Reduced Order Models for Fluid Flow With Generative Adversarial Networks (GANs)

by

Mirko Kemna

Interim Thesis and Literature Report

Student Number: 5606896

Faculty Electrical Engineering, Mathematics,
and Computer Science

Delft University of Technology

Contents

Chapter 1

Introduction

Chapter 2

Modeling Incompressible Fluids

Gases and liquids together make up a vast share of the matter on earth's surface, and as such fluids have long been an important topic of study in physics and mathematics. Today, fluid modeling is a mature field, with many highly developed numerical methods at the disposal of scientists and engineers. Some of the most important applications include modeling flow around structures (aviation, transportation, architecture), through turbomachinery (wind power, hydro power, jet engines) as well as through biological systems (e.g. the heart).

We will restrict our discussion of the topic to incompressible flow, which is sufficient for modeling all fluids, as well as gas flow so long as flow velocity is small compared to sonic speed.

2.1 Governing Equations

Fluids are generally modeled in the framework of continuum mechanics and described by field quantities (velocity, density, temperature etc.) as a function of space and time. To derive the equations that govern fluids, we simply apply well known conservation principles such as conservation of mass, momentum and energy, and demand that they hold for every part of the fluid. For instance, by assuming that no mass is created or destroyed, we get

$$\frac{\partial \rho}{\partial t} + \rho \nabla \cdot \mathbf{u} = 0, \tag{2.1}$$

where we have assumed a fluid of uniform mass density ρ, and $\mathbf{u}$ the vector describing its velocity. Next, we apply newtons second law, which yields

$$\rho \frac{D\mathbf{u}}{Dt} = \nabla \cdot \boldsymbol{\sigma} + \mathbf{f}, \tag{2.2}$$

Here, the left side corresponds to the inertial force of a fluid element, and the right side to the forces acting on it. The expression $\frac{D}{Dt}$ is the so called material derivative, i.e. the derivative in a Lagrangian reference frame. In a fixed Eulerian frame, this evaluates to

$$\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla. \tag{2.3}$$

In Eq. (2.2) we have split the forces into those arising due to stress in the fluid which is described by the stress tensor $\boldsymbol{\sigma}$ and external volumetric forces $\mathbf{f}$, such as gravity. We can go even further and separate out an isotropic term from the stress tensor, which we call the hydrostatic pressure p (we define it to act inwards on a fluid element, giving it a sign opposite to $\boldsymbol{\sigma}$). Thus, we get

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} \right) = -\nabla p + \nabla \cdot \boldsymbol{\tau} + \rho\, \mathbf{g} \tag{2.4}$$

where the tensor $\boldsymbol{\tau}$ contains the remaining stresses in the material and we have replaced the generalized volumetric force with the gravitational force. To close the system of equations, we need a way to relate τ to derivatives of the velocity. This relation we cannot easily derive from first principles of classical mechanics, as it depends on the molecular properties of the specific fluid. However, we may attempt to try the simplest possible relation: a proportional dependence on the velocity gradients. As it turns out, this supposition holds well for many of the most frequently studied liquids, in particular water and air. If we further assume that the fluid has uniform, isotropic properties (and thus in particular a constant density), we arrive at the following relation:

$$\boldsymbol{\tau} = \mu \left(\nabla \mathbf{u} + \nabla \mathbf{u}^{\mathrm{T}} \right) \tag{2.5}$$

where μ is a material constant known as the viscosity, or more specifically dynamic viscosity. Substituting this, we obtain the momentum equation we have been striving for:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = \frac{1}{\rho} \left(-\nabla p + \mu\, \nabla^2 \mathbf{u} \right) + \mathbf{g}. \tag{2.6}$$

Together with eq. (2.1), this constitutes a closed set of equations known as the Navier-Stokes equations for incompressible, uniform fluids.

In this research project, we will assume stationary flow, i.e. no changes of field quantities over time. Moreover, gravity has np effect since we are dealing with a single homogeneous fluid. The Navier-Stokes equations then become

$$\nabla \cdot \mathbf{u} = 0 \tag{2.7a}$$

$$\nu \nabla^2 \mathbf{u} - \rho(\mathbf{u} \cdot \nabla)\mathbf{u} = \nabla p. \tag{2.7b}$$

This is a second order, nonlinear system of partial differential equations (PDEs) with elliptic character. The first equation represents the condition of divergence free flow, and the second represents the force equilibrium. The first term in (**??**) represents the inertial forces acting on the fluid, the second terms represents the viscous forces, and the right hand side represents an internal source in form of the pressure gradient. Mathematically, the pressure gradients acts as an Lagrange multiplier enforcing the incompressibility constraint [1].

