Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering

Kimon Fountoulakis

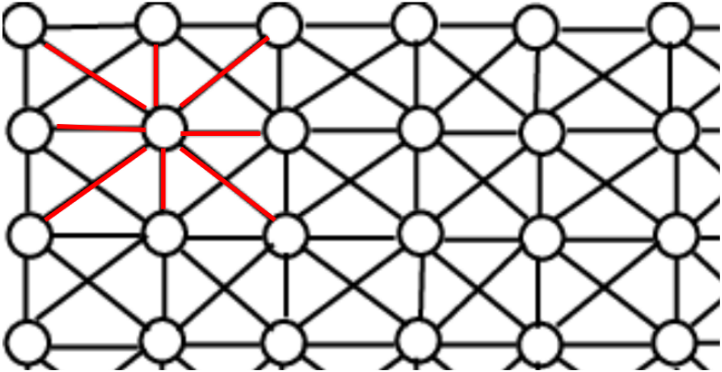
January 2024

Introduction

- We saw before graphs can capture relationship between datapoints
- Graphs can also capture relationship between features
- For this talk, we will assume our graph has n vertices, each associated with a feature
- $x \in \mathbb{R}^n$ is the vector of features for a single data point

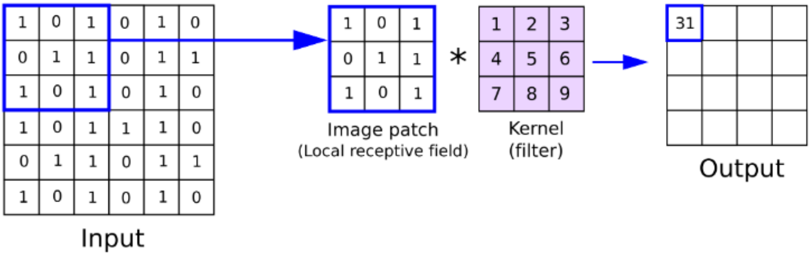
Introduction

- **Example:** Image data: each feature is a pixel. Graph is the grid-graph over pixels



Convolutional Neural Network

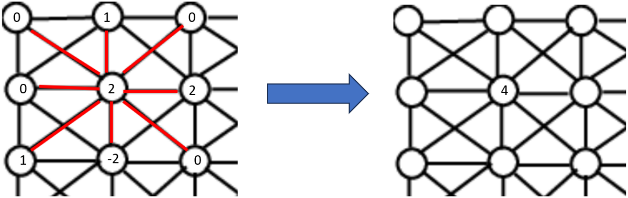
- Convolutional Neural Network capture local structures by looking at relationship between nearby pixels



$$1 \times 1 + 0 \times 2 + 1 \times 3 + 0 \times 4 + 1 \times 5 + 1 \times 6 + 1 \times 7 + 0 \times 8 + 1 \times 9 = 31$$

CNN is a Spatial Graph Convolution

- Sum feature value around neighborhood of vertex
- Can be interpreted as the map $x \mapsto Ax$ where A is the adjacency matrix
- CNN with uniform Kernel = spatial convolution on grid graph



$$0 \times 1 + 1 \times 1 + 0 \times 1 + 0 \times 1 + 2 \times 1 + 2 \times 1 + 1 - 2 \times 1 + 0 \times 1 = 4$$

Question

- Sometimes, relationship between features are captured by arbitrary graphs (ex: temperature collected from weather stations)
- Since graph is not a grid graph, cannot apply CNN
- Question: can we generalize CNNs to arbitrary graphs?

(c) Weather stations across the U.S.

Why is it Important?

- CNNs are perform very well on Images
- CNNs are scalable because they require few training parameters
- Is Natural to try to generalize to aribtrary graphs

Why don't Previous Solution Work

- Vanilla GCN has no learnable kernel like in CNN
- Can add learned edgeweights through graph attention etc. But...

