

“Neural Networks and Machine learning concepts into the applications of Computational Fluid Dynamics”

SUMMER INTERNSHIP REPORT

Under the guidance of

Mr. Kumaran Babu

(CAE Manager, ACRi Infotech Pvt Ltd)



Submitted in Partial Fulfilment of the
Requirement for Award of the Degree
Of

MASTER OF TECHNOLOGY

In

Energy Engineering with specialization in Materials (EEM)

Submitted by

HARSH ARORA

(T18163)

at



SCHOOL OF ENGINEERING

INDIAN INSTITUTE OF MANDI, KAMAND,

DISTT. MANDI, HIMACHAL PRADESH - 175005

July 30, 2019

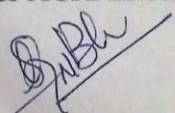
Bangalore.

CERTIFICATE

This is to certify that **Mr. Harsh Arora**, a bonafide student (M.Tech- Energy Engineering) of Indian Institute of Technology - Mandi, Kamand, Himachal Pradesh, India has successfully completed internship from **24th June, 2019** to **30th July, 2019** at **ACRi Infotech Pvt Ltd**. During his internship, he worked on the topic "Neural Networks and Machine learning concepts" in to the "Applications of Computational Fluid Dynamics". During his tenure we found him to be knowledgeable, punctual and hardworking.

We wish him every success in life.

for ACRi Infotech Pvt Ltd.


(Kumaran Babu)

Authorized Signature



AKNOWLEDGEMENT

I feel an enormous feeling of honour and self-achievement after pursuing this highly informative internship at **ACRi infotech Pvt Ltd**. This Internship has been a source of immense practical knowledge and comprehension of modern techniques.

Firstly, I would like to thank my faculty advisor **Dr Jaspreet Randhawa** (Assistant Professor, School of Engineering, Indian Institute of Mandi, Himachal Pradesh) for allowing me to pursue this internship. I would like to thank my guide **Dr Pradeep Kumar** (Assistant Professor, School of Engineering, Indian Institute of Mandi, Himachal Pradesh) for encouraging me to attend this internship. I would like to express my special thanks to **Mr. A. Chakrapani** (Operations Manager, ACRi infotech Pvt Ltd, Bangalore) for provide me this opportunity. I wish to express my profound gratitude to **Dr Madhukar Rao** (Technical Director, ACRi infotech Pvt Ltd, Bangalore) for his valuable direction in my internship project. I would like to express sincere gratitude to my project mentor **Mr. Kumaran Babu** (CAE Manager, ACRi infotech Pvt Ltd, Bangalore) because of whom I could transcend the limits of my tactical approach during the project. He has been the perennial guiding force to me for learning. The cooperation he gave is greatly appreciated. I extend my thanks to all engineers working at **ACRi infotech Pvt Ltd, Bangalore**.

PLACE:
Bangalore

Harsh Arora

TABLE OF CONTENTS

Company's Profile	05
Introduction	06
1. Part-1	
1.1. Machine Learning	06
1.2. Machine Learning models	06
1.3. Artificial Neural Networks (ANN)	07
1.4. Convolutional Neural Networks (CNN)	08
1.5. Long-Short Term Memory (LSTM) Networks	08
2. Part-2	
2.1. How machine learning can be used for CFD applications.....	09
2.2. Deciding a Machine Learning architecture for CFD problems	09
2.3. Advantages and challenges – Data driven vs Numerical approaches	09
2.4. A brief review of related work	10
3. Part-3	
3.1. A Convolutional Neural Network to predict flow field across a 2D Backward Facing Step	11
3.2. An Artificial Neural Network to predict velocities, stream function and vorticity function at a given spatial location and time-step for the case of 2D Backward Facing Step	12
3.2.2 Training accuracy of ANN	13
3.2.3 Validation of ANN predicted data with Numerical simulation	14
3.3. Physics Informed Neural Network	15
Future Work	15
References	16
Appendix - Python code for Artificial Neural Network discussed in section 3.2.....	17

Company's Profile

ACRi - **Analytical and Computational Research Incorporation** is a **software development** and **consulting** organization providing mathematical modelling and computer analysis of environmental pollution and engineering processes involving Fluid Dynamics, Heat & Mass Transfer, Turbulence, and Combustion.

ACRi develops and markets a family of versatile **CFD software tools** that are widely used by leading organizations and universities. These tools are distinguished by their ease of use and versatility for both problem-specific application, but more so for the ability to allow users to quickly implement alternative solvers, physical sub-models, and numerical schemes. Since its founding in 1979, the firm has provided consulting services on an international basis in the various fields of its expertise to over 100 clients. The firm is committed to staying abreast of the state-of-the-art in the rapidly evolving fields of mathematical and computer modelling by in-house development and close contacts with leading universities and professional organizations around the world.

