Embedding for graph classification

Marc Lelarge Alexis Galland

INRIA-ENS

June 2019

A controversial blog

The Bitter Lesson

Rich Sutton

March 13, 2019

"The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. (..) Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation."
`http://www.incompleteideas.net/IncIdeas/BitterLesson.html`

Inductive Bias

The inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict outputs given inputs that it has not encountered.

Inductive Bias

The inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict outputs given inputs that it has not encountered.

Example 1: in a Bayesian model, inductive biases are typically expressed through the choice and parameterization of the prior distribution.

Inductive Bias

The inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict outputs given inputs that it has not encountered.

Example 1: in a Bayesian model, inductive biases are typically expressed through the choice and parameterization of the prior distribution.

Example 2: the weight-sharing architecture of a convolutional neural network induces an inductive bias of translational invariance. It is a spatial inductive bias because it builds in specific assumptions about the spatial structure of natural images.

More on Inductive Bias

An inductive bias might be a regularization term added to avoir overfitting. Inductive biases can be understood in terms of the bias-variance tradeoff.

More on Inductive Bias

An inductive bias might be a regularization term added to avoir overfitting. Inductive biases can be understood in terms of the bias-variance tradeoff.

Ideally, inductive biases both improve the search for solutions without substantially diminishing performance, as well as help find solutions which generalize in a desirable way.

More on Inductive Bias

An inductive bias might be a regularization term added to avoir overfitting. Inductive biases can be understood in terms of the bias-variance tradeoff.

Ideally, inductive biases both improve the search for solutions without substantially diminishing performance, as well as help find solutions which generalize in a desirable way.

Main problem: understand inductive bias for graphs in order to design principled machine learning algorithms.

What I will not discuss: community detection

Community detection in networks: A user guide

Santo Fortunato*

Center for Complex Networks and Systems Research, School of Informatics and Computing and Indiana University Netwo
Science Institute (IUNI), Indiana University, Bloomington, USA and
Department of Computer Science, Aalto University School of Science, P.O. Box 15400, FI-00076

Darko Hric

Department of Computer Science, Aalto University School of Science, P.O. Box 15400, FI-00076

(Dated: November 4, 2016)

Community detection in networks is one of the most popular topics of modern network science. Communities, or clusters, are usually groups of vertices having higher probability of being connected to each other than to members of other groups, though other patterns are possible. Identifying communities is an ill-defined problem. There are no universal protocols on the fundamental ingredients, like the definition of community itself, nor on other crucial issues, like the validation of algorithms and the comparison of their performances. This has generated a number of confusions and misconceptions, which undermine the progress in the field. We offer a guided tour through the main aspects of the problem. We also point out strengths and weaknesses of popular methods, and give directions to their use.

Machine learning with graphs

Basic question : how to represent a graph $G = (V, E)$ as input to a machine learning algorithm?

Machine learning with graphs

Basic question : how to represent a graph $G = (V, E)$ as input to a machine learning algorithm?

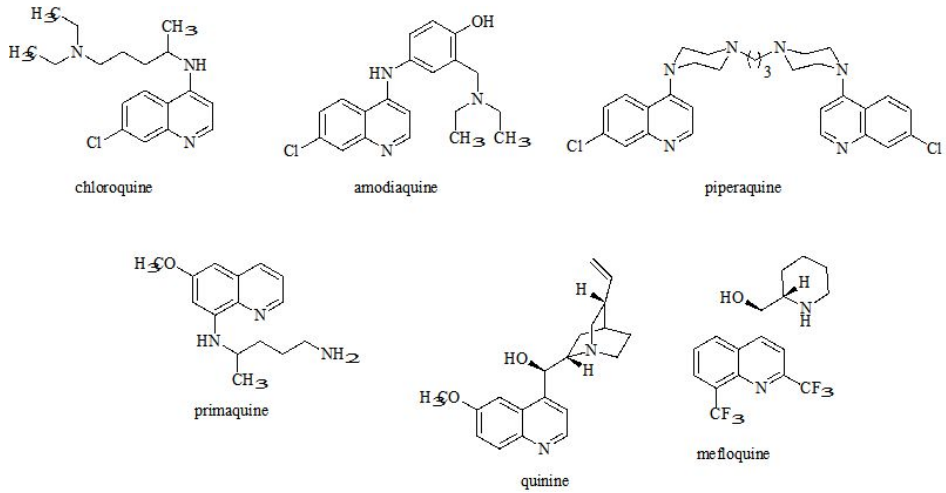
Goal : build a feature vector in $\mathbb{R}^d$ from a graph that is relevant for the learning task.

