Graph ML 2

Yury Kulev

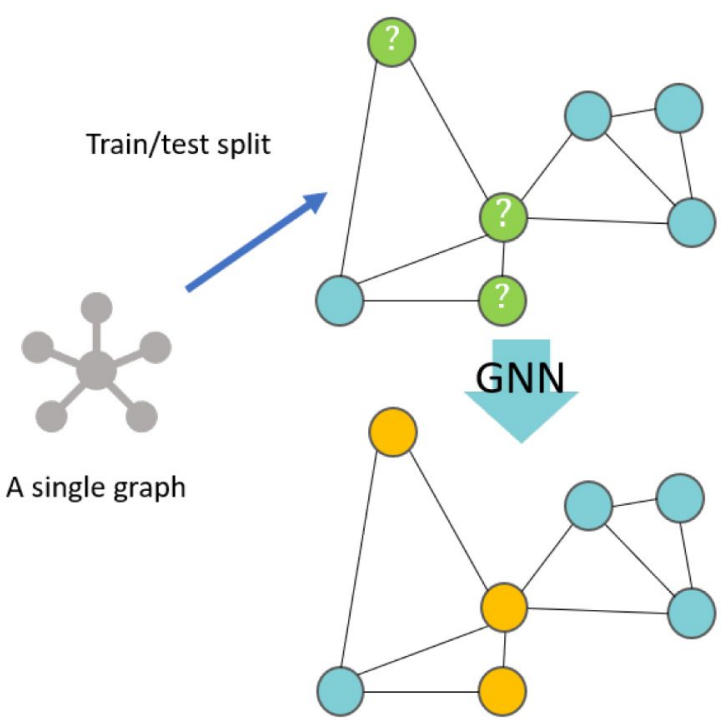
GraphSAGE

Motivation

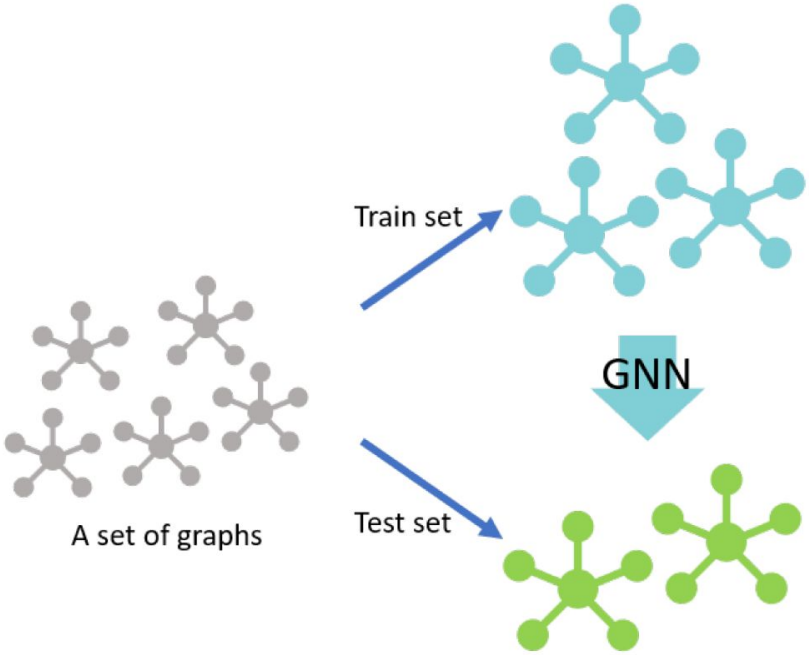
Transductive VS Inductive

- Transductive: make predictions only on the specific data points that are provided in the training set, without trying to create a general model
- Inductive: learning from a set of labeled training data to create a general model that can be used to make predictions on new, unseen data points

What about graphs?



