Walk Aggregation Graph Networks

with Applications to Molecules

Mehmet F. Demirel

Qualifying Presentation

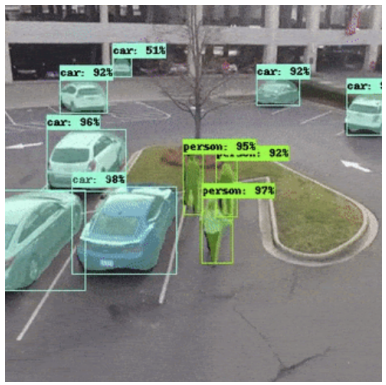
Committee: Yingyu Liang, Jerry Zhu, Dimitris Papailiopoulos

January 13, 2021

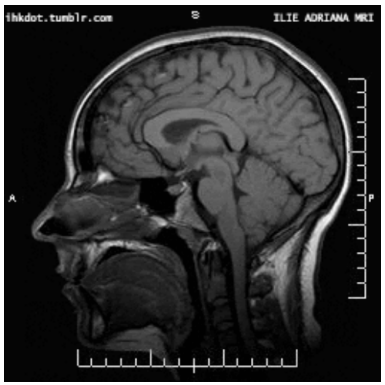
Overview

1. Molecular Property Prediction and Representations for Molecules
2. Existing Approaches to Building Representations for Molecules
3. Our Approach I: N-gram Graph
4. Our Approach II: AWARE

Machine Learning Everywhere

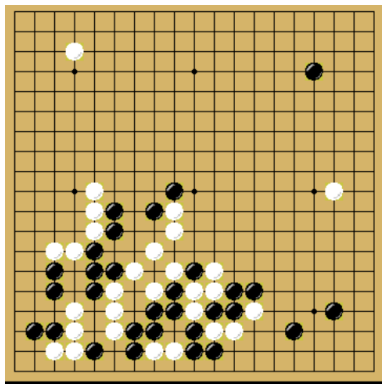


Computer Vision



Medical Imaging

NLP

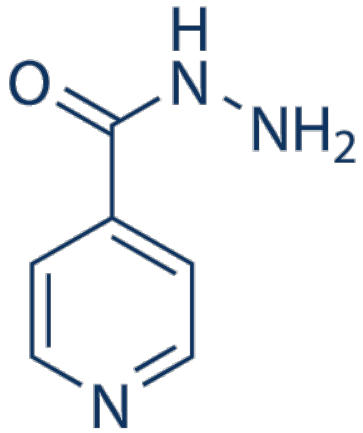


Game Playing

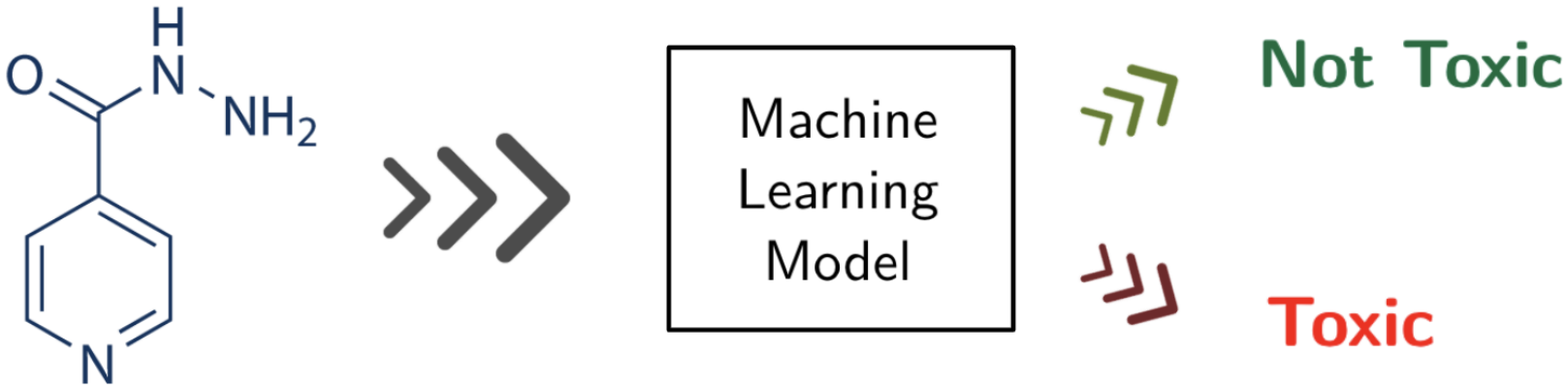
And more!

Molecules?

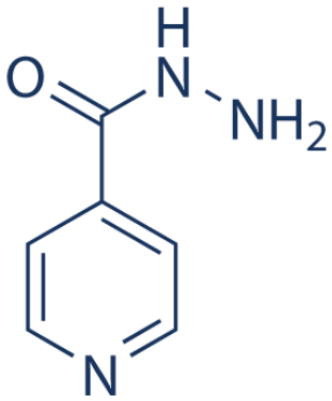
What about molecules?



Molecular Property Prediction



Molecular Property Prediction



HOW?

Machine Learning Model

Not Toxic

Toxic

Challenge: Representation

Input to traditional machine learning models: **vectors**

How to represent a molecule as a vector?

- Fingerprints *e.g.* Morgan fingerprints
- Graph kernels *e.g.* WL-kernel
- Graph neural networks (GNN): GCN, Weave

Fingerprints/kernels are unsupervised and fast to compute.
GNNs are end-to-end supervised, more expensive; but powerful.

Challenge: Representation

Input to traditional machine learning models: **vectors**

How to represent a molecule as a vector?

- Fingerprints *e.g.* Morgan fingerprints
- Graph kernels *e.g.* WL-kernel
- Graph neural networks (GNN): GCN, Weave

Fingerprints/kernels are unsupervised and fast to compute.
GNNs are end-to-end supervised, more expensive; but powerful.

