Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering
by M. Defferrard, X. Bresson, P. Vandergheynst, 2016
Paper presentation

Hubert Leterme

July 8th, 2020
Table of Contents

1 Introduction
2 Background on CNNs
3 Convolution layers on graphs
4 Pooling layers on graphs
5 Numerical experiments
6 Conclusion and perspectives
7 References
Convolutional neural networks (CNNs), introduced in 1989 by Yann LeCun [1], are able to learn local stationary structures which are composed into multi-scale patterns. They led to breakthroughs in image, video and sound recognition tasks.

Defferard and colleagues [2] introduced a generalization of CNNs to graphs, which encode complex data structures lying on irregular or non-euclidean domains.
Convolutional neural networks (CNNs), introduced in 1989 by Yann LeCun [1], are able to learn local stationary structures which are composed into multi-scale patterns. They led to breakthroughs in image, video and sound recognition tasks.

Defferard and colleagues [2] introduced a generalization of CNNs to graphs, which encode complex data structures lying on irregular or non-euclidean domains.

Main challenges:

- Construct a convolution operator on irregular grids;
- Design strictly localized filters, as in standard CNNs;
- Compute forward- and backward-propagation with a linear complexity w.r.t. the filter support’s size and the number of edges;
- Design an efficient pooling operator (which yields smaller graphs by grouping vertices together);
- Obtain high experimental performance on both standard image and more complex data recognition tasks.
Section 2

Background on CNNs
Description of a convolutional layer

Forward-propagation:

\[ X_s \xrightarrow{\text{Conv}(W)} Y_s \xrightarrow{\text{ReLU}} A_s \xrightarrow{\text{Pool}} Z_s \]

with:

- \( X_s \in \mathbb{R}^{C \times M \times M} \), \( Y_s, A_s \in \mathbb{R}^{D \times M \times M} \) and \( Z_s \in \mathbb{R}^{D \times N \times N} \) (feature maps for the s-th training sample);
- \( W \in \mathbb{R}^{D \times C \times \mu \times \mu} \) (convolution kernels – trainable parameters);
- \( C, D > 0 \) (number of input and output feature maps);
- \( M, N > 0 \) such that \( N < M \) (sizes of the input and output feature maps);
- \( \mu \ll M \) (size of the convolution kernels).

- Conv: convolutional layer (see slide 7);
- ReLu: rectified linear unit (non-linear pointwise operation);
- Pool: pooling operator (e.g. max pooling).
Illustration of a convolutional layer

Computation of the $d$-th output feature map, for $d \in [0..D-1]$:

$$Y_{s,d} = b_d + \sum_{c=0}^{C-1} (W_{d,c} \star X_{s,c})$$

where $\star$ denotes the cross-correlation operator (slid sum-product).

**Figure:** $c$-th input feature map $X_{s,c} \in \mathbb{R}^{M \times M}$, for a given $c \in [0..C-1]$
Illustration of a convolutional layer

Computation of the $d$-th output feature map, for $d \in [0..D−1]$:

$$Y_{s,d} = b_d + \sum_{c=0}^{C−1}(W_{d,c} \star X_{s,c})$$

where $\star$ denotes the **cross-correlation** operator (slid sum-product).

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure:** $c$-th input feature map, extended with zeros (padding)
Illustration of a convolutional layer

Computation of the \( d \)-th output feature map, for \( d \in [0..D−1] \):

\[
Y_{s,d} = b_d + \sum_{c=0}^{C-1} (W_{d,c} \ast X_{s,c})
\]

where \( \ast \) denotes the \textbf{cross-correlation} operator (slid sum-product).