(a) Transductive learning on a graph



(b) Inductive learning on graphs

Key idea

“Instead of training individual embeddings for each node, **we learn a function** that generates embeddings by sampling and aggregating features from a node’s local neighborhood”

Forward propagation

Algorithm 1: GraphSAGE embedding generation (i.e., forward propagation) algorithm

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$; input features $\{\mathbf{x}_v, \forall v \in \mathcal{V}\}$; depth K; weight matrices $\mathbf{W}^k, \forall k \in \{1, ..., K\}$; non-linearity σ; differentiable aggregator functions $\text{AGGREGATE}_k, \forall k \in \{1, ..., K\}$; neighborhood function $\mathcal{N} : v \rightarrow 2^{\mathcal{V}}$

Output : Vector representations $\mathbf{z}_v$ for all $v \in \mathcal{V}$

$\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$;

for $k = 1...K$ **do**

  for $v \in \mathcal{V}$ **do**

    $\mathbf{h}_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$;

    $\mathbf{h}_v^k \leftarrow \sigma\left(\mathbf{W}^k \cdot \text{CONCAT}(\mathbf{h}_v^{k-1}, \mathbf{h}_{\mathcal{N}(v)}^k)\right)$

  end

  $\mathbf{h}_v^k \leftarrow \mathbf{h}_v^k / \|\mathbf{h}_v^k\|_2, \forall v \in \mathcal{V}$

end

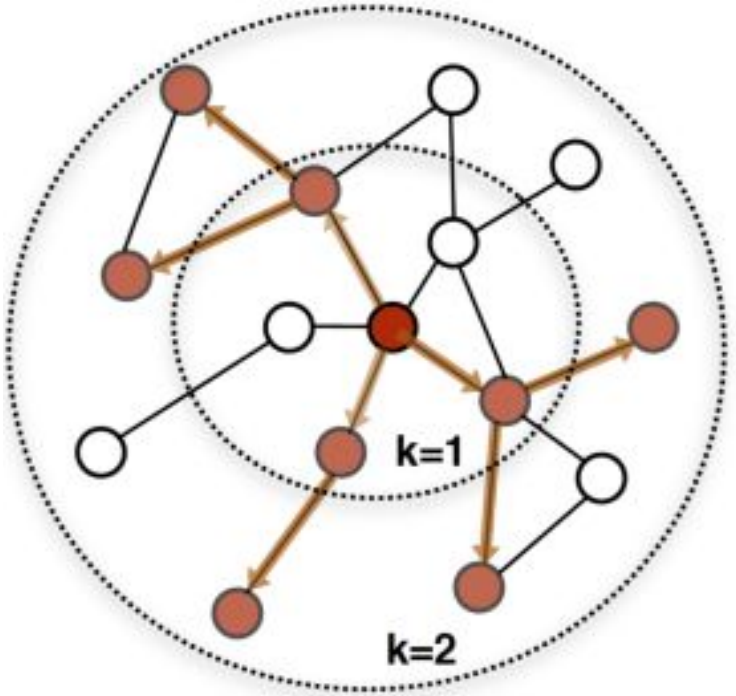
$\mathbf{z}_v \leftarrow \mathbf{h}_v^K, \forall v \in \mathcal{V}$

Neighborhood definition

Uniform draw of size S_k for each k from 1 to K (depth)

Usually, K is equal to 1, 2 or 3

Optimal parameters: K = 2, $S_1 S_2 \leq 500$



Aggregator architectures

1) Mean aggregator

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{\mathbf{h}_v^{k-1}\} \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\})$$

Aggregator architectures

2) LSTM aggregator

Good article with LSTM explanation - https://colah.github.io/posts/2015-08-Understanding-LSTMs/

TLDR: LSTM is a type of recurrent neural network (RNN) architecture that is designed to effectively capture and remember long-term dependencies in sequential data

Aggregator architectures

3) Pooling aggregator

$$\text{AGGREGATE}_k^{\text{pool}} = \max(\{\sigma\left(\mathbf{W}_{\text{pool}}\mathbf{h}_{u_i}^k + \mathbf{b}\right), \forall u_i \in \mathcal{N}(v)\})$$

