A Network Tour of Data Science (NTDS)

EPFL December 8, 2017

Learning on Graphs

Michaël Defferrard

École Polytechnique Fédérale de Lausanne (EPFL)

Learning

$\mathbf{x} =$ 

$y = f(\mathbf{x}) =$ "cat"

Goal: learn the **unknown** function f.

Structured data

Why structure data?

- To incorporate additional information.
- To exploit spatial correlations.
- To decrease learning complexity by making geometric assumptions.

Data structured by Euclidean grids.

- 1D: sound, time-series.
- 2D: images.
- 3D: video, hyper-spectral images.

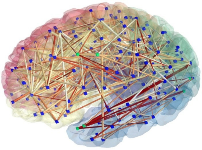
Naturally graph-structured data

Modeling versatility: graphs model heterogeneous pairwise relationships.

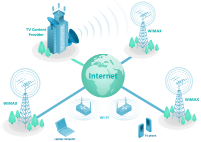
Examples of irregular / graph-structured data:

- Social networks: Facebook, Twitter.
- Biological networks: genes, molecules, brain connectivity.
- Infrastructure networks: energy, transportation, Internet, telephony.

Social network



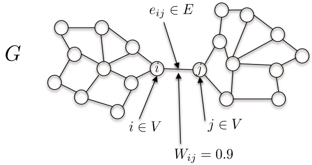
Brain structure



Telecommunication

Notation

$\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$: undirected and connected graph



- $\mathcal{V}$: set of $|\mathcal{V}| = n$ vertices
- $\mathcal{E}$: set of edges
- $\mathbf{W} \in \mathbb{R}^{n \times n}$: weighted adjacency matrix
- $\mathbf{D}_{ii} = \sum_j \mathbf{W}_{ij}$: diagonal degree matrix

Graph Laplacians (core operator to spectral graph theory):

- $\mathbf{L} = \mathbf{D} - \mathbf{W} \in \mathbb{R}^{n \times n}$: combinatorial
- $\mathbf{L} = \mathbf{I}_n - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2} \in \mathbb{R}^{n}$: normalized

The problem

We have:

1. a data matrix $\mathbf{X} \in \mathbb{R}^{N \times d}$,
2. a graph $\mathcal{G}$ represented by its Laplacian $\mathbf{L} \in \mathbb{R}^{N \times N}$.

We want:

- to classify the graph $\mathcal{G}$,
- to classify the vertices v,
- to classify the signals $\mathbf{x} \in \mathbb{R}^{N}$.

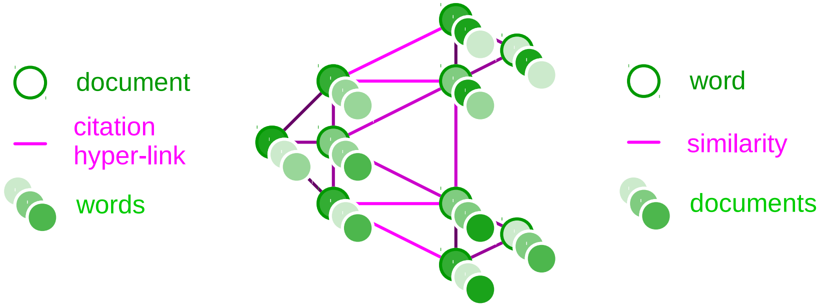
Types of graphs

Sample graph

- Semi-supervised learning.
- Incorporate external information.

Feature graph

- Reduce computations.
- Incorporate external information.



Problems: signals, nodes or graphs classification (regression).

Using the structure

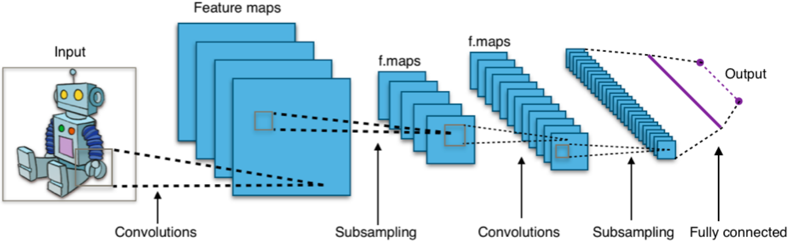
Extrinsic: embed the graph in an Euclidean space.

