Airfran 데이터

simulation is given as a point cloud defined via the nodes of the simulation mesh. Each point of a point cloud is described via 7 features: its position (in meters), the inlet velocity (two components in meter per second), the distance to the airfoil (one component in meter), and the normals (two components in meter, set to 0 if the point is not on the airfoil).

Each point is given a target of 4 components for the underlying regression task: the velocity (two components in meter per second), the pressure divided by the specific mass (one component in meter squared per second squared), the turbulent kinematic viscosity (one component in meter squared per second).

Finally, a boolean is attached to each point to inform if this point lies on the airfoil or not.

The output is a tuple of a list of np.ndarray of shape *(N, 7 + 4 + 1)*, where N is the number of points in each simulation and where the features are ordered as presented in this documentation, and a list of name for the each corresponding simulation.

We highly recommend to handle those data with the help of a Geometric Deep Learning library such as PyTorch Geometric or Deep Graph Library.

Dataset Dataloader

- https://github.com/eric1645/3-2/blob/main/dataset_dataloader.ipynb

Dataset Dataloader

```python
class AirfRANSDataset(Dataset):

    def __init__(self, root, task='scarce', train=True):
        #초기화
        self.root = root
        self.task = task
        self.train = train

        taskk = 'full' if task == 'scarce' and not train else task
        split = 'train' if train else 'test'

        #manifest읽기
        with open(os.path.join(root, 'manifest.json'), 'r') as f:
            manifest = json.load(f)[f"{taskk}_{split}"]

        self.names = manifest
        self.samples = []
        for name in tqdm(manifest, desc=f'Loading AirfRANS ({taskk}, {split})'):
            sim = Simulation(root=root, name=name)     #Simulation 객체 생성

            inlet_velocity = (
                np.array([np.cos(sim.angle_of_attack), np.sin(sim.angle_of_attack)])
                * sim.inlet_velocity  #inlet velocity를 받음각에 따라 분해
            ).reshape(1, 2) * np.ones_like(sim.sdf)

            #데이터 순서대로 모으기
            data = np.concatenate([
                sim.position,
                inlet_velocity,
                sim.sdf,
                sim.normals,
                sim.velocity,
                sim.pressure,
                sim.nu_t,
                sim.surface.reshape(-1, 1)
            ], axis=-1)

            self.samples.append(torch.tensor(data, dtype=torch.float32)) #tensor 변환 후 저장
```

```python
    #데이터 반환
    def __getitem__(self, idx):
        data = self.samples[idx]
        name = self.names[idx]

        x = data[:, :7] #input
        y = data[:, 7:11] #output

        return x, y, name

root = r"C:\airfran\Dataset"
dataset = AirfRANSDataset(root, task='scarce', train=True)
```

```
Loading AirfRANS (scarce, train): 100%|██████████| 200/200 [00:40<00:00,  4.88it/s]
```

Dataset Dataloader

```python
loader = DataLoader(dataset, batch_size=1, shuffle=True)

for x, y, name in loader:
    print(name, x.shape, y.shape)
    break
```

```
('airFoil2D_SST_66.62_-0.743_0.839_4.397_14.875',) torch.Size([1, 178475, 7]) torch.Size([1, 178475, 4])
```

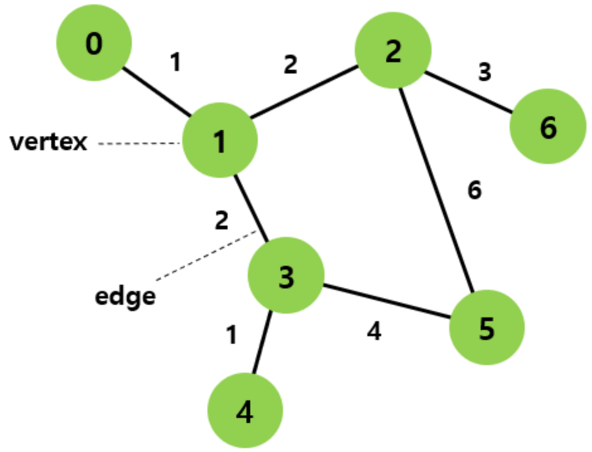
```
('airFoil2D_SST_62.893_-0.786_3.167_0.0_17.071',) torch.Size([1, 180572, 7]) torch.Size([1, 180572, 4])
```

```
('airFoil2D_SST_72.844_3.105_2.426_4.619_19.695',) torch.Size([1, 174181, 7]) torch.Size([1, 174181, 4])
```

GNN(Graph Neural Network)

- 그래프 형태의 데이터에서 노드 간 연결 관계와 특징을 함께 학습하는 신경망.

