

## Master-Thesis

# Convolutional Neural Networks auf Graphrepräsentationen von Bildern

Matthias Fey

15. März 2017

### **Gutachter:**

Prof. Dr. Heinrich Müller

M.Sc. Jan Eric Lenssen

Lehrstuhl Informatik VII  
Graphische Systeme  
TU Dortmund



# Inhaltsverzeichnis

<b>1. Gedachter Inhalt</b>	<b>1</b>
<b>2. Einleitung</b>	<b>2</b>
2.1. Motivation . . . . .	2
2.2. Aufbau der Arbeit . . . . .	2
<b>3. Grundlagen</b>	<b>3</b>
3.1. Notationen . . . . .	3
3.2. Graphentheorie . . . . .	3
3.3. Faltung in CNNs . . . . .	5
<b>4. Graphrepräsentationen von Bildern</b>	<b>6</b>
4.1. Kantengewichte . . . . .	6
<b>5. Räumliche Graphentheorie</b>	<b>7</b>
5.1. Patchy-SAN . . . . .	7
<b>6. Spektrale Graphentheorie</b>	<b>8</b>
6.1. Einführung . . . . .	8
6.2. Probleme . . . . .	8
6.3. Eigenwerte und Eigenvektoren reell symmetrischer Matrizen . . . . .	8
6.4. Der Laplacian und seine Eigenwerte . . . . .	9
6.4.1. Visuelle Interpretation des Laplacian . . . . .	9
6.4.2. Eigenschaften . . . . .	10
6.5. Faltung mittels Graph-Fourier-Transformation . . . . .	11
6.5.1. Kontinuierliche Fourier-Transformation . . . . .	11
6.5.2. Graph-Fourier-Transformation . . . . .	11
6.5.3. Faltung . . . . .	13
6.6. Polynomielle Approximation . . . . .	14
6.6.1. Tschebyschow-Polynome . . . . .	14
6.7. Graph Convolutional Networks . . . . .	16
6.8. Weisfeiler Lehman Analogie . . . . .	17
6.9. Erweiterung für mehrere Kantenattribute . . . . .	17
6.9.1. Übertragung auf räumlich eingebettete Graphen . . . . .	17

6.10. Pooling-Ebene . . . . .	19
6.10.1. Clustering von Graphen . . . . .	19
6.10.2. Pooling-Operation . . . . .	21
6.10.3. INFORMATIONEN . . . . .	21
6.11. Beispiel . . . . .	21
<b>A. Weitere Informationen</b>	<b>23</b>
<b>Abbildungsverzeichnis</b>	<b>25</b>
<b>Algorithmenverzeichnis</b>	<b>27</b>
<b>Literaturverzeichnis</b>	<b>29</b>

# 1. Gedachter Inhalt

Einleitung: Motivation Aufbau der Arbeit

Grundlagen: Graphen, insbesondere planare Graphen Mathematische Notationen: Vektor, Matrix, Tensor Neuronale Netze (Was ist ein CNN, wie ist der Convolution Operator definiert, nicht lineare Aktivierungsfunktion) Faltung, insbesondere Faltung im CNN

Graphrepräsentationen von Bildern Grid Superpixel Superpixelalgorithmen Umwandlung von Kanten von Distanz zu Gauß Merkmalsextraktion (Momente) Merkmalselektion (Cov, PCA)

Lernen auf Graphen: Stand der Forschung: Spatial vs Spectral

Spatial: Patchy Zentralität Canonical Labeling Übertragung auf planare Graphen <- EIGENER ANTEIL (z.B. Grid Spiral) Komplexität Vorteile (einfache Architektur)/Nachteile (keine direkte Nachbarschaftsberücksichtigung möglich, keine Graph Coarsening möglich, Vorverarbeitung ist recht teuer und muss Preprocessed werden weil man das nicht über Matrixoperationen ausdrücken kann)

Spectral: Laplacian, Fourier Transformation GCN und kGCN (weisfeiler Lehman) Übertragung auf planare Graphen (Adjazenzpartitionierung) <- EIGENER ANTEIL Pooling/Coarsening Komplexität Vorteile (z.B. Nachbarschaftsberücksichtigung/keine Ordnung nötig)/Nachteile (rotationsinvariant)

Deep Learning auf variabler Input-Menge (SPP)

Augmentierung von Graphen (ist das überhaupt möglich) COARSENING IST RANDOM (EINE FORM DER AUGMENTIERUNG) PERMUTATE RANDOM REMOVE RANDOM EDGE

Realisierung (Experimente) und Evaluation Adam-Optimizer Sparse Tensors Vorstellung Datensätze (MNIST, PascalVOC, CIFAR-10, ImageNet) Tensorflow Dropout L2-Regularisierung

Zusammenfassung und Ausblick

## 2. Einleitung

### 2.1. Motivation

### 2.2. Aufbau der Arbeit

## 3. Grundlagen

### 3.1. Notationen

$\text{diag}: \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$  einen Vektor  $\mathbf{d}$  in Diagonalform  $\mathbf{D}$  bringt mit  $\mathbf{D}_{ii} = \mathbf{d}_i$  und  $\mathbf{D}_{ij} = 0$  für  $i \neq j$ . Die Inverse  $\text{diag}^{-1}: \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^n$  liefert zu einer beliebigen Diagonalmatrix  $\mathbf{D}$  dessen Diagonalvektor  $\mathbf{d}$ . brauch ich glaub ich nicht mehr

Wir erlauben, dass wir eine eindimensionale Funktion  $f: \mathbb{R} \rightarrow \mathbb{R}$  auch elementweise auf Vektoren, Matrizen bzw. Tensoren anwenden dürfen.

Identitätsmatrix  $\mathbf{I}$

Skalarprodukt  $\langle \cdot, \cdot \rangle$  definiert als

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n \mathbf{x}_i \mathbf{y}_i \quad (3.1)$$

für zwei Vektoren  $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ .

hadamard

### 3.2. Graphentheorie

Graph  $G = (\mathcal{V}, \mathcal{E}, w)$

$\mathcal{V} = \{v_i\}_{i=1}^n$

$|\mathcal{V}| = n < \infty$

Umschreibung in Tensor/Dense Matrix

$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$

Falls  $(u, v) \in \mathcal{E}$ , dann sind  $u$  und  $v$  adjazent und wir schreiben  $u \sim v$

Gewichtsfunktion  $w: \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}_+$

ungewichtet:  $w: \mathcal{V} \times \mathcal{V} \rightarrow \{0, 1\}$

Falls  $(u, v) \notin \mathcal{E}$ , dann  $w(u, v) = 0$

Im ungewichteten Fall ist Gewichtsfunktion implizit durch  $\mathcal{E}$  gegeben

ungerichtet:  $u \sim v$  genau dann, wenn  $v \sim u$  und

$$w(u, v) = w(v, u) \quad (3.2)$$

Fordern wir für den Verlauf dieser Arbeit (also keine gerichteten Graphen)

Als *Schleife* wird eine Kante bezeichnet, die einen Knoten mit sich selbst verbindet, d.h.  $w(v, v) > 0$ . Ein Graph ohne Schleifen wird *schleifenloser Graph* genannt. Für den weiteren Verlauf dieser Arbeit fordern wir schleifenlose Graphen.

Adjazenzmatrix  $\mathbf{A} \in \mathbb{R}_+^{n \times n}$  eines Graphen  $G$  mit  $\mathbf{A}_{ij} = w(v_i, v_j)$   
Wir sagen ein Knoten  $v_i$  hat Position  $i$  in  $\mathbf{A}$ . Umschreibung in Sparse Matrix/Tensor

$G = (\mathcal{V}, \mathcal{E}, w)$  ist eindeutig definiert durch  $\mathbf{A}$ .

Der *Grad* eines Knotens  $v$  ist die Anzahl der Knoten, die adjazent zu ihm sind, d.h.

$$\deg(v) = |\{u: (v, u) \in \mathcal{E}\}| \quad (3.3)$$

Im Falle von gewichteten Graphen wird der Grad eines Knotens von  $v$  auch oft über

$$d(v_i) = \sum_{j=1}^n \mathbf{A}_{ij} \quad (3.4)$$

definiert. Die unterschiedliche Notation macht deutlich, wann wir welchen Grad eines Knotens meinen.

