Objective (page 4-5)

Learning problem Our final goal is to learn a mapping $f_{\boldsymbol{\theta}} : S_n^{++} \to S_n^{++}$ that takes an spd matrix $\boldsymbol{A}$ and predicts a suitable preconditioner $\boldsymbol{P}$ that improves the spectral properties of the system – and therefore also the convergence behavior of the conjugate gradient method.

- Generated preconditioners are restricted to lower triangular matrices with strictly positive elements on the diagonal.
- Required to have the same sparsity pattern as the input matrix, so no fill-ins are allowed (the code gives the option to include fill-ins, if ever more accuracy in the matrix decomposition is needed – at the expense of greater computation time)
    - if the matrix entry in the original matrix A was 0, then the same matrix entry in P should also be 0

Overall algo (page 21)

Algorithm 2 Pseudo-code for NeuralIF preconditioner.

Input: Graph representation of the spd system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$.
Output: Lower-triangular sparse preconditioner for the linear system which is an incomplete factorization.
▷ *NeuralIF preconditioner computation:* $P_\theta(A) = \Lambda_\theta(A)\Lambda_\theta(A)^T$
Compute node features $\boldsymbol{x}_i$ shown in Table 4
Apply graph normalization // stabilizes training → faster convergence, independent of topology
Split graph adjacency matrix into index set for the lower and upper triangular parts, L and U.
for each message passing block l in $0, 1, \ldots, N-1$ **do**
  ▷ *update using the lower-triangular matrix part*
  $z_{ij}^{(l+\frac{1}{2})} \leftarrow \boxed{\phi_{\boldsymbol{\theta}^l_{z,1}}}(\boldsymbol{z}_{ij}^{(l)}, \boldsymbol{x}_i^{(l)}, \boldsymbol{x}_j^{(l)})$ for all $(i,j) \in L$ $\in \mathbb{R}^{1+8+8}$ // update edge features
  $m_i^{(l+\frac{1}{2})} \leftarrow \bigoplus^{(1)}_{j \in \mathcal{N}_L(i)} z_{ji}^{(l+\frac{1}{2})}$ // mean aggregation (based on lower updates)
  $\boldsymbol{x}_i^{(l+\frac{1}{2})} \leftarrow \boxed{\psi_{\boldsymbol{\theta}^l_{e,1}}}(\boldsymbol{x}_i^{(l)}, m_i^{(l+\frac{1}{2})})$ $\in \mathbb{R}^8$ // update node features
  ▷ *share the computed edge updates between the layers*
  $z_{ji}^{(l+\frac{1}{2})} \leftarrow z_{ij}^{(l+\frac{1}{2})}$ for all $(i,j) \in L$ // ensure symmetry of the latent-edge representation when flipping to the upper pass
  ▷ *update using the upper triangular matrix part*
  $z_{ji}^{(l+1)} \leftarrow \boxed{\phi_{\theta^l_{z,2}}}(z_{ji}^{(l+\frac{1}{2})}, x_j^{(l+\frac{1}{2})}, x_i^{(l+\frac{1}{2})})$ for all $(j,i) \in U$
  $m_i^{(l+1)} \leftarrow \bigoplus^{(2)}_{j \in \mathcal{N}_U(i)} z_{ji}^{(l+1)}$ // sum aggregation (based on upper updates)
  $x_i^{(l+1)} \leftarrow \psi_{\theta^l_{e,2}}(x_i^{(l)}, m_i^{(l+1)})$ // update node features again
  if not final layer in the network **then**
    ▷ *add skip connections*
    $z_{ij}^{(l+1)} \leftarrow [z_{ji}^{(l+1)}, a_{ji}]^\mathsf{T}$ for all $(j,i) \in U$ // upper-edge embedding concatenated with original matrix entry
  else
    $z_{ij}^{(l+1)} \leftarrow z_{ji}^{(l+1)}$ for all $(j,i) \in U$ // ensure symmetry
Apply $\sqrt{\exp(\cdot)}$-activation function to final edge embedding of diagonal matrix entries $z_{ii}^{(N)}$. // enforce pd of learned preconditioner
Return lower triangular matrix with elements $z_{ij}^{(N)}$ for $i \le j$.

