DEEP LEARNING FOR FLUID MODELING

Xu Botian
11810424

Lu Jichen
11811815

ABSTRACT

Shape optimization is critical in aerospace engineering. It refers to the process of finding a shape design with particular geometry that can minimize a specific cost or achieve the best performance while satisfying given constraints. This project aims to incorporate deep learning methods into the shape optimization problem with two primary motivations: (1) speed up the evaluation step by replacing numerical simulations, which are often very costly in computation resources and time, with a data-driven modeling alternative; (2) use generative adversarial network (GAN) for generating new solutions with expectedly good quality.

1 INTRODUCTION

1.1 FLUID DYNAMICS

1.1.1 NAVIER-STOKES EQUATIONS (NON-CONSERVATION FORM)

The Navier-Sokes equations are the most widely used **governing equations**. They state the fundamental physical principles upon which all of fluid dynamics is based:

Continuity equation: Mass is conserved

$$\frac{D\rho}{Dt} + \rho\nabla \cdot \mathbf{V} = 0$$

Momentum equation: Newton's law

$$\rho\frac{D\mathbf{V}}{Dt} = -\nabla p + \nabla \cdot \tau + \rho\mathbf{g}$$

Where notations are defined as follows:

1. $\frac{D}{Dt}$ is the material derivative, defined as $\frac{\partial}{\partial t} + \mathbf{V} \cdot \nabla$,
2. ρ is the density,
3. $\mathbf{V}$ is the flow velocity,
4. $\nabla\cdot$ is the divergence,
5. p is the pressure,
6. t is time,
7. τ is the stress tensor,
8. g accounts for body forces, for example gravity.

1.2 AIRFOIL DESIGN

An **airfoil** (American English) or aerofoil (British English) is the 2-d cross-sectional shape of a wing. Aircrafts rely on the aerodynamic force produced by the air moving over this shape. For our purpose, several terminologies are introduced here.

The **angle of attack** (AOA) is the angle between a reference line on a body (often the chord line of an airfoil) and the vector representing the relative motion between the body and the fluid through which it is moving.

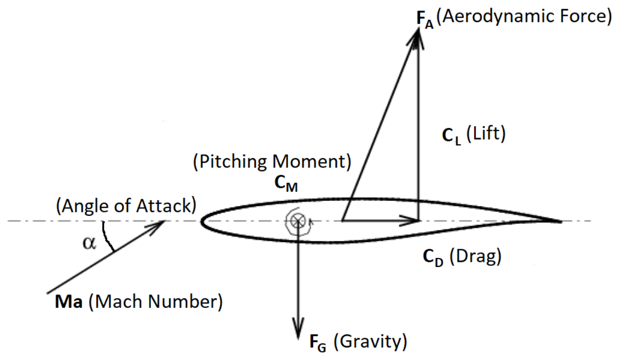


Figure 1: Illustrated terminologies of an airfoil.

1.3 Deep Learning

Deep learning has been gaining more and more attention on its application to engineering problems. A **deep neural network** model is characterized by stacked feed-forward layers with learnable parameters. The model typically learns to fit a specific function by adjusting its parameter using techniques like stochastic gradient descent (SGD) where the gradient is computed with respect to some loss function which measures the distance between the prediction given by the model and the expected target value.

2 Problem Definition

2.1 Shape Optimization

In this project, we work on the design problem of **supercritical airfoil**. As the speed of the aircraft approaches the speed of sound, the air accelerating around the wing reaches Mach 1 and shockwaves begin to form. The formation of these shockwaves causes wave drag. Supercritical airfoils are designed primarily to minimize this effect by flattening the upper surface of the wing. A typical optimization problem consists of the following steps:

1. Given a set of designs $\{a\}$.
2. Evaluate each of $\{a\}$ according to certain metrics.
3. Generate new design $\{a'\}$ from $\{a\}$.
4. Replace (or extend) $\{a\}$ with $\{a'\}$ following a replacement strategy.
5. Repeat 1 through 4 until a termination condition is met.