\textbf{Figure:} Cross-correlation mapping the \( c \)-th input \( X_{s,c} \) (left, in blue) to the \( d \)-th output, using the kernel \( W_{d,c} \in \mathbb{R}^{\mu \times \mu} \) (left, in orange). Right: \( (W_{d,c} \ast X_{s,c}) \).
Computation of the $d$-th output feature map, for $d \in [0..D-1]$:

$$Y_{s,d} = b_d + \sum_{c=0}^{C-1} (W_{d,c} \ast X_{s,c})$$

where $\ast$ denotes the **cross-correlation** operator (slid sum-product).

**Figure:** Cross-correlation mapping the $c$-th input $X_{s,c}$ (left, in blue) to the $d$-th output, using the kernel $W_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(W_{d,c} \ast X_{s,c})$. 

Illustration of a convolutional layer
Illustration of a convolutional layer

Computation of the $d$-th output feature map, for $d \in [0..D-1]$:

$$Y_{s,d} = b_d + \sum_{c=0}^{C-1} (W_{d,c} \star X_{s,c})$$

where $\star$ denotes the cross-correlation operator (slid sum-product).

Figure: Cross-correlation mapping the $c$-th input $X_{s,c}$ (left, in blue) to the $d$-th output, using the kernel $W_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(W_{d,c} \star X_{s,c})$. 
Illustration of a convolutional layer

Computation of the $d$-th output feature map, for $d \in [0..D - 1]$:

$$Y_{s,d} = b_d + \sum_{c=0}^{C-1} (W_{d,c} \ast X_{s,c})$$

where $\ast$ denotes the cross-correlation operator (slid sum-product).

**Figure:** Cross-correlation mapping the $c$-th input $X_{s,c}$ (left, in blue) to the $d$-th output, using the kernel $W_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(W_{d,c} \ast X_{s,c})$. 
Illustration of a convolutional layer

Computation of the $d$-th output feature map, for $d \in [0..D-1]$:

$$Y_{s,d} = b_d + \sum_{c=0}^{C-1} (W_{d,c} \ast X_{s,c})$$

where $\ast$ denotes the **cross-correlation** operator (slid sum-product).

**Figure**: Cross-correlation mapping the $c$-th input $X_{s,c}$ (left, in blue) to the $d$-th output, using the kernel $W_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(W_{d,c} \ast X_{s,c})$. 
Illustration of a convolutional layer

Computation of the $d$-th output feature map, for $d \in [0..D - 1]$:

$$Y_{s,d} = b_d + \sum_{c=0}^{C-1} (W_{d,c} \star X_{s,c})$$

where $\star$ denotes the cross-correlation operator (slid sum-product).

Figure: Cross-correlation mapping the $c$-th input $X_{s,c}$ (left, in blue) to the $d$-th output, using the kernel $W_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(W_{d,c} \star X_{s,c})$. 
Illustration of a convolutional layer

Computation of the $d$-th output feature map, for $d \in [0..D - 1]$:

$$ Y_{s,d} = b_d + \sum_{c=0}^{C-1} (W_{d,c} \star X_{s,c}) $$

where $\star$ denotes the cross-correlation operator (slid sum-product).

Figure: Cross-correlation mapping the $c$-th input $X_{s,c}$ (left, in blue) to the $d$-th output, using the kernel $W_{d,c} \in \mathbb{R}^{\mu \times \mu}$ (left, in orange). Right: $(W_{d,c} \star X_{s,c})$. 
Convolution versus cross-correlation

- **Matrix convolution product:**
  \[(U \ast V)[m, n] = \sum_{i \in \mathbb{Z}} \sum_{j \in \mathbb{Z}} U[i, j] \cdot V[m - i, n - j]\]

- **Cross-correlation:**
  \[(U \star V)[m, n] = \sum_{i \in \mathbb{Z}} \sum_{j \in \mathbb{Z}} U[i, j] \cdot V[m + i, n + j]\]

**Proposition 2.1**

\[U \ast V = \overline{U} \ast V\]

*where \(\overline{U}[m, n] = U[-m, -n]\).*
Learning convolution kernels
Description of a training step

