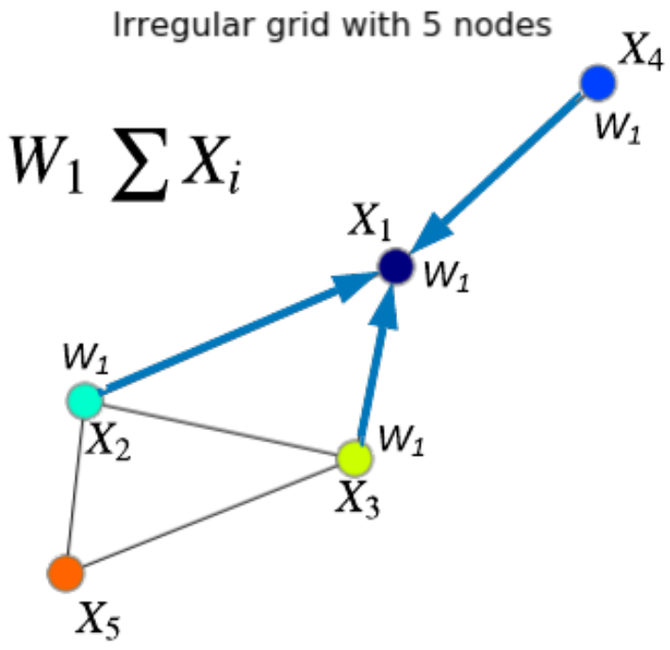
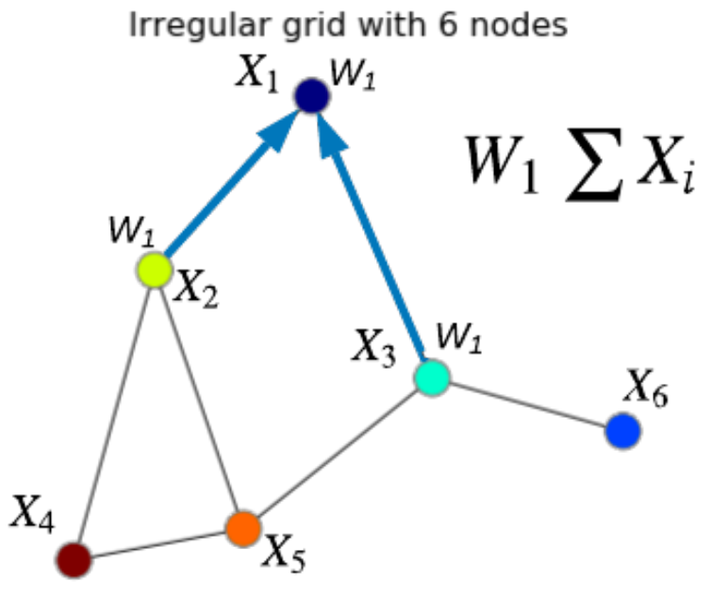
How to make convolution on a graph ?





option 1. Aggregate neighbors info, then update the current node

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right)$$
$$= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right),$$

option 2. Aggregate current node + neighbor info at once

$$\mathbf{h}_u^{(k)} = \text{AGGREGATE}(\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u) \cup \{u\}\}),$$

Message passing

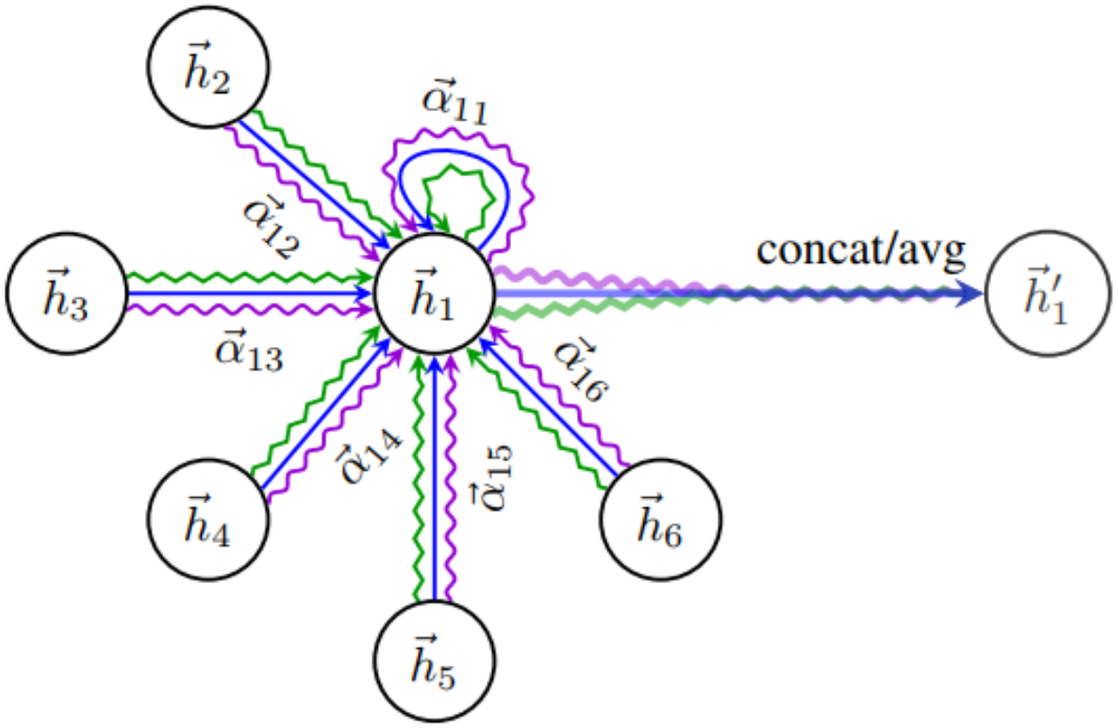
https://medium.com/@BorisAKnyazev/tutorial-on-graph-neural-networks-for-computer-vision-and-beyond-part-1-3d9fada3b80d

