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

Chapter 1

Introduction

Chapter 2

Fluid Modeling

2.1 Governing Equations

2.2 Numerical Solution

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, roughly synonymous with artificial intelligence (AI), 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. To facilitate learning even for difficult problems, most modern AI methods 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.[1]

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, a topic 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.[CITE]

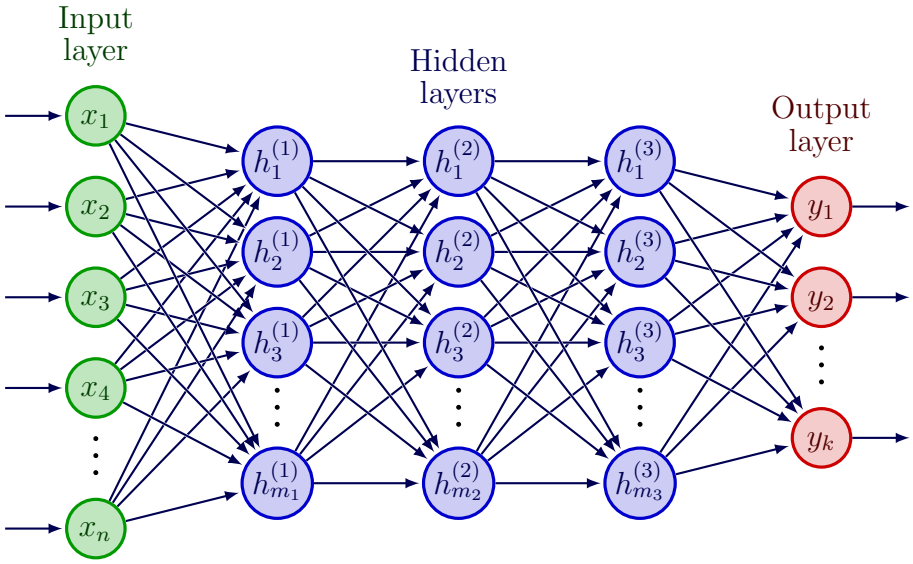

Figure 3.1: A graph visualizing the general structure of an artificial neural network with three hidden layers.

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 analogy to the networks formed by biological neurons found in the brains of animals. As the name suggests, ANNs are made up of individual neurons, which are arranged in connected layers. An example is visualized in figure 3.1. The generally 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.

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 function $f : \mathbb{R} \to \mathbb{R}$ after a fixed offset has been subtracted. Framed differently, the activation function

operates on the euclidean distance of point $\mathbf{x}$ from the hyperplane in the input space parametrized by the weight vector $\mathbf{w}$ and the offset θ, also known as the bias. We write

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

where the resulting output h is called the activation level, which is subsequently fed as input into some or all neurons in the next layer.

3.1.2 Network Architecture

Typically, the layers in a neural network are connected in a sequential order, such that information propagates from input to output, although other variations exist (recurrent neural networks, RNNs). Data is fed into the network at the input layer, in analogy to sensory organs in biological cognition. Then follow a number of so-called hidden layers, each operating on the output of the previous. Finally, the output layer is where the response of the network is read off. The whole network is essentially just a parametrized function $\mathcal{N}$ mapping from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$:

$$\begin{aligned} \mathcal{N} &: \mathcal{X} \to \mathcal{Y} \\ \hat{y} &= \mathcal{N}(x; \omega), \end{aligned} \tag{3.2}$$

where ω represents weights and biases of every neuron in the network, and $\hat{y}$ is the network's output. Typically, we take both x and ω as vectors in $\mathbb{R}$ such that

$$\mathcal{X} \subseteq \mathbb{R}^n, \quad \mathcal{Y} \subseteq \mathbb{R}^k.$$

The size of input and output layer (n and k respectively) is imposed by the problem setting, whereas the number and size of hidden layers is a design parameter. The size of input and output layers may be drastically different; take for instance the task of classifying 256x256 pixel images on the basis of whether or not they depict a cat. In this case, the network will have an input layer with size $n = 256^2 \approx 65\,000$, but only a single, binary output neuron. It should be noted that if the data has tensorial structure, it can make sense to reflect that in the notation. For our example, the mapping would then read $\mathbb{R}^{256\times256} \to \{0, 1\}$.

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 it is typically 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 y, i.e. we can denote the i^{th} training example as

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

...

3.1.4 Convolutional Layers

3.2 Generative Adversarial Networks

3.2.1 Motivation

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 for instance the task of detecting whether an image contains a cat. 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 cats. The key difference is that for the classification problem, we always know exactly in which direction the output should be moved in order to improve the result. For the generative problem on the other hand, this is all but clear.

A naive approach would be to compare generated images to images from the training set using a simple pixel-based loss. However this approach will

likely 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.

Taking a step back and seeing the problem from a higher vantage point, we may come to the conclusion that ideally, we would like to abandon simple cost functions in favor of having an observer that can judge the generated images for their realism. One may at first consider relying on human subjects for this task, similar to how they are usually employed to generate the labels for training data. However, apart from the fact that this would make the training process prohibitively slow and expensive, there is an even more fundamental limitation to this idea. In order to be useful for training, it is not sufficient to obtain a value representing the credibility of the image, but we also need to know how every pixel affected that judgment (and therefore what the local gradients are). This requirement is clearly infeasible with human subjects.

A much more practical approach would therefore be to first train a separate neural network on the aforementioned classification task of detecting images of cats, and afterwards use it to train the generative network (generator). This can be done by using the generated output as input for the discriminative network (discriminator), and back-propagate through both networks to obtain information on how to change the generators weights in order to increase the output of the discriminator (i.e. its confidence that the generated output is real). The issue with this approach is that the generator, since it has in a sense perfect knowledge of the inner workings of the discriminator, will typically find "shortcuts" to overpowering the discriminator by exploiting its imperfections and biases. In order to circumvent this, we train both generator and discriminator together in an adversarial zero sum game. This way both networks can be kept in balance, and improve each other until a satisfactory solution is reached. This framework is known as generative adversarial networks (GANs). It was first proposed in [2].

3.2.2 Fundamentals and Variants

The goal of training a GAN is typically to find a useful mapping from a latent space $\mathbf{z}$ to the probability distribution $p(\mathbf{x})$ underlying the training data $\mathbf{x} \in X$. A classical example is generating images of human faces. However, GANs can also easily be extended to work with conditional distributions $p(\mathbf{x}|\mathbf{c})$. For the example of facial generation, $\mathbf{c}$ could be as simple as a value $\in [0, 1]$ representing the lightness of the skin, or as complex as a sketch acting as basis for the generation.

Generative problems fall on a spectrum; on the one side there are setting 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.

...

3.3 Machine Learning for Fluid Modeling

An emerging field is the use of neural networks in particular as surrogate models for solutions to problems with an underlying partial differential equation (PDE). The hope is that the networks can learn useful relations from data, just as a human can build up a physical intuition. If successful, this could produce surrogate models that can be evaluated at much lower computational effort compared to a full numerical simulation, if perhaps at the cost of some loss of generality.

...

Chapter 4

Initial Experiments

4.1 Methodology

4.2 Results

Chapter 5

Project Scope

Bibliography

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

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