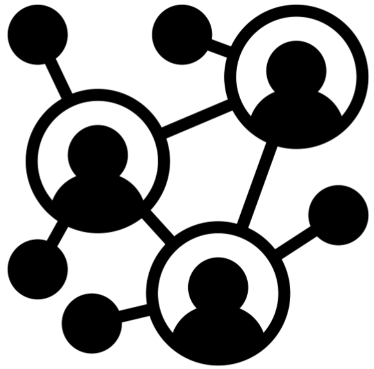
Neural Message Passing

Overview

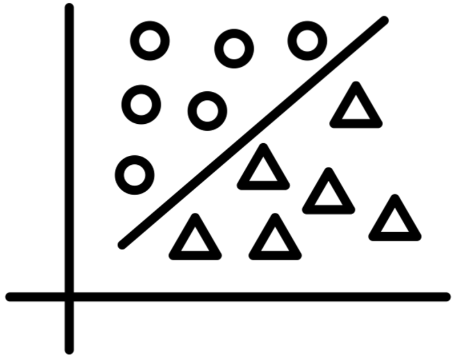
- High-level intuition
- General equations with examples
- Coded example

Background

- Our goal is to assign our nodes meaningful coordinates (i.e., "*embeddings*")
  - Coordinates allow us to create decision boundaries for classification problems
- An embedding of a node should consider its connections
  - I.e., nodes that share many connections should have similar embeddings



