Solving PDE by Deep Learning

Team MnMs

구예찬
김현준

Chapter 1

PDE: 편미분방정식

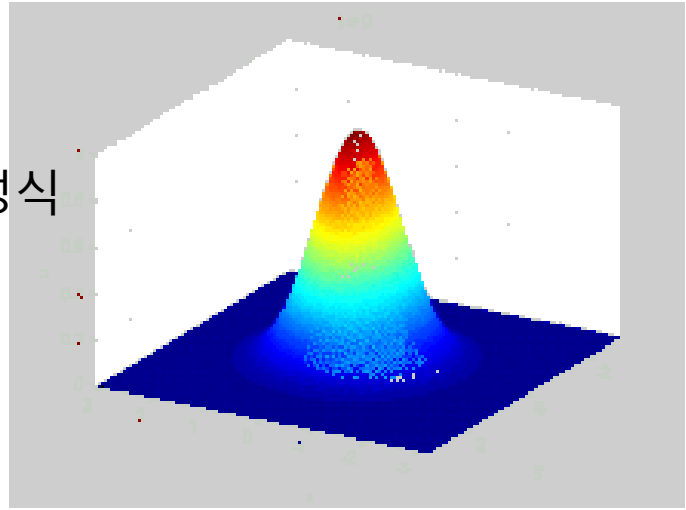
Partial differential equation

- 물리학, 공학, 기타 여러 산업에서 활발히 사용
- 편미분방정식의 analytic solution을 찾는 것은 일반적으로 매우 어렵다.
- 산업에서는 상당 부분 수치해석적 방법을 이용하여 근사해(approximation solution)를 찾아사용함

PINN(physics-informed Neural Network)

- Neural network를 이용하여 편미분방정식의 해를 수하는 시도
- 기존의 수치 해석적 방법보다 더 정확한 해를 구할 수 있다는 연구들이 발표되고 있다.

Burgers' equation



- 유체 역학, 음향, 기체 역학, traffic flow등에 많이 이용되는 비선형 편미분 방정식
- 초기 조건에 따라 해의 연속성이 깨지는 shock solution이 발생할 수 있다.
- Shock이 발생하면 weak solution이 여러 개 존재할 수 있고,

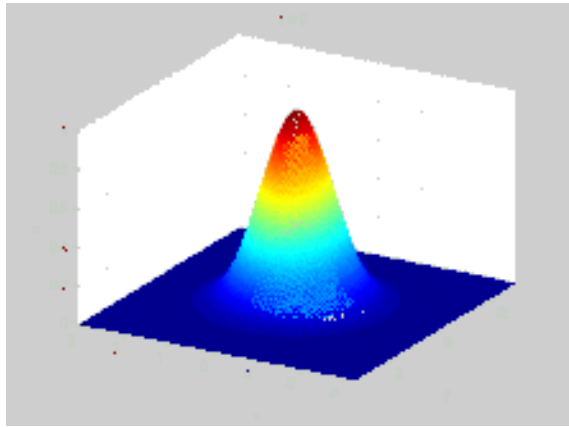
  그들 중 물리학적으로 의미가 없는 해도 있다.

Characteristic method

Characteristics-Informed Neural Networks for Forward and Inverse Hyperbolic Problems

Ulisses Braga-Neto
Department of Electrical and Electronic Engineering
Texas A&M University
E-mail: ulisses@tamu.edu

- 편미분 방정식의 해를 구하는 방법 중 하나
- Inviscid burger's equation에서는 특히 Characteristic method가 많이 사용됨
- Solution function의 값이 일정한 곡선을 생각한다.
- 2022년 Characteristic method를 이용하여 PINN을 강화한 CINN을 이용하는 논문이 arxiv에 등재



2D Burgers' equation

- 유체 역학, 음향, 기체 역학, traffic flow등에 많이 이용되는 비선형 편미분 방정식
- 초기 조건에 따라 해의 연속성이 깨지는 shock solution이 발생할 수 있다.
- Shock이 발생하면 weak solution이 여러 개 존재할 수 있고, 물리학적으로 의미 없는 해가 존재
- If R=inf, inviscid 2D Burger's equation

$$u_t + uu_x + vu_y = \frac{1}{R}(u_{xx} + u_{yy})$$

$$v_t + uv_x + vv_y = \frac{1}{R}(v_{xx} + v_{yy})$$

- 모델 결과의 신뢰성을 위해 analytical solution을 구할 수 있는 condition들로 학습을 진행
- Linear condition : shock 발생x
- Exponential condition: shcck 발생x
- Riemann problem: shock 발생o

Example1. Linear condition

만약 $u(x,y,0)=ax+by,\ v(x,y,0)=cx+dy$라 할 때

characteristic method를 이용하여 해를 구하면

$$(x_0,y_0)^t=\frac{1}{(at+1)(dt+1)-bct^2}\begin{pmatrix}dt+1 & -bt\\ -ct & at+1\end{pmatrix}(x,y)^t$$

$$u=ax_0+by_0,\ v=cx_0+dy_0$$

$$(u,v)=\frac{1}{(ad-bc)t^2+(a+d)t+1}((ad-bc)t+a)x+by,\ cx+((ad-bc)t+d)y)$$

(a,b,c,d)=(1,-2,1,-3)

Example2. Exponential condition

$$u(x,y,t)=\frac{3}{4}-\frac{1}{4(1+e^{((-4x+4y-t)R/32)})}$$

$$v(x,y,t)=\frac{3}{4}+\frac{1}{4(1+e^{((-4x+4y-t)R/32)})}$$

- 모델 결과의 신뢰성을 위해 analytical solution을 구할 수 있는 condition들로 학습을 진행
- Linear condition : shock 발생x
- Exponential condition: shcck 발생x
- Riemann problem: shock 발생o

Example3. Riemann problem (U_L=2, U_R=0, v=0)

$u(x,0)=\begin{cases}u_L & x<0\\ u_R & x>0\end{cases}$과 같이 초기조건을 주었을 때를 Riemann problem이라 한다.

