Convolutional Neural Networks on Graphs

Xavier Bresson

School of Computer Science and Engineering
Nanyang Technological University

NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

NATIONAL RESEARCH FOUNDATION
PRIME MINISTER'S OFFICE
SINGAPORE

Joint work with M. Defferrard (EPFL), P. Vandergheynst (EPFL),
Y. Seo (EPFL), M. Bronstein (USI), F. Monti (USI), T. Laurent (LMU)

Institute for Mathematical Sciences
National University of Singapore
June 1st 2017

Note: Background image is generated by an artificial neural network with random input, i.e. the AI is dreaming...

Outline

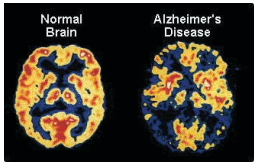
- **Data Science and Deep Learning**
  - *A brief history*
  - *What makes DL great*
- **Convolutional Neural Networks**
  - *Architecture*
- **Convolutional Neural Networks on Graphs**
  - *Non-Euclidean data*
  - *Graph convolution, graph downsampling and pooling*
  - *Numerical experiments*
- **Natural Language Processing**
- **Recommender Systems**
- **Conclusion**

Outline

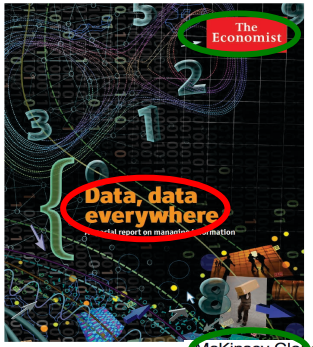
- **Data Science and Deep Learning**
  - *A brief history*
  - *What makes DL great*
- Convolutional Neural Networks
  - *Architecture*
- Convolutional Neural Networks on Graphs
  - *Non-Euclidean data*
  - *Graph convolution, graph downsampling and pooling*
  - *Numerical experiments*
- Natural Language Processing
- Recommender Systems
- Conclusion

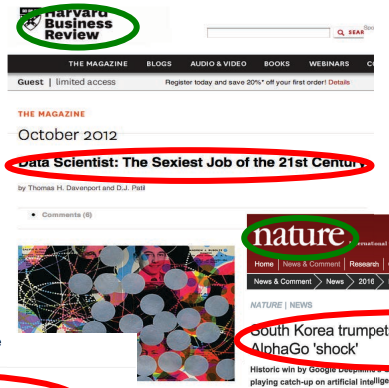
Data Science and Artificial Neural Networks

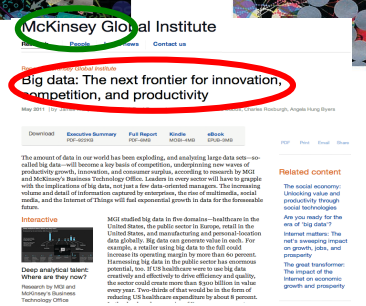
- **Data Science** is an *interdisciplinary* field that aims at *transforming raw data into meaningful knowledge* to provide smart decisions for real-world problems:

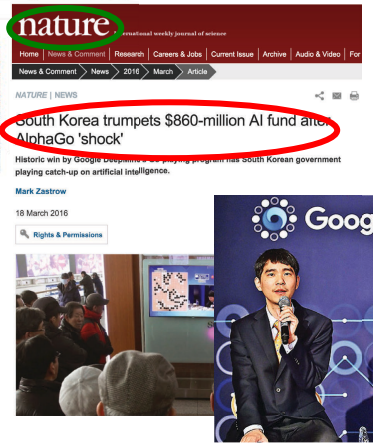


- In the news:

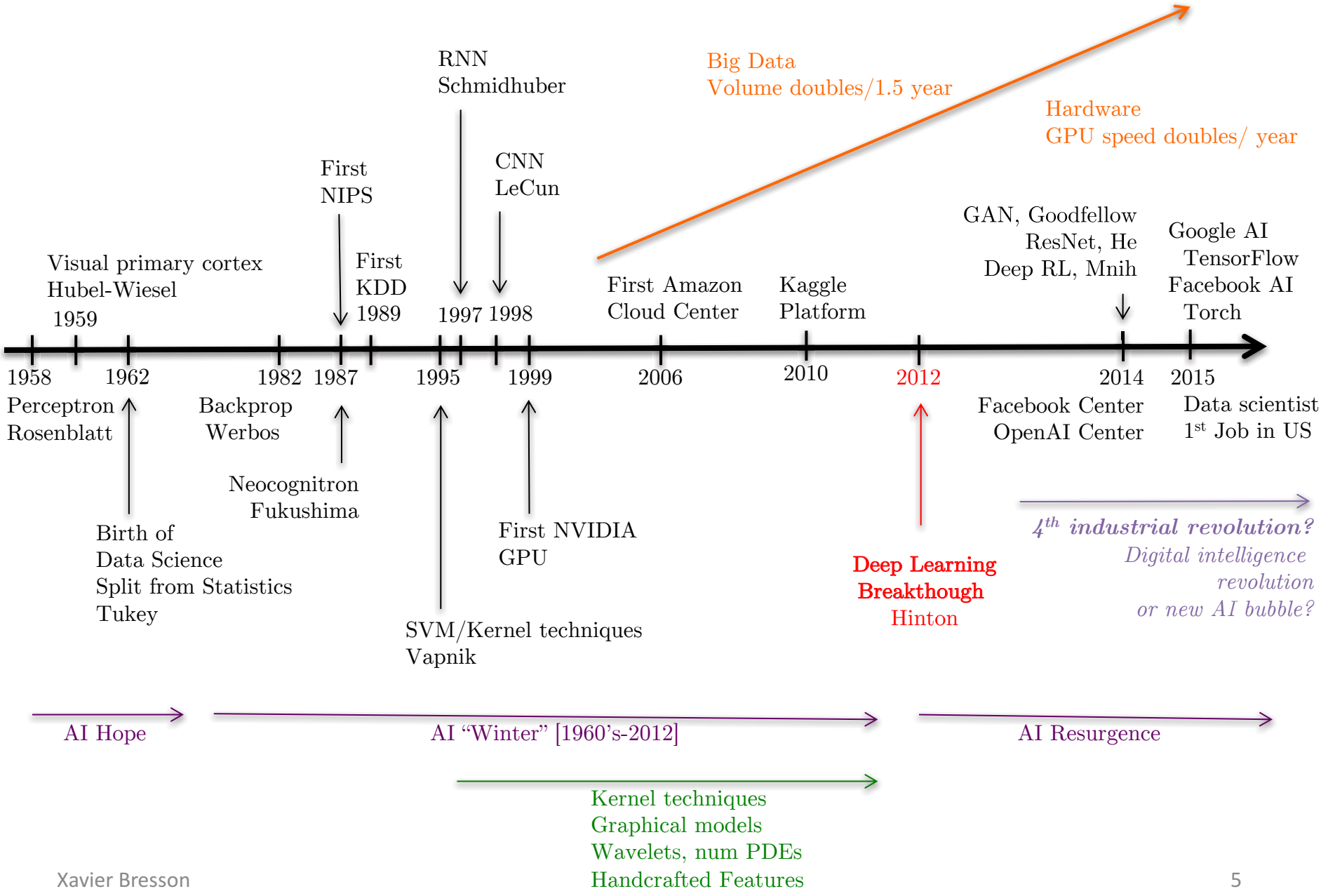








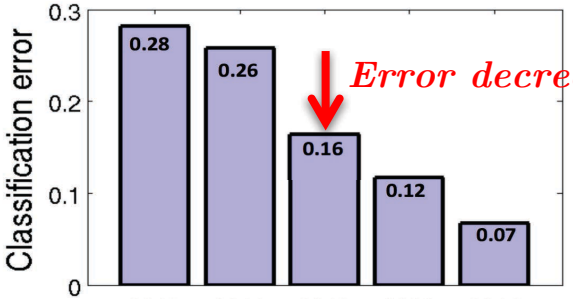
A Brief History of Artificial Neural Networks



2012: Deep Learning Breakthrough

- **ImageNet [Fei Fei-et.al.'09]:** International Image Classification Challenge
  1,000 object classes and 1,431,167 images

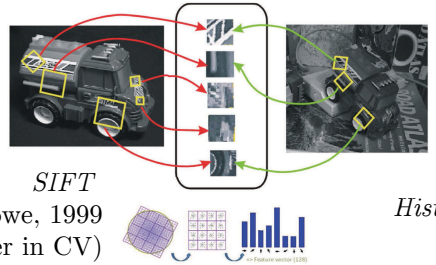
[L. Fei-Fei-et.al.'09]



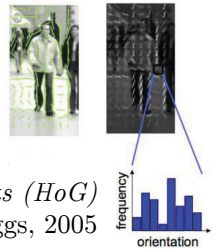
[Russakovsky-et.al.'14]

- **Before 2012:** Handcrafted filters + SVM classification

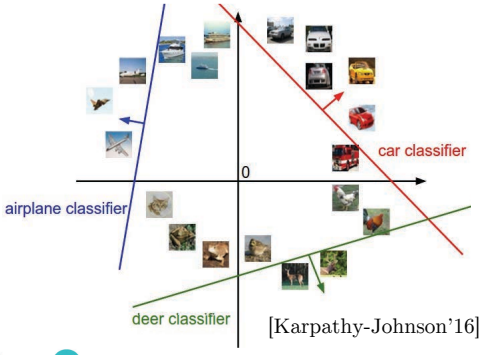
[Fei-Fei Li-et.al.'15]



SIFT
Lowe, 1999
(most cited paper in CV)

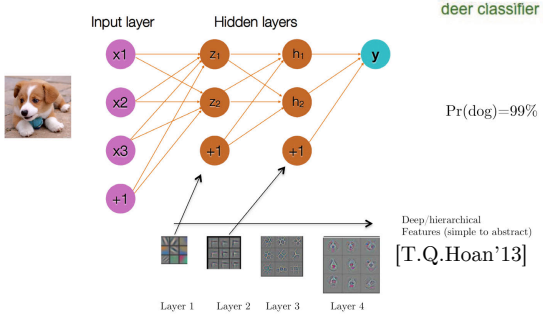


Histogram of Gradients (HoG)
Dalal & Triggs, 2005



[Karpathy-Johnson'16]

After 2012: *Learn filters and classifier with neural networks (end-to-end system).*



[T.Q.Hoan'13]

What Makes Deep Learning Great? [long answer]

Computer Science

Scalable databases for storing, accessing data.

E.g. Cloud computing, Amazon EC2, Hadoop.

Distributed and parallel frameworks for data processing.

E.g. MapReduce, GraphLab.

Mathematical Modeling

Design algorithms that transform data into knowledge.

Use Linear algebra, optimization, graph theory, statistics.

Deep Learning

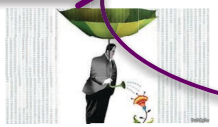
Multidisciplinary field: "1+1=3"

Personalized Services

E.g. Healthcare (enhanced diagnostics)
Commerce (products)

Knowledge Discovery

E.g. Physics, genomics, social sciences.



Data

Collection of massive amounts of data at increasing rate.

E.g. Social networks, sensor networks, mobile devices, biological networks, administrative, economics data

Issues of privacy, security, ownership

Domain Expertise

Sciences

E.g. Economy, Biology, Physics, Neuroscience, sociology.

Government

E.g. Healthcare, Defense, Education, Transportation.

Industry

E.g. E-commerce, Telecommunications, Finance.

Google

Intelligent Systems

E.g. Autonomous cars, security, interactive tools for data organization and exploration.

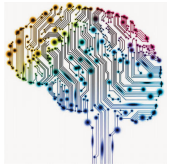
Major challenges: Multidisciplinary integration, large-scale databases, scalable computational infrastructures, design math algorithms for massive datasets, trade-off speed and accuracy for real-time decisions, interactive visualization tools.

What Makes Deep Learning Great? [short answer]

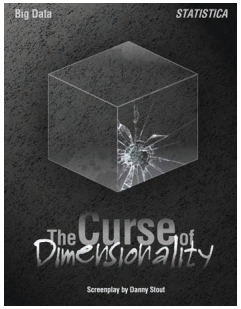
1. **Data.. lots of data:** Inherited from 3rd industrial revolution (digital revolution)

2. **Computational power:** Moore's law (CPUs, GPUs) and cloud computing

   Note: Human brain 10^{14} synapses/10^9 Mflops = Computers in 2040 (10,000 times faster than today)



2040 =



3. **Neural networks can beat the curse of dimentionality** (more next slide).

4. **Neural network is a universal/flexible strategy to solve many challenging real-world problems:**
   DL/AI is the new "electricity" [A. Ng, Baidu], it will change major industries in the coming decade.

Curse of Dimensionality

- **Data lie in high-dim spaces:**

  dim(Go) = 19 x 19 ≈ 10^3

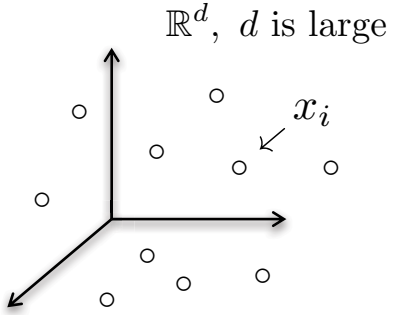
  dim(Images) = 512 x 512 ≈ 10^6

  dim(Books) = 80,000 ≈ 10^5

- **Consequence 1: Curse of dimensionality** *[Beyer'98]*:

  In high dimensions, distances between data is meaningless ⇒ all data are close to each other.

$$lim_{d\to\infty}\ \mathbb{E}\left(\frac{d^{\ell_2}_{\max}(x_i, V - x_i) - d^{\ell_2}_{\min}(x_i, V - x_i)}{d^{\ell_2}_{\min}(x_i, V - x_i)}\right) \to 0$$



- **Consequence 2: All data are intractable.**

  #Go = $3^{19\ \text{x}\ 19}$ ≈ 10^{172}

  #Images = $256^{512\ \text{x}\ 512}$ ≈ $10^{630,000}$

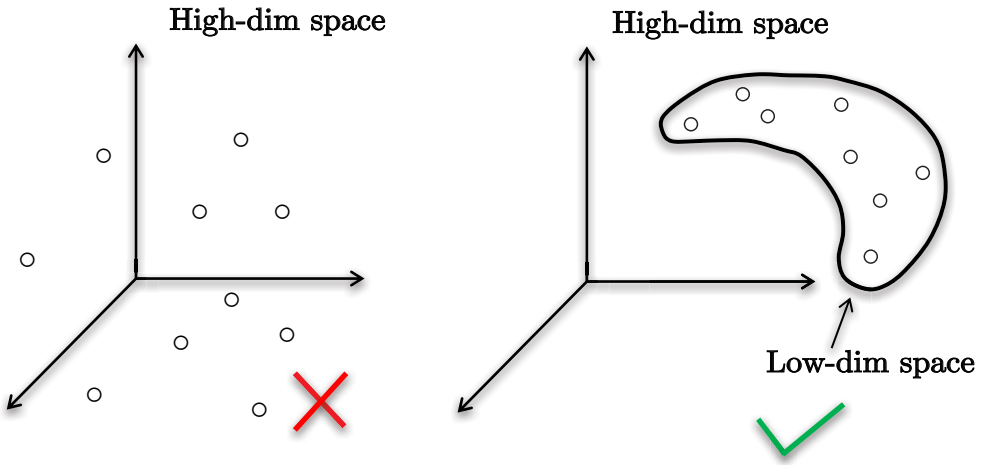
  #Books = $8{,}000^{80,000}$ ≈ $10^{310,000}$

  ⇒ *Can we learn exponential functions f(x), #x=10^{172}? No, in general.*

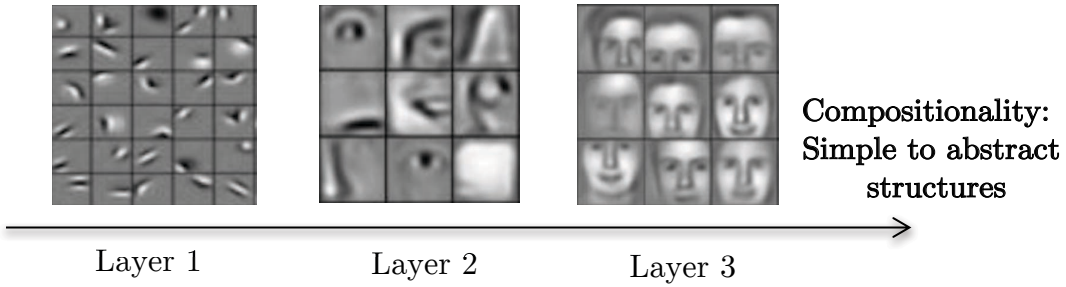
Good News

- **In our universe:**
  1. *Data have structures.*
  2. *These structures are compositional.*

- **Blessing of structure:** Data are not random, they have structures, meaning data are concentrated in low-dim spaces

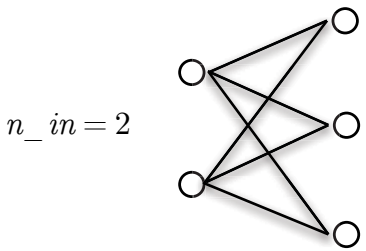


- **Compositionality:** Data are formed of simple structures that agglomerate to compose more complex and abstract structures and so on.