GNN - Aggregators

Attention mechanism

Uses **attention weight** to weight the importance of the neighbor information

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v,$$



the attention head may be learnt or deduced from neighbors info

$$\alpha_{u,v} = \frac{\exp\left(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v]\right)}{\sum_{v' \in \mathcal{N}(u)} \exp\left(\mathbf{a}^\top [\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_{v'}]\right)},$$

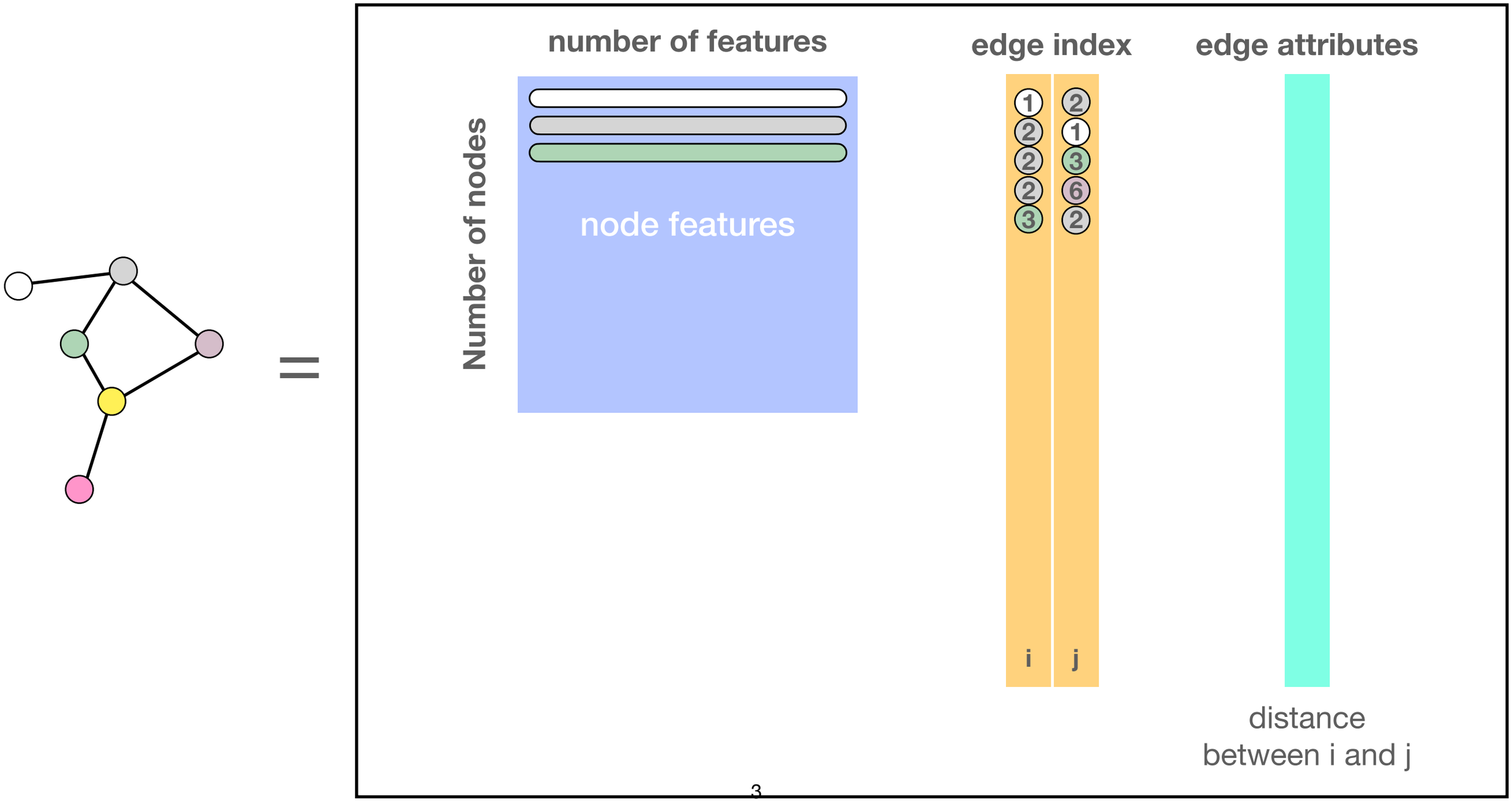
$$\alpha_{u,v} = \frac{\exp\left(\text{MLP}(\mathbf{h}_u, \mathbf{h}_v)\right)}{\sum_{v' \in \mathcal{N}(u)} \exp\left(\text{MLP}(\mathbf{h}_u, \mathbf{h}_{v'})\right)},$$

$$\alpha_{u,v} = \frac{\exp\left(\mathbf{h}_u^\top \mathbf{W}\mathbf{h}_v\right)}{\sum_{v' \in \mathcal{N}(u)} \exp\left(\mathbf{h}_u^\top \mathbf{W}\mathbf{h}_{v'}\right)},$$

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. and Bengio, Y. (2018) **Graph Attention Networks**.

