Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

M. Defferrard\textsuperscript{1} X. Bresson\textsuperscript{2} P. Vandergheynst\textsuperscript{1}

\textsuperscript{1}EPFL, Lausanne, Switzerland

\textsuperscript{2}Nanyang Technological University, Singapore

Itay Boneh, Asher Kabakovitch
Tel-Aviv University, Deep Learning Seminar
2017
Spectral Filtering
Non-parametric

From convolution theorem:

$$x \ast_{G} g = U \left( U^{T} g \odot U^{T} x \right) = U \left( \hat{g} \odot U^{T} x \right)$$

Or algebraically:

$$x \ast_{G} g = U \begin{bmatrix} \hat{g}_{1} & 0 \\ & \ddots \\ 0 & \hat{g}_{n} \end{bmatrix} U^{T} x$$

Not localized

$O(n)$ parameters to train

$O(n^2)$ multiplications (no FFT)
Spectral Filtering
Non-parametric

From convolution theorem:

\[ x \ast g = U \left( U^T g \odot U^T x \right) = U \left( \hat{g} \odot U^T x \right) \]

Or algebrically:

\[ x \ast g = U \begin{bmatrix} \hat{g}_1 & 0 \\ \vdots \\ 0 & \hat{g}_n \end{bmatrix} U^T x \]

- Not localized
- \( O(n) \) parameters to train
- \( O(n^2) \) multiplications (no FFT)
Spectral Filtering
Polynomial Parametrization

$\hat{g}$ is a continuous 1-D function, parametrized by $\theta$:

$$x \star_{\hat{g}} g = U \left( \hat{g}_\theta(\lambda) \odot U^T x \right) = U \begin{bmatrix} \hat{g}_\theta(\lambda_1) & 0 \\ 0 & \hat{g}_\theta(\lambda_n) \end{bmatrix} U^T x = U \hat{g}_\theta(\Lambda) U^T x = \hat{g}_\theta(L) x$$

\[
T \phi = T \tilde{\lambda} \Rightarrow f(T) = \phi f(\Lambda) \phi^{-1}
\]
Spectral Filtering

Polynomial Parametrization

\( \hat{g} \) is a continuous 1-D function, parametrized by \( \theta \):

\[
    x \ast_{\hat{g}} g = U \left( \hat{g}_\theta(\lambda) \odot U^T x \right) = U \begin{bmatrix} \hat{g}_\theta(\lambda_1) & 0 \\ \vdots & \ddots \\ 0 & \hat{g}_\theta(\lambda_n) \end{bmatrix} U^T x 
    = U \hat{g}_\theta(\Lambda) U^T x = \hat{g}_\theta(L)x
\]

Polynomial parametrization:

\[
    \hat{g}_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k
\]

- K-localized: 1-hop for every \( L \) application
- \( K \) parameters to train - independent on the graph’s size
- Still \( O(n^2) \) multiplications (multiplications with the basis \( U \))
Spectral Filtering
Polynomial Parametrization

Impulse response on a 2D Euclidean domain

Impulse response on a graph
Spectral Filtering
Recursive Polynomial Parametrization

Chebyshev polynomials:

\[
\begin{align*}
T_k(\lambda) &= 2\lambda T_{k-1}(\lambda) - T_{k-2}(\lambda) \\
T_0(\lambda) &= 1 \\
T_1(\lambda) &= \lambda
\end{align*}
\]

Parametrization:

\[
\hat{g}_\theta(L) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})
\]

\[
\tilde{L} = 2L\lambda_n^{-1} - I \quad \text{(orthonormal basis in } [-1, 1])
\]
Spectral Filtering
Recursive Polynomial Parametrization

Filtering:
\[
y = \hat{g}_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x = \sum_{k=0}^{K-1} \theta_k \bar{x}_k
\]

Recurrence:
\[
\begin{cases}
\bar{x}_k = T_k(\tilde{L})x = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2} \\
\bar{x}_0 = x \\
\bar{x}_1 = \tilde{L}x
\end{cases}
\]
Spectral Filtering

Recursive Polynomial Parametrization

Filtering:

\[ y = \hat{g}_\theta(L)x = \sum_{k=0}^{K-1} \theta_k \, T_k(\tilde{L})x = \sum_{k=0}^{K-1} \theta_k \tilde{x}_k \]