Learning Exponential Functions

- **Neural networks** leverage structureness and compositionality by learning **parametric** exponential functions with **linear** number of parameters.



⇒ **Number of parameters grow linearly, but function capacity grows exponentially.**

- Another good news: Neural networks can learn **a very large number of parameters** in a consistent way (w/ backpropagation), order of 10^9 ⇒ huge capacity to learn (and to generalize).

- Another good news: **Theorem of Universal Approximation [Cybenko'89] is today obsolete.**

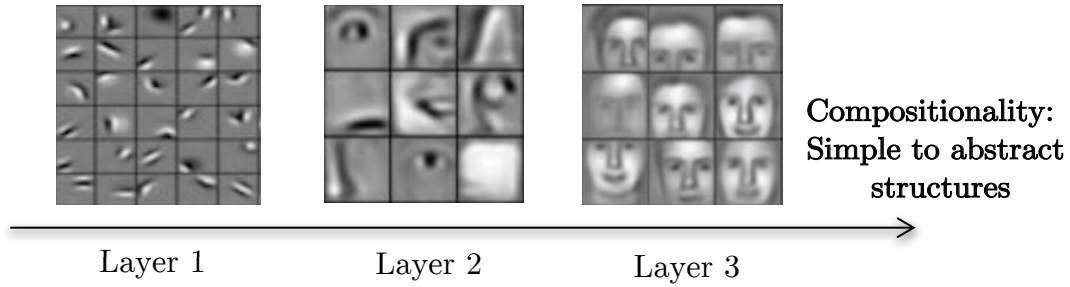
  Theorem states that a neural network with a single hidden layer can approximate nearly every function, but exponential functions of interest require an exponential number of neurons with a single layer.

  Not true with multiple layers ⇒ deep property [Bengio-et.al'14].

Outline

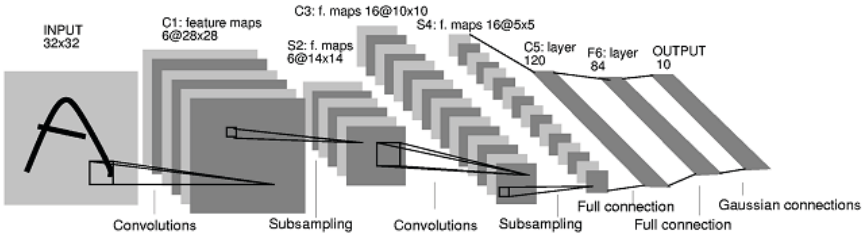
- Data Science and Deep Learning
  - *A brief history*
  - *What makes DL great*
- **Convolutional Neural Networks**
  - *Architecture*
- Convolutional Neural Networks on Graphs
  - *Non-Euclidean data*
  - *Graph convolution, graph downsampling and pooling*
  - *Numerical experiments*
- Natural Language Processing
- Recommender Systems
- Conclusion

How Compositionality is Captured in Images?



- Compositionality is captured by learning **local stationary structures** and compose them into **multiscale hierarchical patterns** (explained next).

- This class of deep learning is called **convolutional neural networks (CNNs)** [LeCun-Bottou-Bengio-Haffner'98].

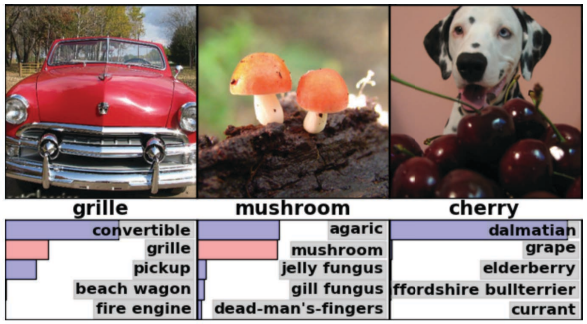


- CNNs are **extremely efficient** at extracting meaningful statistical patterns in large-scale and high-dimensional image datasets

  $\Rightarrow$ CNN has become the Swiss knife of Computer Vision.

CNNs are Great

- CNNs are highly successful for computer vision tasks:



Image/object recognition

[Krizhevsky-Sutskever-Hinton'12]

Image inpainting

[Pathak-Efros-etal'16]

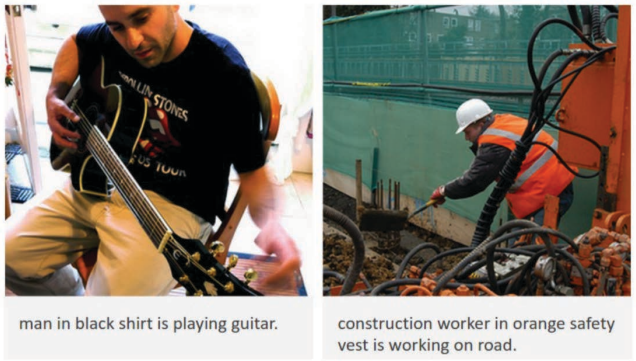


Image captioning

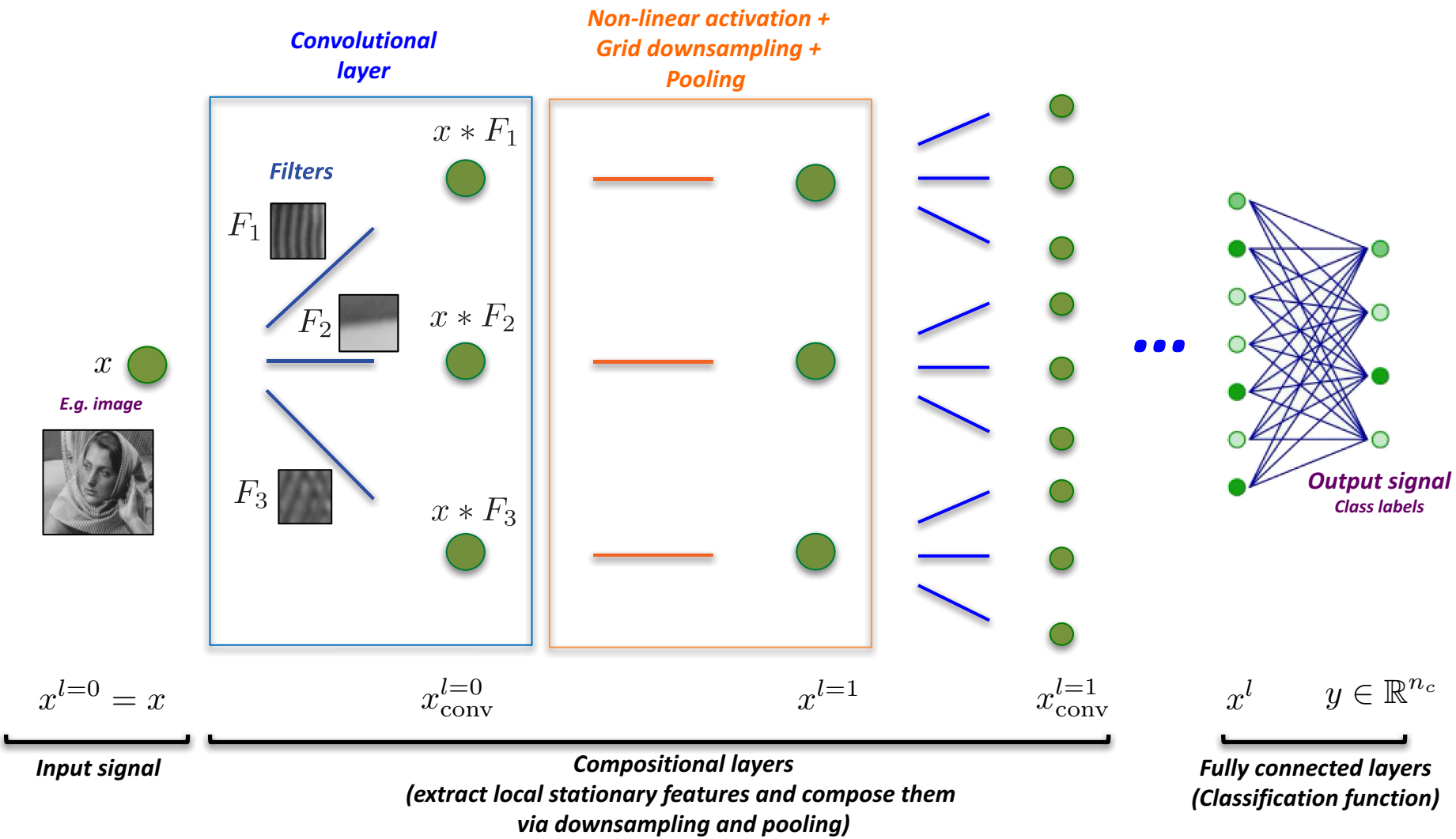
[Karpathy-FeiFei'15]

Self-driving cars

⇒ CNNs are best implemented by:

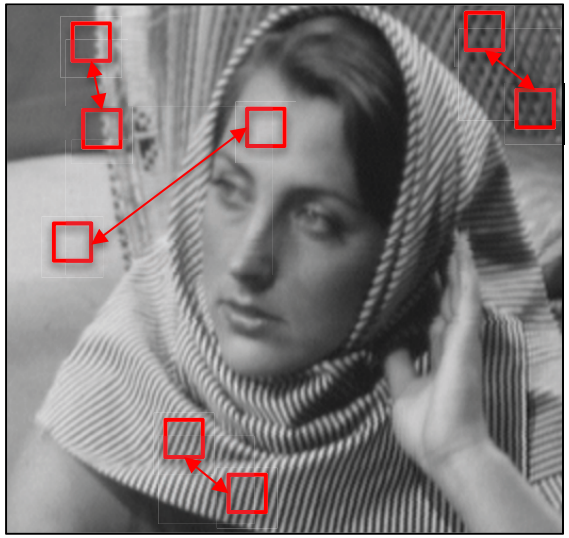
1. Google TensorFlow
2. Facebook Torch

Architecture of CNNs

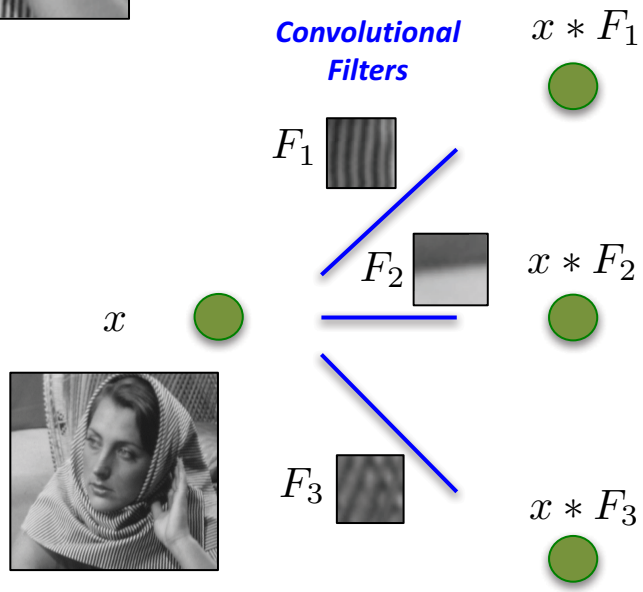


Convolutional Layer

- **Assumption:** Images are *locally stationary* ⇔ similar local patches are shared across the data domain:

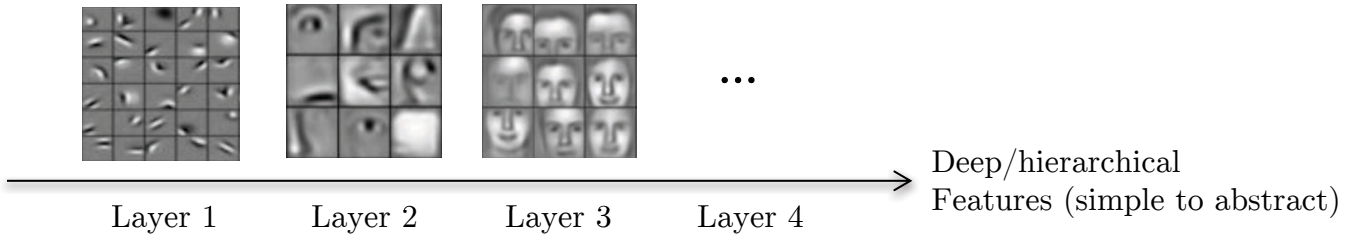


- **How to extract local stationary patterns?**
  Convolutional filters (filters with compact support)

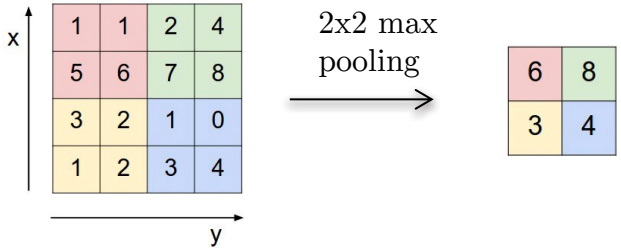


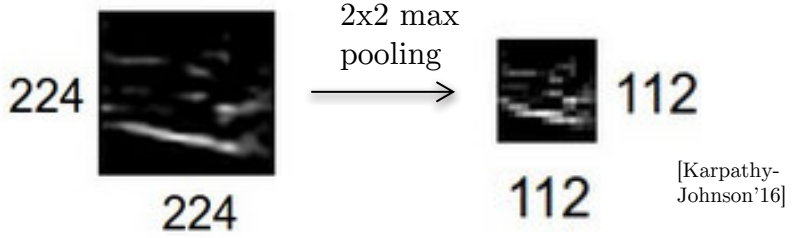
Compositional Layers

- **Assumption:** Images are *compositional* ⇔ local stationary patterns can be composed to form more abstract complex patterns:

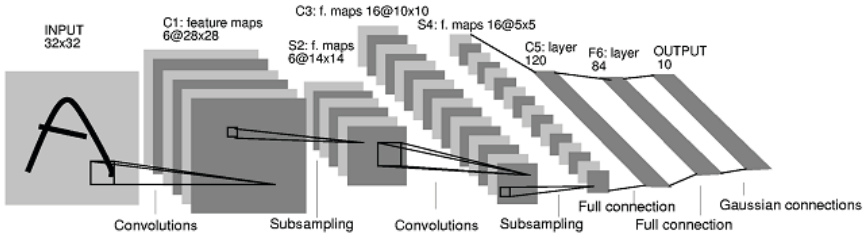


- **How to extract these hierarchical patterns?**
  Downsampling of data domain (image grid) with *pooling* (max, average).





- **Other advantage:** Reduce computational complexity while increasing #filters.



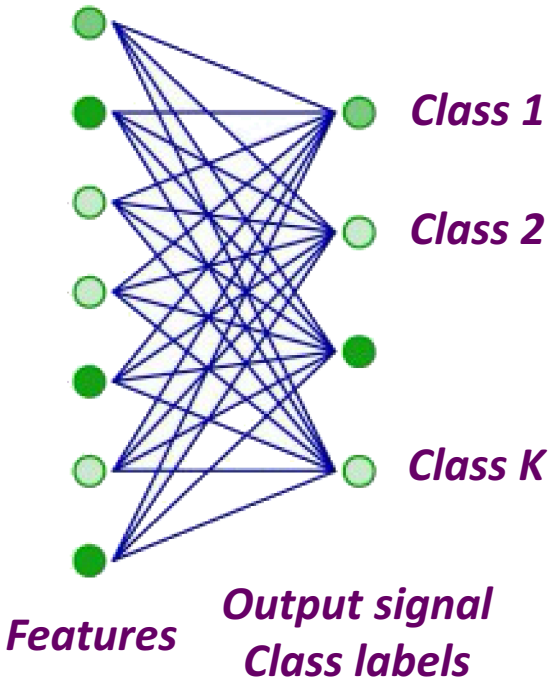
Fully Connected Layer

- **Classifier:** After extracting multiscale locally stationary features, feed them to a classification function.