- Each node is represented by a vector.
- Use that embedding as additional features for a fully connected NN.
- Use a convolutional NN in the embedding space.
  Possibly very high-dimensional!

Intrinsic: a Neural Net defined on graphically structured data.

- Exploit geometric structure for computational efficiency.
- Starting point: ConvNet, an intrinsic formulation for Euclidean grids.

ConvNets: architecture



Ingredients

1. Convolution (local)
2. Non-linearity (point-wise)
3. Down-sampling (global / local)
4. Pooling (local)

ConvNets: why?

ConvNets are extremely efficient at extracting meaningful statistical patterns in large-scale and high-dimensional datasets.

They exploit the geometry.

Key properties

- Convolutional: translation invariance (stationarity).
- Localized: deformation stability & compact filters.
- Multi-scale: hierarchical features extracted by multiple layers.

ConvNets: feature extraction

Zeiler and Fergus 2014

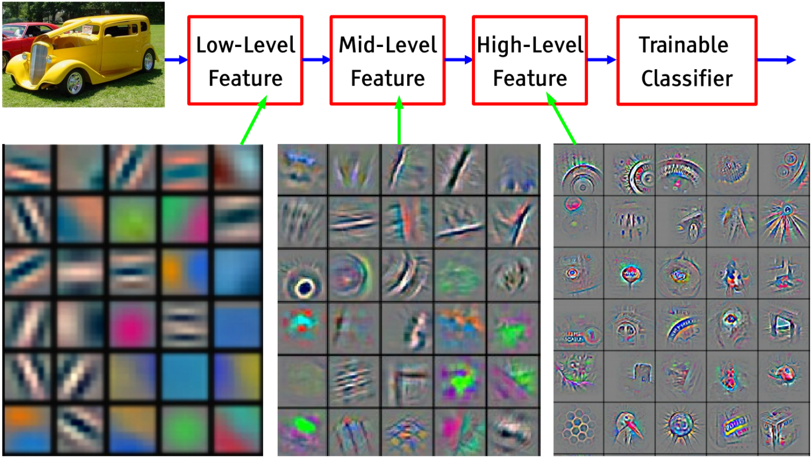


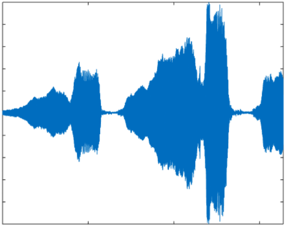
Figure: Features extracted from ImageNet.

Developed for data lying on Euclidean grids

All operations are well defined and computationally efficient:

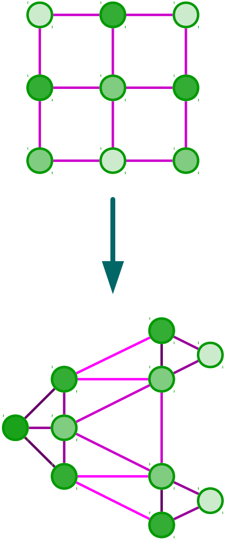
1. Convolution $\rightarrow$ filter translation or fast Fourier transform (FFT).
2. Down-sampling $\rightarrow$ pick one pixel out of n.
3. Non-linearity $\rightarrow$ point-wise operation.
4. Pooling $\rightarrow$ summarize the receptive field.

Image (2D) Video (3D)



Sound (1D)

ConvNets on graphs



Graphs vs Euclidean grids

- Irregular sampling.
- Weighted edges.
- No orientation (in general).