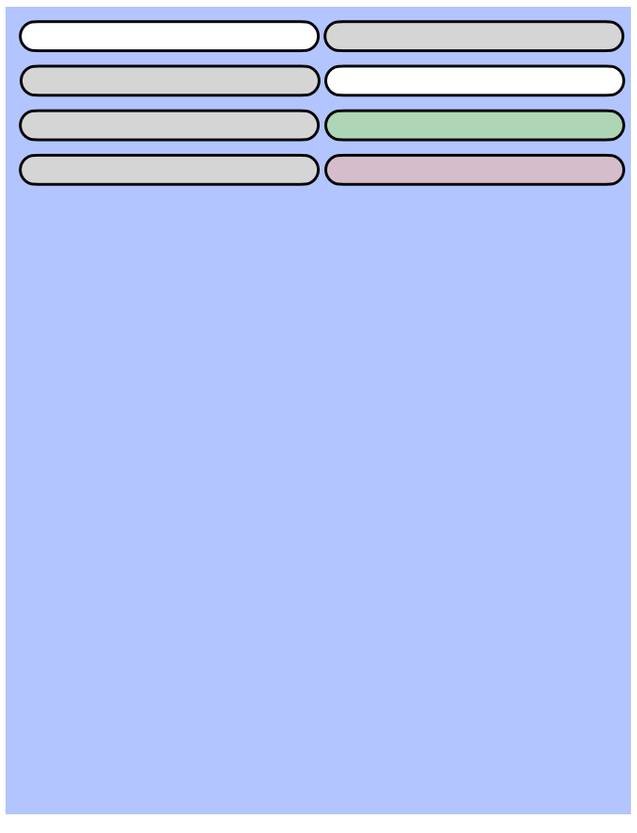
Ex. Graph attention (like) layer as implemented in GraphProt

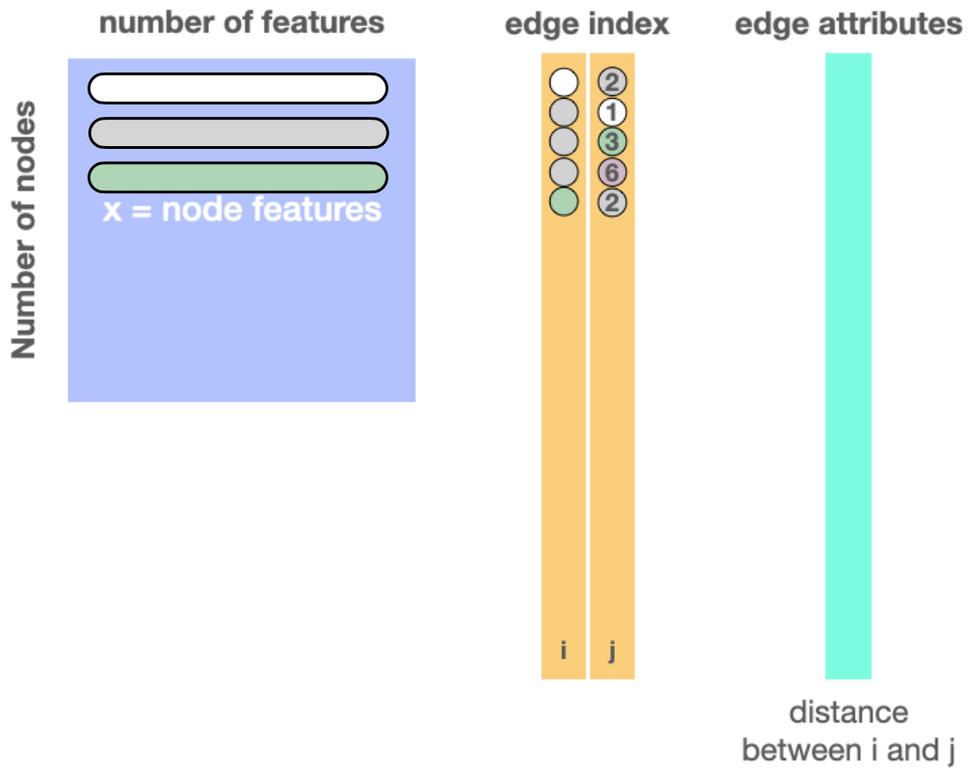
$$z_i = \frac{1}{Ni} \sum_j a_{ij} * [x_i || x_j] * W + b_i$$