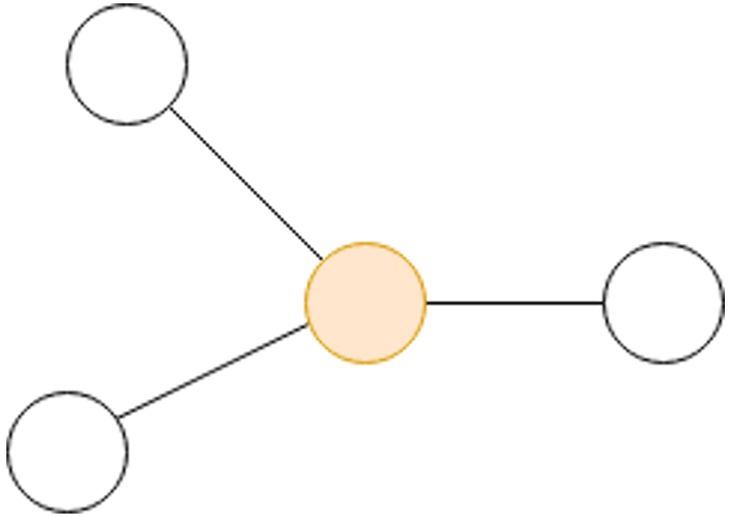
Created by Becris
from Noun Project



Created by Gacem Tachfin
from Noun Project

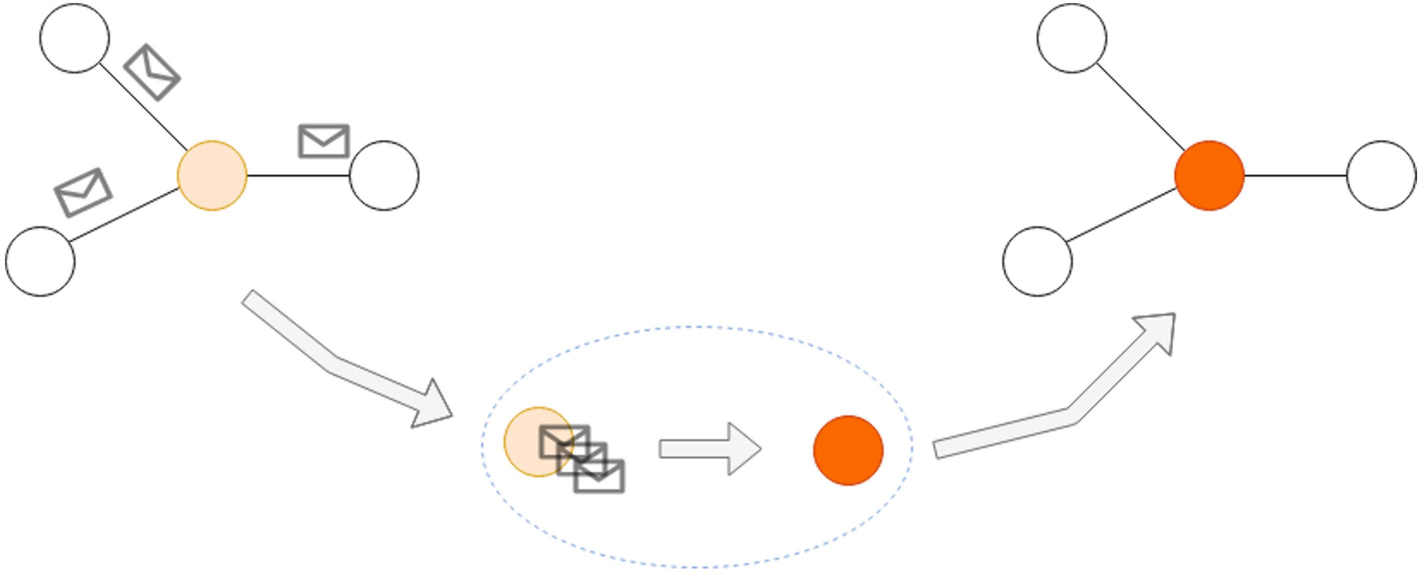
Motivating Example

- To make things concrete, let's use the following setup:
  - Nodes: people
  - Node features: age, net worth
  - Edges: in phone contacts
  - Edge features: number of phone calls in last year
- Will focus on one node in a small subgraph, but process is for all nodes



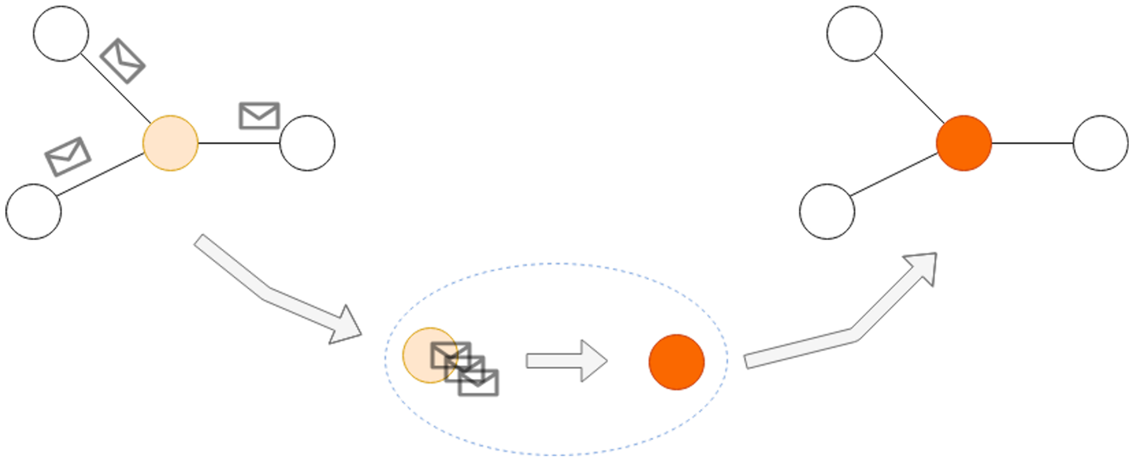
Intuition

- Goal: to calculate "neighborhood-aware" embeddings for nodes
- Approach:
  - Messages are sent between nodes via the edges
  - Nodes use these messages to update its embedding