By reformulating the equation in terms of a natural length scale L and velocity scale U, we can non-dimensionalize the left side of (**??**):

$$\tilde{\nabla}^2 \tilde{\mathbf{u}} - \mathbf{Re}\,(\tilde{\mathbf{u}} \cdot \tilde{\nabla})\tilde{\mathbf{u}} = \frac{L}{\eta U}\tilde{\nabla} p. \tag{2.8}$$

where $(\tilde{\cdot})$ represents non-dimensional variables and

$$\mathbf{Re} = \frac{\rho U L}{\eta} \tag{2.9}$$

is the Reynolds number that describes the characteristic ratio of inertial to viscous forces. Formulating a natural pressure scale is less obvious, but typically is taken as ρU^2 [2, page 53-54]. The full dimensionless system of equation is then given by

$$\tilde{\mathbf{u}} = 0 \tag{2.10a}$$

$$\mathbf{Re}^{-1}\tilde{\nabla}^2 \tilde{\mathbf{u}} - (\tilde{\mathbf{u}} \cdot \tilde{\nabla})\tilde{\mathbf{u}} = \tilde{\nabla} p. \tag{2.10b}$$

Chapter 3

Machine Learning

Enabled by the culmination of sustained exponential growth in computing power over the last decades[CITE], the field of machine learning (ML) has made remarkable progress in the last few years. The term refers to techniques of enabling computers to solve problems, not by directly following a set of rules encoded by a programmer, but by inferring those rules from observation, i.e. data. The vast majority of contemporary artificial intelligence (AI) methods are based on machine learning. Moreover, they often rely on the concept of hierarchical representation, using consecutive layers of information processing units that build on each other to bootstrap more powerful representations of real world data. This paradigm is the origin of the term deep learning.[3]

Ground-breaking results have been achieved across fields such as image & speech recognition, natural language processing and autonomous driving, are in many cases are already powering consumer technology today. [CITE]

Beyond being a very active field of study itself, ML has also been successfully applied to advance frontiers in other areas of scientific research, an discipline generally referred to as scientific machine learning (SciML). A recent milestone highlighting the potential was a breakthrough in the area of protein folding by the AlphaFold 2 AI in 2020.[4]

3.1 Artificial Neural Networks

Artificial neuron networks (ANNs, hereafter also simply referred to as neural networks) are a type of machine learning architecture designed in loose

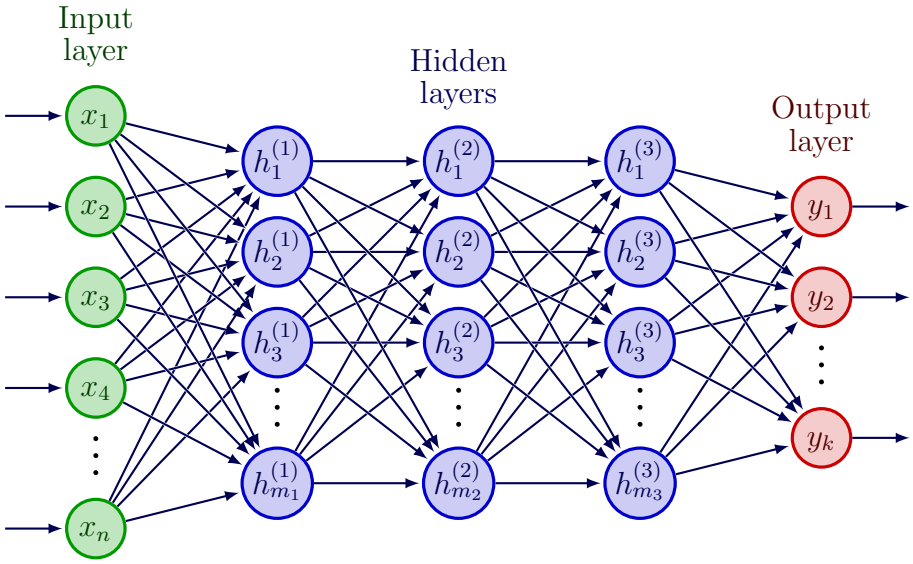

Figure 3.1: A graph visualizing the general structure of an artificial neural network with three hidden layers.

