Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

A Discussion

Naganand Y

Introduction

- **CNNs**: state-of-the art on image, speech, video recognition tasks

Introduction

- **CNNs**: state-of-the art on image, speech, video recognition tasks
  - (Shared) Filters are translation invariant

Introduction

- **CNNs**: state-of-the art on image, speech, video recognition tasks
  - (Shared) Filters are translation invariant
  - # Parameters independent of input size

Introduction

- **CNNs**: state-of-the art on image, speech, video recognition tasks
  - (Shared) Filters are translation invariant
  - # Parameters independent of input size
  - Local dependencies dominate

Introduction

- **CNNs**: state-of-the art on image, speech, video recognition tasks
    - (Shared) Filters are translation invariant
    - # Parameters independent of input size
    - Local dependencies dominate
- **Limitation**: Convolution, pooling defined only for regular grids

Introduction

- **CNNs**: state-of-the art on image, speech, video recognition tasks
  - (Shared) Filters are translation invariant
  - # Parameters independent of input size
  - Local dependencies dominate
- **Limitation**: Convolution, pooling defined only for regular grids
  - Not defined for irregular/non-Euclidean domains

Introduction

- **CNNs**: state-of-the art on image, speech, video recognition tasks
  - (Shared) Filters are translation invariant
  - # Parameters independent of input size
  - Local dependencies dominate
- **Limitation**: Convolution, pooling defined only for regular grids
  - Not defined for irregular/non-Euclidean domains
- **Contribution**: Define convolution, pooling on graphs

Introduction

- **CNNs**: state-of-the art on image, speech, video recognition tasks
    - (Shared) Filters are translation invariant
    - # Parameters independent of input size
    - Local dependencies dominate
- **Limitation**: Convolution, pooling defined only for regular grids
    - Not defined for irregular/non-Euclidean domains
- **Contribution**: Define convolution, pooling on graphs
    - User data on social networks
    - Gene data on biological regulatory networks
    - Log data on telecommunication networks
    - Text documents on word embeddings

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

- **Convolutional Neural Networks on Graphs**

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

- **Convolutional Neural Networks on Graphs**
  - Define convolution and pooling operations on graphs

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

- **Convolutional Neural Networks on Graphs**
  - Define convolution and pooling operations on graphs
- **Fast**

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

- **Convolutional Neural Networks on Graphs**
  - Define convolution and pooling operations on graphs
- **Fast**: same complexity as CNNs

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

- **Convolutional Neural Networks on Graphs**
  - Define convolution and pooling operations on graphs
- **Fast**: same complexity as CNNs
  - Linear computational complexity

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

- **Convolutional Neural Networks on Graphs**
  - Define convolution and pooling operations on graphs
- **Fast**: same complexity as CNNs
  - Linear computational complexity
  - Constant learning complexity

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

- **Convolutional Neural Networks on Graphs**
  - Define convolution and pooling operations on graphs
- **Fast**: same complexity as CNNs
  - Linear computational complexity
  - Constant learning complexity
- **Localized Spectral Filtering**

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

- **Convolutional Neural Networks on Graphs**
  - Define convolution and pooling operations on graphs
- **Fast**: same complexity as CNNs
  - Linear computational complexity
  - Constant learning complexity
- **Localized Spectral Filtering**
  - Ideas from spectral graph theory

Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

- **Convolutional Neural Networks on Graphs**
  - Define convolution and pooling operations on graphs
- **Fast**: same complexity as CNNs
  - Linear computational complexity
  - Constant learning complexity
- **Localized Spectral Filtering**
  - Ideas from spectral graph theory
  - Filters provable to be strictly localized in a ball of radius K

Proposed Technique

1 Design of localized convolutional filters on graphs

Proposed Technique

1 Design of localized convolutional filters on graphs

2 Graph coarsening

Proposed Technique

1 Design of localized convolutional filters on graphs

2 Graph coarsening

- Group similar vertices together

Proposed Technique

1. Design of localized convolutional filters on graphs
2. Graph coarsening
   - Group similar vertices together
   - Build condensed, smaller graphs through these groups

Proposed Technique

1 Design of localized convolutional filters on graphs

2 Graph coarsening
- Group similar vertices together
- Build condensed, smaller graphs through these groups

3 Pooling operation

Proposed Technique

1. Design of localized convolutional filters on graphs
2. Graph coarsening
   - Group similar vertices together
   - Build condensed, smaller graphs through these groups
3. Pooling operation
   - Trade spatial details for higher filter details

Proposed Technique

1 Design of localized convolutional filters on graphs

2 Graph coarsening
   - Group similar vertices together
   - Build condensed, smaller graphs through these groups

3 Pooling operation
   - Trade spatial details for higher filter details

Notations

- **Input**: An undirected, connected, weighted $G = (V, E, W)$

Notations

- **Input**: An undirected, connected, weighted $G = (V, E, W)$
  - **Signal** $x : V \to \mathbb{R}$ regarded as $x \in \mathbb{R}^n$

Notations

- **Input**: An undirected, connected, weighted $G = (V, E, W)$
  - **Signal** $x : V \to \mathbb{R}$ regarded as $x \in \mathbb{R}^n$
- **Graph Laplacian**: $L = I_n - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$

Notations

- **Input**: An undirected, connected, weighted $G = (V, E, W)$
  - **Signal** $x : V \to \mathbb{R}$ regarded as $x \in \mathbb{R}^n$
- **Graph Laplacian**: $L = I_n - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ where $D = diag(\sum_j W_{ij})$