Recurrence:

\[
\begin{cases}
\tilde{x}_k = T_k(\tilde{L})x = 2\tilde{L}\tilde{x}_{k-1} - \tilde{x}_{k-2} \\
\tilde{x}_0 = x \\
\tilde{x}_1 = \tilde{L}x
\end{cases}
\]

- K-localized: 1-hop for every \( L \) application
- \( K \) parameters to train - independent on the graph's size
- \( O(Kn) \) multiplications (actually \( O(K|\varepsilon|) \))
- No EVD of \( L \)
Graph CNN

Input graph signals
- e.g., bags of words

Feature extraction: feature maps
- Convolutional layers

Classification
- Fully connected layers

Output signals
- e.g., labels/ classes

Graph signal filtering
1. Convolution
2. Non-linear activation

Graph coarsening
1. Sub-sampling
2. Pooling

$0 = \lambda_1 < \lambda < \lambda_{M-1}$
Graph CNN
Learning Filters

\[
y_{s,j} = \sum_{i=1}^{F_{in}} \hat{g}_{\theta_{i,j}}(L)x_{s,i}
\]

\[
\frac{\partial L}{\partial \theta_{i,j}} = \sum_{s=1}^{S}[\bar{x}_{s,i,0}, \ldots, \bar{x}_{s,i,K-1}]^T \frac{\partial L}{\partial y_{s,j}}
\]

\[
\frac{\partial L}{\partial x_{s,i}} = \sum_{j=1}^{F_{out}} \hat{g}_{\theta_{i,j}}(L) \frac{\partial L}{\partial y_{s,j}}
\]

\(s = 1, \ldots, S\) - sample index
\(i = 1, \ldots, F_{in}\) - input feature map index
\(j = 1, \ldots, F_{out}\) - output feature map index
\(\theta_{i,j}\) - \(F_{in} \times F_{out}\) Chebyshev coefficients vectors of order \(K\)
\(L\) - Loss over a mini-batch of \(S\) samples
Graph CNN

Learning Filters

\[ y_{s,j} = \sum_{i=1}^{F_{\text{in}}} \hat{g}_{\theta_{i,j}}(L)x_{s,i} \]

\[
\frac{\partial L}{\partial \theta_{i,j}} = \sum_{s=1}^{S} \left[ \bar{x}_{s,i,0}, \ldots, \bar{x}_{s,i,K-1} \right]^T \frac{\partial L}{\partial y_{s,j}}
\]

\[
\frac{\partial L}{\partial x_{s,i}} = \sum_{j=1}^{F_{\text{out}}} \hat{g}_{\theta_{i,j}}(L) \frac{\partial L}{\partial y_{s,j}}
\]

\[ O(K|\varepsilon|F_{\text{in}}F_{\text{out}}S) \]

Easily paralleled
Graph Pooling

Coarsening

- Multilevel clustering algorithm
- Reduce the size of the graph by a specified factor (2)
- Do all this efficiently
Graph Pooling
Coarsening

- Multilevel clustering algorithm
- Reduce the size of the graph by a specified factor (2)
- Do all this efficiently

Graclus multilevel clustering algorithm

- Maximizing local normalized cut
- Greedily pick an unmarked vertex $i$ and match it with an unmatched vertex $j$ which maximizes the local normalized cut $W_{i,j}(1/d_i + 1/d_j)$.
- Extremely fast.
- Dividing the number of nodes by approximately 2.
- Might generate singletons (non matched) nodes. Solved by using fake nodes.
Graph Pooling

Fast Pooling

Example: Pooling by 4

- Graclus generates **singletons**: $\bar{n}_0 = 8 \rightarrow \bar{n}_1 = 5 \rightarrow \bar{n}_2 = 3$
- By adding **fake nodes** we get: $n_2 = 3 \rightarrow n_1 = 6 \rightarrow n_0 = 12$
- $z = [\max(x_0, x_1), \max(x_4, x_5, x_6), \max(x_8, x_9, x_{10})]$]
- Balanced binary trees $\Rightarrow$ efficient on GPUs.
Experiments

