm{LES}} \in \left\{\frac{1}{6}, \frac{1}{8}, \frac{1}{12}\right\}$. After training and validation, the model is used in LESs of two test problems where $\mathrm{Re} \in \{20, 2000\}$ and $h_{\mathrm{LES}} \in \left\{\frac{1}{7}, \frac{1}{15}\right\}$. The neural network correctly learned to generalize for high Reynolds numbers because $\mathrm{Re} = 2000$ is well outside the scope of the training set. Conversely, the tests at $\mathrm{Re} = 20$ show that the neural network learned that no unresolved-scale model is required when the Reynolds number is small. Furthermore, the model is able to handle both large and small elements, even when the elements are much smaller than in the training set, such as in the tests where $h = \frac{1}{15}$. Thus, it can be concluded that the neural network learned a fundamental underlying functional relationship and therefore has the ability to generalize when presented with previously unseen inputs.

8 Conclusion

Revisiting the research questions, the following conclusions are drawn:

Can a neural network establish a functional relationship between large-scale input features and the integral forms of the unresolved-scale terms?

The results of testing the neural networks on both individual data points as well as within actual simulations show without doubt that a neural network can learn a functional relationship between the large-scale input features and the unresolved-scale terms. The models achieve mean absolute errors (MAEs) on the validation sets of less than 0.05 without much tweaking and some optimization of the hyperparameters reduces the MAE to 0.015. To put these numbers into context, consider that the outputs are normalized to approximately the range $[-1, 1]$.

What are suitable large-scale input features?

The variables $\bar{u}$, $\bar{u}_t$, and f sampled at the element boundaries constitutes the so-called local stencil. Both $\bar{u}_x$ and $\mathcal{R}(\bar{u})$ can be computed from the variables in the local stencil and need not be provided explicitly. The local stencil yields a functional model because it is known from variational multiscale theory that u' is driven by $\bar{u}$ and the strong residual of $\bar{u}$. The extended stencil, where the input features are also sampled at both adjacent elements, yields an improvement over the local stencil if the correlation length exceeds the element length.

Can the model be simplified by omitting individual unresolved-scale terms?

The results show that the u'-term is the dominant term and must always be modeled. Modeling the u'_x-term in addition to the u'-term does improve the solution considerably, but not nearly as dramatically as the u'-term. The u'_t-term is a special case; it can be omitted without penalty if the Courant number is small (e.g., $C = 0.01$). The importance of the u'_t-term increases as the Courant number increases; at $C = 0.1$, the u'_t-term can no longer be omitted. Thus, the unresolved scales are quasi-steady at small timesteps whereas at large timesteps, this is no longer true.

What is a successful neural network architecture?

Due to the small number of inputs (12–14) and outputs (2–4), it makes little sense to use an architecture other than a densely connected neural network. It is therefore no surprise that a densely connected neural network with ReLU activation functions is highly successful in the current work. The results show that an architecture with three hidden layers of 64 units per layer is close to optimal, but this optimum will vary per problem. None of the models suffered from overfitting the training dataset; implementing regularization or dropout is therefore unnecessary.

How many examples are required to train a functional model?

The models in Chapter 5, 6, and 7 are trained on training sets consisting of respectively 32 000 examples, 3200 examples, and 270 000 examples. These numbers cannot be directly compared because the training set in Chapter 7 covered a wide range of Reynolds numbers and mesh spacings whereas the Reynolds numbers and mesh spacing were fixed in Chapter 5 and 6. The model trained on 3200 examples achieves a mean absolute error on the validation set of 0.0344, which is in the same order of magnitude as the errors of the remaining models. The dataset in Chapter 7 can therefore likely be reduced to 27 000 examples and still be sufficiently large to train a functional model that is nearly universal.

Can the neural network generalize to new situations?

The models are tested on two test problems with different boundary conditions, forcing functions, Reynolds numbers, and mesh spacings than the training problem. The neural networks perform remarkably well in these tests; the LES solutions are close to the exact solutions in all cases, even if the Reynolds number far exceeds the Reynolds number seen in training. It can therefore be concluded that the neural networks have truly learned an underlying physical relationship and can generalize to new situations.

Under what conditions does the model fail to make accurate predictions and what are the implications of these errors?