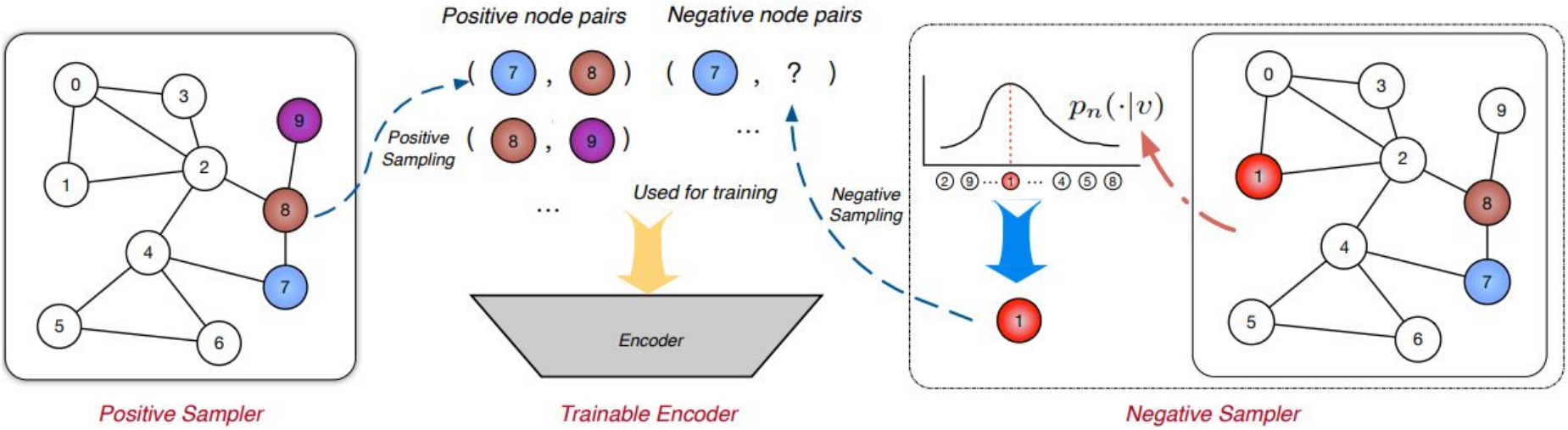
Loss

1) Unsupervised approach:

$$J_{\mathcal{G}}(\mathbf{z}_u) = -\log\left(\sigma(\mathbf{z}_u^\top \mathbf{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\mathbf{z}_u^\top \mathbf{z}_{v_n})\right)$$

2) Another approaches (e.g., cross-entropy loss)

Negative Sampling



Pretty good article about different negative sampling strategies - https://arxiv.org/abs/2005.09863

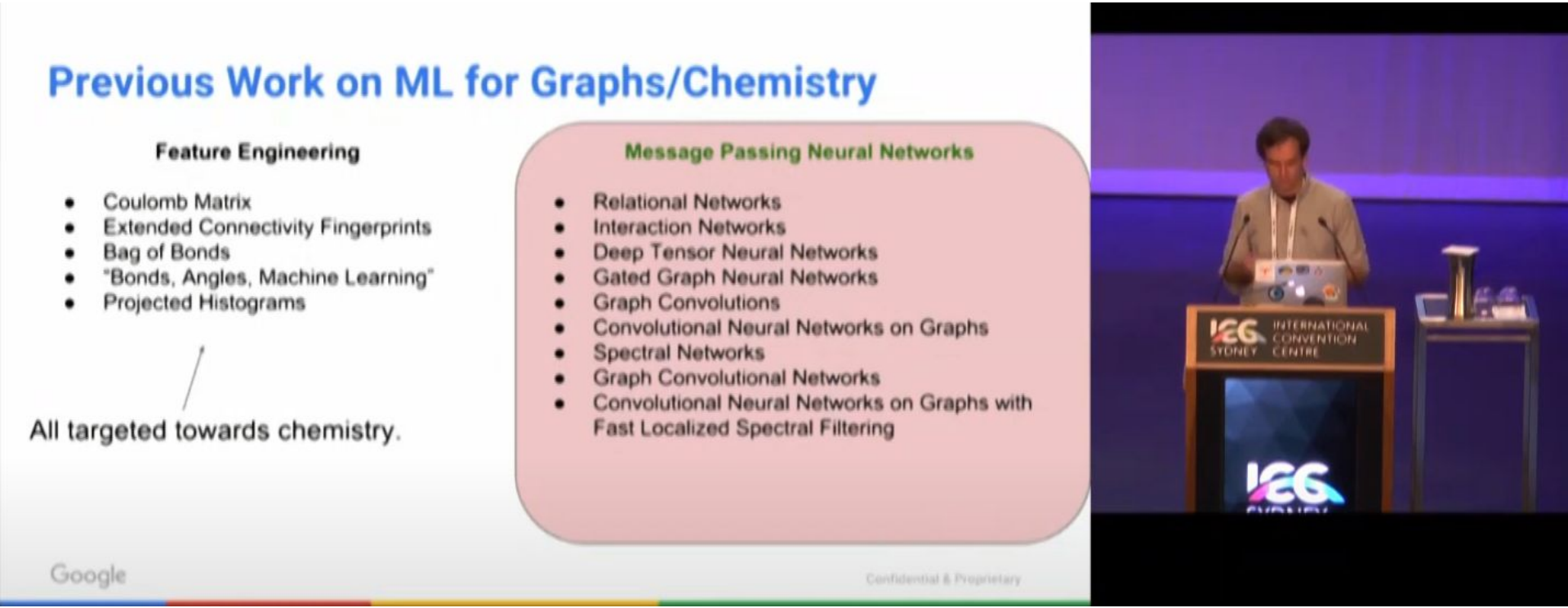
Authors suggest their own method and compare it using different GNNs (including GraphSAGE)

