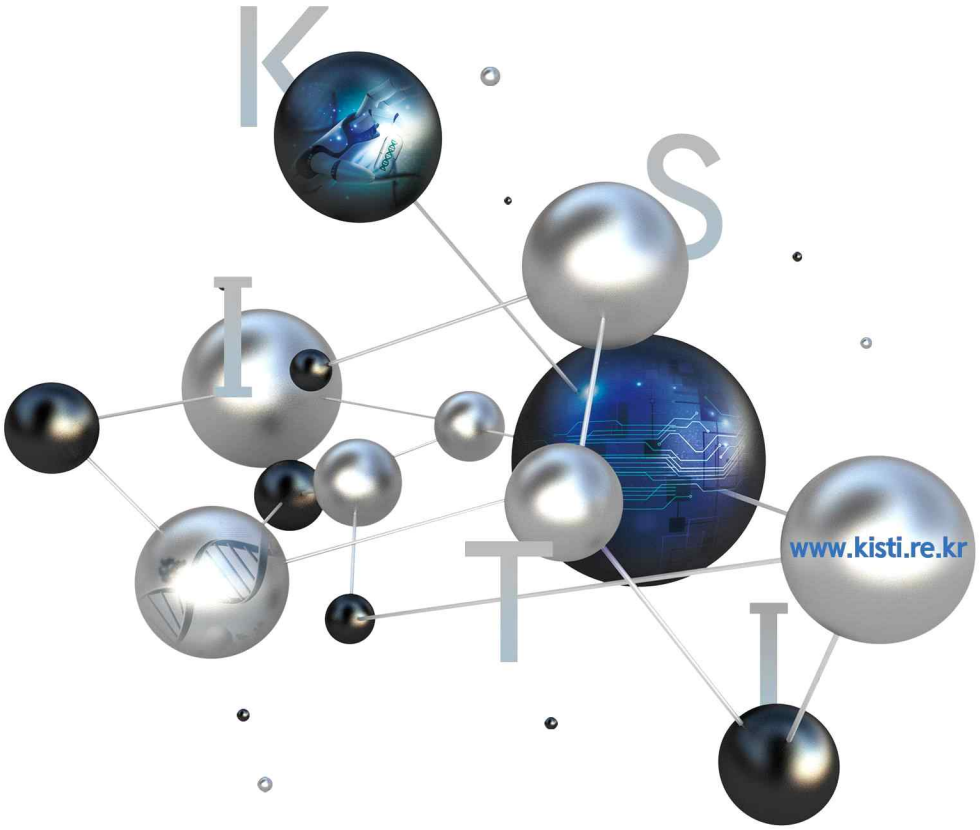
KISTI
RESEARCH
REPORT

Hands-On AI-CFD Technical Report

Jung-Hun Shin, Kum-Won Cho / KISTI

Seong-Im Choi, Won-Tae Hwang / GIST



KiSTi 한국과학기술정보연구원
www.kisti.re.kr
Korea Institute of Science and Technology Information

Hands-On AI-CFD Technical Report

Junghun Shin[1], Seongim Choi[2], Wontae Hwang[2], Kumwon Cho[1]

[1]Korea Institute of Science and Technology Information, Daehak-ro, Yuseong-gu, Daejeon 34141
[2]Gwangju Institute of Science and Technology, 123 Cheomdangwagi-ro, Buk-gu, Gwangju, 61005

Abstract

Data-driven researches based on artificial intelligence techniques, are being actively explored in science and technology areas. In the area of computational fluid dynamics, several studies have already been conducted in a reduced order modeling technique using principal orthogonal basis called proper orthogonal decomposition (POD). Recently the application of artificial intelligence techniques also represented by deep learning which include many modeling infrastructure: fully connected deep neural network (FC-DNN), convolutional neural network (CNN), recurrent neural network (RNN), et cetra to the existing reduced order modeling is increasingly drawing attention with the development of processor hardware technology such as GPU or other AI processors. Since this technical report was initially planned for the AI beginner who were already used to CFD for airfoil, it was developed in a way of hands-on tutorial so that readers can follow the first-time going procedure and learn the characteristic of AI implementation. The content of three complexity levels or chapters were made up with corresponding sample code and dataset provided from repositories: a) unsteady case with fixed airfoil shape and fixed flow condition, b) steady case with fixed airfoil shape and variable flow conditions, and c) steady case with variable airfoil shapes and variable flow conditions. The dataset was produced by a state-of-the-art RANS flow solver using structured curved grid, and learned by recently verified AI techniques to some degree such as POD, CNN (specifically U-net), RNN (specifically LSTM), and their mixed ones. The flow conditions of the target airfoil problem as Reynolds number, Mach number, and angle-of-attack. The comparisons of inference errors between AI techniques were provided at the end of the chapters. Although generative network and reinforcement learning models attracted researchers' attention as well in the field of AI, they were not included in this report. After learning this report, ones could use this report document as a hands-on tutorial material for your neighbors.

Acknowledgment

This work was supported by the National Research Council of Science & Technology (NST) grant by the Korea government (MSIT) (No. CRC21011)

CONTENTS

01 Introduction

02 Case 1) Simple Unsteady Flow

Unsteady Case with fixed airfoil shape and fixed flow condition

Methods : POD, POD-DNN, POD-GPR, POD-LSTM, FC-DNN, Conditional U-net

03 Case 2) Steady Flow in Variable Flow Conditions

Steady Cases with fixed airfoil shape and variable flow conditions

Methods : POD, POD-DNN, POD-GPR, FC-DNN, Conditional U-net

04 Case 3) Steady Flow in Variable Airfoil Shapes and Variable Flow Conditions

Steady Cases with variable airfoil shape and variable flow condition

U-net, Conditional U-net

05 Appendix

Chapter 1 - Introduction

I. Introduction

Calculation speed and accuracy are lacking in reality to apply in a level of Digital Twin in actual calculation even though numerical analysis precision of Computational Fluid Dynamics(CFD) which requires high precision and computing cost is increasedHowever, it is now an era of Bigdata and AI by development of great strides of Artificial Intelligence(AI) infrastructure based on hardware and software since 2010 as provided in Figure 1-1. Various applied research has been greatly continuing which is based on machine learning and deep learning, a key element technology of AI, in areas across society, industry, defense and living under this massive trend of technology. Also there has been many under evaluation regarding usage of AI infrastructure to overcome technological limit in CFD as computer simulation widely used in CFD.

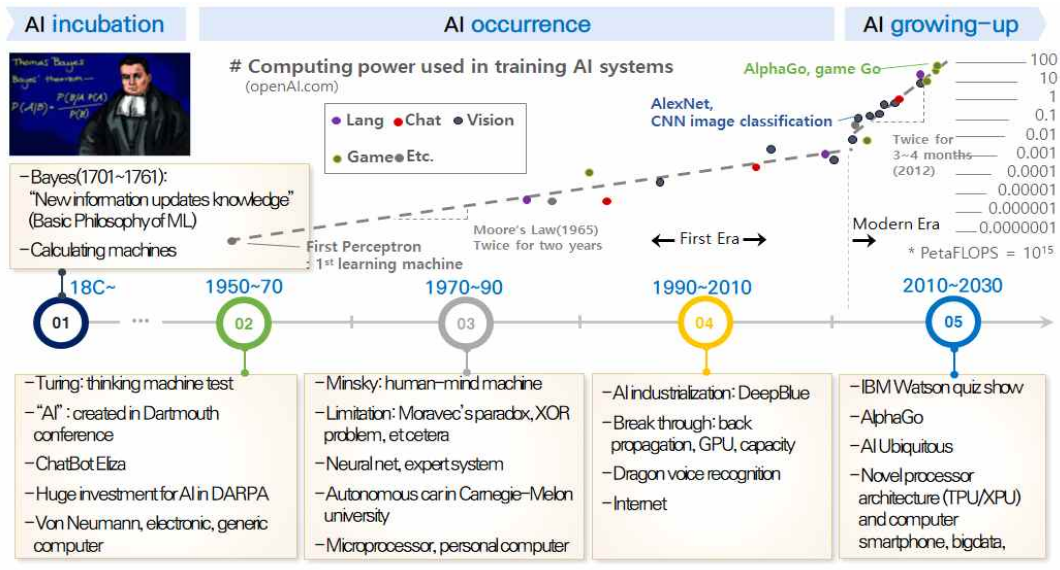


Figure 1-1. History of AI

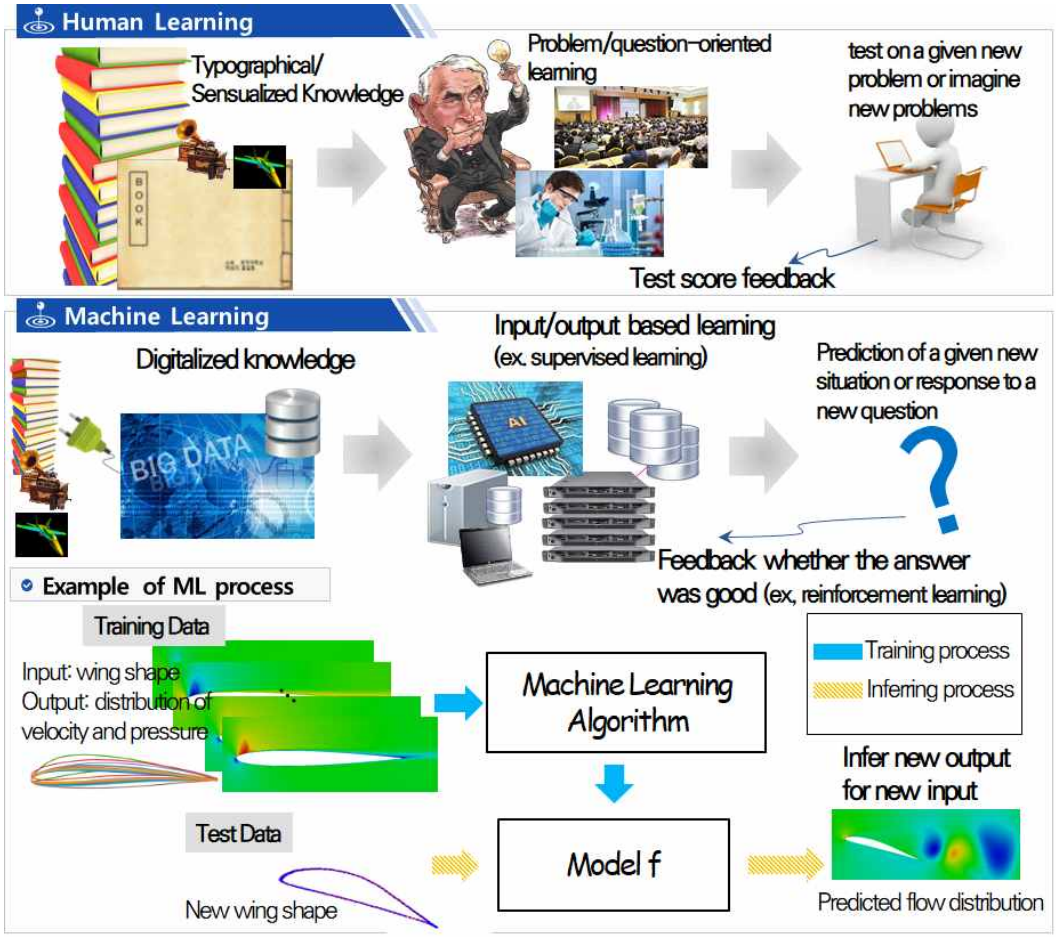


Figure 1-2. Concept of Machine Learning

Machine learning is a concept that is understandable for scholars who have accumulated academic experiences. To describe it shortly machine learning is a finding process of answers under all kinds of conditions. A tool that is driven from this process is capability humankind has been desired for very long period of time. Computer simulation is also a recently developed tool for prediction. There is basically similarity between human and machine learning meanwhile Figure 2-1 provides differences in a learning process of human and machine. Higher data reliance and digitization adapted for learning can be found in machine learning.

It is more readily to understand from an example of machine learning based on airfoil simulation data below Figure 1-2.

Deep learning, a subset of machine learning is recognized distinctively as a separate term as it leads to advancing enhancement of AI infrastructure since 2010. In addition another reason is due to neural network's configuration flexibility

resulting in explosive advent of massive number of detailed technologies. Machine learning and deep learning are basically under common purpose; classification, regression, optimization, control, dimensionality reduction, generative modeling, similarity determination. One reminder is that two key factors such as regression and dimensionality reduction are utilized mainly among AI algorithms features. Reduced order model(ROM) is the main task of finding a a reduced prediction model overall through using AI algorithms to solve challenging matters that demands time and computational resources. Proper Orthorgonal Decomposition(POD), one of reduced modeling originated from traditional machine learning techniques has been conducted in engineering simulation already. Technologies in areas of image and vision are grafted well into engineering simulation used for visualization. Convolutional Neural Network(CNN) which directs to booming of deep learning is also widely applied in areas of image and vision. Not only that, CFD has been increasingly shown many cases of using CNN. Therefore, the Reduced Order Modeling is spotlighted well for its accelerating prediction. Recently existing POD techniques are under revaluation in situation which seeking a method of applying deep learning for dimensionality reduction as processor hardware such as Graphics Processing Unit(GPU) is deveoped.

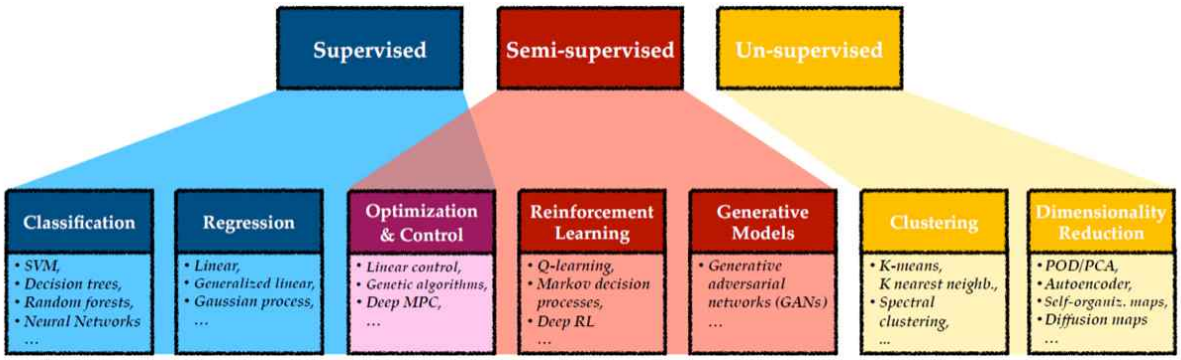


Figure 1-3. Features of Machine Learning·Deep Learning

[Courtesy of] 2020, Brunton, S.L., Noack, B.R., and Koumoutsakos, P., "Machine Learning for Fluid Mechanics," Annual Reviews of Fluid Mechanics, Vol. 52, pp. 477-508.

This technical report is made based on Hands-On Tutorial method targeting CFD researchers who are not familiar using AI to rapidly absorb to learn multiple AI techniques. In addition to introduction notifying beforehand a range of "AI

techniques" can differ depending on AI researchers, we comprehensively use a term "AI techniques" used as following such as machine learning pattern recognition in existing computing science, universal neural networks with few or many number of perceptrons, algorithms in basis of CNN/RNN that leads to 2012 deep learning booming, reward-based reinforcement learning that is used widely in automation control by applied from AlphaGo. In particular, Proper Orthogonal Decomposition(POD) widely used in CFD can be described as a type of machine learning techniques due to the fact that it is derived from Principal Component Analysis(PCA) commonly used for image compression. Followed by complexity of problems is increasing by moving to each chapter, there are several AI techniques stated to solve each problems in each chapter. Leading to absorbing problems and methods systematically by following in order, it is comprised to be available to try practices of any problems of desired complexity. Specifically, chapter 1 is introduction and chapter 2 is background theory. In chapter 3, the analysis part of the simplest problems to understand is fixed, and analysis dataset of changing only one parameter(time) is stated. In other words, it contains making a reduced model based unsteady/fixed shape flow analysis dataset and comparing inferred results using the reduced model with original numerical analysis data. In following chapter 4, compared to one input parameter of a reduced model in chapter 3, learning and reasoning under three flow condition [Reynolds number, Mach number, and Angle of Attack] in problem dataset of flow anlysis are specified. In chapter 5, shapes are input parameter in more complex practice problems. With a reduced model problems which include learning several input parameters mentioned in chapter 3 and 4, chapter 6 will demonstrate problems regarding shapes.

II. Understanding Key Terms

In introduction, basic terms that are used in technological concepts of AI like machine learning and Information technology(IT) are the ones that has to be get adjusted to. Both terms which are essential for machine learning procedures and technological terms that are considered to be necessary for professionals and students in field of numerical analysis simulation are summarized in this chapter. (more of regressions terms than classification are specified)

A. 3 Steps of Machine Learning: Modeling, Evaluation, Optimization

Figure 1-4 illustrates common machine learning process. In order for acquired data to be used for machine learning algorithm, verification and preprocessing have be to conducted firstly. Three steps below go by steps of learning to derive a reduced model.

Step 1) Modeling: First comes deciding a form of prediction machine. Then defining through which process to predict and classify should be selected. Deterministic, statistic method or others are options to choose.

Step 2) Evaluation: Next step is instructing the machine about direction to learn. For example, the higher score spam mail clssifier gets as the spam mail classfier does classification better. Depending on a result of a model, 'loss' is marked.

Step 3: Optimization: Based on the direction given to the machine what to learn, tuning follows in that direction. The most suitable model is minimizing loss.

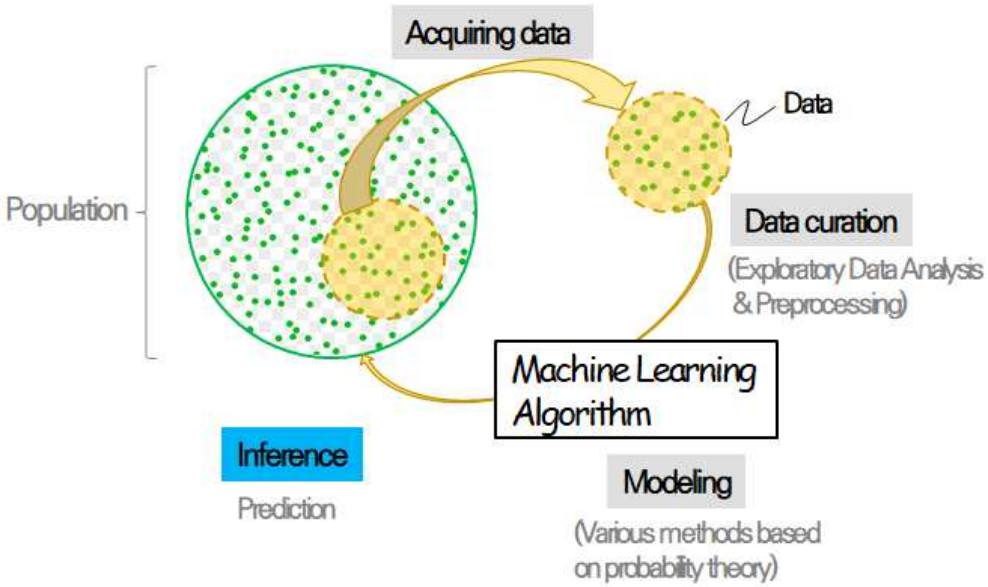


Figure 1-4. Process of Machine Learning

B. List of Key Terms

Accuracy - probability of making a right prediction of predicting model drawn from learning

Application Programming Interface (API)- A series of command, function, protocol, and objects that is usable for programmers to make software or interact with external system.

Artificial General Intlligence (AGI) - A computational system that conducts all intellectual tasks that human can, called as "strong AI". Currently, AGI yet has not reached to the point.

Artificial Intellgence (or Weak AI) - A computational system that simulates partial human intelligence but focuses on narrow tasks. Unlike AGI, it's called "weak AI".

Artificial Neural Network (ANN) - Machine learning method inspired by human central nervous system, especially neural network configurations of human brain. It is comprised of network in a unit of nuron or percpetron.

Convolutional Neural Network (CNN) - Deep artificial neural networks that are mainly used for classifying mostly images(e.g. match animal name with photo), clustering based on similarity(photo search), conducting object recognition within scenes(perceiving intervened human when autonomous driving).

Central Processing Unit (CPU) - The core electronic circuitry conducting corresponding computer program's command by performing computation of coded basic arithmetic, logical, control, and input/output(I/O).

Deep Learning - One type of machine learning based on neural network that gradually extracts a higher level of properties which started by learning data applying multiple layers of processing method.

Deep Neural Network (DNN) - A type of artificial neural network which is a term used to highlight usage of many layers in network(standard is none).

Evaluation - Evaluation of predictive model's accuracy, quality or others. It is a model drawn from learning.

Feature Extraction

1) When learning images with various features can show following features as

example:

- Lines, edges, and ridges
- Localized interest points such as corners, stains, dots
- Additional complex features such as repetitive shapes, sepcific shape, or motion

2) Process trasnforming data which is too large to use for purpose of interest into reduced representation set such as pattern, shape, lines, and edges.

Connected Neural Network (FCDNN) – A method with strength of having flexibility such as deep neural network which can be used as of general-purpose. In other hands, a method with shortcomings such as unability to classify well of image shapes just as CNN.

Generative Adversarial Networks (GAN)-An unsupervised machine learning algorithm in a basis of game theory, which is implemented by two contesting deep neural network to each other. It is used in many researches because of its strength of recommending and creating things such as images that look authentic to observers or objects that are yet constructed.

Graphics Processing Unit (GPU) – An electronic circuit designed to manipulate and change fastly memory for accelerating image production from frame buffer in output of display device. It is attached to embedded systems, mobile phones, personal computers, workstations, game consoles. It is widely used as deep learning's computational resource which needs many computation and for purpose of graphic processing.

Hyperparameter - A value set directly by users in process of modeling to improve performance of machine learning model. It has to be optimized for reducing model's error and enhancing learning capacity.

1) Representative Hyperparameter: in case of neural network machine

Types	Explanation	Application Consideration
Learning Rate	Variable deciding how fast to move in direction of gradient	– If too small, learning speed is slow and if too big, no learning
Cost Function	Function calculating difference between expected and actual values depending on input	–Mean squared error (in general) – cross entropy error(image)
Regularization parameter	L1 or L2 normalization method to avoid overfitting problem	– Classification of generalized variable that is currently used into hyperparameter
Mini–batch Size	Size of equally dividing entire learning data for conducting batch set	–Consideration of usable memory size and epoch capability
Training Loop	Variable determining early termination of learning	– Properly determining time point of decrease in learning efficiency
Hidden Unit	Learning optimization determinants for learning data	– effective when a number of neurons of first hidden layer is bigger than input layer
Weight Initialization	Determinants for learning capacity	– If all initial value is 9 then all neurons comes out to be same

2) Hyperparmeters has no decided optimal value and can try diverse optimizing trials

Image Recognition - An ability to distinguish objects, places, people, writing, and image motion.

Image Segmentation - A process of dividing digital images into several segments. Also it means an ability to simplify or alter image into easily analyzable and bestow name.

Inference - Prediction task applying to untrained cases using learning model.

Input – All forms of data that goes into learning (text, image, audio, music notation, all signals, digital data coming from science).

Labeling - Inputting various information according to a purpose to learn machine learning model by using data processing tool into data such as images, videos, texts

Layer - It refers to each column in artificial neural network.

Long Short-Term Memory (LSTM) - It is a method saving information of certain period of time by processing short-term state and long-term state to compensate shortcomings such as bigger RNN network is, more occurings of vanishing gradient problems.

Machine Learning (ML) - It is an algorithm that can predict and make a decision using new data based on pattern resulted after a process of learning pattern from previous data.

Model - A processing block or black box function returns predicted output after receiving input. Also a function built by hyperparameter optimization through type selection and learning in artificial intelligence.

Optimization - In areas of artificial intelligence, it is parameter optimization defined in modeling step in which machine has upmost capacity.

Output - A prediction value resulted after upload to learning model or after feded input is reprocessed in learning model.

Overfitting - It refers to situation which constructed machine learning algorithm cannot distinguish information related to assigned task within learning data from information that is irrelevant. In ovefitting, algorithm's predictive performance becomes low when dealing with new data.

Parameter - It refers to intrinsic(not manipulated) property value which is usable as criterion to define or classify system, function, and data. In AI, data predicted from algorithm is targeted. For example, weights, bias of neural network, mean and variance of distribution data.

Predictive Model - A model measuring probability different sample or population's remainder of showing same behavior or same results by using value of interest that data sample has.

Proper Orthogonal Decomposition (POD) - Using this method can decide the optimal mode set that is representative of specific data based on L2 norm. Varying on professional area, various name exists for example, principal component

analysis(PCA).

Python – Interpreted(no need of compile) high-level language for general-purpose programming.

Recurrent Neural Network (RNN)– An artificial neural network that has continous loop that recorded information is transferred from one step to next step of a network. It can be seen as a repetition of same network which each information is transferred to the next network.

Regression - A statistical measure used to determine relations between dependent variables and independent variables.

Reinforcement Learning – A type of machine learning that learning machine receives training to accomplish target function through an experimental process which receiving certain reward(score is given in good results). This method does not require status checking or labeling about every individual tasks that algorithm will conduct so that it's fundamentally differnet from general supervised learning method.

Singular Value Decomposition – As of one method of matrix factorization in linear algebra, it is to generalize by deciding eigenvalue which deals with square matrix and eigen decomposition which uses eigenvector to be rectangular matrix

Snapshot – A single vector made by vectorization of scalar field distributed in space. In POD, these vetors conducts SVD in a basis of matrix which is attached according to each time step(or specific parameter).

Supervised Learning:

1) A machine learning method which requires continuous learning process of status checking after human enters input. In supervised learning, there is definite results of machine's data mining. It's target function is defined as reaching this results.

2) It is a machine learning algorithm learning patterns of resulting data. Supervised learning algorithm predicts based on a series of exmaples.

Target Function - The final goal of an algorithm.

TesorFlow - It is open-source software framework(a bundle of library) that is used for machine learning application such as neural network. Started by used in Google research and production, it was released as the Apache 2.0 open source

license.

Test Dataset - In machine learning, it refers to data offered to machine after training and validation step is completed. This test dataset means algorithm's performance(accuracy, sensitivity, etc) made after completion of first two phase(training and validation) when unknown data is marked.

Torch - A scientific computing framework made to widely support machine learning algorithm which is written in C and lua language. Currently it is upgraded and used in Facebook AI Research and Twitter, and if compared often to TensorFlow, it's used general-purpose.

Training Dataset - In machine learning, it refers to training dataset offered to machine in initial "learning" or "training" phase. Through this dataset, machine can obtain black box funtion with insight to complete efficiently assigned mission by distinguishing relations and features between data.

Unsupervised Learning - Machine learning algorithm learning data patterns in situation in which no result value presents. In here, machine can be used to find instrinsic pattern from data or draw own conclusion after offering completely unlabelled data.

Validation Dataset - A data sample used for offering intermediate resulting values for evaluating model fit regarding training dataset while modulating hyperparameter of learning model.

Crawling - It refers to internet bot that browses systematically World Wide Web for web indexing in general. More than web engine indexing of related page, it efficiently processes and searches.

C. Comparison Table of Mixed Words

In addition to the list above, inferential statistics which is academic foundation for artificial intelligence and currently used machine learning terms are compared in below table 1. Both of the two fields mentioned earlier are not accustomed for researchers in field of numerical simulation which is composed of industrial mathematics terms.

Table 1. Comparison of Mixed Words

Machine Learning	Inferential Statistics
Feature	Variable
Output (Label)	Dependent Variable
Input	Independent Variable
Learning	Estimation
Supervised Learning	Regressionn
	Classification
Unsupervised Learning	Clustering
Data	Population
Training Data	Sample

III. Quick Start: Building Curve Fitting Model using ANN

In this section, basic concepts of machine learning will be learned through hands-on practice comprised of learning and inference (prediction). In particular, the practice in this section will use the artificial neural network (ANN) among various machine learning methods, and is directly related to deep learning techniques.

A. Training process

Machine learning process depicted in Figure 1-4 is implemented for deriving a prediction model for $(\hat{x}, \hat{y})$ from given point dataset (x, y). In other words, it is about curve fitting modeling of given point dataset in two-dimensional space.

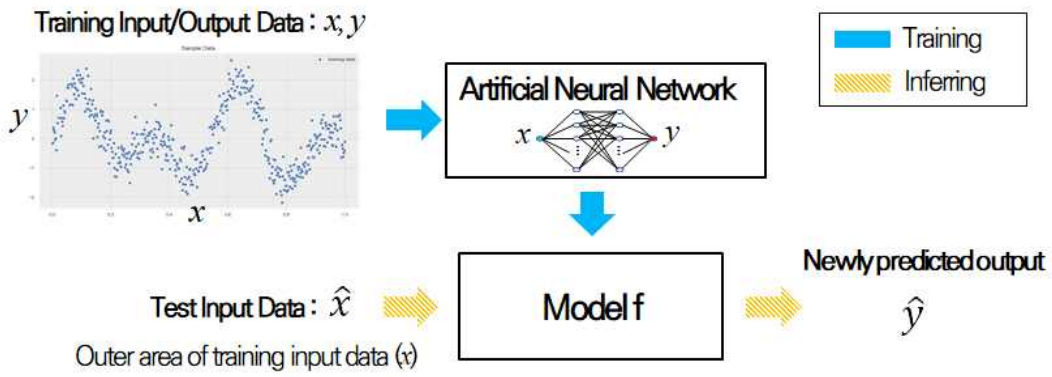


Figure 1-4. Establishment of Machine Learning Modeling

B. Dataset Preparation

$\sin(4\pi x)+\sin(7\pi x)$ is set to be the basis function for training data, and 500 points in range of $0<x<1$ are created based on random number error generation. The source code for data sampling is as follows.