- 복잡한 구조를 가진 패턴(분자 구조, 사회망 등)을 효과적으로 학습할 수 있다.



https://miro.medium.com/max/488/0*UgMHEDLriw2efXbx

GAT(Graph Attention Network)

- 그래프 신경망(GNN)의 한 종류.
- 그래프 데이터 구조에서 어텐션(attention) 메커니즘을 적용해 각 이웃 노드들이 서로 다른 weight를 가지도록 함.

GAT(Graph Attention Network)

- https://github.com/eric1645/3-2/blob/main/gat_dataset.ipynb

GAT(Graph Attention Network)

```python
class AirfRANSGATDataset(Dataset):
    def __init__(self, root, task='scarce', train=True, k=10):
```

```python
#그래프 edge 생성
edge_index = self._build_knn_graph(sim.position, k=self.k)

self.graphs.append(Data(x=x, y=y, edge_index=edge_index))
```

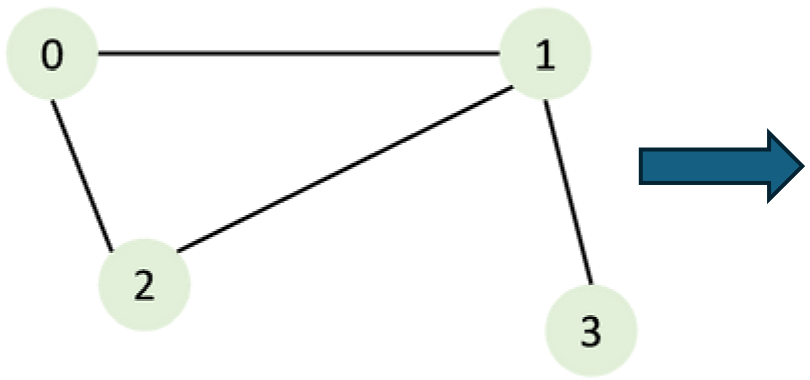
```python
def _build_knn_graph(self, pos, k=10):
    #(N, 2) numpy array를 edge_index tensor (2, E)로 반환
    from sklearn.neighbors import NearestNeighbors
    nbrs = NearestNeighbors(n_neighbors=k + 1).fit(pos)
    distances, indices = nbrs.kneighbors(pos)

    # i → j edges
    src, dst = [], []
    for i in range(len(indices)):
        for j in indices[i][1:]:  # skip self
            src.append(i)
            dst.append(j)
    edge_index = torch.tensor([src, dst], dtype=torch.long)
    return edge_index
```

```python
root = r"C:\airfran\Dataset"
dataset = AirfRANSGATDataset(root, task='scarce', train=True, k=10)
loader = DataLoader(dataset, batch_size=1, shuffle=True)

for batch in loader:
    print(batch.x.shape, batch.y.shape, batch.edge_index.shape)
    break
```

```
Loading AirfRANS (scarce, train): 100%|██████████| 200/200 [03:22<00:00,  1.01s/it]
torch.Size([179064, 7]) torch.Size([179064, 4]) torch.Size([2, 1790640])
```




```
[0, 0, 1, 1]
[1, 2, 2, 3]
```

https://chamdom.blog/static/43dcc5ebdae930f808c5563ac31f4159/c5bb3/directed-and-undirected.png

```python
import torch
from torch_geometric.nn import GATConv

class SimpleGAT(torch.nn.Module):
    def __init__(self, in_channels=7, hidden_channels=64, out_channels=4, heads=4):
        super().__init__()
        self.gat1 = GATConv(in_channels, hidden_channels, heads=heads, concat=True) #첫번째 레이어
        self.gat2 = GATConv(hidden_channels * heads, out_channels, heads=1, concat=False) #두번째 레이어

    def forward(self, x, edge_index):
        x = torch.relu(self.gat1(x, edge_index))
        x = self.gat2(x, edge_index)
        return x

model = SimpleGAT()
for batch in loader:
    pred = model(batch.x, batch.edge_index)
    print(pred.shape)  # (N, 4)
    break
```

```
torch.Size([180442, 4])
```

Evaluation

- **ML-related**: standard ML metrics(e.g. MAE, RMSE, etc) and speed-up with respect to the reference solution computational time.
- **Physical compliance**: respect of underlying physical laws
- **Application-based context**: out-of-distribution (OOD) generalization to extrapolate over minimal variations of the problem depending on the application; speed-up. A solution may perform well in standard machine learning related

$$\text{Score} = \alpha_{ML} \times \text{Score}_{ML} + \alpha_{OOD} \times \text{Score}_{OOD} + \alpha_{Physics} \times \text{Score}_{Physics}$$

ML Score

$$\text{Score}_{ML} = \alpha_A \times \text{Score}_{Accuracy} + \alpha_S \times \text{Score}_{Speed}$$

$$\text{Score}_{Accuracy} = \frac{1}{2N}(2 \times Ng + 1 \times No + 0 \times Nr)$$

N_r :the number of unacceptable results overall
N_o :the number of acceptable results overall
N_g :the number of great results overall
$N = N_r + N_o + N_o$

$$Score_{Speed} = \min\left(\frac{\log_{10}(SpeedUp)}{\log_{10}(SpeedUpMax)}, 1\right)$$

$$Score_{SpeedUp} = \frac{time_{ClassicalSolver}}{time_{Inference}}$$

SpeedUpMax is the maximal speed up allowed for the airfoil use case

OOD Score

- This sub-score will evaluate the capability of the learned model to predict OOD dataset.

