Artificial Intelligence (AI) and Deep Learning For CFD

Technical Report · July 2022

DOI: 10.13140/RG.2.2.22298.59847/6

1 author:

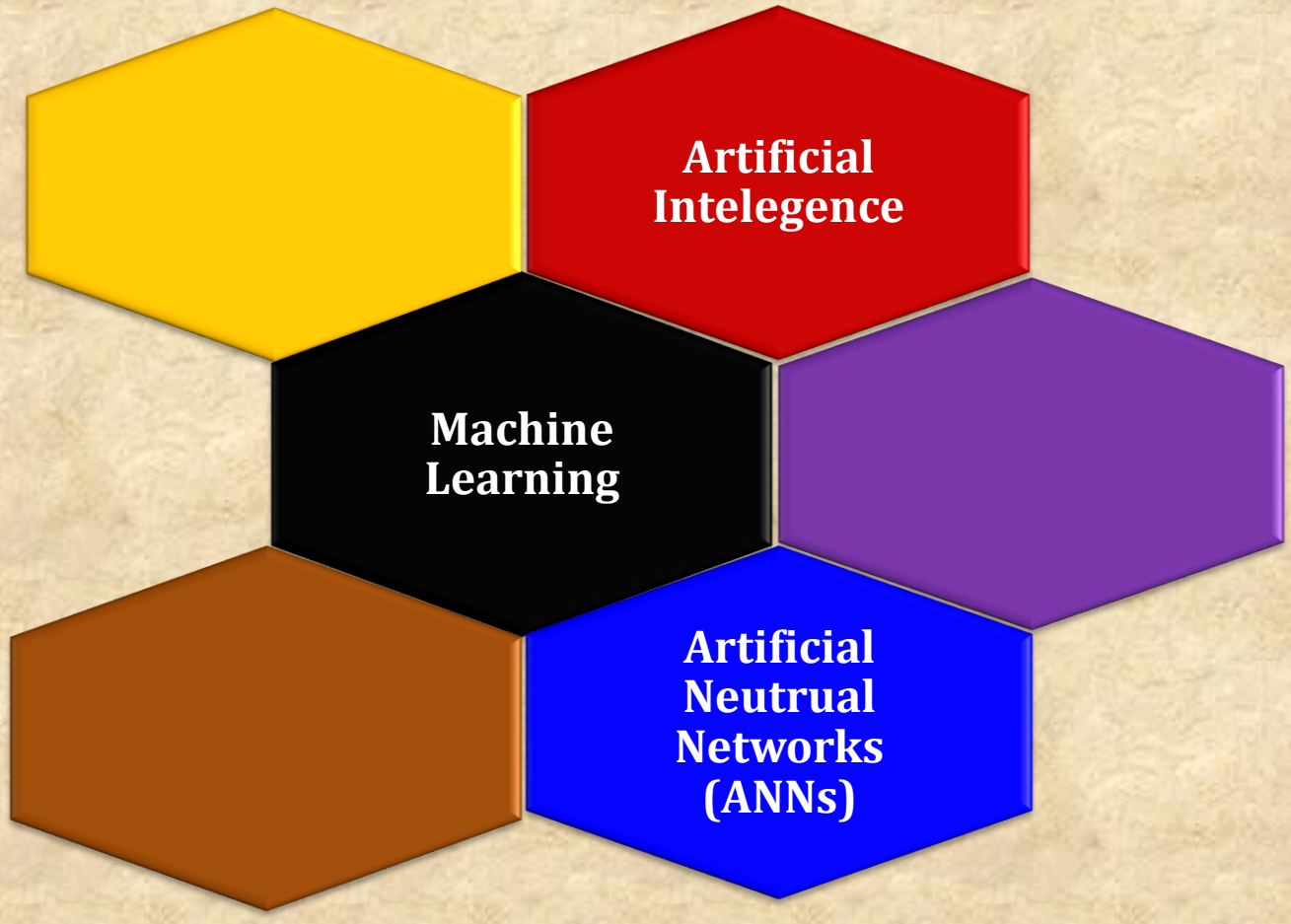
Ideen Sadrehaghighi
CFD Open Series
76 PUBLICATIONS **76** CITATIONS

SEE PROFILE

CFD Open Series/Patch 2.55

Artificial Intelligence (AI) and Deep Learning For CFD

Edited & Adapted by: Ideen Sadrehaghighi

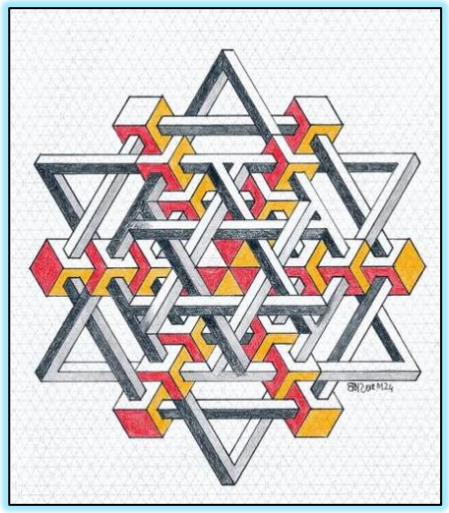


ANNAPOLIS, MD

Contents

List of Tables

List of Figures



1 Artificial Intelligence & Machine Learning in CFD

1.1 Background

Artificial Intelligence (AI) is the broadest way to think about advanced, computer intelligence. In 1956 at the Dartmouth Artificial Intelligence Conference, the technology was described as such: "***Every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.***" *AI* can refer to anything from a computer program playing a game of chess, to a voice-recognition system like Amazon's Alexa interpreting and responding to speech. IBM's Deep Blue, which beat chess grand master Garry Kasparov at the game in 1996, or Google DeepMind's Alpha Go, are examples of AI. According to *HackerEarth Blog*, AI can be classified into the following (see **Figure 1.1.1**) :

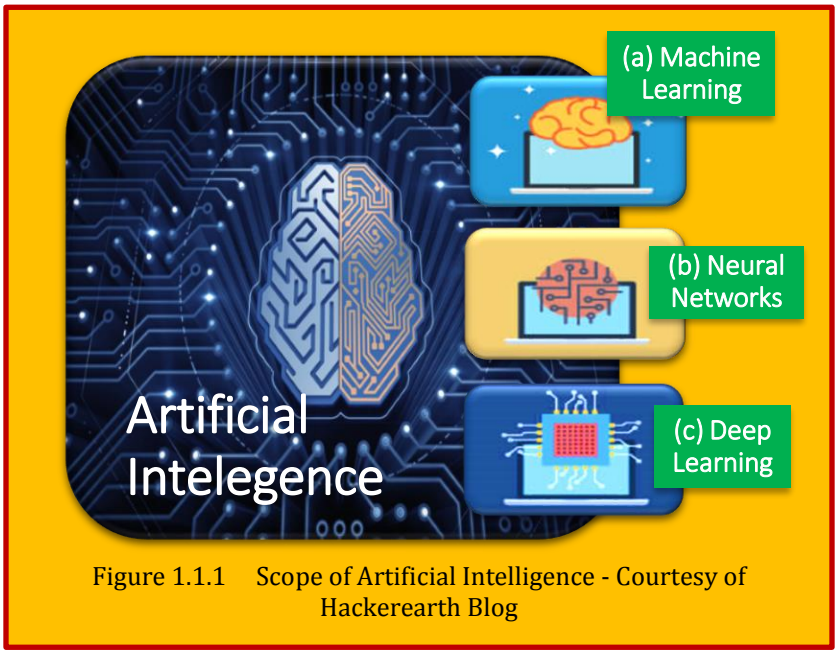


Figure 1.1.1 Scope of Artificial Intelligence - Courtesy of Hackerearth Blog

- Machine Learning
- Neural Networks
- Deep Learning

Other definitions provided by [*Kontos*][1], which defines A.I. as a single and consolidated discipline it might be better to consider as a set of different technologies that are easier to define individually. This set can include *data mining, question answering, self-aware systems, pattern recognition, knowledge representation, automatic reasoning, deep learning, expert systems, information extraction, text mining, natural language processing, problem solving, intelligent agents, logic programming, machine learning, artificial neural networks, artificial vision, computational discovery, computational creativity*. Therefore artificial ``*Self-aware*'' or ``*conscious*'' systems are the products of one of these technologies. Vargas et al.[2] describes the Deep Learning (DL) as an emerging area of Machine Learning (ML) research. It comprises multiple hidden layers of Artificial Neural Networks (ANN). The deep learning methodology applies nonlinear transformations and model abstractions of high level in large databases. The following (**Figure 1.2.1**) indicates the various area of Artificial Intelligence with relevant subject shown in red ellipse.

1.2 Machine Learning

Before we pay tribute to the field of machine learning in CFD, it best to go briefly of what is machine learning itself. ***Machine learning is a type of Artificial Intelligence (AI) that provides computers with the ability to learn without being explicitly programmed***. Machine learning focuses on the development of computer programs that can change when exposed to new data. In the past decade,

[1] John Kontos, "*Artificial Intelligence, Machine Consciousness and Explanation*", Academia Letters preprint, 2012.
[2] Vargas, R., Mosavi, A., & Ruiz, R. (2017). Deep learning: a review.

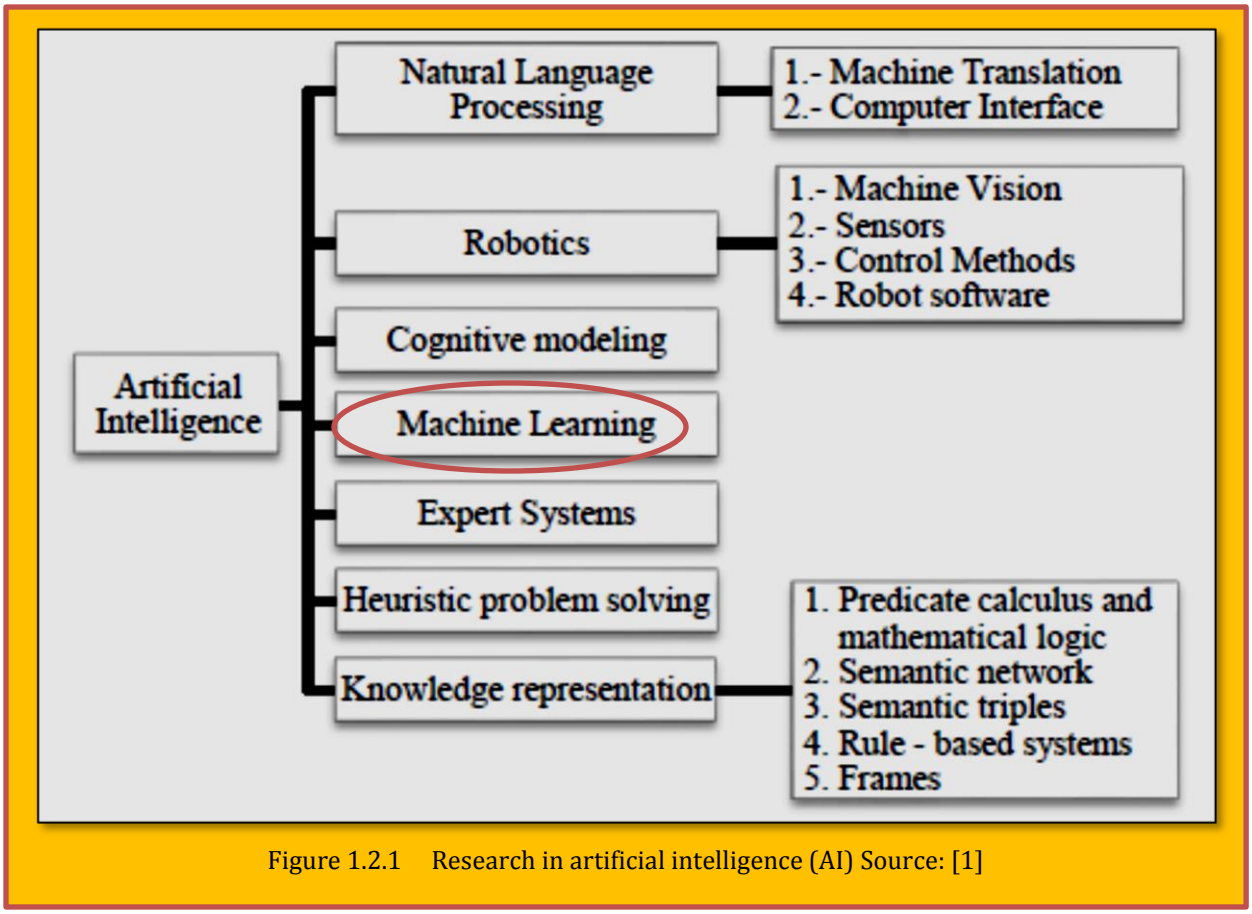


Figure 1.2.1 Research in artificial intelligence (AI) Source: [1]

machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome. Machine learning is so pervasive today that you probably use it dozens of times a day without knowing it. The process of machine learning is similar to that of ***data mining***. Both systems search through data to look for patterns. However, instead of extracting data for human comprehension as is the case in data mining applications machine learning uses that data to detect patterns in data and adjust program actions accordingly. Machine learning algorithms are often categorized as being ***supervised*** or ***un-supervised*** and ***reinforcement learning***. *Supervised* algorithms can apply what has been learned in the past to new data. *Un-supervised* algorithms can draw inferences from datasets. Facebook's News Feed uses machine learning to personalize each member's feed. If a member frequently stops scrolling in order to read or "like" a particular friend's posts, the News Feed will start to show more of that friend's activity earlier in the feed. Behind the scenes, the software is simply using statistical analysis and predictive analytics to identify patterns in the user's data and use to patterns to populate the News will be included in the data set and the News Feed will adjust accordingly. Google and Amazon are other heavy users of Machine Learning***. In essence, Machine learning (ML) is an algorithms that process and extract information from data. They facilitate automation of tasks and augment human domain knowledge. They are linked to learning processes and are categorized as supervised, semi-supervised, or unsupervised*** (Brunton, Noack, & Koumoutsakos, 2020)**.** *Reinforcement learning* is a third, large branch of machine learning research, in which an *agent* learns to make control decisions to interact with an environment for some high level objection[3].

[3] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, vol. 1. MIT Press, Cambridge (1998)

Examples include learning how to play games[4], such as chess. (Brunton, 2021)[5].

1.2.1 Creating Your First Machine Learning Model (Apples & Oranges)

Source : Newark.com

In ML, instead of defining the rules and expressing them in a programming language, answers (typically called *labels*) are provided with the data (see **Figure 1.2.2**). The machine will conclude the rules that determine the relationship between the labels and the data. The data and labels are used to create ML Algorithms, typically called models. Using this model, when the machine gets new data, it predicts or correctly labels them. If we train the model to discern between apples and oranges, the model can predict whether it is an apple or an orange when new data is presented. The problem sounds easy, but it is impossible to solve without ML. You'd need to write tons of rules to tell the difference between apples and oranges. With a new problem, you need to restart the process. There are many aspects of the fruit that we can collect data on, including color, weight, texture, and shape. For our purposes, we'll pick only two simple ones as data: weight and texture. In this article, we will explain how to create a simple ML algorithm that discerns between an apple and an orange. To discern between an apple and an orange, we create an algorithm that can figure out the rules so we don't have to write them by hand. And for that, we're going to train what's called a classifier. You can think of a classified as a function. It takes some data as input and assigns a label to it as output. The technique of automatically writing the classifier is called supervised learning.

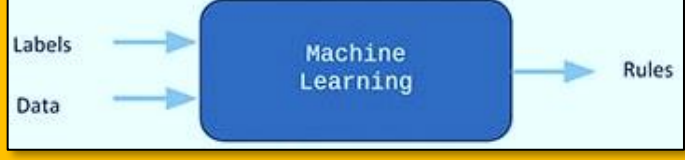


Figure 1.2.2 Machine Learning Programming

1.2.1.1 Supervised Learning

In supervised learning, the *training data* will have expert labels that should be predicted or modeled with the machine learning algorithm (Brunton, 2021)[4]. These output labels may be discrete, such as a categorical label of a "dog" or a "cat" given an input image, in which case the task is one of *classification*. If the labels are continuous, such as the average value of lift or drag given a specified airfoil geometry, then the task is one of *regression*. To use supervised learning, we follow a simple procedure with a few standard steps. The first step is to collect training data. These are essentially examples of the problem we want to solve. Step two is to use these examples to train a classifier. Once we have a trained classifier, the next step is to make predictions and classify a new fruit.

1.2.2 Collect Training Data

To collect training data, assume we head out to an orchard and collect some data. We look at different

[4] Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature 518, 529 (2015)

[5] arXiv:2110.02083 [physics.flu-dyn]

apples and oranges and write down their descriptive measurements in a table. In ML, these measurements are called features. To keep things simple, we've used only two types of data – how much each fruit weighs in grams and its texture, which can be bumpy or smooth. Each row in our training data depicts an example. It describes one piece of fruit. The last column is known as the label. It identifies what type of fruit is in each row, and in this case, there are only two possibilities – apples or oranges. The more training data you have, the better a classifier you create. (see **Table 1.2.1**).

Weight	*Texture*	*Label*
155	rough	Orange
180	rough	Orenge
135	smooth	apple
110	smooth	apple

Table 1.2.1 Data Considered

1.2.3 Training the Classifier

With the dataset prepared, the next step is to set up our training data and code it. Before we set up our training data, ensure the *scikit-learn* package is loaded. *Scikit-learn* provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. Now let's write down our training data in code. We will use two variables – features and labels.

```
features = [[155, "rough"], [180, "rough"],[135, "smooth"],[110, "smooth"]]
labels = ["orange", "orange", "apple", "apple"]
```

In the preceding code, the features contain the first two columns, and labels contain the last. Since scikit-learn works best with integers, we're going to change the variable types of all features to integers instead of strings – using 0 for rough and 1 for smooth. We will do the same for our labels – using 0 for apple and 1 for orange. The next step involves using these example features to train a classifier. The type of classifier we will use is called a decision tree. There are many different classifiers, but for simplicity, you think of a classified as a box of rules. Before we use our classifier, we must import the decision tree into the environment. Then on the next line in our script, we will create the classifier. (**Figure 1.2.3**).

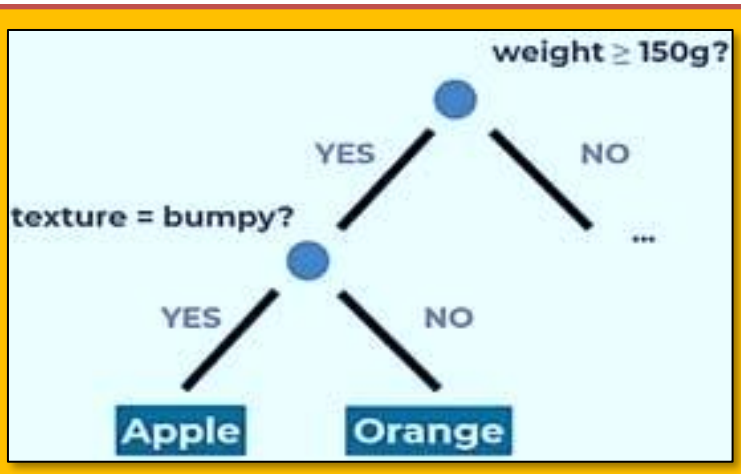


Figure 1.2.3 Decision Tree Classifier

1.2.4 Make Predictions

We have a trained classifier. Let's test it and use it to classify a new fruit. The input to the classifier is the feature for a new example. Let's say the fruit we want to classify is 150 grams and bumpy. Let's see if our ML algorithm can make such a prediction:

```
print (clf.predict(X = [[150, 0]]))
(1)
```

It works! The output is what we expected: 1 (orange). If everything worked for you, then congratulations! You have completed your first ML project in Python. You can create a new classifier for a new problem just by changing the training data. Fortunately, with the abundance of open source libraries and resources available today, programming with ML has become more comfortable and accessible to a rising number of users every day. Once you have a basic understanding of ML software programs and algorithms, you can scale your project using AI-based development boards. Decide on a hardware platform based on your application, and you are ready to go for real-world deployment.

1.2.5 Warming Up: Quadratic Equation

Consider a prototypical problem of finding roots of quadratic equation, $ax^2 + bx + c = 0$,

$$r_L, r_R = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Eq. 1.2.1

We would like to learn the **Eq. 1.2.1**

$$(a, b, c) \rightarrow (r_L, r_R)$$

Eq. 1.2.2

without relying on the knowledge of the underlying processes (Gyrya, Shashkov, Skurikhin, & Tokareva, 2019)[6]. For example, the relationship **Eq. 1.2.2** may represent a physical process for which some observations are available but the analytical relation **Eq. 1.2.1** has not yet been established. The prototypical problem of finding roots of a quadratic equation was selected as a proxy for the following reasons that are relevant to many complex practical problems:

- It is a fairly simple problem that is familiar to everyone who would be reading this paper. Yet, it is good representative a wide class of approximation problem in scientific computing.
- Finding solution involves different arithmetic operations some of which could be difficult to model by machine learning techniques. For example, division and taking of a square root represent a challenge for neural networks to capture exactly using activation functions.
- There are situations when a particular form of analytic expression/algorithm may exhibit loss of accuracy. For example, the analytic expression **Eq. 1.2.1** for the larger root is numerically inaccurate when b is much larger than 4ac.
- The roots of quadratic equation under certain condition exhibit some non-trivial behavior. There are several branches in the solution: if a = 0, the quadratic equation becomes a linear equation, which has one root – this is a qualitative change from one regime to a different one; depending on the discriminant the number of roots as well as the nature of the roots changes (real vs. complex).
- Probably, the most significant challenge from the standpoint of ML is that there is a small range of input parameters for which output values are increasingly large (corresponding to small values of a).

We will now explain what we mean by learning the relation **Eq. 1.2.2**. Assume we are provided a number of observations (training set):

$$\left(a^i, b^i, c^i\right) \rightarrow \left(r_L^i, r_R^i\right) \quad , i = 1, 2, , , , , , , N$$

Eq. 1.2.3

where $(r^i_L; r^i_R)$ satisfy **Eq. 1.2.1** exactly. From the training data **Eq. 1.2.3** we will try to predict the relation **Eq. 1.2.2** on a new previously unseen data $(a^j; b^j; c^j)$:

$$\left(a^j, b^j, c^j\right) \rightarrow \left(\bar{r}_L^j, \bar{r}_R^j\right) \approx \left(r_L^j, r_R^j\right) \quad , \quad j = N + 1, , , , , , , , N + K$$

Eq. 1.2.4

The goal is to minimize mismatches between the estimates $(\tilde{r}^j_L; \tilde{r}^j_R)$ and the testing data $(r^j_L; r^j_R)$

---

[6] Gyrya, V., Shashkov, M., Skurikhin, A., & Tokareva, S. (2019). Machine learning approaches for the solution of the Riemann problem in fluid dynamics: a case study. *Journal of Computational Physics*.

$$\text{Cost} = \sum_{j} \left(r_L^j - \bar{r}_L^j \right)^2 + \sum_{j} \left(r_R^j - \bar{r}_R^j \right)^2$$

Eq. 1.2.5

Since the testing data is not available during the training process the minimization is performed on the training set with the idea that the training and the testing set are selected from the same pool. The above setup is the typical ML setup. In this work our goal was to compare the performance of several existing ML approaches for the case of a quadratic equation.

1.3 Difference Between Artificial Intelligence and Machine Learning

Artificial Intelligence (AI) is a computer program that does something smart. It can be a pile of statements or a complex statistical model. Usually, when a computer program designed by AI researchers actually succeeds at something; like winning at chess many people say it's "not really intelligent", because the algorithms internals are well understood. So you could say that true AI is whatever computers can't do yet. Machine learning, as others here have said, is a subset of AI. In short, ***Machine learning is a science that involves development of self-learning algorithms*****.** These algorithms are more generic in nature that it can be applied to various domain related problems. Machine learning uses statistics (mostly inferential statistics) to develop self-learning algorithms. ***Artificial Intelligence is a science to develop a system or software to mimic human to respond and behave in a circumstance.*** As field with extremely broad scope, AI has defined its goal into multiple chunks. Later each chuck has become a separate field of study to solve its problem. The "learning" part of Machine Learning means that ML algorithms attempt to optimize along a certain dimension; i.e. they usually try to minimize error or maximize the likelihood of their predictions being true. How does one minimize error? Well, one way is to build a framework that multiplies inputs in order to make guesses as to the inputs' nature. Different outputs/guesses are the product of the inputs and the algorithm. Usually, the initial guesses are quite wrong, and if you are lucky enough to have ground-truth labels pertaining to the input, you can measure how wrong your guesses are by contrasting them with the truth, and then use that error to modify your algorithm. That's what ***Artificial Neural Networks (ANN)*** do. They keep on measuring the error and modifying their parameters until they can't achieve any less error. They are, in short, an optimization algorithm. If you tune them right, they minimize their error by guessing and guessing and guessing again.
Another point of view expressed by (Pandey, Schumacher, & Sreenivasan, 2020)[7] is that ***while ML is sometimes regarded as a subset of AI, there are some differences in usage. AI mimics natural intelligence to solve complex problems and enables decision making; efficiency is not its main driver, and it is an intelligence capability which we want to build into all machines. Machine learning, on the other hand, is about improving and maximizing performance by means of self-learning algorithms.*** Both of them require large databases from which to learn: the more the high-quality data that becomes available, the better the results, hence the close connection of AI and ML to Big Data.

[7] Pandey, S., Schumacher, J., & Sreenivasan, K. R. (2020). A perspective on machine learning in turbulent flows. *Journal of Turbulence*.

1.4 Deep Learning

Using ***Machine Learning*** to build data-driven models in fluid mechanics is usually achieved by ***Artificial Neural Networks (ANN).*** The situation is commonly refer to as ***Deep learning***. It is also important to realize that machine learning is not an automatic or turn-key procedure for extracting models from data. Instead, it requires expert human guidance at every stage of the process, from deciding on the problem, to collecting and curating data that might inform the model, (Brunton, 2021)[8]. The ***ANNs*** are inspired by our understanding of the biology of our brains all those interconnections between the neurons (Copeland, 2010)[9]. But, unlike a biological brain where any neuron can connect to any other neuron within a certain physical distance, these artificial neural networks have discrete layers, connections, and directions of data propagation. You might, for example, take an image, chop it up into a bunch of tiles that are inputted into the first layer of the neural network. In the first layer individual neurons, then passes the data to a second layer. The second layer of neurons does its task, and so on, until the final layer and the final output is produced. Each neuron assigns a weighting to its input; how correct or incorrect it is relative to the task being performed. The final output is then determined by the total of those weightings. So think of our stop sign example. Attributes of a stop sign image are chopped up and "examined" by the neurons its octagonal shape, its fire-engine red color, its distinctive letters, its traffic-sign size, and its motion or lack thereof. The neural network's task is to conclude whether this is a stop sign or not. It comes up with a "***probability vector***," really a highly educated guess, based on the weighting. In our example the system might be 86% confident the image is a stop sign, 7% confident it's a speed limit sign, and 5% it's a kite stuck in a tree ,and so on and the network architecture then tells the neural network whether it is right or not. In short, ***Deep Learning is a technique for implementing Machine Learning.*** Deep Learning has enabled many practical applications of Machine Learning and by extension the overall field of AI as perceived in **Figure 1.4.1**. Deep Learning breaks down tasks in ways that makes all kinds of machine assists seem possible, even likely. Driverless cars, better preventive healthcare, even better movie recommendations, are all here today or on the horizon. Today, image recognition by machines trained via deep learning in some scenarios is better than humans, and that ranges from cats to identifying indicators for cancer in blood and tumors in MRI scans. Google's AlphaGo learned the game, and trained for its Go match it tuned its neural network by playing against itself over and over and over. As a starting point, users are encouraged to the applicability of a ***Deep Neural Network***

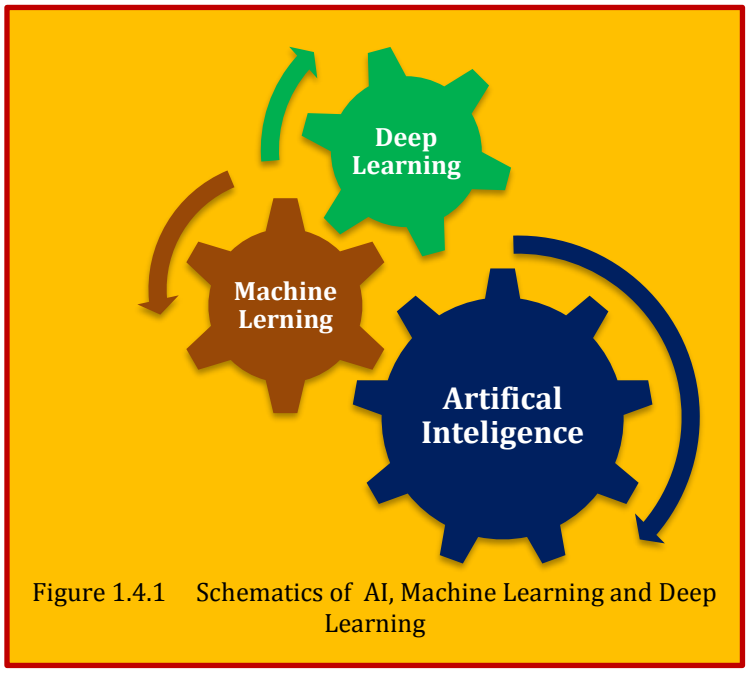


Figure 1.4.1 Schematics of AI, Machine Learning and Deep Learning

[8] arXiv:2110.02083 [physics.flu-dyn]

[9] Copeland, M. (2010). *What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?*

(DNN) approach to simulate one-dimensional non-relativistic fluid dynamics, as presented by (Taradiy et al.)[10].

1.5 Types of Problems and Tasks

Machine learning tasks are typically classified into three broad categories, depending on the nature of the learning "signal" or "feedback" available to a learning system. These are:

1.5.1 Supervised Learning

This algorithm consist of a target/outcome variable (or dependent variable) which is to be predicted from a given set of predictors (independent variables). Using these set of variables, we generate a function that map inputs to desired outputs. The training process continues until the model achieves a desired level of accuracy on the training data. Examples of Supervised Learning: Regression, Decision Tree, Random Forest, KNN, Logistic Regression etc.[11]

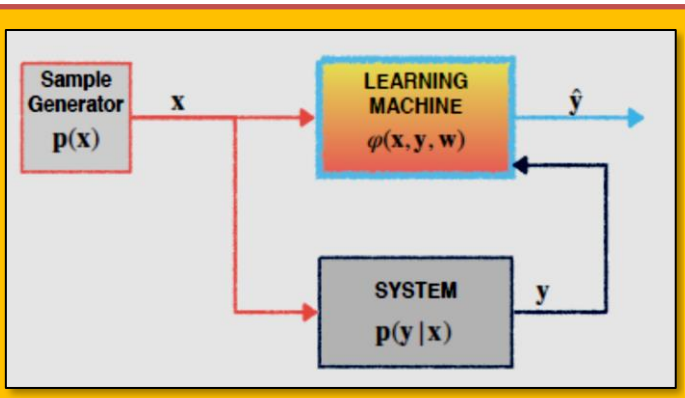


Figure 1.5.1 A Learning Machine Uses Inputs From a Sample Generator and Observations from a System to Generate an Approximation of its Output (Credit: Cherkassky & Mulier (2007))

Supervised learning implies the availability of corrective information to the learning machine. In its simplest and most common form, this implies labeled training data, with labels corresponding to the output of the ML. Minimization of the cost function, which implicitly depends on the training data, will determine the unknown parameters of the LM. In this context, supervised learning dates back to the regression and interpolation methods proposed centuries ago (**Figure 1.5.1**). A commonly employed loss function is :

$$L(y, \varphi(x, y, w) = \|y - \varphi(x, y, w)\|^2$$

Eq. 1.5.1

Alternative loss functions may reflect different constraints on the learning machine such as sparsity. The choice of the approximation function reflects prior knowledge about the data and the choice between linear and nonlinear methods directly bears on the computational cost associated with the learning methods[12].

1.5.2 Unsupervised Learning

In this algorithm, we do not have any target or outcome variable to predict/estimate. It is used for clustering population in different groups, which is widely used for segmenting customers in different groups for specific intervention. Examples of Unsupervised Learning: A priori algorithm, K-means.

1.5.3 Reinforcement Learning

Using this algorithm, the machine is trained to make specific decisions. It works this way: the machine is exposed to an environment where it trains itself continually using trial and error. This machine

---

[10] Kirill Taradiy, Kai Zhou, Jan Steinheimer, Roman V. Poberezhnyuk, Volodymyr Vovchenko, and Horst Stoecker, "*Machine learning based approach to fluid dynamics*", arXiv:2106.02841v1 [physics .comp-ph], 2021.
[11] Sunil, Ray, "Essentials of Machine learning Algorithms (with Python and R codes)", August 2015.
[12] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos, "*Machine Learning for Fluid Mechanics*", Annual Review of Fluid Mechanics , January 2020.

learns from past experience and tries to capture the best possible knowledge to make accurate business decisions. Example of Reinforcement Learning: Markov Decision Process[13].

1.6 List of Common Machine Learning Algorithms

Here is the list of commonly used machine learning algorithms. These algorithms can be applied to almost any data problem where we explain a little bit of the first four.

- Linear Regression
- Logistic Regression
- Decision Tree
- Artificial Neural Networks (ANNs)
- Support Vector Machine (SVM)
- Naive Bayes
- K-Nearest Neighbors (KNN)
- K-Means
- Random Forest
- Dimensionality Reduction Algorithms
- Gradient Boost

1.6.1 Linear Regression

It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation Y = a * X + b. The best way to understand linear regression is to relive this experience of childhood. Let us say, you ask a child in fifth grade to arrange people in his class by increasing order of weight, without asking them their weights! What do you think the child will do? He / she would likely look (visually analyze) at the height and build of people and arrange them using a combination of these visible parameters. This is linear regression in real life! The child has actually figured out that height and build would be correlated to the weight by a relationship, which looks like the equation above. In this equation, Y-Dependent Variable, a-Slope, X-Independent variable and b-Intercept.

These coefficients a and b are derived based on minimizing the sum of squared difference of distance between data points and regression line. Look at the below example. Here we have identified the best fit line having linear equation y = 0.2811 x +13.9 (see **Figure 1.6.1**). Now using this equation, we can find the weight, knowing the height of a person. Linear Regression is of mainly two types: Simple Linear Regression and Multiple Linear Regression. Simple Linear Regression is characterized by one independent variable. And, Multiple Linear Regression(as the name suggests) is characterized by multiple (more than 1) independent

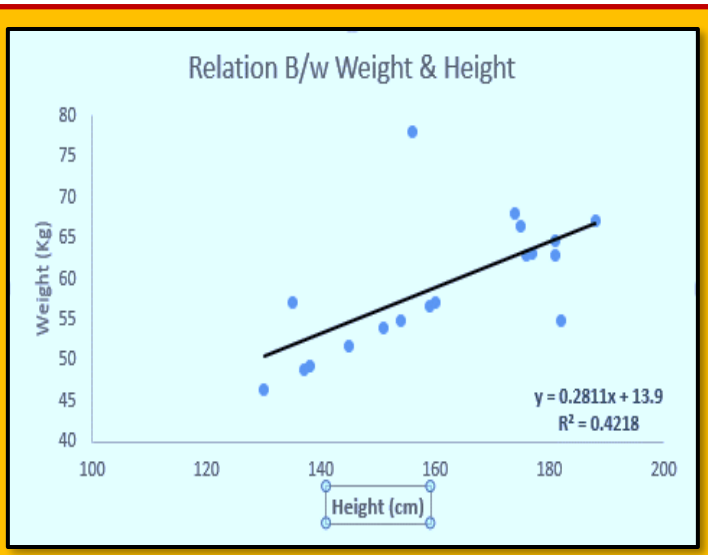


Figure 1.6.1 Linear Regression

[13] same as previous.

variables. While finding best fit line, you can fit a polynomial or curvilinear regression. And these are known as polynomial or curvilinear regression[14].

1.6.2 Logistic Regression

Don't get confused by its name! It is a classification not a regression algorithm. It is used to estimate discrete values (Binary values like 0/1, yes/no, true/false) based on given set of independent variable(s). In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function. Hence, it is also known as logistic regression. Since, it predicts the probability, its output values lies between 0 and 1 (as expected). Again, let us try and understand this through a simple example. Let's say your friend gives you a puzzle to solve. There are only 2 outcome scenarios ; either you solve it or you don't. Now imagine, that you are being given wide range of puzzles/ quizzes in an attempt to understand which subjects you are good at. The outcome to this study would be something like this ; if you are given a trigonometry based tenth grade problem, you are 70% likely to solve it. On the other hand, if it is grade fifth history question, the probability of getting an answer is only 30%. This is what Logistic Regression provides you. Coming to the math, the log odds of the outcome is modeled as a linear combination of the predictor variables odds = p/(1 - p) = probability of event occurrence / probability of not event occurrence. ln(odds) = ln(p/(1 - p)), logit(p) = ln(p/(1 - p)). Above, p is the probability of presence of the characteristic of interest. It chooses parameters that maximize the likelihood of observing the sample values rather than that minimize the sum of squared errors (like in ordinary regression). Now, you may ask, why take a log? For the sake of simplicity, let's just say that this is one of the best mathematical way to replicate a step function. It can go in more details, but that will beat the purpose of this article.

1.6.3 Decision Tree

This is favorite algorithm and used it quite frequently. It is a type of supervised learning algorithm that is mostly set for classification problems[15]. Surprisingly, it works for both categorical and continuous dependent variables. In this algorithm, we split the population into two or more homogeneous sets. This is done based on most significant attributes/ independent variables to make as distinct groups as possible. In the image above, you can see that population is classified into four different groups based on multiple attributes to identify 'if they will play or not'. To split the population into different heterogeneous groups, it uses various techniques (**Figure 1.6.2**).

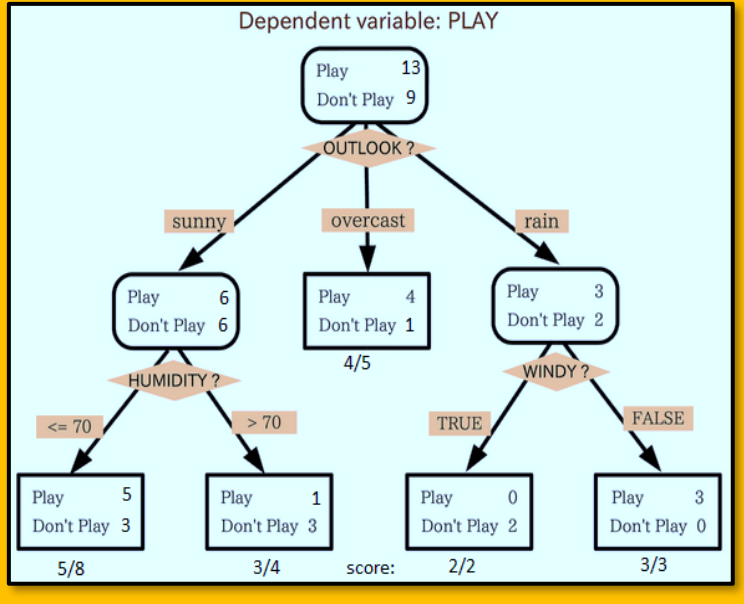


Figure 1.6.2 Decision Tree

[14] Sunil, Ray, "Essentials of Machine learning Algorithms (with Python and R codes)", August 2015.
[15] Sunil, Ray, "*Essentials of Machine learning Algorithms (with Python and R codes)*", August 2015.

1.7 Artificial Neural Networks (ANNs)

Computational model used in machine learning, computer science and other research disciplines, which is based on a large collection of connected simple units called ***artificial neurons,*** loosely analogous to axons in a biological brain. Connections between neurons carry an activation signal of varying strength. If the combined incoming signals are strong enough, the neuron becomes activated and the signal travels to other neurons connected to it. Such systems can be trained from examples, rather than explicitly programmed, and excel in areas where the solution or feature detection is difficult to express in a traditional computer program. Like other machine learning methods, neural networks have been used to solve a wide variety of tasks, like computer vision and speech recognition, that are difficult to solve using ordinary rule-based programming. Typically, neurons are connected in layers, and signals travel from the first (input), to the last (output) layer. Modern neural network projects typically have a few thousand to a few million neural units and millions of connections; their computing power is similar to a worm brain, several orders of magnitude simpler than a human brain. The signals and state of artificial neurons are real numbers, typically between 0 and 1. There may be a threshold function or limiting function on each connection and on the unit itself, such that the signal must surpass the limit before propagating. Back propagation is the use of forward stimulation to modify connection weights, and is sometimes done to train the network using known correct outputs. However, the success is unpredictable: after training, some systems are good at solving problems while others are not. Training typically requires several thousand cycles of interaction, (see **Figure 1.7.1**).

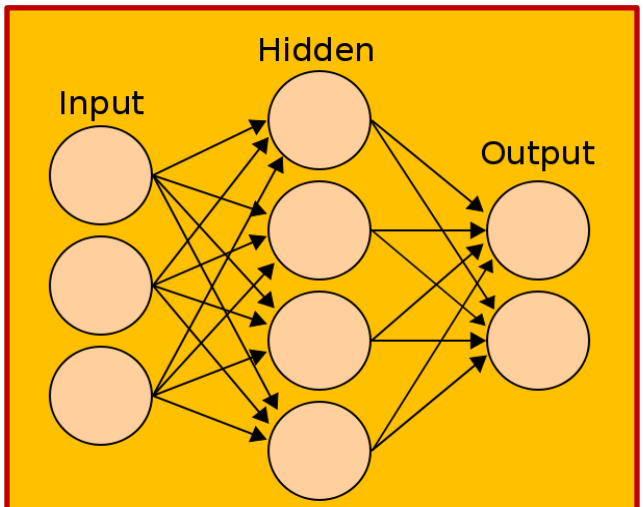


Figure 1.7.1 Artificial Neural Network (ANN)

The goal of the neural network is to solve problems in the same way that a human would, although several neural network categories are more abstract. New brain research often stimulates new patterns in neural networks. One new approach is use of connections which span further to connect processing layers rather than adjacent neurons. Other research being explored with the different types of signal over time that axons propagate, such as deep learning, interpolates greater complexity than a set of Boolean variables being simply on or off. Newer types of network are more free flowing in terms of stimulation and inhibition, with connections interacting in more chaotic and complex ways. Dynamic neural networks are the most advanced, in that they dynamically can, based on rules, form new connections and even new neural units while disabling others. Historically, the use of neural network models marked a directional shift in the late 1980s from high-level (symbolic) artificial intelligence, characterized by expert systems with knowledge embodied in ***if-then*** rules, to low-level (sub-symbolic) machine learning, characterized by knowledge embodied in the parameters of a cognitive model with some dynamical system. A simple example provided below demonstrates a better explanation.

1.7.1 Types of Neutral Networks

1.7.1.1 Perceptron

The Perceptron is the most basic and oldest form of neural networks (Pavan Vadapalli)[16]. It consists of just 1 neuron which takes the input and applies activation function on it to produce a binary output.

[16] **Pavan Vadapalli** Director of Engineering @ upGrad, 2020.

It doesn't contain any hidden layers and can only be used for binary classification tasks. The neuron does the processing of addition of input values with their weights. The resulted sum is then passed to the activation function to produce a binary output. (see **Figure 1.7.2**).

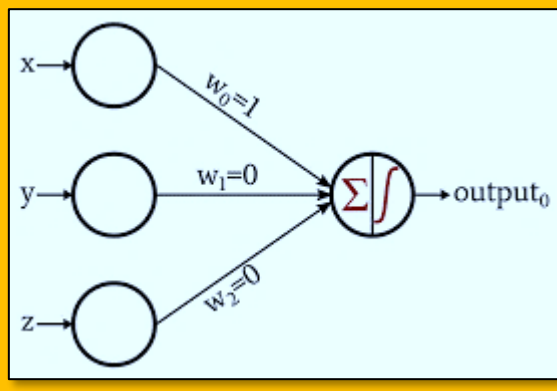


Figure 1.7.2 Perceptron

1.7.1.2 Feed Forward Network

The ***Feed Forward (FF)*** networks consist of multiple neurons and hidden layers which are connected to each other. These are called "feed-forward" because the data flow in the forward direction only, and there is no backward propagation (**Figure 1.7.1**). Hidden layers might not be necessarily present in the network depending upon the application. More the number of layers more can be the customization of the weights. And hence, more will be the ability of the network to learn. Weights are not updated as there is no backpropagation. The output of multiplication of weights with the inputs is fed to the activation function which acts as a threshold value.

1.7.1.3 Multi-Layer Perceptron

The main shortcoming of the Feed Forward networks was its inability to learn with backpropagation. ***Multi-layer Perceptron's*** are the neural networks which incorporate multiple hidden layers and activation functions (see **Figure 1.7.3**). The learning takes place in a Supervised manner where the ***weights*** are updated by the means of ***Gradient Descent***. Multi-layer Perceptron is bi-directional, i.e., Forward propagation of the inputs, and the backward propagation of the weight updates. The activation functions can be changes with respect to the type of target. These are also called dense networks because all the neurons in a layer are connected to all the neurons in the next layer. They are used in Deep Learning based applications but are generally slow due to their complex structure.

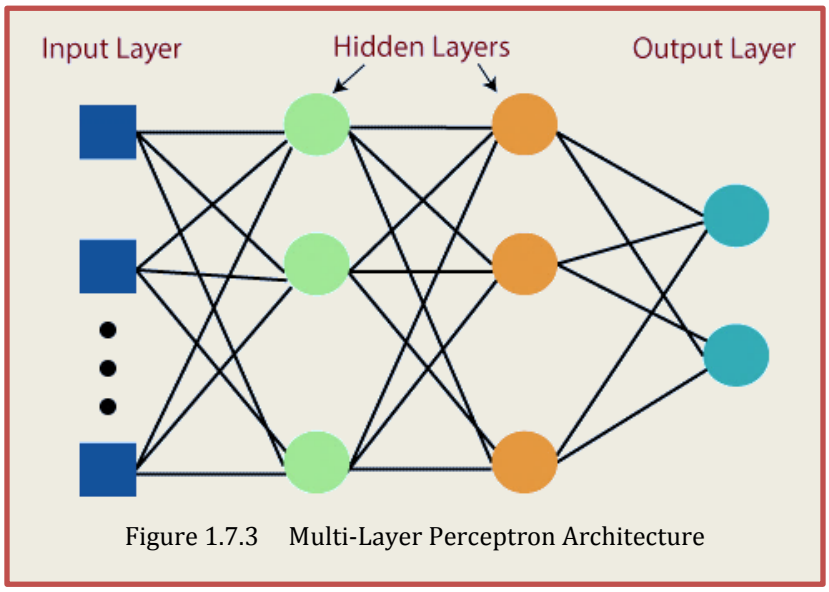


Figure 1.7.3 Multi-Layer Perceptron Architecture

1.7.1.4 Radial Basis Networks

Radial Basis Networks (RBN) use a completely different way to predict the targets. It consists of an input layer, a layer with RBF neurons and an output. The RBF neurons store the actual classes for

each of the training data instances. The RBN are different from the usual Multilayer perceptron because of the Radial Function used as an activation function.
When the new data is fed into the neural network, the RBF neurons compare the Euclidian distance of the feature values with the actual classes stored in the neurons. This is similar to finding which cluster to does the particular instance belong. The class where the distance is minimum is assigned as the predicted class. The RBNs are used mostly in function approximation applications like Power Restoration systems. (**Figure 1.7.4**).

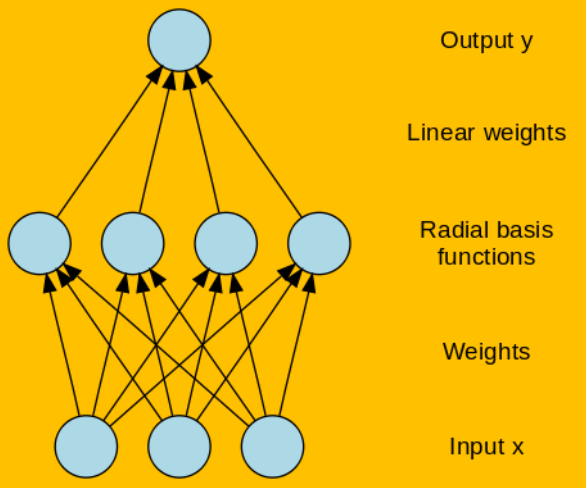


Figure 1.7.4 Radial Basis Function

1.7.1.5 Convolutional Neural Networks

When it comes to image classification, the most used neural networks are ***Convolution Neural Networks (CNN)***. CNN contain multiple convolution layers which are responsible for the extraction of important features from the image (**Figure 1.7.5**). The earlier layers are responsible for low-level details and the later layers are responsible for more high-level features. The Convolution operation uses a custom matrix, also called as filters, to convolute over the input image and produce maps. These filters are initialized randomly and then are updated via backpropagation. One example of such a filter is the Canny Edge Detector, which is used to find the edges in any image.

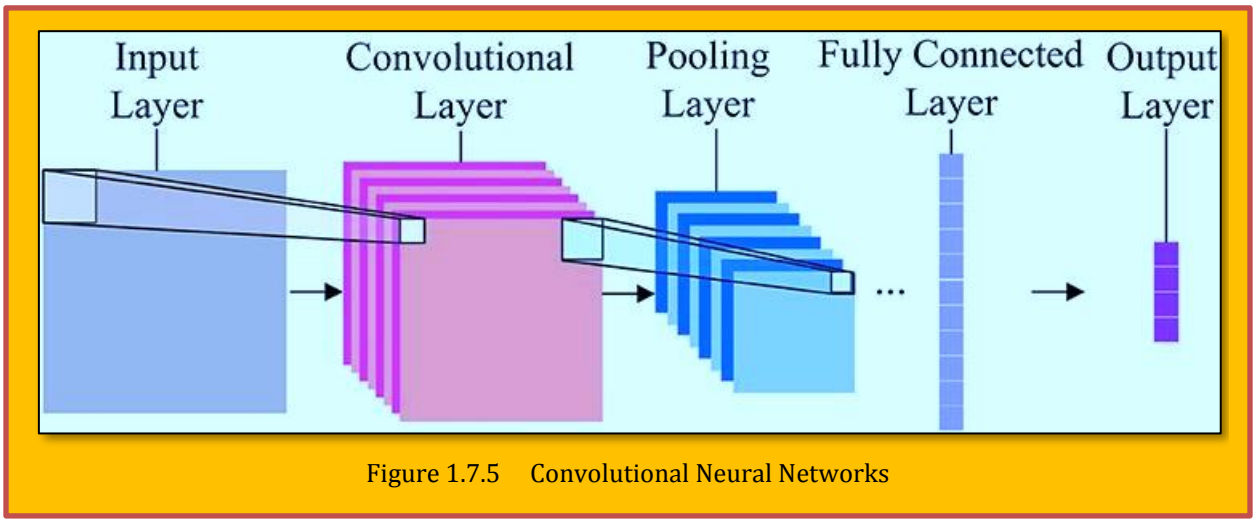


Figure 1.7.5 Convolutional Neural Networks

After the convolution layer, there is a pooling layer which is responsible for the aggregation of the maps produced from the convolutional layer. It can be Max Pooling, Min Pooling, etc. For regularization, CNNs also include an option for adding dropout layers which drop or make certain neurons inactive to reduce overfitting and quicker convergence. CNNs use ReLU (Rectified Linear Unit) as activation functions in the hidden layers. As the last layer, the CNNs have a fully connected dense layer and the activation function mostly as *Softmax* for classification, and mostly ReLU for regression.

1.7.1.6 Recurrent Neural Networks

Recurrent Neural Networks (RNN) come into picture when there's a need for predictions using sequential data (**Figure 1.7.6**). Sequential data can be a sequence of images, words, etc. The RNN have a similar structure to that of a Feed-Forward Network, except that the layers also receive a time-delayed input of the previous instance prediction. This instance prediction is stored in the RNN cell which is a second input for every prediction. However, the main disadvantage of RNN is the Vanishing Gradient problem which makes it very difficult to remember earlier layers' weights.

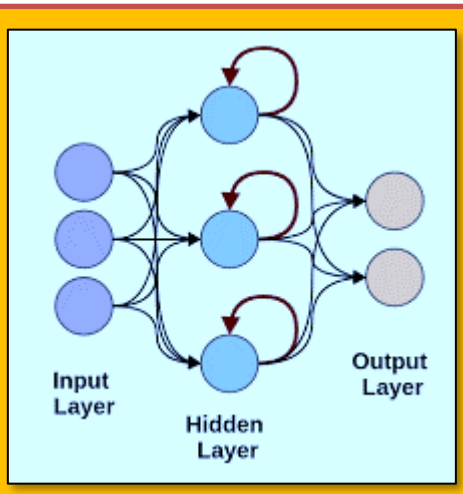


Figure 1.7.6 Recurrent Neural Networks (RNN)

1.7.1.7 Physics-informed neural networks

Physics-informed neural networks (PINNs) are a type of universal function approximators that can embed the knowledge of any physical laws that govern a given data-set in the learning process, and can be described by (PDEs)[17] (see **Figure 1.7.7**). They overcome the low data availability of some biological and engineering systems that makes most state-of-the-art machine learning techniques lack robustness, rendering them ineffective in these scenarios. The prior knowledge of general physical laws acts in the training of neural networks (NNs) as a regularization agent that limits the space of admissible solutions, increasing the correctness of the function approximation. This way, embedding this prior information into a neural network results

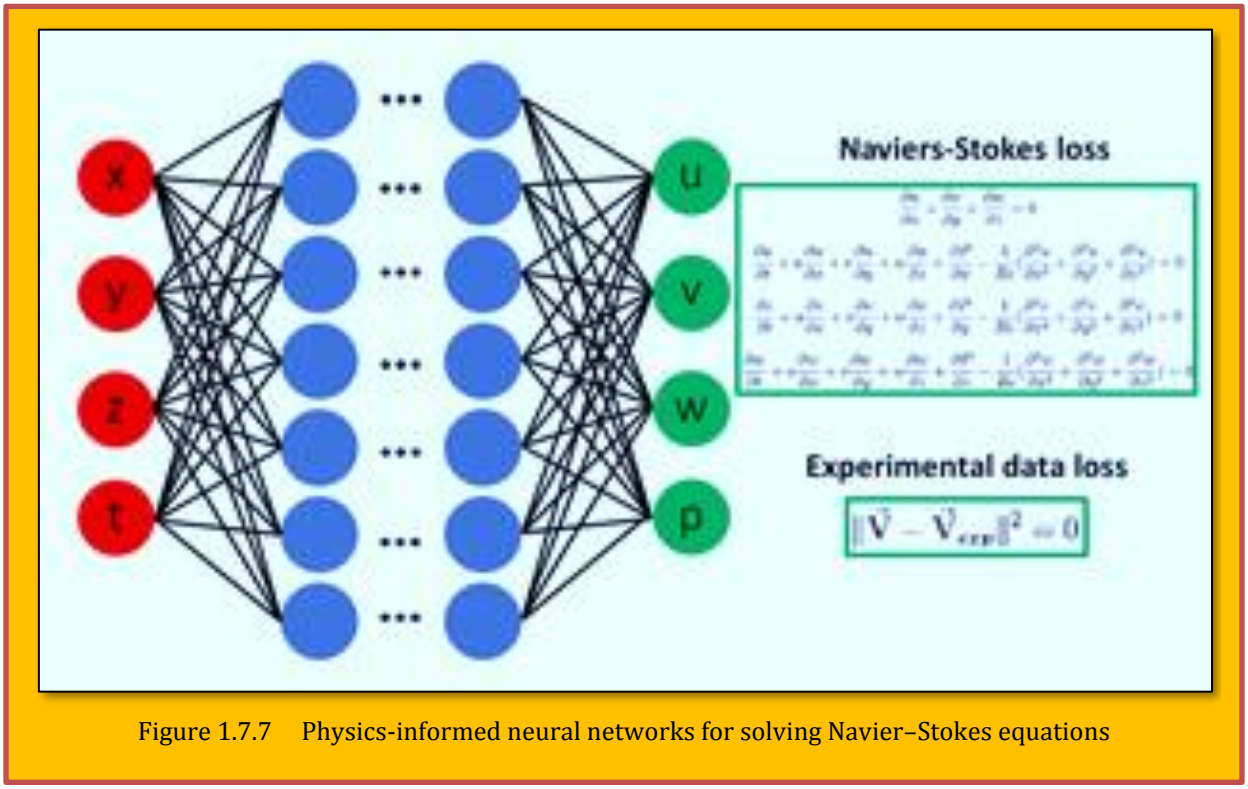


Figure 1.7.7 Physics-informed neural networks for solving Navier–Stokes equations

[17] Wikipedia

in enhancing the information content of the available data, facilitating the learning algorithm to capture the right solution and to generalize well even with a low amount of training

1.7.2 Case Study - Prediction & Comparison of the Maximal Wall Shear Stress (MWSS) for Carotid Artery Bifurcation

Steady state simulations for 1886 geometries were undertaken and MWSS values were calculated for each of them. This dataset was used for training and testing following data mining algorithms; k-nearest neighbors, linear regression, neural network: multilayer perceptron, random forest and support vector machine. The results are based on ***Relative Root Mean Square (RMSE):***

$$\mathrm{RMSE} = \frac{\sum_{i=1}^{n}(f_i - \hat{f}_i)^2}{\sum_{i=1}^{n}(f_i - \bar{f}_i)^2} \quad , \quad \begin{cases} f_i = \text{desired value (target)} \\ \hat{f}_i = \text{predicted value (predicted using dataminig algorithm)} \\ \bar{f}_i = \text{average value (average value of MWS for all 1886 samples)} \end{cases}$$

$$0 \leq \mathrm{RMSE} \leq 1$$

Eq. 1.7.1

Model	RMSE
K-Nearest Neighbors	0.760
Linear Regression	0.748
Neural Network	0.140
Random Forest	1.127
Support Vector Machin	0.612

Table 1.7.1 Results of Different Methods

Visualization of the global importance of features used for modeling MWSS. The horizontal axis of each diagram denotes the values of particular feature and the vertical axis denotes the respective average contribution value for that particular feature value. The application of the model explanation methodology results in quantitatively describing how much features and their individual values, on average, influence the target prediction values of the model. Visualization of the global importance of features used for modeling MWSS. The horizontal axis of each diagram denotes the values of particular feature and the vertical axis denotes the respective average contribution value for that particular feature value. The application of the model explanation methodology results in quantitatively describing how much features and their individual values, on average, influence the target prediction values of the model. (See **Table 1.7.1** and **Figure 1.7.8**).

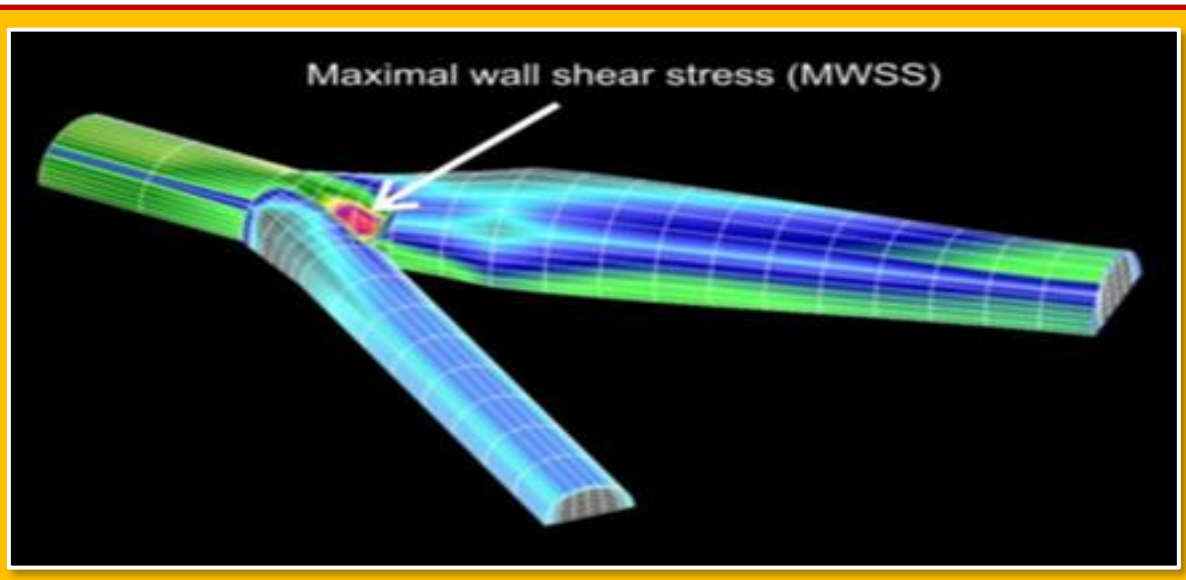


Figure 1.7.8 Maximal Wall Shear Stress (MWSS) Value for Carotid Artery Bifurcation

1.8 Machine Learning & Fluid Dynamics

According to editorial by (Brunton et al.)[8], machine learning (i.e., modern data-driven optimization and applied regression) is a rapidly growing field of research that is having a profound impact across many fields of science and engineering. In the past decade, machine learning has become a critical complement to existing experimental, computational, and theoretical aspects of fluid dynamics. In this short article, we are excited to introduce this special issue highlighting a number of promising avenues of ongoing research to integrate machine learning and data-driven techniques in the field of fluid dynamics. We will also attempt to provide a broader perspective, outlining recent successes, opportunities, and open challenges, while balancing optimism and skepticism.

Continuing from the same source; in the field of fluid dynamics, there is an interesting parallel between the rise of machine learning in recent years and the rise of computational science decades earlier. Neither approach fundamentally changes the scientific questions being asked nor the higher-level objectives. Rather, both approaches provide sophisticated tools for analysis based on emerging technologies, enabling the community to address scientific questions at a greater scale and a broader scope than was previously possible. In the early years of computational fluid dynamics, there were voices of both extreme skepticism and open-ended optimism that these new approaches would supplant existing techniques. In reality, computational techniques have provided another valuable perspective for scientific inquiry, complementing more traditional approaches. It is therefore reasonable to believe that machine learning and data-intensive analysis will have a similar impact, complementing other well-established techniques to expand our collective capabilities.

The use of machine-learning and data-science inspired approaches should be encouraged to solve problems in fluid dynamics, especially those that are difficult to solve with traditional methods. Many goals in fluid dynamics, such as analysis, modeling, sensing, estimation, design optimization, and control, may be posed as optimization problems. These problems are challenging because fluids are nonlinear and multiscale in space and time, resulting in high-dimensional and non-convex optimization landscapes. Fortunately, machine learning is improving our ability to tackle these traditionally intractable optimization problems. In addition to the critical tasks of benchmarking and validating new approaches with canonical problems, we must strive to use these emerging techniques to go beyond what is possible with existing techniques to reveal physical insights or improve analytical capabilities.

1.8.1 Motivation and Objectives

Flow control has been a fundamental concept in fluid mechanics research in this century. We develop flow modeling and optimization techniques using biologically inspired algorithms such as Artificial ***Neural Networks (ANN)*** and ***evolution strategies***. The applications presented herein encompass a variety of problems such as cylinder drag minimization, neural net modeling of the near wall structures, enhanced jet mixing, and parameter optimization in turbine blade film cooling. The unifying concept is the utilization of automated processes for the solution of these problems, devised from machine learning algorithms. The results presented herein encompass a wide variety of problems such as drag minimization, neural net modeling of the near wall structures, enhanced jet mixing, and parameter optimization in turbine blade film cooling. a challenging problem that, when solved, could lead to drastically improved designs. We envision neural network approaches as an effective way of developing such models and incorporating them in feedback control algorithms. In We present some preliminary results from the application of (ANN) as a method to construct low order models, describing the near wall dynamics in turbulent flows. Neural Networks are viewed as a general procedure of model formation encompassing schemes such as the ***Proper Orthogonal Decomposition (POD)***.

