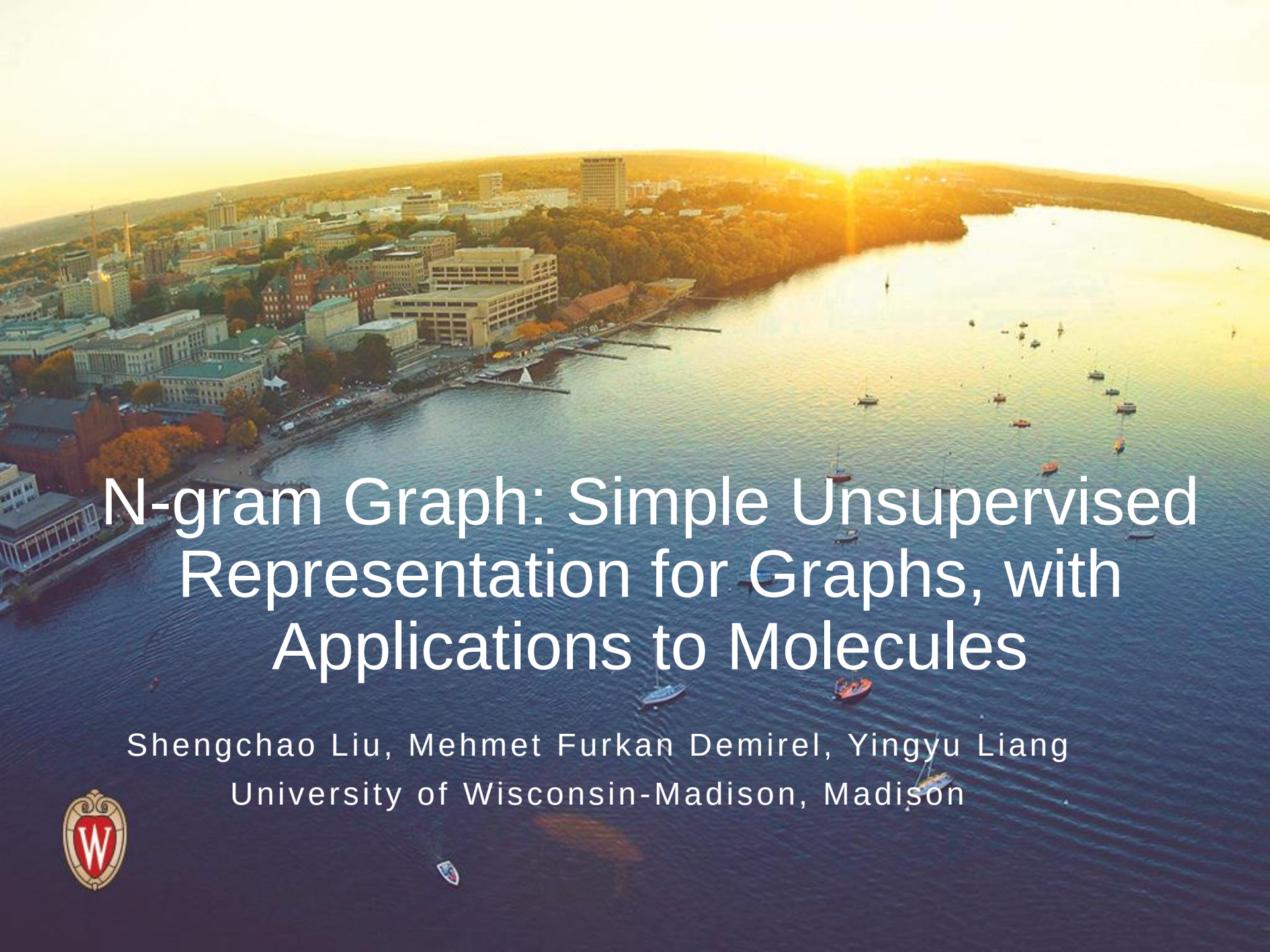
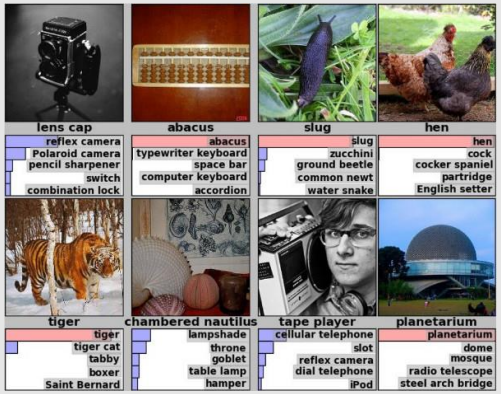