This process is illustrated in 2, for designing and optimizing an airfoil.

2.2 Efficient Evaluation: Flow Field Prediction

A supercritical airfoil is evaluated by analysing the flow field as solution of a time-averaged version Navier-Stokes equations. Unfortunately, solving RANS using numerical methods is computationally too expensive both in time and resources to be practical for a large scale local-search based optimization method. So we seek a more efficient and still effect alternative: predict the solution directly via a deep neural network model. Define $NN(a, \alpha; \theta)$ to be a network parameterized by θ. a is the geometry as will be discussed later and α is the vector of flow parameters (angle of attack and mach number). We train the network by

$$\hat{u} = NN(a, \alpha; \theta), \theta^* = \arg\min L(\hat{u}, u)$$

where L is some loss function.

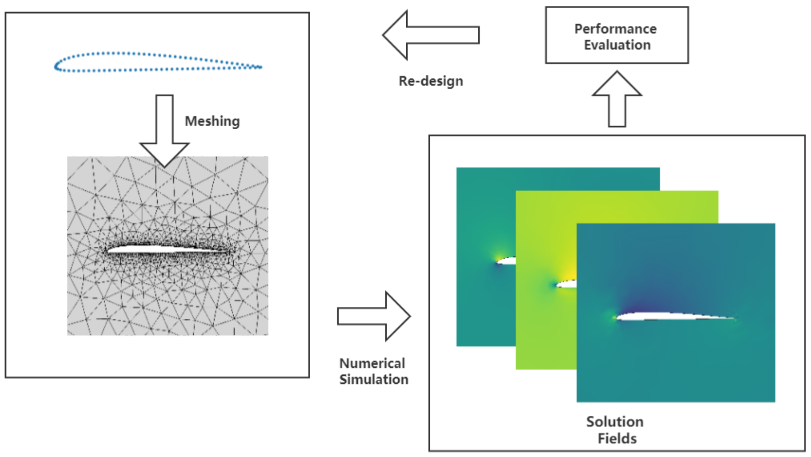


Figure 2: A Typical Design Workflow. In our setting, the **numerical simulation** step is replaced with a neural network for directly predicting the solution, and the **re-design** is done by a deep generative model.

2.3 Search Over Design Space: Generative Models

The optimization part of the whole modeling combines two sub-model: Gerative Adversarial Network (GAN) and Nondominated Sorting Genetic Algorithm II (NSGA-II). The first model, GAN, is widely used in image generation field. Since the airfoil outline could be seen as a curve image, we try to generate new airfoil outline using GAN model. For the second algorithm, we conbine one of the classic evolutionary algorithm NSGAII, aiming to select optima sample from both the input samples and the generated samples. Besides, we also through the embeded evolutionary algorithm to reduce the possibility of local optimization and enlarge the search range.

3 Methods: Prediction

3.1 Structured Mesh

A mesh is an approach to **spatial discretization** we choose to represent our problem. It divides the space into **elements** defined by nodes and their connectivity. **Structured mesh** is one type of mesh generated by transforming the original space. It is identified by regular connectivity. Elements can be quadrilaterals in 2-d or hexahedra in 3-d.

The main difference between structured grids and cartesian grids is that it can be irregular in the physical plane, but remains regular in the computation plane. This property enables it to be used for approximating the gradient of a field using the finite difference method.

Moreover, when used as input/output to a convolutional neural network, it can be stored and processed conveniently as n-dimensional arrays (tensors) due to their regular 4-connectivity:

$$a \in \mathbb{R}^{m\times n\times k}$$

where m, n is the two axes, and each of the $m \times n$ nodes holds k attributes. The attributes may be coordinates of a position and the physical field quantity at that position. Specifically,

$$a^{(input)} \in \mathbb{R}^{m\times n\times 3} \text{ with } a^{(input)}_{i,j} = (\mathbf{x}_x, \mathbf{x}_y, I)^T$$

where I takes either 0 or 1 to indicate whether this node is on the surface of airfoil.