Die Gradmatrix  $\mathbf{D} \in \mathbb{R}_+^{n \times n}$  eines Graphen  $G$  ist definiert als Diagonalmatrix

$$\mathbf{D} = \text{diag}\left([d(v_1), \dots, d(v_n)]^\top\right) \quad (3.5)$$

Umschreibung in Sparse Matrix/Tensor

Ein Graph heißt *k-regulär* falls  $\deg(v_i) = k$  für alle  $1, \dots, n$ .

Ein *ebener Graph* ist eine konkrete Darstellung eines Graphen auf der zweidimensionalen Ebene  $\mathbb{R}^2$ . Jedem Knoten  $v$  ist eine Positionsfunktion  $p: \mathcal{V} \rightarrow \mathbb{R}^2$  zugeordnet, die die Position eines Knotens auf der Ebene eindeutig definiert.

Ein *Weg* ist eine Folge von Knoten  $(v_{x(1)}, v_{x(2)}, \dots, v_{x(k)})$ , sodass  $v_{x(i)} \sim v_{x(i+1)}$  für alle  $1 \leq i < k$  mit Länge  $k$ , wobei  $x: \{1, \dots, n\} \rightarrow \{1, \dots, n\}$  eine Permutation auf der Anzahl der Knoten.

Ein Graph ist *verbunden*, falls er nur eine Komponente hat. Ein Graph ist *verbunden*, falls es von jedem Knoten  $u$  einen Weg zu jedem Knoten  $v$  gibt. Für den weiteren Verlauf dieser Arbeit fordern wir, dass  $G$  verbunden ist.

Ein *Pfad* ist ein Weg, sodass  $v_{x(i)} \neq v_{x(i+1)}$ . Im Kontext von schleifenlosen Graphen sind die Begriffe Weg und Pfad äquivalent. Wir schreiben  $s(u, v)$  einer Funktion  $s: \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{N}$  für die Länge des kürzesten Pfades von  $u$  nach  $v$ .

In Graphen mit *Mehrfachkanten*, auch *Multigraphen* genannt, können zwei Knoten durch mehrere Kanten verbunden sein. Multigraphen lassen sich als Tensor über einen Vektor von Adjazenzmatrizen  $[\mathbf{A}_1, \dots, \mathbf{A}_m] \in \mathbb{R}_+^{m \times n \times n}$  schreiben. Graphen mit Mehrfachkanten können ebenso als eine Menge von Graphen mit gleicher Knotenmenge betrachtet werden.

nimmt noch  
ganz, ge-  
a auch  
ere Kno-

nd graph  
onenten,  
eren

te knoten

o Nachbar-  
 $\mathcal{N}(v, k)$



### 3.3. Faltung in CNNs

In der Funktionalanalysis beschreibt die *Faltung* einen mathematischen Operator, der für zwei Funktion  $f$  und  $g$  eine dritte Funktion  $f * g$  liefert. Die Faltung kann als ein Produkt von Funktionen verstanden werden.

Anschaulich ist  $(f * g)(x)$  der *gewichtete Mittelwert* von  $f$ , wobei die Gewichtung durch  $g$  gegeben ist.

Angenommen wir wollen über einer Matrix mit einem *Filter* falten. Sei unsere Eingangsmatrix  $3 \times 4$  und unsere Filtergröße  $2 \times 2$ .

Dann gilt zum Beispiel für den Faltungsoperator  $*$  in einem Convolutional Neural Network:

$$\begin{pmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 12 & 16 & 11 \\ 24 & 28 & 17 \end{pmatrix} \quad (3.6)$$

$f : 3 \times 4 \rightarrow \mathbb{R}$  und  $g : 2 \times 2 \rightarrow \mathbb{R}$ , dann ist  $*$  definiert als

$$(f * g)(x, y) = \sum_{x_i \in [x, x+1] y_i \in [y, y+1]} f(x_i, y_i) g(x - x_i, y - y_i) \quad (3.7)$$

## 4. Graphrepräsentationen von Bildern

planarer Graph (MUSS NICHT UNBEDINGT SEIN), gegenbeispiel, ist aber auch egal

### 4.1. Kantengewichte

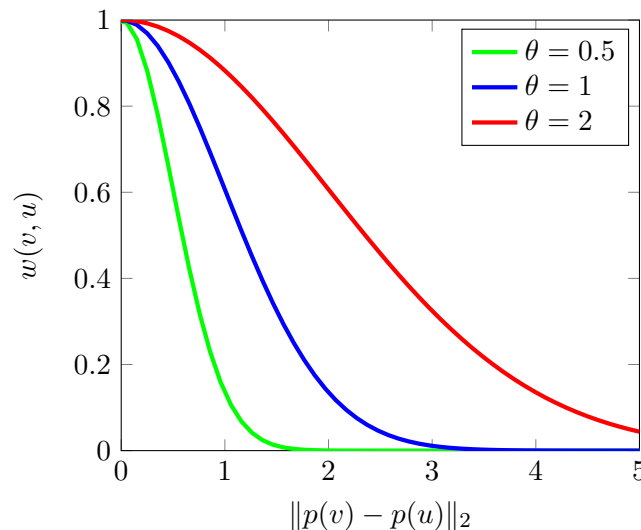
Kantengewichte werden ermittelt aus der euklidischen Distanz der Zentren zweier adjazenter Regionen  $\|p(v) - p(u)\|_2$ . Distanz entspricht aber nicht der üblichen Bedeutung von Kantengewichten auf Graphen. Je höher das Gewicht, desto ähnlicher bzw. näher sind zwei Knoten.

Ein üblicher Weg die Distanz zweier Knoten zueinander als Kantengewicht darzustellen ist über einen gewichteten *Gaussian-Kernel* [?]

$$w(v, u) = \exp\left(-\frac{\|p(v) - p(u)\|_2^2}{2\theta^2}\right) \quad (4.1)$$

falls  $v \sim u$  und einem Parameter  $\theta \in \mathbb{R}$ .

Die Wahl von  $\theta$  ist dabei abhängig von der Ausdehnung der Distanzen eines Graphen.



## 5. Räumliche Graphentheorie

Isomorphismus, Automorphismus, Canonical Labeling  
Labeling / Node Partitions

### 5.1. Patchy-SAN

## 6. Spektrale Graphentheorie

### 6.1. Einführung

- *Spektrum* eines Graphen zur Untersuchung seiner Eigenschaften
- *algebraische* oder *spektrale Graphentheorie* genannt

Algebraische Methoden sind sehr effektiv bei Graphen, die regulär und symmetrisch sind.

### 6.2. Probleme

Rotationsinvariant

### 6.3. Eigenwerte und Eigenvektoren reell symmetrischer Matrizen

$$\mathbf{M}\mathbf{u} = \lambda\mathbf{u}$$

Zu einem Eigenwert  $\lambda$  gibt es unendlich viele (skalierte) Eigenvektoren  $\mathbf{u}$ . Wir definieren einen Eigenvektor  $\mathbf{u}$  dann eindeutig über  $\|\mathbf{u}\|_2 = 1$ . Sei  $\mathbf{M}$  weiterhin reell symmetrisch, d.h.  $\mathbf{M} = \mathbf{M}^\top \in \mathbb{R}^{n \times n}$ . Dann gilt für zwei unterschiedliche Eigenvektoren  $\mathbf{u}_1$  und  $\mathbf{u}_2$ , dass  $\mathbf{u}_1 \perp \mathbf{u}_2$ .  $\mathbf{M}$  hat dann genau  $n$  reelle Eigenwerte mit  $\{\lambda_i\}_{i=1}^n$ . Wir definieren  $\mathbf{\Lambda} = \text{diag}([\lambda_1, \dots, \lambda_n])$ .

Wir definieren die orthogonale Matrix  $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_n] \in \mathbb{R}^{n \times n}$ . Dann gilt  $\mathbf{M}\mathbf{U} = \mathbf{U}\mathbf{\Lambda}$ . und insbesondere ist  $\mathbf{M}$  diagonalisierbar über

$$\mathbf{M} = \mathbf{M}\mathbf{U}\mathbf{U}^\top = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top \quad (6.1)$$

mit  $\mathbf{U}\mathbf{U}^\top = \mathbf{I}$ .