Notations

- **Input**: An undirected, connected, weighted $G = (V, E, W)$
  - **Signal** $x : V \to \mathbb{R}$ regarded as $x \in \mathbb{R}^n$

- **Graph Laplacian**: $L = I_n - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ where $D = diag(\sum_j W_{ij})$
  - Non-negative eigenvalues $\{\lambda_l\}_{l=0}^{n-1}$

Notations

- **Input**: An undirected, connected, weighted $G = (V, E, W)$
  - **Signal** $x : V \to \mathbb{R}$ regarded as $x \in \mathbb{R}^n$
- **Graph Laplacian**: $L = I_n - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ where $D = diag(\sum_j W_{ij})$
  - Non-negative eigenvalues $\{\lambda_l\}_{l=0}^{n-1}$
  - Orthonormal (Fourier) basis $\{u_l\}_{l=0}^{n-1}$

Notations

- **Input**: An undirected, connected, weighted $G = (V, E, W)$
  - **Signal** $x : V \to \mathbb{R}$ regarded as $x \in \mathbb{R}^n$
- **Graph Laplacian**: $L = I_n - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ where $D = diag(\sum_j W_{ij})$
  - Non-negative eigenvalues $\{\lambda_l\}_{l=0}^{n-1}$
  - Orthonormal (Fourier) basis $\{u_l\}_{l=0}^{n-1}$
  - Diagonalization $L = U \Lambda U^T$

Notations

- **Input**: An undirected, connected, weighted $G = (V, E, W)$
  - **Signal** $x : V \to \mathbb{R}$ regarded as $x \in \mathbb{R}^n$
- **Graph Laplacian**: $L = I_n - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ where $D = diag(\sum_j W_{ij})$
  - Non-negative eigenvalues $\{\lambda_l\}_{l=0}^{n-1}$
  - Orthonormal (Fourier) basis $\{u_l\}_{l=0}^{n-1}$
  - Diagonalization $L = U \Lambda U^T$
- **GFT**: $\hat{x} = U^T x$

Notations

- **Input**: An undirected, connected, weighted $G = (V, E, W)$
  - **Signal** $x : V \to \mathbb{R}$ regarded as $x \in \mathbb{R}^n$
- **Graph Laplacian**: $L = I_n - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ where $D = diag(\sum_j W_{ij})$
  - Non-negative eigenvalues $\{\lambda_l\}_{l=0}^{n-1}$
  - Orthonormal (Fourier) basis $\{u_l\}_{l=0}^{n-1}$
  - Diagonalization $L = U \Lambda U^T$
- **GFT**: $\hat{x} = U^T x$
  - Inverse, $x = U\hat{x}$

Spectral Filtering of Graph Signals

- **Convolution operator**: $a * b = U\Big((U^T a) \odot (U^T b)\Big)$

Spectral Filtering of Graph Signals

- **Convolution operator**: $a * b = U\Big((U^T a) \odot (U^T b)\Big)$
  - Defined on fourier domain rather than vertex domain

Spectral Filtering of Graph Signals

- **Convolution operator**: $a * b = U\Big((U^T a) \odot (U^T b)\Big)$
  - Defined on fourier domain rather than vertex domain
- **Filtering**: A signal x is filtered by g_θ as

Spectral Filtering of Graph Signals

- **Convolution operator**: $a * b = U\Big((U^T a) \odot (U^T b)\Big)$
  - Defined on fourier domain rather than vertex domain
- **Filtering**: A signal x is filtered by g_θ as

$$y = g_\theta(L)x$$

Spectral Filtering of Graph Signals

- **Convolution operator**: $a * b = U\Big((U^T a) \odot (U^T b)\Big)$
  - Defined on fourier domain rather than vertex domain
- **Filtering**: A signal x is filtered by g_θ as

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x$$

Spectral Filtering of Graph Signals

- **Convolution operator**: $a * b = U\Big((U^T a) \odot (U^T b)\Big)$
  - Defined on fourier domain rather than vertex domain
- **Filtering**: A signal x is filtered by g_θ as

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = Ug_\theta(\Lambda)U^T x$$

Spectral Filtering of Graph Signals

- **Convolution operator**: $a * b = U\Big((U^T a) \odot (U^T b)\Big)$
  - Defined on fourier domain rather than vertex domain
- **Filtering**: A signal x is filtered by g_θ as

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = Ug_\theta(\Lambda)U^T x$$

- Could use a non-parametric filter i.e. $g_\theta(\Lambda) = diag(\theta)$. But

Spectral Filtering of Graph Signals

- **Convolution operator**: $a * b = U\Big((U^T a) \odot (U^T b)\Big)$
  - Defined on fourier domain rather than vertex domain
- **Filtering**: A signal x is filtered by g_θ as

$$y = g_\theta(L)x = g_\theta(U \Lambda U^T)x = U g_\theta(\Lambda) U^T x$$

- Could use a non-parametric filter i.e. $g_\theta(\Lambda) = diag(\theta)$. But
  - Not localized in space

Spectral Filtering of Graph Signals

- **Convolution operator**: $a * b = U\Big((U^T a) \odot (U^T b)\Big)$
  - Defined on fourier domain rather than vertex domain

- **Filtering**: A signal x is filtered by g_θ as

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = Ug_\theta(\Lambda)U^T x$$

- Could use a non-parametric filter i.e. $g_\theta(\Lambda) = diag(\theta)$. But
  - Not localized in space
  - Learning complexitiy is $O(n)$