Challenges

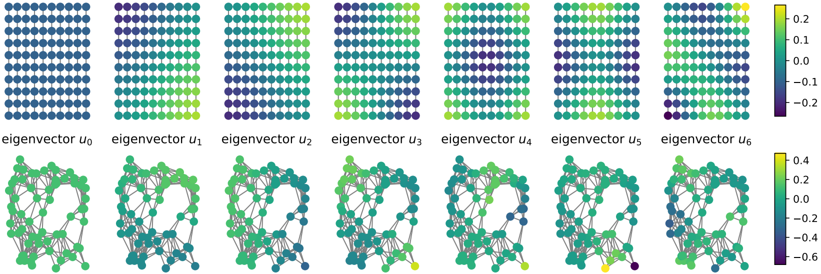
1. Formulate convolution and down-sampling on graphs.
2. Make them efficient!

Graph Fourier basis

Shuman, Narang, Frossard, Ortega, and Vandergheynst 2013

L is symmetric and positive semidefinite $\rightarrow$ $L = U \Lambda U^T$ (EVD)

- Graph Fourier basis $U = [u_1, \ldots, u_n] \in \mathbb{R}^{n \times n}$
- Graph "frequencies" $\Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{bmatrix} \in \mathbb{R}^{n \times n}$

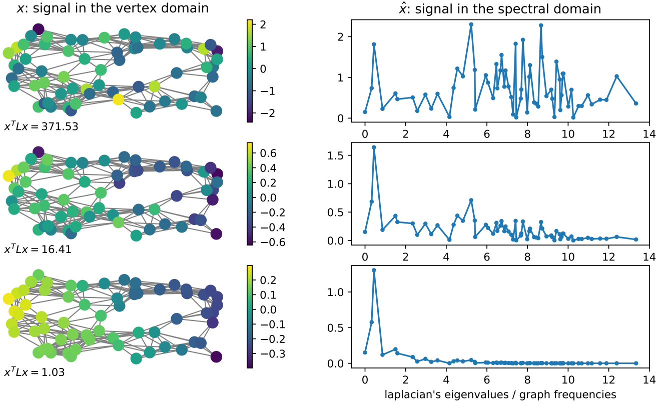


eigenvector u_0 eigenvector u_1 eigenvector u_2 eigenvector u_3 eigenvector u_4 eigenvector u_5 eigenvector u_6

Graph Fourier Transform

Shuman, Narang, Frossard, Ortega, and Vandergheynst 2013

- Graph signal $x : \mathcal{V} \to \mathbb{R}$ seen as $x \in \mathbb{R}^n$
- Transform: $\hat{x} = \mathcal{F}_\mathcal{G}\{x\} = U^T x \in \mathbb{R}^n$
- Inverse: $x = \mathcal{F}_\mathcal{G}^{-1}\{x\} = U\hat{x} = UU^T x = x$

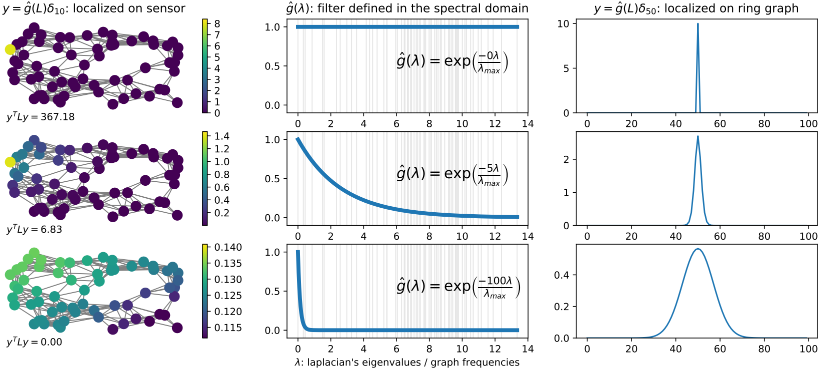


Filtering with convolution on graphs

Shuman, Narang, Frossard, Ortega, and Vandergheynst 2013

Convolution theorem: $y = x *_\mathcal{G} g = U\left(U^T g \odot U^T x\right) = U\left(\hat{g} \odot U^T x\right)$

$$y = x *_\mathcal{G} g = U \begin{bmatrix} \hat{g}(\lambda_0) & & 0 \\ & \ddots & \\ 0 & & \hat{g}(\lambda_{n-1}) \end{bmatrix} U^T x = U\hat{g}(\Lambda)U^T x = \hat{g}(L)x$$