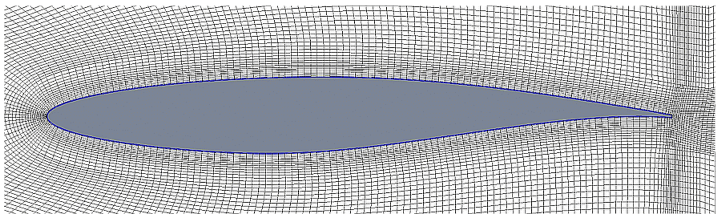


Figure 3: Example mesh around RAE2822 airfoil (not complete).

3.2 Network Architecture

The overall architecture of the neural network model is shown in 4. It follows a encoder-decoder scheme with residual ”bottleneck” blocks in the middle.

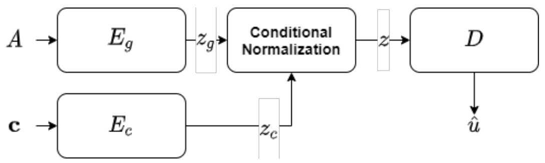


Figure 4: Network architecture.

3.2.1 Encoder

The model $NN(a, \alpha)$ (we don't explicitly state θ again in the rest of this text for convenience) takes two kinds of input:

$$a \in \mathbb{R}^{m \times n \times 3} \text{ and } \alpha \in \mathbb{R}^2$$

so we need different structures to process the input.

- A convolutional encoder $E_g : \mathbb{R}^{m \times n \times 3} \to \mathbb{R}^{m' \times n' \times z}$ is used to encode the input geometry a.
- A multi-layer perceptron encoder $E_c : \mathbb{R}^2 \to \mathbb{R}^{2z}$ is used to encode the flow parameters α in to a latent vector $\mathbf{z}$.

To combine the $\mathbf{z}_g = E_g(a)$ and $\mathbf{z}_c = E_c(\alpha)$, we use a **conditional normalization** (Dumoulin et al. (2017)) operator, which is defined as

$$\mathbf{z} = \mathbf{z}_c^{(1)} \left(\frac{z_g - E[z_g]}{\sqrt{Var[z_g]}} \right) + \mathbf{z}_c^{(2)}$$

where the mean and variance of z_g are computed per-dimension separately.

3.2.2 Decoder

A convolutional decoder $D : \mathbb{R}^{m' \times n' \times z} \to \mathbb{R}^{m \times n \times 3}$. The solution vector u or $\hat{u} = D(\mathbf{z})$ consists of the physical quantities of interest, namely the velocity components $\mathbf{v}_x, \mathbf{v}_y$ and pressure p.

The upsampling method used here is PixelShuffle which is widely used in the literature of super resolution.

3.2.3 Training

In the current stage, we use a loss function that is a combination of Mean Square Error (MSE) and Mean Absolute Error:

$$L(u, \hat{u}) = \lambda_1 \langle ||u - \hat{u}||_2 \rangle + \lambda_2 \langle ||u - \hat{u}||_1 \rangle$$

4 METHODS: OPTIMIZATION

4.1 GERATIVE ADVERSARIAL NETWORK (GAN)

Gerative Adversarial Network (GAN) is widely used in image generation. There are totally two module in a classic GAN model: generator and discriminator. The generator is used to generate new images from a random noise data. While discriminator is used to judge whether the generated image is true or false. It could be regarded as a game process.