Let’s assume the following values have already been computed:
- \( E \): loss computed over a minibatch of \( S \) samples;
- \( \left\{ \nabla_{(Y_{s,d})} E \bigg| s \in [0..S - 1], d \in [0..D - 1] \right\} \): gradients w.r.t. the outputs;

Then, backpropagates the gradient in \( \mathcal{O}(SCD \cdot \mu^2 N^2) \):

\[
\nabla_{(W_{d,c})} E = \sum_{s=1}^{S} \left( \nabla_{(Y_{s,d})} E \right) \ast X_{s,c}
\]

\[
\nabla_{(X_{s,c})} E = \sum_{d=1}^{D} \left( \nabla_{(Y_{s,d})} E \right) \ast W_{d,c}
\]

Finally, update the weights using stochastic gradient descent:

\[
W_{d,c} \leftarrow \left( W_{d,c} - \eta \cdot \nabla_{(W_{d,c})} E \right)
\]
Learning convolution kernels
Description of a training step

Let’s assume the following values have already been computed:
- \( E \): loss computed over a minibatch of \( S \) samples;
- \( \left\{ \nabla_{(y_s,d)} E \middle| s \in [0..S-1], d \in [0..D-1] \right\} \): gradients w.r.t. the outputs;

Then, backpropagates the gradient in \( \mathcal{O}(SCD \cdot \mu^2 N^2) \):
\[
\nabla_{(w_{d,c})} E = \sum_{s=1}^{S} \left( \nabla_{(y_s,d)} E \right) \ast x_{s,c}
\]
\[
\nabla_{(x_{s,c})} E = \sum_{d=1}^{D} \left( \nabla_{(y_s,d)} E \right) \ast w_{d,c}
\]

Finally, update the weights using stochastic gradient descent:
\[
\text{w}_{d,c} \leftarrow (\text{w}_{d,c} - \eta \cdot \nabla_{(w_{d,c})} E)
\]
For any sample $s$ and any output $d$:

$$Z_{s,d}[m, n] = \max_{i, j \in \{0, 1\}} \left( Y_{s,d}[2m + i, 2n + j] \right)$$

with $Y_{s,d} \in \mathbb{R}^{N \times N}$ and $Z_{s,d} \in \mathbb{R}^{(N/2) \times (N/2)}$. 
Illustration of a pooling layer
Example of max pooling with size $2 \times 2$

For any sample $s$ and any output $d$:

$$Z_{s,d}[m, n] = \max_{i,j \in \{0,1\}} \left( Y_{s,d}[2m + i, 2n + j] \right)$$

with $Y_{s,d} \in \mathbb{R}^{N \times N}$ and $Z_{s,d} \in \mathbb{R}^{(N/2) \times (N/2)}$. 
For any sample $s$ and any output $d$:

$$Z_{s,d}[m, n] = \max_{i, j \in \{0, 1\}} \left( Y_{s,d}[2m + i, 2n + j] \right)$$

with $Y_{s,d} \in \mathbb{R}^{N \times N}$ and $Z_{s,d} \in \mathbb{R}^{(N/2) \times (N/2)}$. 

Illustration of a pooling layer
Example of max pooling with size $2 \times 2$
For any sample $s$ and any output $d$:

$$Z_{s,d}[m, n] = \max_{i,j \in \{0,1\}} \left( Y_{s,d}[2m + i, 2n + j] \right)$$

with $Y_{s,d} \in \mathbb{R}^{N \times N}$ and $Z_{s,d} \in \mathbb{R}^{(N/2) \times (N/2)}$. 
For any sample $s$ and any output $d$:

$$Z_{s,d}[m, n] = \max_{i,j \in \{0,1\}} \left( Y_{s,d}[2m + i, 2n + j] \right)$$

with $Y_{s,d} \in \mathbb{R}^{N \times N}$ and $Z_{s,d} \in \mathbb{R}^{(N/2) \times (N/2)}$. 
For any sample $s$ and any output $d$:

$$Z_{s,d}[m, n] = \max_{i,j \in \{0,1\}} \left( Y_{s,d}[2m + i, 2n + j] \right)$$

with $Y_{s,d} \in \mathbb{R}^{N \times N}$ and $Z_{s,d} \in \mathbb{R}^{(N/2) \times (N/2)}$. 