1.8.2 Accelerating the Poisson Equation

According to (Vinuesa & Brunton, 2021)[18], it is possible to accelerate CFD calculation by solving the ***Poisson equation*** by means of deep learning, as proposed by several research groups in various areas [1,2]. The Poisson equation is frequently used in operator-splitting methods to discretize the Navier–Stokes equations [3], where first the velocity field is advected, and the resulting field u* does not satisfy the continuity equation (i.e. for incompressible flows, u* does not satisfy divergence free condition). The second step involves a correction to ensure that u* is divergence free, leading to the following Poisson equation:

$$\frac{\Delta t}{\rho}\nabla^2 p = -\nabla \cdot u^*$$

Eq. 1.8.1

where Δt is the simulation time step, ρ is the fluid density, and p is the pressure. Solving this equation is typically the most computationally expensive step of the numerical solver. Therefore, devising alternative strategies to solve it more efficiently is an area of great promise. Ajuria et al. [2] proposed using a ***convolutional neural network (CNN)*** coupled with a CFD code to solve **Eq. 1.8.1** in incompressible cases and tested it in a plume configuration. Their results indicate that it is possible to outperform the traditional Jacobi solver with good accuracy at low Richardson numbers Ri; the Richardson number measures the ratio between the buoyancy and the shear in the flow. However, the accuracy degrades at higher Ri, motivating the authors to develop a hybrid CNN-CFD approach. Fully-convolutional neural networks were also used to solve the Poisson problem by decomposing the original problem into a homogeneous Poisson problem plus four inhomogeneous Laplace subproblems [92]. This decomposition resulted in errors below 10%, which motivates using this approach as a first guess in iterative algorithms, potentially reducing computational cost. These approaches may also be used to accelerate simulations of lower fidelity that rely on turbulence models, which will be discussed in the next section.

Numerical simulations can also be accelerated by decreasing the size of the computational domain needed to retain physical properties of the system. For example, artificially producing turbulent inflow conditions or far-field pressure gradients can significantly reduce the size of a computational domain. Fukami et al. [6] developed a time dependent inflow generator for wall-bounded turbulence simulations using a convolutional autoencoder with an MLP. They tested their method in a turbulent channel flow at Re_τ = 180, which is the friction Reynolds number based on channel half height and friction velocity, and they maintained turbulence for an interval long enough to obtain converged turbulence statistics. This is a promising research direction due to the fact that current inflow generation methods show limitations in terms of generality of the inflow conditions, for instance at various flow geometries and Reynolds numbers. A second approach to reduce the computational domain in external flows is to devise a strategy to set the right pressure-gradient distribution without having to simulate the far field. This was addressed by Morita et al. [7] through Bayesian optimization based on Gaussian-process regression, achieving excellent results when imposing concrete pressure-gradient conditions on a turbulent boundary layer.

1.8.3 References

[1] T. Shan, W. Tang, X. Dang, M. Li, F. Yang, S. Xu, and J. Wu. Study on a Poisson's equation solver based on deep learning technique. 2017 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS), pages 1–3, 2017.

[2] Z. Zhang, L. Zhang, Z. Sun, N. Erickson, R. From, and J. Fan. Solving Poisson's equation using deep learning in particle simulation of PN junction. 2019 Joint International Symposium on Electromagnetic Compatibility, Sapporo and Asia-Pacific International Symposium on Electromagnetic Compatibility (EMC Sapporo/APEMC), pages 305–308, 2019.

[3] R. Bridson. Fluid simulation. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[18] arXiv:2110.02085v1 [physics.flu-dyn]

[4] E. Ajuria, A. Alguacil, M. Bauerheim, A. Misdariis, B. Cuenot, and E. Benazera. Towards a hybrid computational strategy based on deep learning for incompressible flows. AIAA AVIATION Forum, June 15–19, pages 1–17, 2020.
[5] A. O¨ zbay, A. Hamzehloo, S. Laizet, P. Tzirakis, G. Rizos, and B. Schuller. Poisson CNN: Convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh. Data-Centric Engineering, 2:E6, 2021.
[6] K. Fukami, Y. Nabae, K. Kawai, and K. Fukagata. Synthetic turbulent inflow generator using machine learning. Physical Review Fluids, 4:064603, 2019.
[7] Y. Morita, S. Rezaeiravesh, N. Tabatabaei, R. Vinuesa, K. Fukagata, and P. Schlatter. Applying Bayesian optimization with Gaussian-process regression to Computational Fluid Dynamics problems. Preprint arXiv:2101.09985, 2021.
[8] Brunton, S.L., Hemati, M.S. & Taira, K. Special issue on machine learning and data-driven methods in fluid dynamics. *Theor. Comput. Fluid Dyn.* 34, 333–337 (2020). https://doi.org/10.1007/s00162-020-00542-y

1.8.4 Case Study - Applying Machine Learning to Study Fluid Mechanics

Author : *Steven L. Brunton*
Affiliation : *Department of Mechanical Engineering, University of Washington, Seattle, WA 98195, United States*
Citation : *Brunton, S.L. (2022). Applying Machine Learning to Study Fluid Mechanics. ArXiv, abs/2110.02083.*

1.8.4.1 Abstract

This paper provides a short overview of how to use machine learning to build data-driven models in fluid mechanics. The process of machine learning is broken down into five stages:

(1)formulating a problem to model,
(2) collecting and curating training data to inform the model,
(3) choosing an architecture with which to represent the model,
(4) designing a loss function to assess the performance of the model, and
(5) selecting and implementing an optimization algorithm to train the model.

At each stage, we discuss how prior physical knowledge may be embedding into the process, with specific examples from the field of fluid mechanics.

Keywords : Machine learning, fluid mechanics, physics-informed machine learning, neural networks, deep learning

1.8.4.2 Introduction

The field of fluid mechanics is rich with data and rife with problems, which is to say that it is a perfect playground for machine learning. Machine learning is the art of building models from data using optimization and regression algorithms. Many of the challenges in fluid mechanics may be posed as optimization problems, such designing a wing to maximize lift while minimizing drag at cruise velocities, estimating a flow field from limited measurements, controlling turbulence for mixing enhancement in a chemical plant or drag reduction behind a vehicle, among myriad others. These optimization tasks fit well with machine learning algorithms, which are designed to handle nonlinear and high-dimensional problems. In fact, machine learning and fluid mechanics both tend to rely on

the same assumption that there are patterns that can be exploited, even in high-dimensional systems [1]. Often, the machine learning algorithm will model some aspect of the fluid, such as the lift profile given a particular airfoil geometry, providing a surrogate that may be optimized over. Machine learning may also be used to directly solve the fluid optimization task, such as designing a machine learning model to manipulate the behavior of the fluid for some engineering objective with active control [2-4].

In either case, it is important to realize that machine learning is not an automatic or turn-key procedure for extracting models from data. Instead, it requires expert human guidance at every stage of the process, from deciding on the problem, to collecting and curating data that might inform the model, to selecting the machine learning architecture best capable of representing or modeling the data, to designing custom loss functions to quantify performance and guide the optimization, to implementing specific optimization algorithms to train the machine learning model to minimize the loss function over the data. A better name for machine learning might be "expert humans teaching machines how to learn a model to fit some data," although this is not as catchy. Particularly skilled (or lucky!) experts may design a learner or a learning framework that is capable of learning a variety of tasks, generalizing beyond the training data, and mimicking other aspects of intelligence. However, such artificial intelligence is rare, even more so than human intelligence. The majority of machine learning models are just that, models, which should fit directly into the decades old practice of model-based design, optimization, and control [5].

With its unprecedented success on many challenging problems in computer vision and natural language processing, machine learning is rapidly entering the physical sciences, and fluid mechanics is no exception. The simultaneous promise, and over-promise, of machine learning is causing many researchers to have a healthy mixture of optimism and skepticism. In both cases, there is a strong desire to understand the uses and limitations of machine learning, as well as best practices for how to incorporate it into existing research and development workflows. It is also important to realize that while it is now relatively simple to train a machine learning model for a well-defined task, it is still quite difficult to create a new model that outperforms traditional numerical algorithms and physics-based models. Incorporating partially known physics into the machine learning pipeline well tend to improve model generalization and improve interpretability and explain ability, which are key elements of modern machine learning [6,7].

1.8.4.3 Physics Informed Machine Learning for Fluid Mechanics

Applied machine learning may be separated into a few canonical steps, each of which provides an opportunity to embed prior physical knowledge: (1) choosing the problem to model or the question to answer; (2) choosing and curating the data used to train the model; (3) deciding on a machine learning architecture to best represent or model this data; (4) designing loss functions to quantify performance and to guide the learning process; and (5) implementing an optimization algorithm to train the model to minimize the loss function over the training data. See **Figure 1.8.1** for a schematic of this process on the example of reduced-order modeling. This organization of steps is only approximate, and there are considerable overlaps and tight interconnections between each stage. For example, choosing the problem to model and choosing the data to inform this model are two closely related decisions. Similarly, designing a custom loss function and implementing an optimization algorithm to minimize this loss function are tightly coupled. In most modern machine learning workflows, it is common to iteratively revisit earlier stages based on the outcome at later stages, so that the machine learning researcher is constantly asking new questions and revising the data, the architecture, the loss functions, and the optimization algorithm to improve performance. Here, we discuss these canonical stages of machine learning, investigate how to incorporate physics, and review examples in the field of fluid mechanics. This discussion is largely meant to be a high-level overview, and many more details can be found in recent reviews [5, 8–10].

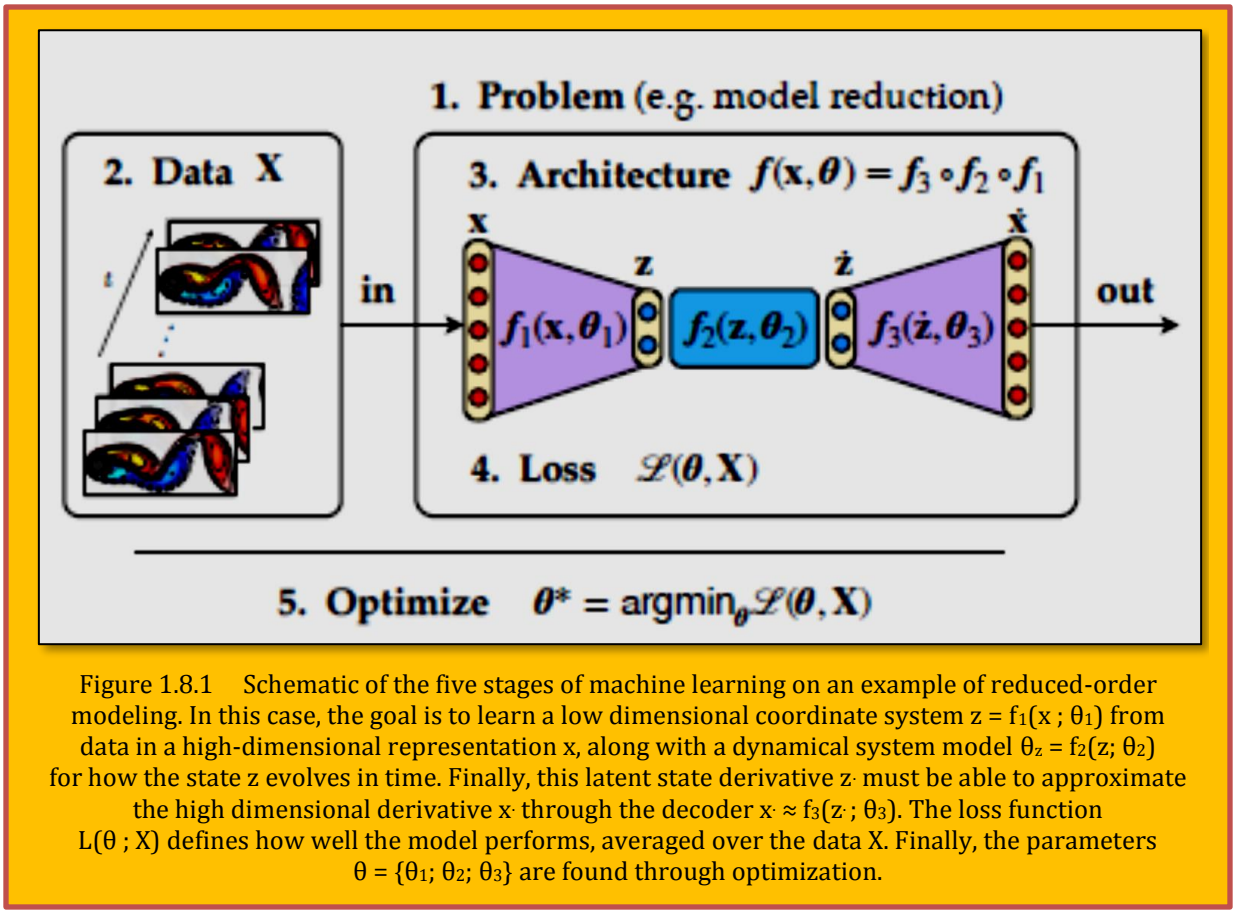


Figure 1.8.1 Schematic of the five stages of machine learning on an example of reduced-order modeling. In this case, the goal is to learn a low dimensional coordinate system $z = f_1(x ; \theta_1)$ from data in a high-dimensional representation x, along with a dynamical system model $\theta_z = f_2(z; \theta_2)$ for how the state z evolves in time. Finally, this latent state derivative $z\cdot$ must be able to approximate the high dimensional derivative $x\cdot$ through the decoder $x\cdot \approx f_3(z\cdot ; \theta_3)$. The loss function $L(\theta ; X)$ defines how well the model performs, averaged over the data X. Finally, the parameters $\theta = \{\theta_1; \theta_2; \theta_3\}$ are found through optimization.

1.8.4.3.1 The Problem

Data science is the art of asking and answering questions with data. The sub-field of machine learning is concerned with leveraging historical data to build models that may be deployed to automatically answer these questions, ideally in real-time, given new data. It is critical to select the right system to model, motivated by a problem that is both important and tractable. Choosing a problem involves deciding on input data that will be readily available in the future, and output data that will represent the desired output, or prediction, of the model. The output data should be determinable from the inputs, and the functional relationship between these is precisely what the machine learning model will be trained to capture.

The nature of the problem, specifically what outputs will be modeled given what inputs, determines the large classes of machine learning algorithms: supervised, unsupervised, and reinforcement learning. In supervised learning, the training data will have expert labels that should be predicted or modeled with the machine learning algorithm. These output labels may be discrete, such as a categorical label of a 'dog' or a 'cat' given an input image, in which case the task is one of classification. If the labels are continuous, such as the average value of lift or drag given a specified airfoil geometry, then the task is one of regression. In unsupervised learning, there are no expert labels, and structure must be extracted from the input data alone; thus, this is often referred to as data mining, and constitutes a particularly challenging field of machine learning. Again, if the structure in the data is assumed to be discrete, then the task is clustering. After the clusters are identified and characterized, these groupings may be used as proxy labels to then classify new data. If the structure in the data is assumed to be continuously varying, then the task is typically thought of as an embedding or dimensionality reduction task. Principal component analysis (PCA) or proper orthogonal decomposition (POD) may be thought of as unsupervised learning tasks that seek a continuous

embedding of reduced dimension [11]. Reinforcement learning is a third, large branch of machine learning research, in which an agent learns to make control decisions to interact with an environment for some high level objection [12]. Examples include learning how to play games [13,14], such as chess and go.

Embedding physics: Deciding on what phenomena to model with machine learning is often inherently related to the underlying physics. Although classical machine learning has been largely applied to "static" tasks, such as image classification and the placement of advertisements, increasingly it is possible to apply these techniques to model physical systems that evolve in time according to some rules or physics. For example, we may formulate a learning problem to find and represent a conserved quantity, such as a Hamiltonian, purely from data [15]. Alternatively, the machine learning task may be to model time-series data as a differential equation, with the learning algorithm representing the dynamical system [16–20]. Similarly, the task may involve learning a coordinate transformation where these dynamics become simplified in some physical way; i.e., coordinate transformations to linearize or diagonalize/decouple dynamics [21–28].

Examples in fluid mechanics: There are many physical modeling tasks in fluid mechanics that are benefiting from machine learning [5, 9]. A large field of study focuses on formulating turbulence closure modeling as a machine learning problem [8, 29], such as learning models for the Reynolds stresses [30, 31] or sub-gridscale turbulence [32, 33]. Designing useful input features is also an important way that prior physical knowledge is incorporated into turbulence closure modeling [34–36]. Similarly, machine learning has recently been focused on the problem of improving computational fluid dynamics (CFD) solvers [37–40]. Other important problems in fluid mechanics that benefit from machine learning include super-resolution [41,42], robust modal decompositions [1,43,44], network and cluster modeling [45-47], control [4, 48] and reinforcement learning [49, 50], and design of experiments in cyber physical systems [51]. Aerodynamics is a large related field with significant data-driven advances [52]. The very nature of these problems embeds the learning process into a larger physics-based framework, so that the models are more physically relevant by construction.

1.8.4.3.2 The Data

Data is the lifeblood of machine learning, and our ability to build effective models relies on what data is available or may be collected. As discussed earlier, choosing data to inform a model is closely related to choosing what to model in the first place, and therefore this stage cannot be strictly separated from the choice of a problem above. The quality and quantity of data directly affects the resulting machine learning model. Many machine learning architectures, such as deep neural networks, are essentially sophisticated interpolation engines, and so having a diversity of training data is essential to these models being useful on unseen data. For example, modern deep convolutional neural networks rose to prominence with their unprecedented classification accuracy [53] on the ImageNet data base [54], which contains over 14 million labeled images with over 20; 000 categories, providing a sufficiently large and rich set of examples for training. This pairing of a vast labeled data set with a novel deep learning architecture is widely regarded as the beginning of the modern era of deep learning [55].

Embedding physics: The training data provides several opportunities to embed prior physical knowledge. If a system is known to exhibit a symmetry, such translational or rotational invariance, then it is possible to augment and enrich the training data with shifted or rotated examples. More generally, it is often assumed that with an abundance of training data, these physical invariances will automatically be learned by a sufficiently expressive architecture. However, this approach tends to require considerable resources, both to collect and curate the data, as well as to train increasingly large models, making it more appropriate for industrial scale, rather than academic scale, research. In contrast, it is also possible to use physical intuition to craft new features from the training data, for example by applying a coordinate transformation that may simplify the representation or

training. Physical data often comes from multiple sources with different fidelity, such as from numerical simulations, laboratory experiments, and in-flight tests. This is an important area of research for flight testing and unsteady aerodynamics [52], and recently physics informed neural networks have been used with multi fidelity data to approximate PDEs [56].

Examples in fluid mechanics: Fluids data is notoriously vast and high-dimensional, with individual flow fields often requiring millions (or more!) degrees of freedom to characterize. Moreover, these flow fields typically evolve in time, resulting in a time series of multiple snapshots. Although vast in the spatial and/or temporal dimensions, data is often rather sparse in parameter space, as it is expensive to numerically or experimentally investigate multiple geometries, Reynolds numbers, etc. Thus there are many algorithms designed for both rich and sparse data. Other considerations involve exciting transients and observing how the system evolves when it is away from its natural state. In many other cases, fluids data might be quite limited, for example given by time-series data from a few pressure measurements on the surface of an airfoil, or from force recordings on an experimental turbine.

1.8.4.3.3 The Architecture

Once a problem has been identified, and data is collected and curated, it is necessary to choose an architecture with which to represent the machine learning model. Typically, a machine learning model is a function that maps inputs to outputs

$$y = f(x\,;\theta)$$

Eq. 1.8.2

and this function is generally represented within a specified family of functions parameterized by values in θ. For example, a linear regression model would model outputs as a linear function of the inputs, with θ parameterizing this linear map, or matrix. Neural networks have emerged as a particularly powerful and flexible class of models to represent functional relationships between data, and they have been shown to be able to approximate arbitrarily complex functions with sufficient data and depth [57, 58]. There is a tremendous variety of potential neural network architectures [11], limited only by the imagination of the human designer. The most common architecture is a simple feedforward network, in which data enters through an input layer and maps sequentially through a number of computational layers until an output layer. Each layer consists of nodes, where data from nodes in the previous layer are combined in a weighted sum and processed through an activation function, which is typically nonlinear. In this way, neural networks are fundamentally compositional in nature. The parameters θ determine the network weights for how data is passed from one layer to the next, i.e. the weighted connectivity matrices for how nodes are connected in adjacent layers. The overarching network topology (i.e., how many layers, how large, what type of activation functions, etc.) is specified by the architect or determined in a meta-optimization, thus determining the family of functions that may be approximated by that class of network. Then, the network weights for the specific architecture are optimized over the data to minimize a given loss function; these stages are described next.

It is important to note that not all machine learning architectures are neural networks, although they are one of the most powerful and expressive modern architectures, powered by increasingly big data and high performance computing. Before the success of deep convolutional networks on the ImageNet dataset, neural networks were not even mentioned in the list of top ten machine learning algorithms [59]. Random forests [60] and support vector machines [61] are two other leading architectures for supervised learning. Bayesian methods are also widely used, especially for dynamical systems [62]. Genetic programming has also been widely used to learn human interpretable, yet flexible representations of data for modeling [16,63-65] and control [4]. In addition, standard linear regression and generalized linear regression are still widely used for modeling time-series data, especially in fluids. The dynamic mode decomposition (DMD) [1,17,66]

employs linear regression with a low-rank constraint in the optimization to find dominant spatiotemporal coherent structures that evolve linearly in time. The *sparse identification of nonlinear dynamics (SINDy)* [18] algorithm employs generalized linear regression, with either a sparsity promoting loss function [67] or a sparse optimization algorithm [18, 68], to identify a differential equation model with as few model terms as are necessary to fit the data.

Embedding physics: Choosing a machine learning architecture with which to model the training data is one of the most intriguing opportunities to embed physical knowledge into the learning process. Among the simplest choices are convolutional networks for translationally invariant systems, and recurrent networks, such as ***long-short-time memory (LSTM)*** networks [20] or reservoir computing [19,69], for systems that evolve in time. LSTMs have recently been used to predict aeroelastic responses across a range of Mach numbers [70]. More generally, equivariant networks seek to encode various symmetries by construction, which should improve accuracy and reduce data requirements for physical systems [71–74]. Autoencoder networks enforce the physical notion that there should be low-dimensional structure, even for high-dimensional data, by imposing an information bottleneck, given by a constriction of the number of nodes in one or more layers of the network. Such networks uncover nonlinear manifolds where the data is compactly represented, generalizing the linear dimensionality reduction obtained by PCA and POD.

It is also possible to embed physics more directly into the architecture, for example by incorporating Hamiltonian [75, 76] or Lagrangian [77, 78] structure. There are numerous successful examples of ***physics-informed neural networks (PINNs)*** [79–83], which solve supervised learning problems while being constrained to satisfy a governing physical law. Graph neural networks have also shown the ability to learn generalizable physics in a range of challenging domains [64, 84,85]. Deep operator networks [86] are able to learn continuous operators, such as governing partial differential equations, from relatively limited training data.

Examples in fluid mechanics: There are numerous examples of custom neural network architectures being used to enforce physical solutions for applications in fluid mechanics. The work of Ling et al. [30] designed a custom neural network layer that enforced Galilean invariance in the Reynolds stress tensors that they were modeling. Related Reynolds stress models have been developed using the *SINDy* sparse modeling approach [87-89]. Hybrid models that combine linear system identification and nonlinear neural networks have been used to model complex aeroelastic systems [90]. The hidden fluid mechanics (HFM) approach is a physics-informed neural network strategy that encodes the Navier-Stokes equations while being flexible to the boundary conditions and geometry of the problem, enabling impressive physically quantifiable flow field estimations from limited data [91]. Sparse sensing has also been used to recover pressure distributions around airfoils [92]. The Fourier neural operator is a novel operator network that performs super-resolution upscaling and simulation modeling tasks [93]. Equivariant convolutional networks have been designed and applied to enforce symmetries in high-dimensional complex systems from fluid dynamics [73]. Physical invariances have also been incorporated into neural networks for sub grid-scale scalar flux modeling [94]. Lee and Carlberg [95] recently showed how to incorporate deep convolutional autoencoder networks into the broader reduced-order modeling framework [96–98], taking advantage of the superior dimensionality reduction capabilities of deep autoencoders.

1.8.4.3.4 The Loss Function

The loss function is how we quantify how well the model is performing, often on a variety of tasks. For example, the L_2 error between the model output and the true output, averaged over the input data, is a common term in the loss function. In addition, other terms may be added to regularize the optimization (e.g., the L_1 or L_2 norm of the parameters θ to promote parsimony and prevent overfitting). Thus, the loss function typically balances multiple competing objectives, such as model performance and model complexity. The loss function may also incorporate terms used to promote a specific behavior across different sub-networks in a neural network architecture. Importantly, the

loss function will provide valuable information used to approximate gradients required to optimize the parameters.

Embedding physics: Most of the physics-informed architectures described above involve custom loss functions to promote the efficient training of accurate models. It is also possible to incorporate physical priors, such as sparsity, by adding L_1 or L_0 regularizing loss terms on the parameters in θ. In fact, parsimony has been a central theme in physical modeling for century, where it is believed that balancing model complexity with descriptive capability is essential in developing models that generalize. The sparse identification of nonlinear dynamics algorithm [18] learns dynamical systems models with as few terms from a library of candidate terms as are needed to describe the training data. There are several formulations involving different loss terms and optimization algorithms that promote additional physical notions, such as stability [99] and energy conservation [100]. Stability promoting loss functions based on notions of Lyapunov stability have also been incorporated into autoencoders, with impressive results on fluid systems [101].

Examples in fluid mechanics: Sparse nonlinear modeling has been used extensively in fluid mechanics, adding sparsity-promoting loss terms to learn parsimonious models that prevent overfitting and generalize to new scenarios. *SINDy* has been used to generate reduced-order models for how dominant coherent structures evolve in a flow for a range of configurations [100,102-105]. These models have also been extended to develop compact closure models [87–89]. Recently, the physical notion of boundedness of solutions, which is a fundamental concept in reduced-order models of fluids [106], has been incorporated into the *SINDy* modeling framework as a novel loss function. Other physical loss functions may be added, such as adding the divergence of a flow field as a loss term to promote solutions that are incompressible [107].

1.8.4.3.5 The Optimization Algorithm

Ultimately, machine learning models are trained using optimization algorithms to find the parameters, that best fit the training data. Typically, these optimization problems are both high dimensional and non-convex, leading to extremely challenging optimization landscapes with many local minima. While there are powerful and generic techniques for convex optimization problems [108, 109], there are few generic guarantees for convergence or global optimality in nonconvex optimization. Modern deep neural networks have particularly high-dimensional parameters, and require large training data sets, which necessitate stochastic gradient descent algorithms.

In a sense, the optimization algorithm is the engine powering machine learning, and as such, it is often abstracted from the decision process. However, developing advanced optimization algorithms is the focus of intense research efforts. It is also often necessary to explicitly consider the optimization algorithm when designing a new architecture or incorporating a novel loss term.

Embedding physics: There are several ways that the optimization algorithm may be customized or modified to incorporate prior physical knowledge. One approach is to explicitly add constraints to the optimization, for example that certain coefficients must be non-negative, or that other coefficients must satisfy a specified algebraic relationship with each other. Depending on the given machine learning architecture, it may be possible to enforce energy conservation [100] or stability constraints [99] in this way. Another approach involves employing custom optimization algorithms required to minimize the physically motivated loss functions above, which are often non-convex. In this way, the line between loss function and optimization algorithm are often blurred, as they are typically tightly coupled. For example, promoting sparsity with the L_0 norm is non-convex, and several relaxed optimization formulations have been developed to approximately solve this problem. The sparse relaxed regularized regression (SR3) optimization framework [68] has been developed specifically to handle challenging non-convex loss terms that arise in physically motivated problems.

Examples in fluid mechanics: Loiseau [100] showed that it is possible to enforce energy conservation for incompressible fluid flows directly by imposing skew-symmetry constraints on the

quadratic terms of a sparse generalized linear regression (i.e. *SINDy*) model. These constraints manifest as equality constraints on the sparse coefficients θ of the *SINDy* model. Because the standard SINDy optimization procedure is based on a sequentially thresholder least-squares procedure, it is possible to enforce these equality constraints at every stage of the regression, using the *Karush–Kuhn–Tucker (KKT)* conditions. The *SR3* optimization package [68] was developed to generalize and extend these constrained optimization problems to more challenging constraints, and to more generic optimization problems. This is only one of many examples of custom optimization algorithms being developed to train machine learning models with novel loss functions or architectures.

1.8.4.4 Parting Thoughts

This brief paper has attempted to provide a high level overview of the various stages of machine learning, how physics can be incorporated at each stage, and how these techniques are being applied today in fluid mechanics. Machine learning for physical systems requires careful consideration in each of these steps, as every stage provides an opportunity to incorporate prior knowledge about the physics. A working definition of physics is the part of a model that generalizes, and this is one of the central goals of machine learning models for physical systems. It is also important to note that machine learning is fundamentally a collaborative effort, as it is nearly impossible to master every stage of this process.

The nature of this topic is mercurial, as new innovations are being introduced every day that improve our capabilities and challenge our previous assumptions. Much of this work has deliberately oversimplified the process of machine learning and the field of fluid mechanics. Machine learning is largely concerned with fitting functions from data, and so it is important to pick the right functions to fit. The inputs to the function are the variables and parameters that we have access to or control over, and the outputs are quantities of interest that we would like to accurately and efficiently approximate in the future. It is a fruitful exercise to revisit classically important problems where progress was limited by our ability to represent complex functions. For example , Ling et al. [30] had great success revisiting the classical Reynolds stress models of Pope [110] with powerful modern techniques. More fundamentally, machine learning is about asking and answering questions with data. We can't forget why we are asking these questions in the first place: because we are curious, and there is value in knowing the answer.

Disclaimer

Any omission or oversight was the result of either ignorance, forgetfulness, hastiness, or lack of imagination on my part. These notes are not meant to be exhaustive, but rather to provide a few concrete examples from the literature to guide researchers getting started in this field. This field is growing at an incredible rate, and these examples provide a tiny glimpse into a much larger effort. I have tried to sample from what I consider some of the most relevant and accessible literature. However, a disproportionate number of references are to work by my close collaborators, as this is the work I am most familiar with. If I have missed any important references or connections, or mis-characterized any works cited here, please let me know and I'll try to incorporate corrections in future versions of these notes.

Acknowledgments

SLB acknowledges many valuable discussions and perspectives gained from collaborators and coauthors Petros Koumoutsakos, J. Nathan Kutz, Jean-Christophe Loiseau, and Bernd Noack.

1.8.4.5 References

[1] Kunihiko Taira, Steven L Brunton, Scott Dawson, Clarence W Rowley, Tim Colonius, Beverley J McKeon, Oliver T Schmidt, Stanislav Gordeyev, Vassilios Theofilis, and Lawrence S Ukeiley. Modal analysis of fluid flows: An overview. AIAA Journal, 55(12):4013–4041, 2017.

[2] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse R´eglade, and Nicolas Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. Journal of fluid mechanics, 865:281–302, 2019.
[3] Feng Ren, Hai-bao Hu, and Hui Tang. Active flow control using machine learning: A brief review. Journal of Hydrodynamics, 32(2):247–253, 2020.
[4] Yu Zhou, Dewei Fan, Bingfu Zhang, Ruiying Li, and Bernd R Noack. Artificial intelligence control of a turbulent jet. Journal of Fluid Mechanics, 897, 2020.
[5] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. Annual Review of Fluid Mechanics, 52:477–508, 2020.
[6] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. Communications of the ACM, 63(1):68–77, 2019.
[7] Christoph Molnar. Interpretable machine learning. Lulu. com, 2020.
[8] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence modeling in the age of data. Annual Reviews of Fluid Mechanics, 51:357–377, 2019.
[9] MP Brenner, JD Eldredge, and JB Freund. Perspective on machine learning for advancing fluid mechanics. Physical Review Fluids, 4(10):100501, 2019.
[10] Michael P Brenner and Petros Koumoutsakos. Machine learning and physical review fluids: An editorial perspective. Physical Review Fluids, 6(7):070001, 2021.
[11] S. L. Brunton and J. N. Kutz. Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control. Cambridge University Press, 2019.
[12] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction, volume 1. MIT press Cambridge, 1998.
[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529, 2015.
[14] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. nature, 550(7676):354–359, 2017.
[15] Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Discovering conservation laws from data for control. In 2018 IEEE Conference on Decision and Control (CDC), pages 6415–6421. IEEE, 2018.
[16] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. Science, 324(5923):81–85, 2009.
[17] P. J. Schmid. Dynamic mode decomposition of numerical and experimental data. Journal of Fluid Mechanics, 656:5–28, August 2010.
[18] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proceedings of the National Academy of Sciences, 113(15):3932–3937, 2016.
[19] Jaideep Pathak, Zhixin Lu, Brian R Hunt,Michelle Girvan, and Edward Ott. Using machine learning to replicate chaotic attractors and calculate lyapunov exponents from data. Chaos: An Interdisciplinary Journal of Nonlinear Science, 27(12):121102, 2017.
[20] Pantelis R Vlachas, Wonmin Byeon, Zhong Y Wan, Themistoklis P Sapsis, and Petros Koumoutsakos. Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. Proc. R. Soc. A, 474(2213):20170844, 2018.
[21] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. Nature Communications, 9(1):4950, 2018.
[22] Christoph Wehmeyer and Frank No´e. Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. The Journal of Chemical Physics, 148(241703):1–9, 2018.
[23] Andreas Mardt, Luca Pasquali, Hao Wu, and Frank No´e. VAMPnets: Deep learning of molecular kinetics. Nature Communications, 9(5), 2018.

[24] Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. In Advances in Neural Information Processing Systems, pages 1130–1140, 2017.
[25] Qianxiao Li, Felix Dietrich, ErikMBollt, and Ioannis G Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator. Chaos: An Interdisciplinary Journal of Nonlinear Science, 27(10):103111, 2017.
[26] Enoch Yeung, Soumya Kundu, and Nathan Hodas. Learning deep neural network representations for koopman operators of nonlinear dynamical systems. arXiv preprint arXiv:1708.06850, 2017.
[27] Samuel E Otto and ClarenceWRowley. Linearly-recurrent autoencoder networks for learning dynamics. SIAM Journal on Applied Dynamical Systems, 18(1):558–593, 2019.
[28] K. Champion, B. Lusch, J. Nathan Kutz, and Steven L. Brunton. Data-driven discovery of coordinates and governing equations. Proceedings of the National Academy of Sciences, 116(45):22445–22451, 2019.
[29] Shady E Ahmed, Suraj Pawar, Omer San, Adil Rasheed, Traian Iliescu, and Bernd R Noack. On closures for reduced order models a spectrum of first-principle to machine-learned avenues. arXiv preprint arXiv:2106.14954, 2021.
[30] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. Journal of Fluid Mechanics, 2016.
[31] J Nathan Kutz. Deep learning in fluid dynamics. Journal of Fluid Mechanics, 814:1–4, 2017.
[32] Romit Maulik, Omer San, Adil Rasheed, and Prakash Vedula. Sub grid modelling for two dimensional turbulence using neural networks. Journal of Fluid Mechanics, 858:122–144, 2019.
[33] Guido Novati, Hugues Lascombes de Laroussilhe, and Petros Koumoutsakos. Automating turbulence modelling by multi-agent reinforcement learning. Nature Machine Intelligence, 2021.
[34] Jian-XunWang, Jin-LongWu, and Heng Xiao. Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on dns data. Physical Review Fluids, 2(3):034603, 2017.
[35] Linyang Zhu, Weiwei Zhang, Jiaqing Kou, and Yilang Liu. Machine learning methods for turbulence modeling in subsonic flows around airfoils. Physics of Fluids, 31(1):015105, 2019.
[36] Linyang Zhu, Weiwei Zhang, Xuxiang Sun, Yilang Liu, and Xianxu Yuan. Turbulence closure for high reynolds number airfoil flows by deep neural networks. Aerospace Science and Technology, 110:106452, 2021.
[37] Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P Brenner. Learning data-driven discretization for partial differential equations. Proceedings of the National Academy of Sciences, 116(31):15344–15349, 2019.
[38] Stephan Thaler, Ludger Paehler, and Nikolaus A Adams. Sparse identification of truncation errors. Journal of Computational Physics, 397:108851, 2019.
[39] Ben Stevens and Tim Colonius. Enhancement of shock-capturing methods via machine learning. Theoretical and Computational Fluid Dynamics, 34:483–496, 2020.
[40] Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning accelerated computational fluid dynamics. arXiv preprint arXiv:2102.01010, 2021.
[41] N Benjamin Erichson, Lionel Mathelin, Zhewei Yao, Steven L Brunton, Michael W Mahoney, and J Nathan Kutz. Shallow neural networks for fluid flow reconstruction with limited sensors. Proceedings of the Royal Society A, 476(2238):20200097, 2020.
[42] Kai Fukami, Koji Fukagata, and Kunihiko Taira. Super-resolution reconstruction of turbulent flows with machine learning. Journal of Fluid Mechanics, 870:106–120, 2019.
[43] Kunihiko Taira, Maziar S Hemati, Steven L Brunton, Yiyang Sun, Karthik Duraisamy, Shervin Bagheri, Scott Dawson, and Chi-An Yeh. Modal analysis of fluid flows: Applications and outlook. AIAA Journal, 58(3):998–1022, 2020.

[44] Isabel Scherl, Benjamin Strom, Jessica K Shang, Owen Williams, Brian L Polagye, and Steven L Brunton. Robust principal component analysis for particle image velocimetry. Physical Review Fluids, 5(054401), 2020.
[45] Aditya G Nair and Kunihiko Taira. Network-theoretic approach to sparsified discrete vortex dynamics. Journal of Fluid Mechanics, 768:549–571, 2015.
[46] E. Kaiser, B. R. Noack, L. Cordier, A. Spohn, M. Segond, M. Abel, G. Daviller, J. Osth, S. Krajnovic, and R. K. Niven. Cluster-based reduced-order modelling of a mixing layer. J. Fluid Mech. , 2014.
[47] Daniel Fernex, Bernd R Noack, and Richard Semaan. Cluster-based network modeling from snapshots to complex dynamical systems. Science Advances, 7(25):eabf5006, 2021.
[48] Guy Y Cornejo Maceda, Yiqing Li, Franc¸ois Lusseyran, Marek Morzy´ nski, and Bernd R Noack. Stabilization of the fluidic pinball with gradient-enriched machine learning control. Journal of Fluid Mechanics, 917, 2021.
[49] Dixia Fan, Liu Yang, ZhichengWang, Michael S Triantafyllou, and George Em Karniadakis. Reinforcement learning for bluff body active flow control in experiments and simulations. Proceedings of the National Academy of Sciences, 117(42):26091–26098, 2020.
[50] Siddhartha Verma, Guido Novati, and Petros Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. Proceedings of the National Academy of Sciences, 115(23):5849–5854, 2018.
[51] Dixia Fan, Gurvan Jodin, TR Consi, L Bonfiglio, Y Ma, LR Keyes, George E Karniadakis, and Michael S Triantafyllou. A robotic intelligent towing tank for learning complex fluid structure dynamics. Science Robotics, 4(36), 2019.
[52] Jiaqing Kou and Weiwei Zhang. Data-driven modeling for unsteady aerodynamics and aeroelasticity. Progress in Aerospace Sciences, 125:100725, 2021.
[53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, 2012.
[54] Jia Deng,Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. IEEE, 2009.
[55] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016.
[56] Xuhui Meng and George Em Karniadakis. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse pde problems. Journal of Computational Physics, 401:109020, 2020.
[57] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366, 1989.
[58] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. Neural networks, 4(2):251–257, 1991.
[59] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, S Yu Philip, et al. Top 10 algorithms in data mining. Knowledge and Information Systems, 14(1):1–37, 2008.
[60] Leo Breiman. Random forests. Machine learning, 45(1):5–32, 2001.
[61] Bernhard Sch¨olkopf and Alexander J Smola. Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press, 2002.
[62] Antoine Blanchard and Themistoklis Sapsis. Bayesian optimization with output-weighted optimal sampling. Journal of Computational Physics, 425:109901, 2021.
[63] Josh Bongard and Hod Lipson. Automated reverse engineering of nonlinear dynamical systems. Proceedings of the National Academy of Sciences, 104(24):9943–9948, 2007.
[64] Miles D Cranmer, Rui Xu, Peter Battaglia, and Shirley Ho. Learning symbolic physics with graph networks. arXiv preprint arXiv:1909.05862, 2019.

[65] Miles Cranmer, Alvaro Sanchez-Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. arXiv preprint arXiv:2006.11287, 2020.
[66] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems. SIAM, 2016.
[67] Robert Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), pages 267–288, 1996.
[68] Peng Zheng, Travis Askham, Steven L Brunton, J Nathan Kutz, and Aleksandr Y Aravkin. Sparse relaxed regularized regression: SR3. IEEE Access, 7(1):1404–1423, 2019.
[69] Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott. Model-free prediction of large spatiotemporally chaotic systems from data: a reservoir computing approach. Physical review letters, 120(2):024102, 2018.
[70] Kai Li, Jiaqing Kou, and Weiwei Zhang. Deep neural network for unsteady aerodynamic and aeroelastic modeling across multiple mach numbers. Nonlinear Dynamics, 96(3):2157–2177, 2019.
[71] Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. arXiv preprint arXiv:1802.08219, 2018.
[72] Benjamin Kurt Miller, Mario Geiger, Tess E Smidt, and Frank No´e. Relevance of rotationally equivariant convolutions for predicting molecular properties. arXiv preprint arXiv:2008.08461, 2020.
[73] RuiWang, RobinWalters, and Rose Yu. Incorporating symmetry into deep dynamics models for improved generalization. arXiv preprint arXiv:2002.03061, 2020.
[74] Simon Batzner, Tess E Smidt, Lixin Sun, Jonathan P Mailoa, Mordechai Kornbluth, Nicola Molinari, and Boris Kozinsky. Se (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. arXiv preprint arXiv:2101.03164, 2021.
[75] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. Advances in Neural Information Processing Systems, 32:15379–15389, 2019.
[76] Marc Finzi, Ke Alexander Wang, and Andrew Gordon Wilson. Simplifying hamiltonian and Lagrangian neural networks via explicit constraints. Advances in Neural Information Processing Systems, 33, 2020.
[77] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. arXiv preprint arXiv:2003.04630, 2020.
[78] Yaofeng Desmond Zhong and Naomi Leonard. Unsupervised learning of lagrangian dynamics from images for prediction and control. Advances in Neural Information Processing Systems, 2020.
[79] M Raissi, P Perdikaris, and GE Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378:686–707, 2019.
[80] Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. SIAM Journal on Scientific Computing, 41(4):A2603–A2626, 2019.
[81] Liu Yang, Dongkun Zhang, and George Em Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. SIAM Journal on Scientific Computing, 2020.
[82] Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. Computer Methods in Applied Mechanics and Engineering, 360:112789, 2020.
[83] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. Nature Reviews Physics, 3(6):422–440, 2021.
[84] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261, 2018.

[85] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In International Conference on Machine Learning, pages 8459–8468. PMLR, 2020.
[86] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nature Machine Intelligence, 3(3):218–229, 2021.
[87] S Beetham and J Capecelatro. Formulating turbulence closures using sparse regression with embedded form invariance. Physical Review Fluids, 5(8):084611, 2020.
[88] Sarah Beetham, Rodney O Fox, and Jesse Capecelatro. Sparse identification of multiphase turbulence closures for coupled fluid–particle flows. Journal of Fluid Mechanics, 914, 2021.
[89] Martin Schmelzer, Richard P Dwight, and Paola Cinnella. Discovery of algebraic Reynolds stress models using sparse symbolic regression. Flow, Turbulence and Combustion, 104(2):579-603, 2020.
[90] Jiaqing Kou andWeiwei Zhang. A hybrid reduced-order framework for complex aeroelastic simulations. Aerospace science and technology, 84:880–894, 2019.
[91] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Science, 367(6481):1026–1030, 2020.
[92] Xuan Zhao, Lin Du, Xuhao Peng, Zichen Deng, and Weiwei Zhang. Research on refined reconstruction method of airfoil pressure based on compressed sensing. Theoretical and Applied Mechanics Letters, page 100223, 2021.
[93] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. arXiv preprint arXiv:2010.08895, 2020.
[94] Hugo Frezat, Guillaume Balarac, Julien Le Sommer, Ronan Fablet, and Redouane Lguensat.
Physical invariance in neural networks for sub grid-scale scalar flux modeling. Physical Review Fluids, 6(2):024607, 2021.
[95] Kookjin Lee and Kevin T Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. Journal of Computational Physics, 404:108973, 2020.
[96] B. R. Noack, K. Afanasiev, M. Morzynski, G. Tadmor, and F. Thiele. A hierarchy of lowdimensional models for the transient and post-transient cylinder wake. Journal of Fluid Mechanics, 2003.
[97] Peter Benner, Serkan Gugercin, and Karen Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. SIAM review, 57(4):483–531, 2015.
[98] Clarence W Rowley and Scott TM Dawson. Model reduction for flow analysis and control. Annual Review of Fluid Mechanics, 49:387–417, 2017.
[99] Alan A Kaptanoglu, Jared L Callaham, Christopher J Hansen, Aleksandr Aravkin, and Steven L Brunton. Promoting global stability in data-driven models of quadratic nonlinear dynamics. arXiv preprint arXiv:2105.01843, 2021.
[100] J. C. Loiseau and S. L. Brunton. Constrained sparse Galerkin regression. Journal of Fluid Mechanics, 838:42–67, 2018.
[101] N Benjamin Erichson, Michael Muehlebach, and Michael W Mahoney. Physics-informed autoencoders for lyapunov-stable fluid flow prediction. arXiv preprint arXiv:1905.10866, 2019.
[102] J. C. Loiseau, B. R. Noack, and S. L. Brunton. Sparse reduced-order modeling: sensor-based dynamics to full-state estimation. Journal of Fluid Mechanics, 844:459–490, 2018.
[103] Jean-Christophe Loiseau. Data-driven modeling of the chaotic thermal convection in an annular thermosyphon. Theoretical and Computational Fluid Dynamics, 34(4):339–365, 2020.
[104] Nan Deng, Bernd R Noack, Marek Morzy´ nski, and Luc R Pastur. Low-order model for successive bifurcations of the fluidic pinball. Journal of fluid mechanics, 884, 2020.
[105] Nan Deng, Bernd R Noack, Marek Morzy´ nski, and Luc R Pastur. Galerkin force model for transient and post-transient dynamics of the fluidic pinball. Journal of Fluid Mechanics, 918, 2021.
[106] M. Schlegel and B. R. Noack. On long-term boundedness of Galerkin models. Journal of Fluid Mechanics, 765:325–352, 2015.

[107] Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1457–1466, 2020.
[108] Michael Grant, Stephen Boyd, and Yinyu Ye. Cvx: Matlab software for disciplined convex programming, 2008.
[109] Stephen Boyd and Lieven Vandenberghe. Convex optimization. Cambridge university press, 2009.
[110] SBj Pope. Amore general effective-viscosity hypothesis. Journal of Fluid Mechanics, 72(2):331

1.9 Direct Numerical Simulation (DNS) and Turbulence Issues

Direct numerical simulation (DNS) is a high-fidelity approach where the governing Navier–Stokes equations are discretized and integrated in time with enough degrees of freedom (# of Cells) to resolve all flow structures (Vinuesa & Brunton, 2021)[19]. Turbulent flows exhibit a pronounced multi-scale character, with vorticial structures across a range of sizes and energetic content. This complexity requires fine meshes and accurate computational methods to avoid distorting the underlying physics with numerical artifacts. With properly designed DNS, it is possible to obtain a representation of the flow field with the highest level of detail among CFD methods. However, the

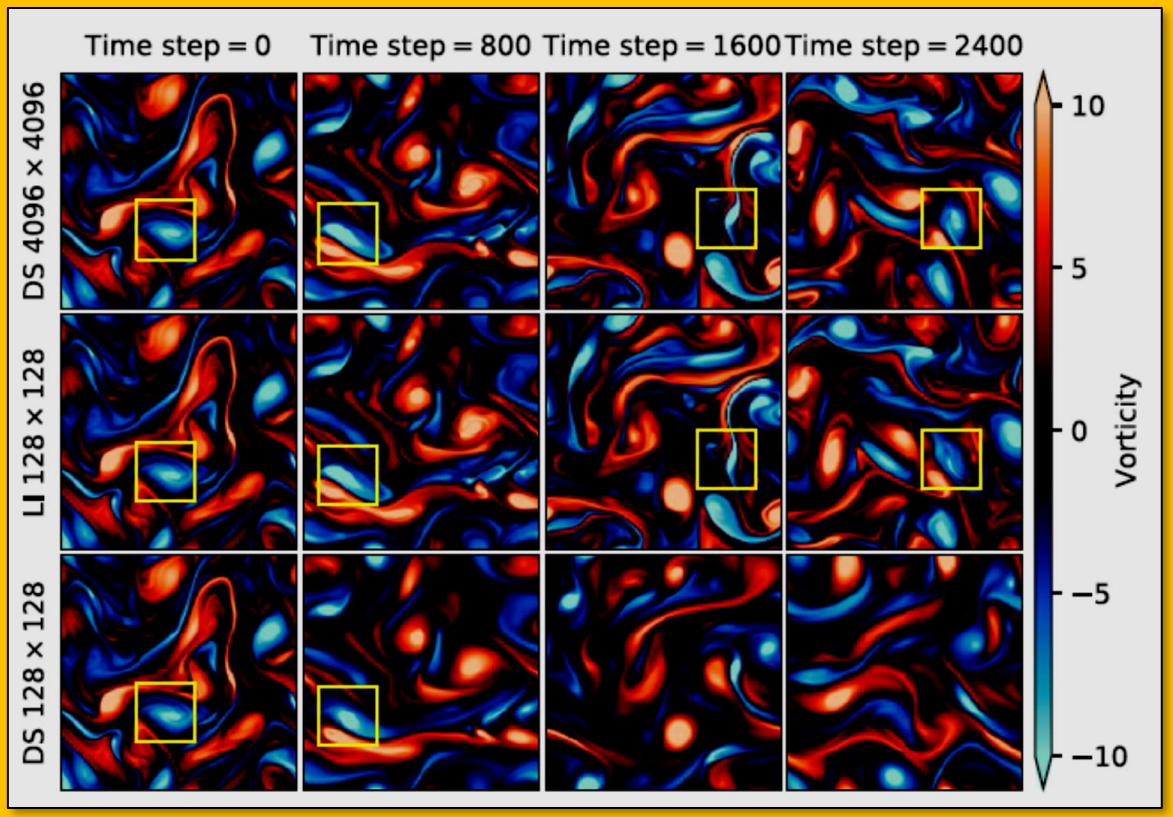


Figure 1.9.1 Sample results from the work by Kochkov et al. [56], where the instantaneous vorticity field is shown for (top) the simulation with original resolution, (middle) low-resolution data based on the ML model and (bottom) low-resolution data based on a simulation with the same coarse resolution. Four different time steps are shown, and some key vortical structures are highlighted with yellow squares. Reprinted from Ref. [56], with permission of the publisher (United States National Academy of Sciences)

[19] arXiv:2110.02085v1 [physics.flu-dyn]

fine computational meshes required to resolve the smallest scales lead to exceedingly high computational costs, which increase with the Reynolds number [1].

A number of machine learning approaches have been developed recently to improve the efficiency of DNS. Bar-Sinai et al. [2] proposed a technique based on deep learning to estimate spatial derivatives in low-resolution grids, outperforming standard finite-difference methods. A similar approach was developed by Stevens and Colonius [3] to improve the results of fifth-order finite-difference schemes in the context of shock-capturing simulations. Other strategies to improve the performance of PDE solvers in coarser meshes have been developed by Li et al. [4-6]. Recently, Kochkov et al. [7] considered the two-dimensional Kolmogorov flow [8], which maintains fluctuations via a forcing term. They leveraged deep learning to develop a correction between fine and coarse resolution simulations, obtaining excellent agreement with reference simulations in meshes from 8 to 10 times coarser in each dimension, as shown in **Figure 1.9.1.** These results promise to significantly reduce the computational cost of relevant fluid simulations, including weather [9], climate [10], engineering [11], and astrophysics [12].

Jeon and Kim [13] proposed to use a deep neural network to simulate the well-known finite-volume discretization scheme [14] employed in fluid simulations. They tested their method with reactive flows, obtaining excellent agreement with reference high-resolution data at one tenth the computational cost. However, they also documented errors with respect to the reference solution which increased with time. Another deep-learning approach, based on a fully-convolutional/long-short-term-memory (LSTM) network, was proposed by Stevens and Colonius [15] to improve the accuracy of finite-difference/finite-volume methods. Recent developments in CFD for turbomachinery, with emphasis in turbulence, which make use of machine learning techniques to augment prediction accuracy, speed up prediction times, analyze and manage uncertainty and reconcile simulations with available data [16].

1.9.1 References

[1] H. Choi and P. Moin. Grid-point requirements for large eddy simulation: Chapman's estimates revisited. Physics of Fluids, 24:011702, 2012.

[2] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning data-driven discretization's for partial differential equations. Proceedings of the National Academy of Sciences, 116(31), 2019.

[3] B. Stevens and T. Colonius. Enhancement of shock-capturing methods via machine learning. Theoretical and Computational Fluid Dynamics, 34:483–496, 2020.

[4] K. Lee and K. T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. Journal of Computational Physics, 404:108973, 2020.

[5] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. arXiv preprint arXiv:2010.08895, 2020.

[6] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Multipole 10 graph neural operator for parametric partial differential equations. arXiv preprint arXiv:2006.09535, 2020.

[7] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. Machine learning-accelerated computational fluid dynamics. Proceedings of the National Academy of Sciences, 118:e2101784118, 2021.

[8] G. J. Chandler and R. R. Kerswell. Invariant recurrent solutions embedded in a turbulent two-dimensional Kolmogorov flow. Journal of Fluid Mechanics, 722:554–595, 2013.

[9] P. Bauer, A. Thorpe, and G. Brunet. The quiet revolution of numerical weather prediction. Nature, 525:47–55, 2015.

[10] F. Schenk, M. V¨aliranta, F. Muschitiello, L. Tarasov, M. Heikkil¨a, S. Bj ¨orck, J. Brandefelt, A. V. Johansson, J. O. N¨aslund, and B. Wohlfarth. Warm summers during the Younger Dryas cold reversal. Nature Communications, 9:1634, 2018.

[11] R. Vinuesa, P. S. Negi, M. Atzori, A. Hanifi, D. S. Henningson, and P. Schlatter. Turbulent boundary layers around wing sections up to Rec = 10^6. International Journal of Heat and Fluid Flow, 2018.
[12] C. Aloy Tor´as, P. Mimica, and M. Mart´ınez-Sober. Towards detecting structures in computational astrophysics plasma simulations: using machine learning for shock front classification. In Artificial Intelligence Research and Development. Z. Falomir et al. (Eds.), 2018.
[13] J. Jeon and S. J. Kim. FVM Network to reduce computational cost of CFD simulation. Preprint arXiv:2105.03332, 2021.
[14] R. Eymard, T. Gallou¨et, and R. Herbin. Finite volume methods. Handbook of Numerical Analysis, 7:713–1018, 2000.
[15] B. Stevens and T. Colonius. Finitenet: A fully convolutional LSTM network architecture for time-dependent partial differential equations. arXiv preprint arXiv:2002.03014, 2020.
[16] Hammond, J.; Pepper, N.; Montomoli, F.; Michelassi, V. *Machine Learning Methods in CFD for Turbomachinery: A Review*. Int. J. Turbomach. Propuls. Power 2022, 7, 16.

1.10 Design and Optimization Issue

Another key issue in the effort to reduce time to market of new engineering designs is the optimization of the design parameters in an efficient manner. The design cycle usually involves multi-objective and multi-disciplinary optimization problems, requiring the iterative solution of empirical formulas, the appropriate integration of numerical simulations, and the incorporation of physical understanding of the various aspects of the problem. At the same time, the optimization cycle of the physical problem must take into consideration financial and manufacturing constraints. In flow related problems, this optimization cycle has benefited from advances in optimization theory which usually aim at tackling the most costly aspects of the optimization problem such as the solution of the Navier-Stokes equations. Powerful techniques such as the adjoint procedure have been implemented successfully in the design cycle of aircrafts. However, such optimization strategies are usually based on the efficient calculation of gradients of functions relating the quantity to be optimized to the parameters of the problem. Such gradients are not always readily available as often the optimization cycle would involve empirical formulas and cost functions that are difficult to express analytically in terms of the optimization problem. Moreover, gradient based algorithms are usually converging to local extrema. Therefore, the result strongly depends on the initial selection of parameters.
Evolution strategies (Rechenberg, I., 1973)[20] are optimization techniques that avoid the problems associated with the use of gradients as they require only the calculation of the cost function at each point in the parameter space. They operate based on natural principles of evolution such as mutation, recombination, and selection. These operations are adapted so that the algorithm automatically develops and attempts to optimize a ***model landscape*** relating the cost function to its parameters. Compared with gradient based techniques, their convergence rate is usually much lower, thus requiring large numbers of iterations that could be unrealistic for some problems of engineering interest. On the other hand, they are highly parallel algorithms that efficiently exploit today's powerful parallel computer architectures and they are more likely than gradient based algorithms to identify a global optimum. This latter aspect makes them attractive in many engineering applications where the fitness landscape cannot be assumed unimodal.

1.10.1 Accomplishments

Data methods are certainly not new in the fluids community. Computational fluid dynamics has capitalized on Machine Learning efforts with dimensionality-reduction techniques such as proper orthogonal decomposition or dynamic mode decomposition, which compute interpretable low-rank

[20] Rechenberg, I., "*Evolutions strategie: Optimierung technischer systeme nach prinzipien der biologischen evolution*". Fromann-Holzboog, Stuttgart, 1973.

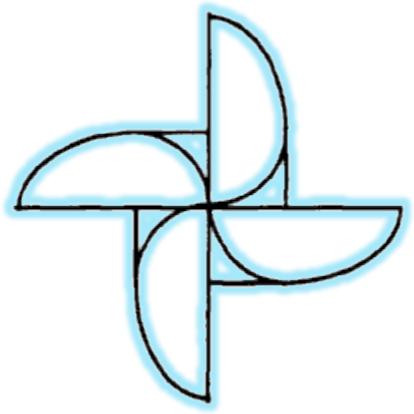
modes and subspaces that characterize spatial-temporal flow data[21]. ***Proper Orthogonal Decomposition (POD)*** and ***Dynamic Mode Decomposition (DMD)*** are based on the singular value decomposition which is ubiquitous in the dimensionality reduction of physical systems. When coupled with Galerkin projection, POD reduction forms the mathematical basis of reduced-order modelling, which provides an enabling strategy for computing high-dimensional discretization of complex flows[22]. The success of dimensionality reduction in fluids is enabled by

- Significant performance gains in computational speed and memory,
- Generation of physically interpretable spatial and/or spatial-temporal modes that dominate the physics.

Thus computations are enabled and critical physical intuition gained. Such success is tempered by two well-known failings of POD/DMD based reductions:

- Their inability to capture transient, intermittent and/or multi-scale phenomenon without significant tuning,
- Their inability to capture invariances due to translation, rotation and/or scaling.

ANNs are almost diametrically opposed in their pros and cons. Specifically, ANNs are well suited for extracting multi-scale features as the ANN decomposition shares many similarities with wavelet decompositions, which are the computational work horse of multi-resolution analysis. Moreover, translations, rotations and other invariances are known to be easily handled in the ANN architecture. These performance gains are tempered by the tremendous computational cost of building a ANN from a large training set and the inability of ANN to produce easily interpretable physical modes and/or features.



[21] Holmes, P., Lumley, J. & Berkooz, G., "Turbulence, Coherent Structures, Dynamical Systems and Symmetry", Cambridge University Press, 1998.
[22] Benner, P., Gugercin, S. & Willcox, K., " A survey of projection-based model reduction methods for parametric dynamical systems", SIAM Rev. 57, 483–531, 2015.

2 Artificial Neutral Networks (ANNs)

2.1 Field Inversion and Machine Learning in Support of Data Driven Environment

A machine learning technique such as an ***Artificial Neural Network (ANN)*** can adequately describe by its field inversion on data driven context. The Calibration Cases (offline data) where few configuration data (DNS or Experimental data) such as the one showing in **Figure 2.1.1**. The Prediction cases (Machine Learning with no data) has similar configuration with different; (1) Twist, (2) Sweep angles, and (3) Airfoil shape[23]. The challenge in predictive modeling, however, is to extract an optimal model form that is sufficiently accurate. Constructing such a model and demonstrating its predictive capabilities for a class of problems is the objective.

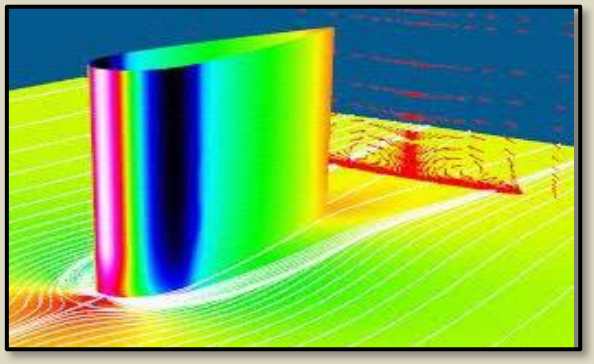


Figure 2.1.1 Calibration Cases for off Line Data

2.2 Kernel of the Artificial Neural Networks (ANNs)

The functional relationship b(η), where $\eta = [\eta_1, \eta_2, , , , \eta_M]^T$ are input features derived from mean-field variables that will be available during the predictive solution process. The functional relationship must be developed by considering the output of a number of inverse problems representative of the modeling deficiencies relevant to the predictive problem. Further, as explained below, elements of the feature vector η are chosen to be locally non-dimensional quantities. The standard NN algorithm operates by constructing linear combinations of inputs and transforming them through nonlinear activation functions (Singh, Medida, & Duraisamy, 2016)[24]. The process is

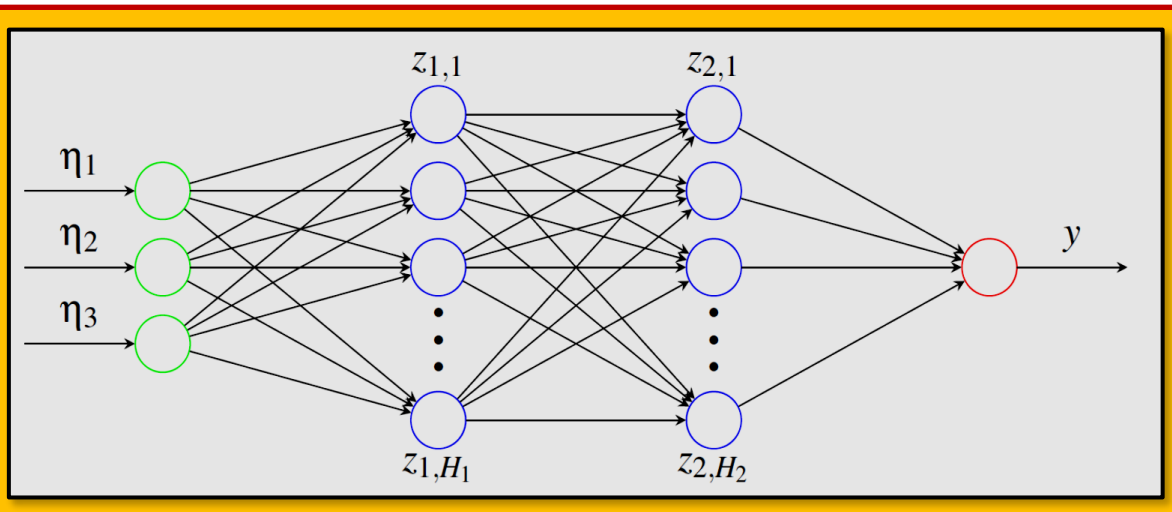


Figure 2.2.1 Network Diagram for a feed-forward NN with three inputs and one output

[23] Heng Xiao, "Physics-Informed Machine Learning for Predictive Turbulence Modeling: Status, Perspectives, and Case Studies", Machine Learning Technologies and Their Applications to Scientific and Engineering Domains Workshop, August 17, 2016.