analogy to the networks formed by biological neurons found in the brains of humans and animals. As the name suggests, ANNs are made up of individual artificial neurons, which are arranged in connected layers. An example is visualized in figure 3.1. The most general description of the setting in which neural networks are applied is to model the relation underlying a set of observations (x, y) i.e. to find a way of relating a feature x to a label y. We generally distinguish between regression tasks, where the label is continuous (e.g. predicting the market value of a house), and classification tasks, where it is discrete (e.g. detecting a handwritten digit).

3.1.1 Artificial Neurons

A single neuron can be represented as a mapping from a number of inputs x_i to a single output y. Specifically, this takes the form of a weighted sum of all inputs, which is passed through a so-called activation function $f : \mathbb{R} \to \mathbb{R}$ after a fixed offset known as the bias has been added. We collect the inputs in a vector $\mathbf{x}$ and the corresponding weights in a vector $\mathbf{w}$. Together with

the bias b we thus write

$$h = f\left(\mathbf{w} \cdot \mathbf{x} + b\right), \tag{3.1}$$

where the resulting output h is called the activation level. The weights and bias parametrize a hyperplane in the input space known as the decision boundary, which is defined by $\mathbf{w} \cdot \mathbf{x} + b = 0$. A geometric interpretation is that the activation function operates on the euclidean distance d of the input point $\mathbf{x}$ to the decision boundary, scaled by the norm of the weight vector:

$$h = f\left(d\,|\mathbf{w}|\right). \tag{3.2}$$

3.1.2 Network Architecture

As mentioned before, the neurons in an ANN are arranged in layers. These are typically connected in a sequential order, such that information propagates from input to output, although other variations exist (recurrent neural networks, RNNs). Each layer defines an operation acting on the output of the previous. Data is always fed into the network at the input layer, in analogy to sensory organs in biological cognition. Then it passes through a number of so-called hidden layers, until it reaches the output layer where the response of the network is read off. While the size of input and output layer (n and k respectively) is imposed by the problem setting, the number and size of hidden layers is a design parameter. If we represent all layers (including the input) as one dimensional structures, then the operation that a layer j performs can be written as

$$\mathbf{z}_j = F\left(\mathbf{W}_j \mathbf{z}_i + \mathbf{b}\right), \tag{3.3}$$

where i is the index of the previous layer, $\mathbf{W}_j$ is an matrix of dimensions $m_j \times m_i$ (i.e. the sizes of layers i and j respectively), $\mathbf{b}$ is a vector containing the biases and F is the element wise application of an activation function f. If $\mathbf{W}_j$ is a dense matrix, we say layer j is fully connected. The connectivity structure is an important design parameter, and will be explored further in Section 3.2.

The whole network then is essentially a parametrized function $\mathcal{N}$ mapping from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$:

$$\begin{aligned} \mathcal{N} &: \mathcal{X} \to \mathcal{Y} \\ \hat{y} &= \mathcal{N}(x; \omega), \end{aligned} \tag{3.4}$$

where ω represents weights and bias of every neuron in the network, and $\hat{y}$ is the network's output. Typically, both input and output are taken as vectors in $\mathbb{R}$. However, if the data has a grid-like 2D or 3D structure it can make sense to reflect that in the mathematical representation of the network. We will use this in later sections, where we will be dealing with translating from an image of geometry to a flow field image of the same size. We correspondingly take input and output layers as 2-dimensional, i.e.

$$\mathcal{X} \subseteq \mathbb{R}^{k \times l}, \quad \mathcal{Y} \subseteq \mathbb{R}^{k \times l}$$

where k and l are the image dimensions. Note that size of input and output layers may also be drastically different; take for instance the task of classifying 256x256 pixel images on the basis of whether or not they depict a dog. In this case, the network will have an input layer with a size of approx. 65 000, but only a single, binary output neuron.

3.1.3 Training

A large neural network can have millions of parameters in the form of weights and biases. Our goal is to choose these in such a way that the network can approximate the unknown law underlying our observations. This law could be a simple functional relation, $y(x)$, but typically it is modeled in the more general framework of stochastics, i.e. as conditional distribution $p(y|x)$.