Experiments

	Citation		Reddit		PPI	
Name	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1	Unsup. F1	Sup. F1
Random	0.206	0.206	0.043	0.042	0.396	0.396
Raw features	0.575	0.575	0.585	0.585	0.422	0.422
DeepWalk	0.565	0.565	0.324	0.324	—	—
DeepWalk + features	0.701	0.701	0.691	0.691	—	—
GraphSAGE-GCN	0.742	0.772	**0.908**	0.930	0.465	0.500
GraphSAGE-mean	0.778	0.820	0.897	0.950	0.486	0.598
GraphSAGE-LSTM	0.788	0.832	**0.907**	**0.954**	0.482	**0.612**
GraphSAGE-pool	**0.798**	**0.839**	0.892	0.948	**0.502**	0.600
% gain over feat.	39%	46%	55%	63%	19%	45%

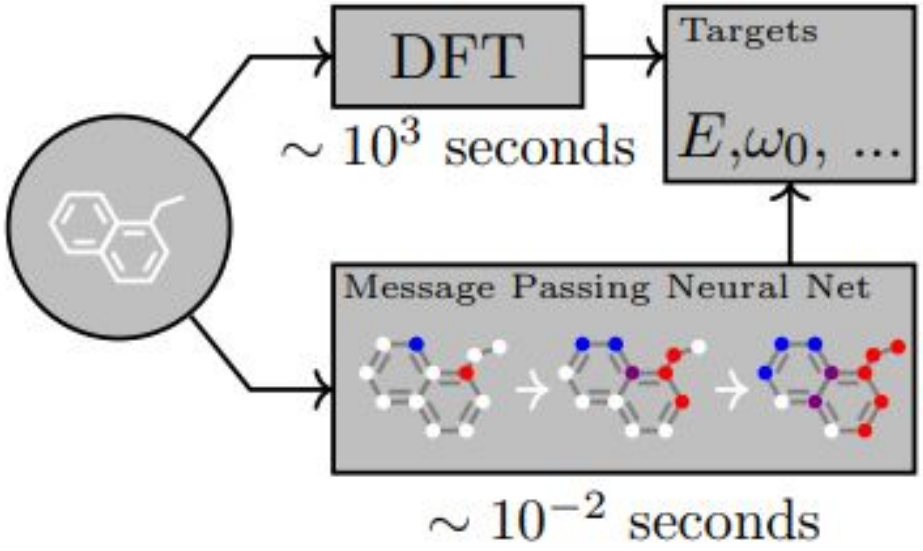
MPNN

Motivation



Presentation from article author - https://www.youtube.com/watch?v=NJEb5sqjv2w

Problem


DFT
Targets
E,ω_0, ...
$\sim 10^3$ seconds
Message Passing Neural Net
$\sim 10^{-2}$ seconds