데이터 준비 Data preparation

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Number of samples and input variable x (training set).
n = 500
x = np.linspace(start=0, stop=1, num=n)

# x, y는 학습 데이터
def f(x):
"""The base function"""
    f = np.sin((4*np.pi)*x) +np.sin((7*np.pi)*x)
    return(f)

# Error standard deviation.
sigma_n =0.4

# Errors.
epsilon =np.random.normal(loc=0, scale=sigma_n, size=n)

# Observed target variable.
f_x = f(x)
y = f_x + epsilon

# Plotting with seaborn library
fig, ax = plt.subplots()

# Plot training data.
sns.scatterplot(x=x, y=y, label='training data', ax=ax)

# Plot "true" linear fit.
sns.lineplot(x=x, y=f_x, color='red', label='f(x)', ax=ax)
ax.set(title='Sample Data')
ax.legend(loc='upper right')
```

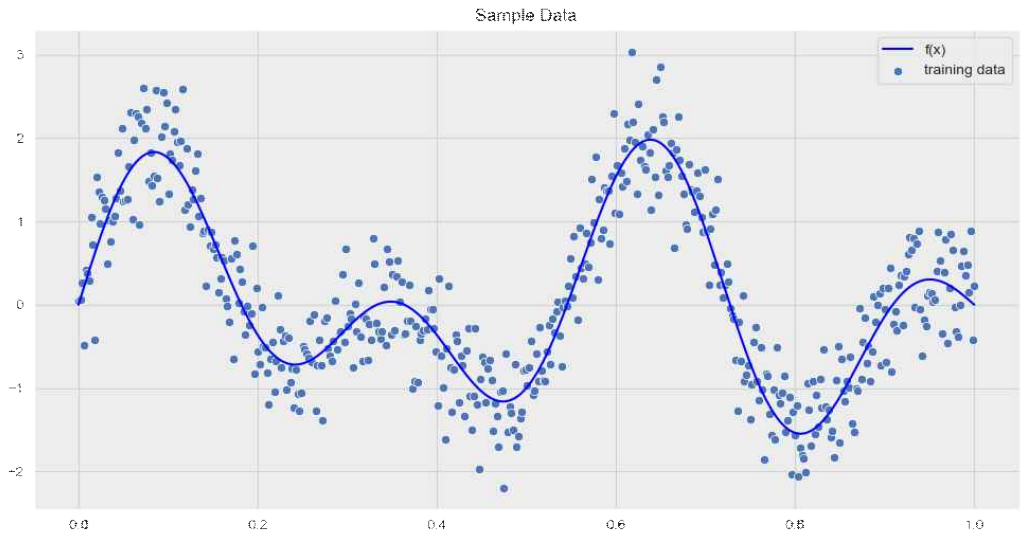


Figure 1-5. Preparation of Arbitrary Training Dataset(x,y)

C. Network configuration and hyper-parameters

Composed of a network of neurons or perceptions as shown in Figure 1-6, artificial network provides universal approximator which can model any form of data to moderate level if hyper-parameters such as network structure, activation function, optimizer, learning rate, et cetera are successfully selected. In this quick start, the network is composed of four layers (an input layer + two hidden layers + an output layer). The number of layers and neurons in each hidden layer are determined arbitrarily, outnumbers a normal neural network, and selected less than that used for deep learning. Exponential linear unit (ELU) which guarantees the continuity of differentiation is selected as nonlinear activation function of a single neuron.

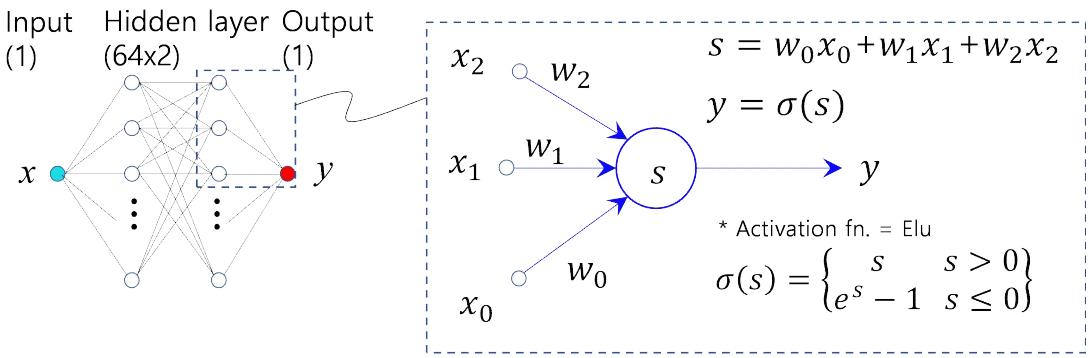


Figure 1-6. Single Neuron in Artificial Neural Network

Training Network Modeling (ANN)

```
import tensorflow astf
from tensorflow importkeras
from tensorflow.keras importlayers

# Create the model
model = keras.Sequential()
model.add(keras.layers.Dense(units = 1, activation ='linear', input_shape=[1]))
model.add(keras.layers.Dense(units = 64, activation ='elu'))
model.add(keras.layers.Dense(units = 64, activation ='elu'))
model.add(keras.layers.Dense(units = 1, activation ='linear'))
model.compile(loss='mse', optimizer="adam")

# Display the model
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 1)                 2
 dense_9 (Dense)             (None, 64)                128
 dense_10 (Dense)            (None, 64)                4160
 dense_11 (Dense)            (None, 1)                 65
=================================================================
Total params: 4,355
Trainable params: 4,355
Non-trainable params: 0
_________________________________________________________________
```

D. Optimization of Prediction Model Parameters

Once artificial neural network model is built, 5,000 iterations for parameter (not hyper-parameters) optimization during training process. Below the result shows that a model with MSE (mean-squre-error) 0.146 was derived.

Training

```
# Training
model.fit(x, y, epochs=5000, verbose=1)
```

```
Epoch 4993/5000
16/16 [==============================] - 0s 4ms/step - loss: 0.1508
Epoch 4994/5000
16/16 [==============================] - 0s 4ms/step - loss: 0.1512
Epoch 4995/5000
16/16 [==============================] - 0s 5ms/step - loss: 0.1489
Epoch 4996/5000
16/16 [==============================] - 0s 4ms/step - loss: 0.1553
Epoch 4997/5000
16/16 [==============================] - 0s 4ms/step - loss: 0.1488
Epoch 4998/5000
16/16 [==============================] - 0s 4ms/step - loss: 0.1520
Epoch 4999/5000
16/16 [==============================] - 0s 4ms/step - loss: 0.1568
Epoch 5000/5000
16/16 [==============================] - 0s 4ms/step - loss: 0.1486
```

A: 0s - **loss: 0.146**

E. Inference Process

Figure 1-7 demonstrates original point data, base function, and two inference (prediction) models whose epoch numbers are different. One of the inference models was established after 5,000 epochs named 'pred5000', but the other model was done by 500 epochs named 'pred500'. One can find that the 'pre5000' model shows better performance than 'pre500' model does.

Inference and Plotting

```
# Compute the output
y_predict = model.predict(x)
print(x.shape, y_predict[:,0].shape, f_x.shape)
fig, ax =plt.subplots()
handles, labels =ax.get_legend_handles_labels()

# Plot training data.
A =sns.scatterplot(x=x, y=y, label='training data', ax=ax)

# Plot "true" linear fit.
B =sns.lineplot(x=x, y=f_x, color='green', label='base func.', ax=ax)

# Plot prediction.
C =sns.lineplot(x=x, y=y_predict[:,0], color='red', label='pred', ax=ax)
ax.set_title('Neural Net Prediction & True')
ax.legend(loc='upper right');
```

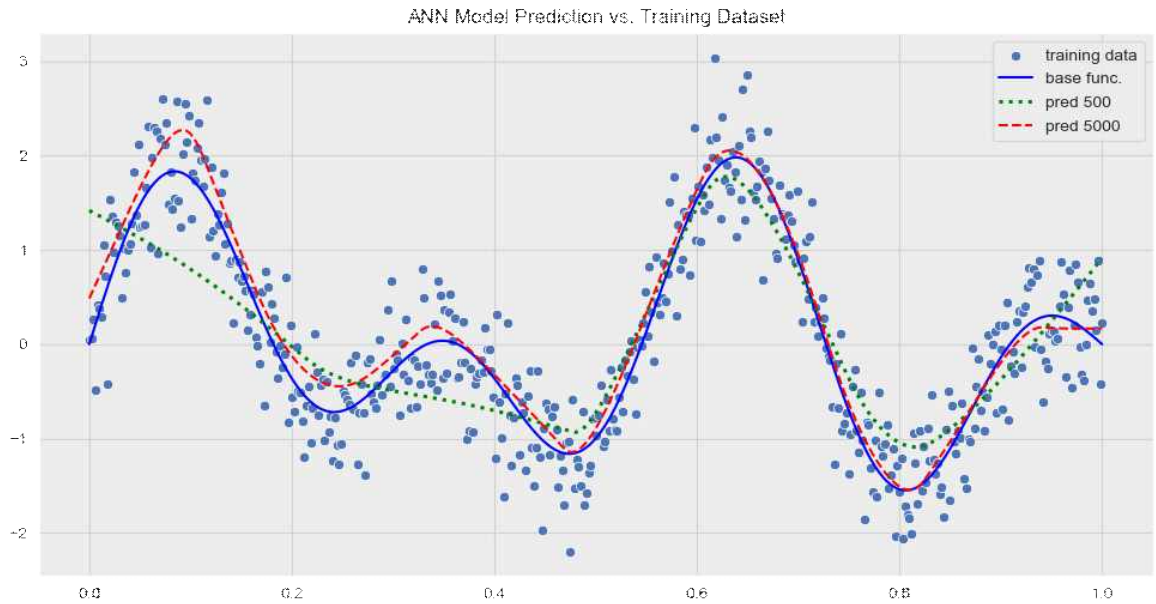


Figure 1-7. Drawn Results of Neural Network Learning Model

In the description so far, a process of establishing a prediction model for curve fitting through machine learning, especially artificial neural network method that is a foundation of deep learning algorithm, is well demonstrated. Further algorithms will be introduced in following chapters, but the basic scheme will be similar in this supervised learning cases.

IV. Repositories

As this technical report mainly focuses on hands-on practice, related dataset and source codes were made up into some repositories. The comprising flow of datasets and codes is depicted in Figure 1-8.

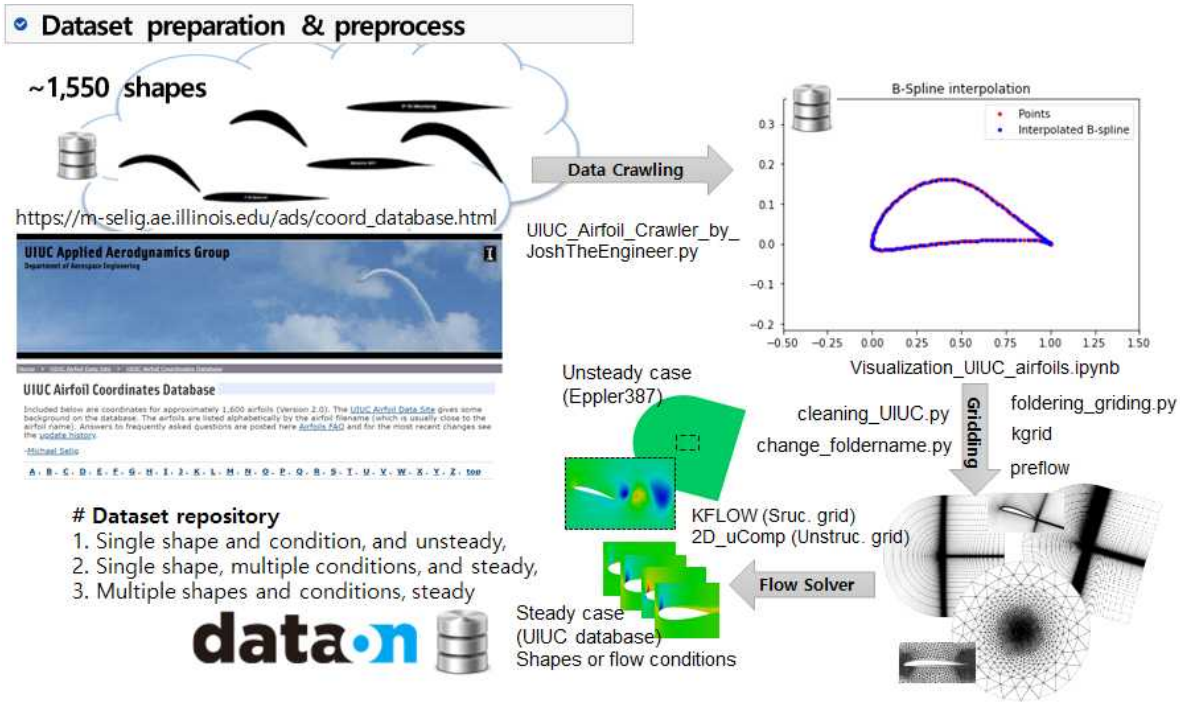


Figure 1-8. Preparation of Dataset and Preprocessing Process

A. Dataset repository

All data used in this report were uploaded under the following dataset names in DataON (https://www.dataon.re.kr/ which is a research data repository.

- Repository: DataON (https://www.dataon.re.kr/)
- Dataset: 1. UIUC Airfoil Shape Data, Grid, Aerodynamic Simulation Dataset
  2. NACA0012 CFD dataset in various flow conditions

There are several preprocessed datasets made based on 1,550 shapes crawled on

the airfoil shape data site created at the University of Illinois in Urbana-Champine (UIUC). The list is as follows:

1) Dotted airfoil shape data (chord length = 1)
2) Grid of plot3d format regarding each shape
3) CFD using structured grid unsteady RANS (40 time steps) of fixed shape (Eppler387), Mach number 0.6, angle of attack 16°, Reynolds number $1x10^5$, Sparart-Allmaras model (Chapter 2)
4) Steady CFD data under 135 flow conditions in range of Mach number 0.3 to 0.5, Reynolds number $1x10^5$, angle of attack (AOA) 0 to 7 degrees for single shape (NACA0012) (Chapter 3)
5) 26 steady CFD cases in range of Mach number 0.4, angle of attach 1° to 16° for 1,550 individual airfoil shapes (Chapter 4)
6) All analysis results are made using Tecplot format

The dataset could be updated in future if it is necessary

B. Code repository

The sample codes used in this hands-on report were uploaded in an open source code repository, GitHub.

- Repository: GitHub (https://github.com/)
- Code set: https://github.com/Jameshin/AI-CFD-Technical-Report

Detailed list of ML codes and tools including crawling webtsite information and calculating coefficient of drag are written below.

1) ./UIUC_Airfoil_Dataset_Handler/UIUC_Airfoil_Crawler_by_JoshTheEngineer.py

: Crawling of shape information of UIUC airfoil datasite (refer to Josh The Engineer website)

2) ./UIUC_Airfoil_Dataset_Handler/Visualization_UIUC_airfoils.ipynb
: Visualization of point data shape (jupyter notebook format)

3) ./UIUC_Airfoil_Dataset_Handler/ExtracData_###.py
: Preprocessing task to change into data usable for machine learning

4) ./UIUC_Airfoil_Dataset_Handler/ folder's extra files
: Modification of structure and name of dataset folder, etc

5) ./Unsteady/POD-###/POD_ExtracData_POD_Unsteady.py
: Preprocessing task to change into data usable for machine learning

6) ./Unsteady/POD-###/POD_SVD_Unsteady.py
: Extraction and analysis of POD regarding unsteady flow

7) ./Unsteady/POD-###/POD_Evaluator_###_Unsteady.py
: Prediction of POD mode coefficient using machine learning method and flow-field reconstruction

8) ./Unsteady/POD-###/CFDLib.py
: Data error calculation predicted by using FOM CFD data and ROM

9) ./Unsteady/POD-LSTM/Generate_windowed_dataset.py
: Preprocessing time series data into window data for LSTM learning

10) ./Unsteady/POD-LSTM/lstm_encoder_decoder.py
: LSTM model construction

11) ./Unsteady/FCDNN/FCDNN_Unsteady.py
: Fully-connected DNN method

12) ./Unsteady/FCDNN/PODNN_Unsteady.py
: Fully-connected DNN + POD method

13) ./Unsteady/Conditional Unet/ExtracData_Unsteady_cuttail.py
: Transformation to data usable for machine learning and deletion of far-field area

14) ./Unsteady/Conditional Unet/Unet_UIUC_Unsteady.py
: Flow-field prediction method based on CNN-Autoencoder (time condition)

15) ./Steady FC/POD-GPR/ExtracData_POD_FC_cuttail.py
: Transformation to data usable for machine learning, deletion of far-field area, formation of flow condition label

16) ./Steady FC/POD-GPR/POD_Evaluator_GP_FC.py
: Prediction of POD mode coefficient under flow condition on the the basis of GPR method and flow-field reconstruction

17) ./Steady FC/Conditional Unet/ExtracData_FC_cuttail.py
: Transformation to data usable for machine learning, deletion of far-field area, formation of flow condition label

18) ./Steady FC/Conditional Unet/Unet_UIUC_FC.py
: Flow-field prediction method based on CNN-Autoencoder (flow condition)

19) ./Steady Shape FC/Conditional Unet/###.py
: Flow-field prediction method based on CNN-Autoencoder (shape+flow condition)

20) ./Aero_Performance/CLCD_FOM.py
: Extraction of coefficient of drag(Lift and Drag) from CFD data

21) ./Aero_Performance/CLCD_ROM.py
: Extraction of coefficient of drag(Lift and Drag) from inference data

Python, an easy and common script language, is a language should be learned to bring each framework functionAPI as diverse range of AI framework (Tensorflow 1.x, Tensorflow 2.x, PyTorch) is applied in each code. It can be learned through each framework manual and basic tutorial, and compared to high-level coding language can be learned within much faster time as long as theoretical procedure of each method is understood.

Current Codes could be updated if necessary.

V. Practice Cases of This Book

Main contexts of the report is hands-on practice about CFD cases of airfoils. Below are targeted CFD Cases.

A. (Case 1) Unsteady CFD + Single Shape + Fixed Flow Conditions

First case is to learn CFD dataset in which airfoil shape (or grid) and flow conditions are fixed and whose outputs change only depending on time. In other words, this case is about comprising a learned model of single factor, t. Chapter 2 refers to shape, flow condition, time step, et cetera in detail. It should be noted that this chapter contains essential contents generally understanding applying machine learning in area of CFD since most variety of methods are applied to this simplest case.

B. (Case 2) Steady CFD + Single Shape + Variable Flow Conditions

In Case 2, several ML/DL algorithms were used for the CFD dataset whose labeling factors are three flow conditions (Reynolds number, Mach number, angle of attack) with steady state (fixed time) whereas the CFD dataset was based only on time factor in Case 1. In other words, flow field under arbitrary flow condition in given airfoil shape can be predicted through the learned ROM model. Further details can be found in Chapter 3.

C. (Case 3) Steady CFD + Variable Shapes + Variable Flow Conditions

The last step of this report is the case which usage of deep learning method on a basis of dataset in which airfoil shape is arbitrarily dotted and a single flow condition (angle of attack) is variable as well. More complexity it is, more limit in selecting a AI method. In Case 3, CNN-based network (named U-net) was applied since it is most tractable of current shape sensitivity due to the recently proved convolution performance. Detailed information can be referred Chapter 4.

Based on 3 cases that was explained above, it is thought that readers to deal with fundamental AI methods to CFD simulation data can get ready regardless of the fact that there are other advanced practices.

Chapter 2 - Case 1) Simple Unsteady Flow

Unsteady Cases with Fixed Airfoil Shape and Fixed Flow Condition
AI methods : POD, POD-GPR, PODNN, POD-LSTM, FCDNN, Conditional U-Net

I. Learning Data Formation

In chapter 2, full order model(FOM) CFD analysis is conducted for formation of a reduced model(ROM) which is for unsteady flow analysis. For learning data, unsteady analysis result of shedding generated by airfoil in angle of attack 16° is used. For airfoil, Eppler 387 is used, and the analysis is analyzed under condition which Mach number 0.6, angle of attack 16°. Below Figure 2-1 shows Eppler 387 grid that is usedd in analysis of unsteady CFD.

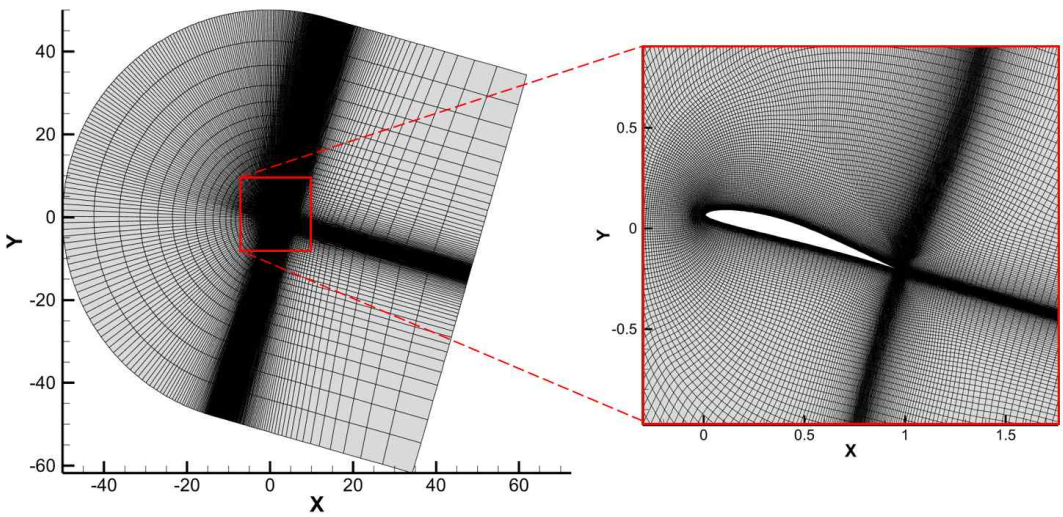


Figure 2-1. Eppler 387 Grid used for Dataset Formation

Table 1 below summarizes CFD analysis condition for forming learning data.

Table 2. CFD Analysis Condition

Solver	Airfoil	Grid	Flow condition
KFLOW	Eppler 387	689x145 (C-type)	Mach 0.6 AoA 16°

Governing equation	Flux function	Time integration	Nondimensional Time step ($=tU_\infty/L$)
RANS S-A model	Roe flux 3rd order	DADI	0.1

Based on previously explained conditions, analysis interprets vortex shedding occuring at airfoil at angle of attack 16° through unsteady CFD anlysis, and by that translates 0 to 18 dimenionless time within 0.1 time interval. From analysis, calculation results from 10.1 to 18 are used as data set. Performance of a reduced model is evaluated by predicting two flow fields: flow field within total of 20 time such as 10.1,10.5,10.9... 17.7 and within total of 20 time such as 10.3, 10.7,11.1, ...17.9.

II. POD-ML Reduced Model

In chapter 2 demonstrates that linear combination of POD basis vector and mode coefficient can lead to possibility of dimensionality reduction and reconstruction within flow field by using POD method. Therefore, this page will introduce API tutorial using reduced order modeling in unsteady flow with applied POD method. For predicting POD mode coefficient, two types of representative interpolation method(local linear interpolation, Gaussian process regression) and long short-term memory(LSTM) which is one of AI model for time series learning are applied.

A. Local linear interpolation

In order for predicting mode coefficient, the most simple way is applying linear

interpolation between two time period. Just as Figure 2-2, connecting points(circular point) extracted from exact solution into a straight line can predict solution between points on the line. By following this rule, mode coefficient at unknown time t^* can be found, and reconstructing to original dimension to predict field flow is capable.

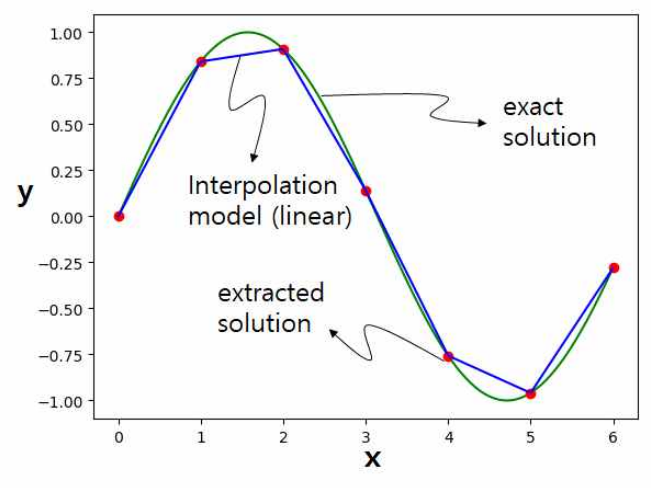


Figure 2-2. Local Linear Interpolation

1) Preprocessing and Extraction of Snapshot Data (*ExtracData_POD_Unsteady.py*)

Snap shot data calculated by CFD before applying POD method has to be transformed to the form suitable for POD. Just as the example code below, environment and variables for data processing and extraction are set to start. noCol is a number of flow variable included in snap shot data, and numd is a number of snap shot data used in machine learning. For example, inc_time of below code means that among 80 data comprised of time step in range of 201 to 280, only 20 are to learn in a time interval of 4.

```
import numpy as np
import pandas as pd
noCol = 11      # number of flow field variables
numd = 20       # number of snapshots
initial_time = 101
inc_time = 4    # time interval of training data
zone1_i = 689
zone1_j = 145
sim_data_path = "../../../Database/sol01RANS3.8208da/"
Tecplot_header_in = "variables=X, Y, Z, Rho, U, V, W, P, T, Vor, Qcri"
Tecplot_header_out = "variables=X, Y, Rho, U, V, P"
```

Below code is preprocessing process after snapshot data file which is calculated by CFD is loaded. In total of 11 flow variables, only four flow variables which are density(q), speed(U,V), pressure(P) are arranged as one column and stored in array. About the given unsteady flow, grid form is consistent that xy is stored separately and flow field data is stored as snapshot.

```
## stack snapshots
snapshot_data = np.array([])
POD = np.array([])
for i in range(0,numd):
    # read data
    pd_data = pd.read_csv(sim_data_path+'flo001.0000'+str(initial_time+i*inc_time).
              rjust(3,'0')+'uns', dtype='float64', delimiter=' ',
              skipinitialspace=True,skiprows=2, header=None)
    data = np.nan_to_num(pd_data.values)
    array_data = data.flatten()
    array_data = array_data.reshape(-1,noCol)
    if i==0:
       snapshot_data = array_data
        xy = snapshot_data[:,0:2]
        N = snapshot_data.shape[0]
        POD = array_data[:,[3,4,5,7]].flatten()[:,None]
    else:
        snapshot_pod = array_data[:,[3,4,5,7]].flatten()[:,None]
        POD = np.hstack((POD,snapshot_pod))
## export snapshots
np.savez('../Results/PODarray.npz', xy=xy, snapshot=POD)
```

2) Extraction of POD basis vector and mode coefficient (*POD_SVD_Unsteady.py*)

Preprocessed snapshot data from previous page is retrieved, calculate mean averaged flow. and subtract average value from all snapshot data for data normalization. After that, singular value decomposition (SVD) is conducted using numpy library. For resulting vectors, singular values is rearranged in order of big numbers. Since singular values and singular vector in preceding order contain orginal data's most of information that only using this values and vector lead to substantial rate of reconstruction when reconstructed to original flow field. In POD, left singular vector is used as basis vector, and regularized snapshot matrix is divided by basis vector to obtain mode coefficient.

```
#mean center the data
mean_array = None
mean_data_tensor = np.mean(snapshot_data, axis=1)
```

```
print(mean_data_tensor)
mean_centered_data = np.subtract(snapshot_data, np.tile(mean_data_tensor, (numd,1)).T)

#Singular Value Decomposition
u, s, v = np.linalg.svd(mean_centered_data, compute_uv=True, full_matrices=False)

# POD coefficients
compute_coeffs = np.matmul(np.transpose(mean_centered_data),u)
e = np.sum(s)
s_energy = np.divide(s,e)*100
coeffs = compute_coeffs
mean_array = mean_data_tensor
```

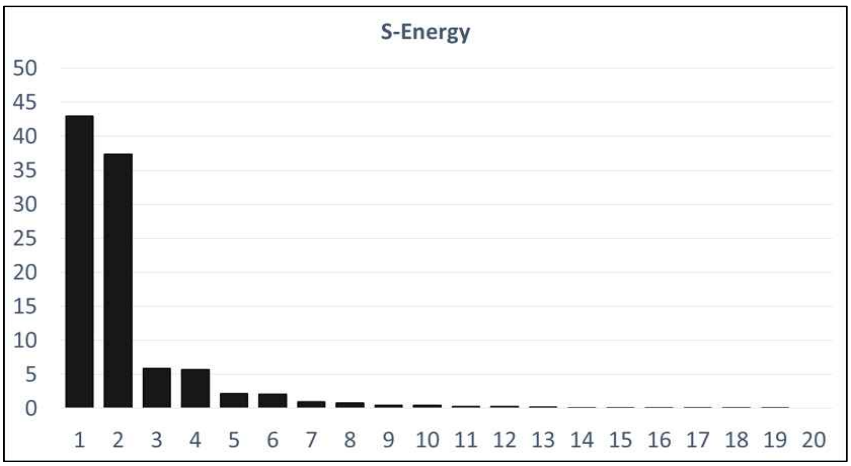


Figure 2-3. Comparison of S-energy Size Depending on Mode

3) Prediction of Mode Coefficient and Reconstruction of Flow Field (*POD_Evaluator_Unsteady.py*)

In infer_times variable, time not used for learning ,$t^*$is entered. Then just as the code below, mode coefficient for time t^* is predictable through linear interpolation. Predicted mode coefficient is multiplied to basis vector, and then added flow field average value leading to construction of snap shot data of time t^*. Reconstructed snap shot data is comparable to exact solution calculated by CFD. In below code, relative error with exact solution of speed and pressure prediction value is calculated.

