Supervized learning on graphs with deep learning

Michaël Defferrard

Slides by Jean-Baptiste Cordonnier, Andreas Loukas.

Plan

Structure is key to learning

How do Convolutional Neural Networks exploit grid structures?

Learning from graph-structured data

From classical CNNs to graph CNNs
- Convolution on Graphs
- Graph coarsening

Applications and other topics

Why does deep learning work?

Why does deep learning work?

- Availability of data and hardware
- Openness and goal-oriented mentality
- Successes in non-convex optimization

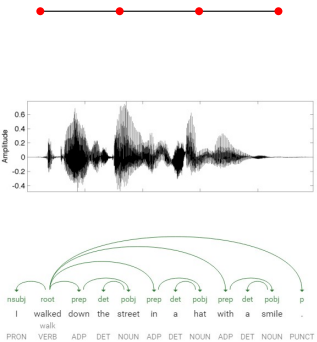
Structure is key to learning

"In general, as the number of dimensions grows, the number of samples we need (to attain statistical significance) increases exponentially."

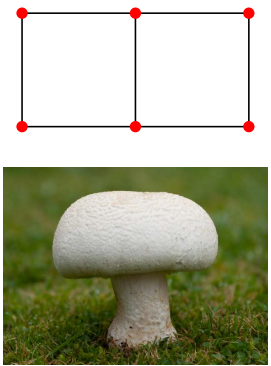
Exploit the structure to deal with this curse of dimensionality.

How is data structured?

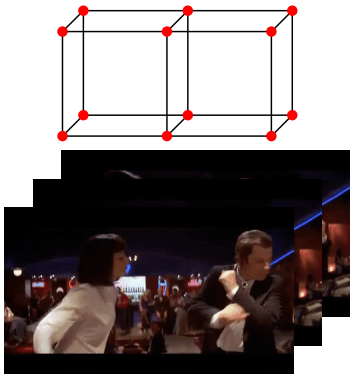
Sequences



Images

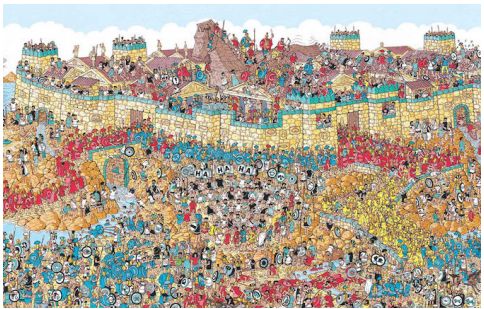


Videos



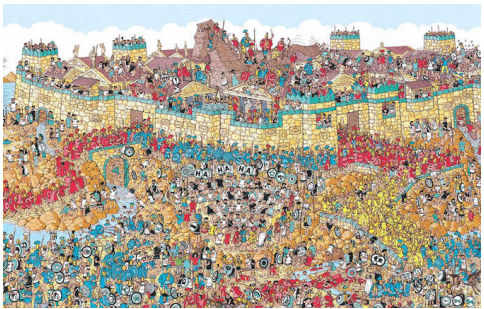
The importance of structure: a toy example

A toy image classification problem: *does the image contain Waldo?*



The importance of structure: a toy example

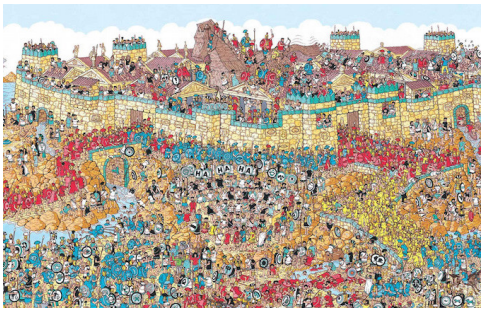
A toy image classification problem: *does the image contain Waldo?*



- An MLP needs $\mathcal{O}(e^{\#\text{ pixels}})$ samples,

The importance of structure: a toy example

A toy image classification problem: *does the image contain Waldo?*



- An MLP needs $\mathcal{O}(e^{\#\text{ pixels}})$ samples,
- however, the problem becomes easy once we use the pixel-grid (single layer CNN).

Plan

Structure is key to learning

How do Convolutional Neural Networks exploit grid structures?

Learning from graph-structured data

From classical CNNs to graph CNNs
- Convolution on Graphs
- Graph coarsening

Applications and other topics

Image invariants

Which image invariants are exploited by CNNs to decrease the number of parameters?

Image invariants

Which image invariants are exploited by CNNs to decrease the number of parameters?

Translation invariance

Composability

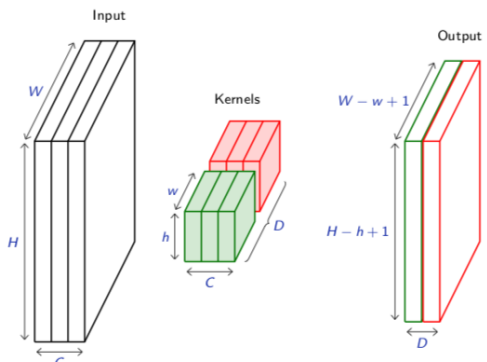
How do CNNs work?

Image from EE559, F. Fleuret

CNNs exploit image properties

- **translation:** a face on left/right of the image is still a face,

Learn a bank of filters/kernels, apply them to a part of the image (point wise multiplication and sum), translate to identify patterns everywhere (stripes, circle, eye, ...).



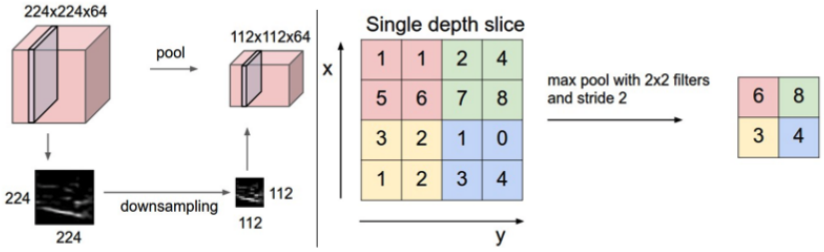
How do CNNs work?

Image from CS231 Stanford

CNN exploit image properties

- **composability:** 2 eyes, a nose and a mouth make a face.

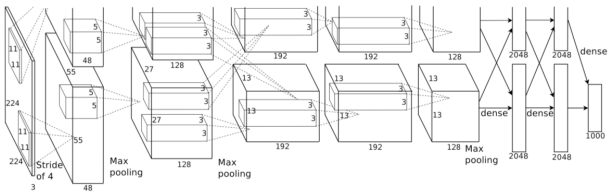
Aggregate local filters by pooling meaningful pixels to identify bigger patterns.



How do CNNs work?

Image from (Krizhevsky et al., 2012)

By repeating convolution with parametrized bank of filters and pooling, we decrease the dimension of the image and encode its information into the channels (depth).



- Number of parameters independent from N (due to abundant weight-sharing).
- Apply a fully connected layer on the output to produce a class label.
- Use miss-classification loss, backpropagation, and SGD to train them.

Plan

Structure is key to learning

How do Convolutional Neural Networks exploit grid structures?

Learning from graph-structured data

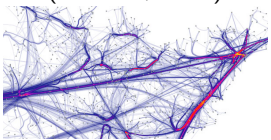
From classical CNNs to graph CNNs
- Convolution on Graphs
- Graph coarsening

Applications and other topics

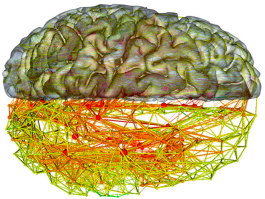
Beyond the grid: graph-structured data

Images by Danny Holten, Martijn can den Heuvel, Spruston Lab

Transporation networks
(Li et al., 2018)

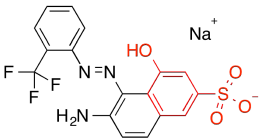


Connectome
(Parisot et al., 2017)

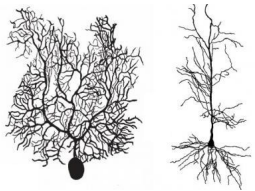


Wikipedia network
(Yin et al., 2017)

Molecules
(Kearnes et al., 2016)



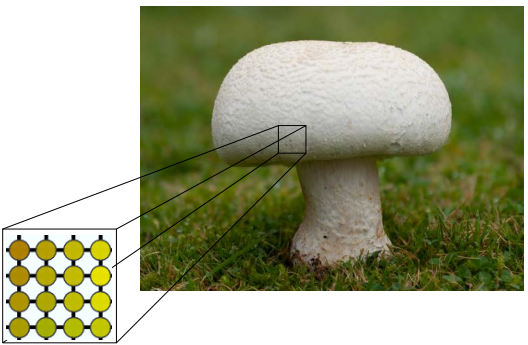
Neural cells



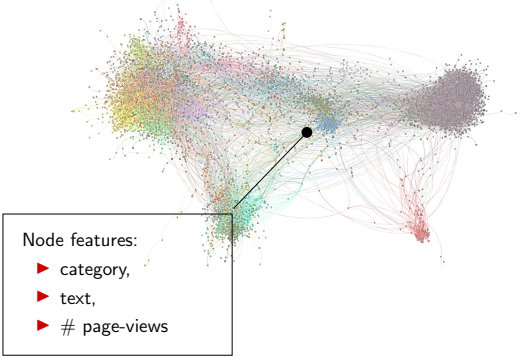
Emerging research question

How can we teach machines to learn from graph-structured data?