ACRi has subsidiaries in **France** and **India**, and distributors in the **United Kingdom, Spain, Korea** and **France**. In India ACRi is currently present in **Bangalore** and **Dharamshala, Himachal Pradesh**.

Source: [ACRi website](#)

Introduction

The present report mentions the detailed procedure and concepts I have come across and learnt throughout my internship tenure. The part-1 of the report briefly explains about various Machine Learning concepts which were later implemented to solve problems. Part-2 mentions the comparison between traditional CFD methods and the data-driven Machine Learning approaches as well as the advantage and disadvantage of Data driven approach. Also, some literature work where data-driven approach is used to solve for CFD problems is discussed. Part-3 of the report mentions the CFD problems I addressed using different ML (Machine Learning) models, and the future work.

Part-1

1.1 Machine Learning

‘Machine learning’ as the name itself depicts is a technique through which a machine learns like humans do and can perform intelligence tasks similar to humans - like classifying the objects or detecting similarities in different data. (for example, google photos itself detects the same person appearing in different pictures inside the mobile phone’s gallery). ML is currently used in plenty of applications, we are also using it regularly in some or the other way through accessing web and different apps.

To explain how ML model is different than traditional computer code let’s consider an example- if a user has to compute the output using some input, the user will define a function which contains a set of instructions (say an equation which takes input and determines output). But in a machine learning model, a user does not define the equation rather he will just show the input data and the model will predict the output. How this is achieved is through ‘training’ the model before using it for predicting the values.

Training a Machine Learning model involves providing the data – the user has to provide the input and the corresponding output data for a large number of instances. The model itself figure out how the output is changing with input and it itself maps a complex function which is later used later for prediction. The training is done for a single time, once a ML model has learnt it is able to predict output very quickly and as many times a user want. The larger the number of instances used in training (the training data set) the more accurate the model will be.

1.2 Machine Learning Models

There are many different types of Machine learning models which differs from each other in terms of their architecture. These different architectures are specialized for addressing different types of problems based on their data structures i.e. what form of input is there and what form of output is desired or say what a user wants to get out of the data. Whether it’s a classification problem, a regression, a clustering or something else, the architecture and hence the type of model is chosen based on this. The explanation for each of the ML model is out of the scope for this report, hence only the models which are later used for dealing with CFD problems are briefly explained below.

1.3 Artificial Neural Networks (ANN)

An artificial neural network is a machine learning architecture inspired from the biological neural network. An artificial neural network is similar to a biological neural network inside our body, a single neuron receives the information from other neurons, process it and then transfer that information to the next n number of neurons. In an ANN also a single node (acting as a neuron) takes information from a number of similar nodes (neurons) in its previous layer, process it and transmits it to the neurons present in the next layer. So typically, an ANN consists of layers of neurons, where first layer is known as **input layer**, the final layer is the **output layer** and the layers in between them are called as **hidden layers**. The layers contain different number of nodes (neurons), and different activation functions (which decides how they process the input). In an ANN weights are assigned to each of the output information coming from any neuron of any layer and feeding to any other neuron of the other layer. [Fig.2]