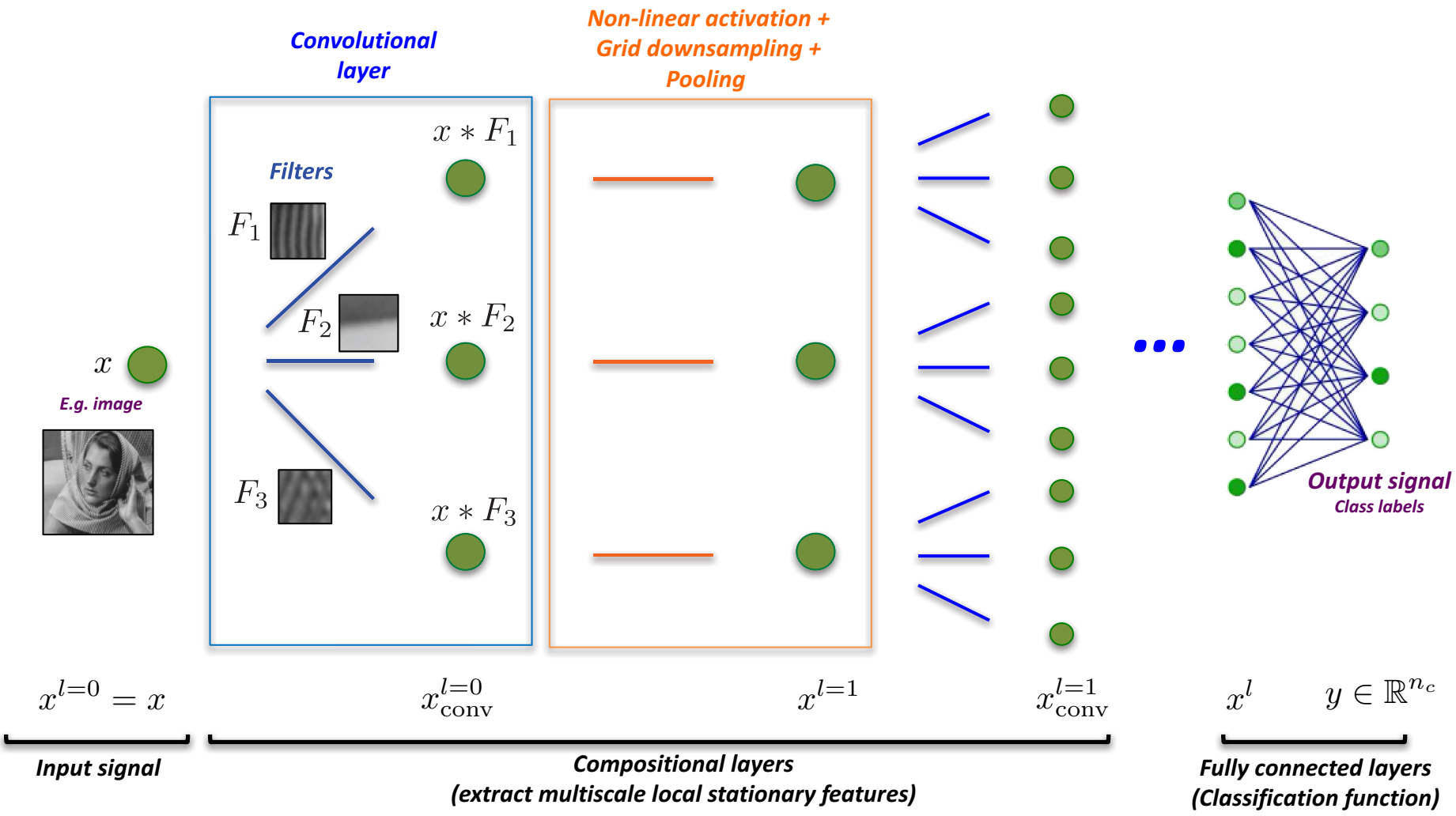
- **How to design a (linear) classifier?**

  Fully connected neural networks.

$$x_{\text{class}} = W x_{\text{feat}}$$



Architecture of CNNs



Outline

- Data Science and Deep Learning
  - *A brief history*
  - *What makes DL great*
- Convolutional Neural Networks
  - *Architecture*
- **Convolutional Neural Networks on Graphs**
  - *Non-Euclidean data*
  - *Graph convolution, graph downsampling and pooling*
  - *Numerical experiments*
- Natural Language Processing
- Recommender Systems
- Conclusion

CNNs are Great but...

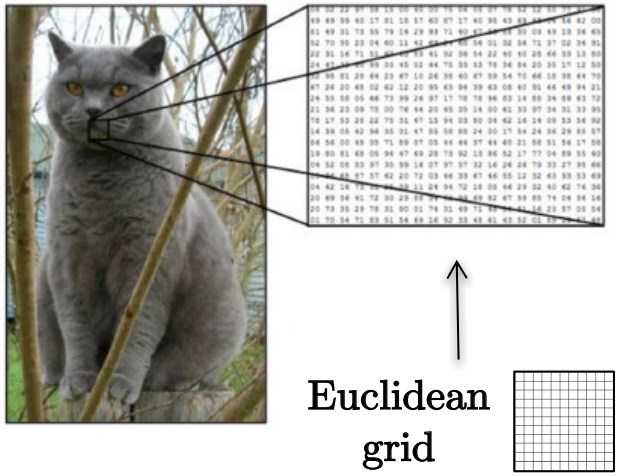
- **CNNs are designed for data lying on *Euclidean* grids:**

  (1) Convolution on Euclidean grids (FFT)

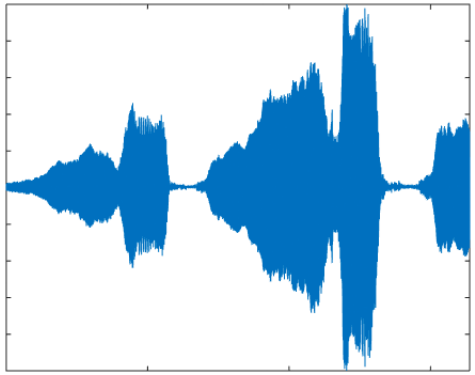
  (2) Downsampling on Euclidean grids

  (3) Pooling on Euclidean grids

  ⇒ *Everything mathematically well defined and computationally fast!*



Images (2D, 3D)
videos (2+1D)



Sound (1D)

- **But not all data lie on Euclidean grids.**

Non-Euclidean Data

- **Examples of non-Euclidean data:**

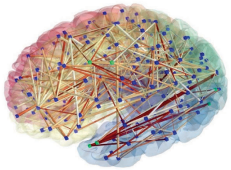
  (1) Social networks (Facebook, Twitter)

  (2) Biological networks (gene, brain connectivity)

  (3) Communication networks (road, wireless, internet)

  (4) Word semantic networks (NLP, translation, Q&A)

Social networks



Brain structure

Telecommunication networks

=

Graph/network-structured data

- **Main challenges:**

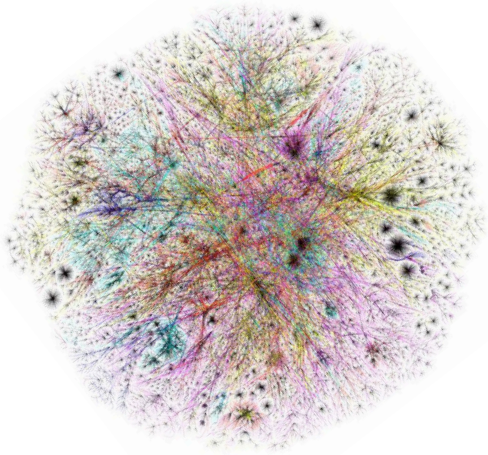
  1. *How to define compositionality on graphs (i.e. convolution, downsampling and pooling on graphs)?*
  2. *And how to make them numerically fast? (as fast as standard CNNs)*

- **Current solution:** Map graph-structured data to regular/Euclidean grids with e.g. kernel methods and apply standard CNNs. But handcrafting the mapping is (to my opinion) against CNN principle.

CNNs for Graph-Structured Data
[Defferrard-B-Vandergheynst'16]

- **Our contribution:** Generalizing CNNs to any graph-structured data with ***same linear time complexity*** as standard CNNs.

$\Rightarrow$



- **What mathematical tools?**
  1. *Graph spectral theory,*
  2. *Graph clustering.*

- This research was published at NIPS'16.

  Code/data are available on GitHub: https://github.com/mdeff/cnn_graph

Related Works

- **Categories of graph CNNs:**

  (1) Spatial approach

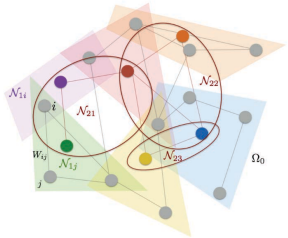
  (2) Spectral (Fourier) approach

- **Spatial approach:**
  - *Local reception fields* [Coates-Ng'11, Gregor-LeCun'10]:

    Find compact groups of similar features, but no defined convolution.
  - *Locally Connected Networks* [Bruna-Zaremba-Szlam-LeCun'13]:

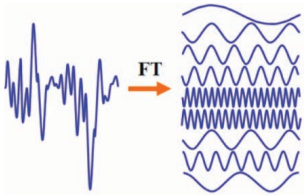
    Exploit multiresolution structure of graphs, but no defined convolution.
  - *Haar Scattering on Graphs* [Chen-Cheng-Mallat'14]:

    Use scattering transform on graphs to invariant signal descriptors.
  - *ShapeNet* [Bronstein-et.al.'15'16]:
    Generalization of CNNs to 3D-meshes. Convolution well-defined in these smooth low-dimensional non-Euclidean spaces. Handle multiple graphs. Obtained state-of-the-art results for 3D shape recognition.

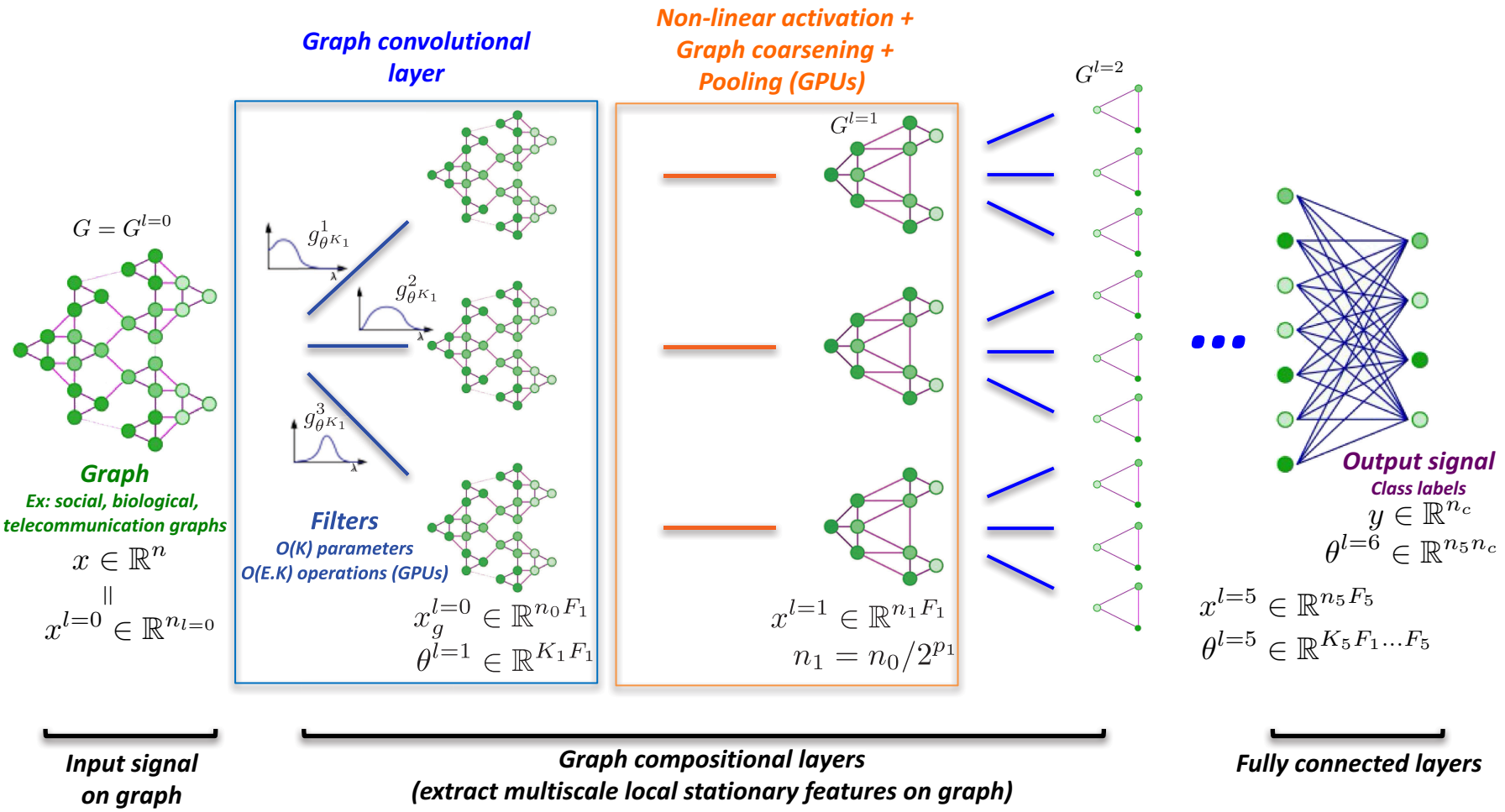


- **Spectral approach:**
  - *Deep Spectral Networks* [Henaff-Bruna-LeCun'15]:

    Related to this work. We will compare later.



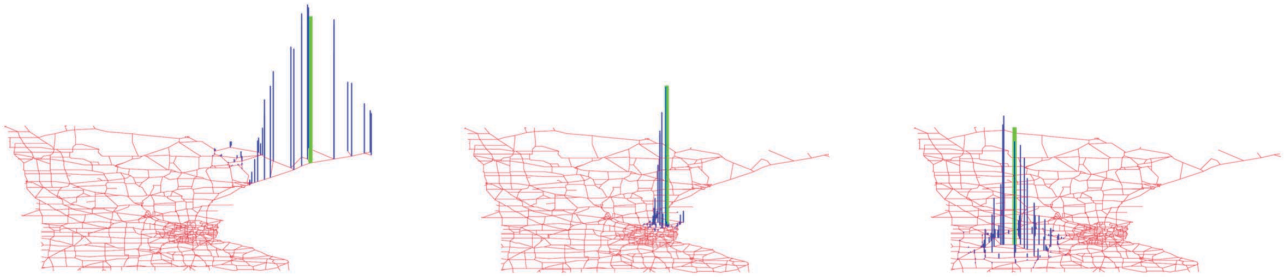
Architecture of Graph CNNs



Redefining Standard Layers

- **Step 1. Convolutional layer: How to perform convolution on graphs?**

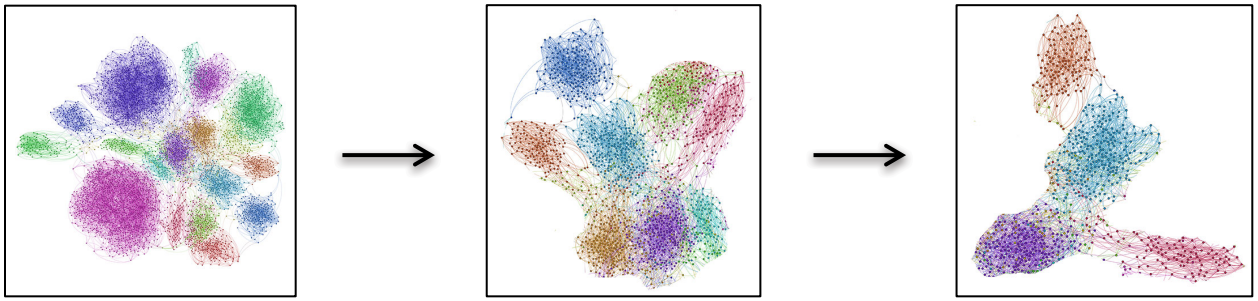
  $\Rightarrow$ *Graph spectral theory*



- **Step 2. Compositional layer: How to "downsample" and pool on graphs?**

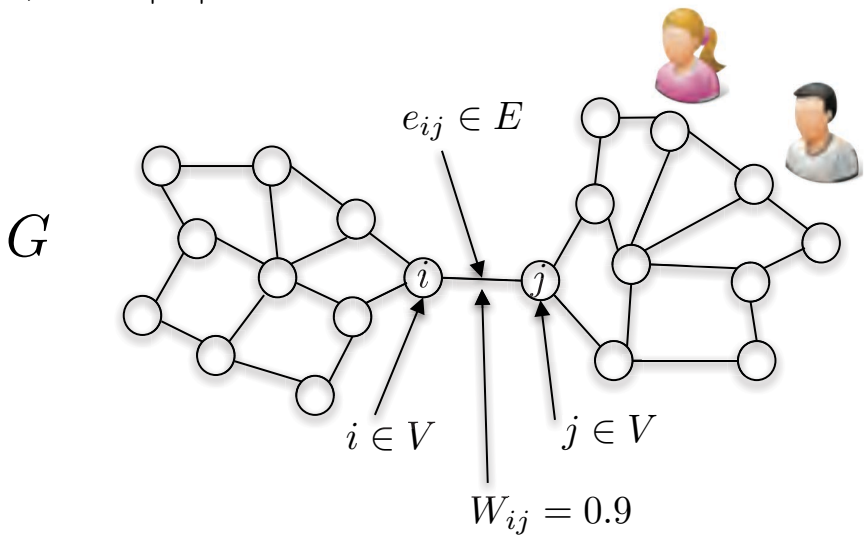
  $\Rightarrow$ *Graph clustering*

  $\Rightarrow$ *Graph pooling*



Step 1. Graph Convolutional Layer

- **Graphs:** $G=(V,E,W)$, with V set of vertices, E set of edges, W similarity matrix, and $|V|=n$.



