Artificial Intelligence (AI) & Machine Learning (ML/DL/NNs)

Technical Report · April 2024

DOI: 10.13140/RG.2.2.20926.05444

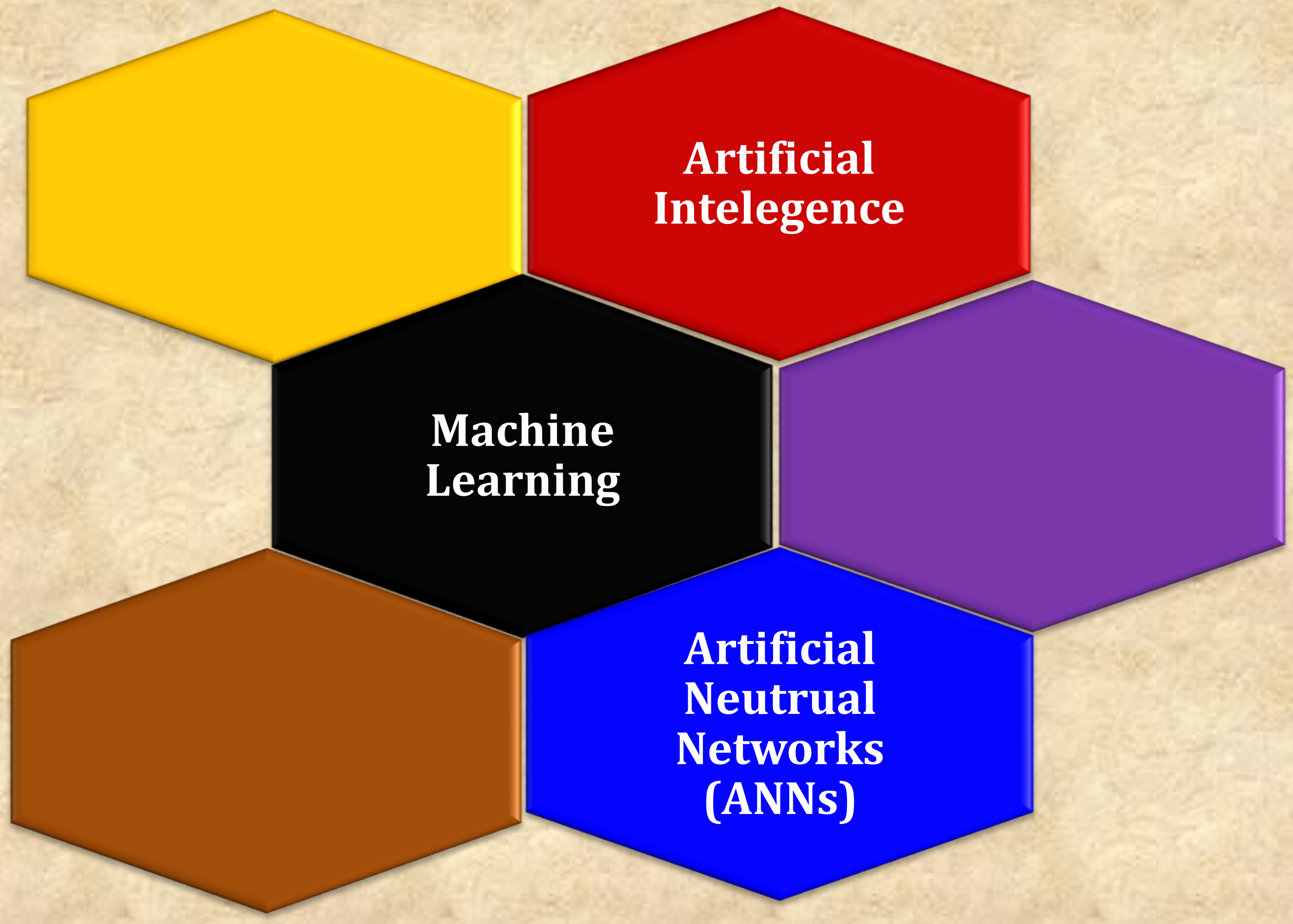
Ideen Sadrehaghighi
CFD Open Series

CFD Open Series / Patch 2.75

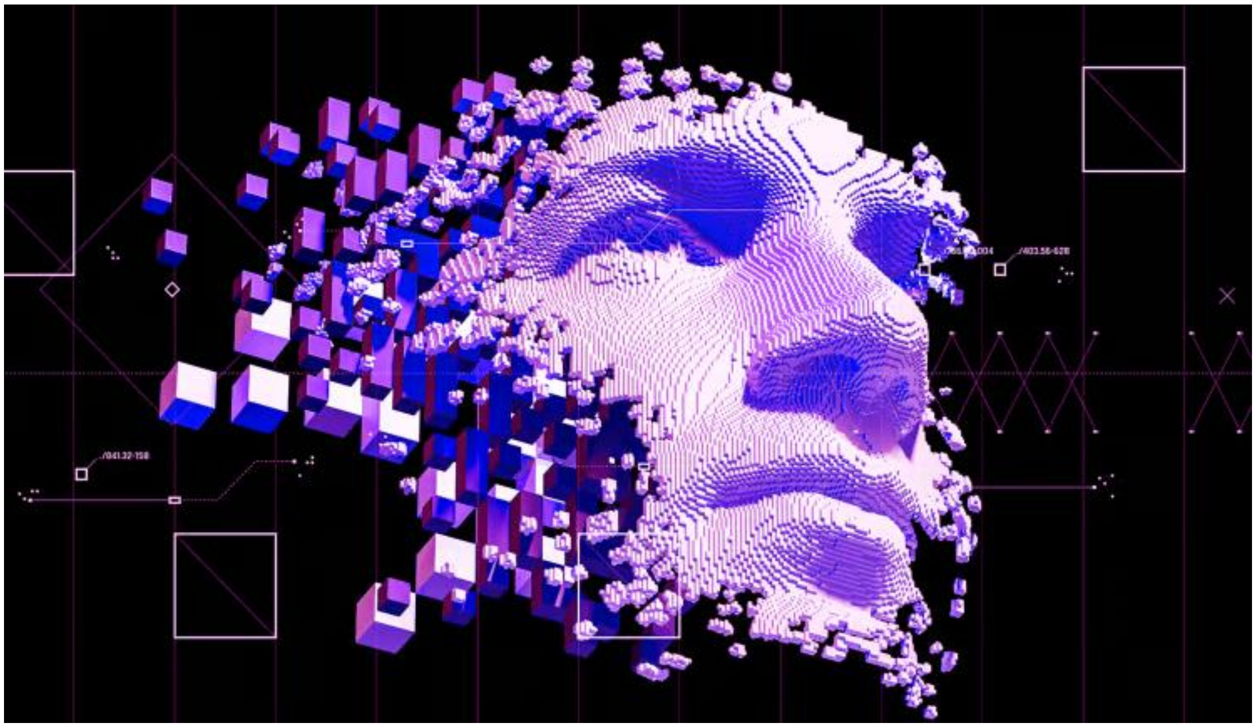
Artificial Intelligence (AI) & Machine Learning (ML/DL/NNs)

ARTIFICIAL INTELLIGENCE

Edited & Adapted by: Ideen Sadrehaghighi



ANNAPOLIS, MD



Contents

List of Tables

List of Figures

1 Artificial Intelligence

1.1 Definitions

Artificial Intelligence (AI) can be outlined as the analysis of mental and psychological abilities by using various computational patterns and sequences [1,2]. The term "intelligence" in this field can be very deceptive. For instance, we usually apply this word when we want to describe someone displaying unusual inventiveness and mind-blowing skills. This results in giving the impression that artificial intelligence is a reliable method for generating loads of clever ideas and insights but in reality, it revolves around the basic idea of duplicating the physiological and mental abilities of the "ordinary" people. It can also be defined as the science of creating sophisticated machines and devices as well as various computerized programs to analyze human intelligence [3] for solving the practical problems that the world presents us with. The ultimate aim of artificial intelligence is to create devices that have human-level intelligence, as some might think that this practice is immoral and indecent [2]. In broadest way, **Artificial Intelligence (AI)** can be think of 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.**" A.I. 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 A.I. It also used to classify machines that mimic human intelligence and human cognitive functions, like problem-solving and learning. AI uses predictions and automation to optimize and solve complex tasks that humans have historically done, such as facial and speech recognition, decision making and translation (IBM Blog, 2023).

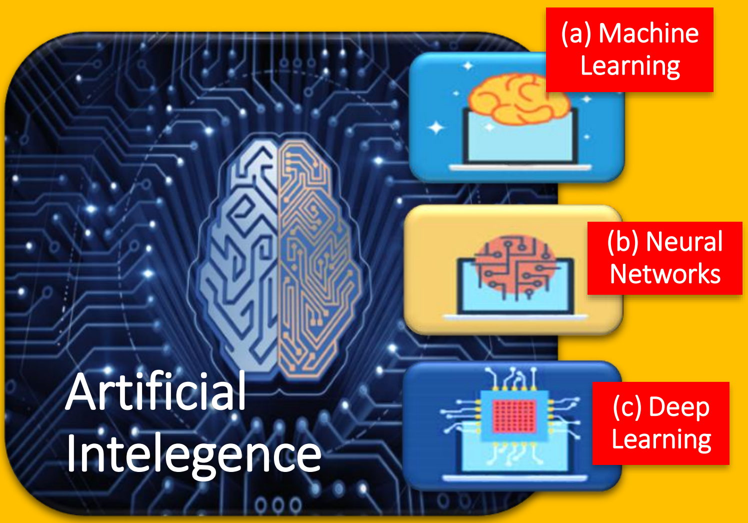


Figure 1.1.1 Scope of Artificial Intelligence - Courtesy of Hackerearth Blog

1.2 Categories of AI

Three main categories of AI are:

- Artificial Narrow Intelligence (ANI)
- Artificial General Intelligence (AGI)
- Artificial Super Intelligence (ASI)

ANI is considered "weak" AI, whereas the other two types are classified as "strong" AI. We define weak AI by its ability to complete a specific task, like winning a chess game or identifying a particular individual in a series of photos. Natural language processing (NLP) and computer vision, which let companies automate tasks and underpin chatbots and virtual assistants such as *Siri* and *Alexa*, are