이는 다음과 같이 나눌 수 있다.

1) $u_L > u_R$일 때

이를 만족하는 weak solution은 아래와 같이 유일하다.

$$u(x,t)=\begin{cases}u_L & x<st\\ u_R & x>st\end{cases}, \text{ where } s=\frac{u_L+u_R}{2}$$

이는 임의의 시간 t에 대해 $x=st$일 때 불연속성이 나타남을 알 수 있다.

Conditions – points domain

- Example 1 : $x \in [0,1],\ y \in [0,1],\ t \in [0,1]$
- Example 2 : $x \in [0,0.5],\ y \in [0,0.5],\ t \in [0,1]$
- Example 3 : $x \in [-1,1],\ y \in [-1,1],\ t \in [0,1]$

- Google Colab 사용
- Pytorch 라이브러리 활용
- Reference : DeepXDE (arXiv:1907.04502v2 [cs.LG] 14 Feb 2020)
- 학습 결과 확인시 $u(x, y, t)$의 결과를 주로 확인

Chapter 2

Main ideas

Index

- **utils.py**
    1. point_sampler
    2. validate_visualization
- **main**
    1. linear_condition, known_condition, unknown_condition
    2. loss_fn
    3. RAR
    4. Train – Adam
    5. Train – LBFGS

자세한 코드 설명은 첨부 파일 주석 참고 부탁드립니다

- collocation point 및 boundary, initial 데이터 포인트를 생성하는 함수
  - collocation point : 주어진 x,y,t 범위 내의 랜덤한 포인트
  - initial point : (x,y,0)
  - boundary point : 의도한 경계조건에 부합하는 포인트

- collocation point 생성시 initial point 및 boundary point가 포함되도록 한다
  -initial/boundary point들도 PDE loss 계산에 영향이 있도록 의도

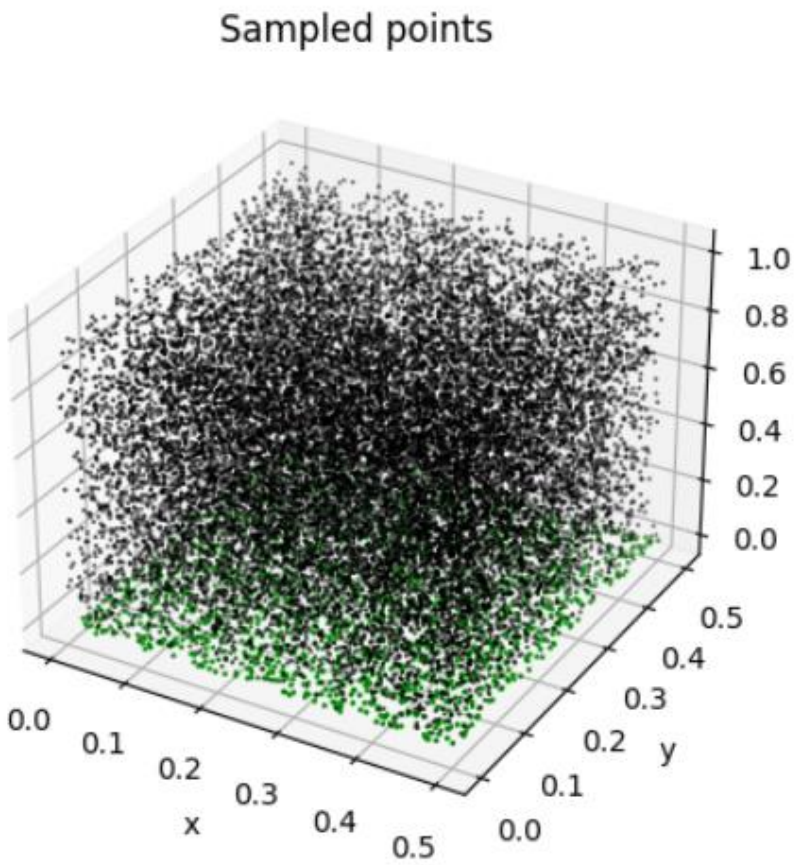
- initial/boundary point 개수는 collocation point 개수의 약 10%~20% 정도가 가장 학습에 적합하였음

- DeepXDE 논문을 참고하면 포인트를 매 epoch마다 새로 sampling 하거나 처음에 sampling 하는 것을 전략적으로 선택할 수 있다고 언급되어 있다 -> 처음에 sampling 된 것을 계속 training에 사용하기로 결정