Damit gilt insbesondere für symmetrisch reelle Matrizen  $\mathbf{M}$ ,  $k \in \mathbb{N}$

$$\mathbf{M}^k = (\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top)^k = \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^\top \quad (6.2)$$

Diesen Zusammenhang kann man sich leicht erklären, wenn man die Potenz ausschreibt:

$$(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top)^k = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top \prod_{i=1}^{k-2} \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \mathbf{U}\mathbf{\Lambda}^2\mathbf{U}^\top \prod_{i=1}^{k-2} \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top = \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^\top \quad (6.3)$$

Falls  $\mathbf{M}$  weiterhin *schwach diagonaldominant* ist, d.h

$$\sum_{j=1}^n |\mathbf{M}_{ij}| \leq |\mathbf{M}_{ii}| \quad (6.4)$$

für alle  $i \in \{1, \dots, n\}$  sind ihre Eigenwerte  $\lambda_i \in \mathbb{R}_+$  positiv reell und wir können auf diesen eine Ordnung definieren mit  $0 \leq \lambda_1 \leq \dots \leq \lambda_n$ . Insbesondere ist  $\mathbf{M}$  dann *positiv-semidefinit*, das bedeutet

$$\mathbf{x}^\top \mathbf{M} \mathbf{x} \geq 0 \quad (6.5)$$

## 6.4. Der Laplacian und seine Eigenwerte

Der *kombinatorische Laplacian*  $\mathbf{L}$  eines Graphen  $\mathcal{G}$  ist definiert als  $\mathbf{L} = \mathbf{D} - \mathbf{A}$  [?].  $\mathbf{L}$  ist eine symmetrische Matrix.

Der *normalisierte Laplacian*  $\tilde{\mathbf{L}}$  ist definiert als  $\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{L} \mathbf{D}^{-\frac{1}{2}}$  [?]. Es gilt die Konvention, dass  $(\mathbf{D}^{-\frac{1}{2}})_{ii} = 0$  falls  $\mathbf{D}_{ii} = 0$  (d.h.  $v_i$  ist isolierter Knoten)

Für verbundene Graphen gilt damit  $\tilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$  [?]. Jeder Eintrag der Diagonalen von  $\tilde{\mathbf{L}}$  ist damit 1.  $\tilde{\mathbf{L}}$  ist weiterhin symmetrisch, das wäre bei einer Normierung der Form  $\mathbf{D}^{-1} \mathbf{L}$  nicht der Fall.

$\mathbf{L}$  und  $\tilde{\mathbf{L}}$  sind keine ähnlichen Matrizen. Insbesondere sind ihre Eigenvektoren unterschiedlich. Die Nutzung von  $\mathbf{L}$  oder  $\tilde{\mathbf{L}}$  ist damit abhängig von dem Problem, welches man betrachtet. [?].

Wir schreiben  $\mathcal{L}$  wenn die Wahl des Laplacian  $\mathbf{L}$  oder  $\tilde{\mathbf{L}}$  irrelevant ist.

### 6.4.1. Visuelle Interpretation des Laplacian

- diskrete Analogie des  $\nabla^2$  Operators
- man nimmt eine Funktion und approximiert sie mit Hilfe eines Graphen, so dass Knoten, die dichter beieinander liegen eine größere zweite Ableitung besitzen.

$$\nabla^2 f = \nabla \cdot \nabla f$$

Die Divergenz eines Vektorfeldes ist ein Skalarfeld, das an jedem Punkt angibt, wie sehr die Vektoren in einer kleinen Umgebung des Punktes auseinanderstreben.

The Laplace operator measures how much a function differs at a point from the average of the values of the function over small spheres centered at that point. As it turns out, the Laplacian of a graph does something completely analogous: namely, it measures how much a function on a graph differs at a vertex from the average of the values of the function over the neighbors of the vertex.

Im  $n$ -dimensionalen euklidischen Raum

$$\nabla^2 f = \sum_{i=1}^n \frac{\partial^2 f}{\partial x_i^2} \quad (6.6)$$

in einer Dimension reduziert sich der Laplace-Operator auf die zweite Ableitung  $\nabla^2 f = f''$ .

Der *diskrete Laplace-Operator* ist eine Analogie zum diskreten Laplace-Operator, der finite Differenzen  $x \pm h$  zur Approximation von  $\nabla^2 f$  nutzt. Approximation des Laplace-Operators für finite Elemente

Sei  $f: \mathcal{V} \rightarrow \mathbb{R}$  eine Funktion auf den Knoten eines Graphen.  $f$  kann ebenso als Vektor  $\mathbf{f} \in \mathbb{R}^n$  betrachtet werden mit der Ordnung der Knoten, die die Adjazenzmatrix vorgibt.

Dann gilt für  $\mathcal{L}$ , dass

$$(\mathcal{L}\mathbf{f})_i = \sum_{\substack{j=0 \\ j \neq i}}^n -\mathcal{L}_{ij}(\mathbf{f}_i - \mathbf{f}_j) \quad (6.7)$$

Für einen Graphen, der ein reguläres Gitter aufspannt mit gleichen Kantengewichten  $\frac{1}{h^2} \in \mathbb{R}$  gilt für einen Knoten an Position  $(x, y)$ : Abusing the index notation

$$(\mathbf{L}\mathbf{f})_{x,y} = \frac{4\mathbf{f}_{x,y} - \mathbf{f}_{x+1,y} - \mathbf{f}_{x-1,y} - \mathbf{f}_{x,y+1} - \mathbf{f}_{x,y-1}}{h^2} \quad (6.8)$$

beschreibt die *5-Punkte-Stern* Approximation  $-\nabla^2 f$ . mit  $\nabla^2 f$  definiert auf den fünf Punkten  $\{(x, y), (x + h, y), (x - h, y), (x, y + h), (x, y - h)\}$ .

$$\mathcal{L}f \approx -\nabla^2 f \quad (6.9)$$

Damit kann der Graph Laplacian als eine Generalisierung des diskreten Laplacian auf einem Gitter verstanden werden.

Eigenwerte und Eigenvektoren werden benutzt, um zu verstehen was passiert, wenn wir einen Operator (hier  $\mathcal{L}$ ) mehrfach auf einen Vektor  $\mathbf{x}$  anwenden (hier Merkmal auf den Knoten).

Wir können  $\mathbf{x}$  als Linearkombination der *Eigenbasis* schreiben mit

$$\mathbf{x} = \sum_i c_i \mathbf{u}_i \quad (6.10)$$

und berechnen dann

$$\mathcal{L}^k \mathbf{x} = \sum_i c_i \mathcal{L}^k \mathbf{u}_i = \sum_i c_i \lambda_i \mathcal{L}^{k-1} \mathbf{u}_i = \sum_i c_i \lambda_i^k \mathbf{u}_i \quad (6.11)$$

Wenn wir einen Operator haben, der einen Graphen beschreibt, dann können Eigenschaften dieses Operators und damit des Graphen selber durch dessen Eigenwerte und Eigenvektoren beschrieben werden.

### 6.4.2. Eigenschaften

$\mathcal{L}$  ist eine reell symmetrische, schwach diagonaldominante Matrix und damit insbesondere positiv semidefinit.

$\mathcal{L} \in \mathbb{R}^{n \times n}$  hat genau  $n$  Eigenwerte  $\{\lambda_i\}_{i=1}^n \in \mathbb{R}_+$  mit  $\lambda_i \leq \lambda_{i+1}$ .

Anzahl der Eigenvektoren gleich Null ist die Anzahl an Komponenten, die ein Graph besitzt.