We want to approach the problem empirically, i.e. have the ANN learn from training examples. Our training set $\mathbb{X}$ consists of features x and associated labels (or solutions) y, i.e. we can denote the i^{th} training example as

$$\{x^{(i)}, y^{(i)}\} \in \mathbb{X}.$$

For each feature sample x, the ANN can generate an output $\hat{y}$. In order to improve, we need a measure of how good (or bad) the network performed. This is provided by the cost function $c(y, \hat{y})$, which assigns a cost to every combination of y and $\hat{y}$:

$$\begin{aligned} c : \mathcal{Y} \times \mathcal{Y} &\to \mathbb{R} \\ c(y, y) &= 0. \end{aligned} \tag{3.5}$$

With the grading system laid out, the last and most important step is to optimize the trainable parameters of the network such that we can reduce

the average cost across the training set. Solving the problem using direct methods is infeasible due to its size and complexity. Therefore, gradient based methods are typically used. These work by computing the derivative of the cost function with respect to every trainable parameter, and use that information to update them. The update is then given by

$$\Delta\omega = \eta\frac{\partial c}{\partial \omega} = \eta\frac{\partial c}{\partial \hat{y}}\frac{\partial \mathcal{N}}{\partial \omega}, \tag{3.6}$$

where η is a parameter scaling the update step, also known as learning rate. We may perform the update based one sample at a time (Stochastic Gradient Descent, SGD) or first average over a batch of samples (Batch Gradient Descent) and then update the weights.

3.2 Convolutional Networks

Much of the data encountered in real world problems has grid-like topology. In a sense this means that the position of a data point in relation to others matters, not just its value. Examples include images or time series such as audio recordings. A common way to reflect this in the structure of an ANN is through use of convolutional layers.

3.2.1 Convolutional Layers

Traditional image or signal processing often employs discrete convolutions for feature detection. Here, the "feature" is encoded in a small filter kernel or stencil, which is shifted across the imageor signal (in the following we will focus on application to images). At each point, the data in the kernel range is weighted by the corresponding kernel elements and summed up. The resulting grid is called a feature map, as it indicates the presence of the feature in the original image. A simple example is edge detection.

The same idea is used in convolutional layers, except now the features to search for are determined as part of the learning process. In the regular ANN framework, this corresponds to a sparsely connected layer (each neuron is connected only to its neighborhood), where additionally weights are shared between all neurons in the layer. The number of parameters therefore only depends on kernel size, not on the number of nodes in the layer. This drastically reduces the parameter space, as shown in Figure 3.2. However,

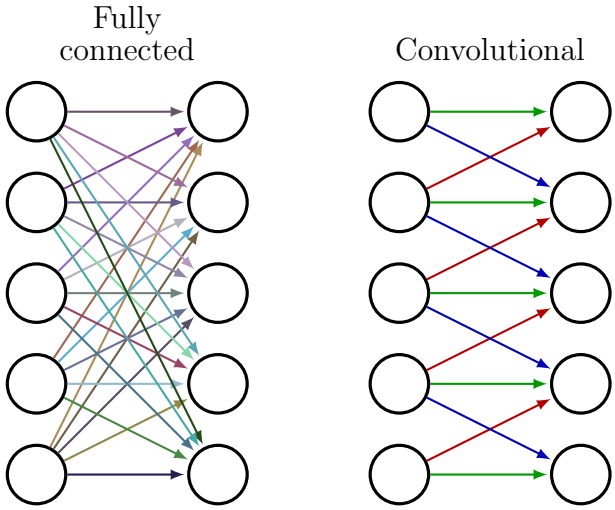

Figure 3.2: Comparison between fully connected and convolutional layer. Same colors indicate shared weights; the fully connected layer has 25 independent weights, while the convolutional layer has only 3.