  1. Number of learned parameters = number of edges, which is too high
  2. In CNN Kernel size is much smaller than n

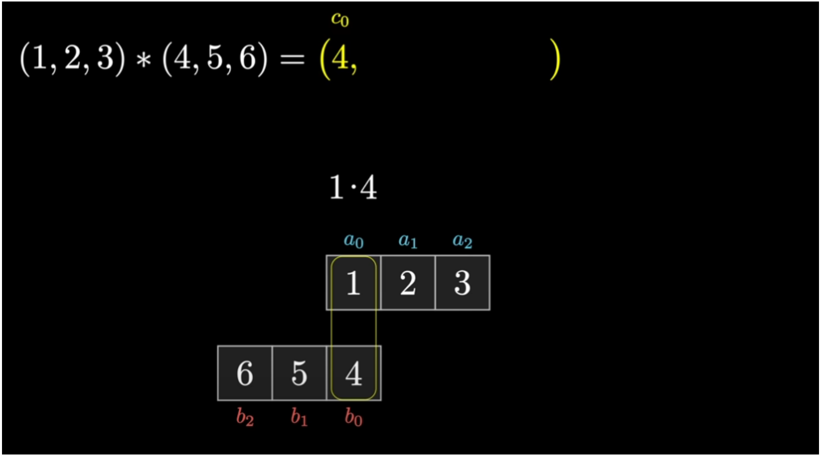
Signals and Convolutions (in discrete terms)

- We can interpret a **signal** as a vector $x \in \mathbb{R}^n$
- let $g \in \mathbb{R}^k$ be a filter. Then the convolution $g * x \in \mathbb{R}^{n+k}$ is defined by

$$g * x(t) = \sum_{i=1}^{t} x(i)g(t-i)$$

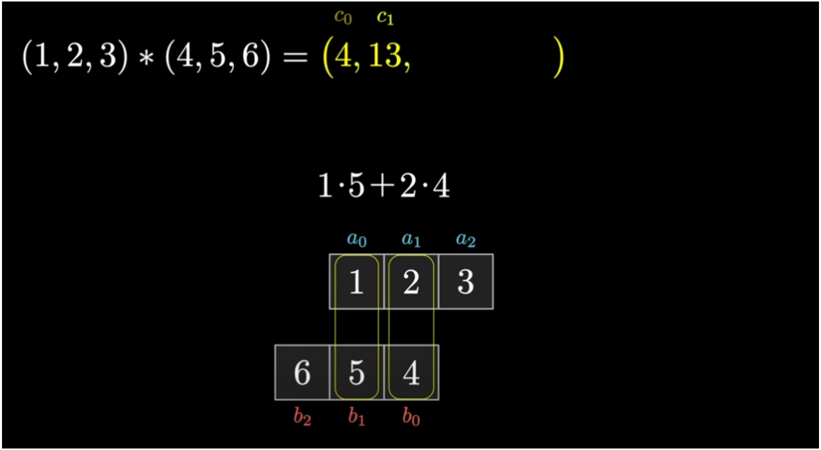
Example: (from 3Blue1Brown)

- $a = (1, 2, 3)$ and $b = (4, 5, 6)$



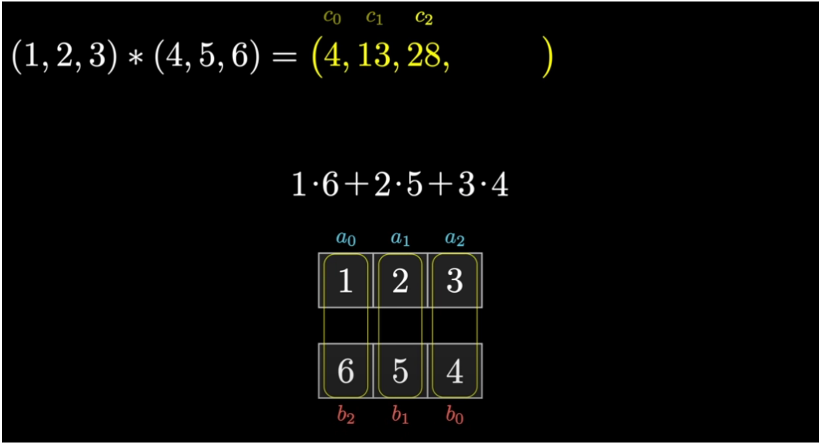
Example: (from 3Blue1Brown)