Challenge: Representation

Input to traditional machine learning models: **vectors**

How to represent a molecule as a vector?

- Fingerprints *e.g.* Morgan fingerprints
- Graph kernels *e.g.* WL-kernel
- Graph neural networks (GNN): GCN, Weave

Fingerprints/kernels are unsupervised and fast to compute.
GNNs are end-to-end supervised, more expensive; but powerful.

N-gram Graph

Our previous work [1] is inspired by the **N-gram approach in NLP**.

- Unsupervised
- Fast to compute
- Overall better performance than traditional methods

[1] Liu, Shengchao, **Mehmet F. Demirel**, and Yingyu Liang. "N-gram graph: Simple unsupervised representation for graphs, with applications to molecules." Advances in Neural Information Processing Systems. 2019.

N-gram approach in NLP

n-gram is a contiguous sequence of n words from a given sentence.

"I love living in Madison"

N-gram approach in NLP

n-gram is a contiguous sequence of n words from a given sentence.

"I love living in Madison"

N-gram approach in NLP

n-gram is a contiguous sequence of n words from a given sentence.

"**I love** living in Madison"

- **2-grams:** "I love"

N-gram approach in NLP

n-gram is a contiguous sequence of n words from a given sentence.

"I **love living** in Madison"

- **2-grams:** "I love", "love living"

N-gram approach in NLP

n-gram is a contiguous sequence of n words from a given sentence.

"I love **living in** Madison"

- **2-grams:** "I love", "love living", "living in"

N-gram approach in NLP

n-gram is a contiguous sequence of n words from a given sentence.

"I love living **in Madison**"

- **2-grams:** "I love", "love living", "living in", "in Madison"

N-gram approach in NLP

n-gram is a contiguous sequence of n words from a given sentence.

"I love living in Madison"

- **1-grams:** "I", "love", "living", "in ", "Madison"
- **2-grams:** "I love", "love living", "living in", "in Madison"
- **3-grams:** "I love living", "love living in", "living in Madison"

⋮

N-gram approach in NLP

n-gram is a contiguous sequence of n words from a given sentence

N-gram count vector $c_{(n)}$ is a numeric representation vector:

- its coordinates correspond to all n-grams
- its coordinate values are the number of times the corresponding n-gram shows up in the sentence

Notice that $c_{(1)}$ is just the histogram of the words in the sentence.

N-gram approach in NLP

n-gram is a contiguous sequence of n words from a given sentence

N-gram count vector $c_{(n)}$ is a numeric representation vector:

- its coordinates correspond to all n-grams
- its coordinate values are the number of times the corresponding n-gram shows up in the sentence

Notice that $c_{(1)}$ is just the histogram of the words in the sentence.

Dimensionality Reduction

Problem: N-gram vector $c_{(n)}$ has high dimensions: $|V|^n$ for vocabulary V.

Solution: Dimensionality reduction by word embeddings: $f_{(1)} = W c_{(1)}$

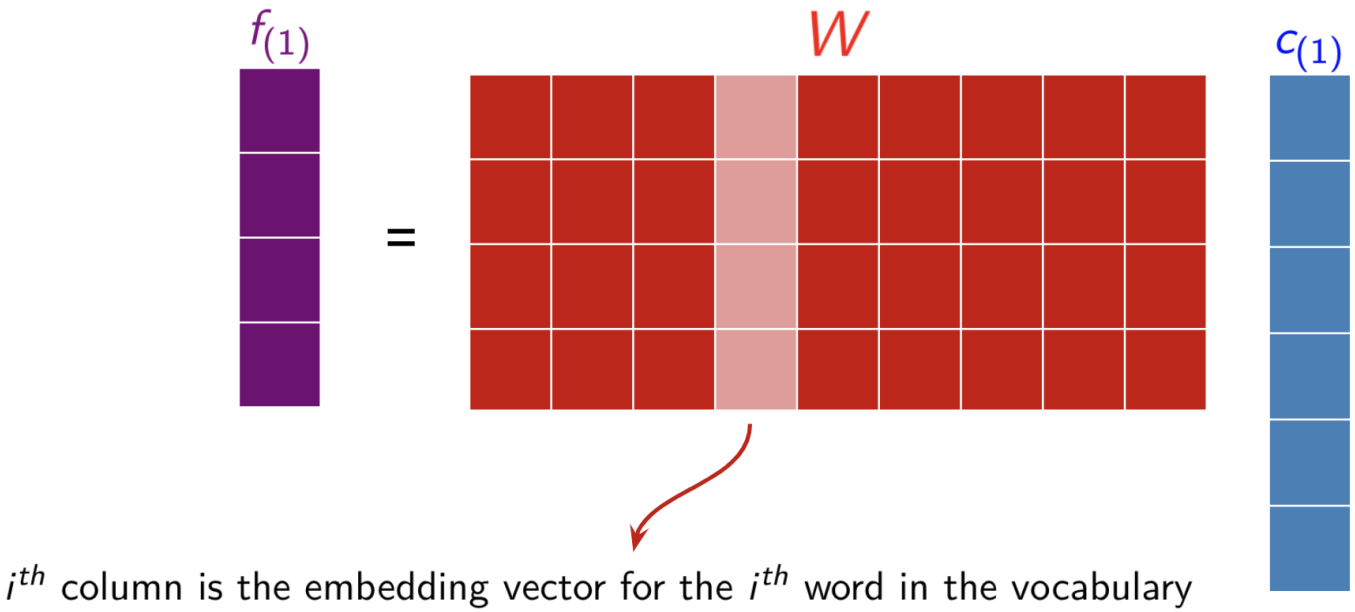
Dimensionality Reduction

Problem: N-gram vector $c_{(n)}$ has high dimensions: $|V|^n$ for vocabulary V.

Solution: Dimensionality reduction by word embeddings: $f_{(1)} = W c_{(1)}$

Problem: N-gram vector $c_{(n)}$ has high dimensions: $|V|^n$ for vocabulary V.