- **Graph Laplacian** *(core operator to spectral graph theory [1])*: 2nd order derivative operator on graphs

$$L = D - W$$
$$\text{with} \quad D = \text{diag}\Big(\sum_j W_{ij}\Big)$$

[1] Chung, 1997

Graph Fourier Transform

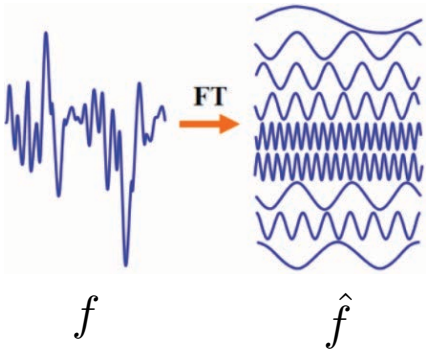
- **Fourier transform on graphs [2]:**

Eigen-decomposition of L: $L = U \Lambda U^T$

Fourier modes ***Graph frequencies***

The Graph Fourier Transform of $f \in \mathbb{R}^n$ is

$$\mathcal{F}_G f = \hat{f} = U^T f$$



The inverse GFT is defined as:

$$\mathcal{F}_G^{-1} \hat{f} = U \hat{f} = U U^T f = f,$$

[2] Hammond, Vandergheynst, Gribonval, 2011

Convolution on Graphs

- **Convolution on graphs is easy in the Fourier domain**:

$$f *_G g = \mathcal{F}_G^{-1}\left(\mathcal{F}_G f \odot \mathcal{F}_G g\right)$$

Note (for later): It is also convenient to see:

$$f *_G g = \hat{g}(L) f,$$

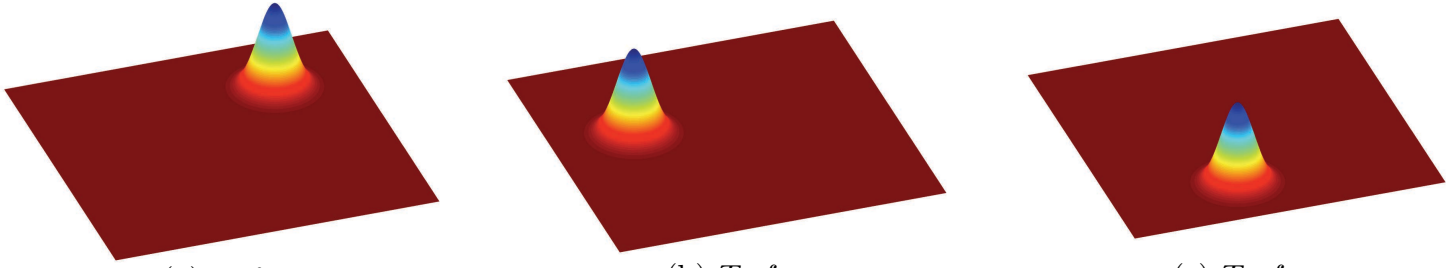
- **[Bonus] Translation on graphs:** A signal f defined on the graph can be translated to any vertex i using the graph convolution operator [3]:

$$T_i f := f *_G \delta_i,$$

[3] Shuman, Ricaud, Vandergheynst, 2016

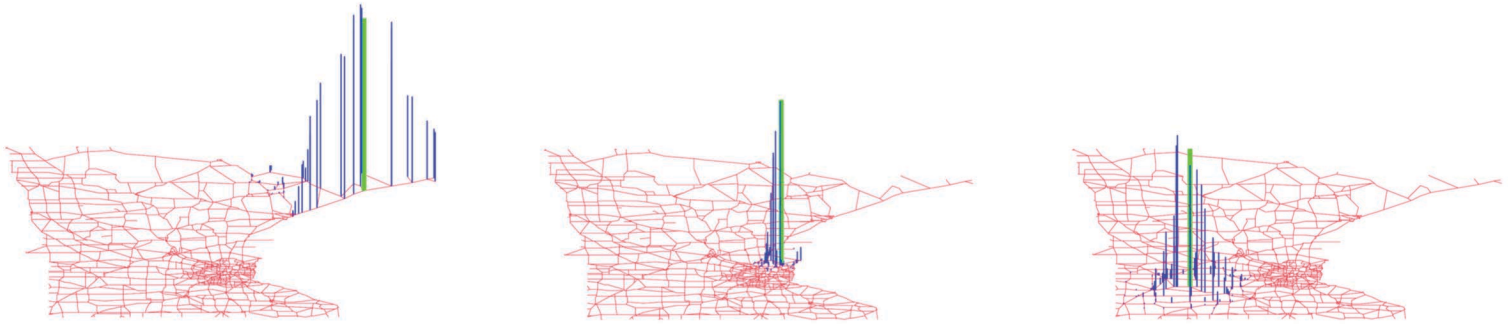
Translation: Euclidean vs Non-Euclidean Domains

- **Translated signals in the Euclidean domain:**



(a) $T_s f$ (b) $T_{s'} f$ (c) $T_{s''} f$

- **Translated signals in a non-Euclidean domain:**

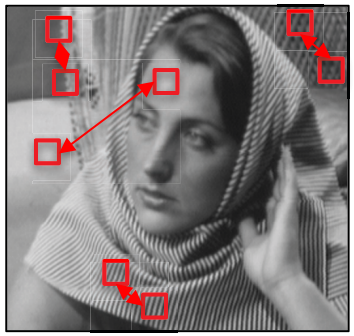


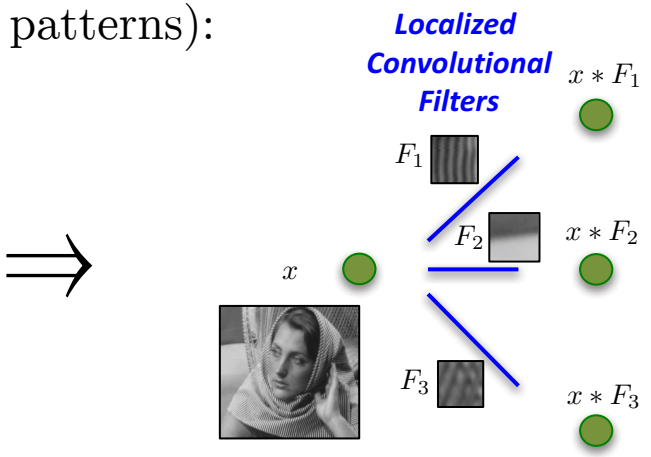
(d) $T_i f$ (e) $T_{i'} f$ (f) $T_{i''} f$

[Shuman-Ricaud-Vandergheynst'16]

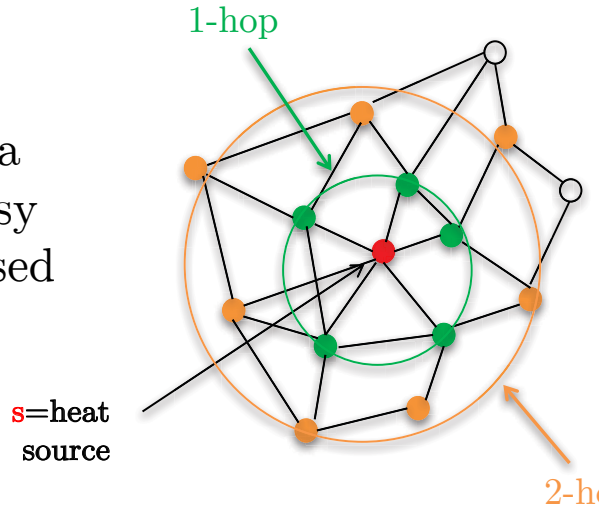
Localized Filters on Graphs

- As in standard CNNs, we need to define **localized** filters **on graphs** (to find local stationary patterns):





- **Key observation:** Graph Laplacian operation increases the support of a Dirac function by 1-hop ⇒ it is easy to control the size of Laplacian-based filters!

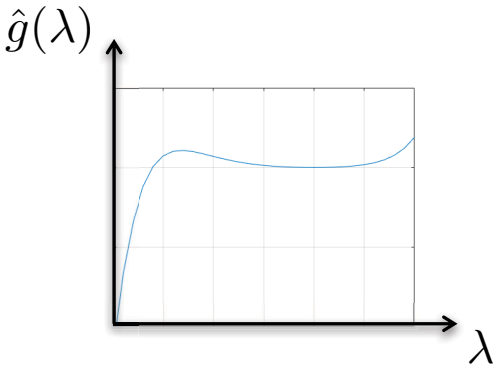


Heat propagation with Laplacian:

$$s^{t+1} = Ls^t$$

Localized Filters on Graphs

- **Spectral kernels as Laplacian polynomial [2]:**

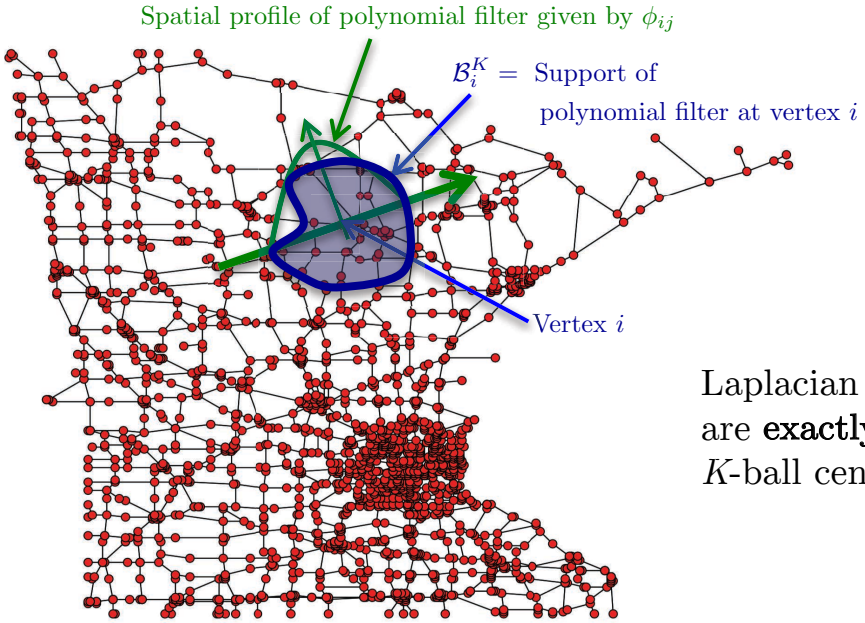


$$\hat{g}(\lambda_l) = \hat{p}_K(\lambda_l) := \sum_{k=0}^{K} a_k \lambda_l^k$$

Corollary 1. *Consider the function ϕ_{ij} such that*

$$\phi_{ij} = \big(T_i p_K\big)(j) = \big(p_K *_G \delta_i\big)(j) = \big(\hat{p}_K(L)\delta_i\big)(j) = \big(\sum_{k=0}^{K} a_k L^k \delta_i\big)(j).$$

Then $\phi_{ij} = 0 \quad if \quad d_G(i,j) > K.$



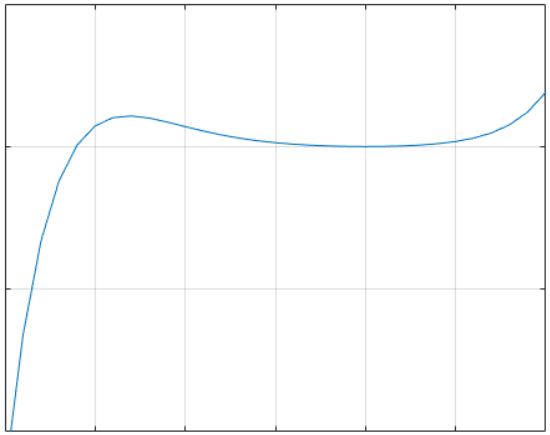
Laplacian spectral kernels are **exactly** localized in a K-ball centered at vertex i.

Localized Filters on Graphs

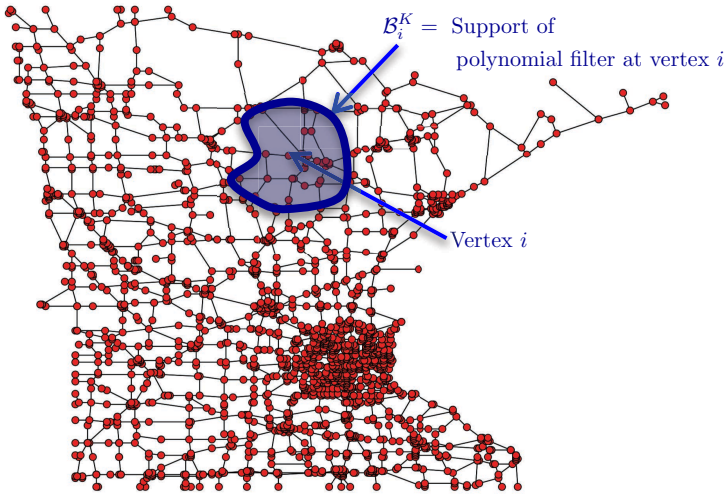
Corollary 2. *Localized filters on graphs are defined according to the principle:*

Frequency smoothness $\Leftrightarrow$ *Spatial graph localization*

This is obviously the Heisenberg's uncertainty principle extended to the graph setting. Recent papers have studied the uncertainty principle on graphs.



$\Leftrightarrow$



Frequency/spectral domain **Spatial/graph domain**

How fast is Convolution on Graphs?

- **Graph convolution:**

$$y \quad = \quad x *_G p_K = \hat{p}_K(L)x = \sum_{k=0}^{K} a_k L^k x$$

- **Complexity is linear: *O(n)***

  Denote $X_k = LX_{k-1}$ with $X_0 = x$ and rewrite $y = \Sigma_k a_k X_k$.
  Observe that all $\{X_k\}$ are generated by multiplication of the *sparse* L matrix and a vector $\Rightarrow$ Complexity is $O(EK)=O(n)$ for *sparse* graphs.

- **GPU parallel implementation:** Linear algebra operations can be done in parallel, allowing a fast GPU implementation of graph convolution.

Convolution on Graphs with Chebyshev

- **Monomial graph convolution is actually not optimal for learning:**

$$y \quad = \quad x *_G p_K = \hat{p}_K(L)x = \sum_{k=0}^{K} a_k L^k x$$

Monomial basis $\{1,x,x^2,\ldots,x^K\}$ do **not** form an **orthogonal** basis, which prevents learning good spectral filters.

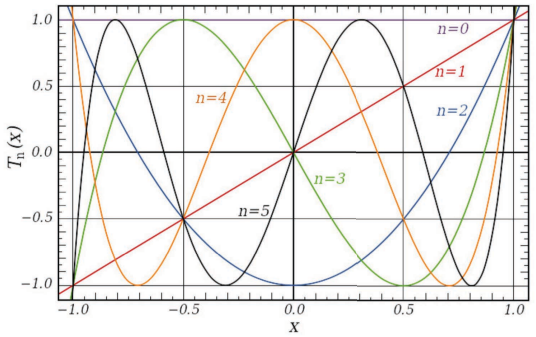
- **Graph convolution with (orthogonal) Chebyshev polynomials :**

$$y \quad = \quad x *_G q_K = \sum_{k=0}^{K} \theta_k T_k(L)x,$$

- **Complexity is also linear: *O(n)***

Denote $X_k = T_k(L)X_{k-1} = 2LX_{k-1} - X_{k-2}$ and rewrite $y = \Sigma_k \theta_k X_k$.

Complexity is $O(n)$ for *sparse* graphs.

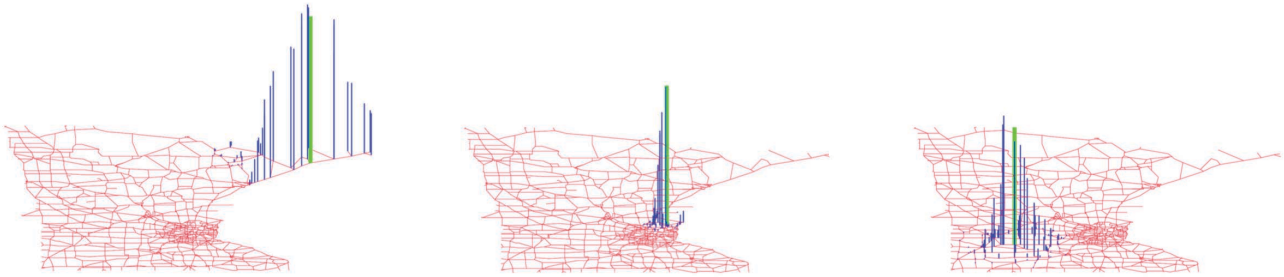


First six Chebyshev functions $\{T_0, T_1, T_2, T_3, T_4, T_5\}$.

Redefining Standard Layers

- **Step 1. Convolutional layer: How to perform convolution on graphs?**

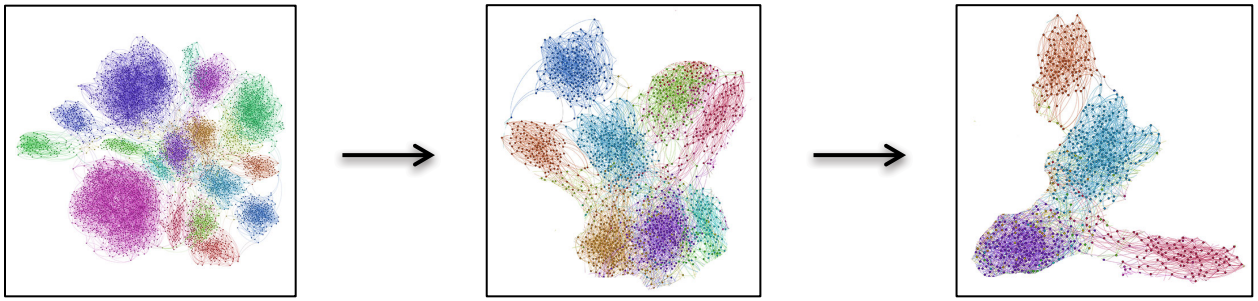
  $\Rightarrow$ *Graph spectral theory*



- **Step 2. Compositional layer: How to "downsample" and pool on graphs?**

  $\Rightarrow$ *Graph clustering*

  $\Rightarrow$ *Graph pooling*



Step 2. Graph "Downsampling"

- **Graph downsampling is graph coarsening, also graph clustering/partitioning.**
  This step is essential to compose lower-level features to more complex patterns.