When steep moving shocks cross into an element, the unresolved scales change dramatically. In these cases, the models sometimes make erroneous predictions that exceed the exact solution by two orders of magnitude. The implication of these erroneous predictions is a surplus of energy that triggers the solution to grow uncontrollably. This shows that measures must be taken to combat the effects of errors.

What measures can be taken to improve the resiliency to erroneous predictions?

Tweaking the hyperparameters and improving the training dataset can improve the accuracy of the neural network, but neural networks are statistical tools and erroneous predictions can therefore never completely be avoided. Solutions to instabilities caused by erroneous predictions must therefore be implemented at the level of the simulation. A rudimentary stabilization scheme consists of filtering extreme outliers by imposing lower and upper limits to the outputs of the neural network. This rudimentary stabilization scheme is capable of preventing the solution to grow uncontrollably. Testing shows that setting the lower and upper limits to respectively the 1st and 99th percentiles of the training set yields results that closely match the exact solutions.

9 Discussion and Recommendations

The unconstrained approach in the current work stands in contrast to the constrained approach in the previous work at TU Delft. In the current work, the neural network represents the unresolved-scale model in its entirety whereas in the previous work, the neural network is integrated into an existing unresolved-scale model. The constrained approach retains the residual-based formulation which improves the robustness of the unresolved-scale model. The unconstrained approach results in a larger function space which enables the neural network to learn subtle nuances or otherwise hard-to-understand edge cases. A fair comparison between the unconstrained and the constrained approach is not possible at this point because the previous work does not include performance evaluations in previously unseen test problems. The results in the current work show that the unconstrained approach suffers from robustness issues and an a posteriori stabilization scheme is necessary to get these issues under control. Robustness could be improved by finding a middle ground between the unconstrained and the constrained approaches where the residual-based formulation is restored, i.e., a hybrid method.

The rudimentary stabilization scheme employed in the current work imposes lower and upper limit on the outputs of the neural network where the limits are based on a statistical analysis of the training examples. While this scheme can filter extreme outliers, it cannot prevent erroneous predictions that are within the bounds of the limits. A physics informed stabilization scheme could monitor the local transfer of energy to the subgrid scales to prevent erroneous predictions. It is known that the unresolved-scales are predominantly dissipative and large amounts of backscatter should therefore raise a red flag. Furthermore, implementing methods akin to successive over-relaxation in the corrector passes of Newton's method may also prevent the solution from growing uncontrollably while at the same time improving the rate of convergence. Investigating advanced stabilization scheme could be considered a topic of future work.

Now that both the unconstrained and the constrained approach have been applied to Burgers' equation, there are enough results to justify continued research on neural-network-based wall models for the Navier-Stokes equations. Developing a neural-network-based unresolved-scale model for the three-dimensional Navier-Stokes equations is in many ways more complicated than developing a model for the one-dimensional Burgers' equation. There are many individual unresolved-scale terms, there are more large-scale flow parameters that should be considered as input features, the extended stencil now includes adjacent elements in all dimensions, etc. As a first step, the process in Chapter 4 should be repeated for the Navier-Stokes equations. That is, running a DNS of a representative model problem, computing the unresolved-scale terms, and substituting the (modified) precomputed terms into an LES of the same model problem. It is worthwhile to investigate if any of the unresolved-scale terms can safely be omitted from a model. Thereafter, a first neural network can be trained on a dataset derived from a DNS. A one-dimensional Navier-Stokes problem can be used initially to smoothen the transition from Burgers' equation to the Navier-Stokes equations.

References

[1] J. Slotnick et al. *CFD vision 2030 study: a path to revolutionary computational aerosciences*. Technical Report NASA/CR-2014-218178. NASA Langley Research Center, 2014.

[2] T. J. R. Hughes. "Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods". *Computer Methods in Applied Mechanics and Engineering* 127.1-4 (1995), pp. 387–401. DOI: 10.1016/0045-7825(95)00844-9.

[3] J. M. Burgers. "A mathematical model illustrating the theory of turbulence". *Advances in Applied Mechanics* 1 (1948), pp. 171–199. DOI: 10.1016/S0065-2156(08)70100-5.