90's - to-date



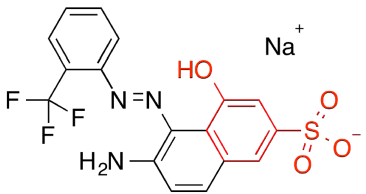
2015 - to-date



Wikipedia network (Yin et al., 2017)

Two types of problems

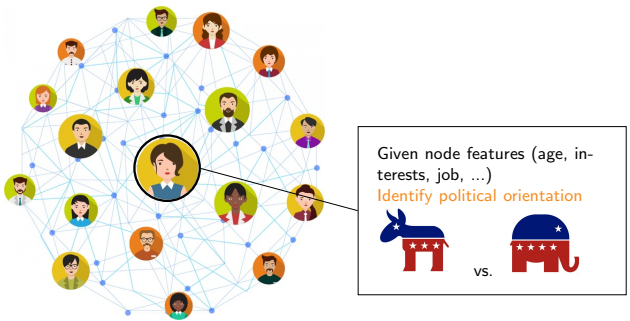
Global problems



- solubility
- drug efficacy
- photo-voltaic efficiency

(Kearnes et al., 2016)

Local problems



(Kipf & Welling, 2016a)

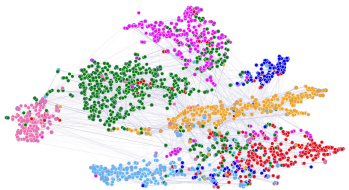
Citation networks analysis

local problem

Infer the scientific community of a set of articles given

- some features for each article (word counts, university, journal...),
- a graph of citations (two articles are connected if one cite the other).

Semi-supervised learning problem: during training we are given the labels of only a subset of the articles. But we can use all unlabeled articles (even from the test set).



Cora dataset colored by inferred class (Velickovic et al., 2017).

Citation networks analysis

local problem

Input:

- Features for each node and a graph of relations between nodes/labels.

Output:

- probability that each node is in a given class (softmax).

Learning:

- *Train* by binary cross entropy loss on the training set; *evaluate* accuracy on test set.

Citation networks analysis

local problem

Input:

- Features for each node and a graph of relations between nodes/labels.

Output:

- probability that each node is in a given class (softmax).

Learning:

- *Train* by binary cross entropy loss on the training set; *evaluate* accuracy on test set.

What is the difference with the problem solved by TV regularization, i.e., $\min_x \|Ax - y\|_2^2 + \alpha \|Sx\|_1$?

Citation networks analysis

local problem

Input:

- Features for each node and a graph of relations between nodes/labels.

Output:

- probability that each node is in a given class (softmax).

Learning:

- *Train* by binary cross entropy loss on the training set; *evaluate* accuracy on test set.

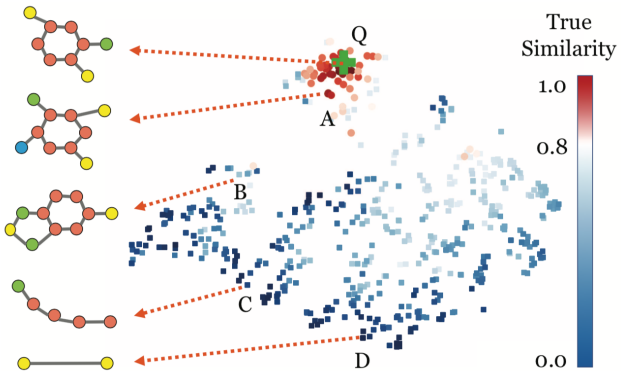
What is the difference with the problem solved by TV regularization, i.e., $\min_x \|Ax - y\|_2^2 + \alpha \|Sx\|_1$?

- Inverse problems try to understand label relationship.
- Here, we search for a function from features to labels, that takes into account neighboring features.

Find similar graphs

global problem

Application: Find in a database the closest chemical compounds to a new molecule Q (Bai et al., 2018).



Find similar graphs

global problem

Comparison between graphs is a hard problem:

- two graphs can have very different size,
- no correspondence between nodes and we can't check all permutations

Find similar graphs

global problem

Comparison between graphs is a hard problem:

- two graphs can have very different size,
- no correspondence between nodes and we can't check all permutations

"Hard" definitions of graph distance:

- isomorphism-based (hard problem, belongs in NP but is not NP-complete)
- Graph edit distance (GED): the distance between two graphs is the number of operation (add/remove edge/node) to transform one into the other. (NP-hard)

Find similar graphs

global problem

Comparison between graphs is a hard problem:

- two graphs can have very different size,
- no correspondence between nodes and we can't check all permutations

"Hard" definitions of graph distance:

- isomorphism-based (hard problem, belongs in NP but is not NP-complete)
- Graph edit distance (GED): the distance between two graphs is the number of operation (add/remove edge/node) to transform one into the other. (NP-hard)

Machine learning point of view:

> No single distance measure suffices. It depends on what you want to do.

Neural networks learn the distance function from examples.

Find similar graphs

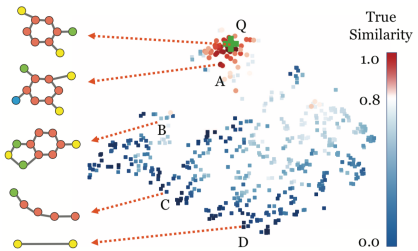
global problem

Input: two graphs,
Output: similarity measure in $[0, 1]$
Loss: MSE of output with "true" similarity

What does "true" mean?

- can be either user defined (chemists)
- output of another algorithm (GED)
- ground-truth labels—distance should allow me to better distinguish between graphs of different classes

Evaluation: compare ranking with output (Spearman's Rank Correlation Coefficient, or top k match).



Plan

Structure is key to learning

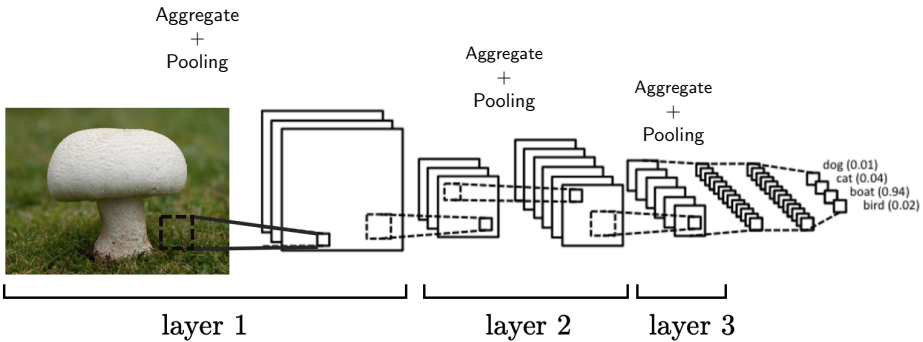
How do Convolutional Neural Networks exploit grid structures?

Learning from graph-structured data

From classical CNNs to graph CNNs
- Convolution on Graphs
- Graph coarsening

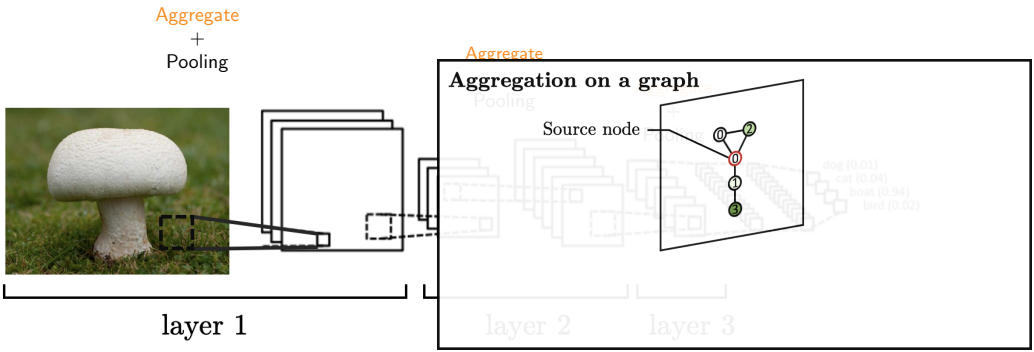
Applications and other topics

From CNN to GCNN



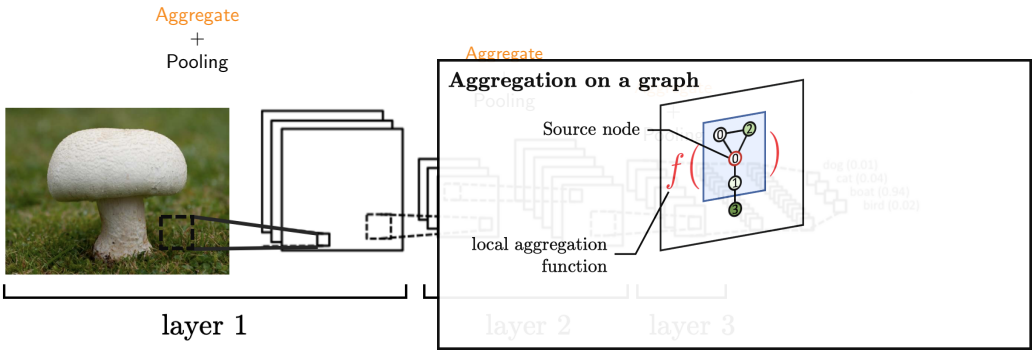
CNNs hierarchically aggregate and pool images along pixel-grid.

From CNN to GCNN



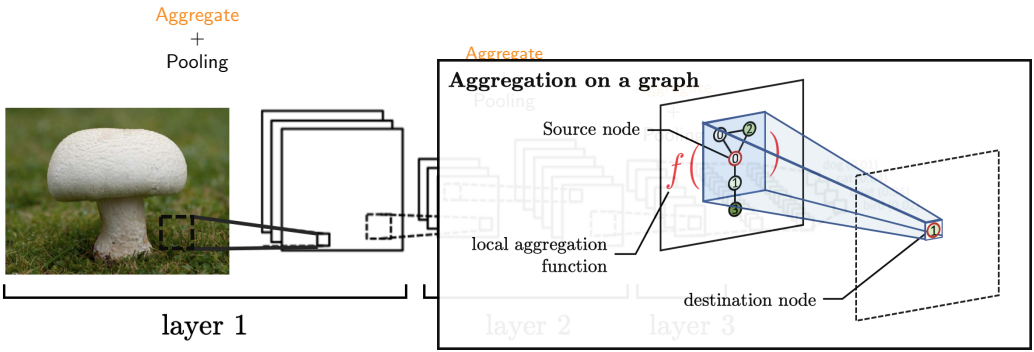
CNNs hierarchically aggregate and pool images along pixel-grid.