Machine learning with graphs

Basic question : how to represent a graph $G = (V, E)$ as input to a machine learning algorithm?

Goal : build a feature vector in $\mathbb{R}^d$ from a graph that is relevant for the learning task.

Example : graph classification task.
To make it even more precise, classify active vs. inactive molecules.



What is a good graph representation?

Requirement 1 : for two isomorphic graphs, the representations should be the same.

What is a good graph representation?

Requirement 1 : for two isomorphic graphs, the representations should be the same.

Requirement 2 : for two non-isomorphic graphs, the representations should be distinct.

What is a good graph representation?

Requirement 1 : for two isomorphic graphs, the representations should be the same.

Requirement 2 : for two non-isomorphic graphs, the representations should be distinct.

But then we solve the graph isomorphism problem!
In the worst case, to get fast computable representations, we need to relax our requirements.

What is a good graph representation?

Requirement 1 : for two isomorphic graphs, the representations should be the same.

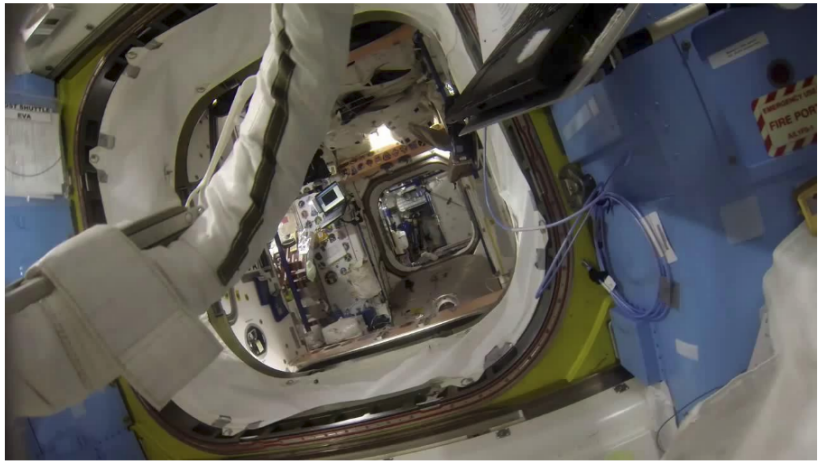
Requirement 2 : for two non-isomorphic graphs, the representations should be distinct.

But then we solve the graph isomorphism problem!
In the worst case, to get fast computable representations, we need to relax our requirements.

We are not dealing with worst case but with datasets. We want to extract features relevant for the classification for a specific dataset.

Graphs are not images!

Result of seeing an image where nodes are pixels and where we replace the grid by the complete graph:



Invariant graph embedding

A graph embedding (or graph feature) is a function $\mathcal{F}$ mapping graphs to vectors in $\mathbb{R}^d$, where d is called the dimension of the embedding.
A graph embedding is invariant if for any two isomorphic graphs G and H, we have $\mathcal{F}(G) = \mathcal{F}(H)$.

Invariant graph embedding

A graph embedding (or graph feature) is a function $\mathcal{F}$ mapping graphs to vectors in $\mathbb{R}^d$, where d is called the dimension of the embedding.
A graph embedding is invariant if for any two isomorphic graphs G and H, we have $\mathcal{F}(G) = \mathcal{F}(H)$.