MNIST

- 28×28 pixels + 192 fake nodes ⇒ $n = |\mathcal{V}| = 976$
- 8-NN graph ⇒ $|\varepsilon| = 3198$
- Based on LeNet-5 ⇒ $K = 5$

\[ W_{i,j} = e^{-\|x_i - x_j\|_2^2 / \sigma^2} \]
Experiments

MNIST

- $28 \times 28$ pixels + 192 fake nodes $\Rightarrow n = |\mathcal{V}| = 976$
- 8-NN graph $\Rightarrow |\mathcal{E}| = 3198$
- Based on LeNet-5 $\Rightarrow K = 5$

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Architecture</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical CNN</td>
<td>C32-P4-C64-P4-FC512</td>
<td>99.33</td>
</tr>
<tr>
<td>Proposed graph CNN</td>
<td>GC32-P4-GC64-P4-FC512</td>
<td>99.14</td>
</tr>
</tbody>
</table>

Comparison to classical CNNs.
Experiments

MNIST

\[ W_{i,j} = e^{-\frac{\|x_i - x_j\|_2^2}{\sigma^2}} \]

- 28\times28 \text{ pixels} + 192 \text{ fake nodes} \Rightarrow n = |\mathcal{V}| = 976
- 8-NN graph \Rightarrow |\varepsilon| = 3198
- Based on LeNet-5 \Rightarrow K = 5
- Isotropic filters (no orientation)
- Uninvestigated optimizations and initializations
Experiments

20NEWS

- 18,846 text documents associated with 20 classes
- 10K most common words from the 94K unique words \( \Rightarrow n = |\mathcal{V}| = 10K \)
- 16-NN,
  \[ W_{i,j} = e^{-\|z_i - z_j\|_2^2 / \sigma^2} \]  
  \( z_i \) - word2vec
  \( \Rightarrow |\varepsilon| = 132,834 \)
- \( x \) - bag-of-words model
Experiments

20NEWS

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear SVM</td>
<td>65.90</td>
</tr>
<tr>
<td>Multinomial Naive Bayes</td>
<td>68.51</td>
</tr>
<tr>
<td>Softmax</td>
<td>66.28</td>
</tr>
<tr>
<td>FC2500</td>
<td>64.64</td>
</tr>
<tr>
<td>FC2500-FC500</td>
<td>65.76</td>
</tr>
<tr>
<td>GC32</td>
<td>68.26</td>
</tr>
</tbody>
</table>

Comparison with other methods ($K = 5$)

- Slightly worse than multinomial naive Bayes classifier.
- Defeats fully-connected networks with much less parameters.
Experiments

20NEWS

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear SVM</td>
<td>65.90</td>
</tr>
<tr>
<td>Multinomial Naive Bayes</td>
<td>68.51</td>
</tr>
<tr>
<td>Softmax</td>
<td>66.28</td>
</tr>
<tr>
<td>FC2500</td>
<td>64.64</td>
</tr>
<tr>
<td>FC2500-FC500</td>
<td>65.76</td>
</tr>
<tr>
<td>GC32</td>
<td>68.26</td>
</tr>
</tbody>
</table>

Comparison with other methods ($K = 5$)

- Slightly worse than multinomial naive Bayes classifier.
- Defeats fully-connected networks with much less parameters.

Total training time divided by # of gradient steps

- Scales as $O(n)$ as opposed to $O(n^2)$
Experiments
Graph Quality

<table>
<thead>
<tr>
<th>Architecture</th>
<th>8-NN on 2D Euclidean grid</th>
<th>random</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC32</td>
<td>97.40</td>
<td>96.88</td>
</tr>
<tr>
<td>GC32-P4-GC64-P4-FC512</td>
<td>99.14</td>
<td>95.39</td>
</tr>
</tbody>
</table>

Classification accuracies with different graph constructions on MNIST.

⇒ The data structure is important

<table>
<thead>
<tr>
<th>word2vec</th>
<th>bag-of-words</th>
<th>pre-learned</th>
<th>learned</th>
<th>approximate</th>
<th>random</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>67.50</td>
<td>66.98</td>
<td>68.26</td>
<td>67.86</td>
<td>67.75</td>
</tr>
</tbody>
</table>

Accuracies of GC32 with different graph constructions on 20NEWS.

⇒ A well constructed graph is important
⇒ Proper approximations (LSHForest) can be used for larger DBs.
Conclusions and Future Work

- Introduced a model with linear complexity.
- The quality of the input graph is of paramount importance.
- Local stationarity and compositionality are verified for text documents as long as the graph is well constructed.