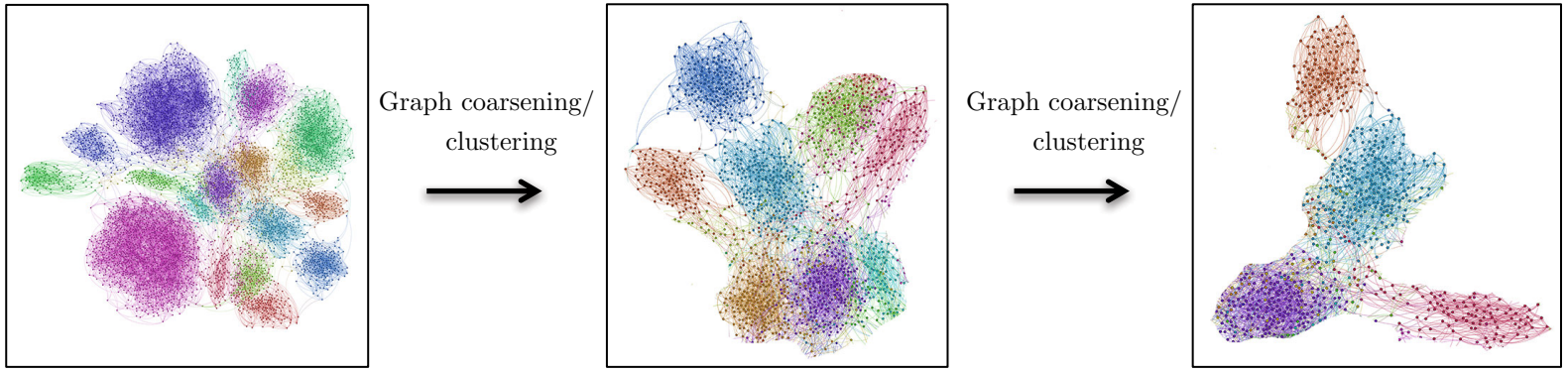
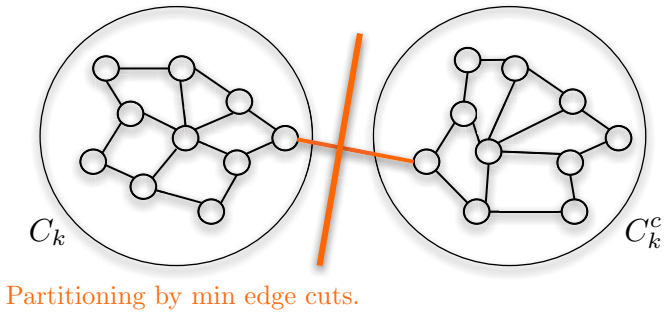


Fig. Illustration of graph coarsening.

- **Graph partitioning** is NP-hard ⇒ Approximation needed.

Graph Partitioning

- **Balanced Cuts [4]:** Two powerful measures of graph clustering are the *Normalized Cut* and *Normalized Association* defined as:

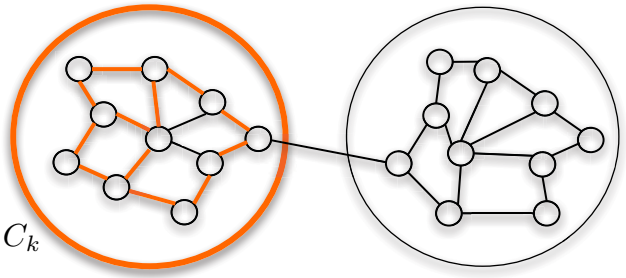


Partitioning by min edge cuts.

Normalized Cut:

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} \frac{\mathrm{Cut}(C_k, C_k^c)}{\mathrm{Vol}(C_k)}$$

⇕ Equivalence by complementarity



Partitioning by max vertex matching.

Normalized Association:

$$\max_{C_1,\ldots,C_K} \sum_{k=1}^{K} \frac{\mathrm{Assoc}(C_k)}{\mathrm{Vol}(C_k)}$$

Figure 5: Equivalence between NCut and NAssoc partitioning.

where $\mathrm{Cut}(A,B) := \sum_{i\in A, j\in B} W_{ij}$, $\mathrm{Assoc}(A) := \sum_{i\in A, i\in B} W_{ij}$, $\mathrm{Vol}(A) := \sum_{i\in A, j\in B} d_i$, and $d_i := \sum_{j\in V} W_{ij}$ is the degree of vertex i.

[4] Shi, Malik, 2000

Graclus Graph Partitioning

- **Graclus [5]:** It is a greedy (fast) technique that computes clusters that locally maximize the Normalized Association.

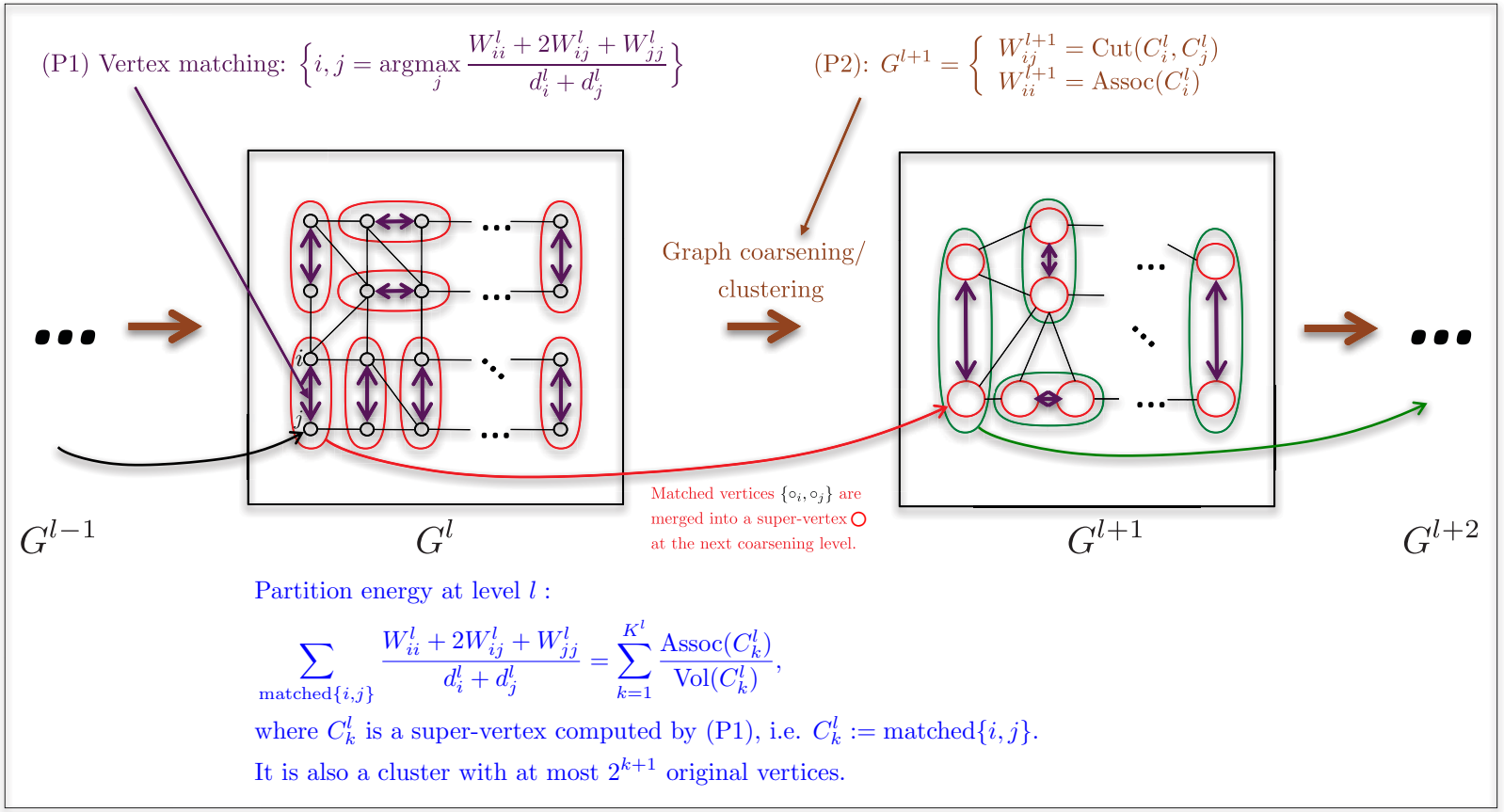


Figure 6: Graph coarsening with Graclus. Graclus proceeds by two successive steps: (P1) Vertex matching, and (P2) Graph coarsening. These two steps provide a local solution to the Normalized Association clustering problem at each coarsening level l.

[5] Dhillon, Guan, Kulis, 2007

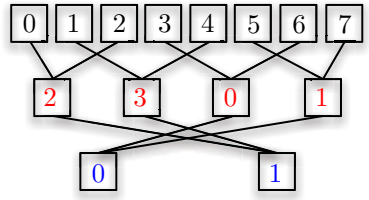
Fast Graph Pooling

- **Graph pooling** such as **max** pooling or **average** pooling transfer feature value to the next layer.

- **Unstructured pooling is (highly) inefficient**: The graph and all its coarsened versions require to store a table with all matched vertices.

  ⇒ Memory consuming and not (easily) parallelizable.

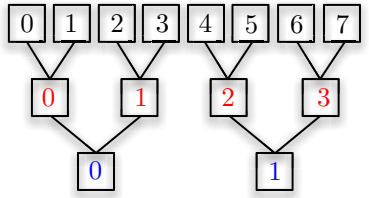
Graph pooling:



Unstructured arrangement of vertices

- **Structured pooling can be as efficient as a 1D grid pooling (GPUs):** Use a ***binary tree arrangement*** of the nodes such that adjacent nodes are hierarchically merged at the next coarser level.

Graph pooling:



Binary tree arrangement of vertices

Reindexing Process

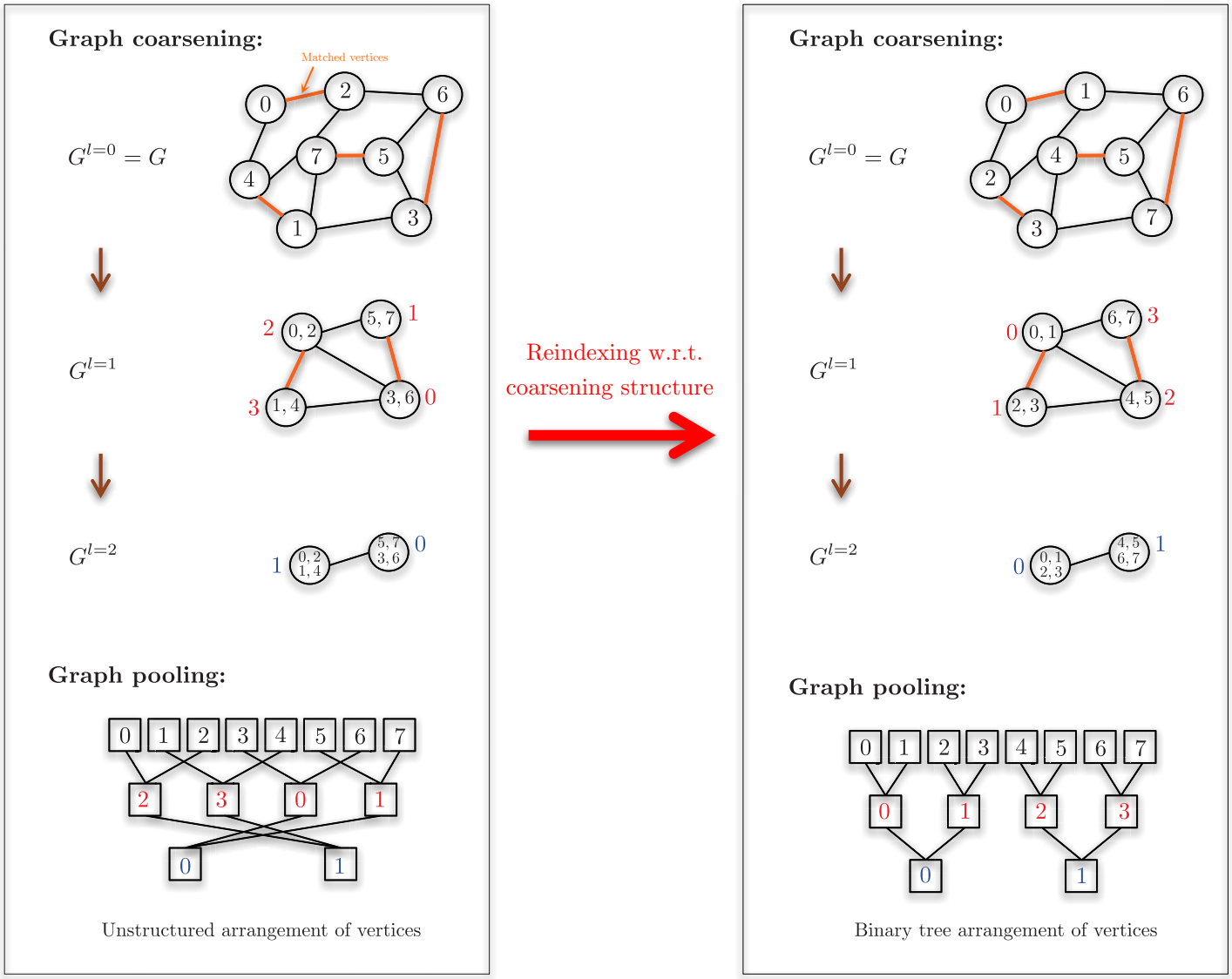
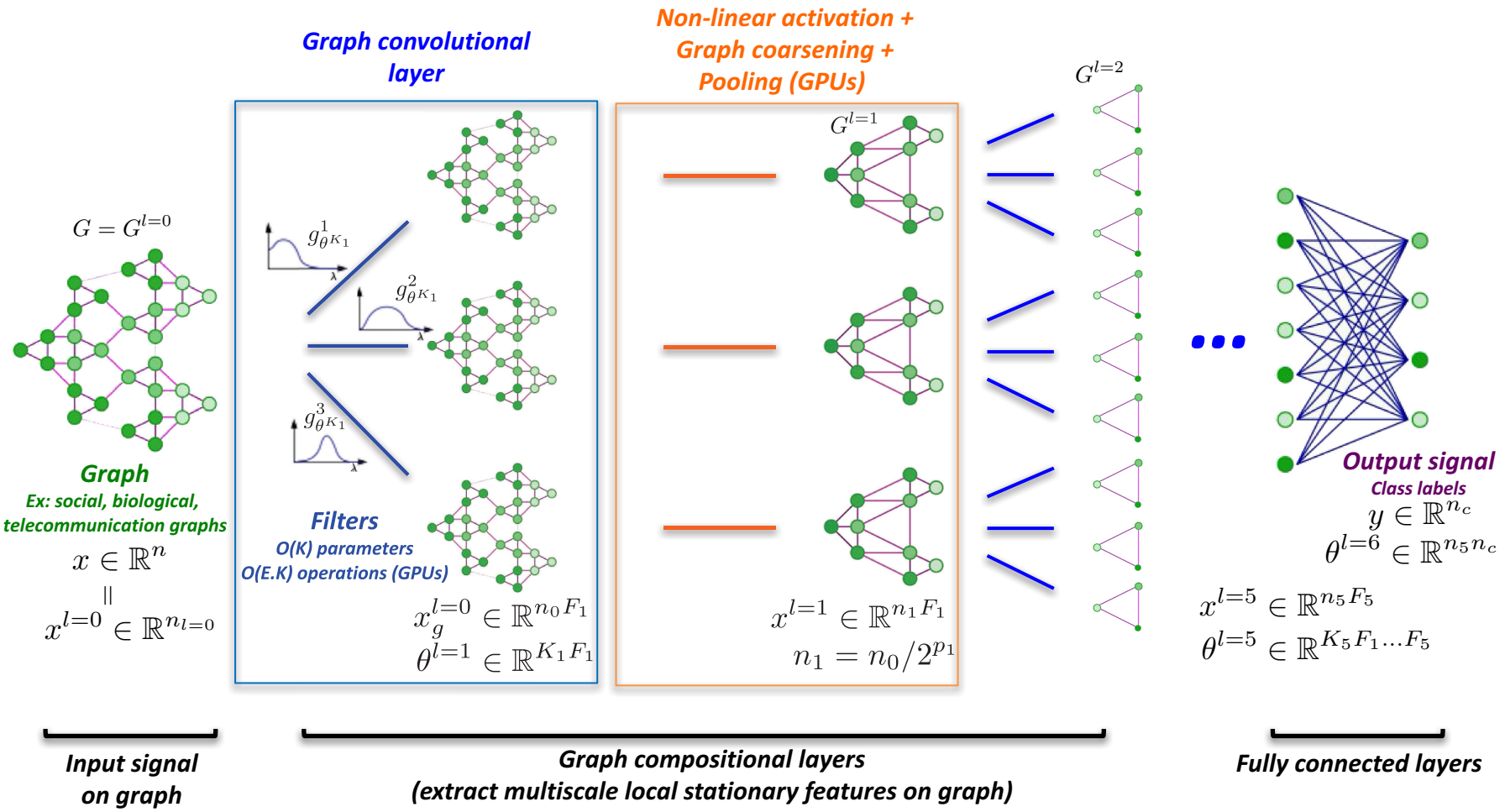