we typically apply not just one but many kernels on a single layer, and thus we get multiple feature maps, also known as channels, as output. For the case of a 2D image, we can therefore represent a convolutional layer as a 3D block, where each sublayer in the block represents the response to a different filter. If another convolutional layer follows, it will act on the whole block, i.e. each kernel will have not just a width and height, but also a depth equal to that of the previous block. If we represent input and output of vectors, we can write the operation as

$$\mathbf{y}_i = \sum_j \mathbf{C}_{ij}\mathbf{x}_j, \tag{3.7}$$

where $\{\mathbf{x}_i\}_{i=1\ldots I}$ are the I input channels, $\{\mathbf{y}_i\}_{i=1\ldots J}$ are the J output channels and $\{\mathbf{C}_{ij}\}$ are the weight matrices defining the convolution. Each $\mathbf{C}_{ij}$ has the same sparsely diagonal structure, with each row containing the elements of the respective kernel. Not by coincidence, this structure is very similar to that of matrices resulting from finite element discretizations of PDE problems, for instance (here the kernel corresponds to the stencil). If $\mathbf{C}_{ij}$ is a square matrix, then the input and output feature maps are of the

same size, corresponding to the kernel being evaluated at every point of the input grid. Often however, the kernel is only evaluated at larger regularly spaced intervals, a method known as striding. This corresponds to removing row from $\mathbf{C}_{ij}$, and effectively downsamples the input. For example, a stride of two in x and y direction reduces the rows in $\mathbf{C}_{ij}$ and thereby the length of $\mathbf{y}_i$ by a factor of four. Moreover, there are different strategies for dealing with the points on the boundary, where the kernel extends beyond the input data (e.g. padding the data with zeros). For more details, see [5]. Another operation that is often combined with convolutions is so-called pooling. In a pooling operation, each pixel in the feature map is replaced with the average (average pooling) or maximum (max pooling) of the values in its neighborhood. For more details on convolutional layers and their use, see and [3, Chapter 9].

3.2.2 Structure of Convolutional Networks

Neural networks that make use of convolutional layers are called convolutional neural networks (CNNs). A common feature of CNN architecture is the repeated use of down-sampling convolutional layers, combined with a simultaneous increase in number of channels. The idea here is that each layer can assemble higher level features based on the lower level ones from the previous layer. The striding also causes the receptive field to increase from layer to layer, allowing deeper layer to detect features much larger than kernel size. The receptive field of a kernel is made up by all pixels in the input image which influence a given pixel in the feature map. Sometimes fully connected layers are placed at the end of the down-convolutional pipeline to perform nonlinear operations on the detected features.

Consider the example of classifying images based on whether they contain a dog. The first layers might detect basic features such as edges and circles, while later layers can use this information to detect more complex features such as a snout or tail, and finally a whole dog.

For tasks where both input and output have tensorial structure, CNNs commonly use a bottleneck architecture, similar to an autoencoder, as visualized in Figure 3.2.2. It consists of an encoding pipeline, which "featureizes" data, a bottleneck combining the high level features in latent space, and a decoder part that synthesizes the output using transpose convolutions or upconvolutions (sometimes misleadingly referred to as deconvolutions). Transpose convolutions are convolutions where striding is applied at the output

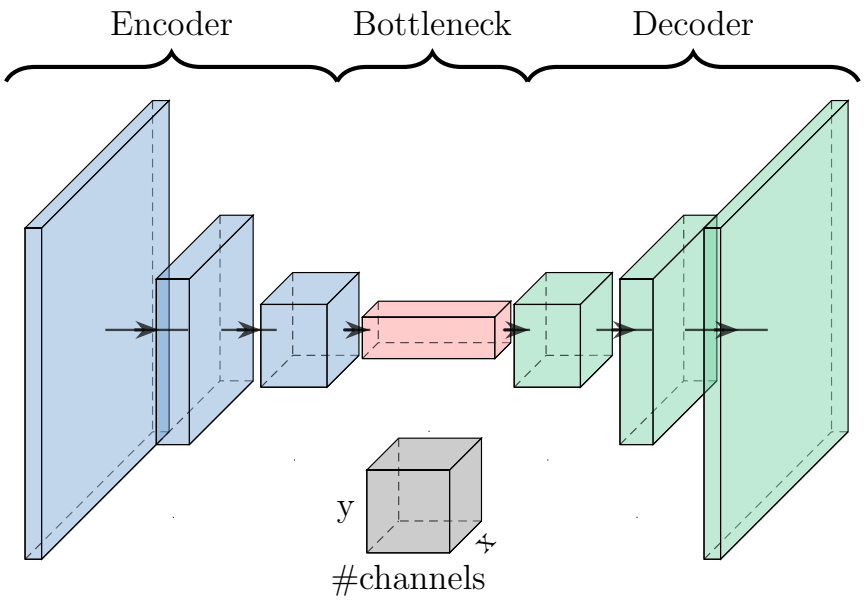

Figure 3.3: Convolutional bottleneck architecture (schematically). Visualization using [6].

instead of the input. The name stems from the fact that we can transform the down-convolution defined in (3.7) into an up-convolution by simply exchanging input with output and transposing the convolution matrix (see also [5]). It should be noted here that kernel weights of the up-convolutions are not inferred from those in the down convolution, but are learned during training.