Learning

- Ideal unknown function: $\mathbf{y} = f(\mathbf{x})$.
- Parametrized approximation: $\mathbf{y} \approx f_\theta(\mathbf{x})$, where θ are the parameters.
- Learning a function: $\min_\theta E(\mathbf{y}, f_\theta(\mathbf{x}))$.
- Example of energy/loss/objective: $E(\mathbf{y}, \mathbf{x}) = \|\mathbf{y} - \mathbf{x}\|_2^2$
- In our case, f is graph filtering: $f(\mathbf{x}) = \hat{g}_\theta(\mathbf{L})\mathbf{x}$
- Learning by gradient descent (and backpropagation).

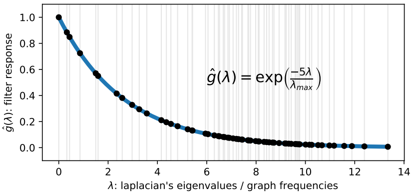
$$\theta^{t+1} = \theta^t - \frac{\partial E}{\partial \theta} = \theta^t - \frac{\partial E}{\partial f}\frac{\partial f}{\partial \theta}$$

$\rightarrow$ we want a differentiable function f!

Spectral filtering of graph signals

Non-parametric filter, can learn all possible filters:

$$\hat{g}_\theta(\Lambda) = \mathrm{diag}(\theta), \; \theta \in \mathbb{R}^n$$



- Non-localized in vertex domain
- Learning complexity in $\mathcal{O}(n)$
- Computational complexity in $\mathcal{O}(n^2)$ (& memory)

Variation: a smooth function such as $\hat{g}_\theta(\Lambda) = B\theta$ where B is the cubic spline basis (Bruna, Zaremba, Szlam, and LeCun 2014).

Polynomial parametrization

Shuman, Ricaud, and Vandergheynst 2016

$$\hat{g}_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k, \ \theta \in \mathbb{R}^K$$

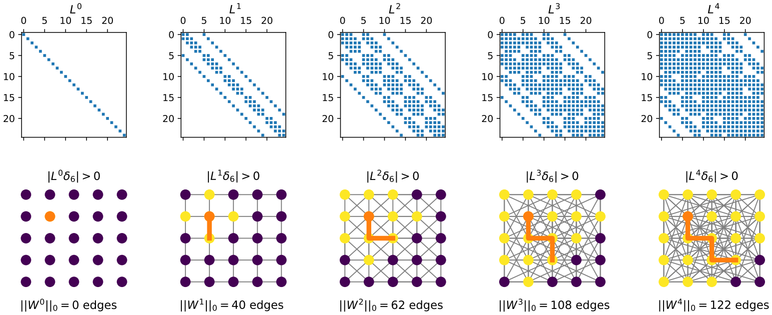
- Can learn all K-localized filters.
- Distributed computing: only need access to the K-neighborhood.

- K-localized
- Learning complexity in $\mathcal{O}(K)$
- Computational complexity in $\mathcal{O}(n^2)$

Filter localization

Hammond, Vandergheynst, and Gribonval 2011, Lemma 5.2

- Value at j of g_θ centered at i: $(\hat{g}_\theta(L)\delta_i)_j = (\hat{g}_\theta(L))_{i,j} = \sum_k \theta_k (L^k)_{i,j}$
- $d_{\mathcal{G}}(i,j) > K$ implies $(L^K)_{i,j} = 0$



Filter localization

Shuman, Ricaud, and Vandergheynst 2016

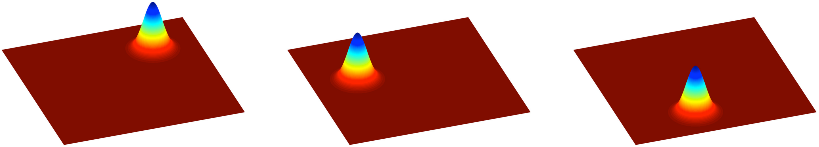


Figure: Localization on regular Euclidean grid.