- In the OOD testset, the input data are from a different distribution than those used for training.

- The computation of this sub-score is similar to $Score_{ML}$.

PHYSICS Score

- For the Physics compliance sub-score, we evaluate the relative errors of physical variables.

- For each criterion, the score is also calibrated based on 2 thresholds and gives 0/1/2 points, similarly to $Score_{Accuracy}$, depending on the result provided by the metric considered.

```
allmetrics={"ML":{
                 "x-velocity":0.03,
                 "y-velocity":0.03,
                 "pressure":0.01,
                 "turbulent_viscosity":0.08,
                 "pressure_surfacic":0.27
                 },
            "Physics":
                 {"spearman_correlation_drag":0.2,
                  "spearman_correlation_lift":0.6,
                  "mean_relative_drag":0.18,
                  "mean_relative_lift":0.25,
                  },
            "OOD":
                 {
                  "x-velocity":0.08,
                  "y-velocity":0.07,
                  "pressure":0.055,
                  "turbulent_viscosity":0.06,
                  "pressure_surfacic":0.45,
                  "spearman_correlation_drag":0.1,
                  "spearman_correlation_lift":0.6,
                  "mean_relative_drag":0.28,
                  "mean_relative_lift":0.35,
                  }
           }

speedUp= {"ML":1300,"OOD":1300}
```

Spearman correlation

- 예측값과 실제값 간 순위의 상관관계를 측정.

- ρ가 1이면 예측 순서가 실제 순서와 완전히 일치
- ρ가 -1이면 예측 순서가 실제 순서와 완전히 반대

- 절대적인 값을 맞추지 못하더라도 상대적인 순위도 중요.

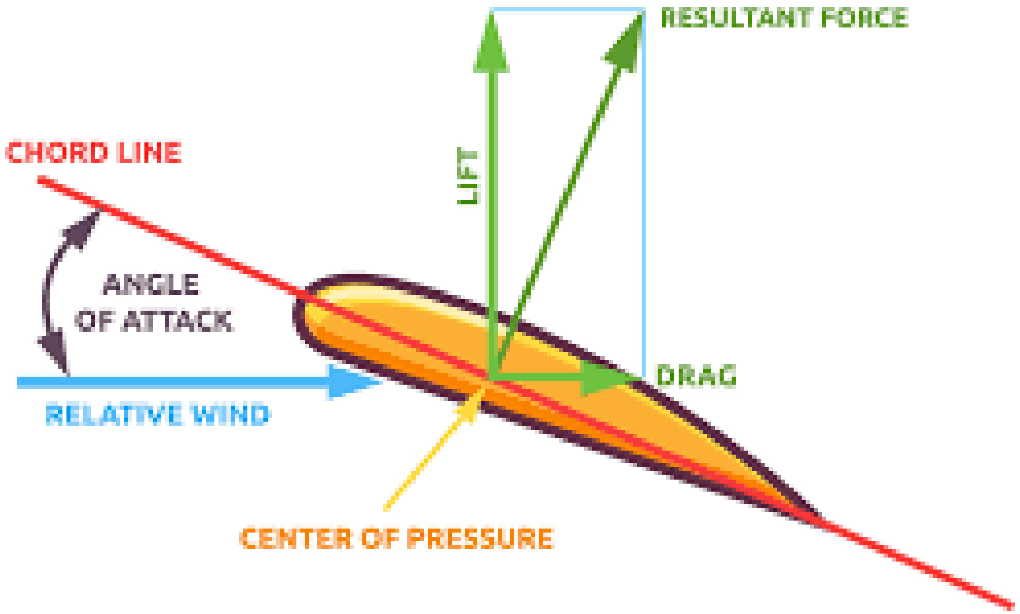
Airfoil 관련 물리량

- **Lift(양력, $\boldsymbol{C_L}$)**
- **Drag(항력, $\boldsymbol{C_D}$)**
- Lift-to-drag ratio(C_L, C_D)
- Pressure Coefficient(C_P)
- Friction Coefficient(C_f)
- Boundary Layer Thickness(δ)

양력, 항력

- $L = -\int_A p sin\theta dA$
- $D = -\int_A p cos\theta dA$

- $C_L = \frac{L}{\frac{1}{2}\rho V^2 A}$, $C_D = \frac{D}{\frac{1}{2}\rho V^2 A}$



https://mwi-inc.com/blog-post/what-is-an-airfoil-and-whats-its-purpose/