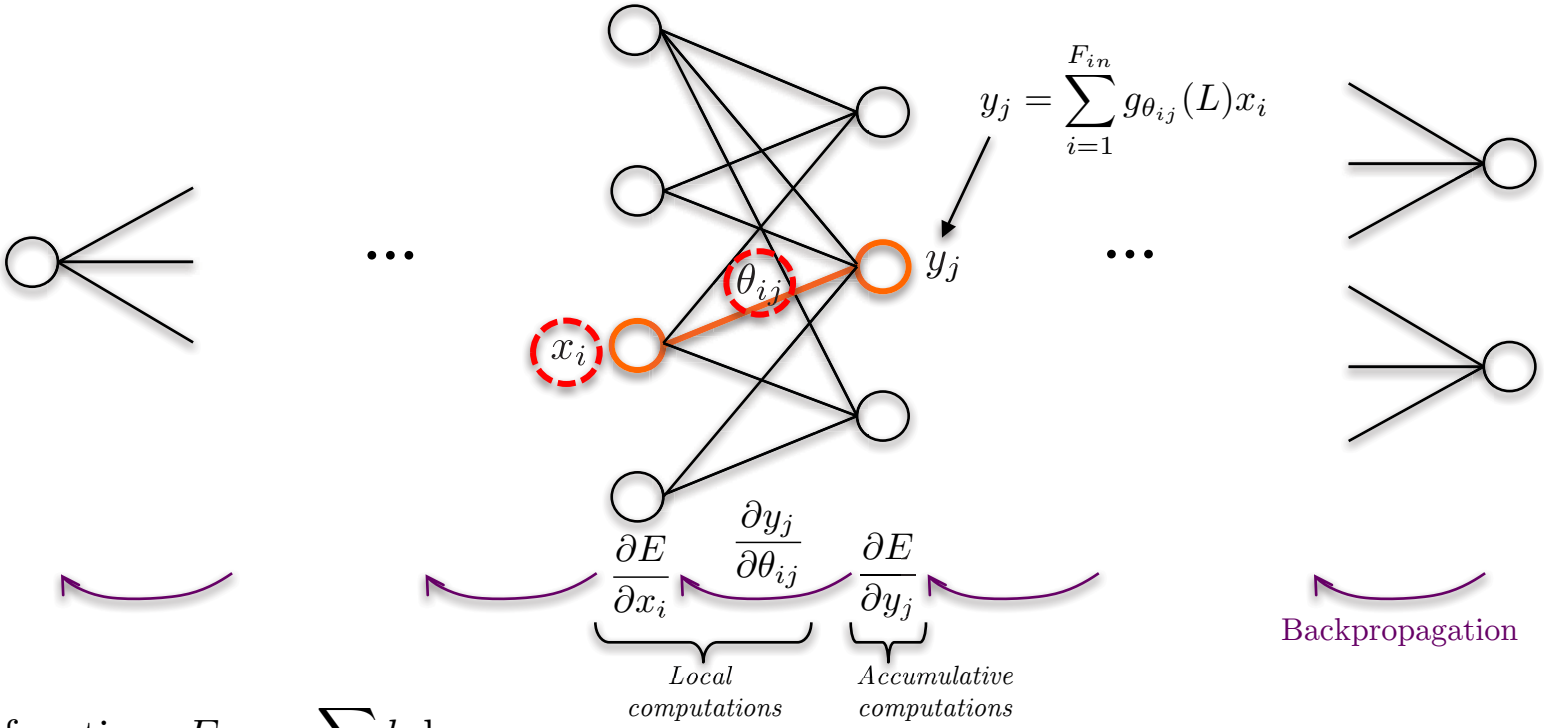
Figure 7: Fast graph pooling using graph coarsening structure. The binary tree arrangement of vertices allows a very efficient pooling on graphs, as fast as a regular 1D Euclidean grid pooling.

Architecture of Graph CNNs



Optimization

- **Backpropagation [6]** = Chain rule applied to the neurons at each layer.



Loss function: $E = -\sum_{s \in S} l_s \log y_s$

Gradient descents: $\begin{cases} \theta_{ij}^{n+1} = \theta_{ij}^n - \tau . \frac{\partial E}{\partial \theta_{ij}} \\ x_i^{n+1} = x_i^n - \tau . \frac{\partial E}{\partial x_i} \end{cases}$

Local gradients:

$$\frac{\partial E}{\partial \theta_{ij}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \theta_{ij}} = \sum_{s \in S} \left[X_{0,s}, ..., X_{K-1,s} \right]^T \frac{\partial E}{\partial y_{j,s}}$$

$$\frac{\partial E}{\partial x_i} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial x_i} = \sum_{j=1}^{F_{out}} g_{\theta_{ij}}(L) \frac{\partial E}{\partial y_j}$$

[6] Werbos 1982 and Rumelhart, Hinton, Williams, 1985

Numerical Experiments

- **Platforms and hardware:** All experiments are carried out with:
  1. *TensorFlow (Google AI software) [7]*
  2. *GPU NVIDIA K40*

- **Reproducibility: Code/data are on GitHub.**

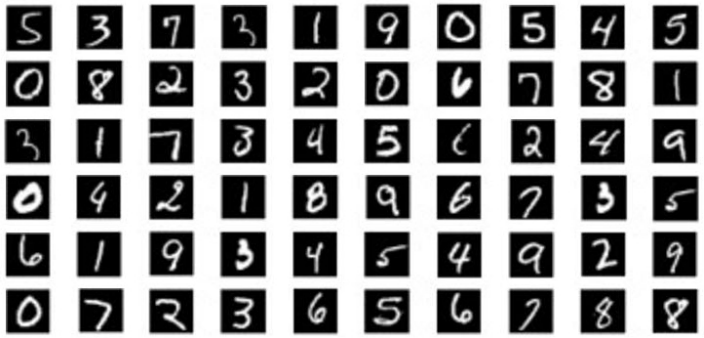
- **Goals of the experiments:**
  1. *Check linear complexity*
  2. *Compare with [LeCun-et al.'15]*

[7] Abadi-et.al. 2015

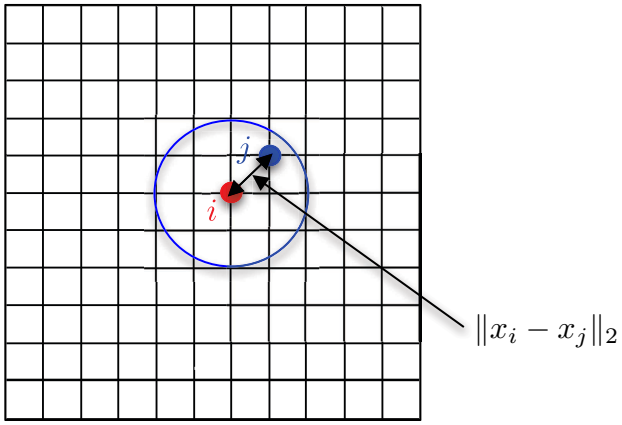
Graph CNNs for Image Classification

- **MNIST** is the most popular dataset in deep learning [8].

  It has of 70,000 images represented on a 2D grid of size 28x28 (dim=28^2=784) of handwritten digits, from 0 to 9.



- **Graph:** A k-NN graph (k=8) of the Euclidean grid:



$$W_{ij} = e^{-\|x_i - x_j\|_2^2/\sigma}$$

[8] LeCun, Bottou, Bengio, 1998

Performances

- **Results:** Classification accuracies

Algorithm	Accuracy
Linear SVM	91.76
Softmax	92.36
CNNs [LeNet5]	99.33
graph CNNs: CN32-P4-CN64-P4-FC512-softmax	99.18

- **CPU vs. GPU:** A GPU implementation of graph CNNs is **8x faster** than a CPU implementation, same order of speed up as standard CNNs.

Architecture	Time	Speedup
CNNs with CPU	210ms	-
CNNs with GPU NVIDIA K40c	31ms	6.77x
graph CNNs with CPU	1600ms	-
graph CNNs with GPU NVIDIA K40c	200ms	8.00x

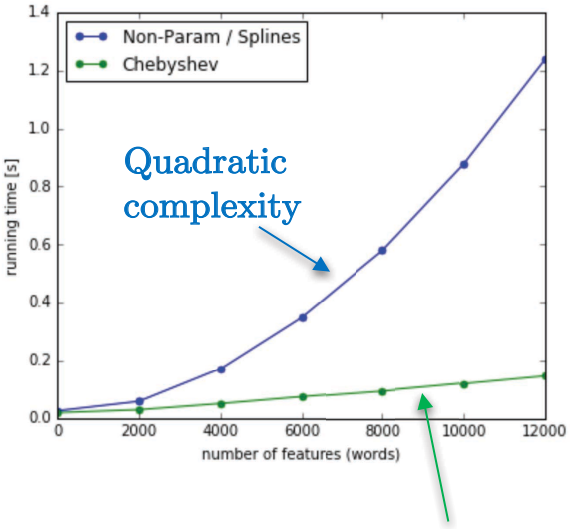
Numerical Linear Complexity

- **Advantages over [Henaff-Bruna-LeCun'15]:**

(1) Computational complexity $O(n)$ vs. $O(n^2)$:

$$y = x *_G h_K = \hat{h}_K(L)x = U(\hat{h}_K(\Lambda)(U^T x)),$$

where $\hat{h}_K$ are spectral filters based on K^{th}-order splines, Λ, U are the eigenvalue, eigenvector matrices of the graph Laplacian L, and U^T, U are full $O(n^2)$ matrices.

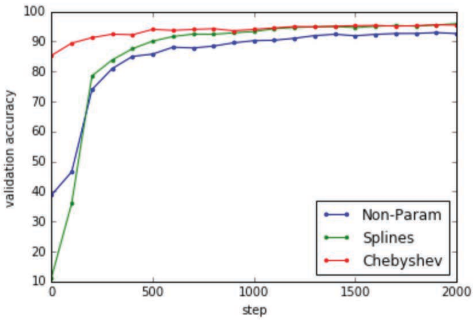


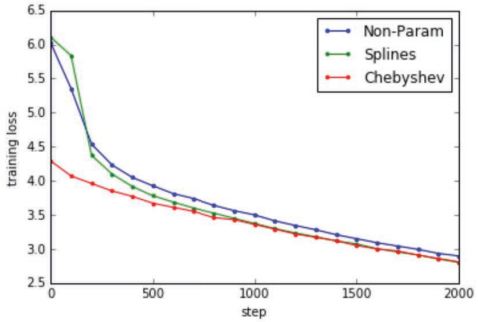
(2) no EVD $O(n^3)$ vs. [HBL15]

(3) Accuracy:

Architecture	Accuracy
CN10 [HBL15]	97.26
CN10 [Ours]	97.48
CN32-P4-CN64-P4-FC512 [HBL15]	97.75
CN32-P4-CN64-P4-FC512 [Ours]	99.18

(4) Learn faster:





Graph CNNs as a Tool for Data Analysis?

- **Standard CNN is the Swiss knife of Computer Vision.**

  $\Rightarrow$ Graph CNNs may have the same potential for any (un)structured data.

- **PI' works:**

  1. Recurrent neural networks on graphs for NLP [Seo-Defferrard-Vandergheynst-B, Structured Sequence Modeling with Graph Convolutional Recurrent Networks'16]

  2. Recommender systems [Monti-Bronstein-B, Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks'17] – state-of-the-art

- **Recent works that build on graph CNNs:**

  1. Brain analysis [Distance Metric Learning using Graph Convolutional Networks: Application to Functional Brain Networks'17]

  2. Temporal time series analysis [Dynamic Graph Convolutional Networks'17]

  3. Traffic Control [Deep Deterministic Policy Gradient for Urban Traffic Light Control'17]

  4. Chemistry [Neural Message Passing for Quantum Chemistry'17]

Outline

- Data Science and Deep Learning
  - *A brief history*
  - *What makes DL great*
- Convolutional Neural Networks
  - *Architecture*
- Convolutional Neural Networks on Graphs
  - *Non-Euclidean data*
  - *Graph convolution, graph downsampling and pooling*
  - *Numerical experiments*
- **Natural Language Processing**
- Recommender Systems
- Conclusion

Natural Language Processing

- **NLP problem:**

  Input: A sequence of words

  Yesterday I went to the beach and I saw a ...

  Output: Probability of the next word →

next word	probability
dolphin	11.5%
boat	8.2%
⋮	
elephant	0.1%
tiger	0.1%
⋮	
cooking	0.0%
with	0.0%

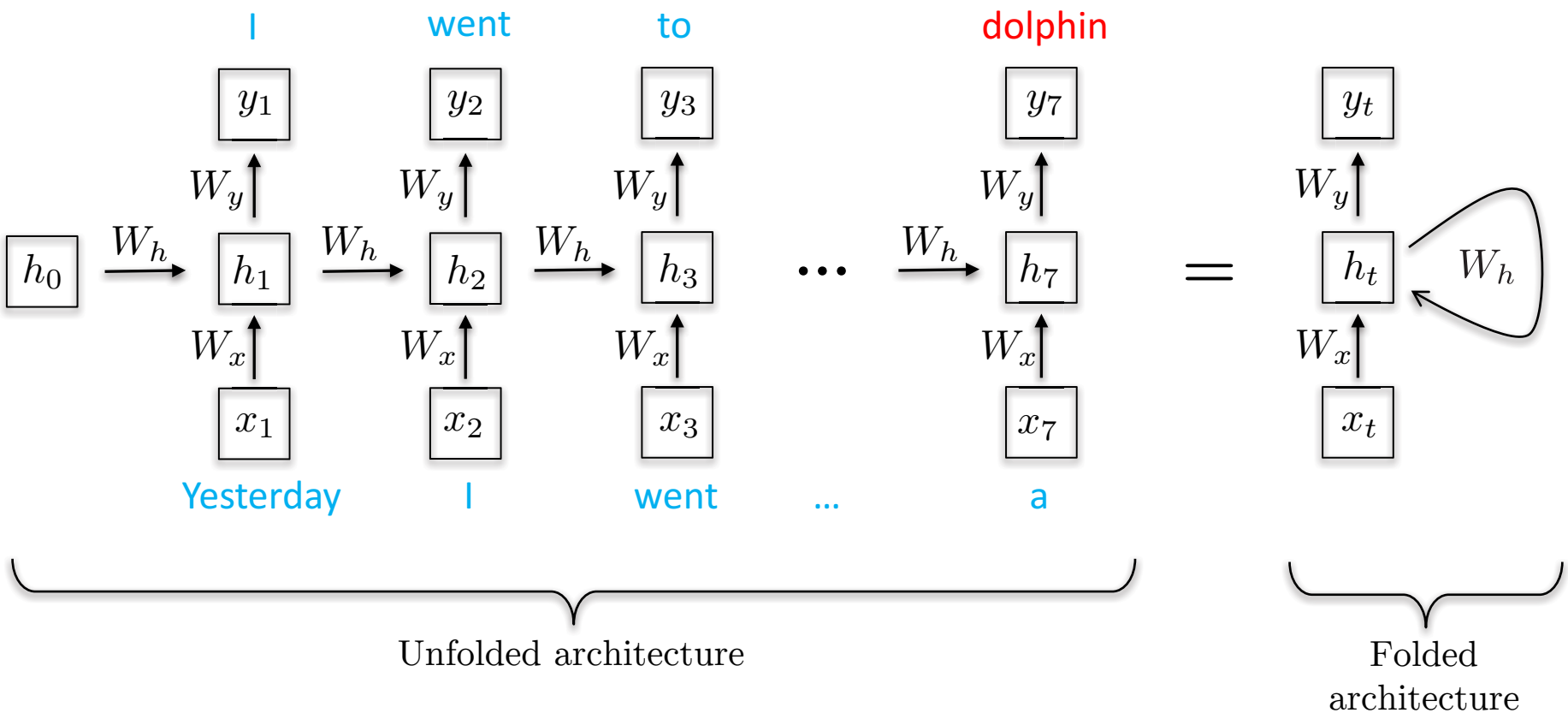
- **State-of-the-art solution since 2015:** Learn to predict next word with RNN.

- **Google's Neural Machine Translation system (Sept 2016):**

  "Today we announce the Google Neural Machine Translation system (GNMT), which utilizes state-of-the-art training techniques to achieve the largest improvements to date for machine translation quality."