[4] P. G. Saffman. "Lectures on Homogeneous Turbulence". *Topics in Nonlinear Physics* (1968), pp. 485–614. DOI: 10.1007/978-3-642-88504-4_6.

[5] D. R. Chapman. "Computational aerodynamics development and outlook". *AIAA Journal* 17.12 (1979), pp. 1293–1313. DOI: 10.2514/3.61311.

[6] U. Piomelli and E. Balaras. "Wall-layer models for large-eddy simulations". *Annual Review of Fluid Mechanics* 34 (2002), pp. 349–374. DOI: 10.1146/annurev.fluid.34.082901.144919.

[7] I. Marusic, R. Mathis, and N. Hutchins. "Predictive model for wall-bounded turbulent flow". *Science* 329.5988 (2010), pp. 193–196. DOI: 10.1126/science.1188765.

[8] A. J. Smits and I. Marusic. "Wall-bounded turbulence". *Physics Today* 66.9 (2013), pp. 25–30. DOI: 10.1063/PT.3.2114.

[9] U. Piomelli. "Wall-layer models for large-eddy simulations". *Progress in Aerospace Sciences* 44.6 (2008), pp. 437–446. DOI: 10.1016/j.paerosci.2008.06.001.

[10] K. Duraisamy, Z. J. Zhang, and A. P. Singh. "New approaches in turbulence and transition modeling using data-driven techniques". *53rd AIAA Aerospace Sciences Meeting*. 2015. DOI: 10.2514/6.2015-1284.

[11] B. Tracey, K. Duraisamy, and J. J. Alonso. "A machine learning strategy to assist turbulence model development". *53rd AIAA Aerospace Sciences Meeting*. 2015. DOI: 10.2514/6.2015-1287.

[12] M. Milano and P. Koumoutsakos. "Neural network modeling for near wall turbulent flow". *Journal of Computational Physics* 182.1 (2002), pp. 1–26. DOI: 10.1006/jcph.2002.7146.

[13] F. Sarghini, G. Felice, and S. Santini. "Neural networks based subgrid scale modeling in large eddy simulations". *Computers and Fluids* 32.1 (2003), pp. 97–108. DOI: 10.1016/S0045-7930(01)00098-6.

[14] J. Bardina, J. H. Ferziger, and W. C. Reynolds. "Improved subgrid-scale models for large-eddy simulation". *AIAA Paper* (1980).

[15] A. Vollant, G. Balarac, and C. Corre. "Subgrid-scale scalar flux modelling based on optimal estimation theory and machine-learning procedures". *Journal of Turbulence* 18.9 (2017), pp. 854–878. DOI: 10.1080/14685248.2017.1334907.

[16] M. Gamahara and Y. Hattori. "Searching for turbulence models by artificial neural network". *Physical Review Fluids* 2.5 (2017). DOI: 10.1103/PhysRevFluids.2.054604.

[17] T. J. R. Hughes, A. A. Oberai, and L. Mazzei. "Large eddy simulation of turbulent channel flows by the variational multiscale method". *Physics of Fluids* 13.6 (2001), pp. 1784–1799. DOI: 10.1063/1.1367868.

[18] A. A. Oberai and T. J. R. Hughes. "A palette of fine-scale eddy viscosity and residual-based models for variational multiscale formulations of turbulence". *Computational Mechanics* 57.4 (2016), pp. 629–635. DOI: 10.1007/s00466-015-1242-2.

[19] F. Shakib, T. J. R. Hughes, and Z. Johan. "A new finite element formulation for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equations". *Computer Methods in Applied Mechanics and Engineering* 89.1-3 (1991), pp. 141–219. DOI: 10.1016/0045-7825(91)90041-4.

[20] T. Durieux. "Exploring the use of artificial neural network based subgrid scale models in a variational multiscale formulation". Master's thesis. Faculty of Aerospace Engineering, Delft University of Technology, 2015.

[21] M. Beekman. "Using artificial neural networks as unresolved-scale models for the Burgers equation". Master's thesis. Faculty of Aerospace Engineering, Delft University of Technology, 2016.

[22] N. Kurian. "Subgrid-scale model using artificial neural networks for wall-bounded turbulent flows". Master's thesis. Faculty of Aerospace Engineering, Delft University of Technology, 2018.