Let $0 = \lambda_1 \leqslant \lambda_2 \leqslant \ldots \leqslant \lambda_n$ be the eigenvalues of the graph Laplacian $L = D - A$, then $\mathcal{F}(G) = (\lambda_2, \ldots, \lambda_{k+1})$ if $n \geqslant k+1$ and $\mathcal{F}(G) = (0, \ldots, 0, \lambda_2, \ldots, \lambda_n)$ if $n \leqslant k$ is an invariant embedding.

Embedding of nodes

An embedding of nodes is a function $\mathcal{E}$ mapping a graph of size n to a vector $(\mathcal{E}(G)(i))_{i \in V} \in \mathbb{R}^{d \times n}$. For a graph G, $\mathcal{E}(G)(i) \in \mathbb{R}^d$ is the embedding of node i from graph G.
An embedding of nodes is equivariant if for any two isomorphic graphs G and H such that $H = \pi(G)$ for a permutation π, we have $\mathcal{E}(H)(i) = \mathcal{E}(G)(\pi(i))$ for all $i \in V$.

Embedding of nodes

An embedding of nodes is a function $\mathcal{E}$ mapping a graph of size n to a vector $(\mathcal{E}(G)(i))_{i \in V} \in \mathbb{R}^{d \times n}$. For a graph G, $\mathcal{E}(G)(i) \in \mathbb{R}^d$ is the embedding of node i from graph G.
An embedding of nodes is equivariant if for any two isomorphic graphs G and H such that $H = \pi(G)$ for a permutation π, we have $\mathcal{E}(H)(i) = \mathcal{E}(G)(\pi(i))$ for all $i \in V$.

From an equivariant embedding of nodes, it is straightforward to create an invariant graph embedding, we just need to apply a symmetric funtion. A very simple choice is just to sum the nodes' embeddings:

$$\mathcal{F}(G) = \sum_{i \in V} \mathcal{E}(G)(i).$$

Spatial graph embedding

Let $P = D^{-1}A$ be the transition matrix for the random walk on G. Then $1^T P^k = (x^k(1), \ldots, x^k(n))$ is a vector where $x^k(j)/n$ is the probability for a random walk started uniformly at random on the graph to be in position j after k steps and is an equivariant embedding of nodes.

Spatial graph embedding

Let $P = D^{-1}A$ be the transition matrix for the random walk on G. Then $1^T P^k = (x^k(1), \ldots, x^k(n))$ is a vector where $x^k(j)/n$ is the probability for a random walk started uniformly at random on the graph to be in position j after k steps and is an equivariant embedding of nodes.

For a graph of size n, we define

$$f(x_1, \ldots, x_n) = (\ell(x_1, \ldots, x_n; t_i))_{1 \leqslant i \leqslant m},$$

where $t_1 < t_2 < \cdots < t_m$ are fixed parameters and

$$\ell(x_1, \ldots, x_n; t) = \frac{\sum_i x_i e^{t x_i}}{\sum_j e^{t x_j}} \in \mathbf{R}.$$

Other spatial graph embeddings: graph kernels

Graphlets represent graphs as counts of all types of subgraphs of size $k \in \{3, 4, 5\}$

Possible variations by counting only trees, cycles or specific patterns.

Other spatial graph embeddings: graph kernels

Graphlets represent graphs as counts of all types of subgraphs of size $k \in \{3, 4, 5\}$
Possible variations by counting only trees, cycles or specific patterns.

Weisfeiler-Lehman graph kernels extend these graphlets using the naive vertex refinement of Weisfeiler-Lehman Test of isomorphism.

Recall that $L = D - A$ is the Laplacian of the graph and the spectral theorem yields $L = U\Lambda U^T$, where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ is the diagonal matrix of eigenvalues of L and $U = (u_1, \ldots, u_n)$ is the matrix of corresponding eigenvectors, with $U^T U = I$ and $u_1 = 1/\sqrt{n}$.

Spectral graph theory

Recall that $L = D - A$ is the Laplacian of the graph and the spectral theorem yields $L = U\Lambda U^T$, where $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$ is the diagonal matrix of eigenvalues of L and $U = (u_1, \ldots, u_n)$ is the matrix of corresponding eigenvectors, with $U^T U = I$ and $u_1 = 1/\sqrt{n}$.