```
# Use Tensorflow 1.x
for i, x in zip(range(total_steps), infer_times):
    print(i)
    ii= x-201
    hi_idx = [idx for idx,v in enumerate(train_times) if v > x][0]
    lo_idx = hi_idx - 1
    #interpolate coefficients
    interp_coeffs = coeffs[lo_idx]+(coeffs[hi_idx]-coeffs[lo_idx])*
                    (x-train_times[lo_idx])/(train_times[hi_idx]-train_times[lo_idx])
    int_coeff_tensor = tf.Variable(interp_coeffs)
    #add a dim to make it a 2-D tensor
```

```
int_coeff_tensor = tf.expand_dims(int_coeff_tensor, 0)
#compute the POD approximation
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    result_op = tf.matmul(int_coeff_tensor, tf.transpose(u) )
    modal_result = sess.run(tf.transpose(result_op))
    modal_result.flatten()
    mean_tensor.eval()
    result_op = tf.add_n([modal_result[:,0], mean_tensor])
    result = sess.run(result_op)
result = result.reshape((int(modal_result.shape[0]/noConcernVar), noConcernVar))
error_u = relative_error(result[:,1], UC[:,ii])
error_v = relative_error(result[:,2], VC[:,ii])
error_p = relative_error(result[:,3], PC[:,ii])
print('Error u: %e, v: %e, p: %e' % (error_u, error_v, error_p))
```

4) Example Results

The left of the Figure 2-4 is a contouring result of velocity field predicted at t=10.3, time that wasn't workable for learning. Right of the Figure is absolute error contour with FOM CFD results. Relative error is estimated to be 2.9%.

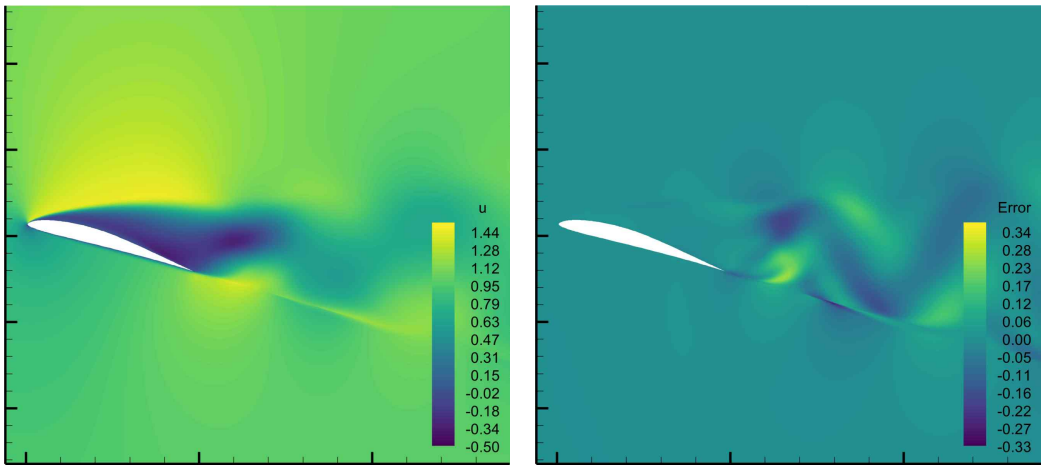


Figure 2-4. Velocity Field Result Using Linear Interpolation(Left) and Error(Right)

B. Gaussian process regression (GPR)

Gaussian process regression(GPR) or called as kriging is a interpolation techniques that is most commonly used in engineering. It can probalistically find function that is most representable of learning data. Also there is a good side such as it can more precisely describe nonlinear data than linear interpolation.

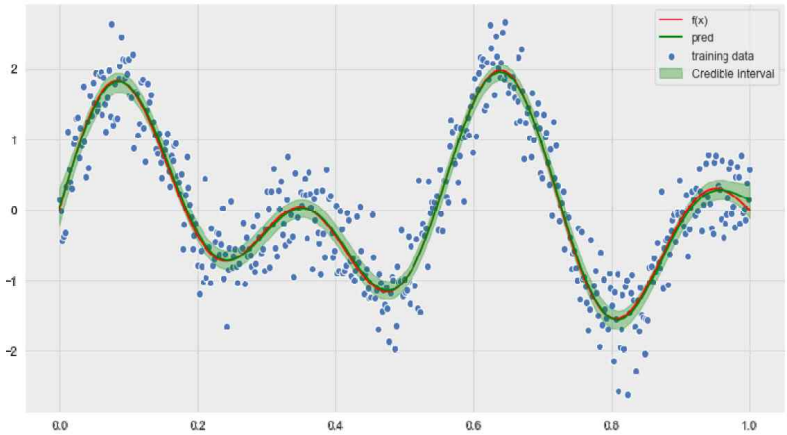


Figure 2-5. Example of Curve fitting Using Kriging

1) Prediction and Reconstruction of Mode Coefficient using GPR (*POD_Evaluator_GP_Unsteady.py*)

snapshot data preprocessing and POD decomposition procedure are same just as described in linear interpolation explained earlier. First GPR model has to be defined for predicting mode coefficient. In this report, correlation function is defined using squared exponential kernel function just as code below.

```
def kernel_function(x, y, sigma_f=1, l=1):
    """Define squared exponential kernel function."""
    kernel = sigma_f * np.exp(- (np.linalg.norm(x - y)**2) / (2 * l**2))
    return kernel
```

Data distribution diagram necessary for covariance calculation is calculated using correlation function as described in code below.

```
def compute_cov_matrices(x, x_star, sigma_f=1, l=1):
    """
    Compute components of the covariance matrix of the joint distribution.
    We follow the notation:
        - K = K(X, X)
        - K_star = K(X_*, X)
        - K_star2 = K(X_*, X_*)
    """
    n = x.shape[0]
    n_star = x_star.shape[0]
    K = [kernel_function(i, j, sigma_f=sigma_f, l=l) for (i, j) in itertools.product(x, x)]
    K = np.array(K).reshape(n, n)
    K_star2 = [kernel_function(i, j, sigma_f=sigma_f, l=l) for (i, j)
    in itertools.product(x_star, x_star)]
    K_star2 = np.array(K_star2).reshape(n_star, n_star)
    K_star = [kernel_function(i, j, sigma_f=sigma_f, l=l) for (i, j)
```

Consequently, both average function for learning data interpolation and covariance which is confidence interval are calculable as described in code below.

```
def compute_gpr_parameters(y, K, K_star2, K_star, sigma_n):
    """Compute gaussian regression parameters."""
    n = K.shape[0]
    # Mean.
    f_bar_star = np.dot(K_star, np.dot(np.linalg.inv(K + (sigma_n**2)*np.eye(n)),
                    y.reshape([n, d])))
    # Covariance.
    cov_f_star = K_star2 - np.dot(K_star, np.dot(np.linalg.inv(K + (sigma_n**2)
                    *np.eye(n)) K_star.T))
    return (f_bar_star, cov_f_star)
```

sigma_f, sigma_n and l are important hyperparameter that decide interpolation performance of GPR, and necessary to set up proper value. Data selectivity is calculated based on this value.

```
# Coefficients calculated by Gaussian Process
    sigma_f = 1
    sigma_n = 0.4
    l = 1
    n_star = train_times.shape[0]
    d = 1
    K, K_star2, K_star = compute_cov_matrices(train_times, train_times,
                                              sigma_f=sigma_f, l=l)
```

Based on code below, average function that interpolated learning data is calculated. Mode coefficient of time that wasn't able to learn is predictable by using this average function.

```
for i, x in zip(range(total_steps), infer_times):
        print(i)
        ii= x-201
        # Sample from prior distribution.
        coeffs_pri = np.random.multivariate_normal(mean=coeffs[i], cov=K)
        print("==============",coeffs_pri)
        # Compute posterior mean and covariance.
        f_bar_star, cov_f_star = compute_gpr_parameters(coeffs_pri, K, K_star2,
                                                        K_star, sigma_n)
        temp = np.zeros(n_star)
        print("=======", f_bar_star.squeeze().shape)
        coeffs_sam = np.random.multivariate_normal(mean=f_bar_star.squeeze(),
                                                   cov=cov_f_star, size=n_star)
        coeffs_pos = np.apply_over_axes(func=np.mean, a=coeffs_sam, axes=0).squeeze()
        coeffs_pos_sd = np.apply_over_axes(func=np.std, a=coeffs_sam, axes=0).squeeze()
        np.savez(res_data_path+"coeffs.npz", t=infer_times,
                 mean=coeffs_pos, sd=coeffs_pos_sd)
        int_coeff_tensor = tf.Variable(coeffs_pos)
```

Mode coefficient predicted using GPR is constructed using method described earlier.

```
        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
            result_op = tf.matmul(int_coeff_tensor,tf.transpose(u))
            modal_result = sess.run(tf.transpose(result_op))
            modal_result.flatten()
            sess.run(mean_tensor)
            result_op = tf.add( modal_result, mean_tensor)
            result = sess.run(result_op)
        result = result.reshape(-1, noConcernVar)
        error_u = relative_error(result[:,1], UC[:,ii])
        error_v = relative_error(result[:,2], VC[:,ii])
        error_p = relative_error(result[:,3], PC[:,ii])
        print('Error u: %e, v: %e, p: %e' % (error_u, error_v, error_p))
```

2) Example Results

In left of Figure 2-6, result of contouring velocity field predicted at $t = 10.3$, time which was not able to learn is demonstrated. On the right, there is absolute error contour from result of FOM CFD. Relative error is estimated to be 6.6%. It can be found that a degree of error is greatly high in number compared to prediction result using linear interpolation. In current code, main reason causing low accuracy seems to be due to the fact that hyperparameter optimization wasn't reachable lacking of Maximum likelihood estimation.

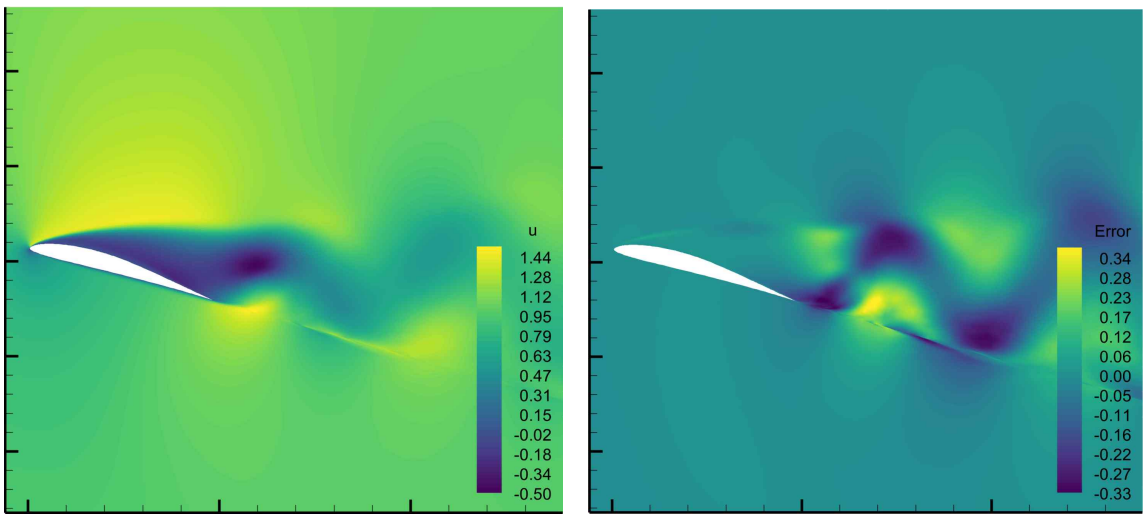


Figure 2-6. Velocity Field Prediction Result Using GPR(Left) and Error(Right)

C. Long Short Term Memory (LSTM)

Long short-term memory method which is a type of recurrent neural network is a reinforcement learning method that is widely used in time series analysis. This method can solve problem such as decline in accuracy in condition of long dependency which existing RNN has had problem with by process of updating cell state using 4 types of layers.

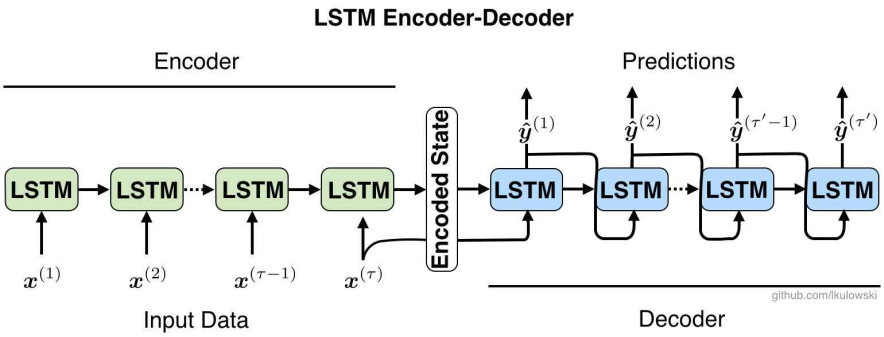


Figure 2-7. Structure of LSTM encoder decoder

1) Data Segmentation of Mode Coefficient for LSTM Learning (*Generate_windowed_dataset.py*)

Extraction of POD mode coefficient is same as linear interpolation and Gaussian process regression method. However, mode coefficent has to split into windowed dataset for time series data learning. First process is dividing all time series data into learning data as much as the ratio of split. Rest of the remaining data is used for testing.

```
def train_test_split(t, y, split = 0.8):

        indx_split = int(split * len(y))
        indx_train = np.arange(0, indx_split)
        indx_test = np.arange(indx_split, len(y))
        t_train = t[indx_train]
        y_train = y[indx_train]
        t_test = t[indx_test]
        y_test = y[indx_test]

        return t_train, y_train, t_test, y_test
```

Second step is preprocessing learning and testing data into windowed dataset that is necessary for time series learning. LSTM, a method specialized in extrapolation,

enters sequent previous time series data and predicts the data corresponding to subsequent time. In Figure 2-8, the example shows predicting process of 2 time series data in constant stride through 3 time series data. In code below, input_window means the number of input time series data, output_window is the number of data to predict, stride refers to time interval that prediction is placed. num_feature is total number of used dataset. In this step, 80 of flow field data are used.

```
def windowed_dataset(y, input_window = 9, output_window = 1, stride = 1, num_features = 80):

        L = y.shape[0]
        num_samples = (L - input_window - output_window) // stride + 1
        X = np.zeros([input_window, num_samples, num_features],dtype=np.double)
        Y= np.zeros([output_window, num_samples, num_features],dtype=np.double)

        for ff in np.arange(num_features):
                for ii in np.arange(num_samples):
                        start_x = stride * ii
                        end_x = start_x + input_window
                        X[:, ii, ff] = y[start_x:end_x, ff]

                        start_y = stride * ii + input_window
                        end_y = start_y + output_window
                        Y[:, ii, ff] = y[start_y:end_y, ff]

        return X, Y
```

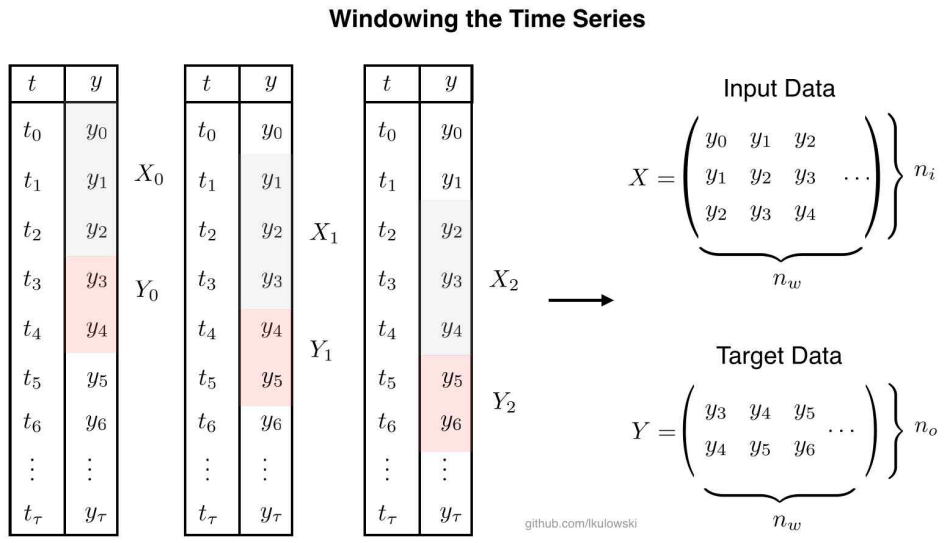


Figure 2-8. Time Series Data Prediction Process Using LSTM

numpy to torch function trasfrom snap shot data which is divided into four types into tensor data type which is easy to use for LSTM learning.

```
def numpy_to_torch(Xtrain, Ytrain, Xtest, Ytest):

    X_train_torch = torch.from_numpy(Xtrain).type(torch.Tensor)
    Y_train_torch = torch.from_numpy(Ytrain).type(torch.Tensor)
    X_test_torch = torch.from_numpy(Xtest).type(torch.Tensor)
    Y_test_torch = torch.from_numpy(Ytest).type(torch.Tensor)

    return X_train_torch, Y_train_torch, X_test_torch, Y_test_torch
```

2) Prediction and Reconstruction of Mode efficient using LSTM (*POD_Evaluator_LSTM_Unsteady.py*)

lstm seq2seq class is a function setting up LSTM model. Using hidden_size specifies the depth of neural network, and using input_size sets the number of data which is entered per time series. Through train_model function, the created model is trained. Initial training speed went up using mixed teacher forcing method which corrects wrong prediction values from learning process through ground truth. Also basic parameter optimization can be conducted by modulating n_epoch, learning_rate, batch_size variables. Through LSTM, predicted mode coefficient is reconstructed as same method as 가 to 나.

```
class lstm_seq2seq(nn.Module):
    def __init__(self, input_size, hidden_size):

    super(lstm_seq2seq, self).__init__()
    self.input_size = input_size
    self.hidden_size = hidden_size
    self.encoder = lstm_encoder(input_size = input_size, hidden_size =
     hidden_size)
    self.decoder = lstm_decoder(input_size = input_size, hidden_size =
      hidden_size)

def  train_model(self,  input_tensor,  target_tensor,  n_epochs,  target_len=output_window,
     batch_size,  training_prediction  =  'mixed_teacher_forcing',  teacher_forcing_ratio  =
     0.6, learning_rate = 0.01, dynamic_tf = False):
    losses = np.full(n_epochs, np.nan)
    optimizer = optim.Adam(self.parameters(), lr = learning_rate)
    criterion = nn.MSELoss()
    n_batches = int(input_tensor.shape[1] / batch_size)
    with trange(n_epochs) as tr:
        for it in tr:
            batch_loss = 0.
            batch_loss_tf = 0.
            batch_loss_no_tf = 0.
            num_tf = 0
            num_no_tf = 0
            for b in range(n_batches):
                input_batch = input_tensor[:, b: b + batch_size, :]
                target_batch = target_tensor[:, b: b + batch_size, :]
                outputs = torch.zeros(target_len, batch_size, input_batch.shape[2])
```

```
                encoder_hidden = self.encoder.init_hidden(batch_size)
                optimizer.zero_grad()
                encoder_output, encoder_hidden = self.encoder(input_batch)
                decoder_input = input_batch[-1, :, :]
                decoder_hidden = encoder_hidden
                if training_prediction == 'mixed_teacher_forcing':
                    # predict using mixed teacher forcing
                    for t in range(target_len):
                        decoder_output, decoder_hidden =
                      self.decoder(decoder_input, decoder_hidden)
                        outputs[t] = decoder_output
                        # predict with teacher forcing
                        if random.random() < teacher_forcing_ratio:
                            decoder_input = target_batch[t, :, :]
                        # predict recursively
                        else:
                            decoder_input = decoder_output
                # compute the loss
                loss = criterion(outputs, target_batch)
                batch_loss += loss.item()
                # backpropagation
                loss.backward()
                optimizer.step()
            # loss for epoch
            batch_loss /= n_batches
            losses[it] = batch_loss
            # dynamic teacher forcing
            if dynamic_tf and teacher_forcing_ratio > 0:
                teacher_forcing_ratio = teacher_forcing_ratio - 0.02
            # progress bar
            tr.set_postfix(loss="{0:.3f}".format(batch_loss))
    return losses
```

3) Example Results

In left side of the Figure 2-9, it is the result of contouring velocity field predicted at $t = 17.0$, time which was not used for learning. In the right side, it is the absolute error contour from FOM CFD result. Related error is measured as 0.33%.

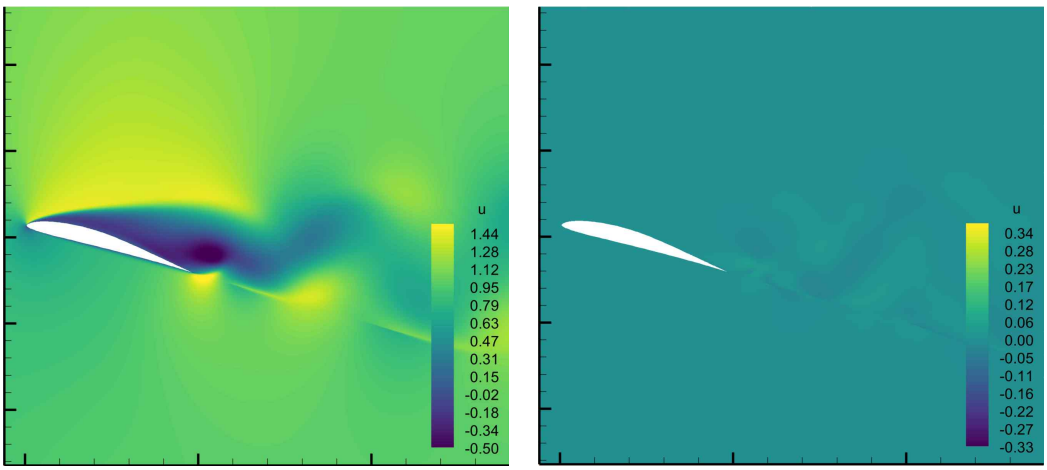


Figure 2-9. Velocity Field Prediction Result Using LSTM(Left)and Error(Right)

III. Fully Connected Deep Neural Network (FCDNN)

FCDNN is comprised of more than two hidden layer between input layer and output layer, and is an artificial neural network which every node is interconnected. Good side of FCDNN is that data's complex features are well trained compared to neural network with a few number of hidden layer. Also due to simple structure, FCDNN can be used for many modeling correlation. FCDNN is useful for for tasks such as classification, numerical prediction, image learning. But due to quantitatively large number of parameter(weight between nodes) that needs learning process, there are issues such as taking long hours to learn, large quantity of required memory compared to CNN.

A. Direct FCDNN

In this part of the report will introduce tutorial directly predicting flow field using FCDNN. In Figure 2-10 shows schematic diagram directly predicting method of flow field using Fully Connected Deep Neural Network(FCDNN).

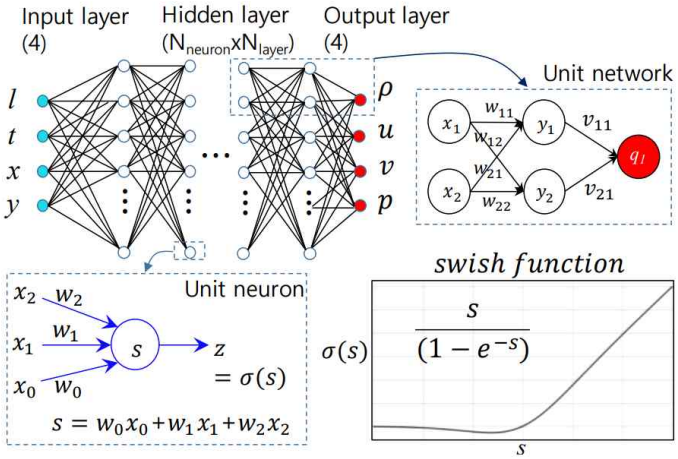


Fig. 3. Fully Connected Deep Neural Network for Direct Field Inference.

Figure 2-10. Schematic Diagram of FCDNN Model

1) Preposcessing and Extration of Snapshot Data (*ExtracData_Unsteady.py*)

Data preprocessing is necessary in order to conduct learning process of flow field which is gained from CFD within FCDNN to predict flow field using FCDNN. The code

below is to preprocess flow field which is gained from CFD. For exmaple, noCol is a number of flow variable included in data, and numb is number of timestep of flow field used for learning. inc_time is an interval of timestep of learning data.

```
import numpy as np
import pandas as pd
noCol = 11   # number of flowfield variables
numd = 20    # number of snapshots
initial_time = 101
inc_time = 4 # time interval of training data
zone1_i = 689
zone1_j = 145
dt = 0.1
wt = 1000
d_inf = 1.225
U_inf = 0.005*343
sim_data_path = "../../../Database/sol01RANS3.8208da/"
Tecplot_header_in = "variables=X, Y, Z, Rho, U, V, W, P, T, Vor, Qcri"
Tecplot_header_out = "variables=X, Y, Rho, U, V, P"
```

The code below is preprocessing into a suitable form to make FCDNN learn after loading flow field data gained from CFD. From total 11 flow variables, density(q), speed(U,V), pressure(P), CFD node-based point(X,Y) and time information are all saved as one compressed file.

```
snapshot_data = np.array([])
for i in range(0,numd):
    pd_data = pd.read_csv(sim_data_path+'flo001.0000'+str(initial_time+i*inc_time)
                    .rjust(3,'0')+'uns', dtype='float64', delimiter=' '
                    , skipinitialspace=True, skiprows=2, header=None)
    # make it an np ndarray
    data = np.nan_to_num(pd_data.values)
    array_data = data.flatten()
    array_data = array_data.reshape(-1,noCol)
    if i==0:
        snapshot_data = array_data
        xy = snapshot_data[:,0:2]
        N = snapshot_data.shape[0]
    else:
        snapshot_data = np.vstack((snapshot_data, array_data))
array_data1 = snapshot_data
shp = array_data1.shape
print(shp)
## Vectorize each variable
t_star = np.arange(Ntime)[:,None]*dt*inc_time
T = t_star.shape[0]
xc_star = array_data1[:,0]
yc_star = array_data1[:,1]
NT = xc_star.shape[0]
dc_star = array_data1[:,3]
uc_star = array_data1[:,4]
vc_star = array_data1[:,5]
```

```
pc_star = array_data1[:,7]
print(xc_star.shape)
DC = np.reshape(dc_star, (T,N)).T
UC = np.reshape(uc_star, (T,N)).T
VC = np.reshape(vc_star, (T,N)).T
PC = np.reshape(pc_star, (T,N)).T
XC = np.reshape(xc_star, (T,N)).T
YC = np.reshape(yc_star, (T,N)).T
TC = np.tile(t_star, (1,N)).T
```

2) Learning and Prediction of Flow Field using FCDNN (*FCDNN_Unsteady.py*)

In this report, layer of FCDNN is defined as code below. In input layer of this neural network, there are time information of flow field and X,Y coordinate value of CFD grid. In output layer, flow variable of interest such as density, speed, pressure values are included and used for learning.

```
if __name__ == "__main__":
    with tf.device('/gpu:0'):
        batch_size = 4096
        layers = [3] + 10*[4*10] + [4]
```

Properly adjusting hyperparameter such as learning rate when conducting learning process of neural network is one of important factor that decides performance of neural network of learned neural network. The code below shows improving performance of neural network by modulating decrease pattern by decreasing learning rate as learning process continues.

```
        def train(self, total_time, learning_rate):
        N_data = self.xydata.shape[0]
        start_time = time.time()
        running_time = 0
        it = 0
        while running_time < total_time:   # The training keeps going until total_time (h)
            idx_data = np.random.choice(self.t_data.shape[0], self.batch_size)
            if it == 50000:   # As iteration goes further, learning rate decreases
                learning_rate = 1e-3
            if it == 100000:
                learning_rate = 5e-4
            if it == 150000:
                learning_rate = 1e-4
            if it == 200000:
                learning_rate = 5e-5
            if it == 250000:
                learning_rate = 1e-5
            if it == 300000:
                learning_rate = 1e-6
            if it == 350000:
                learning_rate = 1e-7
            if it == 400000:
```

```
learning_rate = 1e-8
```

By using flow field data gained from CFD and trained FCDNN, flow field is predicted, and predicted flow field is saved.