![Illustration of a pooling layer](Image)
For any sample $s$ and any output $d$:

$$Z_{s,d}[m, n] = \max_{i,j \in \{0,1\}} \left(Y_{s,d}[2m + i, 2n + j]\right)$$

with $Y_{s,d} \in \mathbb{R}^{N \times N}$ and $Z_{s,d} \in \mathbb{R}^{(N/2) \times (N/2)}$. 

Illustration of a pooling layer
Example of max pooling with size $2 \times 2$
Section 3

Convolution layers on graphs
Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{A})$, with:
- $|\mathcal{V}| = N$;
- $\mathbf{A} \in \mathbb{R}^{N \times N}$ such that $\mathbf{A}_{ij} \neq 0 \implies (i,j) \in \mathcal{E}$.

$L \in \mathbb{R}^{N \times N}$: positive semidefinite reference matrix for $\mathcal{G}$;

$U, \Lambda \in \mathbb{R}^{N \times N}$, with:
- $U = [u_0, \ldots, u_{N-1}]$: eigenvectors of $L$ (graph Fourier modes);
- $\Lambda = \text{diag}(\lambda_0, \ldots, \lambda_{N-1})$: eigenvalues of $L$ (graph frequencies);

such that $L = U \Lambda U^\top$;

Input signal $x \in \mathbb{R}^N$, defined on the nodes of $\mathcal{G}$;

$\hat{x}$: graph Fourier transform of $x$, such that $\hat{x} = U^\top x$. 
From classical to graph convolutions

Forward-propagation for any sample $s$ and any output $d$:

**Classical CNN:**

$$Y_{s,d} = \sum_{c=0}^{C-1} (W_{d,c} \ast X_{s,c})$$

$$= \sum_{c=0}^{C-1} (W_{d,c} \ast X_{s,c})$$

according to proposition 2.1.

**Graph CNN:**

$$y_{s,d} = \sum_{c=0}^{C-1} (\theta_{d,c} \ast(G) x_{s,c})$$

where $\ast(G)$ has to be defined.
Issue with spatial convolution: **no unique definition of translation** on graphs.
From classical to graph convolutions

Issue with spatial convolution: no unique definition of translation on graphs.
Issue with spatial convolution: no unique definition of translation on graphs.
From classical to graph convolutions

Issue with spatial convolution: **no unique definition of translation** on graphs.
From classical to graph convolutions

Issue with spatial convolution: **no unique definition of translation** on graphs.
Graph convolution via Fourier transform

Instead, use convolution properties in the Fourier domain:

\[ G^{(d)}(x_s) = y_{s,d} = \sum_{c=0}^{C-1} (\theta_{d,c} \ast (G) \ x_{s,c}) \]

\[ = \sum_{c=0}^{C-1} U \cdot g^{(d,c)}(\Lambda) \cdot U^\top x_{s,c} \]

with \( g^{(d,c)} : \mathbb{R}_+ \rightarrow \mathbb{R} \) and \( g^{(d,c)}(\Lambda) = \text{diag} \left( g^{(d,c)}(\lambda_0), \ldots, g^{(d,c)}(\lambda_{N-1}) \right) \).

\[ \Rightarrow \] For any input \( c \) and output \( d \), \( g^{(d,c)} \) only needs to be defined on the graph frequencies \( \lambda_0, \ldots, \lambda_{N-1} \), giving a weight to the corresponding eigenspaces.
Let $\theta_{d,c} \in \mathbb{R}^N$ such that $g^{(d,c)}(\lambda_n) = \theta_{d,c}[n]$ for any $n \in [0..N-1]$.