Some architectures, such as the "U-Net" introduced by Ronneberger, Fischer, and Brox in [7] also introduce direct, so-called skip connections between down-convolution and up-convolution layers on the same level. This can help if some structure is shared exactly between input and output, as is the case in predicting a flow field from geometry. It also bear resemblance to the multigrid method from scientific computing, as explored in [8].

3.3 Generative Adversarial Networks

As deep learning was already achieving impressive results for discriminative problems, the field initially struggled to repeat the same for generative tasks. An important step in this area was the introduction of the framework of

adversarial networks in 2015 by Goodfellow et al. [9].

3.3.1 Motivation and Concepts

Before a neural network can be trained, one has to lay out the "grading scheme" that is the cost function. For some types of problems, such as classification, the choice is typically straightforward. Take again the task of detecting whether an image contains a dog. Given a set of labeled training data, we penalize incorrect guesses by the network, taking into account its level of confidence with the cross-entropy cost function. A much more difficult task, however, is to *generate* realistic images of dogs. The key difference is that it is difficult to define a cost function that properly accounts for the target distribution.

A naive approach could be to compare generated images to images from the training set using a simple pixel-based average loss. However this approach would only produce results akin to a superposition of training data, instead of realistic images. One may try to improve upon it by adding additional terms to the cost function such as penalties on blurred edges etc., but the network will still be essentially unaware of any meaningful conditional distributions underlying cat images and truly convincing results are likely to remain elusive.

GANs are designed to solve this problem in an elegant way. Instead of having to hand-craft the cost function that assesses the performance of the generator, it becomes part of the training process. This is done by training a second ANN, known as the discriminator, to classify images based on whether they are from the real data set or were output by the first network (known as the generator). Generator and discriminator are trained together in an adversarial zero sum game, where the former is trying to "fool" the latter, while the latter is trying to expose the "forgeries" of the former.

3.3.2 Fundamentals and Variants

The canonical generative setting in which GANs are applied is finding a useful mapping from a latent space input $\mathbf{z}$ to the probability distribution $p(\mathbf{x})$ underlying the training data $\mathbf{x} \in X$. A classical example is generating novel images of ostensibly human faces. However, GANs can also easily be extended to work with conditional distributions $p(\mathbf{x}|\mathbf{c})$ with conditional c. For the example of facial generation, $\mathbf{c}$ could be as simple as an integer value

$\in [0, 1]$ representing age, or as complex as a sketch acting as basis for the generation.

Generative problems fall on a spectrum; on the one side there are settings where we are actively interested in sampling the data distribution through the latent space, such as when building a universal face generator. On the other side, we have more translative problems, which may or may not be strictly deterministic, but for which we are typically only interested in getting a single, high-quality result (e.g. image upscaling). In this case, we may choose to omit the latent space altogether.

...

Chapter 4

Learning Fluid Dynamics

4.1 Machine Learning for Physical Modeling

The use of neural networks in finding approximative solutions to problems with an underlying partial differential equation (PDE) is an emerging field at the intersection of machine learning and scientific computing. Research activity in this area can be roughly split into at least three groups.

One group contains methods that seek to approximate the PDE solution directly using the parametrized function space given by the neural network, as already proposed by Lagaris et al. in 1997 [10]. In other words, the ANN itself serves as the discretization, in the sense of reducing the original problem to finite dimensionality. Perhaps the purest implementation of this approach are the physics-informed neural networks (PINNs) as introduced by Raissi et al [11, 12]. In this framework, the solution $u(x)$ is directly approximated by the neural network $\mathcal{N}(x;\omega)$.