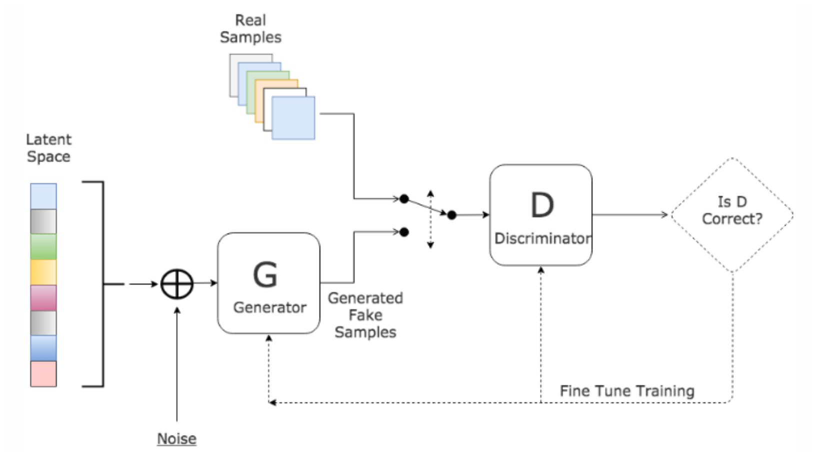


Figure 5: GAN Model Diagram

The core formula of GAN is as follows:

$$\min_G \max_D V(D,G) = E_{x\ p_{data}(x)}[logD(x)] + E_{z\ p_z(z)}[log(1 - D(G(x)))]$$

In the formula, G represents generator and D represent discriminator. When we fix G, we want a good D to judge all real image to true and all fake image to false, then we need to minimize the formula; when we fix D, we want a good G to gnerate more real image to cheat D, which maximum the formula. Therefore, the whole process is a game between generator G and discriminator D.

4.2 NONDOMINATED SORTING GENETIC ALGORITHM II (NSGA-II)

4.2.1 BACKGROUND

Nondominated sorting genetic algorithm II (NSGA-II) is one of the classical and state-of-art multi-objective EAs. NSGA-II is an algorithm based on nondominated sorting genetic algorithm (NSGA). The whole structure is classical since it contains the processes including environmental selection, mating and mutation, generating offspring. There are three main problems for ordinary NSGA **?**:

- $O(MN^3)$ computational complexity (where M is the number of objectives and N is the population size).
- Nonelitism approach.
- The need for specifying a sharing parameter.

4.2.2 BASIC PROCESS

The basic process is shown in Algorithm. 1.

4.2.3 FAST NONDOMINATED SORT

One of the great improvement of NSGA-II is the speed up based on fast nondominated sort, which has reduce the time complexity from original $O(MN^3)$ in NSGA to $O(MN^2)$.

Algorithm 1 NSGA-II Procedure

```
Input: $N(populationsize)$, $T(threshold)$.
Output: $P(final\ population)$.
1: $P \leftarrow Initialization(N, Z)$
2: while $t \leq T$ do
3:     $Q \leftarrow Generating\ offspring$
4:     $R \leftarrow P \cup Q$
5:     $Nondominated\ sortR$
6:     $P \leftarrow Select\ best\ nondominated\ sets\ until\ size \geq N$
7:     if $size \geq N$ then
8:         $Crowding\ distance\ sort\ last\ nondominated\ front$
9:     end if
10: end while
11: $Result \leftarrow Final\ population\ P$
```

Algorithm 2 Fast Nondominated Sort

```
Input: $P(population)$.
Output: $P(sortedpopulation)$.
1: for $eachp \in P$ do
2:     $S_p = \emptyset$
3:     $n_p = 0$
4:     for $eachq \in P$ do
5:         if $p \prec q$ then
6:             $S_p = S_p \cup \{q\}$
7:         else if $q \prec p$ then
8:             $n_p = n_p + 1$
9:         end if
10:     end for
11:     if $n_p = 0$ then
12:         $p_{rank} = 1$
13:         $F_1 = F_1 \cup \{p\}$
14:     end if
15: end for
16: $i = 1$
17: while $F_i \neq \emptyset$ do
18:     $Q = 0$
19:     for $eachp \in F_i$ do
20:         for $eachq \in S_p$ do
21:             $n_q = n_q - 1$
22:             if $n_q = 0$ then
23:                 $q_{rank} = i + 1$
24:                 $Q = Q \cup \{q\}$
25:             end if
26:         end for
27:     end for
28:     $i = i + 1$
29:     $F_i = Q$
30: end while
```

4.2.4 Crowding Distance Assignment

The selection part of NSGA-II has another parameter, named crowding distance assignment, which indicate the diversity of the populatoin. In order to obtain population with larger distribution range, or search range, we need to select individuals with long distances.