입력예시

```
PDE Dimension(excluding t) : 2
Number of collocation points : 10000
x_min : 0
x_max : 1
test_x_grid_num : 50
y_min : 0
y_max : 1
test_y_grid_num : 50
t_max : 1
test_t_grid_num : 50
cha_num: 50
Number of boundary conditions: 4
Boundary 1
Boundary for x or y? (x/y) : x
Boundary1 : x = 0
Boundary 2
Boundary for x or y? (x/y) : x
Boundary2 : x = 1
Boundary 3
Boundary for x or y? (x/y) : y
Boundary3 : y = 0
Boundary 4
Boundary for x or y? (x/y) : y
Boundary4 : y = 1
boundary/initial points rato (%) : 20
torch.Size([2000, 3])
Total sampled points : 20000
```

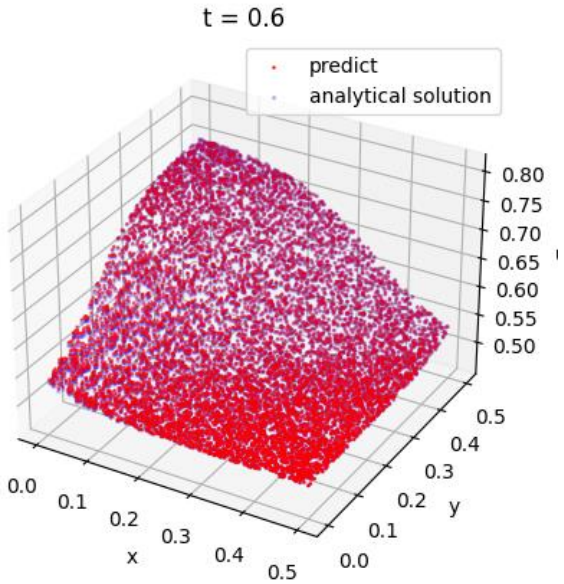
결과



- 모델 학습시 적절한 epoch step 마다 모델을 validation 할 때 사용

- 지정한 x,y 범위 및 원하는 time point 에서 모델의 예측값과 analytical solution 결과값을 동시에 plot

예시



- Neural network를 정의
- 활성화 함수 : Tanh
- hidden layer 들의 feature 개수는 (N,N) 형태로 통일 (모든 hidden layer의 node 개수 동일)
- optuna 라이브러리를 사용하여 최적의 network 구조를 선별

- 기계 학습을 위해 설계된 자동 hyperparameter 최적화 소프트웨어 프레임워크
- 파라미터의 범위를 지정하거나, 파라미터가 될 수 있는 목록을 설정하면 매 Trial 마다 파라미터를 변경하면서, 최적의 파라미터를 찾는다
- Neural Network 의 hidden layer 개수, feature 개수를 다양하게 trial 해 보아 최적의 모델 구조를 찾음
- 구현한 학습 방법 중 상수 parameter를 다양하게 trial 해 보아 최적의 상수 값을 찾음

- PDE에 적용되는 initial condition, boundary condition의 적용을 위한 클래스
- Analytical solution (exact solution)이 존재할 때는 Known_condition을 사용
- Linear_condition은 initial condition이 $u_0 = ax + by,\ v_0 = cx + dy$ 의 형태로 사용될 때 편의를 위해 만듦
- Analytical solution이 없는 경우 Unknown_condition에서 직접 condition들을 지정

- 학습에 필요한 오차함수를 정의 및 역전파를 위한 오차를 계산
- PDE loss + initial condition loss (+ boundary loss)
- PDE loss 계산시 pytorch 자동미분기능을 사용
- $f(x,y,t) = g(x,y,t)$ 형태의 PDE 학습 보다 $h(x,y,t) = 0$ 형태의 PDE 학습이 더 빠른 학습속도를 보임
- 기본 오차함수는 MSE(mean squared error)
- MSE(PDE out , 0) + MSE(initial out, initial condition) 값을 반환

- 특정 epoch마다 일정 수의 random sample한 points들에 대해 pde_loss를 각각 계산하고 등간격의 test_dict에서 구한 pde_loss_0와 비교하여 pde_loss>pde_loss_0 인 점들을 training points로 새로 받아들임

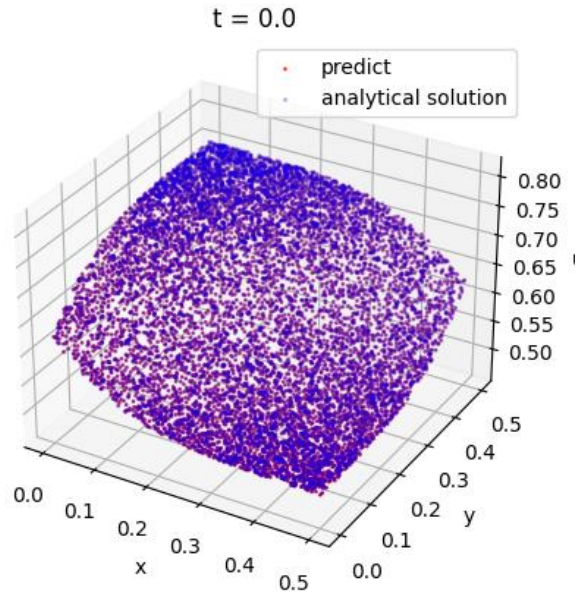
- 특정 epoch마다 계속 반복하여 RAR을 적용한다면, epoch = n에 대해 training시키는 시간 복잡도가 O(n), 전체 training에 필요한 시간 복잡도는 O(n^2)이 된다. 따라서, epoch가 지날 때 마다 더 적은 수의 sample을 선택하는 방법들을 고안하였다.