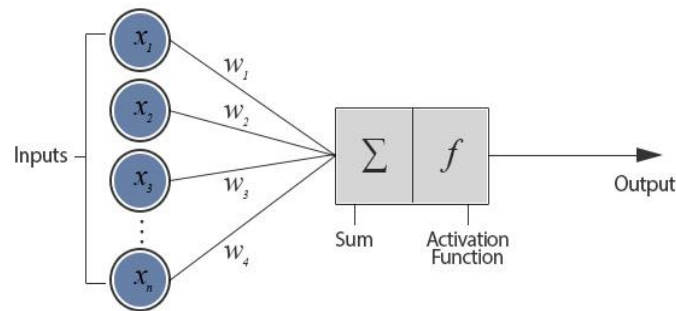


Fig.1

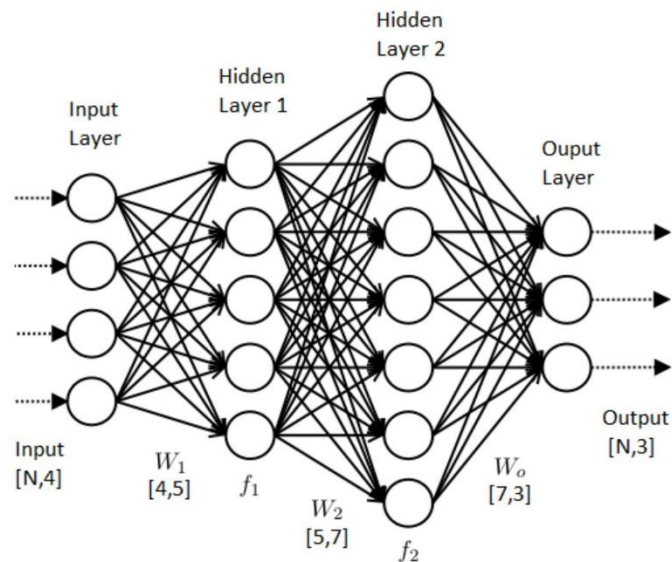


Fig.2

Fig.1 shows a simple perceptron however this is also a Neural Network. A neural network with two or more layers is called a deep neural network and hence the learning associated with it is called as **Deep Learning**. While training an ANN, the weights (as shown in **Fig.3**) get updated as per the loss function which back propagates the error to the network and this is how learning takes place which gradually improves the weights to achieve the accuracy in prediction.

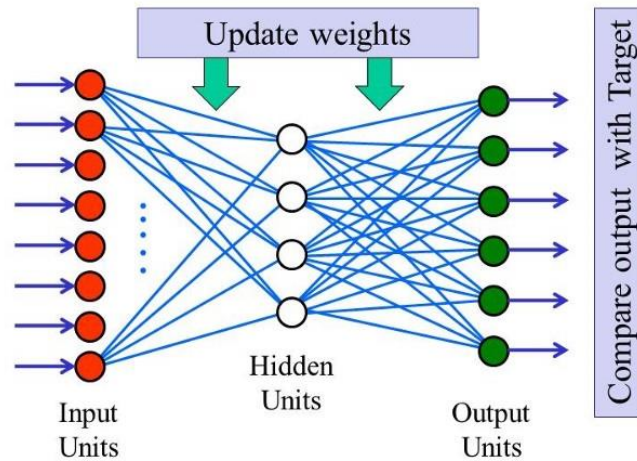


Fig.3

1.4 Convolutional Neural Networks (CNN)

An ANN takes the 1-dimensional or say the tabular column data as input (one row as a single instance). However, Convolutional Neural Networks have special ability to take two and three-dimensional data as input and further process and down sample this input (still keeping the necessary information) with the help of some convolution and pooling functions to feed it to an ANN. In a CNN architecture, input layer is followed by some convolution and pooling layers, a flatten layer is then used to make the data one dimensional to feed it into an ANN. The architecture is shown in Fig.4 below.

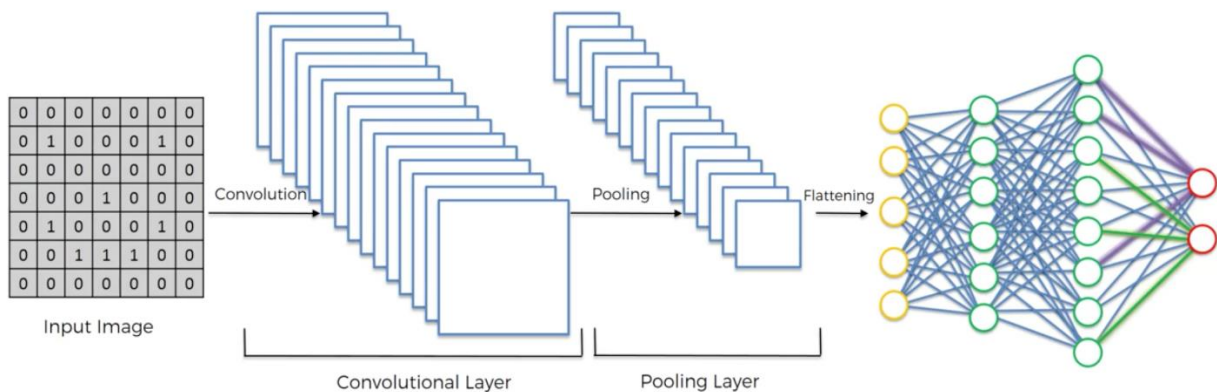


Fig.4

1.5 Long-Short Term Memory (LSTM) Networks

LSTM networks are classified under recurrent type neural networks, they have some memory. An LSTM node determines its output based on its memory of previous N number of times. Therefore, LSTM are great for time-series predictions and are widely used in applications like stock prediction. In the current project I haven't used LSTM but it could be useful for unsteady simulations in CFD.