4.3 Optimization Model

4.3.1 Framework

The whole framework of the optimization model use the GAN-based MOEA (GMOEA). The framework could be summarized as follows:

1. Initialization
   Initialize with a set of data samples. Initialize GAN model.
2. Classification
   Classify the samples into "real" and "fake" with the metric of the problem.
3. Training
   Train the GAN model to generate samples to approach the train set.
4. Offspring generation
   Use the GAN generated samples and operator in MOEA to generate offspring.
5. Environmental Selection
   Use operators in MOEA to select samples with better fitness.
6. Iteration until reach the goal

As for the visulization of the framework, it is shown in Figure. 6. The two parts separately work. And the MOEA actually provide the selection pressure to the GAN model to make the generator generate the images we want.

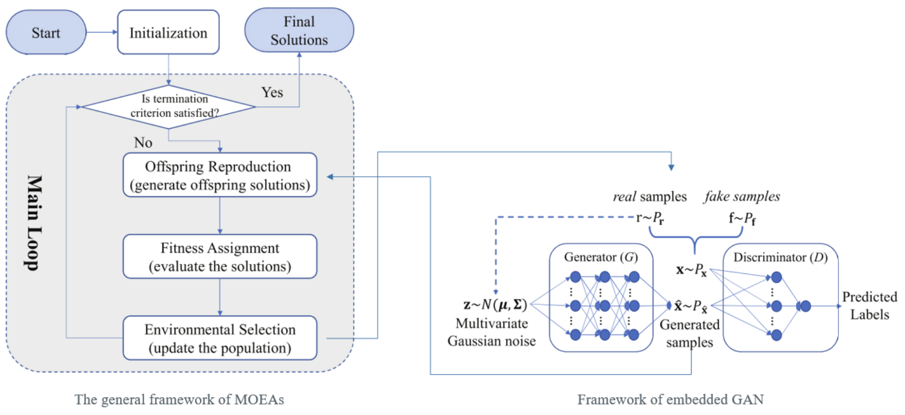

Start
Initialization
Final Solutions
Is termination criterion satisfied?
Yes
No
Main Loop
Offspring Reproduction (generate offspring solutions)
Fitness Assignment (evaluate the solutions)
Environmental Selection (update the population)
real samples
fake samples
r∼P_r
f∼P_f
Generator (G)
Discriminator (D)
x∼P_x
z∼N(μ,Σ) Multivariate Gaussian noise
x̂∼P_x̂ Generated samples
Predicted Labels
The general framework of MOEAs
Framework of embedded GAN

Figure 6: GAN-based MOEA

5 Experiments: Prediction

5.1 Setup

- **Data**.The dataset for this part is provided by the Department of Aerospace Engineering, SUSTech. It contains simulation results for 201 different airfoils (5 each, 1005 in total) under 0.73 mach with angle of attack ranging from 2 to 3 degrees.
- **Encoders**. The convolutional encoder E_g consists of stacked convolution layers with stride which downsample the input by a factor of $2^4 = 16$. Deformable convolutions Dai et al.

(2017) are used to capture the spatial irregularity. E_c is a 3-layer MLP. The latent dimension z is chosen to be 128.

- **Decoder**. The decoder does $2^4\times$ upsample on the encoded z to recover the size of the input mesh. The upsampling technique is PixelShuffle by Shi et al. (2016).

5.2 Results

After 1500 epochs training with learning rate automatically reduced on plateau, the model achieves a relative error (in magnitude) of less than 2%. Examples are shown in 7.