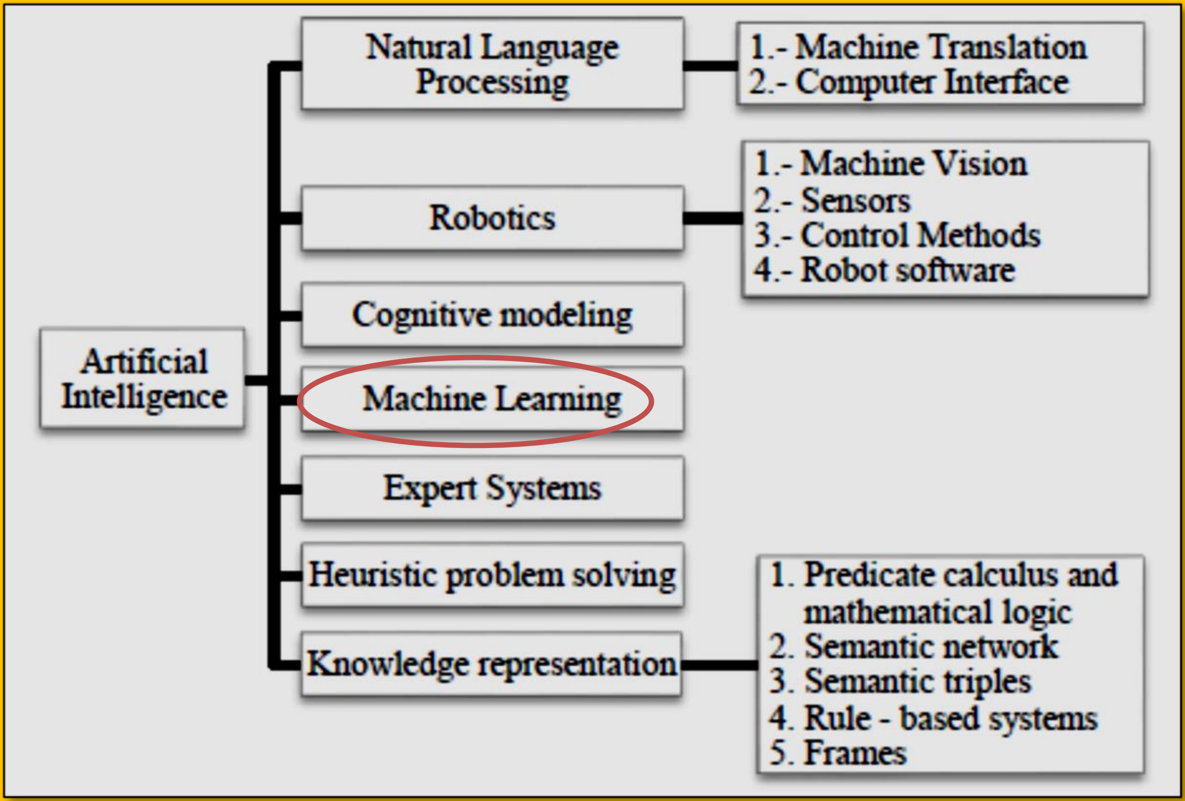


Figure 1.2.1 Research in artificial intelligence [1]

examples of ANI. Computer vision is a factor in the development of self-driving cars. Stronger forms of AI, like AGI and ASI, incorporate human behaviors more prominently, such as the ability to interpret tone and emotion. Strong AI is defined by its ability compared to humans. Artificial General Intelligence (AGI) would perform on par with another human, while Artificial Super Intelligence (ASI), also known as superintelligence, would surpass a human's intelligence and ability. Neither form of Strong AI exists yet, but research in this field is ongoing (IBM newsletter). According to *HackerEarth Blog*, AI can be classified into the following (see **Figure 1.1.1**):

- Machine Learning (ML)
- Deep Learning (DL)
- Neural Networks (NNs)

Other definitions provided by Kontos [4], 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. **Figure 1.2.1** indicates the various area of Artificial Intelligence with attentive subject contoured shown in red ellipse.
While for artificial intelligence (AI), machine learning (ML), deep learning (DL) and neural networks (NN) are related technologies, the terms are often used interchangeably, which frequently leads to confusion about their differences. Vargas et al. [5] describes the Deep Learning (DL) as an emerging

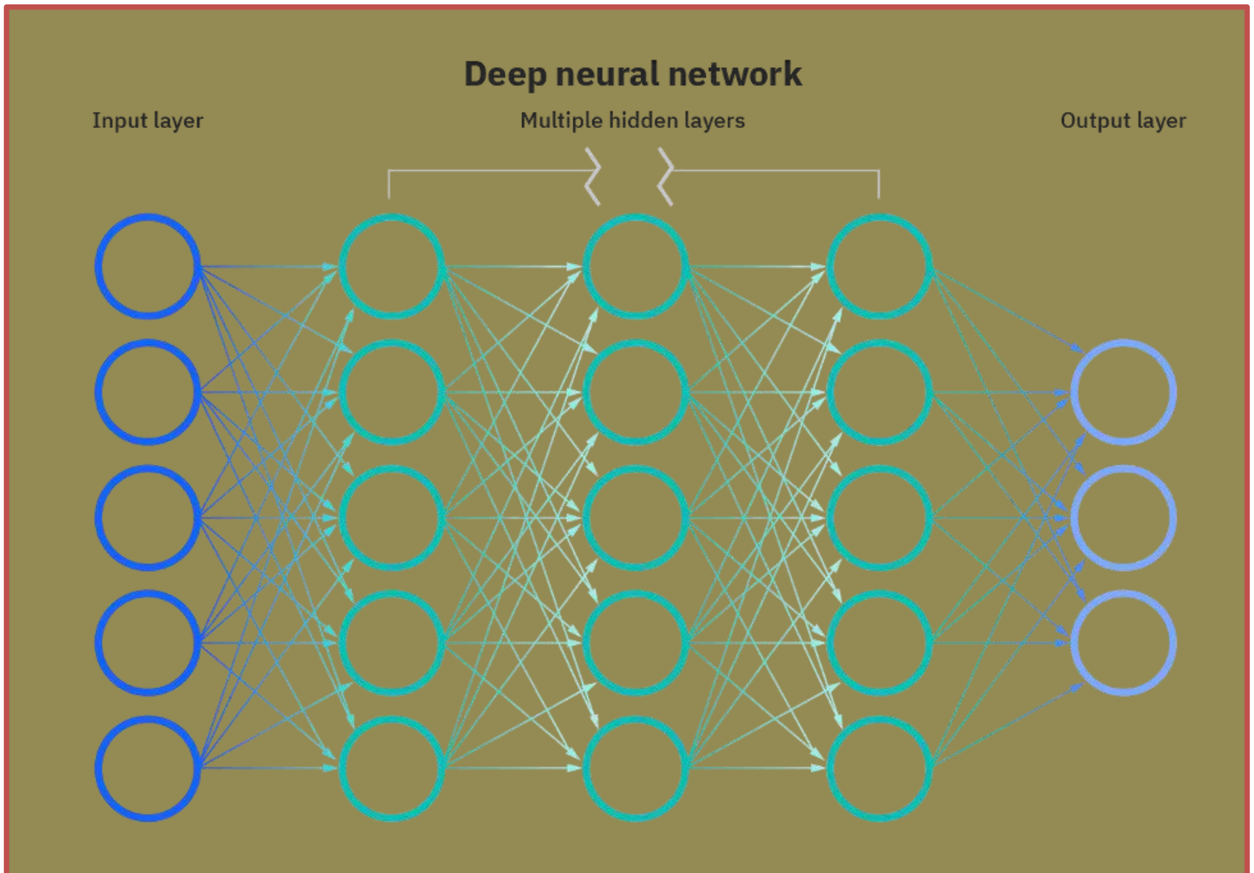


Figure 1.2.2 Schematics of Deep Learning

area of Machine Learning (ML) research (**Figure 1.2.2**). It comprises multiple hidden layers of Artificial Neural Networks (ANNs). The deep learning methodology applies nonlinear transformations and model abstractions of high level in large databases.

1.2.1 References

[1] Charniak, E. (1985). *Introduction to artificial intelligence*. Pearson Education India
[2] Ananya Priyadarshini, "*Artificial Intelligence: The Inescapable*", B.Tech. Computer Science, 2022.
[3] McCarthy, J. (2007). What is artificial intelligence?
[4] John Kontos, "*Artificial Intelligence, Machine Consciousness and Explanation*", Academia Letters preprint, 2012.
[5] Vargas, R., Mosavi, A., & Ruiz, R. (2017). *Deep learning: a review*.

2 Machine Learning (ML)

2.1 Definition

Before we pay tribute to the field of machine learning, it best to go briefly of what is machine learning itself is. **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, 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**,
- **Un-supervised,**
- **Semi-supervised Learning,** 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 [1]. Examples include learning how to play games [2], such as chess. (Brunton, 2021)[3].

2.2 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) [4] 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.

2.2.1 Reference

[1] Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*, vol. 1. MIT Press, Cambridge (1998)
[2] Mnih, V., Kavukcuoglu, K., Silver, D., et al.: *Human-level control through deep reinforcement learning*. Nature 518, 529 (2015)
[3] arXiv:2110.02083 [physics. flu-dyn]
[4] Pandey, S., Schumacher, J., & Sreenivasan, K. R. (2020). *A perspective on machine learning in turbulent flows*. Journal of Turbulence.

2.3 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 2.3.1**). 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 distinguish 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,

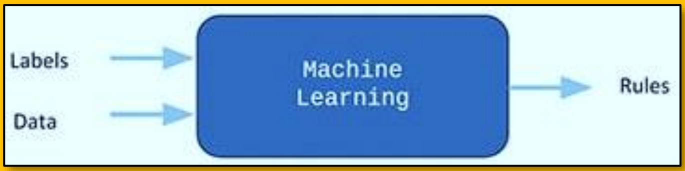
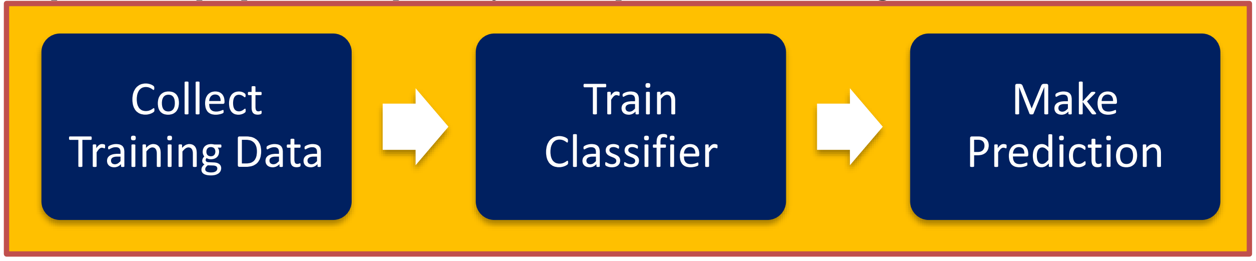


Figure 2.3.1 Machine Learning Programming



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.**

2.3.1 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.