From CNN to GCNN



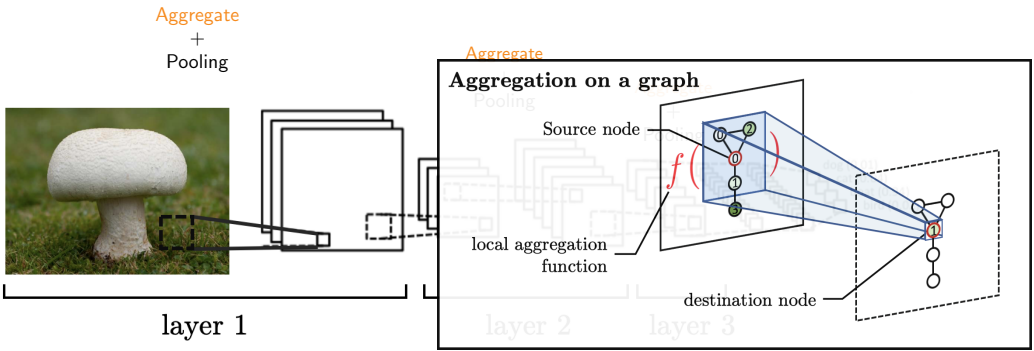
CNNs hierarchically aggregate and pool images along pixel-grid.

From CNN to GCNN



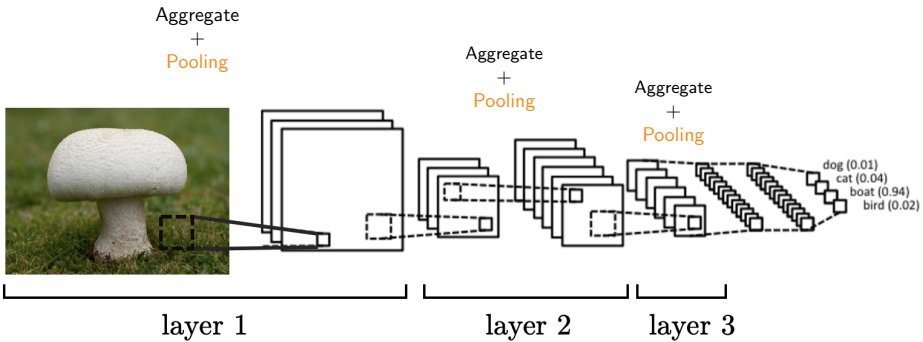
CNNs hierarchically aggregate and pool images along pixel-grid.

From CNN to GCNN



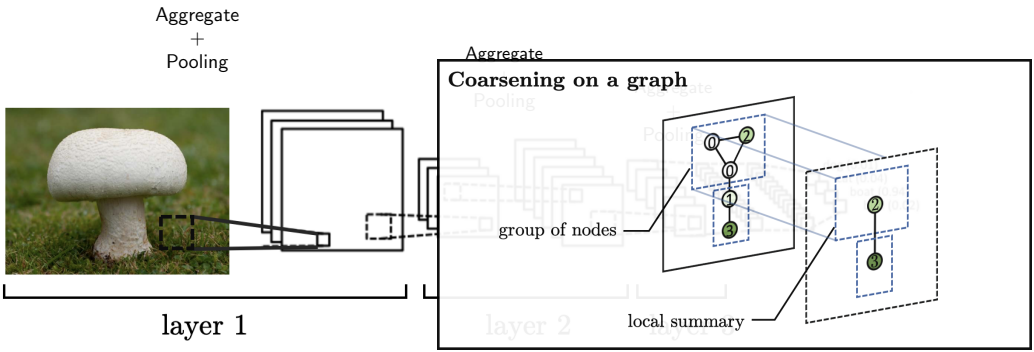
CNNs hierarchically aggregate and pool images along pixel-grid.

From CNN to GCNN



CNNs hierarchically aggregate and pool images along pixel-grid.

From CNN to GCNN



CNNs hierarchically aggregate and pool images along pixel-grid.

Plan

Structure is key to learning

How do Convolutional Neural Networks exploit grid structures?

Learning from graph-structured data

From classical CNNs to graph CNNs
- Convolution on Graphs
- Graph coarsening

Applications and other topics

Graph (or generalized) convolution

Key idea:

Convolution in vertex domain $\Leftrightarrow$ multiplication in graph frequency domain.

Graph (or generalized) convolution

Key idea:

> Convolution in vertex domain $\Leftrightarrow$ multiplication in graph frequency domain.

Let $L = U\Lambda U^\top$ be the Laplacian. Define (generalized) convolution between graph signals h and x as

$$h *_{\mathcal{G}} x = U\left((U^\top h) \circ (U^\top x)\right) = U(\hat{h} \circ \hat{x}).$$

Graph (or generalized) convolution

Key idea:

> Convolution in vertex domain $\Leftrightarrow$ multiplication in graph frequency domain.

Let $L = U\Lambda U^\top$ be the Laplacian. Define (generalized) convolution between graph signals h and x as

$$h *_{\mathcal{G}} x = U\left((U^\top h) \circ (U^\top x)\right) = U(\hat{h} \circ \hat{x}).$$

This is equivalent to filtering by graph filter $h(L) = U\mathsf{diag}(\hat{h})U^\top$ with frequency response $\hat{h}$:

$$h(L)x = U\mathsf{diag}(\hat{h})U^\top x = U\left(\hat{h} \circ \hat{x}\right) = h *_{\mathcal{G}} x$$

Graph (or generalized) convolution

Key idea:

> Convolution in vertex domain $\Leftrightarrow$ multiplication in graph frequency domain.

Let $L = U\Lambda U^\top$ be the Laplacian. Define (generalized) convolution between graph signals h and x as

$$h *_{\mathcal{G}} x = U\left((U^\top h) \circ (U^\top x)\right) = U(\hat{h} \circ \hat{x}).$$

This is equivalent to filtering by graph filter $h(L) = U\text{diag}(\hat{h})U^\top$ with frequency response $\hat{h}$:

$$h(L)x = U\text{diag}(\hat{h})U^\top x = U\left(\hat{h} \circ \hat{x}\right) = h *_{\mathcal{G}} x$$

> Graph convolution $\Leftrightarrow$ graph filtering

Convolution without translation?

1D Euclidean convolution:

$$(x * g)(i) = \sum_{j=-\infty}^{\infty} x(j)g(i-j) = \langle T_i g, x \rangle,$$

where $T_i g$ is a **translation** of the signal g by i steps.

Graph convolution:

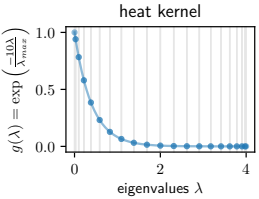
$$(x *_{\mathcal{G}} g)(i) = (g(L)x)(i) = \langle \mathcal{T}_i g(L), x \rangle = \langle g(L)\delta_i, x \rangle,$$

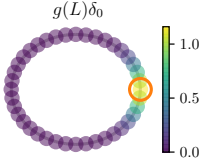
where $\mathcal{T}_i g$ is the **localization** of the kernel g at node v_i.

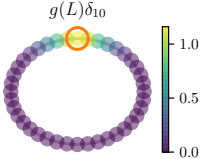
We filter x with a kernel g. We cannot convolve x with another signal!

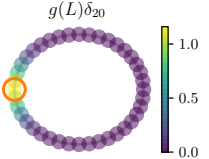
Example: localization vs translation

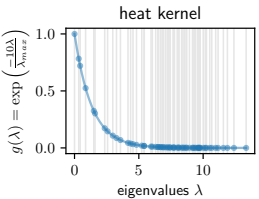
$$\mathcal{T}_i g(L) = g(L)\delta_i$$

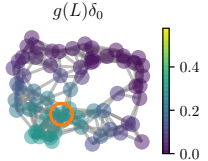


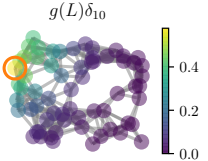


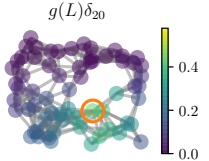








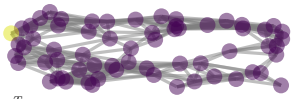




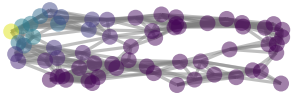
Example: vertex domain kernel visualization

$$\mathcal{T}_i g(L) = g(L)\delta_i$$

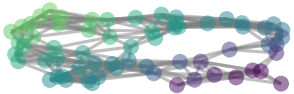
localized $y = g(L)\delta_{10}$ (sensor)



$y^T L y = 367.18$

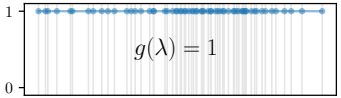


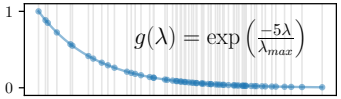
$y^T L y = 6.83$

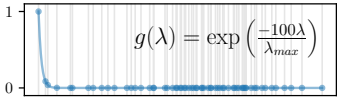


$y^T L y = 0.00$

kernel $g(\lambda)$ defined in the spectral domain

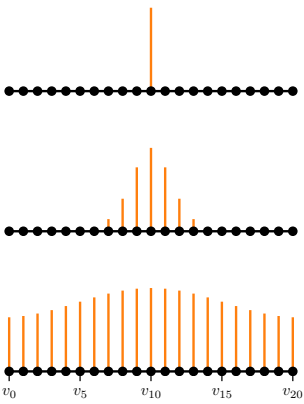






λ: laplacian's eigenvalues / graph frequencies

localized $y = g(L)\delta_{10}$ (path graph)



Graph convolution in the spectral domain

"Spectral Networks and Locally Connected Networks on Graphs" (Bruna et al., 2013)

Given L the normalized Laplacian and the input $X_{l-1} \in \mathbb{R}^{N \times C_{l-1}}$ of layer l, a convolutional layer of C_l channels returns

$$X_l(:, c') = \sigma\left(\sum_{c=1}^{C_{l-1}} h_{l,c,c'}(L)\, X_{l-1}(:, c)\right) \in \mathbb{R}^N \quad \text{for} \quad c' = 1, \ldots, C_l,$$

where $h_{l,c,c'}$ are filters and σ is an element-wise non linearity, such as the ReLU function $[\sigma(x)](i) = \max(0, x_i)$ or the sigmoid function $[\sigma(x)](i) = 1/(1 + e^{-x(i)})$.