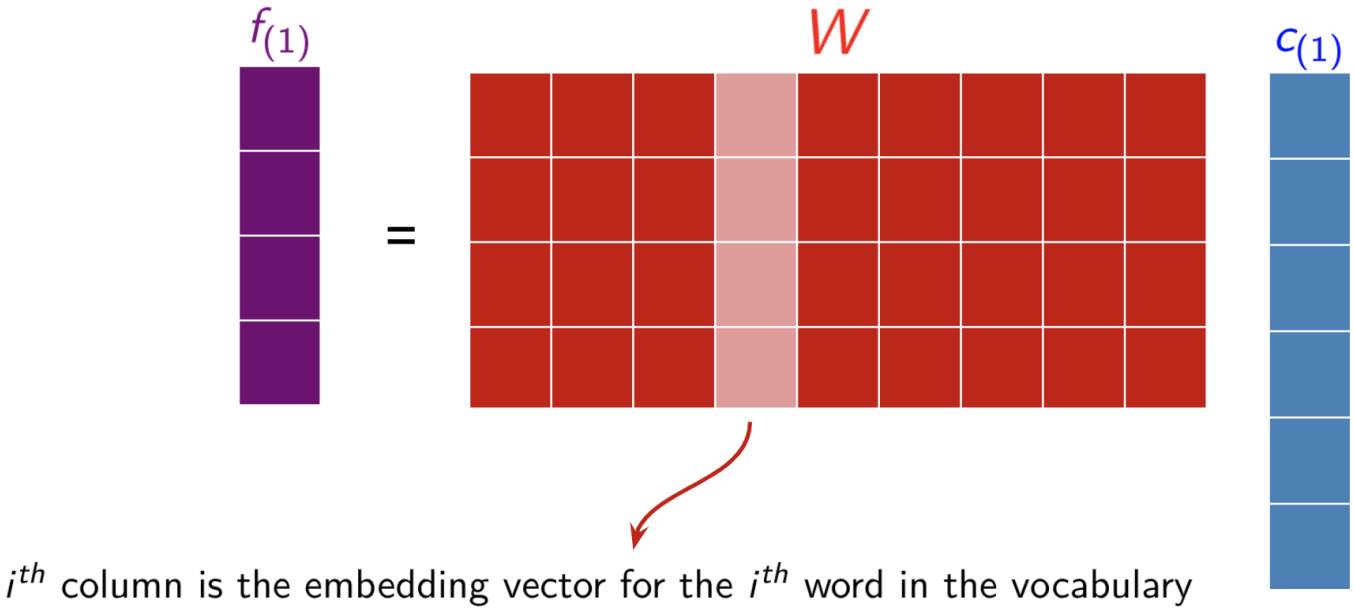
Solution: Dimensionality reduction by word embeddings: $f_{(1)} = W c_{(1)}$



Dimensionality Reduction

Problem: N-gram vector $c_{(n)}$ has high dimensions: $|V|^n$ for vocabulary V.

Solution: Dimensionality reduction by word embeddings: $f_{(1)} = W c_{(1)}$



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

Dimensionality Reduction

Problem: N-gram vector $c_{(n)}$ has high dimensions: $|V|^n$ for vocabulary V.

Solution: Dimensionality reduction by word embeddings: $f_{(1)} = W c_{(1)}$

For general n:

- Embedding of an n-gram is the entry-wise product of its word vectors.
- $f_{(n)}$ is the sum of the embeddings of the n-grams in the sentence.

N-gram *graphs*?

Sentences are linear graphs on words.
Molecules are graphs on atoms with attributes!

- atom symbol, atom degree, is-acceptor, ...

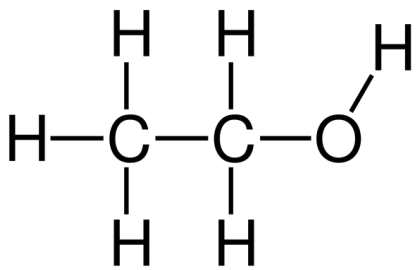
N-gram *graphs*?

Sentences are linear graphs on words.
Molecules are graphs on atoms with attributes!

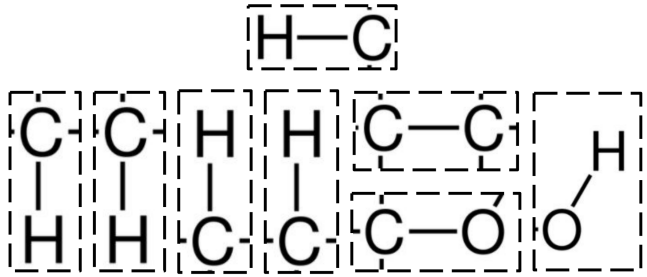
- atom symbol, atom degree, is-acceptor, ...

We can view:

- atoms with different attributes as different words
- walks of length n as n-grams.



A molecule



Its 2-grams

N-gram Graph algorithm

Sentences are linear graphs on words.
Molecules are graphs 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: entry-wise product of its vertex vectors
- $f_{(n)}$: sum of embeddings of all n-grams
- Final N-gram Graph embedding f_G: concatenation of $f_{(1)}, f_{(2)}, \ldots, f_{(T)}$

N-gram Graph as simple GNN

GNN keeps a vector h_i for each vertex in the graph and uses some neighborhood aggregation strategy that iteratively updates by aggregating those of its neighbors.

$$f_i^{(k)} = \textbf{AGGREGATE}^{(k)}\left(\{h_j^{(k-1)} : j \in \text{Neighbor(i)}\}\right)$$
$$h_i^{(k)} = \textbf{COMBINE}^{(k)}\left(h_i^{(k-1)}, f_i^{(k)}\right)$$

N-gram Graph as simple GNN

Given vectors f_i for vertices i and graph adjacency matrix $\mathcal{A}$:

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

Equivalent to a simple GNN without any parameters!