Let $X = \sqrt{\Lambda^+} U^T$ where $\Lambda^+ = \text{diag}(0, 1/\lambda_2, \ldots, 1/\lambda_n)$ denotes the pseudo-inverse of Λ. The columns $x(1), \ldots, x(n)$ of the matrix X define an embedding of the nodes in $\mathbf{R}^n$, each dimension corresponding to an eigenvector of the Laplacian.

Spectral graph theory

Recall that $L = D - A$ is the Laplacian of the graph and the spectral theorem yields $L = U\Lambda U^T$, where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ is the diagonal matrix of eigenvalues of L and $U = (u_1, \ldots, u_n)$ is the matrix of corresponding eigenvectors, with $U^T U = I$ and $u_1 = 1/\sqrt{n}$.
Let $X = \sqrt{\Lambda^+} U^T$ where $\Lambda^+ = \mathrm{diag}(0, 1/\lambda_2, \ldots, 1/\lambda_n)$ denotes the pseudo-inverse of Λ. The columns $x(1), \ldots, x(n)$ of the matrix X define an embedding of the nodes in $\mathbf{R}^n$, each dimension corresponding to an eigenvector of the Laplacian.
The Gram matrix of X is the pseudo-inverse of the Laplacian:

$$X^T X = U\Lambda^+ U^T = L^+.$$

In words, the graph is completly encoded in the nodes' embedding X, without any loss of information.

Geometric interpretation (1)

Let H_{ij} be the mean hitting time of node j from node i for the continuous-time Markov chain with generator matrix $-L$.
Then we have $H_{ij} = n(x(j) - x(i))^T x(j)$, hence the mean commute time between nodes i and j is:

$$C_{ij} = H_{ij} + H_{ji} = n\|x(i) - x(j)\|^2.$$

Geometric interpretation (1)

Let H_{ij} be the mean hitting time of node j from node i for the continuous-time Markov chain with generator matrix $-L$. Then we have $H_{ij} = n(x(j) - x(i))^T x(j)$, hence the mean commute time between nodes i and j is:

$$C_{ij} = H_{ij} + H_{ji} = n\|x(i) - x(j)\|^2.$$

Proposition

We can reconstruct the adjacency matrix form the matrix of commute times.

Geometric interpretation (2)

Also all the information about the graph is contained in the relative geometry of the $x(i)$'s, their intrinsic values also carries relevant information. Here, we state Fiedler's nodal domain theorem:

Theorem (Fiedler (1975))

for any $k \geqslant 2$, let $W_k = \{i \in V, x_k(i) \geqslant 0\}$. Then the graph induced by G on W_k has at most $k-1$ connected components.

This implies that for low values of k, nodes i with non-negative entries $x_k(i)$ tends to be well-connected.

Spectral graph embedding (1)

We can apply the symmetric function $\ell(,;t)$ to the vector of $x(i)$'s componentwise, however this will not produce an invariant graph embedding.

Problem : as soon as an eigenvalue has multiplicity larger than 2, the associated eigenvectors are defined up to a rotation. Even if all eigenvalues have multiplicity one, the eigenvector is only defined up to a sign.

Spectral graph embedding (1)

We can apply the symmetric function $\ell(,;t)$ to the vector of $x(i)$'s componentwise, however this will not produce an invariant graph embedding.
Problem : as soon as an eigenvalue has multiplicity larger than 2, the associated eigenvectors are defined up to a rotation. Even if all eigenvalues have multiplicity one, the eigenvector is only defined up to a sign.

Solution for graphs with all eigenvalues with multiplicity 1: take an even function.
However, we loose the notion of Fiedler's nodal domain.

Spectral graph embedding (2)

To incorporate the information contained in the commute times, we compute the dot products $x(i)^T x(j)$ for $1 \leqslant i, j \leqslant n$. Then, we 'flatten' this matrix to obtain a vector of size n^2 and pass this vector through our symmetric function $\ell(.; t)$.
This is clearly an invariant embedding of the graph.