Insbesondere sind jede Reihen- und Spaltensumme von  $\mathbf{L}$  ist 0, d.h.  $\sum_j \mathcal{L}_{ij} = 0$  und  $\sum_j \mathcal{L}_{ji} = 0$  für alle  $i \in \{1, \dots, n\}$ . Insbesondere gilt  $\lambda_1 = 0$ , da  $\mathbf{u}_1 = \frac{1}{\sqrt{n}}[1, \dots, 1]^\top \in \mathbb{R}^n$  Eigenvektor von  $\mathcal{L}$  mit  $\mathcal{L}\mathbf{u}_1 = 0$ .

gilt nur für  
binatorische

$0 = \lambda_1 < \lambda_2 \leq \dots \leq \lambda_n$  wenn Graph verbunden.

quelle

Für einen Graphen  $G$  definieren wir  $\lambda_G := \lambda_2$  und  $\lambda_{\max} := \lambda_n$ . Für  $\tilde{\mathbf{L}}$  gilt  $\lambda_{\max} \leq 2$ .

was sagt  $\lambda_2$

Für  $\mathcal{L}^k$  mit  $k \in \mathbb{N}$  gilt  $(\mathcal{L}^k)_{ij} = 0$  genau dann, wenn  $s(v_i, v_j) > k$  [?]. Damit beschreibt  $\mathcal{L}^k$  bildlich gesprochen die Menge an Knoten, die maximal  $k$  Kanten entfernt liegen.

quelle

Eine Verschrumpfung eines Graphen  $G$  kann beschrieben werden über zwei verschiedene Knoten  $u$  und  $v$  zu einem neuen Knoten  $v^*$  mit

$$w(x, v^*) = w(x, u) + w(x, v) \quad (6.12)$$

$$w(v^*, v^*) = w(u, u) + w(v, v) + 2w(u, v) \quad (6.13)$$

Für einen Graphen  $G$  gilt für einen Graphen  $H$ , der aus  $G$  verkleinert wurde [?],

$$\lambda_G \leq \lambda_H \quad (6.14)$$

## 6.5. Faltung mittels Graph-Fourier-Transformation

### 6.5.1. Kontinuierliche Fourier-Transformation

kontinuierliche oder klassische Fourier-Transformation: Konvertierung eines Signals in dessen Frequenzspektrum. Signal  $f: \mathbb{R} \rightarrow \mathbb{R}$

Die komplexe Exponentialfunktion  $\exp(i\omega t)$  der Fourier-Transformation beschreibt die *Eigenfunktionen* des eindimensionalen Laplace-Operators  $\nabla^2$ .

Die inverse Fourier-Transformation

das versteht  
nicht

$$f(t) = \frac{1}{2\pi} \int \hat{f}(\omega) \exp(i\omega t) d\omega \quad (6.15)$$

kann damit als die Ausdehnung von  $f$  in Bezug auf die Eigenfunktionen des Laplace-Operators gesehen werden.

### 6.5.2. Graph-Fourier-Transformation

Wir können die *Graph-Fourier-Transformation* analog dazu definieren.

Eigenwerte beschreiben die Frequenzen des Graphen.

Signal  $f: \mathcal{V} \rightarrow \mathbb{R}$  auf Graphen

Signal kann ebenso als Vektor  $\mathbf{f} \in \mathbb{R}^n$  aufgefasst werden (impliziert Ordnung der Knoten, ist durch Adjazenzmatrix gegeben)

weiter  
definiert,  
n grundla-  
nterbrin-

$$\hat{f}(\lambda_i) = \langle \mathbf{f}, \mathbf{u}_i \rangle \quad (6.16)$$

bzw. als Vektor

$$\hat{\mathbf{f}} = \mathbf{U}^\top \mathbf{f} \quad (6.17)$$

Die inverse Transformation ist dann definiert als

$$f(v_i) = \sum_{j=1}^n \hat{f}(\lambda_j) (\mathbf{u}_j)_i \quad (6.18)$$

bzw. als Vektor

$$\mathbf{f} = \mathbf{U} \hat{\mathbf{f}} \quad (6.19)$$

val relati-

Spectral Graph Domain

- *Spectral Graph Domain*: Der Raum der Eigenfunktionen von  $\mathcal{L}$
- Analogon (Nachbildung) einer *Fourier-Transformation* von Funktionen auf gewichteten Graphen

Das ermöglicht uns das Anwenden verschiedener Operation wie Filterung, Translation oder Faltung.

Directly extending this construction to arbitrary weighted graphs is problematic, as it is unclear how to define scaling and translation on an irregular graph. We approach this problem by working in the spectral graph domain, i.e. the space of eigenfunctions of the graph Laplacian  $\mathcal{L}$ . This tool from spectral graph theory, provides an analogue of the Fourier transform for functions on weighted graphs.

Eigenwerte werden als Frequenz aufgefasst, die das Spektrum des Graphen beschreiben.

Die Eigenvektoren beschreiben die Signale zu den gegebenen Frequenzen.

Fourier-Transformation beschreibt die gleiche Funktion  $f$ , aber in einer völlig anderen Domäne. Nicht in der Vertex-Domäne, sondern in der Spectrum-Domäne, d.h. auf Basis der Eigenwerte.

In classical Fourier analysis, the eigenvalues carry a specific notion of frequency: for  $\omega$  close to zero (low frequencies), the associated complex exponential eigenfunctions are smooth, slowly oscillating functions, whereas for  $\omega$  far from zero (high frequencies), the associated complex exponential eigenfunctions oscillate much more rapidly. In the graph setting, the graph Laplacian eigenvalues and eigenvectors provide a similar notion of frequency. For connected graphs, the Laplacian eigenvector  $\mathbf{u}_0$  associated with the eigenvalue



0 is constant and equal to 1 at each vertex. The graph Laplacian eigenvectors associated with low frequencies  $\lambda_2$  vary slowly across the graph; i.e., if two vertices are connected by an edge with a large weight, the values of the eigenvector at those locations are likely to be similar. The eigenvectors associated with larger eigenvalues oscillate more rapidly and are more likely to have dissimilar values on vertices connected by an edge with high weight.

Graph Fourier Transformation und ihre Inverse gibt uns die Möglichkeit, ein Signal in zwei verschiedenen Domänen zu repräsentieren, nämlich die Knotendomäne (das unveränderte Signal auf der Knotenmenge  $\mathbf{f}$ ) und der spektralen Domäne (das transformierte Signal in das Spektrum des Graphen). Signale, die im Spektrum definiert werden, werden *Kernel* genannt.

ok?

Die Fourier-Transformation wird unter anderem gerne genutzt, da sie gute Eigenschaften hat.

bilder bzw.  
ken

### 6.5.3. Faltung

Es ist schwierig, Faltung auf Graphen zu definieren (wir haben keinen Translationsoperator  $(x - t)$ ). In der Fourier-Domäne ist dies aber sehr einfach.

In der Signalverarbeitung versteht man unter der Frequenzfilterung die Transformation eines Signals in die Fourier-Domäne und der verstärkenden oder dämpfenden Veränderung der Amplituden der Frequenzkomponenten.

Formal betrachtet also

$$\hat{f}_{\text{out}}(\omega) = \hat{f}_{\text{in}}(\omega)\hat{g}(\omega) \quad (6.20)$$

Es lässt sich zeigen, dass die Multiplikation in der Fourier-Domäne äquivalent ist zu einer Faltung in der Zeitdomäne

quelle

$$f_{\text{out}}(t) = (f_{\text{in}} \star g)(t) \quad (6.21)$$

Wir können das spektrale Graphfilterung analog definieren mit

$$\hat{f}_{\text{out}}(\lambda_i) = \hat{f}_{\text{in}}(\lambda_i)\hat{g}(\lambda_i) \quad (6.22)$$

Damit entspricht die Faltung eines Signals auf einem Graphen der elementweisen Multiplikation in der Spectral Graph Domain

$$\mathbf{f}_{\text{out}} = \mathbf{f}_{\text{in}} \star \mathbf{g} = \mathbf{U} \left( \mathbf{U}^\top \mathbf{f}_{\text{in}} \odot \mathbf{U}^\top \mathbf{g} \right) \quad (6.23)$$

Das lässt sich in Matrixschreibweise (die zweite obige Formel) umschreiben über

$$\hat{\mathbf{f}}_{\text{out}} = \hat{g}(\mathbf{\Lambda})\hat{\mathbf{f}}_{\text{in}} = \hat{g}(\mathbf{\Lambda})\mathbf{U}^\top \mathbf{f}_{\text{in}} \quad (6.24)$$

bzw.

$$\mathbf{f}_{\text{out}} = \mathbf{U}\hat{g}(\mathbf{\Lambda})\mathbf{U}^\top \mathbf{f}_{\text{in}} \quad (6.25)$$

mit  $\hat{g}(\mathbf{\Lambda}) = \text{diag}([\hat{g}(\lambda_1), \dots, \hat{g}(\lambda_n)]^\top)$

## 6.6. Polynomielle Approximation

bedeutet

- not localized in Space

bedeutet  
localized?