- Not localized in space;
- $N$ trainable parameters ($\ll N$ for standard CNNs);
- Filtering operation: $O(N^2)$ (linear complexity for standard CNNs).
Let $\theta_{d,c} \in \mathbb{R}^K$ ($K \ll N$) such that for any $n \in [0..N - 1]$:

$$g^{(d,c)}(\lambda_n) = \sum_{k=0}^{K-1} \theta_{d,c}[k] \cdot \lambda_n^k$$

- **$K$-localized filters**: for any $i, j \in [0..N - 1]$, $y_{s,d}[j]$ is influenced by $x_{s,c}[i]$ only if $d_G(i, j) \leq (K - 1)$, where $d_G$ denotes the minimum number of edges connecting vertices $i$ and $j$;
- **$K$ trainable parameters**, which is equal to the filter spatial extension (similarly to standard 1D CNNs);
- Filtering operation: $O(N^2)$ (linear complexity for standard CNNs).
Parametrization of graph filters
Parametrization in the basis of Chebyshev polynomials

Let $\theta_{d,c} \in \mathbb{R}^K$ ($K \ll N$) such that for any $n \in [0 .. N - 1]$: 

$$g^{(d,c)}(\lambda_n) = \sum_{k=0}^{K-1} \theta_{d,c}[k] \cdot T_k(\tilde{\lambda}_n)$$

with $\tilde{\lambda} = 2\lambda/\lambda_{\text{max}} - 1$ and $T_k \in \mathcal{P}_k(\mathbb{R})$ (Chebyshev polynomials) such that:

$$
\begin{align*}
T_0(u) &= 1 \\
T_1(u) &= u \\
T_k(u) &= 2uT_{k-1}(u) - T_{k-2}(u) \quad \text{for any } k \geq 2
\end{align*}
$$

- **$K$-localized filters**: for any $i, j \in [0 .. N - 1]$, $y_{s,d}[j]$ is influenced by $x_{s,c}[i]$ only if $d_G(i, j) \leq (K - 1)$, where $d_G$ denotes the minimum number of edges connecting vertices $i$ and $j$;

- **$K$ trainable parameters**, which is equal to the filter spatial extension (similarly to standard 1D CNNs);

- **Fast filtering operation** with complexity $\mathcal{O}(K|\mathcal{E}|) \ll \mathcal{O}(N^2)$ (takes advantage of the sparsity of $L$).
Learning graph convolution filters

Forward-propagation

**Forward-propagation** in $O(SCD \cdot K|E|)$:

$$
y_{s,d} = \sum_{c=0}^{C-1} \left( \theta_{d,c} \ast (g) \; x_{s,c} \right)
$$

$$
= \sum_{c=0}^{C-1} \left( T_{L}(x_{s,c}) \cdot \theta_{d,c} \right)
$$

with:

- $\theta_{d,c} \in \mathbb{R}^K$ vector of Chebyshev coefficients;
- $T_{L} : \mathbb{R}^N \rightarrow \mathbb{R}^{N \times K}$, computed in $O(K|E|)$ with $K$ recursive computations.
Gradient backpropagation in $O(SCD \cdot K|\mathcal{E}|)$ (assuming $|\mathcal{E}| \sim N$):

$$\nabla(\theta_{d,c}) E = \sum_{s=0}^{S-1} \left[ T_L(x_s, c)^\top \cdot \nabla(y_s, d) E \right]$$

$$\nabla(x_s, c) E = \sum_{d=0}^{D-1} \left[ T_L \left( \nabla(y_s, d) E \right) \cdot \theta_{d,c} \right]$$

where the loss $E$ is computed over a minibatch of $S$ samples.

