Wind Farm Power prediction with Graph Neural Network

Junyoung Park
SYSTEMS INTELLIGENCE Lab
Industrial and Systems Engineering (ISysE)
Wind Farm Power Estimation Task

Wind Farm Layout

Wind direction 1

Wind Farm Layout

Wind direction 2
Wind Farm Power Estimation Task

- Farm-level power estimation
  Wind-farm power = ??
- Turbine-level power estimation
  Wind turbine powers = ??
Wind Farm Power Estimation Task

- Farm-level power estimation
  Wind-farm power = ??

- Turbine-level power estimation
  Wind turbine powers = ??
Wind Farm and Its Graph Representation

\[ G = (N, E, g) \]

Node features \( N = \{\text{free flow wind speed}\}_{\forall i \in \text{turbine index}} \)

Edge features \( E = \{ \left( \text{the down–stream wake distance } d, \text{ the radial–wake distance } r \right) \}_{\forall (i,j)}^* \)

Global features \( g = \{\text{free flow wind speed}\} \)

\( i, j \) are turbine index

\* \( \forall (i, j) \in \text{interaction turbines} \)
Details on Edge Features

\[ \mathcal{G} = (N, E, g) \]

Edge features \( E \) = \{ the down-stream wake distance \( d \), the radial-wake distance \( r \) \} \(_{V(i,j)} \)
Neural network is a function approximator that has trainable parameter $\theta$ such $y \approx \hat{y}$ as accurate as possible.

$$\hat{y} = NeuralNetwork(x; \theta)$$
Why Graph Representation?

\[ G = (N, E, g) \]

Matrix (Tensor) Representations

<table>
<thead>
<tr>
<th></th>
<th>X coord.</th>
<th>Y coord.</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>850</td>
<td>713</td>
</tr>
<tr>
<td>T1</td>
<td>303</td>
<td>587</td>
</tr>
<tr>
<td>T2</td>
<td>569</td>
<td>775</td>
</tr>
<tr>
<td>T3</td>
<td>642</td>
<td>290</td>
</tr>
<tr>
<td>T4</td>
<td>217</td>
<td>97</td>
</tr>
</tbody>
</table>
Why Graph Representation?

1. MLP/CNN’s input size tends to be fixed.
   e.g.) MNSIT = [28 X 28]
   If we deploy one more turbine to the farm, then the input dimension would change

2. Input data has no natural order.
   e.g.) time-series has time index!
   Which turbine should be the first input?

<table>
<thead>
<tr>
<th></th>
<th>X coord.</th>
<th>Y coord.</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td>850</td>
<td>713</td>
</tr>
<tr>
<td>T1</td>
<td>303</td>
<td>587</td>
</tr>
<tr>
<td>T2</td>
<td>569</td>
<td>775</td>
</tr>
<tr>
<td>T3</td>
<td>642</td>
<td>290</td>
</tr>
<tr>
<td>T4</td>
<td>217</td>
<td>97</td>
</tr>
</tbody>
</table>
Spatial/Temporal Adjacency does not imply ‘related’

Convolution operation presumes that ‘Nearby pixels are somewhat related’. Since we *share* the convolution filters.

RNNs presumes that ‘Nearby inputs are somewhat related’. Since we *share* the RNN blocks.

Figure source <Left: https://github.com/vdumoulin/conv_arithmetic>, <Right: https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9>
Graph Neural Network

\[ g_y = \text{GraphNeuralNetwork} (g_x; \theta) \]

- Graph Convolution Networks (GCN)
- Attention based approaches
- Relational inductive bias (GN block)
- ...

Imposing Relational Inductive Bias

**Share** edge update function $f$ and node update function $g$ for updating graph represented data.
Imposing Relational Inductive Bias

Share edge update function $f$ and node update function $g$ for updating graph represented data.
Imposing Relational Inductive Bias

\[ f(\cdot) \quad \text{Edge update function} \]
\[ g(\cdot) \quad \text{Node update function} \]

**Share** edge update function \( f \) and node update function \( g \) for updating graph represented data
Imposing Relational Inductive Bias

Share edge update function $f$ and node update function $g$ for updating graph represented data.
Imposing Relational Inductive Bias

Share edge update function $f$ and node update function $g$ for updating graph represented data.
Physics-induced Graph Neural Network On Wind Power Estimations

Physics-induced Graph Neural Network (PGNN)

Wind farm graph $\mathcal{G}$

Physical properties
downstream-wake distance $d$, the radial-wake distance $r$

Power predictions of wind farm
GN (Graph Neural) Block

Input graph $\mathcal{G}$

Graph Neural (GN) Block

- Edge update network: $f(\cdot; \theta_0)$
- Node update network: $f(\cdot; \theta_1)$
- Global update network: $f(\cdot; \theta_2)$