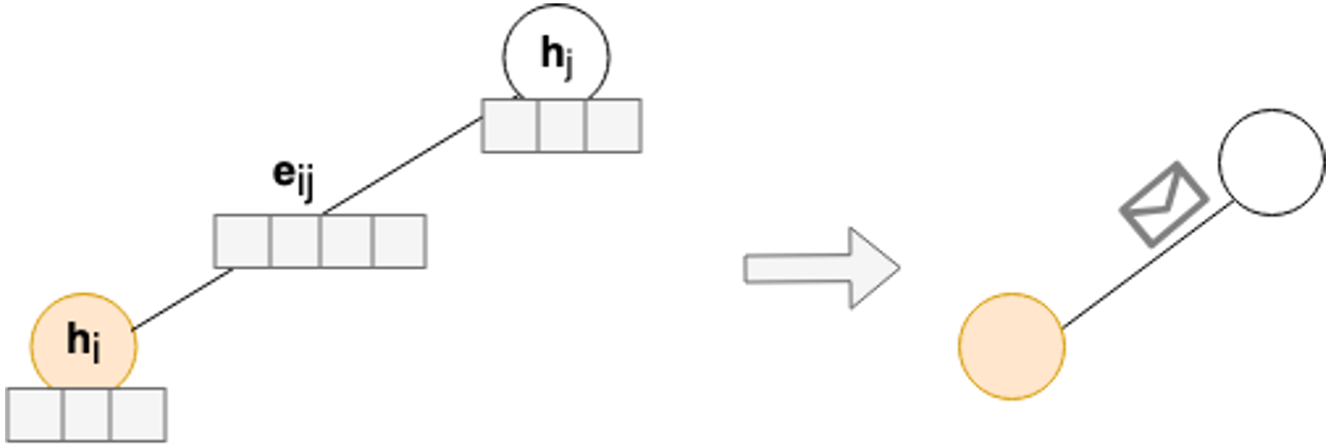
Framing of the problem

- There are three main functions:
  - The **Message** function, which computes the message using node/edge features
  - The **Aggregation** function, which combines the set of messages into a fixed-length vector that represents the neighborhood
  - The **Update** function, which computes the new node embedding using the aggregated messages and the old node embedding



Message function

$$\mathbf{m}_{ij}^{(k)} = \mathcal{M}(\mathbf{h}_i^{(k)}, \mathbf{h}_j^{(k)}, \mathbf{e}_{ij})$$



$$\mathcal{M}(\mathbf{h}_i^{(k)}, \mathbf{h}_j^{(k)}, \mathbf{e}_{ij})$$

Message function examples

$$\mathbf{m}_{ij}^{(k)} = \mathcal{M}(\mathbf{h}_i^{(k)}, \mathbf{h}_j^{(k)}, \mathbf{e}_{ij})$$