Main phases

1) Message passing phase

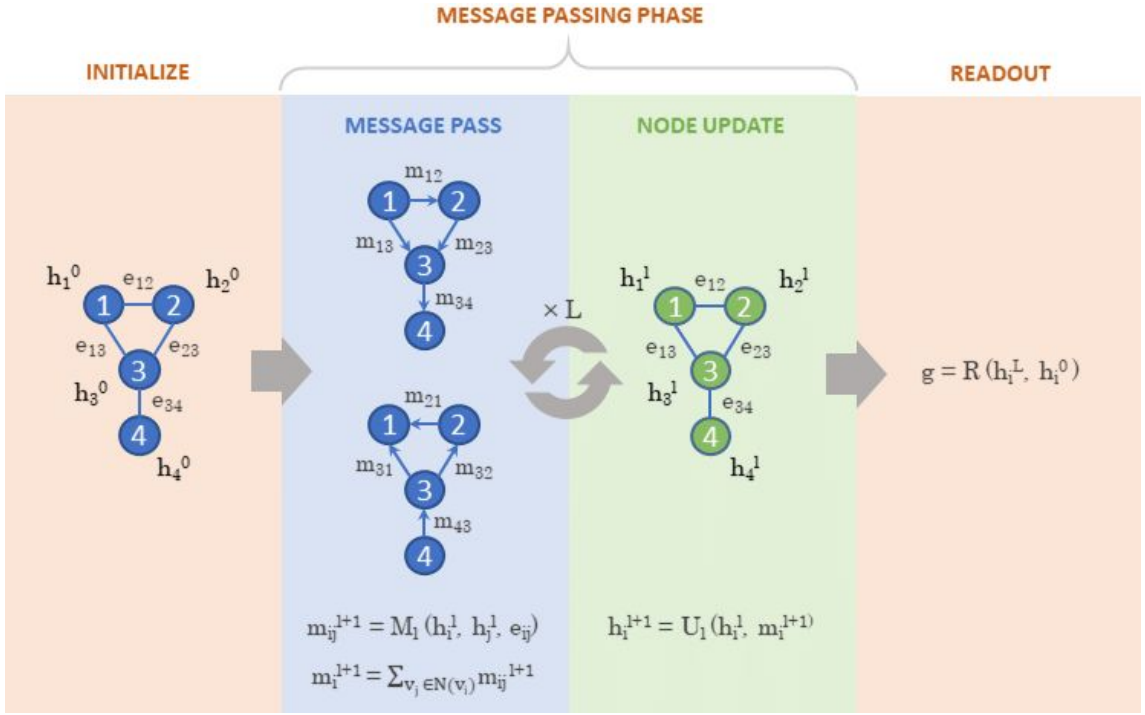
$$m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$

$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1})$$

2) Readout phase

$$\hat{y} = R(\{h_v^T \mid v \in G\}).$$

Main phases


MESSAGE PASSING PHASE
INITIALIZE
READOUT
MESSAGE PASS
NODE UPDATE
× L
$m_{ij}^{l+1} = M_l(h_i^l, h_j^l, e_{ij})$
$m_i^{l+1} = \sum_{v_j \in N(v_i)} m_{ij}^{l+1}$
$h_i^{l+1} = U_l(h_i^l, m_i^{l+1})$
$g = R(h_i^L, h_i^0)$

MPNN examples

- Convolutional Networks for Learning Molecular Fingerprints
- Gated Graph Neural Networks (GG-NN)
- Interaction Networks
- Molecular Graph Convolutions
- Deep Tensor Neural Networks
- Laplacian Based Method

MPNN examples

- Convolutional Networks for Learning Molecular Fingerprints
- **Gated Graph Neural Networks (GG-NN) - baseline**
- Interaction Networks
- Molecular Graph Convolutions
- Deep Tensor Neural Networks
- Laplacian Based Method