[24] Singh, A. P., Medida, S., & Duraisamy, K. (2016). Machine Learning-augmented Predictive Modeling of Turbulent Separated. *arXiv:1608.03990v3 [cs.CE]*.

repeated once for each hidden layer (marked blue in **Figure 2.2.1**) in the network, until the output layer is reached. **Figure 2.2.1** presents a sample ANN where a Network diagram for a feed-forward NN with three inputs, two hidden layers, and one output. For this sample network, the values of the hidden nodes $z_{1,1}$ through z_1,H_1 would be constructed as

$$z_{1,i} = a^1\left(\sum_{i=1}^{3} w_{i,j}^1 \eta_i\right)$$

Eq. 2.2.1

where a^1 and $w^1{}_{i,j}$ are the activation function and weights associated with the first hidden layer, respectively. Similarly, the second layer of hidden nodes is constructed a

$$z_{2,j} = a^2\left(\sum_{i=1}^{H_1} w_{i,j}^2 z_{1,i}\right)$$

Eq. 2.2.2

Finally, the output is

$$y \approx f(\eta) = a^3\left(\sum_{i=1}^{H_2} w_{i,j}^3 z_{2,i}\right)$$

Eq. 2.2.3

Given training data, error back-propagation algorithms[25] are used to find $w^n{}_{ij}$. Once the weights are found, computing the output depends only on the number of hidden nodes, and not on the volume of the training data. Hyper-parameters of the NN method include the number of hidden layers, the number of nodes in each hidden layer, and the forms of the activation functions. Typically, 3 layers and about 100 nodes were employed with a sigmoid activation function.

2.2.1 The POD as Linear Artificial Neural Network (LANN)

A ***model reduction*** can be accomplished by projecting the model equations, i.e. the Navier-Stokes equations, on a properly selected lower dimensional phase subspace. A reasonable choice for a proper selection criterion for the base of this manifold is the maximization of the energy content of the projection[26]. This can be done by applying the ***Karhunen-Loeve*** decomposition to a data set that is representative of the dynamics of the system that we wish to approximate. This operation is called ***Proper Orthogonal Decomposition (POD)***[27]. The linear POD is an approximation of the flow vector *v* by a finite expansion of orthonormal functions φ_n such that:

$$v = V + \sum_{i=1}^{n} a_n(t)\varphi_n(x)$$

Eq. 2.2.4

---

[25] Zhang, Z. J. and Duraisamy, K., "Machine Learning Methods for Data-Driven Turbulence Modeling," 22nd AIAA Computational Fluid Dynamics Conference, AIAA Aviation, (AIAA 2015-2460), Dallas, TX, Jun 2015.

[26] S. Muller , M. Milano and P. Koumoutsakos, "*Application of machine learning algorithms to flow modeling and optimization*", Center for Turbulence Research Annual Research Briefs 1999.

[27] Berkooz, G., Holmes, P. & Lumley, J. L. "*The proper orthogonal decomposition in the analysis of turbulent flows*",*Ann. Rev. Fluid Mech.* 25, 539-575, 1993.

where V is the time averaged flow, φ_n is the set of the first n eigenvectors of the covariance matrix C = E $[(v_i - V)(v_j - V)]$; when this representation for *v* is substituted in the Navier Stokes equations, the original PDE model is transformed in an ODE model, composed by n equations. The POD can be expressed as a multi-layer feed-forward neural network. Such a network is defined by the number of layers, the specification of the output function for the neurons in each layer, and the weight matrices for each layer. [Baldi and Hornik][28] have shown that training a linear neural network structure to perform an identity mapping on a set of vectors is equivalent to obtaining the POD of this set of vectors. A neural network performing the linear POD can be specified as a 2 layer linear network:

$$x = W_1 v$$
$$\hat{v} = W_2 x$$

Eq. 2.2.5

where ^*v* is the reconstructed field, *v* is the original flow field, having N components, x is the reduced order representation of the field, having n components, and W_1 and W_2 are the network weight matrices, of sizes N x n and n x N respectively. Non-linearity can be introduced by a simple extension to this basic network:

$$x = W_2 \tanh(W_1 v)$$
$$\hat{v} = W_4 \tanh(W_3 x)$$

Eq. 2.2.6

This corresponds to a neural network model with 4 layers: the first one, with an m x N weight matrix W_1, nonlinear; the second one, with an n x m weight matrix W_2, linear; the third one, also nonlinear, with an m x n weight matrix W_3, and the last one, linear with an N x m weight matrix W_4. However, the resulting system of ODEs is more involved as compared to the one resulting from the application of the linear POD.

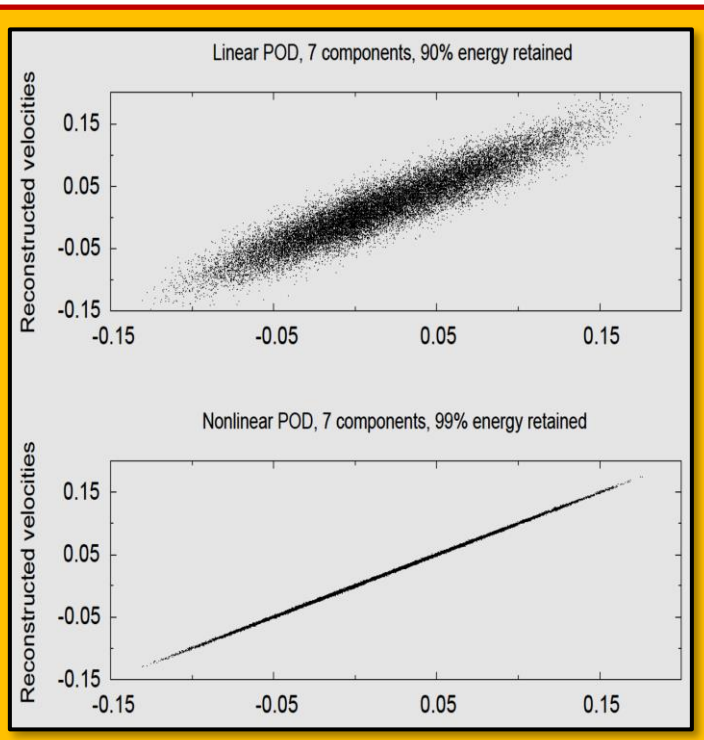


Figure 2.2.2 Comparison of linear POD (top) and Neural Networks (bottom)

2.2.1.1 POD and Nonlinear ANN

A simple comparison of POD and nonlinear ANN is provided by the reconstruction of the velocity field in the stochastically forced Burger's equation a classical 1D model for turbulent flow [Chambers][29]. The linear POD was used to obtain a set of 256 linear Eigen functions using 10000 snapshots extracted from a simulation. Using the first 7 Eigen functions it is possible to reconstruct the original flow field, keeping the 90 percent of the energy. A nonlinear neural network was trained on the same data set to perform the identity mapping: this network is composed by 256 inputs and 4 layers having respectively 64 nonlinear neurons, 7 linear

[28] Baldi, P. & Hornik, K., " *Neural networks and principal component analysis: Learning from examples without local minima*". Neural Networks. 2, 53-58, 1989.

[29] Chambers, D. H., Adrian R. J., Moin, P. & Stewart, S.*,"Karhunen-Loeve expansion of Burgers model of turbulence*". *Phys Fluids.* 31, 2573-2582, 1998.

neurons, 64 nonlinear neurons, and 256 linear neurons. For validation purposes, a data set of 1000 snapshots, not used in the training phase, was used. In **Figure 2.2.2** it is possible to appreciate the reconstruction performances of both the approaches; the proposed nonlinear ANN clearly outperforms the linear POD (top) using a velocity field in Burgers equation.

2.3 Overview of ANNs in Turbulence Applications

Turbulent flows generally exhibit multi-scale (spatial and temporal) physics that are high dimensional with rotational and translational intermittent structures also present. Such data provide an opportunity for ANN to make an impact in the modelling and analysis of turbulent flow fields. (Ling, J., Kurzawski, A. & Templeton, J., 2016)[30] have proposed using ANNs for Reynolds Averaged Navier Stokes (RANS) models which are widely used because of their computational tractability in modelling the rich set of dynamics induced by turbulent flows. In this highlighted body of work, the specific aim is to use ANNs to build an improved representation of the Reynolds stress anisotropy tensor from high-fidelity simulation data. Remarkably, despite the widespread success of ANNs at providing high-quality predictions in complex problems, there have been only limited attempts to apply deep learning techniques to turbulence. Thus far, these attempts have been limited to a couple hidden layers. **Figure 2.3.1** shows Skin Friction Coefficient for *Onera M6* wing to be matched within 2%[31].

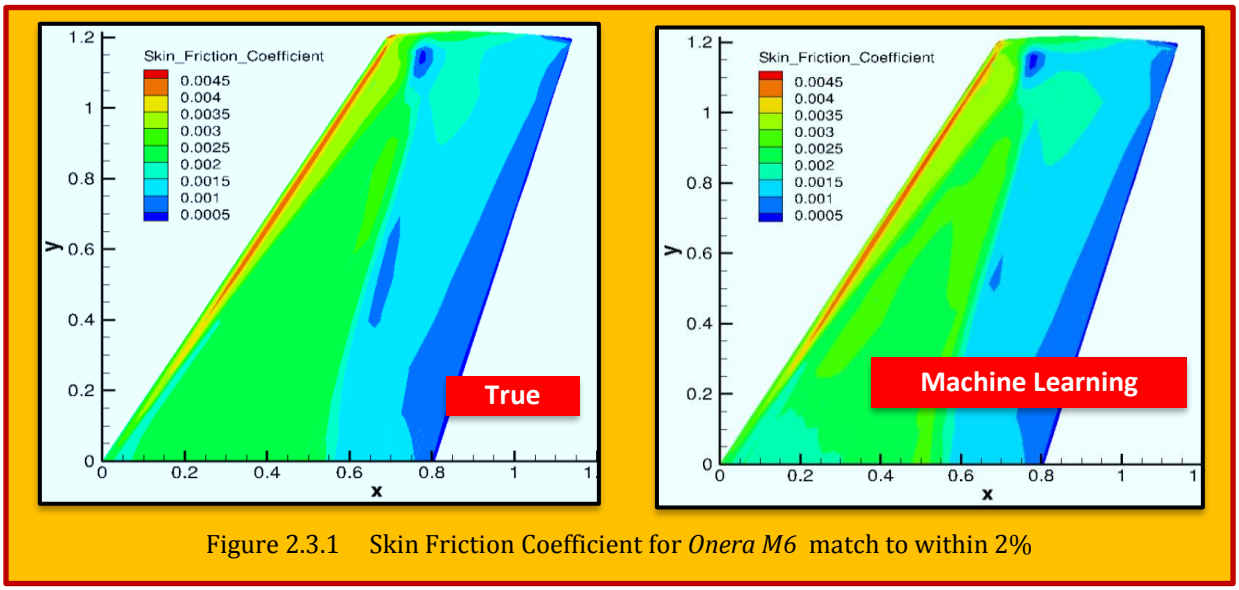


Figure 2.3.1 Skin Friction Coefficient for *Onera M6* match to within 2%

Other researchers such as (Romit Maulik et al.)[32], tried using an open source module (***TensorFlow***), within the *OpenFOAM*. It outline the development of a data science module within *OpenFOAM* which allows for the in-situ deployment of trained deep learning architectures for general-purpose predictive tasks. This is constructed with the *TensorFlow C API* and is integrated into *OpenFOAM* as an application that may be linked at run time. In this experiment, the different geometries are all backward facing steps with varying step heights (h). Once trained, the steady-state eddy-viscosity emulator may be used at the start of the simulation (by observing the initial conditions) following which solely the pressure and velocity equations need to be solved to convergence. We outline

[30] Ling, J., Kurzawski, A. & Templeton, J. "*Reynolds averaged turbulence modelling using deep neural networks with embedded invariance*", J. Fluid Mech 807, 155–166, 2016.
[31] Karthik Duraisamy, "*A Framework for Turbulence Modeling using Big Data*", NASA Aeronautics Research Mission Directorate (ARMD) LEARN/Seedling Technical Seminar January 13-15, 2015.
[32] Maulik, R., Sharma, H., Patel, S., Lusch, B., & Jennings, E. (2020). Deploying deep learning in OpenFOAM with TensorFlow. *ArXiv, abs/2012.00900*.

results from one such experiment (backward steps), where the geometry is 'unseen', in **Figure 2.3.2**.

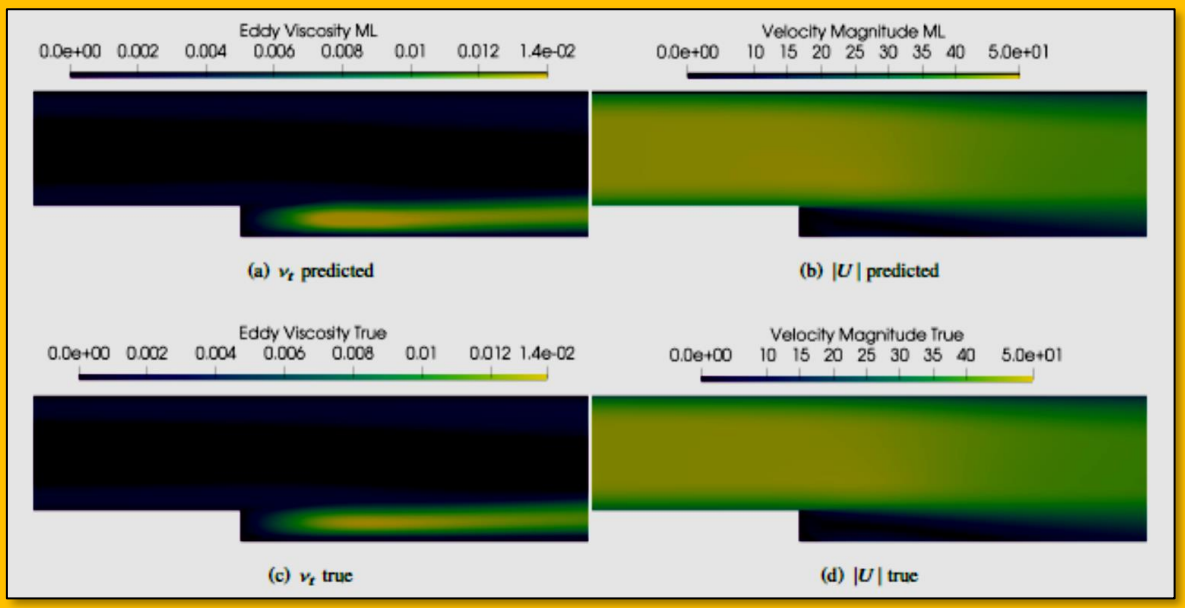


Figure 2.3.2 Contour plots for a backward facing step. Note that the training of the ML surrogate did not include data for the shown step height.

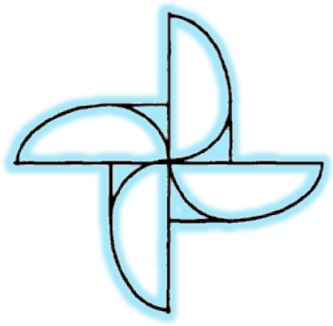
2.4 The Future of ANNs for Fluids Modelling

ANNs will almost certainly have a transformative impact on modelling high dimensional complex systems such as turbulent flows. The successes with many complex data sets will compel researchers to utilize this rapidly emerging data analysis tool for improving predictive capabilities. ANNs represent a paradigm shift for the community. Whereas many innovations have often been inspired from expert-in-the-loop intuition and physically interpretable models, ANNs have challenged these traditional notions by building prediction engines that simply outperform competing methods without providing clear evidence of why they are doing so. To some extent, the application of ANNs to turbulent flows will bring awareness to the fluids community of the two cultures of statistics and data science. These two outlooks are centered around the concepts of machine learning and statistical learning. The former focuses on prediction (ANNs) while the latter is concerned with inference of interpretable models from data (POD/DMD reductions). Although both methodologies have achieved significant success across many areas of big data analytics, the physical and engineering sciences have primarily focused on interpretable methods. Despite its successes, significant challenges remain for ANNs. Simple questions remains:

1. How many layers are necessary for a given data set?
2. How many nodes at each layer are needed?
3. How big must my data set be to properly train the network?
4. What guarantees exist that the mathematical architecture can produce a good predictor of the data?
5. What is my uncertainty and/or statistical confidence in the ANN output?
6. Can I actually predict data well outside of my training data?
7. Can I guarantee that I am not overfitting my data with such a large network?

And the list goes on. These questions remain central to addressing the long-term viability of ANNs. The good news is that such topics are currently being intensely investigated by academic researchers

and industry (Google, Facebook, etc.) alike. Undoubtedly, the next decade will witness significant progress in addressing these issues. From a practical standpoint, the work of determine the number of layers and nodes based upon prediction success, i.e. more layers and more nodes do not improve performance. Additionally, cross-validation is imperative to suppress overfitting. As a general rule, one should never trust results of a ANN unless rigorous cross-validation has been performed. Cross-validation plays the same critical role as a convergence study of a numerical scheme. Given the computational maturity of ANNs and how readily available they are, it is perhaps time for part of the turbulence modelling community to adopt what has become an important and highly successful part of the machine learning culture: challenge data sets.



3 Case Studies

3.1 Case Study 1 - 2D High-Lift Aerodynamic Optimization Using Neural Networks

Citation : Greenman, R. (1998). Two-dimensional high-lift aerodynamic optimization using neural networks.

Artificial neural networks (ANN) were successfully used to minimize the amount of data required to completely define the aerodynamics of a three-element airfoil[33]. The ability of the neural nets to accurately predict the aerodynamic coefficients (lift, drag, and moment coefficients), for any high-lift flap deflection, gap, and overlap, was demonstrated for both computational and experimental training data sets. Multiple input, single output networks were trained using the NASA Ames variation of the ***Levenberg-Marquardt algorithm*** for each of the aerodynamic coefficients. The computational data set was generated using a 2D incompressible Navier-Stokes algorithm with the *Spalart-Allmaras* turbulence model. In high-lift aerodynamics, both experimentally and computationally, it is difficult to predict the maximum lift, and at which angle of attack it occurs. In order to accurately predict the maximum lift in the computational data set, a maximum lift criteria was needed. The "pressure difference rule," which states that there exists a certain pressure difference between the peak suction pressure and the pressure at the trailing edge of the element at the maximum lift condition, was applied to all three elements. In this study it was found that only the pressure difference on the slat element was needed to predict maximum lift. The neural nets were trained with only three different values of each of the parameters stated at various angles of attack. The entire computational data set was thus sparse and yet by using only 55 - 70% of the computed data, the trained neural networks predicted the aerodynamic coefficients within an acceptable accuracy defined to be the experimental error.

A high-lift optimization study was conducted by using neural nets that are trained with computational data. Artificial neural networks have been successfully integrated with a gradient based optimizer to minimize the amount of data required to completely define the design space of a three-element airfoil. This design process successfully optimized flap deflection, gap, overlap, and angle of attack to maximize lift. Once the neural nets were trained and integrated with the optimizer, minimal additional computer resources are required to perform optimization runs with different initial conditions and parameters. Neural networks "within the process" reduced the amount of computational time and resources needed in high-lift rigging optimization.

3.1.1 Discussion and Background

Artificial neural networks are a collection (or network) of simple computational devices which are modeled after the architecture of biological nervous systems. The ability of neural networks to accurately learn and predict nonlinear multiple input and output relationships makes them a promising technique in modeling nonlinear aerodynamic data. ***CFD in conjunction with Artificial Neural Networks (ANN) and optimization, may help reduce the time and resources needed to accurately define the optimal aerodynamics of an aircraft.*** Essentially, the neural networks will reduce the amount of data required to define the aerodynamic characteristics of an aircraft while the optimizer will allow the design space to be easily searched for extreme as. **Figure 3.1.1** shows a visual depiction of the agile artificial intelligence (AI) enhanced design space capture and smart surfing process that is developed.

The design space data source is represented by black dots. In this study, computational fluid dynamics will be used to generate the data. The entire design space analyzed is shown in the figure

[33] Roxana M. Greenman, *"Two-Dimensional High-Lift Aerodynamic Optimization Using Neural Networks",* Ames Research Center, Moffett Field, California, NASA / TM- 1998-112233.

with a carpet map. The neural network will be able to capture the design space with the small amount of data that is generated. Next, the optimizer will be able to locate extreme as in the design space by using the captured design space to calculate the path that must be followed to reach a maxima. The agile artificial intelligence (AI) design space capture and surfing process is shown in **Figure 3.1.1**.

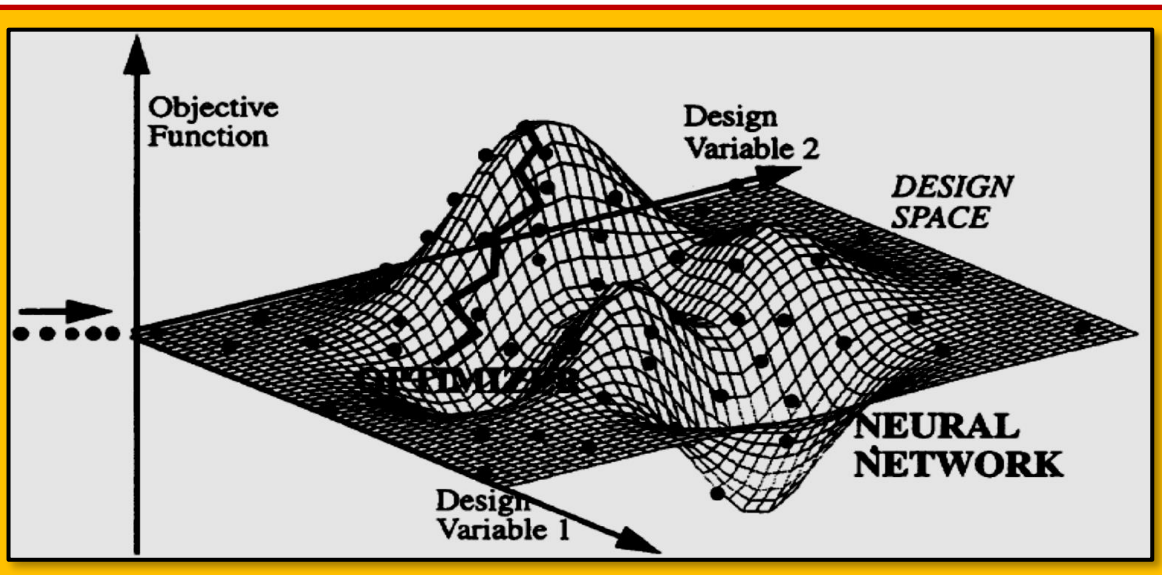


Figure 3.1.1 Agile AI-Enhanced Design Space Capture and Smart Surfing

Recently, neural networks have been applied to a wide range of problems in the aerospace industry. For example, neural networks have been used in aerodynamic performance optimization of rotor blade design[34]. The study demonstrated that for several rotor blade designs, neural networks were advantageous in reducing the time required for the optimization. [Failer and Schreck][35] successfully used neural networks to predict real-time three-dimensional unsteady separated flow fields and aerodynamic coefficients of a pitching wing. It has also been demonstrated that neural networks are capable of predicting measured data with sufficient accuracy to enable identification of instrumentation system degradation. [Steck and Rokhsaz][36] demonstrated

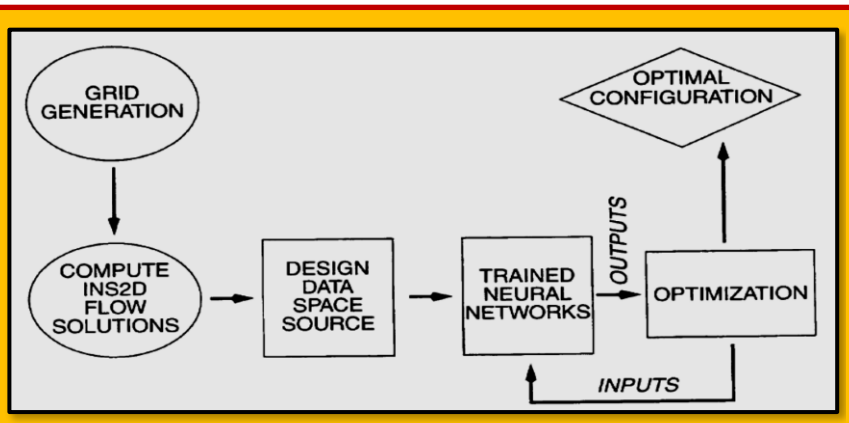


Figure 3.1.2 Illustration of AI-Enhanced Design Process

[34] LaMarsh, W. J.; Walsh, J. L.; and Rogers, J. L.:, "*Aerodynamic Performance Optimization of a Rotor Blade Using a Neural Network as the Analysis*". AIAA Paper 92-4837, Sept. 1992.
[35] Failer, W. E.; and Schreck, S. J., "*Real-Time Prediction of Unsteady Aerodynamics: Application for Aircraft Control and Maneuverability Enhancement*". IEEE Transactions on Neural Networks, vol. 6, no. 6, Nov. 1995.
[36] Steck, J. E.; and Rokhsaz, K.: "*Some Applications of **Artificial** Neural Networks in Modeling of Nonlinear Aerodynamics and Flight Dynamics*". AIAA Paper 97-0338, Jan. 1997.

that neural networks can be successfully trained to predict aerodynamic forces with sufficient accuracy or design and modeling. [Rai and Madavan][37] demonstrated the feasibility of applying neural networks to aerodynamic design of turbomachinery airfoils.

3.1.2 Agile AI-Enhanced Design Process

Here, we describes a process which allows CFD to impact high-lift design. This process has three phases:

1. generation of the training database using CFD;
2. training of the neural networks;
3. integration of the trained neural networks with an optimizer to capture and surf (search) the high-lift design space (refer to **Figure 3.1.2**).

In this reading, an incompressible 2D Navier-Stokes solver is used to compute the flow field about the three-element airfoil shown in **Figure 3.1.3**. The selected airfoil is a cross-section of the Flap-Edge model that was tested in the 7- by 10-Foot Wind Tunnel No. 1 at the NASA Ames Research Center. Extensive wind-tunnel investigations have been carried out for the Flap-Edge geometry shown in **Figure 3.1.3.** The model is a three-element un-swept wing consisting *of* a 12%c LB-546 slat, NACA 632-215 Mod B main element and a 30%c Fowler flap where c is chord and is equal to *c = 30.0* inches for the un-deflected (clean, all high-lift components stowed) airfoil section. the CFD database for this flap optimization problem, there are two different slat deflection settings, six and twenty-six degrees, and for each, 27 different flap riggings (refer to **Figure 3.1.3-b)** are computed for ten different angles *of* attack. [for details see Greenman][38]. The neural networks are trained by using the flap riggings and angles of attack as the inputs and the aerodynamic forces as the outputs. The neural networks are defined to be successfully trained to predict the aerodynamic coefficients when given a set of inputs that are not in the training set, the outputs are predicted within

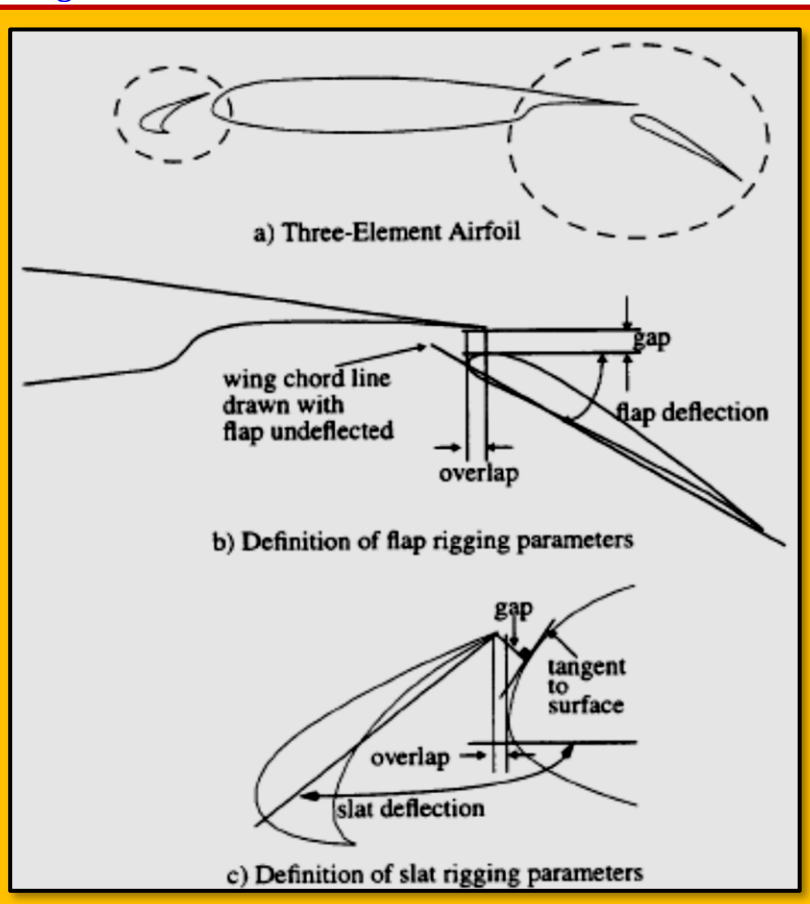


Figure 3.1.3 Edge Geometry and Definition of Flap and Slat High-Lift Rigging

[37] Rai, M. M.; and Madavan, N. K.: "*Application of Artificial Neural Networks to the Design of Turbomachinery Airfoils*". AIAA Paper 98-1003, Jan. 1998.

[38] Roxana M. Greenman, *"Two-Dimensional High-Lift Aerodynamic Optimization Using Neural Networks"*, Ames Research Center, Moffett Field, California, NASA / TM- 1998-112233.

the experimental error. Finally, the trained neural networks are integrated with the optimizer to allow the design space to be easily searched for points of interest. It will be shown that this agile, artificial intelligence enhanced design process minimizes the cost and time required to accurately optimize the high lift flap rigging.

3.1.3 Summary

Multiple input, single output networks were trained using the NASA Ames variation of the Levenberg-Marquardt algorithm. The neural networks were first trained with wind tunnel experimental data of the three-element airfoil to test the validity of the neural networks. The networks did accurately predict the lift coefficients of the individual main and flap elements. However, there was noticeable error in predicting the slat lift coefficient. The prediction error is most likely caused by the sparse training data since there were only five different used to train the neural networks**.** This results from the fact that the errors are also summed and amplify in the prediction of the total lift coefficient. Computational data is next used to train the neural networks to test if computational data can be used to train the neural networks. The neural networks were used to create a computational data base which may be used to impact design. Solutions were obtained by solving the two-dimensional Reynolds-averaged incompressible Navier-Stokes equations. The flow field was assumed to be fully turbulent and the *Spalart-Allmaras* turbulence model been used. The computational data set had to be pre-processed to reduce the prediction error at or beyond maximum lift. In high-lift aerodynamics, both experimentally and computationally, it is difficult to predict the maximum lift, and at which angle of attack it occurs. In order to predict maximum lift and the angle of attack where it occurs, a maximum lift criteria was needed. The pressure difference rule, which states that there exists a certain pressure difference between the peak suction pressure and the pressure at the trailing edge of the element at the maximum lift condition, was applied to all three elements. For this configuration, it was found that only the pressure difference on the slat element was needed to predict maximum lift. By applying the pressure difference rule, the prediction errors of the neural networks were reduced.

The amount of data that is required to train the neural networks was reduced to allow computational fluid dynamics to impact the design phase. Different subsets of the training methods were created by removing entire configurations from the six-degree-deflected slat training set. The mean and standard deviations of the root-mean-square prediction errors were calculated to compare the different methods of training. Even though the entire computational data set was sparse, it was reduced to only 70% of the entire data. It was found that the trained neural networks predicted the aerodynamic coefficients within an acceptable accuracy defined to be the experimental error. The aerodynamic data had to be represented in a nonlinear fashion so that the neural networks could learn and predict accurately. By carefully choosing the training subset, the computational data set was even further reduced to contain only 52% of the configurations.

These trained neural networks also predicted the aerodynamic coefficients within the acceptable error. Thus, the computational data required to accurately represent the flow field of a multi-element airfoil was reduced to allow computational fluid dynamics to be a usable tool for design. This same procedure was followed in the twenty-six-degree-deflected slat computational data. This data set had higher deflected flaps which were actually out of the normal flight envelope. The same trends were found except that the prediction error was much higher in this training set than the previous one. This was caused by the fact that the flow field was severely separated with the higher deflected flaps. Thus, the training data representing the flow field was noisy which leads to prediction errors. The computational design space needs to be easily searched for areas of interest such as maximums or optimal points. An optimization study to search the design space was conducted by using neural networks that were trained with computational data.

Artificial neural networks have been successfully integrated with a gradient based optimizer to minimize the amount of data required to completely define the design space of a three-element

airfoil. The accuracy of the neural networks' prediction was tested for both the initial and modified configurations by generating the grid and computing the INS2D-UP solution. The high-lift flap aerodynamics were optimized for a three-element airfoil by maximizing the lift coefficient. The design variables were flap deflection, gap, and overlap.

3.1.4 Conclusion

Overall, the neural networks were trained successfully to predict the high-lift aerodynamics of a multi-element airfoil. The neural networks were also able to predict the aerodynamics successfully when only 52-70% of the entire computational data set was used to train. The neural networks were integrated with an optimizer thus allowing a quick way to search the design space for points of interest. Optimization with neural networks reduced the turnaround time, CPU time, and cost of multiple optimization runs. Therefore, neural networks are an excellent tool to allow computational fluid dynamics to impact the design space. For a complete study of ANN in design and optimization environments, reader encourage to consult the [Rai et al.][39] and [Timnak et al.][40] beside current author.

[39] Man Mohan Rai, Nateri K. Madavan, and Frank W. Huber, "*Improving the Unsteady Aerodynamic Performance of Transonic Turbines Using Neural Networks*", NASA/TM-1999-208791.

[40] N. Timnak, A. Jahangiriana and S.A. Seyyedsalehi, "*An optimum neural network for evolutionary aerodynamic shape design*", Scientia Iranica B (2017) 24(5), 2490-2500.

3.2 Case Study 2 - Artificial Neural Networks (ANNs) Trained Through Deep Reinforcement Learning Discover Control Strategies For Active Flow Control

Citation : Rabault, J., Kuchta, M., Jensen, A., Réglade, U., & Cerardi, N. (2019). Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics, 865*, 281-302. doi:10.1017/jfm.2019.62

We present the first application of an ***Artificial Neural Network (ANNs)*** trained through a ***Deep Reinforcement Learning agent*** to perform active flow control[41]. It is shown that, in a 2D simulation of the Kármán vortex street at moderate Reynolds number (Re = 100), our Artificial Neural Network is able to learn an active control strategy from experimenting with the mass flow rates of two jets on the sides of a cylinder. By interacting with the unsteady wake, the Artificial Neural Network successfully stabilizes the vortex alley and reduces drag by about 8%. This is performed while using small mass flow rates for the actuation, on the order of 0.5% of the mass flow rate intersecting the cylinder cross section once a new pseudo-periodic shedding regime is found. This opens the way to a new class of methods for performing active flow control.

3.2.1 Introduction and Literature Survey

Drag reduction and flow control are techniques of critical interest for the industry [Brunton & Noack][42]. For example, 20% of all energy losses on modern heavy duty vehicles are due to aerodynamic drag, and drag is naturally the main source of energy losses for an airplane. Drag is also a phenomenon that penalizes animals, and Nature shows examples of drag mitigation techniques. It is for example thought that structures of the skin of fast-swimming sharks interact with the turbulent boundary layer around the animal, and reduce drag by as much as 9% [Dean & Bhushan][43]. This is therefore a proof-of-existence that flow control can be achieved with benefits, and is worth aiming for. In the past, much research has been carried towards so-called passive drag reduction methods, for example using ***Micro Vortex Generators*** for passive control of transition to turbulence. While it should be underlined that this technique is very different from the one used by sharks (preventing transition to turbulence by energizing the linear boundary layer, contra reducing the drag of a fully turbulent boundary layer), benefits in terms of reduced drag can also be achieved. Another way to obtain drag reduction is by applying an active control to the flow. A number of techniques can be used in active drag control and have been proven effective in several experiments, a typical example being to use small jets.

Interestingly, it has been shown that effective separation control can be achieved with even quite weak actuation, as long as it is used in an efficient way [Schoppa & Hussain][44]. This underlines the need to develop techniques that can effectively control a complex actuation input into a flow, in order to reduce drag. Unfortunately, designing active flow control strategies is a complex endeavor [Duriez et al][45]. Given a set of point measurements of the flow pressure or velocity around an object, there is no easy way to find a strategy to use this information in order to perform active control and reduce drag. The high dimensionality and computational cost of the solution domain (set by the complexity

---

[41] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Reglade and Nicolas Cerardi, "*Artificial Neural Networks Trained Through Deep Reinforcement Learning Discover Control Strategies For Active Flow Control*", Journal of Fluid Mechanics, April 2019.

[42] Brunton, Steven L & Noack, Bernd R 2015 *Closed-loop turbulence control: progress and challenges*. Applied Mechanics Reviews 67 (5), 050801.

[43] Dean, Brian & Bhushan, Bharat *Shark-skin surfaces for fluid-drag reduction in turbulent flow: a review. Philosophical transactions. Series A*, Mathematical, physical, and engineering sciences, 2010.

[44] Schoppa, Wade & Hussain, Fazle 1998 *A large-scale control strategy for drag reduction in turbulent boundary layers*. Physics of Fluids 10 (5), 1049–1051.

[45] Duriez, Thomas, Brunton, Steven L & Noack, Bernd R 2016 *Machine Learning Control-Taming Nonlinear Dynamics and Turbulence*. Springer.

and non-linearity inherent to Fluid Mechanics) mean that analytical solutions, and real-time predictive simulations (that would decide which control to use by simulating several control scenarios in real time) seem out of reach. Despite the considerable efforts put into the theory of flow control, and the use of a variety of analytical and semi-analytical techniques [Barbagallo et al.][46]; [Sipp & Schmid][47], bottom-up approaches based on an analysis of the flow equations face considerable difficulties when attempting to design flow control techniques.

A consequence of these challenges is the simplicity of the control strategies used in most published works about active flow control, which traditionally focus on either harmonic or constant control input [Schoppa & Hussain][48]. Therefore, there is a need to develop efficient control methods, that perform complex active control and take full advantage of actuation possibilities. Indeed, it seems that, as of today, the actuation possibilities are large, but only simplistic (and probably suboptimal) control strategies are implemented. To the knowledge of the authors, only few published examples of successful complex active control strategies are available with respect to the importance and extent of the field [Pastoor et al.][49]; [Gautier et al][50] ; [Li et al.][51] ; [Erdmann et al.][52] ; [Guéniat et al.][53]. ***In the present work, we aim at introducing for the first time Deep Neural Networks and Reinforcement Learning to the field of active flow control***. Deep Neural Networks are revolutionizing large fields of research, such as image analysis [Krizhevsky et al.][54], speech recognition, and optimal control. Those methods have surpassed previous algorithms in all these examples, including methods such as genetic programming, in terms of complexity of the tasks learned and learning speed. It has been speculated that Deep Neural Networks will bring advances also to fluid mechanics [Kutz][55], but until this day those have been limited to a few applications, such as the definition of reduced order models [Wang et al.][56], the effective control of swimmers, or performing Particle Image Velocimetry (PIV). As Deep Neural Networks, together with the Reinforcement Learning framework, have allowed recent breakthroughs in the optimal control of complex dynamic systems [Lillicrap et al.][57]; [Schulman et al.][58], it is natural to attempt to use them

---

[46] Barbagallo, Alexandre, Dergham, Gregory, Sipp, Denis, Schmid, Peter J & Robinet, Jeanchristophe 2012 *Closed-loop control of unsteadiness over a rounded backward-facing step*. Journal of Fluid Mechanics 703.

[47] Sipp, Denis & Schmid, Peter J 2016 *Linear closed-loop control of fluid instabilities and noise-induced perturbations: A review of approaches and tools*. Applied Mechanics Reviews 68 (2), 020801.

[48] Schoppa, Wade & Hussain, Fazle 1998 *A large-scale control strategy for drag reduction in turbulent boundary layers*. Physics of Fluids 10 (5), 1049–1051.

[49] Pastoor, Mark, Henning, Lars, Noack, Bernd R, King, Rudibert & Tadmor, Gilead 2008 *Feedback shear layer control for bluff body drag reduction*. Journal of fluid mechanics 608, 161–196.

[50] Gautier, Nicolas, Aider, J-L, Duriez, Thomas, Noack, Br, Segond, Marc & Abel, Markus," *Closed-loop separation control using machine learning*. Journal of Fluid Mechanics, 2015.

[51] Li, Ruiying, Noack, Bernd R., Cordier, Laurent, Borée, Jacques & Harambat, Fabien 2017 *Drag reduction of a car model by linear genetic programming contro*l. Experiments in Fluids 58 (8), 103.

[52] Erdmann, Ralf, Pätzold, Andreas, Engert, Marcus, Peltzer, Inken & Nitsche, Wolfgang 2011 *On active control of laminar–turbulent transition on two-dimensional wings.* Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 369 (1940), 1382–1395.

[53] Guéniat, Florimond, Mathelin, Lionel & Hussaini, M Yousuff 2016 *A statistical learning strategy for closed-loop control of fluid flows*. Theoretical and Computational Fluid Dynamics 30 (6), 497–510.

[54] Krizhevsky, Alex, Sutskever, Ilya & Hinton, Geoffrey E 2012 *Image net classification with deep convolutional neural networks.* In Advances in neural information processing systems, pp. 1097–1105.

[55] Kutz, J. Nathan 2017 *Deep learning in fluid dynamics. Journal of Fluid Mechanics* 814, 1–4.

[56] Wang, Z, Xiao, D, Fang, F, Govindan, R, Pain, Cc & Guo, Y 2018 *Model identification of reduced order fluid dynamics systems using deep learning.* International Journal for Numerical Methods in Fluids 86 (4), 255–268.

[57] Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David & Wierstra, Daan 2015 *Continuous control with deep reinforcement learning*. preprint arXiv:1509.02971

[58] Schulman, John, Levine, Sergey, Moritz, Philipp, Jordan, Michael I. & Abbeel, Pieter 2015 *Trust region policy optimization*. CoRR abs/1502.05477, arXiv: 1502.05477.

for optimal flow control.

Artificial Neural Networks (ANNs) are the attempt to reproduce in machines some of the features that are believed to be at the origin of the intelligent thinking of the brain (LeCun et al., 2015). The key idea consists in performing computations using a network of simple processing units, called neurons. The output value of each neuron is obtained by applying a transfer function on the weighted sum of its inputs. When performing supervised learning an algorithm, such as stochastic gradient descent, is then used for tuning the neurons weights so as to minimize a cost function on a training set. Given the success of this training algorithm, ANNs can in theory solve any problem since they are universal approximations: a large enough feed-forward neural network using a nonlinear activation function can fit arbitrarily well any function, and the Recurrent Neural Network paradigm is even turning complete. Therefore, virtually any problem or phenomenon that can be represented by a function could be a field of experimentation with ANNs.

However, the problem of designing the ANNs, and designing the algorithms that train and use them, is still the object of active research. While the case of supervised learning (i.e., when the solution is known and the ANN should simply be trained at reproducing it, such as image labeling or PIV) is now mostly solved owing to the advance of Deep Neural Networks and Deep Convolutional Networks (He et al., 2016), the case of reinforcement learning (when an agent tries to lean through the feedback of a reward function) is still the focus of much attention [Mnih et al.][59]; [Gu et al.][60]; [Schulman et al][61].

In the case of reinforcement learning, an agent (controlled by the ANN) interacts with an environment through 3 channels of exchange of information in a closed-loop fashion. First, the agent is given access at each time step to an observation o_t of the state s_t of the environment. The environment can be any stochastic process, and the observation is only a noisy, partial description of the environment. Second, the agent performs an action, at, that influences the time evolution of the environment.

Finally, the agent receives a reward rt depending on the state of the environment following the action. The reinforcement learning framework consists in finding strategies to learn from experimenting with the environment, in order to discover control sequences a(t = 1, , , , , T) that maximize the reward. The environment can be any system that provides the interface (o_t, at, r_t), either it is an Atari game emulator, a robot acting in the physical world that should perform a specific task, or a fluid mechanics system whose drag should be minimized in our case.

In the present work, we apply for the first time the ***Deep Reinforcement Learning (DRL, i.e. Reinforcement Learning performed on a deep ANN)*** paradigm to an active flow control problem. We use a ***Proximal Policy Optimization (PPO)*** method together with a Fully ***Connected Artificial Neural Network (FCANN)*** to control two synthetic jets located on the sides of a cylinder immersed in a constant flow in a 2D simulation. The geometry is chosen owing to its simplicity, the low computational cost associated with resolving a 2D unsteady wake at moderate Re number, and the simultaneous presence of the causes that make active flow control challenging (time dependence, non-linearity, high dimensionality). The PPO agent manages to control the jets and to interact with the unsteady wake to reduce the drag. We choose to release all our code as Open Source, to help trigger interest in those methods and facilitate further developments. In the following, we first present the simulation environment, before giving details about the network and reinforcement framework, and finally we offer an overview of the results obtained.

3.2.2 Simulation Environment

---

[59] Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan & Riedmiller, Martin 2013 *Playing atari with deep reinforcement learning*. arXiv preprint arXiv:1312.5602 .

[60] Gu, Shixiang, Lillicrap, Timothy, Sutskever, Ilya & Levine, Sergey 2016 *Continuous deep q-learningwith model-based acceleration*. In International Conference on Machine Learning, pp. 2829–2838.

[61] Schulman, John, Wolski, Filip, Dhariwal, Prafulla, Radford, Alec & Klimov, Oleg 2017 *Proximal policy optimization algorithms.* arXiv preprint arXiv:1707.06347 .

The PPO agent performs active flow control in a 2D simulation environment. In the following, all quantities are considered non-dimensionalized. The geometry of the simulation, adapted from the 2D test case of well-known benchmarks [Schäfer et al][62], consists of a cylinder of non-dimensional diameter D = 1 immersed in a box of total non-dimensional length L = 22 (along the X-axis) and height H = 4.1 (along Y-axis). The origin of the coordinate system is in the center of the cylinder. Similarly to the benchmark of the cylinder is slightly off the centerline of the domain (a shift of 0.05 in the Y direction is used), in order to help trigger the vortex shedding. The inflow profile (on the left wall of the domain) is parabolic, following the formula (cf. 2D-2 test case in [Schäfer et al.][63]):

$$U(y) = \frac{6\left[\left(\frac{H}{2}\right)^2 - y^2\right]}{H^2}$$

Eq. 3.2.1

where (U(y), V(y) = 0) is the non-dimensionalized velocity vector. Using this velocity profile, the mean velocity magnitude is $\bar{U} = 2U(0)/3 = 1$. A no-slip boundary condition is imposed on the top and

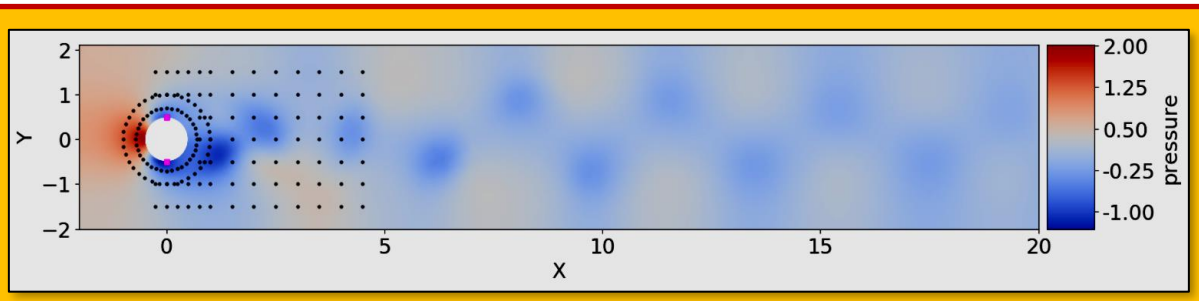


Figure 3.2.1 Unsteady Non-Dimensional Pressure Wake Behind the Cylinder after Flow Initialization without Active Control. The Location of the Velocity Probes is Indicated by the Black Dots While The Location of the Control Jets is Indicated by the Red Dot

bottom walls and on the solid walls of the cylinder. An outflow boundary condition is imposed on the right wall of the domain. The configuration of the simulation is visible in **Figure 3.2.1.**

The Reynolds number based on the mean velocity magnitude and cylinder diameter ($Re = UD/\nu$, with ν the kinematic viscosity) is set to Re = 100. Computations are performed on an unstructured mesh generated with *Gmsh*. The mesh is refined around the cylinder and is composed of 9262 triangular elements. A non-dimensional, constant numerical time step $dt = 5.10^{-3}$ is used. The total instantaneous drag on the cylinder C is computed following:

$$F_D = \int_C (\sigma . n)\ e_x\, dS$$

Eq. 3.2.2

where _ is the Cauchy stress tensor, n is the unit vector normal to the outer cylinder surface, and e_x = (1. 0). In the following, the drag is normalized into the drag coefficient:

[62] Schäfer, M., Turek, S., Durst, F., Krause, E. & Rannacher, R. 1996 *Benchmark Computations of Laminar Flow Around a Cylinder,* pp. 547–566. Wiesbaden: Vieweg+Teubner Verlag.
[63] See Previous.

$$C_D = \frac{F_D}{(1/2)\rho \bar{U}^2 D}$$

Eq. 3.2.3

where $\rho = 1$ is the non-dimensional volumetric mass density of the fluid. Similarly, the lift force F_L and

$$F_L = \int_C (\sigma . n)\ e_y\ dS$$

Eq. 3.2.4

lift coefficient C_L are defined as

$$C_L = \frac{F_L}{(1/2)\rho \bar{U}^2 D}$$

Eq. 3.2.5

where $e_y = (0, 1)$. In the interest of short solution time, the governing Navier-Stokes equations are solved in a segregated manner. More precisely, the Incremental Pressure Correction Scheme (IPCS method with an explicit treatment of the nonlinear term is used. More details are available in ([Rabault][64], Appendix B). Spatial discretization then relies on the finite element method implemented within the FEniCS framework. We remark that both the mesh density and the Reynolds number could easily be increased in a later study, but are kept low here as it allows for fast training on a laptop which is the primary aim of our proof-of-concept demonstration.

In addition, two jets (1 and 2) normal to the cylinder wall are implemented on the sides of the cylinder, at angles $\theta_1 = 90$ and $\theta_2 = 270$ degrees relatively to the flow direction. The jets are controlled through their non-dimensional mass flow rates, Q_i, $i = 1; 2$, and are set through a parabolic-like velocity profile going to zero at the edges of the jet, (see [Rabault][65], Appendix B) for the details. The jet widths are set to 10 degree. Choosing jets normal to the cylinder wall, located at the top and bottom extremities of the cylinder, means that all drag reduction observed will be the result of indirect flow control, rather than direct injection of momentum. In addition, the control is set up in such a way that the total mass flow rate injected by the jets is zero, i.e. $Q_1 + Q_2 = 0$. This synthetic jets condition is chosen as it is more realistic than a case when mass is added or subtracted from the flow, and makes the numerical scheme more stable specially with respects to the boundary conditions of the problem. In addition, it makes sure that the drag reduction observed is the result of actual flow control, rather than some sort of propulsion phenomenon. In the following, the injected mass flow rates are normalized following:

$$Q_i^* = \frac{Q_i}{Q_{ref}} \quad ; \quad Q_{ref} = \int_{-D2}^{D2} \rho U(y) dy$$

Eq. 3.2.6

where Q_{ref} is the reference mass flow rate intercepting the cylinder. During learning, we impose that $|Q| < 0.06$. This helps in the learning process by preventing non-physically large actuation, and prevents problems in the numeric of the simulation by enforcing the CFL condition close to the actuation jets.

---

[64] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Reglade and Nicolas Cerardi, "*Artificial Neural Networks Trained Through Deep Reinforcement Learning Discover Control Strategies For Active Flow Control*", Journal of Fluid Mechanics, April 2019.

[65] See Previous.

Finally, information is extracted from the simulation and provided to the PPO agent. A total of 151 velocity probes, which report the local value of the horizontal and vertical components of the velocity field, are located in several locations in the neighborhood of the cylinder and in its wake (see **Figure 3.2.1**). This means that the network gets detailed information about the flow configuration, which is our objective as this article focuses on finding the best possible control strategy of the vortex shedding pattern. A different question would be to assess the ability of the network to perform control with a partial observation of the system. To illustrate that this is possible with adequate training, we provide some results with an input layer reduced to 11 and 5 probes in (see [Rabault][66], Appendix E) but further parameter space study and sensitivity analysis is out of the scope of the present paper and is let to future work.

An unsteady wake develops behind the cylinder, which is in good agreement with what is expected at this Reynolds number. A simple benchmark of the simulation was performed by observing the pressure fluctuations, drag coefficient and ***Strouhal number*** $St = f D/\bar{U}$, where f is the vortex shedding frequency. The mean value of C_D in the case without actuation (around 3.205) is within 1% of what is reported in the benchmark of [Schäfer et al.][67], which validates our simulations, and similar agreement is found for St (typical value of around 0.30) . In addition, we also performed tests on refined meshes, going up to around 30000 triangular elements, and found that the mean drag varied by less than 1 % following mesh refinement. A pressure field snapshot of the fully developed unsteady wake is presented in **Figure 3.2.1**.

3.2.3 Network and Reinforcement Learning Framework

As stated in the introduction, Deep Reinforcement Learning (DRL) sees the fluid mechanic simulation as yet-another environment to interact with through 3 simple channels: the observation o_t (here, an array of point measurements of velocity obtained from the simulation), the action at (here, the active control of the jets, imposed on the simulation by the learning agent), and the reward r_t (here, the time-averaged drag coefficient provided by the environment, penalized by the mean lift coefficient magnitude; see further in this section). Based on this limited information, DRL trains an ANN to find closed-loop control strategies deciding at from ot at each time step, so as to maximize r_t. Our DRL agent uses the Proximal Policy Optimization (PPO, Schulman et al.][68]) method for performing learning.

PPO is a reinforcement learning algorithm that belongs to the family of policy gradient methods. This method was chosen for several reasons. In particular, it is less complex mathematically and faster than concurring Trust Region Policy Optimization methods (TRPO), and requires little to no meta parameter tuning. It is also better adapted to continuous control problems than Deep Q Learning (DQN) and its variations. From the point of view of the fluid mechanist, the PPO agent acts as a black box (though details about its internals are available in and the referred literature). A brief introduction to the PPO method is provided in ([Rabault][69], Appendix C).

The PPO method is episode-based, which means that it learns from performing active control for a limited amount of time before analyzing the results obtained and resuming learning with a new episode. In our case, the simulation is first performed with no active control until a well-developed unsteady wake is obtained, and the corresponding state is saved and used as a start for each subsequent learning episode. The instantaneous reward function, rt, is computed following:

---

[66] See Previous.

[67] Schäfer, M., Turek, S., Durst, F., Krause, E. & Rannacher, R. 1996 *Benchmark Computations of Laminar Flow Around a Cylinder*, pp. 547–566. Wiesbaden: Vieweg+Teubner Verlag.

[68] Schulman, John, Wolski, Filip, Dhariwal, Prafulla, Radford, Alec & Klimov, Oleg 2017 *Proximal policy optimization algorithms.* arXiv preprint arXiv:1707.06347 .

[69] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Reglade and Nicolas Cerardi, "*Artificial Neural Networks Trained Through Deep Reinforcement Learning Discover Control Strategies For Active Flow Control*", Journal of Fluid Mechanics, April 2019.

$$r_t = -\langle C_D \rangle_T - 0.2|\langle C_L \rangle_T|$$

Eq. 3.2.7

where $< . >_T$ indicates the sliding average back in time over a duration corresponding to one vortex shedding cycle. The ANN tries to maximize this function r_t, i.e. to make it as little negative as possible therefore minimizing drag and mean lift (to take into account long-term dynamics, an actualized reward is actually used during gradient descent; (see the Appendix C in [Rabault][70], for more details). This specific reward function has several advantages compared with using the plain instantaneous drag coefficient. Firstly, using values averaged over one vortex shedding cycle leads to less variability in the value of the reward function, which was found to improve learning speed and stability. Secondly, the use of a penalization term based on the lift coefficient is necessary to prevent the network from 'cheating'. Indeed, in the absence of this penalization, the ANN manages to find a way to modify the configuration of the flow in such a way that a larger drag reduction is obtained (up to around 18 % drag reduction, depending on the simulation configuration used), but at the cost of a large induced lift which is damageable in most practical applications.

The ANN used is relatively simple, being composed of two dense layers of 512 fully connected neurons, plus the layers required to acquire data from the probes, and generate data for the 2 jets. This network configuration was found empirically through trial and error, as is usually done with ANNs. Results obtained with smaller networks are less good, as their modeling ability is not sufficient in regards to the complexity of the flow configuration obtained. Larger networks are also less successful, as they are harder to train. In total, our network has slightly over 300000 weights. For more details, readers are referred to the code implementation (see the [Rabault][71] Appendix A).

At first, no learning could be obtained from the PPO agent interacting with the simulation environment. The reason for this was the difficulty for the PPO agent to learn the necessity to set time-correlated, continuous control signals, as the PPO first tries purely random control and must observe some improvement on the reward function for performing learning. Therefore, we implemented two tricks to help the PPO agent learn control strategies:

- The control value provided by the network is kept constant for a duration of 50 numerical time steps, corresponding to around 7.5 % of the vortex shedding period. This means, in practice, that the PPO agent is allowed to interact with the simulation and update its control only each 50 time steps.
- The control is made continuous in time to avoid jumps in the pressure and velocity due to the use of an incompressible solver. For this, the control at each time step in the simulation is obtained for each jet as $c_{s+1} = c_s + \alpha(a - c_s)$, where c_s is the control of the jet considered at the previous numerical time step, c_s+1 is the new control, a is the action set by the PPO agent for the current 50 time steps, and $\alpha = 0.1$ is a numerical parameter.

Using those technical tricks, and choosing an episode duration $T_{max} = 20.0$ (which spans around 6.5 vortex shedding periods, and corresponds to 4000 numerical time steps, i.e. 80 actions by the network), the PPO agent is able to learn a control strategy after typically about 200 epochs corresponding to 1300 vortex shedding periods or 16000 sampled actions, which requires around 24 hours of training on a modern desktop using one single core. This training time could be reduced easily by at least a factor of 10, using more cores to parallelize the data sampling from the epochs which is a fully parallel process. Fine tuning the policy can take a bit longer time, and up to around 350 epochs can be necessary to obtain a fully stabilized control strategy. A training has also been performed going up to over 1000 episodes to confirm that no more changes were obtained if the

---

[70] See Previous.

[71] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Reglade and Nicolas Cerardi, "*Artificial Neural Networks Trained Through Deep Reinforcement Learning Discover Control Strategies For Active Flow Control*", Journal of Fluid Mechanics, April 2019.

network is let to train for a significantly longer time. Most of the computation time is spent in the flow simulation. This setup with simple, quick simulations makes experimentation and reproduction of our results easy, while being enough for a proof-of-concept in the context of a first application of Reinforcement Learning to active flow control and providing an interesting control strategy for further analysis.

The drag values reported are obtained at each training epoch (including exploration noise), for 10 different trainings using the same meta parameters, but different values of the random seed. Robust learning takes place within 200 epochs, with fine converged strategy requiring a few more epochs to stabilize. The drag reduction is slightly less than what is reported in the rest of the text, as these results include the random exploration noise and are computed over the second half of the training epochs, where some of the transient in the drag value is still present during training.

3.2.4 Results for Drag Reduction Through Active Flow Control

Robust learning is obtained by applying the methodology presented in the previous section. This is illustrated by **Figure 3.2.2** [72], which presents the averaged learning curve and the confidence interval corresponding to 10 different trainings performed using different seeds for the random number generator. In this figure, the drag presented is obtained by averaging the drag coefficient obtained on the second half of each training epoch. This averaging is performed to smooth the effect of both vortex shedding, and drag fluctuations due to the exploration. While it may include part of the initial transition from the undisturbed vortex shedding to the controlled case, it is a good relative indicator of policy convergence. Estimating at each epoch the asymptotic quality of the fully established control regime would be too expansive, which is the reason why we resort to this averaged value. Using different random seeds results in different trainings, as random data are used in the exploration noise and for the random sampling of the replay memory used during stochastic gradient descent. All other parameters are kept constant. The data presented indicate that learning takes place consistently in around 200 epochs, with fine convergence and tuning requiring up to around 400 epochs.

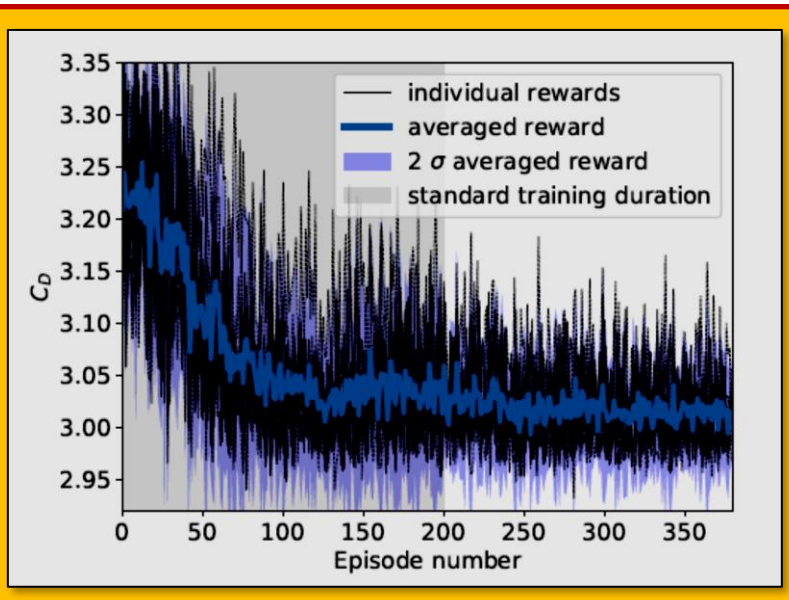


Figure 3.2.2 Illustration of the Robustness of the Learning Process

[72] Illustration of the robustness of the learning process. The drag values reported are obtained at each training
epoch (including exploration noise), for 10 different trainings using the same meta parameters, but different values of the random seed. Robust learning takes place within 200 epochs, with fine converged strategy requiring a few more epochs to stabilize. The drag reduction is slightly less than what is reported in the rest of the text, as these results include the random exploration noise and are computed over the second half of the training epochs, where some of the transient in the drag value is still present during training.

Due to the presence of exploration noise and the averaging being performed on a time window including some of the transition in the flow configuration from free shedding to active control, the quality of the drag reduction reported in this figure is slightly less than in the case of deterministic control in the pseudo periodic actively controlled regime (i.e. when a modified stable vortex shedding is obtained with the most likely action of the optimal policy being picked up at each time step implying that, in the case of deterministic control, no exploration noise is present), which is as expected. The final drag reduction value obtained in the deterministic mode (not shown to not overload the figure) is also consistent across the runs.

Therefore, it is clear that the ANN is able to consistently reduce drag by applying active flow control following training through the DRL/PPO algorithm, and that the learning is both stable and robust. All results presented further in both this section and the next one are obtained using deterministic prediction, and therefore exploration noise is not present in the following figures and results. The time series for the drag coefficient obtained using the active flow control strategy discovered through training in the first run, compared with the baseline simulation (no active control, i.e. $Q_1 = Q_2 = 0$), is presented in **Figure 3.2.3** together with the corresponding control signal (inset). Similar results and control Laws are obtained for all training runs, and the results presented in **Figure 3.2.3** are therefore representative of the learning obtained with all 10 realizations. In the case without actuation (baseline), the drag coefficient C_D varies periodically at twice the vortex shedding frequency, as should be expected. The mean value for the drag coefficient is $< C_D > \approx 3.205$, and the amplitude of the fluctuations of the drag coefficient is around 0.034. By contrast, the mean value for the drag coefficient in the case with active flow control is around $< C'_D > \approx 2.95$, which represents a drag reduction of around 8%.