- Learning Komplexität linear zu der Dimension der Daten  $n$

Wenn wir  $\hat{g}(\lambda_i)$  als ein Polynom vom Grad  $k$  approximieren, d.h.

$$\hat{g}'(\lambda_i) = \sum_{j=0}^k c_j \lambda_i^j \quad (6.26)$$

bzw.

$$\hat{g}'(\mathbf{\Lambda}) = \sum_{j=0}^k c_j \mathbf{\Lambda}^j \quad (6.27)$$

mit Koeffizienten  $c_0, \dots, c_k \in \mathbb{R}$ , dann lässt sich der Filterungsprozess in der Spectral Graph Domain erstaunlich gut auch in der Knotendomäne interpretieren.

Der Term  $\mathbf{U}\hat{g}'(\mathbf{\Lambda})\mathbf{U}^\top$  kann dann weiter vereinfacht werden:

$$\hat{g}'(\mathcal{L}) =: \mathbf{U}\hat{g}'(\mathbf{\Lambda})\mathbf{U}^\top = \mathbf{U} \sum_{j=0}^k c_j \mathbf{\Lambda}^j \mathbf{U}^\top = \sum_{j=0}^k c_j \mathbf{U}\mathbf{\Lambda}^j \mathbf{U}^\top = \sum_{j=0}^k c_j \mathcal{L}^j \quad (6.28)$$

s weiter  
definiert

Das sieht man leicht mit der Beziehung  $\mathcal{L}^k = (\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top)^k = \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^\top$ .

Dann ist die Faltung definiert als

$$\mathbf{f}_{\text{out}} = \hat{g}'(\mathcal{L})\mathbf{f}_{\text{in}} \quad (6.29)$$

bzw.

$$f_{\text{out}}(v_i) = \sum_{j=0}^k f_{\text{in}}(v_j) (\hat{g}'(\mathcal{L}))_{ij} \quad (6.30)$$

Wir erinnern uns, dass  $(\mathbf{L}^l)_{ij} = 0$ , wenn die kürzeste Pfadlänge  $s(v_i, v_j) > l$  von  $v_i$  zu  $v_j$ , d.h. die minimale Anzahl an Kanten, größer ist als  $l$ .

eren

Damit kann die Faltung in der Knotenmenge intuitiv beschrieben werden als eine gewichtete Aufsummierung bzw. lineare Kombination der Knotensignale in einer  $k$ -lokalisierten Nachbarschaft um einen Knoten  $v_i$

$$f_{\text{out}}(v_i) = f_{\text{in}}(v_i) (\hat{g}'(\mathcal{L}))_{ii} + \sum_{v_j \in \mathcal{N}(v_i, k)} f_{\text{in}}(v_j) (\hat{g}'(\mathcal{L}))_{ij} \quad (6.31)$$

### 6.6.1. Tschebyschow-Polynome

- Faltung ist sehr teuer aufgrund der Berechnungen von  $\mathcal{L}^k$
- Idee: beschreibe  $\hat{g}'(\mathcal{L})$  als eine polynomielle Funktion, die rekursiv aus  $\mathcal{L}$  berechnet werden kann.

- Warum sollte das effizienter sein?  $\Rightarrow \mathcal{L}$  ist nicht dicht besetzt, und hat nur  $|\mathcal{E}|+n \ll n^2$  Einträge mit  $n \leq |\mathcal{E}|$

*Tschebyschow-Polynome* (engl. *Chebyshev*) bezeichnen eine Menge von Polynomen  $T_n(x): \mathbb{R} \rightarrow \mathbb{R}$  mit dem rekursiven Zusammenhang

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x) \quad (6.32)$$

mit  $T_0(x) = 1$  und  $T_1(x) = x$ . Ein Tschebyschow-Polynom  $T_n$  ist ein Polynom  $n$ -ten Grads.

Für  $x \in [-1, 1]$  gilt  $T_k(x) \in [-1, 1]$

Rescale  $\mathbf{\Lambda}$  zu  $\tilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{\max}}\mathbf{\Lambda} - \mathbf{I} \in [-1, 1]^{n \times n}$ .

Dann ist  $T_k(\tilde{\mathbf{\Lambda}}) \in [-1, 1]^{n \times n}$

warum noch

$$\hat{g}'(\mathbf{\Lambda}) = \sum_{i=0}^k c_i \mathbf{\Lambda}^i = \sum_{i=0}^k c'_i T_i(\tilde{\mathbf{\Lambda}}) \quad (6.33)$$

Analog lässt sich wiederum  $\hat{g}'(\mathcal{L})$  definieren mit

$$\hat{g}'(\mathcal{L}) = \sum_{i=0}^k c'_i T_i(\tilde{\mathcal{L}}) \quad (6.34)$$

mit  $\tilde{\mathcal{L}} = \mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{U}^\top = \frac{2}{\lambda_{\max}}\mathcal{L} - \mathbf{I}$ .

Jetzt lässt sich  $\mathbf{f}_{\text{out}} = \hat{g}'(\mathcal{L})\mathbf{f}_{\text{in}} = \sum_{i=0}^k c'_i T_i(\tilde{\mathcal{L}})\mathbf{f}_{\text{in}}$  sehr schnell berechnen:

1. berechne  $\bar{\mathbf{f}}_i := T_i(\tilde{\mathcal{L}})\mathbf{f}_{\text{in}}$  für alle  $i \in \{0, 1, \dots, k\}$  mit Hilfe von Rekursion:
  - a)  $\bar{\mathbf{f}}_0 = \mathbf{f}_{\text{in}}$
  - b)  $\bar{\mathbf{f}}_1 = \tilde{\mathcal{L}}\mathbf{f}_{\text{in}}$
  - c)  $\bar{\mathbf{f}}_i = 2\tilde{\mathcal{L}}\bar{\mathbf{f}}_{i-1} - \bar{\mathbf{f}}_{i-2}$
2. berechne  $\mathbf{f}_{\text{out}} = [\bar{\mathbf{f}}_0, \bar{\mathbf{f}}_1, \dots, \bar{\mathbf{f}}_k]\mathbf{c}'$ , wobei  $\mathbf{c}' = [c'_0, \dots, c'_k] \in \mathbb{R}^{k+1}$

### Laufzeit

- anstatt  $\mathcal{L}^k$  zu berechnen mit Komplexität  $\mathcal{O}(n^2)$  haben wir nur noch  $k$  Matrix-Vektor-Multiplikationen mit der Matrix  $\tilde{\mathcal{L}}$
- da  $\mathcal{L}$  für große Graphen sehr dünnbesetzt ist, d.h.  $|\mathcal{E}| \ll n^2$ , haben wir bei Verwendung von *dünnbesetzten Matrizen* nur noch eine Laufzeit von  $\mathcal{O}(k|\mathcal{E}|)$  [?, ?]
- für den zweiten Schritt gilt  $\mathcal{O}(kn)$ , mit  $n \leq |\mathcal{E}|$  bedingt damit nur Schritt Eins die Laufzeit

## 6.7. Graph Convolutional Networks

- aus [?]
- Idee: setze  $k = 1$  für alle Faltungsebenen
- Begründung: Faltung über  $i \leq k = 1$  berücksichtigt nur alle Knoten, die zum jeweiligen Faltungsknoten adjazent sind und den Knoten selber (Begründung weiter oben)
- für CNNs hat sich gezeigt, dass kleine Filtergrößen wie  $3 \times 3$  keine negativen Auswirkungen haben
- viele kleine Faltungsebenen ohne Pooling propagieren die Informationen weit entfernter Knoten weiter

- kann das Problem des Overfitting reduzieren für Graphen mit hohem Grad

Annahme:  $\lambda_{\max} \approx 2$  Es gilt  $0 \leq \lambda_0 \leq \dots \leq \lambda_{n-1} \leq 2$  [?] (aber nur für bitartite Graphen?) Wenn das gilt, dann wird  $\lambda_{\max} := 2$  einfach auf die obere Schranke gesetzt Begründung: Netzparameter werden die Veränderung der Skalierung von  $\tilde{\mathbf{L}}$  annehmen/ausgleichen. Dann ist  $\tilde{\mathbf{L}} = \mathbf{L} - \mathbf{I}$ .