[23] L. C. Navarro Hernández. "Design of residual-based unresolved-scale models using time-averaged solution data". Master's thesis. Faculty of Aerospace Engineering, Delft University of Technology, 2015.

[24] M. G. Larson and F. Bengzon. *The finite element method: theory, implementation, and applications*. Springer, 2010. ISBN: 978-3-642-33287-6. DOI: 10.1007/978-3-642-33287-6.

[25] T. J. R. Hughes, L. Mazzei, and K. E. Jansen. "Large Eddy Simulation and the variational multiscale method". *Computing and Visualization in Science* 3.1-2 (2000), pp. 47–59. DOI: 10.1007/s007910050051.

[26] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[27] François Chollet et al. *Keras*. https://keras.io. 2015.

[28] François Chollet. *Deep learning with Python*. Springer, 2017. ISBN: 978-1-617-29443-3.

[29] Andrew Ng et al. *Machine Learning*. https://www.coursera.org/learn/machine-learning. 2019.

A Derivation of the Variational Multiscale Form of Burgers' Equation

The strong form of Burgers' equation is written as

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{1}{2}\dfrac{\partial}{\partial x}\left(u^2\right) - \dfrac{1}{\mathrm{Re}}\dfrac{\partial^2 u}{\partial x^2} = f(x,t), & x \in (0,1), t \in (0,T] & \text{(PDE)} \\ u(x,0) = I(x), & x \in (0,1) & \text{(initial condition)} \\ u(0,t) = 0, & t \in (0,T] & \text{(boundary condition)} \\ u(1,t) = 0, & t \in (0,T] & \text{(boundary condition)} \end{cases} \tag{A.1}$$

For the sake of simplicity, the boundary conditions in (A.1) are homogeneous, but other types of boundary conditions are equally valid.

A.1 Weak Form of Burgers' Equation

The method of weighted residuals involves multiplying the PDE by a so-called weighting function w and integrating over the domain. The resulting equation is required to hold for all w in the set of weighting functions $\mathcal{W}$. The application of the method of weighted residuals to (A.1) gives

$$\int_0^1 w\frac{\partial u}{\partial t}dx + \frac{1}{2}\int_0^1 w\frac{\partial}{\partial x}\left(u^2\right)dx - \frac{1}{\mathrm{Re}}\int_0^1 w\frac{\partial^2 u}{\partial x^2}dx = \int_0^1 wf\,dx, \qquad \forall w \in \mathcal{W} \tag{A.2}$$

The application of integration by parts to the second and third terms in (A.2) yields

$$\int_0^1 w\frac{\partial u}{\partial t}dx - \frac{1}{2}\int_0^1 \frac{dw}{dx}u^2dx + \frac{1}{2}\left[wu^2\right]_0^1 + \frac{1}{\mathrm{Re}}\int_0^1 \frac{dw}{dx}\frac{du}{dx}dx - \frac{1}{\mathrm{Re}}\left[w\frac{du}{dx}\right]_0^1 = \int_0^1 wf\,dx, \qquad \forall w \in \mathcal{W} \tag{A.3}$$

The boundary terms in (A.3) disappear because the weighting functions are assumed to vanish at the boundaries. To simplify subsequent writing, let

$$(w,u) = \int_0^1 wu\,dx \tag{A.4}$$

In the finite element method, u is approximated as a linear combination of a finite set of functions $\mathcal{V}$ and this thesis uses the Bubnov-Galerkin method where $\mathcal{V} = \mathcal{W}$. Using the notational convention in (A.4), the variational from (or weak form) of (A.1) is written as

$$\begin{cases} \text{Find } u \in \mathcal{W} \text{ such that } \forall w \in \mathcal{W}\text{:} \\ (w,u_t) - \dfrac{1}{2}\left(w_x,u^2\right) + \dfrac{1}{\mathrm{Re}}\left(w_x,u_x\right) = (w,f) \end{cases} \tag{A.5}$$