Experimental Results

60 tasks on 10 datasets from MoleculeNet [2].

Methods:

- WL-Kernel + SVM
- Morgan FP + RF or XGB
- Graph CNN (GCNN), Weave Neural Network, Graph Isomorphism Network (GIN)

- N-gram Graph + RF or XGB
- Vertex embedding dimension $r = 100$ and $T = 6$

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

Experimental Results

N-gram Graph + XGB: top-1 in 21 and top-3 in 48 out of 60 tasks
Overall better performance than other methods

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**

Assumptions.

- Each vertex has $S = 1$ attribute and it takes values from a set of size K. $W \in \mathbb{R}^{r \times K}$ is the vertex embedding matrix.
- Assume for simplicity that no two vertices in a walk p can have the same attribute value.

Idea: There exists a linear mapping $W^{[n]}$ such that $f_{(n)} = W^{[n]} c_{(n)}$.

- For $S = 1$, $W^{[n]}$ is the n-way column Hadamard product of W.
- There are a wide family of prior distributions on W such that $W^{[n]}$ has RIP with high probability.

  $\Longrightarrow$ With sparse $c_{(n)}$, $c_{(n)}$ can efficiently be recovered from $f_{(n)}$.
- A linear classifier learned on $f_{(n)}$ is comparable to the best linear classifier on $c_{(n)}$.

Assumptions.

- Each vertex has $S = 1$ attribute and it takes values from a set of size K. $W \in \mathbb{R}^{r \times K}$ is the vertex embedding matrix.
- Assume for simplicity that no two vertices in a walk p can have the same attribute value.

Idea: There exists a linear mapping $W^{[n]}$ such that $f_{(n)} = W^{[n]} c_{(n)}$.

- For $S = 1$, $W^{[n]}$ is the n-way column Hadamard product of W.
- There are a wide family of prior distributions on W such that $W^{[n]}$ has RIP with high probability.

  $\Longrightarrow$ With sparse $c_{(n)}$, $c_{(n)}$ can efficiently be recovered from $f_{(n)}$.
- A linear classifier learned on $f_{(n)}$ is comparable to the best linear classifier on $c_{(n)}$.

Assumptions.

- Each vertex has $S = 1$ attribute and it takes values from a set of size K. $W \in \mathbb{R}^{r \times K}$ is the vertex embedding matrix.
- Assume for simplicity that no two vertices in a walk p can have the same attribute value.

Idea: There exists a linear mapping $W^{[n]}$ such that $f_{(n)} = W^{[n]} c_{(n)}$.

- For $S = 1$, $W^{[n]}$ is the n-way column Hadamard product of W.
- There are a wide family of prior distributions on W such that $W^{[n]}$ has RIP with high probability.

  $\Longrightarrow$ With sparse $c_{(n)}$, $c_{(n)}$ can efficiently be recovered from $f_{(n)}$.
- A linear classifier learned on $f_{(n)}$ is comparable to the best linear classifier on $c_{(n)}$.

Assumptions.

- Each vertex has $S = 1$ attribute and it takes values from a set of size K. $W \in \mathbb{R}^{r \times K}$ is the vertex embedding matrix.
- Assume for simplicity that no two vertices in a walk p can have the same attribute value.

Idea: There exists a linear mapping $W^{[n]}$ such that $f_{(n)} = W^{[n]} c_{(n)}$.

- For $S = 1$, $W^{[n]}$ is the n-way column Hadamard product of W.
- There are a wide family of prior distributions on W such that $W^{[n]}$ has RIP with high probability.

  $\implies$ With sparse $c_{(n)}$, $c_{(n)}$ can efficiently be recovered from $f_{(n)}$.
- A linear classifier learned on $f_{(n)}$ is comparable to the best linear classifier on $c_{(n)}$.

Assumptions.

- Each vertex has $S = 1$ attribute and it takes values from a set of size K. $W \in \mathbb{R}^{r \times K}$ is the vertex embedding matrix.
- Assume for simplicity that no two vertices in a walk p can have the same attribute value.

Idea: There exists a linear mapping $W^{[n]}$ such that $f_{(n)} = W^{[n]} c_{(n)}$.

- For $S = 1$, $W^{[n]}$ is the n-way column Hadamard product of W.
- There are a wide family of prior distributions on W such that $W^{[n]}$ has RIP with high probability.

  $\implies$ With sparse $c_{(n)}$, $c_{(n)}$ can efficiently be recovered from $f_{(n)}$.
- A linear classifier learned on $f_{(n)}$ is comparable to the best linear classifier on $c_{(n)}$.

Current Work: AWARE

Parametrization of the N-gram graph algorithm:
Attentive **W**alk **A**ggregation G**R**aph N**E**twork

The N-gram graph algorithm has no parameters and requires no training. Therefore, it is efficient in computation. However:

- Huge design space for adding trainable parameters.
- Concatenated with a classifier, it becomes end-to-end.

Q: Why parametrize the algorithm, though?

A: Weighting!

Current Work: AWARE

Parametrization of the N-gram graph algorithm:
Attentive **W**alk **A**ggregation G**R**aph N**E**twork

The N-gram graph algorithm has no parameters and requires no training. Therefore, it is efficient in computation. However:

- Huge design space for adding trainable parameters.
- Concatenated with a classifier, it becomes end-to-end.

Q: Why parametrize the algorithm, though?
A: Weighting!

Weighted Features

The downstream learning tasks may prefer certain directions both in the vertex embedding space and final embedding space.

Given vectors $F = [f_1, \ldots, f_m]$ for m vertices and graph adjacency matrix $\mathcal{A}$:

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

Weighted Features

The downstream learning tasks may prefer certain directions both in the vertex embedding space and final embedding space.