GNN - Graph attention (like) layer

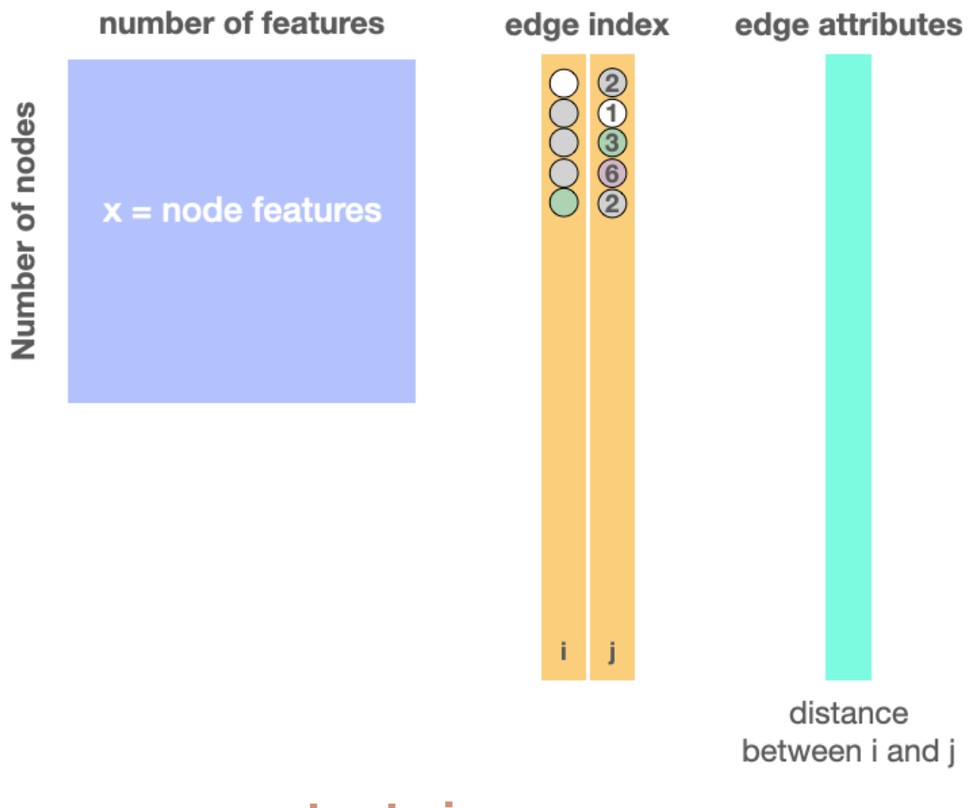
$$z_i = \frac{1}{Ni} \sum_j a_{ij} * [x_i || x_j] * W + b_i$$