Limitations:

- Parameters:

Graph convolution in the spectral domain

"Spectral Networks and Locally Connected Networks on Graphs" (Bruna et al., 2013)

Given L the normalized Laplacian and the input $X_{l-1} \in \mathbb{R}^{N \times C_{l-1}}$ of layer l, a convolutional layer of C_l channels returns

$$X_l(:, c') = \sigma\left(\sum_{c=1}^{C_{l-1}} h_{l,c,c'}(L)\, X_{l-1}(:, c)\right) \in \mathbb{R}^N \quad \text{for} \quad c' = 1, \ldots, C_l,$$

where $h_{l,c,c'}$ are filters and σ is an element-wise non linearity, such as the ReLU function $[\sigma(x)](i) = \max(0, x_i)$ or the sigmoid function $[\sigma(x)](i) = 1/(1 + e^{-x(i)})$.

Limitations:

- Parameters: $\mathcal{O}(N)$ (size of the graph), can be cut at smaller K,
- Complexity:

Graph convolution in the spectral domain

"Spectral Networks and Locally Connected Networks on Graphs" (Bruna et al., 2013)

Given L the normalized Laplacian and the input $X_{l-1} \in \mathbb{R}^{N \times C_{l-1}}$ of layer l, a convolutional layer of C_l channels returns

$$X_l(:, c') = \sigma \left(\sum_{c=1}^{C_{l-1}} h_{l,c,c'}(L)\, X_{l-1}(:, c) \right) \in \mathbb{R}^N \quad \text{for} \quad c' = 1, \ldots, C_l,$$

where $h_{l,c,c'}$ are filters and σ is an element-wise non linearity, such as the ReLU function $[\sigma(x)](i) = \max(0, x_i)$ or the sigmoid function $[\sigma(x)](i) = 1/(1 + e^{-x(i)})$.

Limitations:

- Parameters: $\mathcal{O}(N)$ (size of the graph), can be cut at smaller K,
- Complexity: Computing the basis U is $\mathcal{O}(N^3)$.
- Domain adaptation:

Graph convolution in the spectral domain

"Spectral Networks and Locally Connected Networks on Graphs" (Bruna et al., 2013)

Given L the normalized Laplacian and the input $X_{l-1} \in \mathbb{R}^{N \times C_{l-1}}$ of layer l, a convolutional layer of C_l channels returns

$$X_l(:, c') = \sigma \left(\sum_{c=1}^{C_{l-1}} h_{l,c,c'}(L)\, X_{l-1}(:, c) \right) \in \mathbb{R}^N \quad \text{for} \quad c' = 1, \ldots, C_l,$$

where $h_{l,c,c'}$ are filters and σ is an element-wise non linearity, such as the ReLU function $[\sigma(x)](i) = \max(0, x_i)$ or the sigmoid function $[\sigma(x)](i) = 1/(1 + e^{-x(i)})$.

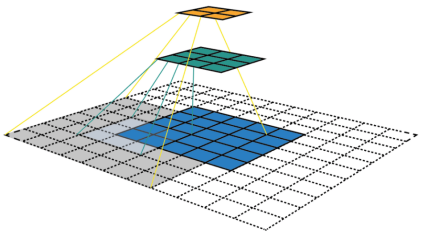
Limitations:

- Parameters: $\mathcal{O}(N)$ (size of the graph), can be cut at smaller K,
- Complexity: Computing the basis U is $\mathcal{O}(N^3)$.
- Domain adaptation: filters are specific to Λ and do not generalize well to other graphs.

Spectral networks are non-localized

Image from Dang Ha The Hien

Receptive field: the part of the original input that is visible to a given neuron.

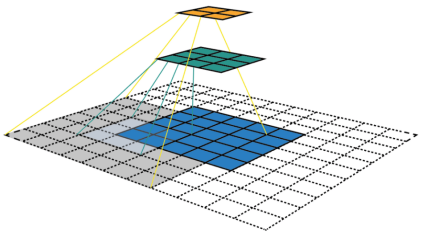


Original 5x5 image (blue), two 3x3 CNN layers (green and orange) stride 2, 2x2 pooling.

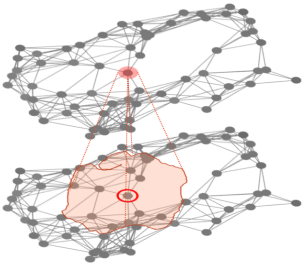
Spectral networks are non-localized

Image from Dang Ha The Hien

Receptive field: the part of the original input that is visible to a given neuron.



Original 5x5 image (blue), two 3x3 CNN layers (green and orange) stride 2, 2x2 pooling.



Example of a localized receptive field, in a single GCN layer.

In general, $h_{l,c,c'}(L)X_{l-1}[:,c]$ is non-localized.

Parametrized graph convolution I

How to overcome the high (computational/parameter) complexity issue?

Parametrized graph convolution I

How to overcome the high (computational/parameter) complexity issue?

Parametrize h with parameters θ!

- Number of parameters can be explicitly controlled.
  - Avoid overfitting.
  - But, need to make sure that we do not restrict the representation capacity of neural network too much.
- Convolution can be performed in the vertex domain:
  - no need for eigendecomposition
  - it becomes easier to use network across different graphs.
- Localization: distributed implementation

Parametrized graph convolution II

Polynomials (Shuman et al., 2011),(Defferrard et al., 2016)

We can parametrize function h as a polynomial

$$h_\theta(\lambda) = \sum_{p=0}^{P} \theta_p \lambda^p$$

Parametrized graph convolution II

Polynomials (Shuman et al., 2011),(Defferrard et al., 2016)

We can parametrize function h as a polynomial

$$h_\theta(\lambda) = \sum_{p=0}^{P} \theta_p \lambda^p$$

- Monomials:

$$h_\theta(L)x = \sum_{p=0}^{P} \theta_p L^p x = \sum_{p=0}^{P} \theta_p x_p, \quad \text{with } x_0 = 1,\ x_1 = x,\ x_p = L x_{p-1}.$$

Parametrized graph convolution II

Polynomials (Shuman et al., 2011),(Defferrard et al., 2016)

We can parametrize function h as a polynomial

$$h_\theta(\lambda) = \sum_{p=0}^{P} \theta_p \lambda^p$$

- Monomials:

$$h_\theta(L)x = \sum_{p=0}^{P} \theta_p L^p x = \sum_{p=0}^{P} \theta_p x_p, \quad \text{with } x_0 = 1,\ x_1 = x,\ x_p = L x_{p-1}.$$

- Chebyshev polynomials

$$h_\theta(L)x = \sum_{t=0}^{P} \theta_t T_t(L)x = \sum_{t=0}^{P} \theta_t x_t, \quad \text{with } x_0 = 1,\ x_1 = Lx,\ x_{t+1} = 2Lx_t - x_{t-1}.$$

Receptive field of polynomials

Polynomials are localized filters (i.e., they only look at the neighborhood of each node):
Every polynomial filter can be written in monomial form:

$$y = h_\theta(L)x = Uh_\theta(\Lambda)U^\top x = U\left(\sum_{p=0}^{P}\theta_p\Lambda^p\right)U^\top x = \sum_{p=0}^{P}\theta_p U\Lambda^p U^\top x = \sum_{p=0}^{P}\theta_p L^p x$$

Receptive field of polynomials

Polynomials are localized filters (i.e., they only look at the neighborhood of each node):
Every polynomial filter can be written in monomial form:

$$y = h_\theta(L)x = Uh_\theta(\Lambda)U^\top x = U\left(\sum_{p=0}^{P}\theta_p\Lambda^p\right)U^\top x = \sum_{p=0}^{P}\theta_p U\Lambda^p U^\top x = \sum_{p=0}^{P}\theta_p L^p x$$

Thus, the output of a single polynomial graph convolution layer at node v_i is dependent on

$$y(i) = \sum_{p=0}^{P}\theta_P \sum_{j=1}^{N}(L^p)(i,j)x(j) = \sum_{j=1}^{N}x(j)\underbrace{\sum_{p=0}^{P}\theta_p(L^p)(i,j)}_{0 \text{ if } \mathrm{dist}(i,j)>P}$$

The receptive field of a polynomial GCN layer is the P-neighborhood!