Given vectors $F = [f_1, \ldots, f_m]$ for m vertices and graph adjacency matrix $\mathcal{A}$:

$F_{(1)} = \sigma(W_1 F)$, $f_{(1)} = F_{(1)}\mathbf{1}$
for each $n \in [2, T]$ **do**
  $F_{(n)} = (F_{(n-1)}\mathcal{A}) \odot F$
  $f_{(n)} = \sigma(W_3 F_{(n)})\mathbf{1}$
end for
$f_G = [f_{(1)}; \ldots; f_{(T)}]$

Weighted Features

The downstream learning tasks may prefer certain directions both in the vertex embedding space and final embedding space.

Given vectors $F = [f_1, \ldots, f_m]$ for m vertices and graph adjacency matrix $\mathcal{A}$:

$F_{(1)} = \sigma(W_1 F)$, $f_{(1)} = F_{(1)}\mathbf{1}$
for each $n \in [2, T]$ **do**
  $F_{(n)} = (F_{(n-1)}\mathcal{A}) \odot F$
  $f_{(n)} = \sigma(W_3 F_{(n)})\mathbf{1}$
end for
$f_G = [f_{(1)}; \ldots; f_{(T)}]$

Then for a vertex embedding f_i, $W_1 f_i$ will stretch its components along W_1's larger singular vectors while relatively shrink the components along the smaller ones.

Same idea for W_3.

Attentive Messages

Some nodes have more impact on their neighbors.

- Weighted sum of latent vectors from neighbors (with attention).

Given vectors $F = [f_1, \ldots, f_m]$ for m vertices and graph adjacency matrix $\mathcal{A}$:

$F_{(1)} = \sigma(W_1 F), f_{(1)} = F_{(1)}\mathbf{1}$
for each $n \in [2, T]$ **do**
  $F_{(n)} = (F_{(n-1)}\mathcal{A}) \odot F$
  $f_{(n)} = \sigma(W_3 F_{(n)})\mathbf{1}$
end for
$f_G = [f_{(1)}; \ldots; f_{(T)}]$

Attentive Messages

Some nodes have more impact on their neighbors.

- Weighted sum of latent vectors from neighbors (with attention).

Given vectors $F = [f_1, \ldots, f_m]$ for m vertices and graph adjacency matrix $\mathcal{A}$:

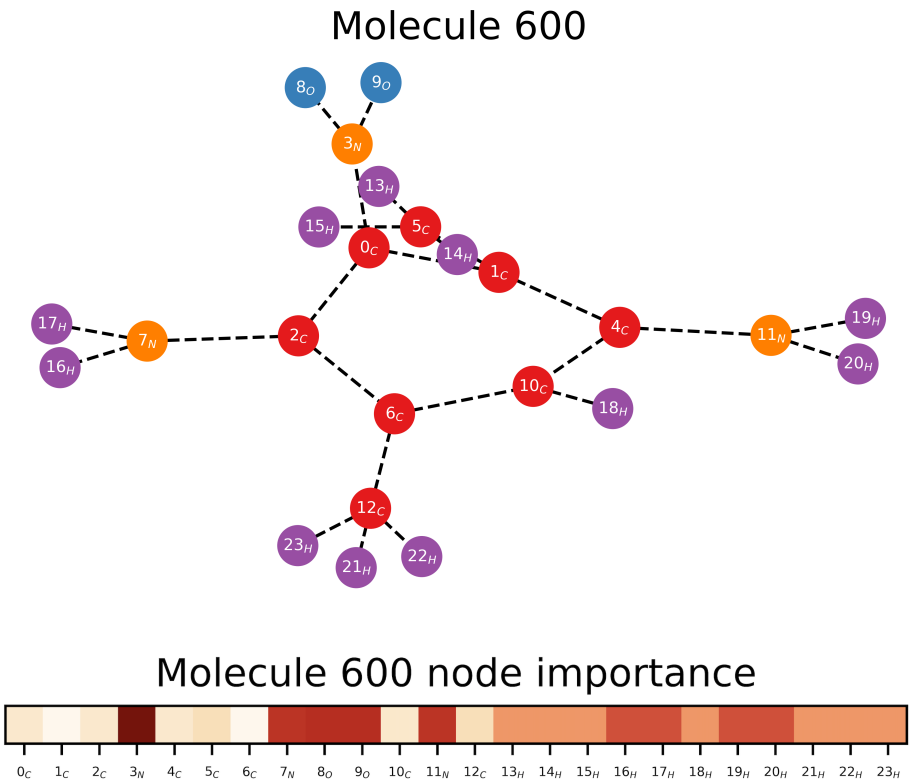
$F_{(1)} = \sigma(W_1 F), f_{(1)} = F_{(1)}\mathbf{1}$
for each $n \in [2, T]$ **do**
$F_{(n)} = (F_{(n-1)}(\mathcal{A} \odot \bar{S}_{(n-1)})) \odot F_{(1)}$
$f_{(n)} = \sigma(W_3 F_{(n)})\mathbf{1}$
end for
$f_G = [f_{(1)}; \ldots; f_{(T)}]$

$$[s_{(n-1)}]_{ji} = [F_{(n-1)}]_j^\top W_2 [F_{(n-1)}]_i$$

$$[\bar{S}_{(n-1)}]_{ji} = \frac{\exp([s_{(n-1)}]_{ji})}{\sum_{k \in \text{neighbors of } i} \exp([s_{(n-1)}]_{ki})}$$

Attentive Messages: Visualization

Mutagenicity dataset: NO_2 and NH_2 atom groups are known to have a mutagenic effect in a molecule.