```
# Write the predictions
        for i in range(20):
            t_test = T_test.T[:,i:i+1]
            x_test = X_star.T[:,i:i+1]
            y_test = Y_star.T[:,i:i+1]
            # Prediction
            d_pred, u_pred, v_pred, p_pred = model.predict(t_test, x_test, y_test)
            p3d_result = np.hstack((xydata[:,0][:,None], xydata[:,1][:,None], d_pred,
                                    u_pred, v_pred, p_pred))
            np.savetxt("../Results/Case_flo8_RANS_NN_cuttail2_t="+str(initial_time+
                       i*inc_time+2)+".dat", p3d_result, delimiter=" ",
                       header="variables = X, Y, c, u, v, p \n zone i="+str(zone1_i
                       -2*cuttail)+" j="+str(glayer)+" ", comments=' ')
            # Error
            error_d = relative_error(d_pred, DC_star[:,i][:,None])
            error_u = relative_error(u_pred, UC_star[:,i][:,None])
            error_v = relative_error(v_pred, VC_star[:,i][:,None])
            error_p = relative_error(p_pred, PC_star[:,i][:,None])
            print('Error d: %e, u: %e, v: %e, p: %e' % (error_d, error_u, error_v, error_p))
            error_d_sum += error_d
            error_u_sum += error_u
            error_v_sum += error_v
            error_p_sum += error_p
        print('Mean of Error d: %e, u: %e, v: %e, p: %e' % (error_d_sum/numd,
              error_u_sum/numd, error_v_sum/numd, error_p_sum/numd))
```

3) Example Result

In left side of the Figure 2-11, it is the result of contouring velocity field predicted at $t = 10.3$, time which was not trained. In the right side, it is the absolute error contour of FOM CFD result. Related error is measured as 7.20%.

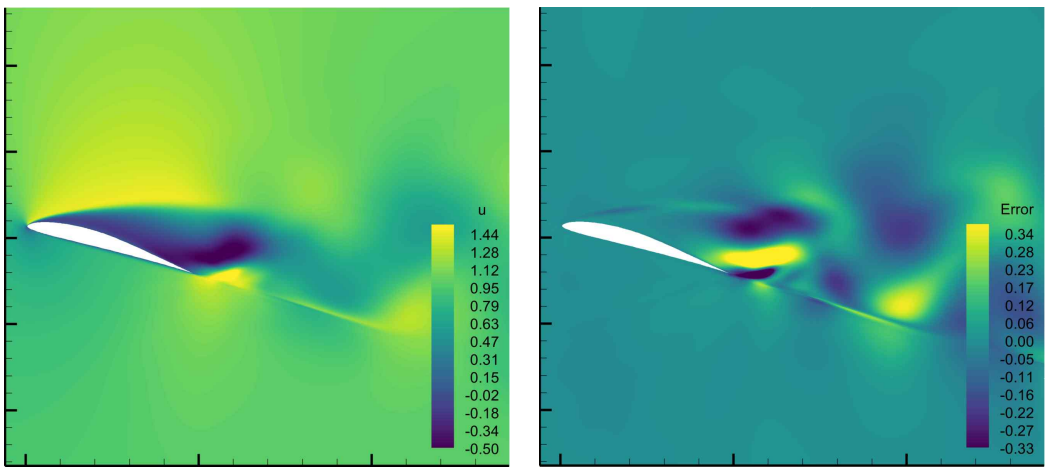


Figure 2-11.Velocity Field Prediction Result Using FCDNN(Left)and Error(Right)

B. PODNN

In this page of the report introduces tutorial predicting mode coefficient of POD(POD-DNN method) by using FCDNN. Figure 2-12 is an schematic diagram of POD-DNN.

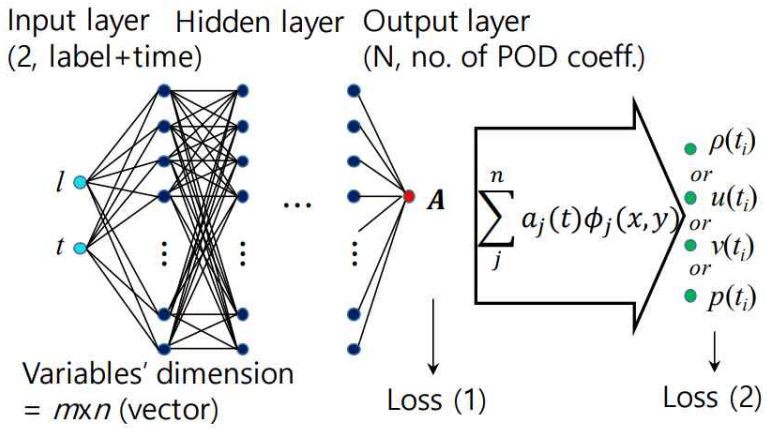


Fig. 4. Schematic of mixed POD-DNN Method

Figure 2-12. Schematic Diagram of POD-DNN

1) Prediction and Reconstruction of Mode Coefficient using FCDNN (*PODNN_Unsteady.py*)

Following two processes are same as the description of local linear interpolation above. One is preprocessing and extraction process of snapshot data using ExtracData_POD_Unsteady.py. Another one is extraction process of POD basis vector and mode coefficient using POD_SVD_Unsteady.py. This report defines layer of FCDNN as the code below. The input layer of this neural network contains time information of flow field and label of time. The output layer contains mode coefficient gained from POD analysis and used for learning.

```
if __name__ == "__main__":
    with tf.device('/gpu:0'):
        batch_size = 400
        layers = [2] + 10*[2*20] + [20]
```

Properly adjusting hyperparamer such as learning rate when conducting learning

process of neural network is one of important factor that decides performance of neural network of learned neural network. The code below shows modulating performance of neural network by adjusting decrease pattern by decreasing learning rate as learning process continues.

```
    def train(self, total_time, learning_rate):
            start_time = time.time()
            running_time = 0
            it = 0
            while running_time < total_time:
                if it == 8000:            # As iteration goes further, learning rate decreases
                    learning_rate = 5e-4
                if it == 15000:
                    learning_rate = 1e-4
                if it == 20000:
                    learning_rate = 5e-5
                if it == 50000:
                    learning_rate = 1e-5
                if it == 100000:
                    learning_rate = 5e-6
                if it == 150000:
                    learning_rate = 1e-6
                if it == 200000:
                    learning_rate = 5e-7
                if it == 250000:
                    learning_rate = 1e-7
                if it == 300000:
                    learning_rate = 5e-8
                if it == 350000:
                    learning_rate = 1e-8
                if it == 400000:
                    learning_rate = 1e-9
```

Using mode coefficient gained from the result of FCDNN, the flow field is reconstructed according to the method explained earlier.

```
    U_pod = np.add(np.transpose(np.matmul(np.transpose(a_pred), np.transpose(phi))),
                           np.tile(mean_data, (1,Ntime)))
            error_d_sum = 0
            error_u_sum = 0
            error_v_sum = 0
            error_p_sum = 0
            for i in range(Ntime):
                U_pred = U_pod[:,i].reshape(-1, noConcernVar)
                d_pred = U_pred[:,0][:,None]
                u_pred = U_pred[:,1][:,None]
                v_pred = U_pred[:,2][:,None]
                p_pred = U_pred[:,3][:,None]
                p3d_result = np.hstack((xydata[:,0][:,None], xydata[:,1][:,None],
                             d_pred, u_pred, v_pred, p_pred))
                np.savetxt("../Results/Case_flo_R_PODDNN_t="+str(initial_time+i*inc_time+2)
                           + ".dat", p3d_result, delimiter=" ",
                           header="variables = X, Y, c, u, v, p \n zone i="+str(zone1_i)
```

```
                    + " j="+str(zone1_j)+" ", comments=' ')
        # Error
        error_d = relative_error(d_pred, DC_star[:,i][:,None])
        error_u = relative_error(u_pred, UC_star[:,i][:,None])
        error_v = relative_error(v_pred, VC_star[:,i][:,None])
        error_p = relative_error(p_pred, PC_star[:,i][:,None])
        print('Error d: %e, u: %e, v: %e, p: %e' % (error_d, error_u, error_v, error_p))
```

2) Example Result

In left side of the Figure 2-11, it is the result of contouring velocity field predicted at $t = 10.3$, time which was untrained. In the right side, it is the absolute error contour of FOM CFD result. Related error is measured as 0.8%.

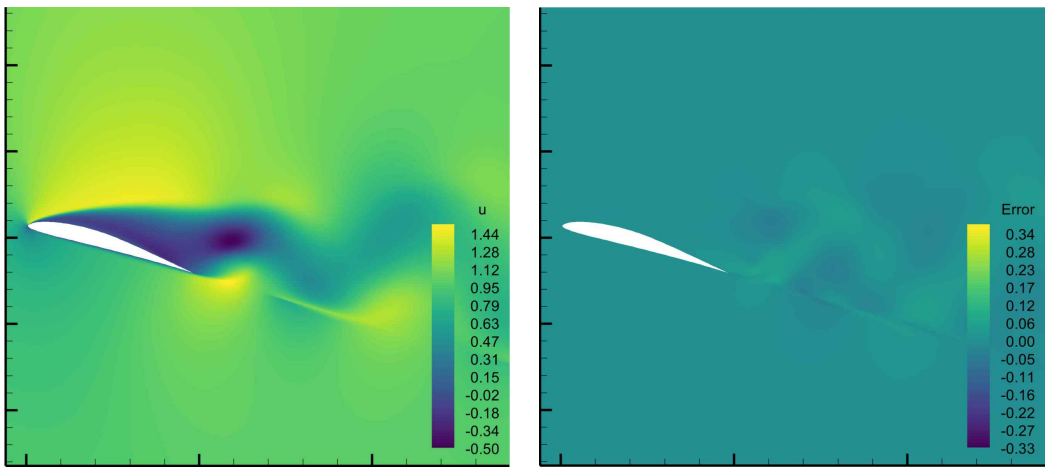


Figure 2-13. Velocity Field Prediction Result Using PODNN(Left) and Error(Right)

IV. U-Net Reduced Model on the basis of CNN

In this part of the report introduces API tutorial of reduced-order modeling which applied Conditional U-Net on the basis of CNN. Conditional U-Net can be greatly applied in wide range because it can affect generating user-oriented output by adding specific condition to battle neck layer of previous U-Net. In this section talks about unsteady flow that time is added as a condition.

A. Preprocessing and Extraction of Snapshot Data (*ExtracData_Unsteady_cuttail.py*)

User can partially extract the part of interest out of total snapshot data just as Figure 2-14 by adjusting cuttail and glayer variable. This way can promote learning speed and lessens memory burden by deleting data in areas of unncessary far-field. Cuttail variable deletes i area of grid as much as set value. glayer variable accumulates j area of data as much as set value and deletes remaining data.

```
noCol = 11
numd = 20
initial_time = 103
inc_time = 4
zone1_i = 689
zone1_j = 145
cuttail = 0     # = crop_i
glayer = 145    # = crop_j
dt = 0.1
```

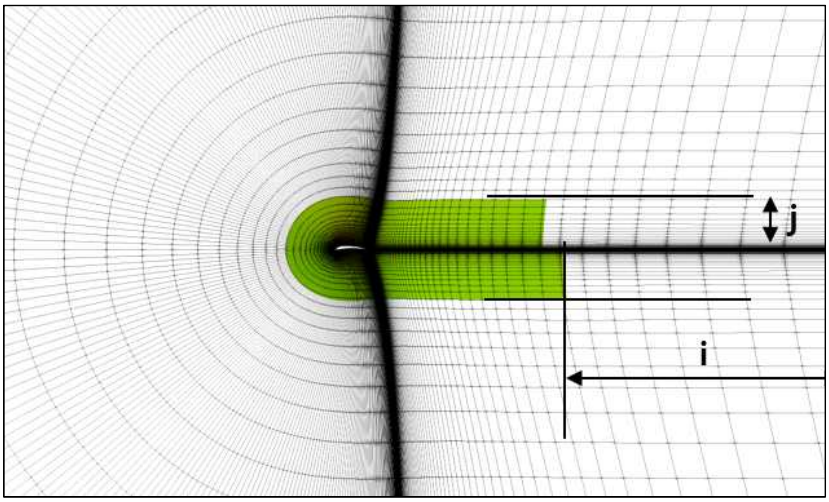


Figure 2-14. Snapshot Data With Far-Field Area Deleted

The code below deletes unnecessary data after rearranging snapshot data to vector form according to set value. Data which completed preprocessing can be used for learning. Also initial_time variable and inc_time variable can be modulated to divide data for each of learning and model evaluation.

```
# cut grid for memory efficiency
idx_x_slice = np.array([])
for i in range(glayer):
    idx_x_slice = np.append(idx_x_slice, np.arange
    ((cuttail)+i*zone1_i,(zone1_i-(cuttail))+i*zone1_i)).astype('int32')
#print(idx_x_slice)
print(idx_x_slice.shape[0])
```

```
DC_star = DC[idx_x_slice,:]
UC_star = UC[idx_x_slice,:]
VC_star = VC[idx_x_slice,:]
PC_star = PC[idx_x_slice,:]
XC_star = XC[idx_x_slice,:]
YC_star = YC[idx_x_slice,:]
TC_star = TC[idx_x_slice,:]
```

B. Formation and Learning of Network Model (*Unet_UIUC_Unsteady.py*)

Information below is hyperparameters affecting network model learning. There is no predetermined value, and user can set up optimal value empirically.

```
# Model parameters
BATCH_SIZE = 20     # 학습 데이터 개수
validation_split = 0.1  # 학습 데이터 중에서 validation 데이터 비율
EPOCHS = 30000      # 학습 횟수
LATENT_DEPTH = 64   # Bottle neck layer의 크기
NB_CHANNELS = 1     # Condition 채널의 크기
LR = 1e-3           # Learning rate
IMAGE_SHAPE = [zone1_i, zone1_j]
```

Since input and ouput size have to be same, U-Net uses raster data, x and y as input values. For output value, flow field data of corresponding U,V,P are used. At this time, cost function of shape vector is defined to maintain airfoil including raster data in output value. The code below calculates shape vector necessary for cost function calculation.

```
#Shape vector
idx_bottom = np.where(xydata[:,0] == xydata[1,0])[0]
i = 1
for i in range(1,idx_bottom[1]):
    if(xydata[i,1] != xydata[idx_bottom[1]-i+1,1]):
        break
idx_tip = [i-1, idx_bottom[1]-i+2]
idx_x_bd1 = np.arange(1, idx_tip[0]+1)
idx_x_bd2 = np.arange(idx_tip[1], idx_bottom[1]+1)
idx_x_sur = np.arange(idx_tip[0],idx_tip[1]+1)
T = numd
N = xydata.shape[0]
```

The code below is conducting second preprocessing in form that is needed for Unet learning and testing after loading preprocessed learning and testing data.

```
# Train set
XI_star = np.reshape(XC_star.T, [numd, glayer, zone1_i-2*cuttail])
XI_field = XI_star[:,:-1,:-1]
YI_star = np.reshape(YC_star.T, [numd, glayer, zone1_i-2*cuttail])
YI_field = YI_star[:,:-1,:-1]
UI_star = np.reshape(UC_star.T, [numd, glayer, zone1_i-2*cuttail])
UC_field = UI_star[:,:-1,:-1]
VI_star = np.reshape(VC_star.T, [numd, glayer, zone1_i-2*cuttail])
VC_field = VI_star[:,:-1,:-1]
PI_star = np.reshape(PC_star.T, [numd, glayer, zone1_i-2*cuttail])
PC_field = PI_star[:,:-1,:-1]
# Test set
XT_star = np.reshape(XT_star.T, [numT, glayer, zone1_i-2*cuttail])
XT_field = XT_star[:,:-1,:-1]
YT_star = np.reshape(YT_star.T, [numT, glayer, zone1_i-2*cuttail])
YT_field = YT_star[:,:-1,:-1]
UT_star = np.reshape(UT_star.T, [numT, glayer, zone1_i-2*cuttail])
UT_field = UT_star[:,:-1,:-1]
VT_star = np.reshape(VT_star.T, [numT, glayer, zone1_i-2*cuttail])
VT_field = VT_star[:,:-1,:-1]
PT_star = np.reshape(PT_star.T, [numT, glayer, zone1_i-2*cuttail])
```

The code below defines encoder structure of a model. It is about down sampling feature map in a row by 3x3 convolution and Maxpooling function. Layer number and size can suitably be comprised by users just like hyperparameter.

```
# encoder
input_e = tf.keras.Input(shape=(glayer-1, zone1_i-2*cuttail-1, 2))
pooling_size = 2
n_ch = 12
conv1 = layers.Conv2D(n_ch, (3,3), activation='elu', padding = 'same')(input_e)
mp1 = layers.MaxPooling2D((pooling_size,pooling_size))(conv1)
conv2 = layers.Conv2D(n_ch*2, (3,3), activation='elu', padding = 'same')(mp1)
mp2 = layers.MaxPooling2D((pooling_size,pooling_size))(conv2)
```

In bottle neck layer, output of encorder can be flattened as one vector. Flattened vector can add time vector using concatenate function. The vector which added condition such as time, can be changed to be used as deconder input again through fully connected layer.

```
# bottle neck layer
flat1 = layers.Flatten()(mp4)
dense1 = layers.Dense(LATENT_DEPTH, activation='elu')(flat1)
time = tf.keras.Input(shape=(1,))
dense2 = layers.Concatenate(axis=1)([time, dense1])
dense3 = layers.Dense(LATENT_DEPTH+1, activation='elu')(dense2)
dense4 = layers.Dense(LATENT_DEPTH, activation='elu')(dense3)
dense5 = layers.Dense(n_ch*8*(zone1_i-2*cuttail-1)/pooling_size/pooling_size
                      /pooling_size/pooling_size*(glayer-1)/pooling_size
```

```
                        /pooling_size/pooling_size/pooling_size, activation='elu')(dense4)
    output_f = layers.Reshape((int((glayer-1)/pooling_size/pooling_size/pooling_size
                               /pooling_size),int((zone1_i-2*cuttail-1)/pooling_size
                               /pooling_size/pooling_size/pooling_size), n_ch*8))(dense5)
```

In decoder, snapshot data is predicted by up sampling vectors which is inputed from bottle neck layer into dimension of original learning data. Feature of U-Net can conduct concatenate of encoder layer's results which is in horizontal relationship with corresponding layer of each layer that is decoded. Also this process can well reflect data feature better by fusing through convolution.

```
    # decoder
    convt1 = layers.Conv2DTranspose(n_ch*8, (3,3), activation='elu', padding='same')(output_f)
    upsamp1 = layers.UpSampling2D((pooling_size,pooling_size))(convt1)
    skipcon1 = layers.Concatenate(axis=3)([conv4, upsamp1])
    conv6 = layers.Conv2D(n_ch*8, (3,3), activation = 'elu', padding='same')(skipcon1)
    convt2 = layers.Conv2DTranspose(n_ch*4, (3,3), activation='elu', padding='same')(conv6)
    upsamp2 = layers.UpSampling2D((pooling_size,pooling_size))(convt2)
    skipcon2 = layers.Concatenate(axis=3)([conv3, upsamp2])
    conv7 = layers.Conv2D(n_ch*4, (3,3), activation = 'elu', padding='same')(skipcon2)
```

The code below defines cost function. It defines as sum of cost function regarding surface grid's shape vector and regarding flow field data. The reason to use shape vector for cost function calculation and the corresponding calculation are explained in earlier pages.

```
    # Loss calculation
    def custom_mse(idx_x_bd1, idx_x_bd2):
        def loss(y_true,y_pred):
            # Extract boundary values
            train_bd1 = y_pred[:,0:1,:,:][:,:,idx_tip[0]:1:-1,:]
            train_bd2 = y_pred[:,0:1,:,:][:,:,idx_tip[1]:idx_bottom[1],:]
            # calculating squared difference between target and predicted values
            loss1 = tf.keras.backend.square(y_true - y_pred)
            loss2 = tf.keras.backend.square(train_bd1 - train_bd2)
            # summing both loss values along batch dimension
            loss1 = tf.keras.backend.mean(tf.keras.backend.sum(loss1, axis=1))
            loss2 = tf.keras.backend.mean(tf.keras.backend.sum(loss2, axis=1))
            print(loss1.shape, loss1.shape)
            return loss1 + loss2
        return loss
```

The code below is defined earlier which is used for U-Net model training. This code can set up model's learning data, cost function, a number of iteration or others For input value, time data added in Bottle neck layer and x,y grid data are used. For

output value, flow field data is used. The below part of model loading is a code used for retrieving when necessary after saving model which is done learning earlier. It has to go through annotation processing while learning. At the lowest below, it's a code testing untrained data using model.

```
#Train model
unet = Model(inputs=[input_e, time], outputs=output_d)
lr_schedule = optimizers.schedules.ExponentialDecay(
    initial_learning_rate=LR,
    decay_steps=300,
    decay_rate=0.9)
reduce_lr = LearningRateScheduler(lr_schedule)
optimizer = optimizers.Adam(learning_rate=lr_schedule)
unet.compile(loss=losses.MeanSquaredError(), metrics=["mae"])
#optimizers.Adam(learning_rate=0.005) optimizer='adam',
unet.fit([train_inp, train_inp_t], train_env, validation_split=0.1,
epochs=EPOCHS, verbose=2, callbacks=[reduce_lr])
unet.save(res_data_path+"my_model.h5")
```

C. Example Results

In left side of the Figure 2-15, it is the result of contouring velocity field predicted at $t = 10.3$, time which was untrained. In the right side, it is the absolute error contour of FOM CFD result. Related error is measured as 3.2%

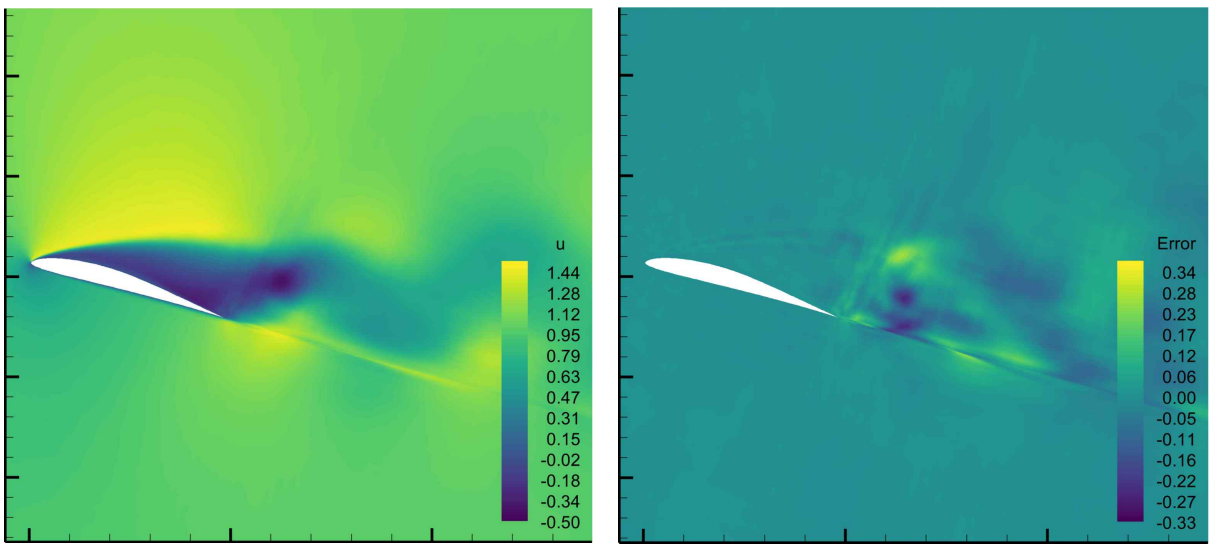


Figure 2-15. Velocity Field Prediction Results Using CNN U-Net(Left) and Error(Right)

V. Result Comparison

In this chapter compares performance of reduced models which is explained earlier. The Figure 2−16 below is about contouring velocity field absolute error in 5 types of reduced model of FOM CFD. The table 2−16 shows average relative error of testing data. POD−LSTM method is demonstrated separately due to distinctive characteristics of data preprocessing.

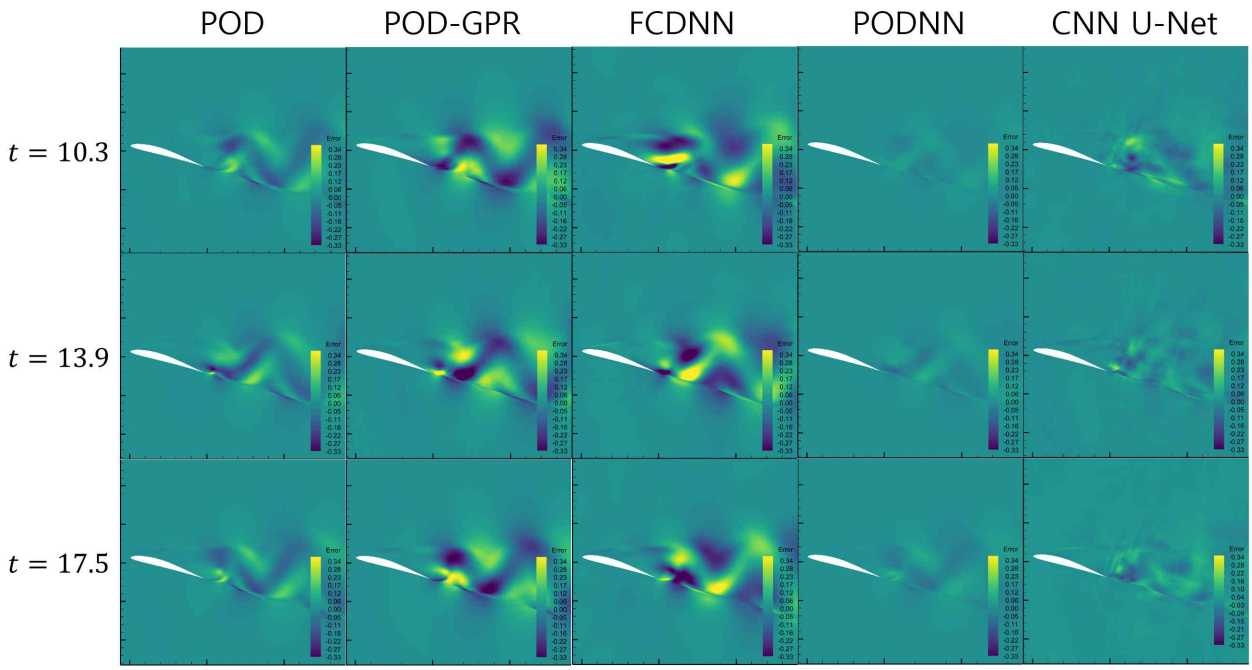


Figure 2-16.Comparison of Velocity Field's Absolute Error of 5 Reduced Models

As a result of comprising reduced models using learning data explained earlier, model that show highest accuracy of unsteady flow prediction is PODNN, followed by POD and CNN−Unet showing the accuracy. Various reduced models introduced in this section show very much sensitive in performance depending on data feature and size and degree of optimization of model's parameter. So that it is very hard to conclude which model is superior to another regarding model performance based only on the results above. For accurate comparison, testing of more diverse learning data has be be followed. Therefore, users has to select and apply a model that suitable for them.

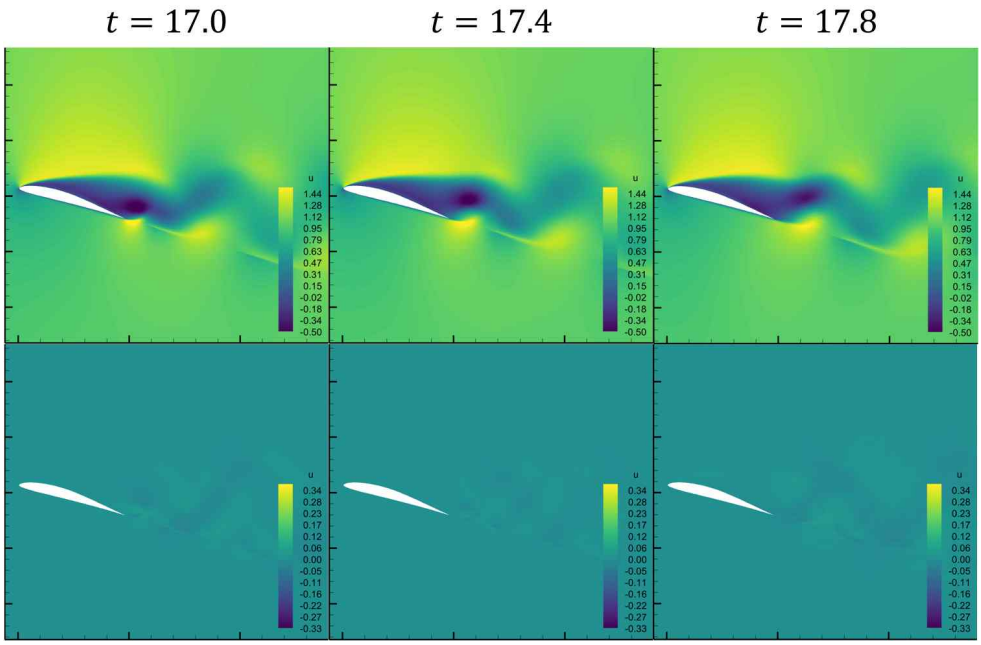


Figure 2-17. Comparison of Velocity Field's Absolute Error of POD-LSTM

The Figure 2-17 above gives prediction accuracy of POD-LSTM model of time-series data. Upper Figure shows velocity prediction of the model, and lower Figure is absolute error contour of FOM CFD data. Previous 5 types of models are models for interpolation which includes in learning data. POD-LSTM is a model for extrapolation of learning data. Therefore time-series data for LSTM learning uses flow field of sequential 9 time as input value. Following 10^{th} time flow field is used for prediction. The result above shows that learning data from 10.1 to 16.0 second is used to comprise a learning model and predicts time after 17.0 second.

Chapter 3 - Case 2) Steady Flow in Variable Flow Condition

Steady Cases with Fixed Airfoil Shape and Variable Flow Condition
AI methods : POD-GPR, Conditional U-Net

I. Learning Data Formation

In this chapter 3, analysis of full order model(FOM) CFD is conducted to create a reduced model(ROM) for analyzing steady flow of various flow condition. Total 15 angle of attack from 1° to 7° in an interval of 0.5 are used. For Mach numbers, total 9 Mach number from 0.3 to 0.025 in an interval of 0.5 are used. So total number of CFD data is 135. For airfoil, NACA0012 is used, and below Figure 3-1 is NACA0012 grid used for steady CFD analysis.

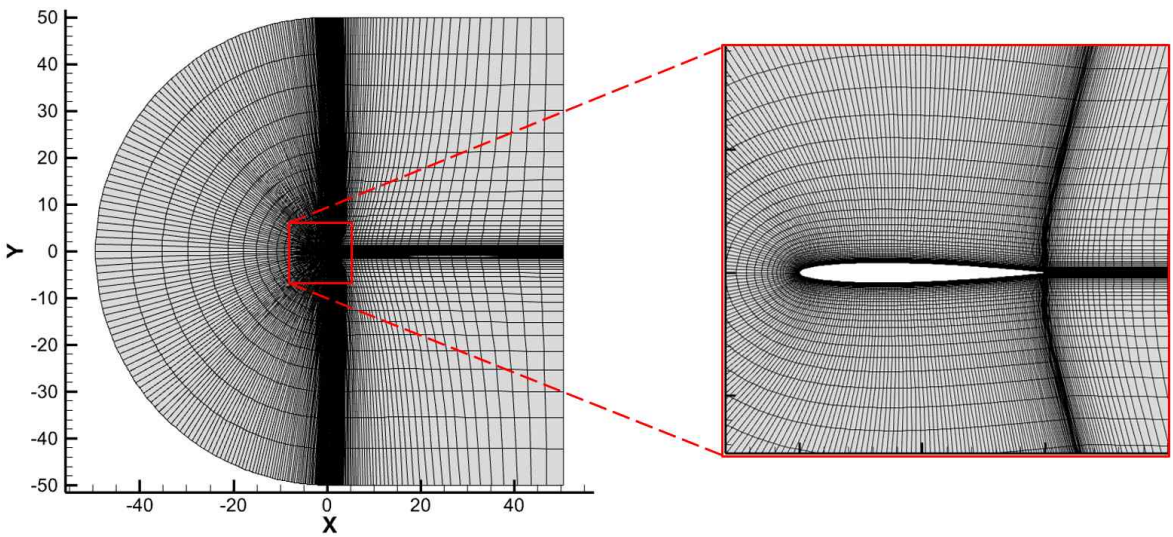


Figure 3-1. NACA0012 Grid Used For Dataset Formation

The table 3 below summarizes CFD anlysis condition to creat learning data.

Table 3. CFD Analysis Condition

Solver	Airfoil	Grid	Flow condition	Governing equation	Flux function
KFLOW	NACA0012	401x81 (C-type)	Mach 0.3~0.5 AoA 0~7°	RANS S-A model	Roe flux 3rd order

Based on previously described condition, flow field generated around airfoil goes thorugh CFD analysis. Among them, flow field at Mach number 0.325, 0.375, 0.425, 0.475 and each angle of attack 1°, 3.5°,6° are predicted so that total 12 testing data is used to evaluate performance of reduced model.

II. POD-GPR Reduced Model

In this section introduces API tutorial of reduced modeling method which can predicts flow field under diverse flow condition by using POD and GPR. Contents below is based on chapter 2, and this section is about additional contents.

A. Prediction and Reconstruction of Mode Coefficient (*POD_Evaluator_GP_FC.py*)

Below is the code for loading learning and testing data, and changing data's form into suitable for POD, The data for learning and testing is classified as mentioned in of data generation section above.