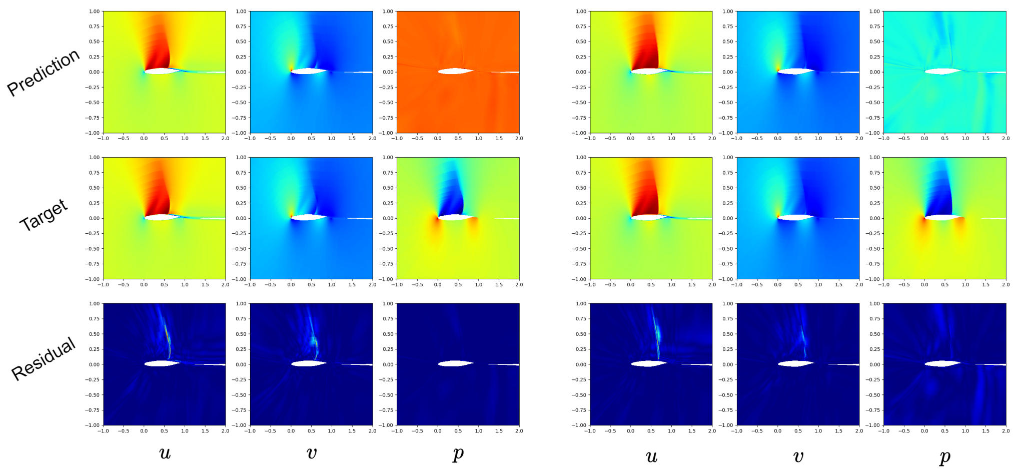


Figure 7: Example prediction results. The model performs relatively better on the velocity fields (u and v) but suffers some saturation on the pressure field. This may be caused by the difference in data distribution and is yet to be examined.

6 Experiments: Optimization

6.1 Setup

6.1.1 Agent Model

To complete the optimization model, we use a simple agent model using ANN. Its input is airfoil sample with 76 sampling points and output the corresponding aerodynamic metrics. Its output is 5 aerodynamic metrics. There are two hidden layer of the ANN model with neural number of 100 and activation function ReLu. The train data set is the 200 airfoil outline data with their aerodynamic metrics. The loss value during training of embeded evaluation model is like follows (with lr=0.02 and iteration = 100):

6.1.2 Metric

To judge the quality of generated airfoil outline, we use a metric smoothness to measure whether the sample points are smooth to imitate the real designed airfoils. The formula is:

$$smoothness = \sum_{i=1}^{N-1} Distance_{p_n \perp |p_{n-1}p_{n+1}|}$$

As for the explanation of the smoothness formula, it uses the vertical distance between the point and the line formed by its two adjacent points, sum the distance to get the metric of a airfoil composed by sample points. Note that the smaller the smoothness metric, the better generated airfoil outlines.

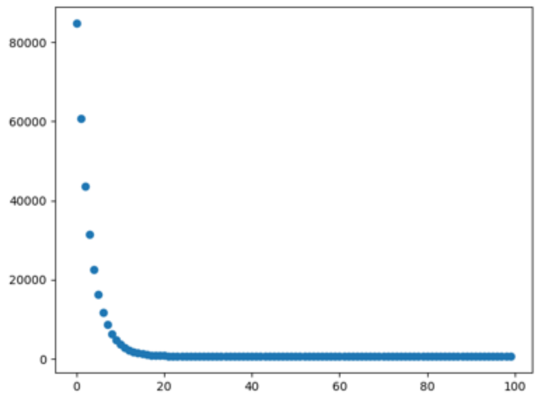


Figure 8: Loss of agent model

6.2 Results

Based on the above framework, we construct the optimization model and test with certain parameters.