AWARE performs top-1 in 27 and top-3 in 48 out of 60 tasks
Overall better performance than other methods, including N-gram Graph

Table 3. Overall performance: (# 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	Number of Tasks	Eval Metric	FP + RF	FP + XGB	WL + SVM	GCNN	GAT	GIN	N-Gram + XGB	N-Gram + RF	AWARE
Delaney	1	RMSE							0,1	0,1	1,1
Malaria	1	RMSE	1,1	0,1						0,1	
CEP	1	RMSE		0,1				1,1			0,1
QM7	1	MAE							0,1	0,1	1,1
QM8	12	MAE				5,7		2,6	0,11		5,11
QM9	12	MAE				3,7		4,7	1,11	0,6	4,7
Tox21	12	ROC	0,4		0,2				5,12	1,7	6,11
clintox	2	ROC				0,1	1,1	0,2			1,2
HIV	1	ROC	1,1	0,1					0,1		
MUV	17	ROC	1,7	1,7	3,5	1,9	0,1	1,2	1,4	0,2	9,14
Total	60		3,13	1,10	3,7	9,24	1,2	8,18	7,41	1,18	**27,48**

Theoretical Analysis

Assumption. Each vertex has $S = 1$ attribute and it takes values from a set of size K. $W \in \mathbb{R}^{r \times K}$ is the vertex embedding matrix. Also assume that $W_1 = W_3 = I$ and σ is linear.

Idea: There exists a linear mapping $W^{[n]}$ such that $f_{(n)} = W^{[n]} \Lambda_{(n)} c_{(n)}$ where $\Lambda_{(n)}$ is a diagonal weighting matrix that depends on the scoring matrix $\bar{S}$.

- A similar analysis can be applied to $\Lambda_{(n)} c_{(n)}$ instead of $c_{(n)}$.
- Benefit of weighting: If $\Lambda_{(n)}$ emphasizes important features for prediction, then better to learn over $\Lambda_{(n)} c_{(n)}$ rather than $c_{(n)}$. Thus, we can learn over $f_{(n)} = W^{[n]} \Lambda_{(n)} c_{(n)}$.

Assumption. Each vertex has $S = 1$ attribute and it takes values from a set of size K. $W \in \mathbb{R}^{r \times K}$ is the vertex embedding matrix. Also assume that $W_1 = W_3 = I$ and σ is linear.

Idea: There exists a linear mapping $W^{[n]}$ such that $f_{(n)} = W^{[n]} \Lambda_{(n)} c_{(n)}$ where $\Lambda_{(n)}$ is a diagonal weighting matrix that depends on the scoring matrix $\bar{S}$.

- A similar analysis can be applied to $\Lambda_{(n)} c_{(n)}$ instead of $c_{(n)}$.
- Benefit of weighting: If $\Lambda_{(n)}$ emphasizes important features for prediction, then better to learn over $\Lambda_{(n)} c_{(n)}$ rather than $c_{(n)}$. Thus, we can learn over $f_{(n)} = W^{[n]} \Lambda_{(n)} c_{(n)}$.

Assumption. Each vertex has $S = 1$ attribute and it takes values from a set of size K. $W \in \mathbb{R}^{r \times K}$ is the vertex embedding matrix. Also assume that $W_1 = W_3 = I$ and σ is linear.

Idea: There exists a linear mapping $W^{[n]}$ such that $f_{(n)} = W^{[n]} \Lambda_{(n)} c_{(n)}$ where $\Lambda_{(n)}$ is a diagonal weighting matrix that depends on the scoring matrix $\bar{S}$.

- A similar analysis can be applied to $\Lambda_{(n)} c_{(n)}$ instead of $c_{(n)}$.
- Benefit of weighting: If $\Lambda_{(n)}$ emphasizes important features for prediction, then better to learn over $\Lambda_{(n)} c_{(n)}$ rather than $c_{(n)}$. Thus, we can learn over $f_{(n)} = W^{[n]} \Lambda_{(n)} c_{(n)}$.

Assumption. Each vertex has $S = 1$ attribute and it takes values from a set of size K. $W \in \mathbb{R}^{r \times K}$ is the vertex embedding matrix. Also assume that $W_1 = W_3 = I$ and σ is linear.

Idea: There exists a linear mapping $W^{[n]}$ such that $f_{(n)} = W^{[n]} \Lambda_{(n)} c_{(n)}$ where $\Lambda_{(n)}$ is a diagonal weighting matrix that depends on the scoring matrix $\bar{S}$.

- A similar analysis can be applied to $\Lambda_{(n)} c_{(n)}$ instead of $c_{(n)}$.
- Benefit of weighting: If $\Lambda_{(n)}$ emphasizes important features for prediction, then better to learn over $\Lambda_{(n)} c_{(n)}$ rather than $c_{(n)}$. Thus, we can learn over $f_{(n)} = W^{[n]} \Lambda_{(n)} c_{(n)}$.

Thank you!

Appendix

Theorem 1 *If* $r = \Omega(ns_n^3 \log K)$ *where* s_n *is the sparsity of* $c_{(n)}$*, then there is a prior distribution over* W *so that* $f_{(n)} = T_{(n)}c_{(n)}$ *for a linear mapping* $T_{(n)}$*. If additionally* $c_{(n)}$ *is the sparsest vector satisfying* $f_{(n)} = T_{(n)}c_{(n)}$*, then with probability* $1 - O(S\exp(-(r/S)^{1/3}))$*,* $c_{(n)}$ *can be efficiently recovered from* $f_{(n)}$*.*

Appendix

Prediction Power. Consider a prediction task and let $\ell_{\mathcal{D}}(g)$ denote the risk of a prediction function g over the data distribution $\mathcal{D}$.