N-gram Graph: Simple Unsupervised Representation for Graphs, with Applications to Molecules

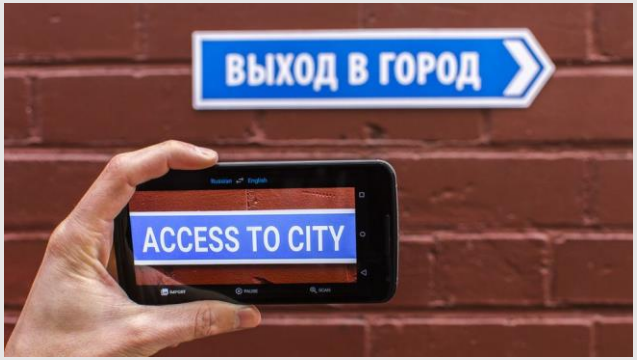
Shengchao Liu, Mehmet Furkan Demirel, Yingyu Liang
University of Wisconsin-Madison, Madison

Machine Learning Progress

- Significant progress in Machine Learning

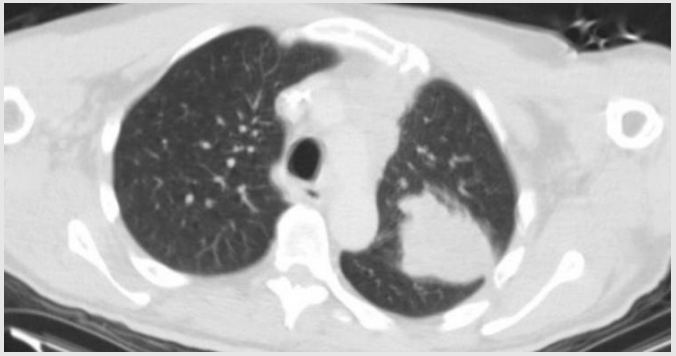


Computer vision



Machine translation

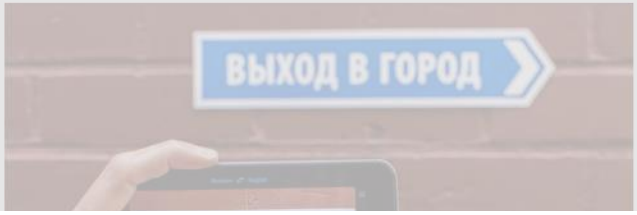
Game Playing

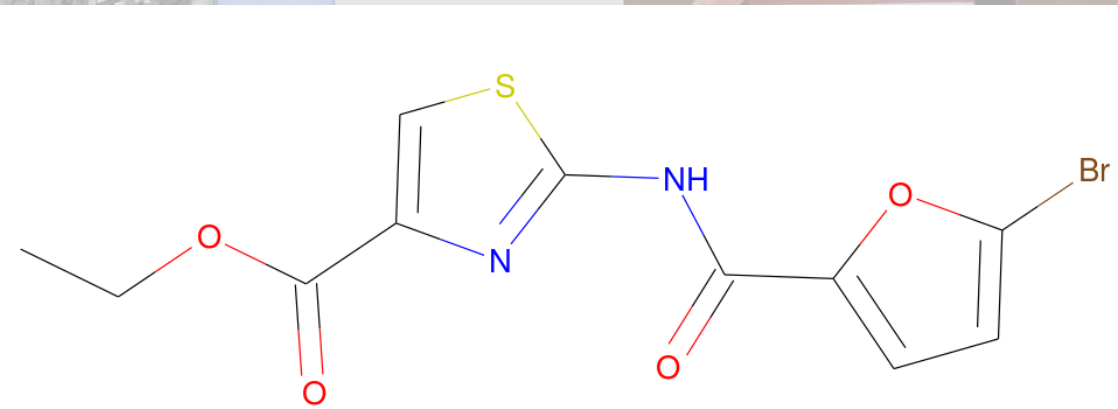


Medical Imaging

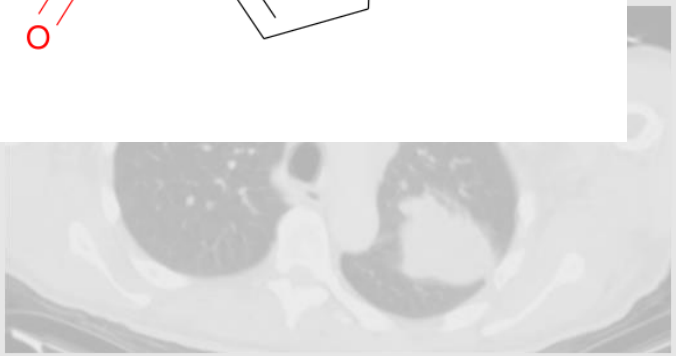
ML for Molecules?

- Significant progress in Machine Learning





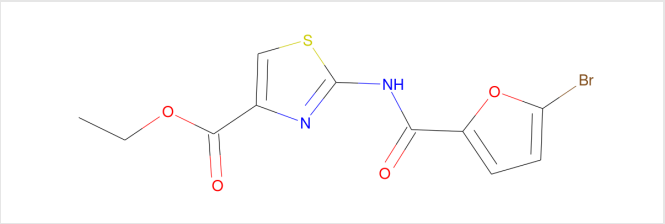
Game Playing

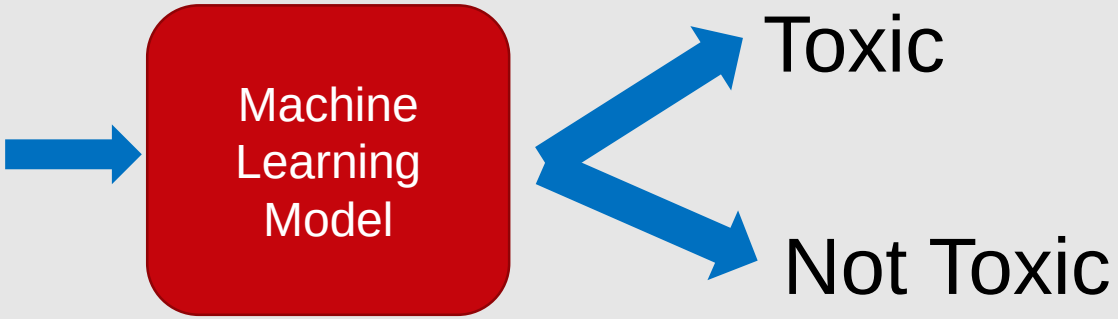


Medical Imaging

ML for Molecules?

- Molecule property prediction





Challenge: Representations

- Input to traditional ML models: vectors

- How to represent molecules as vectors?
  - Fingerprints: Morgan fingerprints, etc
  - Graph kernels: Weisfeiler-Lehman kernel, etc
  - Graph Neural Networks (GNN): Graph CNN, Weave, etc

- Fingerprints/kernels: unsupervised, fast to compute
- GNN: supervised end-to-end, more expensive; powerful

Our method: N-gram Graphs

- Unsupervised
- Relatively fast to compute
- Strong prediction performance
  - Overall better than traditional fingerprint/kernel and popular GNNs

- Inspired by N-gram approach in Natural Language Processing

N-gram Approach in NLP

- n-gram is a consecutive sequence of n words in a sentence

- Example: “this molecule looks beautiful”
- Its 2-grams: “this molecule”, “molecule looks”, “looks beautiful”

N-gram Approach in NLP

- n-gram is a consecutive sequence of n words in a sentence

- Example: "this molecule looks beautiful"
- Its 2-grams: "this molecule", "molecule looks", "looks beautiful"