Dann gilt für die Filterung eines Signals  $\mathbf{x}$  über  $g_\theta$

$$g_\theta \star \mathbf{x} = \mathbf{U} g_\theta(\boldsymbol{\Lambda}) \mathbf{U}^\top \mathbf{x} \approx g'_{\theta'}(\mathbf{L}) \mathbf{x} = \sum_{i=0}^{k-1} \theta'_i T_i(\mathbf{L} - \mathbf{I}) \mathbf{x} = \theta'_0 \mathbf{x} + \theta'_1 (\mathbf{L} - \mathbf{I}) \mathbf{x} \quad (6.35)$$

Wenn wir den normalisierten Laplacian benutzen, gegeben durch  $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ , dann lässt sich weiter vereinfachen mit

$$g_\theta \star \mathbf{x} \approx \theta'_0 \mathbf{x} - \theta'_1 \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \mathbf{x} \quad (6.36)$$

Um die Gefahr des Overfittings und die Anzahl an Berechnungen weiter zu reduzieren, können die Anzahl der Parameter weiter reduziert werden. Mit  $\theta = \theta'_0 = -\theta'_1$  gilt dann

$$g_\theta \star \mathbf{x} \approx \theta \left( \mathbf{I} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \right) \mathbf{x} \quad (6.37)$$

$\mathbf{I} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$  hat nun Eigenwerte im Bereich  $[0, 2]$ . Wiederholte Anwendungen dieses Operators können daher zu numerischen Instabilitäten und dann zu explodierenden oder verschwindenden Gradienten führen. Um dies zu verhindern, wird folgende Renormalisierung vorgenommen:  $\mathbf{I} + \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}} \rightarrow \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}}$  mit  $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$  und  $\tilde{\mathbf{D}}$  ist nun die Gradmatrix der renormalisierten Adjazenzmatrix  $\tilde{\mathbf{A}}$ .

$$g_\theta \star \mathbf{x} \approx \theta \left( \tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} \right) \mathbf{x} \quad (6.38)$$

$$H^{(l+1)} = f(H^{(l)}, A) \quad (6.39)$$

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)}) \quad (6.40)$$

$$D_{ii} = \sum_j A_{ij} \quad (6.41)$$

Für die Potenz  $x \in \mathbb{R}$  einer Diagonalmatrix  $D \in \mathbb{R}^{N \times N}$  gilt:

$$D^x = \begin{pmatrix} d_{11} & 0 & \cdots & 0 \\ 0 & d_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{nn} \end{pmatrix}^x = \begin{pmatrix} d_{11}^x & 0 & \cdots & 0 \\ 0 & d_{22}^x & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{nn}^x \end{pmatrix} \quad (6.42)$$

## 6.8. Weisfeiler Lehman Analogie

## 6.9. Erweiterung für mehrere Kantenattribute

Graph Convolutional Networks berücksichtigen nur eine Adjazenzmatrix. Das bedeutet insbesondere, dass ein Graph nur über ein Kantenattribut verfügen kann. Das ist für ungewichtete Graphen die Markierung einer Kante ( $a_{ij} \in \{0, 1\}$ ) oder für gewichtete Graphen das Gewicht einer Kante ( $a_{ij} \in \mathbb{R}_+$ ). Eine Menge von Kantenattributen kann über mehrere Adjazenzmatrizen definiert werden. Damit ist es ebenfalls möglich unterschiedliche Kanten für unterschiedliche Attribute zu definieren.

Eine Menge von Adjazenzmatrizen  $\mathcal{A} = \{A_1, A_2, \dots, A_m\}$  mit  $A_i \in \mathbb{R}^{n \times n}$  beschreibt damit eine Menge von  $m$  Graphen über der gleichen Knotenmenge  $\mathcal{V}$  mit Kardinalität  $n$ .

$\mathcal{A} \in \mathbb{R}^{m \times n \times n}$  kann zu einer zweidimensionalen Matrix  $A \in \mathbb{R}^{m \cdot n \times n}$  geglättet werden.

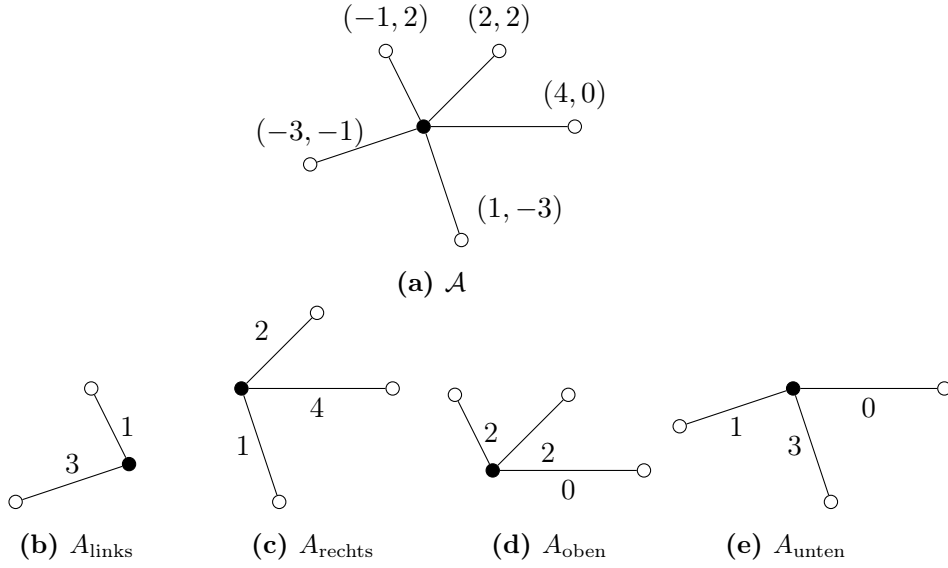
Dann ist  $A \cdot H^{(l)} \in \mathbb{R}^{m \cdot n \times d}$ . Reshape zu  $\mathbb{R}^{n \times m \cdot d}$  und Gewichtsmatrix  $G \in \mathbb{R}^{m \cdot d \times x}$ .

$$H^{(l+1)} = f(H^{(l)}, \tilde{\mathcal{A}}) = \sigma \left( \frac{1}{|\tilde{\mathcal{A}}|} \sum_{\tilde{A}_i \in \tilde{\mathcal{A}}} \tilde{D}_i^{-\frac{1}{2}} \tilde{A}_i \tilde{D}_i^{-\frac{1}{2}} H^{(l)} W_i^{(l)} \right) \quad (6.43)$$

$\sigma(\cdot)$  kennzeichnet eine Aktivierungsfunktion wie zum Beispiel  $\text{ReLU}(\cdot) = \max(0, \cdot)$ .