2.3.2 Collect Training Data

To collect training data, assume we head out to an orchard and collect some data. We look at different 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 2.3.1**).

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

Table 2.3.1 Data Considered

2.3.3 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.

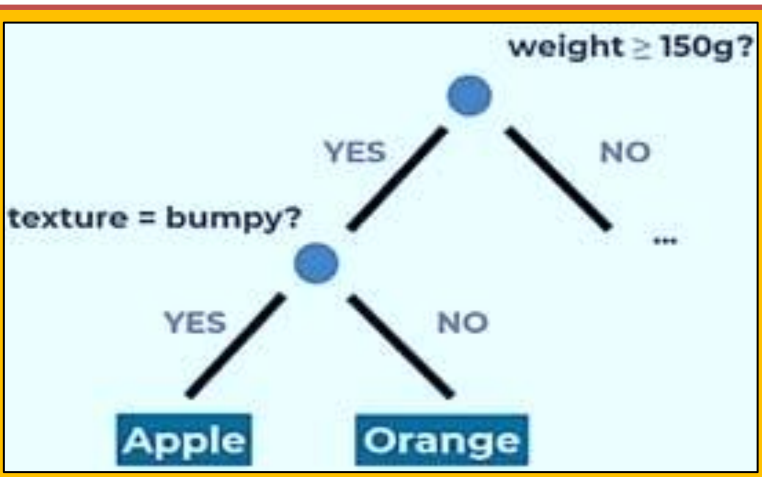


Figure 2.3.2 Decision Tree Classifier

```
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 2.3.2).

2.3.4 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.

2.3.5 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. 2.3.1

We would like to learn the **Eq. 2.3.1**

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

Eq. 2.3.2

without relying on the knowledge of the underlying processes (Gyrya, Shashkov, Skurikhin, & Tokareva, 2019)[4]. For example, the relationship **Eq. 2.3.2** may represent a physical process for which some observations are available but the analytical relation **Eq. 2.3.1** 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. 2.3.1** 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. 2.3.2**. Assume we are provided a number of observations (training set):

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

Eq. 2.3.3

where (r^i_L; r^i_R) satisfy **Eq. 2.3.1** exactly. From the training data **Eq. 2.3.3** we will try to predict the relation **Eq. 2.3.2** on a new previously unseen data (a^j; b^j; c^j):

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

Eq. 2.3.4

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)

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

Eq. 2.3.5

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.

2.3.6 References

[1] Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*, vol. 1. MIT Press, Cambridge (1998)
[2] Mnih, V., Kavukcuoglu, K., Silver, D., et al.: *Human-level control through deep reinforcement learning*. Nature 518, 529 (2015)
[3] arXiv:2110.02083 [physics. flu-dyn]
[4] 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.
[5] Pandey, S., Schumacher, J., & Sreenivasan, K. R. (2020). *A perspective on machine learning in turbulent flows*. Journal of Turbulence.

2.4 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)[1]. The **ANNs** are inspired by our understanding of the biology of our brains all those interconnections between the neurons (Copeland, 2010)[2]. 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 2.4.1**. 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 (DNN)** approach to simulate one-dimensional non-relativistic fluid dynamics, as presented by (Taradiy et al.)[3]. A prime example of deep learning would be **ChatGPT** from *Open AI*, which is a conversational AI-powered chatbot designed to answer questions and respond to queries in text form in a way that sounds natural and human.

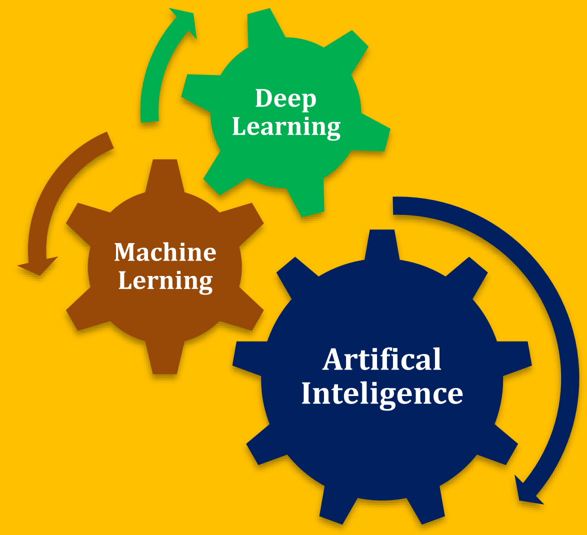


Figure 2.4.1 Schematics of AI, Machine Learning and Deep Learning

2.4.1 References

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

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

[3] 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.

2.5 Algorithms and Types of Problems

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. As identified before, these are:

2.5.1 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. [1]
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 2.5.1**). A commonly employed loss function is :

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

Eq. 2.5.1

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 [2].

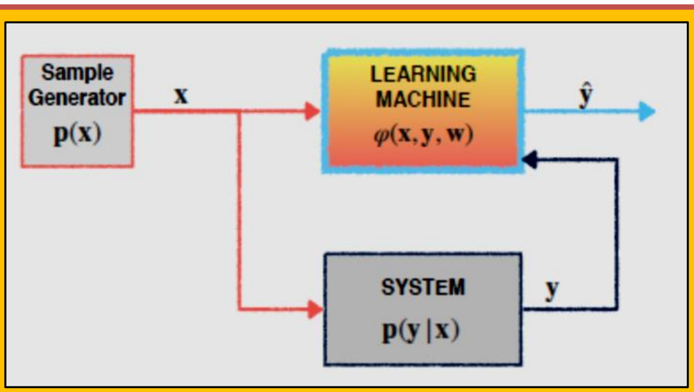


Figure 2.5.1 A Learning Machine Uses Inputs From a Sample Generator and Observations from a System to Generate an Approximation of its Output (Credit: Cherkassky & Mulier (2007))

2.5.2 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.

2.5.3 Semi-supervised Learning

Semi-supervised learning is a hybrid machine learning approach that combines labeled and unlabeled data for training. It leverages the limited labeled data and a larger set of unlabeled data to improve the learning process. The idea is that the unlabeled data provide additional information and context to enhance the model's understanding and performance. By utilizing the unlabeled data effectively, semi-supervised learning can overcome the limitations of relying solely on labeled data. This approach is particularly useful when acquiring labeled data is expensive or time-consuming. Semi-supervised learning techniques can be applied to various tasks, such as classification, regression, and anomaly detection, allowing models to make more accurate predictions and generalize better in real-world scenarios (Simon Tavasoli)[1].

2.5.4 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 learns from past experience and tries to capture the best possible knowledge to make accurate business decisions. Example of Reinforcement Learning: Markov Decision Process [3].

2.5.5 References

[1] Sunil, Ray, "*Essentials of Machine learning Algorithms (with Python and R codes)*", August 2015.
[2] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos, "*Machine Learning for Fluid Mechanics*", Annual Review of Fluid Mechanics , January 2020.

[1] Top 10 Machine Learning Algorithms For Beginners: Supervised, and More

[3] Same as Previous

2.6 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. (see **Table 2.6.1**).

Algorithms	*Machine Learning*
Linear Regression	supervised
Logistic Regression	supervised
Decision Tree	supervised
Artificial Neural Networks (ANNs)	supervised
Support Vector Machine (SVM)	supervised