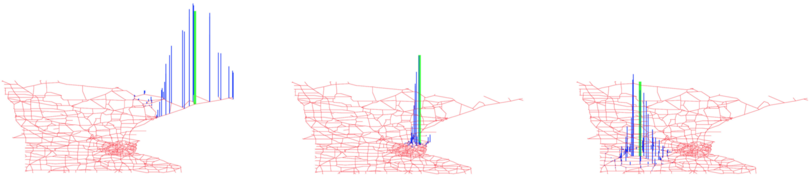


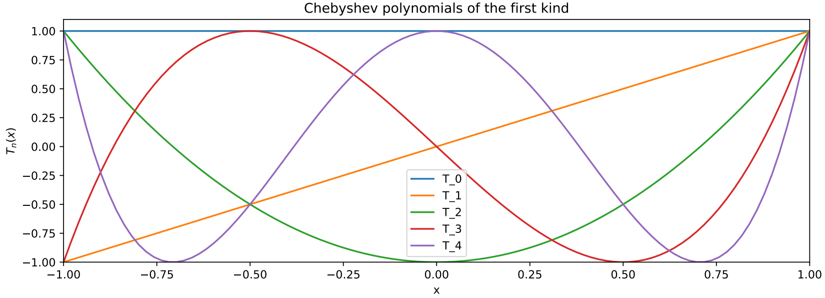
Figure: Localization on graph with $(\hat{g}_\theta(L)\delta_i)_j = (\hat{g}_\theta(L))_{i,j}$.

Recursive formulation with Chebyshev polynomials

Hammond, Vandergheynst, and Gribonval 2011

$$\hat{g}_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}), \quad \tilde{\Lambda} = 2\lambda_n^{-1}\Lambda - I_n$$

Chebyshev polynomials: $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0 = 1$ and $T_1 = x$



Recursive formulation with Chebyshev polynomials

$$y = \hat{g}_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x, \quad \tilde{L} = 2\lambda_n^{-1} L - I_n$$

Recurrence:
$$\begin{aligned} y &= \hat{g}_\theta(L)x = [\bar{x}_0, \ldots, \bar{x}_{K-1}]\theta \\ \bar{x}_k &= T_k(\tilde{L})x = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2} \\ \bar{x}_0 &= x \\ \bar{x}_1 &= \tilde{L}x \end{aligned}$$

- K-localized
- Learning complexity in $\mathcal{O}(K)$
- Computational complexity in $\mathcal{O}(K|\mathcal{E}|)$ (same as classical ConvNets!)

Learning filters

Defferrard, Bresson, and Vandergheynst 2016

$$y_{s,j} = \sum_{i=1}^{F_{in}} \hat{g}_{\theta_{i,j}}(L) x_{s,i} \in \mathbb{R}^n$$

- $x_{s,i}$: feature map i of sample s
- $\theta_{i,j}$: trainable parameters
  ($F_{in} \cdot F_{out}$ vectors of K Chebyshev coefficients)

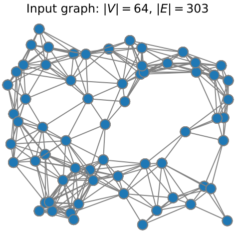
Gradients for backpropagation:

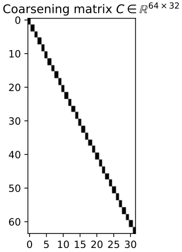
- $\frac{\partial E}{\partial \theta_{i,j}} = \sum_{s=1}^{S} [\bar{x}_{s,i,0}, \ldots, \bar{x}_{s,i,K-1}]^T \frac{\partial E}{\partial y_{s,j}}$
- $\frac{\partial E}{\partial x_{s,i}} = \sum_{j=1}^{F_{out}} g_{\theta_{i,j}}(L) \frac{\partial E}{\partial y_{s,j}}$

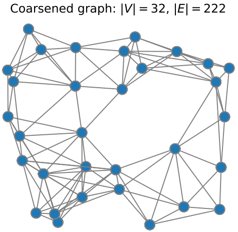
Overall cost of $\mathcal{O}(K|\mathcal{E}|F_{in}F_{out}S)$ operations, $|\mathcal{E}| \propto n$ for sparse graphs

Coarsening

Defferrard, Bresson, and Vandergheynst 2016



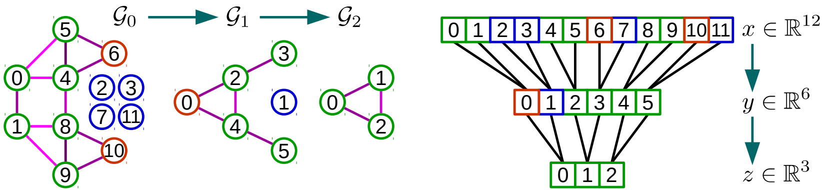




- Inherently combinatorial problem.
- Can be done as pre-processing.
- Greedy node merging with Graclus / Metis (very fast).

Pooling

Defferrard, Bresson, and Vandergheynst 2016

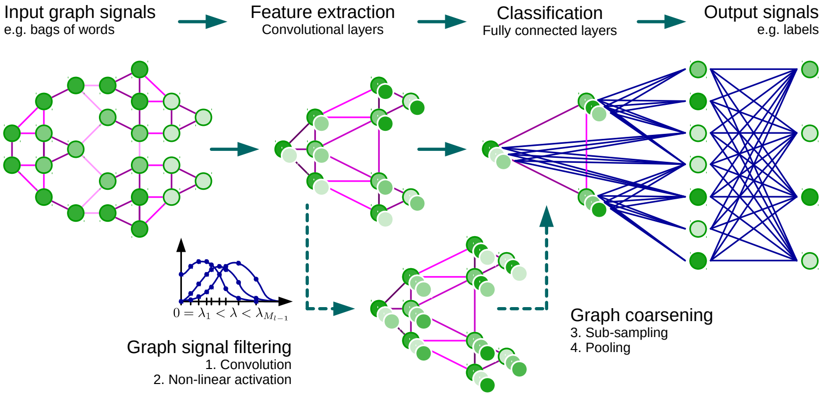


Pooling as any regular 1D signal

- Node order does not matter $\rightarrow$ arrange them for local access.
- Nodes at multiple levels are ordered as a tree.
- Satisfies parallel architectures like GPUs.