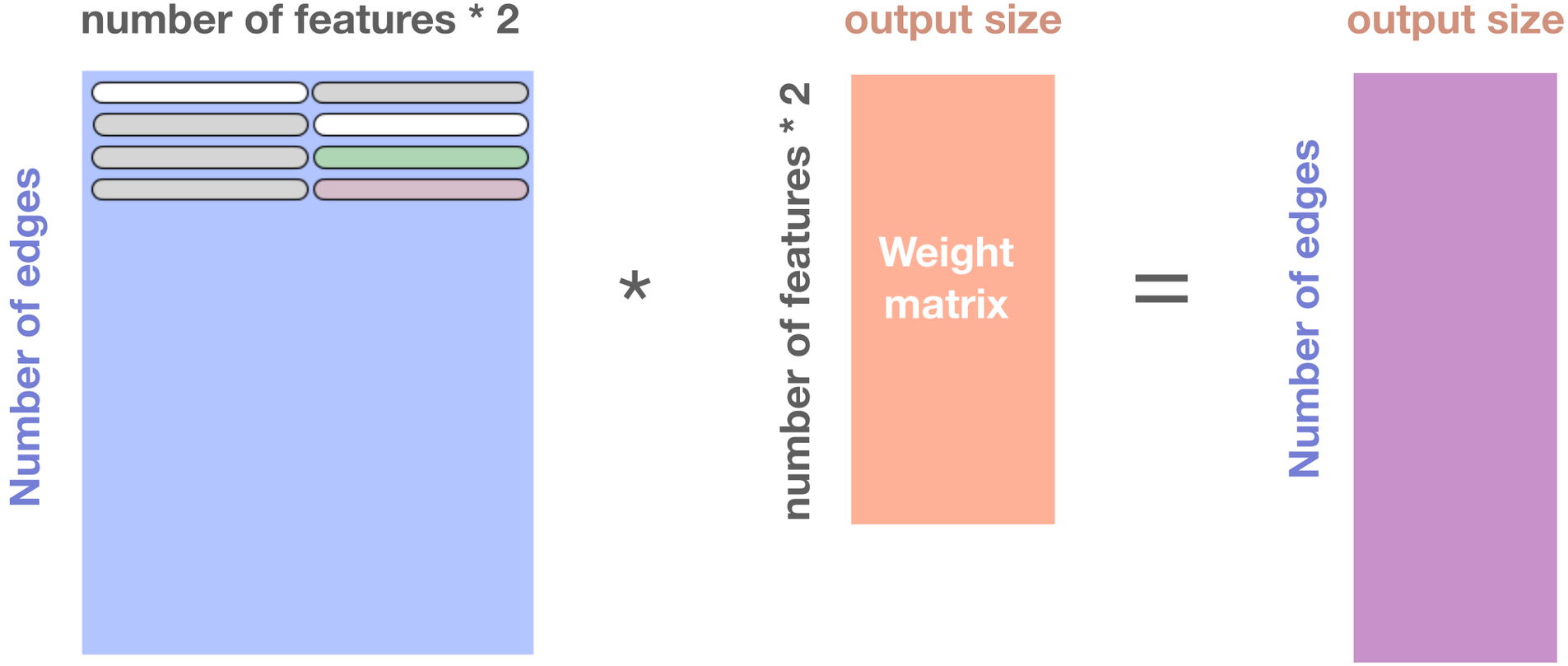




GNN - Graph attention (like) layer

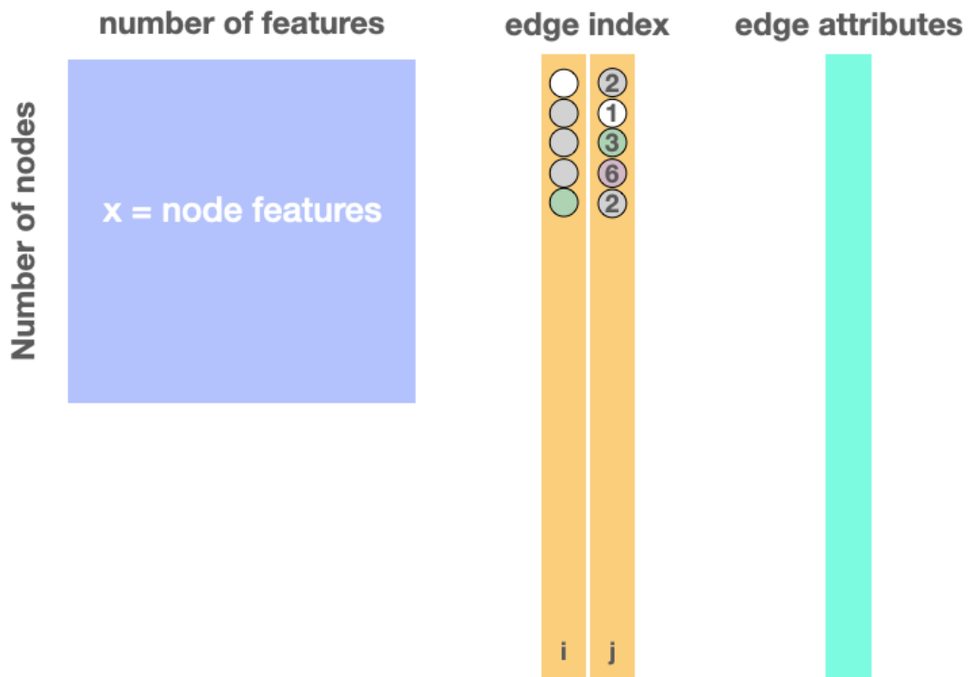
$$z_i = \frac{1}{Ni} \sum_j a_{ij} * [x_i || x_j] * W + b_i$$