Theorem 2 *Let g_c be a prediction function on the count statistics $c_{[T]}$. In the same setting as in Theorem 1, with probability $1 - O(TS\exp(-(r/S)^{1/3}))$, there is a function g_f on the N-gram graph embeddings f_G with risk $\ell_{\mathcal{D}}(g_f) = \ell_{\mathcal{D}}(g_c)$.*

Consider the binary classification task with the logistic loss function $\ell(g, y)$ where g is the prediction and y is the true label. Let $\ell_{\mathcal{D}}(\theta) = \mathbb{E}_{\mathcal{D}}[\ell(g_\theta, y)]$ denote the risk of a linear classifier g_θ with weight vector θ over the data distribution $\mathcal{D}$. Let θ^* denote the weight of the classifier over $c_{[n]}$ minimizing $\ell_{\mathcal{D}}$. Suppose we have a dataset $\{(G_i, y_i)\}_{i=1}^M$ i.i.d. sampled from $\mathcal{D}$, and $\hat{\theta}$ is the weight over f_G which is learned via ℓ_2-regularization with regularization coefficient λ:

$$\hat{\theta} = \arg\min_\theta \frac{1}{M} \sum_{i=1}^{M} \ell(\langle \theta, f_{G_i} \rangle, y_i) + \lambda \|\theta\|_2. \tag{4}$$

Theorem 3 *Assume that f_G is scaled so that $\|f_G\|_2 \leq 1$ for any graph from $\mathcal{D}$. There exists a prior distribution over W, such that with $r = \Omega(\frac{n s_{\max}^3}{\epsilon^2} \log K)$ for $s_{\max} = \max\{s_n : 1 \leq n \leq T\}$ and appropriate choice of regularization coefficient, with probability $1 - \delta - O(TS \exp(-(r/S)^{1/3}))$, the $\hat{\theta}$ minimizing the ℓ_2-regularized logistic loss over the N-gram graph embeddings f_{G_i}'s satisfies*

$$\ell_{\mathcal{D}}(\hat{\theta}) \leq \ell_{\mathcal{D}}(\theta^*) + O\left(\|\theta^*\|_2 \sqrt{\epsilon + \frac{1}{M} \log \frac{1}{\delta}} \right). \tag{5}$$

Theorem 4 (Restatement of Theorem 1.1 in [12]) *Suppose A is $(2k, \epsilon)$-RIP for an $\epsilon < \sqrt{2} - 1$. Let $\hat{x}$ denote the solution to (9), and let x_k denote the vector x with all but the k-largest entries set to zero. Then*

$$\|\hat{x} - x\|_1 \leq C_0 \|x_k - x\|_1$$

and

$$\|\hat{x} - x\|_2 \leq C_0 k^{-1/2} \|x_k - x\|_1.$$

In particular, if x is k-sparse, the recovery is exact.

Furthermore, it has been shown that A is (k, ϵ)-RIP with overwhelming probability when $d = \Omega(k \log \frac{N}{k})$ and $\sqrt{d} A_{ij} \sim \mathcal{N}(0, 1)(\forall i, j)$ or $\sqrt{d} A_{ij} \sim \mathcal{U}\{-1, 1\}(\forall i, j)$.

Definition 3 (ℓ-way Column Hadamard Product) *Let A be a $d \times N$ matrix, and let ℓ be a natural integer. The ℓ-way column Hadamard-product of A is a $d \times \binom{N}{\ell}$ matrix denoted as $A^{(\ell)}$, whose columns indexed by a sequence $1 \leq i_1 < i_2 \cdots < i_\ell \leq d$ is the element-wise product of the $i_1, i_2, \ldots, i_\ell$-th columns of A, i.e., $(i_1, i_2, \ldots, i_\ell)$-th column in $A^{(\ell)}$ is $A_{i_1} \odot A_{i_2} \odot \cdots \odot A_{i_\ell}$ where A_j for $j \in [N]$ is the j-th column in A.*

We have the following theorems:

Theorem 5 (Restatement of Theorem 4.1 in [32]) *Let X be an $n \times d$ matrix, and let A be a $d \times N$ random matrix with independent entries R_{ij} such that $\mathbb{E}[R_{ij}] = 0, \mathbb{E}[R_{ij}] = 1$, and $|R_{ij}| \leq \tau$ almost surely. Let $\epsilon \in (0, 1)$, and let k be an integer satisfying $sr(X) \geq \frac{C\tau^8}{\epsilon^2} k^2 \log \frac{N^2}{k\epsilon}$ for some universal constant $C > 0$. Then with probability at least $1 - \exp(-c\epsilon^2 sr(X)/(k^2\tau^8))$ for some universal constant $c > 0$, the matrix $XA^{(\ell)}/\|X\|_F$ is (k, ϵ)-RIP.*

Here, $\mathrm{sr}(X) = \|X\|_F^2/\|X\|^2$ is the stable rank of X. In our case, we will apply the theorem with X being $\mathbf{I}_{d\times d}/\sqrt{d}$ where $\mathbf{I}_{d\times d} \in \mathbb{R}^{d\times d}$ is the identity matrix.

Theorem 6 (Restatement of Theorem 4.3 in [32]) *Let X be an $n \times d$ matrix, and let A be a $d \times N$ random matrix with independent entries R_{ij} such that $\mathbb{E}[R_{ij}] = 0, \mathbb{E}[R_{ij}] = 1$, and $|R_{ij}| \leq \tau$ almost surely. Let $\ell \geq 3$ be a constant. Let $\epsilon \in (0, 1)$, and let k be an integer satisfying $sr(X) \geq \frac{C\tau^{4\ell}}{\epsilon^2} k^3 \log \frac{N^\ell}{k\epsilon}$ for some universal constant $C > 0$. Then with probability at least $1 - \exp(-c\epsilon^2 sr(X)/(k^2\tau^{4\ell}))$ for some universal constant $c > 0$, the matrix $XA^{(\ell)}/\|X\|_F$ is (k, ϵ)-RIP.*

Appendix

Theorem 7 (Restatement of Theorem 4.2 in [3]) *Suppose A is $(\Delta\mathcal{X}, \epsilon)$-RIP. Then with probability at least $1-\delta$,*

$$\ell_{\mathcal{D}}^{A}(\hat{\theta}_A) \leq \ell_{\mathcal{D}}(\theta^*) + O\left(\lambda_\ell B\|\theta^*\|\sqrt{\epsilon + \frac{1}{M}\log\frac{1}{\delta}}\right)$$

for appropriate choice of C. Here, $\Delta\mathcal{X} = \{x - x' : x, x' \in \mathcal{X}\}$ for any $\mathcal{X} \subseteq \mathbb{R}^N$.