The second group encompasses techniques that use machine learning to improve upon classical numerical methods. Published work on this includes areas such as RNN assisted solution upscaling for multigrid schemes [13], learning based preconditioning for domain decomposition methods [14] and use of neural networks for obtaining closure terms in turbulence modeling [15].

The method presented here belongs to a third group of using neural networks as reduced order surrogate models. Different to the first group, these operate on existing discretizations. The network is typically trained to map from a raster image of the domain, together with information about bound-

ary / initial conditions as well as additional non-dimensional parameters, onto one or multiple images showing the approximated solution fields. Due to the regular grid-like structure of inputs and outputs, these methods typically rely on a convolutional architecture. Examples include using neural networks to approximate steady state flow [16, 17]

As is always the case with reduced order models, we want to sacrifice some generality of our model for easier evaluation. The motivation for using ANNs in constructing the surrogate model is to be able to infer the essential relations in the examined model range directly from data. This could potentially reduce the need for hand-crafted domain specific models using assumptions based on expert knowledge. To draw an analogy, the learning process could be akin to how humans can build up a physical intuition without ever studying physics. Reduced order models can be very useful in many applications. One potential application is design space exploration, where the it could enable scanning a large number of configurations for optimality, or even providing design feedback to an engineer in close to real-time.

4.2 Use of GANs in Physical Modeling

Since the GAN framework was first proposed in 2015 [9], numerous studies have looked at applying it to modeling problems. Most of those fall into the second group as described above, i.e. they are surrogate models operating on existing discretizations.

One of the earliest attempts was by Farimani et al. in 2017 [18]. The authors trained GANs to solve 2D boundary value problems, specifically Laplace's equation and the incompressible steady-state Navier-Stokes equations (2.7). The generator was tasked with mapping from an image encoding the domain and boundary conditions, which is supplied as conditional input, to an output image showing the respective solution field. The generator loss function was a combination of the discriminator loss and a L1 loss with respect to the ground truth. For the case of Navier-Stokes, there are three input and output channels, one for each of the variables (velocity components u,v and pressure p). The PatchGAN architecture was used for the discriminator. The authors were able to obtain high accuracy on a test set with a relative mean absolute error (MAE) of less than 1%, and show that the neural network model outperforms state-of-the-art finite difference solvers in terms of prediction speed by an order of magnitude. The authors did not investigate

how much the adversarial part of the loss function actually improved results compared to a direct L1 ground-truth based training.

A number of studies have since applied GAN models to solve fluid problems in particular. In a 2020 paper, the authors used a GAN for predicting time series of convective flow with energy transport in a 2D square domain from initial and boundary conditions [19]. The method was found to provide fast and accurate solutions for the analyzed test cases. In another recent publication, the authors applied a GAN setup to model stationary flow through a more complex 3D domain of dispersed spherical obstacles, a relevant setting for modeling certain multiphase flow [20]. The authors found the GAN based result to outperform an older reduced order model that was developed specifically for this application, but did not comment on the relative computational effort.

Others have implemented GAN models for predicting stress in solids on a 2D domain with complex geometry [21]. In their publication, the authors supply separate images showing shape, loads and boundary conditions in the domain as input and extract a single output image of the domain showing von Mises stress. The generator uses a regular bottleneck architecture. The authors found the GAN to consistently outperform a previous model using a regular CNN architecture.

While all the work mentioned so far relies on a purely data-driven approach, other authors have investigated the effects of incorporating physics constraints into the training process. In one publication, a cGAN was trained to predict how a given flow field around a cylinder would evolve one time step into the future [22]. The authors compared four variants: Regular CNNs and GANs, one of each trained with and without a physics based loss contribution (conservation of mass and momentum). While all versions offered some success for prediction in unseen flow regimes, the GAN trained without physics based loss was found most successful at predicting recursively multiple time steps into the future.

There have already been more application-related studies published using GANs for fluid modeling as well. An architecture firm used a pix2pix GAN implementation to predict wind speeds in urban settings based on a building height map [23]. Again they trained the generator both on the discriminator and on the ground truth. Generalization performance was mixed, but the authors did not benchmark it against a comparable reduced order model.

Chapter 5

Initial Experiments

5.1 Methodology

5.1.1 Numerical Solution

5.2 Results

Chapter 6

Project Outlook & Scope

Bibliography

[1] Wojciech Ożański. "The Lagrange multiplier and the stationary Stokes equations". In: *Journal of Applied Analysis* 23.2 (Dec. 2017). DOI: 10.1515/jaa-2017-0017.