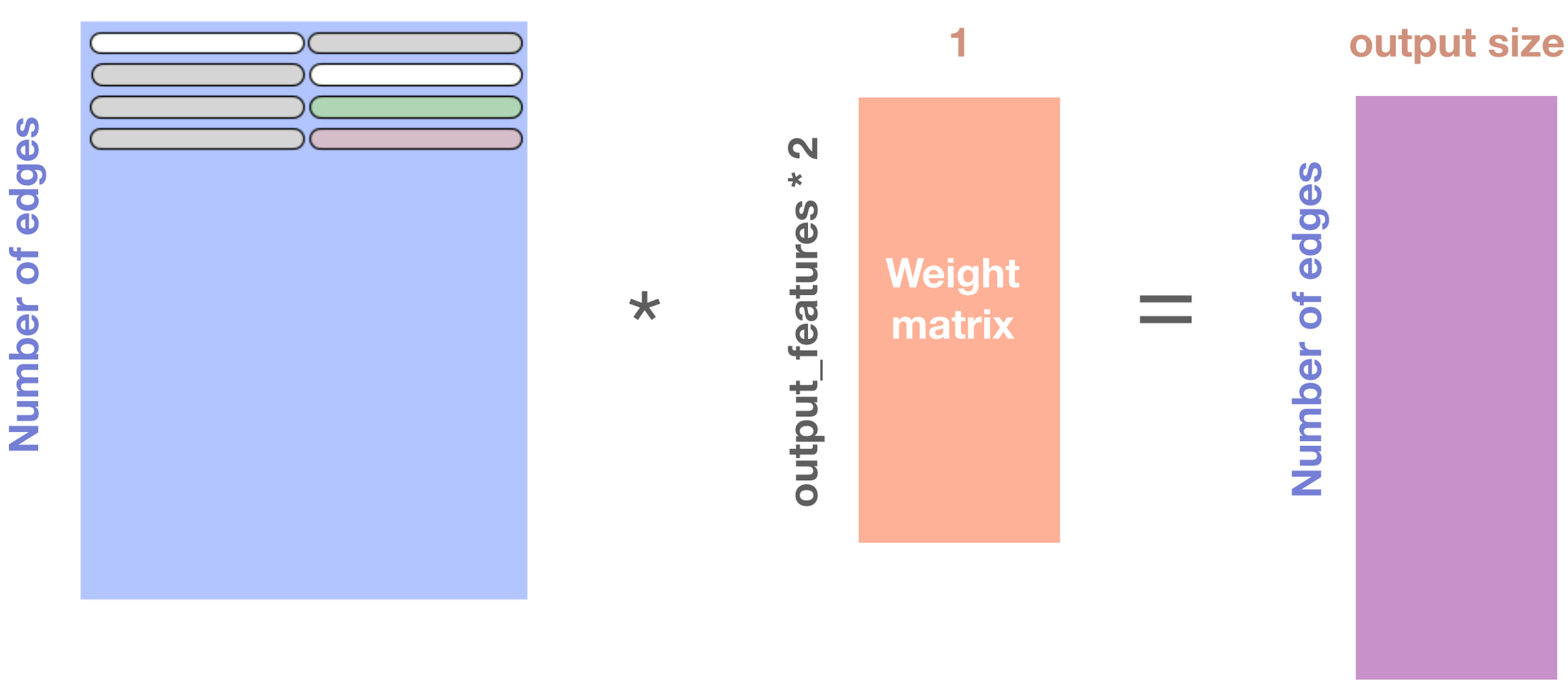




GNN - Graph attention (like) layer

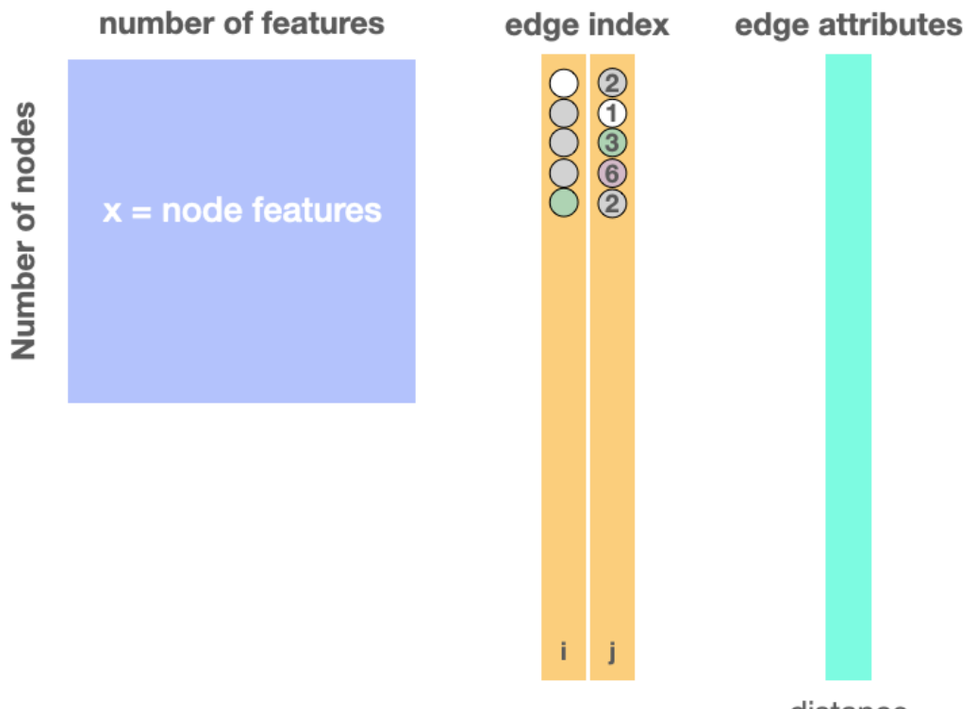
$$z_i = \frac{1}{Ni} \sum_j a_{ij} * [x_i || x_j] * W + b_i$$