Definition 3 (Walk Weights). *Define the weight for one attribute value v as $w(v) = 1$, and define the weight for a sequence of attribute values $(v_0, \ldots, v_{n-1})$ with $n > 1$ as:*

$$w(v_0, \ldots, v_{n-1}) := \prod_{i=0}^{n-2} S(F_{(1)}(v_i), F_{(1)}(v_{i+1})) \quad (4)$$

where $S(\cdot, \cdot)$ is the score function in Eq (2), $F_{(1)}(v_i) = Wh(v_i)$, and $h(v_i)$ is the one-hot vector for the attribute value v_i.

Theorem 1. *The embedding $f_{(n)}$ is a linear mapping of the walk statistics $c_{(n)}$:*

$$f_{(n)} = W^{[n]} \Lambda_{(n)} c_{(n)} \tag{5}$$

where $W^{[n]}$ is the n-way column products of W, and $\Lambda_{(n)}$ is a K^n-dimensional diagonal matrix, whose columns correspond to all possible length-n sequences of attribute values, with the diagonal entry in the column indexed by a sequence of attribute values $(v_0, \ldots, v_{n-1})$ being its weight $w(v_0, \ldots, v_{n-1})$. Then

$$f_{[T]} := \mathcal{M} \Lambda c_{[T]} \tag{6}$$

where $\mathcal{M}$ is a block-diagonal matrix with diagonal blocks $W, W^{[2]}, \ldots, W^{[T]}$, and Λ a block-diagonal matrix with diagonal blocks $\Lambda_{(1)}, \Lambda_{(2)}, \ldots, \Lambda_{(T)}$.

The intuition for the benefit of appropriate weighting is simple. Suppose that the label is given by a linear function on $c_{[T]}$ with parameter θ^*, i.e., $y = \langle \theta^*, c_{[T]} \rangle$. When we learn over the weighted features $\Lambda c_{[T]}$, if the support of the diagonal entries $\mathrm{diag}(\Lambda)$ contains the support of θ^* (i.e., if $\theta^*_i \neq 0$ then $\Lambda_{ii} \neq 0$), the parameter $\Lambda^\dagger \theta^*$ on $\Lambda c_{[T]}$ has the same predictions and loss as θ^* on $c_{[T]}$. So we only need to learn $\Lambda^\dagger \theta^*$ on $\Lambda c_{[T]}$, which can require fewer data samples for learning (equivalently, smaller loss for a fixed amount of samples). Learning over $f_{[T]}$ is learning over a compressed version of $\Lambda c_{[T]}$, which shares the same intuition.

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{M} \sum_{i=1}^{M} \ell(\langle \theta, f_{[T]}(G_i) \rangle, y_i) + \lambda \|\theta\|_2. \quad (7)$$

Theorem 2. *Assume that $\mathcal{M}$ satisfies the $(2s, \epsilon)$-RIP, and $c_{[T]}$ is s-sparse. Also assume that the support of $diag(\Lambda)$ contains the support of θ^*. For any $\delta \in (0, 1)$, the $\hat{\theta}$ minimizing the ℓ_2-regularized logistic loss over the embeddings $f_{[T]}(G_i)$'s satisfies with probability at least $1 - \delta$:*

$$\ell_{\mathcal{D}}(\hat{\theta}) \leq \ell_{\mathcal{D}}(\theta^*) + O\left(B_\Lambda \sqrt{\epsilon + \frac{1}{M} \log \frac{1}{\delta}} \right) \quad (8)$$

for an appropriate choice of regularization coefficient λ. Here the term B_Λ is defined as

$$B_\Lambda := \|\Lambda^\dagger \theta^*\|_2 \max_{G \sim \mathcal{D}} \|\Lambda c_{[T]}(G)\|_2 \quad (9)$$

and $\Lambda^\dagger$ is the pseudo-inverse of Λ.

Definition 7 (Walk Statistics for the General Case). *Given a walk $p = (i_1, \ldots, i_n)$ of length n, the vector $\mathbf{e}_p \in \{0,1\}^{(\prod_j k_j)^n}$ is defined as the one-hot vector for the attribute value sequence $(\mathcal{V}_{i_1}, \ldots, \mathcal{V}_{i_n})$ along the walk. The walk statistics vector $c_{(n)}(G) = \sum_p \mathbf{e}_p$ with the sum over all walks p of length n in the graph G. Furthermore, let the walk statistics $c_{[T]}(G)$ be the concatenation of $c_{(1)}(G), \ldots, c_{(T)}(G)$. When G is clear from the context, we write $c_{(n)}$ and $c_{[T]}$ for short.*

So the definition is similar to that for the case with $S = 1$, except that now it is in dimension $K_n = (\prod_{j=0}^{S-1} k_j)^n$.

To describe the linear mapping from $c_{(n)}$ to $f_{(n)}$, we need to introduce the following notation.

Definition 8. *Let $(W_1 W)^{\{n\}}$ be a matrix with K_n column corresponding to all possible length-n sequences of attribute values, with the column indexed by a sequence of attribute values $(v_0, \ldots, v_{n-1})$ being $(W_1 W h(v_0)) \odot (W_1 W h(v_1)) \odot \cdots \odot (W_1 W h(v_{n-1}))$, where $h(v_i) = [h(v_i^0), \ldots, h(v_i^{S-1})]$ and $h(v_i^j)$ is the one-hot vector for the attribute value v_i^j for the j-th attribute in v_i.*

The following theorem then shows that $f_{(n)}$ can be a compressed version of the walk statistics, weighted by the weighting parameter matrix W_1, W_3 and also by the attention scores $\bar{S}$.

Theorem 6. *The embedding $f_{(n)}$ is a linear mapping of the walk statistics $c_{(n)}$:*

$$f_{(n)} = W_3 (W_1 W)^{\{n\}} \Lambda_{(n)} c_{(n)}. \tag{40}$$