[2] C. S. Jog. *Fluid Mechanics*. Cambridge University Press, 2015. DOI: 10.1017/cbo9781316134030.

[3] Ian Goodfellow, Joshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press Ltd, 2016. 800 pp.

[4] Robert Service. "'The game has changed.' AI triumphs at solving protein structures". In: *Science* (Nov. 2020). DOI: 10.1126/science.abf9367.

[5] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: (Mar. 2016). arXiv: 1603.07285 [stat.ML].

[6] Haris Iqbal. *HarisIqbal88/PlotNeuralNet v1.0.0*. 2018. DOI: 10.5281/ZENODO.2526396.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Lecture Notes in Computer Science*. Springer International Publishing, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.

[8] Quang Tuyen Le and Chin Chun Ooi. "Surrogate Modeling of Fluid Dynamics with a Multigrid Inspired Neural Network Architecture". In: (May 2021). arXiv: 2105.03854 [physics.flu-dyn].

[9] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[10] I. E. Lagaris, A. Likas, and D. I. Fotiadis. "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations". In: (May 1997). DOI: 10.1109/72.712178. arXiv: physics/9705023 [physics.comp-ph].

[11] M. Raissi, P. Perdikaris, and G.E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (Feb. 2019), pp. 686–707. DOI: 10.1016/j.jcp.2018.10.045.

[12] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations". In: (Nov. 2017). arXiv: 1711.10566 [cs.AI].

[13] Nils Margenberg et al. "A neural network multigrid solver for the Navier-Stokes equations". In: (Aug. 2020). DOI: 10.1016/j.jcp.2022.110983. arXiv: 2008.11520 [physics.comp-ph].

[14] Alexander Heinlein et al. "Machine Learning in Adaptive Domain Decomposition Methods - Predicting the Geometric Location of Constraints". In: *SIAM Journal on Scientific Computing* 41.6 (Jan. 2019), A3887–A3912. DOI: 10.1137/18m1205364.

[15] Andrea Beck, David Flad, and Claus-Dieter Munz. "Deep neural networks for data-driven LES closure models". In: *Journal of Computational Physics* 398 (Dec. 2019), p. 108910. DOI: 10.1016/j.jcp.2019.108910.

[16] Xiaoxiao Guo, Wei Li, and Francesco Iorio. "Convolutional Neural Networks for Steady Flow Approximation". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, Aug. 2016. DOI: 10.1145/2939672.2939738.

[17] Axel Klawonn Matthias Eichinger Alexander Heinlein. *Surrogate Convolutional Neural Network Models for Steady Computational Fluid Dynamics Simulations*. Tech. rep. University of Cologne, 2020.

[18] Amir Barati Farimani, Joseph Gomes, and Vijay S. Pande. "Deep Learning the Physics of Transport Phenomena". In: (Sept. 2017). arXiv: 1709.02432 [cs.LG].

[19] Changlin Jiang and Amir Barati Farimani. "Deep Learning Convective Flow Using Conditional Generative Adversarial Networks". In: (May 2020). arXiv: 2005.06422 [physics.flu-dyn].

[20] B. Siddani et al. "Machine learning for physics-informed generation of dispersed multiphase flow using generative adversarial networks". In: *Theoretical and Computational Fluid Dynamics* 35.6 (Oct. 2021), pp. 807–830. DOI: 10.1007/s00162-021-00593-9.

[21] Haoliang Jiang et al. "StressGAN: A Generative Deep Learning Model for Two-Dimensional Stress Distribution Prediction". In: *Journal of Applied Mechanics* 88.5 (Feb. 2021). DOI: 10.1115/1.4049805.

[22] Sangseung Lee and Donghyun You. "Data-driven prediction of unsteady flow over a circular cylinder using deep learning". In: *Journal of Fluid Mechanics* 879 (Sept. 2019), pp. 217–254. DOI: 10.1017/jfm.2019.700.

[23] Aleksandra Sojka Sarah Mokhtar and Carlos Cerezo Davila. "Conditional Generative Adversarial Networks forPedestrian Wind Flow Approximation". In: *Proceedings of SimAUD*. 2020, pp. 25–27.