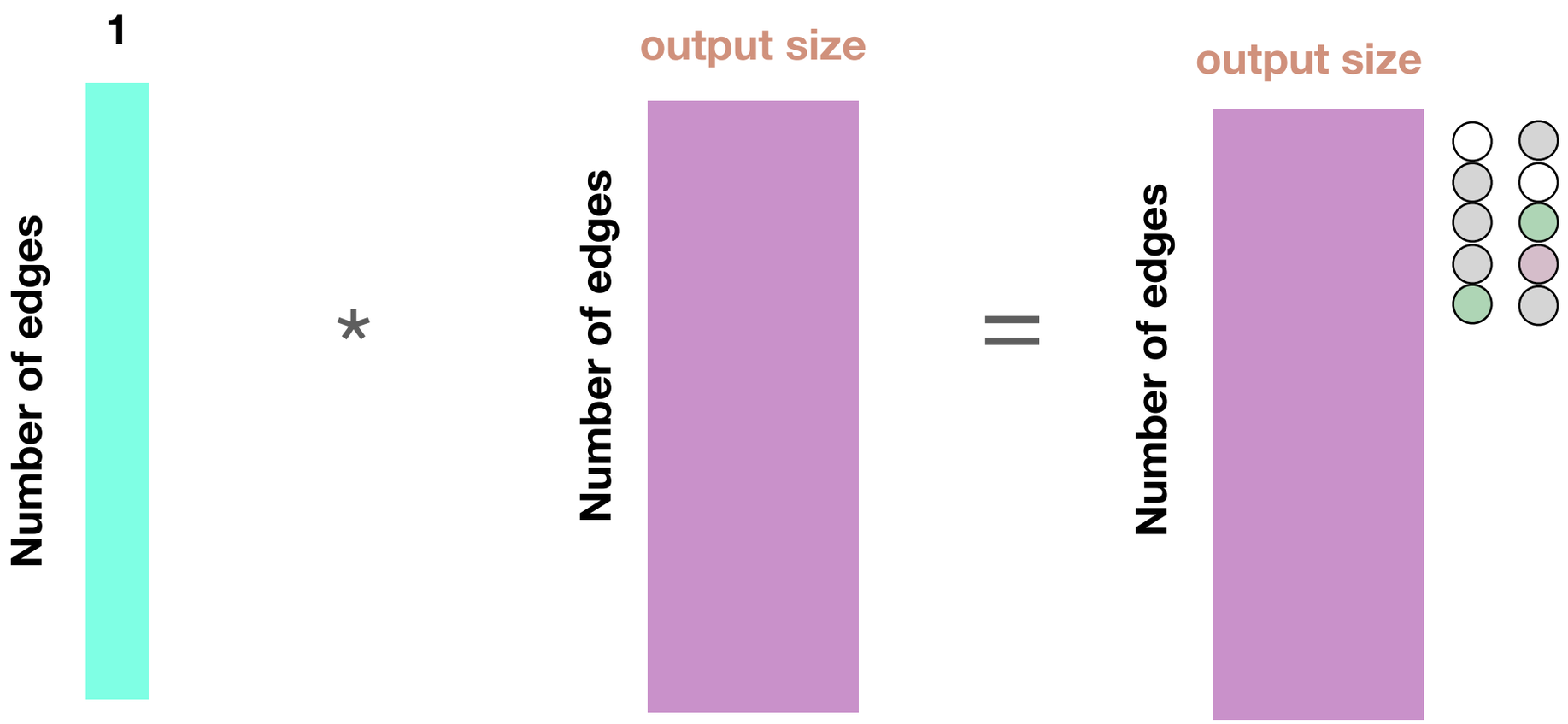




GNN - Graph attention (like) layer

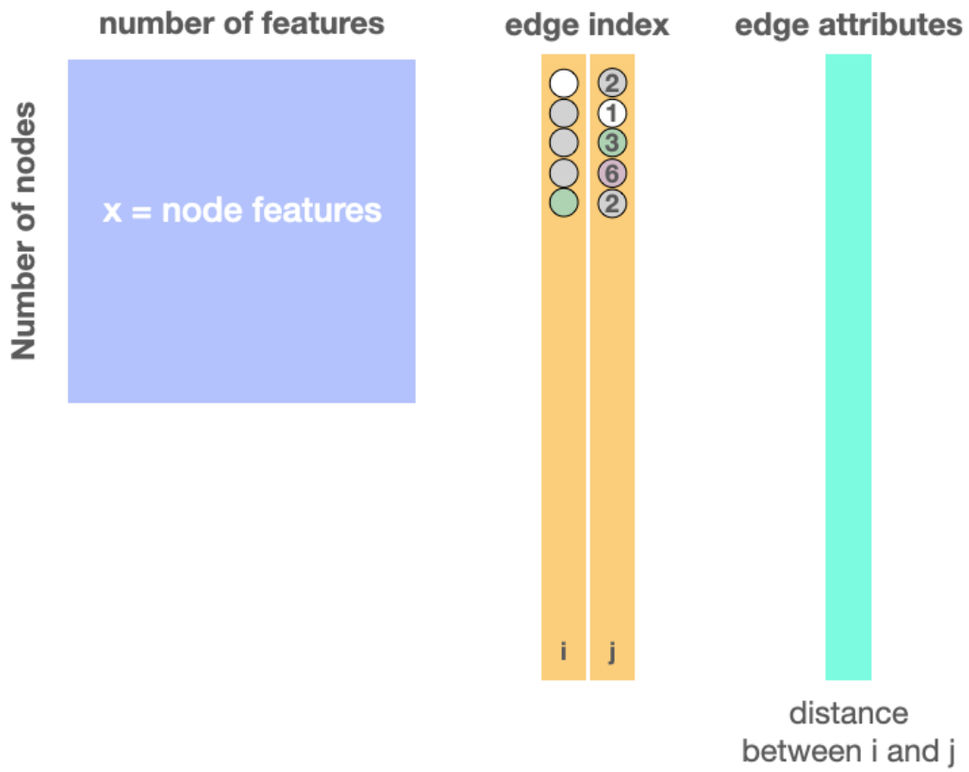
$$z_i = \frac{1}{Ni} \sum_j a_{ij} * [x_i || x_j] * W + b_i$$