1. Input sample number:200
2. Sampling points: 76
3. Input aerodynamic metrics: 5
4. Population number: 50
5. Iteration for embeded evaluation model: 100
6. Evaluation Iteration for GMOEA: 100000

The generated airfoil outlines are as follows with The generated airfoil reflect following things:

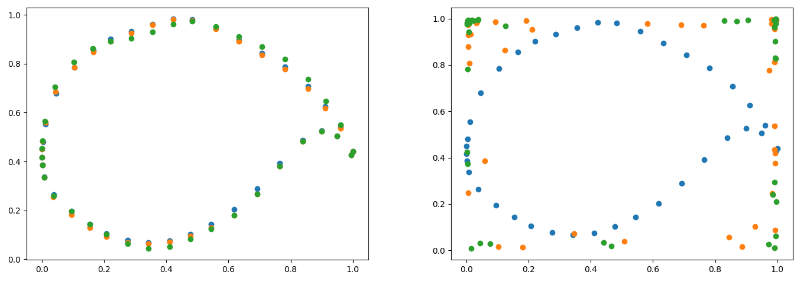

(a) Generated airfoils (b) Some failed generated airoils

Figure 9: Generated airfoils of GMOEA

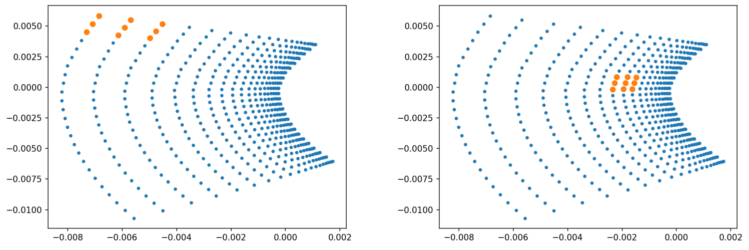
- The optimization model can generate airfoil successfully.
- The generated airfoil model has little variance, the reason could be the low variance of input training sample.
- The current agent model is rough, the metrics may not be accurate enough.
- Some of the failed generated airfoil has not been evicted.

7 Discussion

7.1 Limitations

7.1.1 Inconsistent Convolution Operators

Due to the fact that the mesh is irregular in space, a convolution kernel in inconsistently defined when applied to different spatial locations, as depicted in 10.



(a) (b)

Figure 10: A convolution kernel defines different impulse responses when applied at different locations on the mesh.

7.1.2 Generalization

The generalization performance of the deep neural network model suffers from the fact that the training data provided greatly resemble each other. This poses a severe problem in the sense that the learned model may not be able to work properly with unseen samples, but has just remembered the training data.

Without loss of generality, assuming we are using MSE as loss function, the learning process is equivalent to the optimization problem:

$$f^* = \arg\min \mathbb{E}_{x,y\sim p_{data}}[||y - f(x)||_2]$$

which yields

$$f^*(x) = \mathbb{E}_{y\sim p_{data}(y|x)}[y].$$

However, if the numerical variation of x is too small such that the conditioning becomes negligible, the model degenerates to

$$f^*(x) = \mathbb{E}_{y\sim p_{data}}[y]$$

thus the model effectively just predicts the average of what it has seen during training.

8 Future Work

8.1 A Dynamical System View of Neural Network

Recently, there are several groups of people who proposed similar ideas about the dynamical system's view of deep learning to shed light on its application to computational science.

- People from Applied Mathematics/Physics aim to solve equations (E et al. (2017)) or simulate physical processes.
- Those from AI try to achieve a performance comparable to or better than existing deep models on both common AI tasks (e.g. image classification) and tasks like learning physical rules.

Neural ODE (Chen et al. (2019)) is the most famous but not the only one of them (a collection of similar or related papers is available at `https://github.com/Zymrael/awesome-neural-ode`). For a formal definition I personally prefer what's proposed by Weinan (2017). They differ in the details of formulation but generally they are offering a new interpretation:

- Conventional: Neural networks are said to be extracting and processing features through layers.
- Dynamical System: The deep model unfolds the time evolution of a dynamical process controlled or defined by layer(s) of neural network. In other words, we are going through a trajectory (or a ”flow”) of hidden states.

