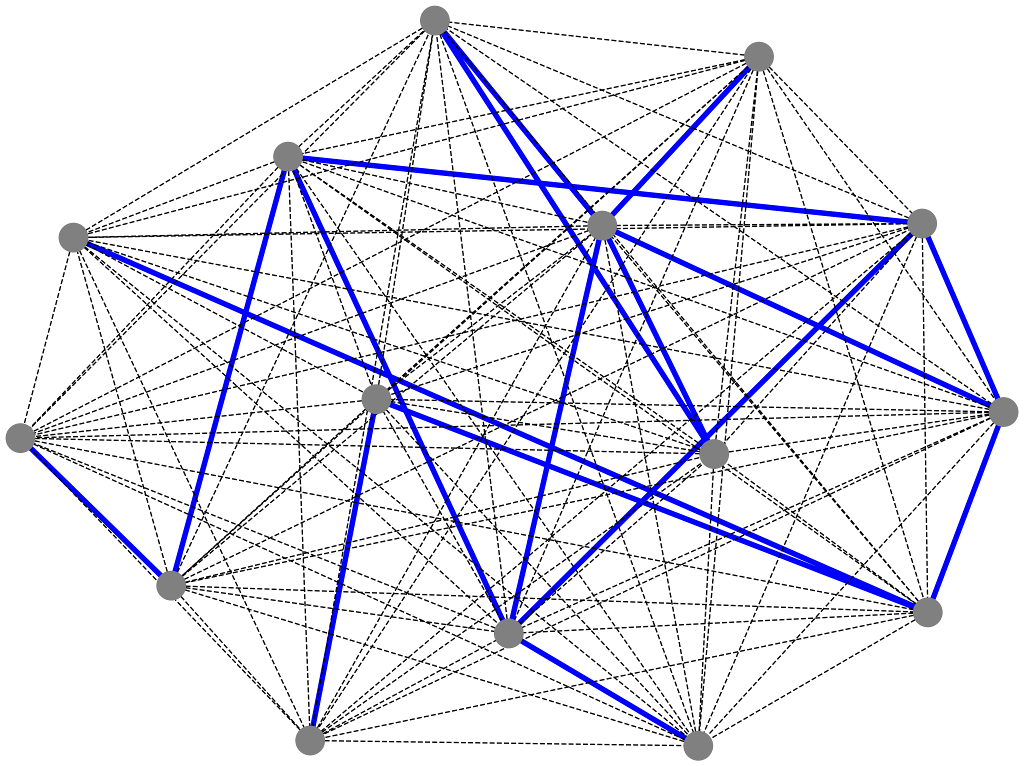
LMP Spread Modeling with Message Passing Neural Networks

William Colglazier



Zoomed in version of toy electricity graphs. With a fully connected graph (black lines) to model LMP spreads between all nodes, and a sparsely connected graph (blue lines) represents actual electricity lines with edge features.

Toy Dataset Generation using DC Power Flow Equation

We start by randomly generating per node variables Generation G_i, Load L_i, and per edge variables Reactance x_k and Line Capacity (limit). Then solve for the net injected power P_i at all nodes:

$$P_i = G_i - L_i$$

Givining an injection vector

$$\mathbf{P}_{\text{inj}} = \begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_n \end{bmatrix}$$

Given the branch reactances x_k, calculate the diagonal branch matrix

$$\mathbf{B}_{\text{branch}} = \text{diag}\left(\frac{1}{x_1}, \frac{1}{x_2}, \dots, \frac{1}{x_m}\right)$$

A denotes the branch-to-bus incidence matrix, which encodes the network's connectivity. Using this, the bus admittance matrix is defined as

$$\mathbf{B}_{\text{bus}} = \mathbf{A}^{\top}\mathbf{B}_{\text{branch}}\mathbf{A}$$

Solve for the voltage angles $\boldsymbol{\theta}$ using

$$\mathbf{B}_{\text{bus}}\boldsymbol{\theta} = \mathbf{P}_{\text{inj}}$$

This equation enforces Kirchhoff's Current Law (KCL) at every bus in the grid.

For each branch k between buses i and j, the power flow is:

$$f_k = \frac{\theta_i - \theta_j}{x_k}$$

The LMP spread is then calculated, which is used as our edge label between each pair of connected nodes.

$$\Delta\text{LMP}_{ij} = 1000(\theta_i - \theta_j) + 50 \cdot (0, |f_{ij}| - \text{limit}_{ij})$$

This accounts for both base LMP differences (from angle) and congestion penalties (from flow limits).

Data file

The data generator script creates a `.npz` file containing a user defined number of graphs and graph sizes. Each graph contains:

```
{
    ’generation’:
        shape: (num_samples, num_nodes)
        [[...], [...], ...],

    ’load’:
        shape: (num_samples, num_nodes)
        [[...], [...], ...],

    ’net_injection’:
        shape: (num_samples, num_nodes)
        [[...], [...], ...],

    ’edge_index’:
        shape: (num_samples, 2, num_edges)
        [[[...], [...]], [[...], [...]], ...],

    ’reactance’:
        shape: (num_samples, num_edges)
        [[...], [...], ...],

    ’line_limit’:
        shape: (num_samples, num_edges)
        [[...], [...], ...],

    ’power_flow’:
        shape: (num_samples, num_edges)
        [[...], [...], ...],

    ’lmp_spread_edge’:
        shape: (num_samples, num_edges)
        [[...], [...], ...],

}
```

Data Loader

```
data = Data(
    x = x,
    edge_index = edge_index ,
    edge_attr = edge_attr ,
    y = y,
    num_nodes = num_nodes
)
```

- `x` is the node feature matrix.
- `edge_index` is the connectivity matrix that defines which nodes are connected to which.
- `edge_attr` is the edge feature matrix.
- `y` is the label containing LMP spread values for each edge.
- `num_nodes` is the number of nodes in the graph.