### 6.9.1. Übertragung auf räumlich eingebettete Graphen

Graphknoten haben im Allgemeinen keine Position oder Lage im Raum. Knoten, die Regionen in einer vorhandenen Segmentierung darstellen, haben jedoch offensichtlich eine gewisse Lage im Raum, die zum Beispiel über das Zentrum der Region definiert werden kann. Diese Information ist vorhanden und wichtig und sollte demnach auch nicht verloren



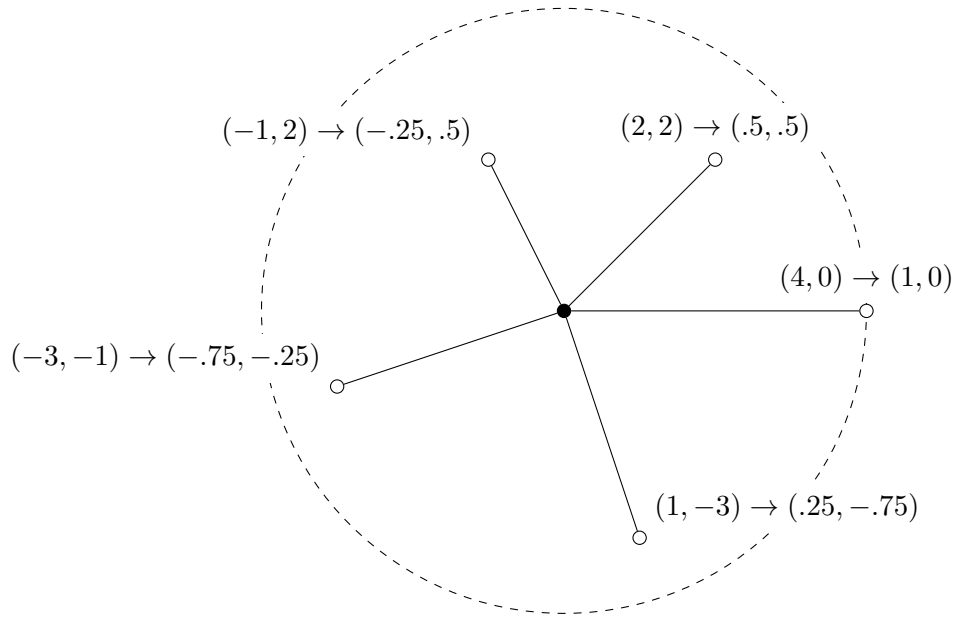
**Abbildung 6.1.:** Aufteilung einer Adjazenzmatrix in vier räumlich eingebettete Bereiche.

gehen. Anstatt diese lokal im Knoten zu speichern, bietet es sich eher an diese Information in den Kanten zu speichern um eine bessere Faltung zu garantieren. Die euklidische Distanz zwischen zwei benachbarten Regionszentren wahrt zwar die Information der Distanz zweier Knoten zueinander, verliert aber die Information der Position zweier Knoten zueinander. Es bietet sich daher an, die horizontalen und vertikalen Abstände in einer Koordinate an den Kanten zu speichern. Es ist zu beachten, dass wir dadurch zu einem gerichteten Graphen übergehen, bei dem jede Kante von  $v$  nach  $w$  auch eine Kante von  $w$  nach  $v$  besitzt.

Wir haben damit zwei Adjazenzmatrizen. Da Graph Convolutional Networks nicht mit negativen Gewichten funktionieren, müssen wir negative Koordinaten in eine weitere Adjazenzmatrix schreiben. Wir gelangen damit zu vier Adjazenzmatrizen, die die Verbindungen von einem Knoten beschreibt, die links, rechts, oben oder unten zu ihm liegen. Wir definieren diese Adjazenzmatrizen respektive als  $A_{\text{links}}$ ,  $A_{\text{rechts}}$ ,  $A_{\text{oben}}$  und  $A_{\text{unten}}$  (vgl. Abbildung 6.1). Falls eine Kante horizontal bzw. vertikal liegt, so definieren wir  $a_{ij} = 1$  respektive für beide „gegenüberliegenden“ Adjazenzmatrizen.

Kantenattribute bzw. Positionen von Knoten sollten skalierungsinvariant gespeichert werden. Dafür werden die Abstände auf den Einheitskreis gemappt, wobei der Knoten mit der längsten Distanz zum Wurzelknoten genau auf dem Einheitskreis liegt (vgl. Abbildung 6.2).

Für die Anwendung auf das Graph Convolutional Network müssen die Gewichte aller Adjazenzmatrizen  $a_{xij} \in [0, 1]$  invertiert werden, damit nähere Knoten einen größeren Einfluss haben. Ebenso müssen *Self Loops* für alle Knoten hinzugefügt werden. Wir definieren unsere Adjazenzmatrix  $\tilde{A} \in \mathbb{R}^{N \times N}$  aus einer Adjazenzmatrix  $A \in \mathbb{R}^{N \times N}$  dann über



**Abbildung 6.2.:** Abbildung der lokalen Nachbarschaftsknoten auf den Einheitskreis.

$$\tilde{A}_{ij} = \begin{cases} 1, & \text{falls } i = j, \\ (a_{ij} + 1)^{-1}, & \text{falls } a_{ij} \neq 0, \\ 0, & \text{sonst.} \end{cases} \quad (6.44)$$

Dann ist  $\tilde{a}_{ij} \in [1, 0.5]$

Diagonalmatrix ist schwierig. Man will ja die Normalisierung damit  $H^{(l)}$  nicht überskalieren. Ich würde auch die gewichtete Matrix normalisieren. Denke das macht Sinn. Dann fallen die Werte ab, wenn viele Knoten weit entfernt sind.

## 6.10. Pooling-Ebene

### 6.10.1. Clustering von Graphen

Pooling-Ebenen des Netzes sollen über das Clustering bzw. die logische Zusammenfassung von Knoten realisiert werden.

Anforderungen:

- mehrstufiges Clustering von Graphen für mehrere Pooling-Ebenen
- Reduzierung der Knotenanzahl soll den Blick auf einen Graphen bei unterschiedlichen Auflösungen zeigen
- Cluster-Algorithmen, die die Größe eines Graphen um den Faktor zwei für jede Anwendung reduzieren erlauben eine feine Kontrolle über die zu benutzenden Pooling-Größen.

- effiziente Approximation, da Graph-Clustering NP-schwer (vgl. 5)

Es existieren einige Cluster-Techniken auf Graphen wie das populäre *spektrale Clustering* [?].

Dieser erfüllt aber nicht die Voraussetzungen (warum nicht?). Stimmt doch garnicht!!

Defferrard et al. [?] benutzen für die Pooling-Ebene eines Netzes auf Graphen die *Vergrößerungsphase* des mehrstufigen Cluster-Algorithmus *Gracclus* [?]. Dabei wird der initiale Graph  $G_0$  sukzessive in kleinere Graphen  $G_1, G_2, \dots, G_m$  mit  $|\mathcal{V}_0| > |\mathcal{V}_1| > \dots > |\mathcal{V}_m|$  transformiert. Für die Transformation von einem Graphen  $G_i$  zu einem Graphen  $G_{i+1}$  mit kleinerer Knotenanzahl  $|\mathcal{V}_{i+1}| < |\mathcal{V}_i|$  werden aus disjunkten Knotenuntermengen von  $\mathcal{V}_i$  *Superknoten* für  $\mathcal{V}_{i+1}$  gebildet.

Die Auswahl der Untermengen erfolgt gierig. Die Knoten des Graphen werden als unmarkiert initialisiert und zufällig durchlaufen. Für jeden Knoten  $v \in \mathcal{V}_i$ , der noch unmarkiert ist, wird ein lokaler, ebenfalls noch unmarkierter, Nachbarschaftsknoten  $u \in \mathcal{N}(v)$  nach einer zuvor definierten Strategie bestimmt und  $v$  sowie  $w$  zu einem Superknoten  $v^* := \{v, w\} \in \mathcal{V}_{i+1}$  verschmelzt. Anschließend werden  $v$  und  $w$  markiert. Falls  $v$  keinen unmarkierten Nachbarn besitzt, wird  $v$  allein als *Singleton-Superknoten*  $v^* := \{v\} \in \mathcal{V}_{i+1}$  deklariert und markiert [?].

Strategien für die Nachbarschaftsauswahl basieren üblicherweise auf der Maximierung von  $w_{uv}$  oder  $w_{uv} \left( \frac{1}{d_u} + \frac{1}{d_v} \right)$  (*Normalized Cut*).

Gracclus reduziert die Knotenanzahl eines beliebigen Graphen näherungsweise um die Hälfte, d.h.  $2 \cdot |\mathcal{V}_{i+1}| \approx |\mathcal{V}_i|$ . Ausnahmen sind zum Beispiel Graphen  $G = (\mathcal{V}, \mathcal{E})$  mit  $\mathcal{E} = \emptyset$ . In der Praxis zeigt sich jedoch, dass Gracclus nur sehr wenige Singleton-Knoten generiert [?].