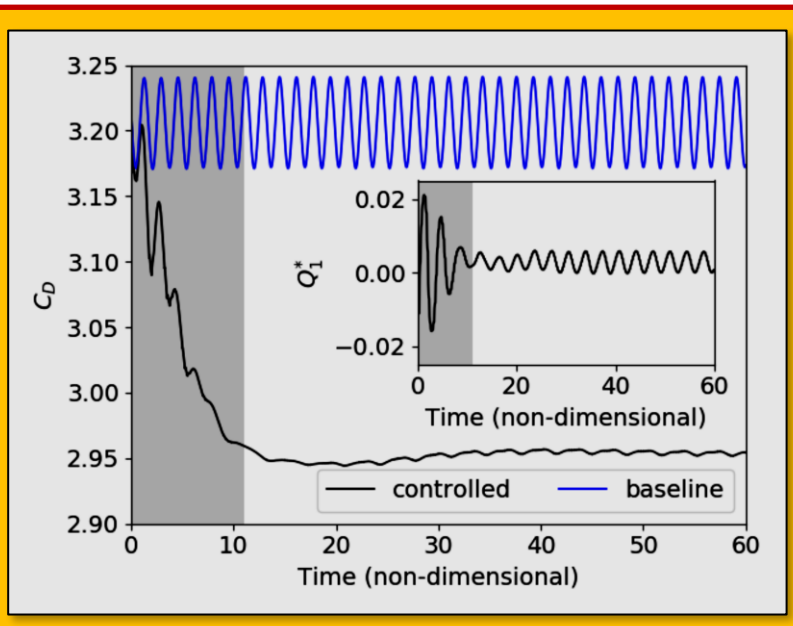


Figure 3.2.3 Time-Resolved Value of the Drag Coefficient C_D in the case without (baseline curve) and with (controlled curve) Active Flow Control, and Corresponding Normalized Mass Flow Rate of the Control Jet 1 (Q^*_1 inset)

To put this drag reduction into perspective, we estimate the drag obtained in the hypothetical case where no vortex shedding is present. For this, we perform a simulation with the upper half domain and a symmetric boundary condition on the lower boundary (which cuts the cylinder through its equator). More details about this simulation are presented in Appendix D of [Rabault][73]. The steady-state drag obtained on a full cylinder in the case without vortex shedding is then CDs = 2.93 (see Appendix D) of [Rabault][74], which means that the active control is able to suppress around 93% of the drag increase observed in the baseline without control compared with the hypothetical reference case where the flow would be kept completely stable.

[73] See Previous.
[74] See Previous.

In addition to this reduction in drag, the fluctuations of the drag coefficient are reduced to around 0:0016 by the active control, i.e. a factor of around 20 compared with the baseline. Similarly, fluctuations in lift are reduced, though by a more modest factor of around 5.7. Finally, a Fourier analysis of the drag coefficients obtained shows that the actuation slightly modifies the characteristic frequency of the system. The actively controlled system has a shedding frequency around 3.5% lower than the baseline. Several interesting points are visible from the active control signal imposed by the ANN presented in **Figure 3.2.3**. Firstly, the active flow control is composed of two phases. In the first one, the ANN changes the configuration of the flow by performing a relatively large transient actuation (non-dimensional time ranging from 0 to around 11). This changes the flow configuration, and sets the system in a state in which less drag is generated. Following this transient actuation, a second regime is reached in which a smaller actuation amplitude is used. The actuation in this new regime is pseudo-periodic. Therefore, it appears that the ANN has found a way to both set the flow in a modified configuration in which less drag is present, and keep it in this modified configuration at a relatively small cost. In a separate simulation, the small actuation present in the pseudo-periodic regime once the initial actuation has taken place was suppressed. This led to a rapid collapse of the modified flow regime, and the original base flow configuration was recovered. As a consequence, it appears that the modified flow configuration is unstable, though only small corrections are needed to keep the system in its neighborhood.

Secondly, it is striking to observe that the ANN resorts to quite small actuations. The peak value for the norm of the non-dimensional control mass flow rate Q^*_1 , which is reached during the transient active control regime, is only around 0.02, i.e. a factor 3 smaller than the maximum value allowed during training. Once the pseudo-periodic regime is established, the peak value of the actuation is reduced to around 0.006. This is an illustration of the sensitivity of the Navier-Stokes equations to small perturbations, and a proof that this property of the equations can be exploited to actively control the flow configuration, if forcing is applied in an appropriate manner.

3.2.5 Analysis of the Control Strategy Results

The ANN trained through DRL learns a control strategy by using a trial-and-error method. Understanding which strategy an ANN decides to use from the analysis of its weights is known to be challenging, even on simple image analysis tasks. Indeed, the strategy of the network is encoded in the complex combination of the weights of all its neurons. A number of properties of each individual network, such as the variations in architecture, make systematic analysis challenging [Rauber et al.][75]. Through the combination of the neuron weights, the network builds its own internal representation of how the flow in a given state will be affected by actuation, and how this will affect the reward value. This is a sort of private, 'encrypted' model obtained through experience and interaction with the flow.

Therefore, it appears challenging to directly analyze the control strategy from the trained network, which should be considered rather as a black box in this regard. Instead, we can look at macroscopic flow features and how the active control modifies them. This pinpoints the effect of the actuation on the flow and separation happening in the wake. Representative snapshots of the flow configuration in the baseline case (no actuation), and in the controlled case when the pseudo-periodic regime is reached (i.e., after the initial large transient actuation), are presented in **Figure 3.2.4**. As visible in **Figure 3.2.4**, the active control leads to a modification of the 2D flow configuration. In particular, the Kármán alley is altered in the case with active control and the velocity fluctuations induced by the vortexes are globally less strong, and less active close to the upper and lower walls.

[75] Rauber, Paulo E, Fadel, Samuel G, Falcao, Alexandre X & Telea, Alexandru C 2017 *Visualizing the hidden activity of artificial neural networks. IEEE transactions on visualization and computer graphics 23 (1),* 101–110.

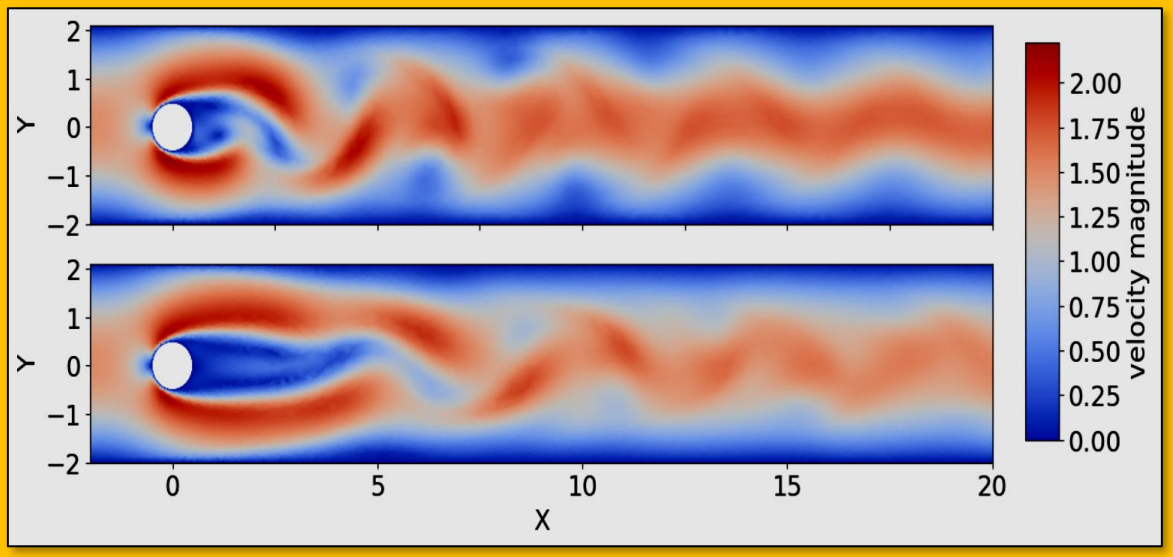


Figure 3.2.4 Comparison of representative snapshots of the velocity magnitude in the case without actuation (top), and with active flow control (bottom). The bottom figure corresponds to the established pseudo-periodic modified regime, which is attained after the initial transient control.

More strikingly, the extent of the recirculation area is dramatically increased. Defining the recirculation area as the region in the downstream neighborhood of the cylinder where the horizontal component of the velocity is negative, we observe a 130% increase in the recirculation area, averaged over the pseudo-period. The recirculation area in the active control case represents 103% of what is obtained in the hypothetical stable configuration of Appendix D of ([Rabault][76], (so, the recirculation area is slightly larger in the controlled case than in the hypothetical stable case, though the difference is so small that it may be due to a side effect such as slightly larger separation close to the jets, rather than a true change in the extent of the developed wake), while the recirculation area in the baseline configuration with vortex shedding is only 44% of this same stable configuration value. This is, similarly to what was observed for CD, an illustration of the efficiency of the control strategy at reducing the effect of vortex shedding. To go into more details, we look at the mean and the Standard Deviation (STD) of the flow velocity magnitude and pressure, averaged over a large number of vortex shedding periods (in the case with active flow control, we consider the pseudo-periodic regime).

3.2.6 Conclusion

We show for the first time that the Deep Reinforcement Learning paradigm (DRL, and more specifically the Proximal Policy Optimization algorithm, PPO) can discover an active flow control strategy for synthetic jets on a cylinder, and control the configuration of the 2D Kármán vortex street. From the point of view of the ANN and DRL, this is just yet-another-environment to interact with. The discovery of the control strategy takes place through the optimization of a reward function, here defined from the fluctuations of the drag and lift components experienced by the cylinder. A drag reduction of up to around 8 % is observed. In order to reduce drag, the ANN decides to increase the area of the separated region, which in turn induces a lower pressure drop behind the cylinder, and therefore lower drag. This brings the flow into a configuration that presents some similarities with what would be obtained from boat-tailing.

---

[76] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Reglade and Nicolas Cerardi, "*Artificial Neural Networks Trained Through Deep Reinforcement Learning Discover Control Strategies For Active Flow Control*", Journal of Fluid Mechanics, April 2019.

The value of the drag coefficient and extent of the recirculation bubble when control is turned on are very close to what is obtained by simulating the flow around a half cylinder using a symmetric boundary condition at the lower wall, which allows to estimate the drag expected around a cylinder at comparable Reynolds number if no vortex shedding was present. This implies that the active control is able to effectively cancel the detrimental effect of vortex shedding on drag. The learning obtained is remarkable, as little meta parameter tuning was necessary, and training takes place in about one day on a laptop. In addition, we have resorted to strong regularization of the output of the DRL agent through under sampling of the simulation and imposing a continuous control for helping the learning process. It could be expected that relaxing those constraints, i.e. giving more freedom to the network, could lead to even more efficient strategies.

These results are potentially of considerable importance for Fluid Mechanics, as they provide a proof that DRL can be used to solve the high dimensionality, analytically intractable problem of active flow control. The ANN and DRL approach has a number of strengths which make it an appealing methodology. In particular, ANNs allow for an efficient global approximation of strongly nonlinear functions, and they can be trained through direct experimentation of the DRL agent with the flow which makes it in theory easily applicable to both simulations and experiments without changes in the DRL methodology. In addition, once trained, the ANN requires only few calculations to compute the control at each time step. In the present case when two hidden layers of width 512 are used, most of the computational cost comes from a matrix multiplication, where the size of the matrices to multiply is [512, 512]. This is much less computationally expensive than the underlying problem. Finally, we are able to show that learning takes place in a timely manner, requiring a reasonable number of vortex shedding periods to obtain a converged strategy.

This work opens a number of research directions, including applying the DRL methodology to more complex simulations, for example more realistic 3D LES/DNS on large computer clusters, or even applying such an approach directly to a real world experiment. In addition, a number of interesting questions arise from the use of ANNs and DRL. For example, can some form of transfer learning be used between simulations and the real world if the simulations are realistic enough (i.e., can one train an ANN in a simulation, and then use it in the real world)? The use of DRL for active flow control may provide a technique to finally take advantage of advanced, complex flow actuation possibilities, such as those allowed by complex jet actuator arrays.

3.3 Case Study 3 - Solving Partial Differential Equations By A Supervised Learning Technique, Applied for the Reaction–Diffusion Equation

Authors : *Behzad Zakeri, Morteza Khashehchi, Sanaz Samsam, Atoosa Tayebi, Atefeh Rezaei*
Appearance: *Springer Nature Switzerland AG 2019*
Citation : *Zakeri, B., Khashehchi, M., Samsam, S. et al. Solving partial differential equations by a supervised learning technique, applied for the reaction–diffusion equation. SN Appl. Sci. 1, 1589 (2019). https://doi.org/10.1007/s42452-019-1630-x*

Deep learning is a crucial point of valuable intelligence resources to deal with complicated mathematical problems [Zakeri et al.][77]. The effectiveness of deep learning in solving differential equations has been considered over the past few years. Supervision of the learning process requires significant information to be marked in order to train the network. Nevertheless, this approach could not be a helpful strategy in case of unknown differential equations that we have no identified data. In order to address this problem, a new method for solving differential equations will be introduced in this paper using only the boundary and initial conditions. As an efficient method, inadequate monitoring can provide an ideal bed to fix boundary and initial value issues. For verification of the proposed method, a reaction–diffusion equation was performed. This equation has a variety of applications in engineering and science.

3.3.1 Introduction

Reaction–diffusion equation (RDE) is one of the well-known partial differential equations (PDEs) in engineering sciences as well as chemistry and finance [12]. RDE, by considering the simultaneous diffusion and dissipation of a property in the system gives an accurate model to predict the value of that property in the space-time domain. Coefficients, variables, boundary conditions type and dimensions of the RDE are highly dependent on the case which is studying, and based on the way that they are chosen, proper technique must be taken. All techniques aim to represent a precise solution (C(x, t)) of the RDE which describe diffusive property concentration in each time and position [21]. Although there are numerous numerical and analytical techniques to solve PDEs with different conditions, there is no global method which can handle all kinds of PDEs.

In recent years, by the explosive increment in available data, several heuristic approaches which all rely on the Machine Learning (ML) have been introduced for solving linear and non-linear PDEs [1, 2]. In all of them, models, by the utility of available data set tries to discovery the governing rule between initial/boundary conditions and the solution. Nevertheless, data-driven models can adequately learn the physics of the PDEs, but it requires the solution of the PDE before the solving process, which is not available in most cases.

In this work, weakly supervised learning was utilized to solve the one-dimensional RDE with constant coefficients and Dirichlet boundary condition. The benefit of this technique is that the network is trained whit only the boundary condition without providing any solution before the solving process. For the validation purpose, the analytical solution of the corresponding problem has been provided by taking advantage of Danckwert's transform. Also, the dimensional analysis method was used to assess the function of the neural network in solving pure reaction and pure diffusion without any training process.

3.3.2 Related Work and Literature Survey

The present study is a connection between artificial intelligence and dynamical systems, which each of them has been conducted in a variety of research studies. The former subject is researched by data scientists and AI developers. The main objective of weakly supervised learning is to provide a general

[77] Behzad Zakeri, Morteza Khashehchi, Sanaz Samsam, Atoosa Tayebi, Atefeh Rezaei, "*Solving Partial Differential Equations By A Supervised Learning Technique, Applied For The Reaction–Diffusion Equation*", Springer Nature Switzerland AG 2019.

platform technique for learning algorithms to be able to learn at the training stage with minimal initial marked data. The latter topic is aiming for a reliable method for solving PDEs. The use of various mathematical techniques for the estimation of the solution of PDEs is one of the essential aspects of this work.

Changing traditional numerical methods to alternative meshless approaches, such as machine learning, has become increasingly popular in recent years. Particularly in the event of problems with complex mathematical formulation, machine learning schemes are replaced by classical models [11]. [Oquab et al.] have used a weakly supervised convolutional neural network to identify objects in image processing to reduce the number of input labelled images. This method was a general concept and has been used in various applications, such as automated identification, medical image analysis and differential equation resolution [3, 8, 18]. [Sharma et al] trained an encoder–decoder U-Net architecture, which was a completely convoluted neural network to solve a steady-state two-dimensional heat equation on a regular square.

For this reason, weakly supervised learning techniques have been used to describe the proper convolutional kernel and loss function to train the network only by using the boundary conditions of the PDE, rather than having a large number of marked data sets [17]. [Han et al.] have introduced a new approach to use deep learning to solve high-dimensional PDEs. In the form of backward stochastic differential equations (BSDEs), they reformulate the PDEs and then use deep learning to approximate the gradient of the solution. Though their method is effective in dealing with high-dimensional situations, this method's limitations justify looking for the comprehensive approach to resolving linear and low-dimensional PDEs [6, 19].

Then again, customary numerical strategies, for example, FDM and FVM, have been broadly created to handle various sorts of scientific issues which portrayal of a precise answer for them is not available [12]. For example, for the above case of the utilization of the response dispersion condition in sulfate assault, [Zuo et al.] have utilized the limited contrast technique to discover the fixation dissemination of sulfate particles in concrete. Additionally, expanded looks into have led on the use of AI in various designing fields, for example, handling with violent streams and control theory [5, 10, 20].

3.3.3 Physics

3.3.3.1 Reaction–Diffusion Equation

For explaining 1-D RDE, a primary line has been considered as a domain with Dirichlet boundary condition at the parts of the bargains. By doling out the self-assertive consistent to the dissemination coefficients, we are able to control the part of the fabric on the transport phenomena. Too, the response coefficient indicated the impact of the interaction between the diffusive substance and medium. In this recreation, a high concentration connected to the boundaries, and the point is modelling the engendering of that substance among the space. The general form of the RDE in one-dimensional space is shown in **Eq. 3.3.1**, including the initial/boundary conditions , and we want to determine C(x, t), the concentration field in arbitrary time.

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2} - RC \quad \text{B.C. } C(0 < x < L, 0) = 0 \text{ and } C(0, t) = C(L, t) = C_0$$

Eq. 3.3.1

where D, R > 0 are the diffusion coefficient and reaction rate between specified material and domain respectively. The analytical procedures are not accessible in most cases. Be that as it may, by expecting the straight line as space and Dirichlet boundary condition, the expository arrangement of the RDE is appeared in taking after. Since convergence analysis within the neural networks is an impossible task [14], this arrangement plays a critical part within the approval of profound learning comes about.

3.3.3.2 Analytical Solution

For the utility of *Danckwert's method* to represent the analytical solution and by considering **Eq. 3.3.1** with constant coefficients (D and R), firstly, the equation must be solved without any source term (reaction term). Consequently, **Eq. 3.3.1** reform to the **Eq. 3.3.2** as follows:

$$\frac{\partial C_1}{\partial t} = D\frac{\partial^2 C_1}{\partial x^2}$$

Eq. 3.3.2

where C_1 represents the solution of the RDE without any reactive term. Let us attempt to find a nontrivial solution of (**Eq. 3.3.2**) satisfying the boundary conditions (**Eq. 3.3.1**) using ***separation of variables***:

$$C_1(x,t) = X(x)T(t)$$

Eq. 3.3.3

Substituting $C_1(x, t)$ back into **Eq. 3.3.2** one obtains:

$$\frac{1}{D}\frac{T'(t)}{T(t)} = \frac{X''(x)}{X(x)} = -\lambda^2$$

Eq. 3.3.4

where λ is an arbitrary positive coefficient. The solution of the corresponding ODEs of **Eq. 3.3.4** are:

$$T(t) = e^{-\lambda^2 Dt}$$

Eq. 3.3.5

$$X(x) = A\sin(\lambda x) + B\cos(\lambda x)$$

Eq. 3.3.6

leading to a of **Eq. 3.3.2** of the form:

$$C_1(x,t) = [A\sin(\lambda x) + B\cos(\lambda x)]e^{-\lambda^2 Dt}$$

Eq. 3.3.7

where A and B are constants of integration. Since **Eq. 3.3.2** is a linear equation, the most general solution is obtained by summing solutions of type **Eq. 3.3.7**, so that we have:

$$C_1(x,t) = \sum_{\alpha=1}^{\infty}[A_\alpha \sin(\lambda_\alpha x) + B_\alpha \cos(\lambda_\alpha x)]e^{-\lambda_\alpha{}^2 Dt}$$

Eq. 3.3.8

where A_α , B_α , and λ_α are determined by the initial and boundary conditions for any particular problem. The boundary conditions **Eq. 3.3.1** demand that:

$$A_\alpha = 0 \quad , \quad \lambda_\alpha = \frac{\alpha\pi}{L}$$

Eq. 3.3.9

By utility of Fourier series, the final solution of the **Eq. 3.3.2** have been extracted as the following form:

$$C_1(x,t) = \frac{4C_0}{\pi} \sum_{n=0}^{\infty} \frac{1}{2n+1} \exp\left(\frac{-D(2n+1)^2\pi^2 t}{L^2}\right) \cos\frac{(2n+1)\pi x}{L}$$

Eq. 3.3.10

Based on the *Danckwert's transform* [9], the solution of the **Eq. 3.3.1** can be calculated by the following integral transform:

$$C(x,t) = k\int_0^t C_1 e^{-kt'} dt' + C_1 e^{-kt}$$

Eq. 3.3.11

Finally, after the integration, the final solution of the 1-D RDE (**Eq. 3.3.1**) can be shown as follow:

$$C(x,t) = \frac{-4C_0}{\pi} \sum_{n=0}^{\infty} \langle a_n \cos(\omega_n x) \left\{ k\Psi_n \left[\exp\left(\frac{t}{\Psi_n}\right) - 1\right] + \exp\left(\frac{t}{\Psi_n}\right)\right\} k\rangle + C_0$$

Eq. 3.3.12

where a_n , ω_n and ψ_n are represented as follows:

$$a_n = \frac{(-1)^n}{(2n+1)'} \quad , \quad \omega_n = \frac{(2n+1)\pi}{2L} \quad , \quad \Psi_n = \frac{4L^2}{-D_e(2n+1)^2\pi^2 - 4kL^2}$$

Eq. 3.3.13

3.3.3.3 Finite Difference Method

The finite-difference is considered as a powerful numerical strategy which is utilized to compute the precise solution of the PDEs in arbitrary domains. In this strategy, governing physical equations and space both discretized, and all equations solve iteratively on the discrete space. Considering the discretization of the domain 16 and time 17, the discretized shape of RDE for position *m* and time *n* would be:

$$\frac{C_m^{n+1} - C_m^n}{\Delta t} = -D\frac{C_{m-1}^n - 2_m^n + C_{m+1}^n}{\Delta x^2} - RC_m^n$$

$$\frac{x_L - x_0}{m} = \Delta x \quad , \quad \frac{t_\infty - t_0}{n} = \Delta t$$

Eq. 3.3.14

where $C^n{}_m$ is the concentration in time n and position m, also indices 0, L and ∞ represent the boundaries in time and positions, ends of the domain and last time step respectively. With solving the **Eq. 3.3.14** iteratively, the value of the C in each time and position converge to the correct value.

3.3.3.4 Deep Learning Solver

The point of this work is utilizing the deep neural network to solve the RDE with as it was utilizing boundary and initial conditions, without knowing the numerical or analytical solution or indeed having any labelled information. For this reason, the differential form of the equation has been decoded into a ***physical-informed loss function***. This method makes a difference for us to discover the solution of the PDE without utilizing supervision within the frame of information. To define the initial and boundary conditions of the problem into the deep neural network, we utilized a n × m

matrix which its columns and rows represent the positions and time steps respectively. All of the matrix components for the input matrix are zero except the primary and final columns which their values represent to the boundary condition values (which in this case is C_0). Moreover, in this matrix each row demonstrates the concentration distribution in a specified time-frame.

3.3.3.5 Deep Learning Architecture

A fully convolutional encoder–decoder organize within the shape of U-Net engineering has been utilized in this consider, as [Ronneberger et al.] have utilized this design for biomedical picture segmentation. The most reason for choosing a fully convolutional design among other structures is the adaptability of this structure to fathom issues at different scales. The arrange contains a few encoding convolutional and translating pooling layers which spare the input matrix estimate amid the learning handle. At last, the output matrix gives the solution of the PDE within the discretized space-time space. The schematic structure of the network has appeared in **Figure 3.3.1**. As it appears in **Figure 3.3.1**, each encoding layer has been connected to the corresponding decoder layer utilizing Fusion link. The reason for the utility of fusion link is to pass the boundary values of the

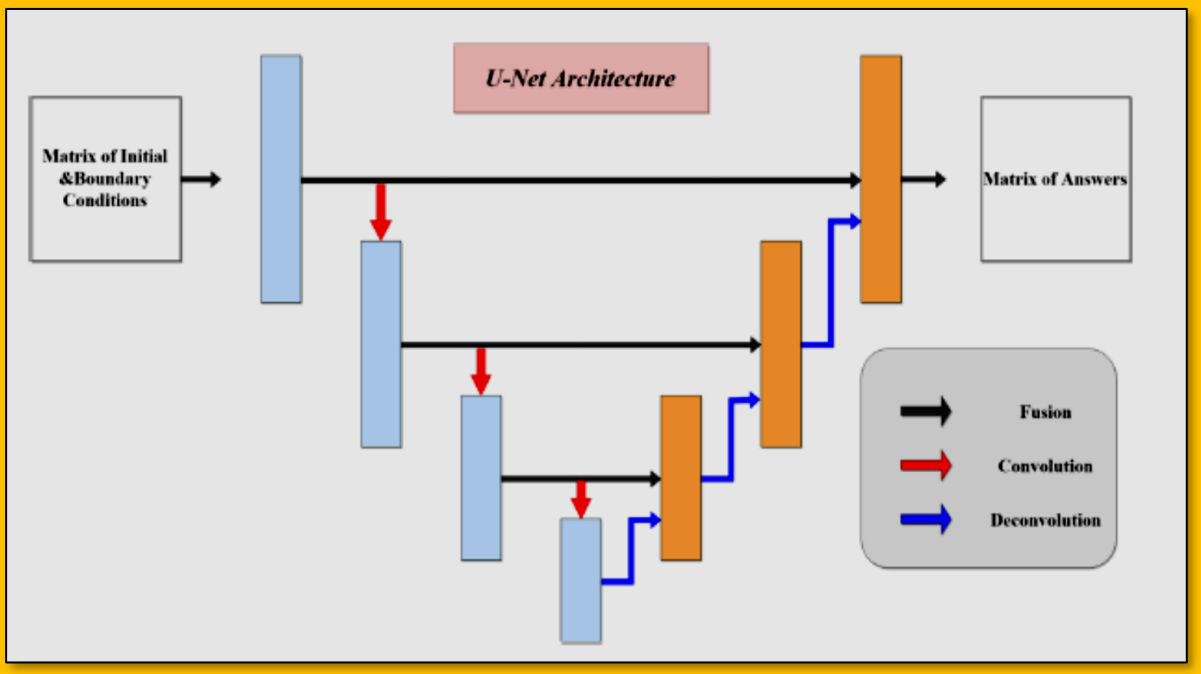


Figure 3.3.1 Schematic Diagram of the Network Architecture

input to the output layers, and by this procedure, the network is not constrained to memorize the structure of the input in its bottleneck layers. The number of layers in our design is self-assertive, and it is conceivable to include layers into the network as much as essential.

3.3.3.6 Kernel

To make an intelligent network that can solve the equation in any time and position, it is necessary to define the governing rule in that equation in a simple way for the neural network. It is similar to the method that FDM use for solving the discretized equation. In fact, by discretization of a continues equation and transferring that equation into the algebraic form we can observe the governing rule for every point in space and time. By reforming the **Eq. 3.3.14**, we can find the state of an arbitrary point in the space-time domain based on its neighbors as shown in:

$$C_m^{n+1} = C_m^n + B(C_{m-1}^n - 2C_m^n + C_{m+1}^n) - RC_m^n \Delta t$$

Eq. 3.3.15

And B defined as follow:

$$B = \frac{D\Delta t}{\Delta x^2}$$

Eq. 3.3.16

For transferring the relation among variables into the neural network, **Eq. 3.3.15** have been decoded into the 3 × 3convolutional kernel as follow:

$$\begin{bmatrix} -a & -b & -c \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Eq. 3.3.17

Where

$$a \to Bc \to Bb \to (1 - 2B - R\Delta t)$$

Eq. 3.3.18

Discussed Kernel has been convolved into the across the input matrix, and the output matrix after normalization was used to calculate the Loss function:

$$\sum_{ij} \left(\text{Conv2D}(\text{Kernel}, \text{Output})_{ij}\right)^2$$

Eq. 3.3.19

By minimizing the **Eq. 3.3.19**, the deep neural network tries to make its' solution closer to the real values which can be found in **Eq. 3.3.15** and changing in boundary and initial conditions train the network for solving any type of problems governed by reaction–diffusion physics.

3.3.4 Results

In this section, the solutions of the RDE which have been conducted by analytical and Deep Learning methods analyze and compare with previously validated solutions. The solution of the proposed equation has been represented by taking advantage of the numerical method (FDM) for determining the concentration of the sulfate ion in concrete. To have better judgment, boundary and initial condition of the demonstrated solution in this section have been chosen precisely similar to the [Zuo et al.]. As it was raised in the deep learning section, the output of the U-Net network for specific input is a matrix which its columns and rows represent the position and time, and the value of each element demonstrates the concentration in *i*th time and *j*th position. On the other hand, the proposed analytical solution in **Sect.**

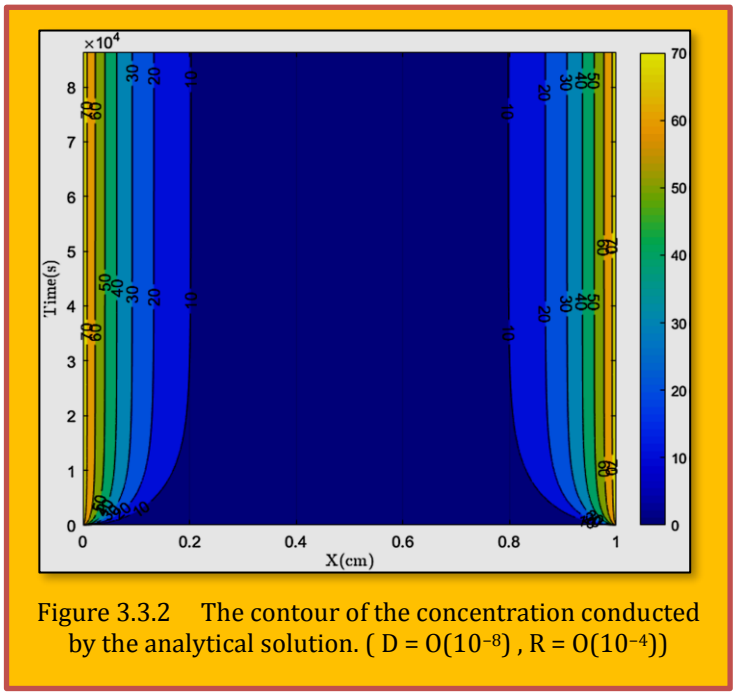


Figure 3.3.2 The contour of the concentration conducted by the analytical solution. ($D = O(10^{-8})$, $R = O(10^{-4})$)

3.2 gives the continues field of solution. **Figure 3.2.1** and **Figure 3.3.3** illustrate the contour of the analytical solution of the proposed equation.

Looking at **Figure 3.3.2** in more detail, the concentration diffusion along the time axis is apparent. Both the left and right edges of the demonstrated illustration represent the boundary conditions of the problem within this case are the same and equal to (C_0). By moving through the y-axis (Time-axis) of the contour, the propagation of the concentration advance, although this phenomenon is less visible in this image and more evident in **Figure 3.3.3**, because of the way the coefficients have chosen.

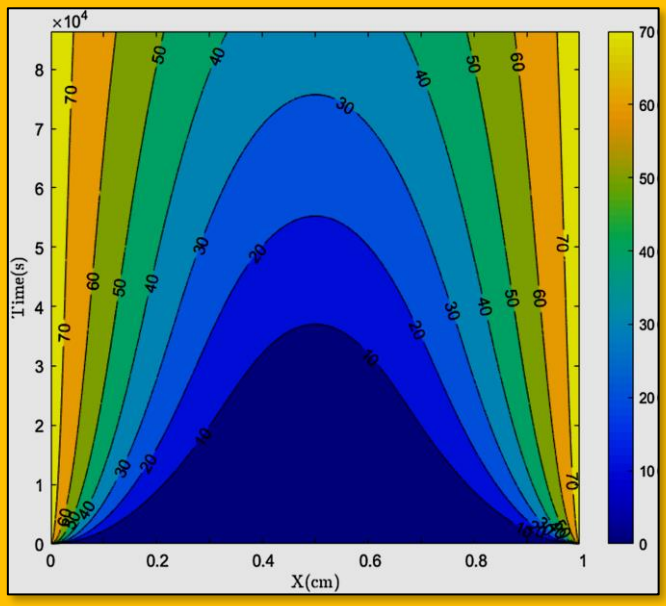


Figure 3.3.3 The contour of the concentration conducted by the analytical solution. ($D = O(10^{-8})$, $R = O(10^{-6})$)

Figure 3.3.3 shows the contours of the RDE solution with different coefficients compares to **Figure 3.3.2**. Reaction and Diffusion coefficients have chosen in a way that after some time, the diffusion process fills all the domain rather than **Figure 3.3.2**. In both **Figure 3.3.2** and **Figure 3.3.3,** the reaction and diffusion process plays a significant role, and none of them cannot be neglected. One of the demanding features of **Figure 3.3.3** is the bell shape of the iso-contours of the concentration. The reason for this shape is the nature of the propagation in such cases which behave such as waves.

Although the wave base solution of such systems is much developed for semi-infinite domains with pure diffusion, the nature of the wave behavior of the RDE is evident in **Figure 3.3.3**. Several cases of wave solution of such systems can be found in the mathematical literature of diffusion phenomena [4, 7, 16].

Figure 3.3.4 compares the deep learning solution with finite difference method comes about in terms of concentration propagation along with space at various times. With looking more accurately to **Figure 3.3.4**, we watch that deep learning comes about to have high consistency with FDM comes

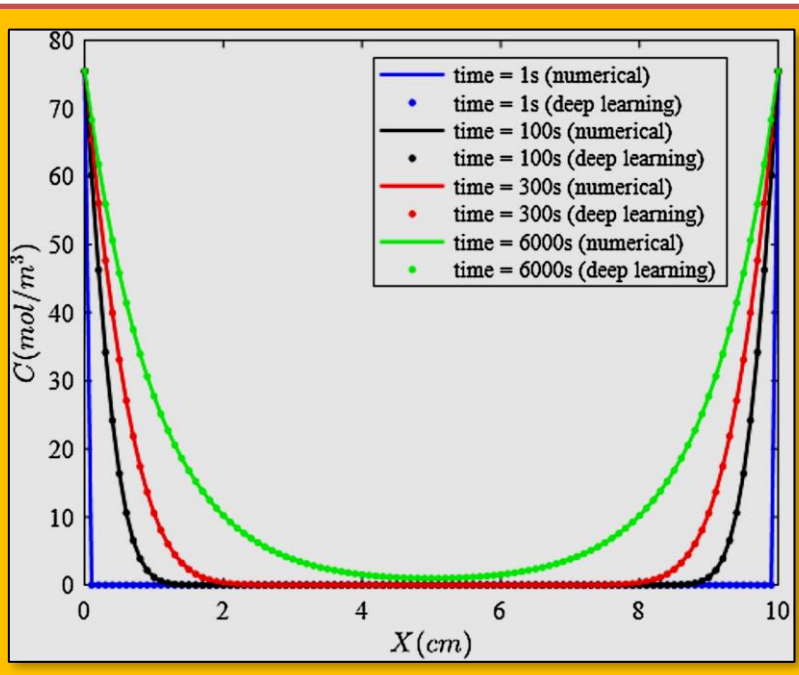


Figure 3.3.4 2D Comparison of Deep Learning With FDM Solver In Different Times

about. However, it seems that deep learning cannot accurately forecast the concentrations values in primary timesteps such as 1*s*. Many reasons can be named for the lack of deep learning in predicting the correct values in primary time-steps, but the most critical and compelling reason is the high gradient in this space-time.

In **Figure 3.3.5**, 3D results of the reaction–diffusion solutions conducted by deep learning and analytical solution have been demonstrated. To have a better perspective of the physics of the reaction–diffusion process, the proportion of the reaction rate and diffusion coefficient have chosen meticulously. The proportion of reaction and diffusion coefficient can determine the physics of the system among pure diffusion, pure reaction, and simultaneous reaction–diffusion. In fact, by choosing the correct range of coefficient, the role of the arbitrary term in the equation(reaction or diffusion) can outweigh the other term.

For this reason, a dimensionless coefficient has been characterized which offer assistance us to calculate the right extent of response and dissemination coefficients to have all three state of the arrangement in our computation. ***Damköhler number*** is an important dimensionless parameter in chemical engineering which clarifies the role of diffusion, reaction or simultaneous reaction–diffusion phenomena in transport phenomena and define as follow:

$$D_a = \frac{\text{Rate of Reaction}}{\text{Diffusion Rate}}$$

Eq. 3.3.20

In our model **Eq. 3.3.1**, ***Damköhler number*** is defined as:

$$D_a = \frac{RL^2}{D}$$

Eq. 3.3.21

This number represents the states of reaction–diffusion in different states where $D_a \cong 1$, $D_a \gg 1$, and $D_a \ll 1$ mean the physics of ***Reaction–Diffusion, pure Reaction***, and ***pure Diffusion*** respectively. The Mean Square Error (MSE) index has been utilized to calculate the deep learning faults in predicting the correct values of the equation. It has been observed that the final value of the concentration in space-time is dependent on the reaction and diffusion coefficients. However, this dependency is not as much as affect the accuracy of the final results in a way that they become unreliable. To have a quantitative assessment of deep learning solution, we assumed one of the coefficients constant, and by changing the other coefficient MSE value has been computed, and the result of this analysis is reported in **Table 3.3.1**.

R coefficient	*MSE* value
(a) $D = 2.7 \times 10^{-9}$	
2.25×10^{-2}	0.587
2.25×10^{-5}	0.495
2.25×10^{-7}	0.623
2.25×10^{-10}	0.341
D coefficient	*MSE* value
(b) $R = 2.25 \times 10^{-7}$	
2.7×10^{-2}	0.305
2.7×10^{-7}	0.576
2.7×10^{-8}	0.588
2.7×10^{-10}	0.534

Table 3.3.1 Accuracy Analyze Based On Changing Coefficients

3.3.5 Conclusion

In this paper, the capability of weakly supervised learning in comprehending the transient one-dimensional reaction–diffusion equation has been studied. Also, an analytical solution for the RDE based on the separation of variable technique and the utility of Danchwert's transform has proposed. It appeared that the results conducted by deep learning method have grate consistency with analytical and numerical results. Moreover, it was observed that the values of the reaction and diffusion coefficients could cause the miss estimation by deep learning. Although these noises were not in such a level that manipulates our results in this case, it could be the source of destructive faults in other problems like BSDEs. Finally, it is worth emphasizing that weakly supervised learning, cloud

successfully tackle the lack of sufficient labelled data to learn the physics of the governing equations. Furthermore, this method can be considered for complex problems with limited labelled data and complex governing equation.

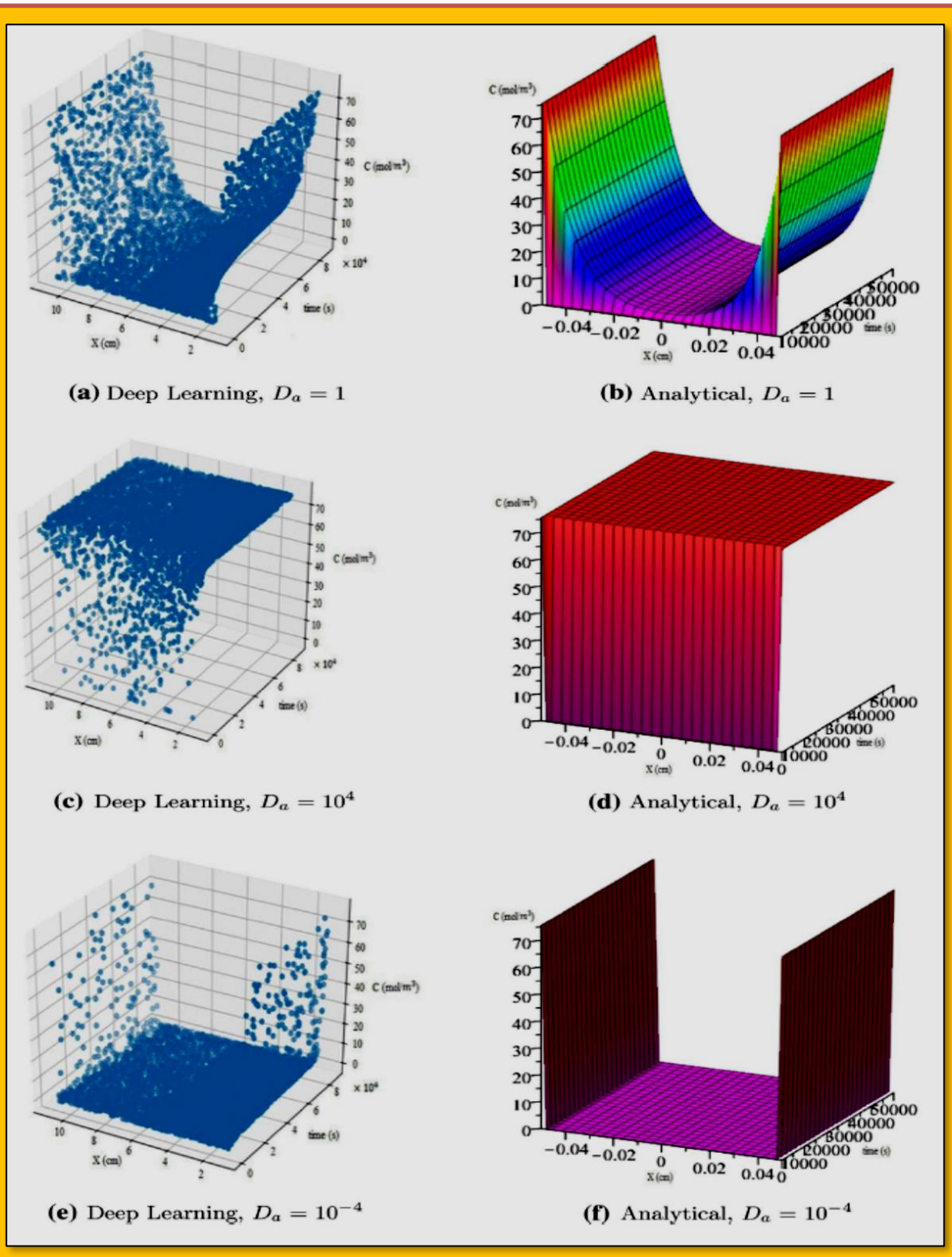


Figure 3.3.5 3D Comparison of Deep Learning with Analytical Solution

3.3.6 References

1. Berg J, Nyström K (2018) *A unified deep artificial neural network approach to partial differential equations in complex geometries*. Neurocomputing 317:28–41.
2. Berg J, Nyström K (2019) *Data-driven discovery of PDES in complex datasets*. J Com. Phys.
3. Ciompi F, de Hoop B, van Riel SJ, Chung K, Scholten ET, Oudkerk M, de Jong PA, Prokop M, van Ginneken B (2015) *Automatic classification of pulmonary peri-fissure nodules in computed tomography using an ensemble of 2d views and a convolutional neural network out-of-the-box*. Med Image Anal 26(1):195–202.
4. Crank J et al (1979) *The mathematics of diffusion*. Oxford University Press, Oxford.
5. Guo X, Yan W, Cui R (2019) *Integral reinforcement learning-based adaptive NN control for continuous-time nonlinear MIMO systems with unknown control directions*. IEEE Trans Syst Man Cybern Syst. https ://doi.org/10.1109/TSMC.2019.28972 21.
6. Han J, Jentzen A, Weinan E (2018) *Solving high-dimensional partial differential equations using deep learning.* Proc Natl Academy Sci 115(34):8505–8510.
7. Kuttler C (2011) *Reaction–diffusion equations with applications*. In: Internet seminar 8. Liu P, Gan J, Chakrabarty RK (2018) *Variational autoencoding the Lagrangian trajectories of particles in a combustion system*. arXiv preprint arXiv :1811.11896.
9. McNabb A (1993) *A generalized Danckwerts transformation.* Eur J Appl Math 4(2):189–204.
10. Mohan AT, Gaitonde DV (2018) *A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks*. arXiv preprint arXiv :1804.09269
11. Monsefi AK, Zakeri B, Samsam S, Khashehchi M. (2019) *Performing software test oracle based on deep neural network with fuzzy inference system*. Grandinetti L, Mirtaheri SL, Shahbazian R (eds) High-performance computing and big data analysis. Springer, Cham, pp 406–417
12. Morton KW, Mayers DF (2005) *Numerical solution of partial differential equations: an introduction*. Cambridge University Press, Cambridge
13. Oquab M, Bottou L, Laptev I, Sivic J (2015) *Is object localization for free?-weakly-supervised learning with convolutional neural networks*. Proceedings of the IEEE conference on computer vision and pattern recognition, pp 685–694.
14. Raissi M, Perdikaris P, Karniadakis GE (2019) *Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*. J Comput Phys 378:686–707.
15. Ronneberger O, Fischer P, Brox T (2015) *U-net: convolutional networks for biomedical image segmentation. In: International conference on medical image computing and computer-assisted intervention*. Springer, Berlin, pp 234–241.
16. Schell KG, Fett T, Bucharsky EC (2019) *Diffusion equation under swelling stresses*. SN Appl Sci 1(10):1300. https ://doi. org/10.1007/s4245 2-019-1343-1.
17. Sharma R, Farimani AB, Gomes J, Eastman P, Pande V (2018) *Weakly-supervised deep learning of heat transport via physics informed loss*. arXiv preprint arXiv :1807.11374
18. Tajbakhsh N, Shin JY, Gurudu SR, Hurst RT, Kendall CB, Gotway MB, Liang J (2016) *Convolutional neural networks for medical image analysis: full training or fine tuning?* IEEE Trans Med Imaging 35(5):1299–1312.
19. Weinan E, Han J, Jentzen A (2017) *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations*. Common Math Stat 5(4):349–380
20. Wu JL, Xiao H, Paterson E (2018) *Physics-informed machine learning approach for augmenting turbulence models: a comprehensive framework*. Phys Rev Fluids 3(7):074602
21. Zakeri B, Monsefi AK, Samsam S, Monsefi BK (2019) *Weakly supervised learning technique for solving partial differential equations; case study of 1-d reaction–diffusion equation*. Grandinetti L, Mirtaheri SL, Shahbazian R (eds) High-performance computing and big data analysis. Springer, Cham.

22. Zuo XB, Sun W, Yu C (2012) *Numerical investigation on expansive volume strain in concrete subjected to sulfate attack*. Const Build Mater 36:404–410.

3.4 Case Study 4 - Physics Informed Machine Learning (A Review)

Citation : *Karniadakis, G.E., Kevrekidis, I.G., Lu, L. et al. Physics-informed machine learning. Nat Rev Phys 3, 422–440 (2021). https://doi.org/10.1038/s42254-021-00314-5*
Original Appearance : *Nature Reviews | Physics*
Adaptation Mode : *Slight modification due to formatting*

Despite great progress in simulating multi physics problems using the numerical discretization of partial differential equations (PDEs), one still cannot seamlessly incorporate noisy data into existing algorithms, mesh generation remains complex, and high- dimensional problems governed by parameterized PDEs cannot be tackled. Moreover, solving inverse problems with hidden physics is often prohibitively expensive and requires different formulations and elaborate computer codes. Machine learning has emerged as a promising alternative, but training deep neural networks requires big data, not always available for scientific problems. Instead, such networks can be trained from additional information obtained by enforcing the physical laws (for example, at random points in the continuous space-time domain). Such physics informed learning integrates (noisy) data and mathematical models, and implements them through neural networks or other kernel based regression networks. Moreover, it may be possible to design specialized network architectures that automatically satisfy some of the physical invariants for better accuracy, faster training and improved generalization. Here, we review some of the prevailing trends in embedding physics into machine learning, present some of the current capabilities and limitations and discuss diverse applications of physics- informed learning both for forward and inverse problems, including discovering hidden physics and tackling high- dimensional problems.

Key Points

- Physics informed machine learning integrates seamlessly data and mathematical physics models, even in partially understood, uncertain and high- dimensional contexts.
- Kernel based or neural network- based regression methods offer effective, simple and meshless implementations.
- Physics informed neural networks are effective and efficient for ill posed and inverse problems, and combined with domain decomposition are scalable to large problems.
- Operator regression, search for new intrinsic variables and representations, and equivariant neural network architectures with built- in physical constraints are promising areas of future research.
- There is a need for developing new frameworks and standardized benchmarks as well as new mathematics for scalable, robust and rigorous next- generation physics- informed learning machines.

Modelling and forecasting the dynamics of multi-physics and multiscale systems remains an open scientific problem. Take for instance the Earth system, a uniquely complex system whose dynamics are intricately governed by the interaction of physical, chemical and biological processes taking place on spatiotemporal scales that span[17] orders of magnitude[1]. In the past 50 years, there has been tremendous progress in understanding multiscale physics in diverse applications, from geophysics to biophysics, by numerically solving partial differential equations (PDEs) using finite differences, finite elements, spectral and even meshless methods. Despite relentless progress, modelling and predicting the evolution of nonlinear multiscale systems with inhomogeneous cascades- of- scales by using classical analytical or computational tools inevitably faces severe challenges and introduces prohibitive cost and multiple sources of uncertainty. Moreover, solving inverse problems (for inferring material properties in functional materials or discovering missing physics in reactive transport, for example) is often prohibitively expensive and requires complex formulations, new

algorithms and elaborate computer codes. Most importantly, solving real- life physical problems with missing, gappy or noisy boundary conditions through traditional approaches is currently impossible. This is where and why observational data play a crucial role. With the prospect of more than a trillion sensors in the next decade, including airborne, seaborne and satellite remote sensing, a wealth of multi- fidelity observations is ready to be explored through data- driven methods.

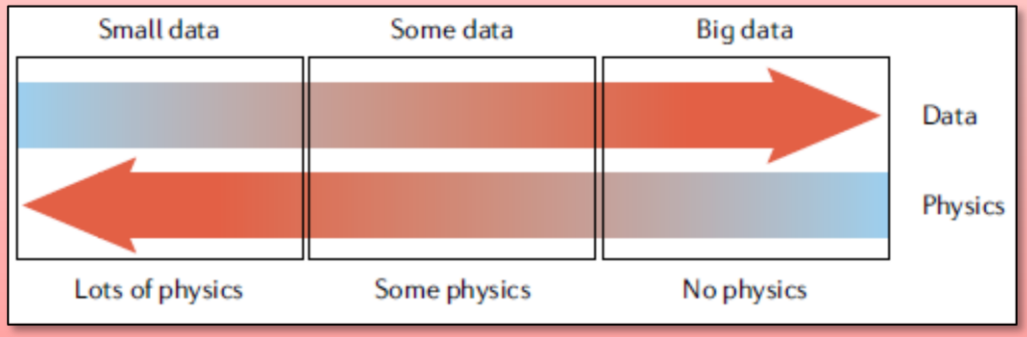
However, despite the volume, velocity and variety of available (collected or generated) data streams, in many real cases it is still not possible to seamlessly incorporate such multi- fidelity data into existing physical models.

Mathematical (and practical) data- assimilation efforts have been blossoming; yet the wealth and the spatiotemporal heterogeneity of available data, along with the lack of universally acceptable models, underscores the need for a transformative approach. This is where machine learning (ML) has come into play. It can explore massive design spaces, identify multi- dimensional correlations and manage ill- posed problems. It can, for instance, help to detect climate extremes or statistically predict dynamic variables such as precipitation or vegetation productivity[2,3]. Deep learning approaches, in particular, naturally provide tools for automatically extracting features from massive amounts of multi- fidelity observational data that are currently available and characterized by unprecedented spatial and temporal coverage[4]. They can also help to link these features with existing approximate models and exploit them in building new predictive tools. Even for biophysical and biomedical modelling, this synergistic integration between ML tools and multiscale and multi physics models has been recently advocated[5].

A common current theme across scientific domains is that the ability to collect and create observational data far outpaces the ability to assimilate it sensibly, let alone understand it[4] (**Box 1**).

Box 1

The figure below schematically illustrates three possible categories of physical problems and associated available data. In the small data regime, it is assumed that one knows all the physics, and data are provided for the initial and boundary conditions as well as the coefficients of a partial differential equation. The ubiquitous regime in applications is the middle one, where one knows some data and some physics, possibly missing some parameter values or even an entire term in the partial differential equation, for example, reactions in an advection–diffusion–reaction system. Finally, there is the regime with big data, where one may not know any of the physics, and where a data- driven approach may be most effective, for example, using operator regression methods to discover new physics. Physics- informed machine learning can seamlessly integrate data and the governing physical laws, including models with partially missing physics, in a unified way. This can be expressed compactly using automatic differentiation and neural networks[7] that are designed to produce predictions that respect the underlying physical principles.



Despite their towering empirical promise and some preliminary success[6], most ML approaches currently are unable to extract interpretable information and knowledge from this data deluge.

Moreover, purely data- driven models may fit observations very well, but predictions may be physically inconsistent or implausible, owing to extrapolation or observational biases that may lead to poor generalization performance. Therefore, there is a pressing need for integrating fundamental physical laws and domain knowledge by 'teaching' ML models about governing physical rules, which can, in turn, provide 'informative priors' that is, strong theoretical constraints and inductive biases on top of the observational ones.

To this end, physics- informed learning is needed, hereby defined as the process by which prior knowledge stemming from our observational, empirical, physical or mathematical understanding of the world can be leveraged to improve the performance of a learning algorithm. A recent example reflecting this new learning philosophy is the family of '***physics informed neural networks' (PINNs)***[7]. This is a class of deep learning algorithms that can seamlessly integrate data and abstract mathematical operators, including PDEs with or without missing physics (**Boxes 2,3**). The leading motivation for developing these algorithms is that such prior knowledge or constraints can yield more interpretable ML methods that remain robust in the presence of imperfect data (such as missing or noisy values, outliers and so on) and can provide accurate and physically consistent predictions, even for extrapolatory/generalization tasks.

Despite numerous public databases, the volume of useful experimental data for complex physical systems is limited. The specific data- driven approach to the predictive modelling of such systems depends crucially on the amount of data available and on the complexity of the system itself, as illustrated in **Box 1**. The classical paradigm is shown on the left side of the figure in **Box 1**, where it is assumed that the only data available are the boundary conditions and initial conditions whereas the specific governing PDEs and associated parameters are precisely known. On the other extreme (on the right side of the figure), a lot of data may be available, for instance, in the form of time series, but the governing physical law (the underlying PDE) may not be known at the continuum level[7-9]. For the majority of real applications, the most interesting category is sketched in the center of the figure, where it is assumed that the physics is partially known (that is, the conservation law, but not the constitutive relationship) but several scattered measurements (of a primary or auxiliary state) are available that can be used to infer parameters and even missing functional terms in the PDE while simultaneously recovering the solution.

It is clear that this middle category is the most general case, and in fact it is representative of the other two categories, if the measurements are too few or too many. This 'mixed' case may lead to much more complex scenarios, where the solution of the PDEs is a stochastic process due to stochastic excitation or an uncertain material property. Hence, stochastic PDEs can be used to represent these stochastic solutions and uncertainties. Finally, there are many problems involving long- range spatiotemporal interactions, such as turbulence, visco-elasto-plastic materials or other anomalous transport processes, where non local or fractional calculus and fractional PDEs may be the appropriate mathematical language to adequately describe such phenomena as they exhibit a rich expressivity not unlike that of ***deep neural networks (DNNs).***

Over the past two decades, efforts to account for uncertainty quantification in computer simulations have led to highly parameterized formulations that may include hundreds of uncertain parameters for complex problems, often rendering such computations infeasible in practice. Typically, computer codes at the national labs and even open- source programs such as *OpenFOAM10* or *LAMMPS11* have more than 100,000 lines of code, making it almost impossible to maintain and update them from one generation to the next. We believe that it is possible to overcome these fundamental and practical problems using physics informed learning, seamlessly integrating data and mathematical models, and implementing them using PINNs or other nonlinear regression based ***physics informed networks (PINs)*** (**Box 2**).

In this Review, we first describe how to embed physics in ML and how different physics can provide guidance to developing new neural network (NN) architectures. We then present some of the new capabilities of physics informed learning machines and highlight relevant applications. This is a very

Box 2

Making a learning algorithm physics- informed amounts to introducing appropriate observational, inductive or learning biases that can steer the learning process towards identifying physically consistent solutions (see the figure).

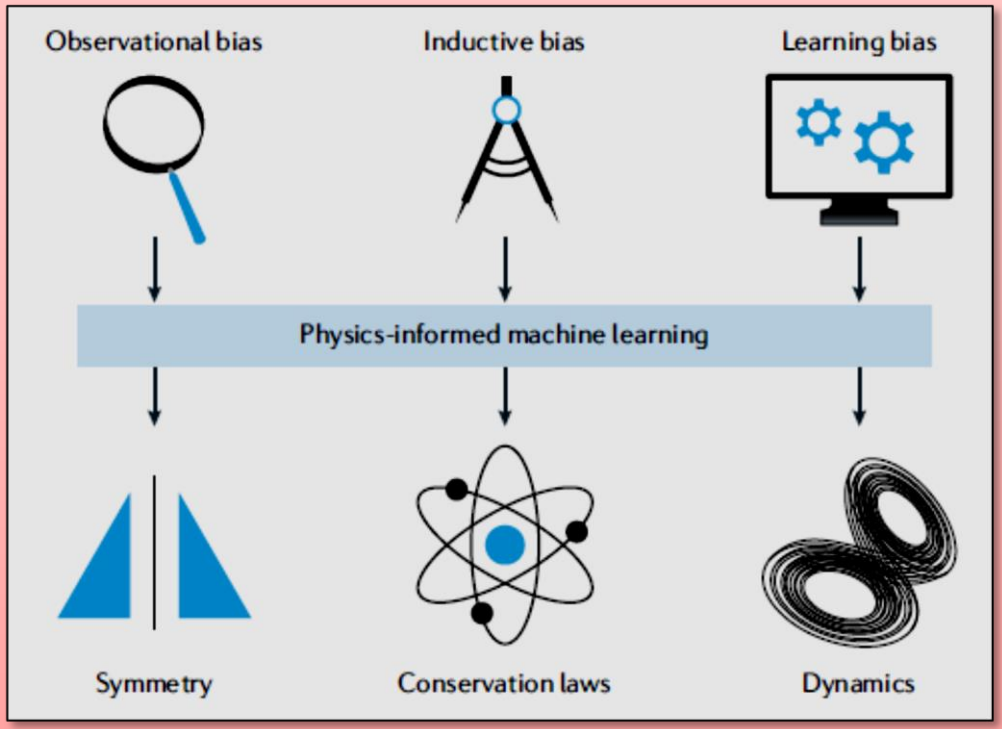
• Observational biases can be introduced directly through data that embody the underlying physics or carefully crafted data augmentation procedures. Training a machine learning (ML) system on such data allows it to learn functions, vector fields and operators that reflect the physical structure of the data.

• Inductive biases correspond to prior assumptions that can be incorporated by tailored interventions to an ML model architecture, such that the predictions sought are guaranteed to implicitly satisfy a set of given physical laws, typically expressed in the form of certain mathematical constraints. One would argue that this is the most principled way of making a learning algorithm physics- informed, as it allows for the underlying physical constraints to be strictly satisfied. However, such approaches can be limited to accounting for relatively simple symmetry groups (such as translations, permutations, reflections, rotations and so on) that are known a priori, and may often lead to complex implementations that are difficult to scale.

• Learning biases can be introduced by appropriate choice of loss functions, constraints and inference algorithms that can modulate the training phase of an ML model to explicitly favor convergence towards solutions that adhere to the underlying physics.

By using and tuning such soft penalty constraints, the underlying physical laws can only be approximately satisfied; however, this provides a very flexible platform for introducing a broad class of physics- based biases that can be expressed in the form of integral, differential or even fractional equations.

These different modes of biasing a learning algorithm towards physically consistent solutions are not mutually exclusive and can be effectively combined to yield a very broad class of hybrid approaches for building physics- informed learning machines.



fast moving field, so at the end we provide an outlook, including some thoughts on current

limitations. A taxonomy of several existing physics- based methods integrated with ML can also be found in ref.[12].

3.4.1 How to Embed Physics in ML

No predictive models can be constructed without assumptions, and, as a consequence, no generalization performance can be expected by ML models without appropriate biases. Specific to physics informed learning, there are currently three pathways that can be followed separately or in tandem to accelerate training and enhance generalization of ML models by embedding physics in them (**Box 2**).

3.4.1.1 Observational Biases

Observational data are perhaps the foundation of the recent success of ML. They are also conceptually the simplest mode of introducing biases in ML. Given sufficient data to cover the input domain of a learning task, ML methods have demonstrated remarkable power in achieving accurate interpolation between the dots, even for high- dimensional tasks. For physical systems in particular, thanks to the rapid development of sensor networks, it is now possible to exploit a wealth of variable fidelity observations and monitor the evolution of complex phenomena across several spatial and temporal scales. These observational data ought to reflect the underlying physical principles that dictate their generation, and, in principle, can be used as a weak mechanism for embedding these principles into an ML model during its training phase. Examples include NNs proposed in refs[13–16].

However, especially for over parameterized deep learning models, a large volume of data is typically necessary to reinforce these biases and generate predictions that respect certain symmetries and conservation laws. In this case, an immediate difficulty relates to the cost of data acquisition, which for many applications in the physical and engineering sciences could be prohibitively large, as observational data may be generated via expensive experiments or large scale computational models.

3.4.1.2 Inductive Biases

Another school of thought pertains to efforts focused on designing specialized NN architectures that implicitly embed any prior knowledge and inductive biases associated with a given predictive task. Without a doubt, the most celebrated example in this category are convolutional NNs[17], which have revolutionized the field of computer vision by craftily respecting invariance along the groups of symmetries and distributed pattern representations found in natural images[18]. Additional representative examples include graph neural networks (GNNs)[19], equivariant networks[20], kernel methods such as Gaussian processes[21–26], and more general PINs[27], with kernels that are directly induced by the physical principles that govern a given task. Convolutional networks can be generalized to respect more symmetry groups, including rotations, reflections and more general gauge symmetry transformations[19,20]. This enables the development of a very general class of NN architectures on manifolds that depend only on the intrinsic geometry, leading to very effective models for computer vision tasks involving medical images[28], climate pattern segmentation[20] and others.

Translation invariant representations can also be constructed via wavelet based scattering transforms, which are stable to deformations and preserve high- frequency information[29]. Another example includes covariant NNs[30], tailored to conform with the rotation and translation invariances present in many body systems (**Figure 3.4.1a**). A similar example is the equivariant transformer networks[31], a family of differentiable mappings that improve the robustness of models for predefined continuous transformation groups. Despite their remarkable effectiveness, such approaches are currently limited to tasks that are characterized by relatively simple and well defined physics or symmetry groups, and often require craftsmanship and elaborate implementations. Moreover, their extension to more complex tasks is challenging, as the underlying invariances or conservation laws that characterize many physical systems are often poorly understood or hard to implicitly encode in a neural architecture.