K-Nearest Neighbors (KNN)	supervised
K-Means	Un-supervised
Random Forest	supervised
Dimensionality Reduction Algorithms (POD/PCA)	Un-supervised
Genetic algorithms	supervised
Q-learning	Semi-supervised
Markov Decision Process	reinforcement

Table 2.6.1 Machine learning algorithms may be categorized into supervised, unsupervised, and semi-supervised, depending on the extent and type of information available

2.6.1 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

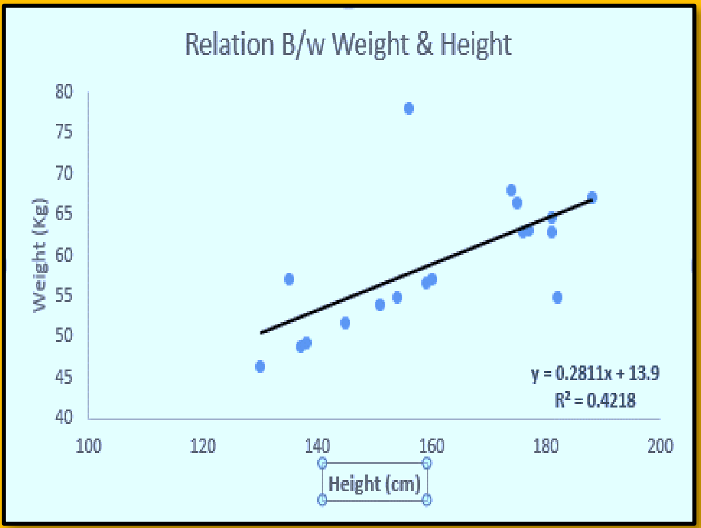


Figure 2.6.1 Linear Regression

identified the best fit line having linear equation y = 0.2811 x +13.9 (see **Figure 2.6.1**). 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 variables. While finding best fit line, you can fit a polynomial or curvilinear regression. And these are known as polynomial or curvilinear regression [1].

2.6.2 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.

2.6.3 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 [2]. 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 2.6.2**).

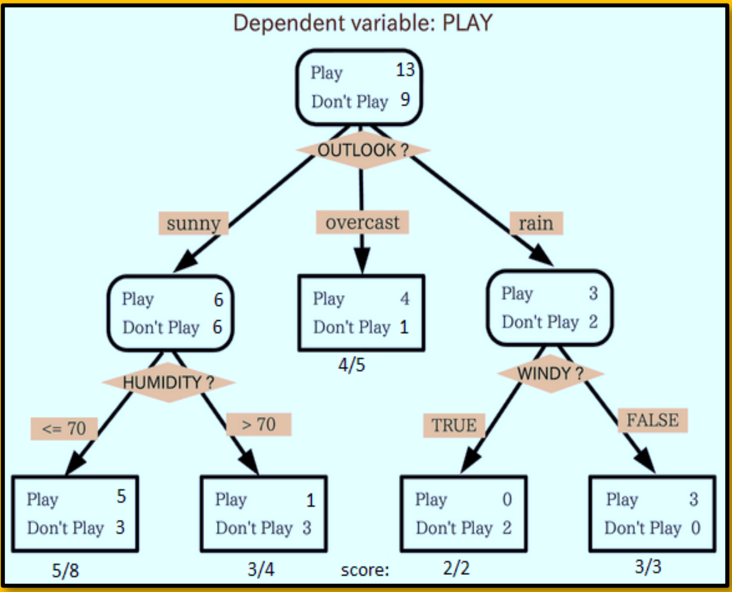
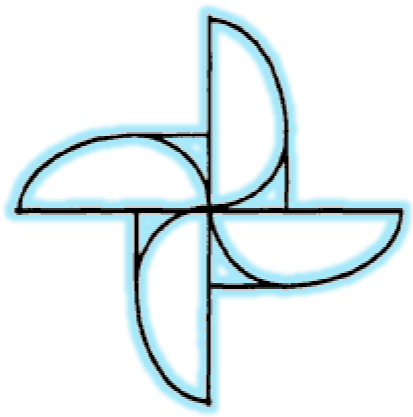


Figure 2.6.2 Decision Tree

2.6.4 References

[1] Sunil, Ray, “Essentials of Machine learning Algorithms (with Python and R codes)”, August 2015.
[2] Same as Above



3 Artificial Neutral Networks (ANNs)

3.1 Preliminaries

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 3.1.1**).

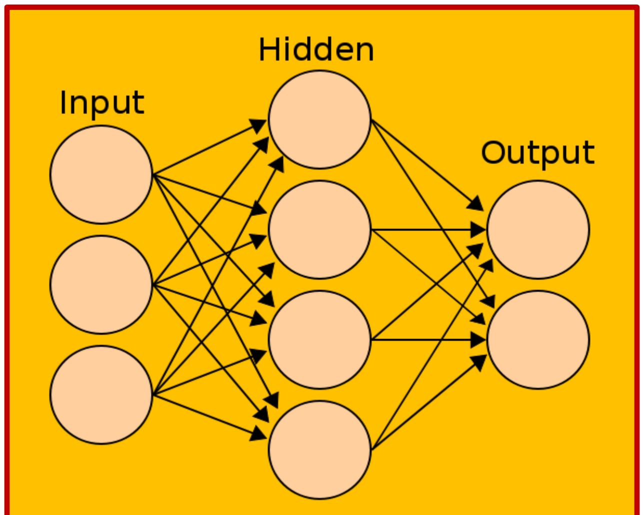


Figure 3.1.1 Artificial Neural Network (ANN)

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.

3.2 Types of Neutral Networks

3.2.1 Perceptron

The **Perceptron** is the most basic and oldest form of neural networks (Pavan Vadapalli)[2]. It consists of just 1 neuron which takes the input and applies activation function on it to produce a binary output. 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 3.2.1**).

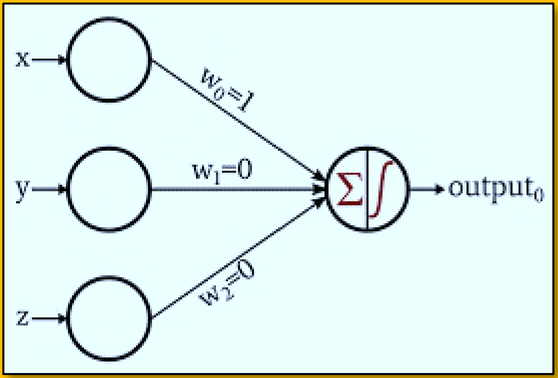


Figure 3.2.1 Perceptron

3.2.2 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 3.1.1**). 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.

3.2.3 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 (**Figure 3.2.2**). 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.

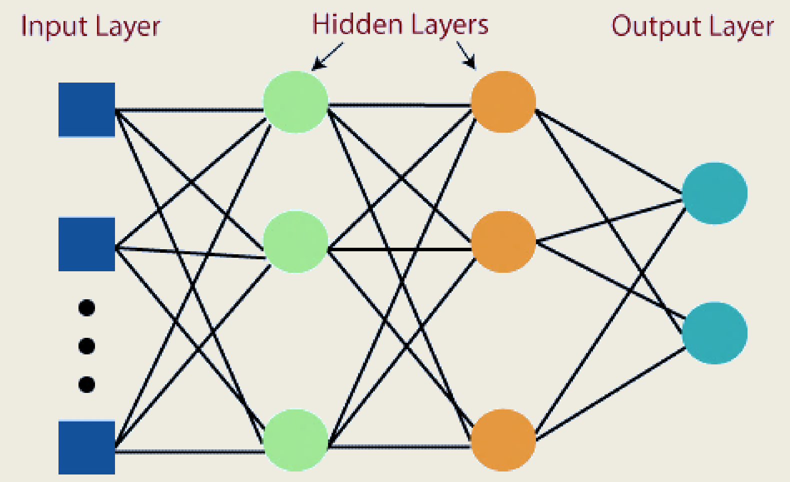


Figure 3.2.2 Multi-Layer Perceptron Architecture

3.2.4 Radial Basis Networks

Radial Basis Networks (RBN) use a completely different way to predict the targets. It consists of

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

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 3.2.3**).

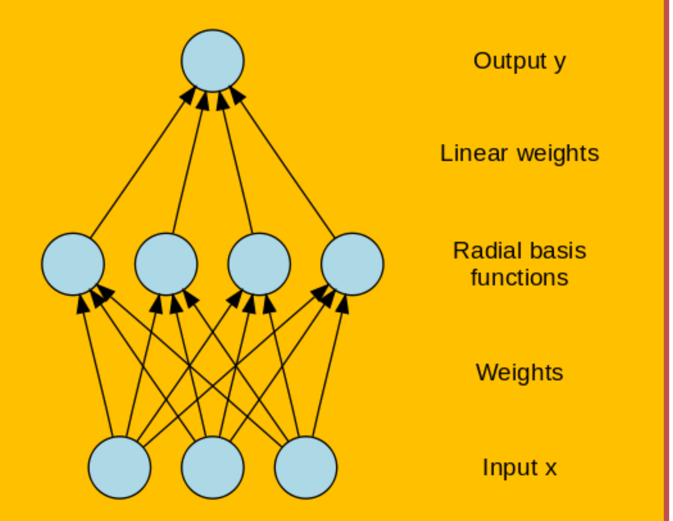


Figure 3.2.3 Radial Basis Function

3.2.5 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 3.2.4**). 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.

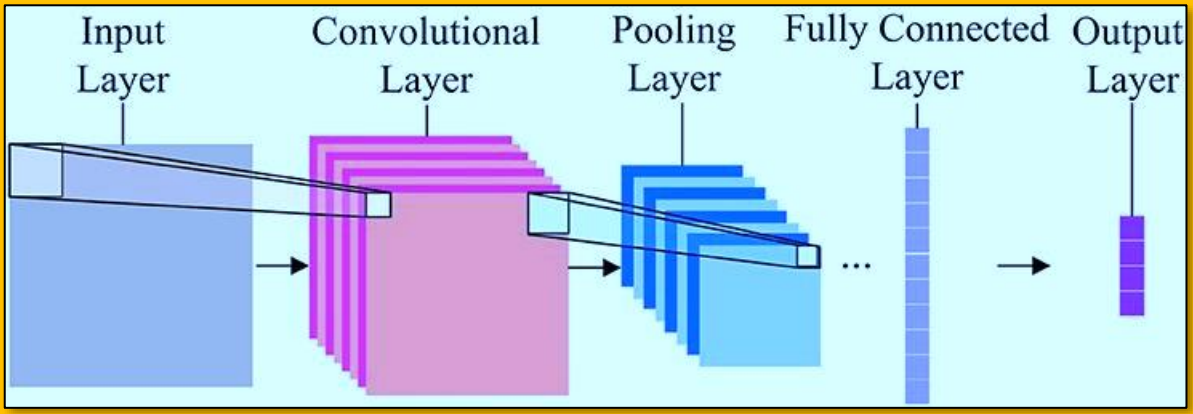


Figure 3.2.4 Convolutional Neural Networks

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.

3.2.6 Recurrent Neural Networks

Recurrent Neural Networks (RNN) come into picture when there's a need for predictions using sequential data (**Figure 3.2.5**). 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.

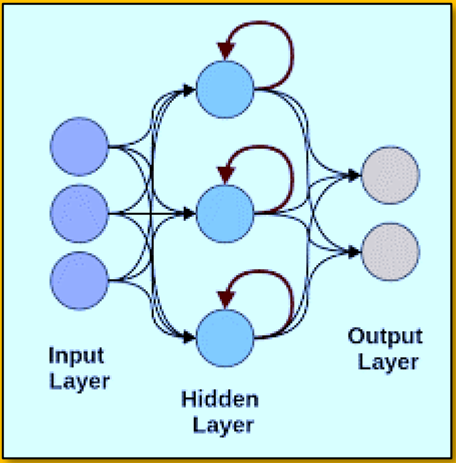


Figure 3.2.5 Recurrent Neural Networks (RNN)

3.2.7 Physics-Informed Neural Networks (PINN)

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)[3] (**Figure 3.2.6**). 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 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

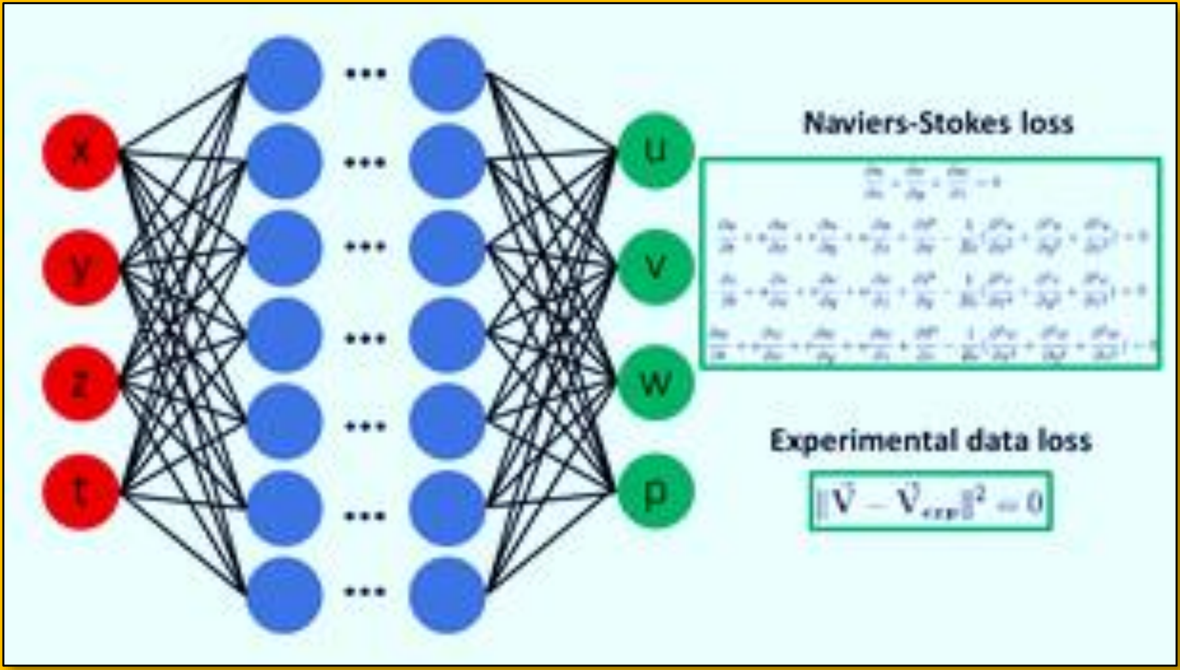


Figure 3.2.6 Physics-informed neural networks for solving Navier–Stokes equations

[3] Wikipedia

3.3 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.

3.4 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 Error! Reference source not found.. The Prediction cases (Machine Learning with no data) has similar configuration with different; (1) Twist, (2) Sweep angles, and (3) Airfoil shape[4]. 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.

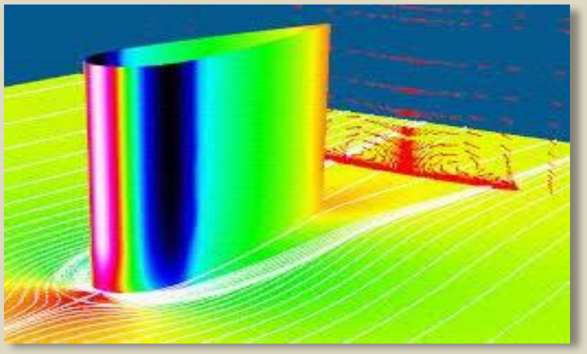


Figure 3.4.1 Calibration Cases for off Line Data

3.5 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

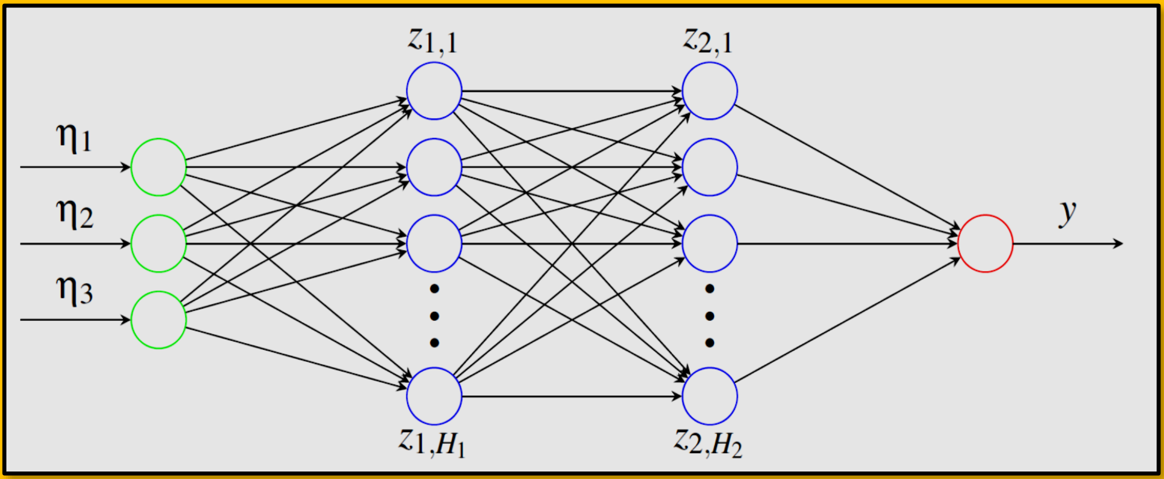


Figure 3.5.1 Network Diagram for a feed-forward NN with three inputs and one output

[4] 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.

them through nonlinear activation functions[5]. The process is repeated once for each hidden layer (marked blue in Error! Reference source not found.) in the network, until the output layer is reached. Error! Reference source not found. 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. 3.5.1

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. 3.5.2

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. 3.5.3

Given training data, error back-propagation algorithms[6] 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.

3.5.1 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[7]. 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)**[8]. The linear POD is an approximation of the flow vector *v* by a finite expansion of orthonormal functions φ_n such that:

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

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

[6] 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.

[7] 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.

[8] 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.

Eq. 3.5.4

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][9] 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. 3.5.5

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. 3.5.6

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.