```
#Load train data
saved_npz = np.load(res_data_path +"Staedy_airfoil_cuttail_train.npz")
snapshot_data4 = saved_npz['DC']
snapshot_data1 = saved_npz['UC']
snapshot_data2 = saved_npz['VC']
snapshot_data3 = saved_npz['PC']
snapshot_data = np.vstack((snapshot_data1 ,snapshot_data2 ,snapshot_data3))
xy1 = saved_npz['XC']
xy2 = saved_npz['YC']
xydata =np.column_stack((xy1[:,0],xy2[:,0]))

#Load test data
saved_npz_e = np.load (res_data_path +"Staedy_airfoil_cuttail_test.npz")
```

```
U_true = saved_npz_e['UC']
V_true = saved_npz_e['VC']
P_true = saved_npz_e['PC']
```

Below is a process calculating three things. One is mode coefficient necessary for GPR learning through POD. Another is average value necessary for POD construction. Lastly calculating basis vector is included. From above, it is a code for calculating data regularization, singlular value decomposition, mode coefficient. More details about theoretical background is specified in chapter 2 and appendix.

```
#Mean center the data
mean_array = None
mean_data_tensor = np.mean(snapshot_data, axis =1)
mean_centered_data = np.subtract(snapshot_data, np.tile(mean_data_tensor, (numd,1)).T)

#Singular Value Decomposition
u, s, v = np.linalg.svd(mean_centered_data, compute_uv =True, full_matrices =False

#POD coefficients
compute_coeffs = np.matmul(np.transpose(mean_centered_data), u)
e = np.sum(s)
s_energy = np.divide(s, e)*100
coeffs = compute_coeffs
mean_data = mean_data_tensor
mean_tensor = tf.constant(mean_data , name ="mean_data_tensor")
```

The code above calculates mode coefficient which is an output value necessary for GPR learning. In other hands, the code below defines input value. In this section, total of 2 which are Mach number and angle of attack are used for input value. This input value is classified for learning and testing according to corresponding mode coefficient.

```
#Make label
Nm = 1; Na = 1
k = 0; kk = 0
for i in range(Totnum):
  if Nm%2 == 0 and (Na+2)%5 == 0:
    test_ipt_M[k] = Nm
    test_ipt_A[k] = Na
    k += 1
  else :
    train_ipt_M[kk] = Nm
    train_ipt_A[kk] = Na
    kk += 1

  if Na %15 == 0:
    Nm += 1
    Na = 0
  Na += 1
```

```
train_x[:,0] = train_ipt_M
train_x[:,1] = train_ipt_A
test_x[:,0] = test_ipt_M
test_x[:,1] = test_ipt_A
```

The code below is GPR learning code which used GPy Python package. For kernel function, Matern52 function is used. Also multivariate GPR is used as a model because input value is two dimensional due to Mach number and angle of attack. In addition, hyperparameter optimization is conducted through Maximum likelihood estimation.

```
#Run GPR
ker = GPy.kern.Matern52(n_var, ARD =True)
m = GPy.models.GPRegression(train_x,coeffs,ker)
m.optimize(messages =True, max_f_eval =1000)
```

Untrained mode coefficient is predictable under flow condition using testing data's input value which is ready to use once GPR learning is completed. Predicted mode coefficient is reconstructed as entire flow field based on calculation of basis vector and average value which is saved separately after POD.

```
for i in range(numT):
  print(i)
  x_pred[0] = test_x[i]
  interp_coeffs = m.predict(x_pred)[0][0]
  int_coeff_tensor = tf.Variable(interp_coeffs)
  #add a dim to make it a 2-D tensor
  int_coeff_tensor = tf.expand_dims(int_coeff_tensor, 0)
  #compute the POD approximation
  with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    result_op = tf.matmul(int_coeff_tensor, tf.transpose(u))
    modal_result = sess.run(tf.transpose(result_op))
    modal_result.flatten()
    mean_tensor.eval()
    result_op = tf.add_n([modal_result[:,0], mean_tensor])
    result = sess.run(result_op)
  U_pred = result[:23104]
  V_pred = result[23104:46208]
  P_pred = result[46208:]
```

B. 나. Example Results

Figure 3-2 shows results of predicted pressure field at arbitrary Mach numbers and angle of attack which is untrained. It is about FOM CFD result, the prediction result using POD-GPR, and relative error contour between this two as described in

preceeding order. Average relative error of testing data is estimated to be approximately 0.03%.

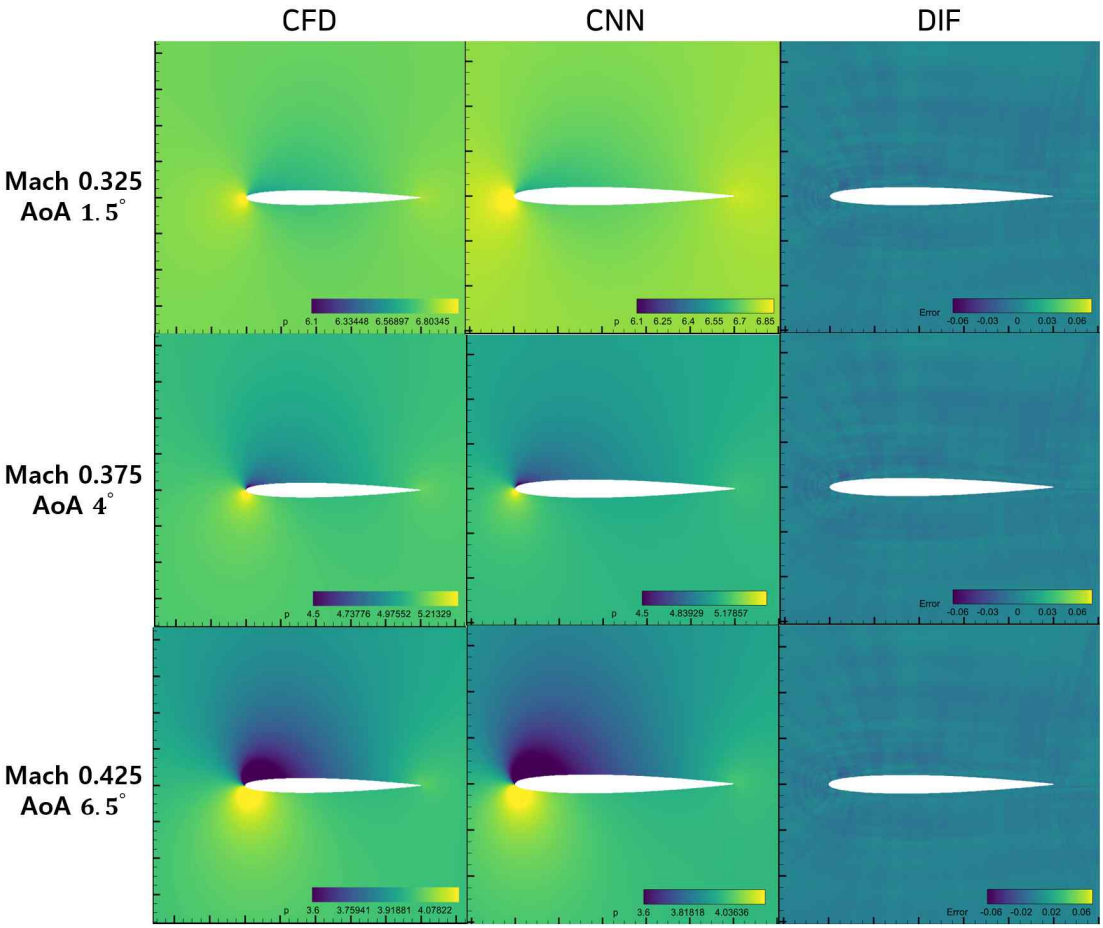


Figure 3-2. Presure Field Prediction Result Using POD-GPR and Error Contour

III. Conditional U-Net Reduced Model on the basis of CNN

In this section predicts flow field of airfoil of which flow conditions changes by using conditional U-Net. In chapter 2 includes condition such as time into battle neck layer, but this section adds Mach number and angle of attack as condition because steady flow under diverse flow condition is dealt with.

A. Snapshot Data Preprocessing and Extraction (*ExtracData_FC_cuttail.py*)

Unncessary far-field area is deleted and reduced into 360x64 by using cuttail and glyer variables described in chapter 2. Flow field data is loaded by Mach numbers and angle of attack for loading CFD data which is previously analyzed.

```
Mach =['1', '2', '3', '4', '5', '6', '7', '8', '9']
Mach =np.array(Mach)
AoA = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12',
       '13', '14', '15']
AoA =np.array(AoA)
NMa = 4
NAoA = 5
noCol = 11
numd = 123
zone1_i = 401
zone1_j = 81
cuttail = 20   # = crop_i
glayer = 64   # = crop_j
filenames = []
merged = []
Ntime = 0
for i in (Mach):
  for j in (AoA):
    filenames.append(sim_data_path +"flo_" + str(i) +"_" + str(j) + ".dat")
  Ntime += 1
print (Ntime, filenames)
```

Below is a code for decomposing learning data and testing data from total 135 data. Total 12 of testing data are used. Flow condition is as explained in data generation section. In code below, learning and testing data are classified depending on Mach number and angle of attack and saved in Traindata and Testdata variable since testing data is defined ahead of time.

```
if k == 0:
    Traindata = np.zeros((123,snapshot_data.shape[0], snapshot_data.shape[1]))
    Testdata = np.zeros((12,snapshot_data.shape[0], snapshot_data.shape[1]))
    Trainlabel = np.zeros((123, 2))
    Testlabel = np.zeros((12, 2))
if  Testname == file:
    print(Testname)
    Testdata [kk,:,:] = snapshot_data
    kk += 1
    Nat +=5
    if kk%3 == 0:
        Nmt +=2
        Nat = 3
    Testlabel[kk-1, 0] = Nm
    Testlabel[kk-1, 1] = Na
else:
    Traindata[k,:,:] = snapshot_data
    k += 1
```

```
        Trainlabel[k-1, 0] = Nm
        Trainlabel[k-1, 1] = Na
    if Na%15 == 0:
        Nm +=1
        Na = 1
    Na += 1
```

Data decomposition for learning and evaluation is done as described in the code below and separately saved under different compressed file.

```
    for i in range (2):
        if i == 0 :
        array_data = Traindata
        save_path = res_data_path +"Staedy_airfoil_cuttail_train.npz"

        if i == 1 :
        array_data = Testdata
        save_path = res_data_path +"Staedy_airfoil_cuttail_test.npz"
```

B. Creation and Learning of Network Model (*Unet_UIUC_FC.py*)

These are hyperparameters affecting learning.

```
    # Model parameters
    BATCH_SIZE = 123          # 학습 데이터 수
    validation_split = 0.1    # 학습 데이터 중에서 validation 데이터 비율
    EPOCHS = 30000            # 학습 횟수
    LATENT_DEPTH = 64         # Bottle neck layer의 크기
    NB_CHANNELS = 2           # Condition 채널의 크기
    LR = 1e-3                 # Learning rate
```

Below is a code retrieving learning data. x, y grid and corresponding flow field data are used. Following is a task to bring testing data.

```
    #read xy-coordinates
    saved_npz = np .load (res_data_path +"Staedy_airfoil_cuttail_train.npz")
    XC_star = saved_npz ['XC']
    YC_star = saved_npz ['YC']
    xydata = np .hstack ((XC_star [:,:][:,None ], YC_star [:,:][:,None ]))
    #Train data extraction
    UC_star = saved_npz ['UC']
    VC_star = saved_npz ['VC']
    PC_star = saved_npz ['PC']
    #Exact data extraction
    saved_npz_e = np .load (res_data_path +"Staedy_airfoil_cuttail_test.npz")
    XT_star = saved_npz_e ['XC']
    YT_star = saved_npz_e ['YC']
    UT_star = saved_npz_e ['UC']
    VT_star = saved_npz_e ['VC']
    PT_star = saved_npz_e ['PC']
```

In Bottle neck layer, output of encoder goes through flattening into one vector. This

vector can add vector of Mach number and angle of attack using concatenate function. Vector which added flow condition changes into the form which is usable as deconder input once again through fully connected layer.

```
    # bottle neck layer
    flat1 = layers.Flatten()(mp2 )
    dense1 = layers.Dense(LATENT_DEPTH , activation ='elu')(flat1)
    Mach = tf .keras.Input(shape =(1 ,)) # 마하수
    AoA = tf .keras.Input(shape =(1 ,))  # 받음각
    dense2 = layers.Concatenate(axis =1 )([Mach, AoA, dense1])
    dense3 = layers.Dense(LATENT_DEPTH+1, activation ='elu')(dense2)
    dense4 = layers.Dense(LATENT_DEPTH, activation ='elu')(dense3)
    dense5 = layers.Dense(n_ch*2*(zone1_i-2*cuttail-1)/(pooling_size**2)*
           (glayer)/(pooling_size**2), activation ='elu')(dense4)
    output_f = layers.Reshape((int((glayer)/(pooling_size**2)), int((zone1_i-2
                *cuttail-1)/(pooling_size**2)), n_ch*2))(dense5)
```

C. Example Result

Figure 3-3 shows predicted results of pressure field at arbitrary Mach numbers and angle of attack which is untrained. It is about FOM CFD result, the prediction result using POD-GPR, and relative error contour between this two as described earlier. Average relative error of testing data is estimated to be approximately 0.1%.

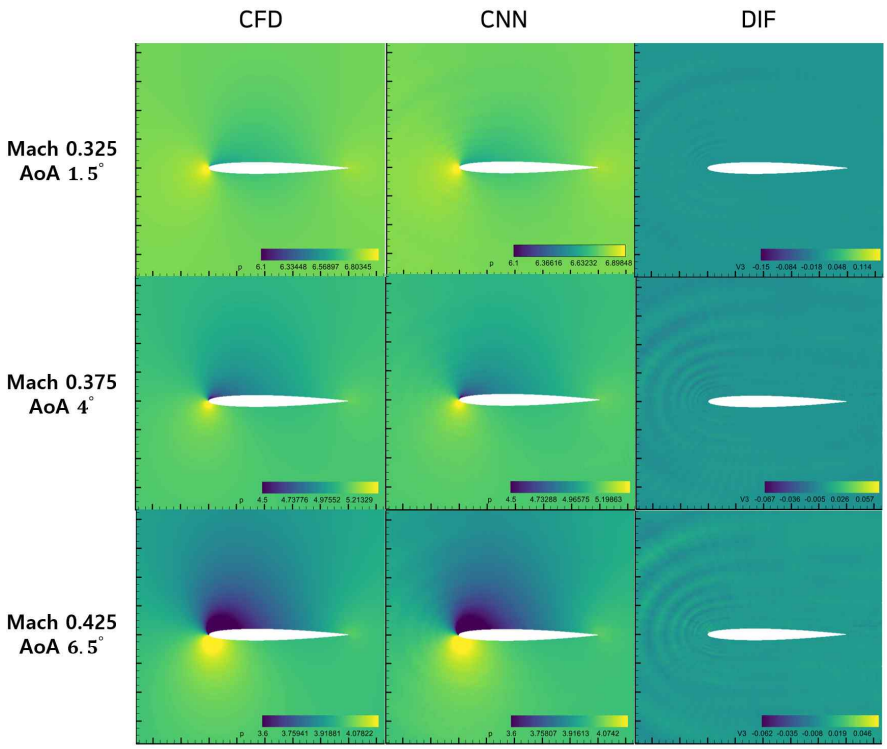


Figure 3-3. Presure Field Prediction Result Using CNN U-Net and Error Contour

Chapter 4 - Case 3) Steady Flow in Variable Airfoil Shapes and Variable Flow Conditions

Steady Cases with Variable Airfoil Shape and Variable Flow Condition
AI methods : U-Net, Conditional U-Net

I. Learning Data Formation

In chapter 4 talks about steady flow analysis under diverse shape and flow condition by using a reduced model. Also CFD analysis of Full order model(FOM) is conducted for model learning. Among 1550 of shape data of UIUC Airfoil, total 500 of airfoil are used. By adjusting 26 types of angle of attack, simulation of each shape is conducted. Angle of attack is in range of 1~16°. From 1~10°, angle of attack is divided in an interval of 0.5, and from 11~16°, it is divided in interval of 1. Figure 4-1 below is UIUC airfoil's grid which is used for CFD analysis.

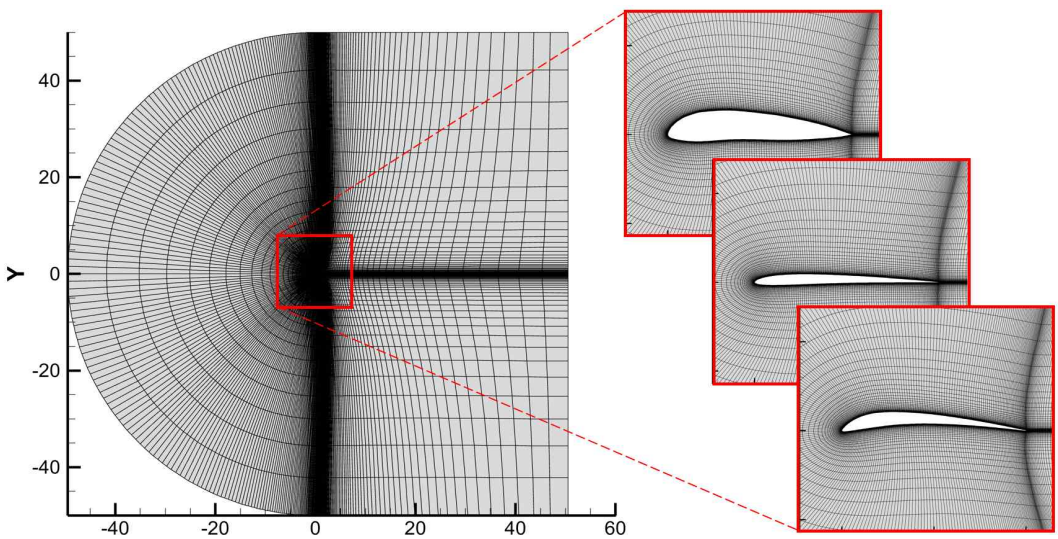


Figure 4-1. UIUC Airfoil Gird used for Data Formation

Figure 4-1 below summarizes CFD analysis condition for generating learning data.

Table 4-1. CFD Anlaysis Condition

Solver	Airfoil	Grid	Flow condition	Governing equation	Flux function
KFLOW	NACA0012	401x81 (C-type)	Mach 0.4 AoA 1~16°	RANS S-A model	Roe flux 3rd order

Based on previously described adjustment, total 13,000 of learning data are extracted. Out of 12,900 data which is used for learning, 10% is used for validation while conducting model learning. Remaining 100 of the data is for as testing data to evaluate model performance after learning is completed. Testing data is comprised of flow field data as a result of analyzing of angle of attack in 3°,6°,9°,12°,15° among 20 randomly chosed shape. In this section, only applying a reduced model using Conditional U-Net which can best track shape sensitivity than any others is dealt due to situation which data complexity is relatively high and considering both shape and flow condition at the same time. Also shape parametric techniques such as Free-from deformation(FFD), Hicks-Henne bump function or others can be applied to consider diverse shape other than learning data used in the example. In addition, flow condition other than angle of attack such as Reynolds number and Mach number can be considered that wide range of application is possible.

II. U-Net Reduced Model in the basis of CNN

In this section, cases of U-Net applied to steady flow prediction regarding fixed flow condition and diverse shape are introduced before U-Net is applied to steady flow under diverse shape and flow condition. Flow field data that is used is calculated at Mach number 0.5 and angle of attack 1°. Vanilla U-Net model of which bottle neck layer is not included in condition is used because talking about fixed flow condition even though this section is in the basis of Chapter 2 section 4.

A. Preprocessing and Extraction of Snapshot Data (*ExtracData_Shape_cuttail.py*)

Only areas of interest excluding far-field among snapshot data are extracted using crop_i and crop_j for data preprocessing. Among 1550 airfoil data, 1070 data is sorted

that 1050 of them are used for learning data and remaning 20 of them is used for testing data.

```
zone1_i = 401
zone1_j = 81
crop_i = 20
crop_j = 64
```

The code below is to delete unnecessary data depending on variable setting which is specified earlier.

```
#### Slicing to get cropped airfoils
idx_x_slice = np.array([])
for i in range(crop_j):
    idx_x_slice = np.append(idx_x_slice, np.arange(crop_i+i*zone1_i,
        (zone1_i-crop_i)+i*zone1_i)).astype('int32')
XC_star = np.transpose(XC[:,:,idx_x_slice], (0,1,2))
YC_star = np.transpose(YC[:,:,idx_x_slice], (0,1,2))
CC_star = np.transpose(CC[:,:,idx_x_slice], (0,1,2))
DC_star = np.transpose(DC[:,:,idx_x_slice], (0,1,2))
UC_star = np.transpose(UC[:,:,idx_x_slice], (0,1,2))
VC_star = np.transpose(VC[:,:,idx_x_slice], (0,1,2))
PC_star = np.transpose(PC[:,:,idx_x_slice], (0,1,2))
```

Predicting diverse shape requires relatively many number of snapshot data. So that instead of simple loop, processes-based parallel processing method is used when converting data into vector form. The code below converts flow field information of diverse shape into vector form using pool class of multi-processing module and cpu_count function.

```
def snapshot_dataset(i,j):
    pd_data1 = pd.read_csv(foilcases[i*Ncon+j], na_filter=True,
        dtype='float64', delimiter=' ', skipinitialspace=True, skiprows=2,
        header=None)
    data = np.nan_to_num(pd_data1.values)
    array_data = data.flatten()
    array_data = array_data.reshape(-1,noCol)
    snapshot_data = array_data
    return snapshot_data
### Distributed Session
num_cores = cpu_count()
pool = Pool(num_cores)
snapshot_dataset = np.concatenate(pool.starmap(snapshot_dataset,
product(range(Nfoil), range(Ncon))))
pool.close()
pool.join()
```

B. Formation of Network Model and Learning (*Unet_UIUC_Shape.py*)

Below are hyperparmeters affecting network model learning. It is similar to Conditional U-Net but doesn't use LATENT_DPETH variable because bottle neck layer is not necessary in fixed flow condition.

```
# Model parameters
BATCH_SIZE = 1050   # 학습 데이터 개수
validation_split = 0.1 # 학습 데이터 중에서 validation 데이터 비율
EPOCHS = 6000       # 학습 횟수
NB_CHANNELS = 1     # Condition Channel의 크기
LR = 1e-3           # Learning rate
IMAGE_SHAPE = [zone1_i, zone1_j]
```

After setting of hyperparameter, distinction between shape vector calculation for cost function calculation and input/output data uses completely same code as Conditional U-Net. Also a process of defining model is similar to Conditional U-Net. However, there is a difference in this section like a concatenate process of vector under flow condition in battle neck layer is skipped since it is unnecessary. Then snapshot data is predictable by decoding vector right away which is inputed from encoder. Below is the code creating a model by connecting immediately encoder and decoder. After creating a model, definition of cost function and U-Net model learning are conducted same as described in chapter 2.

```
# Encoder
input_e = tf.keras.Input(shape=(glayer, zone1_i-2*cuttail-1, 2))
pooling_size = 2
n_ch = 12
conv1 = layers.Conv2D(n_ch, (3,3), activation='elu', padding = 'same')(input_e)
mp1 = layers.MaxPooling2D((pooling_size,pooling_size))(conv1)
conv2 = layers.Conv2D(n_ch*2, (3,3), activation='elu', padding = 'same')(mp1)
mp2 = layers.MaxPooling2D((pooling_size,pooling_size))(conv2)
conv3 = layers.Conv2D(n_ch*4, (3,3), activation='elu', padding = 'same')(mp2)
mp3 = layers.MaxPooling2D((pooling_size,pooling_size))(conv3)
output_e = layers.Conv2D(n_ch*8, (3,3), activation='elu', padding = 'same')(mp3)

# Decoder
convt1 = layers.Conv2DTranspose(n_ch*4, (3,3), activation='elu', padding='same')(output_e)
upsamp1 = layers.UpSampling2D((pooling_size,pooling_size))(convt1)
skipcon1 = layers.Concatenate(axis=3)([conv3, upsamp1])
conv6 = layers.Conv2D(n_ch*4, (3,3), activation = 'elu', padding='same')(skipcon1)
convt2 = layers.Conv2DTranspose(n_ch*2, (3,3), activation='elu', padding='same')(conv6)
upsamp2 = layers.UpSampling2D((pooling_size,pooling_size))(convt2)
skipcon2 = layers.Concatenate(axis=3)([conv2, upsamp2])
conv7 = layers.Conv2D(n_ch*2, (3,3), activation = 'elu', padding='same')(skipcon2)
convt3 = layers.Conv2DTranspose(n_ch*1, (3,3), activation='elu', padding='same')(conv7)
```

```
upsamp3 = layers.UpSampling2D((pooling_size,pooling_size))(convt3)
skipcon3 = layers.Concatenate(axis=3)([conv1, upsamp3])
conv8 = layers.Conv2D(n_ch*1, (3,3), activation='elu', padding='same')(skipcon3)
```

C. Example Result

Figure 4-2 is contouring pressure field of 3 random airfoil among 20 shapes unused in learning model. Relative error for shape 1 is estimated to be 0.28%, for shape 2 is 0.15%, for shape 3 is 0.19%.

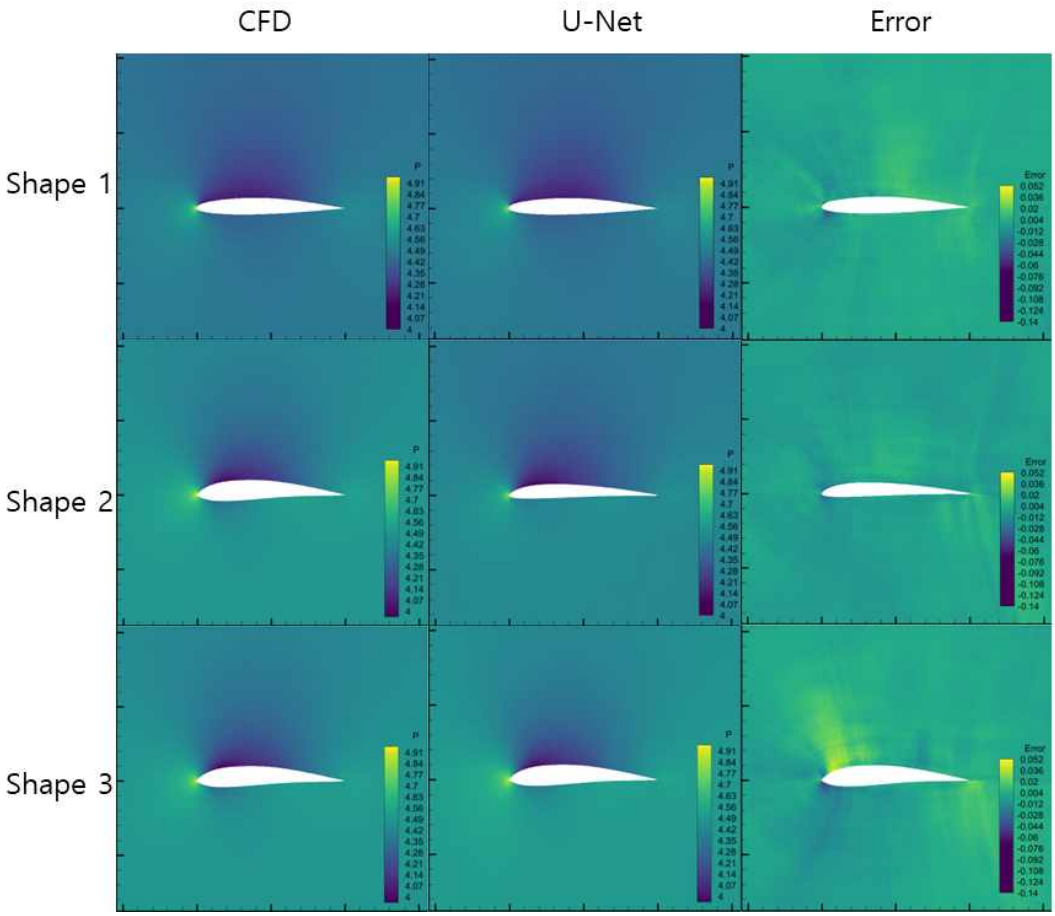


Figure 4-2. Pressure Field Prediction Result Using CNN U-Net and Error Contour

III. Conditional U-Net Reduced Model on the basis of CNN.

In this section introduces API tutorial of reduced modeling technique which can consider both diverse shape and flow condition. To predict flow field, Conditional U-Net on the basis of CNN is used. Used as an example the shape information of

UIUC airfoil is used for input value of a model. The flow condition is about outputing flow field which reflected both shape and flow condition in case of decoding by adding low dimensional vector of battle neck layer. The contents below is based on chapter 2 and chapter 3, and in this section only talks about additional information.

A. Preprocessing and Extraction of Snapshot data (*ExtracData_Shape_FC_cuttail.py*)

More data is complex, more amount of data to learn, larger burden of memory become. The example data of this section has the large capacity of 60GB, and learning is impossible in computer in general if it does not undergo preprocessing unless separate workstation and server is equipped for learning. Therefore, data preprocessing before learning is very much important, and is throughly demonstrated in chapter 2.

The code below is a process of selecting random 20 out of 500 airfoil by using random function for splitting testing data among total number of data. File name of selected airfoil is separately saved and used to classify data learning data and testing data with setted angle of attack.