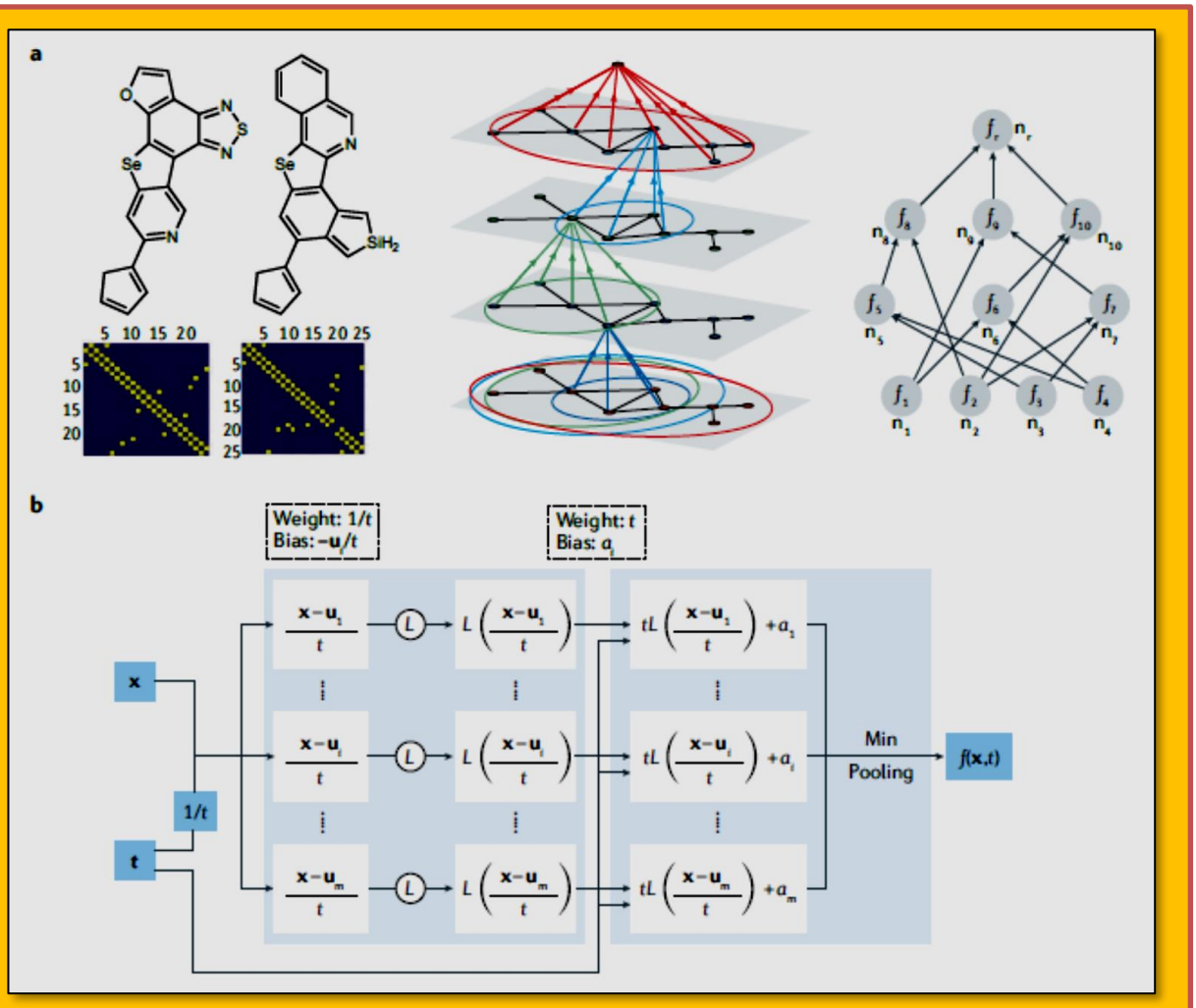


Figure 3.4.1 **Physics-inspired neural network architectures. a |** Predicting molecular properties with covariant compositional networks[204]. The architecture is based on graph neural networks 19 and is constructed by decomposing into a hierarchy of sub- graphs (middle) and forming a neural network in which each 'neuron' corresponds to one of the sub graphs and receives inputs from other neurons that correspond to smaller sub- graphs (right). The middle panel shows how this can equivalently be thought of as an algorithm in which each vertex receives and aggregates messages from its neighbors. Also depicted on the left are the molecular graphs for C18H9N3OSSe and C22H15NSeSi from the Harvard Clean Energy Project (HCEP) data set205 with their corresponding adjacency matrices. **b |** A neural network with the Lax–Oleinik formula represented in the architecture. *f* is the solution of the Hamilton–Jacobi partial differential equations, *x* and *t* are the spatial and temporal variables, *L* is a convex and Lipschitz activation function, *ai* _R and *ui* _R *n* are the neural network parameters, and *m* is the number of neurons. Panel **a** is adapted with permission from ref.204, AIP Publishing. Panel **b** image courtesy of J. Darbon and T. Meng, Brown University.

Generalized convolutions are not the only building blocks for designing architectures with strong implicit biases. For example, anti-symmetry under the exchange of input variables can be obtained in NNs by using the determinant of a matrix- valued function[32].

Reference[33] proposed to combine a physics- based model of bond- order potential with an NN and divide structural parameters into local and global parts to predict interatomic potential energy surface in large- scale atomistic modelling. In another work[34], an invariant tensor basis was used to embedded Galilean invariance into the network architecture, which significantly improved the NN

prediction accuracy in turbulence modelling. For the problem of identifying Hamiltonian systems, networks are designed to preserve the simplistic structure of the underlying Hamiltonian system[35]. For example, ref.[36] modified an auto encoder to represent a Koopman operator for identifying coordinate transformations that recast nonlinear dynamics into approximately linear ones.

Specifically for solving differential equations using NNs, architectures can be modified to satisfy exactly the required initial conditions[37], Dirichlet boundary conditions[37,38], Neumann boundary conditions[39,40], Robin boundary conditions[41], periodic boundary conditions[42,43] and interface conditions[41]. In addition, if some features of the PDE solutions are known a priori, it is also possible to encode them in network architectures, for example, multiscale features[44,45], even/odd symmetries and energy conservation[46], high frequencies[47] and so on.

For a specific example, we refer to the recent work in ref.[48], which proposed new connections between NN architectures and viscosity solutions to certain Hamilton–Jacobi PDEs (HJ- PDEs). The two- layer architecture depicted in **Figure 3.4.1b** defines

$$f(x,t) := \underbrace{\min}_{i\in\{1,,,,,m\}} \left[tL\left(\frac{x-u_i}{t}\right) + a_i \right]$$

Eq. 3.4.1

which is reminiscent of the celebrated Lax–Oleinik formula. Here, x and t are the spatial and temporal variables, L is a convex and Lipschitz activation function, $a_i \in R$ and $u_i \in R^n$ are the NN parameters, and *m* is the number of neurons. It is shown in ref.[48] that *f* is the viscosity solution to the following HJ-PDE

$$\frac{\partial f}{\partial t}(x,t) + H[\nabla_x f(x,t)] = 0 \quad , \quad x \in R^n \ , \quad t \in (0,\infty)$$
$$f(x,0) = J(x) \quad , \quad x \in R^n$$

Eq. 3.4.2

where both the Hamiltonian *H* and the initial data J are explicitly obtained by the parameters and the activation functions of the networks. The Hamiltonian H must be convex, but the initial data J are not. Note that the results of ref.[48] do not rely on universal approximation theorems established for NNs. Rather, the NNs in ref.[48] show that the physics contained in certain classes of HJ-PDEs can be naturally encoded by specific NN architectures without any numerical approximation in high dimensions.

3.4.1.3 Learning Bias

Yet another school of thought approaches the problem of endowing an NN with prior knowledge from a different angle. Instead of designing a specialized architecture that implicitly enforces this knowledge, current efforts aim to impose such constraints in a soft manner by appropriately penalizing the loss function of conventional NN approximations. This approach can be viewed as a specific use- case of multi task learning, in which a learning algorithm is simultaneously constrained to fit the observed data, and to yield predictions that approximately satisfy a given set of physical constraints (for example, conservation of mass, momentum, monotonicity and so on). Representative examples include the deep Galerkin method[49] and PINNs and their variants[7,37,50–52]. The framework of PINNs is further explained in **Box 3**, as it accurately reflects the key advantages and limitations of enforcing physics via soft penalty constraints.

The flexibility of soft penalty constraints allows one to incorporate more general instantiations of domain- specific knowledge into ML models. For example, ref.[53] presented a statistically constrained generative adversarial network (GAN) by enforcing constraints of covariance from the training data, which results in an improved ML based emulator to capture the statistics of the training data generated by solving fully resolved PDEs. Other examples include models tailored to learn contact-induced discontinuities in robotics[54], physics- informed auto- encoders[55], which use an additional

soft constraint to preserve the Lyapunov stability, and *InvNet*[56], which is capable of encoding invariances by soft constraints in the loss function.

Box 3

Physics informed neural networks (PINNs)[7] seamlessly integrate the information from both the measurements and partial differential equations (PDEs) by embedding the PDEs into the loss function of a neural network using automatic differentiation. The PDEs could be integer- order PDEs[7], integrodifferential equations[154], fractional PDEs[103] or stochastic PDEs[42,102]. Here, we present the PINN algorithm for solving forward problems using the example of the viscous Burgers' equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \upsilon\frac{\partial^2 u}{\partial x^2}$$

Eq. 3.4.3

with a suitable initial condition and Dirichlet boundary conditions. In the figure, the left (physics-uninformed) network represents the surrogate of the PDE solution u(x , t), while the right (physics-informed) network describes the **Eq. 3.4.3** residual. The loss function includes a supervised loss of data measurements of u from the initial and boundary conditions and an unsupervised loss of PDE:

$$\mathcal{L} = w_{data}\mathcal{L}_{data} + w_{PDE}\mathcal{L}_{PDE}$$

Eq. 3.4.4

Where

$$\mathcal{L}_{data} = \frac{1}{N_{DATA}}\sum_{i=1}^{N_{DATA}}[u(x_i, t_i) - u_i]$$

And

$$\mathcal{L}_{data} = \frac{1}{N_{PDE}}\sum_{j=1}^{N_{PDE}}\left.\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \upsilon\frac{\partial^2 u}{\partial x^2}\right|_{x_j t_j}$$

Here $\{x_i, t_i\}$ and $\{x_j, t_j\}$ are two sets of points sampled at the initial/boundary locations and in the entire domain, respectively, and u_i are values of u at (x_i, t_i); w_{data} and w_{PDE} are the weights used to balance the interplay between the two loss terms. These weights can be user- defined or tuned automatically, and play an important role in improving the trainability of PINNs[76,173]. The network is trained by minimizing the loss via gradient- based optimizers, such as Adam[196] and L-BFGS[206], until the loss is smaller than a threshold ε. The PINN algorithm is shown below, and more details about PINNs and a recommended Python library *DeepXDE* can be found in ref.[154].

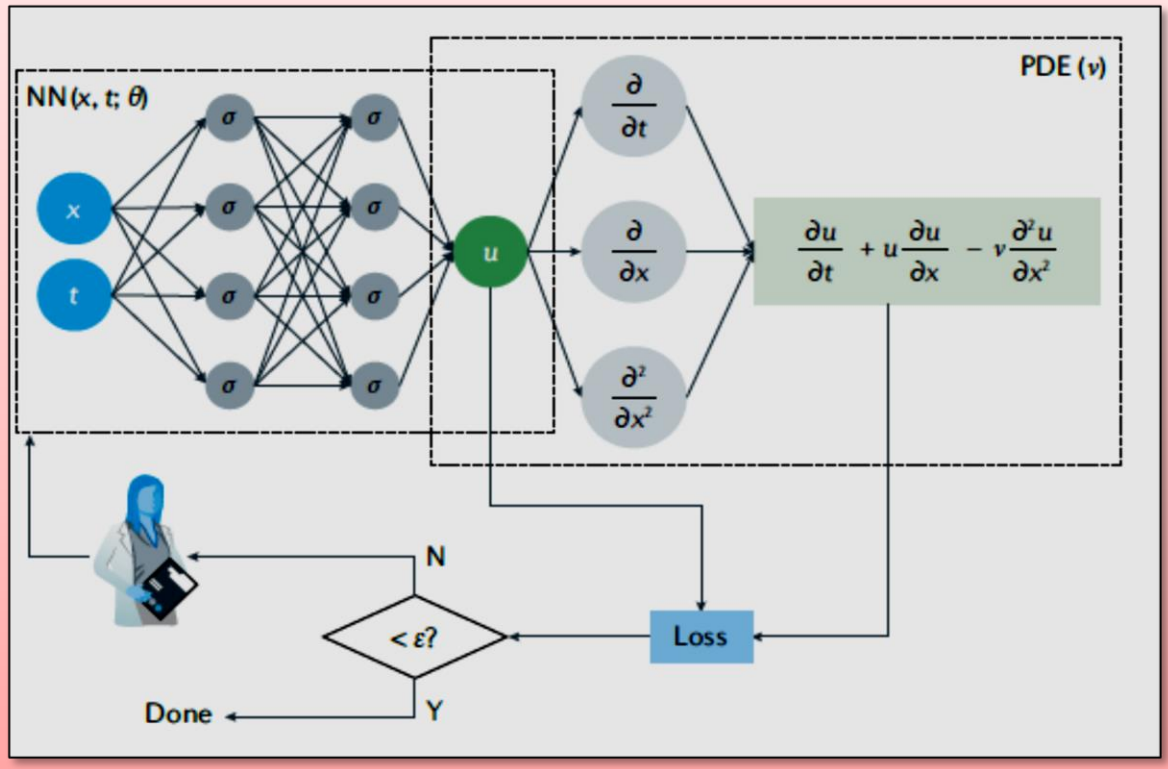
Algorithm 1: The PINN algorithm.

Construct a neural network (NN) $u(x, t; \boldsymbol{\theta})$ with $\boldsymbol{\theta}$ the set of trainable weights w and biases b, and σ denotes a nonlinear activation function. Specify the measurement data $\{x_i, t_i, u_i\}$ for u and the residual points $\{x_j, t_j\}$ for the PDE. Specify the loss L in **Eq. 3.4.4** by summing the weighted losses of the data and PDE. Train the NN to find the best parameters $\boldsymbol{\theta}^*$ by minimizing the loss *L*.

Further extensions include convolutional and recurrent architectures, and probabilistic formulations[51,52,57]. For example, ref.[52] includes a Bayesian framework that allows for uncertainty quantification of the predicted quantities of interest in complex PDE dynamical systems.

Note that solutions obtained via optimization with such soft penalty constraints and regularization can be viewed as equivalent to the maximum a posteriori estimate of a Bayesian formulation

Box 3 - Continued



stemming from physics- based likelihood assumptions. Alternatively, a fully Bayesian treatment using Markov chain Monte Carlo methods or variational inference approximations can be used to quantify the uncertainty arising from noisy and gappy data, as discussed below.

3.4.2 Hybrid Approaches

The aforementioned principles of physics informed ML have their own advantages and limitations. Hence, it would be ideal to use these different principles together, and indeed different hybrid approaches have been proposed. For example, non- dimensionalization can recover characteristic properties of a system, and thus it is beneficial to introduce physics bias via appropriate non-dimensional parameters, such as Reynolds, Froude or Mach numbers. Several methods have been proposed to learn operators that describe physical phenomena[13,15,58,59]. For example, *DeepONets*[13] have been demonstrated as a powerful tool to learn nonlinear operators in a supervised data- driven manner.

What is more exciting is that by combining *DeepONets* with physics encoded by PINNs, it is possible to accomplish real- time accurate predictions with extrapolation in multi-physics applications such as electro- convection[60] and hypersonics[61]. However, when a low- fidelity model is available, a multi-fidelity strategy[62] can be developed to facilitate the learning of a complex system. For example, ref.[63] combines observational and learning biases through the use of large eddy simulation data and constrained NN training methods to construct closures for lower fidelity Reynolds- averaged Navier–Stokes models of turbulent fluid flow.

Additional representative user cases include the multi fidelity NN used in ref.[64] to extract material properties from instrumented indentation data, the PINs in ref.[65] used to discover constitutive laws of non- Newtonian fluids from rheological data, and the coarse graining strategies proposed in ref.[66]. Even if it is not possible to encode the low fidelity model into the learning directly, the low fidelity

model can be used through data augmentation that is, generating a large amount of low fidelity data via inexpensive low- fidelity models, which could be simplified mathematical models or existing computer codes, such as ref.[64].

Other representative examples include *FermiNets*[32] and graph neural operator methods[58]. It is also possible to enforce the physics to an NN by embedding a network into a traditional numerical method (such as finite element). This approach was applied to solve problems in many different fields, including nonlinear dynamical systems[67], computational mechanics to model constitutive relations[68,69], subsurface mechanics[70–72], stochastic inversion[73] and more[74,75].

3.4.3 Connections to Kernel Methods

Many of the presented NN- based techniques have a close asymptotic connection to kernel methods, which can be exploited to produce new insight and understanding. For example, as demonstrated in refs[76,77], the training dynamics of PINNs can be understood as a kernel regression method as the width of the network goes to infinity. More generally, NN methods can be rigorously interpreted as kernel methods in which the underlying warping kernel is also learned from data[78,79]. Warping kernels are a special kind of kernels that were initially introduced to model non- stationary spatial structures in geostatistics[80] and have been also used to interpret residual NN models[27,80]. Furthermore, PINNs can be viewed as solving PDEs in a reproducing kernel Hilbert space spanned by a feature map (parametrized by the initial layers of the network), where the latter is also learned from data. Further connections can be made by studying the intimate connection between statistical inference techniques and numerical approximation.

Existing works have explored these connections in the context of solving PDEs and inverse problems[81], optimal recovery[82] and Bayesian numerical analysis[83–88]. Connections between kernel methods and NNs can be established even for large and complicated architectures, such as attention-based transformers[89], whereas operator valued kernel methods[90] could offer a viable path of analyzing and interpreting deep learning tools for learning nonlinear operators. In summary, analyzing NN models through the lens of kernel methods could have considerable benefits, as kernel methods are often interpretable and have strong theoretical foundations, which can subsequently help us to understand when and why deep learning methods may fail or succeed.

3.4.4 Connections to Classical Numerical Methods

Classical numerical algorithms, such as Runge–Kutta methods and finite- element methods, have been the main workhorses for studying and simulating physical systems in silico. Interestingly, many modern deep learning models can be viewed and analyzed by observing an obvious correspondence and specific connections to many of these classical algorithms. In particular, several architectures that have had tremendous success in practice are analogous to established strategies in numerical analysis. Convolutional NNs, for example, are analogous to finite different stencils in translationally equivariant PDE discretizations[91,92] and share the same structures as the multigrid method[93]; residual NNs (*ResNets*, networks with skip connections)[94] are analogous to the basic forward Euler discretization of autonomous ordinary differential equations[95–98]; inspection of simple Runge–Kutta schemes (such as an RK4) immediately brings forth the analogy with recurrent NN architectures (and even with Krylov- type matrix- free linear algebra methods such as the generalized minimal residual method)[95,99]. Moreover, the representation of DNNs with the ReLU activation function is equivalent to the continuous piecewise linear functions from the linear finite element method[100].

Such analogies can provide insights and guidance for cross- fertilization, and pave the way for new ‘mathematics- informed’ meta- learning architectures. For example, ref.[7] proposed a discrete- time NN method for solving PDEs that is inspired by an implicit Runge–Kutta integrator: using up to 500 latent stages, this NN method can allow very large time- steps and lead to solutions of high accuracy.

3.4.5 Merits of Physics Informed Learning

There are already many publications on physics informed ML across different disciplines for specific applications. For example, different extensions of PINNs cover conservation laws[101] as well as stochastic and fractional PDEs for random phenomena and for anomalous transport[102,103]. Combining domain decomposition with PINNs provides more flexibility in multiscale problems, while the formulations are relatively simple to implement in parallel since each subdomain may be represented by a different NN, assigned to a different GPU with very small communication cost[101,104,105]. Collectively, the results from these works demonstrate that PINNs are particularly effective in solving ill- posed and inverse problems, whereas for forward, well- posed problems that do not require any data assimilation the existing numerical grid based solvers currently outperform PINNs. In the following, we discuss in more detail for which scenarios the use of PINNs may be advantageous and highlight these advantages in some prototypical applications.

3.4.6 Incomplete Models and Imperfect Data

As shown in **Box 1**, physics- informed learning can easily combine both information from physics and scattered noisy data, even when both are imperfect. Recent research[106] demonstrated that it is possible to find meaningful solutions even when, because of smoothness or regularity inherent in the PINN formulation, the problem is not perfectly well posed. Examples include forward and inverse problems, where no initial or boundary conditions are specified or where some of the parameters in the PDEs are unknown scenarios in which classical numerical methods may fail. When dealing with imperfect models and data, it is beneficial to integrate the Bayesian approach with physics informed learning for uncertainty quantification, such as Bayesian PINNs (B-PINNs)[107]. Moreover, compared with the traditional numerical methods, physics- informed learning is mesh- free, without computationally expensive mesh generation, and thus can easily handle irregular and moving-domain problems[108]. Lastly, the code is also easier to implement by using existing open- source deep learning frameworks such as *TensorFlow* and *PyTorch*.

3.4.7 Strong Generalization in Small Data Regime

Deep learning usually requires a large amount of data for training, and in many physical problems it is difficult to obtain the necessary data at high accuracy. In these situations, physics- informed learning has the advantage of strong generalization in the small data regime. By enforcing or embedding physics, deep learning models are effectively constrained on a lower- dimensional manifold, and thus can be trained with a small amount of data. To enforce the physics, one can embed the physical principles into the network architecture, use physics as soft penalty constraints or use data augmentation as discussed previously. In addition, physics- informed learning is capable of extrapolation, not only interpolation: that is, it can perform spatial extrapolation in boundary value problems[107].

3.4.8 Understanding Deep Learning

In addition to enhancing the trainability and generalization of ML models, physical principles are also being used to provide theoretical insight and elucidate the inner mechanisms behind the surprising effectiveness of deep learning. For example, in refs[109–112], the authors use the jamming transition of granular media to understand the double- descent phenomenon of deep learning in the over-parameterized regime. Shallow NNs can also be viewed as interacting particle systems and hence can be analyzed in the probability measure space with mean field theory, instead of the high- dimensional parameter space[113].

Another work[114] rigorously constructed an exact mapping from the variational renormalization group to deep learning architectures based on restricted Boltzmann machines. Inspired by the successful density matrix renormalization group algorithm developed in physics, ref.[115] proposed a framework for applying quantum- inspired tensor networks to multi class supervised learning tasks, which introduces considerable savings in computational cost. Reference[116] studied the landscape of deep networks from a statistical physics viewpoint, establishing an intuitive connection between NNs

and the spin- glass models. In parallel, information propagation in wide DNNs has been studied based on dynamical systems theory[117,118], providing an analysis of how network initialization determines the propagation of an input signal through the network, hence identifying a set of hyper parameters and activation functions known as the 'edge of chaos' that ensure information propagation in deep networks.

3.4.9 Tackling High Dimensionality

Deep learning has been very successful in solving high dimensional problems, such as image classification with fine resolution, language modelling, and high- dimensional PDEs. One reason for this success is that DNNs can break the curse of dimensionality under the condition that the target function is a hierarchical composition of local functions[119,120]. For example, in ref.[121] the authors reformulated general high- dimensional parabolic PDEs using backward stochastic differential equations, approximating the gradient of the solution with DNNs, and then designing the loss based on the discretized stochastic integral and the given terminal condition. In practice, this approach was used to solve high- dimensional Black–Scholes, Hamilton–Jacobi–Bellman and Allen–Cahn equations. GANs[122] have also proven to be fairly successful in generating samples from high- dimensional distributions in tasks such as image or text generation[123–125]. As for their application to physical problems, in ref.[102] the authors used GANs to quantify parametric uncertainty in high- dimensional stochastic differential equations, and in ref.[126] GANs were used to learn parameters in high-dimensional stochastic dynamics.

These examples show the capability of GANs in modelling high- dimensional probability distributions in physical problems. Finally, in refs[127,128] it was demonstrated that even for operator regression and applications to PDEs, deep operator networks (*DeepONets*) can tackle the curse of dimensionality associated with the input space.

3.4.10 Uncertainty Quantification

Forecasting reliably the evolution of multiscale and Multi-physics systems requires uncertainty quantification. This important issue has received a lot of attention in the past 20 years, augmenting traditional computational methods with stochastic formulations to tackle uncertainty due to the boundary conditions or material properties[129–131]. For physics- informed learning models, there are at least three sources of uncertainty: uncertainty due to the physics, uncertainty due to the data, and uncertainty due to the learning models.

The first source of uncertainty refers to stochastic physical systems, which are usually described by stochastic PDEs (SPDEs) or stochastic ordinary differential equations (SODEs). The parametric uncertainty arising from the randomness of parameters lies in this category.

In ref.[132] the authors demonstrate the use of NNs as a projection function of the input that can recover a low- dimensional nonlinear manifold, and present results for a problem on uncertainty propagation in an SPDE with uncertain diffusion coefficient. In the same spirit, in ref.[133] the authors use a physics-informed loss function, that is, the expectation of the energy functional of the PDE over the stochastic variables , to train an NN parameterizing the solution of an elliptic SPDE. In ref.[51], a conditional convolutional generative model is used to predict the density of a solution, with a physics- informed probabilistic loss function so that no labels are required in the training data. Notably, as a model designed to learn distributions, GANs offer a powerful approach to solving stochastic PDEs in high dimensions.

The physics informed GANs in refs.[102,134] represent the first such attempts. Leveraging data collected from simultaneous reads at a limited number of sensors for the multiple stochastic processes, physics- informed GANs are able to solve a wide range of problems ranging from forward to inverse problems using the same framework. Also, the results so far show the capability of GANs, if properly formulated, to tackle the curse of dimensionality for problems with high stochastic dimensionality.

The second source of uncertainty, in general, refers to aleatoric uncertainty arising from the noise in

data and epistemic uncertainty arising from the gaps in data. Such uncertainty can be well tackled in the Bayesian framework.

If the physics informed learning model is based on Gaussian process regression, then it is straightforward to quantify uncertainty and exploit it for active learning and resolution refinement studies in PDEs[23,135], or even design better experiments[136].

Another approach was proposed in ref.[107] using B-PINNs. The authors of ref.[107] showed that B- PINNs can provide reasonable uncertainty bounds, which are of the same order as the error and increase as the size of noise in data increases, but how to set the prior for B-PINNs in a systematic way is still an open question.

The third source of uncertainty refers to the limitation of the learning models ; for example, the approximation, training and generalization errors of NNs and is usually hard to rigorously quantify. In ref.[137], a convolutional encoder–decoder NN is used to map the source term and the domain geometry of a PDE to the solution as well as the uncertainty, trained by a probabilistic supervised learning procedure with training data coming from finite- element methods.

Notably, a first attempt to quantify the combined uncertainty from learning was given in ref.[138], using the dropout method of ref.[139] and, due to physical randomness, using arbitrary polynomial chaos. An extension to time- dependent systems and long- time integration was reported in ref.[42]: it tackled the parametric uncertainty using dynamic and bi- orthogonal modal decomposition of the stochastic PDE, which are effective methods for long- term integration of stochastic systems.

3.4.11 Applications Highlights

In this section, we discuss some of the capabilities of physics informed learning through diverse applications. Our emphasis is on inverse and ill- posed problems, which are either difficult or impossible to solve with conventional approaches. We also present several ongoing efforts on developing open- source software for scientific ML.

3.4.12 Some Examples

3.4.12.1 Flow Over an Espresso Cup

In the first example, we discuss how to extract quantitative information on the 3D velocity and pressure fields above an espresso coffee cup140. The input data is based on a video of temperature gradient (**Figure 3.4.2**). This is an example of the 'hidden fluid mechanics' introduced In ref.[106]. It is an ill posed inverse problem as no boundary conditions or any other information are provided. Specifically, 3D visualizations obtained using tomographic background oriented Schlieren (Tomo-BOS) imaging that measures density or temperature are used as input to a PINN, which seamlessly integrates the visualization data and the flow and passive scalar governing equations, to infer the latent quantities. Here, the physical assumption is that of the Boussinesq approximation, which is valid if the density variation is relatively small. The PINN uses the space and time coordinates as inputs and infers the velocity and pressure fields; it is trained by minimizing a loss function including a data mismatch of temperature and the residuals of the conservation laws (mass, momentum and energy). Independent experimental results from particle image velocimetry have verified that the Tomo-BOS/PINN approach is able to provide continuous, high- resolution and accurate 3D flow fields.

3.4.12.2 Physics Informed Deep Learning for 4D Flow MRI

Next, we discuss the use of PINNs in biophysics using real magnetic resonance imaging (MRI) data. Because it is non- invasive and proves a range of structural and physiological contrasts, MRI has become an indispensable tool for quantitative in vivo assessment of blood flow and vascular function in clinical scenarios involving patients with cardiac and vascular disease. However, MRI measurements are often limited by the very coarse resolution and may be heavily corrupted by noise,

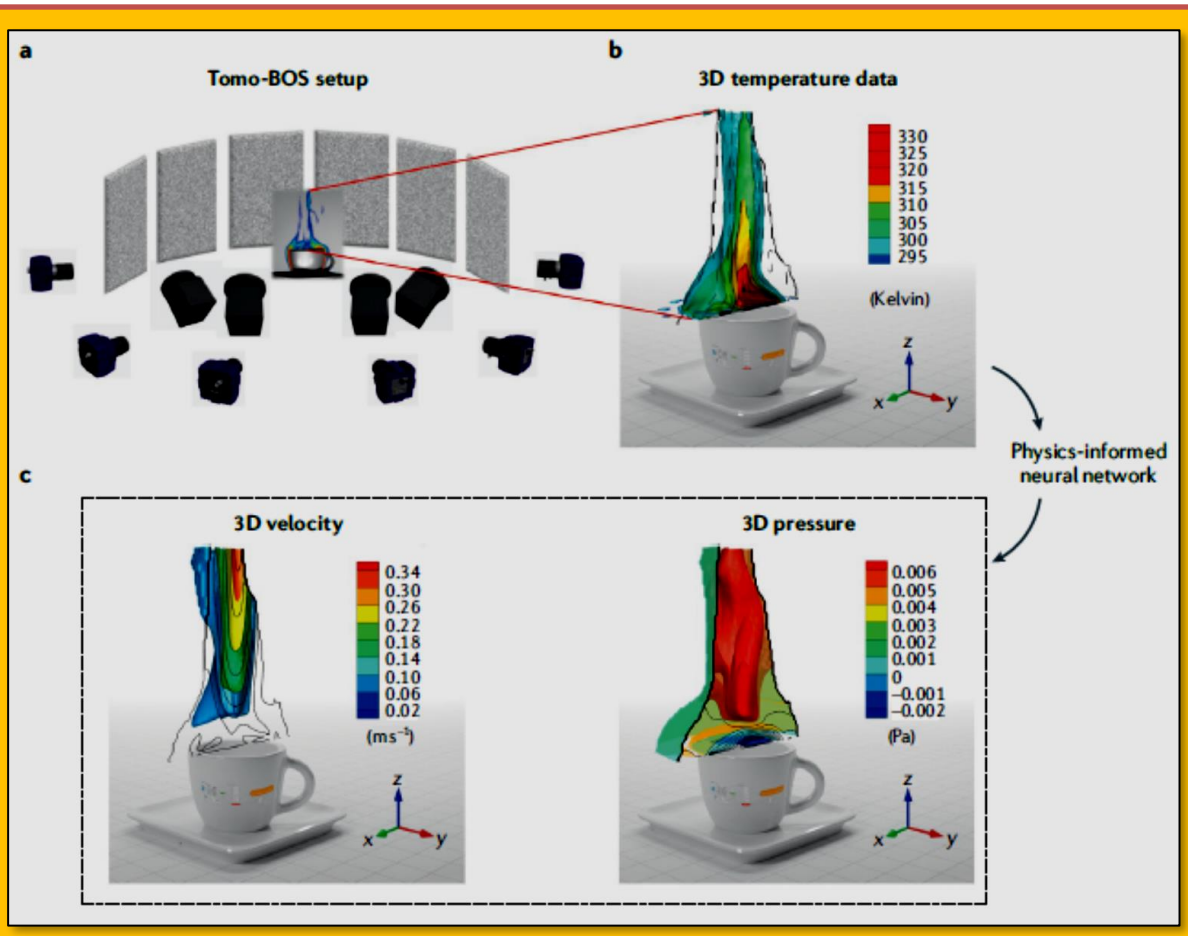


Figure 3.4.2 **inferring the 3D flow over an espresso cup based using the Tomo-BOs imaging system and physics-informed neural networks (PiNNs). a** | Six cameras are aligned around an espresso cup, recording the distortion of the dot- patterns in the panels placed in the background, where the distortion is caused by the density variation of the airflow above the espresso cup. The image data are acquired and processed with LaVision's Tomographic BOS software (DaVis 10.1.1). **b** | 3D temperature field derived from the refractive index field and reconstructed based on the 2D images from all six cameras. **c** | Physics- informed neural network (PINN) inference of the 3D velocity field (left) and pressure field (right) from the temperature data. The Tomo BOS experiment was performed by F. Fuest, Y. J. Jeon and C. Gray from LaVision. The PINN inference and visualization were performed by S. Cai and C. Li at Brown University. Image courtesy of S. Cai and C. Li, Brown University.

leading to tedious and empirical workflows for reconstructing vascular topologies and associated flow conditions.

Recent developments on physics- informed deep learning can greatly enhance the resolution and information content of current MRI technologies, with a focus on 4D-flow MRI. Specifically, it is possible to construct DNNs that are constrained by the Navier–Stokes equations in order to effectively de-noise MRI data and yield physically consistent reconstructions of the underlying velocity and pressure fields that ensure conservation of mass and momentum at an arbitrarily high spatial and temporal resolution. Moreover, the filtered velocity fields can be used to identify regions of no-slip flow, from which one can reconstruct the location and motion of the arterial wall and infer important quantities of interest such as wall shear stresses, kinetic energy and dissipation (**Figure 3.4.3**). Taken together, these methods can considerably advance the capabilities of MRI technologies in research and clinical scenarios. However, there are potential pitfalls related to the robustness of

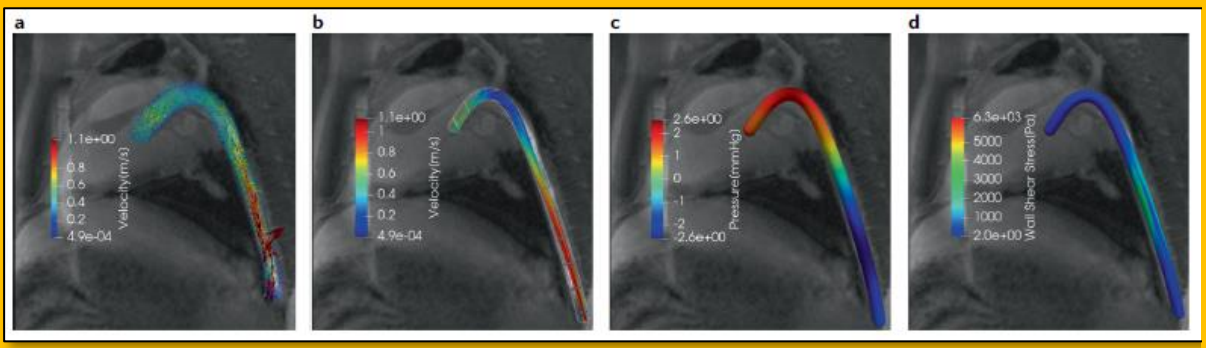


Figure 3.4.3 **Physics-informed filtering of in-vivo 4D-flow magnetic resonance imaging data of blood flow in a porcine descending aorta.** Physics- informed neural network (PINN) models can be used to de-noise and reconstruct clinical magnetic resonance imaging (MRI) data of blood velocity, while constraining this reconstruction to respect the underlying physical laws of momentum and mass conservation, as described by the incompressible Navier–Stokes equations. Moreover, a trained PINN model has the potential to aid the automatic segmentation of the arterial wall geometry and to infer important biomarkers such as blood pressure and wall shear stresses. **a** | Snapshot of in- vivo 4D- flow MRI measurements. **b**–**d** | A PINN reconstruction of the velocity field (panel **b**), pressure (panel **c**), arterial wall surface geometry and wall shear stresses (panel **d**). The 4D- flow MRI data were acquired by E. Hwuang and W. Witschey at the University of Pennsylvania. The PINN inference and visualization were performed by S. Wang, G. Kissas and P. Perdikaris at the University of Pennsylvania.

PINNs, especially in the presence of high signal- to- noise ratio in the MRI measurements and complex patterns in the underlying flow (for example, due to boundary layers, high vorticity regions, transient turbulent bursts through a stenosis, tortuous branched vessels and so on). That said, under physiological conditions, blood flow is laminar, a regime under which current PINN models usually remain effective.

3.4.12.3 Uncovering Edge Plasma Dynamics Via Deep Learning From Partial Observations

Predicting turbulent transport on the edge of magnetic confinement fusion devices is a longstanding goal spanning several decades, currently presenting significant uncertainties in the particle and energy confinement of fusion power plants. In ref.[141] it was demonstrated that PINNs can accurately learn turbulent field dynamics consistent with the two fluid theory from just partial observations of a synthetic plasma, for plasma diagnosis and model validation in challenging thermonuclear environments. **Figure 3.4.4** displays the turbulent radial electric field learned by PINNs from partial observations of a 3D synthetic plasma's electron density and temperature[141].

3.4.12.4 Studying Transitions Between Metastable States of a Distribution

Next, we discuss how physics informed learning can be creatively used to tackle high- dimensional problems. In ref.[142], the authors proposed to use physics informed learning to study transitions between two metastable states of a high- dimensional probability distribution. In particular, an NN was used to represent the committer function, trained with a physics- informed loss function defined as the variational formula for the committer function combined with a soft penalty on the boundary conditions. Moreover, adaptive importance sampling was used to sample rare events that dominate the loss function, which reduces the asymptotic variance of the solution and improves generalization. Results for a probability distribution in a 144 dimensional Allen–Cahn type system are illustrated in **Figure 3.4.5**.

Although these computational results suggest that this approach is effective for high- dimensional problems, the application of the method to more complicated systems and the selection of the NN architecture in adapting it to a given system remain challenging.

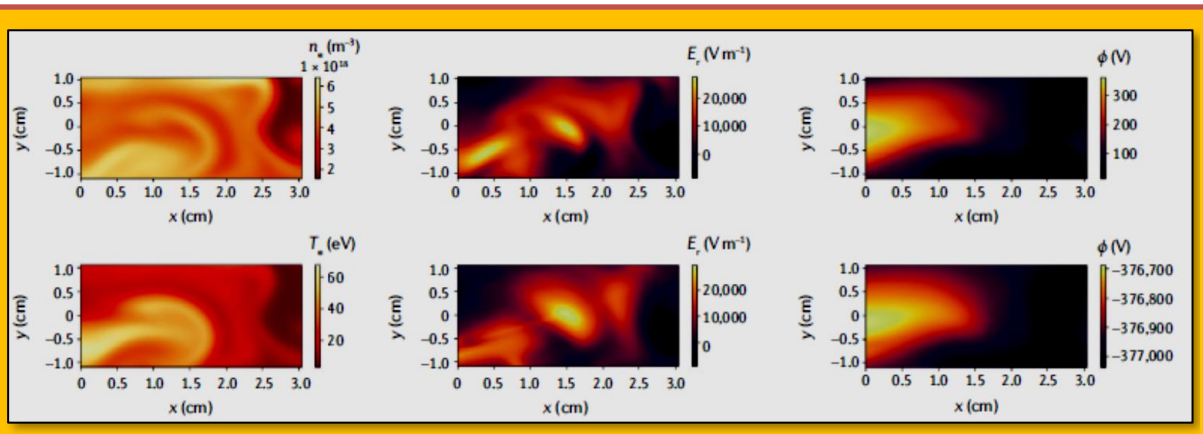


Figure 3.4.4 **Uncovering edge plasma dynamics.** One of the most intensely studied aspects of magnetic confinement fusion is edge plasma behavior, which is critical to reactor performance and operation. The drift- reduced Braginskii two- fluid theory has for decades been widely used to model edge plasmas, with varying success. Using a 3D magnetized two- fluid model, physics- informed neural networks (PINNs) can be used to accurately reconstruct[141] the unknown turbulent electric field (middle panel) and underlying electric potential (right panel), directly from partial observations of the plasma's electron density and temperature from a single test discharge (left panel). The top row shows the reference target solution, while the bottom row depicts the PINN model's prediction. These 2D synthetic measurements of electron density and temperature over the duration of a single plasma discharge constitute the only physical dynamics observed by the PINNs from the 3D collisional plasma exhibiting blob- like filaments. ϕ, electric potential; *E*r, electric field; *n*e, electron density; *T*e, electron temperature. Figure courtesy of A. Matthews, MIT.

3.4.12.5 Thermodynamically Consistent PINNs

The physics regularization generally pursued in PINNs admits an interpretation as a least- squares residual of point evaluations using an NN basis. For hyperbolic problems involving shocks, where point evaluation of the solution is ill- defined, it is natural to consider alternative physics stabilization requiring reduced regularity. The control volume PINN (cvPINN) pursued by ref.[143] generalizes traditional finite- volume schemes to deep learning settings. In addition to offering increased accuracy due to reduced regularity requirements, connections to traditional finite- volume schemes allow natural adaptation of total variation diminishing limiters and recovery of entropy solutions. This framework has allowed the estimation of black- box equations of state for shock hydrodynamics models appropriate for materials such as metals. For scenarios such as phase transitions at extreme pressures and temperatures, DNNs provide an ideal means of addressing unknown model form, whereas the finite- volume structure provided by cvPINNs allows enforcement of thermodynamic consistency.

3.4.12.6 Application to Quantum Chemistry

In some other applications, researchers have also used the physics to design specific new architectures together with the principles of physics informed learning. For example, in ref.[32], a fermionic NN (*FermiNet*) was proposed for the abolition calculation of the solution of the many-electron Schrödinger equation. *FermiNet* is a hybrid approach for embedding physics. First, to parameterize the wavefunction, the NN has a specialized architecture that obeys Fermi–Dirac statistics: that is, it is anti- symmetric under the exchange of input electron states and the boundary conditions (decay at infinity). Second, the training of *FermiNet* is also physics- informed: that is, the loss function is set as the variational form of the energy expectation value, with the gradient estimated by the Monte Carlo method. Although the application of NNs leads to eliminating the basis-set extrapolation, which is a common source of error in computational quantum chemistry, the

performance of NNs, in general, depends on many factors, including the architectures and optimization algorithms, which require further systematic investigation.

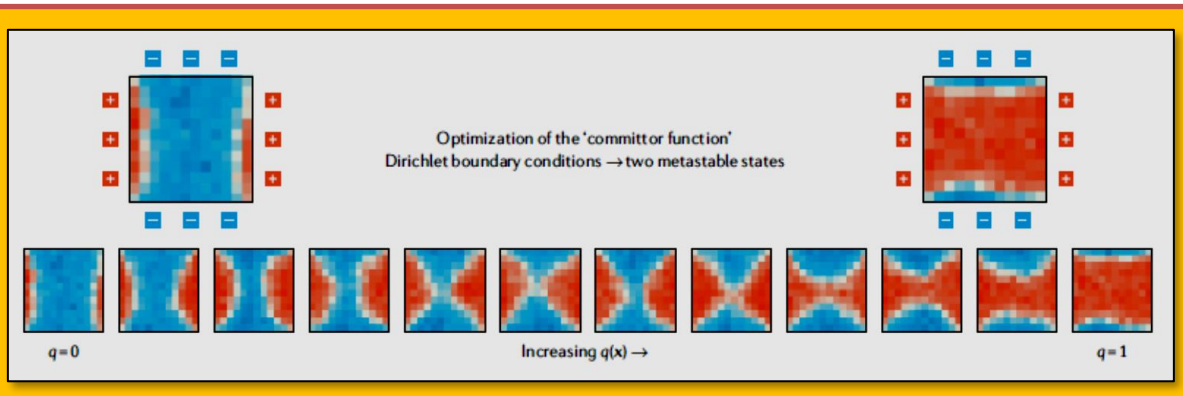


Figure 3.4.5 **Transitions between metastable states.** Results obtained from studying transitions between metastable states of a distribution in a 144- dimensional Allen–Cahn type system. The top part of the figure shows the two metastable states. The lower part of the figure shows, from left to right, a learned sample path with the characteristic nucleation pathway for a transition between the two metastable states. Here, q is the committor function. Figure courtesy of G. M. Rotskoff, Stanford University, and E. Vanden-Eijnden, Courant Institute.

3.4.12.7 Application to Material Sciences

In applications to materials, from the characterization of the material properties to the non-destructive evaluation of their strength, physics- informed learning can play an important role as the underlying problems are typically ill- posed and of inverse type. In ref.[144], the authors introduced an optimized PINN trained to identify and precisely characterize a surface breaking crack in a metal plate. The PINN was supervised with realistic ultrasonic surface acoustic wave data acquired at a frequency of 5 MHz and physically informed by the acoustic wave equation, with the unknown wave speed function represented as an NN. A key element in training was the use of adaptive activation functions, which introduced new trainable hyper parameters and substantially accelerated convergence even in the presence of significant noise in the data. An alternative approach to introducing physics into ML is through a multi- fidelity framework as in ref.[64] for extracting mechanical properties of 3D- printed materials via instrumented indentation. By solving the inverse problem of depth- sensing indentation, the authors could determine the elastoplastic properties of 3D- printed titanium and nickel alloys. In this framework, a composite NN consisting of two *ResNets* was used. One is a low fidelity *ResNet* that uses synthetic data (a lot of finite element simulations) and the other is a high fidelity *ResNet* that uses as input the sparse experimental data and the output of the low fidelity data. The objective was to discover the nonlinear correlation function between the low- and high- fidelity data, and subsequently predict the modulus of elasticity and yield stress at high fidelity. The results reported in ref.[64] show impressive performance of the multi fidelity framework, reducing the inference error for the yield stress from over 100% with existing techniques to lower than 5% with the multi fidelity framework.

3.4.12.8 Application to Molecular Simulations

In ref.[145], an NN architecture was proposed to represent the potential energy surfaces for molecular dynamics simulations, where the translational, rotational and permutational symmetry of the molecular system is preserved with proper pre- processing. Such an NN representation could be further improved in deep potential molecular dynamics (*DeePMD*)[146]. With traditional artificially designed potential energy functions replaced by the NN trained with data from ab initio simulations, *DeePMD* achieves an ab initio level of accuracy at a cost that scales linearly with the system size. In

ref.[147], the limit of molecular dynamics simulations was pushed with ab initio accuracy to simulating more than 1- ns- long trajectories of over 100 million atoms per day, using a highly optimized code for DeePMD on the Summit supercomputer. Before this work, molecular dynamics simulations with ab initio accuracy were performed in systems with up to 1 million atoms[147,148].

3.4.12.9 Application to Geophysics.

Physics informed learning has also been applied to various geophysical inverse problems. The work in ref.[71] estimates subsurface properties, such as rock permeability and porosity, from seismic data by coupling NNs with full- waveform inversion, subsurface flow processes and rock physics models. Furthermore, in ref.[149], it was demonstrated that by combining DNNs and numerical PDE solvers as we discussed in the section on hybrid approaches, physics- informed learning is capable of solving a wide class of seismic inversion problems, such as velocity estimation, fault rupture imaging, earthquake location and source–time function retrieval.

3.4.13 Software

To implement PINNs efficiently, it is advantageous to build new algorithms based on the current ML libraries, such as *TensorFlow*[150], *PyTorch*[151], *Keras*[152] and *JAX*[153]. Several software libraries specifically designed for physics-informed ML have been developed and are contributing to the rapid development of the field (**Table 3.4.1**). At the present time, some of the actively developed libraries include *DeepXDE*[154], *SimNet*[155], *PyDEns*[156], *NeuroDiffEq*[157], *NeuralPDE*[158], *SciANN*[159] and *ADCME*[160]. Because Python is the dominant programming language for ML, it is more convenient to use Python for physics- informed ML, and thus most of these libraries are written in Python, except the *NeuralPDE*[158] and *ADCME*[160], which are written in Julia.

All these libraries use the automatic differentiation mechanism provided in other software's such as *TensorFlow*[150]. Some of these libraries (such as *DeepXDE*[154] and *SimNet*[155]) can be used as a solver, that is, users only need to define the problem and then the solver will deal with all the underlying details and solve the problem, whereas some (such as *SciANN*[159] and *ADCME*[160]) only work as a wrapper, meaning they wrap low- level functions of other libraries (such as *TensorFlow*) into relatively high level functions for easier implementation of physics informed learning and users still need to implement all the steps to solve the problem.

Software name	Usage	Language	Backend	Ref.
DeepXDE	Solver	Python	TensorFlow	154
SimNet	Solver	Python	TensorFlow	155
PyDEns	Solver	Python	TensorFlow	156
NeuroDiffEq	Solver	Python	PyTorch	157
NeuralPDE	Solver	Julia	Julia	158
SciANN	Wrapper	Python	TensorFlow	159
ADCME	Wrapper	Julia	TensorFlow	160
GPyTorch	Wrapper	Python	PyTorch	161
Neural Tangents	Wrapper	Python	JAX	162

Table 3.4.1 Major software libraries specifically designed for physics- informed machine learning

Software packages such as *GPyTorch*[161] and *Neural Tangents*[162] also enable the study of NNs and PINNs through the lens of kernel methods. This viewpoint has produced new understanding of the training dynamics of PINNs, subsequently motivating the design of new effective architectures and training algorithms[76,77].

DeepXDE not only solves integer order ODEs and PDEs, but it can also solve integra differential equations and fractional PDEs. *DeepXDE* supports complex domain geometries via the technique of constructive solid geometry, and enables the user code to stay compact, resembling closely the mathematical formulation. *DeepXDE* is also well- structured and highly configurable, since all its components are loosely coupled. We note that in addition to being used as a research tool for solving problems in computational science and engineering, *DeepXDE* can also be used as an educational tool in diverse courses. Although *DeepXDE* is suitable for education and research, SimNet[155] developed by Nvidia is specifically optimized for *Nvidia GPU*s for large scale engineering problems.

In PINNs (**Box 3**), one needs to compute the derivatives of the network outputs with respect to the network inputs. One can compute the derivatives using automatic differentiation provided by ML packages such as *TensorFlow*[150]. For example, $\partial U/\partial t$ -can be computed using *TensorFlow* as *tf.gradients* (U, t), and second order derivatives can be computed by applying *tf.gradients* twice. *DeepXDE* provides a more convenient way to compute higher order derivatives, for example using *dde.grad.hessian* to compute the Hessian matrix. Moreover, there are two extra advantages to using *dde.grad.hessian*: first, it is lazy evaluation, meaning it will only compute an element in the Hessian matrix until that element is needed, rather than computing the whole Hessian matrix.

Second, it memorizes all the gradients that have already been computed to avoid duplicate computation, even if the user calls the function multiple times in different parts of the code. These two features could speed up the computation in problems where one needs to compute the gradients many times, for example in a system of coupled PDEs. Most of these libraries (such as *DeepXDE* and *SimNet*) use physics as the soft penalty constraints (**Box 3**), and ADCME embeds DNNs in standard scientific numerical schemes (such as Runge–Kutta methods for ODEs, and the finite- difference, finite element and finite- volume methods for PDEs) to solve inverse problems. *ADCME* was recently extended to support implicit schemes and nonlinear constraints[163,164]. To enable truly large- scale scientific computations on large meshes, support for MPI based domain decomposition methods is also available and was demonstrated to scale very well on complex problems[165].

3.4.13.1 Which Model, Framework, Algorithm To Use?

With a growing collection of methodologies and software tools, a series of questions naturally arises: given a physical system and/or governing law and some observational data, which ML framework should one use? Which training algorithm to choose? How many training samples to consider? Although at present there are no rule- of- thumb strategies for answering these questions, and some degree of experience is required to set up a physics- informed ML model properly, meta- learning techniques[166-168] could automate this process in the future. The choices intimately depend on the specific task that needs to be tackled. In terms of providing a high- level taxonomy, we note that PINNs are typically used to infer a deterministic function that is compatible with an underlying physical law when a limited number of observations is available (either initial/boundary conditions or other measurements).

The underlying architecture of a PINNs model is determined by the nature of a given problem: multi-layer perceptron architectures are generally applicable but do not encode any specialized inductive biases, convolutional NN architectures are suitable for gridded 2D domains, Fourier feature networks are suitable for PDEs whose solution exhibits high frequencies or periodic boundaries, and recurrent architectures are suitable for non Markovian and time- discrete problems. Moreover, probabilistic variants of PINNs can also be used to infer stochastic processes that can allow capturing epistemic/model uncertainty (via Bayesian inference or frequentist ensembles) or aleatoric uncertainty (via generative models such as variational auto encoders and GANs).

However, the *DeepONet* framework can be used to infer an operator (instead of a function). In *DeepONet*, the choice of the underlying architecture can also vary depending on the nature of available data, such as scattered sensor measurements (multi-layer perceptron), images (convolutional NNs) or time series (recurrent NNs). In all the aforementioned cases, the required sample complexity is typically not known a priori and is generally determined by: the strength of inductive biases used in the architecture; the compatibility between the observed data, and the underlying physical law used as regularization; and the complexity of the underlying function or operator to be approximated.

3.4.14 Current Limitations

3.4.14.1 Multiscale and Multi-Physics Problems

Despite the recent success of physics- informed learning across a range of applications, multiscale and multi physics problems require further developments. For example, fully connected NNs have difficulty learning high- frequency functions, a phenomenon referred to in the literature as the 'F-principle'[169] or 'spectral bias'[170]. Additional work[171,172] rigorously proved the existence of frequency bias in DNNs and derived convergence rates of training as a function of target frequency. Moreover, high frequency features in the target solution generally result in steep gradients, and thus PINN models often struggle to penalize accurately the PDE residuals[45]. As a consequence, for multiscale problems, the networks struggle to learn high- frequency components and often may fail to train[76,173]. To address the challenge of learning high frequency components, one needs to develop new techniques to aid the network learning, such as domain decomposition[105], Fourier features[174] and multiscale DNN45[175].

However, learning multi-physics simultaneously could be computationally expensive. To address this issue, one may first learn each physics separately and then couple them together. In the method of *DeepM&M* for the problems of electro- convection[60] and hypersonics[61], several *DeepONets* were first trained for each field separately and subsequently learned the coupled solutions through either a parallel or a serial *DeepM&M* architecture using supervised learning based on additional data for a specific multi-physics problem. It is also possible to learn the physics at a coarse scale by using the fine- scale simulation data only in small domains[176].

Currently in NN based ML methods, the physics informed loss functions are mainly defined in a pointwise way. Although NNs with such loss functions can be successful in some high dimensional problems, they may also fail in some special low dimensional cases, such as the diffusion equation with non- smooth conductivity/permeability[177].

3.4.15 New Algorithms and Computational Frameworks

Physics informed ML models often involve training large scale NNs with complicated loss functions, which generally consist of multiple terms and thus are highly non convex optimization problems[178]. The terms in the loss function may compete with each other during training. Consequently, the training process may not be robust and sufficiently stable, and thus convergence to the global minimum cannot be guaranteed[179].

To resolve this issue, one needs to develop more robust NN architectures and training algorithms for diverse applications. For example, refs[76,77,173] have identified two fundamental weaknesses of PINNs, relating spectral bias[170] to a discrepancy in the convergence rate of different components in a PINN loss function. The latter is manifested by training instabilities leading to vanishing back propagated gradients. As discussed in these refs[76,77,173], these pathologies can be mitigated by designing appropriate model architectures and new training algorithms for PINNs. Also, ref.[104] used the weak form of the PDE and hp- refinement via decomposition to enhance the approximation capability of networks.

Other examples include adaptively modifying the activation functions[180] or sampling the data points and the residual evaluation points during training[181], which accelerate convergence and improve the

performance of physics informed models. Moreover, the design of effective NN architectures is currently done empirically by users, which could be very time consuming.
However, emerging meta learning techniques can be used to automate this search[166–168]. What is interesting here is that the architecture may be changing as the bifurcation parameters of the system (such as the Reynolds number) increase. The training and optimization of deep learning models is expensive, and it is crucial to speed up the learning, for instance through transfer learning via *DeepONets* as in the example of crack propagation reported in ref.[182]. In addition, scalable and parallel training algorithms should be developed by using hardware like GPUs and tensor processing units, using both data- parallel and model- parallel algorithms.
Unlike classic classification or regression tasks, where the first order derivative is required for gradient descent, physics- informed ML usually involves higher order derivatives. Currently, their efficient evaluation is not well supported in popular software frameworks such as *TensorFlow* and *PyTorch*. An ML software library that is more efficient for computing high- order derivatives (for example, via Taylor mode automatic differentiation)[183,184] could greatly reduce the computational cost and boost the application of physics- informed ML across different disciplines. In addition to integer order derivatives, other operators such as integral operators and even fractional order derivatives[103] are very useful in physics informed learning.

3.4.16 Data Generation & Benchmarks

In the ML community dealing with imaging, speech and natural language processing problems, the use of standard benchmarks is very common in order to assess algorithm improvement, reproducibility of results, and expected computational cost. The *UCI Machine Learning Repository*[185], which was created over three decades ago, is a collection of databases and data generators that are often used to compare the relative performance of new algorithms. Currently, they also include experimental data sets in the physical sciences, for example noise generated by an aero foil, ocean temperature and current measurements related to El Niño, and hydrodynamic resistance related to different yacht designs.
These data sets are useful and are intended for data driven modelling in ML, but in principle they can also be used for benchmarking physics- informed ML methods, assuming that proper parameterized physical models can be explicitly included in the databases. However, in many different applications in physics and chemistry, full field data are required, which cannot be obtained experimentally (for example in density- functional theory and molecular dynamics simulation or in direct numerical simulations of turbulence), and which tax computational resources heavily both in terms of time and memory. Hence, careful consideration should be given to how to make these data publicly available, how to curate such valuable data, and how to include the physical models and all parameters required for the generation of these databases.
In addition, it will take a concerted effort by researchers to design meaningful benchmarks that test accuracy and speed up of the new proposed physics informed algorithms, which is a non-trivial task. Indeed, even for the aforementioned imaging and other established ML applications, there are still new developments on refining existing benchmarks and metrics, especially if software and hardware considerations are also factored in such evaluations (for example, an in- depth analysis for image recognition)[186]. In physical systems, these difficulties are exacerbated by the fact that the aim is to predict dynamics, and it will be complicated, for example, to determine how to capture or identify bifurcations in dynamical systems and chaotic states. However, new metrics such as the valid- time prediction introduced in ref.[187] may be appropriate and offer a promising direction to follow.

3.4.17 New Mathematics

Despite the empirical success of physics-informed learning models, little is known about the theoretical foundation of such constrained NNs. A new theory is required to rigorously analyses the capabilities and limitations of physics informed learning (for example, the learning capacity of NNs). More specifically, a fundamental question is: can a network find solutions to PDE via gradient- based

optimization? To answer this question, one should analyze the total error in deep learning, which can be decomposed into three types of errors: approximation error (can a network approximate a solution to PDE with any accuracy?), optimization error (can one attain zero or very small training loss?) and generalization error (does smaller training error mean more accurate predicted solution?).

It is important to analyses the well posed-ness of the problem and the stability and convergence in terms of these errors. In particular, if the operator to be solved is (possibly partially) learned by the data themselves, establishing how well posed any problem involving this operator is becomes an exciting mathematical challenge. The challenge is exacerbated when the initial/boundary/internal conditions are provided themselves as (possibly uncertain) data. This well-posedness issue must be analyzed mathematically, aided by ML computational exploration.

The first mathematical analysis for PINNs in solving forward problems appeared in ref.[188], where the Hölder regularization was introduced to control generalization error. Specifically, ref.[188] analyzed the second order linear elliptic and parabolic type PDEs and proved the consistency of results. References[189,190] used quadrature points in the formulation of the loss and provided an abstract error estimate for both forward and inverse problems.

However, no convergence results were reported, as the use of quadrature points does not quantify the generalization error. In subsequent work, ref.[191] studied linear PDEs and proposed an abstract error estimates framework for analyzing both PINNs[7] and variational PINNs[104,192]. Based on the compactness assumptions and the norm equivalence relations, sufficient conditions for convergence to the underlying PDE solution were obtained. The generalization error was handled by the Rademacher complexity. For the continuous loss formulation, refs[49,193–195] derived some error estimates based on the continuous loss formulations of PINNs.

Although known error bounds involved with continuous norms (from PDE literature) may serve as error bounds for (continuous) PINNs, data samples have to be taken into account to quantify the generalization error.

In general, NNs are trained by gradient- based optimization methods, and a new theory should be developed to better understand their training dynamics (gradient descent, stochastic gradient descent, Adam[196] and so on).

In ref.[197], over parameterized two layer networks were analyzed, and it was proved that the convergence of gradient descent for second order linear PDEs, but the boundary conditions were not included in the analysis. In ref.[76], the neural tangent kernel theory[198] was extended to PINNs, and it was shown that the training dynamics of PINNs sometimes can be regarded as a kernel regression as the width of network goes to infinity.

It is also helpful to understand the training process of networks by visualizing the landscape of loss function of different formulations (strong form, weak form and so on). Furthermore, more methods are being rapidly developed nowadays, and thus it is also important to understand the equivalence between models and the equivalence between different loss functions with different norms.

Analyzing the physics- informed ML models based on rigorous theory calls for a fruitful synergy between deep learning, optimization, numerical analysis and PDE theory that not only has the potential to lead to more robust and effective training algorithms, but also to build a solid foundation for this new generation of computational methods.

3.4.18 Outlook

Physics informed learning integrates data and mathematical models seamlessly even in noisy and high dimensional contexts, and can solve general inverse problems very effectively. Here, we have summarized some of the key concepts in **Boxes 1–3** and provided references to frameworks and open- source software for the interested reader to have a head start in exploring physics informed learning. We also discussed current capabilities and limitations and highlighted diverse applications from fluid dynamics to biophysics, plasma physics, transition between metastable states and other

applications in materials. Next, we present possible new directions for applications of physics-informed learning machines as well as research directions that will contribute to their faster training, more accurate predictions, and better interpretability for diverse physics applications and beyond.
Although there have been tools like Tensor Board to visualize the model graph, track the variables and metrics, and so on, for physical problems, extended requirements may include incorporating multiple physics and complicated geometry domain into the learning algorithm, visualizing the solution field (even high dimensional ones), as in traditional computing platforms such as *FEniCS199*, *OpenFOAM10* and others.
A user friendly, graph based ML development environment that can address the above issues could help more practitioners to develop physics- informed ML algorithms for applications to a wide range of diverse physical problems.

3.4.19 Future Directions

3.4.19.1 Digital Twins

'Digital twins', a concept first put forth by General Electric to describe the digital copy of an engine manufactured in their factories, are now becoming a reality in a number of industries. By assimilating real measurements to calibrate computational models, a digital twin aims to replicate the behavior of a living or non- living physical entity in silico. Before these emerging technologies can be translated into practice, a series of fundamental questions need to be addressed. First, observational data can be scarce and noisy, are often characterized by vastly heterogeneous data modalities (images, time series, lab tests, historical data, clinical records and so on), and may not be directly available for certain quantities of interest.
Second, physics based computational models heavily rely on tedious pre-processing and calibration procedures (such as mesh generation or calibration of initial and boundary conditions) that typically have a considerable cost, hampering their use in real- time decision making settings. Moreover, physical models of many complex natural systems are, at best, 'partially' known as conservation laws, and do not provide a closed system of equations unless appropriate constitutive laws are postulated.
Thanks to its natural capability of blending physical models and data as well as the use of automatic differentiation that removes the need for mesh generation, physics- informed learning is well placed to become an enabling catalyst in the emerging era of digital twins.