$$\mathbf{m}_{ij}^{(k)} = \mathbf{h}_j^{(k)}$$

Neighbor Copy

$$\mathbf{m}_{ij}^{(k)} = \frac{1}{c_{ij}}\mathbf{h}_j^{(k)}$$

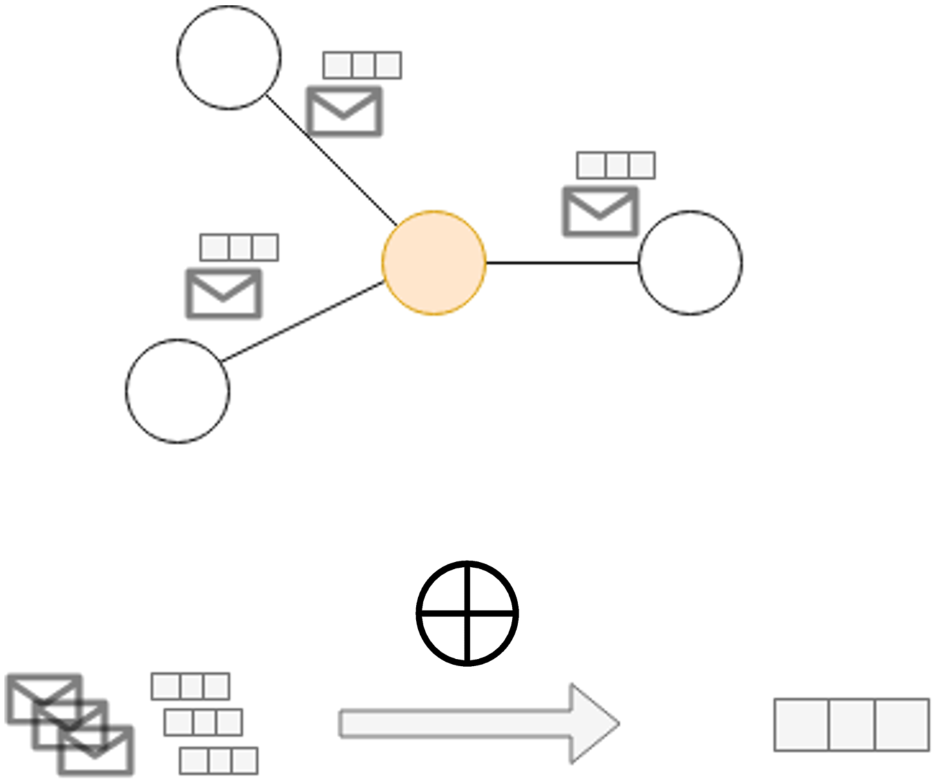
Structurally Normalized

$$\mathbf{m}_{ij}^{(k)} = a\left(\mathbf{h}_i^{(k)}, \mathbf{h}_j^{(k)}\right)\mathbf{h}_j^{(k)}$$

Attention

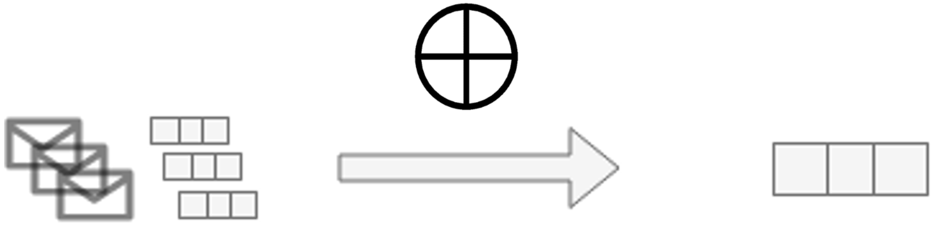
Aggregation function

$$\widehat{\mathbf{m}}_i^{(k)} = \bigoplus_{j \in \mathcal{N}_i} \mathbf{m}_{ij}^{(k)}$$



Aggregation function

- Fixed-length representation regardless of neighborhood size



- Permutation invariant: gives the same answer regardless of how you order the inputs

Aggregation function examples

$$\widehat{\mathbf{m}}_i^{(k)} = \bigoplus_{j \in \mathcal{N}_i} \mathbf{m}_{ij}^{(k)}$$

$$\widehat{\mathbf{m}}_i^{(k)} = \sum_{j \in \mathcal{N}_i} \mathbf{m}_{ij}^{(k)}$$

Sum

$$\widehat{\mathbf{m}}_i^{(k)} = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{m}_{ij}^{(k)}$$