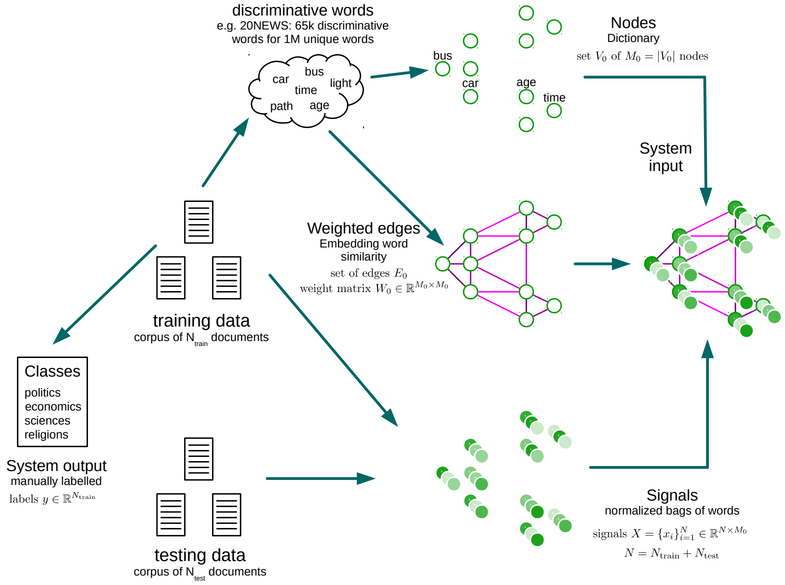
Graph ConvNet architecture

Defferrard, Bresson, and Vandergheynst 2016



Structuring documents with a feature graph

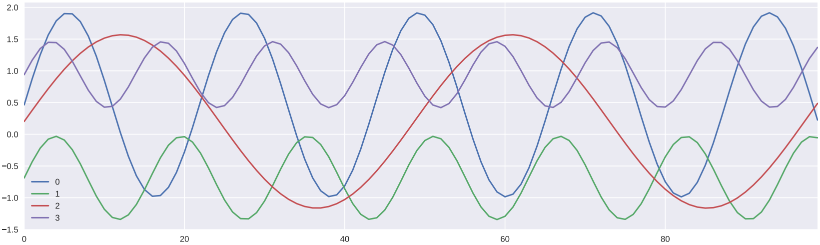
Defferrard, Bresson, and Vandergheynst 2016



Various applications

- Transductive learning
  [Kipf and Welling 2016; Manessi, Rozza, and Manzo 2017]
- Quantum Chemistry
  [Duvenaud et al. 2015; Gilmer, Schoenholz, Riley, Vinyals, and Dahl 2017]
- High Energy Physics
- Computer Graphics [Monti, Boscaini, et al. 2016; Yi, Su, Guo, and Guibas 2016; Wang, Gan, Zhang, and Shui 2017; Simonovsky and Komodakis 2017]
- Community detection [Bruna and Li 2017]
- Brain analysis
  [Ktena et al. 2017; Parisot et al. 2017; Anirudh and Thiagarajan 2017]
- Matrix completion for recommendation
  [Monti, Bronstein, and Bresson 2017]
- Neural machine translation
  [Bastings, Titov, Aziz, Marcheggiani, and Sima'an 2017]
- Link prediction and entity classification in knowledge bases
  [Schlichtkrull et al. 2017]

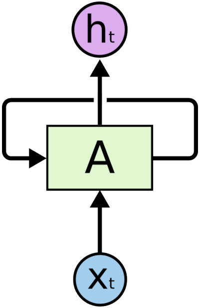
Time Series

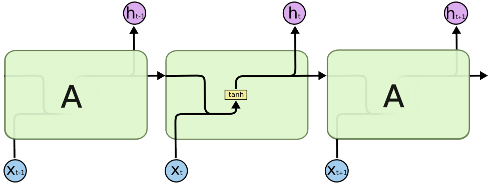


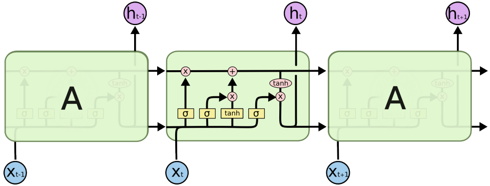
- Sensors: temperature, wind, pressure, body signals, etc.
- Stock market
- Text (series of discrete symbols, i.e. words)
- Network activity: energy, transportation, communication, brain

Recurrent Neural Networks & LSTM

Figures by Colah, 2015







Recurrent Graph Convolutional Network

Seo, Defferrard, Bresson, and Vandergheynst 2016

1D signals

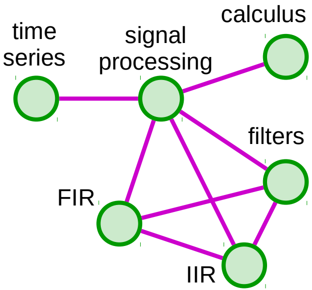
- $h_t = \tanh(W_x x_t + W_h h_{t-1})$
- $y_t = W h_t$
- State stored in hidden units