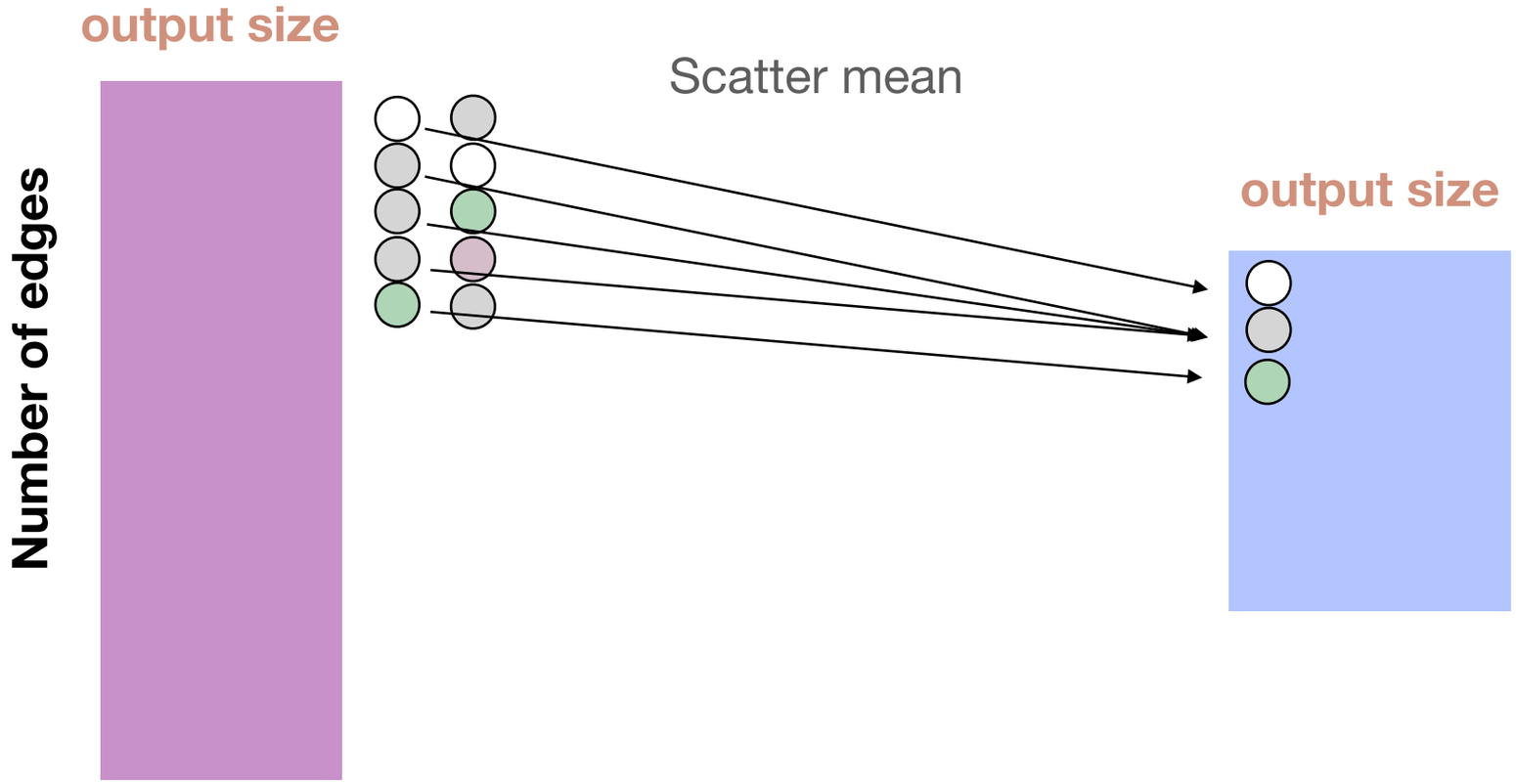




GNN - Graph attention (like) layer

$$z_i = \frac{1}{Ni} \sum_j a_{ij} * [x_i || x_j] * W + b_i$$





```python
def __init__(self, input_shape):
    super(GINet, self).__init__()

    self.conv1 = GraphAttention(input_shape, 16)
    self.conv2 = GraphAttention(16, 32)

    self.fc1 = torch.nn.Linear(32, 64)
    self.fc2 = torch.nn.Linear(64, 1)

    self.clustering = 'mcl'

def forward(self, data):

    act = nn.Tanhshrink()
    act = F.relu
    #act = nn.LeakyReLU(0.25)

    # first conv block
    data.x = act(self.conv1(
        data.x, data.edge_index, data.edge_attr))
cluster = get_preloaded_cluster(data.cluster0, data.batch)
data = community_pooling(cluster, data)

# second conv block
data.x = act(self.conv2(
    data.x, data.edge_index, data.edge_attr))
cluster = get_preloaded_cluster(data.cluster1, data.batch)
x, batch = max_pool_x(cluster, data.x, data.batch)

# FC
x = scatter_mean(x, batch, dim=0)
x = act(self.fc1(x))
x = self.fc2(x)
#x = F.dropout(x, training=self.training)

return x
```