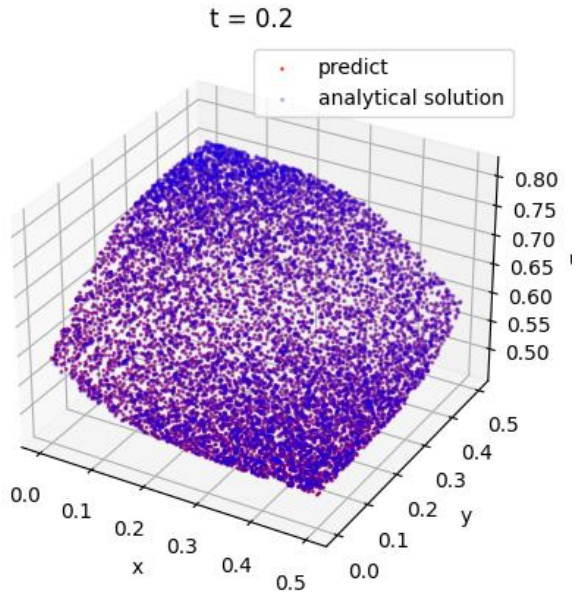
- constant: 특정 epoch마다 동일한 수의 random sample 생성
- fraction: 특정 epoch =n마다 (1/n)에 비례한 수의 random sample 생성
- Exponential: 특정 epoch=n 마다 2^(-n)에 비례한 수의 random sample 생성
- Zero: w/o RAR

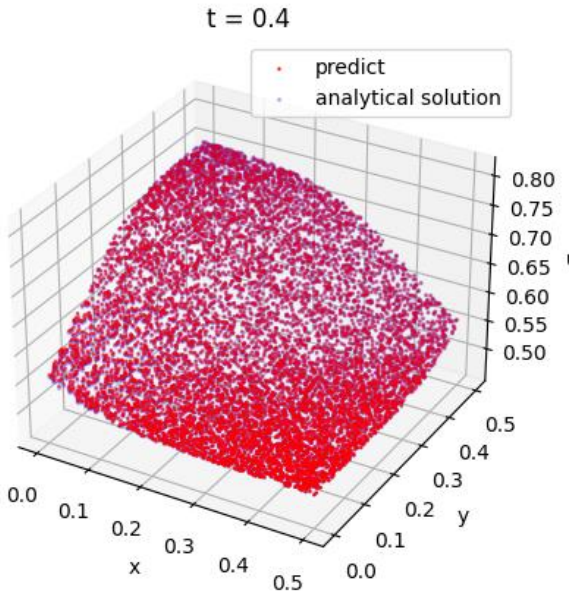
Optimizer - LBFGS

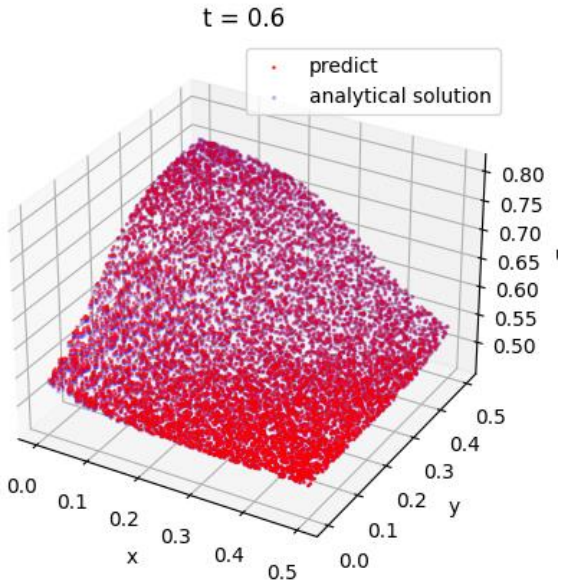
- Adam 보다 빠른 학습속도를 보임
- batch 단위 없이 모든 데이터를 한번에 학습하는 것이 특징
- 그러나 LBFGS 알고리즘 상 local minimum에 빠져 나오지 못하는 문제가 존재하여 학습이 매번 성공하지는 않았음
- 어느정도 간단한 PDE의 학습에서는 Adam 보다 빠르고 정확한 결과를 보여준다
- 학습한 time 범위 밖의 time 값에 대해서도 어느정도 좋은 정확성을 보임

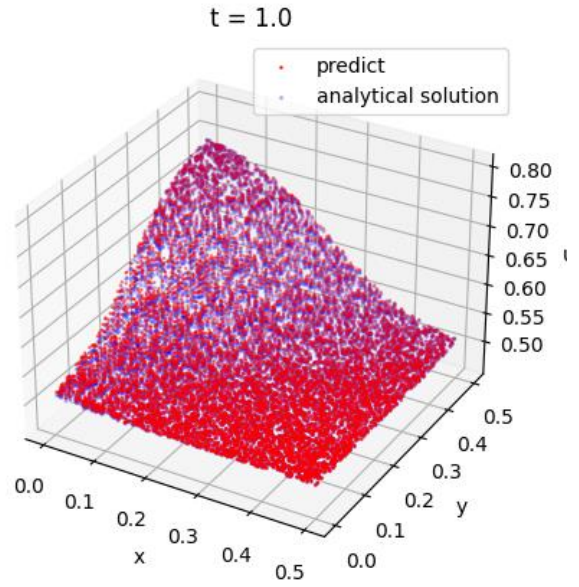
main – Train(LBFGS) 결과 (exponential condition)