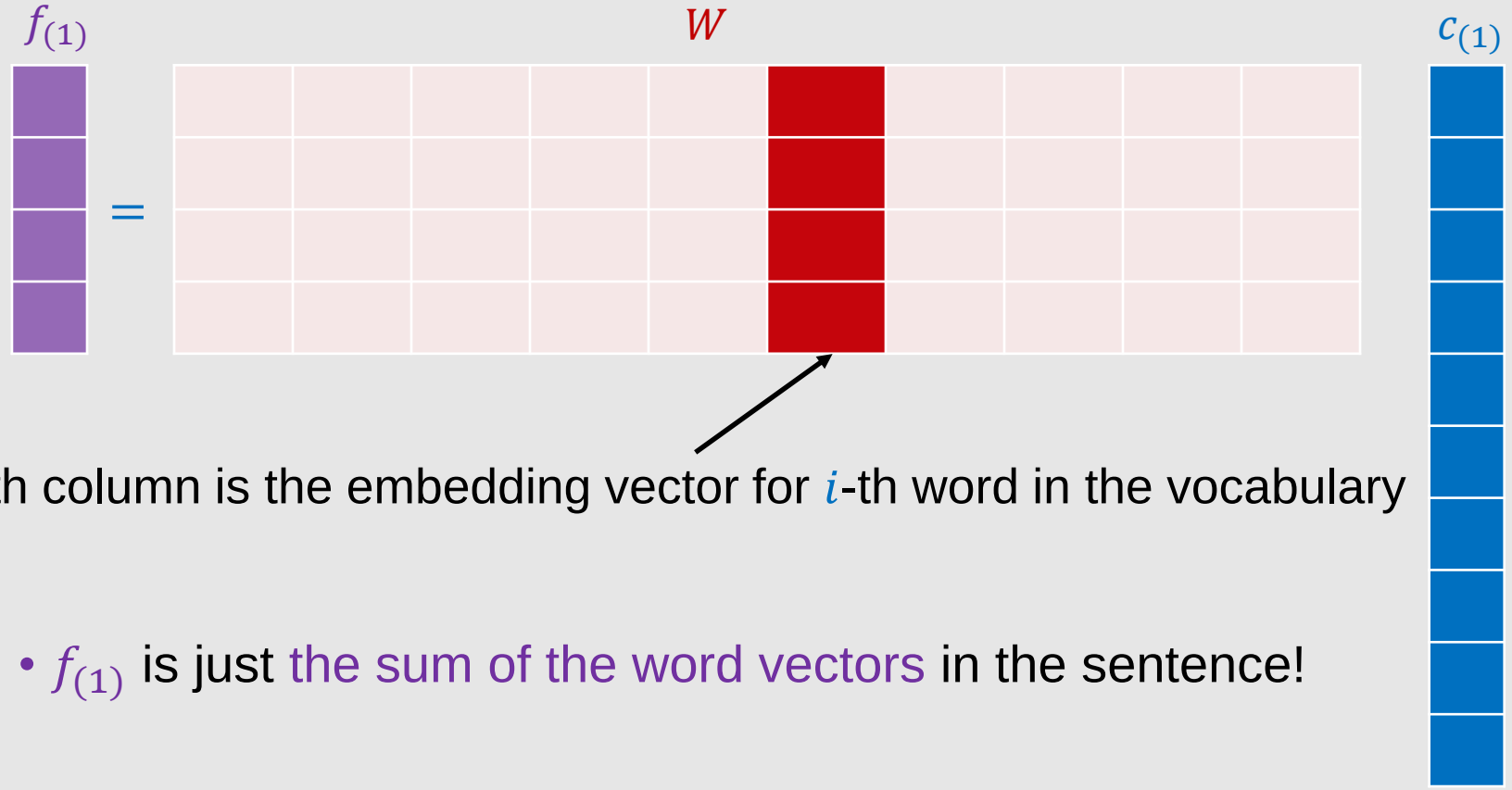
- N-gram count vector $c_{(n)}$ is a numeric representation vector
  - coordinates correspond to all n-grams
  - coordinate value is the number of times the corresponding n-gram shows up in the sentence
- Example: $c_{(1)}$ is just the histogram of the words in the sentence