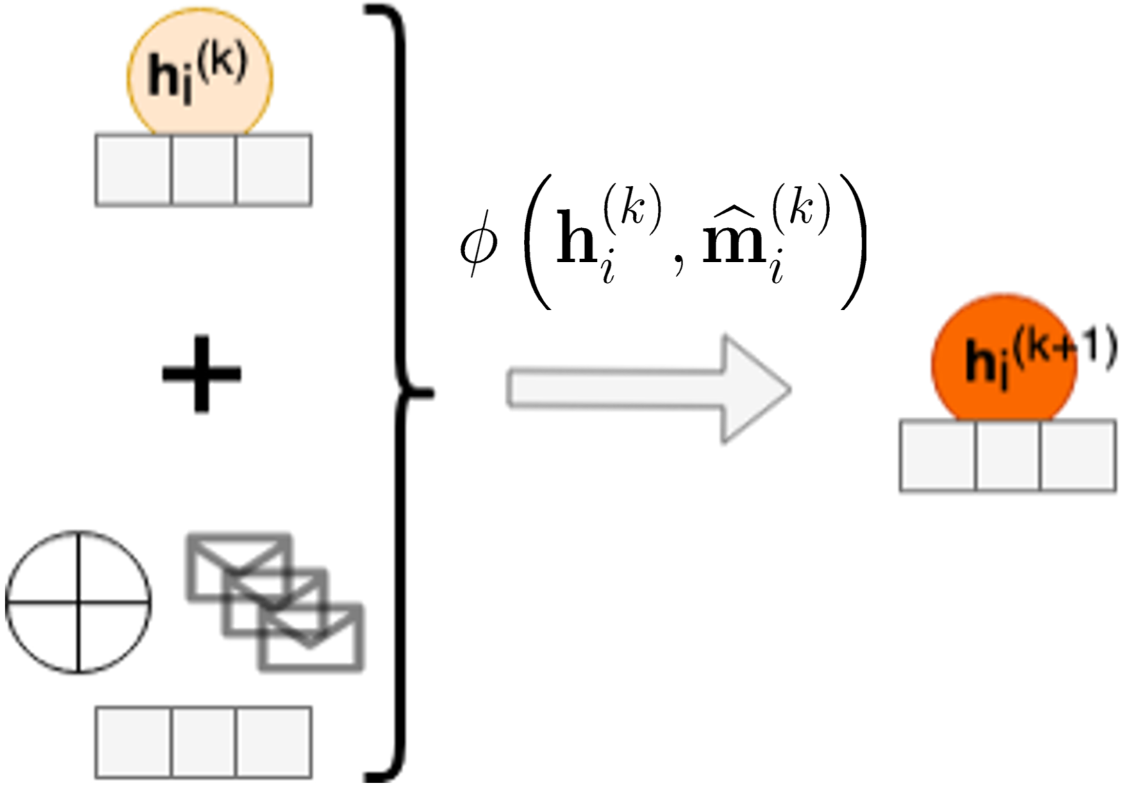
Average

$$\widehat{\mathbf{m}}_i^{(k)} = \max_{j \in \mathcal{N}_i} \left(\mathbf{m}_{ij}^{(k)}\right)$$

Max

Update function

$$\mathbf{h}_i^{(k+1)} = \phi\left(\mathbf{h}_i^{(k)}, \widehat{\mathbf{m}}_i^{(k)}\right)$$



Update function examples

$$\mathbf{h}_i^{(k+1)} = \phi\left(\mathbf{h}_i^{(k)}, \widehat{\mathbf{m}}_i^{(k)}\right)$$

$$\mathbf{h}_i^{(k+1)} = \sigma\left(\mathbf{W}^{(k+1)}\widehat{\mathbf{m}}_i^{(k)}\right)$$

$$\mathbf{h}_i^{(k+1)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k+1)}\mathbf{h}_i^{(k)} + \mathbf{W}_{\text{neigh}}^{(k+1)}\widehat{\mathbf{m}}_i^{(k)} + \mathbf{b}^{(k+1)}\right)$$

$$\mathbf{h}_i^{(k+1)} = \sigma\left(\mathbf{W}^{(k+1)}\text{CONCAT}\left(\mathbf{h}_i^{(k)}, \widehat{\mathbf{m}}_i^{(k)}\right)\right)$$