Receptive field of polynomials

Polynomials are localized filters (i.e., they only look at the neighborhood of each node):
Every polynomial filter can be written in monomial form:

$$y = h_\theta(L)x = Uh_\theta(\Lambda)U^\top x = U\left(\sum_{p=0}^{P}\theta_p\Lambda^p\right)U^\top x = \sum_{p=0}^{P}\theta_p U\Lambda^p U^\top x = \sum_{p=0}^{P}\theta_p L^p x$$

Thus, the output of a single polynomial graph convolution layer at node v_i is dependent on

$$y(i) = \sum_{p=0}^{P}\theta_P \sum_{j=1}^{N}(L^p)(i,j)x(j) = \sum_{j=1}^{N}x(j)\underbrace{\sum_{p=0}^{P}\theta_p(L^p)(i,j)}_{0 \text{ if } \mathrm{dist}(i,j)>P}$$

The receptive field of a polynomial GCN layer is the P-neighborhood!

Parametrized graph convolution III

Rational designs (Isufi et al., 2017), (Levie et al., 2019)

We can parametrize function h as a ratio of polynomials

$$h_\theta(\lambda) = \frac{b_Q(\lambda)}{a_P(\lambda)} = \frac{\sum_{q=0}^{Q} b_q \lambda^q}{1 + \sum_{p=1}^{P} a_p \lambda^p} \quad \text{with} \quad \theta = [a_0, a_1, \ldots, a_P, b_1, \ldots, b_P].$$

Parametrized graph convolution III

Rational designs (Isufi et al., 2017), (Levie et al., 2019)

We can parametrize function h as a ratio of polynomials

$$h_\theta(\lambda) = \frac{b_Q(\lambda)}{a_P(\lambda)} = \frac{\sum_{q=0}^{Q} b_q \lambda^q}{1 + \sum_{p=1}^{P} a_p \lambda^p} \quad \text{with} \quad \theta = [a_0, a_1, \ldots, a_P, b_1, \ldots, b_P].$$

- Higher modeling capacity
- Non-localized
- Implementation: solve the sparse linear system $a_P(L)y = b_Q x$

A simplified graph domain design

Graph Convolution Networks (Kipf & Welling, 2016b) – without normalization trick

A 1-hop GCN layer with C_l channels ($H_l \in \mathbb{R}^{C_{l-1} \times C_l}$) amounts to:

$$X_l(:, c') = \sigma \left(\sum_{c=1}^{C_{l-1}} H_l(c, c') L \, X_{l-1}(:, c) \right) \quad \forall c'$$

A simplified graph domain design

Graph Convolution Networks (Kipf & Welling, 2016b) – without normalization trick

A 1-hop GCN layer with C_l channels ($H_l \in \mathbb{R}^{C_{l-1} \times C_l}$) amounts to:

$$X_l(:, c') = \sigma \left(\sum_{c=1}^{C_{l-1}} H_l(c, c') L \, X_{l-1}(:, c) \right) \quad \forall c'$$

or $X_l = \sigma(L X_{l-1} H_l)$

A simplified graph domain design

Graph Convolution Networks (Kipf & Welling, 2016b) – without normalization trick

A 1-hop GCN layer with C_l channels ($H_l \in \mathbb{R}^{C_{l-1} \times C_l}$) amounts to:

$$X_l(:, c') = \sigma \left(\sum_{c=1}^{C_{l-1}} H_l(c, c') L \, X_{l-1}(:, c) \right) \quad \forall c'$$

or $X_l = \sigma(L X_{l-1} H_l)$ or

$$X_l(i, :) = \sigma \left(\sum_j L(i, j) X_{l-1}(j, :) \, H_l \right)$$

A simplified graph domain design

Graph Convolution Networks (Kipf & Welling, 2016b) – without normalization trick

A 1-hop GCN layer with C_l channels ($H_l \in \mathbb{R}^{C_{l-1} \times C_l}$) amounts to:

$$X_l(:, c') = \sigma \left(\sum_{c=1}^{C_{l-1}} H_l(c, c') L \, X_{l-1}(:, c) \right) \quad \forall c'$$

or $X_l = \sigma(L X_{l-1} H_l)$ or

$$X_l(i, :) = \sigma \left(\sum_j L(i, j) X_{l-1}(j, :) \, H_l \right)$$

- combine your features with those of your neighbors
- apply a linear transformation H_l on feature vector
- apply an element-wise non-linearity σ

A simplified graph domain design

Graph Convolution Networks (Kipf & Welling, 2016b) – without normalization trick

A 1-hop GCN layer with C_l channels ($H_l \in \mathbb{R}^{C_{l-1} \times C_l}$) amounts to:

$$X_l(:, c') = \sigma \left(\sum_{c=1}^{C_{l-1}} H_l(c, c') L \, X_{l-1}(:, c) \right) \quad \forall c'$$

or $X_l = \sigma(L X_{l-1} H_l)$ or

$$X_l(i, :) = \sigma \left(\sum_j L(i, j) X_{l-1}(j, :) \, H_l \right)$$

- combine your features with those of your neighbors
- apply a linear transformation H_l on feature vector
- apply an element-wise non-linearity σ

Question: What is the receptive field of a node at layer l ?

Plan

Structure is key to learning

How do Convolutional Neural Networks exploit grid structures?

Learning from graph-structured data

From classical CNNs to graph CNNs
- Convolution on Graphs
- Graph coarsening

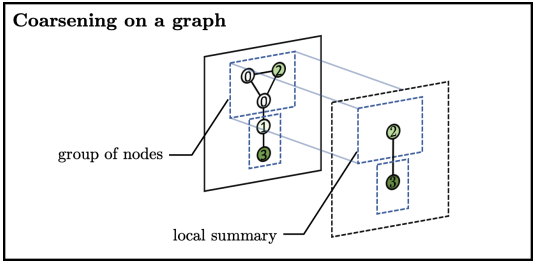
Applications and other topics

Graph coarsening for GCNN

- Analogous to pooling in CNN.
- Useful to decrease the size of the graph.
- Hierarchical coarsening allows to apply a fully connected layer to predict graph properties.

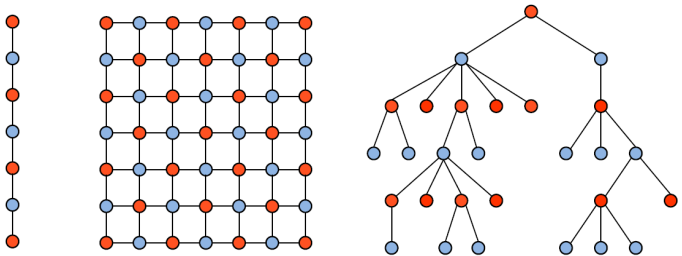
No single way to do coarsening.
We will have a look at 2:

- Kron reduction
- Heavy-edge contraction



What is pooling on a graph?

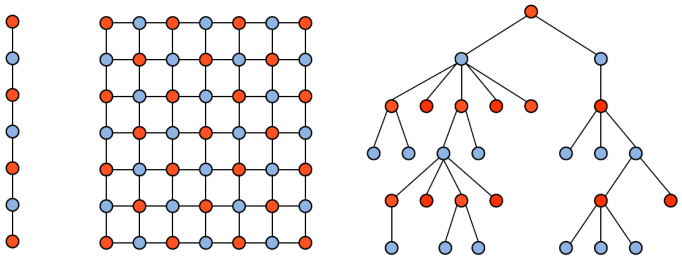
If we interpret pooling as downsampling on images or sounds: select every other node.



We keep the blue nodes S, remove the red nodes T and connect 2 blue nodes if they have a common red neighbor.

What is pooling on a graph?

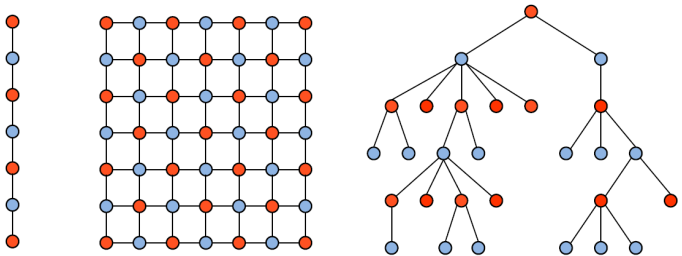
If we interpret pooling as downsampling on images or sounds: select every other node.



We keep the blue nodes S, remove the red nodes T and connect 2 blue nodes if they have a common red neighbor. Not that simple...

What is pooling on a graph?

If we interpret pooling as downsampling on images or sounds: select every other node.



We keep the blue nodes S, remove the red nodes T and connect 2 blue nodes if they have a common red neighbor. Not that simple...

For which graphs direct downsampling works?

Kron reduction I

Graph downsampling

Bipartite graph: V partitioned in S and T and $\forall e_{ij} \in E : v_i \in S$ and $v_j \in T$ or vice versa.

For a bipartite graph, the two groups can be recovered from polarity of the eigenvector $u_{\max}$ associated to the largest eigenvalue $\lambda_{\max}$.

$$S = \{v_i \in V \,|\, u_{\max}(i) \geq 0\}$$

Kron reduction I

Graph downsampling

Bipartite graph: V partitioned in S and T and $\forall e_{ij} \in E : v_i \in S$ and $v_j \in T$ or vice versa.

For a bipartite graph, the two groups can be recovered from polarity of the eigenvector $u_{\max}$ associated to the largest eigenvalue $\lambda_{\max}$.

$$S = \{v_i \in V \,|\, u_{\max}(i) \geq 0\}$$

Exercise

Show that for the normalized Laplacian of a bipartite graph, $\lambda_{\max} = 2$ and

$$u_{\max}(i) = \begin{cases} +N^{-1/2}, & v_i \in S \\ -N^{-1/2}, & v_i \in T \end{cases}$$

Kron reduction I

Graph downsampling

Bipartite graph: V partitioned in S and T and $\forall e_{ij} \in E : v_i \in S$ and $v_j \in T$ or vice versa.