Part-2

2.1 How machine learning can be used for CFD applications

From a CFD perspective Machine Learning doesn't solve a fluid flow problem (we are not providing any governing equations to the model), rather the model learns the solution for the instances we are feeding while training it. Later, when trying to make predictions for a new input instance it will predict the output based on the mapping it developed at the time of learning. For a CFD case the training dataset includes input instances as the Mesh information (spatial co-ordinates), thermophysical properties, boundary conditions etc. and the output instances includes what we want our Machine Learning model to predict i.e. let's say u and v velocities or maybe any other physical quantities. For generating the whole training set we have to first perform the simulations for a large number of times (using a traditional numerical approach- using a standard solver like OpenFoam, ANSYS Fluent, or maybe any stand-alone code). This could be very time consuming but is to be done only a single time. Later, when the ML model will be used it can perform the same simulation with in small fraction of time consumed by a standard CFD solver (or maybe a couple of seconds).

2.2 Deciding a Machine Learning architecture for CFD problems

A Machine Learning model is all about the data we feed into it and the data we want as its output, no matter how complex the governing equations are an ML model has capabilities to deal with non-linearities and develop complex mappings between the inputs and outputs. Choosing an ML model however depends upon the data structure of input and output, the complexities between the input and output etc. Looking at these aspects only the complex models like Artificial Neural Networks (explained in section 1.3) can deal with such cases. Therefore, deep learning is used to address such problems. In CFD problems we mostly deal with grids – for a 2D fluid flow problem the input and output could be a set of 2D tensors (matrices), on the other hand if the problem is of 1-dimensional the input and output data structure can be taken as tabular data with some number of columns. For both the above cases different ML models have to be chosen, for the former one a Convolutional Neural Network (explained in section 1.4) would be required to take 2D tensors as input, for the latter case an ANN could do the job. Once the architecture is decided the next step is training the model, for this we have to feed the model with large training data set to perform learning. However, in some cases the tuning of hyperparameters to achieve good accuracy is a big deal and this issue can be expected while dealing with non-linear CFD problems. In Part-3 of this report two CFD problems are discussed in detail, in both of them different ML models are used.

2.3 Advantages and challenges – Data driven vs Numerical approaches

The advantages of using ML or say a data-driven approach to deal with CFD problems are: Firstly, it could be time efficient - specially for CFD problems in which iterative schemes are used in the solver or for cases with large computational grids. Let's understand this with an example- solving for fluid flow across a domain mostly requires solving Pressure Poisson equation in pressure corrector step (in PISO as well as SIMPLE algorithm which are widely used currently in most of the commercial CFD software). Since it uses an iterative scheme to solve the pressure equation it is the most time-consuming part while performing a fluid flow simulation. Talking about the ML approach, though initially training the model will take time but while using it for prediction time taken would be very less

as compared to the case of a standard commercial solver. Secondly, it can be used to solve very complex problems for which experimental results are available but numerical approach doesn't yield accurate approximations, maybe where multiple phenomenon is taking place simultaneously and the simulation can't be done using standard solvers available for CFD problems.

The challenges in the Machine Learning data-driven approach is to tune the hyperparameters while dealing with 2D and 3D data as inputs and outputs to the neural network. Since the number of hyperparameters are large for such complex Neural Networks consisting of many Convolution layers and pooling layers followed by dense hidden layers and further deconvolutions or convolution transpose layers. The other challenges could be preparing large data for training set. The predicted data using an ML model may not satisfy the physical laws very precisely, however it can provide values close to what will satisfy the governing equations. An attempt is made to solve this issue using a Physics Informed Neural Network (PINN) which is explained later under section 3.4.

2.4 A brief review of related literature work

A deep learning-based approach is followed by [1] for predicting turbulent flow field data for a range of time using available previous temporal range data as an input. Reduced Order Modelling is used, the data is first converted from a high dimensional space to a low dimensional subspace this is done by utilizing a dimensionality reduction technique called Proper Orthogonal Decomposition (POD). Then the LSTM neural network (introduced in section 1.5 above) is used to make time series prediction in POD modes. This work sets a great example for demonstrating the capability of LSTMs and their advantages over other methods as they have very low computational cost as compared to LES/DNS for high fidelity simulations.