Spectral graph embedding (2)

To incorporate the information contained in the commute times, we compute the dot products $x(i)^T x(j)$ for $1 \leqslant i, j \leqslant n$. Then, we 'flatten' this matrix to obtain a vector of size n^2 and pass this vector through our symmetric function $\ell(.; t)$.
This is clearly an invariant embedding of the graph.

Even if we can reconstruct the original graph from the matrix $(x(i)^T x(k))_{1 \leqslant i,k \leqslant n}$, our embedding will not allow to reconstruct the graph because it only characterizes the multi-set $\{x(i)^T x(k), 1 \leqslant i, k \leqslant n\}$ which does not a priori characterize the graph up to isomorphisms.

Spectral graph embedding (2)

To incorporate the information contained in the commute times, we compute the dot products $x(i)^T x(j)$ for $1 \leqslant i, j \leqslant n$. Then, we 'flatten' this matrix to obtain a vector of size n^2 and pass this vector through our symmetric function $\ell(.; t)$.
This is clearly an invariant embedding of the graph.

Even if we can reconstruct the original graph from the matrix $(x(i)^T x(k))_{1 \leqslant i,k \leqslant n}$, our embedding will not allow to reconstruct the graph because it only characterizes the multi-set $\{x(i)^T x(k), 1 \leqslant i, k \leqslant n\}$ which does not a priori characterize the graph up to isomorphisms.
Question : what information on the graph did we loose by considering only the statistics of the commute times?

Results

Classification accuracy for some bioinformatic network datasets.

Dataset	MUTAG	PTC	PROTEINS	NCI1	NCI109	D&D
Max	28	109	620	111	111	5748
Avg	17.93	25.56	39.06	29.87	29.68	284.32
#Graphs	188	344	1113	4110	4127	1178
DCNN	66.51	55.79	65.22	63.10	60.67	-
DGK	86.17	59.88	71.69	64.40	67.14	72.95
FSGD	**92.12**	62.80	73.42	79.80	78.84	77.10
IGE	90.01 $\pm$ 6.6	**64.54 $\pm$ 5.8**	**74.21 $\pm$ 3.1**	**81.46 $\pm$ 0.9**	**79.36 $\pm$ 1.0**	**78.86 $\pm$ 3.2**

Table 1: Classification accuracy on bioinformatics datasets

Results

Classification accuracy for some social network datasets.

Dataset	IMDB-B	IMDB-M	REDDIT-B	REDDIT-M	COLLAB
Avg	19.77	13.00	429.63	508.52	74.49
#Graphs	1000	1500	2000	4999	5000
DGK	66.96	44.55	78.04	41.27	73.09
FSGD	**73.62**	**52.41**	86.50	47.76	**80.02**
IGE	**74.10** $\pm$ **3.2**	48.87 $\pm$ 2.1	**89.12** $\pm$ **2.2**	**50.92** $\pm$ **2.4**	78.95 $\pm$ 2.0

Table 2: Classification accuracy on social datasets

Towards learnable embeddings

In order to learn our graph embedding, we assume that we start from an embedding of the nodes $(\mathcal{E}(G)(i))_{i\in V}$ and need to learn the symmetric function f able to create the best graph embedding $\mathcal{F}(G) = f(\mathcal{E}(G)(i)_{i\in V})$ for the classification task on the given dataset.

Towards learnable embeddings

In order to learn our graph embedding, we assume that we start from an embedding of the nodes $(\mathcal{E}(G)(i))_{i\in V}$ and need to learn the symmetric function f able to create the best graph embedding $\mathcal{F}(G) = f(\mathcal{E}(G)(i)_{i\in V})$ for the classification task on the given dataset.