For a bipartite graph, the two groups can be recovered from polarity of the eigenvector $u_{\max}$ associated to the largest eigenvalue $\lambda_{\max}$.

$$S = \{v_i \in V \,|\, u_{\max}(i) \geq 0\}$$

Exercise

Show that for the normalized Laplacian of a bipartite graph, $\lambda_{\max} = 2$ and

$$u_{\max}(i) = \begin{cases} +N^{-1/2}, & v_i \in S \\ -N^{-1/2}, & v_i \in T \end{cases}$$

Heuristic: use $u_{\max}$ polarity to decide which vertex to keep for any graph (Shuman et al., 2016).

Kron reduction II

Weights of the new graph

Now that we have S and T we need to compute the new coarsened normalized Laplacian L_c $(n \times n)$ from L $(N \times N)$ which can be written as

$$L = \begin{bmatrix} L_S & L_{ST} \\ L_{ST}^\top & L_T \end{bmatrix}$$

Kron reduction II

Weights of the new graph

Now that we have S and T we need to compute the new coarsened normalized Laplacian L_c $(n \times n)$ from L $(N \times N)$ which can be written as

$$L = \begin{bmatrix} L_S & L_{ST} \\ L_{ST}^\top & L_T \end{bmatrix}$$

We can use the Kron reduction/Schur complement:

$$L_c = L_S - L_{ST} L_T^{-1} L_{ST}^\top$$

Kron reduction III

Resistance distance preservation

The resistance distance $R_G(i,j)$ between v_i, v_j is equal to

$$R_G(i,j) = (\delta_i - \delta_j)^\top L^+ (\delta_i - \delta_j)$$

Electrical network interpretation:

- edges correspond to 1 ohm resistances
- unit current enters at v_i and leaves from v_j

R_G is a proper metric!

Kron reduction III

Resistance distance preservation

The resistance distance $R_G(i,j)$ between v_i, v_j is equal to

$$R_G(i,j) = (\delta_i - \delta_j)^\top L^+ (\delta_i - \delta_j)$$

Electrical network interpretation:

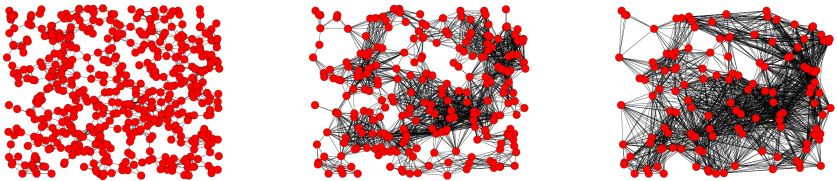
- edges correspond to 1 ohm resistances
- unit current enters at v_i and leaves from v_j

R_G is a proper metric!

Kron reduction preserves resistance distance (Dörfler & Bullo, 2013):

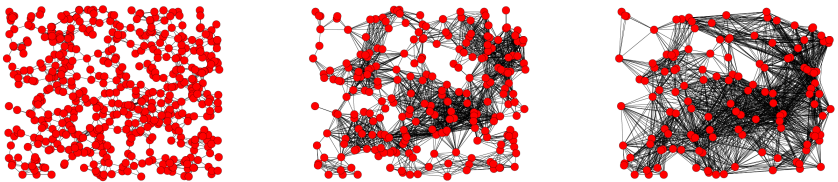
$$R_G(i,j) = R_{G_c}(i,j) \quad \text{for all} \quad v_i, v_j \in S$$

Kron reduction IV



Two levels of kron reduction.

Kron reduction IV



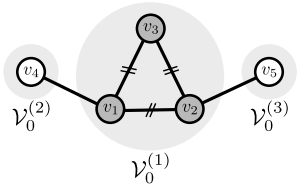
Two levels of kron reduction.

Limitations:

- graph downsampling is heuristic (for non bipartite graph)
- breaks sparsity pattern
- no guaranty of spectral domain preservation
- high complexity!

Graph coarsening I

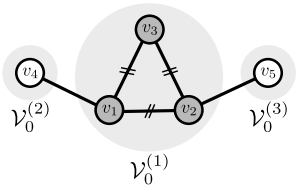
(Karypis & Kumar, 1998),(Loukas, 2018)



Graph G

Graph coarsening I

(Karypis & Kumar, 1998),(Loukas, 2018)



Graph G

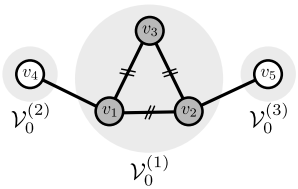
Coarse graph G_c.

To coarsen $G = (V, E)$ into $G_c = (V_c, E_c)$:

- Partition G into n connected subgraphs $G^{(r)} = (V^{(r)}, E^{(r)})$. Call $V^{(r)}$ a contraction set.
- Form a vertex $v'_r \in V_c$ for every contraction set.
- The weight of edge $(v'_r, v'_p) \in E_c$ is equal to $w(V^{(r)}, V^{(p)})$.

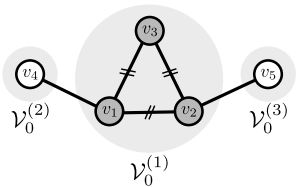
Graph coarsening II

Toy example



Graph coarsening II

Toy example



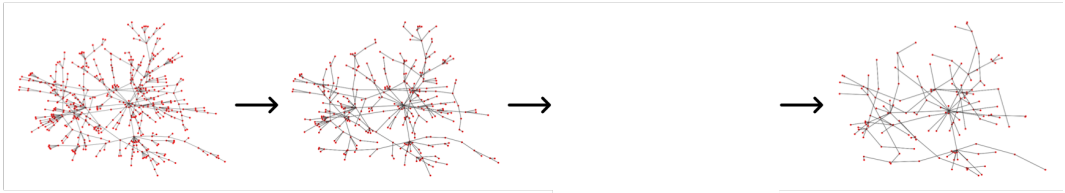
$$P = \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad P^+ = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and coarsening results in

$$L_c = P^{\mp} L P^+ = \begin{bmatrix} 2 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}, \quad x_c = Px = \begin{bmatrix} (x(1)+x(2)+x(3))/3 \\ x(4) \\ x(5) \end{bmatrix}.$$

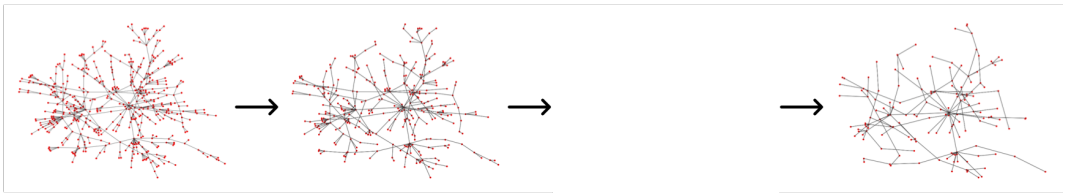
Graph coarsening III

Multilevel construction



Graph coarsening III

Multilevel construction



Benefits:

- Speed: only $O(M)$ operations needed to coarsen (trivial pseudo-inversion)!
- Interpretability: the sparsity structure and cuts are preserved.

Graph coarsening IV

Heavy-edge contraction (Karypis & Kumar, 1998)

Which sets do we want to contract?

Graph coarsening IV

Heavy-edge contraction (Karypis & Kumar, 1998)

Which sets do we want to contract? The heavy ones, i.e., having large weight.
Intuition: nodes are close and can be merged.

Graph coarsening IV

Heavy-edge contraction (Karypis & Kumar, 1998)

Which sets do we want to contract? The heavy ones, i.e., having large weight.
Intuition: nodes are close and can be merged.

Algorithm for edge-based sets

- Find a max-weight matching $M \subset E$ (complexity $O(N^3) \rightarrow$ approximation in $\tilde{O}(|E|)$),

Graph coarsening IV

Heavy-edge contraction (Karypis & Kumar, 1998)

Which sets do we want to contract? The heavy ones, i.e., having large weight.
Intuition: nodes are close and can be merged.

Algorithm for edge-based sets

- Find a max-weight matching $M \subset E$ (complexity $O(N^3) \rightarrow$ approximation in $\tilde{O}(|E|)$),
- Define your contraction sets as $\{\{v_i, v_j\} : e_{ij} \in M\}$,

Graph coarsening IV

Heavy-edge contraction (Karypis & Kumar, 1998)

Which sets do we want to contract? The heavy ones, i.e., having large weight.
Intuition: nodes are close and can be merged.

Algorithm for edge-based sets

- Find a max-weight matching $M \subset E$ (complexity $O(N^3) \rightarrow$ approximation in $\tilde{O}(|E|)$),
- Define your contraction sets as $\{\{v_i, v_j\} : e_{ij} \in M\}$,
- Build the P fat matrix of size $(N - |M|) \times N$,

Graph coarsening IV

Heavy-edge contraction (Karypis & Kumar, 1998)

Which sets do we want to contract? The heavy ones, i.e., having large weight.
Intuition: nodes are close and can be merged.

Algorithm for edge-based sets

- Find a max-weight matching $M \subset E$ (complexity $O(N^3) \rightarrow$ approximation in $\tilde{O}(|E|)$),
- Define your contraction sets as $\{\{v_i, v_j\} : e_{ij} \in M\}$,
- Build the P fat matrix of size $(N - |M|) \times N$,
- Compute the new Laplacian as $L_c = P^{\mp} L P^{+}$ and the signal $x_c = Px$,

Graph coarsening IV

Heavy-edge contraction (Karypis & Kumar, 1998)

Which sets do we want to contract? The heavy ones, i.e., having large weight.
Intuition: nodes are close and can be merged.

Algorithm for edge-based sets

- Find a max-weight matching $M \subset E$ (complexity $O(N^3) \rightarrow$ approximation in $\tilde{O}(|E|)$),
- Define your contraction sets as $\{\{v_i, v_j\} : e_{ij} \in M\}$,
- Build the P fat matrix of size $(N - |M|) \times N$,
- Compute the new Laplacian as $L_c = P^{\mp} L P^{+}$ and the signal $x_c = Px$,
- Repeat until n is small enough.