- $a = (1, 2, 3)$ and $b = (4, 5, 6)$



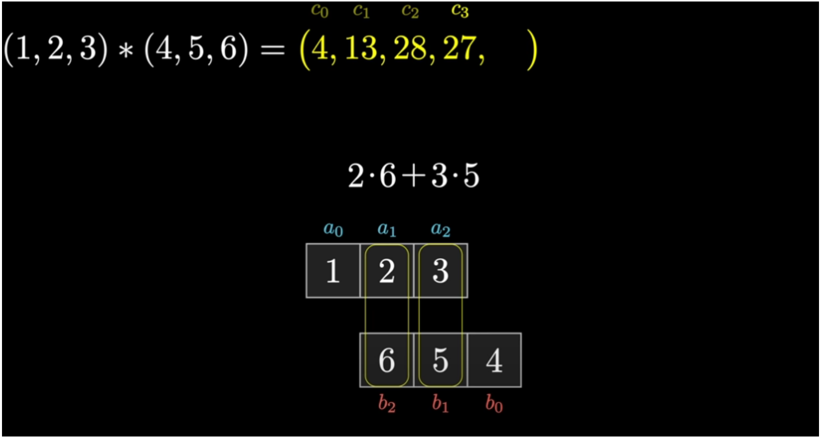
Example: (from 3Blue1Brown)

- $a = (1, 2, 3)$ and $b = (4, 5, 6)$



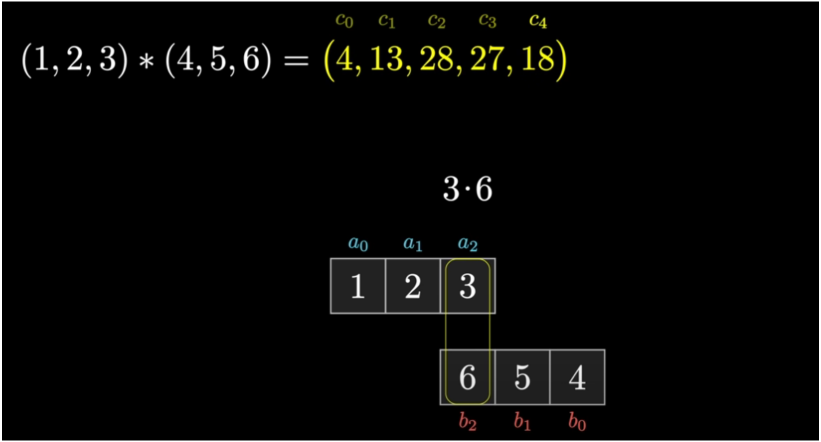
Example: (from 3Blue1Brown)

- $a = (1, 2, 3)$ and $b = (4, 5, 6)$



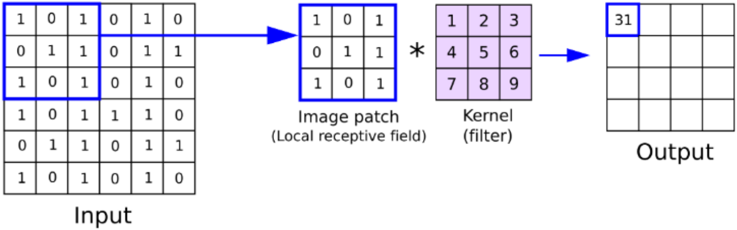
Example: (from 3Blue1Brown)

- $a = (1, 2, 3)$ and $b = (4, 5, 6)$



Example: CNN

- CNNs are prominent examples of discrete convolutions
- x = Input, g = Kernel, $g * x$ = Output
- The filter/Kernel contains learned parameters



Discrete Fourier Transform (Informal)