3.4.19.2 Data and Model Transformations, Fusion And Interpretability

As the interactions between physics based modelling and ML intensify, one will encounter with increasing frequency situations in which different researchers arrive at different data driven models of the same phenomenon, even if they use the same training data (or equally informative data, observed through different sensors). For example, two research groups using the same or equivalent alternative data may end up having differently trained networks (differently learned latent spaces, differently learned operators) even though their predictions are practically indistinguishable on the training set. Recognizing that there is often no unique physical interpretation of an observed phenomenon, here we foresee the importance of building ML based transformations between predictive models, models at different fidelities, and theories, that are one-to-one (transformable, 'dual', calibratable) to each other in a verifiable manner.
Researchers are increasingly discovering such transformations (for example from nonlinear dynamics to the corresponding Koopman model; from a Poisson system to the corresponding Hamiltonian one; from Nesterov iterations to their corresponding ODEs) in a data driven manner. Such transformations will allow data and models to be systematically fused.
Transformations between the ML latent space features and the physically interpretable observables, or ML learned operators and closed- form equations, will obviously bolster the interpretability of the ML model.

Ultimately, one needs to test how far these transformations generalize: for what range of observations an ML model can be mapped to a different ML model, or to a physical model, and what the generalization limit is, beyond which they cannot be transformed or calibrated to each other.

3.4.19.3 Searching For Intrinsic Variables And Emergent, Useful Representations

Most of the current physics informed ML methods follow this paradigm: first define a set of (humanly interpretable) observables/variables; then collect data; formulate the physics completely or incompletely using a 'reasonable' dictionary of operators based on the chosen observables; and finally apply the learning algorithm of choice. An emerging paradigm fueled by advances in ML is to use observations and learning methods to automatically determine good/intrinsic variables and to also find useful or informative physical model formulations. Stepping beyond principal component analysis, manifold learning techniques (from ISO MAP to t- SNE and diffusion maps) and their deep learning counterparts of generative models and (possibly variational) auto encoders are used to embed raw observations in reduced, mathematically useful latent spaces, in which evolution rules can be learned.

Remarkably, these useful representations can go beyond embedding the relevant features, the dependent variables in a PDE. For spatiotemporally disordered data, one can also create ML driven emergent spaces[200,201] in terms of ML learned independent variables: emergent 'spacetimes' in which the model operators will be learned.

The *DARPA* Shredder Challenge[202] of 2011 recreated space by effectively solving puzzles: documents shredded using a variety of paper shredding techniques. Today, disorganized spatiotemporal observations can be embedded in informative 'independent variable' emergent spaces.

For example, evolution operators in the form of PDEs or SODEs will then be learned in terms of these new, emergent space even possibly time independent variables; there is a direct analogy here with the discussion of emergent space time in modern physics[203].

Such new paradigms could play a critical role in design optimization or in building a digital twin for complicated systems, even systems of systems, where humans can hardly write down a neat physical formulation in closed form. Moreover, instead of collecting data from experiments first and then performing the learning algorithm, it becomes important to integrate both in an active learning framework. In this way, a judicious selection of new and informative data can be aided by exploiting the geometry of latent space of the learning algorithms, while the algorithms can gradually improve the choice of latent space descriptors, as well as the mathematical formulation governing the physics, so as to yield realistic predictions as the experiments go on.

Ultimately, the main element we see changing is what we mean by 'understanding'. Up to now, understanding meant that, say, each term in a PDE had a physical or mechanistic interpretation operated on some physically meaningful observables (dependent variables) and also operated in terms of some physically meaningful space/time (independent) variables. Now, it becomes possible to make accurate predictions without understanding' this type of mechanistic understanding and 'is something that may be redefined in the process.

3.4.20 References

1. Hart, J. K. & Martinez, K. Environmental sensor networks: a revolution in the earth system science? *Earth Sci. Rev.* 78, 177–191 (2006).
2. Kurth, T. et al. Exascale deep learning for climate analytics (IEEE, 2018).
3. Reddy, D. S. & Prasad, P. R. C. Prediction of vegetation dynamics using NDVI time series data and LSTM. *Model. Earth Syst. Environ.* 4, 409–419 (2018).
4. Reichstein, M. et al. Deep learning and process understanding for data- driven earth system science. *Nature* 566, 195–204 (2019).
5. Alber, M. et al. Integrating machine learning and multiscale modeling perspectives, challenges, and opportunities in the biological, biomedical, and behavioral sciences. *NPJ Digit. Med.* 2, 1–11 (2019).
6. Iten, R., Metger, T., Wilming, H., Del Rio, L. & Renner, R. Discovering physical concepts with neural

networks. *Phys. Rev. Lett.* 124, 010508 (2020).
7. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Physics- informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* 378, 686–707 (2019).
8. Schmidt, M. & Lipson, H. Distilling free- form natural laws from experimental data. *Science* 324, 81–85 (2009).
9. Brunton, S. L., Proctor, J. L. & Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc. Natl Acad. Sci. USA* 113, 3932–3937 (2016).
10. Jasak, H. et al. OpenFOAM: A C++ library for complex physics simulations. *Int. Workshop Coupled Methods Numer. Dyn.* 1000, 1–20 (2007).
11. Plimpton, S. Fast parallel algorithms for short- range molecular dynamics. *J. Comput. Phys.* 117, 1–19 (1995).
12. Jia, X. et al. Physics- guided machine learning for scientific discovery: an application in simulating lake temperature profiles. Preprint at *arXiv* https://arxiv.org/abs/2001.11086 (2020).
13. Lu, L., Jin, P., Pang, G., Zhang, Z. & Karniadakis, G. E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nat. Mach. Intell.* 3, 218–229 (2021).
14. Kashefi, A., Rempe, D. & Guibas, L. J. A point- cloud deep learning framework for prediction of fluid flow fields on irregular geometries. *Phys. Fluids* 33, 027104 (2021).
15. Li, Z. et al. Fourier neural operator for parametric partial differential equations. in *Int. Conf. Learn. Represent.* (2021).
16. Yang, Y. & Perdikaris, P. Conditional deep surrogate models for stochastic, high- dimensional, and multi- fidelity systems. *Comput. Mech.* 64, 417–434 (2019).
17. LeCun, Y. & Bengio, Y. et al. Convolutional networks for images, speech, and time series. *Handb. Brain Theory Neural Netw.* 3361, 1995 (1995).
18. Mallat, S. Understanding deep convolutional networks. *Phil. Trans. R. Soc. A* 374, (2016).
19. Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A. & Vandergheynst, P. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Process. Mag.* 34, 18–42 (2017).
20. Cohen, T., Weiler, M., Kicanaoglu, B. & Welling, M. Gauge equivariant convolutional networks and the icosahedral CNN. *Proc. Machine Learn. Res.* 97, 1321–1330 (2019).
21. Owhadi, H. Multigrid with rough coefficients and multiresolution operator decomposition from hierarchical information games. *SIAM Rev.* 59, 99–149 (2017).
22. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Inferring solutions of differential equations using noisy multi- fidelity data. *J. Comput. Phys.* 335, 736–746 (2017).
23. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Numerical Gaussian processes for time- dependent and nonlinear partial differential equations. *SIAM J. Sci. Comput.* 40, A172–A198 (2018).
24. Owhadi, H. Bayesian numerical homogenization. *Multiscale Model. Simul.* 13, 812–828 (2015).
25. Hamzi, B. & Owhadi, H. Learning dynamical systems from data: a simple cross- validation perspective, part I: parametric kernel flows. *Physica D* 421, 132817 (2021).
26. Reisert, M. & Burkhardt, H. Learning equivariant functions with matrix valued kernels. *J. Mach. Learn. Res.* 8, 385–408 (2007).
27. Owhadi, H. & Yoo, G. R. Kernel flows: from learning kernels from data into the abyss. *J. Comput. Phys.* 389, 22–47 (2019).
28. Winkens, J., Linmans, J., Veeling, B. S., Cohen, T. S. & Welling, M. Improved semantic segmentation for histopathology using rotation equivariant convolutional networks. in *Conf. Med. Imaging Deep Learn.* (2018).
29. Bruna, J. & Mallat, S. Invariant scattering convolution networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 35, 1872–1886 (2013).
30. Kondor, R., Son, H. T., Pan, H., Anderson, B. & Trivedi, S. Covariant compositional networks for learning graphs. Preprint at *arXiv* https://arxiv.org/abs/1801.02144 (2018).

31. Tai, K. S., Bailis, P. & Valiant, G. Equivariant transformer networks. *Proc. Int. Conf. Mach. Learn.* 97, 6086–6095 (2019).
32. Pfau, D., Spencer, J. S., Matthews, A. G. & Foulkes, W. M. C. Ab initio solution of the many electron Schrödinger equation with deep neural networks. *Phys. Rev. Res.* 2, 033429 (2020).
33. Pun, G. P., Batra, R., Ramprasad, R. & Mishin, Y. Physically informed artificial neural networks for atomistic modeling of materials. *Nat. Commun.* 10, 1–10 (2019).
34. Ling, J., Kurzawski, A. & Templeton, J. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *J. Fluid Mech.* 807, 155–166 (2016).
35. Jin, P., Zhang, Z., Zhu, A., Tang, Y. & Karniadakis, G. E. SympNets: intrinsic structure- preserving symplectic networks for identifying Hamiltonian systems. *Neural Netw.* 132, 166–179 (2020).
36. Lusch, B., Kutz, J. N. & Brunton, S. L. Deep learning for universal linear embeddings of nonlinear dynamics. *Nat. Commun.* 9, 4950 (2018).
37. Lagaris, I. E., Likas, A. & Fotiadis, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* 9, 987–1000 (1998).
38. Sheng, H. & Yang, C. PFNN: A penalty- free neural network method for solving a class of second-order boundary- value problems on complex geometries. *J. Comput. Phys.* 428, 110085 (2021).
39. McFall, K. S. & Mahan, J. R. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transac. Neural Netw.* (2009).
40. Beidokhti, R. S. & Malek, A. Solving initial- boundary value problems for systems of partial differential equations using neural networks and optimization techniques. *J. Franklin Inst.* (2009).
41. Lagari, P. L., Tsoukalas, L. H., Safarkhani, S. & Lagaris, I. E. Systematic construction of neural forms for solving partial differential equations inside rectangular domains, subject to initial, boundary and interface conditions. *Int. J. Artif. Intell. Tools* 29, 2050009 (2020).
42. Zhang, D., Guo, L. & Karniadakis, G. E. Learning in modal space: solving time- dependent stochastic PDEs using physics- informed neural networks. *SIAM J. Sci. Comput.* (2020).
43. Dong, S. & Ni, N. A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks. *J. Comput. Phys*. 435, 110242 (2021).
44. Wang, B., Zhang, W. & Cai, W. Multi- scale deep neural network (MscaleDNN) methods for oscillatory stokes flows in complex domains. *Commun. Comput. Phys.* 28, 2139–2157 (2020).
45. Liu, Z., Cai, W. & Xu, Z. Q. J. Multi- scale deep neural network (MscaleDNN) for solving Poisson–Boltzmann equation in complex domains. *Commun. Comput. Phys.* 28, 1970–2001 (2020).
46. Mattheakis, M., Protopapas, P., Sondak, D., Di Giovanni, M. & Kaxiras, E. Physical symmetries embedded in neural networks. Preprint at *arXiv* https://arxiv.org/abs/1904.08991 (2019).
47. Cai, W., Li, X. & Liu, L. A phase shift deep neural network for high frequency approximation and wave problems. *SIAM J. Sci. Comput.* 42, A3285–A3312 (2020).
48. Darbon, J. & Meng, T. On some neural network architectures that can represent viscosity solutions of certain high dimensional Hamilton- Jacobi partial differential equations. *J. Comput. Phys.* (2021).
49. Sirignano, J. & Spiliopoulos, K. DGM: a deep learning algorithm for solving partial differential equations. *J. Comput. Phys.* 375, 1339–1364 (2018).
50. Kissas, G. et al. Machine learning in cardiovascular flows modeling: predicting arterial blood pressure from non- invasive 4D flow MRI data using physics informed neural networks. *Comput. Methods Appl. Mech. Eng.* 358, 112623 (2020).
51. Zhu, Y., Zabaras, N., Koutsourelakis, P. S. & Perdikaris, P. Physics- constrained deep learning for high- dimensional surrogate modeling and uncertainty quantification without labeled data. *J. Comput. Phys.* 394, 56–81 (2019).
52. Geneva, N. & Zabaras, N. Modeling the dynamics of PDE systems with physics- constrained deep auto- regressive networks. *J. Comput. Phys.* 403, 109056 (2020).
53. Wu, J. L. et al. Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems. *J. Comput. Phys.* 406, 109209 (2020).

54. Pfrommer, S., Halm, M. & Posa, M. Contactnets: learning of discontinuous contact dynamics with smooth, implicit representations. Preprint at *arXiv* https://arxiv.org/abs/2009.11193 (2020).
55. Erichson, N.B., Muehlebach, M. & Mahoney, M. W. Physics- informed autoencoders for Lyapunov-stable fluid flow prediction. Preprint at *arXiv* https://arxiv.org/abs/1905.10866 (2019).
56. Shah, V. et al. Encoding invariances in deep generative models. Preprint at *arXiv* https://arxiv.org/abs/1906.01626 (2019).
57. Geneva, N. & Zabaras, N. Transformers for modeling physical systems. Preprint at *arXiv* https://arxiv.org/abs/2010.03957 (2020).
58. Li, Z. et al. Multipole graph neural operator for parametric partial differential equations. in *Adv. Neural Inf. Process. Syst.* (2020).
59. Nelsen, N. H. & Stuart, A. M. The random feature model for input–output maps between Banach spaces. Preprint at *arXiv* https://arxiv.org/abs/2005.10224 (2020).
60. Cai, S., Wang, Z., Lu, L., Zaki, T. A. & Karniadakis, G. E. DeepM&Mnet: inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *J. Comput. Phys*. 436, 110296 (2020).
61. Mao, Z., Lu, L., Marxen, O., Zaki, T. A. &Karniadakis, G. E. DeepM&Mnet for hypersonics: predicting the coupled flow and finite- rate chemistry behind a normal shock using neural- network approximation of operators. Preprint at *arXiv* https://arxiv.org/abs/2011.03349 (2020).
62. Meng, X. & Karniadakis, G. E. A composite neural network that learns from multi- fidelity data: application to function approximation and inverse PDE problems. *J. Comput. Phys.* 401, (2020).
63. Sirignano, J., MacArt, J. F. & Freund, J. B. DPM: a deep learning PDE augmentation method with application to large- eddy simulation. *J. Comput. Phys.* 423, 109811 (2020).
64. Lu, L. et al. Extraction of mechanical properties of materials through deep learning from instrumented indentation. *Proc. Natl Acad. Sci. USA* 117, 7052–7062 (2020).
65. Reyes, B., Howard, A. A., Perdikaris, P. & Tartakovsky, A. M. Learning unknown physics of non- Newtonian fluids. Preprint at *arXiv* https://arxiv.org/abs/2009.01658 (2020).
66. Wang, W. & Gómez- Bombarelli, R. Coarse- graining auto- encoders for molecular dynamics. *NPJ Comput. Mater.* 5, 1–9 (2019).
67. Rico- Martinez, R., Anderson, J. & Kevrekidis, I. Continuous- time nonlinear signal processing: a neural network based approach for gray box identification (IEEE, 1994).
68. Xu, K., Huang, D. Z. & Darve, E. Learning constitutive relations using symmetric positive definite neural networks. Preprint at *arXiv* https://arxiv.org/abs/2004.00265 (2020).
69. Huang, D. Z., Xu, K., Farhat, C. & Darve, E. Predictive modeling with learned constitutive laws from indirect observations. Preprint at *arXiv* https://arxiv.org/abs/1905.12530 (2019).
70. Xu, K., Tartakovsky, A. M., Burghardt, J. & Darve, E. Inverse modeling of viscoelasticity materials using physics constrained learning. Preprint at *arXiv* https://arxiv.org/abs/2005.04384 (2020).
71. Li, D., Xu, K., Harris, J. M. & Darve, E. Coupled time- lapse full- waveform inversion for subsurface flow problems using intrusive automatic differentiation. *Water Resour. Res.* 56, (2020).
72. Tartakovsky, A., Marrero, C. O., Perdikaris, P., Tartakovsky, G. & Barajas- Solano, D. Physics-informed deep neural networks for learning parameters and constitutive relationships in subsurface flow problems. *Water Resour. Res.* 56, e2019WR026731 (2020).
73. Xu, K. & Darve, E. Adversarial numerical analysis for inverse problems. Preprint at *arXiv* https://arxiv.org/abs/1910.06936 (2019).
74. Yang, Y., Bhouri, M. A. & Perdikaris, P. Bayesian differential programming for robust systems identification under uncertainty. *Proc. R. Soc. A* 476, 20200290 (2020).
75. Rackauckas, C. et al. Universal differential equations for scientific machine learning. Preprint at *arXiv* https://arxiv.org/abs/2001.04385 (2020).
76. Wang, S., Yu, X. & Perdikaris, P. When and why PINNs fail to train: a neural tangent kernel perspective. Preprint at *arXiv* https://arxiv.org/abs/2007.14527 (2020).

77. Wang, S., Wang, H. & Perdikaris, P. On the eigenvector bias of Fourier feature networks: from regression to solving multi- scale PDEs with physics- informed neural networks. Preprint at *arXiv* https://arxiv.org/abs/ 2012.10047 (2020).
78. Pang, G., Yang, L. & Karniadakis, G. E. Neural- net-induced Gaussian process regression for function approximation and PDE solution. *J. Comput. Phys.* 384, 270–288 (2019).
79. Wilson, A. G., Hu, Z., Salakhutdinov, R. & Xing, E. P. Deep kernel learning. *Proc. Int. Conf. Artif. Intell. Stat.* 51, 370–378 (2016).
80. Owhadi, H. Do ideas have shape? Plato's theory of forms as the continuous limit of artificial neural networks. Preprint at *arXiv* https://arxiv.org/abs/2008.03920 (2020).
81. Owhadi, H. & Scovel, C. *Operator- Adapted Wavelets, Fast Solvers, and Numerical Homogenization: From a Game Theoretic Approach to Numerical Approximation and Algorithm Design* (Cambridge Univ. Press, 2019).
82. Micchelli, C. A. & Rivlin, T. J. in *Optimal Estimation in Approximation Theory* (eds. Micchelli, C. A. & Rivlin, T. J.) 1–54 (Springer, 1977).
83. Sard, A. *Linear Approximation* (Mathematical Surveys 9, American Mathematical Society, 1963).
84. Larkin, F. Gaussian measure in Hilbert space and applications in numerical analysis. *Rocky Mt. J. Math.* 2, 379–421 (1972).
85. Sul'din, A. V. Wiener measure and its applications to approximation methods. I. *Izv. Vyssh. Uchebn. Zaved. Mat.* 3, 145–158 (1959).
86. Diaconis, P. Bayesian numerical analysis. *Stat. Decision Theory Relat. Top. IV* 1, 163–175 (1988).
87. Kimeldorf, G. S. & Wahba, G. A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. *Ann. Math. Stat.* 41, 495–502 (1970).
88. Owhadi, H., Scovel, C. & Schäfer, F. Statistical numerical approximation. *Not. Am. Math. Soc.* 66, 1608–1617 (2019).
89. Tsai, Y. H. H., Bai, S., Yamada, M., Morency, L. P. & Salakhutdinov, R. Transformer dissection: a unified understanding of transformer's attention via the lens of kernel. Preprint at *arXiv* https://arxiv.org/abs/ 1908.11775 (2019).
90. Kadri, H. et al. Operator- valued kernels for learning from functional response data. *J. Mach. Learn. Res.* 17, 1–54 (2016).
91. González- García, R., Rico- Martínez, R. & Kevrekidis, I. G. Identification of distributed parameter systems: a neural net based approach. *Comput. Chem. Eng.* 22, S965–S968 (1998).
92. Long, Z., Lu, Y., Ma, X. & Dong, B. PDE- Net: learning PDEs from data. *Proc. Int. Conf. Mach. Learn.* 80, 3208–3216 (2018).
93. He, J. & Xu, J. MgNet: a unified framework of multigrid and convolutional neural network. *Sci. China Math.* 62, 1331–1354 (2019).
94. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition (IEEE, 2016).
95. Rico- Martinez, R., Krischer, K., Kevrekidis, I., Kube, M. & Hudson, J. Discrete- vs. continuous- time nonlinear signal processing of Cu electro dissolution data. *Chem. Eng. Commun.* 118, 25–48 (1992).
96. Weinan, E. A proposal on machine learning via dynamical systems. *Commun. Math. Stat.* (2017).
97. Chen, T. Q., Rubanova, Y., Bettencourt, J. & Duvenaud, D. K. Neural ordinary differential equations. *Adv. Neural Inf. Process. Syst.* 31, 6571–6583 (2018).
98. Jia, J. & Benson, A. R. Neural jump stochastic differential equations. *Adv. Neural Inf. Process. Syst.* 32, 9847–9858 (2019).
99. Rico- Martinez, R., Kevrekidis, I. & Krischer, K. in *Neural Networks for Chemical Engineers* (ed. Bulsari, A. B.) 409–442 (Elsevier, 1995).
100. He, J., Li, L., Xu, J. & Zheng, C. ReLU deep neural networks and linear finite elements. *J. Comput. Math.* 38, 502–527 (2020).
101. Jagtap, A. D., Kharazmi, E. & Karniadakis, G. E. Conservative physics- informed neural networks on discrete domains for conservation laws: applications to forward and inverse problems. *Comput. Methods Appl. Mech. Eng.* 365, 113028 (2020).

102. Yang, L., Zhang, D. & Karniadakis, G. E. Physics- informed generative adversarial networks for stochastic differential equations. *SIAM J. Sci. Comput.* 42, A292–A317 (2020).
103. Pang, G., Lu, L. & Karniadakis, G. E. fPINNs: fractional physics- informed neural networks. *SIAM J. Sci. Comput.* 41, A2603–A2626 (2019).
104. Kharazmi, E., Zhang, Z. & Karniadakis, G. E. hp- VPINNs: variational physics- informed neural networks with domain decomposition. *Comput. Methods Appl. Mech. Eng.* 374, 113547 (2021).
105. Jagtap, A. D. & Karniadakis, G. E. Extended physics informed neural networks (XPINNs): a generalized space- time domain decomposition based deep learning framework for nonlinear partial differential equations. *Commun. Comput. Phys.* 28, 2002–2041 (2020).
106. Raissi, M., Yazdani, A. & Karniadakis, G. E. Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations. *Science* 367, 1026–1030 (2020).
107. Yang, L., Meng, X. & Karniadakis, G. E. B- PINNs: Bayesian physics- informed neural networks for forward and inverse PDE problems with noisy data. *J. Comput. Phys.* 415, 109913 (2021).
108. Wang, S. & Perdikaris, P. Deep learning of free boundary and Stefan problems. *J. Comput. Phys.* 428, 109914 (2020).
109. Spigler, S. et al. A jamming transition from under- to over- parametrization affects generalization in deep learning. *J. Phys. A* 52, 474001 (2019).
110. Geiger, M. et al. Scaling description of generalization with number of parameters in deep learning. *J. Stat. Mech. Theory Exp.* 2020, 023401 (2020).
111. Belkin, M., Hsu, D., Ma, S. & Mandal, S. Reconciling modern machine- learning practice and the classical bias–variance trade- off. *Proc. Natl Acad. Sci. USA* 116, 15849–15854 (2019).
112. Geiger, M. et al. Jamming transition as a paradigm to understand the loss landscape of deep neural networks. *Phys. Rev. E* 100, 012115 (2019).
113. Mei, S., Montanari, A. & Nguyen, P. M. A mean field view of the landscape of two- layer neural networks. *Proc. Natl Acad. Sci. USA* 115, E7665–E7671 (2018).
114. Mehta, P. & Schwab, D. J. An exact mapping between the variational renormalization group and deep learning. Preprint at *arXiv* https://arxiv.org/abs/1410.3831 (2014).
115. Stoudenmire, E. & Schwab, D. J. Supervised learning with tensor networks. *Adv. Neural Inf. Process. Syst.* 29, 4799–4807 (2016).
116. Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B. & LeCun, Y. The loss surfaces of multilayer networks. *Proc. Artif. Intell. Stat.* 38, 192–204 (2015).
117. Poole, B., Lahiri, S., Raghu, M., Sohl- Dickstein, J. & Ganguli, S. Exponential expressivity in deep neural networks through transient chaos. *Adv. Neural Inf. Process. Syst.* 29, 3360–3368 (2016).
118. Yang, G. & Schoenholz, S. Mean field residual networks: on the edge of chaos. *Adv. Neural Inf. Process. Syst.* 30, 7103–7114 (2017).
119. Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B. & Liao, Q. Why and when can deep but not shallow networks avoid the curse of dimensionality: a review. *Int. J. Autom. Comput.* 14, 503–519 (2017).
120. Grohs, P., Hornung, F., Jentzen, A. & Von Wurstemberger, P. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations. Preprint at *arXiv* https://arxiv.org/abs/1809.02362 (2018).
121. Han, J., Jentzen, A. & Weinan, E. Solving high dimensional partial differential equations using deep learning. *Proc. Natl Acad. Sci. USA* 115, 8505–8510 (2018).
122. Goodfellow, I. et al. Generative adversarial nets. *Adv. Neural Inf. Process. Syst.* 27, (2014).
123. Brock, A., Donahue, J. & Simonyan, K. Large scale GAN training for high fidelity natural image synthesis. in *Int. Conf. Learn. Represent.* (2019).
124. Yu, L., Zhang, W., Wang, J. & Yu, Y. SeqGAN: sequence generative adversarial nets with policy gradient (AAAI Press, 2017).
125. Zhu, J.Y., Park, T., Isola, P. & Efros, A. A. Unpaired image- to-image translation using cycle-consistent adversarial networks (IEEE, 2017).

126. Yang, L., Daskalakis, C. & Karniadakis, G. E. Generative ensemble- regression: learning particle dynamics from observations of ensembles with physics- informed deep generative models. Preprint at *arXiv* https://arxiv.org/abs/2008.01915 (2020).
127. Lanthaler, S., Mishra, S. & Karniadakis, G. E. Error estimates for DeepONets: a deep learning framework in infinite dimensions. Preprint at *arXiv* https://arxiv.org/abs/2102.09618 (2021).
128. Deng, B., Shin, Y., Lu, L., Zhang, Z. & Karniadakis, G. E. Convergence rate of DeepONets for learning operators arising from advection–diffusion equations. Preprint at *arXiv* https://arxiv.org/abs/2102.10621 (2021).
129. Xiu, D. & Karniadakis, G. E. The Wiener–Askey polynomial chaos for stochastic differential equations. *SIAM J. Sci. Comput.* 24, 619–644 (2002).
130. Marzouk, Y. M., Najm, H. N. & Rahn, L. A. Stochastic spectral methods for efficient Bayesian solution of inverse problems. *J. Comput. Phys.* 224, 560–586 (2007).
131. Stuart, A. M. Inverse problems: a Bayesian perspective. *Acta Numerica* 19, 451 (2010).
132. R. K. & Bilionis, I. Deep UQ: learning deep neural network surrogate models for high dimensional uncertainty quantification. *J. Comput. Phys.* 375, 565–588 (2018).
133. Karumuri, S., Tripathy, R., Bilionis, I. & Panchal, J. Simulator- free solution of high- dimensional stochastic elliptic partial differential equations using deep neural networks. *J. Comput. Phys.* (2020).
134. Yang, Y. & Perdikaris, P. Adversarial uncertainty quantification in physics- informed neural networks. *J. Comput. Phys.* 394, 136–152 (2019).
135. Raissi, M., Perdikaris, P. & Karniadakis, G. E. Machine learning of linear differential equations using Gaussian processes. *J. Comput. Phys.* 348, 683–693 (2017).
136. Fan, D. et al. A robotic intelligent towing tank for learning complex fluid- structure dynamics. *Sci. Robotics* 4, eaay5063 (2019).
137. Winovich, N., Ramani, K. & Lin, G. ConvPDE-UQ: convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *J. Comput. Phys.* 394, 263–279 (2019).
138. Zhang, D., Lu, L., Guo, L. & Karniadakis, G. E. Quantifying total uncertainty in physics- informed neural networks for solving forward and inverse stochastic problems. *J. Comput. Phys.* 397, (2019).
139. Gal, Y. & Ghahramani, Z. Dropout as a Bayesian approximation: representing model uncertainty in deep learning. *Proc. Int. Conf. Mach. Learn.* 48, 1050–1059 (2016).
140. Cai, S. et al. Flow over an espresso cup: inferring 3-D velocity and pressure fields from tomographic background oriented Schlieren via physics- informed neural networks. *J. Fluid Mech*. 915 (2021).
141. Mathews, A., Francisquez, M., Hughes, J. & Hatch, D. Uncovering edge plasma dynamics via deep learning from partial observations. Preprint at *arXiv* https://arxiv.org/abs/2009.05005 (2020).
142. Rotskoff, G. M. & Vanden- Eijnden, E. Learning with rare data: using active importance sampling to optimize objectives dominated by rare events. Preprint at *arXiv* https://arxiv.org/abs/2008.06334 (2020).
143. Patel, R. G. et al. Thermodynamically consistent physics informed neural networks for hyperbolic systems. Preprint at https://arxiv.org/abs/2012.05343 (2020).
144. Shukla, K., Di Leoni, P. C., Blackshire, J., Sparkman, D. & Karniadakis, G. E. Physics- informed neural network for ultrasound nondestructive quantification of surface breaking cracks. *J. Nondestruct. Eval.* 39, 1–20 (2020).
145. Behler, J. & Parrinello, M. Generalized neural network representation of high dimensional potential energy surfaces. *Phys. Rev. Lett.* 98, 146401 (2007).
146. Zhang, L., Han, J., Wang, H., Car, R. & Weinan, E. Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics. *Phys. Rev. Lett.* 120, 143001 (2018).
147. Jia, W. et al. Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning. Preprint at *arXiv* https://arxiv.org/abs/2005.00223 (2020).
148. Nakata, A. et al. Large scale and linear scaling DFT with the CONQUEST code. *J. Chem. Phys.* 152,

164112 (2020).
149. Zhu, W., Xu, K., Darve, E. & Beroza, G. C. A general approach to seismic inversion with automatic differentiation. Preprint at *arXiv* https://arxiv.org/abs/2003.06027 (2020).
150. Abadi, M. et al. Tensorflow: a system for large- scale machine learning. *Proc. OSDI* 16, 265–283 (2016).
151. Paszke, A. et al. PyTorch: an imperative style, high performance deep learning library. *Adv. Neural Inf. Process. Syst.* 32, 8026–8037 (2019).
152. Chollet, F. et al. Keras — Deep learning library. *Keras* https://keras.io (2015).
153. Frostig, R., Johnson, M. J. & Leary, C. Compiling machine learning programs via high level tracing. in *Syst. Mach. Learn.* (2018).
154. Lu, L., Meng, X., Mao, Z. & Karniadakis, G. E. DeepXDE: a deep learning library for solving differential equations. *SIAM Rev.* 63, 208–228 (2021).
155. Hennigh, O. et al. NVIDIA SimNet: an AI- accelerated multi physics simulation framework. Preprint at *arXiv* https://arxiv.org/abs/2012.07938 (2020).
156. Koryagin, A., Khudorozkov, R. & Tsimfer, S. PyDEns: a Python framework for solving differential equations with neural networks. Preprint at *arXiv* https://arxiv.org/abs/1909.11544 (2019).
157. Chen, F. et al. NeuroDiffEq: A python package for solving differential equations with neural networks. *J. Open Source Softw.* 5, 1931 (2020).
158. Rackauckas, C. & Nie, Q. DifferentialEquations.jl a performant and feature- rich ecosystem for solving differential equations in Julia. *J. Open Res. Softw.* 5, 15 (2017).
159. Haghighat, E. & Juanes, R. SciANN: a Keras/TensorFlow wrapper for scientific computations and physics- informed deep learning using artificial neural networks. *Comput. Meth. Appl. Mech. Eng.* 373, 113552 (2020).
160. Xu, K. & Darve, E. ADCME: Learning spatially- varying physical fields using deep neural networks. Preprint at *arXiv* https://arxiv.org/abs/2011.11955 (2020).
161. Gardner, J. R., Pleiss, G., Bindel, D., Weinberger, K. Q. & Wilson, A. G. Gpytorch: black box matrix–matrix Gaussian process inference with GPU acceleration. *Adv. Neural Inf. Process. Syst.* 31, 7587–7597 (2018).
162. Novak, R. et al. Neural Tangents: fast and easy infinite neural networks in Python. in *Conf. Neural Inform. Process. Syst.* (2020).
163. Xu, K. & Darve, E. Physics constrained learning for data- driven inverse modeling from sparse observations. Preprint at *arXiv* https://arxiv.org/abs/2002.10521 (2020).
164. Xu, K. & Darve, E. The neural network approach to inverse problems in differential equations. Preprint at *arXiv* https://arxiv.org/abs/1901.07758 (2019).
165. Xu, K., Zhu, W. & Darve, E. Distributed machine learning for computational engineering using MPI. Preprint at *arXiv* https://arxiv.org/abs/2011.01349 (2020).
166. Elsken, T., Metzen, J. H. & Hutter, F. Neural architecture search: a survey. *J. Mach. Learn. Res.* 20, 1–21 (2019).
167. He, X., Zhao, K. & Chu, X. AutoML: a survey of the state- of-the- art. *Knowl. Based Syst.* 212, 106622 (2021).
168. Hospedales, T., Antoniou, A., Micaelli, P. & Storkey, A. Meta- learning in neural networks: a survey. Preprint at *arXiv* https://arxiv.org/abs/2004.05439 (2020).
169. Xu, Z.-Q. J., Zhang, Y., Luo, T., Xiao, Y. & Ma, Z. Frequency principle: Fourier analysis sheds light on deep neural networks. *Commun. Comput. Phys.* 28, 1746–1767 (2020).
170. Rahaman, N. et al. On the spectral bias of neural networks. *Proc. Int. Conf. Mach. Learn.* 97, 5301–5310 (2019).
171. Ronen, B., Jacobs, D., Kasten, Y. & Kritchman, S. The convergence rate of neural networks for learned functions of different frequencies. *Adv. Neural Inf. Process. Syst.* 32, 4761–4771 (2019).
172. Cao, Y., Fang, Z., Wu, Y., Zhou, D. X. & Gu, Q. Towards understanding the spectral bias of deep learning. Preprint at *arXiv* https://arxiv.org/abs/1912.01198 (2019).

173. Wang, S., Teng, Y. & Perdikaris, P. Understanding and mitigating gradient pathologies in physics-informed neural networks. Preprint at *arXiv* https://arxiv.org/abs/2001.04536 (2020).
174. Tancik, M. et al. Fourier features let networks learn high frequency functions in low dimensional domains. *Adv. Neural Inf. Process. Syst.* 33 (2020).
175. Cai, W. & Xu, Z. Q. J. Multi- scale deep neural networks for solving high dimensional PDEs. Preprint at *arXiv* https://arxiv.org/abs/1910.11710 (2019).
176. Arbabi, H., Bunder, J. E., Samaey, G., Roberts, A. J. & Kevrekidis, I. G. Linking machine learning with multiscale numeric: data- driven discovery of homogenized equations. *JOM* 72, (2020).
177. Owhadi, H. & Zhang, L. Metric- based upscaling. *Commun. Pure Appl. Math.* 60, 675–723 (2007).
178. Blum, A. L. & Rivest, R. L. Training a 3-node neural network is NP- complete. *Neural Netw.* 5, 117–127 (1992).
179. Lee, J. D., Simchowitz, M., Jordan, M. I. & Recht, B. Gradient descent only converges to minimizers. *Annu. Conf. Learn. Theory* 49, 1246–1257 (2016).
180. Jagtap, A. D., Kawaguchi, K. & Em Karniadakis, G. Locally adaptive activation functions with slope recovery for deep and physics- informed neural networks. *Proc. R. Soc. A* 476, 20200334 (2020).
181. Wight, C. L. & Zhao, J. Solving Allen–Cahn and Cahn–Hilliard equations using the adaptive physics informed neural networks. Preprint at *arXiv* https://arXiv.org/abs/2007.04542 (2020).
182. Goswami, S., Anitescu, C., Chakraborty, S. & Rabczuk, T. Transfer learning enhanced physics informed neural network for phase- field modeling of fracture. *Theor. Appl. Fract. Mech.* (2020).
183. Betancourt, M. A geometric theory of higher order automatic differentiation. Preprint at *arXiv* https://arxiv.org/abs/1812.11592 (2018).
184. Bettencourt, J., Johnson, M. J. & Duvenaud, D. Taylor- mode automatic differentiation for higher-order derivatives in JAX. in *Conf. Neural Inform. Process. Syst.* (2019).
185. Newman, D, Hettich, S., Blake, C. & Merz, C. UCI repository of machine learning databases. *ICS* http://www.ics.uci.edu/~mlearn/MLRepository.html (1998).
186. Bianco, S., Cadene, R., Celona, L. & Napoletano, P. Benchmark analysis of representative deep neural network architectures. *IEEE Access* 6, 64270–64277 (2018).
187. Vlachas, P. R. et al. Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks* (2020).
188. Shin, Y., Darbon, J. & Karniadakis, G. E. On the convergence of physics informed neural networks for linear second- order elliptic and parabolic type PDEs. *Commun. Comput. Phys.* 28, 2042–2074 (2020).
189. Mishra, S. & Molinaro, R. Estimates on the generalization error of physics informed neural networks (PINNs) for approximating PDEs. Preprint at *arXiv* https://arxiv.org/abs/2006.16144 (2020).
190. Mishra, S. & Molinaro, R. Estimates on the generalization error of physics informed neural networks (PINNs) for approximating PDEs II: a class of inverse problems. Preprint at *arXiv* https://arxiv.org/abs/2007.01138 (2020).
191. Shin, Y., Zhang, Z. & Karniadakis, G.E. Error estimates of residual minimization using neural networks for linear PDEs. Preprint at *arXiv* https://arxiv.org/abs /2010.08019 (2020).
192. Kharazmi, E., Zhang, Z. & Karniadakis, G. Variational physics- informed neural networks for solving partial differential equations. Preprint at *arXiv* https://arxiv.org/abs/1912.00873 (2019).
193. Jo, H., Son, H., Hwang, H. Y. & Kim, E. Deep neural network approach to forward inverse problems. *Netw. Heterog. Media* 15, 247–259 (2020).
194. Guo, M. & Haghighat, E. An energy- based error bound of physics- informed neural network solutions in elasticity. Preprint at *arXiv* https://arxiv.org/abs/2010.09088 (2020).
195. Lee, J. Y., Jang, J. W. & Hwang, H. J. The model reduction of the Vlasov–Poisson–Fokker–Planck system to the Poisson–Nernst–Planck system via the deep neural network approach. Preprint at *arXiv* https://arxiv.org/abs/2009.13280 (2020).

196. Kingma, D. P. & Ba, J. Adam: a method for stochastic optimization. in *Int. Conf. Learn. Represent.* (2015).
197. Luo, T. & Yang, H. Two- layer neural networks for partial differential equations: optimization and generalization theory. Preprint at *arXiv* https://arxiv.org/abs/2006.15733 (2020).
198. Jacot, A., Gabriel, F. & Hongler, C. Neural tangent kernel: convergence and generalization in neural networks. *Adv. Neural Inf. Process. Syst.* 31, 8571–8580 (2018).
199. Alnæs, M. et al. The FEniCS project version 1.5. *Arch. Numer. Softw.* 3, 9–23 (2015).
200. Kemeth, F. P. et al. An emergent space for distributed data with hidden internal order through manifold learning. *IEEE Access* 6, 77402–77413 (2018).
201. Kemeth, F. P. et al. Learning emergent PDEs in a learned emergent space. Preprint at *arXiv* https://arxiv.org/abs/2012.12738 (2020).
202. Defense Advanced Research Projects Agency. DARPA shredder challenge rules. *DARPA* https://web.archive.org/web/20130221190250/http://archive.darpa.mil/shredderchallenge/Rules.aspx (2011).
203. Rovelli, C. Forget time. *Found. Phys.* 41, 1475 (2011).
204. Hy, T. S., Trivedi, S., Pan, H., Anderson, B. M. & Kondor, R. Predicting molecular properties with covariant compositional networks. *J. Chem. Phys.* 148, 241745 (2018).
205. Hachmann, J. et al. The Harvard clean energy project: large- scale computational screening and design of organic photovoltaics on the world community grid. *J. Phys. Chem. Lett.* 2, 2241–2251 (2011).
206. Byrd, R. H., Lu, P., Nocedal, J. & Zhu, C. A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.* 16, 1190–1208 (1995).

Acknowledgements
We thank H. Owhadi (Caltech) for his insightful comments on the connections between NNs and kernel methods. G.E.K. acknowledges support from the DOE PhILMs project (no. DE- SC0019453) and OSD/AFOSR MURI grant FA9550-20-1-0358. I.G.K. acknowledges support from DARPA (PAI and ATLAS programmers) as well as an AFOSR MURI grant through UCSB. P.P. acknowledges support from the DARPA PAI programme (grant HR00111890034), the US Department of Energy (grant DE- SC0019116), the Air Force Office of Scientific Research (grant FA9550-20-1-0060), and DOE- ARPA (grant 1256545).

Author Contributions
Authors are listed in alphabetical order. G.E.K. supervised the project. All authors contributed equally to writing the paper.

Competing Interests
The authors declare no competing interests.

Peer review Information
Nature Reviews Physics thanks the anonymous reviewers for their contribution to the peer review of this work.

Publisher's note
Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations. Related links
ADCMe: https://kailaix.github.io/ADCME.jl/latest
DeepXDe: https://deepxde.readthedocs.io/
GPyTorch: https://gpytorch.ai/
NeuroDiffeq: https://github.com/NeuroDiffGym/neurodiffeq
NeuralPDe: https://neuralpde.sciml.ai/dev/
Neural Tangents: https://github.com/google/neural- tangents
PyDens: https://github.com/analysiscenter/pydens

PyTorch: https://pytorch.org
sciANN: https://www.sciann.com/
simNet: https://developer.nvidia.com/simnet
TensorFlow: www.tensorflow.org

3.5 Case Study 5 - Classification of Machine Learning (ML) Frameworks for Data-Driven Thermal Fluid Models

Authors : *Chih-Wei Chang and Nam T. Dinh*
Affiliations : *Department of Nuclear Engineering North Carolina State University, Raleigh, NC.*
Title of Paper : *Classification of Machine Learning Frameworks for Data-Driven Thermal Fluid Models*
Citation : *(Chang & Dinh, 2019)*
Bibliography : *Chang , C.-W., & Dinh, N. T. (2019). Classification of Machine Learning Frameworks for Data-Driven Thermal Fluid Models. arXiv:1801.06621 [physics.flu-dyn]*

We focus on data-driven ***Thermal Fluid Simulation (TFS)***, specifically on their development using ***Machine Learning (ML)***. Five ML frameworks are introduced by (Chang & Dinh, 2019) including

1. Physics-Separated ML (PSML or ***Type-I***),
2. Physics-Evaluated ML (PEML or ***Type-II***),
3. Physics-Integrated ML (PIML or ***Type-III***),
4. Physics-Recovered ML (PRML or ***Type-IV***),
5. Physics-Discovered ML (PDML or ***Type-V***).

The frameworks vary in their performance for different applications depending on the level of knowledge of governing physics, source, type, amount and quality of available data for training. Notably, outlined for the first time in this investigation, ***Type-III*** models present stringent requirements on modeling, substantial computing resources for training, and high potential in extracting value from "big data" in thermal fluid research. The current investigation demonstrates and explores ML frameworks example such as the heat diffusion equation with a nonlinear conductivity model, formulated by ***convolutional neural networks (CNNs)*** and ***feed forward neural networks (FNNs)***. To illustrate the applications of ***Type-I, Type-II, Type-III***, and ***Type-V*** ML. The results indicate a preference for ***Type-II*** ML under deficient data support. ***Type-III*** ML can effectively utilize field data, potentially generating more robust predictions than ***Type-I*** and ***Type-II*** ML. CNN-based closures exhibit more predictability than FNN-based closures, but CNN-based closures require more training data to obtain accurate predictions. Second, we illustrate how to employ ***Type-I*** ML and ***Type-II*** ML frameworks for data-driven turbulence modeling using reference works. Third, we demonstrate ***Type-I*** ML by building a deep FNN-based slip closure for two-phase flow modeling. The results show that deep FNN-based closures exhibit a bounded error in the prediction domain.

3.5.1 Machine Learning (ML) for Thermal Fluid Simulation

Machine learning (ML) can be used to develop closure models by learning from the available, relevant, and adequately evaluated data[78] with nonparametric models. While the concept of ML is not new, the past decade has witnessed a significant growth of capability and interest in machine learning thanking advances in algorithms, computing power, affordable memory, and abundance of data. There is a wide range of applications of machine learning in different areas of engineering practice. In a narrow context of the present study, the machine learning is defined as the capability to create effective surrogates for a massive amount of data from measurements and simulations. **Figure 3.5.1** depicts a workflow of employing ML for developing thermal fluid closures. The objective is to construct a function to represent the unknown model that correlates inputs and targets. Since the supervised learning is interested, inputs and targets are essential that can be obtained from ARAED.

[78] Thermal fluid simulations involve conservation equations with various degrees of averaging from the first principle based on distinct hypotheses. The underlying physics of the conservation equations should be consistent with the experiment or simulation where the Available, Relevant, and Adequately Evaluated Data (ARAED) are obtained.

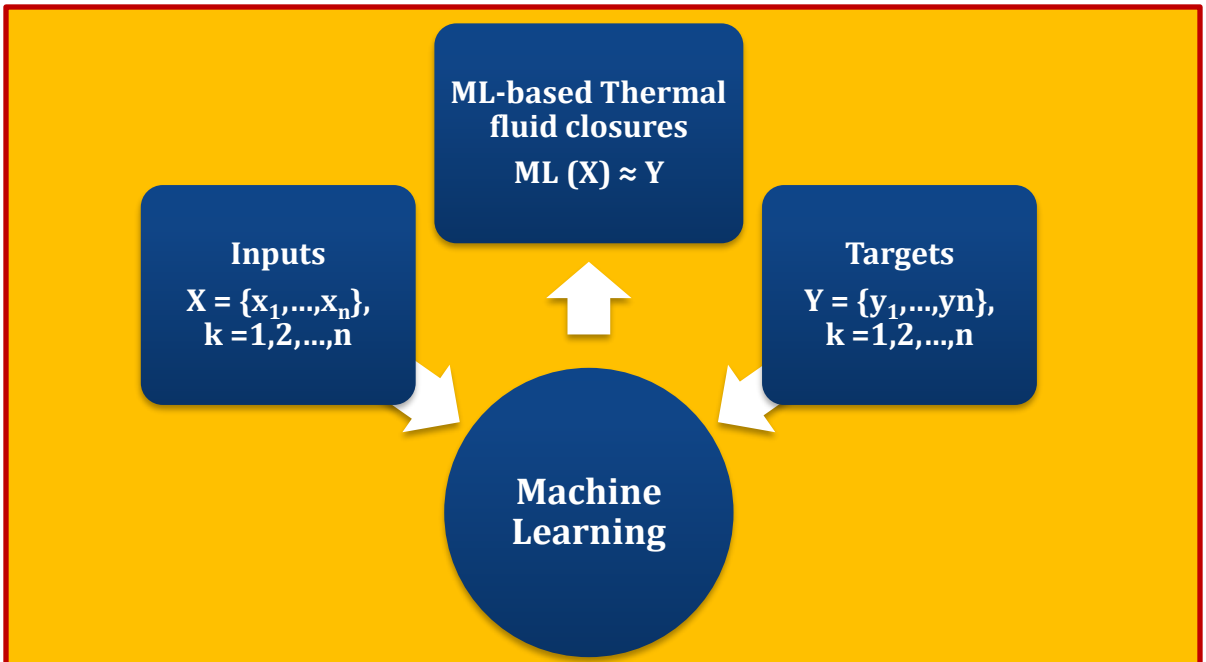


Figure 3.5.1 Workflow of Employing ML methods for Developing Thermal fluid closures – (Courtesy of Chang & Dinh)

The ***X*** denotes the flow feature space as inputs. The ***Y*** presents the response space as targets that are associated with flow features. The subscript k denotes the k^{th} measurement at a certain location. After collecting all relevant datasets, ML models are generalized by a set of nonlinear functions with hyper parameters to represent a thermal fluid closure. Based on different ML methods, various algorithms are employed to seek an optimal solution that allows a ML-based model to fit the observed data. Based on distinct learning purposes, (Domingos, 2015)[79] classified ML methods into five tribes including symbolists, evolutionary, analogizes, connectionists, and Bayesians. [Ling & Templeton][80] evaluated the predictability of various ML algorithms for predicting the averaged Navier-Stoke uncertainty in a high Reynolds region.

3.5.2 Thermal Fluid Data

Figure 3.5.2 provides an overall characterization of thermal fluid data by data type, data source, and data quality. The global data are system conditions and integrated variables such as system pressure, mass flow rate, pressure drop, and total heat input. The local data are time series data at specific locations. The field data are measurements of field variables resolved in space and in time. Traditionally, experiments are a primary source of data, including so-called ***integral effect tests (IETs)*** and ***separate effect tests (SETs)***. As the name suggests, SETs and IETs are designed to investigate isolated phenomena and complex (tightly coupled) phenomena, respectively. Increasingly, appropriately validated numerical simulations become a credible source of data. This includes high-fidelity numerical simulations (e.g., DNS, and other CFD methods), as well as system-level simulation using computer models in parameter domains that are extensively calibrated and validated. It is noted that datasets vary by their quality regarding the quantity and uncertainty. The amount of data affects the performance of inverse modeling since sufficient data can reduce the model parameter uncertainty in the domain of interest. Within a narrow context of ML for thermal fluid simulation, the data quality can be characterized by the amount of relevant and adequately

[79] Domingos P., The Master Algorithm, Basic Books, 2015.

[80] Ling J., Templeton J., *Evaluation of machine learning algorithms for prediction of regions of high Reynolds averaged Navier Stokes uncertainty*, Physics of Fluids, 2015.

evaluated data (i.e., data quantity) and associated uncertainty (including measurement uncertainty and other biases, e.g., scaling, processing).

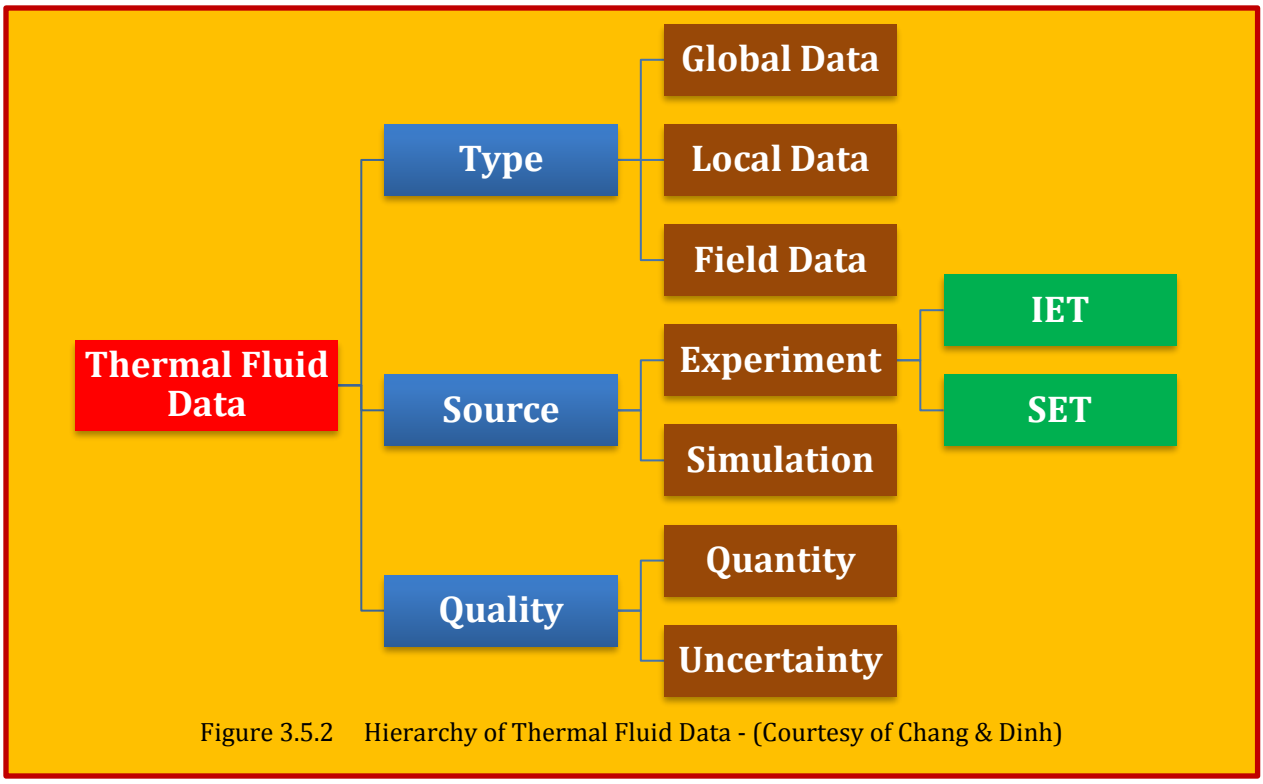


Figure 3.5.2 Hierarchy of Thermal Fluid Data - (Courtesy of Chang & Dinh)

3.5.3 Machine Learning Frameworks for Data-Driven Thermal Fluid Models

As evident from the preceding discussion, the time is ripe for applying ML methods to assist developments of data-driven thermal fluid models. There is a large variety of ways to embed ML in the models. The three major factors are knowledge of physics (overall model form), available data, and machine learning techniques. It is necessary to incorporate knowledge of underlying physics whenever such knowledge is available and trustworthy into ML-based models. The benefits of so doing are wide-ranging, from preventing models from generating unphysical results to narrowing the search space by reducing the dimension of problems. Here, the focus is placed on the use of neural networks, specifically deep learning (or multilayer neural networks), which has recently emerged as capable and universal approximate. Notably, their hierarchical structures deem appropriate for describing complex models that involve multiple scales. The objectives is to develop a system to characterize different approaches to use ML to aid developments of data-driven models in thermal-fluid simulation to help navigate an inter-disciplinary domain (of thermal-fluid simulation, data-driven modeling, and deep learning). The technical approach stems from literature analysis, implementation, and investigation of major types of ML frameworks on synthetic examples.

3.5.4 Criteria for Classifying ML Frameworks for Thermal Fluid Simulation

Each framework has its distinct goal and approach to leverage data. Since we classify five frameworks, we build the classification system based on four conditions. First, we examine whether solutions are converged meaning that solutions conserve the mass-momentum-energy balance in a control volume. Second, we check if the framework focuses on developing fluid closures. Third, we distinguish ***Type-III*** ML from other frameworks because it inherently ensures data-model consistency. Finally, the last condition is about the separation of scales. Accounting for all four conditions, we categorize five distinct types of ML frameworks for thermal fluid simulation based on the following four criteria:

3.5.4.1 Criterion 1- Is PDE Involved in Thermal Fluid Simulation?

The first criterion examines whether conservation equations are involved in thermal fluid simulation. ***Type-V*** ML relies on ML to discover the underlying physics directly from data and to deliver equivalent surrogates of governing equations. ***Type-V*** ML is an extreme case when there is no prior knowledge, and we must purely depend on the observed data. By this criterion, we can distinguish ***Type-V*** ML from other four ML frameworks.

3.5.4.2 Criterion 2 - Is the Form of PDEs Given?

The second criterion inspects if the form of conservation models is known. ***Type-IV*** ML does not make biases on selecting physics models; instead, it recovers the exact form of conservation models based on data. Therefore, we can distinguish ***Type-IV*** ML from ***Type-I, Type-II***, and ***Type-III*** ML.

3.5.4.3 Criterion 3 - Is the PDE Involved in the Training of Closure Relations?

PDEs are involved in ***Type-I, Type-II***, and ***Type-III*** ML. Therefore, the goal is to develop closure models in nonparametric forms to close conservation equations. Criterion 3 checks whether conservation equations are involved in the training of ML-based closures. Traditionally, the assumptions of scale separation and physics decomposition are essential to develop closure models. The former allows us to set up SETs for various scales while the latter decomposes closure relations into different physics within the same scale. However, in many thermal fluid processes, the physics (physical mechanisms) is tightly coupled. ***Type-III*** ML avoids these two assumptions by training closure models that are embedded in PDEs. By this criterion, we can distinguish Type III ML from Type I and Type II ML.

3.5.4.4 Criterion 4 - Is a Scale Separation Assumption Required for the Model Development?

This criterion tests whether the model development requires the separation of scales. This hypothesis isolates closure relations from conservation equations so that the models can be separately built and calibrated by SETs. The scale separation is essential for ***Type-I*** ML because it only relies on data to construct closure models. However, the data by SETs may have been distorted, while IETs are designed to capture (a selected set of) multi-physics phenomena. **Table 3.5.1** summarizes the criteria to classify the five distinct types of ML frameworks for thermal fluid simulation.

Classification	Type I	Type II	Type III	Type IV	Type V
Is PDE involved in thermal fluid simulation?	Yes	Yes	Yes	Yes	No
Is the form of PDEs given?	Yes	Yes	Yes	No	No
Is the PDE involved in the training of closure relations?	No	No	Yes	No	No
Is a scale separation assumption required for the model development?	Yes	No	No	No	No

Table 3.5.1 Criteria for the ML Framework Classification - (Courtesy of Chang & Dinh)

3.5.4.5 Type-I : Physics-Separated Machine Learning (PSML)

Type-I ML or so-called physics-separated ML (PSML) aims at developing closure models by using SET data. ***Type-I*** ML assumes that conservation equations and closure relations are scale separable, for

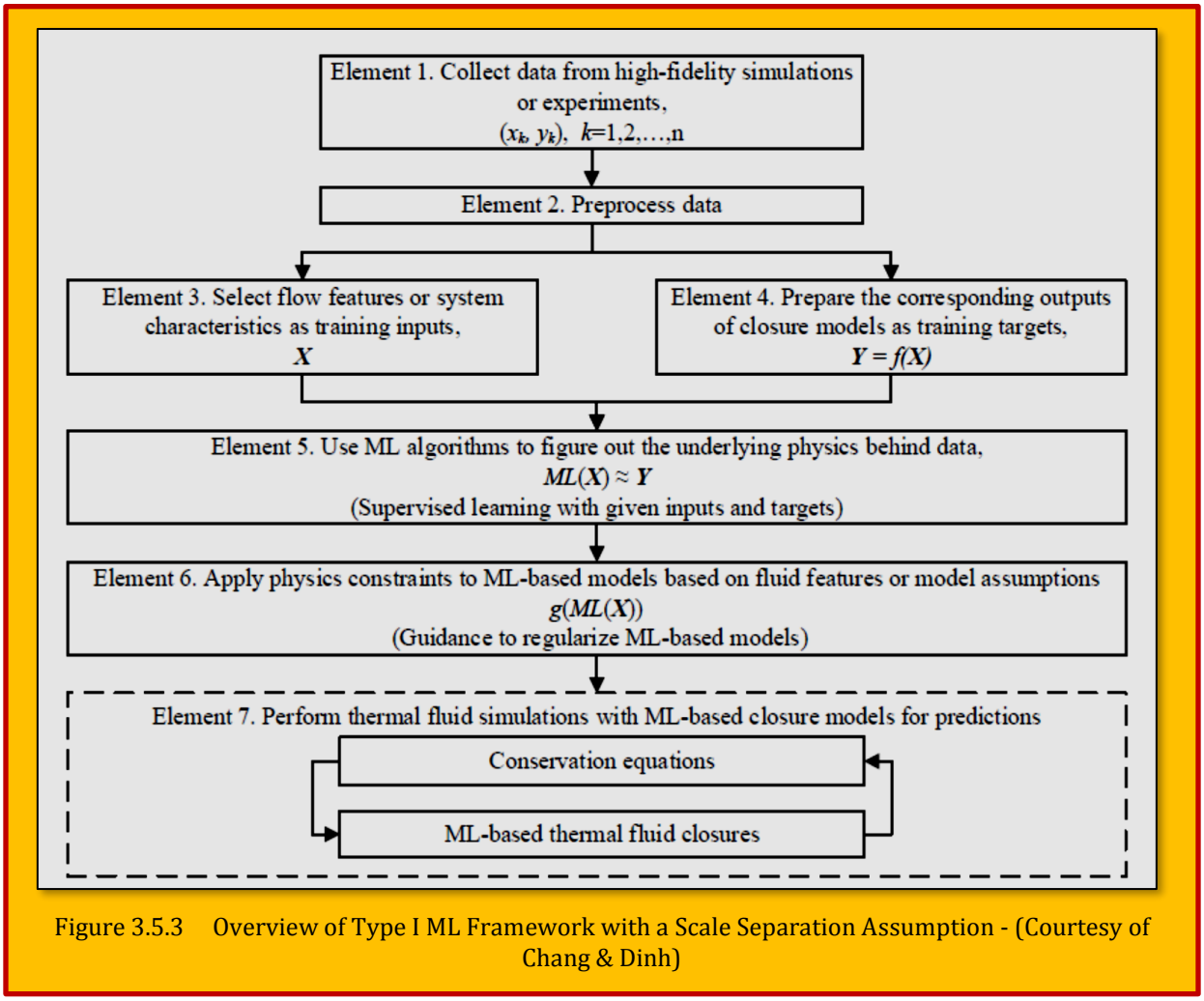


Figure 3.5.3 Overview of Type I ML Framework with a Scale Separation Assumption - (Courtesy of Chang & Dinh)

which the models are local. ***Type-I*** ML requires a thorough understanding of the system so that SETs can be designed to support model developments. We can apply ML-based closures to assimilate data to achieve data-driven thermal fluid simulation. **Figure 3.5.3** depicts the architecture of ***Type-I*** ML framework, and it is forward data-driven modeling. The procedure includes the following elements:

3.5.4.5.1 Element 1

Assume a scale separation is achievable such that closure models can be built from SETs. From either high-fidelity simulations or experiments, collect training data, (x_k, y_k).

3.5.4.5.2 Element 2

Preprocess data from element 1 to ensure that data from multi-sources have the same dimension and manipulation such as the selection of averaging methods. Additionally, consider normalizing data so that we can approximately equalize the importance for each data source. For large datasets, employ principal component analysis can be helpful to reduce the dimension of data.

3.5.4.5.3 Element 3

Compute flow features or system characteristics, ***X***, as training inputs for element 5.

3.5.4.5.4 Element 4

Calculate the corresponding outputs (***Y***) of the desired closures from data as training targets that can supervise ML algorithms to learn from data.

3.5.4.5.5 Element 5

Utilize ML algorithms to build a correlation between inputs and targets. After the training, output the ML-based closure model, ML(**X**), to element 6.

3.5.4.5.6 Element 6

Constrain the ML-based closure, g(ML(**X**)), to satisfy model assumptions and to ensure the smoothness of model outputs since it needs to be solved with PDEs. It is noted that this element is not essential if assumptions are not applicable.

3.5.4.5.7 Element 7

Implement the ML-based closure into conservation equations, and solve PDEs for predictions with the embedded ML-based closure that is iteratively queried.

Type-I ML satisfies the criteria from **Table 3.5.1** except the third criterion. The quality of SET data largely controls the performance of closure models obtained by ***Type-I*** ML. While the experimental uncertainty in each SET may be controlled and reduced, the process uncertainty (dominated by design assumptions) is irreducible. We refer that PDEs and closure relations are decoupled in ***Type-I*** ML. It can cause model biases between conservation equations and closure relations. It is noted that inferring model parameters from data belong to inverse problems which are ill-posed. For ML models, a small change in inputs can result in large uncertainty in outputs. While implementing ML based closures in PDEs, the uncertainty can lead to a discontinuity that fails numerical simulation. For more practices related to ***Type-I*** ML, readers are referred to [Ma et al.][81-82], [Parish & Duraisamy][83], [Zhang & Duraisamy][84], among numerous others.