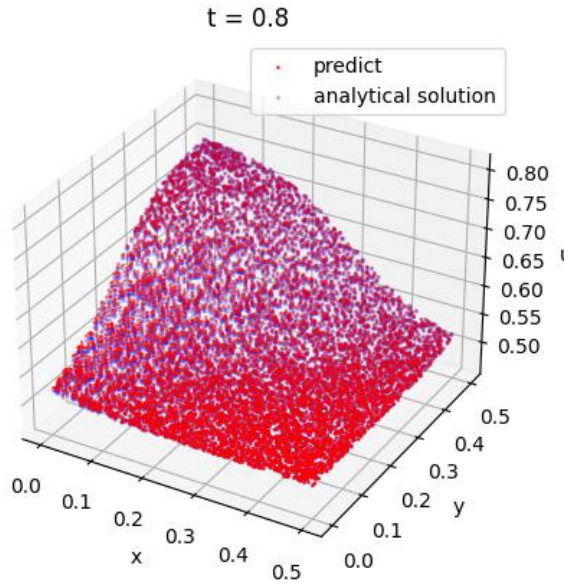










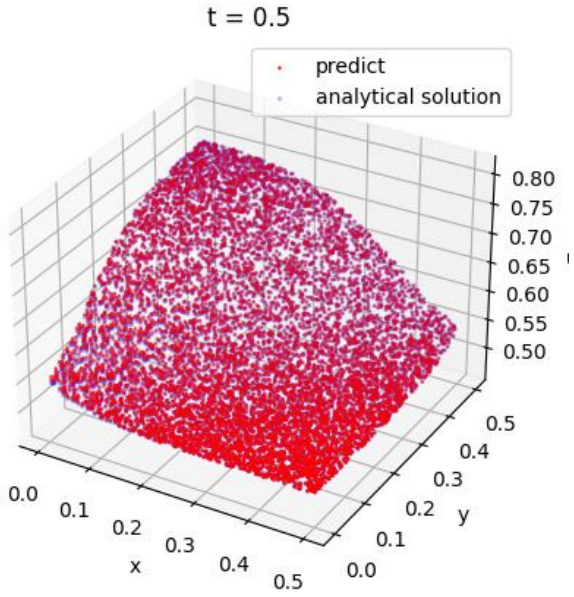


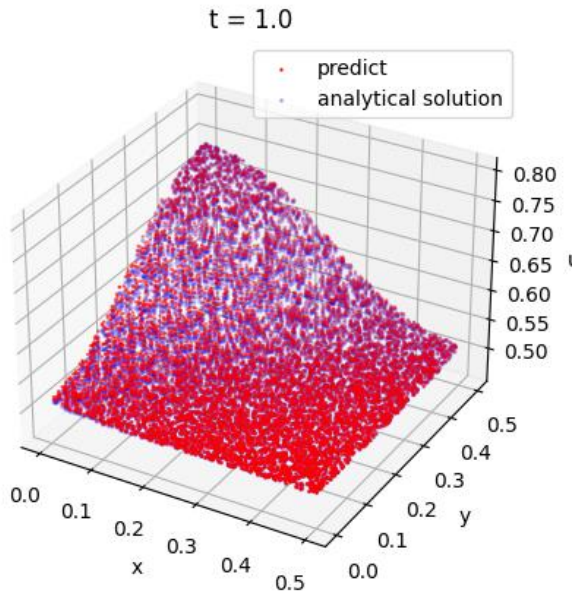
main – Train(LBFGS) 결과 (exponential condition)

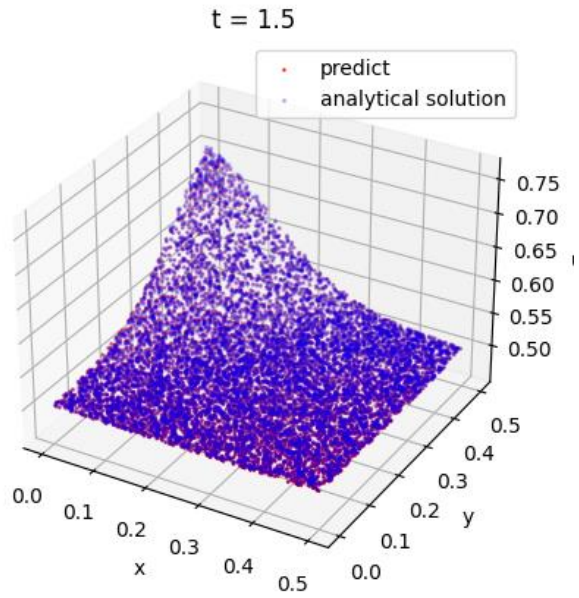
- 중심점 및 t = 0.05 (임의의 값) 대입하여 analytical solution과 예측 값 비교