- **Fourier Basis:** a set of n orthonormal vectors $b_1, ...b_n$
- We can also express x as a linear combination of an orthonormal fourier basis i.e. $x = \sum_i c_i b_i$
- c_i's are **Fourier Coefficients** of x
- **Fourier Transform** maps Φ is a change of basis from standard basis to Fourier basis

$$\Phi : (x(1), x(2), ...x(n)) = (c_1, c_2, ...c_n)$$

Fourier Transform and Convolution

- **Theorem:** Convolution in the standard basis in is equalled to point-wise vector multiplication in the fourier basis

$$\Phi(g * x) = \Phi g \circ \Phi x$$

Where $u \circ v(t) = u(t)v(t)$ denotes the point-wise vector product

- Since Fourier Transforms are invertible, we have

$$g * x = \Phi^{-1}(\Phi g \circ \Phi x)$$

Example of Convolution Theorem

- say x is a vector of coefficients for polynomial

$$p_x(y) = x(0) + x(1)y + ...x(n)y^{n-1}$$

- $\Phi(x)$ is the vector of evaluating the polynomial at some n points. Let's say...the n roots of unity

$$\Phi x = (p_x(e^{2\pi i/n}), p_x(e^{4\pi i/n}), ...p(e^{2\pi i}))$$

- Multiplying two polynomials means...
  1. Convolution in terms of their coefficients
  2. Point-wise Multiplications in terms of their values

Graph Fourier Transform

Given a graph, we want to find a Fourier basis that captures important information about the graph

Graph Laplacian

- **Adjacency Matrix:** A
- **Normalized Adjacency Matrix:** $A_N = D^{-1/2} A D^{-1/2}$, $D =$ diagonal matrix of degrees
- **Spectral Theorem:** Any $n \times n$ real-symmetric matrix M can be diagonalized with real eigenvalues and n orthogonal eigenvectors.
- Since A and A_N are symmetric, they can be diagonalized orthogonal eigenvectors

Spectral Decomposition

- Let $u_1, ... u_n$ be the eigenvectors of A or A_N
- Let $U = [u_1, u_2, ... u_n]$ be the matrix with with eigenvectors as columns, then we can express

$$A = U \Lambda U^\top$$

$$\Lambda = \begin{bmatrix} \alpha_1 & & & \\ & \alpha_2 & & \\ & & ... & \\ & & & \alpha_n \end{bmatrix}$$

Where $\alpha_1 \geq, ... \alpha_n$ are the eigenvalues

- Note $UU^\top = I$

Discrete Fourier Transform on Graphs

- Pick $u_1, \ldots u_n$ as our Fourier Basis
- **Fourier Transform:** Given $x \in \mathbb{R}^n$, $\Phi x = U^\top x$ and $\Phi^{-1} x = Ux$
- **Spectral Convolution:** given a filter vector $g \in \mathbb{R}^n$, we define $g * x = U\text{Diag}(U^\top g)U^\top x$

Discrete Fourier Transform on Graphs

$$U\text{Diag}(U^\top g)U^\top x$$

$$= U \underbrace{\begin{bmatrix} \langle u_1, g\rangle & & & \\ & \langle u_2, g\rangle & & \\ & & \dots & \\ & & & \langle u_n, g\rangle \end{bmatrix}}_{\text{Fourier Coefficients of } \mathrm{g}} \underbrace{\begin{bmatrix} \langle u_1, x\rangle \\ \langle u_2, x\rangle \\ \dots \\ \langle u_n, x\rangle \end{bmatrix}}_{\text{Fourier Coefficients of } \mathrm{x}}$$

$$= \underbrace{U}_{\text{inverse Fourier Transform}} \underbrace{\begin{bmatrix} \langle u_1, g\rangle\langle u_1, x\rangle \\ \langle u_2, g\rangle\langle u_2, x\rangle \\ \dots \\ \langle u_n, g\rangle\langle u_n, x\rangle \end{bmatrix}}_{\text{point-wise multiplication}}$$

$$= g * x$$

Some Examples:

- By spectral decomp. $Ax = U\Lambda U^\top x = (\alpha_i u_i) * x$ or convolution of x with filter $\sum_i \alpha_i u_i$

- Vanilla GCN is also a spectral convolution with a "fixed" filter defined by eigenvalues

Problem Solution

- Perform spectral convolution with a learned filter
- Let $\theta \in \mathbb{R}^n$ be a vector of learning parameters:
- **Convolutional Layer:**

$$x \mapsto U\mathrm{Diag}(\theta)U^\top x$$

- u_i's are eigenvectors of Normalized Adjacency Matrix
- Runtime: $\tilde{O}(n^2)$ time to compute eigenvectors...too slow, and n training parameters is too many

Polynomial transform

- Let g_θ be a polynomial function parameterized by θ, say...$\theta_0 + \theta_1 x + \theta_2 x^2$
- Polynomial on matrices: $g_\theta(A) = \theta_0 I + \theta_1 A + \theta_2 A^2$
- Applying g_θ to A gives

$$g_\theta(A) = U g_\theta(\Lambda) U^\top = U \begin{bmatrix} g_\theta(\alpha_1) & & & \\ & g_\theta(\alpha_2) & & \\ & & \dots & \\ & & & g_\theta(\alpha_n) \end{bmatrix} U^\top$$

- For example: $A^2 = U\Lambda U^\top U \Lambda U^\top = U\Lambda^2 U^\top$

Localized Spectral Filter

- **Idea:** let g_θ be a degree k polynomial parameterized by $\theta \in \mathbb{R}^k$. Convolutional Layer is $x \mapsto g_\theta(A)x$
- Fourier Convolution: $g_\theta(A)x = Ug_\theta(\Lambda)U^\top x$, fourier coefficients of our leanred filter are $g_\theta(\alpha_i)_{i=1}^n$
- Convolution is **Local**: each vertex only recieves data from vertices at most k-hops away
- $A^k(i,j) =$ number of (or total weight of) paths of length k from i to j
- g_θ is degree k and $A^1(i,j), \dots A^k(i,j) = 0$ if i and j have hop distance greater than k

Localized Spectral Filter with Chebyshev Polynomials

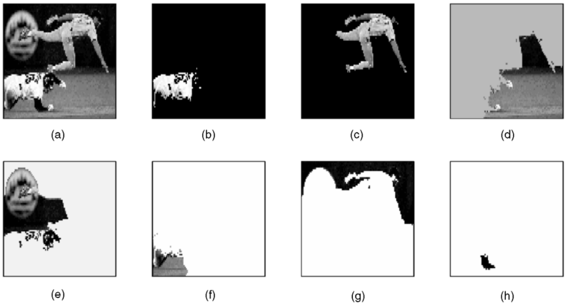
- **Chebyshev Polynonmials:** $T_0 = 1,\ T_1 = x$, $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, easy to compute
- Given k, we take a linear combination of $T_0, \dots T_k$ with learned filter $\theta \in \mathbb{R}^k$
- **Spectral Filter:** $g_\theta(A_N)x = \sum_{1=1}^{K} \theta_i T_i(A_N)x$ (normaized adjacency matrix have eigenvalues in $[-1, 1]$)
- **Computation:** $x^{(k)} = T_k(A_N)x$ and $x^{(i)} = 2A_N x^{(i-1)} - x^{(i-2)}$. Total runtime $= O(K|E|)$

Laplacian Matrix

- In the paper, the matrix they use is slightly different
- Laplacian $L = D - A$, normalized Laplacian $L_N = I - A_N$
- Actual filter they used was $x \mapsto g_\theta(\tilde{L})x$ where $\tilde{L} = \frac{1}{\lambda_{\max}} L - I$
- L_N has same eigenvectors as A_N with eigenvalues $\lambda_1 = 1 - \alpha_1, ...\lambda_n = 1 - \alpha_n$
- Using L_N and A_N are essentially the same