Graph coarsening IV

Heavy-edge contraction (Karypis & Kumar, 1998)

Which sets do we want to contract? The heavy ones, i.e., having large weight.
Intuition: nodes are close and can be merged.

Algorithm for edge-based sets

- Find a max-weight matching $M \subset E$ (complexity $O(N^3) \rightarrow$ approximation in $\tilde{O}(|E|)$),
- Define your contraction sets as $\{\{v_i, v_j\} : e_{ij} \in M\}$,
- Build the P fat matrix of size $(N - |M|) \times N$,
- Compute the new Laplacian as $L_c = P^{\mp} L P^{+}$ and the signal $x_c = Px$,
- Repeat until n is small enough.

(Randomized) heavy-edge contraction preserves the principal eigenvalues and eigenvectors of a graph Laplacian (Loukas & Vandergheynst, 2018).

Graph coarsening V

Advanced coarsening schemes (Chen & Safro, 2011; Livne & Brandt, 2012; Loukas, 2018)

Edge weight does not capture global graph structure.

Advanced edge-contraction: substitute edge weight w_{ij} by a score function σ_{ij}.

- σ_{ij} is large: contract edges within clusters
- σ_{ij} is small: avoid contracting bridges

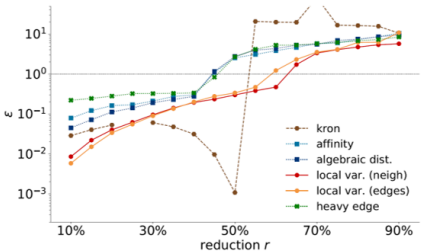
Graph coarsening V

Advanced coarsening schemes (Chen & Safro, 2011; Livne & Brandt, 2012; Loukas, 2018)

Edge weight does not capture global graph structure.

Advanced edge-contraction: substitute edge weight w_{ij} by a score function σ_{ij}.

- σ_{ij} is large: contract edges within clusters
- σ_{ij} is small: avoid contracting bridges



Plan

Structure is key to learning

How do Convolutional Neural Networks exploit grid structures?

Learning from graph-structured data

From classical CNNs to graph CNNs
- Convolution on Graphs
- Graph coarsening

Applications and other topics

Beyond graph convolution

Limitations of graph convolution:

- Does not incorporate edge features (only an edge weight).
- Edge weights remain fixed.
- Spectral interpretation breaks for directed graphs.

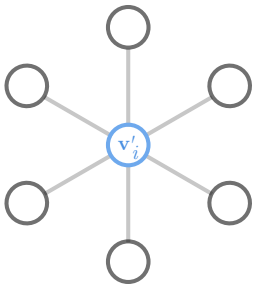
Beyond graph convolution

Limitations of graph convolution:

- Does not incorporate edge features (only an edge weight).
- Edge weights remain fixed.
- Spectral interpretation breaks for directed graphs.

Convolution on a graph amounts to aggregating the values in the neighborhood of each node.

- No fixed size neighborhood
- No order on the nodes



Beyond graph convolution

Limitations of graph convolution:

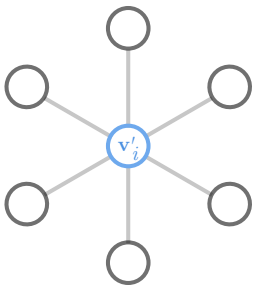
- Does not incorporate edge features (only an edge weight).
- Edge weights remain fixed.
- Spectral interpretation breaks for directed graphs.

Convolution on a graph amounts to aggregating the values in the neighborhood of each node.

- No fixed size neighborhood
- No order on the nodes

There are many alternatives:

- sum, average, minimum, maximum ...
- sort by some criterion and use CNN



Beyond graph convolution

Limitations of graph convolution:

- Does not incorporate edge features (only an edge weight).
- Edge weights remain fixed.
- Spectral interpretation breaks for directed graphs.

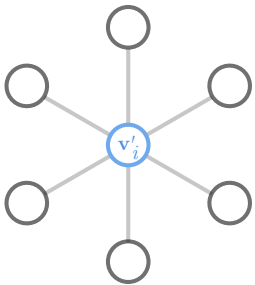
Convolution on a graph amounts to aggregating the values in the neighborhood of each node.

- No fixed size neighborhood
- No order on the nodes

There are many alternatives:

- sum, average, minimum, maximum ...
- sort by some criterion and use CNN

Learn the correct aggregation !!!



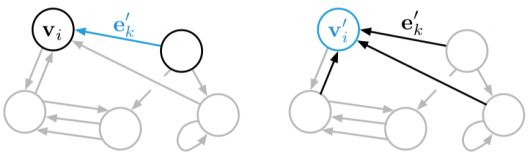
Message-passing framework

(Precup & Teh, 2017; Battaglia et al., 2018)

Let $x_i \in \mathbb{R}^d$ be the features of v_i and $x_{i \to j} \in \mathbb{R}^{d'}$ the features of edge $e_{i \to j}$.

Repeat until convergence:

- **edge update**: $x'_{i \to j} = f_E(x_i, x_j, x_{i \to j})$ use current edge value and values of the 2 linked nodes,
- **node update**: $x'_i = f_V(\{x_{j \to i} \text{ for all } v_j\})$ use incoming edge values (messages).



(a) Edge update (b) Node update

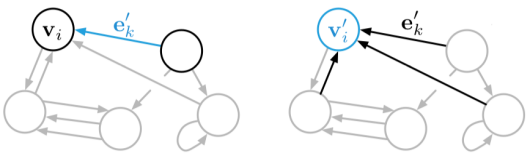
Message-passing framework

(Precup & Teh, 2017; Battaglia et al., 2018)

Let $x_i \in \mathbb{R}^d$ be the features of v_i and $x_{i \to j} \in \mathbb{R}^{d'}$ the features of edge $e_{i \to j}$.

Repeat until convergence:

- **edge update**: $x'_{i \to j} = f_E(x_i, x_j, x_{i \to j})$ use current edge value and values of the 2 linked nodes,
- **node update**: $x'_i = f_V(\{x_{j \to i} \text{ for all } v_j\})$ use incoming edge values (messages).



(a) Edge update (b) Node update

How do we recover $x'_i = \sum_j w_{ij} x_j$ (1-hop graph convolution)?

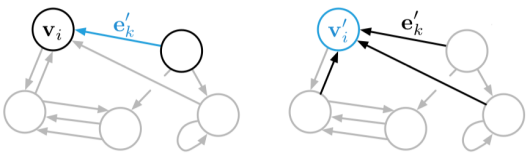
Message-passing framework

(Precup & Teh, 2017; Battaglia et al., 2018)

Let $x_i \in \mathbb{R}^d$ be the features of v_i and $x_{i \to j} \in \mathbb{R}^{d'}$ the features of edge $e_{i \to j}$.

Repeat until convergence:

- **edge update**: $x'_{i \to j} = f_E(x_i, x_j, x_{i \to j})$ use current edge value and values of the 2 linked nodes,
- **node update**: $x'_i = f_V(\{x_{j \to i} \text{ for all } v_j\})$ use incoming edge values (messages).



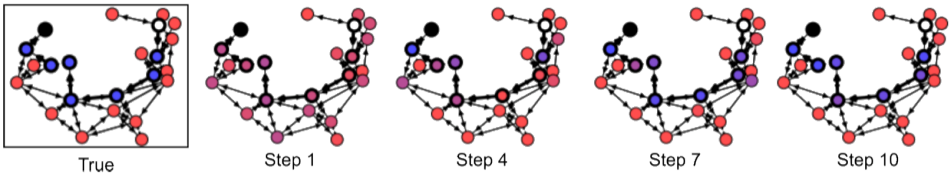
(a) Edge update (b) Node update

How do we recover $x'_i = \sum_j w_{ij} x_j$ (1-hop graph convolution)?

$$f_E(x_i, x_j, x_{i \to j}) = w_{ij}\, x_i \quad \text{and} \quad f_V(\{x_{j \to i} \text{ for all } v_j\}) = \sum x_{j \to i}.$$

Message-passing

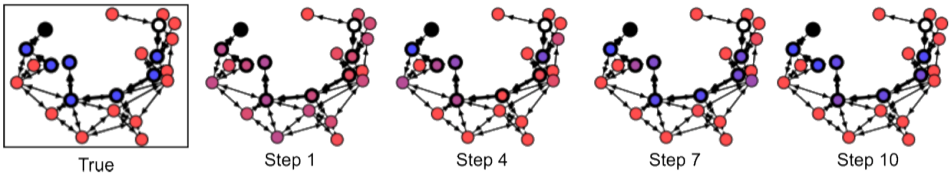
Toy application



Find nodes in the shortest path (in blue) between source (white) and target (black)

Message-passing

Toy application



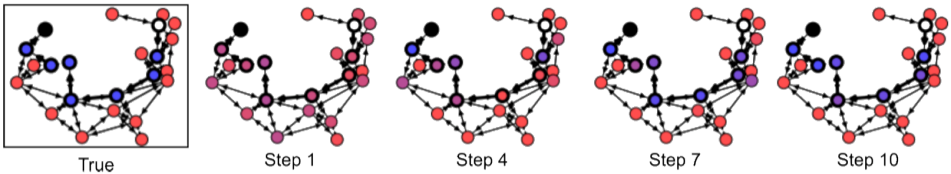
Find nodes in the shortest path (in blue) between source (white) and target (black)

Objective: Find the shortest path between two vertices.

Input: a graph, marked starting node and ending node,
Output: a label for each node = is contained in a shortest path or not.
Training: Examples from graphs in a training set.

Message-passing

Toy application



Find nodes in the shortest path (in blue) between source (white) and target (black)

Objective: Find the shortest path between two vertices.

Input: a graph, marked starting node and ending node,
Output: a label for each node = is contained in a shortest path or not.
Training: Examples from graphs in a training set.