```
# Test file names
shapet = []
testfilenames = []
testlist = np.zeros((Ntest,2))
for i in range (Nshapet):
   a = random.randint(1,Nshape)
   while a in shapet:
       a = random.randint(1,Nshape)
   shapet.append(a)
shapet.sort()

jj = 0
for i in (shapet):
    for j in (AoAt):
        testfilenames.append("../database/Airfoil_"+str (i ).rjust (4 ,'0')+
                             "/result_1_1_" +str (j ).rjust (2 ,'0')+ "/flo001.dat")
        testlist[jj,:] = np.array([i,j])
        jj += 1
```

The code below loads data and saves after splitting learning and testing data depending on airfoil and angle of attack which are selected earlier. Also data is loaded and deletes far-field area and remaining flow information excluding x, y, u, v, p at the same time for preprocessing efficacy. Lastly, angle of attack of learning and testing

data are separately saved and used as condition in case of model learning. After that the data completed preprocessing is saved as binary compressed file with relatively small capacity. At this point of capacity differs depending of preprocessing setting, but can reduces to 5GB.

```
for file in filenames :
   snapshot_data = []
   pd_data = pd.read_csv(file, dtype ='float32', delimiter =' ',
                         skipinitialspace =True, skiprows =2, header =None)
   pd_data = pd_data.drop([2 ,3 ,6 ,8 ,9 ,10], axis =1)
   pd_data = pd_data.loc [idx_x_slice,:]
   snapshot_data = pd_data
   if k == 0 :
     Traindata = []
     Testdata = []
     trainlab = np.zeros(Ntotal-Ntest)
     testlab = np.zeros(Ntest)
   if  file in testfilenames :
     print(file, testlab[kkk])
     Testdata.append(snapshot_data)
     testlab[kkk] = kk
     kkk += 1
   else :
     Traindata.append(snapshot_data)
     trainlab[k] = kk
     print(k+1, trainlab[k])
     k += 1
   if kk%26 == 0 :
     kk = 0
   kk += 1
pd.DataFrame(trainlab).astype('int').to_csv(res_data_path + 'TrainLabel.dat'.format (),
                                            header =False, index =False)
pd.DataFrame(testlab).astype('int').to_csv(res_data_path + 'TestLabel.dat'.format (),
                                           header =False , index =False )
```

B. Creating and Learning Network Model (*Unet_UIUC_Shape_FC.py*)

Below are codes for loading learning data, testing data, angle of attack which are saved separately.

```
#Train data extraction
saved_npz = np.load(res_data_path +"Staedy_airfoil_cuttail_train.npz")
XC_star = saved_npz['XC']
YC_star = saved_npz['YC']
xydata = np.hstack((XC_star [:,:][:,None , YC_star [:,:][:,None]))
UC_star = saved_npz['UC']
VC_star = saved_npz['VC']
PC_star = saved_npz['PC']

#Test data extraction
saved_npz_e = np.load (res_data_path +"Staedy_airfoil_cuttail_test.npz")
XT_star = saved_npz_e ['XC']
YT_star = saved_npz_e ['YC']
```

```
UT_star = saved_npz_e ['UC']
VT_star = saved_npz_e ['VC']
PT_star = saved_npz_e ['PC']

 #Load AoA label
 trainlab = pd.read_csv (res_data_path +"TrainLabel.dat", dtype ='float32', delimiter =',',
                         skipinitialspace =True , header =None)
 testlab = pd.read_csv (res_data_path +"TestLabel.dat", dtype ='float32', delimiter =',',
                         skipinitialspace =True , header =None)
 train_ipt = np.zeros (numd)
 test_ipt = np.zeros (numT)
 train_ipt = np.array (trainlab)[:,0 ]
 test_ipt = np.array (testlab)[:,0 ]
```

Unlike previously demonstrated chapters, this section talks bout learning after making min-batch and then splitting total number of data instead of conducting learning process of total number of data at once. It is to reduce memory burden in middle of learning process. The datalaoder function below splits total number of data into setted up batch size. Then it prepares next min-batch ahead before completing mini-batch learning by using prefetch function, and leads to promoting learning efficacy. bn variable divides total number of data into batch size, and decides a number of mini batch to create. Following information of generating model and its learning refers to previous chapters.

```
@tf.function
def dataloader(paths):
  dataset = tf.data.Dataset.from_tensor_slices(paths)
  dataset = dataset.batch(BATCH_SIZE)
  dataset = dataset.prefetch(1)
  return dataset