Polynomial Parametrization for Localized Filters

- **Polynomial Filter**:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$

[1]Hammond et. al, Wavelets on Graphs via Spectral Graph Theory, Applied and Computational Harmonic Analysis 2011

Polynomial Parametrization for Localized Filters

- **Polynomial Filter**:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$

$\theta \in \mathbb{R}^K$

[1] Hammond et. al, Wavelets on Graphs via Spectral Graph Theory, Applied and Computational Harmonic Analysis 2011

Polynomial Parametrization for Localized Filters

- **Polynomial Filter**:
$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$
$\theta \in \mathbb{R}^K$

- [1]If minimum # edges between i and j is $> K$ then $(L^K)_{i,j} = 0$

[1]Hammond et. al, Wavelets on Graphs via Spectral Graph Theory, Applied and Computational Harmonic Analysis 2011

Polynomial Parametrization for Localized Filters

- **Polynomial Filter**:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$

$\theta \in \mathbb{R}^K$

- [1]If minimum # edges between i and j is $> K$ then $(L^K)_{i,j} = 0$
  - Filters are K-localized

[1]Hammond et. al, Wavelets on Graphs via Spectral Graph Theory, Applied and Computational Harmonic Analysis 2011

Polynomial Parametrization for Localized Filters

- **Polynomial Filter**:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$

$\theta \in \mathbb{R}^K$

- [1]If minimum # edges between i and j is $> K$ then $(L^K)_{i,j} = 0$
  - Filters are K-localized

- Learning complexity is $O(K)$ where K is support size of filter

[1]Hammond et. al, Wavelets on Graphs via Spectral Graph Theory, Applied and Computational Harmonic Analysis 2011

Polynomial Parametrization for Localized Filters

- **Polynomial Filter**:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k$$

$\theta \in \mathbb{R}^K$

- [1]If minimum # edges between i and j is $> K$ then $(L^K)_{i,j} = 0$
  - Filters are K-localized
- Learning complexity is $O(K)$ where K is support size of filter
  - Same as classical CNNs

[1]Hammond et. al, Wavelets on Graphs via Spectral Graph Theory, Applied and Computational Harmonic Analysis 2011

Recursive Formulation for Fast Filtering

- Multiplication with U in $y = Ug_\theta(\Lambda)U^T x$ costs $O(n^2)$ operations

Recursive Formulation for Fast Filtering

- Multiplication with U in $y = Ug_\theta(\Lambda)U^T x$ costs $O(n^2)$ operations
- **Solution**: Exploit sparsity of L

Recursive Formulation for Fast Filtering

- Multiplication with U in $y = Ug_\theta(\Lambda)U^T x$ costs $O(n^2)$ operations
- **Solution**: Exploit sparsity of L
  - **Idea**: Compute polynomial recursively from L

Recursive Formulation for Fast Filtering

- Multiplication with U in $y = Ug_\theta(\Lambda)U^T x$ costs $O(n^2)$ operations
- **Solution**: Exploit sparsity of L
  - **Idea**: Compute polynomial recursively from L
  - Use Chebyshev expansion

Recursive Formulation for Fast Filtering

- Multiplication with U in $y = Ug_\theta(\Lambda)U^T x$ costs $O(n^2)$ operations
- **Solution**: Exploit sparsity of L
  - **Idea**: Compute polynomial recursively from L
  - Use Chebyshev expansion
  - $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0 = 1$ and $T_1 = x$

Recursive Formulation for Fast Filtering

- Multiplication with U in $y = Ug_\theta(\Lambda)U^T x$ costs $O(n^2)$ operations
- **Solution**: Exploit sparsity of L
  - **Idea**: Compute polynomial recursively from L
  - Use Chebyshev expansion
  - $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0 = 1$ and $T_1 = x$
- $$g_\theta(L) = \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{L})$$

Recursive Formulation for Fast Filtering

- Multiplication with U in $y = Ug_\theta(\Lambda)U^Tx$ costs $O(n^2)$ operations
- **Solution**: Exploit sparsity of L
  - **Idea**: Compute polynomial recursively from L
  - Use Chebyshev expansion
  - $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0 = 1$ and $T_1 = x$
- $$g_\theta(L) = \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{L})$$
  - $\widetilde{L} = 2L/\lambda_{max} - I_n$

Recursive Formulation for Fast Filtering

- Multiplication with U in $y = Ug_\theta(\Lambda)U^T x$ costs $O(n^2)$ operations
- **Solution**: Exploit sparsity of L
  - **Idea**: Compute polynomial recursively from L
  - Use Chebyshev expansion
  - $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0 = 1$ and $T_1 = x$
- $$g_\theta(L) = \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{L})$$
  - $\widetilde{L} = 2L/\lambda_{max} - I_n$
  - Entire filtering takes $O(K|E|)$ operations (linear in $|E|$)

Recursive Formulation for Fast Filtering

- Multiplication with U in $y = Ug_\theta(\Lambda)U^T x$ costs $O(n^2)$ operations
- **Solution**: Exploit sparsity of L
  - **Idea**: Compute polynomial recursively from L
  - Use Chebyshev expansion
  - $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0 = 1$ and $T_1 = x$
-
$$g_\theta(L) = \sum_{k=0}^{K-1} \theta_k T_k(\widetilde{L})$$
  - $\widetilde{L} = 2L/\lambda_{max} - I_n$
  - Entire filtering takes $O(K|E|)$ operations (linear in $|E|$)
  - $y = g_\theta(L)x = \left[x'_0, \cdots, x'_{K-1}\right]\theta$ where $x'_k = T_k(\widetilde{L})x$