Another appraisable work is done by [2], they have presented five different types of Machine Learning frameworks suitable for dealing with different cases of Thermal Fluid Simulations. These frameworks include physics-separated ML (PSML or Type I ML), physics-evaluated ML (PEML or Type II ML), physics-integrated ML (PIML or Type III ML), physics-recovered (PRML or Type IV ML), and physics-discovered ML (PDML or Type V ML). Few case studies are also provided with the justification for which type of the framework is most suitable for which case. Deep learning has also been used for industrial applications of thermal fluid simulation such as a data-driven approach based on deep feedforward neural networks is studied.[3] The proposed model uses near wall local features to predict the boiling heat transfer. The inputs of Neural network include the local momentum and energy convective transport, pressure gradients, turbulent viscosity, and surface information. The outputs of the networks are the quantities of interest of a typical boiling system, including heat transfer components, wall superheat, and near wall void fraction. The networks are trained by the high-fidelity data processed from first principle simulation of pool boiling under varying input heat fluxes. Application of Machine learning in Fluid dynamics is an emerging area of research with several efforts from researchers as Zhang and Duraisamy [4], Anand [5], Ling [6] etc.

Part-3

During my internship tenure, I made attempts to address the following work related to solve CFD problems using Machine Learning approach:

3.1 A Convolutional Neural Network (CNN) to predict flow field across a 2D Backward Facing Step:

Convolutional Neural Networks are used for image classification in which the input is a coloured image which is mathematically a pair of three 2D tensors since pixels are made of Red, Green and Blue colours so a coloured image can be represented as a collection of three (one for each RGB) different 2D tensors with each element of the matrices having a number between 0 to 1 representing the intensity of that colour (R,G or B) at that particular location. In data structure terminology it can be called as 3D tensor with 3 channels, hence a coloured image has the input shape of $N \times M \times 3$, where $N \times M$ is the dimension of pixels of image. An analogous approach can be made for CFD problems with input data as matrices (the same dimension as of a 2D grid) and with each channel representing a different physical quantity across the space, so we can input any number of channels depending upon what parameters we want to give it as input. The input shape as we see is similar for the image classification problem as well as for a CFD problems (only differs in terms of number of channels and that's completely fine). However, the output shape for image classification task is only a flat layer with few numbers of nodes (depending upon how many different categories are there in the data set). For say a classification problem to classify dogs and cats will have only two nodes in the output layer. On the other hand, for CFD problems we need output as a physical quantity across the whole grid space which makes the output shape similar to the input shape – a 3D tensor or at least a 2D tensor if we want to predict a single physical quantity. This is where it differs from a typical CNN commonly used for image processing and this also leads to a whole new architecture to deal with CFD problems.

The training data information for the problem case:

	shape	Data
Inputs	(150,48,48,2)	Channel 1: U-velocity at boundary Channel 2: time-step
Outputs	(150, 48,48,1)	U-velocity across the grid for the corresponding time-step
No. of instances for training	150	

Since, the output shape is also 2 dimensional so this special architecture involves some Convolution transpose layers to up sample the data from a single flatten layer (obtained after dense hidden layers) to a 2D tensor shape. The Fig.5 below shows the details about the layers in the used architecture.

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	(None, 48, 48, 2)	0
conv2d_13 (Conv2D)	(None, 46, 46, 16)	304
max_pooling2d_13 (MaxPooling)	(None, 23, 23, 16)	0
conv2d_14 (Conv2D)	(None, 21, 21, 16)	2320
max_pooling2d_14 (MaxPooling)	(None, 11, 11, 16)	0
conv2d_15 (Conv2D)	(None, 9, 9, 32)	4640
max_pooling2d_15 (MaxPooling)	(None, 5, 5, 32)	0
conv2d_16 (Conv2D)	(None, 3, 3, 32)	9248
max_pooling2d_16 (MaxPooling)	(None, 2, 2, 32)	0
flatten_4 (Flatten)	(None, 128)	0
dense_7 (Dense)	(None, 128)	16512
dense_8 (Dense)	(None, 144)	18576
reshape_4 (Reshape)	(None, 12, 12, 1)	0
conv2d_transpose_7 (Conv2DTr	(None, 24, 24, 1)	2
conv2d_transpose_8 (Conv2DTr	(None, 48, 48, 1)	2
Total params: 51,604		

Fig. 5

For the current case with the above shown architecture the model was trained using the training set consisting of data from 150 simulations (performed using 15 different velocity boundary conditions and the data was obtained at 10 different time steps for each of the boundary condition). With the current hyperparameters the model was not able to perform learning. Since this a complex architecture it requires optimization with tuning the hyperparameters like number of dense layers, number of nodes in each of those layers, activation functions across the layers, optimizer, loss function.