3.5.2 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][10]. 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

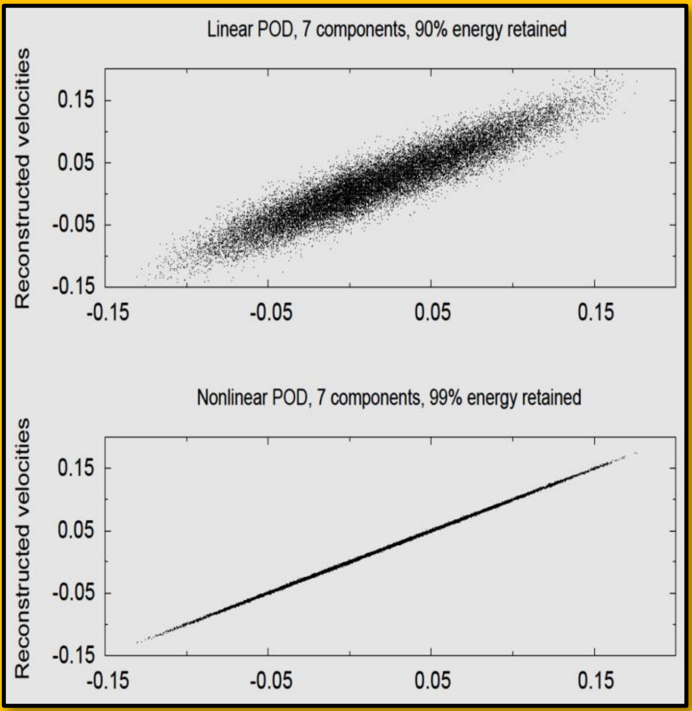


Figure 3.5.2 Comparison of linear POD (top) and Neural Networks (bottom)

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

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

network is composed by 256 inputs and 4 layers having respectively 64 nonlinear neurons, 7 linear 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 Error! Reference source not found. 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.

3.6 Case Study 1 - 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):**

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

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

Eq. 3.6.1

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 3.6.1 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

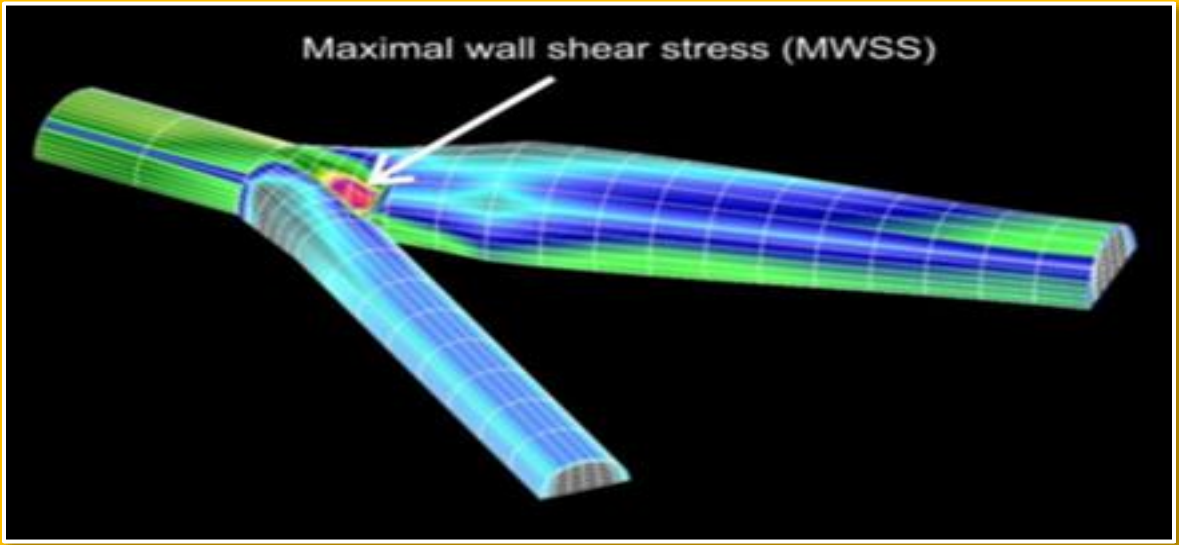


Figure 3.6.1 Maximal Wall Shear Stress (MWSS) Value for Carotid Artery Bifurcation

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 3.6.1** and **Figure 3.6.1**).

3.7 Case Study 2 - Deep Learning-Based Prediction of Aerodynamic Performance for Airfoils in Transonic Regime

Authors: Tarek Ayman[1], Mayar A. Elrefaie[1], Eman Sayed[1], Mohamed Elrefaie[2], Mahmoud Ayyad[3]; Ahmed A. Hamada[4], Ahmed A. Hamada[5]

Affiliations : [1]Aerospace Engineering Department Cairo University, Giza, 12613, Egypt
[2]Department of Aerospace and Geodesy , Technical University of Munich, Germany
[3]Davidson Lab, CEOE Department Stevens Institute of Technology, Hoboken, NJ, USA
[4]Department of Ocean Engineering, Texas A&M University, College state, TX, USA
[5]Aerospace Engineering Department Cairo University, Giza, 12613, Egypt

Citation : Ayman, T., Elrefaie, M. A., Sayed, E., Elrefaie, M., Ayyad, M., Hamada, A. A., & Abdelrahman, M. M. (2023, October). Deep Learning-Based Prediction of Aerodynamic Performance for Airfoils in Transonic Regime. In *2023 5th Novel Intelligent and Leading Emerging Sciences Conference (NILES)* (pp. 157-160). IEEE.

3.7.1 Abstract

This paper presents an approach to estimate the aerodynamic coefficients of airfoils in the transonic regime using **Artificial Neural Networks**. The transonic regime is a critical and challenging aerodynamic domain, and our approach utilizes data generated by the **OpenFOAM®** to train our model. Our dataset encompasses a wide range of transonic flow conditions and different airfoil shapes, enabling our Artificial Neural Networks to capture the complex behavior of aerodynamic phenomena in this regime. Our proposed framework achieves high accuracy, with the lift and moment coefficient predictions demonstrating an unprecedented accuracy level of 99.7% with respect to the test dataset obtained by OpenFOAM. Our results demonstrate the potential of Artificial Neural Networks to accurately predict aerodynamic coefficients in the transonic regime, which could have significant implications for the design and optimization of high-performance aircraft.

Index Terms - Artificial Neural Network, Computational Fluid, Dynamics, Transonic airfoils, Aerodynamic performance, Compressible flow, OpenFOAM

3.7.2 Introduction

The optimization of airfoil shape is a crucial aspect of designing efficient aircraft, wind turbines, and unmanned aerial vehicles. However, achieving the optimal design requires extensive numerical simulations that account for different combinations of operational conditions, such as speed, Angle Of Attack (AOA), and altitude. These numerical simulations are a computationally intense task, which poses a significant challenge during the preliminary stages of the optimization process. To alleviate the computational burden, reduced order models using machine learning can be used for the initial designs. Different approaches have been considered to address solutions to reduce the computational burden. For example, proper orthogonal decomposition has been applied in previous studies [1–3], along with machine learning techniques, to construct surrogate models that can predict airfoil performance. Yilmaz and German [4] proposed a classifier based on a Convolutional Neural Network (CNN), that used airfoil images as input, to predict airfoil performance. The classifier predicted discrete pressure distribution on the wing surface with an accuracy of more than 80%. Moin et al. [5] proposed a data-driven model to predict aerodynamic coefficients using sparse normalized 2D airfoil coordinates and artificial neural networks (ANN).

The results show that ANNs can capture aerodynamic characteristics with limited geometric information, making them a promising approach for predicting aerodynamic coefficients with

adequate accuracy and rapid pace. Zhang et al. [6] investigated three types of architectures to predict the lift coefficient of airfoils with a variety of shapes under multiple flow conditions. Hui et al. [7] employed a data-driven approach with Convolutional Neural Network (CNN) and Feedforward Neural Network (FNN) to predict pressure distribution over airfoils achieving less than 2% Mean Square Error (MSE) using Signed Distance Function (SDF) parameterization method.
Thuerey et al. [8] focused on a modernized U-net architecture and evaluated a large number of trained neural networks with respect to their accuracy for calculating the pressure and velocity distribution. Ahmed et al. [9] conducted a study wherein they developed an artificial neural network architecture to predict the drag and lift coefficients generated by airfoils under various aerodynamic conditions. The proposed Back-Propagation Neural Network (BPNN) was trained as a regression analysis tool specifically designed to determine the coefficients of the airfoils.