Learning Filters

- Use mini-batch GD (S samples in each batch)

Learning Filters

- Use mini-batch GD (S samples in each batch)
  - j-th feature of sample s is

$$y_{s,j} = \sum_{i=1}^{F_{in}} g_{\theta_{i,j}}(L) x_{s,i}$$

Learning Filters

- Use mini-batch GD (S samples in each batch)
  - j-th feature of sample s is

$$y_{s,j} = \sum_{i=1}^{F_{in}} g_{\theta_{i,j}}(L) x_{s,i}$$

- Backprop requires

$$\frac{\partial J}{\partial \theta_{i,j}} = \sum_{s=1}^{S} \left[x'_{s,i,0}, \cdots, x'_{s,i,K-1} \right] \frac{\partial J}{\partial y_{s,j}}$$

Learning Filters

- Use mini-batch GD (S samples in each batch)
  - j-th feature of sample s is

$$y_{s,j} = \sum_{i=1}^{F_{in}} g_{\theta_{i,j}}(L) x_{s,i}$$

- Backprop requires

$$\frac{\partial J}{\partial \theta_{i,j}} = \sum_{s=1}^{S} \left[x'_{s,i,0}, \cdots, x'_{s,i,K-1} \right] \frac{\partial J}{\partial y_{s,j}}$$

and

$$\frac{\partial J}{\partial x_{s,i}} = \sum_{j=1}^{F_{out}} g_{\theta_{i,j}}(L) \frac{\partial J}{\partial y_{s,j}}$$

Learning Filters

- Use mini-batch GD (S samples in each batch)
  - j-th feature of sample s is

$$y_{s,j} = \sum_{i=1}^{F_{in}} g_{\theta_{i,j}}(L) x_{s,i}$$

- Backprop requires

$$\frac{\partial J}{\partial \theta_{i,j}} = \sum_{s=1}^{S} \left[x'_{s,i,0}, \cdots, x'_{s,i,K-1} \right] \frac{\partial J}{\partial y_{s,j}}$$

and

$$\frac{\partial J}{\partial x_{s,i}} = \sum_{j=1}^{F_{out}} g_{\theta_{i,j}}(L) \frac{\partial J}{\partial y_{s,j}}$$

- J is cross-entropy with l_2 regularization on weights

Learning Filters

- Use mini-batch GD (S samples in each batch)
  - j-th feature of sample s is

$$y_{s,j} = \sum_{i=1}^{F_{in}} g_{\theta_{i,j}}(L) x_{s,i}$$

- Backprop requires

$$\frac{\partial J}{\partial \theta_{i,j}} = \sum_{s=1}^{S} \left[x'_{s,i,0}, \cdots, x'_{s,i,K-1} \right] \frac{\partial J}{\partial y_{s,j}}$$

and

$$\frac{\partial J}{\partial x_{s,i}} = \sum_{j=1}^{F_{out}} g_{\theta_{i,j}}(L) \frac{\partial J}{\partial y_{s,j}}$$

- J is cross-entropy with l_2 regularization on weights
- Requires $O(K|E|F_{in}F_{out}S)$ operations

Graph Coarsening

- Pooling requires meaningful neighbourhoods on graphs

[2] Dhillon et. al, Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach, PAMI 2007

Graph Coarsening

- Pooling requires meaningful neighbourhoods on graphs
  - Cluster similar vertices together

[2] Dhillon et. al, Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach, PAMI 2007

Graph Coarsening

- Pooling requires meaningful neighbourhoods on graphs
  - Cluster similar vertices together
  - But graph clustering is NP-hard

[2] Dhillon et. al, Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach, PAMI 2007

Graph Coarsening

- Pooling requires meaningful neighbourhoods on graphs
  - Cluster similar vertices together
  - But graph clustering is NP-hard
- Graclus [2]graph clustering software

[2]Dhillon et. al, Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach, PAMI 2007

Graph Coarsening

- Pooling requires meaningful neighbourhoods on graphs
  - Cluster similar vertices together
  - But graph clustering is NP-hard
- Graclus [2]graph clustering software
  - Uses a greedy algorithm

[2] Dhillon et. al, Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach, PAMI 2007

Graph Coarsening

- Pooling requires meaningful neighbourhoods on graphs
  - Cluster similar vertices together
  - But graph clustering is NP-hard
- Graclus [2]graph clustering software
  - Uses a greedy algorithm
  - Pick an umarked i and match with one of its unmarked neighbours j that maximizes normalized cut

$$W_{ij}\left(\frac{1}{d_i} + \frac{1}{d_j}\right)$$

[2]Dhillon et. al, Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach, PAMI 2007

Graph Coarsening

- Pooling requires meaningful neighbourhoods on graphs
  - Cluster similar vertices together
  - But graph clustering is NP-hard
- Graclus [2]graph clustering software
  - Uses a greedy algorithm
  - Pick an umarked i and match with one of its unmarked neighbours j that maximizes normalized cut

$$W_{ij}\left(\frac{1}{d_i}+\frac{1}{d_j}\right)$$

  - Repeat until all nodes explored

[2] Dhillon et. al, Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach, PAMI 2007