RNN for Natural Language Processing

- **Vanilla RNN architecture for NLP:**

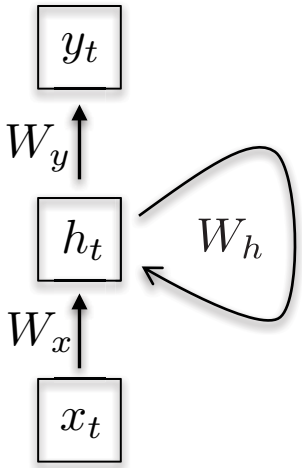


x_t : Input at t
h_t : Memory vector at t
y_t : Probability of next word at t

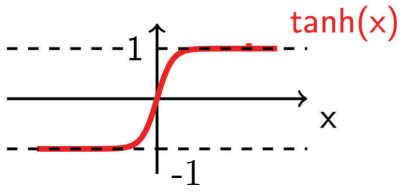
W_x : Embedding matrix
W_h : Memory matrix
W_y : Decoding matrix

Vanilla RNN

- **Iterative scheme:**



$\Rightarrow$



$$h_t = \tanh(W_h h_t + W_x x_t)$$
$$y_t = \text{softmax}(W_y h_t)$$

$$\text{Softmax}\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}} \\ \frac{e^{x_2}}{e^{x_1}+e^{x_2}+e^{x_3}} \\ \frac{e^{x_3}}{e^{x_1}+e^{x_2}+e^{x_3}} \end{bmatrix}$$

- **Major shortcoming:**

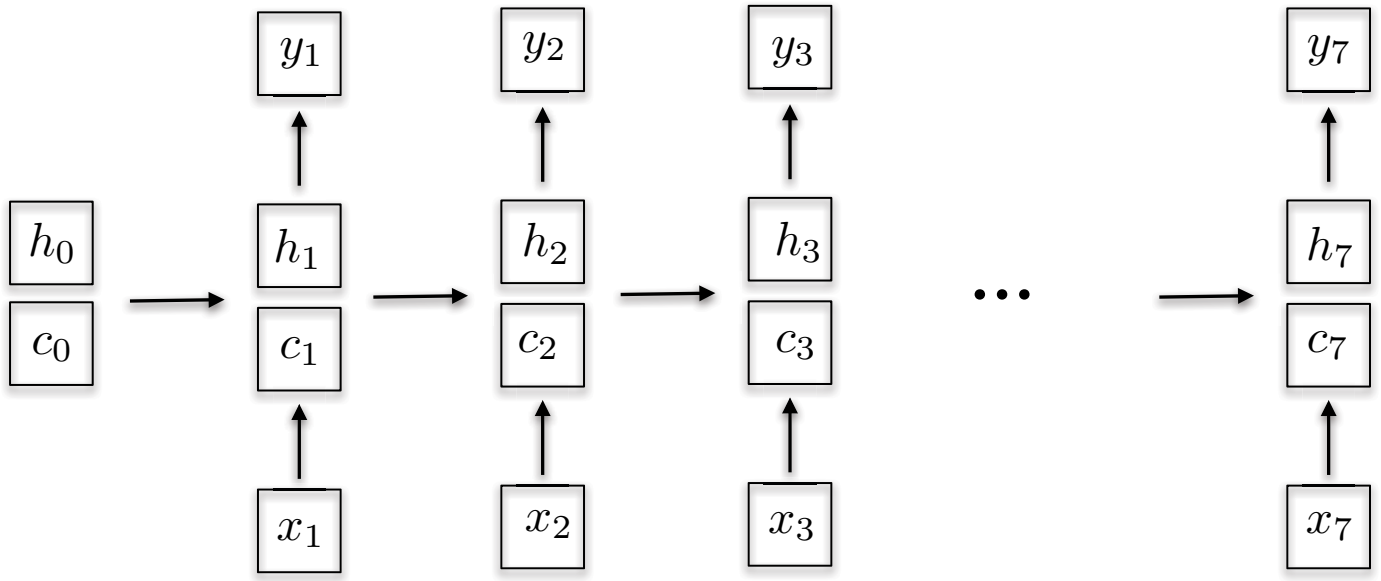
  VRNN do not learn long-term decencies

  Vanishing gradient problem (largest eigenvalue of W_h is less than 1)

$$\textbf{Ex. Softmax}\left(\begin{bmatrix} -1.5 \\ 3 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} .01 \\ .87 \\ .12 \end{bmatrix}$$

Long Short-Term Memory Networks
[Hochreiter-Schmidhuber'97]

- **LSTM architecture**



x_t : Input at t
h_t : *Short-term* memory vector at t
c_t : *Long-term* memory vector at t
y_t : Probability of next word at t

Non-Linear Dynamics

- **Iterative scheme:**

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(z_t)$$
$$h_t = o_t \odot \tanh(c_t)$$
$$y_t = \text{softmax}(W_y h_t)$$

with

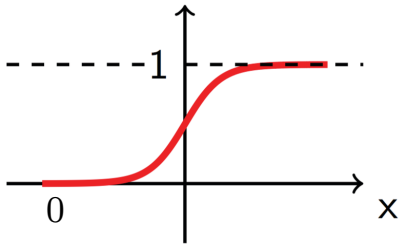
$$i_t = \sigma(W_h^i h_t + W_x^i x_t)$$ **Input gate**

$$f_t = \sigma(W_h^f h_t + W_x^f x_t)$$ **Forget gate (f=0,1)**

$$o_t = \sigma(W_h^o h_t + W_x^o x_t)$$ **Output gate**

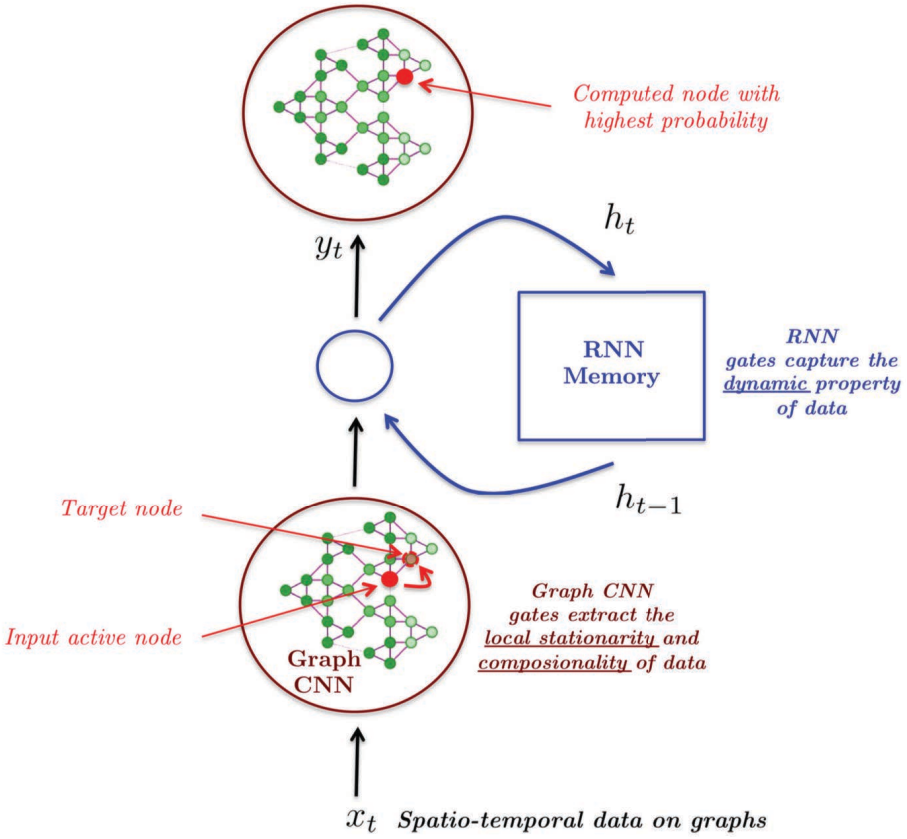
$$z_t = \sigma(W_h^z h_t + W_x^z x_t)$$

Sigmoid function:
$\sigma(x) = \frac{1}{1+e^{-x}}$



LSTM on Graphs
[Seo-Defferrard-Vandergheynst-B'16]

- **Combine CNN + LSTM:**
  - *CNN analyzes static information from data; finds multiscale local stationary patterns.*
  - *LSTM identifies data dynamics; non-linear temporal properties of data.*



- **Graph CNN:** Observe that words form a space of topics that can be modeled by graphs of word relationships.

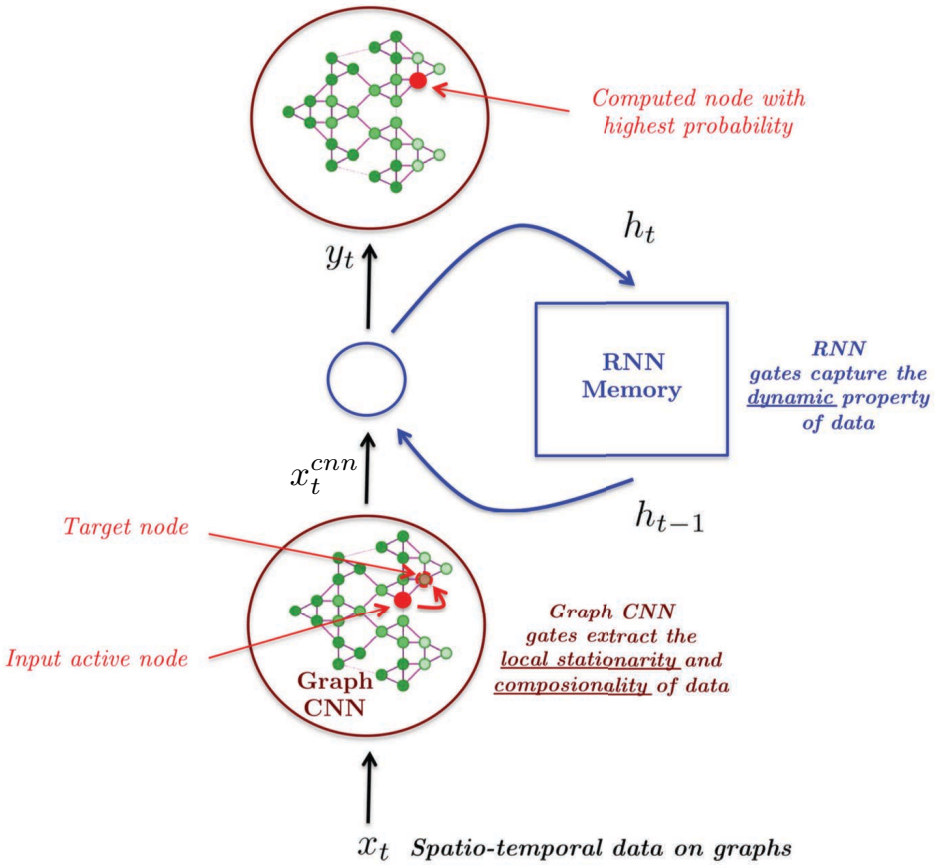
Iterative scheme

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(z_t)$$
$$h_t = o_t \odot \tanh(c_t)$$
$$y_t = \text{softmax}(W_y h_t)$$

with

$$i_t = \sigma(W_h^i h_t + W_c^i x_t^{cnn})$$
$$f_t = \sigma(W_h^f h_t + W_c^f x_t^{cnn})$$
$$o_t = \sigma(W_h^o h_t + W_c^o x_t^{cnn})$$
$$z_t = \sigma(W_h^z h_t + W_c^z x_t^{cnn})$$

x_t : Input at t
h_t : Short-term memory vector at t
c_t : Long-term memory vector at t
y_t : Probability of next word at t
x_t^{cnn} : Output of CNN at t



Numerical Experiments

- **Benchmark Penn Treebank Corpus:**

  Length of training set = 1 million words.

  Length of test set = 100,000 words.

- **Note:** Experiments on T=20 time steps of LSTM.

Architecture	Representation	Parameters	Train Perplexity	Test Perplexity
Zaremba et al. (2014) code[6]	embedding	681,800	36.96	117.29
Zaremba et al. (2014) code[6]	one-hot	34,011,600	53.89	118.82
LSTM	embedding	681,800	48.38	120.90
LSTM	one-hot	34,011,600	54.41	120.16
LSTM, dropout	one-hot	34,011,600	145.59	112.98
GCRN-M1	one-hot	42,011,602	18.49	177.14
GCRN-M1, dropout	one-hot	42,011,602	114.29	**98.67**

Table 1. Performance (perplexity) on the PTB dataset.

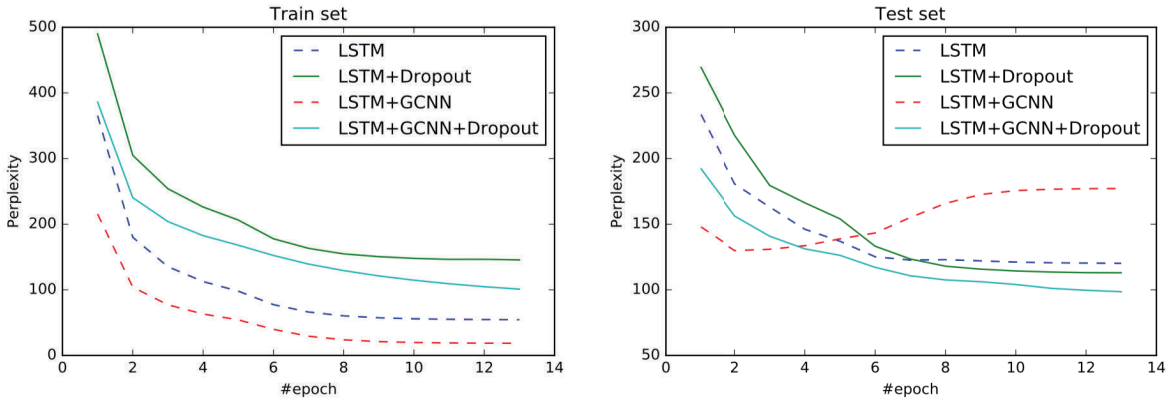


Figure 1. Learning dynamic of LSTM with and without graph structure.

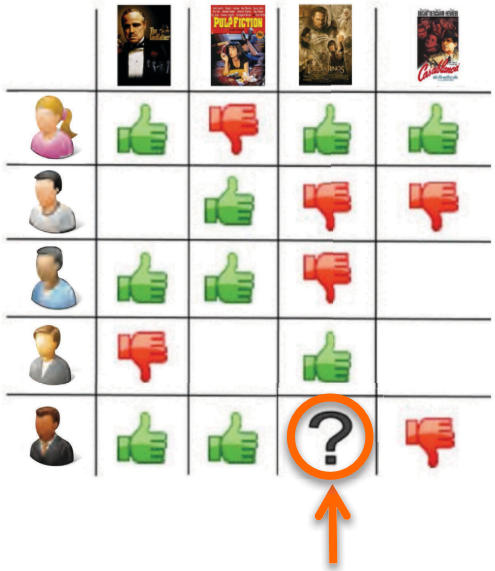
Outline

- Data Science and Deep Learning
  - *A brief history*
  - *What makes DL great*
- Convolutional Neural Networks
  - *Architecture*
- Convolutional Neural Networks on Graphs
  - *Non-Euclidean data*
  - *Graph convolution, graph downsampling and pooling*
  - *Numerical experiments*
- Natural Language Processing
- **Recommender Systems**
- Conclusion

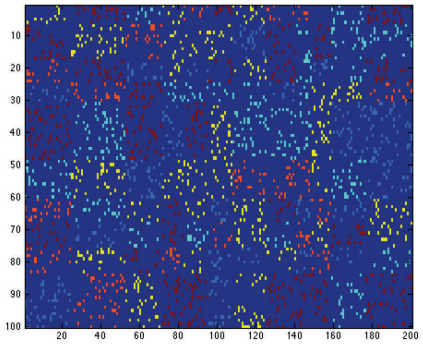
Recommender Systems

- **Goal:** Predict user ratings for items based on previous ratings.

  Ex. Netflix



- **Formalization:** Given a few ratings/entries M_{ij} of item j and user i, find a low-rank matrix X that best fits the ratings.

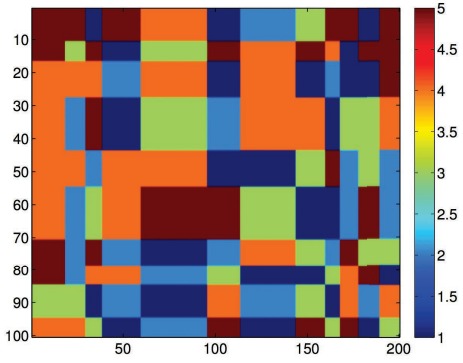


M

Recommendation

$=$

Matrix completion

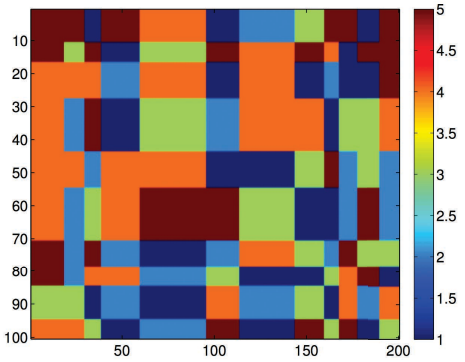


X

Low-Rank Assumption

- **Assumption: Rating matrix X is low-rank.**

  Definition: A low-rank matrix has many columns and rows that are linearly dependent.



- **For Netflix movie recommendation:**

  1. *There exist communities of users who rate movies the same way.*
  2. *There are groups of movies that receive same ratings.*

  Same valid assumptions for Amazon (users, products), LinkedIn (users, jobs), Facebook (users, ads), etc.