Here, we investigate the applicability of ANNs to predict the performance of 2D airfoils under transonic operating conditions. The efficacy of this approach is contingent upon the accuracy of both the training dataset and the selected deep-learning model. Through alleviating the dependence on costly and time-consuming Computational Fluid Dynamics (CFD) simulations, our proposed solution holds noteworthy implications for enhancing the design and optimization of transonic airfoils. Transonic airfoils are exposed to complex phenomena such as shock waves and shock wave/boundary layer interaction (SWBLI), making it challenging to collect a dataset with the full degree of the problem. Due to limited time and resources, we used Euler CFD simulations using OpenFOAM software to collect the data set required to train, validate, and test the ANN model. The ANNs model is capable of predicting the aerodynamic performance; the lift (C_l) and moment (C_m) coefficients of a selected transonic airfoil at flight conditions similar to its training data using pure input-output mapping without prior knowledge of the physical phenomena. The accuracy and efficiency of the ANNs model are compared with the results obtained from CFD simulations. In the following section, we show the dataset generation for the ANN model. The ANN model is discussed in section **3.7.3.6**. The results are presented in section **3.7.4**. The conclusions are presented in section **3.7.6.**

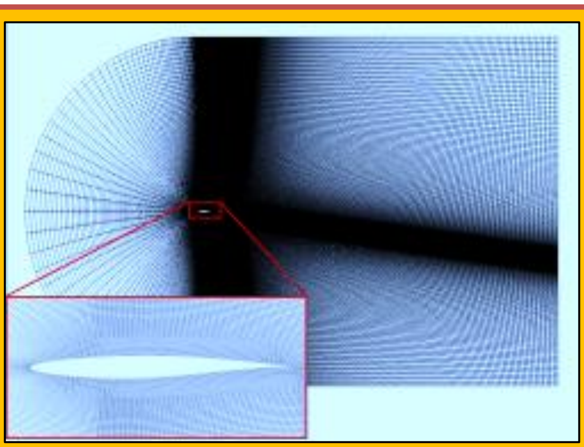


Figure 3.7.1 The discretized domain of the RAE2822 airfoil

3.7.3 Data Set Generation

3.7.3.1 Governing Equations

The transonic flow over a 2D airfoil is simulated by the unsteady compressible Euler equations, which are expressed as:

$$\frac{\partial}{\partial t}\begin{bmatrix}\rho\\ \rho u\\ \rho v\\ \rho e\end{bmatrix}+\frac{\partial}{\partial x}\begin{bmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ u(\rho e+p)\end{bmatrix}+\frac{\partial}{\partial y}\begin{bmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ v(\rho e+p)\end{bmatrix}=0$$

$$\text{where}\quad p=\rho(\gamma-1)\left(e-\frac{u^2+v^2}{2}\right)$$

Eq. 3.7.1

where u and v are respectively the velocity components in x and y-coordinates. ρ, e, and p are the density, internal energy, and pressure of the flow field, respectively. γ is the specific heat ratio of air and equals 1.4. The *rhoCentralFoam* solver in OpenFOAM is adopted to handle the transient, density-based compressible flows.

3.7.3.2 Computational Domain

The **Figure 3.7.1** shows the discretized computational domain of the RAE2822 airfoil. Here, we used a C-section computational domain, with the horizontal quarter center of the airfoil positioned at the origin. This origin point was located at a distance of 20C from the exit, where C represents the chord

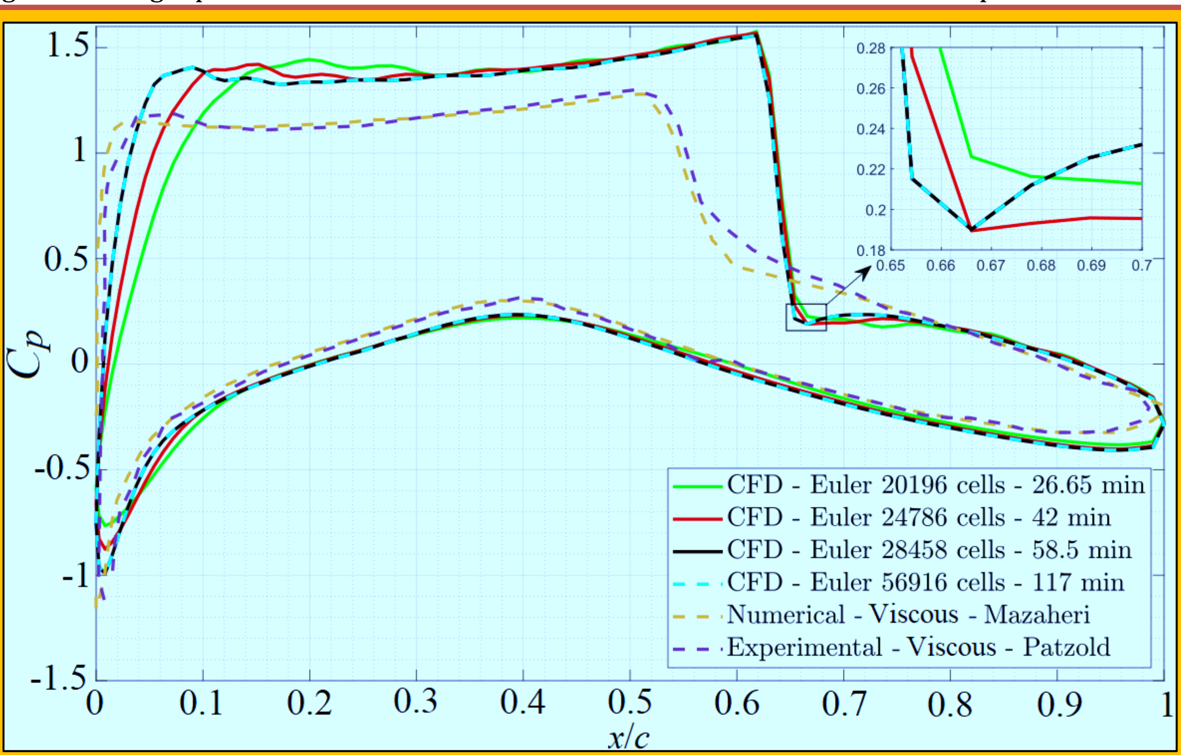


Figure 3.7.2 Grid Convergence Study for the inviscid case about RAE2822 airfoil at M∞ = 0.73, and AOA= 3.19◦

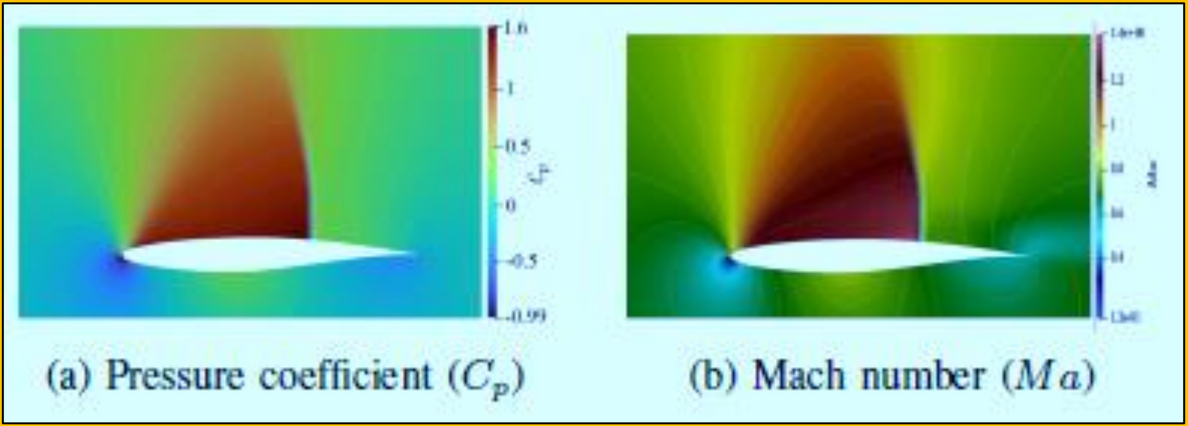


Figure 3.7.3 Pressure coefficient and Mach contours for an inviscid flow over RAE2822 airfoil at M∞ = 0.73, and AOA= 3.19◦

length of the airfoil. The C-section had a radius equal to 10C. We used quadrilateral body-fitted cells around the airfoil. The construction of the mesh was accomplished using an automated airfoil C-grid generator [11].

3.7.3.3 Grid Convergence Study and Validation

A Grid Convergence Study was conducted to verify the accuracy of the computational results obtained for the RAE2822 airfoil. In this study, four grids were generated, to assess the convergence of the solution. The results were then compared and validated against both experimental and numerical data [12]. It is important to note that the data used in [12] pertains to the viscous case, which differs from our specific case, where we focused on solving the inviscid case due to the limited resources (FASTER Texas A&M University HPRC 1 node 64 processors) and the time frame that was available to complete this study. In addition, a validation process to estimate the level of error between the viscous and inviscid cases was conducted, ensuring the reliability of our findings, see **Figure 3.7.2**. The results indicate a significant difference between the 28458-cell and 56916-cell grids. Therefore, the 28458-cell grid is selected. It is important to note that the computational time required for generating data using this grid is still substantial. As a result, we made a conscious decision to prioritize data generation by compromising accuracy. Consequently, we used the 20196-cell grid, which captured the underlying physics and showed the shock wave formation over the airfoils, see **Figure 3.7.3**.

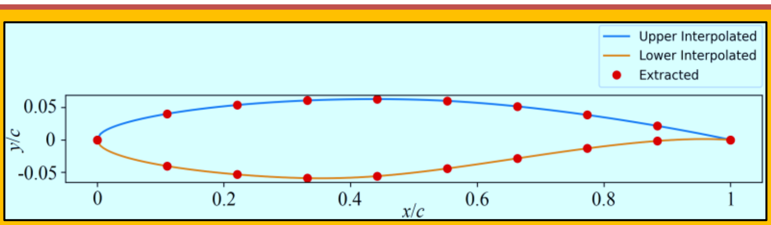


Figure 3.7.4 RAE2822 Airfoil Representation

3.7.3.4 Data Generation

Here, we generate a training dataset, consisting of eight airfoils, namely RAE2822, RAE5212, NACA0012, NACA2412, NACA4412, NACA23012, NACA24112, and NACA25112. For each airfoil, we consider different combinations of wide range values of operational conditions, including the Angle Of Attack (AOA), and freestream Mach numbers (M_∞). The chosen values of AOA range from -2° to 15° with a step size of 0.5°, while that of $M\infty$ range from 0.65 to 0.9 with a step size of 0.05. The airfoil shape is represented by a number of coordinate points N, as discussed in the following section. This comprehensive approach yielded a dataset that consists of 1, 362 data points and (2 + N) input parameters.