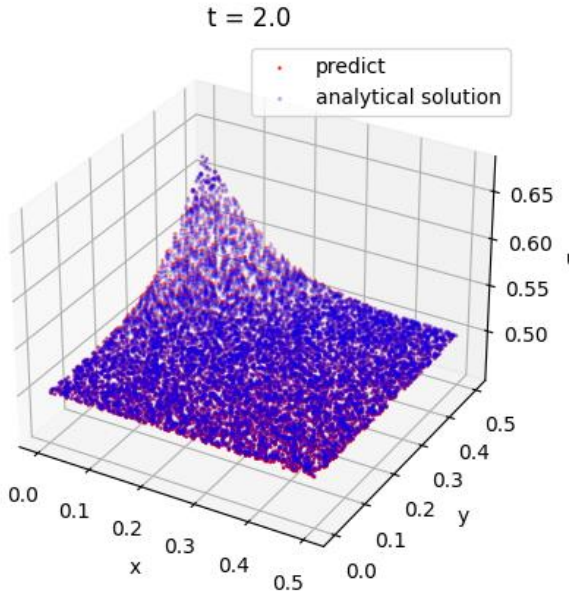
```
x,y,t : 0.25,0.25,0.05
PREDICTED : tensor([[0.6154, 0.8846]])
ANALYTICAL : tensor([[0.6153, 0.8847]])
```

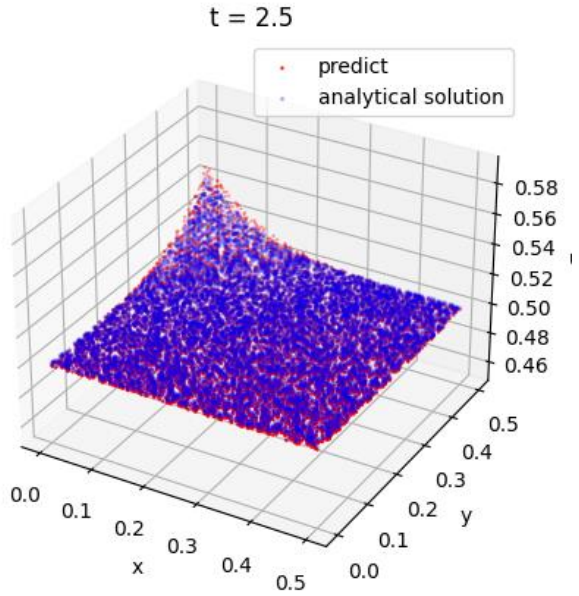
main – Train(LBFGS) 결과 (time > 1)

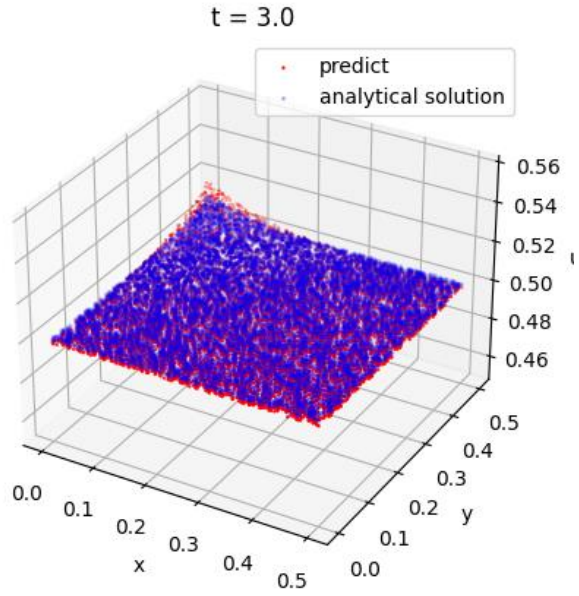












Optimizer - ADAM

- Inviscid form burger's equation에서 characteristic method를 통해 다음과 같은 성질이 밝혀졌다.

편미분 방정식의 해 u가 smooth하다고 가정할 때, 특정한 점 x_0, t_0에 대해
u가 $u(x_0, y_0)$로 일정해지는 곡선$(x(t), t)$를 생각할 수 있다. 이러한 곡선을 (x_0, t_0)의
characteristics이라 하며, 이를 이용하여 편미분 방정식을 비교적 쉽게 해결할 수 있다.

$$u_t + \mu u_x = 0$$

특정한 점 (x_0, t_0)에서의 characteristics를 $(x(t), t)$라 하자.
그렇다면 $u(x(t), t) = constant$이므로 양변을 t에 대해 미분하면 다음과 같은 식이 발생한다.

$$u(x(t), t) = constant \Leftrightarrow u_x x'(t) + u_t = 0$$

$x'(t) = u$라 한다면, 식이 성립한다.
즉 $x'(t) = u$, $x(t_0) = x_0$를 만족하는 $x(t)$는 (x_0, t_0)의 characteristic이다.
그런데 $x'(t) = u(x(t), t) = u(x_0, t_0) = constant$ 이므로 $x(t) = u(x_0, t_0)(t - t_0) + x_0$

$$\therefore \; u(u(x_0, t_0)(t - t_0) + x_0, t) = u(x_0, t_0)($$

- 2D Inviscid form burger's equation에서도 같은 방법을 이용하여 다음이 성립한다.

$$u(x_0, y_0, t_0) = u(u(x_0, y_0, t_0)(t - t_0) + x_0, v(x_0, y_0, t_0)(t - t_0) + y_0, t)$$

$$v(x_0, y_0, t_0) = v(u(x_0, y_0, t_0)(t - t_0) + x_0, v(x_0, y_0, t_0)(t - t_0) + y_0, t)$$

- Random sample을 생성하여 각 sample point의 characteristic line을 구하여 u, v값을 sample point의 u,v와 mse로 loss term 생성
- 만약 solution이라면 characteristic loss = 0이어야 한다.
- 특히, Initial/boundary point의 경우, 조건에 의해 함수값을 알고 있기 때문에 characteristic line은 정확하다.
- Lnitial/boundary points 와 그렇지 않은 일반적인 collocation points로 구분하여 loss를 생성한다.