Update graph $\mathcal{G}'$
GN Block – Edge update steps

\[ Edge'_{0,1} = f(Edge_{0,1} Node_1, Node_0, g; \theta_0) \]

Update edge features with \( f(Edge \text{ features}, Receiver \text{ features}, Sender \text{ features}, g; \theta_0) \)
GN Block – Edge update steps

Update edge features with $f(Edge\ features, Receive\ features, Sender\ features, g; \theta_0)$
GN Block – Edge update steps

Update edge features with $f(\text{Edge features}, \text{Receiver features}, \text{Sender features}, g; \theta_0)$
GN Block – Node update steps

Input graph $\mathcal{G}$

Updated edge features

$g$

Node’$_0 = f(\text{Edge}_0; \theta_1)$

$\text{Edge}_0 = \text{mean}(\text{concat}(\text{Edge}_{0,i}, \text{Node}_1, \text{Node}_i))$

$\forall i$ incoming edges

Aggregation function: any function obeys ‘input-order invariant’ and ‘input-number invariant’ properties. e.g., Mean, Max, Min, etc.
GN Block – Node update steps

Input graph $\mathcal{G}$

Updated edge features
GN Block – Global feature update

Input graph $g$

Updated edge features

$g' = f(Edge', Node', g; \theta_2)$

$Edge' = \text{mean}(Edge'_{i,j}) \forall\text{edges } i,j$

$Node' = \text{mean}(Node_i) \forall\text{nodes } i$
Revisit Aggregation Method

Input graph $\mathcal{G}$

Updated edge features

$Node'_0 = f(Edge_0; \theta_1)$

$Edge_0 = \text{mean}(\text{concat}(Edge_{0,i}, Node_1, Node_i))$

$\forall i$ incoming edges

**Aggregation function**: any function obeys ‘input-order invariant’ and ‘input-number invariant’ properties.

e.g., Mean, Max, Min, etc.
Weighted “__” ≈ Attention (in Deep Learning)

Figure source <Agile Amulet: Real-Time Salient Object Detection with Contextual Attention>
Consider weighted Aggregations

<Robot soccer>  <Visualized weights>

Figure source <Left: https://www.youtube.com/watch?v=HHIN0TDgllE>, <Right: VAIN: Attentional Multi-agent Predictive Modeling>
How can we get the weights?

Learn to weight!
GN Block – Edge update steps Revisit

Input graph $\mathcal{G}$

Update edge features with $f(Edge \ features, Receiver \ features, Sender \ features, g; \theta_0)$

$W_{4,1} = f(some \ possible \ inputs; \theta_3)$

$Edge'_4,1 = W_{4,1} \times f(Edge_{4,1} \ Node_4, Node_1, g; \theta_0)$
JK Park and K.H. Law suggest the continuous deficit factor $\delta u(d, r, \alpha)$ as

$$\delta u(d, r) = 2\alpha \left( \frac{R_0}{R_0 + \kappa d} \right)^2 \exp \left( - \left( \frac{r}{R_0 + \kappa d} \right)^2 \right)$$

* $R_0$: Rotor diameter
* $d$: Downstream wake distance
* $r$: Radial wake distance
* $\alpha, \kappa$: Tunable parameters

Figure source <Cooperative wind turbine control for maximizing wind farm power using sequential convex programming by Jinkyoo Park, Kincho H. Law>
Physics-induced Attention

\[ \delta u(d, r) = 2\alpha \left( \frac{R_0}{R_0 + \kappa d} \right)^2 \exp \left( - \left( \frac{r}{R_0 + \kappa d} \right)^2 \right) \]

\( \delta u(d, r) \) indicates
‘How much the down stream turbine get affected
Due to the upstream turbines’
\[ \rightarrow \text{Weighting Factor } W! \]

However, they tuned the parameters \( \alpha, \kappa \) to the observed data

Figure source <Cooperative wind turbine control for maximizing wind farm power using sequential convex programming by Jinkyo Park, Kincho H.Law>
Physics-induced Attention

Let neural network learn $\alpha, \kappa, R_0$!
Physics-induced Graph Neural Network On Wind Power Estimations
Physics-induced Graph Neural Network On Wind Power Estimations