Graph Coarsening

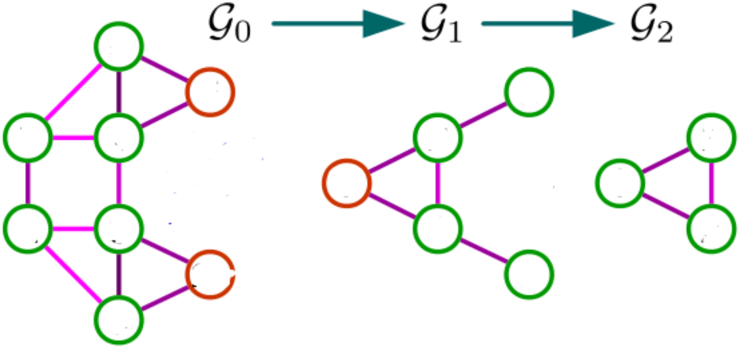
- Pooling requires meaningful neighbourhoods on graphs
  - Cluster similar vertices together
  - But graph clustering is NP-hard
- Graclus [2]graph clustering software
  - Uses a greedy algorithm
  - Pick an umarked i and match with one of its unmarked neighbours j that maximizes normalized cut

$$W_{ij}\left(\frac{1}{d_i}+\frac{1}{d_j}\right)$$

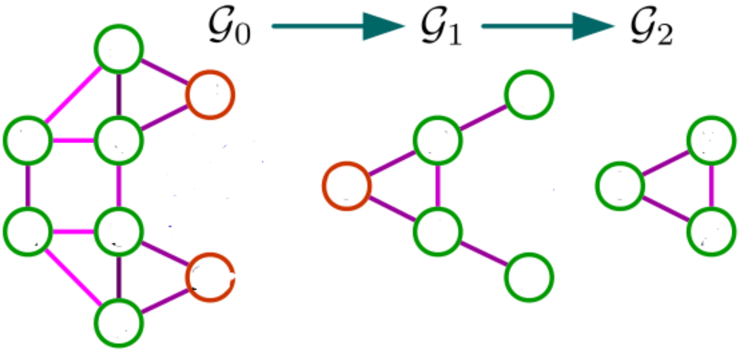
  - Repeat until all nodes explored
- Divides # nodes by approximately 2 (there may exist a few singletons, non-matched nodes)

[2]Dhillon et. al, Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach, PAMI 2007

Graph Coarsening: Example

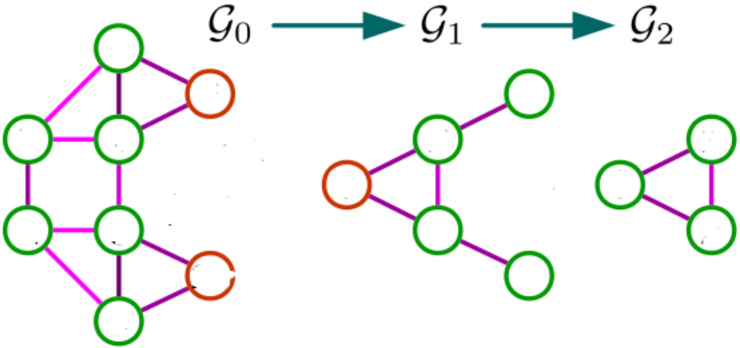


Graph Coarsening: Example



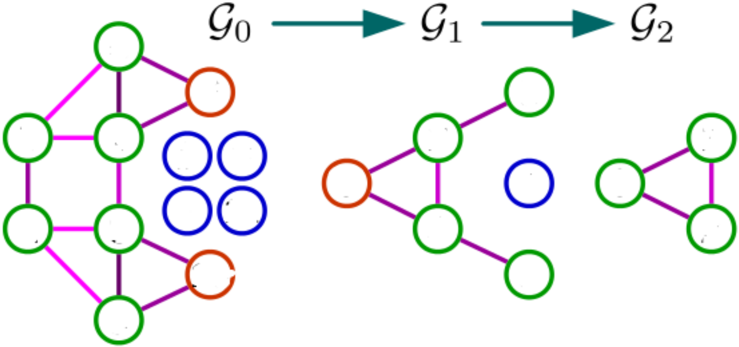
Graclus produces G_1 and G_2 on G_0 as input

Max Pooling: Example



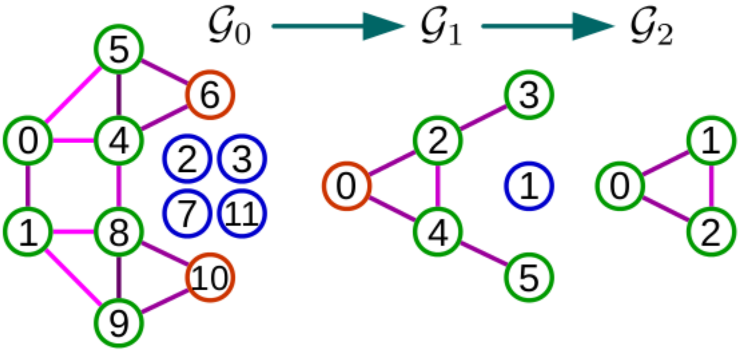
Suppose max pooling of size 4 needs to be carried out (Note: $x \in \mathbb{R}^8$)