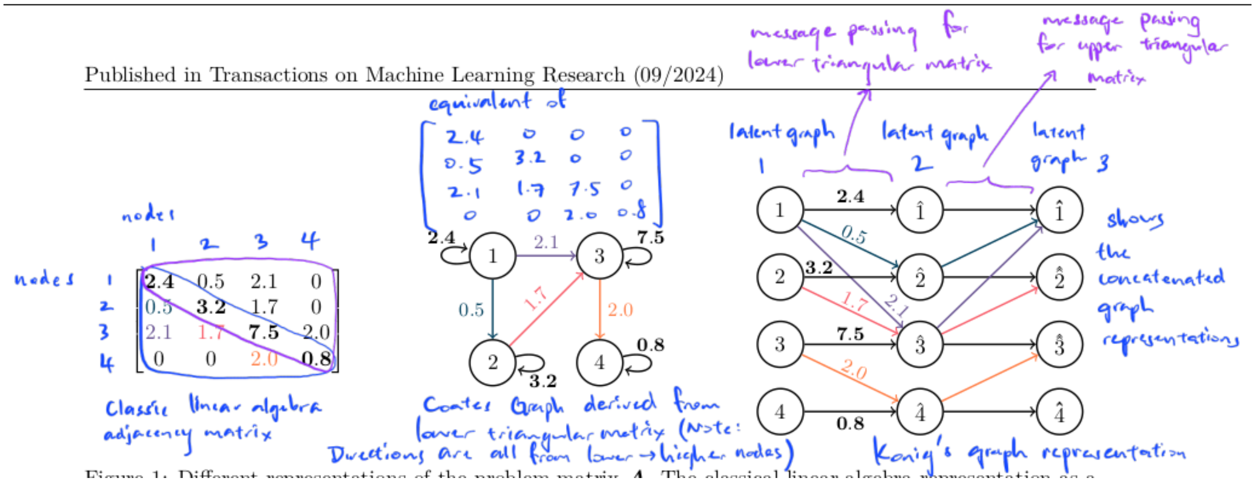
2 FC layers with 8 hidden units each & tanh activation

$\in \mathbb{R}$

Another MLP

- Three message passing blocks used

Graph representations (page 6)



<u>Left</u> – symmetric adjacency matrix

<u>Middle</u> – Coates graph, the equivalent representation of the lower triangular matrix from the left matrix, but now in graph form. This graph representation is the input graph to the GNN

<u>Right</u> – Konig's graph (not that important), basically a concatenated graph representation to show that it is possible to represent matrix multiplications in graph form.

Message-passing (page 4)

1. Update edge features

to describe the update functions for a simple message-passing GNN layer. In each layer l of the network the edge features are updated first by the network computing the features of the next layer $l+1$ as

output message (for the next layer) passed btwn i & j

$$z_{ij}^{(l+1)} = \phi_{\theta_z^{(l)}}\left(z_{ij}^{(l)}, x_i^{(l)}, x_j^{(l)}\right), \qquad (2)$$

eg. perceptron

combination of messages from current & node features for nodes i & j

where ϕ is a parameterized function. The outputs of this function are also referred to as *messages*. Then,

2. Perform a permutation-invariant aggregation
   Incoming messages aggregated using the mean, and the sum function in the 1st and 2nd message-passing steps in the block (see **Message-passing block**)

Any such permutation-invariant aggregation function is denoted here by $\oplus$. The aggregation of incoming messages over the neighborhood $\mathcal{N}$ of node i, which is defined as the set of adjacent nodes in the graph $\mathcal{N}(i) = \{j \mid (i,j) \in E\}$, is computed as

permutation-invariant aggregation function

$$m_i^{(l+1)} = \bigoplus_{j \in \mathcal{N}(i)} z_{ji}^{(l+1)}. \qquad (3)$$

3. Update the node features using a MLP

$$x_i^{(l+1)} = \psi_{\theta_x^{(l)}}\left(x_i^{(l)}, m_i^{(l+1)}\right).$$

Message-passing block (page 6-7)
Step 1: execute message-passing over the lower-triangular matrix (of the adjacency matrix)
Step 2: execute message-passing over the upper-triangular matrix (of the adjacency matrix)

But why compute based on both lower and upper-triangular matrices?

- Note that the Coates graph representation shown in **Graph representations** takes as input the lower-triangular matrix of the original adjacency matrix.
- From the directions of the arrows we know that that the higher labeled nodes are receiving information from the lower labeled nodes, but not the other way around
- So we need to compute message-passing for the upper triangular matrix also, so that the lower labeled nodes can receive information from the higher labeled nodes.

Complexity analysis (page 7-8)
<u>Space complexity - O(n)</u>
Only the nonzero elements in A (O(nnz)) and the node features (O(n), if there are n nodes and 8 features per node) need to be stored.

<u>Time complexity - O(nnz)</u>
Dependent on the 3 key operations - edge update, aggregation, node update (refer to **Message-passing)**

Edge update (O(nnz)):

$$\boldsymbol{z}_{ij}^{(l+1)} = \phi_{\boldsymbol{\theta}_z^{(l)}}\left(\boldsymbol{z}_{ij}^{(l)}, \boldsymbol{x}_i^{(l)}, \boldsymbol{x}_j^{(l)}\right)$$

This function is executed on each nonzero entry Aij. The MLP (constant cost) is applied once per edge per layer.

Aggregation of edge messages (O(nnz)):

$$\boldsymbol{m}_i^{(l+1)} = \bigoplus_{j \in \mathcal{N}(i)} \boldsymbol{z}_{ji}^{(l+1)}.$$

function

If each node on avg has d neighbours (O(d)), we have O(n*d) for n nodes, which is just the total number of edges nnz.

Node update (O(n)):

$$\boldsymbol{x}_i^{(l+1)} = \psi_{\boldsymbol{\theta}_x^{(l)}}\left(\boldsymbol{x}_i^{(l)}, \boldsymbol{m}_i^{(l+1)}\right).$$

Run a MLP on the node's own feature xi plus the aggregated message to produce the new node feature. There are n nodes the MLP cost is constant.