3.2 An Artificial Neural Network to predict velocities, stream function and vorticity function at a given spatial location and time-step for the case of 2D Backward Facing Step:

An ANN model is developed for predicting different physical quantities (U-velocity, V-velocity, Stream function value and vorticity function value) across the grid for different time -steps for a fixed geometry and boundary condition. This sounds a very trivial case which is of no direct use though but it will help

for the understanding of how the ML model could learn the simulation results and perform well if we want the information at a different time-step for which we have not trained it. This study could also be useful later to implement other models for relevant CFD problems.

Unlike the previous case which uses inputs and outputs as 2D tensors, this model is an ANN model which accepts tabular data with columns as different features and rows as different instances of data. The simulation data obtained was first reshaped from a 2D matrix to a vector, so a physical quantity across a 51×31 dimensional grid is now reshaped to 1581 cells in a single column. Obtaining simulation data at 15 different time steps results into 23715 instances of data. This set is then split for training and testing as 80-20 percent.

The training data information for this case:

	shape	Data
Inputs	(18972,3)	X co-ordinate, Y co-ordinate, time-step
Outputs	(18972,4)	U-velocity, V-velocity, Stream Function, Vorticity Function
No. of instances for training	18972	

The architecture is quite simple in this case as follows:

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	(None, 3)	0
dense_12 (Dense)	(None, 256)	1024
dense_13 (Dense)	(None, 256)	65792
dense_14 (Dense)	(None, 4)	1028
Total params: 67,844		

Fig.6

3.2.2 Training accuracy of ANN

In the first few epochs itself the model was able to achieve more than 95 percent accuracy, training it till 100 epochs gave 97.5 percent accuracy which is good without hyperparameter tuning and as per the size of training data. Increasing the training data size and a little hyperparameter tuning could result into accuracy as high as 99% or maybe more.

```

Train on 18972 samples, validate on 4743 samples
Epoch 1/100
18972/18972 [=====] - 5s 272us/step - loss: 0.1866 - acc: 0.7038 -
val_loss: 0.1821 - val_acc: 0.7105
Epoch 2/100
18972/18972 [=====] - 3s 165us/step - loss: 0.1408 - acc: 0.7747 -
val_loss: 0.1048 - val_acc: 0.8535
Epoch 3/100
18972/18972 [=====] - 3s 167us/step - loss: 0.0830 - acc: 0.8892 -
val_loss: 0.0706 - val_acc: 0.9300
Epoch 4/100
18972/18972 [=====] - 3s 168us/step - loss: 0.0631 - acc: 0.9339 -
val_loss: 0.0573 - val_acc: 0.9464
Epoch 5/100
18972/18972 [=====] - 3s 165us/step - loss: 0.0553 - acc: 0.9469 -
val_loss: 0.0505 - val_acc: 0.9538

.....

Epoch 100/100
18972/18972 [=====] - 3s 166us/step - loss: 0.0249 - acc: 0.9723 -
val_loss: 0.0244 - val_acc: 0.9751

```

Fig. 7 Training information

3.2.3 Validation of ANN predicted data with Numerical simulation

A numerical code to solve flow field across a 2D backward facing step was used to generate the training set for the Machine Learning model, later after training the predicted data from the model was compared with the Numerical simulation. (for the same boundary condition and time-step), Fig. 8 below shows the stream lines plotted with the predicted data from the ANN model and from the Numerical simulation.