Nach der spektralen Graphentheorie [?] gilt für Kanten eines Graphen  $G = (\mathcal{V}, \mathcal{E})$  nach Verschmelzung von  $u$  und  $v$  zu  $v^*$

$$w_{xv^*} = w_{xu} + w_{xv} \quad (6.45)$$

$$w_{v^*v^*} = w_{uu} + w_{vv} + 2w_{uv} \quad (6.46)$$

für einen Knoten  $x \in \mathcal{V}$ ,  $x \neq v^*$ . Insbesondere gilt für einen Graphen  $H$ , der auf diese Weise konstruiert wurde,  $\lambda_G \leq \lambda_H$ , wobei  $\lambda_G$ ,  $\lambda_H$  jeweils die ersten Eigenvektoren  $\lambda_1$  von  $G$  respektive  $H$  [?].

## Übertragung auf planare Graphen

- Mittelwert der Positionen wird gebildet (auch gewichtet über  $d_i$ ?)

- $\mathcal{E}_{v^*} = \mathcal{E}_u \cup \mathcal{E}_v$



### 6.10.2. Pooling-Operation

Anhand eines kleineren, vergrößerten Graphen  $G_{i+1}$  und der eindeutigen Zuweisung von Knoten  $u, v \in \mathcal{V}_i$  zu  $v^* \in \mathcal{V}_{i+1}$  können nun die Pooling-Operation der Knotenattribute von  $\mathcal{V}_i$  zu  $\mathcal{V}_{i+1}$  definiert werden:

- **Max-Pooling:**  $v^* := \max(u, v)$
- **L2-Pooling:**  $v^* := \|u, v\|_2$

### 6.10.3. INFORMATIONEN

Normalized Cut ist daher sinnvoll, dass Kanten als wichtiger gezählt werden, wenn ihre entsprechenden Knoten einen geringen Grad haben, das heißt, von nicht so vielen Knoten abhängen. DAS MACHT DURCHAUS SINN.

Ebenfalls ist der ganze Prozess randomisiert. Das heißt wir erreichen bei mehrmaliger Anwendung auf dem gleichen Graphen unterschiedliche Ergebnisse. Damit ist eine Augmentierung der Daten bereits in der Pooling-Ebene vorhanden. DAS IST SEHTR GUT

## 6.11. Beispiel

Wir betrachten eine einfache  $3 \times 3$  Adjazenzmatrix, d.h.  $|\mathcal{V}| = n = 3$ .

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad (6.47)$$

mit Diagonalmatrix  $D = \text{diag}(1, 2, 1)$ .

Der Laplacian  $\mathcal{L} = D - A$  ist dann

$$\mathcal{L} = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix} \quad (6.48)$$

Nun müssen die Eigenvektoren der Matrix und dessen Eigenwerte bestimmt werden, d.h. wir müssen das folgende Eigenwertproblem lösen

$$\mathcal{L} \cdot \vec{u} = \lambda \cdot \vec{u} \quad (6.49)$$

Wir erhalten 3 Eigenvektoren und Eigenwerte mit

$$\lambda_0 = 0, \vec{u}_0 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.58 \\ 0.58 \\ 0.58 \end{pmatrix}, \lambda_1 = 1, \vec{u}_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \approx \begin{pmatrix} -0.71 \\ 0 \\ 0.71 \end{pmatrix}, \lambda_2 = 3, \vec{u}_2 = \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.41 \\ -0.82 \\ 0.41 \end{pmatrix} \quad (6.50)$$

Dann sind  $U$ ,  $\Lambda$  und  $U^T$  definiert als

$$U \approx \begin{pmatrix} 0.58 & -0.71 & 0.41 \\ 0.58 & 0 & -0.82 \\ 0.58 & 0.71 & 0.41 \end{pmatrix}, \Lambda = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix}, U^T \approx \begin{pmatrix} 0.58 & 0.58 & 0.58 \\ -0.71 & 0 & 0.71 \\ 0.41 & -0.82 & 0.41 \end{pmatrix} \quad (6.51)$$

Angenommen wir haben ein Signal  $x = (100, 10, 1)^T$ , dann ist der Wert dieses Signals transformiert in die Fourier Domäne definiert als  $\hat{x} \approx (64.09, -70.00, 33.07)^T$ . Führen wir  $\hat{x}$  auf  $x$  mittels  $U \cdot \hat{x}$  zurück, erhalten wir korrekterweise  $x = (100, 10, 1)^T$ .

Es gilt  $\lambda_{\max} = 3$  Jetzt ist  $\tilde{\Lambda}$  definiert als

$$\tilde{\Lambda} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -\frac{1}{3} & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.52)$$

Wir überprüfen die Approximation durch die Polynome mit  $k = 2$ :

$$g_\theta(\Lambda) = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, g_{\theta_{prime}} = (wd) \quad (6.53)$$

## A. Weitere Informationen

# Symbolverzeichnis

$T$  Tschebyschow-Polynom. 15

$\deg$  Gradfunktion der Knoten eines Graphen  $G$  mit  $\deg: \mathcal{V} \rightarrow \mathbb{N}$ . 4

$\lambda$  Eigenwert. 8–15

$\langle \cdot, \cdot \rangle$  Skalarprodukt. 3

$\mathbb{N}$  Menge der natürlichen Zahlen. 4, 8, 11

$\mathbb{R}_+$  Menge der positiven reellen Zahlen inklusive Null. 3, 4, 9, 11, 17

$\mathbb{R}$  Menge der reellen Zahlen. 3, 4, 6, 8, 10–12, 14, 15, 17

$\mathcal{E}$  Kantenmenge eines Graphen  $G$  mit  $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ . 3, 4, 15

$\mathcal{O}$  O-Notation. 15

$\mathcal{V}$  Knotenmenge  $\{v_i\}_{i=1}^n$  eines Graphen  $G$ . 3, 4, 10, 12

$\tilde{\mathcal{L}}$  reskalierter Laplacian  $\mathcal{L}$ . 15

$\text{diag}$  Diagonalfunktion. 4, 8, 13

$\odot$  Hadamard-Produkt. 3, 13

$\perp$  Orthogonalität. 8

$\sim$  Adjazenzrelation zweier Knoten eines Graphen  $G$  mit  $u \sim v$  genau dann, wenn  $u$  und  $v$  adjazent. 3, 4, 6

$d$  gewichtete Gradfunktion der Knoten eines Graphen  $G$  mit  $d: \mathcal{V} \rightarrow \mathbb{R}_+$ . 4

$p$  Positionsfunktion auf den Knoten  $\mathcal{V}$  mit  $p: \mathcal{V} \rightarrow \mathbb{R}^2$ . 4, 6

$s$  kürzeste Distanzfunktion mit  $s: \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{N}$ . 4, 11, 14

$w$  Gewichtsfunktion der Kanten eines Graph  $G$  mit  $w: \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}_+$ . 3, 4, 6, 11

$\mathbf{A}$  Adjazentmatrix eines Graphen  $G$ . 4, 9

$\mathbf{D}$  gewichtete Gradmatrix. 4, 9

**I** Identitätsmatrix. 3, 8, 9, 15  
**L** Laplacian, unnormalisiert. 9–11, 14  
**U** Eigenvektormatrix. 8, 12–15  
 **$\Lambda$**  Diagonalmatrix der Eigenwerte des Laplacian. 8, 13–15  
 **$\mathcal{L}$**  Laplacian, normalisiert oder unnormalisiert. 9–12, 14, 15  
 **$\tilde{\mathbf{L}}$**  Laplacian, normalisiert. 9, 11  
 **$\tilde{\Lambda}$**  reskalierte Diagonalmatrix der Eigenwerte des Laplacian. 15  
 **$G$**  Graph. 3, 4, 9, 11  
 **$\mathbf{u}$**  Eigenvektor mit  $\|\mathbf{u}\|_2 = 1$ . 8, 10–12



# Abbildungsverzeichnis

- 6.1. Aufteilung einer Adjazenzmatrix in vier räumlich eingebettete Bereiche. . . 18
- 6.2. Abbildung der lokalen Nachbarschaftsknoten auf den Einheitskreis. . . . . 19





## List of Algorithms