A.2 Variational Multiscale Form of Burgers' Equation

The goal of multiscale LES is computing $\bar{u}$, the large-scale component of u that lives in $\mathcal{W}$. In this thesis, $\mathcal{W}$ is a set of piecewise-linear functions. Substituting $u = \bar{u} + u'$ into (A.5) yields

$$(w, (\bar{u} + u')_t) - \frac{1}{2}\left(w_x, (\bar{u} + u')^2\right) + \frac{1}{\mathrm{Re}}\left(w_x, (\bar{u} + u')_x\right) = (w, f), \qquad \forall w \in \mathcal{W} \tag{A.6}$$

and can be rearranged as

$$(w, \bar{u}_t) - \frac{1}{2}\left(w_x, \bar{u}^2\right) + \frac{1}{\mathrm{Re}}\left(w_x, \bar{u}_x\right) + \left(w, u'_t\right) - \left(w_x, \bar{u}u' + \frac{1}{2}u'^2\right) + \frac{1}{\mathrm{Re}}\left(w_x, u'_x\right) = (w, f), \qquad \forall w \in \mathcal{W} \tag{A.7}$$

The terms in (A.7) that contain u', u'_x, and u'_t are referred to as the unresolved-scale terms. The variational multiscale form of (A.1) is therefore written as

$$\begin{cases} \text{Find } \bar{u} \in \mathcal{W} \text{ such that } \forall w \in \mathcal{W}\text{:} \\ (w, \bar{u}_t) - \frac{1}{2}\left(w_x, \bar{u}^2\right) + \frac{1}{\mathrm{Re}}\left(w_x, \bar{u}_x\right) + \text{unresolved-scale terms} = (w, f) \end{cases} \tag{A.8}$$

where

$$\text{unresolved-scale terms} = \underbrace{\left(w, u'_t\right)}_{u'_t\text{-term}} - \underbrace{\left(w_x, \bar{u}u' + \frac{1}{2}u'^2\right)}_{u'\text{-term}} + \underbrace{\frac{1}{\mathrm{Re}}\left(w_x, u'_x\right)}_{u'_x\text{-term}}, \qquad \forall w \in \mathcal{W} \tag{A.9}$$

For the sake of brevity, the individual integral forms of the unresolved-scale terms are referred to as the u'-, u'_t-, and u'_x-terms.

B Derivation of the Large-Scale Energy Balance

As shown in Appendix A, the variational multiscale form of Burgers' equation is written as

$$\begin{cases} \text{Find } \bar{u} \in \mathcal{W} \text{ such that } \forall w \in \mathcal{W}: \\ (w, \bar{u}_t) - \frac{1}{2}\left(w_x, \bar{u}^2\right) + \frac{1}{\mathrm{Re}}\left(w_x, \bar{u}_x\right) + \text{unresolved-scale terms} = (w, f) \end{cases} \tag{B.1}$$

where

$$\text{unresolved-scale terms} = (w, u'_t) - \left(w_x, \bar{u}u' + \frac{1}{2}u'^2\right) + \frac{1}{\mathrm{Re}}\left(w_x, u'_x\right), \qquad \forall w \in \mathcal{W} \tag{B.2}$$

Substituting $w = \bar{u}$ into (B.1) and (B.2) yields an evolution equation for the large-scale energy $\bar{E}$. This evolution equation would be incomplete because the boundary terms that follow from integration by parts cannot be omitted if $w = \bar{u}$. Instead, reverting integration by parts and then substituting $w = \bar{u}$ yields

$$\int_\Omega \bar{u}\frac{\partial \bar{u}}{\partial t}dx + \frac{1}{2}\int_\Omega \bar{u}\frac{\partial}{\partial x}\left(\bar{u}^2\right)dx - \frac{1}{\mathrm{Re}}\int_\Omega \bar{u}\frac{\partial^2 \bar{u}}{\partial x^2}dx + \text{unresolved-energy terms} = \int_\Omega \bar{u}f\,dx \tag{B.3}$$

where

$$\text{unresolved-energy terms} = \int_\Omega \bar{u}\frac{\partial u'}{\partial t}dx + \int_\Omega \bar{u}\frac{\partial}{\partial x}\left(\bar{u}u' + \frac{1}{2}u'u'\right)dx - \frac{1}{\mathrm{Re}}\int_\Omega \bar{u}\frac{\partial^2 u'}{\partial x^2}dx \tag{B.4}$$