datasetI = dataloader(Input_field)
train_inp = []
for batch in datasetI.take(bn):
  for i ,img_inp in enumerate(batch):
    img_inp_np = img_inp.numpy()
    train_inp.insert(i, img_inp_np)
train_inp = np.array(train_inp, dtype ="float32")
```

C. Example Results

In Figure 4-3 is coutouring pressure field at angle of attach in each of 3°, 6°, 9° regarding 3 types of airfoil among 20 random airfoil which is used for testing data. Average relative error of total 100 testing data is 0.28% that shows flow field prediction performance of which reduced model is high enough at untrained shape and angle of attack.

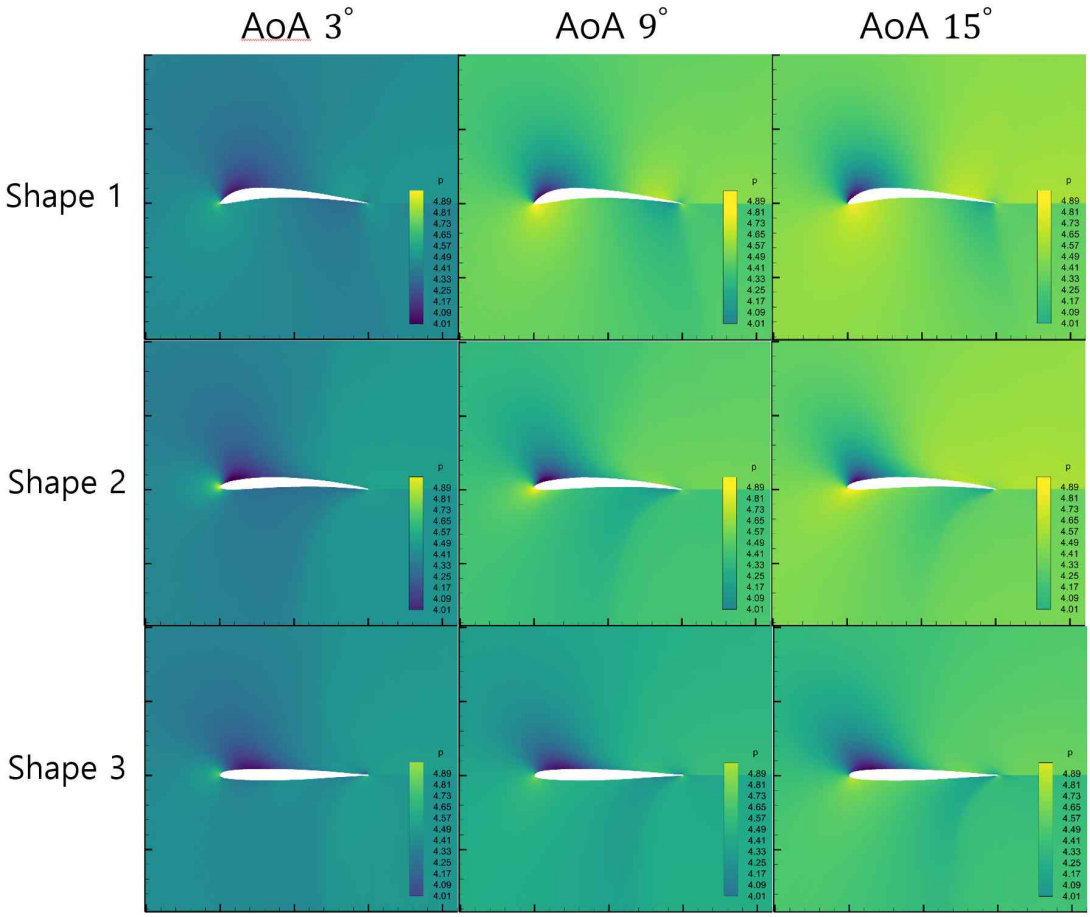


Figure 4-3. Pressure Field Prediction Result at Variable Shapes and Angle of Attack Using Conditional U-Net

Appendix

1. Brunton, S. L., Noack, B. R., and Koumoutsakos, P., 2020, “Machine learning for fluid mechanics,” Annual Review of Fluid Mechanics, Vol. 52, pp. 477~508.
2. Kutz, J. N., 2013, Data-driven modeling & scientific computation: methods for complex systems & big data, Oxford University Press.
3. Holmes, P., Lumley, J. L., Berkooz, G., and Rowley, C. W., 2012, Turbulence, coherent structures, dynamical systems and symmetry. Cambridge University Press.
4. Hartmann, D., Herz, M., & Wever, U., 2018, Model order reduction a key technology for digital twins: In Reduced order modeling for simulation and optimization, Springer, pp. 167~179.
5. Taira, K. et al., 2017, “Modal analysis of fluid flows: An overview,” AIAA Journal, Vol. 55, No. 12, pp. 4013~4041.
6. Taira, K. et al., 2020, “Modal analysis of fluid flows: Applications and outlook,” AIAA journal, Vol. 58, No. 3, pp. 998~1022.
7. Rowley, C. W., Colonius, T., and Murray, R. M., 2004, “Model reduction for compressible flows using POD and Galerkin projection,” Physica D. Nonlinear Phenomena. Vol. 189, No. 1-2, pp. 115-129.
8. Hesthaven, J. S., and Ubbiali, S., 2018, “Non-intrusive reduced order modeling of nonlinear problems using neural networks,” Journal of Computational Physics, Vol. 363, pp. 55~78.
9. Suram, S. 2016, Strategies for including cloud-computing into an engineering modeling workflow, Doctoral dissertation, Iowa State University.
10. http://turbgate.engin.umich.edu/
11. http://cddcp.sites.uofmhosting.net/
12. https://web.stanford.edu/group/frg
13. http://kiwi.ices.utexas.edu/index.php
14. http://www.georgehaller.com/
15. Ghavamian, F., Tiso, P., and Simone, A., "POD-DEIM Model Order Reduction for Strain-softening Viscoplasticity," Computer Methods in Applied Mechanics and Engineering, Vol. 317, 2017, pp. 458-479.
16. Carlberg, K., Bou-Mosleh, C. and Farhat, C., "Efficient non-linear model reduction via a least-squares Petrov -Galerkin projection and compressive tensor approximations," International Journal for Numerical Methods in Engineering, Vol. 86, No. 2, 2011, pp. 155-181.
17. Choi, Y., Coombs, D., and Anderson, R., "SNS: A Solution-based Nonlinear Subspace method for time-dependent nonlinear model order reduction," arXiv:1809.04064, 2018.
18. Xiao, D., Yang, P., Fang, F., Xiang, J., Pain, C. C., Navon, I. M., and Chen, M., "A Non-intrusive Reduced-order Model for Compressible Fluid and Fractured Solid Coupling and Its Application to Blasting," Journal of Computational Physics, Vol. 330, 2017, pp. 221-244.
19. Kim, E. and Cho, M., "Design of a Planar Multibody Dynamic System with ANCF Beam Elements Based on an Element-wise Stiffness Evaluation Procedure," Structural and Multidisciplinary Optimization, Vol. 58, No. 3, 2018, pp. 1059-1107.
20. Park, K. H., Jun, S. O., Baek, S. M., Cho, M. H., Yee, K. J., & Lee, D. H., “Reduced-order model with an artificial neural network for aerostructural design optimization,” Journal of Aircraft, Vol. 50, No. 4, 2013, pp. 1106-1116.
21. Kim, Y., Cho, H., Park, S., Kim, H., and Shin, S.-J., "Advanced Structural Analysis Based on Reduced Order Modeling for Gas Turbine Blade," AIAA Journal, Vol. 56, No. 8, 2018, pp. 3369-3373.
22. Lee, J. and Cho, M., "Efficient design optimization strategy for structural dynamic systems using a reduced

basis method combined with an equivalent static load," Structural and Multidisciplinary Optimization, Vol. 58, No. 4, 2018, pp. 1489-1504.
23. Sirovich, L., "Turbulence and the Dynamics of Coherent Structures. Part 1: Coherent Structures," Quarterly of Applied Mathematics, Vol. 45, No. 3, 1987, pp. 561-571.
24. Bui-Thanh, T., Damodaran, M. and Willcox, K., "Aerodynamic Data Reconstruction and Inverse Design Using Proper Orthogonal Decomposition," AIAA Journal, Vol. 42, No. 8, 2004, pp. 1505-1516.
25. Sirvoich, L., and Kirby, M., "Low-Dimensional Procedure for the Characterization of Human Face," Journal of the Optical Society fo America A, Vol. 4, No. 3, 1987, pp. 519-524.
26. Amsallem, D., Zahr, M. J., Farhat, C., "Nonlinear Model Order Reduction Based on Local Reduced-order Bases," International Journal for Numerical Methods in Engineering, Vol. 92, 2012, pp. 891-916.
27. Galletti,B., Bottaro, A., Bruneau, C.-H., and Iollo, A., "Accurate model reduction of transient flows and forced wakes," European Journal of Mechanics B/Fluids, Vol. 26, No. 3, 2007, pp. 354-366.
28. Rowley, C. W., Colonius, T., and Basu, A. J., "On Self-sustained Oscillations in Two-dimensional Compressible Flow over Rectangular Cavities," Journal of Fluid Mechanics, Vol. 455, 2002, pp. 315-346.
29. Amsallem, D., and Farhat, C., "An Interpolation Method for Adapting Reduced-order Models and Application to Aeroelasticity," AIAA Journal, Vol. 46, 2008, pp. 1803-1813.
30. Chaturantabut, S., Sorensen, D.C., "Nonlinear model reduction via discrete empirical interpolation," SIAM Journal on Scientific Computing, Vol. 32, 2010, pp. 2737-2764.
31. Carlberg, K., Farhat, C., Cortial, J. and Amsallem, D., "The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows," Journal of Computational Physics, Vol. 242, 2013, pp. 623-647.
32. Peherstorfer, B., Drmac, Z., and Gugercin, S., "Stabilizing Discrete Empirical Interpolation via Randomized and Deterministic Oversampling," arXiv:1808.10473, 2018.
33. Burkardt, J., Gunzburger, M., and Lee, H.-C., "POD and CVT-based reduced-order modeling of Navier-Stokes flows," Computer Methods in Applied Mechanics and Engineering, Vol. 196, No. 1-3, 2006, pp. 337-355.
34. Xiao, D., Yang, P., Fang, F., Xiang, J., Pain, C. C., and Navon, I. M., "Non-intrusive reduced order modelling of fluid-structure interactions," Computer Methods in Applied Mechanics and Engineering, Vol. 303, 2016, pp. 35-54.
35. URL: https://github.com/SpectralPOD/spod_matlab
36. Towne, A., Schmidt, O. T., & Colonius, T., 2018, "Spectral proper orthogonal decomposition and its relationship to dynamic mode decomposition and resolvent analysis," Journal of Fluid Mechanics, Vol. 847, pp. 821~867.
37. Schmidt, O. T., & Towne, A., 2019, "An efficient streaming algorithm for spectral proper orthogonal decomposition," Computer Physics Communications, Vol. 237, pp. 98~109.
38. Schmidt, O. T., Towne, A., Colonius, T., Cavalieri, A. V., Jordan, P. and Brès, G. A., 2017, "Wavepackets and trapped acoustic modes in a turbulent jet: coherent structure eduction and global stability," Journal of Fluid Mechanics, Vol. 825, pp. 1153~1181.
39. URL: https://github.com/mengaldo/PySPOD
40. Choi, Y., Arrighi, W. J., Copeland, D. M., Anderson, R. W. and Oxberry, G. M., 2019, libROM, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States).
41. URL: https://github.com/mendezVKI/MODULO
42. Mendez, M. A., Balabane, M., and Buchlin, J. M., 2019, "Multi-scale proper orthogonal decomposition of complex fluid flows," Journal of Fluid Mechanics, Vol. 870, pp. 988~1036.
43. Mendez, M. A., Balabane, M., and Buchlin, J. M. 2018, "Multi-scale proper orthogonal decomposition (mPOD)," AIP Conference Proceedings, Vol. 1978, No. 1, pp. 060018.

44. Esposito, C., Peveroni, L., Gouriet, J. B., Steelant, J., and Vetrano, M. R., 2021, "On the influence of thermal phenomena during cavitation through an orifice," International Journal of Heat and Mass Transfer, Vol. 164, 120481.
45. URL: https://github.com/vassilikitsios/snapshot_pod_rom_cpp
46. URL: https://github.com/vassilikitsios/snapshot_pod_rom_py
47. Kitsios, V., Cordier, L., Bonnet, J. P., Ooi, A. and Soria, J., 2011, "On the coherent structures and stability properties of a leading-edge separated aerofoil with turbulent recirculation," Journal of Fluid Mechanics, Vol. 683, pp. 395~416.
48. URL: https://github.com/BJUT-SIPL/POD-NN
49. Wang, Q., Hesthaven, J. S. and Ray, D., 2019, "Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem," Journal of computational physics, Vol. 384, pp. 289~307.
50. URL: https://github.com/stefaniafresca/POD-DL-ROM
51. Fresca, S. and Manzoni, A., 2021, "POD-DL-ROM: enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition," arXiv preprint arXiv:2101.11845.
52. https://github.com/mathLab/ITHACA-FV
53. Demo,N., Tezzele, M., Gustin, G., Lavini, G. Rozza, G., 2018, "Shape optimization by means of proper orthogonal decomposition and dynamic mode decomposition," Technology and Science for the Ships of the Future: Proceedings of NAV 2018: 19th International Conference on Ship & Maritime Research, IOS Press, pp. 212~219.
54. Taewan Kim's Blog - http://taewan.kim/post/cnn/#fnref:2
55. Stevens, Eli, Luca Antiga, and Thomas Viehmann. Deep learning with PyTorch. Manning Publications, 2020.
56. Chen, D, Gao, X, Xu, C et al. (4 more authors) (2020) FlowGAN: A Conditional Generative Adversarial Network for Flow Prediction in Various Conditions. In: 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence. 32nd International Conference on Tools with Artificial Intelligence (ICTAI 2020), 09-11 Nov 2020, Baltimore, MD, USA. IEEE . ISBN 978-1-7281-9228-4
57. An article in the Christopher Olah's blog, "Understanding LSTM Networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 27-Aug-2015.
58. Rasmussen, C. E. and Edward, Carl. "Gaussian processes in machine learning," Summer school on machine learning. Springer, Berlin, Heidelberg, 2003.
59. https://juanitorduz.github.io/gaussian_process_reg/
60. Shin, J. H. and Sa, J. H., (2022). UIUC airfoil dataset, Grid, aerodynamics, computational fluid dynamics, Simulation. doi:10.22711/idr/955
61. Hwang, W. T.. (2022). NACA0012 CFD dataset in various flow conditions. doi:10.22711/idr/962
62. https://github.com/Jameshin/AI_Airfoil_CFD
63. Shin, J. H. and Cho, K. W., Comparative study on reduced models of unsteady aerodynamics using proper orthogonal decomposition and deep neural network. Journal of Mechanical Science and Technology 36 (9) (2022) 4491~4499. http://doi.org/10.1007/s12206-022-0813-3
62. URL: http://github.com/SheffieldML/GPy
63. URL: https://github.com/lkulowski/LSTM_encoder_decoder

Appendix

Theory Explanation of Each Machine Learning & Deep Learning Method

I. Proportion Orthogonal Decomposition

Proportion orthogonal decomposition (POD) is a way of linear combination with optimal (or minimal) basis (or mode) extracted from a series of snapshot data. This chapter introduces the way of extracting minimal basis and constructing POD reduced model with proper coefficients. The flow of model order reduction is shown in Figure #1.

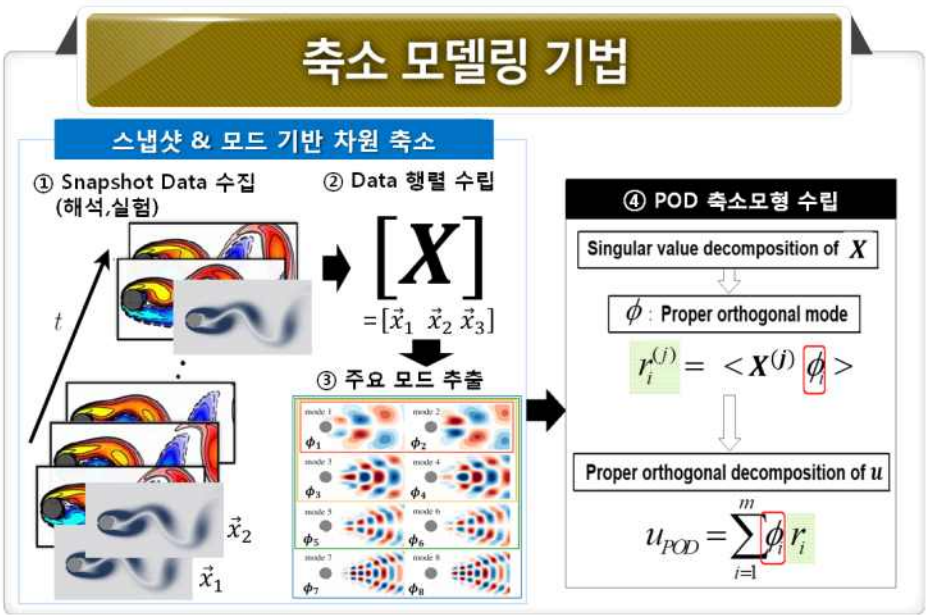


Figure #1. Overview of Reduced Modeling based on POD

A. Basic Concept

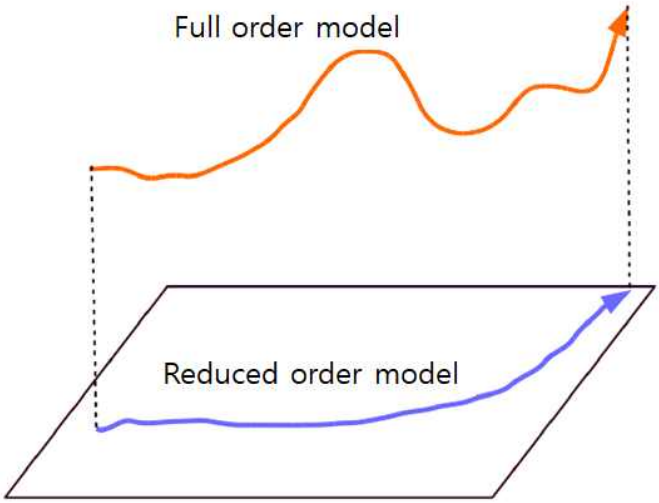


Figure #2. Concept of Order Reduction of Model

Figure #2 shows an example of order reduction from three dimensional to two dimensional. Likewise let us assume two state variables, $u_1(t)$ and $u_2(t)$, depending on time parameter. In Figure #3, we call u_1 and u_2 as snapshots which can also be regarded as orthogonal coordinate system $[x_1, x_2]$ varying in accordance with time interval $(t_1 \sim t_2)$. New coordinate system can be built to understand system better. This new coordinate system is called principal coordinate system.

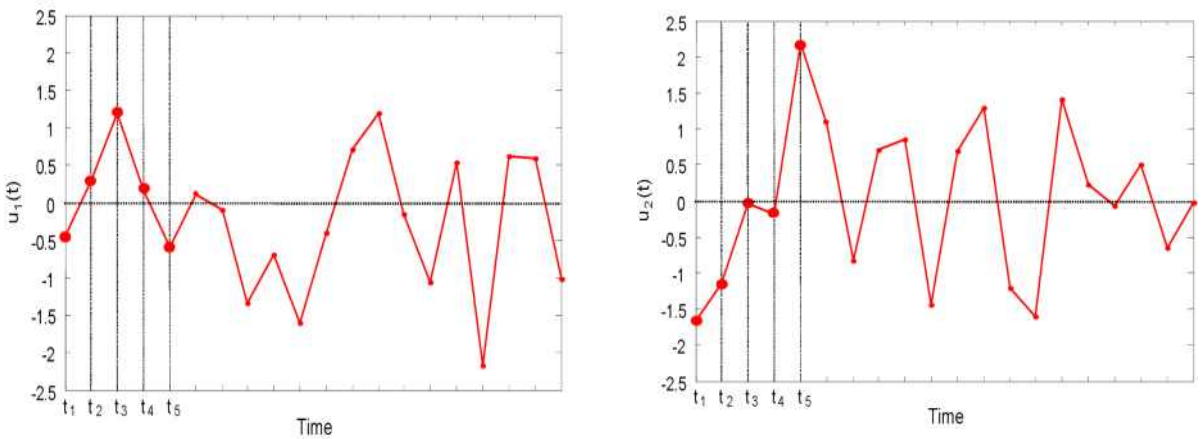
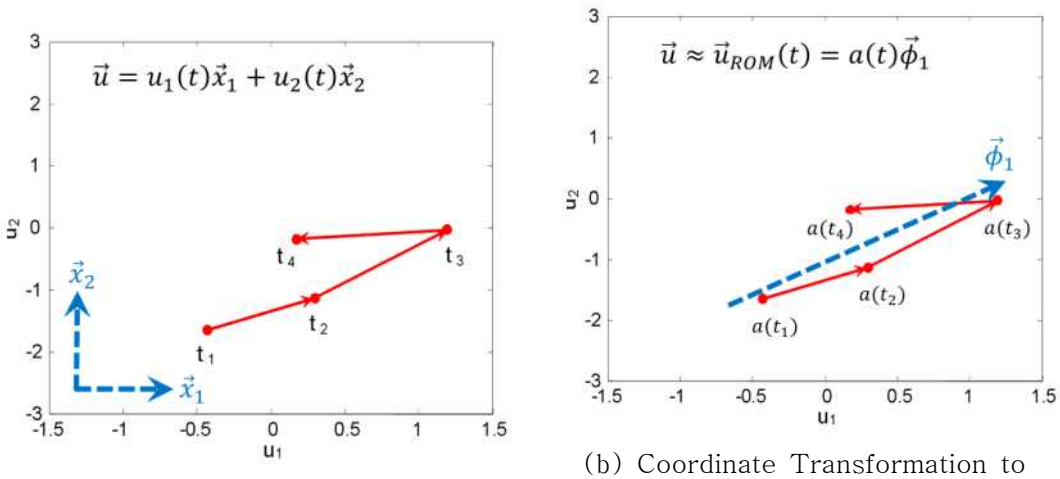


Figure #3. Example of Time Response(Snapashot) at Two Points

Figure #4 refers to the order reduction from two dimensional coordinate system into one dimension. To obtain the principal coordinate system, principal coordinate axis which can describe as maximal information of snapshots as possible must be founded ($\vec{\phi}_1$ in Figure #4). Therefore, data $\vec{u}$ can be described as reduced way such as $\vec{u}_{ROM}$ as expressed in Eq. (#1).

$$\vec{u} \approx \vec{u}_{ROM} = a(t)\vec{\phi}_1 \qquad (\#1)$$



(a) Existing Coordinates

(b) Coordinate Transformation to Reduced Basis

Figure #4. Existing Coordinate (u_1, u_2) and POD reduced Coordinate a

The coordinate axis $\vec{\phi}_1$ and coefficient $a(t)$ can be called POD basis (or mode) and POD coefficient respectively.

Expanding the above explanation into multi-dimension, let us put complete coordinates of n dimensions. If proper number of coordinate axis, such as m, best describe variation feature of original snapshots, they can be expressed with the new reduced POD coordinate system (n > m). The next section explains the extraction of principal modes, $\vec{\phi}_i$, and the calculation of POD coefficients, a_i.

B. Theoretical Description

1) Snapshot Matrix

Assume a matrix, $\boldsymbol{X}$, composed of specific column vector, $\vec{q}_i$, to implement proper orthogonal decomposition previously explained overall.

$$\boldsymbol{X}(\boldsymbol{\xi},\alpha) = [\vec{q}_1\ \vec{q}_2\ \vec{q}_3 \cdots \vec{q}_m] \tag{#2}$$

Here $\boldsymbol{\xi}$ refers to spatial coordinates and each column vector is labeled by numbering parameter, α. In POD method, as specified in basic concept, the column vectors can be described as linear superposition of m principal modes in form of Eq. (#3).

$$\vec{q}(\boldsymbol{\xi},\alpha) \approx \sum_{i}^{m} a_i(\alpha)\vec{\phi}_i(\boldsymbol{\xi}) \tag{#3}$$

In case of using reduced order model, $\vec{q}$ can be predicted at arbitrary α's which even did not exist at originally given points. Now Eq. (#2) can be depicted as Figure #5.

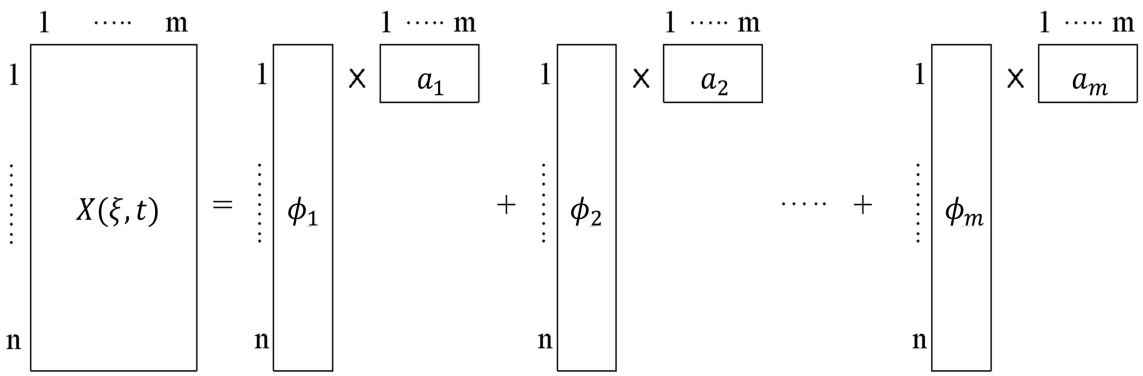


Figure #5. Linear Superposition of POD Mode

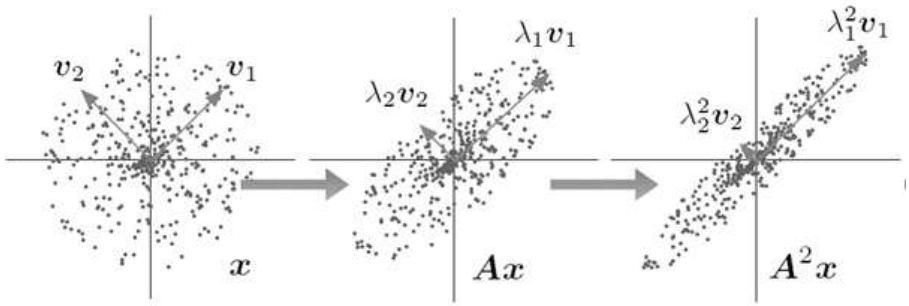
In principal, mode vectors should be described simultaneously on time and space , that is, $\phi_j(\xi,\alpha)$, but in order for better mathematical formulation, method of separation of time and space is employed in our work.

Now we need singular value decomposition (SVD) technique which is an essential mathematical method used when extracting POD basis vectors, $\phi_j(\xi)$, and POD expansion coefficents, $a_j(\alpha)$, for establishing Eq. (#3).

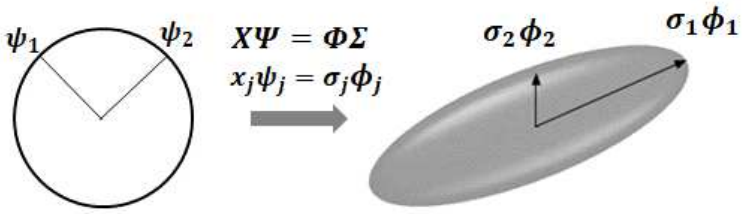
2) Extraction of Principal Basis Using SVD

(a) The Egienvalue and Singular Value Problem

Eigenvalue problem is a technique widely used among ones extracting various modes and corresponding coefficient for each mode. POD is tightly associated with the aforementioned eigen decomposition, but the decomposition technique is slightly different.



a) Eigen value problem



b) Singular value problem

Figure #6. Egien Value and Singular Value Problem

Figure #6 explains eigenvalue and singular value. First for eigenvalue, corresponding operation is done once then the points increase or decrease in certain ratio in direction specifically when assuming matix A which converts arbitrary

distributed points. Here we call the direction as eigenvector or eigenmode(v_1, v_2). The ratio is called as eigenvalue(λ_1, λ_2). Transformation matrix A is given in diverse fields such as linear system, image processing, etc. Eigenvector and eigen value can be calculated at once through eigenvalue decomposition of given system matrix A that it is very useful matrix factorization. Therefore, in summary, the singular value decomposition is more generalized technique than eigen decomposition since the basis vectors in the eigenvalue case are fixed with (v_1, v_2) but singular decomposition is operated between different couples of vectors, (ψ_1, ψ_2) and (ϕ_1, ϕ_2).

Figure 6(b) shows singular value problem which can be regarded as more generalized eigenvalue problem. Just as eigenvalue problem by viewing in perspective of direction and ratio of geometric length transformation, the direction goes from converting orthogonal coordinate vectors (ψ_1, ψ_2) to another orthogonal coordinate vectors (ϕ_1,ϕ_2). The transformation ratios are as much of (σ_1, σ_2) along each orthogonal direction. The mode vectors in the singular value problem are also orthogonal to each other. One thing to highlight is that eigenmode direction does not change in case when converting in eigenvalue problem but change in singular value problem. Also another point to focus regarding POD is system matrix is square (A in Figure #6) in eigenvalue problem, but SVD can deals with rectangular matrix (X). Therefore, complex problems with left/right singular vectors of different dimensions can be handled as shown in Figure 6(b).

(b) Basis and Singular Value Extraction

Data matrix, $\boldsymbol{X}$, of dimension $n \times m$ can be decomposed as shown below.

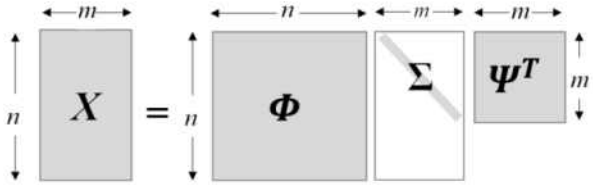
$$\boldsymbol{X} = \boldsymbol{\Phi}\boldsymbol{\Sigma}\boldsymbol{\Psi}^{\boldsymbol{T}} \in \mathrm{R}^{n \times m} \quad (\#4)$$

$$\boldsymbol{\Phi} = \left[\vec{\phi}_1\ \vec{\phi}_2 \ldots \vec{\phi}_n\right] \in \mathrm{R}^{n \times n} \quad (\#5)$$

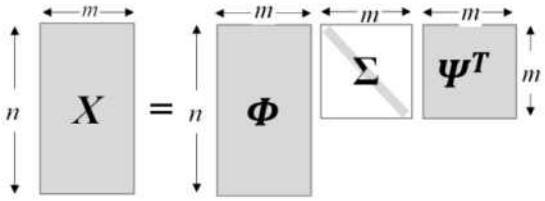
$$\boldsymbol{\Psi} = \left[\vec{\psi}_1\ \vec{\psi}_2 \ldots \vec{\psi}_m\right] \in \mathrm{R}^{m \times m} \quad (\#6)$$

Here for $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ are called right singular matrix and left singular matrix respectively. Each diagonal in diagonal matrix $\boldsymbol{\Sigma}$ ($\sigma_1 > \sigma_2 >$, ..., $> \sigma_m$) is called singular value. The diagonal matrix has a shape of rectangle, that only m column

vectors among left matrix ($\boldsymbol{\Phi}$) are actual used as modes. Therefore, original SVD form is as shown in Figure #7(a), but considering computational cost, can reduce into an economy-sized form as shown in Figure #7(b).



(a) Full-sized



(b) Economy-sized

Figure #7. Singular value decomposition

As mentioned above, left and right matrix ($\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ respectively) means principal modes, and a diagonal at the center matrix ($\boldsymbol{\Sigma}$) means sizing in direction of corresponding mode. Certainly the physical meaning of each mode and its size differs depending on data matrix($\boldsymbol{X}$).

(c) Example: Size Compression of Photo Image Data

Even though differing from the CFD topic, we briefly explains image compression which is a representative application to make better understanding of SVD. Actually POD method is almost the same as principal component analysis (PCA) of this image processing technique, and even recently deep learning techniques associated with image processing are widely applied to flow problems. Figure #8 shows an example of four different images which are compressed through SVD method. Here, let us set matrix $\boldsymbol{X}$ intensity for pixel of gray rectangular photo image. Then left and right matrixes ($\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ respectively) refer to the combination of column vectors which indicate main modes of respective horizontal and longitudinal directions of each photo images. The singular values in diagonal matrix aligns in order of the magnitudes of singular values (intensity) associated with

corresponding modes. By this way, using SVD can extract data features as limited basis and singular values, and original image can be compressed by partially selecting SVD basis and singular values and reconstructing images again by them. Consequently Figure #8 shows a degree of compression depending on the number of extracted basis and singular values (k in equal in upper and lower direction). ted basis/sigular value numbers(k numbers in equal in upper and lower direction).

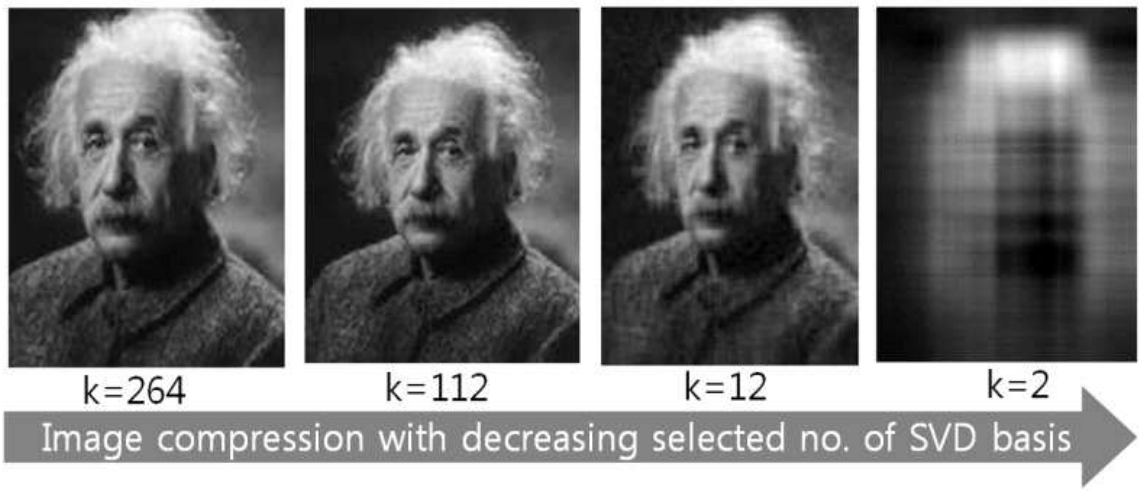


Figure #8. Image compression in accordance to the number of SVD basis (k is the number of basis)

3) Determination of POD Coefficients

Expanding the aforementioned SVD method to a reduced order model problem which has to predict column vector $\vec{q}$ in arbitrary point α, it remains to find POD coefficients a_j of Eq. (#3). There are multiple ways to obtain these coefficients at unknown point α, but can be classified into just two ways which will be explained in this section.

(a) Invasive method: a way to calculate POD coefficients by projecting basis into system matrix (governing equation)

Projection-based reduced order modeling is a method reducing existing large order problem normally requiring large computational cost into smaller order problem (probably ordinary differential equation) by projecting reduced basis

vectors into the large order governing equation which is defined in the form of a partial differential equation (Navier-Stokes equation in CFD). Engineering problems in general include nonlinear terms in governing equations so that in most cases repetitive calculation is required for getting a solution. For such a part, various approaches can be proposed.

Assuming the problem in which scalar α is a variable parameter of interest and we need obtain reduced order model as written in Eq. (#7),

$$\boldsymbol{r}(t) = \sum_{j}^{m} a_j(\alpha)\vec{\phi}_j \tag{#7}$$

And Eq. (#8) is established due to the orthogonality of reduced basis.

$$\langle \vec{\phi}_j, \vec{\phi}_k \rangle = \delta_{jk}, \quad j,k = 1, \ldots, m \tag{#8}$$

$\langle \bullet \rangle$ refers to inner product symbol and means spatially integration of Eq. (#9).

$$\langle \vec{u}, \vec{v} \rangle = \int_{\Omega} \vec{u}(\xi) \bullet \vec{v}(\xi) dV \tag{#9}$$

Now let us assume a partial differential equation for obtaining $u = u(\vec{x}, t)$,

$$\dot{u} = \boldsymbol{X}(u) \tag{#10}$$

When Eq. (#10) substitutes Eq. (#7) and basis vector ($\vec{\phi}$) is projected (or inner product), an ordinary differential equation is derived as Eq. (#11) after applying Eq. (#8). This process is commonly called as Galerkin projection.

$$\frac{d\vec{a}_j}{d\alpha} = \langle \boldsymbol{X}(r(\alpha)), \vec{\phi}_j \rangle, \quad j = 1, 2, \ldots, m \tag{#11}$$

Mostly ordinary differential equation needs less calculation cost along with simple numerical analysis method so that expansion coefficient can be obtained in faster time.

However governing equation of given problem differs depending on problems. Therefore one need to restart from first stage mathematical formulation.

(b) Non-invasive method: a method of estimating unknown coefficients by regression from the coefficient values obtained through SVD stage.

Expansion coefficient a_j pairing with POD mode $\vec{\phi}_j$ extracted through SVD can be calculated through projection or inner product as written in Eq. (#12).

$$a_j(\alpha_k) = \langle \vec{x}(\alpha_k), \vec{\phi}_j \rangle \tag{#12}$$

Various kinds of interpolation or regression (such as linear interpolation, kriging, neural network, et cetera) can be applied based on extracted expansion coefficients which are on the dimension of m (equal to the number of snapshots).

This non-invasive method is useful to make application than invasive method because coefficients can be calculated only using data matrix. In other words, governing equation of targeting problem does not have to be known in non-invasive reduced order modeling.

I. Fully-Connected Deep Neural Network

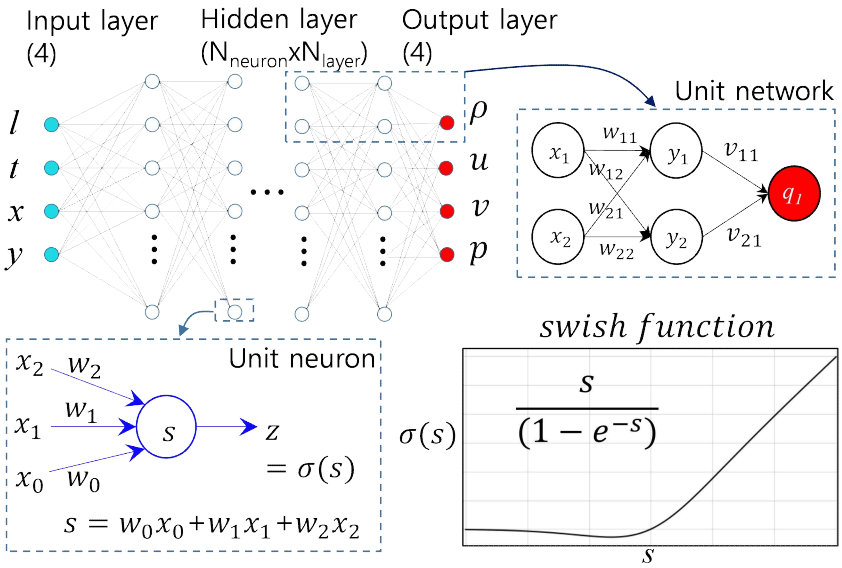


Figure #9. Example of Fully-Connected Deep Neural Network

A. Basic Concept

Fully-connected deep neural network is a universal approximator of multi-dimensional data. Figure #9 comprises three parts: One is neural network (upper left) comprised(or modeling) based on general CFD problems (or dataset). Others are unit perceptron (lower left) and unit network (upper right). Each neuron (or perceptron) has its weight, bias, and activation function, and each column layer has a finite number of neurons. In deep neural network, it's comprised of an input layer, an output layer, and hidden layers. Target goal of this network is finding all weights and bias through optimizing.

B. Implementation

Unit neuron's mathematical description in the upper right of Figure #9 is as written in Eq. (#13).

$$y_j = \sigma\left(\sum_i \left(w_{ij}x_i + b_j\right)\right) \tag{\#13}$$

$$\sigma(s) = s/(1 - e^{-s}) \tag{#14}$$

Recently in image and vision deep learning, many activation functions are being variously selected such as hyperbolic tangent function or rectified linear unit (ReLU). Meanwhile this report employs 'Swish' activation function because it looks like tackling problems which includes vanishing and fluctuation during mathematical differentiation process.