Possibly learns Djikstra's algorithm...

Multihead attention GCN

(Veličković et al., 2018)

Some neighbors state might be more relevant to you than others.

Learn the weights of the graph!

Multihead attention GCN

(Veličković et al., 2018)

Some neighbors state might be more relevant to you than others.

Learn the weights of the graph!

Step 1. Replace

$$X_l(i,:) = \sigma\left(\sum_j L(i,j) X_{l-1}(j,:)\, H_l\right)$$

by

$$X_l(i,:) = \sigma\left(\sum_j \alpha_{j\to i} X_{l-1}(j,:)\, H_l\right),$$

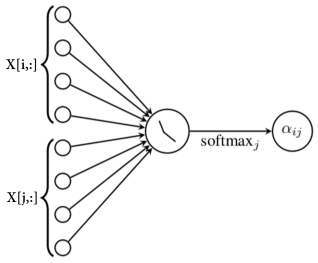
where attention weights $\alpha_{j\to i} \geq 0$ verifying $\sum_j \alpha_{j\to i} = 1$ and $\alpha_{j\to i} = 0$ if $e_{j\to i} \notin E$.

Multihead attention GCN

(Veličković et al., 2018)

Step 2. Compute $\alpha_{j \to i}$ as a function of $X_l(i,:)$ and $X_l(j,:)$.

$$\alpha_{j \to i} = \mathrm{softmax}_j([X_l(i,:)|X_l(j,:)]\, w)$$

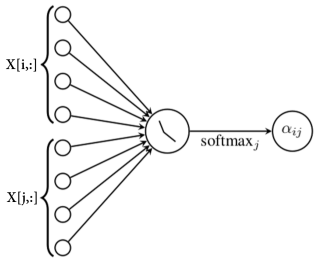


Multihead attention GCN

(Veličković et al., 2018)

Step 2. Compute $\alpha_{j \to i}$ as a function of $X_l(i,:)$ and $X_l(j,:)$.

$$\alpha_{j \to i} = \text{softmax}_j([X_l(i,:)|X_l(j,:)]\, w)$$



Can extend to multiple attention heads.

Semi-supervised learning comparison

(Shchur, Mumme, Bojchevski, & Günnemann, 2018)

	Classes	Features	Nodes	Edges	Label rate	Edge density
CORA	7	1,433	2,485	7,554	0.0563	0.0024
CiteSeer	6	3,703	2,110	5,778	0.0569	0.0026
PubMed	3	500	19,717	64,041	0.0030	0.0003
CORA-Full	67	8,710	18,703	81,124	0.0716	0.0005

Semi-supervised learning comparison

(Shchur et al., 2018)

	Classes	Features	Nodes	Edges	Label rate	Edge density
CORA	7	1,433	2,485	7,554	0.0563	0.0024
CiteSeer	6	3,703	2,110	5,778	0.0569	0.0026
PubMed	3	500	19,717	64,041	0.0030	0.0003
CORA-Full	67	8,710	18,703	81,124	0.0716	0.0005

	CORA	CiteSeer	PubMed	CORA Full
GCN	81.5 ± 1.3	$\mathbf{71.9} \pm 1.9$	77.8 ± 2.9	$\mathbf{62.2} \pm 0.6$
GAT	$\mathbf{81.8} \pm 1.3$	71.4 ± 1.9	$\mathbf{78.7} \pm 2.3$	51.9 ± 1.5
MoNet	81.3 ± 1.3	71.2 ± 2.0	78.6 ± 2.3	59.8 ± 0.8
GS-mean	79.2 ± 7.7	71.6 ± 1.9	77.4 ± 2.2	58.6 ± 1.6
GS-maxpool	76.6 ± 1.9	67.5 ± 2.3	76.1 ± 2.3	40.7 ± 1.5
GS-meanpool	77.9 ± 2.4	68.6 ± 2.4	76.5 ± 2.4	40.5 ± 1.5
MLP	58.2 ± 2.1	59.1 ± 2.3	70.0 ± 2.1	36.8 ± 1.0
LogReg	57.1 ± 2.3	61.0 ± 2.2	64.1 ± 3.1	40.5 ± 0.8
LabelProp	74.4 ± 2.6	67.8 ± 2.1	70.5 ± 5.3	50.5 ± 1.5
LabelProp NL	73.9 ± 1.6	66.7 ± 2.2	72.3 ± 2.9	51.0 ± 1.0

Summary

- Need to exploit graph structure.
- Two types of problems on graphs: **local** and **global**.

Summary

- Need to exploit graph structure.
- Two types of problems on graphs: **local** and **global**.
- Graph Convolution Networks need 2 ingredients:
  - Neighborhood aggregation (graph convolution, message passing, attention)
  - Graph coarsening (Kreon reduction and coarsening).

Summary

- Need to exploit graph structure.
- Two types of problems on graphs: **local** and **global**.
- Graph Convolution Networks need 2 ingredients:
  - Neighborhood aggregation (graph convolution, message passing, attention)
  - Graph coarsening (Kreon reduction and coarsening).
- Exciting and fast developing field!

References I

Bai, Y., Ding, H., Bian, S., Chen, T., Sun, Y., & Wang, W. (2018). Graph edit distance computation via graph neural networks. *CoRR*, *abs/1808.05689*. Retrieved from http://arxiv.org/abs/1808.05689

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V. F., Malinowski, M., ... Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. *CoRR*, *abs/1806.01261*. Retrieved from http://arxiv.org/abs/1806.01261

Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *CoRR*, *abs/1312.6203*. Retrieved from http://arxiv.org/abs/1312.6203

Chen, J., & Safro, I. (2011). Algebraic distance on graphs. *SIAM Journal on Scientific Computing*, *33*(6), 3468–3490.

Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *CoRR*, *abs/1606.09375*. Retrieved from http://arxiv.org/abs/1606.09375

References II

Dörfler, F., & Bullo, F. (2013). Kron reduction of graphs with applications to electrical networks. *IEEE Trans. on Circuits and Systems*, *60*(1), 150–163.

Isufi, E., Loukas, A., Simonetto, A., & Leus, G. (2017). Autoregressive moving average graph filtering. *IEEE Trans. Signal Processing*, *65*(2), 274–288.

Karypis, G., & Kumar, V. (1998). A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. *University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN*.

Kearnes, S. M., McCloskey, K., Berndl, M., Pande, V. S., & Riley, P. (2016). Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, *30*(8), 595–608. Retrieved from `https://doi.org/10.1007/s10822-016-9938-8` doi: 10.1007/s10822-016-9938-8

Kipf, T. N., & Welling, M. (2016a). Semi-supervised classification with graph convolutional networks. *CoRR*, *abs/1609.02907*. Retrieved from `http://arxiv.org/abs/1609.02907`

References III

Kipf, T. N., & Welling, M. (2016b). Semi-supervised classification with graph convolutional networks. *CoRR*, *abs/1609.02907*. Retrieved from http://arxiv.org/abs/1609.02907

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems 25: 26th annual conference on neural information processing systems 2012. proceedings of a meeting held december 3-6, 2012, lake tahoe, nevada, united states.* (pp. 1106–1114). Retrieved from http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

Levie, R., Monti, F., Bresson, X., & Bronstein, M. M. (2019). Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Trans. Signal Processing*, *67*(1), 97–109. Retrieved from https://doi.org/10.1109/TSP.2018.2879624 doi: 10.1109/TSP.2018.2879624

Livne, O. E., & Brandt, A. (2012). Lean algebraic multigrid (lamg): Fast graph laplacian linear solver. *SIAM Journal on Scientific Computing*, *34*(4), B499–B522.

References IV

Loukas, A. (2018). Graph reduction by local variation. *CoRR*, *abs/1808.10650*. Retrieved from http://arxiv.org/abs/1808.10650

Loukas, A., & Vandergheynst, P. (2018, 10–15 Jul). Spectrally approximating large graphs with smaller graphs. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (Vol. 80, pp. 3237–3246). Stockholmsmässan, Stockholm Sweden: PMLR. Retrieved from http://proceedings.mlr.press/v80/loukas18a.html

Parisot, S., Ktena, S. I., Ferrante, E., Lee, M. C. H., Moreno, R. G., Glocker, B., & Rueckert, D. (2017). Spectral graph convolutions for population-based disease prediction. In *Medical image computing and computer assisted intervention - MICCAI 2017 - 20th international conference, quebec city, qc, canada, september 11-13, 2017, proceedings, part III* (pp. 177–185). Retrieved from https://doi.org/10.1007/978-3-319-66179-7_21 doi: 10.1007/978-3-319-66179-7_21

References V

Precup, D., & Teh, Y. W. (Eds.). (2017). *Proceedings of the 34th international conference on machine learning, ICML 2017, sydney, nsw, australia, 6-11 august 2017* (Vol. 70). PMLR. Retrieved from http://jmlr.org/proceedings/papers/v70/

Shchur, O., Mumme, M., Bojchevski, A., & Günnemann, S. (2018). Pitfalls of graph neural network evaluation. *CoRR*, *abs/1811.05868*. Retrieved from http://arxiv.org/abs/1811.05868

Shuman, D. I., Faraji, M. J., & Vandergheynst, P. (2016). A multiscale pyramid transform for graph signals. *IEEE Transactions on Signal Processing*, *64*(8), 2119–2134.

Shuman, D. I., Vandergheynst, P., & Frossard, P. (2011). Chebyshev polynomial approximation for distributed signal processing. In *Distributed computing in sensor systems and workshops (dcoss), 2011 international conference on* (pp. 1–8).

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2017). Graph attention networks. *CoRR*, *abs/1710.10903*. Retrieved from http://arxiv.org/abs/1710.10903

References VI

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph Attention Networks. *International Conference on Learning Representations*. Retrieved from `https://openreview.net/forum?id=rJXMpikCZ` (accepted as poster)