Max Pooling: Example



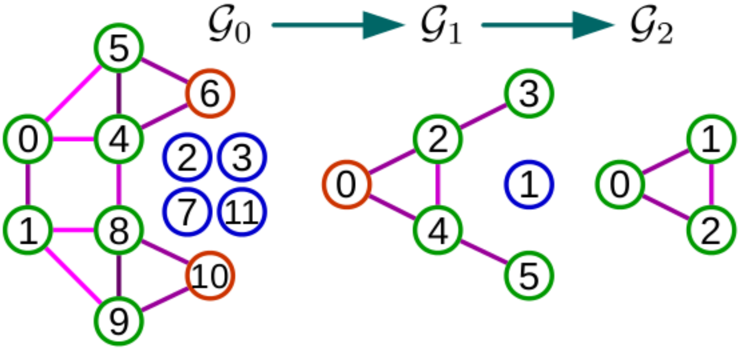
Add fake nodes to pair with singleton nodes

Max Pooling: Example



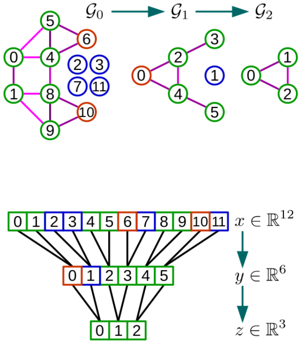
Nodes in V_2: ordered arbitrarily, nodes in V_1, V_0: ordered consequently

Max Pooling: Example

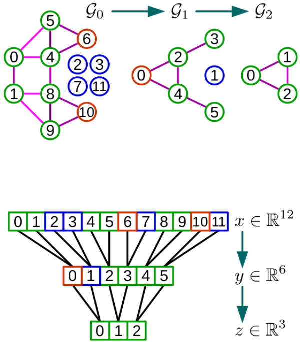


Observation: Node k has $2k$ and $2k+1$ as children

Max Pooling: Example

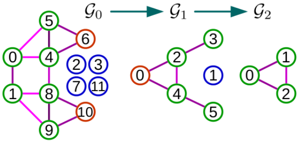


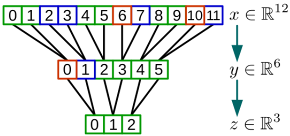
Max Pooling: Example



- $z = \Big(max(x_0, x_1), max(x_4, x_5, x_6), max(x_8, x_9, x_{10})\Big) \in \mathbb{R}^3$

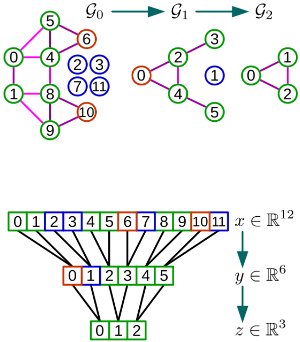
Max Pooling: Example





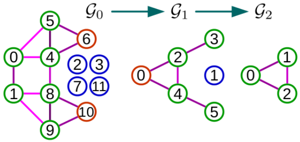
- $z = \Big(max(x_0, x_1), max(x_4, x_5, x_6), max(x_8, x_9, x_{10})\Big) \in \mathbb{R}^3$
  - x_2, x_3, x_7, x_{11} are set to a neutral value

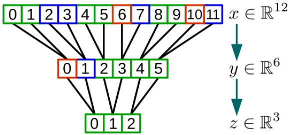
Max Pooling: Example



- $z = \Big(max(x_0, x_1), max(x_4, x_5, x_6), max(x_8, x_9, x_{10}) \Big) \in \mathbb{R}^3$
  - x_2, x_3, x_7, x_{11} are set to a neutral value
  - Analogous to pooling a regular 1D signal

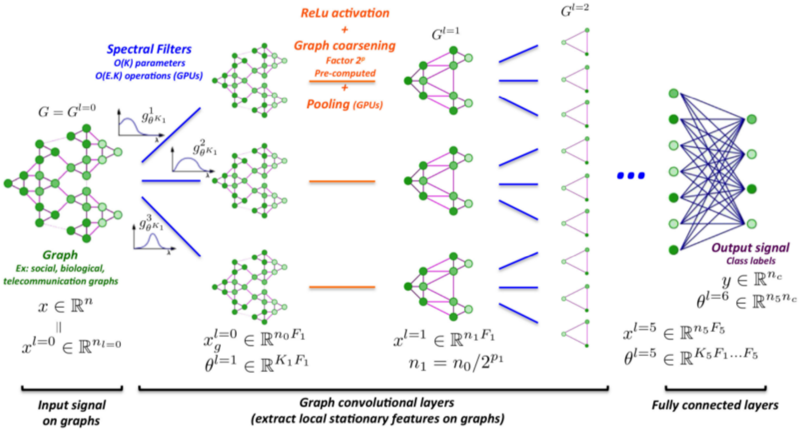
Max Pooling: Example





- $z = \Big(max(x_0, x_1), max(x_4, x_5, x_6), max(x_8, x_9, x_{10}) \Big) \in \mathbb{R}^3$
    - x_2, x_3, x_7, x_{11} are set to a neutral value
    - Analogous to pooling a regular 1D signal
    - Very efficient, satisfies parallel architectures such as GPUs as memory accesses are local

Architecture



Experiments: MNIST as a Sanity Check

- Construct an 8-NN graph

Experiments: MNIST as a Sanity Check

- Construct an 8-NN graph
  - 70000 digits on 2D grid becomes graph with 976 ($28^2 + 192$) nodes