Wind farm graph $\mathcal{G}$

Physical properties
- downstream-wake distance $d$
- radial-wake distance $r$

Power predictions of wind farm
Graph Dense Layer

\[ \hat{\mathbf{P}}_0 = f(N'_0; \theta_5) \]
Graph Dense Layer

\[ \hat{P}_0 = f(N'_0; \theta_5) \]
Graph Dense Layer

\[ \hat{P}_0 = f(N'_0; \theta_5) \]
Graph Dense Layer

\[ \hat{P}_0 = f(N'_0; \theta_5) \]
How to train your PGNN

\[ \hat{P} - P \]

We use mean-squared-error as a loss function of PGNN
Lovely but Dreadful Exponential functions

\[ f(x) = \exp(x) \]
Simple approximation for exponential functions

\[
\exp(x) := \sum_{k=0}^{\infty} \frac{x^k}{k!} \approx \sum_{k=0}^{D} \frac{x^k}{k!}
\]

We set \(D = 5\)
Bottom side of power-series approximation

The suggested approximation works (relatively) properly when $x$ is small.

Question?
“why don’t you use Taylor's expansion?”
Answer:
“You may encounter exponential again!”
Instead of using raw the down stream distance $d$, and the radial wake distance $r$ as inputs,

\[
\begin{align*}
d' &= \frac{d}{\sigma(d)} \times \max(0, s_d) \\
r' &= \frac{d}{\sigma(r)} \times \max(0, s_r)
\end{align*}
\]

$s_d, s_r$ are learnable parameters
Dissect Scale-only normalization

Instead of using raw the down stream distance $d$, and the radial wake distance $r$ as inputs,

$$d' = \frac{d}{\sigma(d)} \times \max(0, s_d)$$

1. Why do not subtract means?
   → We want the scaled values to be positive
2. What are $\max(0, s)$ for?
   → Since $s$’s are learnable parameters, w/o $\max(0, s)$ could be negative
3. How do you get $\sigma(\cdot)$?
   → We employed EWMA to get $\mu(\cdot), \sigma(\cdot)$ estimation
4. Why do you multiply $\max(0, s)$ again?
   → If not scaling was the best, then we can recover the original values.
   Same intuition Batch Normalization did.
Approximated weighting function

\[ f_w(\cdot ; \theta_3) \]

downstream-wake distance \( d \)
radial-wake distance \( r \)
Weight \( w \)

Normalized \( d \)
Scale-norm
Normalized \( r \)
Training Procedure

$n = \{5,10,15,20\}$

Sample wind-farm layout

Simulator

PGNN

# turbines $n$

Wind speed $S$

Wind direction $\theta$

Power simulations with FLORIS

Graph representation

Graph encoding

Sample $s \sim U(5.0\,m/s, 15.0\,m/s)$, $\theta \sim U(0^\circ, 360^\circ)$
Generalization Tests

- Wind farm layouts

Generalization over environmental factors:
- wind directions, wind speeds
- Generalization over wind farm layouts
Generalization Over Environmental Factors

Wind speed = 8.0 m/s

Error = 0.0172

Error = 0.022
- Sample *20 wind farm layouts* and Estimate average estimation errors.
- Each layout has 20 wind turbines in it.
Qualitative Analysis on Physics-induced Bias

\[ W_{4,1} = f(\text{inputs}; \theta_3) \]

\[ Edge'_{4,1} = W_{4,1} \times f(Edge_{4,1} \ Node_4, Node_1, g; \theta_0) \]

\[ f \text{ is another neural network} \]

\[ f(r, d; \alpha, \kappa, R_0) = 2\alpha \left( \frac{R_0}{R_0 + \kappa d} \right)^2 \exp \left( -\left( \frac{r}{R_0 + \kappa d} \right)^2 \right) \]
PGNN achieved 11% smaller validation error than DGNN.
Case Study on Inferred Weights

Wind direction: 0°
Wind direction: 90°
Wind direction: 180°
Wind direction: 270°

Physics-induced Weight
Data-induced Weight

Weight values  Ignored edges
Case Study on a Regularized Grid Layout

**PGNN**

<table>
<thead>
<tr>
<th>Wind farm y size</th>
<th>Wind farm x size</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td>4000</td>
</tr>
</tbody>
</table>

Wind speed: 6.0 m/s

**Error = 0.0642**

**DGNN**

<table>
<thead>
<tr>
<th>Wind farm y size</th>
<th>Wind farm x size</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td>3000</td>
</tr>
<tr>
<td></td>
<td>4000</td>
</tr>
</tbody>
</table>

Wind speed: 6.0 m/s

**Error = 0.0702**
Anyway the wind blows

Junyoung Park
SYSTEMS INTELLIGENCE Lab
Industrial and Systems Engineering (ISysE)

Normalizing powers