Stochastic gradient descent: $\theta_{d,c} \leftarrow \left( \theta_{d,c} - \eta \cdot \nabla(\theta_{d,c}) E \right)$
**Gradient backpropagation** in $O(SCD \cdot K|E|)$ (assuming $|E| \sim N$):

$$\nabla_{(\theta_d,c)} E = \sum_{s=0}^{S-1} \left[ \mathcal{T}_L(x_s,c)^T \cdot \nabla_{(y_s,d)} E \right]$$

$$\nabla_{(x_s,c)} E = \sum_{d=0}^{D-1} \left[ \mathcal{T}_L \left( \nabla_{(y_s,d)} E \right) \cdot \theta_{d,c} \right]$$

where the loss $E$ is computed over a minibatch of $S$ samples.

**Stochastic gradient descent**: $\theta_{d,c} \leftarrow \left( \theta_{d,c} - \eta \cdot \nabla_{(\theta_d,c)} E \right)$
Section 4

Pooling layers on graphs
From classical to graph pooling layers

Forward-propagation for any sample $s$ and any output $d$:

**Classical CNN:**

$$Z_{s,d}[n] = \max_{i \in \{0, 1\}^2} \left( Y_{s,d}[2n + i] \right)$$

**Graph CNN:**

$$z_{s,d}[n] = \max_{m \in \pi_n} \left( y_{s,d}[m] \right)$$

where $\pi_n \subset [0 \ldots N - 1]$ denotes the set of neighboring nodes that are reduced into one in the output graph.

**Goal:** find a graph structure $G' = (\mathcal{V}', \mathcal{E}', \mathbf{W}')$ with $|\mathcal{V}'| = N' = \lceil N/2 \rceil$ and a grouping $\{\pi_n\}_{n \in [0..N'-1]}$, such that local geometric structures are preserved.
Graph clustering is **NP-hard**.

- Approximation with a **greedy algorithm**: Graclus multilevel clustering.
- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.
Graph clustering is **NP-hard**.

- Approximation with a **greedy algorithm**: Graclus multilevel clustering.
Graph clustering is **NP-hard**.

Approximation with a **greedy algorithm**: Graclus multilevel clustering.
Graph clustering is **NP-hard**.

Approximation with a **greedy algorithm**: Graclus multilevel clustering.
Graph clustering is **NP-hard**.

Approximation with a **greedy algorithm**: Graclus multilevel clustering.
Graph clustering is **NP-hard**.

Approximation with a **greedy algorithm**: Graclus multilevel clustering.
Graph clustering is **NP-hard**.

Approximation with a **greedy algorithm**: Graclus multilevel clustering.
- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.
- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.
- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.
Graph clustering is **NP-hard**.

Approximation with a **greedy algorithm**: Graclus multilevel clustering.
- Graph clustering is **NP-hard**.
- Approximation with a **greedy algorithm**: Graclus multilevel clustering.
Fast pooling algorithm

**Idea:** rearrange vertices such that the pooling operation is computed over 2 consecutive nodes:

\[
\forall n \in [0 \ldots N' - 1], \pi(n) = \{2n, 2n + 1\}
\]

Then:

\[
z_{s,d}[n] = \max_{i \in \{0,1\}^2} (y_{s,d}[2n + i])
\]
**Fast pooling algorithm**

**Idea:** rearrange vertices such that the pooling operation is computed over 2 consecutive nodes:

\[ \forall n \in [0..N' - 1], \pi(n) = \{2n, 2n + 1\} \]

Then:

\[ z_{s,d}[n] = \max_{i \in \{0,1\}^2} \left( y_{s,d}[2n + i] \right) \]

**Figure:** From [2]
Section 5

Numerical experiments
Applying graph CNN on image classification

- Sanity check for the model: it should at least perform well on standard image classification tasks.
- 8-NN\(^1\) similarity graph of the 2D grid:

\[
A[i, j] = \exp\left(-\frac{\|z_j - z_i\|^2}{\sigma^2}\right), \text{ where } z_i \in \mathbb{R}^2 \text{ is the coordinate of pixel } i \text{ on the grid.}
\]