Experiments: MNIST as a Sanity Check

- Construct an 8-NN graph
  - 70000 digits on 2D grid becomes graph with 976 ($28^2 + 192$) nodes
  - Weights of k-NN similarity graph are

$$W_{ij} = exp(-\frac{||z_i - z_j||_2^2}{\sigma^2})$$

  z_i is 2D coordinate of pixel i

Experiments: MNIST as a Sanity Check

- Construct an 8-NN graph
  - 70000 digits on 2D grid becomes graph with 976 ($28^2 + 192$) nodes
  - Weights of k-NN similarity graph are

$$W_{ij} = exp(-\frac{||z_i - z_j||_2^2}{\sigma^2})$$

z_i is 2D coordinate of pixel i

Model	Architecture	Accuracy
Classical CNN	C32-P4-C64-P4-FC512	99.33
Proposed graph CNN	GC32-P4-GC64-P4-FC512	99.14

- Ck: Convolutional layer k feature maps
- GCk: Graph convolutional layer k feature maps
- FCk: Fully connected layer k hidden units
- Pk: Pooling layer size, stride k

Text Categorization on 20NEWS

- To demonstrate versatility of model to work with unstructured data

Text Categorization on 20NEWS

- To demonstrate versatility of model to work with unstructured data
  - Extract $10k$ most common words from $93k+$ unique words

Text Categorization on 20NEWS

- To demonstrate versatility of model to work with unstructured data
  - Extract $10k$ most common words from $93k+$ unique words
  - Represent doc x by bag-of-words model (normalized across words)

Text Categorization on 20NEWS

- To demonstrate versatility of model to work with unstructured data
  - Extract $10k$ most common words from $93k+$ unique words
  - Represent doc x by bag-of-words model (normalized across words)
  - Construct a 16-NN graph with $W_{ij} = exp(-\frac{||z_i - z_j||_2^2}{\sigma^2})$ where z_i is word2vec embedding of word i

Text Categorization on 20NEWS

- To demonstrate versatility of model to work with unstructured data
  - Extract $10k$ most common words from $93k+$ unique words
  - Represent doc x by bag-of-words model (normalized across words)
  - Construct a 16-NN graph with $W_{ij} = exp(-\frac{||z_i - z_j||_2^2}{\sigma^2})$ where z_i is word2vec embedding of word i

Model	Accuracy
Linear SVM	65.90
Multinomial Naive Bayes	68.51
Softmax	66.28
FC2500	64.64
FC2500-FC500	65.76
GC32	68.26

Text Categorization on 20NEWS

- To demonstrate versatility of model to work with unstructured data
  - Extract $10k$ most common words from $93k+$ unique words
  - Represent doc x by bag-of-words model (normalized across words)
  - Construct a 16-NN graph with $W_{ij} = exp(-\frac{||z_i - z_j||_2^2}{\sigma^2})$ where z_i is word2vec embedding of word i

Model	Accuracy
Linear SVM	65.90
Multinomial Naive Bayes	68.51
Softmax	66.28
FC2500	64.64
FC2500-FC500	65.76
GC32	68.26

- All models trained 20 epochs by Adam optimizer (initial learning rate $= 0.001$)

Text Categorization on 20NEWS

- To demonstrate versatility of model to work with unstructured data
  - Extract $10k$ most common words from $93k+$ unique words
  - Represent doc x by bag-of-words model (normalized across words)
  - Construct a 16-NN graph with $W_{ij} = exp(-\frac{||z_i - z_j||_2^2}{\sigma^2})$ where z_i is word2vec embedding of word i

Model	Accuracy
Linear SVM	65.90
Multinomial Naive Bayes	68.51
Softmax	66.28
FC2500	64.64
FC2500-FC500	65.76
GC32	68.26

- All models trained 20 epochs by Adam optimizer (initial learning rate $= 0.001$)
- Support of $GC32$ is $K = 5$

Text Categorization on 20NEWS

- To demonstrate versatility of model to work with unstructured data
  - Extract $10k$ most common words from $93k+$ unique words
  - Represent doc x by bag-of-words model (normalized across words)
  - Construct a 16-NN graph with $W_{ij} = exp(-\frac{||z_i - z_j||_2^2}{\sigma^2})$ where z_i is word2vec embedding of word i

Model	Accuracy
Linear SVM	65.90
Multinomial Naive Bayes	68.51
Softmax	66.28
FC2500	64.64
FC2500-FC500	65.76
GC32	68.26

- All models trained 20 epochs by Adam optimizer (initial learning rate = 0.001)
- Support of $GC32$ is $K = 5$
- Proposed model does not outperform multinomial Bayes but defeats FC networks which require much more parameters

Text Categorization on 20NEWS

- To demonstrate versatility of model to work with unstructured data
  - Extract $10k$ most common words from $93k+$ unique words
  - Represent doc x by bag-of-words model (normalized across words)
  - Construct a 16-NN graph with $W_{ij} = exp(-\frac{||z_i - z_j||_2^2}{\sigma^2})$ where z_i is word2vec embedding of word i

Model	Accuracy
Linear SVM	65.90
Multinomial Naive Bayes	68.51
Softmax	66.28
FC2500	64.64
FC2500-FC500	65.76
GC32	68.26

- All models trained 20 epochs by Adam optimizer (initial learning rate $= 0.001$)
- Support of $GC32$ is $K = 5$
- Proposed model does not outperform multinomial Bayes but defeats FC networks which require much more parameters

Comparison of Spectral Filters

- The first formulation[3] of graph CNN defined filter as $g_\theta(\Lambda) = B\theta$