3.5.4.6 Type-II: Physics-Evaluated Machine Learning (PEML)

Type-II ML or so-called physics-evaluated machine learning (PEML) focuses on reducing the uncertainty for conservation equations. It requires prior knowledge on selecting closure models to predict thermal fluid behaviors. ***Type-II*** ML utilizes high-fidelity data to inform low-fidelity simulation. Comparing to high-fidelity models, ROMs can efficiently solve engineering design problems within an affordable time frame. However, ROMs may produce significant uncertainty in predictions. ***Type-II*** ML can improve the uncertainty of low-fidelity simulation by reference data. Since the physics of thermal fluids is nonlinear, ML algorithms are employed to capture the underlying correlation behind high-dimensional data. The framework requires training inputs such as flow features that represent the mean flow properties. Training targets are the responses that correspond to input flow features. ***Type-II*** ML satisfies the first two criteria in **Figure 3.5.3**. We refer that PDEs and closure relations are loosely coupled in ***Type-II*** ML because PDEs are only used for calculating input flow features. The framework provides a one-step solution to improve low-fidelity simulation. Model uncertainty is not accumulated in ***Type-II*** ML because numerical solvers do not interact with ML models. However, ***Type-II*** ML exists an open question about what the magnitude of initial errors can be before it is too late to bring a prior solution to a reference solution. For more detailed examples of ***Type-II*** ML, readers are referred to [Ling & Templeton][85], [Ling, et

---

[81] Ma M., Lu J., Tryggvason G., *Using statistical learning to close two-fluid multiphase flow equations for a simple bubbly system,* Physics of Fluids, 27 (2015).

[82] Tryggvason G., Ma M., Lu J., *DNS–Assisted Modeling of Bubbly Flows in Vertical Channels*, Nuclear Science and Engineering, 184 (2016) 312-320.

[83] Parish E.J., Duraisamy K., *A paradigm for data-driven predictive modeling using field inversion and machine learning,* Journal of Computational Physics, 305 (2016) 758-774.

[84] Zhang Z.J., Duraisamy K., "*Machine Learning Methods for Data-Driven Turbulence Modeling*", American Institute of Aeronautics and Astronautics, 2015.

[85] Ling J., Templeton J., *Evaluation of machine learning algorithms for prediction of regions of high Reynolds averaged Navier Stokes uncertainty*, Physics of Fluids, 27 (2015) 085103.

al.][86], [Wu *et al.*][87], and [Zhu & Dinh][88].

3.5.4.7 Type III - Physics-Integrated Machine Learning (PIML)

To the best knowledge of the authors, ***Type-III*** ML or so-called physics-integrated ML (PIML) is introduced and developed for the first time in this work. ***Type-III*** ML aims at developing closure relations to close thermal fluid models without a scale separation assumption. Closure models are embedded and trained in system dynamics. Training data can be obtained from SETs and IETs. Notably, ***Type-III*** ML can lead the paradigm shift of using ML in thermal fluid simulation because it allows the direct use of field data from IETs. Inputs for ***Type-III*** ML do not directly come from observations; instead, they are solutions of PDEs. ***Type-III*** ML satisfies most criteria in **Table 3.5.1** except for the fourth criterion. We refer that PDEs and closure relations are tightly coupled in ***Type-III*** ML. It is a challenging problem. Such tightly coupled multiscale problems require that numerical solutions (of the governing PDE system) are realized (hence evolving datasets for training) whenever ML algorithms tune model parameters. Therefore, ***Type-III*** ML is computationally expensive. The research on ***Type-III*** ML methodology promises a high-potential impact in complex thermal fluid problems where the separation of scales or physics decomposition may involve significant errors.

3.5.4.8 Type IV - Physics-Recovered Machine Learning (PRML)

Type-IV ML or so-called physics-recovered ML (PRML) aims at recovering the exact form of PDEs. It requires no assumption about the form of governing equations. Instead, the framework requires to construct a candidate library that includes components of governing equations such as time derivative, advection, diffusion, and higher order terms. ***Type-IV*** ML only satisfies the first criterion in **Table 3.5.1**. The challenge of ***Type-IV*** ML can be the recovery of closure relations in thermal fluid models. Closure models are usually complex, and they are hard to be represented by each derivative term. Therefore, it is an open question about how to apply ***Type-IV*** ML for complex flow system such as turbulence modeling. For more practices related to ***Type-IV*** ML, readers are referred to [Brunton et al.][89].

3.5.4.9 Type V - Physics-Discovered Machine Learning (PDML)

Type-V ML or so-called physics-discovered ML (PDML) is the extreme case. ***Type-V*** ML is used for either condition. First, it assumes no prior knowledge of physics. Second, it assumes existing models and modeling tools are not trustworthy or not applicable for thermal fluid systems under consideration. More generally, ***Type-V*** ML is "equation-free" and instrumental in the search for a new modeling paradigm for complex thermal-fluid systems. ***Type-V*** ML does not involve conservation equations nor satisfy any criterion in **Table 3.5.1**. Instead, it wholly relies on data to discover the effective predictive models. However, such situation rarely occurs because there are usually physics principles or hypotheses that can be postulated to reduce the dimension of problems. For the discussion related to ***Type-V*** ML, readers are referred to [Mills et al.][90] and [Hanna et al.][91].

---

[86] Ling J., Jones R., Templeton J., *Machine learning strategies for systems with invariance properties,* Journal of Computational Physics, 318 (2016) 22-35.

[87] Wu J.-L., Wang J.-X., Xiao H., Ling J., *Physics-informed machine learning for predictive turbulence modeling: A priori assessment of prediction confidence*, (2016).

[88] Zhu Y., Dinh N.T., *A Data-Driven Approach for Turbulence Modeling, in: NURETH-17*, American Nuclear Society, Xi'an, China, 2017.

[89] Brunton S.L., Proctor J.L., Kutz J.N., *Discovering governing equations from data by sparse identification of nonlinear dynamical systems*, Proceedings of the National Academy of Sciences, 113 (2016) 3932–3937.

[90] Mills K., Spanner M., Tamblyn I., *Deep learning and the Schrödinger equation*, (2017).

[91] Hanna B.N., Dinh N.T., Youngblood R.W., Bolotnov *I.A., Coarse-Grid Computational Fluid Dynamics (CG-CFD) Error Prediction using Machine Learning*, under review, (2017).

3.5.4.10 Knowledge and Data Requirements for ML Frameworks in Thermal Fluid Simulation

In the present context of ML, knowledge refers to a body of theoretical and empirical evidence that is available and trustworthy for understanding and description of physical mechanisms that underlie thermal fluid processes under consideration. This knowledge can guide selecting model forms, including conservation equations and corresponding closure relations, designing experiments, and performing high-fidelity simulations. The data requirements refer to characteristics of the body of data (e.g., types, amount, quality) needed to enable thermal fluid simulation with the required accuracy. In other words, the required data must be sufficient to complement the "knowledge" for building closure models and recovering/discovering the physics. The form of PDEs are known for Type I, Type II, Type III ML, and the focus is to build closure relations. In traditional modeling approaches, closure models are local, relating a group of (local) source terms (i.e., sub-grid-scale interactions) to a group of (local) flow features. Even when in engineering literature, source terms are expressed regarding global parameters (like flow rate, system pressure), they are used as surrogates for local-valued parameters (through the assumptions that equate global and local conditions).

Type - I ML build closure relations independently from PDEs, but it requires a thorough or assumed understanding of the physics that is essential to set up SETs for acquiring data. Globally measured data or locally measured data (using point instruments) are very small amount of data. In such case, complicated ML-based closures are not necessarily the best choice. Therefore, among the frameworks, ***Type - I*** ML exhibits a minimal data requirement with a maximal knowledge requirement.

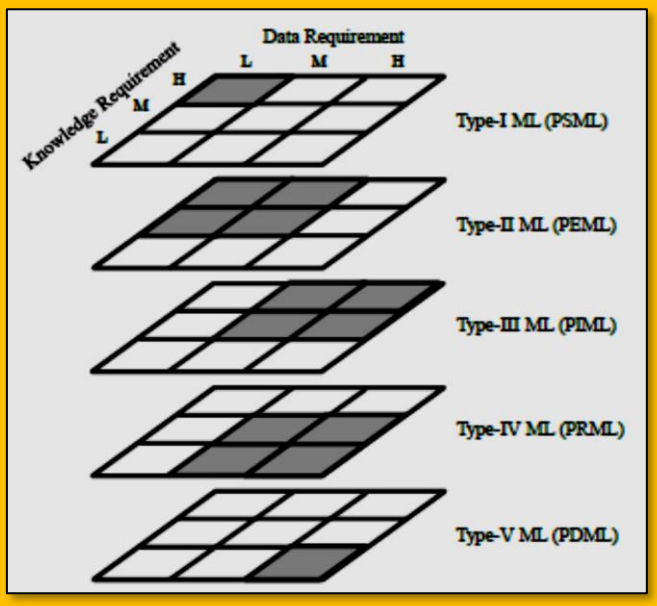


Figure 3.5.4 Domain of Various ML Frameworks where **L**, **M**, and **H** Denote Low, Medium, and High - (Courtesy of Chang & Dinh)

Type-II ML assumes prior knowledge of physics that guide the selection of closure relations for thermal fluid simulation. However, the use of prior models yields uncertainty in thermal fluid analyses. This uncertainty (or error) can be inferred by comparing the model prediction to reference solutions from high-fidelity simulations, high-resolution experiments as well as data obtained in IETs that include multi-physics phenomena. Correspondingly, ***Type-II*** ML requires larger data quantities but less knowledge than ***Type-I*** ML.

Type-III ML trains closure relations that are embedded in conservation equations without invoking a scale separation assumption. IET data can be directly adapted into simulation by applying ***Type-III*** ML. While the term ML is broad, in the present work ML refers to the use of non-parametric models or even narrower, use of DNNs. This means no prior knowledge of model forms of closure relations. Thus, ***Type-III*** ML requires less knowledge than ***Type-II*** ML (which "best-estimated" closure models on the basis of past data). Consequently, ***Type-III*** ML requires a large body of data to represent models than that of ***Type-II*** ML. ***Type-IV*** ML intends not to make any bias on selecting conservation equations; instead, it recovers the exact PDE form from data. It assumes less prior knowledge but requires more extensive training data than the previous three frameworks.

Type-V ML is an extreme case that makes no assumption about prior knowledge or reference solutions for thermal fluid systems under consideration. The aim is to apply ML methods to learn from data, and to establish a data-driven predictive capability. For thermal fluid simulation, it means discovering the effective model form of conservation equations and closure relations. Accordingly, among the frameworks, ***Type-V*** ML is the most stringent with respect to data requirements (types, quantity, and quality). **Figure 3.5.4** depicts the domain of ML frameworks regarding prior knowledge and data requirements

3.5.4.11 Case Study 1.1 - Heat Conduction Investigation by Type I ML Framework

The heat conduction case study is formulated to demonstrate how to employ Type I, Type II, and Type III ML to build ML-based thermal conductivity and to compare results by each framework. [Chanda et al.] used ANN with genetic algorithm[92] to solve inverse modeling for heat conduction problems. In this work, ***Deep Learning (DL)***[93] is selected as the ML methodology in this task. Principally, any neural network (NN) with more than two layers (one hidden layer with an output layer) is considered as to be [DL][94]. [Hornik][95] proved that multilayer NNs are universal approximators, and it can capture the properties of any measurable information. This capability makes DL attractive for the closure development in thermal fluid simulation. Notably, we implement NN-based thermal conductivity by FNNs and ***convolutional neural networks (CNNs)*** to evaluate the performance of closure relations by distinct NNs.

3.5.4.11.1 Problem Formulation

We formulate the synthetic task using a 2D (two-dimensional) heat conduction model given by where k(T) is nonlinear thermal conductivity. To generate training data, shows a temperature-dependent model for k(T) where c, σ, and μ are constant parameters. **Table 3.5.2** gives two parameter sets (baseline and prior sets) to generate data. While demonstrating ML frameworks, k(T) becomes NN-based thermal conductivity.

$$\frac{\partial}{\partial x}\left[k(t)\frac{\partial T}{\partial x}\right]+\frac{\partial}{\partial y}\left[k(t)\frac{\partial T}{\partial y}\right]=0 \quad , \quad k(t)=\frac{c}{\sigma\sqrt{2\pi}}e^{\frac{(T-\mu)^2}{2\sigma^2}}$$

Eq. 3.5.1

Data Set	c (W/m)	σ (K)	μ (K)
Baseline set for producing synthetic data	7.2x104	300	1200
Prior set for producing inputs required by Type II ML	7.2x104	600	2100

Table 3.5.2 Parameter Sets for the Thermal Conductivity Model - (Courtesy of Chang & Dinh)

Two numerical experiments are designed to emulate IETs and SETs for manufacturing synthetic data by solving using parameters sets in **Table 3.5.2**. IETs provide field data, for instance, 2D temperature fields by an infrared camera. SETs offer global data such as a 1D measurement by thermocouples. Synthetic data are used for training and validating NN-based thermal conductivity.

---

[92] Hanna B.N., Dinh N.T., Youngblood R.W., Bolotnov *I.A., Coarse-Grid Computational Fluid Dynamics (CG-CFD) Error Prediction using Machine Learning, under review*, (2017).
[93] LeCun Y., Bengio Y., Hinton G., *Deep learning, Nature*, 521 (2015) 436-444.
[94] Heaton J., Artificial Intelligence for Humans, Volume 3: *Deep Learning and Neural Networks*, Heaton Research, Inc., Chesterfield, MO, 2015.
[95] Hornik K., Stinchcombe M., White H., *Multilayer Feedforward Networks are Universal Approximators, Neural Networks*, 2 (1989) 359-366

Type I ML can only use SET data because of a scale separation assumption. Type II ML can only use SET data because the goal is to improve the prior thermal conductivity by the baseline. Type III and Type V ML use field data. We compare Type I and Type II ML using training data from SETs. Then Type III and Type V ML are compared by field data from IETs.

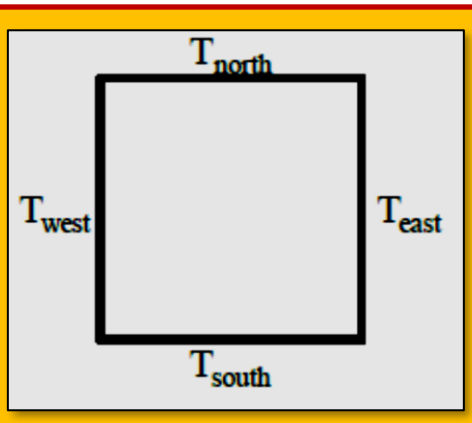


Figure 3.5.5 Schematic of integral effects tests (IETs) for measuring Temperature fields - (Courtesy of Chang & Dinh)

3.5.4.11.2 Manufacturing IET Data

IETs are measurements of temperature fields. Synthetic IET data are generated by **Eq. 3.5.1** with the baseline set in **Table 3.5.2**. **Figure 3.5.5** illustrates the layout of IET experiments with four constant boundary temperatures. We change T_{west} for various observations and fix the boundary temperature (1300K) at the east side. The T_{north} and T_{south} are linearly dependent on the west boundary condition. We prepare three training datasets by including distinct data quantities and three validating datasets by changing T_{west}. **Table 3.5.3** gives the metadata of each training or validating dataset. All observations are uniformly sampled within a given temperature range.

Data Set	Data Quantity	Temperature Range at T_{west}	Description
T1	11 observations	[1000K, 1100K]	Training data set
T2	100 observations	[1000K, 1100K]	Training data set
T3	1000 observations	[1000K, 1100K]	Training data set
P1	1000 observations	[1000K, 1100K]	Validating data set
P2	1000 observations	[900K, 1000K]	Validating data set
P3	1000 observations	[800K, 900K]	Validating data set

Table 3.5.3 Summary of IET Training and Validating Data Sets - (Courtesy of Chang & Dinh)

3.5.4.11.3 Manufacturing SET Data

SETs are global measurements by thermocouples. **Figure 3.5.6** depicts the layout of SETs for obtaining mean temperature and heat conductivity data. A heater is on top of the sample to maintain a constant temperature (T_H). Thermal insulations are installed on the outside surface. The coolant at the bottom removes the heat with a constant heat transfer coefficient. **Eq. 3.5.2** calculates temperature profiles within the sample using parameter sets in **Table 3.5.3**. **Eq. 3.5.2** also calculates the observed heat conductivity (k_{obs}), and the mean temperature is obtained by arithmetic averaging T_H and T_C.

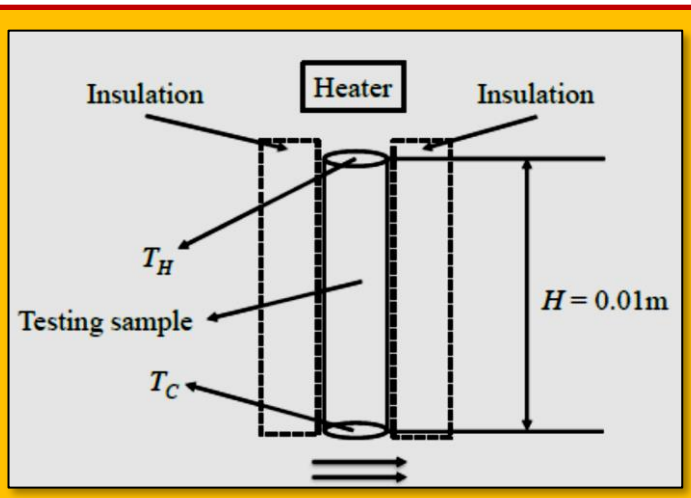


Figure 3.5.6 Schematic of Separate Effects Tests (SETs) for Measuring Thermal Conductivity as the Function of Sample's Mean Temperature - (Courtesy of Chang & Dinh)

$$\frac{\partial}{\partial x}\left[k(t)\frac{\partial T}{\partial x}\right]=0 \quad , \quad k_{obs}\frac{T_H-T_C}{H}=h(T_C-T_{coolant})$$

Eq. 3.5.2

We generate two training datasets with two coolant temperatures to explore the effect by different data qualities shows the metadata of SET datasets. A large temperature gradient across the testing sample increases the nonlinearity of temperature profiles. For each training set, we uniformly sample 41 T_H from **Eq. 3.5.3** to keep mean temperatures in SETs within the same range as IETs.

$$(H_{H,max}, H_{H,min})=(2T_{IET,max}-T_{coolant}, 2T_{IET,min}-T_{coolant})$$

Eq. 3.5.3

Data Set	Data Quantity	Data Quality	$T_{coolant}$(K)	Description
S1	41 Observations	Low	800	Training dataset
S2	41 Observations	High	900	Training dataset

Table 3.5.4 Summary of SET Training Datasets - (Courtesy of Chang & Dinh)

3.5.4.11.4 Implementation of the Heat Conduction by Type I ML Frameworks

We present ***Type-I*** ML in ***Algorithm 1***. SET data are generated by **Eq. 3.5.2** with the baseline set in **Table 3.5.4**. Inputs and targets are temperatures and thermal conductivities. After the training, FNN-based thermal conductivity is implemented in **Eq. 3.5.1** for predictions. For additional information, or how to construct the type II, III, IV, V algorithm, please consult the work by [Chang and Dinh][96] .

Algorithm 1 - Type I ML for 2D Heat Conduction Problem with Dirichlet B.C.	
Input	Training Input ($T_{baseline}$, elemnt 3 in **Error! Reference source not found.**,), and t raining targets ($k_{baseline}$, element 4 in **Error! Reference source not found.**) from SETs (element 1 in **Error! Reference source not found.**)
Output	Temperature fields for predictions (element 7 in **Error! Reference source not found.**)
1	for all epochs < maximum_epoch do (element 5 in **Error! Reference source not found.**)
2	// Build a conductivity model using FNNs
3	K(T) ← FNN(T)
4	for all inputs (T) $_{baseline}$, $k_{baseline}$ Tk) ∈ training datasets do
5	Update hyperparameters for each layer in FNNs
6	Implement ()kT into Eq. (1) (element 7 in **Error! Reference source not found.**)

3.5.4.11.5 CNN-Based Thermal Conductivity Model

Figure 3.5.7 depicts the architecture[97] of CNN-based thermal conductivity that includes three convolutional layers and three fully connected layers. We use the [ReLU][98] activation for layers in CNNs to accelerate the training. Inputs are temperature fields. After the first convolutional layer,

---

[96] Chih-Wei Chang and Nam T. Dinh, "*Classification of Machine Learning Frameworks for Data-Driven Thermal Fluid Models*", North Carolina State University, Raleigh NC 27695-7909.

[97] Lecun Y., Bottou L., Bengio Y., Haffner P., *Gradient-based learning applied to document recognition,* Proceedings of the IEEE, 86 (1998) 2278-2324.

[98] Nair V., Hinton G.E., *Rectified linear units improve restricted Boltzmann machines*, 2010, pp. 807–814.

eight feature maps are generated, and each feature map detects the patterns from temperature fields. The second convolutional layer takes inputs from the previous layer, and it outputs 12 feature maps. The third convolutional layer receives inputs from the previous layer, and it delivers 24 feature maps to fully connected layers. ***Finally, we obtain thermal conductivity fields from CNN's outputs. Learning is an optimization process, and we need to define a cost function based on distinct types of data to inform ML algorithms to tune NN hyper parameters.*** **Eq. 3.5.4** defines the cost function where N, $y_{i,data}$, and $y_{i,model}$ are the total number of training data, ith training data, and ith model solution. To prevent overfitting, we add a regularization term in **Eq. 3.5.4** where i, and N_L denote the ith layer and total layer number. λ is the regularization strength, and **W** is the matrix of total weights in ith layer. We implement NN-based thermal conductivity using ***Tensor flow***[99] which is the DL framework developed by Google. Weights and biases of NNs are tuned based on data using [Adam][100] algorithm.

$$E = \frac{1}{2N}\sum_{i=1}^{N}\left(y_{i,model} - y_{i,data}\right)^2 + \sum_{i=1}^{N_L}\lambda_i \|W_i\|^2$$

Eq. 3.5.4

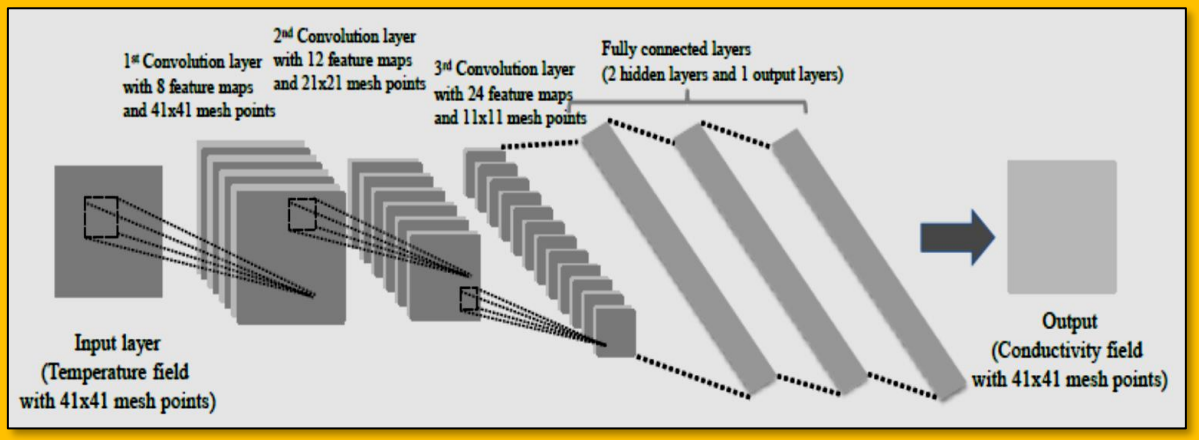


Figure 3.5.7 Architecture of CNN-Based Thermal Conductivity (adopted after LeCun)

3.5.4.11.6 Closing Remarks

The present study is motivated by the growing interest and development of machine learning models in thermal fluid simulation. The trend is powered by the advent of data-intensive research methods, such as modern thermo-fluid experiments and high-fidelity numerical simulations, affordable computing (data processing) power and memory, and progress in machine learning methods, particularly in deep learning using multilayer neural networks. We introduced a classification of machine learning frameworks for thermal fluid simulation, including five types. The selection of the optimal ML framework is problem-dependent, and to a substantial extent, depends on characteristics of supporting data, including data source, data type, data quantity, and quality. Although examples of Type I and Type II models existed in the literature, their developments are still in infancy. While technically straightforward, both Type I and Type II models are limited to systems whose governing physics are open to "***divide-to-conquer***".

---

[99] Abadi M., Agarwal A., Barham P., Brevdo E., Chen Z., Citro C., Corrado G.S., Davis A., Dean J., Devin M., others, *Tensor flow: Large-scale machine learning on heterogeneous distributed systems*, (2016).

[100] Kingma D.P., Ba J., Adam: *A Method for Stochastic Optimization*, (2014).

In Type III, PDEs are involved in the training of machine-learning models, thus alleviating the requirements on the scale separation assumption, and potentially reducing the necessity on the physics decomposition. Correspondingly, Type III models present more stringent requirements on modeling and substantially higher computing resources for training. Based on insights from the case study performed, Type III ML has the highest potential in extracting the value from "big data" in thermal fluid research, while ensuring data-model consistency. There are technical challenges that need to be addressed before Type III models deliver their promises in practical thermal fluid simulation, namely,

- Complex interactions of ML-based closures with a system of PDEs (including discontinuity in hyperbolic systems);
- Effect of the non-local character of ML-based models on PDE solution methods; and
- Implementation and effect of multiple closure models, particularly in multiphase and thermal flows.

3.6 Case Study 6 - Machine Learning and Simulation For Grid Optimization

Authors : *Peer Breier, Qunsheng Huang, Moritz Krügener, Oleksandr Voloshyn, Mengjie Zhao supervision by Dr. Dirk Hartmann, Andres Botero Halblaub, Friedrich Menhorn*
Citation : (Breier, Huang, Krügener, Voloshyn, & Zhao)
Bibliography : *Breier, P., Huang, Q., Krügener, M., Voloshyn, O., & Zhao, M. (n.d.). Machine Learning and Simulation For Grid Optimization. Bavarian Graduate School of Computational Engineering.*

3.6.1 Motivation

Mesh creation and refinement is one of the most time-consuming steps in any CFD simulation; even automated mesh generation requires high levels of expertise and fine-tuning. This project attempts to circumvent some of this complexity by leveraging deep convolutional neural networks to predict mesh densities for arbitrary geometries. Such a refined mesh can be seen in **Figure 3.6.1**. An automated pipeline was created to generate random geometries and run CFD simulations, iteratively performing targeted mesh refinement utilizing adjoint sensitives. A comprehensive 6TB dataset consisting of 65,000 geometry-mesh pairs was assembled via an extensive post-processing and evaluation setup.

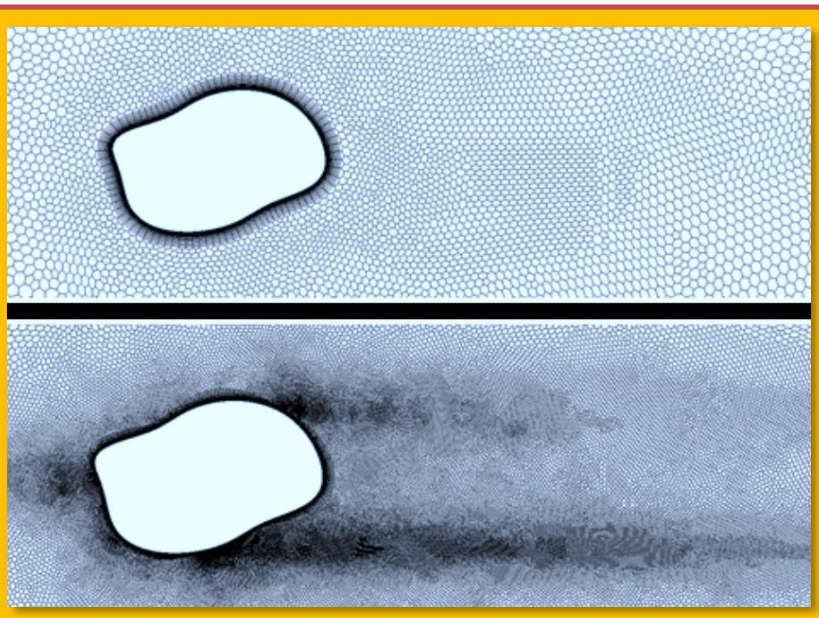


Figure 3.6.1 Unrefined Mesh (top) and Result of Iterative Refinement Strategy for Random Geometry (bottom)

Current literature indicated that the UNet architecture extended by [Thuerey et al.] was suitable to predict flow-related quantities, but had never been used for mesh prediction. In this work, we present a deep, fully convolutional network that estimates mesh densities based off geometry data. The most recent model, tuned with hyper-parameters like network depth, channel size and kernel size, had an accuracy of 98% on our validation dataset. The current pipeline provides a proof-of-concept that convolutional neural networks can, for specific use-cases, generate accurate mesh densities without the need manual fine-tuning. Such a product, if further tuned and extended, can provide significant time savings in future CFD workflows, completely independent of personnel expertise.

3.6.2 Setup

As with all machine learning, training a successful network requires a lot of training data. For our particular case, there was no large dataset of refined meshes for random geometries available, so we had to create it ourselves. This is not a trivial task. As mentioned above, creating refined meshes for complex geometries for CFD simulation is usually an experts task. In our case, it would have been too time consuming to create tens of thousands of meshes manually. Therefore, we used the adjoint solver built in to Star-CCM+© to investigate the sensitivity of the drag force with respect to refinement of individual mesh cells. If a cell shows a high sensitivity (as seen in **Figure 3.6.2**), it is very likely

that "cutting the cell" into smaller sub cells would reduce the residual with regard to the drag force. Our iterative refinement strategy therefore used these results to determine which cells to refine and re-run the simulation with the new mesh densities. This would be repeated until a certain threshold either in residual or in refinement steps would be reached.

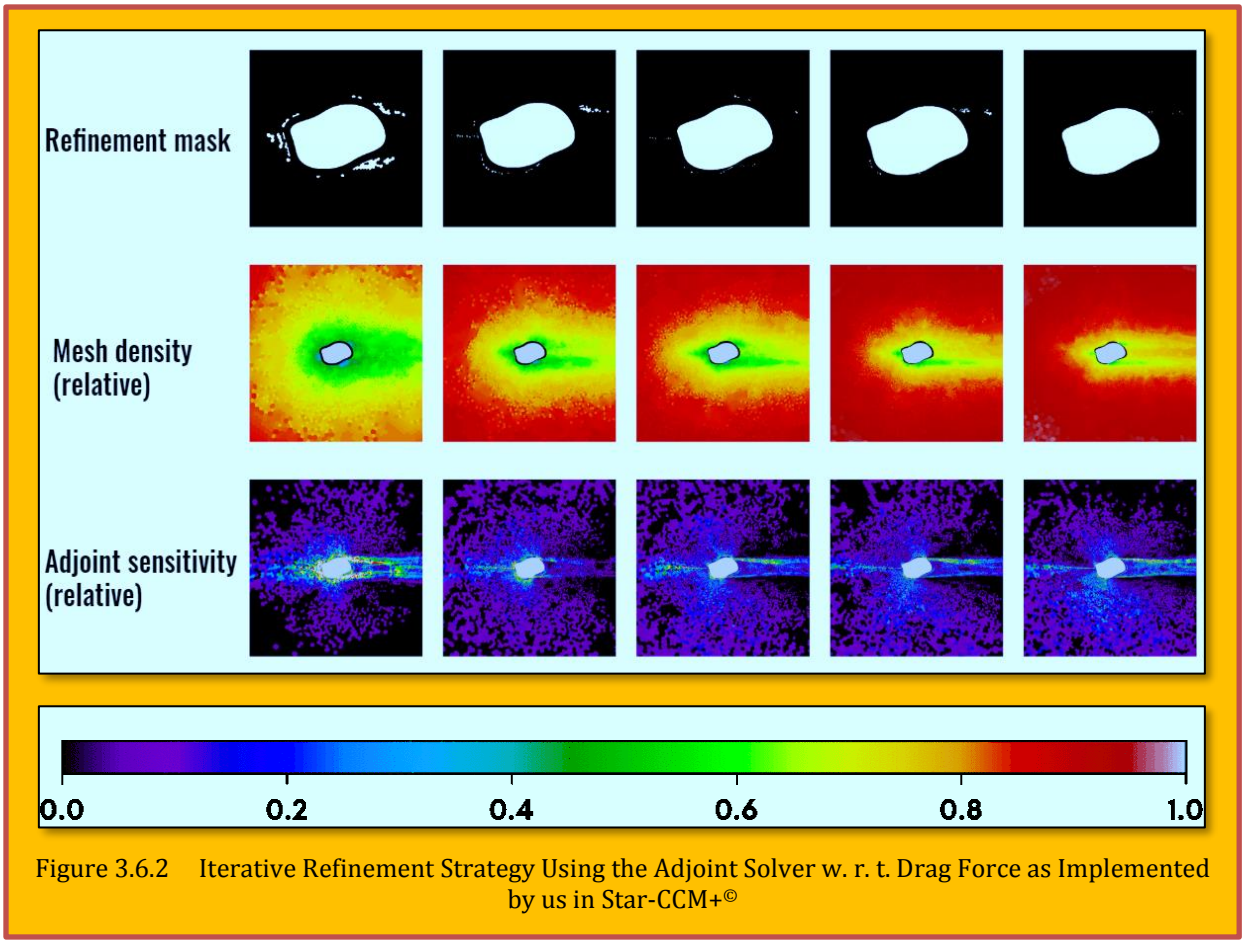


Figure 3.6.2 Iterative Refinement Strategy Using the Adjoint Solver w. r. t. Drag Force as Implemented by us in Star-CCM+©

Using the *CoolMUC2/3* and *IvyMUC* clusters at LRZ, we automated the simulation pipeline and thus simulated and optimized about 65.000 geometries and 325.000 meshes respectively. To handle the massive amounts of data we deployed a *MongoDB* database. We handled post-processing of both the logs coming from the simulations as well as the corresponding mesh pictures in parallel with up to 4 of our personal machines participating with their CPU and GPU power. At the end of post-processing. We had selected a set of around 10.000 geometry-mesh pairs that had gone through the refinement iterations most successfully. The images had been down-sampled and smoothed via custom kernels implemented in *CUDA* to preserve sharp geometry edges. A lot of effort was put into performance-optimizing every step of the pipeline shown in **Figure 3.6.3**. Even with all of this work done, a full pass through of all simulation data through post-processing still takes multiple days with the help of all of our personal machines. During machine learning, a similar distributed approach was used. Different networks and training strategies were dispatched to multiple GPU-equipped machines and would report the final training results back to the database. During this phase, the networks were only trained for a couple of hours each to be able to test as many configurations as possible. In total, more than 200 unique network/training conditions were tested.

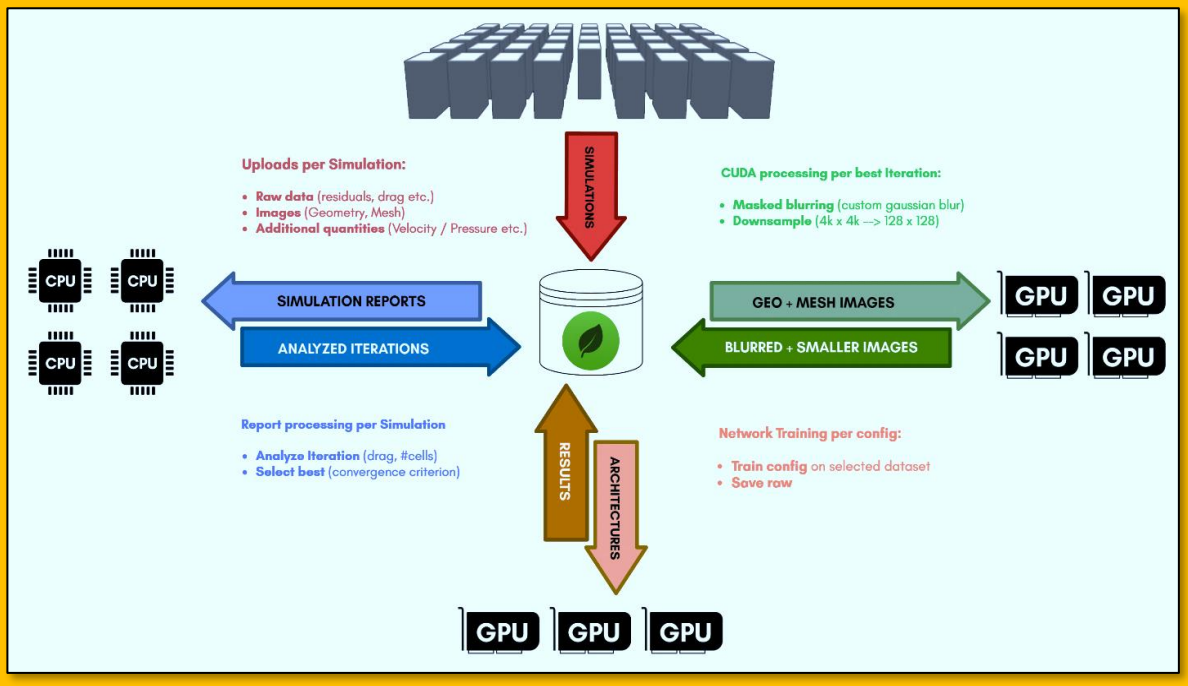


Figure 3.6.3 Data Pipeline from Simulation Creation to Post-Processing and Neural Network (NN) Training

3.6.3 Results

We specifically tested fully-convolutional networks with 3 different architectures. A simple down- and up-sample network, the same network with skip connections and finally a staircase architecture (**Figure 3.6.4**). After initial testing, promising configurations were tested on the full training set and for up to 24 hours with different training hyper-parameters like learning rate and beta-decay for the Adam Optimizer. Finally, the best performing configuration was a staircase UNet architecture 8 layers deep and 2 layers wide on each depth with tensor skip connections at levels 2,3,4,5 and maximal kernel sizes of 16x16. During learning we used beta values beta1=0.9 and beta2=0.999. In total, this network has 8.5 x 10^7 degrees of freedom, which means a capacity of less than 10% of the training set. The network showed no signs of overfitting and reached a final accuracy of 98.7% on the validation set.

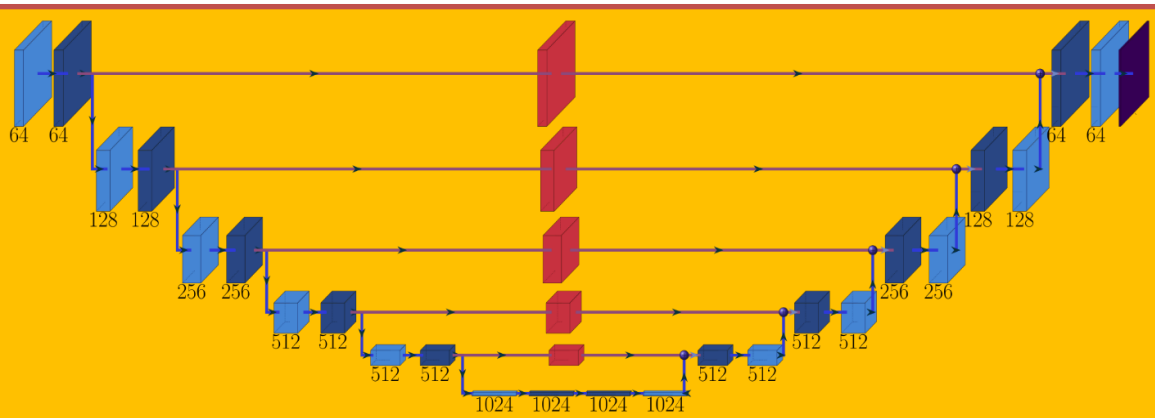


Figure 3.6.4 Schematic of best performing fully convolutional U-Net staircase network. Actual best performing is 2 layers deeper, but has same number of skip connections

Figure 3.6.5 shows a selection of other geometries that were part of the validation set and the performance of our trained network. The predictions regarding optimal mesh densities show a good alignment with the results from the iterative refinement algorithm. A single forward pass through the network for a given geometry takes less than 100ms compared to about 1 hour for the iterative approach and yields reasonably similar mesh densities for even complex shapes. The network therefore could be used as a good first approximation step for getting optimized meshes and therefore save users a lot of fine tuning work when dealing with mesh creation.

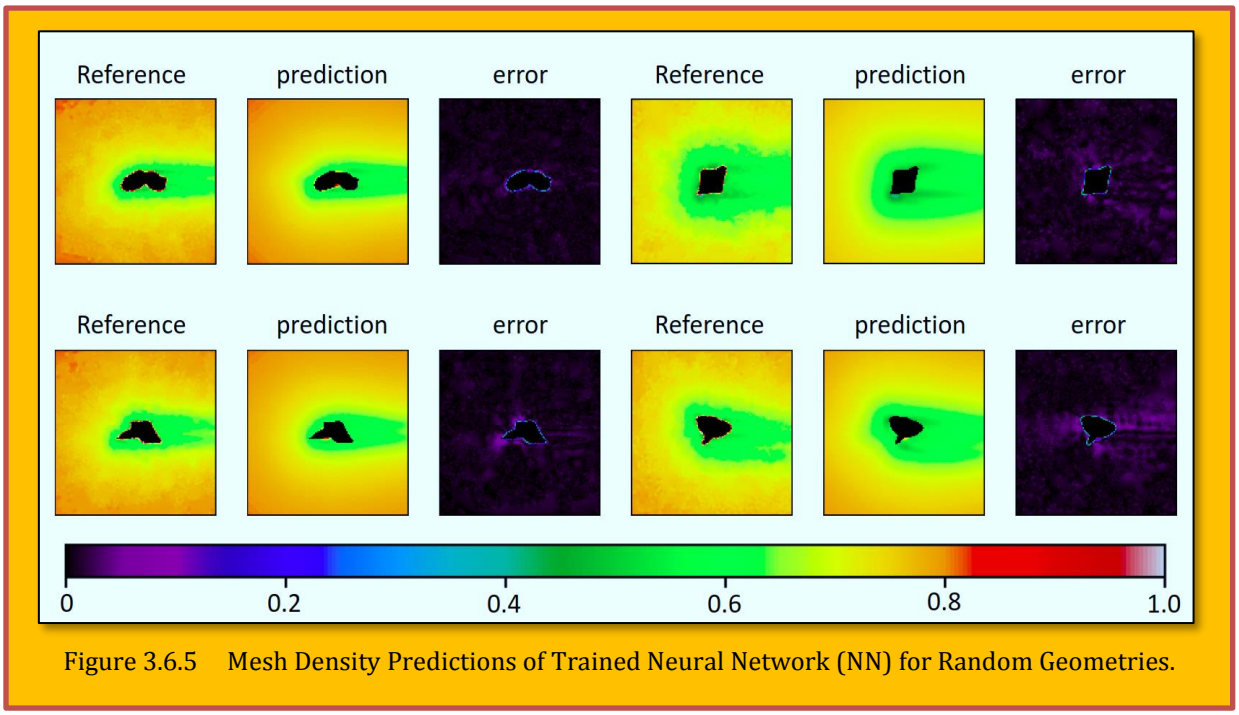


Figure 3.6.5 Mesh Density Predictions of Trained Neural Network (NN) for Random Geometries.

We would like to thank LRZ for the compute time, NVIDIA for granting us a TitanXp GPU for this project and finally our customers at Siemens and CD-Adapco for providing us with the exciting project and continued support.

3.7 Case Study 7 - On Deep-learning-based Geometric Filtering in Aerodynamic Shape Optimization

Authors : *Jichao Li , Mengqi Zhang*
Affiliations : *National University of Singapore, Singapore 117575, Republic of Singapore*
Citation : *Jichao Li, Mengqi Zhang, On deep-learning-based geometric filtering in aerodynamic shape optimization, Aerospace Science and Technology, Volume 112, 2021, 106603, ISSN 1270-9638, https://doi.org/10.1016/j.ast.2021.106603.*
(https://www.sciencedirect.com/science/article/pii/S1270963821001140)

Geometric filtering based on deep-learning models has been shown to be effective to shrink the design space and improve the efficiency of aerodynamic shape optimization. However, since the deep-learning models are trained by existing airfoils, it is criticized that geometric filtering would prevent optimization from finding innovative aerodynamic shapes. This work is conducted to address the concern. By performing 216 airfoil design optimization and several wing design optimization of a conventional wing-body-tail configuration and a blended-wing-body configuration, we find that using the geometric filtering with a lower bound of ~ 0.7 does not exclude innovative aerodynamic shapes that maximize cruise efficiency. The results strengthen the confidence of applying deep-learning-based geometric filtering in aerodynamic shape optimization. Then, two applications of geometric filtering in aerodynamic shape optimization are showcased: the geometric validity constraint and global modal shape derivation. The former is shown to enable aerodynamic shape optimization in a large design space, and the latter provides an efficient parameterization approach to aerodynamic modeling of three-dimensional aircraft configurations.

3.7.1 I. Introduction & Literature Survey

Aircraft shape design increasingly relies on numerical optimization [1–3]. Since suitable geometric freedom is not known in advance, a large design space with hundreds of design variables [3] is usually used to fully tap the potential of aerodynamic shape optimization. This generally leads to a geometric design space that is much larger than required [4]. Consequently, many domains in the design space correspond to abnormal aerodynamic shapes that bring an unnecessary difficulty to aircraft design. Thus, it is of great interest to define a compact geometric design space in aircraft design. Dimension reduction of the shape design variables is a popular choice to obtain a compact design space. With fewer design variables being used, the design space becomes more concise. Robinson and Keane [5] used the singular value decomposition (SVD) to extract modal shapes from a family of supercritical airfoils and found that fewer design variables based on the modal shapes were required to eliminate the shock wave in airfoil design. Poole et al. [6], Masters et al. [7], Li et al. [8] and Kedward et al. [9] further investigated the SVD-based modal parameterization method using the airfoil database of the University of Illinois Urbana-Champaign (UIUC). Allen et al. [10] showed the application of airfoil modes in wing shape optimization. Viswanath et al. [11] presented a dimension reduction method called generative topographic mapping based on nonlinear latent variable models and applied the method to a 30-dimensional airfoil problem. Using the gradient of spatial samples, the active subspace method (ASM) [12] provides another approach to dimension reduction of the design variables. Li et al. [13] reduced the 220 shape design variables to four ASM modes in a wing design shape optimization problem and found surrogate-based optimization in the reduced design space achieved a better solution than that solved in the original design space. Recently, deep-learning techniques have been used to extract low-dimensional latent variables from a high-dimensional aerodynamic shape design space. Chen et al. [14] applied the generative adversarial networks (GAN) to construct airfoil shape representations and achieved better optimization results by using GAN mode shapes than other parameterization methods. Du et al. [15] coupled B-spline with GAN to drive *BSplineGAN* modes for airfoil design. A common criticism of the dimension reduction approach, no matter which method is used to derive modal shapes, is that the suitable number of modes to use is

not known priorly. Using insufficient modes, the performance of aerodynamic shape optimization may be significantly reduced. Using too many modes loses the compactness of the design space. Another approach to establishing a compact design space is to use validity functions that filter out abnormal aerodynamic shapes. This approach does not reduce the number of design variables but alternatively shrinks the design space using computationally cheap models. Kedward et al. [16] proposed a constraint on curve derivatives to ensure smooth shapes in aerodynamic shape design, which improved both the optimization convergence rate and the optimization result in *Aerodynamic Design Optimization Discussion Group (ADODG)* Case 1. Bons et al. [17] used a curvature constraint in the design optimization of the Aerion AS2 supersonic business jet to improve the shape smoothness for ease of manufacture. Li et al. [8] defined a series of marginal functions between the dominant airfoil modes and higher-order modes derived from the UIUC airfoils, and the functions successfully excluded abnormal airfoils from the design space. However, suitable bounds of the curvature-based constraints are usually not known in advance, and there was no guarantee that the margin-based validity functions [8] did not filter out realistic airfoils. The desired validity model should be an accurate discriminator of geometric abnormalities. Benefiting from the strong learning capability of deep-learning models, Li et al. [4] developed such a validity model using the convolutional neural networks (CNN). The model provided validity scores on geometric abnormalities by checking airfoils and wing sectional shapes. Nevertheless, the CNN-based validity function may not cover the desired design space of geometric innovation beyond the UIUC airfoils, because the CNN model is trained via the airfoil GAN model, which is pre-trained by UIUC airfoils. Even worse, if modal collapse occurs in the GAN model, the geometric filtering model may exclude conventional airfoil shapes. Thus, the deep-learning-based filtering model [4] faces criticism: Does the geometric filtering model prevent finding the optimal designs with innovative aerodynamic shapes? This work aims to address the concern on the deep-learning-based geometric filtering and showcase the appealing benefits of applying it in aerodynamic shape optimization. First, we analyze the mode collapse issue in the airfoil GAN model and propose to use the Wasserstein GAN in the generation of realistic airfoils. In order to avoid over-fitting, a large number of training airfoils are generated to train the CNN-based discriminative model for geometric filtering. Then, to investigate whether the optimal designs are excluded by the filtering model, we perform a series of airfoil design optimizations based on different flight missions of commercial aircraft. More persuasively, such investigations are performed in the aerodynamic shape optimization of a wing-body-tail configuration, the Common Research Model (CRM), and a blended-wing-body (BWB) configuration. After analyzing the results, we perform aerodynamic shape optimization of a unit circle to highlight the necessity of geometric filtering in a large high-dimensional design space. Moreover, global wing modes of the CRM and BWB configurations are derived to showcase the utility of deep-learning-based geometric filtering in three-dimensional aircraft modeling.

3.7.2 II. Deep-learning-based Geometric

3.7.2.1 Filtering A. Wasserstein GAN for Airfoils

To train the geometric filtering model, a large number of airfoils, both realistic and abnormal ones, are required. However, about 1600 airfoils can be found from publicly accessible databases worldwide, which may be much less than required, and some of them cannot be used due to missing data [18]. GAN can produce similar synthetic versions of the real dataset by adversarial training a generative model and a discriminative model. Via minimizing the distance between distributions of the training data and the synthetic data, GAN models have been successfully used in the generation of realistic and even hyper-realistic synthetic images and voices [19]. In the aerospace field, various GAN models have been used in airfoil parameterization [15] and shape optimization [4]. To generate a large number of realistic airfoils for the training of the geometric filtering model, we use GAN with a CNN-based generative model, which has been shown to be robust in producing smooth airfoil shapes [4]. The details of this model are explained in the appendix.

One major concern with GAN is the mode collapse issue. When a GAN model encounters mode collapse, the synthetic airfoils are distributed in a small domain and cannot represent the design space spanned by the training airfoils.

Using static metrics like the inception score and the maximum mean discrepancy, Li et al. [4] showed that the model collapse issue was alleviated by normalizing the input data. Nevertheless, such an issue may still exist. To intuitively show whether there is a mode collapse, we make a comparison of the spatial distributions of training airfoils and synthetic airfoils. Since each airfoil is represented by hundreds of points, it is noisy and difficult to compare within the high-dimensional data. We adopt the airfoil mode method [8] to represent these airfoils and alternatively compare the coefficient

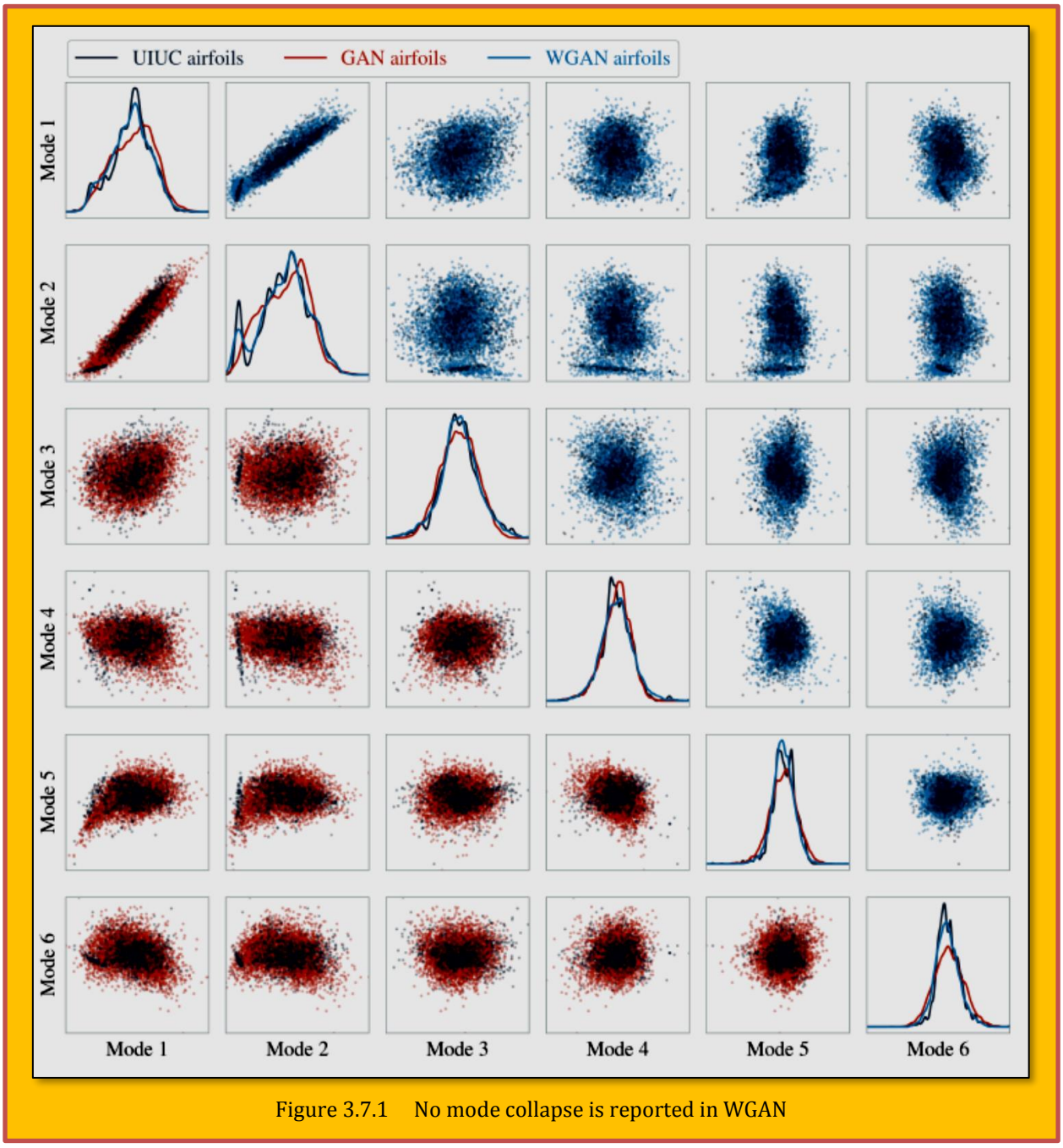


Figure 3.7.1 No mode collapse is reported in WGAN

distributions of dominant airfoil modes. For an arbitrary airfoil y, the mode coefficients c can be computed using the orthogonal mode basis Φ, i.e., $c = \Phi^T y$.

We generate 5000 synthetic airfoils using the GAN model of Li et al. [4], and as shown in **Figure 3.7.1**, their mode coefficients are compared with those of UIUC airfoils. In the diagonal sub-figures, the coefficients of the six dominant modes are compared by using one-dimensional probability distribution curves. Significant distribution differences are reported in the first, second, and fifth modes. In the lower triangle sub-figures, to have a more intuitive comparison, UIUC airfoils (black dots) and GAN synthetic airfoils (red dots) are marked based on their mode coefficients in the corresponding two-dimensional planes. Although the airfoil GAN model presented by Li et al. [4] does not show significant mode collapse, there are still some domains that cannot be captured by the GAN model, which may be regarded as a slight mode collapse issue.

To overcome the issue, we use the Wasserstein GAN (WGAN) proposed by Arjovsky et al. [20] to train the synthetic airfoil generator. In WGAN, the Wasserstein distance is used as the metric among distributions of the real dataset (P_r) and the generated samples (P_g). Based on the Kantorovich-Rubinstein duality, the Wasserstein distance can be defined as

$$W(P_r, P_g) = \frac{1}{K} \underbrace{\sup}_{|f|_L \le K} \left(\mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_g}[f(x)] \right)$$

Eq. 3.7.1

where sup is the least upper bound and E represents the expected value. f is a K-Lipschitz continuous function and is implemented by the discriminator in WGAN. Thus, the discriminator is not directly used as a discriminative model to tell whether a sample comes from the real data or the generated data. Instead, it is trained to learn a K-Lipschitz continuous function that estimates the Wasserstein distance between real and generated samples. We use the same CNN-based generator and discriminator as in the GAN model [4] to train the WGAN model, and the same 1407 UIUC airfoils are used as the training data of the WGAN model. To enforce the Lipschitz constraint on the discriminator, the gradient penalty approach proposed by Gulrajani et al. [21] is used. As shown in **Figure 3.7.1**, WGAN airfoils are distributed in all domains of UIUC airfoils. This implies that the airfoil WGAN model accurately captures the underlying distribution of UIUC airfoils and does not suffer from mode collapse. Besides, the WGAN model exhibits an exploration capability. Some WGAN airfoils are distributed in the adjacent domains with no UIUC airfoils. The feature may help to enable the geometric filtering model to contain innovative shapes. We use WGAN to generate realistic training airfoils for the geometric filtering model.

3.7.2.2 B. Geometric Filtering Model

A large number of abnormal airfoils are required to train the geometric filtering model. Inspired by the abnormal sampling results in a large design space [8], we generate abnormal airfoils by using the Latin hypercube sampling (LHS) method. The free-form deformation (FFD) [22] method is used for airfoil shape deformation. Since there is a large number of samples to generate, a Python script is written to perform this process automatically. First, the FFD control box for each baseline airfoil is generated. The bounds of each FFD control points are defined as 20% of the local thickness. Then, for each baseline airfoil, 30 sample airfoils are generated using LHS. In the end, 42,210 sample airfoils are generated in total. As shown in **Figure 3.7.2**, most of the sample airfoils can be visually identified to be abnormal.

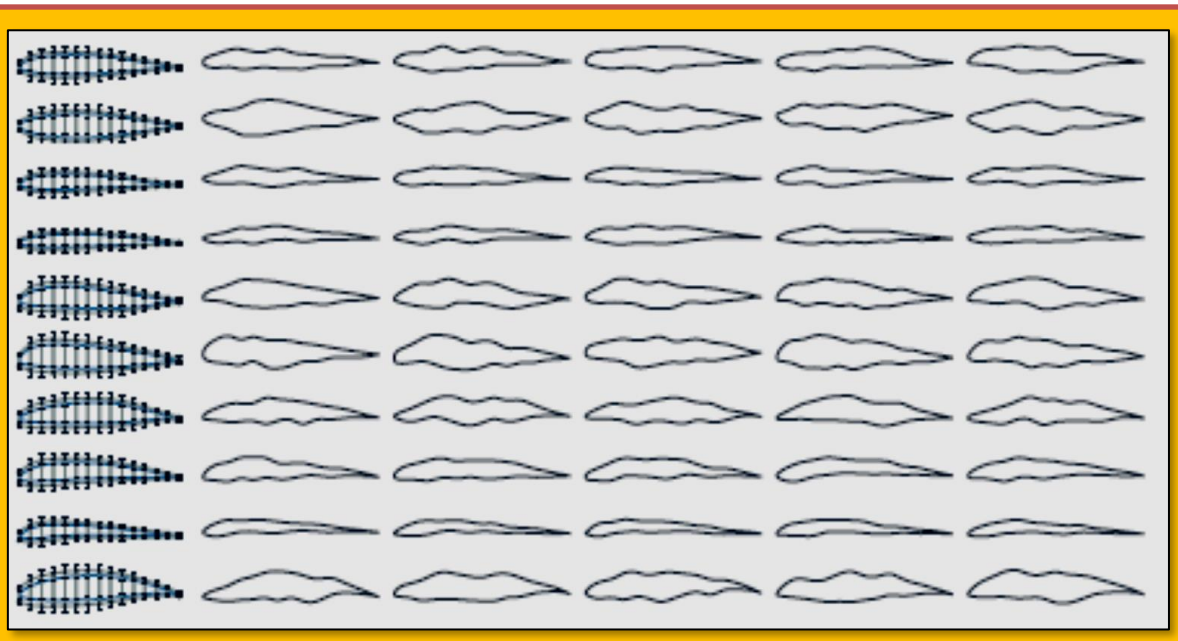


Figure 3.7.2 Abnormal airfoils generated by perturbing FFD control points

We seek to construct a filtering model to automatically detect geometric abnormality for airfoil-like shapes. One important application of the geometric filtering model is high-fidelity aerodynamic shape optimization in order to constrain the search domain of the optimizer. Thus, it is important to have a fast and accurate evaluation of geometric abnormalities. Commonly-used methods include nearest neighbors, decision trees, random forest, neural networks, and so on. The nearest neighbors method may be too costly due to a large number of training data. To maintain the capability of being used in gradient-based optimization, the model should be differentiable. Thus, tree-based methods such as the decision tree and random forest are not suitable. In this work, we merely investigate models based on neural networks because the gradient can be accurately evaluated by automatic differentiation. Two kinds of neural networks, multi-layer perceptron (MLP) and CNN, are investigated. Neural networks are trained by minimizing the loss function.

In the training of neural networks, the loss functions get smaller as a sign of accuracy improvement. We investigate two typical loss functions, the binary cross-entropy (BCE) and the mean absolute error (MSE). The loss functions are defined as

$$f_{BCE} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i) \quad , \quad f_{MSE} = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Eq. 3.7.2

where y_i and $\hat{y}_i$ are the original score (label) and predicted score of the ith airfoil. Each realistic airfoil (WGAN airfoil) is labeled as 1.0. Abnormal airfoils are labeled to be 0.0 and -1.0 for models trained with f_{BCE} and f_{MSE}, respectively.

We investigate MLP models with different layers and numbers of neurons. However, we find that the prediction accuracy is rather low whatever hyperparameters are used. The results imply that it is unsuitable to use MLP to detect geometric abnormality of airfoils. A possible reason may be that it is difficult for MLP networks to learn the underlying features of airfoils. CNN is effective in extracting airfoil geometric features [4]. We use a CNN model with four convolutional layers to extract the underlying features step by step. The stride step is two in each convolutional layer to realize the

down sampling process. The filter size (n_{size}) and the number of filters (n_{filter}) in each layer are important hyperparameters in CNN.

We investigate the influence of the two hyperparameters in the training of the geometric filtering model. The training processes with different hyperparameters and loss functions are shown in **Figure 3.7.3**. Sub-figures in the first and second rows are CNN models trained using f_{MSE} and f_{BCE}, respectively. Sub-figures in different columns are models with different n_{filter}. In each sub-figure, the models with different n_{size} are distinguished by line colors, and the solid and dashed lines represent the loss functions in the training and testing datasets, respectively. The accuracy of the CNN models increases with the rise of n_{size} and n_{filter}, and there is no sign of over-fitting. The training of CNN models with both loss functions tends to be unstable when a large n_{size} or n_{filter} is used. We find that CNN models using the BCE function cannot provide validity scores with a smooth transition from realistic airfoils to abnormal airfoils. This feature makes BCE-based CNN models unsuitable as a constraint in gradient-based aerodynamic shape optimization. Based on the results, we use the CNN model with n_{size} = 5 and n_{filter} = 64 trained by the MSE loss function in this work.