The dataset is randomly divided into:

- 80% Training
- 10% Validation
- 10% Testing

1 Model Architecture

1.1 Data Input

The model is a custom message passing neural network designed for edge level regression tasks on graph structured data. It takes in node features and edge features to predict scalar labels for each edge.

1.2 Message Passing Block

The core of the model is a stack of **GINEConv** layers, which propagate information between nodes while incorporating edge features. The edge attributes are first processed through an MLP, then passed to each GINE layer.

$$h_i^{(l+1)} = \text{GINEConv}\left(h_i^{(l)}, \left\{h_j^{(l)} \mid j \in \mathcal{N}(i)\right\}, e_{j,i}\right)$$

Variable Definitions:

- $h_i^{(l)}$: Feature vector of node i at layer l. This represents the current hidden state of node i.
- $h_i^{(l+1)}$: Updated feature vector of node i after the $(l+1)^{\text{th}}$ GINEConv layer.
- $\mathcal{N}(i)$: The neighborhood of node i, i.e., the set of nodes j such that there is an edge between j and i.
- $h_j^{(l)}$: Feature vector of neighboring node $j \in \mathcal{N}(i)$ at layer l.
- $e_{j,i}$: Feature vector of the edge connecting node j to node i. In undirected graphs, this may also be written as $e_{i,j}$.

Residual Connections: To improve learning and gradient flow, **residual connections** are applied every two layers:

$$h_i^{(l+2)} = h_i^{(l)} + \text{GINEConv}^{(l+2)}(\text{GINEConv}^{(l+1)}(h_i^{(l)}))$$

Node and Edge Embeddings

Node features are projected into a hidden space using a linear transformation:

$$x_{\text{node}} = \text{Linear}(x)$$

Edge features are processed by a two layer MLP with ReLU activations:

$$x_{\text{edge}} = \text{MLP}(e)$$

Non Linearity (ReLU)

The model uses ReLU activations to apply non linearity:

$$\text{ReLU}(x) = \max(0, x)$$

Regularization (Dropout)

Uses dropout regularization so a certain set percentage of the neurons in training are ignored set by the user here is 10%.

2 Output Block

After the message passing phase, edge level predictions are made by concatenating the embeddings of the two connected nodes along with their shared edge features:

$$\text{input}_{\text{MLP}} = [x_i \parallel x_j \parallel e_{ij}]$$

The final edge level prediction output is computed using a prediction MLP:

$$\hat{y}_{ij} = \text{MLP}_{\text{predictor}}([x_i, x_j, e_{ij}])$$

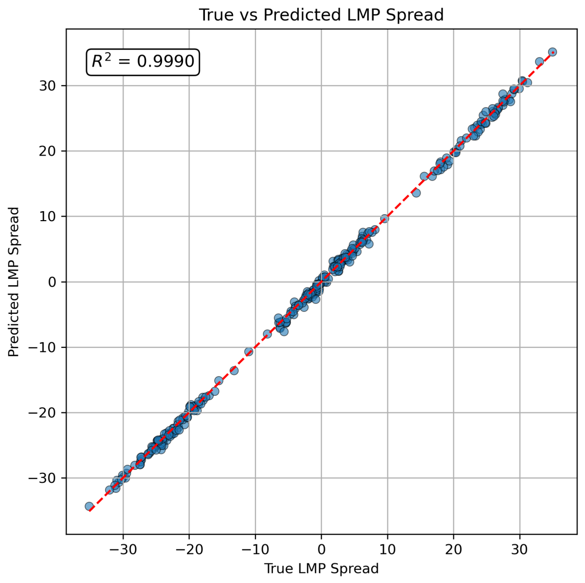
3 Train

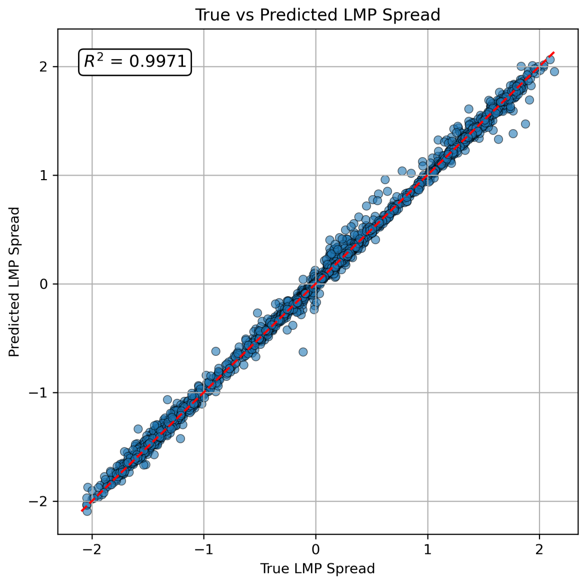
Loss Function: Training minimizes the Mean Squared Error (MSE) between the predicted and true edge labels:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

Evaluation: After training, the model is evaluated on a test set using MAE, RMSE, and R^2 score.

4 Results





Predicted vs true LMP spreads on two different sized datasets, using slightly larger and smaller toy graphs.