3.7.3.5 Airfoil Representation

Here, we represent the airfoil by a set of points along the airfoil surface. We used the University of Illinois at Urbana-Champaign database to obtain the airfoil coordinates x and y of the upper and lower surfaces. For consistency between different airfoils, we normalized each airfoil with its corresponding chord length, so that the chord length equals one. Therefore, the x-coordinate is no longer of interest to be used as the input parameter to the ANN model. Thus, we only use the y-coordinates of the upper and lower surfaces at fixed x-stations

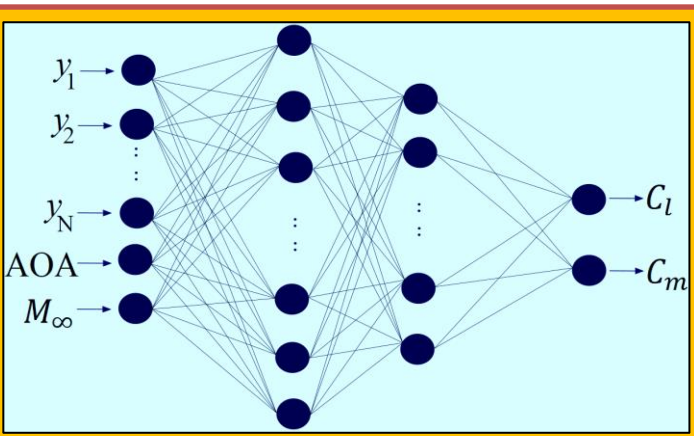


Figure 3.7.5 A schematic of the used Feedforward neural network showing our N +2 inputs as well as our two outputs, C_l, and C_m respectively

among all the airfoils to represent their shape as shown in **Figure 3.7.4**.

3.7.3.6 Artificial Neural Network Architecture

The ANN model is implemented to predict C_l and C_m from the shape and flow parameters of the transonic airfoils. A schematic of the ANN model used is shown in **Figure 3.7.5**. The model is composed of a number of dense fully connected layers. The existence of multiple layers with a large number of nodes per layer helps in capturing complex nonlinear mappings between the input and output using the dataset rather than traditional approaches such as statistical methods. To develop the ANN model, we divided the dataset into three sets using a Train-Validation-Test (TVT) split of 60%, 20%, and 20%, respectively, to ensure appropriate data utilization. The Keras API [13] in Python was used to train the ANN model. The ANN hyperparameters, including the number of layers, neurons in each layer, number of epochs, batch size, learning rate, etc. were tuned using the cross-validation grid search method by trying all possible combinations between the hyperparameters and getting the best performing configuration for training. We found that the best performance occurred when we used a three-layer network with 128, 64, and 32 neurons in the first, second, and third layers, respectively. The optimal number of epochs and batch size are, respectively, 350 and 12. The loss function used is defined as the Mean Squared Error (MSE) function as the loss function, which is given by

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Eq. 3.7.2

where n is the number of data points in the dataset, and yi and ˆyi represent the actual and predicted values for the i-th data point. The Rectified Linear Unit (ReLU) function ($\sigma(x) = \max(x, 0)$) was used as the activation function for the ANN, where x is any arbitrary value. The Adam optimizer was used to train the ANN model. The performance of the ANN model is evaluated using Root Mean Squared Error (RMSE), the square root of the MSE, and the coefficient of determination (R^2), which is given by

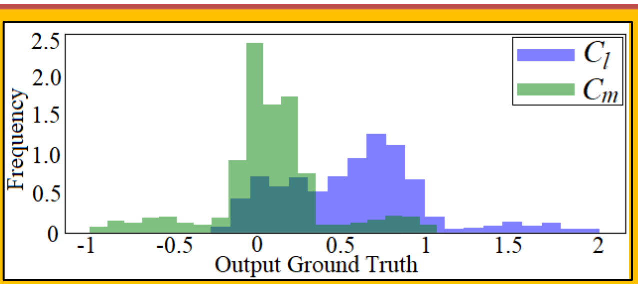


Figure 3.7.6 Histograms of lift (C_l) and moment (C_m) coefficients

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

Eq. 3.7.3

where $\bar{y}$ represents the mean of the true values. For the best performance, RMSE and R^2 should yield zero and one, respectively.

3.7.4 Results and Discussion

The histograms of the C_l and C_m of the training data set are shown in **Figure 3.7.6**. The values of C_l range from −0.2739 to 2.0456 with a mean and standard deviation of 0.5741 and 0.4349, respectively, while the values of C_m range from −0.9958 to 1.0744 with a mean and standard deviation of 0.0714 and 0.3392. There are values of C_l less than zero because the training dataset contains a number of negative angles of attack. The airfoil shape is represented by the y-coordinates of the airfoil's upper and lower surfaces at predefined evenly distributed x-stations, as discussed previously. **Table 3.6.2** shows the performance of the ANN model when we choose a different number of stations. The table shows that evenly distributed 16 points (eight on each surface) perform better than points 10 and 20.

The ANN model is trained using the back-propagation algorithm which consists of forward and backward passes. In the forward pass, the ANN model estimates the output using randomly initialized weights and biases. Then, a backward pass is performed to adjust the weights and biases using the optimization technique by calculating the loss function between the estimated and true values. Each forward and backward pass is called an epoch. **Figure 3.7.7** shows the evolution of the loss function of the validation and training data sets versus the number of epochs. The plot shows that the ANN model does not overfit because the curves concurrently decrease as the number of epochs increases until they asymptotically approach a constant value equal to 0.000134. **Figure 3.7.8** shows the scatter plot of the predicted C_l and C_m values against their corresponding ground truth values using the test dataset. Ideally, the scatter points should be on the top of the regressed diagonal line with a slope equal to one, which represents a perfect fit. Also, the correlation coefficients of C_l and C_m are found to be 0.999, while the corresponding RMSE are respectively 0.00291 and 0.00319. Given that the mean values of C_l and C_m are respectively 0.5741 and 0.0714, and the good agreement in the scatter plot, these results assert the goodness of the ANN predictions.

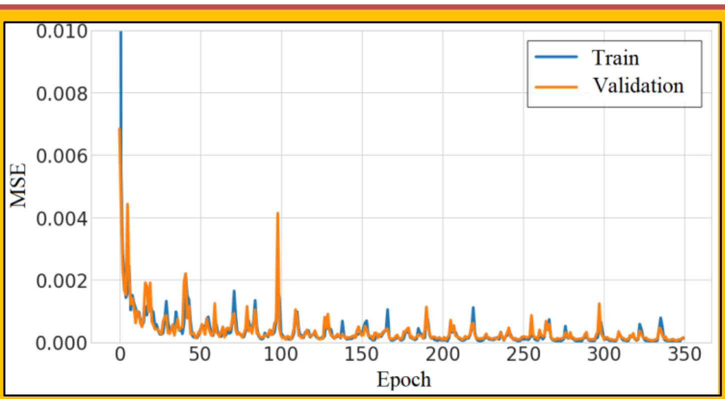


Figure 3.7.7 Validation and Training MSE vs Number of Epochs

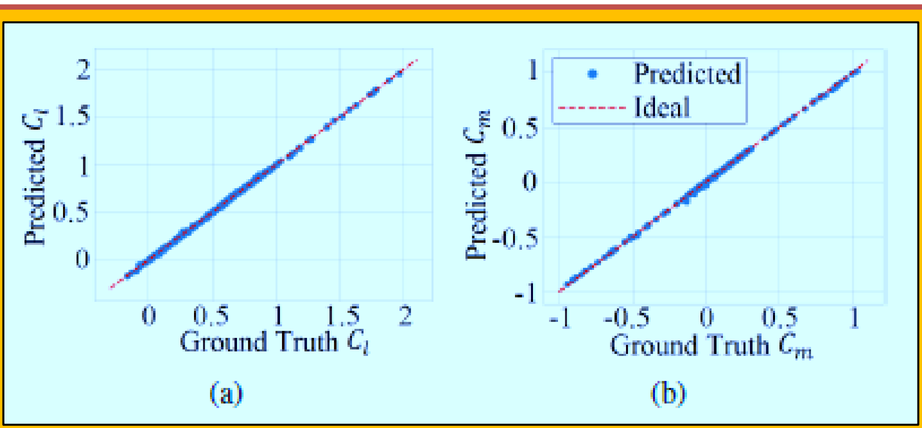