  I think there is indeed some connections between this point of view and how a brain actually works, as you’ve mentioned today.

Although people from different fields focus on different problems, they share some common arguments:

- It is possible to have better (at least as good) function approximation by introducing the notion of infinite number of layers (Lu et al. (2020),Bai et al. (2019)) And Li et al. (2020) gives some theoretical analysis.
- Can impose structure on the dynamical system. Examples of learning exact physical rules (Greydanus et al. (2019), Cranmer et al. (2020)).

Additionally, it may also:

- provide some engineering benefits (e.g. less memory cost Bai et al. (2019)).

8.2 On a Specific Problem: Fluid Dynamics

This part is partly inspired by a series of work byRaissi et al. (2019) which use constraints as loss terms. The dynamics of inviscid fluid is governed by simplified Navier-Stokes equations:

Continuity: $$\frac{D\rho}{Dt} = -\nabla\rho \cdot \mathbf{v}$$

Momentum: $$\frac{D\mathbf{v}}{Dt} = -\frac{1}{\rho}\nabla p + \mathbf{g}$$

The pressure p is given implicitly

where t is time, ρ is the density and $\mathbf{v}$ is the fluid velocity.

Define u to be the solution vector and let $\mathbf{f}$ be a neural network parameterized by θ:

$$\mathbf{u} = (\rho, \mathbf{v}, p)^T, \ \ \frac{d\mathbf{u}}{dt} = \mathbf{f}(\mathbf{u}; \theta)$$

then the solution at time t is

$$\mathbf{u}(t) = \mathbf{u}(t_0) + \int_{t_0}^{t} \mathbf{f}(\mathbf{u}; \theta) dt$$

Setting

$$\mathbf{h}(\mathbf{u}) = (\nabla \cdot \rho\mathbf{v}, -\frac{1}{\rho}\Delta p + \mathbf{g}, \ \cdot \ )^T$$

and train the network by

$$\theta = \arg\min \int_{t_0}^{t} L(\mathbf{f}(\mathbf{u}; \theta), \mathbf{h}(\mathbf{u}))dt$$

where L is some loss function. In this way, we relate the left- and right- hand sides of the PDE and force the model to satisfy the equation. One possible nice thing about this formulation is that there is a convenient way to get the spatial derivatives.

If we further parameterize $\mathbf{u}(t, \mathbf{x}; \theta)$ to be a function of time t and spatial position $\mathbf{x}$ and include its parameters into θ of our model. Then the differential operators $\nabla\cdot$ and ∇ can be obtained by numerical auto-differentiation provided by a lot of deep learning libraries.

Of course things might be much more tricky in practice and the method described above may not work at all.Unfortunately I didn't have the time to make a running demo of it yet. I'm planing to do so in the next few months after taking a deeper look into the related works and try to re-implement some of their experiments.

Possible applications:

- Directly employ this kind of models to do simulation. The advantages of it over traditional numerical methods are yet to be discovered and discussed, but one thing we aim for is to reduce the computational cost by leveraging deep models' ability of dimensionality reduction.
- Solving inverse problem. The equations themselves usually can only describe some but not all of the dynamics ongoing in the real world. In other words, there are something too complex to be included in close-form equations. Deep learning models can serve as tools in discovering these hidden dynamics. The simplest example may be estimating an coefficient from observed data.

REFERENCES

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models, 2019.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks, 2020.

Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks, 2017.

Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style, 2017.

Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, Nov 2017. ISSN 2194-671X. doi: 10.1007/s40304-017-0117-6. URL `http://dx.doi.org/10.1007/s40304-017-0117-6`.

Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks, 2019.

Qianxiao Li, Ting Lin, and Zuowei Shen. Deep learning via dynamical systems: An approximation perspective, 2020.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations, 2020.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, 2016.

E. Weinan. A proposal on machine learning via dynamical systems. 2017.