N-Gram Graph: Simple Unsupervised Representation for Graphs with Applications to Molecules

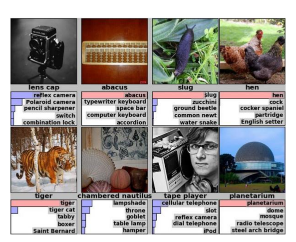
Shengchao Liu, Mehmet Furkan Demirel, Yingyu Liang

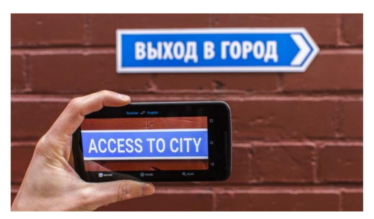
{shengchao, demirel, yliang}@cs.wisc.edu

WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

Motivation

Empirical success of machine learning

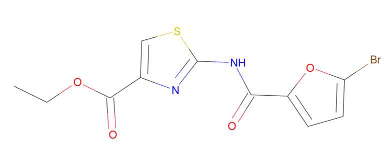

Computer vision

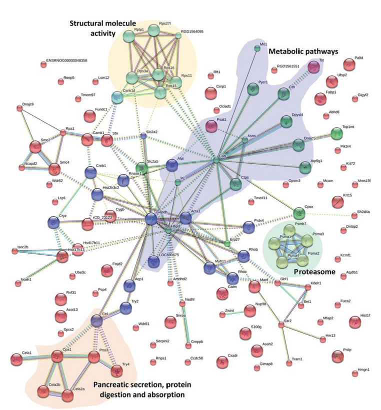


Machine translation

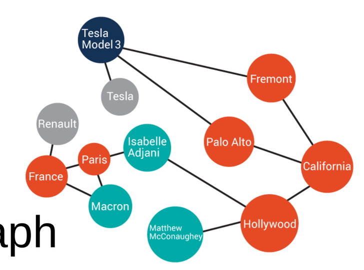
Game playing

What about graph-structured data?


molecule


Protein interaction network


Social networks


Knowledge graph

Such data are ubiquitous in applications in social networks, knowledge graphs, chemistry, biology, material science, etc.

Key challenge: representation as numeric feature

- Fingerprints: Morgan fingerprints via hashing, …
- Graph kernels: Weisfeiler-Lehman kernel, …
- Graph Neural Networks (GNN): GCNN, Weave, …

Our Results

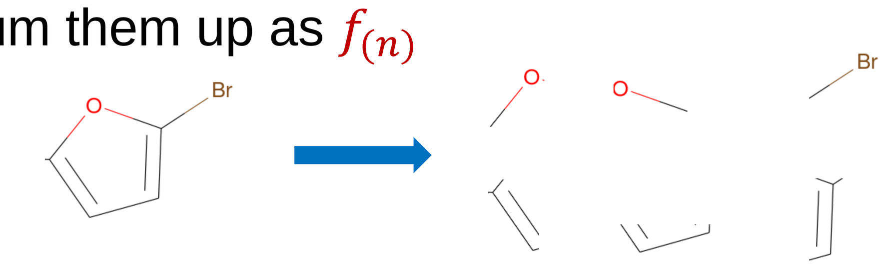
A new representation method for graphs

- **Unsupervised**, so can be used by various learning methods
- **Simple**, relatively fast to compute
- **Strong empirical performance**
  - Outperforms traditional fingerprints/kernels and recent popular GNNs on molecule datasets
  - Preliminary results on other types of data are also strong
- **Strong theoretical power** for representation/prediction
- Inspired by **the N-gram approach in NLP**

N-gram Graph Embedding

Key idea: view a graph as a Bag of Walks

- Enumerate all walks of length n (called n-grams), embed each walk, sum them up as $f_{(n)}$


A molecular graph

Its 2-grams

N-gram Graph (suppose the embeddings for vertices are given):
1. Embed each n-gram: entry-wise product of its vertex embeddings
2. Sum up the embeddings of all n-grams: denote the sum as $f_{(n)}$
3. Repeat for $n = 1, 2, \ldots, T$, and concatenate $f_{(1)}, \ldots, f_{(T)}$

- If vertex embeddings not given: we also provide a method

Equivalent to a simple Graph Neural Network

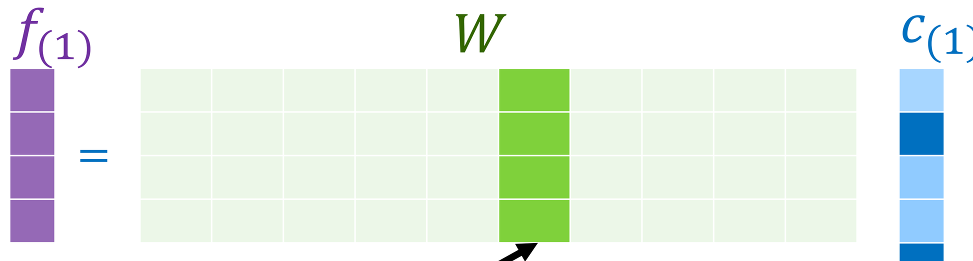
- An efficient version of the method using dynamic programming
- Each vertex holds a latent vector f_i. At each iteration, each vertex updates its latent vector by entry-wise multiplying with the sum of those of its neighbors. Let $\mathcal{A}$ be the adjacent matrix.