Equation (B.3) can be written[1] as

$$\frac{\partial}{\partial t}\int_\Omega \frac{1}{2}\bar{u}^2 dx + \frac{1}{2}\int_\Omega \bar{u}\frac{\partial}{\partial x}\left(\bar{u}^2\right)dx - \frac{1}{\mathrm{Re}}\int_\Omega \bar{u}\frac{\partial^2 \bar{u}}{\partial x^2}dx + \text{unresolved-energy terms} = \int_\Omega \bar{u}f\,dx \tag{B.5}$$

or

$$\frac{\partial \bar{E}}{\partial t} = \underbrace{-\frac{1}{2}\int_\Omega \bar{u}\frac{\partial}{\partial x}\left(\bar{u}^2\right)dx}_{\text{energy flux}} + \underbrace{\frac{1}{\mathrm{Re}}\int_\Omega \bar{u}\frac{\partial^2 \bar{u}}{\partial x^2}dx}_{\text{viscous dissipation}} \underbrace{- \text{unresolved-energy terms}}_{\text{unresolved-scale model}} + \underbrace{\int_\Omega \bar{u}f\,dx}_{\text{work done}} \tag{B.6}$$

Equation (B.6) is the evolution equation of the large-scale energy $\bar{E}$.

[1] Application of the chain rule on the time derivative gives

$$\frac{\partial}{\partial t}\left(\frac{1}{2}\bar{u}^2\right) = \frac{1}{2}\frac{\partial}{\partial t}\left(\bar{u}^2\right) = \frac{1}{2}\cdot 2\bar{u}\cdot\frac{\partial \bar{u}}{\partial t} = \bar{u}\frac{\partial \bar{u}}{\partial t}$$

B.1 Evaluating the Energy Balance

Application of integration by parts to the energy flux and viscous terms in (B.6) yields

$$\frac{\partial \bar{E}}{\partial t} = \frac{1}{2}\int_\Omega \bar{u}_x \bar{u}^2 dx - \frac{1}{2}\left[\bar{u}^3\right]_{\partial\Omega} - \frac{1}{\mathrm{Re}}\int_\Omega \bar{u}_x^2 dx + \frac{1}{\mathrm{Re}}\left[\bar{u}\bar{u}_x\right]_{\partial\Omega} - \mathrm{UETs} + \int_\Omega \bar{u} f\, dx \tag{B.7}$$

where UETs is shorthand for unresolved-energy terms. The element-wise evaluation of (B.7) is complicated because the $[\bar{u}\bar{u}_x]_{\partial\Omega}$-term requires knowledge of $\bar{u}_x$ at the boundaries of the elements, but $\bar{u}_x$ is discontinuous because piecewise-linear elements are used. Instead, the energy balance can be evaluated globally by using homogeneous or periodic boundary conditions. In that case, the boundary terms disappear and the energy balance is given by

$$\frac{\partial \bar{E}}{\partial t} = \frac{1}{2}\int_\Omega \bar{u}_x \bar{u}^2 dx - \frac{1}{\mathrm{Re}}\int_\Omega \bar{u}_x^2 dx - \mathrm{UETs} + \int_\Omega \bar{u} f\, dx \tag{B.8}$$

All terms in (B.8) can be evaluated by numerical integration.

B.2 Evaluating the Unresolved-Energy Terms

Equation (B.8) contains the unresolved-energy terms given by (B.4). Since u' is unknown, the unresolved-energy terms must be expressed in terms of the integral forms of the unresolved-scale terms predicted by the neural network. Application of integration by parts to (B.4) yields

$$\text{unresolved-energy terms} = \int_\Omega \bar{u}u_t'\, dx - \int_\Omega \bar{u}_x\left(\bar{u}u' + \frac{1}{2}u'u'\right) dx + \frac{1}{\mathrm{Re}}\int_\Omega \bar{u}_x u_x'\, dx \tag{B.9}$$

where the boundary terms disappear because the energy balance is evaluated globally using homogeneous or periodic boundary conditions. Recall from Section 2.1.1 that $\bar{u}$ is a linear combination of the weighting functions, i.e.,

$$\bar{u}(x) = \sum_{i=1}^{N} c_i w_i(x), \qquad \bar{u}_x(x) = \sum_{i=1}^{N} c_i w_{x,i}(x) \tag{B.10}$$