Message functions

1) Matrix Multiplication (Vanilla GG-NN)

$$M(h_v, h_w, e_{vw}) = A_{e_{vw}} h_w$$

Message functions

2) Edge Network

$$M(h_v, h_w, e_{vw}) = A_{e_{vw}} h_w$$

… but A is a neural network now with matrix output

Message functions

3) Pair Message

$$m_{wv} = f\left(h_w^t, h_v^t, e_{vw}\right)$$

It's different because we pass h_v directly to NN and use h_w additionally (unlike previous message functions)

Readout functions

1) Vanilla GG-NN
2) set2set

Experiments for enn-s2s

Table 2. Comparison of Previous Approaches (left) with MPNN baselines (middle) and our methods (right)

Target	BAML	BOB	CM	ECFP4	HDAD	GC	GG-NN	DTNN	enn-s2s	enn-s2s-ens5
mu	4.34	4.23	4.49	4.82	3.34	0.70	1.22	-	**0.30**	0.20
alpha	3.01	2.98	4.33	34.54	1.75	2.27	1.55	-	**0.92**	0.68
HOMO	2.20	2.20	3.09	2.89	1.54	1.18	1.17	-	**0.99**	0.74
LUMO	2.76	2.74	4.26	3.10	1.96	1.10	1.08	-	**0.87**	0.65
gap	3.28	3.41	5.32	3.86	2.49	1.78	1.70	-	**1.60**	1.23
R2	3.25	0.80	2.83	90.68	1.35	4.73	3.99	-	**0.15**	0.14
ZPVE	3.31	3.40	4.80	241.58	1.91	9.75	2.52	-	**1.27**	1.10
U0	1.21	1.43	2.98	85.01	0.58	3.02	0.83	-	**0.45**	0.33
U	1.22	1.44	2.99	85.59	0.59	3.16	0.86	-	**0.45**	0.34
H	1.22	1.44	2.99	86.21	0.59	3.19	0.81	-	**0.39**	0.30
G	1.20	1.42	2.97	78.36	0.59	2.95	0.78	.84[2]	**0.44**	0.34
Cv	1.64	1.83	2.36	30.29	0.88	1.45	1.19	-	**0.80**	0.62
Omega	0.27	0.35	1.32	1.47	0.34	0.32	0.53	-	**0.19**	0.15
Average	2.17	2.08	3.37	53.97	1.35	2.59	1.36	-	**0.68**	0.52

Virtual Graph Elements

Sometimes we want messages to pass through the whole graph

1) Adding a separate “virtual” edge type for pairs of nodes that are not connected
2) “Master” node

Experiments without spatial information

Table 3. Models Trained Without Spatial Information

Model	Average Error Ratio
GG-NN	3.47
GG-NN + Virtual Edge	2.90
GG-NN + Master Node	2.62
GG-NN + set2set	**2.57**

Towers

Problem: single step of the message passing phase for a dense graph requires $O(n^2d^2)$ floating point multiplications

Idea: let's break d-dimensional node embedding h into d/k embeddings and then concatenate them with NN

Using this hack we can achieve $O(n^2d^2/k)$ for this step

Experiments for towers8

Besides the speed towers also improve model quality!

Table 4. Towers vs Vanilla GG-NN (no explicit hydrogen)

Model	Average Error Ratio
GG-NN + joint training	1.92
towers8 + joint training	**1.75**
GG-NN + individual training	1.53
towers8 + individual training	**1.37**

Graphormer

Motivation

Transformer architecture shows great results in NLP and CV

Let’s use it in graphs!