Low-Rank Matrix Completion [Candes-Recht'09]

- **Formalization:**

$$\min_X \ \text{rank}(X) \text{ s.t. } \underbrace{X_{ij} = M_{ij}}_{\text{No noise in measurements}}, \forall ij \in \Omega$$

Set of observed entries of M

$$\Downarrow$$

$$\min_X \ \text{rank}(X) + \underbrace{\frac{\lambda}{2}\|\Omega \circ (X - M)\|_F^2}_{\text{Robustness w.r.t. noise in measurements}}$$

- **Solution: Combinatorial problem is NP-hard** (because of rank).

Relaxation

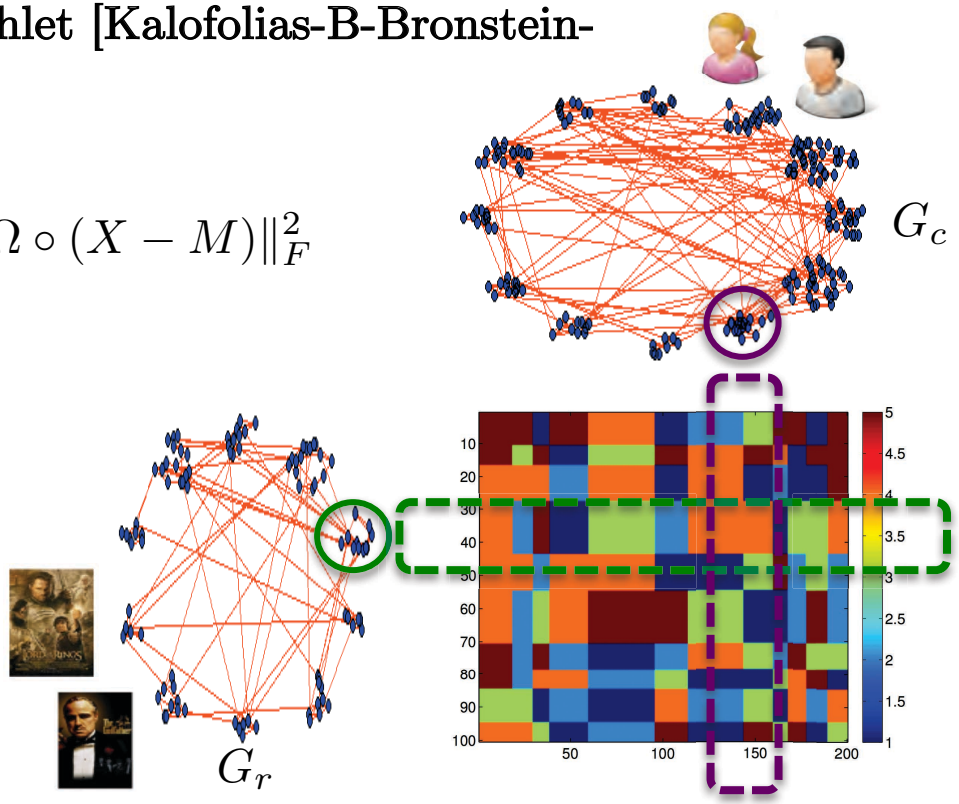
- **There exits two classes of rank relaxation:**
  1. *Nuclear norm relaxation* $\Rightarrow$ ***Collaborative*** *filtering recom systems (Netflix)*
  2. *Graph relaxation* $\Rightarrow$ ***Content*** *filtering recom systems (Amazon)*

- **Graph relaxation with graph Dirichlet [Kalofolias-B-Bronstein-Vandergheynst'14]:**

$$\min_X \ \frac{1}{2}\|X\|^2_{G_c} + \frac{1}{2}\|X\|^2_{G_r} + \frac{\lambda}{2}\|\Omega \circ (X - M)\|^2_F$$

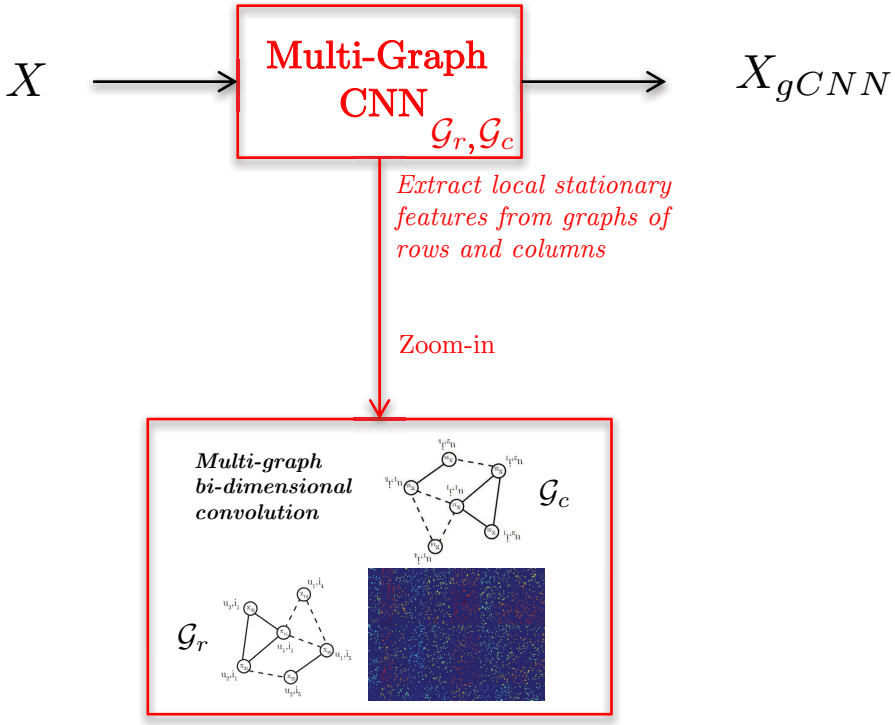
$$\text{with} \quad \|X\|^2_{G_r} = \mathrm{tr}(X^T L_r X)$$
$$\|X\|^2_{G_c} = \mathrm{tr}(X L_c X^T)$$

$\Rightarrow$ **Diffusion term is designed to force the ratings to be smooth on graphs.**



How Graph CNNs can Help?

- Graph CNNs can extract highly meaningful patterns on graphs of users and items, and use them for rating predictions.



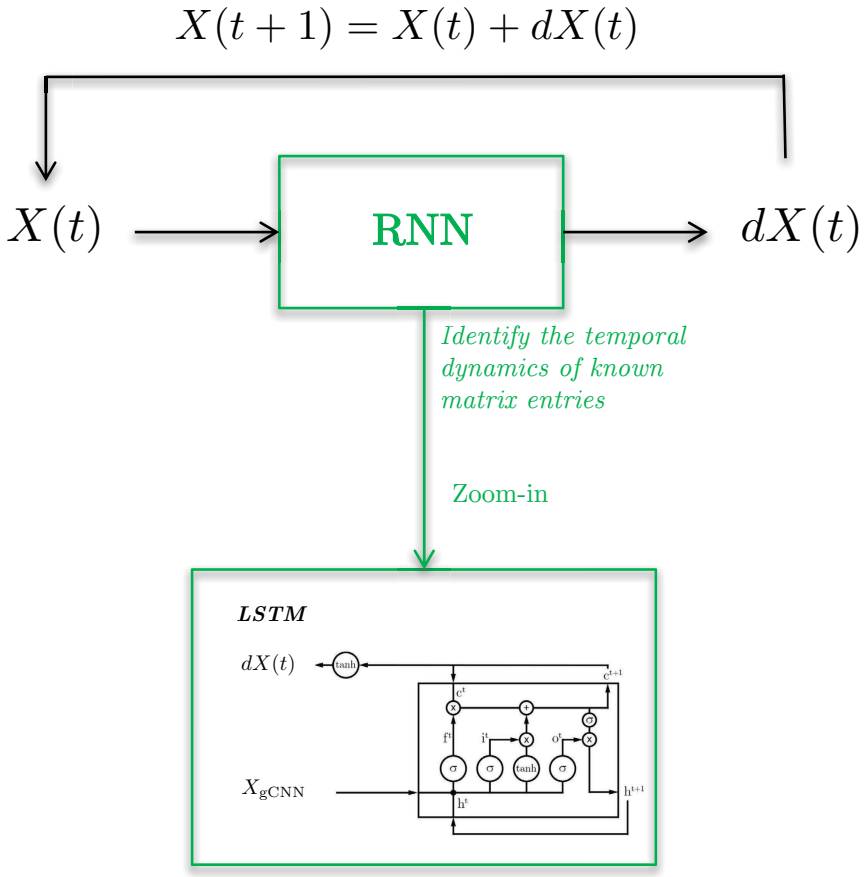
- Observe that CNNs are designed on multiple graphs.

Capturing Ratings Dynamics

- **Assumption: The collected ratings M are the "snapshot" of a stochastic non-linear diffusion process.** For example, movies can go in and out of popularity as triggered by external events such as an actor's appearance in a new movie or chit-chats on social network [Koren-et-al'09].

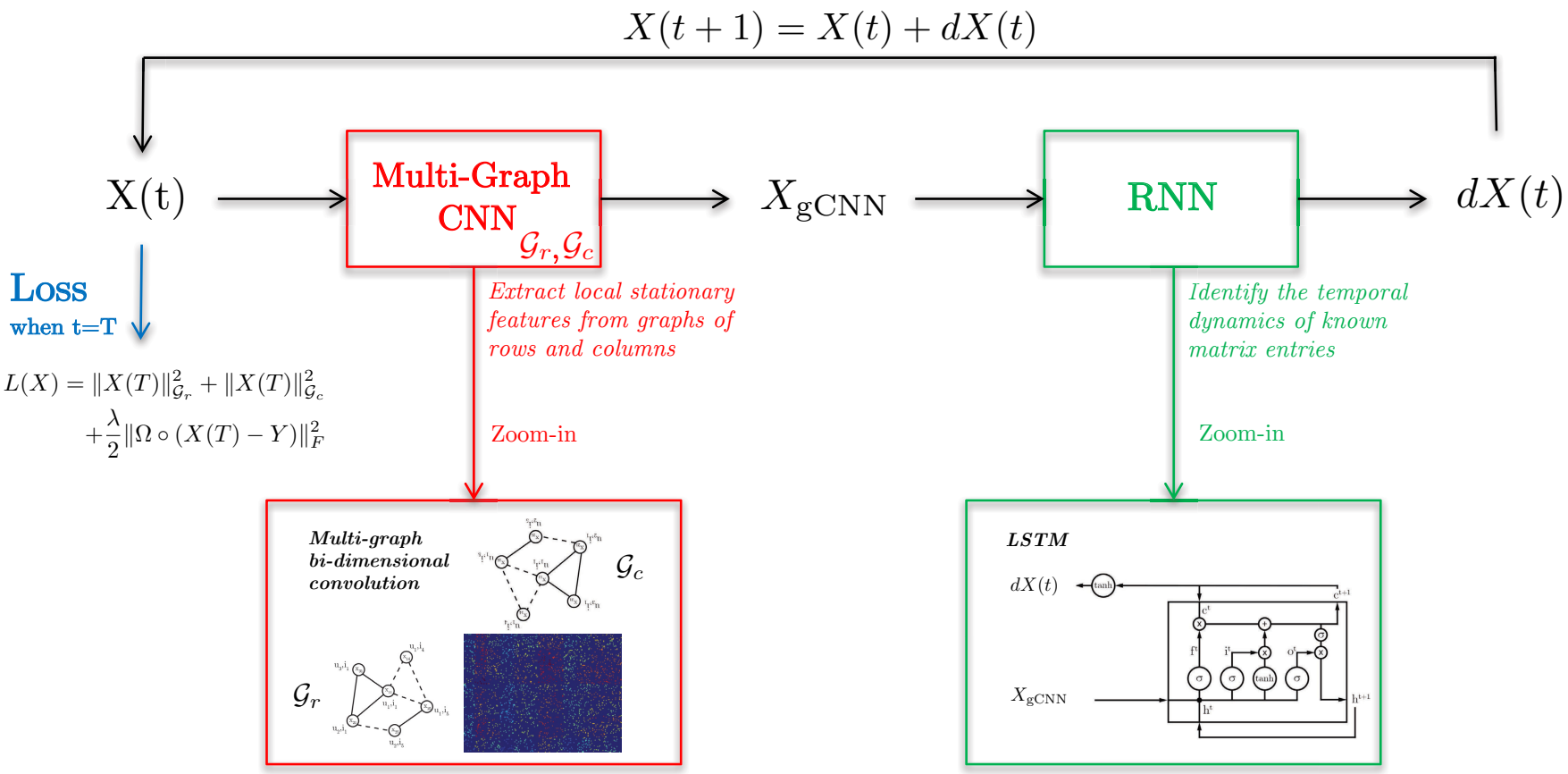
- **How to capture the dynamics of ratings generation?**

  It can be learned by a RNN/LSTM [Hochreiter-Schmidhuber'97].



Final Architecture
[Monti-Bronstein-B'17]

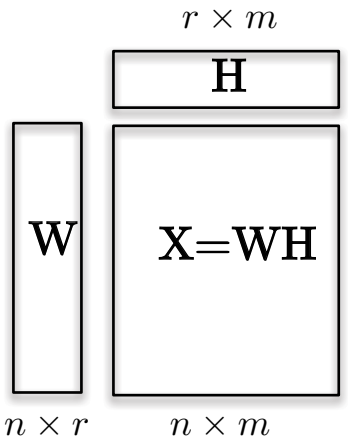
- **Proposed recommender system combines gCNN + LSTM:**



- **If $X \in R^{m \times n}$, then learning complexity is $O(m.n) \Rightarrow$ Quadratic, no scale up.**

Factorized Models

- Factorized matrices:



$$r \ll n, m$$

- New loss function:

$$\min_X \frac{1}{2}\|W\|^2_{G_r} + \frac{1}{2}\|H\|^2_{G_c} + \frac{\lambda}{2}\|\Omega \circ (WH - M)\|^2_F$$

$$\text{with} \quad \|W\|^2_{G_r} = \text{tr}(W^T L_r W)$$
$$\|H\|^2_{G_c} = \text{tr}(H L_c H^T)$$

- If $X \in R^{m \times n}$, then learning complexity is $O(m+n) \Rightarrow$ Linear

Numerical Experiments

- We evaluated the system on benchmark datasets.

METHOD	RMSE
GLOBAL MEAN	1.154
USER MEAN	1.063
MOVIE MEAN	1.033
MC (CANDES & RECHT, 2012)	0.973
IMC (JAIN & DHILLON, 2013; XU ET AL., 2013)	1.653
GMC (KALOFOLIAS ET AL., 2014)	0.996
GRALS (RAO ET AL., 2015)	0.945
sRGCNN	**0.929**

Table 1. Performance (RMS error) on the MovieLens dataset.

METHOD	FLIXSTER	DOUBAN	YAHOOMUSIC
GRALS	1.3126 / 1.2447	0.8326	38.0423
sRGCNN	**1.1788 / 0.9258**	**0.8012**	**22.4149**

Table 2. Performance (RMS error) on Flixster, Douban and YahooMusic.

Outline

- Data Science and Deep Learning
  - *A brief history*
  - *What makes DL great*
- Convolutional Neural Networks
  - *Architecture*
- Convolutional Neural Networks on Graphs
  - *Non-Euclidean data*
  - *Graph convolution, graph downsampling and pooling*
  - *Numerical experiments*
- Natural Language Processing
- Recommender Systems
- **Conclusion**

Conclusion

- **Proposed contributions [NIPS'16]:**
  1. *Generalization of CNNs to non-Euclidean data*
  2. *Exact localized filters on graphs*
  3. *Same linear learning complexity as CNNs while being universal to any graph*
  4. *GPU implementation*

- **Several potential applications including**

  Biological networks (gene, brain connectivity)

Workshop Announcement

New Deep Learning Techniques

Feb 5-9, 2018

Organizers

Xavier Bresson, NTU

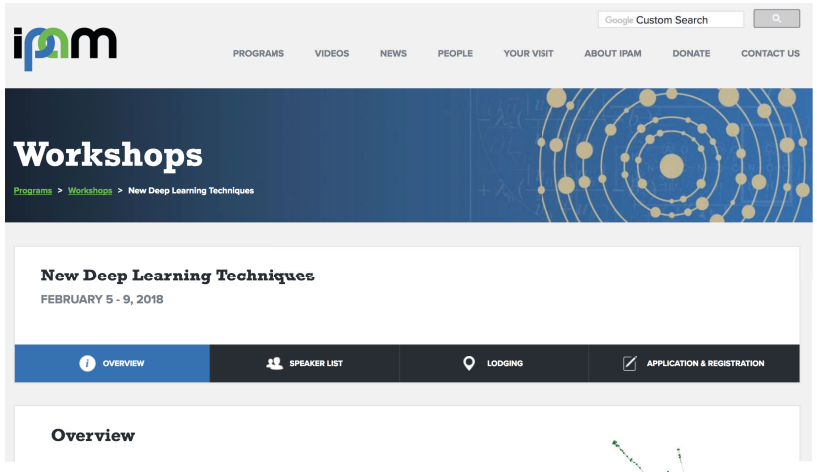
Michael Bronstein, USI/Intel

Joan Bruna, Berkeley

Yann LeCun, Facebook FAIR Director

Stanley Osher, UCLA

Arthur Szlam, Facebook



UCLA

http://www.ipam.ucla.edu/programs/workshops/new-deep-learning-techniques

Thank you.