Architecture examples - GCN

$$\mathbf{h}_i^{(k+1)} = \sigma\left(\mathbf{W}^{(k+1)}\widehat{\mathbf{m}}_i^{(k)}\right)$$

$$\widehat{\mathbf{m}}_i^{(k)} = \underbrace{\sum_{j \in \mathcal{N}_i}}_{\text{Aggregation}} \mathbf{m}_{ij}^{(k)} = \sum_{j \in \mathcal{N}_i} \underbrace{\frac{1}{c_{ij}}\mathbf{h}_j^{(k)}}_{\text{Message}}$$

$$\mathbf{h}_i^{(k+1)} = \sigma\left(\mathbf{W}^{(k+1)}\sum_{j \in \mathcal{N}_i}\frac{1}{c_{ij}}\mathbf{h}_j^{(k)}\right)$$

Examples in code - DGL

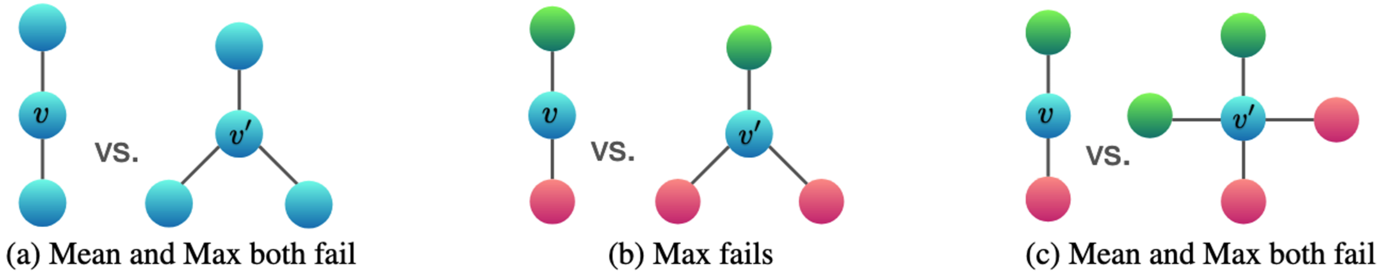
$$\mathbf{h}_i^{(k+1)} = \sigma\left(\mathbf{W}^{(k+1)}\mathrm{CONCAT}\left(\mathbf{h}_i^{(k)}, \frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i}\mathbf{h}_j^{(k)}\right)\right)$$

```python
import dgl.function as fn

class SAGEConv(nn.Module):
    """Graph convolution modul

    Parameters
    ----------
    in_feat : int
        Input feature size.
    out_feat : int
        Output feature size.
    """
    def __init__(self, in_feat, out_feat):
        super(SAGEConv, self).__init__()
        # A linear submodule for projecting the input and neighbor feature to the output.
        self.linear = nn.Linear(in_feat * 2, out_feat)

    def forward(self, g, h):
        """Forward computation

        Parameters
        ----------
        g : Graph
            The input graph.
        h : Tensor
            The input node feature.
        """
        with g.local_scope():
            g.ndata['h'] = h
            # update_all is a message passing API.
            g.update_all(message_func=fn.copy_u('h', 'm'), reduce_func=fn.mean('m', 'h_N'))
            h_N = g.ndata['h_N']
            h_total = torch.cat([h, h_N], dim=1)
            return self.linear(h_total)
```

https://docs.dgl.ai/tutorials/blitz/3_message_passing.html

Limitations

- The aggregation function applied to pair-wise messages has limited ability to distinguish certain structures^



- Repeating message passing multiple times will cause the well-known "over-smoothing" problem, which makes the node embeddings become a self-similar blob

^ Xu, Keyulu, et al. "How powerful are graph neural networks?." *arXiv preprint arXiv:1810.00826* (2018).

Additional Resources

- Gilmer, Justin, et al. "Neural message passing for quantum chemistry." *International conference on machine learning*. PMLR, 2017. (arXiv)
- Battaglia, P. et al. “Relational inductive biases, deep learning, and graph networks.” *ArXiv* abs/1806.01261 (2018): n. pag. (arXiv)