Theorem
Let $f : [0,1]^{k\times n} \to \mathbf{R}^m$ *be a smooth symmetric function: for all* $x, y \in [0,1]^{k\times n}$ *such that* $\inf_\sigma \sup_i |x_{\sigma(i)} - y_i| \leqslant \delta$, *we have* $|f(x) - f(y)| \leqslant \epsilon$. *In this case, for* $K = \lceil \frac{3}{\delta} \rceil^k$, *there exists a function* $g : \mathbf{R}^K \to \mathbf{R}$ *and a function* $h : \mathbf{R}^k \to \mathbf{R}^K$ *such that for all* $x_i \in [0,1]^k$,

$$\left| f(x_1, \ldots, x_n) - g\left(\sum_{i=1}^{n} h(x_i)\right)\right| \leqslant \epsilon.$$

GNNs use the graph structure and node features X_v to learn a representation vector of a node h_v by following a neighborhood aggregation strategy: we iteratively update the representation of a node by aggregating representations of its neighbors.
Formally for the k-th layer, we get:

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right), \quad h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right),$$

What can you take for the aggregate and combine functions?

Learning node embedding with Graph Neural Networks

GNNs use the graph structure and node features X_v to learn a representation vector of a node h_v by following a neighborhood aggregation strategy: we iteratively update the representation of a node by aggregating representations of its neighbors.
Formally for the k-th layer, we get:

$$a_v^{(k)} = \text{AGGREGATE}^{(k)}\left(\left\{h_u^{(k-1)} : u \in \mathcal{N}(v)\right\}\right), \quad h_v^{(k)} = \text{COMBINE}^{(k)}\left(h_v^{(k-1)}, a_v^{(k)}\right),$$

What can you take for the aggregate and combine functions?

$$h_v^{(k)} = \text{MLP}^{(k)}\left(\left(1+\epsilon^{(k)}\right)\cdot h_v^{(k-1)} + \sum\nolimits_{u\in\mathcal{N}(v)} h_u^{(k-1)}\right).$$

Graph Invariant Network: GIN (ICLR 2019).

End-to-end learning with Graph Invariant Network

Published as a conference paper at ICLR 2019

How Powerful are Graph Neural Networks?

Keyulu Xu *†
MIT
keyulu@mit.edu

Weihua Hu *‡
Stanford University
weihuahu@stanford.edu

Jure Leskovec
Stanford University
jure@cs.stanford.edu

Stefanie Jegelka
MIT
stefje@mit.edu

	Datasets	IMDB-B	IMDB-M	RDT-B	RDT-M5K	COLLAB	MUTAG	PROTEINS	PTC	NCI1
Datasets	# graphs	1000	1500	2000	5000	5000	188	1113	344	4110
	# classes	2	3	2	5	3	2	2	2	2
	Avg # nodes	19.8	13.0	429.6	508.5	74.5	17.9	39.1	25.5	29.8
Baselines	WL subtree	73.8 ± 3.9	50.9 ± 3.8	81.0 ± 3.1	52.5 ± 2.1	78.9 ± 1.9	90.4 ± 5.7	75.0 ± 3.1	59.9 ± 4.3	**86.0 ± 1.8** *
	DCNN	49.1	33.5	–	–	52.1	67.0	61.3	56.6	62.6
	PATCHYSAN	71.0 ± 2.2	45.2 ± 2.8	86.3 ± 1.6	49.1 ± 0.7	72.6 ± 2.2	**92.6 ± 4.2** *	75.9 ± 2.8	60.0 ± 4.8	78.6 ± 1.9
	DGCNN	70.0	47.8	–	–	73.7	85.8	75.5	58.6	74.4
	AWL	74.5 ± 5.9	51.5 ± 3.6	87.9 ± 2.5	54.7 ± 2.9	73.9 ± 1.9	87.9 ± 9.8	–	–	–
GNN variants	SUM–MLP (**GIN-0**)	**75.1 ± 5.1**	**52.3 ± 2.8**	**92.4 ± 2.5**	**57.5 ± 1.5**	**80.2 ± 1.9**	**89.4 ± 5.6**	**76.2 ± 2.8**	**64.6 ± 7.0**	**82.7 ± 1.7**
	SUM–MLP (**GIN-ϵ**)	**74.3 ± 5.1**	**52.1 ± 3.6**	**92.2 ± 2.3**	**57.0 ± 1.7**	**80.1 ± 1.9**	**89.0 ± 6.0**	**75.9 ± 3.8**	63.7 ± 8.2	**82.7 ± 1.6**
	SUM–1-LAYER	74.1 ± 5.0	**52.2 ± 2.4**	90.0 ± 2.7	55.1 ± 1.6	**80.6 ± 1.9**	**90.0 ± 8.8**	**76.2 ± 2.6**	63.1 ± 5.7	82.0 ± 1.5
	MEAN–MLP	73.7 ± 3.7	**52.3 ± 3.1**	50.0 ± 0.0	20.0 ± 0.0	79.2 ± 2.3	83.5 ± 6.3	75.5 ± 3.4	**66.6 ± 6.9**	80.9 ± 1.8
	MEAN–1-LAYER (GCN)	74.0 ± 3.4	51.9 ± 3.8	50.0 ± 0.0	20.0 ± 0.0	79.0 ± 1.8	85.6 ± 5.8	76.0 ± 3.2	64.2 ± 4.3	80.2 ± 2.0
	MAX–MLP	73.2 ± 5.8	51.1 ± 3.6	–	–	–	84.0 ± 6.1	76.0 ± 3.2	64.6 ± 10.2	77.8 ± 1.3
	MAX–1-LAYER (GraphSAGE)	72.3 ± 5.3	50.9 ± 2.2	–	–	–	85.1 ± 7.6	75.9 ± 3.2	63.9 ± 7.7	77.7 ± 1.5