Next, Eq. (#15) shows an output variable as a way to write each flow variables with network model equations.

$$q_j = \sum_i v_{jk} y_j \tag{#15}$$

Eq. (#16) shows MSE (mean-square-error) cost function which indicates error between inferred and original data.

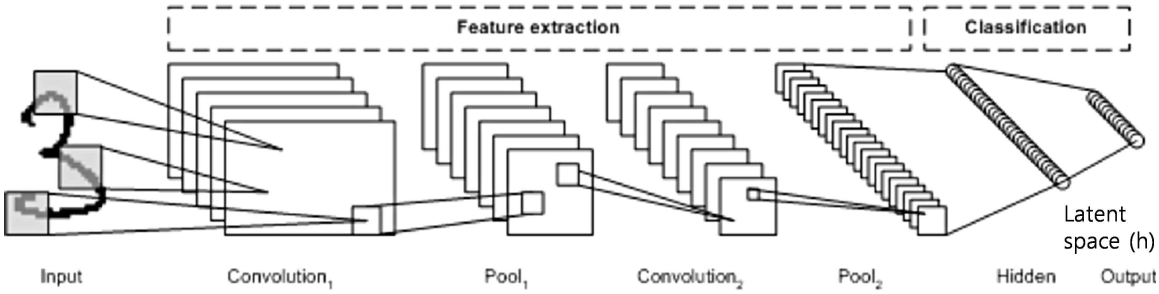
$$MSE = \frac{1}{M} \sum_{j=\rho,u,v,p} \sum_{i}^{M} \left(q_{j,infer}(t_j, x, y) - q_{j,CFD} \right)^2 \tag{#16}$$

Lastly lots of computation has to be conducted for optimizing DNN parameters of weights and biases. In this procedure, GPU and related mathematical libraries are generally used. Several open-source frameworks such as Tensorflow and PyTorch are recently developed for it. Using them, automatic differentiation which is necessary for back propagation process which is a core algorithm of various neural network and deep learning can be implemented by script coding.

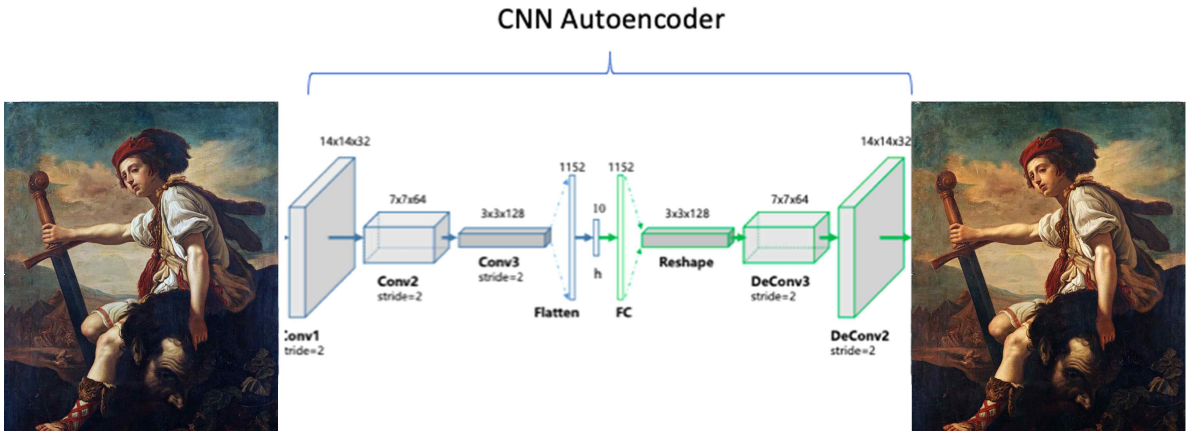
One thing to focus, it is important to select a number of layer and neuron, form of activation function, optimizer, batch/epoch size, learning rate, et cetera. These are called hyperparameters. Unfortunately it is left unestablished how to determine them appropriately.

II. Convolutional Neural Network

[Courtesy of] http://taewan.kim/post/cnn/#fnref:2



a) CNN process of handwriting image (for classification)



b) CNN Autoencoder process of image reproduction (for regression)

Figure #10. Schematic Diagram of CNN Learning Concept of Numerical Image

A. Basic Concept

Images that convolutional neural network (CNN) is applied to can be regarded as two dimensional data (called as three dimensional if RGB color channel is included). two dimensional flow field in CFD data has the similar data topology as well so that it is possible to implement identical CNN algorithm. Since previously explained input data of artificial neural network which is comprised in fully connected form is limited to one dimensional array form, in case that fully connected neural network is implemented in the image data, vectorizing (or flattening) of two dimensional image data is required. But the vectorization of the image data leads to spatial

information loss. Consequently there is inefficiency of artificial neural network for two dimensional problems due to the lacking information which occurs by loss of spatial information. CNN is a representative AI model that enables learning and inferring in state of maintaining spatial information of image.

There are many forms that exists in details in CNN. In large it can be divided into two types. One is classification (refers to Figure #10(a)) which recognizes or segments specific part from overall images. Another is regression (refers to Figure #10(b)) which includes image reproducing or resolution improving. In this report, problems which uses 'Autoencoder', a regression model of CNN, is employed. As shown in Figure #10(b), there are two network parts. One is called convolution network which is the same as classification model that extracts a single vector h as a feature and the other is called deconvolution network which generates new image from the feature vector.

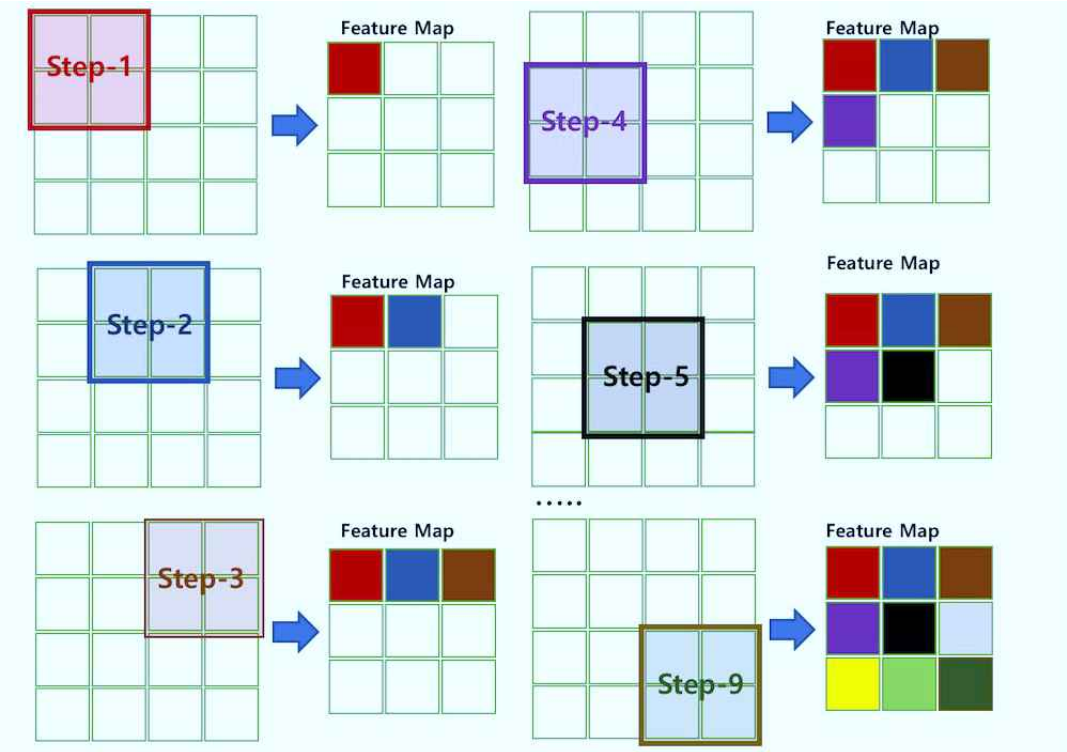
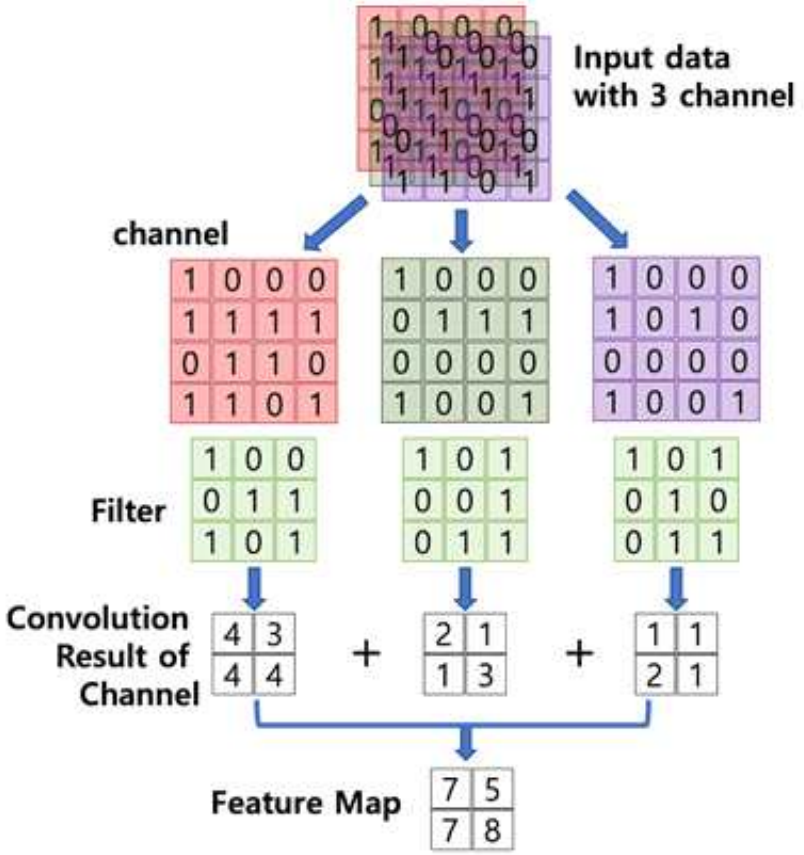
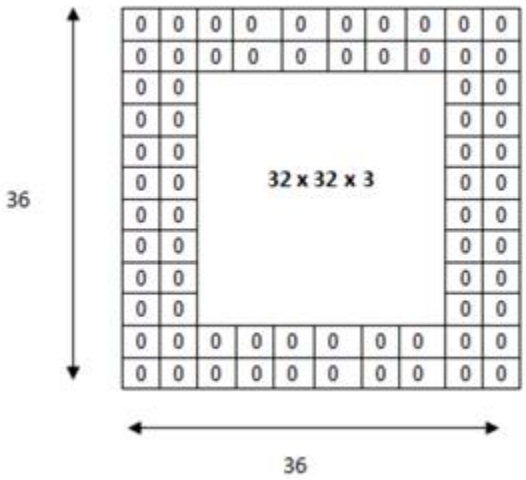


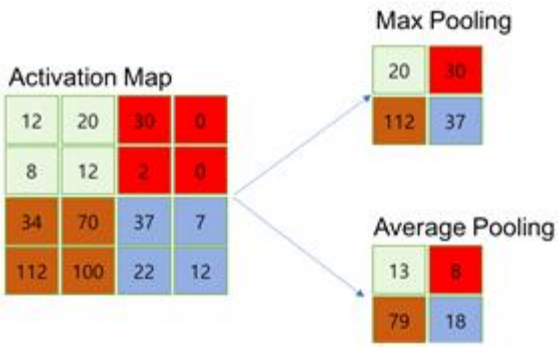
Figure #11. Extraction Process of Convolution Feature Map



a) Convolution Computation



b) Padding



c) Pooling

Figure #12. Example of CNN Computation

B. Implementation

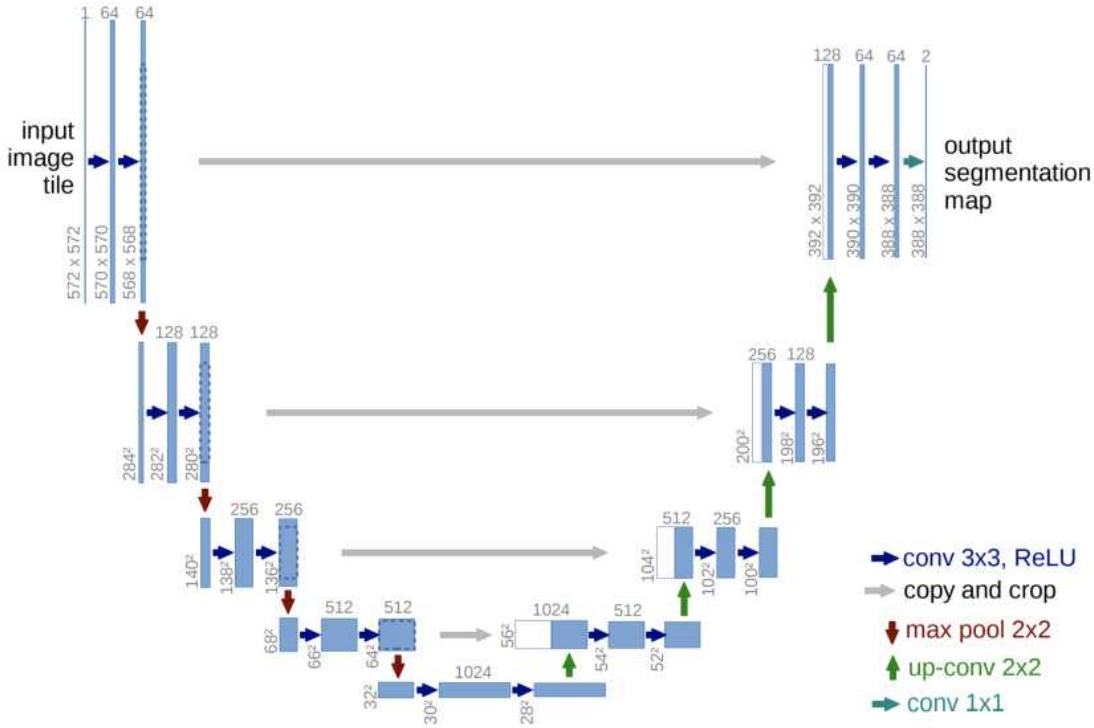
Feature extraction in CNN, previously mentioned, is comprised of piling convolutional layer and pooling layer repetitively. Convolutional layer is a computing layer of reflecting activation function after applying filter to input data. Next to it, a pooling layer is optional often used for improving learning efficiency. As shown in Figure #11, in CNN for image feature extraction, convolution filter shifts all around the input data, and feature map resulted through the convolution calculation. Size and shape of output data varies depending on following factors in the convolutional layer: filter size, stride, padding, and maximum pooling size. In Figure #12, we can find an example of whole CNN process for three channel image input.

A. U-Net Neural Network

[Courtesy of] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015.

[Courtesy of] https://keras.io/examples/vision/oxford_pets_image_segmentation/

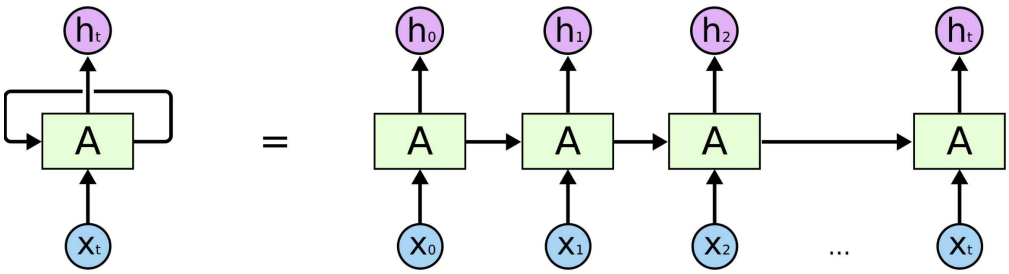
Autoencoder, a representative method applying CNN to image reproduction, is an AI model possible of multiple application by giving asymmetric variation to the network in addition to having simple functions such as reproducing and restoring images. U-Net (because it is U-shaped), specifically developed for the segmentation of medical images and videos, is a kind of Autoencoder network which equalizes the shapes of two corresponding layers which lies symmetrically between convolution and deconvolution processes. It is widely used due to its high accuracy, but with additional memory consumption. Likewise input and output fields' correlation in CFD can be learned by appropriately adding variables of interest to latent feature vector in the middle (called conditional U-net).



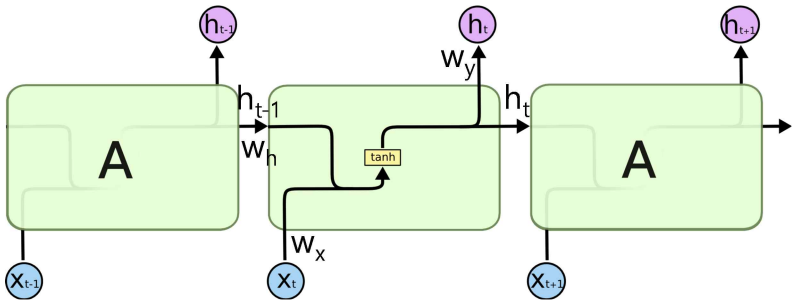
Reference Figure 13. Example of U-net Model

III. Recurrent Neural Network

[Courtesy of] An article in the Christopher Olah's blog, "Understanding LSTM Networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 27-Aug-2015.



(a) Structure of Recurrent Neural Network



(b) Main Flowchart

Figure #14. Schematic Diagram of Recurrent Neural Network

A. Basic Concept

Recurrent neural network (RNN) has achieved many successful tasks in various areas such as speech recognition, language modeling, translation, and image annotation past few years. It is an AI model developed to surpass shortcomings of traditional neural networks which are unable to remember later occurrings based on previously happened incidents. RNN is of structure which maintains information obtained from previous steps by reiterating itself. As shown in Figure #14, block A receiving input X_t and then releasing h_t as output becomes a unit of RNN. A loop in the unit indicates that information from previous step is received in the network of the next step. This RNN's reiterating structure tells that it is not very different from existing neural networks. In other words, RNN is equivalent to a

network that keeps copying one network and delivers information in order as shown in Figure #14(a).

B. Implementation

In Figure #14(b) defines h_t as hidden state value at present time point t. There are total two weights for calculating memory cell of a hidden layer. One weight is W_x which is for an input layer. Another weight is W_h for hidden state value, h_{t-1} at previous time point, $t-1$.

Below is the equation of above.

- Hidden layer: $h_t = \tanh(W_x x_t + W_h x_{t-1} + b)$ (#17)
- Output layer: $y_t = f(W_y h_t + b)$ (#18)

In here, f is one of the nonlinear activation functions. If expressing RNN's hidden layer computation as vector and matrix computation, vector dimension can be called d and size of hidden state can be called D_h. Then each size of vector and matrix are as below.

- Dimension : $x_t : (d \times 1)$, $W_x : (D_h \times d)$, $W_h : (D_h \times D_h)$, $h_{t-1} : (D_h \times 1)$, $b : (D_h \times 1)$

Assuming batch size is 1 and d and D_h equal to 4, RNN's hidden layer computation is shown as below in Figure #15.

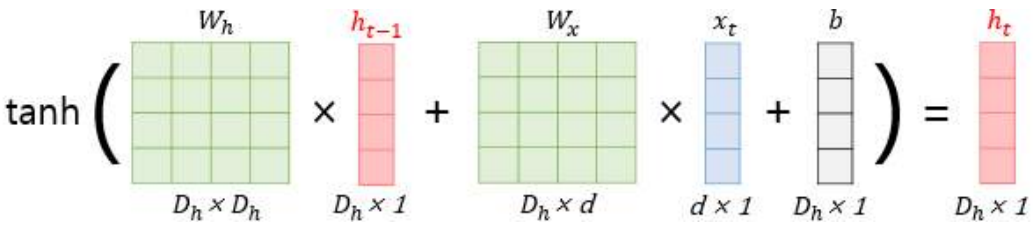


Figure #15. Dimensional Composition of Basic Operation

At this point, mainly hyperbolic tangent function (tanh) is used as an activation function for calculating h_t. Upon the equations above, weight values, W_x, W_y, W_h, are equally shared at every time point in one layer. But if hidden layers are more than two, weights are each different for each hidden layer.

Output layer as of activation function for calculating result value, y_t can be different depending on problems to be solved. For classification application which is mainly dealt in computer science, logistic regression function and Sigmoid function can be used in an output layer. For cases such as multi-class classification, Softmax function can be used in an output layer. In a regression application like CFD if antiderivative and derivative function uses tanh and Swish function which are continous functions in output layer, it is presented to be modest by many.

C. LSTM Networks

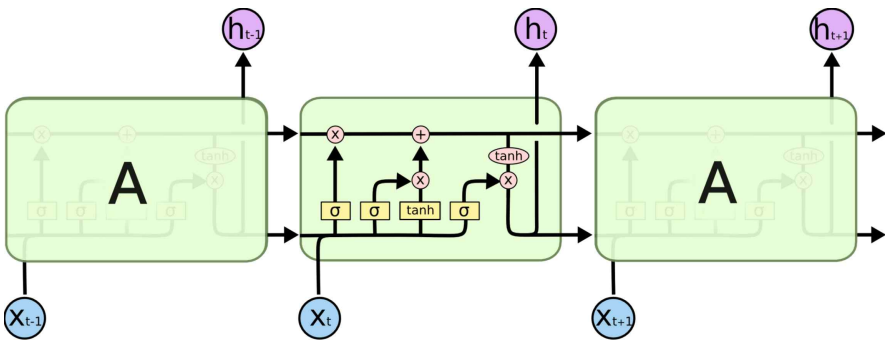


Figure #16. Schematic Diagram of Recurrent Neural Network

Long Short Term Memory networks - usually just called "LSTMs" - are a special kind of RNN, capable of learning long-term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn.

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure as shown in Fogure #16. Instead of having a single neural network layer, there are four, interacting in a specific way.

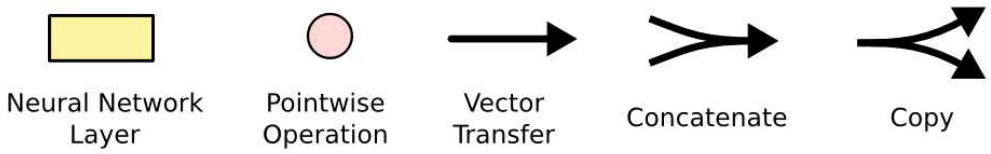


Figure #17. Operator Flow Symbol

Figure #17 shows that each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

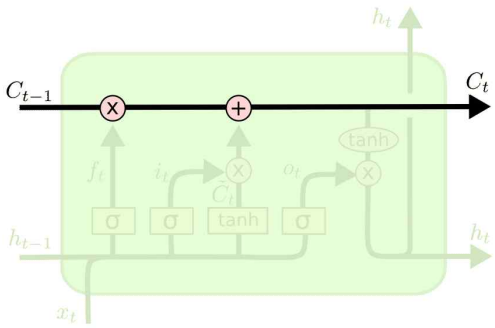


Figure #18. Cell State of LSTM

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

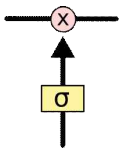
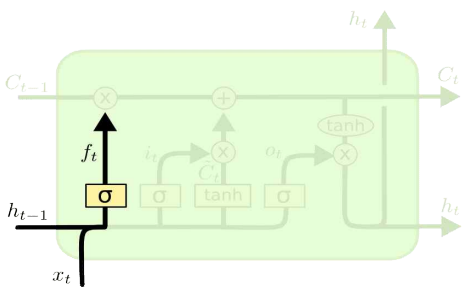


Figure #19. First Gate of LSTM

Sigmoid layer outputs numbers between zero and one. The number indicates a measure of how many information that each component has to transer. If the number is 0, it means “let nothing through”. If the number is 1, it means “let everything through”. An LSTM contains three Gate, and these Gate protects and controls the cell state.

Step-by-Step LSTM Walk Through

The first step of LSTM is to decide what information to withdraw fromn the cell state as a result of determination by sigmoid layer. That’s why this step of Gate is called “forget gate layer”. In this step, h_{t-1} and x_t are received. and then number between 0 and 1 is sent to C_{t-1}. If the number is 1, it means “completely keep this”, and it if the number is 0, it means “completely get rid of this”.

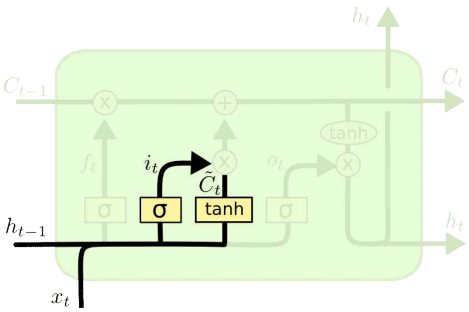


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure #20. Forget Gate Layer of LSTM

(In case of natural language AI) By going to the language model problem which predicts forthcoming words based on earlier preceded words, cell state may have gender information of current subject. Therefore cell state would be preparing to have use of pronoun suitable for gender. However we may not want to think of gender information of existing subject when new subject comes in.

Next step is to decide what new forthcoming information will be saved in the cell state. First, sigmoid layer which is called “input gate layer”, wll choose which value will be updated. Followed by that, tanh layer makes vector, $\tilde{C}_t$ which are new candidate values and prepares to be added to the cell state. Information resulting from these two steps are combined and makes ingredients to update the status.

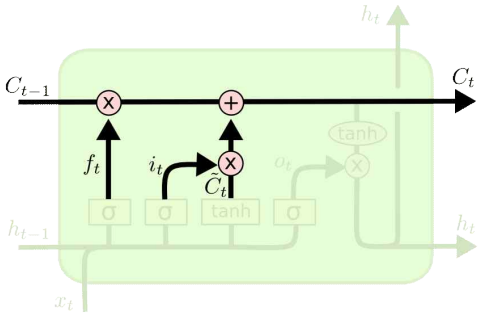


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure #21. Input Gate Layer of LSTM

(In case of natural language AI) existing subject’s gender is to be forgotten and instead, gender information of new subject is wanted to be added to cell state.

Now new cell state, C_t will be made by updating C_{t-1} which is in past status. As it is already set which value will be updated in which degree from previous step that only thing left is to proceed it. Firstly, f_t is multiplied to old state leading to forgetting things that are decided to forget at the very first step. Then $i_t \times \tilde{C}_t$ is added. This resulting value is an scaled value as much as the degree how much to update the value that is decided to upgrade in second step.



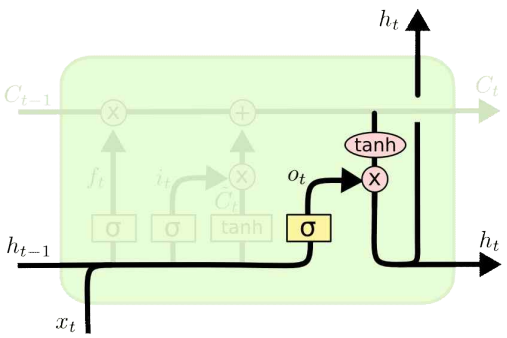
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure #22. Cell State Update of LSTM

(In case of natural language AI) this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push the values to be between -1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

(In case of natural language AI) since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure #23. Output Gate Layer of LSTM

IV. Gaussian Process Regression

[Courtesy of]

- C. E. Rasmussen, Carl Edward. "Gaussian processes in machine learning," Summer school on machine learning. Springer, Berlin, Heidelberg, 2003.

- C. E. Rasmussen & C. K. I. Williams, Gaussian Processes for Machine Learning, the MIT Press, Chapter 2 Regression, 2006, ISBN 026218253X.

[Courtesy of] https://juanitorduz.github.io/gaussian_process_reg/

A. Summary

Supervised learning of regression and classification form is an important comprising factor of dataset analysis, statistics and machine learning. Before advent of machine learning and deep learning, parametric modeling method which parameter is set by operator and then model is constructed used to play this role. One limit is parametric modeling lacks making an expression due to limited number of parameters in cases when dataset is complex even though parametric modeling is easy to analyze. More complicated models, like deep neural network, are hard to proceed due to hardware infrastructure limitation. In situation as this, kernel machin method such as gaussian proccess (GP) can flexibly conduct regression works in which this report is interested.

GP can be interpreted as a method to define a distribution over functions, and inference taking place directly in the space of functions. If new data is detected, then GP prior model is updated into posterior based on an inference statistic such as Baysian inference. This is a stochastic process which collects random variables from entire time or space. Each random variable has a normal distribution. Also GP is completely implemented as mean and covariance functions which are updated from the prior to the posterior in the light of the training data.

B. Gaussian Processes

A Gaussian process is fully specified by its mean function $m(x)$ and covariance function $k(x,x')$. This is a natural generalization of the Gaussian distribution

whose mean and covariance is a vector and matrix, respectively. The Gaussian distribution is over vectors, whereas the Gaussian process is over functions. We will write:

$$f \sim GP(m, k) \tag{#19}$$

Describing the above function as a sentence, "the function f is distributed as a GP with mean function m and covariance function k".

Although the generalization from distribution to process is straight forward, we will be a bit more explicit about the details, because it may be unfamiliar to some readers. The individual random variables in a vector from a Gaussian distribution are indexed by their position in the vector. For the Gaussian process it is the argument x (of the random function $f(x)$) which plays the role of index set: for every input x there is an associated random variable f(x), which is the value of the (stochastic) function f at that location. Let's assume Gaussian process as written in Eq. (#20).

$$f \sim GP(m, k), \quad \text{where } m(x) = \frac{1}{4}x^2,\ k(x, x') = \exp\left(-\frac{1}{2}(x - x')^2\right) \tag{#20}$$

In order to understand this process we can draw samples from the function f. In order to work only with finite quantities, we request only the value of f at a distinct finite number n of locations. How do we generate such samples? Given the x-values we can evaluate the vector of means and a covariance matrix using Eq. (#20), which defines a regular Gaussian distribution:

$$\mu_i = m(x_i) = \frac{1}{4}x_i^2,\ i = 1, ..., n, \tag{#21}$$

$$\Sigma_{ij} = k(x_i, x_j) = \exp\left(-\frac{1}{2}(x_i - x_j)^2\right),\ i, j = 1, ..., n \tag{#22}$$

μ, Σ are symbols to distinguish process and distribution. Now random vector can be generated based on this distribution. The vector has a function $f(x)$ for the corresponding x's.

$$\mathbf{f} \sim \aleph(\mu, \Sigma) \qquad (\#23)$$

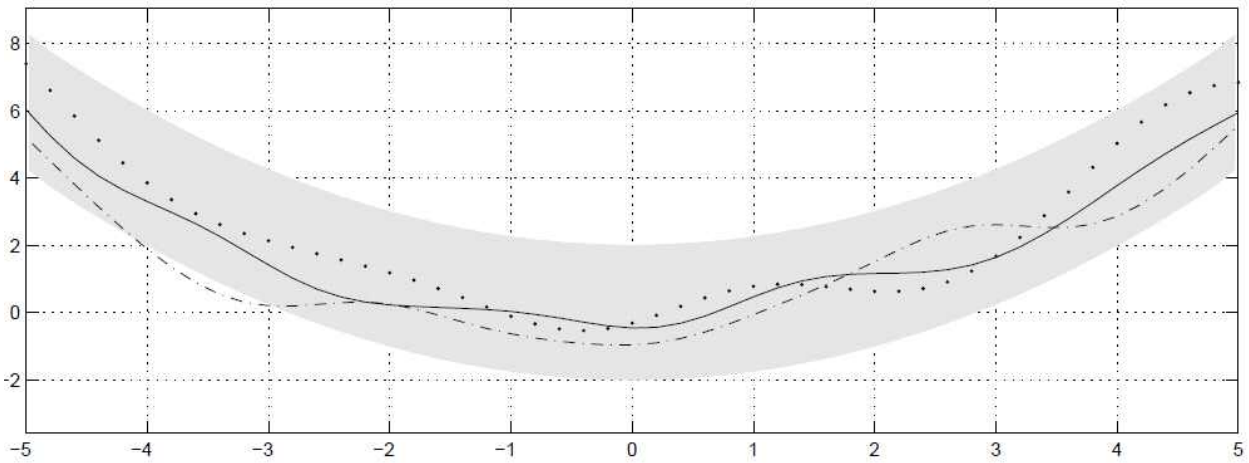


Figure #24. Function values extracted from three functions drawn randomly from a GP as specified in Eq(2). The dot graph is derived from Eq(4). Two other curves resulted from sampled points. The shaded area represents the 95% confidence intervals.

This example has illustrated how we move from process to distribution and also shown that the Gaussian process defines a distribution over functions. Up until now, we have only been concerned with random functions - in the next section we will see how to use the GP framework in a very simple way to make inferences about functions given some training examples.

C. Posterior Gaussian Process

Previously, we saw how to define distributions over functions using GPs. This GP will be used as a prior for Bayesian inference. The prior does not depend on the

training data, but specifies some properties of the functions; for example, in Figure #24 the function is smooth, and close to a quadratic. The goal of this section is to derive the simple rules of how to update this prior in the light of the training data.

One of the primary goals computing the posterior is that it can be used to make predictions for unseen test cases. Let $\mathbf{f}_*$ be the known function values of the training cases, and let $\mathbf{f}$ be a set of function values corresponding to the test set inputs, $\mathbf{X}_*$. Again, we write out the joint distribution of everything we are interested in:

$$\begin{bmatrix}\mathbf{f}\\ \mathbf{f}_*\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}_*\end{bmatrix}, \begin{bmatrix}\Sigma & \Sigma_*\\ \Sigma_*^\top & \Sigma_{**}\end{bmatrix}\right) \tag{#24}$$

Here, we've introduced the following shorthand: $\mu = m(x)$, $i = 1, \ldots, n$ for the training means and analogously for the test means μ_*; for the covariance we use Σ for training set covariances, Σ_* for training–test set covariances and Σ_{**} for test set covariances. Since we know the values for the training set $\mathbf{f}$ we are interested in the conditional distribution of $\mathbf{f}_*$ given f which is expressed as:

$$\mathbf{f}_*|\mathbf{f} \sim \mathcal{N}\left(\mu_* + \Sigma_*^\top \Sigma^{-1}(\mathbf{f} - \boldsymbol{\mu}),\ \Sigma_{**} - \Sigma_*^\top \Sigma^{-1}\Sigma_*\right) \tag{#25}$$

This is the posterior distribution for a specific set of test cases. It is easy to verify (by inspection) that the corresponding posterior process is:

$$\begin{aligned} f|\mathcal{D} &\sim \mathcal{GP}(m_\mathcal{D}, k_\mathcal{D}),\\ m_\mathcal{D}(x) &= m(x) + \Sigma(X,x)^\top \Sigma^{-1}(\mathbf{f} - \mathbf{m})\\ k_\mathcal{D}(x,x') &= k(x,x') - \Sigma(X,x)^\top \Sigma^{-1}\Sigma(X,x') \end{aligned} \tag{#26}$$

Here, $\Sigma(X,x)$ is a vector of covariances between every training case and x. These are the central equations for Gaussian process predictions. Let's examine

these equations for the posterior mean and covariance. Notice that the posterior variance $k_D(x,x)$ is equal to the prior variance $k(x,x)$ minus a positive term, which depends on the training inputs; thus the posterior variance is always smaller than the prior variance, since the data has given us some additional information.

We need to address one final issue: noise in the training outputs. It is common to many applications of regression that there is noise in the observations. The most common assumption is that of additive i.i.d. Gaussian noise in the outputs. In the Gaussian process models, such noise is easily taken into account; the effect is that every $f(x)$ has a extra covariance with itself only (since the noise is assumed independent), with a magnitude equal to the noise variance:

$$\begin{aligned} y(x) &= f(x) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \sigma_n^2), \\ f &\sim \mathcal{GP}(m, k), \qquad y \sim \mathcal{GP}(m, k + \sigma_n^2 \delta_{ii'}) \end{aligned} \tag{#27}$$

Here, $\delta_{ii'}$ is Knonecker's delta. Thus, the covariance function for a noisy process is the sum of the signal covariance and the noise covariance.

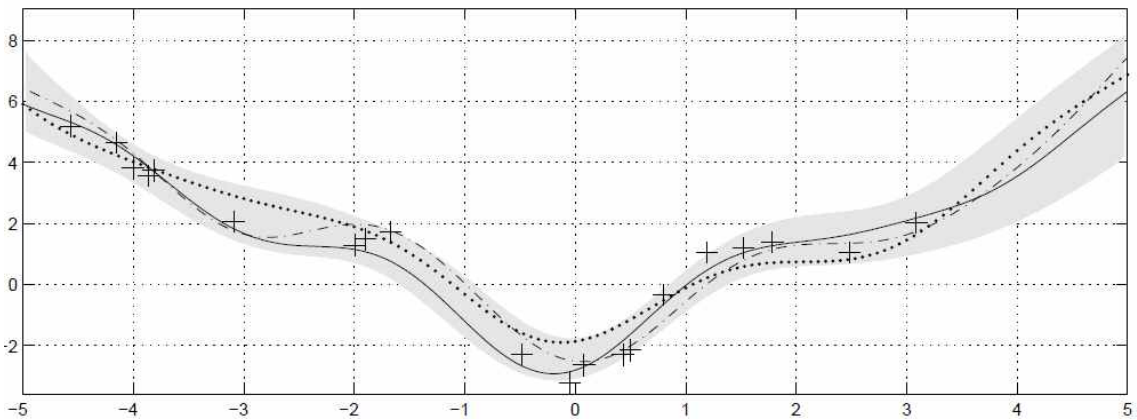


Figure #25 3 functions drawn randomly from the posterior based on given 20 training data points, the GP defined in Eqs. (#21, 22), and a noise level at 0.7. The shaded gray area represents 95% confidence region. Comparing to Figure 24, uncertainty is confirmed that decreased.

D. Training a Gaussian Process

In the previous section we saw how to update the prior Gaussian process in the light of training data. This is useful if we have enough prior information about a dataset at hand to confidently specify prior mean and covariance functions. However, the availability of such detailed prior information is not the typical case in machine learning applications. In order for the GP techniques to be of value in practice, we must be able to chose between different mean and covariance functions in the light of the data. This process will be referred to as training the GP model.

In the light of typically vague prior information, we use a hierarchical prior, where the mean and covariance functions are parameterized in terms of hyper-parameters. For example, Eq. (#20) can be generalized and used just as Eq. (#28).

$$f \sim \mathcal{GP}(m,\ k),$$
$$m(x) = ax^2 + bx + c, \quad \text{and} \quad k(x,x') = \sigma_y^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right) + \sigma_n^2 \delta_{ii'} \qquad \text{(\#28)}$$

Here, we have introduced hyper-parameters $\theta = \{a, b, c, \sigma_y, \sigma_n, l\}$. The purpose of this hierarchical specification is that it allows us to specify vague prior information in a simple way. For example, we've stated that we believe the function to be close to a second order polynomial, but we haven't said exactly what the polynomial is, or exactly what is meant by "close". In fact the discrepancy between the polynomial and the data is a smooth function plus independent Gaussian noise, but again we're don't need exactly to specify the characteristic length scale l or the magnitudes of the two contributions. We want to be able to make inferences about all of the hyperparameters in the light of the data.

In order to do this we compute the probability of the data given the hyper-parameters. Fortunately, this is not difficult, since by assumption the distribution of the data is Gaussian:

$$L = \log p(\mathbf{y}|\mathbf{x},\theta) = -\tfrac{1}{2}\log|\Sigma| - \tfrac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{y}-\boldsymbol{\mu}) - \tfrac{n}{2}\log(2\pi) \qquad \text{(\#29)}$$

In general, this value is referred as log marginal likelihood. The word “marginal” is used to emphasize that non-parametric model is dealt with.

Now hyper-parameters that optimizes the marginal likelihood value above can be evaluated:

$$\begin{aligned}\frac{\partial L}{\partial \theta_m} &= -(\mathbf{y}-\boldsymbol{\mu})^\top \Sigma^{-1}\frac{\partial m}{\partial \theta_m},\\ \frac{\partial L}{\partial \theta_k} &= \tfrac{1}{2}\,\mathrm{trace}\left(\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta_k}\right) + \tfrac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^\top \frac{\partial \Sigma}{\partial \theta_k}\Sigma^{-1}\frac{\partial \Sigma}{\partial \theta_k}(\mathbf{y}-\boldsymbol{\mu})\end{aligned} \qquad (\#30)$$

Here, θ_m and θ_k indicate hyper-parameters of mean and covariance functions respectively. Eq. (#30) can conveniently be used in conjunction with a numerical optimization routine such as conjugate gradients to find good hyper-parameter settings.

Figure #26 shows an example of mean and posterior confidence interval by using established GP model to which applied hyper-parameters optimizing marginal likelihood.

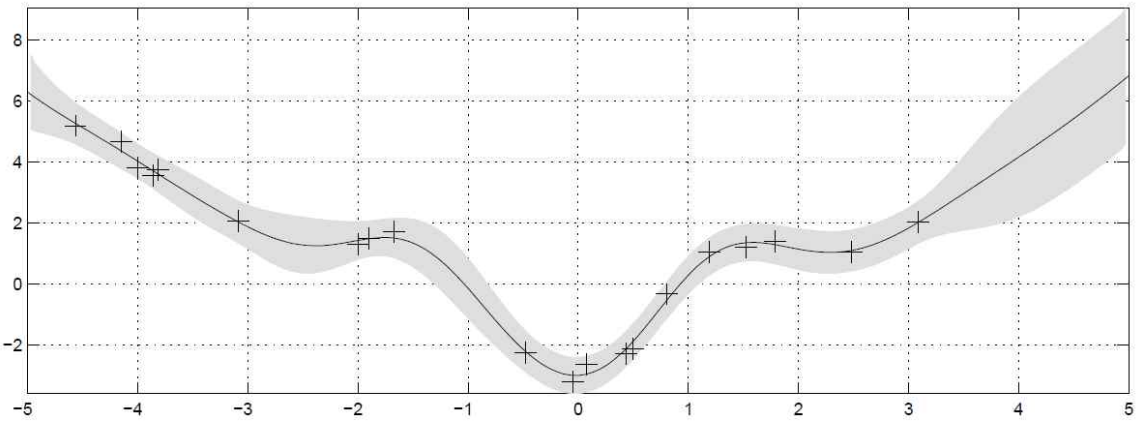


Figure #26. Mean and 95% posterior confidence interval region with parameters that is learned in direction of optimizing marginal likelihood which is defined in Eq. (10). Hyperparameters obtained by optimization are as follows.

$$a=0.3, b=0.03, c=-0.7, l=0.7, \sigma_y=1.1, \sigma_n=0.25$$

Comparing interchangeably of Figure #24~#26 which show modeling results of the same example data explained up to now, performance of GP can be vaguely confirmed. Research outcome (Lee, Jaehoon, et al. "Deep neural networks as gaussian processes." arXiv preprint arXiv:1711.00165 , 2017) which tells that GP is equal to results when a number of neurons in artificial neural network is infinitely extended also confirms GP's efficiency.