Dimension Reduction by Embeddings

- N-gram vector $c_{(n)}$ has high dimensions: $|V|^n$ for vocabulary V
- Dimension reduction by word embeddings: $f_{(1)} = W c_{(1)}$

Dimension Reduction by Embeddings

- N-gram vector $c_{(n)}$ has high dimensions: $|V|^n$ for vocabulary V
- Dimension reduction by word embeddings: $f_{(1)} = W c_{(1)}$



i-th column is the embedding vector for i-th word in the vocabulary

- $f_{(1)}$ is just the sum of the word vectors in the sentence!

Dimension Reduction by Embeddings

- N-gram vector $c_{(n)}$ has high dimensions: $|V|^n$ for vocabulary V
- Dimension reduction by word embeddings: $f_{(1)} = W c_{(1)}$

For general n:

- Embedding of an n-gram: entrywise product of its word vectors
- $f_{(n)}$: sum of embeddings of the n-grams in the sentence

N-gram Graphs

- Sentence: linear graph on words
- Molecule: graph on atoms with attributes

Analogy:

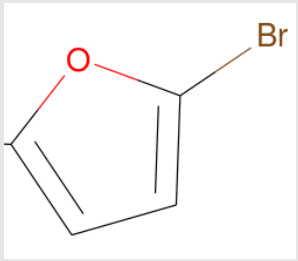
- Atoms with different attributes: different words
- Walks of length n: n-grams

N-gram Graphs

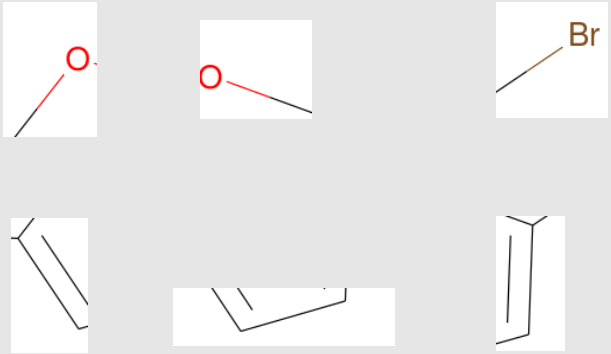
- Sentence: linear graph on words
- Molecule: graph on atoms with attributes

Analogy:

- Atoms with different attributes: different words
- Walks of length n: n-grams



A molecular graph



Its 2-grams

N-gram Graph Algorithm

- Sentence: linear graph on words
- Molecule: graph on atoms with attributes

Given the embeddings for the atoms (vertex vectors)

- Enumerate all n-grams (walks of length n)
- Embedding of an n-gram: entrywise product of its vertex vectors
- $f_{(n)}$: sum of embeddings of the n-grams
- Final N-gram Graph embedding f_G: concatenation of $f_{(1)}, \ldots, f_{(T)}$

N-gram Graph Algorithm

- Sentence: linear graph on words
- Molecule: graph on atoms with attributes

Given the embeddings for the atoms (vertex vectors)

- Enumerate all n-grams (walks of length n)
- Embedding of an n-gram: entrywise product of its vertex vectors
- $f_{(n)}$: sum of embeddings of the n-grams
- Final N-gram Graph embedding f_G: concatenation of $f_{(1)}, \ldots, f_{(T)}$

- Vertex vectors: trained by an algorithm similar to node2vec

N-gram Graphs as Simple GNNs

- Efficient dynamic programming version of the algorithm

- Given vectors f_i for vertices i, and the graph adjacent matrix A

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

- Equivalent to a simple GNN without parameters!

Experimental Results

- 60 tasks on 10 datasets from [1]

- 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)
  - Vertex embedding dimension $r = 100,$ and $T = 6$

[1] Wu, Zhenqin, et al. "MoleculeNet: a benchmark for molecular machine learning." *Chemical science* 9.2 (2018): 513-530.