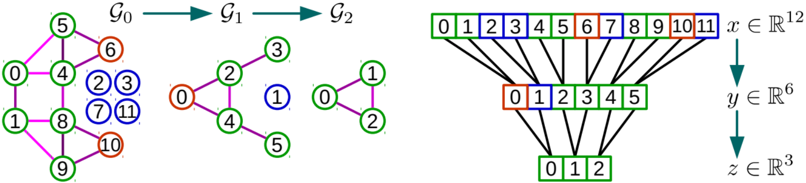
Laplacian Matrix Properties

- Is positive semidefinite, meaning eigenvalues are non-negative
- $\lambda_1 = 0$ and λ_2 is small iff graph can be partitioned into two "balanced" components with few edges in between
- Second eigenvector can be used to find such sparse cuts
- Useful in Image segmentation, finding key features in images

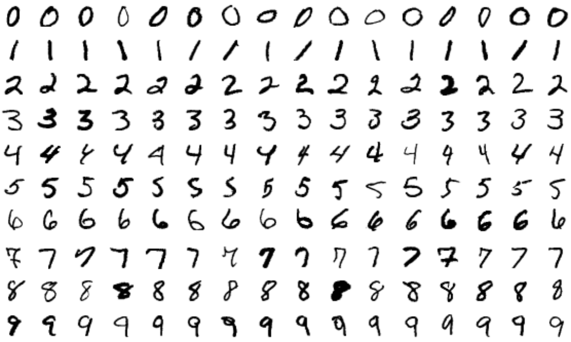


Coarsening and Pooling Layer

- Aggregate neighboring vertices to reduce size and reduce sensitivity to local variation
- Greedily contract edges and take max signal from vertices along contracted edges



Performance on MNIST Data against CNN



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

Table 1: Classification accuracies of the proposed graph CNN and a classical CNN on MNIST.

Comparison with Other Spectral GCN

- Non-Param: $x \mapsto U\text{Diag}(\theta)U^\top x$, $\theta \in \mathbb{R}^n$
- $x \mapsto U\text{Diag}(B\theta)U^\top x$, where B is spline basis

		Accuracy		
Dataset	Architecture	Non-Param (2)	Spline (7) [4]	Chebyshev (4)
MNIST	GC10	95.75	97.26	97.48
MNIST	GC32-P4-GC64-P4-FC512	96.28	97.15	99.14

Table 3: Classification accuracies for different types of spectral filters ($K = 25$).

Runtime Comparison

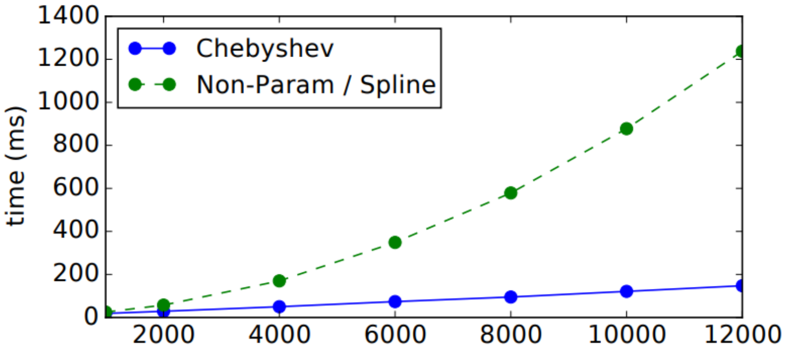


Figure 3: Time to process a mini-batch of $S = 100$ 20NEWS documents w.r.t. the number of words n.

20News Data

- data: set of 18846 texts, want to classify them into 1 of 20 categories
- features: 10,000 most important words, map each word to vector embedding via Word2Vec
- Graph: 16 Nearest-neighbor graph over 10,000 word embedding vertices
- Edge-weights: inverse exponential in embedding vector distance distance

Performance

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

Table 2: Accuracies of the proposed graph CNN and other methods on 20NEWS.

Future Questions

- Can we apply this to datasets whose features are more naturally represented as graphs?
- Can we get better intuition as to what Graph Fourier Transform is doing?
- (Related) what if we just project onto small eigenvectors?