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical CNN</td>
<td>99.33</td>
</tr>
<tr>
<td>Proposed graph CNN</td>
<td>99.14</td>
</tr>
</tbody>
</table>

**Figure:** Classical vs graph CNN

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Architecture</th>
<th>Non-Param (2)</th>
<th>Spline (7) [4]</th>
<th>Chebyshev (4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>GC10</td>
<td>95.75</td>
<td>97.26</td>
<td>97.48</td>
</tr>
<tr>
<td>MNIST</td>
<td>GC32-P4-GC64-P4-FC512</td>
<td>96.28</td>
<td>97.15</td>
<td>99.14</td>
</tr>
</tbody>
</table>

**Figure:** Different models of graph CNNs

\(^1\) nearest neighbors
- **Text categorization** problem on the 20NEWS dataset [4].
- Using a **bag-of-words model** [5]: each document (input data) is represented as a vector \( \mathbf{x} \in \mathbb{R}^N \) with \( N = 10,000 \) (**most common words** in the corpus). \( x[i] \) is the number of occurrences of word \( i \) in the document.
Text categorization problem on the 20NEWS dataset [4].

Using a **bag-of-words model** [5]: each document (input data) is represented as a vector $x \in \mathbb{R}^N$ with $N = 10,000$ (*most common words in the corpus*). $x[i]$ is the number of occurrences of word $i$ in the document.

Word2vec embedding [5]: each word $i$ is **semantically represented** as a vector $z_i \in \mathbb{R}^d$ using (e.g. $d = 640$).

Data structure: **16-NN graph** $G = (\mathcal{V}, \mathcal{E}, A)$, with:
- $|\mathcal{V}| = 10,000$;
- $|\mathcal{E}| = 132,834$ edges (connections between the nearest neighbors, using the Euclidean distance induced by the word2vec embedding);
- weights: $A[i,j] = \exp \left( - \frac{||z_j - z_i||^2}{\sigma^2} \right)$.

Model trained for 20 epochs using **Adam optimizer** [6] and initial learning rate $\eta = 0.001$. 

27 / 32
**Text categorization**

Results

<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>

**Figure:** Proposed model (GC32) is beaten by multinomial Bayes classifier but outperforms fully-connected networks with much less parameters.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Architecture</th>
<th>Non-Param (2)</th>
<th>Spline (7) [4]</th>
<th>Chebyshev (4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>GC10</td>
<td>95.75</td>
<td>97.26</td>
<td>97.48</td>
</tr>
<tr>
<td>MNIST</td>
<td>GC32-P4-GC64-P4-FC512</td>
<td>96.28</td>
<td>97.15</td>
<td>99.14</td>
</tr>
</tbody>
</table>

**Figure:** Different models of graph CNNs
Influence of graph structure on prediction accuracy

<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>

**Figure:** MNIST

<table>
<thead>
<tr>
<th>Architecture</th>
<th>word2vec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bag-of-words</td>
</tr>
<tr>
<td></td>
<td>67.50</td>
</tr>
</tbody>
</table>

**Figure:** 20NEWS

- “bag-of-words”: naive embedding;
- “learned”: embedding learned with word2vec [5];
- “approximate”: approximate nearest-neighbors algorithm used for larger datasets.

⇒ The quality of the results **strongly depend on the graph structure**. It must be designed in order to fulfill **assumptions of locality and stationarity**, as in classical CNNs.
Defferrard and colleagues [2] proposed a model of graph CNN able to extract local and stationary features from the data. Improvements with respect to previous graph CNNs [7] are:

- **strictly localized** convolution filters;
- **computational efficiency** which is comparable to classical CNNs;
- **higher test accuracy**.

**Future work:**

- Explore applications to fields where the data naturally lies on graphs, with explicit information about its structure;
- Learn optimal graph structure in parallel to CNN parameters (instead of using a pre-defined one).

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst.
Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner.
Gradient-based learning applied to document recognition.

Thorsten Joachims.
A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.
References II