How ~~powerful~~ useful are Graph Neural Networks?

How ~~powerful~~ useful are Graph Neural Networks?

Answer: as useful as the dataset they are trained on!

How ~~powerful~~ useful are Graph Neural Networks?

Answer: as useful as the dataset they are trained on!

2. How Many Different Proteins Are Necessary to Support Human Function? Go to:

The number of different proteins comprising the human proteome is a core proteomics issue. Researchers propose numbers between 10,000 [10] and several billion [6] different protein species. Here,

How ~~powerful~~ useful are Graph Neural Networks?

Answer: as useful as the dataset they are trained on!

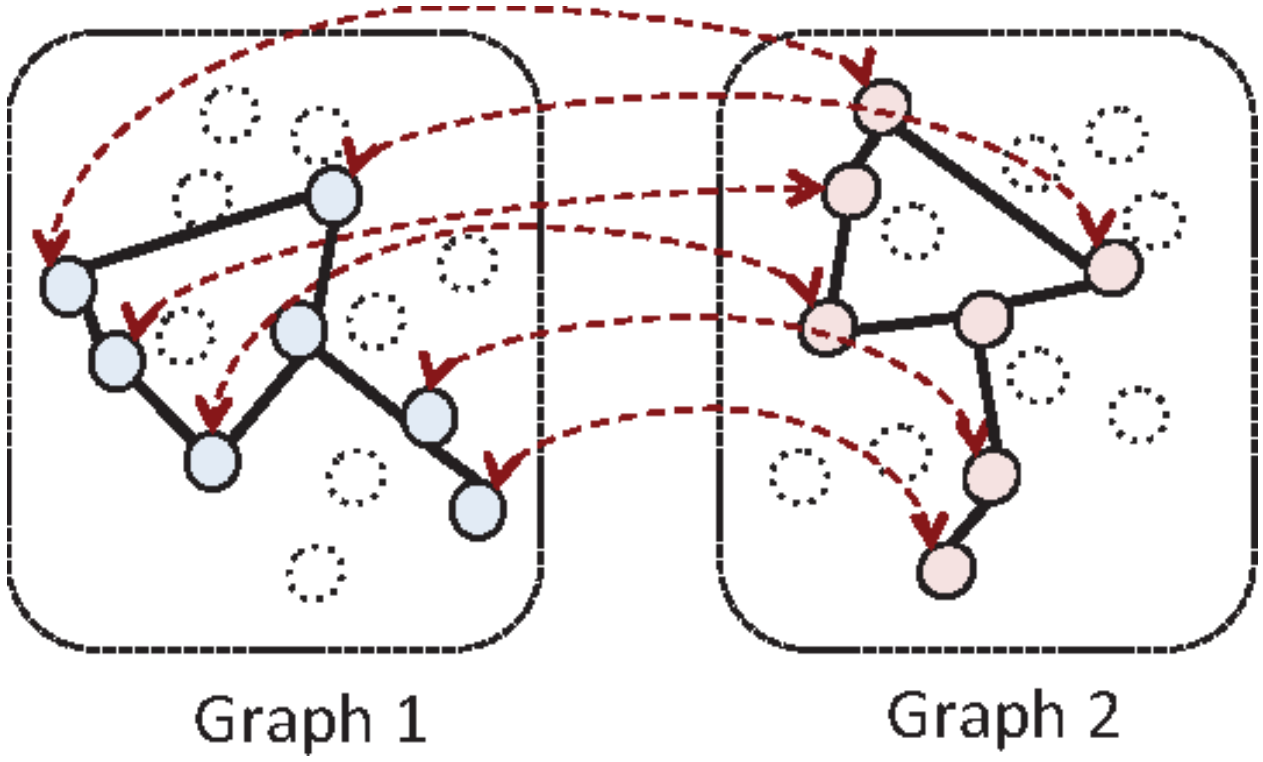
2. How Many Different Proteins Are Necessary to Support Human Function? Go to:

The number of different proteins comprising the human proteome is a core proteomics issue. Researchers propose numbers between 10,000 [10] and several billion [6] different protein species. Here,

3. How Many Protein Species Are Detectable Today? Go to:

According to the Plasma Proteome Database (ver. 06_2015) [19], 10.5 thousand blood-plasma proteins have been detected and less than 10% (1278 of 20,043 human proteins) have been measured in a quantitative manner. The primary issue concerning experimental validation of existing sets of theoretically predicted proteins is the limit of analytical sensitivity of proteomic technology. Analytical sensitivity is determined by instrument-dependent detection limit and biomaterial-dependent dynamic protein concentration ranges.

Graph alignement problem:



Average case analysis: inductive bias is captured by the probabilistic model!

A new perspective on the GAP

Main idea: create your own dataset!
From a set of graphs, you can generate as many permuted graphs as you want like in self-supervision.

A new perspective on the GAP

Main idea: create your own dataset!
From a set of graphs, you can generate as many permuted graphs as you want like in self-supervision.

Hope: so far, algorithms rely on heuristics, here a ML approach could find better heuristics working on a given dataset.

Many variants to explore: use ideas from Adversarial Networks or Reinforcement Learning for better exploration of the space of permutations.

Conclusion

What you need to keep in mind when dealing with graphs!

There are a lot of cases where we have access to a labeled dataset covering a small fraction of the domain and in these cases, leveraging the human knowledge of the domain seems the only sensible thing to do. Black-box approaches are useless and interpretable features need to be provided to the experts.

Conclusion

What you need to keep in mind when dealing with graphs!

There are a lot of cases where we have access to a labeled dataset covering a small fraction of the domain and in these cases, leveraging the human knowledge of the domain seems the only sensible thing to do. Black-box approaches are useless and interpretable features need to be provided to the experts.

There are a lot of cases where we have access to a large unlabeled dataset. In such unsupervised settings, the best way to help the experts is to concentrate on hard computational tasks and try to leverage computation in the most effective way given the dataset...

Conclusion

What you need to keep in mind when dealing with graphs!

There are a lot of cases where we have access to a labeled dataset covering a small fraction of the domain and in these cases, leveraging the human knowledge of the domain seems the only sensible thing to do. Black-box approaches are useless and interpretable features need to be provided to the experts.

There are a lot of cases where we have access to a large unlabeled dataset. In such unsupervised settings, the best way to help the experts is to concentrate on hard computational tasks and try to leverage computation in the most effective way given the dataset...

Playing the race for the best accuracy on noisy datasets is useless but... probably a good strategy to get published!

Thank you for your attention !