Experimental Results

- N-gram+XGB: top-1 for 21 among 60 tasks, and top-3 for 48
- Overall better than the other methods

Table 2: Performance overview: top-1 and top-3 tasks when applying each method on each dataset, marked by (# of tasks with top-1 performance, # of tasks with top-3 performance). For method has no top-1 and top-3 performance on all the tasks in the corresponding dataset, it is left blank.

Dataset	Type	# Task	Eval Metric	WL	Morgan RF	Morgan XGB	GCNN	Weave	GIN	N-Gram RF	N-Gram XGB
delaney	Regression	1	RMSE					1, 1	–	0, 1	0, 1
malaria	Regression	1	RMSE		1, 1				–	0, 1	0, 1
cep	Regression	1	RMSE		1, 1				–	0, 1	0, 1
qm7	Regression	1	MAE					0, 1	–	0, 1	1, 1
qm8	Regression	12	MAE		1, 4	0, 1	7, 12	2, 6	–	0, 2	2, 11
qm9	Regression	12	MAE	–		0, 1	4, 7	1, 8	–	0, 8	7, 12
tox21	Classification	12	ROC-AUC	0, 2	0, 7		0, 2	0, 1		3, 12	9, 12
clintox	Classification	2	ROC-AUC	0, 1			1, 2	0, 1			1, 2
muv	Classification	17	PR-AUC	4, 12	5, 11	5, 11			0, 7	2, 4	1, 6
hiv	Classification	1	ROC-AUC		1, 1					0, 1	0, 1
Overall	Classification	60		4, 15	9, 25	5, 13	12, 23	4, 18	0, 7	5, 31	**21, 48**

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	Vertex Emb GPU	Graph Emb GPU
delaney	delaney	2.46	0.25	44.29	53.17	49.63	2.90
malaria	malaria	128.81	5.28	435.73	560.96	1152.80	19.58
cep	cep	1113.35	17.69	721.30	889.24	2695.57	37.40
qm7	qm7	60.24	0.98	118.13	79.70	173.50	10.60
E1-CC2	qm8	584.98	3.60	437.25	273.45	966.49	33.43
mu	qm9	–	19.58	2984.59	1570.70	8279.03	169.72
NR-AR	tox21	70.35	2.03	161.62	152.50	525.24	10.81
CT-TOX	clintox	4.92	0.63	77.89	101.26	191.93	3.83
MUV-466	muv	276.42	6.31	508.36	472.97	1221.25	25.50
hiv	hiv	2284.74	17.16	1412.11	2287.12	3975.76	139.85

Theoretical Analysis

- Recall $f_{(1)} = W c_{(1)}$
  - W is the vertex embedding matrix
  - $c_{(1)}$ is the count vector
- With sparse $c_{(1)}$ and random W, $c_{(1)}$ can be recovered from $f_{(1)}$
  - Well-known in compressed sensing

Theoretical Analysis

- Recall $f_{(1)} = W c_{(1)}$
  - W is the vertex embedding matrix
  - $c_{(1)}$ is the count vector
- With sparse $c_{(1)}$ and random W, $c_{(1)}$ can be recovered from $f_{(1)}$
  - Well-known in compressed sensing

- In general, $f_{(n)} = T_{(n)} c_{(n)}$, for some linear mapping $T_{(n)}$ depending on W
- With sparse $c_{(n)}$ and random W, $c_{(n)}$ can be recovered from $f_{(n)}$

Theoretical Analysis

- Recall $f_{(1)} = W c_{(1)}$
  - W is the vertex embedding matrix
  - $c_{(1)}$ is the count vector
- With sparse $c_{(1)}$ and random W, $c_{(1)}$ can be recovered from $f_{(1)}$
  - Well-known in compressed sensing

- In general, $f_{(n)} = T_{(n)} c_{(n)}$, for some linear mapping $T_{(n)}$ depending on W
- With sparse $c_{(n)}$ and random W, $c_{(n)}$ can be recovered from $f_{(n)}$

- So $f_{(n)}$ preserves the information in $c_{(n)}$
- Furthermore, can prove: regularized linear classifier on $f_{(n)}$ is competitive to the best linear classifier on $c_{(n)}$


THANK YOU!