$F_{(1)} = F = [f_1, \ldots, f_m], f_{(1)} = F_{(1)}\mathbf{1}$
for each $n \in [2, T]$ **do**
  $F_{(n)} = (F_{(n-1)}\mathcal{A}) \odot F$
  $f_{(n)} = F_{(n)}\mathbf{1}$
end for

Theoretical Analysis

Key idea: compressed sensing on graph statistics

- Count statistics $c_{(n)}$: histogram of different types of n-grams
- Let V be vocabulary of different vertices. $c_{(1)}$ is of dimension $|V|$, i-th coordinate is the times i-th type vertex appears in the graph
- Then $f_{(1)} = Wc_{(1)}$. So $f_{(1)}$ is compressed sensing of $c_{(1)}$ with proper assumptions on the vertex embedding matrix W. This can be used to prove its strong representation and prediction power.



Vertex embedding W: i-th column is the embedding vector for i-th type vertex

- Similar for general n-grams but need more sophisticated analysis

Experiments

- 60 tasks on 10 benchmark molecule datasets
- Evaluated methods
  - Weisfeiler-Lehman kernel + SVM
  - Morgan fingerprints + Random Forest (RF) or XGBoost (XGB)
  - GNN: Graph CNN (GCNN), Weave Neural Network (Weave), Graph Isomorphism Network (GIN)
  - N-gram Graphs + Random Forest (RF) or XGBoost (XGB), with vertex embedding dimension r=100, and T=6
- **Evaluation:** count #times each method gets top-1 and top-3

Table 2: Performance overview: (# of tasks with top-1 performance, # of tasks with top-3 performance) is listed for each model and each dataset. For cases with no top-3 performance on that dataset are left blank. Some models are not well tuned or too slow and are left in "-".

Dataset	# Task	Eval Metric	WL SVM	Morgan RF	Morgan XGB	GCNN	Weave	GIN	N-Gram RF	N-Gram XGB
Delaney	1	RMSE					1, 1	–	0, 1	0, 1
Malaria	1	RMSE		1, 1				–	0, 1	0, 1
CEP	1	RMSE		1, 1				–	0, 1	0, 1
QM7	1	MAE					0, 1	–	0, 1	1, 1
QM8	12	MAE		1, 4	0, 1	7, 12	2, 6	–	0, 2	2, 11
QM9	12	MAE	–		0, 1	4, 7	1, 8	–	0, 8	7, 12
Tox21	12	ROC-AUC	0, 2	0, 7		0, 2	0, 1		3, 12	9, 12
clintox	2	ROC-AUC	0, 1			1, 2	0, 1			1, 2
MUV	17	PR-AUC	4, 12	5, 11	5, 11			0, 7	2, 4	1, 6
HIV	1	ROC-AUC		1, 1					0, 1	0, 1
Overall	60		4, 15	9, 25	5, 13	12, 23	4, 18	0, 7	5, 31	**21, 48**

  - N-gram+XGB: top-1 for 21 in 60 tasks, and top-3 for 48
  - **N-gram graph overall better than the other methods**
- **Runtime: relatively fast**

Table 4: Representation construction time in seconds. One task from each dataset as an example. Average over 5 folds, and including both the training set and test set.

Task	Dataset	WL CPU	Morgan FPs CPU	GCNN GPU	Weave GPU	GIN GPU	Vertex, Emb GPU	Graph, Emb GPU
Delaney	Delaney	2.46	0.25	39.70	65.82	–	49.63	2.90
Malaria	Malaria	128.81	5.28	377.24	536.99	–	1152.80	19.58
CEP	CEP	1113.35	17.69	607.23	849.37	–	2695.57	37.40
QM7	QM7	60.24	0.98	103.12	76.48	–	173.50	10.60
E1-CC2	QM8	584.98	3.60	382.72	262.16	–	966.49	33.43
mu	QM9	–	19.58	9051.37	1504.77	–	8279.03	169.72
NR-AR	Tox21	70.35	2.03	130.15	142.59	608.57	525.24	10.81
CT-TOX	Clintox	4.92	0.63	62.61	95.50	135.68	191.93	3.83
MUV-466	MUV	276.42	6.31	401.02	690.15	1327.26	1221.25	25.50
HIV	HIV	2284.74	17.16	1142.77	2138.10	3641.52	3975.76	139.85

- **Transferrable vertex embeddings: vertex embeddings can be pre-trained** on one dataset and used for different datasets; even random vertex embeddings get competitive results

Table 3: AUC-ROC of N-Gram graph with XGB on 12 tasks from Tox21. Six vertex embeddings are considered: non-transfer (trained on Tox21), vertex embeddings generated randomly and learned from 4 other datasets.

	Non-Transfer	Random	Delaney	CEP	MUV	Clintox
NR-AR	0.791	0.790	0.785	0.787	0.796	0.780
NR-AR-LBD	0.864	0.846	0.863	0.849	0.864	0.867
NR-AhR	0.902	0.895	0.903	0.892	0.901	0.903
NR-Aromatase	0.869	0.858	0.867	0.848	0.858	0.866
NR-ER	0.753	0.751	0.752	0.740	0.735	0.747
NR-ER-LBD	0.838	0.820	0.843	0.820	0.827	0.847
NR-PPAR-gamma	0.851	0.809	0.862	0.813	0.832	0.857
SR-ARE	0.835	0.823	0.841	0.814	0.835	0.842
SR-ATAD5	0.860	0.830	0.844	0.817	0.845	0.857
SR-HSE	0.812	0.777	0.806	0.768	0.805	0.810
SR-MMP	0.918	0.909	0.918	0.902	0.916	0.919
SR-p53	0.868	0.856	0.869	0.841	0.856	0.870

- **Code available**: https://github.com/chao1224/n_gram_graph