(charac_loss_ini, charac_loss_col)

Example1, 3에 대해 시행하였다.

step1. define_model의 depth,(1~30) feature(1~50) 수, loss_fn에서 loss_ratio(initial_loss와 pde_loss와의 비율) (0~1)
Cha_loss_fn의 loss값들에 곱해지는 상수 cha_col, loss_ratio_cha_ini가 어떤 조건에서 모델의 성능이 좋은지
Obtuna를 통해 hyperparameter tunin을 시행한다. (epoch = 20, w/o RAR)

step2. RAR의 방법들 (zero, exponential, constant, fraction)에 대해 obtuna를 이용하여 테스트한다.
(epoch=20, 10번씩 반복) 반복하였을 때, 유의미한 차이를 가진다 판단하면 그 방법을 채택한다. 그렇지
않다면, 시간복잡도가 작은순 (zero<exponential<fraction<constant)로 채택한다.

step3. 1, 2에서 tunin한 hyperparameter를 바탕으로 10000epoch trainin하여 결과를 비교한다.

Adam-Result(Step1)

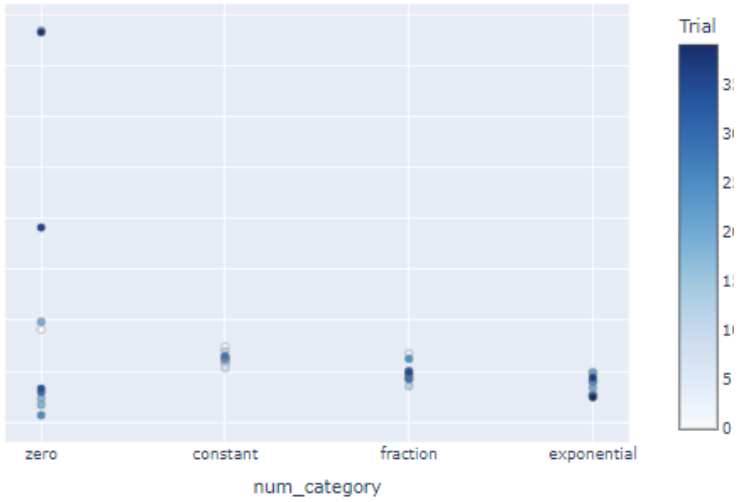
1000번 trial을 시행했었고, 구글 코랩 런타임 문제로
Example1의 경우 320번째 trial을 마치고 강제 종료되었다.
Best parameter
depth: 10
feature: 48
loss_ratio: 0.9276
loss_ratio_cha_col: 2.41e-08
loss_ratio_cha_ini: 1.34e-07

Example3의 경우 385번째 trial을 마치고 강제 종료되었다.
depth: 24
feature: 50
loss_ratio: 0.89
loss_ratio_cha_col: 0.46
loss_ratio_cha_ini: 0.67

RAR을 사용하지 않은 것(zero)보다 사용한 것이 좋은 성과를 나타내며
Constant, fraction, exponential은 모두 비슷한 값을 보였다.

Example1

Example2



Adam-Result(Step3)

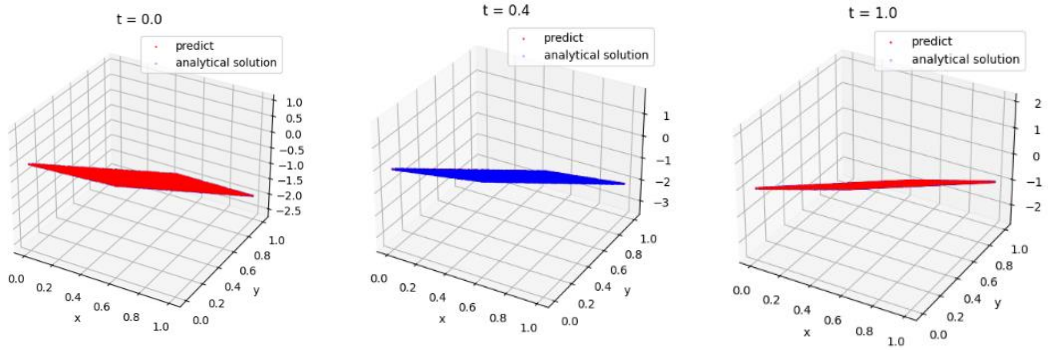
RAR (num_category="exponential)을 적용하여, 10000epoch, 9000epoch 시행하였다.

Example1(10000epoch)

```
test_model = torch.load('./pt/2D_BURGERS_EPOCH_LAST_linear_condition.pt', map_location = 'cpu')
total_error = real_test(test_dict, test_model, 1, loss_fn, condition, device)
```

```
tensor(6.1674e-06, grad_fn=<MseLossBackward0>)
```