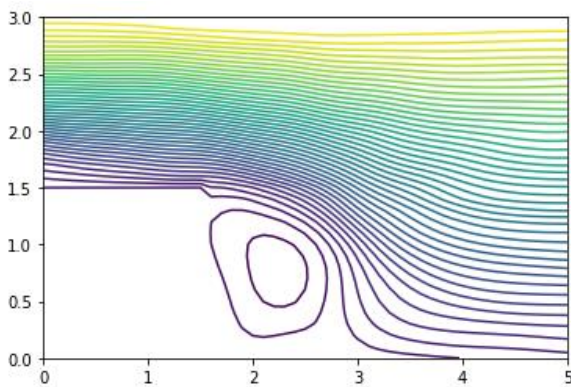


Fig. 8(a): streamline plot using ML predicted data

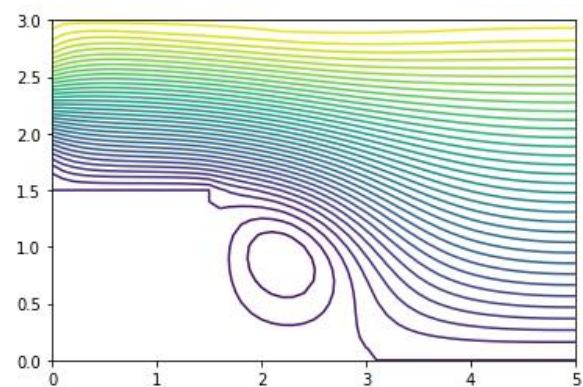


Fig 8(b): streamline plot of Numerical Simulation

3.3 Physics Informed Neural Network (PINN) – A Neural Network Learning Physics

Physics informed Neural Network is a modification to any Neural Network in a way that it satisfies the governing equations, this could be done by defining a custom loss function which will take the predicted values from the network output calculates the gradients and equates them to satisfy the governing equation this way the error is determined as the residual (the deviation from the actual value), this residual is then back propagates through the network (as a loss function) and the weights get updated after each batch in order to minimize the loss (residual) and make the predictions more accurate. A pictorial representation for this is shown in Fig. 8 below. Physics Informed Neural Network is just a concept introduced by Raissi Et al. [7], this has not been incorporated in any standard models as of now. It is estimated that with using PINN the training could become highly efficient as compared to a normal ANN and within very few epochs the network could achieve high accuracy. Raissi in his research work used TensorFlow to develop Neural Networks and train them. Keras and TensorFlow are the most popular libraries used for practicing Machine learning with Python. TensorFlow is a low-level library however Keras is a high-level library built above TensorFlow just to make codes short and easier.

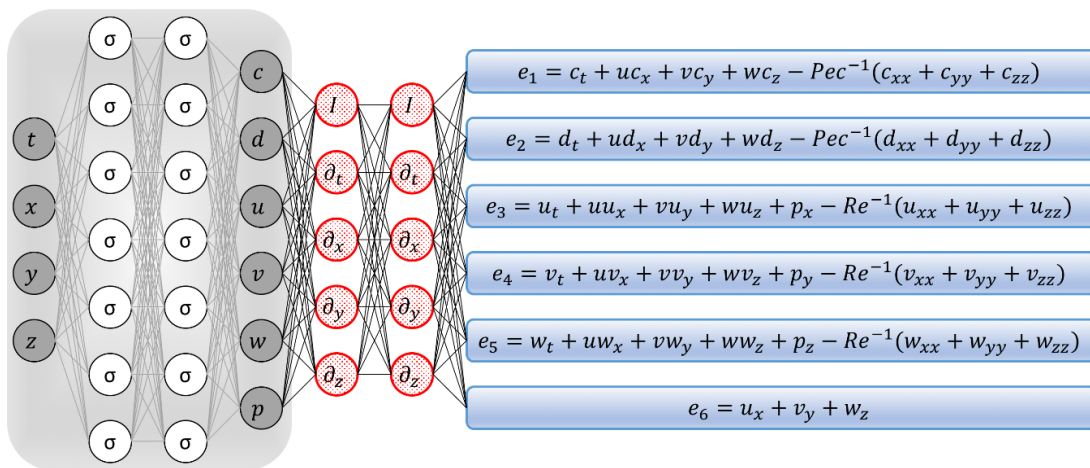


Fig. 9 Physics Informed Neural Network

Future work

In my internship project, I have used Keras for developing all the codes. I also tried to define the custom loss function for the ANN explained in the section 3.2, but I figured out that it can only be possible using TensorFlow, since customisation requires to access the loss function from the root which can only be possible through a low-level library which offers more flexibility in terms of defining each and every function. So, the next thing is to practice TensorFlow for defining things in a custom manner to deal with CFD or any other physics problems. The long term plan is to use deep learning approach in the applications of turbulence modelling.

References

- [1] A. T. Mohan and D. V. Gaitonde, “A Deep Learning based Approach to Reduced Order Modeling for Turbulent Flow Control using LSTM Neural Networks,” 2018.
- [2] C. W. Chang and N. T. Dinh, “Classification of machine learning frameworks for data-driven thermal fluid models,” *Int. J. Therm. Sci.*, vol. 135, pp. 559–579, 2019.
- [3] Y. Liu, N. Dinh, Y. Sato, and B. Niceno, “Data-driven modeling for boiling heat transfer: Using deep neural networks and high-fidelity simulation results,” *Appl. Therm. Eng.*, vol. 144, no. August, pp. 305–320, 2018.
- [4] Z. J. Zhang and K. Duraisamy, “Machine learning methods for data-driven turbulence modeling,” *22nd AIAA Comput. Fluid Dyn. Conf.*, no. June, pp. 1–18, 2015.
- [5] A. P. Singh, S. Medida, and K. Duraisamy, “Machine-Learning-Augmented Predictive Modeling of Turbulent Separated Flows over Airfoils,” *AIAA J.*, vol. 55, no. 7, pp. 2215–2227, 2017.
- [6] J. Ling, A. Kurzawski, and J. Templeton, “Reynolds averaged turbulence modelling using deep neural networks with embedded invariance,” *J. Fluid Mech.*, vol. 807, pp. 155–166, 2016.
- [7] M. Raissi, A. Yazdani, and G. E. Karniadakis, “Hidden Fluid Mechanics : A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data arXiv : 1808 . 04327v1 [cs . CE] 13 Aug 2018.”