Graph signals

- $h_t = \tanh(W_x *_{\mathcal{G}} x_t + W_h *_{\mathcal{G}} h_{t-1})$
- $y_t = W *_{\mathcal{G}} h_t$
- State stored locally on the nodes

- Graph filtering x as $y = [\bar{x}_0, \ldots, \bar{x}_{K-1}]\theta$ is a weighted sum of diffused versions $\bar{x}$ of x.
- Data exchanged locally around the K-neighborhood.
- Reduces to independent signals if $K = 1$ or graph has no edge.

Real data: Wikipedia

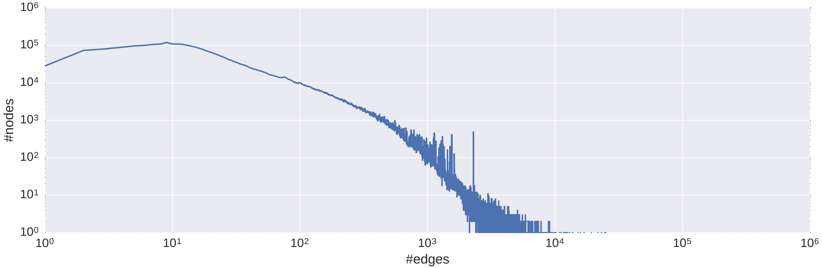


Goal: structured times series forecasting

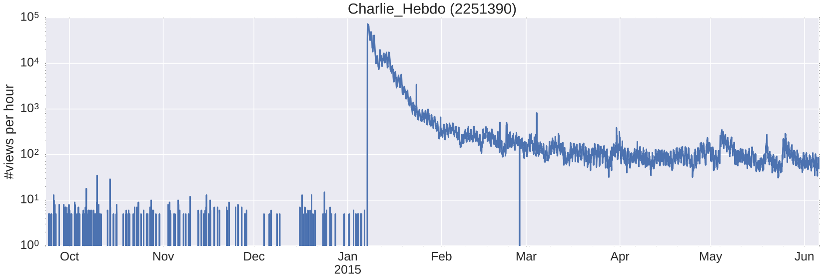
- Anomaly / event detection
- Regulation & Control
- Generative process understanding

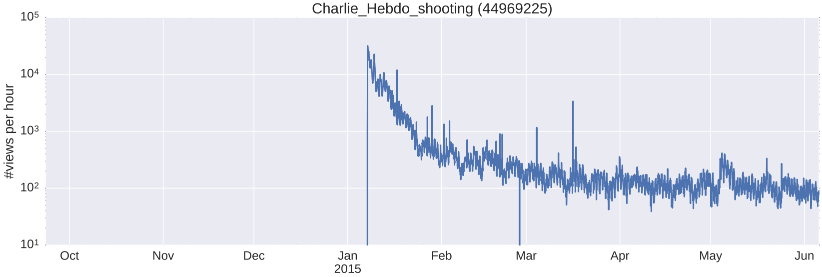
Wikipedia network & signals

- Nodes: articles
- Edges: hyper-links
- Signals: number of hits per hour



Structured Time Series





Conclusion

Instead of engineering feature extractors (filters), learn them.

- Graph are versatile tools to structure real data.
- Neural networks are the most effective ML algorithm today.

References

- Paper: Defferrard, Bresson and Vandergheynst, Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, NIPS, 2016.
- Code: `https://github.com/mdeff/cnn_graph`
- Paper: Seo, Defferrard, Bresson and Vandergheynst, Structured Sequence Modeling with Graph Convolutional Recurrent Networks, arXiv, 2017.
- Code: `https://github.com/youngjoo-epfl/gconvRNN`