[3] Bruna et. al, Spectral Networks and Deep Locally Connected Networks on Graphs

Comparison of Spectral Filters

- The first formulation[3] of graph CNN defined filter as $g_\theta(\Lambda) = B\theta$

[3] Bruna et. al, Spectral Networks and Deep Locally Connected Networks on Graphs

Comparison of Spectral Filters

- The first formulation[3] of graph CNN defined filter as $g_\theta(\Lambda) = B\theta$ where $B \in \mathbb{R}^{n \times K}$ is cubic B-spline basis

[3] Bruna et. al, Spectral Networks and Deep Locally Connected Networks on Graphs

Comparison of Spectral Filters

- The first formulation[3] of graph CNN defined filter as $g_\theta(\Lambda) = B\theta$ where $B \in \mathbb{R}^{n \times K}$ is cubic B-spline basis

		Accuracy		
Dataset	Architecture	Non-Param	Spline	Chebyshev
MNIST	GC10	95.75	97.26	97.48
MNIST	GC32-P4-GC64-P4-FC512	96.28	97.15	99.14

[3] Bruna et. al, Spectral Networks and Deep Locally Connected Networks on Graphs

- The first formulation[3] of graph CNN defined filter as $g_\theta(\Lambda) = B\theta$ where $B \in \mathbb{R}^{n \times K}$ is cubic B-spline basis

		Accuracy		
Dataset	Architecture	Non-Param	Spline	Chebyshev
MNIST	GC10	95.75	97.26	97.48
MNIST	GC32-P4-GC64-P4-FC512	96.28	97.15	99.14

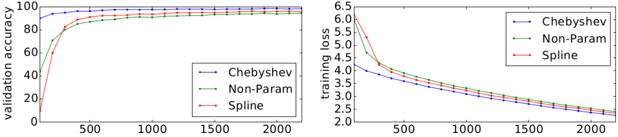


Figure 4: Plots of validation accuracy and training loss for the first 2000 iterations on MNIST.

[3] Bruna et. al, Spectral Networks and Deep Locally Connected Networks on Graphs

Comparison of Computational Efficiency

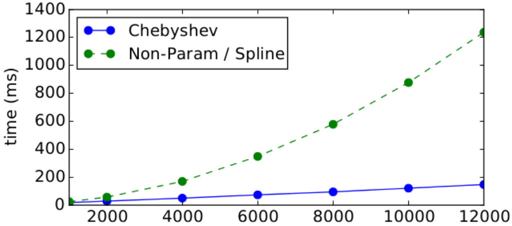


Figure 3: Time to process a mini-batch of $S = 100$ 20NEWS documents w.r.t. the number of words n.

Comparison of Computational Efficiency

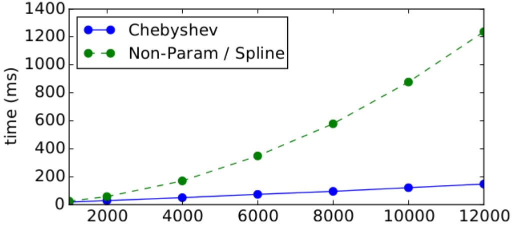


Figure 3: Time to process a mini-batch of $S = 100$ 20NEWS documents w.r.t. the number of words n.

- $O(n)$ vs. $O(n^2)$

Comparison of Computational Efficiency

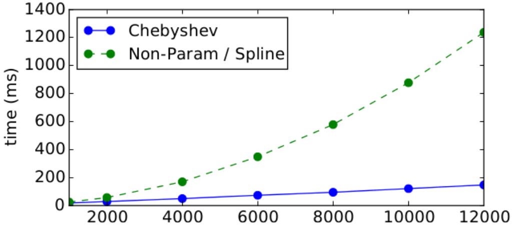


Figure 3: Time to process a mini-batch of $S = 100$ 20NEWS documents w.r.t. the number of words n.

- $O(n)$ vs. $O(n^2)$
- Measured runtime is total training time divided by #gradient steps

Comparison of Computational Efficiency

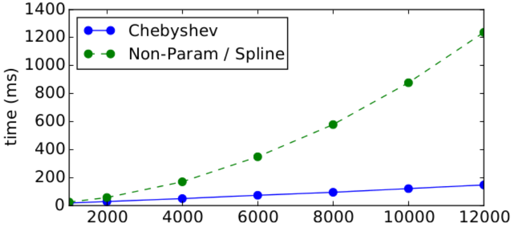


Figure 3: Time to process a mini-batch of $S = 100$ 20NEWS documents w.r.t. the number of words n.

- $O(n)$ vs. $O(n^2)$
- Measured runtime is total training time divided by #gradient steps

Model	Architecture	Time (ms) CPU	Time (ms) GPU	Speedup
Classical CNN	C32-P4-C64-P4-FC512	210	31	6.77x
Proposed graph CNN	GC32-P4-GC64-P4-FC512	1600	200	8.00x

Table 4: Time to process a mini-batch of $S = 100$ MNIST images.

Conclusion

- Introduced mathematical and computational foundations of efficient generalization of CNNs on graphs

Conclusion

- Introduced mathematical and computational foundations of efficient generalization of CNNs on graphs
- Introduced model whose computational complexity linear with dimensionality of data

Conclusion

- Introduced mathematical and computational foundations of efficient generalization of CNNs on graphs
- Introduced model whose computational complexity linear with dimensionality of data
- Experiments have shown ability of model to extract local, stationary features through convolutional layers