Substituting (B.10) into (B.9) yields

$$\text{unresolved-energy terms} = \sum_{i=0}^{N} c_i \left[\int_\Omega w_i u_t'\, dx - \int_\Omega w_{x,i}\left(\bar{u}u' + \frac{1}{2}u'u'\right) dx + \frac{1}{\mathrm{Re}}\int_\Omega w_{x,i} u_x'\, dx\right] \tag{B.11}$$

Comparing (B.11) to (B.2), it is easy to see that

$$\text{unresolved-energy terms} = \sum_{i=1}^{N} c_i \cdot \text{unresolved-scale terms} \tag{B.12}$$

Recall that $c_i = \bar{u}_i$ if piecewise-linear basis functions are used. Thus, (B.12) can be written as

$$\text{unresolved-energy terms} = \sum_{i=1}^{N} \bar{u}_i \cdot \text{unresolved-scale terms} \tag{B.13}$$

The contribution of the neural-network-based predictions of the unresolved-scale terms to the large-scale energy balance can thus be quantified with little effort.

C Implementation Details

The variational multiscale form of Burgers' equation is written as

$$\begin{cases} \text{Find } \bar{u} \in \mathcal{W} \text{ such that } \forall w \in \mathcal{W}: \\ (w, \bar{u}_t) - \frac{1}{2}\left(w_x, \bar{u}^2\right) + \frac{1}{\mathrm{Re}}\left(w_x, \bar{u}_x\right) + \mathrm{USTs} = (w, f) \end{cases} \tag{C.1}$$

where USTs is shorthand for unresolved-scale terms. For the sake of illustration, the backward Euler time-marching scheme is used here. The actual implementation uses a higher-order trapezoidal scheme. The backward Euler time-marching scheme is given by

$$\frac{\partial \bar{u}}{\partial t} = \frac{\bar{u} - \bar{u}_{\text{previous}}}{\Delta t} \tag{C.2}$$

where $\bar{u}_{\text{previous}}$ denotes the solution at the previous timestep. Substituting (C.2) into (C.1) and reverting to an integral notation yields

$$\int_\Omega w \frac{\bar{u} - \bar{u}_{\text{previous}}}{\Delta t} dx - \frac{1}{2}\int_\Omega w_x \bar{u}^2 dx + \frac{1}{\mathrm{Re}}\int_\Omega w_x \bar{u}_x dx + \mathrm{USTs} = \int_\Omega w f \, dx, \qquad \forall w \in \mathcal{W} \tag{C.3}$$

Let $\{w_i(x)\}_{i=1}^N$ denote the functions in $\mathcal{W}$ and define the index set $\mathcal{I}_s = \{1, \ldots, N\}$. Equation (C.3) can then be written as

$$\int_\Omega w_i \frac{\bar{u} - \bar{u}_{\text{previous}}}{\Delta t} dx - \frac{1}{2}\int_\Omega w_{i,x} \bar{u}^2 dx + \frac{1}{\mathrm{Re}}\int_\Omega w_{i,x} \bar{u}_x dx + \mathrm{USTs} = \int_\Omega w_i f \, dx, \qquad i \in \mathcal{I}_s \tag{C.4}$$

which is a nonlinear system of N equations with N unknowns.

C.1 Newton's Method

Systems of nonlinear equations can be solved using Newton's method, which requires a vector of residuals and the Jacobian matrix. The N residuals $r_i(\boldsymbol{c}) = 0$ are given by

$$r_i(\boldsymbol{c}) = \int_\Omega \left(w_i \frac{\bar{u} - \bar{u}_{\text{previous}}}{\Delta t} - \frac{1}{2} w_{i,x} \bar{u}^2 + \frac{1}{\mathrm{Re}} w_{i,x} \bar{u}_x - w_i f \right) dx + \mathrm{USTs}, \qquad i \in \mathcal{I}_s \tag{C.5}$$

where $\boldsymbol{c}$ is the vector of coefficients. The derivatives of the unresolved-scale terms are not included in the Jacobian. This is not problematic because Newton's method will also converge without a perfectly correct Jacobian. The following two derivatives are used in the derivation of the Jacobian:

$$\bar{u}(x) = \sum_{i \in \mathcal{I}_s} c_i w_i(x), \qquad \frac{\partial \bar{u}}{\partial c_j} = \frac{\partial}{\partial c_j} \sum_{k=1}^{N} c_k w_k = w_j, \qquad \frac{\partial \bar{u}_x}{\partial c_j} = \frac{\partial}{\partial c_j} \sum_{k=1}^{N} c_k w_{k,x} = w_{j,x} \tag{C.6}$$