L2_error(u) + L2_error(v) /2 = 6.1e-06

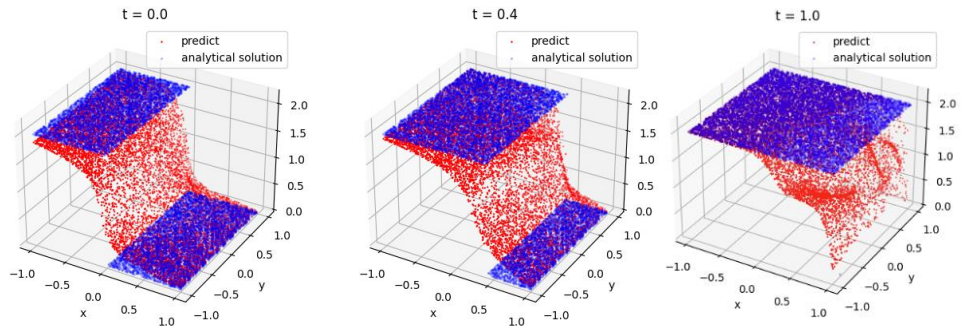


Example3(9000epoch)

```
test_model = torch.load('./pt/2D_BURGERS_EPOCH_9000.pt', map_location = 'cpu')
total_error = real_test(test_dict, test_model, 1, loss_fn, condition, device)
```

```
tensor(0.0797, grad_fn=<MseLossBackward0>)
```

L2_error(u) + L2_error(v) /2 = 0.0797



Conclusion

1. A = initial/boundary_loss와 B = pde_loss는 다른 방법으로 계산되어진다. (mse vs error of equation)
따라서 training에 있어서 동등하게 더하는 것보다 다른 비율로 더해는 것에 대해 고려해야한다.
pA + (1-p)B에서 p는 Example1, 3에서 각각 0.93, 0.89 가 best parameter로 나타났다.

2. characteristic_loss는 이 대회에서 처음 공개된 형식의 loss로 inviscid burger's equation에서 사용할 수 있는 방법이다.
characteristic_loss에 곱해지는 상수 loss_ratio_cha_col, loss_ratio_cha_ini는 Example1에서 2.41e-8, 1.34e-7
Example2에서는 0.46, 0.67이 best paramete였다. Example1과 같이 연속적인 smooth한 해를 구할 때는 characteristic loss를 쓰지 않는 것이 좋은 성능을 낸다고 볼 수 있다. 하지만, Example3과 같이 shock을 발생시키는 조건에서는, 미분계수가 발산하는 경우가 생기기 때문에 단순히 pde_loss를 이용하여 학습하기는 어려울 수 있다. 이런 상황에서 characteristic loss를 사용하는 것이 모델의 성능을 높일 수 있다.

3. characteristic method를 활용하여 PINN의 성능을 높일 수 있음을 보였다. 이처럼 neural network를 포함한 AI는 loss function을 이용하기 때문에, 특정 편미분방정식에 대해 밝혀진 성질들을 잘 이해한다면, 이를 이용한 loss term을 만들어 model의 성능을 높일 수 있다.

4. RAR을 사용하지 않는 것 보다 RAR을 사용하는 것이 더 좋은 성능을 보였다. Exponential, constant, fraction에 대해서는 크게 차이를 보이지 않았다. 시간복잡도를 고려하였을 때 특정 epoch마다 sample point를 더하는 것은 비효율적일 수 있다.

5. 편미분 방정식은, 초기조건이 조금이라도 바뀐다면 해가 크게 바뀌는 경우가 존재할 수 있다. 대부분의 수치해석적 방법에서는 초기/경계조건이 명확하게 지켜지나, PINN에서는 조금이나마 지켜지지 않은 경우가 있다. 이는 특정상황에서 학습이 제대로 안될 가능성이 있다.

- 시간 제약으로 충분한 sample과 trial을 거치지 못하였다. 때문에 추후에 step1과 step2에 더 많은 trial과 sample을 통해 검증이 필요하다. Step3에서는 Example3에서 9000epoch 학습한 모델의 L2_error값이 0.079인데 Step2실험에서 epoch20으로 학습한 것 중 가장 좋게 나온 것이 0.063이었다. 학습이 잘 이루어지지 않았으며, 시간 제약상 1회만 진행한 실험이기 때문에 더 많은 sample과 trial로 추가 실험이 필요하다.

- Step2에서 사실, Zero와 나머지를 비교하기 위해선 나머지 방식에서 추가된 sample point만큼 Zero에서 처음부터 random하게 선택된 점들을 포함하는 것이 맞으나, 시간 제약으로 이를 고려한 실험을 하지 못하였다. 즉, RAR의 성능이 단순히 sample수를 늘렸기 때문에 성능이 좋은 것 뿐일 가능성을 배제할 수 없다. 이를 위한 추가적인 실험이 필요하다.

- 적은 Example로의 실험밖에 진행하지 못하였다. Shock인 경우와 그렇지 않은 solution의 경우로 나누어 여러 상황에서 실험을 반복하여 각 상황에서 loss_ratio나 loss_ratio_cha값에 대한 검증이 필요하다.

- Weak solution이 여러 개 발생하는 경우에 대해 다루지 못하였다. 대표적으로 Example3의 Riemann problem의 경우 만약 조건을 U_L < U_R이 되게 바꾼다면 해가 여러 개 발생할 수 있게 된다. 그 중에서 오직 하나의 해를 제외하고는 모두 물리적으로 의미가 없는 해가 되는데, 이러한 경우 모델이 물리적으로 의미가 있는 해에 근사할 수 있도록 하는 실험을 제안할 수 있다.특히 이에 대해서 학습이 잘 안 될 때, 물리적으로 의미가 있는 해를 판별하는 entropy condition을 이용하여 새로운 loss term을 만드는 방법을 생각할 수 있을 것이다.

- 초기, 경계조건을 명확히 지키도록 model을 새로 구축하는 방법에 대해 추가 연구를 진행하면 좋을 것이다.