Figure 3.7.8 Scatter plots of the predicted (a) C_l, and (b) C_m versus the corresponding ground truth values using the Unseen Test Dataset

No. of Coordinate points (N)	$RMSE$		R^2	
	C_l	C_m	C_l	C_m
10	0.00387	0.00344	0.998	0.997
16	0.00291	0.00319	0.999	0.999
20	0.00771	0.00428	0.999	0.999

Table 3.7.1 Network performance due to varying airfoil Coordinate points

3.7.5 Conclusion

In this paper, we show the usefulness of using artificial neural networks (ANNs) to reduce the computational burden of predicting the lift and moment coefficients of airfoils in the transonic regime. The ANN model was trained using a dataset that was generated using OpenFOAMR software by employing Euler CFD simulations. The training data set consists of 1, 362 data points, which are formed from a combination of eight airfoils operating under different flight conditions, that is, angle of attack and freestream Mach number. The inputs to the ANN are the flight conditions and the airfoil shape, which is represented by the coordinates of a number of points along the airfoil's upper and lower surfaces. The results demonstrated the efficiency of the data-driven method in accurately predicting aerodynamic coefficients in a fraction of the time compared to the CFD simulations.

3.7.6 References

[1] Koki Nankai, Y. Ozawa, Taku Nonomura, and K. Asai, "Linear Reducedorder Model Based on PIV Data of Flow Field around Airfoil," vol. 62, no. 4, pp. 227–235, Jan. 2019, doi: https://doi.org/10.2322/tjsass.62.227.

[2] Y. Zhu, Y. Ju, and C. Zhang, "Proper orthogonal decomposition assisted inverse design optimisation method for the compressor cascade airfoil," Aerospace Science and Technology, vol. 105, p. 105955, Oct. 2020, doi: https://doi.org/10.1016/j.ast.2020.105955.

[3] S. Suresh, S. N. Omkar, V. Mani, and T. N. Guru Prakash, "Lift coefficient prediction at high angle of attack using recurrent neural network," Aerospace Science and Technology, vol. 7, no. 8, pp. 595–602, Dec. 2003, doi: https://doi.org/10.1016/s1270-9638(03)00053-1.

[4] E. Yilmaz and B. German, "A Convolutional Neural Network Approach to Training Predictors for Airfoil Performance," Jun. 2017, doi: https://doi.org/10.2514/6.2017-3660.

[5] H. Moin, H. Zeeshan Iqbal Khan, S. Mobeen and J. Riaz, "Airfoil's Aerodynamic Coefficients Prediction using Artificial Neural Network," 2022 19th International Bhurban Conference on Applied Sciences and Technology (IBCAST), Islamabad, Pakistan, 2022, pp. 175-182, doi: http://doi.org/10.1109/IBCAST54850.2022.9990112.

[6] Y. Zhang, W. J. Sung, and D. N. Mavris, "Application of Convolutional Neural Network to Predict Airfoil Lift Coefficient," 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Jan. 2018, doi: https://doi.org/10.2514/6.2018-1903.

[7] X. Hui, J. Bai, H. Wang, and Y. Zhang, "Fast pressure distribution prediction of airfoils using deep learning," Aerospace Science and Technology, vol. 105, p. 105949, Oct. 2020, doi: https://doi.org/10.1016/j.ast.2020.105949.

[8] N. Thuerey, K. Weisenow, L. Prantl, and X. Hu, "Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows," AIAA Journal, vol. 58, no. 1, pp. 25–36, Jan. 2020, doi: https://doi.org/10.2514/1.j058291.

[9] S. Ahmed et al., "Aerodynamic Analyses of Airfoils Using Machine Learning as an Alternative to RANS Simulation," Applied Sciences, vol. 12, no. 10, p. 5194, May 2022, doi: https://doi.org/10.3390/app12105194.

[10] H. Moin, H. Zeeshan Iqbal Khan, S. Mobeen, and J. Riaz, "Airfoil's Aerodynamic Coefficients Prediction using Artificial Neural Network," IEEE Xplore, Aug. 01, 2022. https://ieeexplore.ieee.org/abstract/document/9990112.

[11] curiosityFluidsAdmin1, "Automatic Airfoil C-Grid Generation for OpenFOAM – Rev 1," curiosityFluids, Apr. 22, 2019. https://curiosityfluids.com/2019/04/22/automatic-airfoil-cmeshgeneration-for-openfoam-rev-1/

[12] K. Mazaheri, K. C. Kiani, A. Nejati, M. Zeinalpour, and R. Taheri, "Optimization and analysis of shock wave/boundary layer interaction for drag reduction by Shock Control Bump," Aerospace Science and Technology, vol. 42, pp. 196–208, Apr. 2015, doi: https://doi.org/10.1016/j.ast.2015.01.007.

[13] Chollet, F. & others, 2015. Keras. Available at: https://github.com/fchollet/keras.

3.8 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.

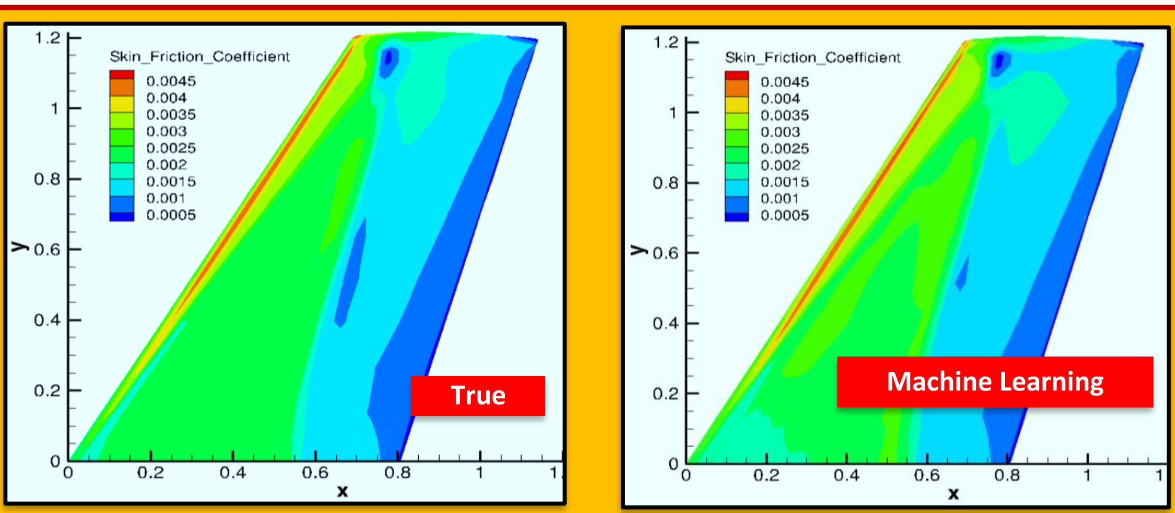


Figure 3.8.1 Skin Friction Coefficient for *Onera M6* match to within 2%

(Ling, J., Kurzawski, A. & Templeton, J., 2016)[11] 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 3.8.1** shows Skin Friction Coefficient for *Onera M6* wing to be matched within 2%[12].

Other researchers such as (Romit Maulik et al.)[13], 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 results from one such experiment (backward steps), where the geometry is 'unseen', in **Figure 3.8.2**.

[11] Ling, J., Kurzawski, A. & Templeton, J. "*Reynolds averaged turbulence modelling using deep neural networks with embedded invariance*", J. Fluid Mech 807, 155–166, 2016.

[12] Karthik Duraisamy, "*A Framework for Turbulence Modeling using Big Data*", NASA Aeronautics Research Mission Directorate (ARMD) LEARN/Seedling Technical Seminar January 13-15, 2015.

[13] Maulik, R., Sharma, H., Patel, S., Lusch, B., & Jennings, E. (2020). Deploying deep learning in OpenFOAM with TensorFlow. *ArXiv, abs/2012.00900*.

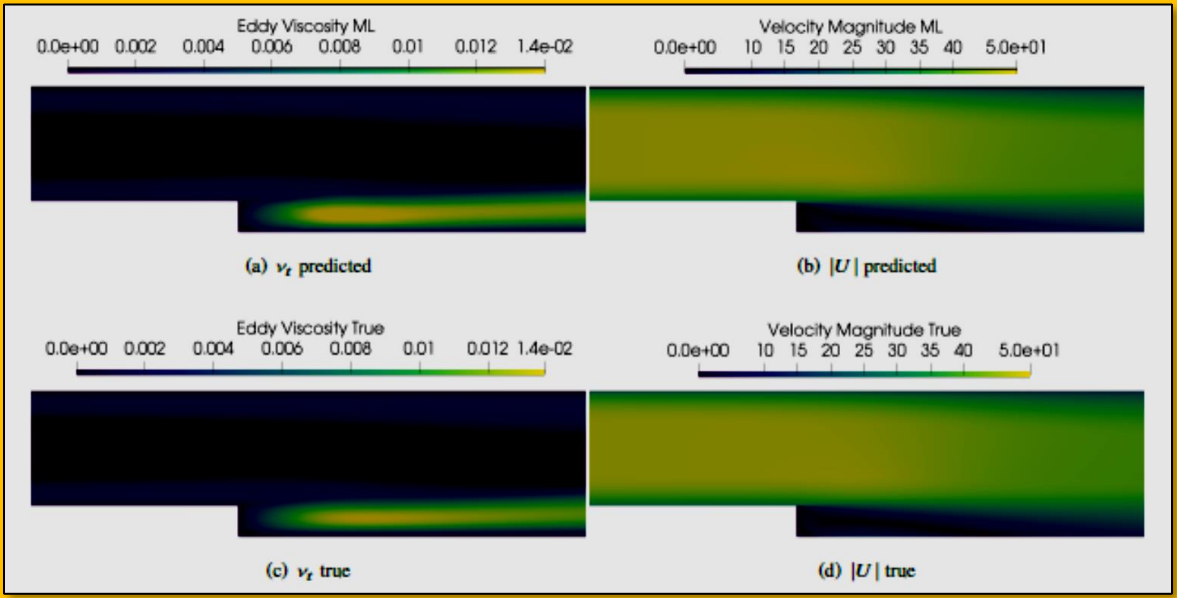


Figure 3.8.1 Contour plots for a backward facing step. Note that the training of the ML surrogate did not include data for the shown step height.

3.9 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.