Using the two derivatives in (C.6), the Jacobian of $\boldsymbol{r}(\boldsymbol{c})$ is derived as follows:

$$\begin{aligned} J_{i,j} = \frac{\partial r_i}{\partial c_j} &= \int_\Omega \frac{\partial}{\partial c_j}\left(w_i \frac{\bar{u}-\bar{u}_{\text{previous}}}{\Delta t} - \frac{1}{2} w_{i,x}\bar{u}^2 + \frac{1}{\text{Re}} w_{i,x}\bar{u}_x - w_i f\right) dx, \qquad i,j \in \mathcal{I}_s \\ &= \int_\Omega \left(\frac{1}{\Delta t} w_i \frac{\partial \bar{u}}{\partial c_j} - w_{i,x}\bar{u}\frac{\partial \bar{u}}{\partial c_j} + \frac{1}{\text{Re}} w_{i,x}\frac{\partial \bar{u}_x}{\partial c_j}\right) dx, \qquad i,j \in \mathcal{I}_s \\ &= \int_\Omega \left(\frac{1}{\Delta t} w_i w_j - w_{i,x}\bar{u} w_j + \frac{1}{\text{Re}} w_{i,x} w_{j,x}\right) dx, \qquad i,j \in \mathcal{I}_s \end{aligned} \tag{C.7}$$

The slightly incorrect Jacobian may have an impact on the number of corrector passes required for convergence. The Newton-Galerkin method for Burgers' equation that results from (C.5) and (C.7) is given in Algorithm 1. The starting guess is either the solution of the previous timestep or the initial condition if there is no previous solution.

Algorithm 1 Newton-Galerkin method for Burgers' equation

1: Choose a starting guess $\boldsymbol{c}^1$
2: **for** $k = 1, 2, 3, \ldots$ **do** ▷ Corrector passes
3: Assemble the Jacobian matrix $\boldsymbol{J}^k$ and the residual vector $\boldsymbol{r}^k$ with entries

$$r_i^k = \int_\Omega \left(w_i \frac{\bar{u}^k - \bar{u}_{\text{previous}}}{\Delta t} - \frac{1}{2} w_{i,x}\left(\bar{u}^k\right)^2 + \frac{1}{\text{Re}} w_{i,x}\bar{u}_x^k - w_i f\right) dx + \text{USTs}, \qquad i \in \mathcal{I}_s$$

$$J_{i,j}^k = \int_\Omega \left(\frac{1}{\Delta t} w_i w_j - w_{i,x}\bar{u}^k w_j + \frac{1}{\text{Re}} w_{i,x} w_{j,x}\right) dx, \qquad i,j \in \mathcal{I}_s$$

4: Modify $\boldsymbol{J}^k$ and $\boldsymbol{r}^k$ to account for the boundary conditions
5: Solve the linear system

$$\boldsymbol{J}^k \boldsymbol{\delta}^k = -\boldsymbol{r}^k$$

6: $\boldsymbol{c}^{k+1} \leftarrow \boldsymbol{c}^k + \boldsymbol{\delta}^k$
7: **end for**

C.2 Assembly of the Residual Vector

The assembly of the vector of residuals $\boldsymbol{r}$ within the corrector passes is detailed in Algorithm 2. The neural network is invoked once for all local node in all elements, but not for all integration points because the neural network already predicts integral values. There are only two local nodes per element because the basis functions are piecewise linear.

Algorithm 2 Assembly of the vector of residuals $\boldsymbol{r}$

1: **for** $e = 1, 2, 3, \ldots$ **do** ▷ Loop over all elements
2: **for** $j \in \{1, 2\}$ **do** ▷ Loop over the two local nodes
3: **for** ip $= 1, 2, 3, \ldots$ **do** ▷ Loop over all integration points
4: Compute the partial integral and add it to r_i where $i = e + j - 1$
5: **end for**
6: Invoke neural network and add prediction to r_i where $i = e + j - 1$
7: **end for**
8: **end for**