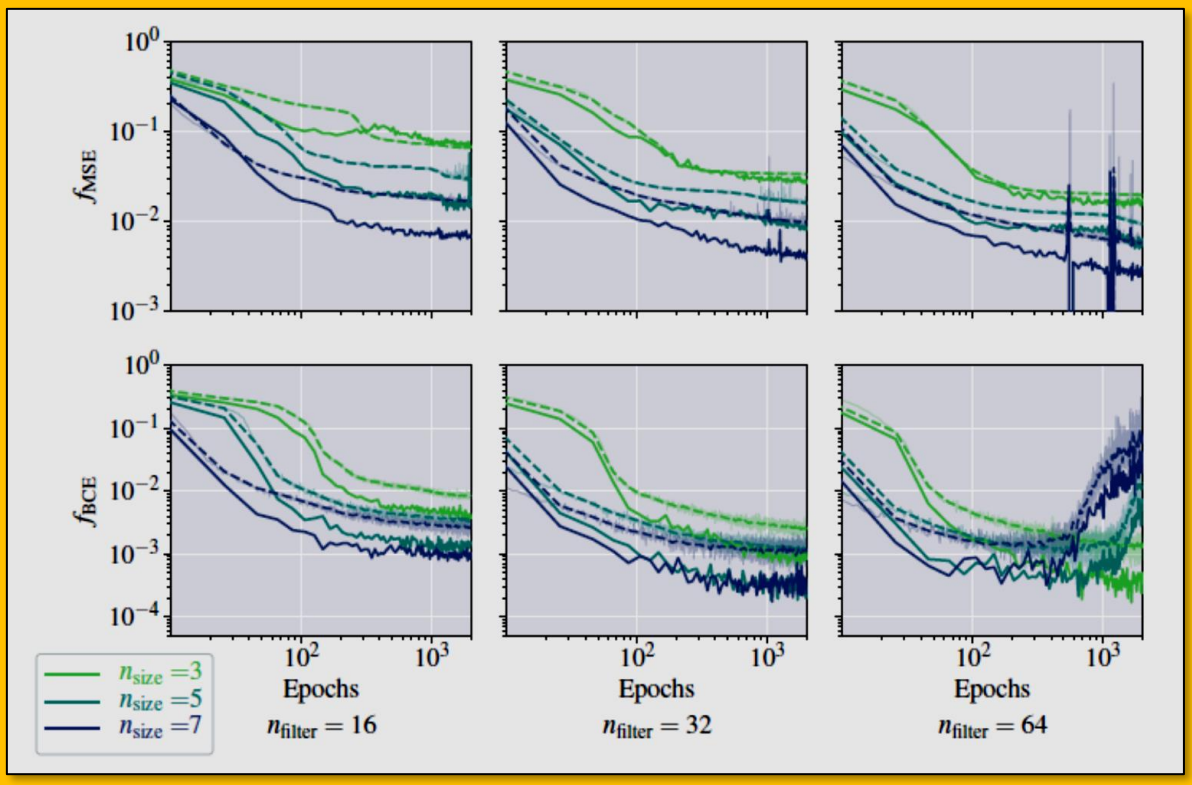


Figure 3.7.3 Investigation of CNN hyperparameters in the training of airfoil filtering models

3.7.3 III. Aerodynamic Shape Design Optimization

3.7.3.1 A. Gradient-based Optimization Framework

We use the MACH-Aero framework[101] to investigate possible shape deformations in aerodynamic design optimization of commercial aircraft. This framework has shown to be robust in many state-of-the-art aerodynamic shape design optimization work [23–26] including the ADODG standard cases [3, 27, 28]. A summary of relevant work can be found in [29]. The sequential least squares programming (SLSQP) algorithm implemented in *pyOptSparse* [30, 31] is used as the optimizer.

[101] https://github.com/mdolab/MACH-Aero

ADflow [32, 33], which is an open-source finite-volume structured multi-block computational fluid dynamics (CFD) code, is used to solve the Reynolds-averaged Navier-Stokes equations with a Spalart–Allmaras turbulence model [34]. The adjoint solver [35] embedded in *ADflow* is used to compute the gradient. The analytic inverse-distance method implemented in *IDWarp*[102] is used to automatically deform the volume mesh. FFD [22] is used as the parameterization method for aerodynamic shapes. In this section, the filtering model is used to check the optimized shapes, so it is not coupled with the optimization framework.

3.7.3.2 B. Airfoil Shape Design Optimization

A series of airfoil design optimization problems are defined based on the ADODG Case 2 and various flight missions of commercial aircraft. The key parameter bounds of the missions considered are shown in **Table 3.7.1**. Several assumptions based on industrial practice are used to transfer the aircraft design problem to an airfoil design problem:

	Mach number	Altitude(m)	Mass(kg)
Lower bound	0.80	9,000	180,000
Upper bound	0.90	12,000	220,000

Table 3.7.1 Bounds of key parameters of flight missions considered

1) The wings provide 87% lift;
2) The lift constraint from wing design to airfoil design: $C_l = 1.1 \times C_L / \cos^2 \Lambda$, where Λ is the wing sweep angle;
3) The operating Mach number for airfoil design is $M = M_{cruise} \times \cos \Lambda$.

For an arbitrary mission, the lift constraint for wing design is

$$C_L = \frac{0.87 \times mass \times g \times 0.5}{\frac{1}{2}\rho U^2 S} \quad \textbf{Eq. 3.7.3}$$

where g is the gravitational acceleration; ρ, U, S are the air density, cruise speed, and wing area. ρ and g are functions of the altitude; U is a function of the altitude (H) and Mach number (M), which is defined by the ICAO Standard Atmosphere. We use the wing area and wing sweep angle of the CRM aircraft to compute M and C_l.

	Functions	Description	Quantity
Minimize	C_d	Drag coefficient	1
With respect to	α	Angle of attack	1
	y_{shape}	y displacements of control points	30
		Total design variables	31
Subject to	$C_l = C_l^{con}$	Lift constraint	1
	$C_m \geq -0.092$	Moment constraint	1
	$A \geq s \times A_{RAE2822}$	Area constraint	1
	$t_{TE} \geq 0.25\%$	Trailing edge thickness constraint	1
	$\Delta y_{LE}^{upper} = -\Delta y_{LE}^{lower}$	Fixed leading edge constraint	1
	$\Delta y_{TE}^{upper} = -\Delta y_{TE}^{lower}$	Fixed trailing edge constraint	1
		Total constraints	6

Table 3.7.2 Airfoil shape design optimization problem statement

[102] https://github.com/mdolab/idwarp

We consider 6 x 4 x 3 = 72 flight missions, which are evenly distributed in the Mach-altitude-mass space with bounds defined in **Table 3.7.1**. Airfoil design optimization is performed to minimize the drag coefficient. Since the choice of starting point does not affect the optimized shape in airfoil design [8, 27, 36], we use the same baseline, the RAE2822 airfoil, in all airfoil design cases. The FFD control points, the baseline airfoil, and the CFD mesh are shown in **Figure 3.7.4**. In addition to the lift constraint, a pitching moment constraint is involved. An area constraint is specified to ensure enough space for the wing structure and fuel tanks. Different airfoil areas are required for different wing sections. In the CRM wing, the areas decrease from 1.17 x $A_{RAE2822}$ to 0.80 x$A_{RAE2822}$ from the wing root to the wing tip, where $A_{RAE2822}$ is the area of the RAE2822 baseline airfoil. We consider three area constraints with s = 0.8, s = 1.0, and s = 1.2 to investigate various area demands. Thus, 216 airfoil design problems are considered in total. The details of airfoil design optimization are described in **Table 3.7.2**.

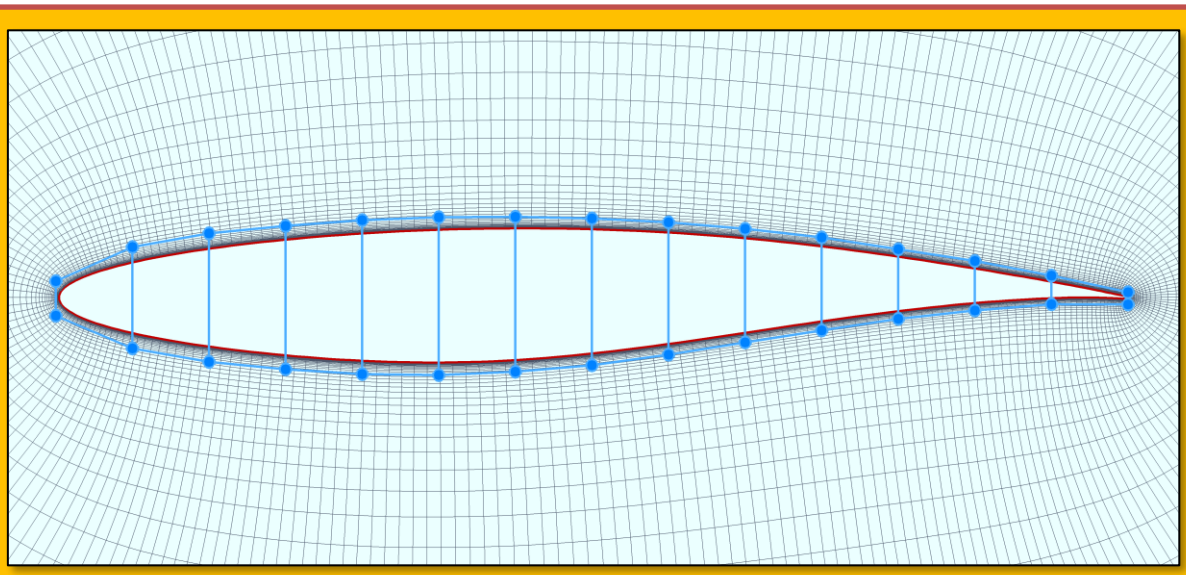


Figure 3.7.4 FFD control points and the CFD mesh for airfoil design

The optimized airfoils are shown in **Figure 3.7.5**. Due to the large number of airfoils, it is difficult to show all details in one figure. So we provide all the optimized airfoils, including the initial and optimized aerodynamic coefficients, in Mendeley Data [37]. Different optimal airfoil shapes are desired in different flight missions. When operated in missions with high Mach numbers, the upper surface of the airfoil tends to be flat to reduce the strength of shock waves, which makes the optimized airfoils similar to supercritical airfoils. This phenomenon becomes more obvious with the increase of the flight altitude, where a larger C_l is required due to the decrease of the air density and sound speed. The increase of aircraft mass leads to a direct increase in the lift constraint. When the mass equals to 220000 kg and H = 12000 m, the optimized airfoils become sunken at the lower surface near the leading edge to provide more lift without violating the pitching moment constraint. This character is different from typical supercritical airfoils, which tend to generate more lift by

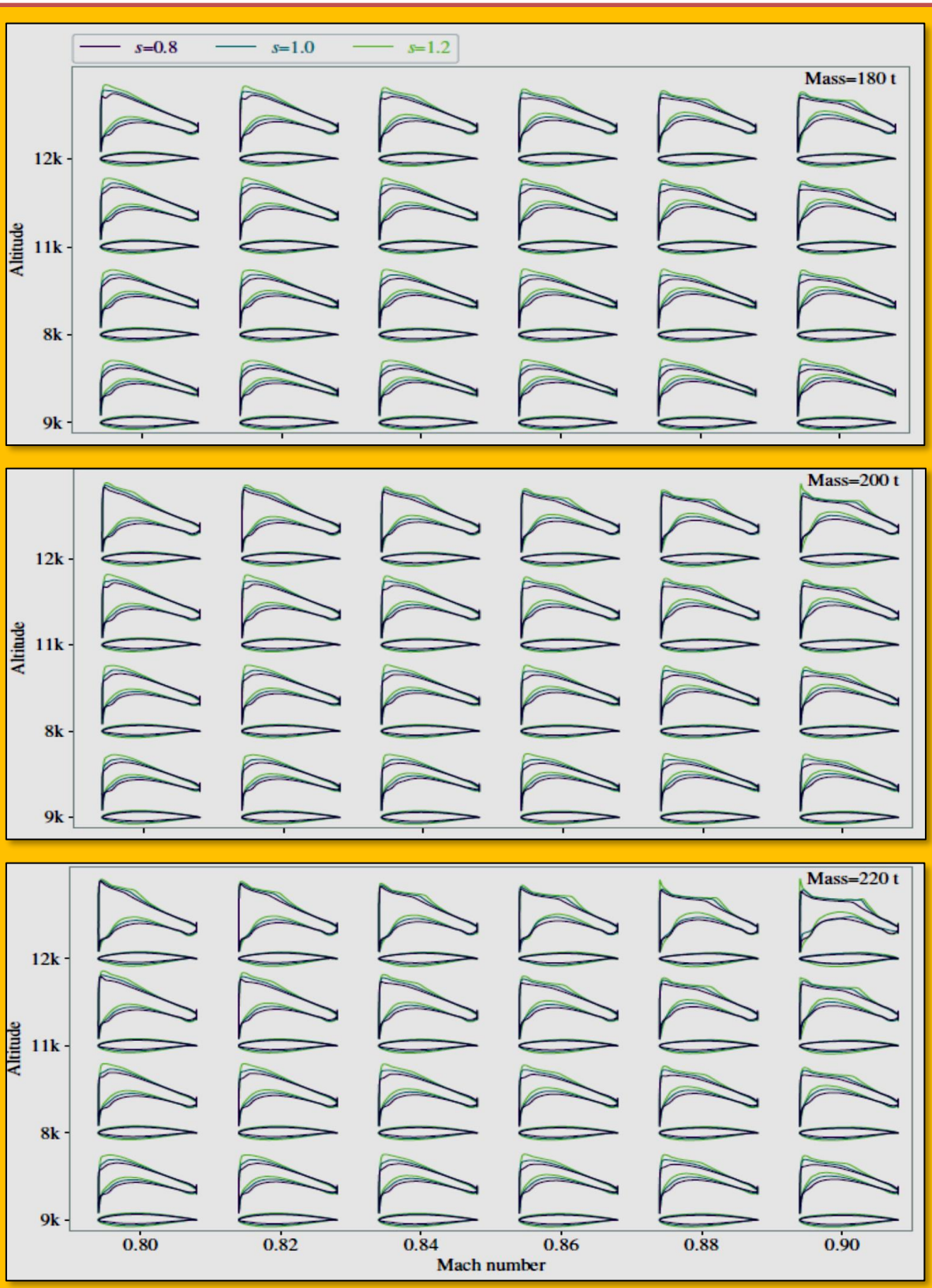


Figure 3.7.5 Optimized airfoils for different flight8missions subject to different area constraints

increasing trailing edge camber. The choice herein may be due to the strict pitching moment

constraint and a lack of constraints for low-speed performance [18].
The optimized airfoils for different missions show rich shape variations and geometric innovations. We evaluate these airfoils using the geometric filtering model, and the validity scores are shown in **Figure 3.7.6**. The score distribution of optimized airfoils is a bit different from that of UIUC airfoils. There is a sharp decrease in the number of optimized airfoils with $S_{validity} > 1.15$. Nevertheless, optimized airfoils are inside the scope of UIUC airfoil scores. The smallest score of optimized airfoils is 0.74, which is clearly different from the scores of abnormal airfoils ($-1.3 < S_{validity} < 0.0$).
Three optimized airfoils with the smallest scores are shown in **Figure 3.7.6**. The airfoils are optimized at missions with H = 12 000 m and show the same innovative character in the leading edge. This implies that the innovative character would reduce the geometric validity score. Nevertheless, using the lower bound of UIUC airfoil scores as the constraint does not prevent aerodynamic shape optimization from finding these innovative airfoils.

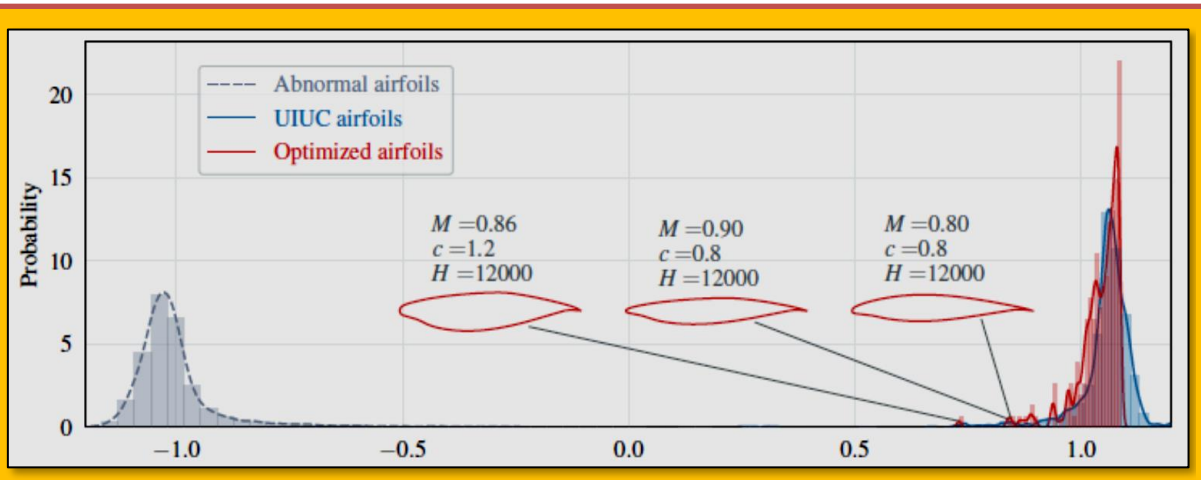


Figure 3.7.6 Scores evaluated by the geometric filtering model

3.7.3.3 C. Aircraft Wing Shape Design Optimization

The airfoil design problems discussed in the previous section are defined using industrial simplifications, so they may not truly reflect the shape deformations in aircraft wing design optimization. To investigate whether the deep-learning-based model would filter out innovative wing shapes, we consider two wing design optimization problems based on the CRM aircraft [38] and a BWB configuration [23], respectively. The optimization is performed to improve cruise efficiency at M = 0.85 and H = 10670 m by minimizing the drag subject to a lift constraint (C^{con}_L). For the CRM aircraft, the lift constraint is chosen as $C^{con}_L = 0.500$ by referring to the AIAA Aerodynamic Design Optimization Discussion Group Case 5. For the BWB aircraft, due to the increase of the wing area, a much smaller lift coefficient is required. By assuming a payload of 800,000 lbs, we use a lift constraint $C^{con}_L = 0.20056$ in the BWB design. A pitching moment constraint ($C_M = 0.0$) is used in the optimization of both aircraft. Volume constraints are involved to make the optimized aircraft have the same wing volume as the initial baseline. Besides, two kinds of thickness constraints, a loose thickness constraint and a strict thickness constraint, are investigated. Loose thickness constraints allow the wing thicknesses to decrease to a level not smaller than 25% of the baseline thicknesses. This choice can gain a significant drag reduction in the transonic regime, but the optimized design may sacrifice too much low-speed aerodynamic performance [18]. So, we also investigate strict thickness constraints where the thicknesses are not allowed to decrease in the optimization. The FFD control points of the two aircraft configurations are shown in **Figure 3.7.7** and **Figure 3.7.8,** respectively. Sectional airfoils distributed across the wing span are monitored to investigate the concern on geometric filtering.

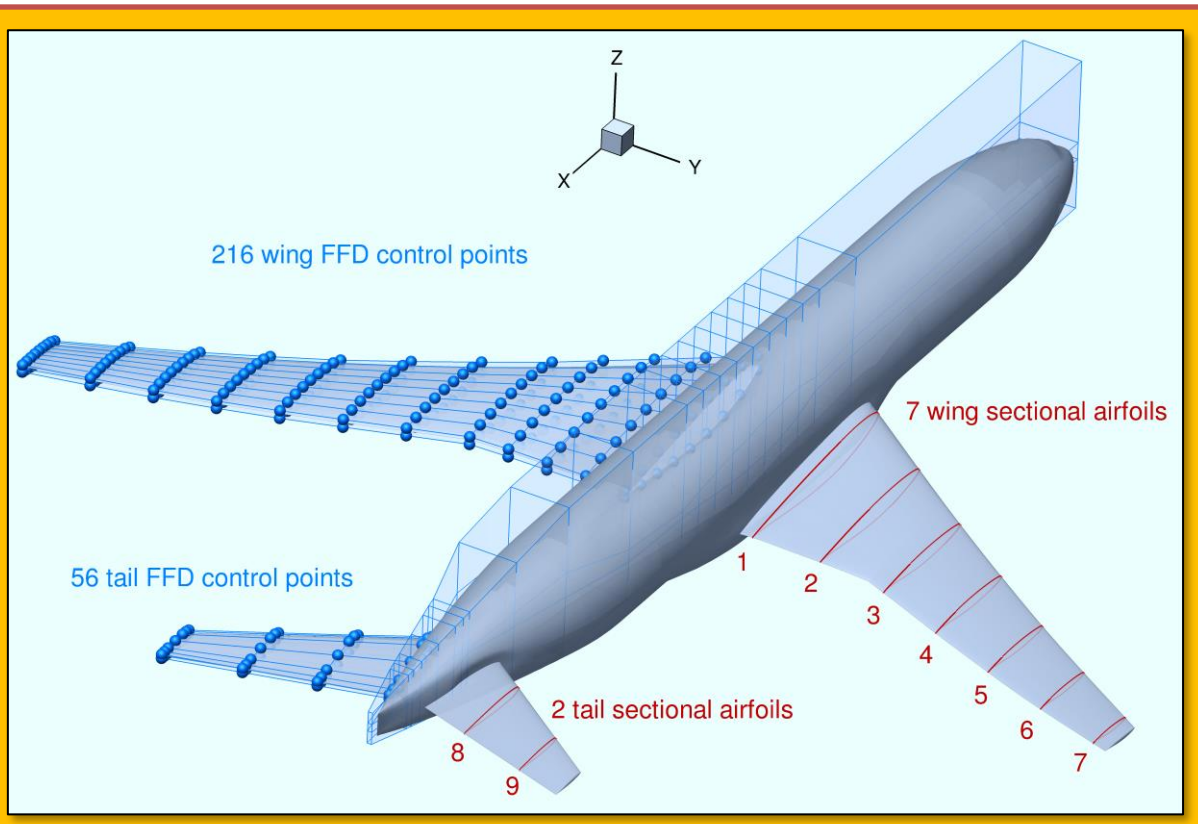


Figure 3.7.7 Nine sectional airfoils are monitored in the CRM optimization

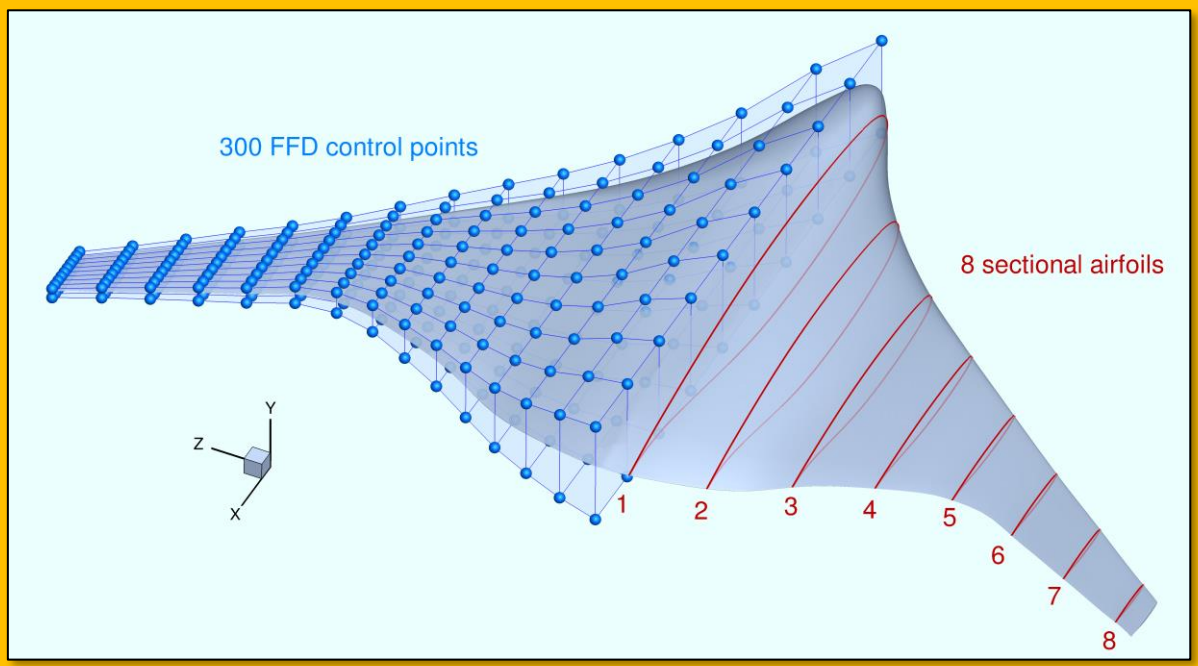


Figure 3.7.8 Eight sectional airfoils are monitored in the BWB optimization

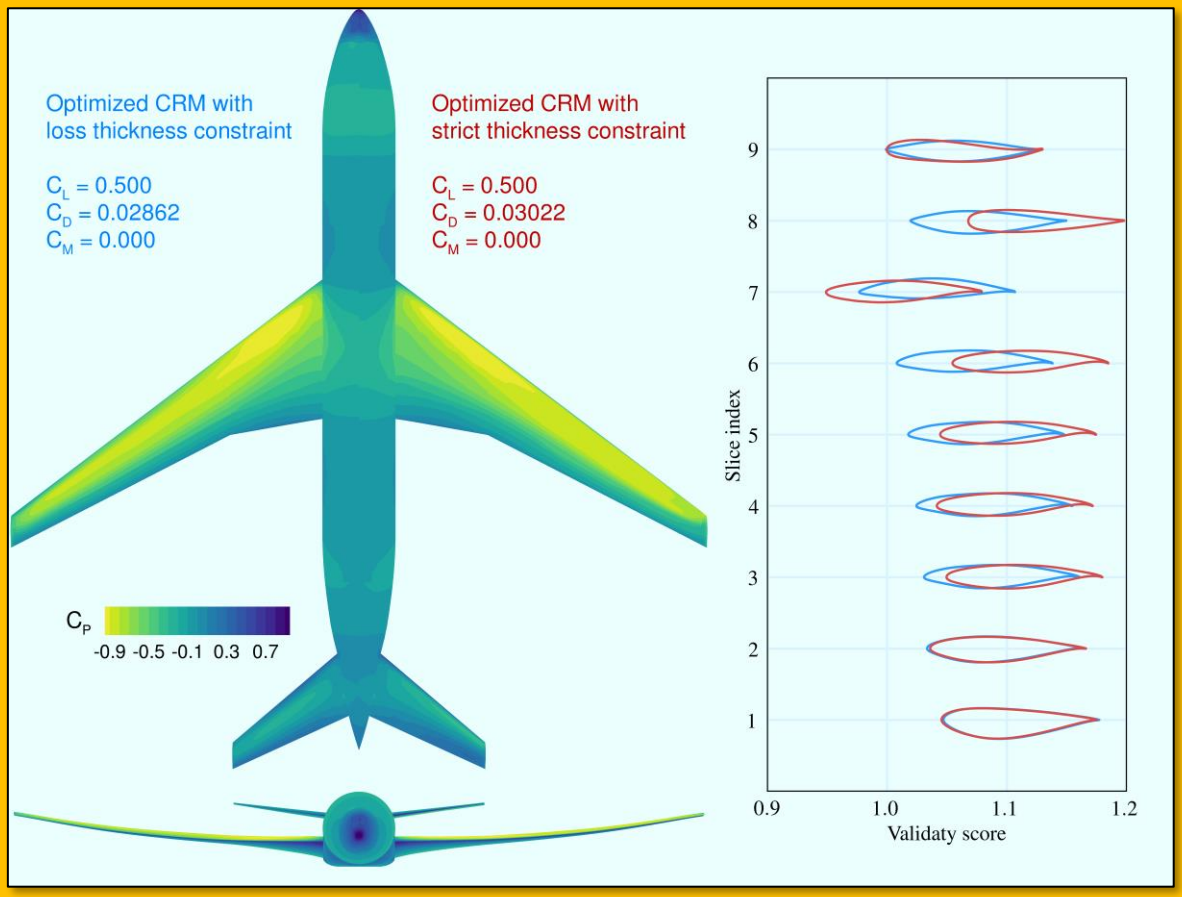


Figure 3.7.9 Geometric filtering constraint with $S_{validity}$ > 1:0 does not filter out the optimal shapes in the CRM design

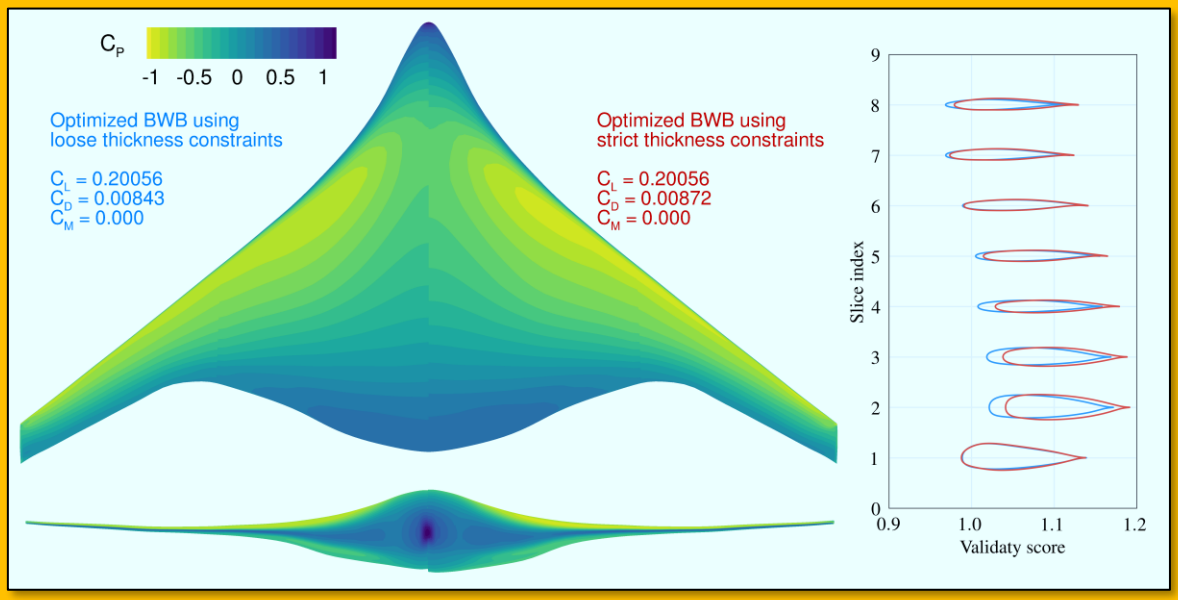


Figure 3.7.10 Geometric filtering constraint with $S_{validity}$ > 1:0 does not filter out the optimal shapes in the BWB design

almost shock-free with the drag coefficient significantly reduced from the baseline value (C_D = 0.0353). The monitored sectional airfoils are shown in **Figure 3.7.9** as well, which are arranged along the horizontal axis based on the validity scores evaluated by the deep-learning filtering model. Using the loose thickness constraints leads to more geometric freedom in design optimization, and the optimized shapes tend to have smaller validity scores. Nevertheless, all wing sectional airfoils in both optimized CRM configurations achieve scores larger than 1.0. Thus, the deep-learning geometric filtering constraint will not filter out the optimal shapes in the CRM design optimization. As shown in **Figure 3.7.10**, two kinds of thickness constraints in the BWB design lead to a significant difference in the optimized shapes. Nevertheless, both optimized aircraft are shock-free and lead to significant drag reductions from the baseline value (C_D = 0.0128). Validity scores of wing sectional airfoils are all larger than 1.0 as well. Similarly to the CRM design optimization, using deep-learning geometric filtering constraint with $S_{validity} > 1.0$ does not filter out these optimal designs.

3.7.4 IV. Applications of Deep-learning-based Geometric Filtering

Multiple tests in Section III show that the deep learning model developed in Section II does not filter out optimal solutions in aerodynamic shape optimization. The results encourage the use of the filtering model to address geometric abnormality issues in aerodynamic shape optimization. In this section, two promising applications based on the model are showcased.

3.7.4.1 A. Geometric validity constraint in aerodynamic shape optimization

Conventional geometric parameterization methods, for example, the FFD method, usually lead to high-dimensional design spaces. Part of the design space corresponds to abnormal aerodynamic shapes with poor aerodynamic performance, which do not contribute to aerodynamic design but bring a significant difficulty to aerodynamic analyses. For surrogate-based optimization methods, Li et al. [4] have shown that using the deep-learning-based geometric filtering model as a validity constraint can significantly improve optimization efficiency in both airfoil and wing shape design. The geometric abnormality issue may not cause too much concern in the aerodynamic design community since gradient-based optimization using the adjoint method is much less sensitive to the curse of dimensionality than surrogate-based optimization. However, we find that gradient-based optimization would also be problematic if a large domain in the design space is associated with geometric abnormalities. Shape design optimization of a bluff body is one of the cases.

In the work of He et al. [27], it was shown that an adaptive FFD method had to be involved to optimize starting from a circular shape. At the beginning of the optimization, the adaptive method defined a simple design space with four geometric design variables so that the circle was quickly modified to an airfoil-like shape by reducing the thickness.

Then the optimization restarted from the airfoil-like shape and the adaptive method switched to a dense design space by adding more control points. However, if a dense FFD control box was used at the beginning, the optimization convergence cannot be achieved [27] even through a robust CFD solver was used [32]. This implies that the geometric issue is an obstacle for the robust gradient-based aerodynamic design optimization method as well.

To showcase the issue, we revisit the optimization problem in He et al. [27]. An optimization from a unit circle is performed using the MACH-Aero framework to minimize the drag coefficient with M = 0.734 and H = 11740 m. The circular shape is parameterized by a dense FFD box with 30 control points. As shown in **Figure 3.7.11**, the optimization fails after 20 CFD evaluations due to failures in CFD analyses of abnormal shapes. These CFD failures are marked as red crosses in **Figure 3.7.11** and the corresponding CFD meshes in these iterations are shown. It can be seen that the optimizer tends to decrease the thickness of the middle part to reduce the drag. This leads to geometric abnormality and brings difficulties to the deformation of CFD meshes. Although a robust inverse

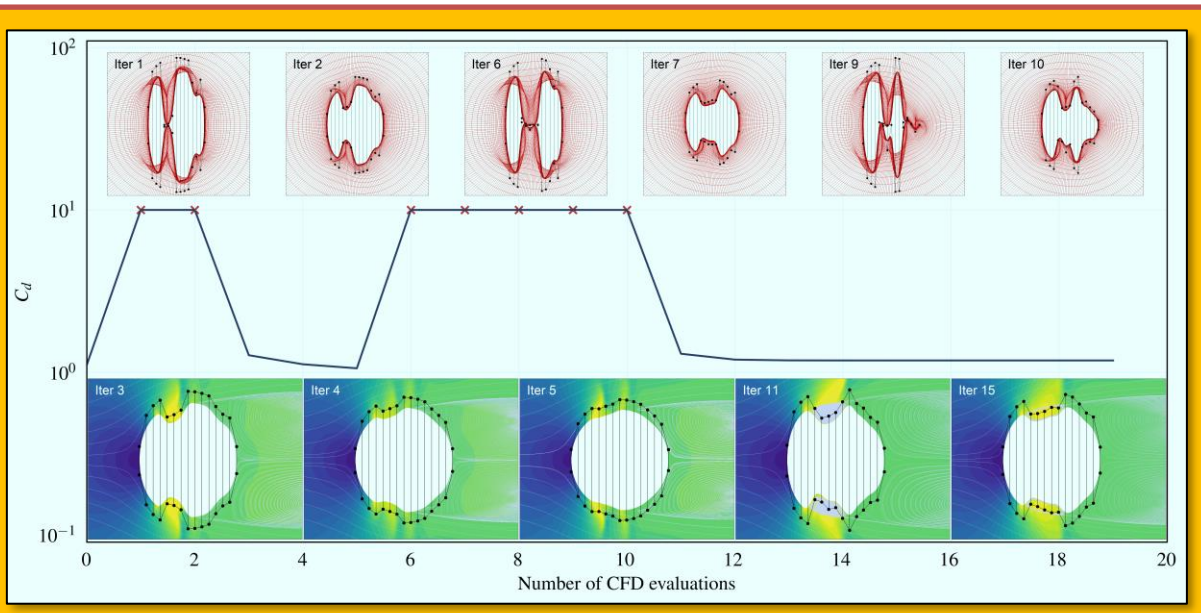


Figure 3.7.11 Gradient-based optimization starting from a circle fails due to too much geometric abnormality

distance mesh deformation method [39] is used, it is difficult to ensure the mesh quality in this circumstance.

The deformed "dumbbell" shapes can be easily identified as abnormal by the deep-learning-based geometric filtering model. We use the geometric validity function as an inequality constraint ($S_{validity} > 0.7$) in the optimization and perform the gradient-based optimization from the unit circle. An area constraint is imposed to ensure the optimized shape with the same area as the RAE2822 airfoil. A lift constraint with $C_l = 0.824$ and a pitching moment constraint with $C_m \geq -0.092$ are used. As shown in **Figure 3.7.12**, the optimization gradually converges to a supercritical airfoil merely using 200 CFD evaluations. Inequality constraints are not necessarily satisfied in the line searching process of SLSQP. Thus, directly using the geometric validity function as an inequality constraint cannot ensure

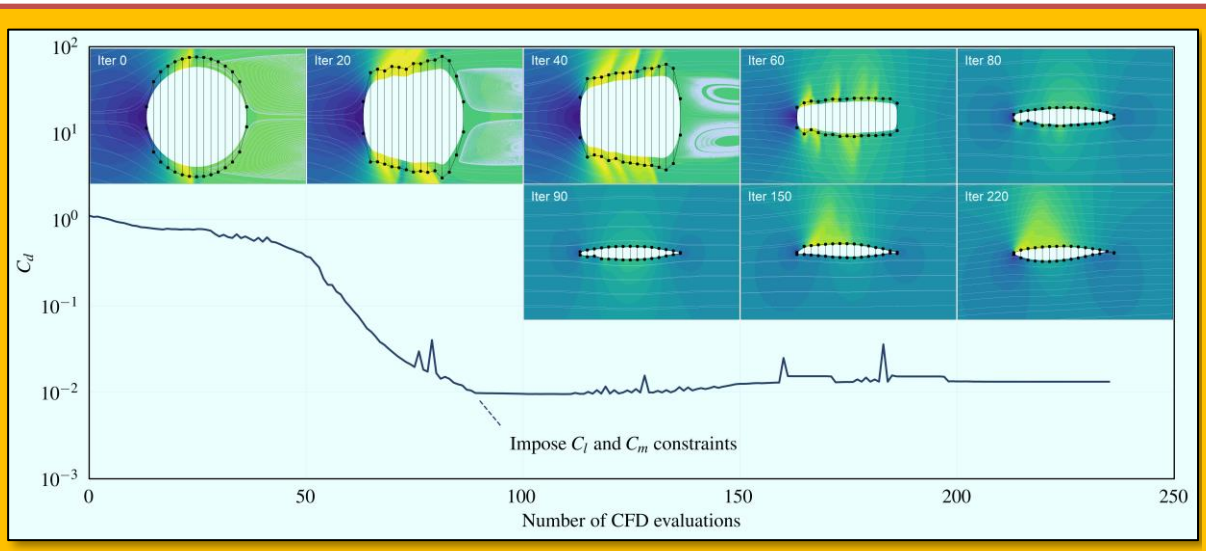


Figure 3.7.12 The deep-learning-based filtering constraint ensures the success of gradient-based optimization from a circle

that “dumbbell” shapes are always avoided during the optimization. To address this issue, we add a judgment before calling the CFD evaluation module. When the validity score of the updated shape is smaller than 0.0, C_d is enforced to a large value, i.e., $C_d = 1.0 - S_{validity}$, without the need of performing CFD simulations. This circumstance merely occurs in the early part of optimization, when the drag is large. For simplification, lift and pitching moment constraints are imposed after the drag is significantly reduced, and before wards, there is no need to estimate C_l or C_m. This approach is not generic, and a modified gradient-based optimization algorithm is on-demand to take full advantage of geometric filtering. Nevertheless, the result shows that deep-learning-based geometric filtering makes gradient-based optimization manage to handle such a challenging case.

3.7.4.2 B. Geometric dimension reduction for aerodynamic modeling

In addition to being used as a geometric validity constraint, the filtering model can be used to derive global mode shapes to construct a compact parameterization for wing-based configurations. The fundamental idea in this context is the deep-learning-based optimal sampling method [40], which combines the GAN airfoil model, the CNN-based filtering model, and an optimization process. First, sectional airfoils distributed in different positions are chosen to represent the wing (or tail) shape. Randomly generated GAN synthetic airfoils are used as the initial sectional shapes.

Then, to make the wing samples satisfy required geometric constraints and ensure the sampling sparsity, an optimization is performed to find the nearest feasible sample wing shape by minimizing the displacement of the wing sectional airfoils. Besides the geometric constraints, the optimization is also subject to the deep-learning-based geometric validity constraints to ensure that the optimized sample wing shapes are realistic. Thus, this sampling method merely selects samples from the feasible domain of the high-dimensional design space.

The optimal sampling method ensures both the sparsity and feasibility of samples, which are a good representation of the desired domain of the high-dimensional design space. We use SVD to derive global mode shapes from the samples. Assuming that m samples are generated, we assemble them in a snapshot matrix

$$A = \begin{bmatrix} a_1^1 - a_1^{baseline} & a_1^2 - a_1^{baseline} & \dots & a_1^m - a_1^{baseline} \\ a_2^1 - a_2^{baseline} & a_2^2 - a_2^{baseline} & \dots & a_2^m - a_2^{baseline} \\ \vdots & \vdots & \ddots & \vdots \\ a_n^1 - a_n^{baseline} & a_n^2 - a_n^{baseline} & \dots & a_n^m - a_n^{baseline} \end{bmatrix}$$

Eq. 3.7.4

where $a^{baseline}_i$ is ith deformable coordinate (z for the CRM and y for the BWB) of the baseline wing. Performing SVD on A, we obtain

$$A = U\Sigma V^T$$

Eq. 3.7.5

where columns in U correspond to global mode shapes of the wing. The global mode shapes have been used in a wing design optimization problem [40, 41], and the compact design space defined by global wing modes makes the adjoint solver unnecessary in high-dimensional wing design. For the CRM and BWB aircraft considered in this work, we generate 200 samples to derive global wing modes. The dominant global wing modes for both configurations are shown in Error! Reference source not found. and **Figure 3.7.13**. The first modes bend the wing sections in a non-parallel manner. Different from local section-based modal shapes, global modal shapes can lead to intersectional deformations. So, the global modal shapes may improve the efficiency of aerodynamic shape parameterization. Its application in aerodynamic modeling of three-dimensional wing-based configurations is worth looking forward to.

3.7.5 V. Conclusions

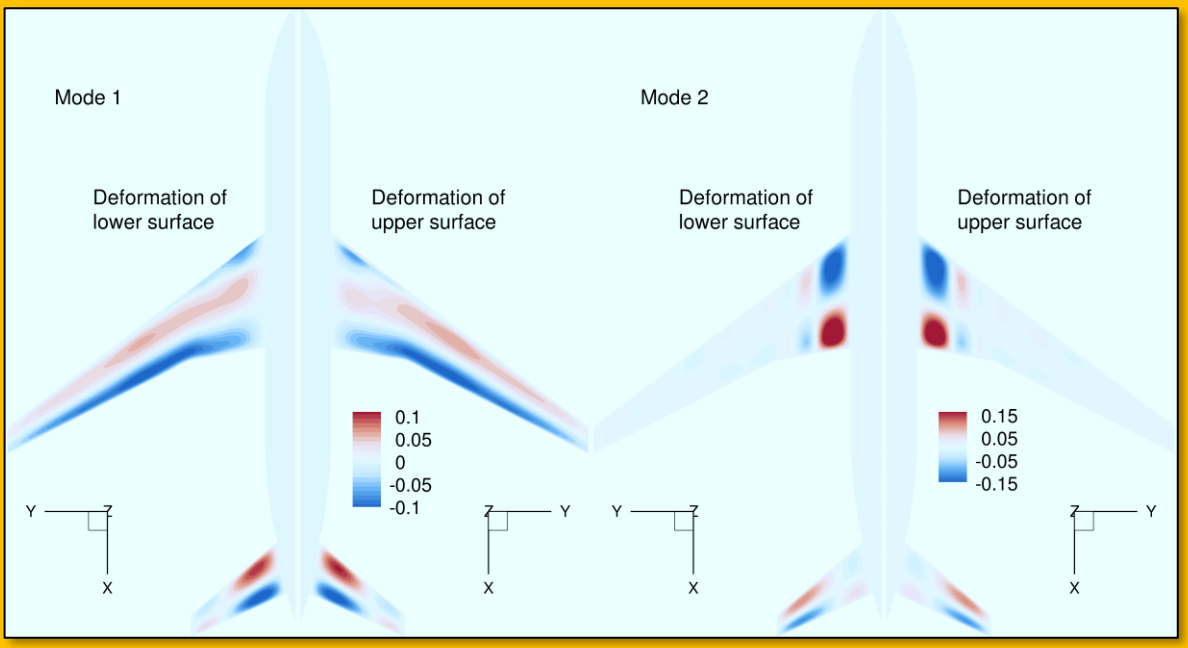


Figure 3.7.14 Dominant global mode shapes for the CRM wing and tail optimization

This work addresses the concern of applying the deep-learning-based geometric filtering model in aerodynamic shape design optimization. The WGAN model is proposed to generate synthetic airfoils. Different types of neural networks are investigated in the training of the geometric validity model. To show whether innovative optimal designs would be filtered out by the validity model, various airfoil and wing shape design optimizations are performed. Airfoil design optimizations are based on 72 flight missions of commercial aircraft and three area constraints. Wing design optimization of two aircraft configurations, the conventional CRM wing-body-tail and an innovative BWB, are considered.

The proposed WGAN-based generative model addresses the model collapse issue that occurs in the airfoil GAN model. WGAN synthetic airfoils are not only a

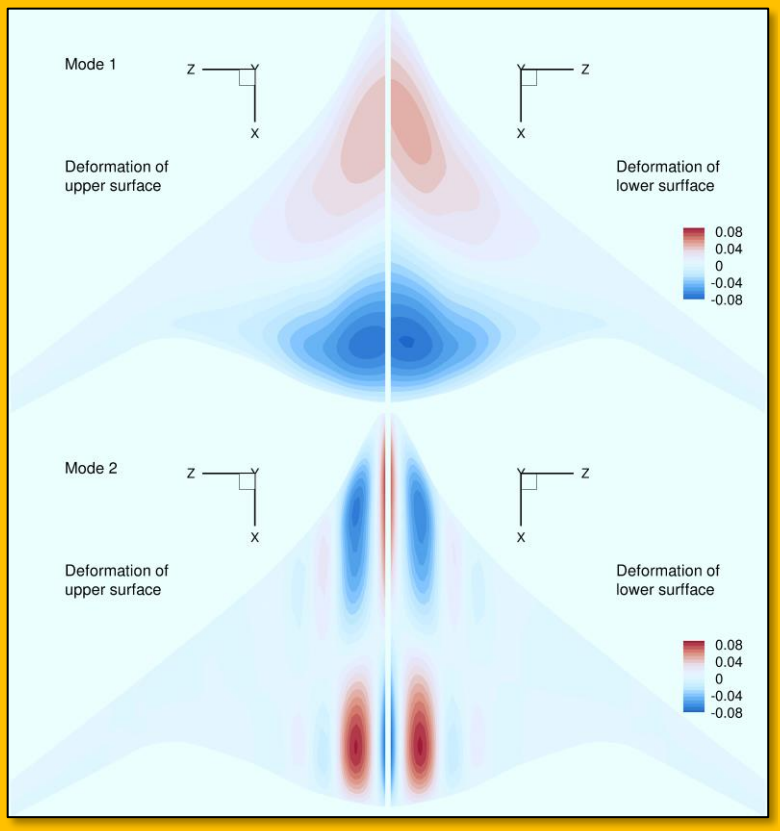


Figure 3.7.13 Dominant global mode shapes for the BWB optimization

good representation of the training airfoils but also are distributed in domains that few training airfoils exist. Thus, the WGAN model has an extrapolation capability, and the synthetic airfoils convey more geometric information than the airfoils by which the model is trained. For the geometric validity model, we find that a CNN-based discriminator trained with the MSE loss function is preferable. This choice results in a smooth and accurate discriminative model for geometric abnormalities. The validity scores of the 216 optimal airfoils are all greater than 0.7 and inside the scope of the UIUC airfoil scores. The wing sectional airfoils in the optimized CRM and BWB earn higher validity scores no matter loose or strict thickness constraints are used. Using a suitable validity constraint, say $S_{validity} > 0.7$, does not prevent the optimizer from finding the optimal designs. These results imply that innovative shapes with preferable aerodynamic performance are inside the recognition scope of the deep-learning-based geometric filtering model, which eases the concern on geometric filtering.

Using the deep-learning geometric filtering model can improve the performance of aerodynamic shape optimization and aerodynamic modeling. Two promising applications of the model are showcased. First, being used as a geometric validity constraint, the model makes gradient-based optimization manage to tackle challenging optimization problems with many geometric abnormalities. Second, the filtering model is used to derive global wing modes, which provides a compact parameterization of three-dimensional aircraft configurations. The deep-learning-based geometric filtering provides a fast and reliable approach to the judgment of aerodynamic shape quality, and further researches are recommended to investigate the pros and cons of relevant applications.

Appendix

For each training (UIUC) airfoil, a uniform x-y format with N = 251 points is used, and the x coordinates are set to

$$x_i = \frac{1}{2}\left(\cos\frac{2\pi(i-1)}{250} + 1\right) \quad i = 1,2,,,,,,251$$

Eq. 3.7.6

Then, the training airfoils are archived by recording corresponding 251 y coordinates in a vector format. The airfoil GAN model proposed by [4], as shown in **Figure 3.7.15**, is explained as follows. GAN is composed of a discriminative model (D) and a generative model (G). D in the airfoil GAN model uses a fully connected layer to perceive the input information. Then, a down sampling process is

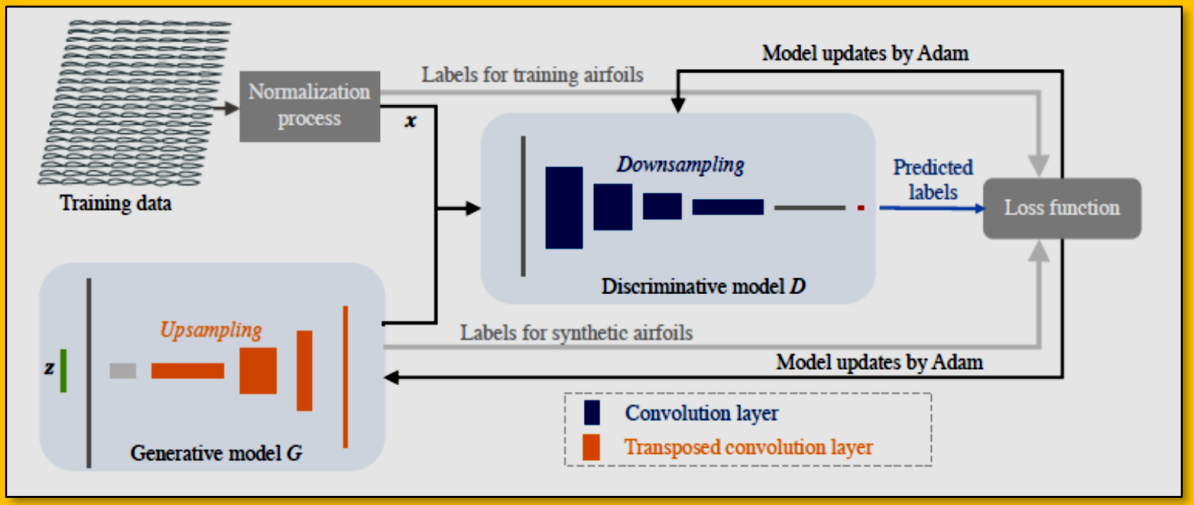


Figure 3.7.15 Flowchart of the airfoil GAN model proposed by Li et al. [4]

followed, where four convolution layers are used to extract the underlying features. These features are connected to a critic neuron, which distinguishes whether an input is from the training data or synthetic data of the generative model. G is trained simultaneously to generate synthetic data using noisy inputs. The noisy input is reshaped and then up sampled by four transposed convolution layers. The output of the last transposed convolution layer in G corresponds to the 251 y coordinates of an airfoil.

Acknowledgments

We acknowledge the Tier 2 grant from the Ministry of Education, Singapore (R-265-000-661-112). The computational resources of the National Supercomputing Centre, Singapore (https://www.nscc.sg) are acknowledged.

3.7.6 VI. Bibliography

3.7.6.1 References

[1] Jameson, A., "Aerodynamic Design via Control Theory," *Journal of Scientific Computing*, Vol. 3, No. 3, 1988, pp. 233–260. doi:10.1007/BF01061285.

[2] Reuther, J. J., Jameson, A., Alonso, J. J., Rimlinger, M. J., and Saunders, D., "Constrained Multipoint Aerodynamic Shape Optimization Using an Adjoint Formulation and Parallel Computers, Part 2," *Journal of Aircraft*, Vol. 36, No. 1, 1999, pp. 61–74. doi:10.2514/2.2414.

[3] Lyu, Z., Kenway, G. K.W., and Martins, J. R. R. A., "Aerodynamic Shape Optimization Investigations of the Common Research Model Wing Benchmark," *AIAA Journal*, Vol. 53, No. 4, 2015, pp. 968–985. doi:10.2514/1.J053318.

[4] Li, J., Zhang, M., Martins, J. R. R. A., and Shu, C., "Efficient Aerodynamic Shape Optimization with Deep-Learning Based Geometric Filtering," *AIAA Journal*, Vol. 58, No. 10, 2020, pp. 4243–4259. doi:10.2514/1.j059254, URL https: //doi.org/10.2514/1.j059254.

[5] Robinson, G. M., and Keane, A. J., "Concise Orthogonal Representation of Supercritical Airfoils," *Journal of Aircraft*, Vol. 38, No. 3, 2001, pp. 580–583. doi:10.2514/2.2803.

[6] Poole, D. J., Allen, C. B., and Rendall, T. C. S., "Metric-Based Mathematical Derivation of Efficient Airfoil Design Variables," *AIAA Journal*, Vol. 53, No. 5, 2015, pp. 1349–1361. doi:10.2514/1.j053427, URL https://doi.org/10.2514/1.j053427.

[7] Masters, D. A., Taylor, N. J., Rendall, T. C. S., Allen, C. B., and Poole, D. J., "Geometric Comparison of Aerofoil Shape Parameterization Methods," *AIAA Journal*, Vol. 55, No. 5, 2017, pp. 1575–1589. doi:10.2514/1.j054943.

[8] Li, J., Bouhlel, M. A., and Martins, J. R. R. A., "Data-based Approach for Fast Airfoil Analysis and Optimization," *AIAA Journal*, Vol. 57, No. 2, 2019, pp. 581–596. doi:10.2514/1.J057129.

[9] Kedward, L., Allen, C. B., and Rendall, T., "Towards Generic Modal Design Variables for Aerodynamic Shape Optimisation," *AIAA Scitech 2020 Forum*, American Institute of Aeronautics and Astronautics, 2020. doi:10.2514/6.2020-0543, URL https://doi.org/10.2514/6.2020-0543.

[10] Allen, C. B., Poole, D. J., and Rendall, T. C. S., "Wing aerodynamic optimization using efficient mathematically-extracted modal design variables," *Optimization and Engineering*, Vol. 19, No. 2, 2018, doi:10.1007/s11081-018-9376-7, URL https://doi.org/10.1007/s11081-018-9376-7.

[11] Viswanath, A., Forrester, A. I. J., and Keane, A. J., "Dimension Reduction for Aerodynamic Design Optimization," *AIAA Journal*,. doi:10.2514/1.j050717, URL https://doi.org/10.2514/1.j050717.

[12] Constantine, P. G., Dow, E., and Wang, Q., "Active Subspace Methods in Theory and Practice: Applications to Kriging Surfaces," *SIAM Journal on Scientific Computing*, Vol. 36, No. 4, 2014, pp. A1500–A1524. doi:10.1137/130916138, URL https://doi.org/10.1137/130916138.

[13] Li, J., Cai, J., and Qu, K., "Surrogate-based aerodynamic shape optimization with the active subspace method," *Structural and Multidisciplinary Optimization*, Vol. 59, No. 2, 2019, pp. 403–419. doi:10.1007/s00158-018-2073-5.

[14] Chen, W., Chiu, K., and Fuge, M. D., "Airfoil Design Parameterization and Optimization Using Bézier Generative Adversarial Networks," *AIAA Journal*, Vol. 58, No. 11, 2020, pp. 4723–4735. doi:10.2514/1.j059317, URL https://doi.org/10.2514/1.j059317.
[15] Du, X., He, P., and Martins, J. R. R. A., "A B-Spline-based Generative Adversarial Network Model for Fast Interactive Airfoil Aerodynamic Optimization," *AIAA SciTech Forum*, AIAA, Orlando, FL, 2020. doi:10.2514/6.2020-2128.
[16] Kedward, L. J., Allen, C. B., and Rendall, T. C. S., "Gradient-Limiting Shape Control for Efficient Aerodynamic Optimization," *AIAA Journal*, Vol. 58, No. 9, 2020, pp. 3748–3764. doi:10.2514/1.j058977, URL https://doi.org/10.2514/1.j058977.
[17] Bons, N., Martins, J., Mader, C. A., McMullen, M. S., and Suen, M., "High-fidelity Aerostructural Optimization Studies of the Aerion AS2 Supersonic Business Jet," *AIAA AVIATION 2020 FORUM*, American Institute of Aeronautics and Astronautics, 2020. doi:10.2514/6.2020-3182, URL https://doi.org/10.2514/6.2020-3182.
[18] Li, J., He, S., and Martins, J. R. R. A., "Data-driven Constraint Approach to Ensure Low-speed Performance in Transonic Aerodynamic Shape Optimization," *Aerospace Science and Technology*, Vol. 92, 2019, pp. 536–550. doi:10.1016/j.ast.2019.06.008.
[19] Wu, H., Zheng, S., Zhang, J., and Huang, K., "GP-GAN: Towards realistic high-resolution image blending," *arXiv preprint arXiv:1703.07195*, 2017.
[20] Arjovsky, M., Chintala, S., and Bottou, L., "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.
[21] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C., "Improved training of wasserstein gans," *Advances in neural information processing systems*, Vol. 30, 2017, pp. 5767–5777.
[22] Kenway, G. K., Kennedy, G. J., and Martins, J. R. R. A., "A CAD-Free Approach to High-Fidelity Aerostructural Optimization," *Proceedings of the 13th AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Fort Worth, TX, 2010. doi: 10.2514/6.2010-9231.
[23] Lyu, Z., and Martins, J. R. R. A., "Aerodynamic Design Optimization Studies of a Blended-Wing-Body Aircraft," *Journal of Aircraft*, Vol. 51, No. 5, 2014, pp. 1604–1617. doi:10.2514/1.C032491.
[24] Kenway, G. K. W., and Martins, J. R. R. A., "Buffet Onset Constraint Formulation for Aerodynamic Shape Optimization," *AIAA Journal*, Vol. 55, No. 6, 2017, pp. 1930–1947. doi:10.2514/1.J055172.
[25] Secco, N. R., Jasa, J. P., Kenway, G. K.W., and Martins, J. R. R. A., "Component-based Geometry Manipulation for Aerodynamic Shape Optimization with Overset Meshes," *AIAA Journal*, Vol. 56, No. 9, 2018, pp. 3667–3679. doi:10.2514/1.J056550.
[26] Shi, Y., Mader, C. A., He, S., Halila, G. L. O., and Martins, J. R. R. A., "Natural Laminar-Flow Airfoil Optimization Design Using a Discrete Adjoint Approach," *AIAA Journal*, Vol. 58, No. 11, 2020, pp. 4702–4722. doi:10.2514/1.j058944, URL https://doi.org/10.2514/1.j058944.
[27] He, X., Li, J., Mader, C. A., Yildirim, A., and Martins, J. R. R. A., "Robust aerodynamic shape optimization—from a circle to an airfoil," *Aerospace Science and Technology*, Vol. 87, 2019, pp. 48–61. doi:10.1016/j.ast.2019.01.051.
[28] Bons, N. P., He, X., Mader, C. A., and Martins, J. R. R. A., "Multimodality in Aerodynamic Wing Design Optimization," *AIAA Journal*, Vol. 57, No. 3, 2019, pp. 1004–1018. doi:10.2514/1.J057294.
[29] Martins, J. R. R. A., "Perspectives on Aerodynamic Design Optimization," *AIAA SciTech Forum*, AIAA, Orlando, FL, 2020. doi:10.2514/6.2020-0043.
[30] Perez, R. E., Jansen, P. W., and Martins, J. R. R. A., "pyOpt: A Python-Based Object-Oriented Framework for Nonlinear Constrained Optimization," *Structural and Multidisciplinary Optimization*, Vol. 45, No. 1, 2012, pp. 101–118. doi:10.1007/s00158-011-0666-3.
[31] Wu, N., Kenway, G., Mader, C., Jasa, J., and Martins, J., "pyOptSparse: APython framework for large-scale constrained nonlinear optimization of sparse systems," *Journal of Open Source Software*, Vol. 5, No. 54, 2020, p. 2564. doi:10.21105/joss.02564, URL https://doi.org/10.21105/joss.02564.

[32] Yildirim, A., Kenway, G. K. W., Mader, C. A., and Martins, J. R. R. A., "A Jacobian-free approximate Newton–Krylov startup strategy for RANS simulations," *Journal of Computational Physics*, Vol. 397, 2019, p. 108741. doi:10.1016/j.jcp.2019.06.018.
[33] Mader, C. A., Kenway, G. K. W., Yildirim, A., and Martins, J. R. R. A., "ADflow: An Open-Source Computational Fluid Dynamics Solver for Aerodynamic and Multidisciplinary Optimization," *Journal of Aerospace Information Systems*, Vol. 17, No. 9, 2020, pp. 508–527. doi:10.2514/1.i010796, URL https://doi.org/10.2514/1.i010796.
[34] Spalart, P., and Allmaras, S., "A One-Equation Turbulence Model for Aerodynamic Flows," *30th Aerospace Sciences Meeting and Exhibit*, 1992. doi:10.2514/6.1992-439.
[35] Kenway, G. K. W., Mader, C. A., He, P., and Martins, J. R. R. A., "Effective Adjoint Approaches for Computational Fluid Dynamics," *Progress in Aerospace Sciences*, Vol. 110, 2019, p. 100542. doi:10.1016/j.paerosci.2019.05.002.
[36] Chernukhin, O., and Zingg, D. W., "Multimodality and Global Optimization in Aerodynamic Design," *AIAA Journal*, Vol. 51, No. 6, 2013, pp. 1342–1354. doi:10.2514/1.j051835.
[37] Li, J., "Optimized airfoils based on different flight missions," Mendeley Data, 2020. doi:10.17632/23yrzbzr3m.1.
[38] Chen, S., Lyu, Z., Kenway, G. K. W., and Martins, J. R. R. A., "Aerodynamic Shape Optimization of the Common Research Model Wing-Body-Tail Configuration," *Journal of Aircraft*, Vol. 53, No. 1, 2016, pp. 276–293. doi:10.2514/1.C033328.
[39] Luke, E., Collins, E., and Blades, E., "A Fast Mesh Deformation Method Using Explicit Interpolation," *Journal of Computational Physics*, Vol. 231, No. 2, 2012, pp. 586–601. doi:10.1016/j.jcp.2011.09.021.
[40] Li, J., and Zhang, M., "Adjoint-Free Aerodynamic Shape Optimization of the Common Research Model Wing," *AIAA Journal*, 2021, pp. 1–11. doi:10.2514/1.j059921, URL https://doi.org/10.2514/1.j059921.
[41] Li, J., and Zhang, M., "Data-based Approach for Wing Shape Design Optimization," *Aerospace Science and Technology*, 2021. (Submitted).