Transformer step

$$h^{'(l)} = \mathbf{MHA}(\mathbf{LN}(h^{(l-1)})) + h^{(l-1)}$$
$$h^{(l)} = \mathbf{FFN}(\mathbf{LN}(h^{'(l)})) + h^{'(l)}$$

LN - layer normalization

MHA - multi-head self-attention

FFN - position-wise feed-forward network

Transformer step

$$h'^{(l)} = \mathrm{MHA}(\mathrm{LN}(h^{(l-1)})) + h^{(l-1)}$$
$$h^{(l)} = \mathrm{FFN}(\mathrm{LN}(h'^{(l)})) + h'^{(l)}$$

LN - layer normalization

MHA - multi-head self-attention - let's take a closer look

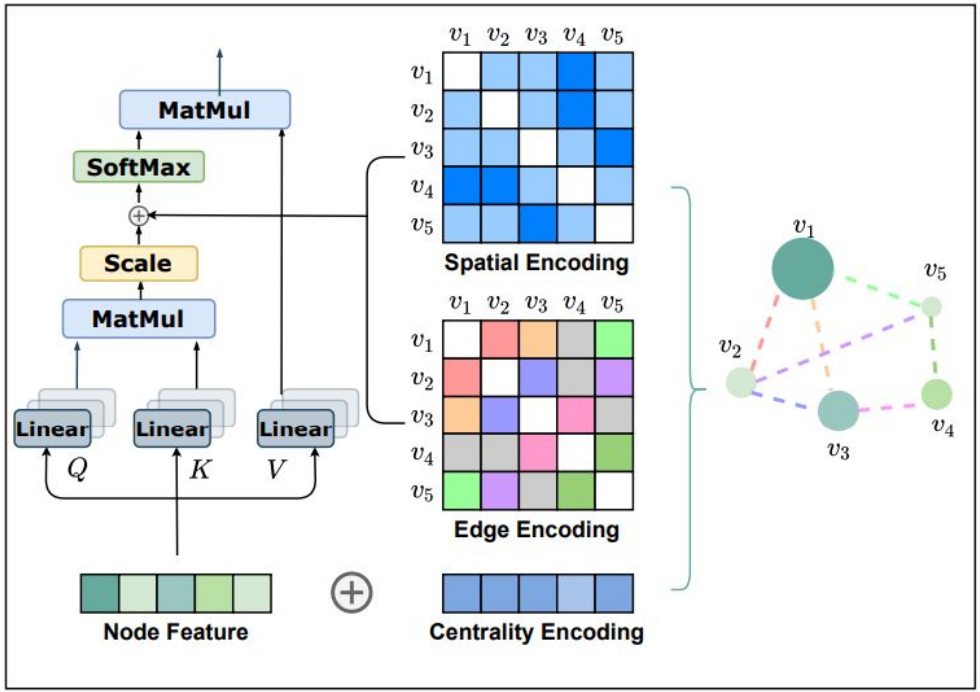
FFN - position-wise feed-forward network

Single-head self-attention

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V,$$

$$A = \frac{QK^\top}{\sqrt{d_K}}, \quad \mathrm{Attn}(H) = \mathrm{softmax}(A)\,V,$$

Self-attention



Encodings

1) Centrality encoding

$$h_i^{(0)} = x_i + z^-_{\deg^-(v_i)} + z^+_{\deg^+(v_i)}$$

Encodings

2) Spatial Encoding

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i, v_j)}$$

Encodings

3) Edge Encoding

$$A_{ij} = \frac{(h_i W_Q)(h_j W_K)^T}{\sqrt{d}} + b_{\phi(v_i, v_j)} + c_{ij}, \text{ where } c_{ij} = \frac{1}{N} \sum_{n=1}^{N} x_{e_n} (w_n^E)^T$$

Experiments

Table 1: Results on PCQM4M-LSC. * indicates the results are cited from the official leaderboard [21].

method	#param.	train MAE	validate MAE
GCN [26]	2.0M	0.1318	0.1691 (0.1684*)
GIN [54]	3.8M	0.1203	0.1537 (0.1536*)
GCN-VN [26, 15]	4.9M	0.1225	0.1485 (0.1510*)
GIN-VN [54, 15]	6.7M	0.1150	0.1395 (0.1396*)
GINE-VN [5, 15]	13.2M	0.1248	0.1430
DeeperGCN-VN [30, 15]	25.5M	0.1059	0.1398
GT [13]	0.6M	0.0944	0.1400
GT-Wide [13]	83.2M	0.0955	0.1408
Graphormer$_{\text{SMALL}}$	12.5M	0.0778	0.1264
Graphormer	47.1M	**0.0582**	**0.1234**

Questions?