Appendix – (Python code for Artificial Neural Network discussed in section 3.2)

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jul 10 22:53:49 2019

****'Artificial neural network for backward facing step'****

@author: Harsh Arora
"""

#data loading and preprocessing
import pandas as pd
import glob
import os

datadir=(r'C:\Users\Harsh Arora\Desktop\ML-python\PINN\simulation_data')
filenames=glob.glob1(datadir,'*.csv')

dataset=pd.read_csv(r'C:\Users\Harsh Arora\Desktop\ML-
python\PINN\simulation_data\t_5.csv',header=None, index_col=None)
for f in range(1,15):
    f1=os.path.join(datadir,filenames[f])
    f2=pd.read_csv(f1,header=None, index_col=None)
    dataset=pd.concat([dataset,f2],axis=0, sort=False)

# Defining input and output features
X=dataset.iloc[:,0:3].values
Y=dataset.iloc[:,3:7].values
```

```

# splitting into training set and testing(validation) set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 1)

# Feature Scaling- coz we don't want one feature to dominate the other
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
Y_train = sc.fit_transform(Y_train)
Y_test = sc.transform(Y_test)

# Defining the ANN architecture
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense

main_input = Input(shape=(3,))
hidden1 = Dense(256, activation='tanh', kernel_initializer="normal")(main_input)
hidden2 = Dense(256, activation='tanh')(hidden1)
main_output = Dense(4, activation='sigmoid')(hidden2)

#compiling the model
model = Model(inputs=main_input, outputs=main_output)
print(model.summary())

#defining a custom loss function
import keras.backend as K
def custom_loss(y_true, y_pred):

```

```

K.print_tensor(y_pred)

return K.sqrt(K.sum(K.square(y_pred-y_true), axis=-1))

#compiling the model
model.compile(optimizer='adam', loss=custom_loss, metrics=['accuracy'])

#fitting the data
model.fit(X_train, Y_train, batch_size = 12, epochs = 100, validation_data=(X_test,Y_test))

# saving model(weights)
model_json = model.to_json()
with open("ANN_bfs_model.json", "w") as json_file:
    json_file.write(model_json)

#serialize weights to HDF5
model.save_weights("ANN_bfs_model.h5",overwrite=True)
print("Saved model to disk")

# loading a saved model architecture (json)
from keras.models import model_from_json
json_file = open('ANN_bfs_model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)

# loading weights into new model
loaded_model.load_weights("ANN_bfs_model.h5")
loaded_model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
print("Loaded and compiled model from disk")

#importing and rearranging data for prediction

```

```

from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler()

import pandas as pd

X_predict=pd.read_csv(r'C:\Users\Harsh Arora\Desktop\ML-
python\PINN\predict.csv',header=None, index_col=None)

X_predict=X_predict.iloc[:,0:3].values

X_predict = sc.fit_transform(X_predict)


#fixing the min-max scaling issue manually
for i in range(1581):

    X_predict[i,2]=0.4922


#predicting using loaded model
Y_pred = loaded_model.predict(X_predict)


u_pred=Y_pred[:,0]          #predicted u velocity
v_pred=Y_pred[:,1]          #predicted v-velocity
stream_pred=Y_pred[:,2]     #predicted stream function
vort_pred=Y_pred[:,3]       #predicted vorticity function


#analyzing predicted stream function
s_min=-0.13763
s_max=1.0002954
import numpy as np
s_transformed=np.zeros((1581,))


#inverse transformation to get the original values
for i in range(1581):

    s_transformed[i] = stream_pred[i]*(s_max-s_min) + s_min

```

```
#reshaping the vector into grid
s_grid=np.reshape(s_transformed,(51,31))

for i in range(16):
    for j in range(16):
        s_grid[i,j]=0

#plotting stream lines
import matplotlib.pyplot as plt
x1=np.linspace(0,5.0,51)
y1=np.linspace(0,3.0,31)
yv,xv=np.meshgrid(y1,x1)
plt.contour(xv,yv,s_grid,50)
```