

## Master-Thesis

# Convolutional Neural Networks auf Graphrepräsentationen von Bildern

Matthias Fey

20. Februar 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>3</b>
<b>3. Spektrale Graphentheorie</b>	<b>5</b>
3.1. Spectral Graph Domain . . . . .	6
3.2. Diskrete Fourier Transformation . . . . .	7
3.3. Faltung . . . . .	7
3.3.1. Faltung in CNNs . . . . .	7
3.3.2. Faltung auf Graphen . . . . .	8
3.3.3. Offene Fragen . . . . .	8
3.3.4. Beispiel . . . . .	8
3.4. Chebyshev Polynome . . . . .	9
3.5. Probleme . . . . .	9
<b>4. Graph Convolutional Networks</b>	<b>11</b>
4.1. Erweiterung für mehrere Kantenattribute . . . . .	11
4.1.1. Übertragung auf räumlich eingebettete Graphen . . . . .	12
4.2. Tschebyschow-Polynome . . . . .	14
4.3. Pooling-Ebene . . . . .	15
4.3.1. Clustering von Graphen . . . . .	15
4.3.2. Pooling-Operation . . . . .	16
<b>A. Weitere Informationen</b>	<b>17</b>
<b>Symbolverzeichnis</b>	<b>19</b>
<b>Abbildungsverzeichnis</b>	<b>21</b>
<b>Algorithmenverzeichnis</b>	<b>23</b>
<b>Literaturverzeichnis</b>	<b>25</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)

Graphrepräsentationen von Bildern Grid Superpixel Superpixelalgorithmen 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)

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

Zusammenfassung und Ausblick



## 2. Einleitung

$\mathbb{R}$  und  $\mathbb{N}$  sind mathematische Symbole [?].





### 3. Spektrale Graphentheorie

- *Spektrum* eines Graphen zur Untersuchung seiner Eigenschaften
- *algebraische* oder *spektrale Graphentheorie* genannt
- als Spektrum eines Graphen bezeichnet man die (nach Größe geordnete) Folge der Eigenwerte  $\lambda$  seiner Adjazenzmatrix, d.h.  $A \cdot x = \lambda x$  ( $x$  Eigenvektoren)

Algebraische Methoden sind sehr effektiv bei Graphen, die regulär und symmetrisch sind. Als *Schleife* wird in der Graphentheorie eine Kante bezeichnet, die einen Knoten mit sich selbst verbindet. Ein Graph ohne Schleifen wird *schleifenloser* Graph genannt.

Sei  $d_v$  der Grad eines Knotens  $v$  eines Graphen  $G$ . Der *Laplacian*  $\mathcal{L}$  eines Graphen ohne Schleifen und Mehrfachkanten ist definiert als

$$\mathcal{L}(u, v) = \begin{cases} d_v, & \text{wenn } u = v, \\ -1, & \text{wenn } u \text{ und } v \text{ adjazent,} \\ 0, & \text{sonst.} \end{cases} \quad (3.1)$$

Der Graph Laplacian ist eine Generalisierung des Laplacian auf einem Gitter.

Damit ist  $\mathcal{L} = D - A$ .  $\mathcal{L}$  kann normalisiert werden über  $\mathcal{L}_{\text{norm}} = T^{-\frac{1}{2}} \mathcal{L} T^{-\frac{1}{2}}$ , wobei  $T$  die Diagonalmatrix beschreibt mit  $T(v, v) = d_v$ . Für einen *isolierten* Knoten  $v$ , d.h.  $d_v = 0$ , gilt die Konvention  $T^{-1}(v, v) = 0$ . Ebenso lässt sich  $\mathcal{L}_{\text{norm}}$  definieren als

$$\mathcal{L}_{\text{norm}}(u, v) = \begin{cases} 1, & \text{wenn } u = v \text{ und } d_v \neq 0, \\ -\frac{1}{\sqrt{d_u d_v}}, & \text{wenn } u \text{ und } v \text{ adjazent,} \\ 0, & \text{sonst.} \end{cases} \quad (3.2)$$

Wenn  $G$   $k$ -regulär ist, d.h.  $T = \text{diag}(k)$ , dann gilt  $\mathcal{L}_{\text{norm}} = I - \frac{1}{k} A$ .

Da  $\mathcal{L}$  symmetrisch ist, sind seine Eigenwerte alle  $\geq 0$  (d.h.  $\mathcal{L}$  ist positiv-semidefinit). Jede Reihen- und Spaltensumme von  $\mathcal{L}$  ist 0.

Einem gewichtetem ungerichteten Graph  $G$  kann eine Gewichtsfunktion  $w : V \times V \rightarrow \mathbb{R}$  zugeschrieben werden, sodass  $w(u, v) = w(v, u)$  und  $w(u, v) \geq 0$ . Falls  $\{u, v\} \notin \mathcal{E}$ , dann  $w(u, v) = 0$ . Damit sind ungewichtete Graphen nur ein Spezialfall bei dem alle Gewichte 0 oder 1 sind. Der Grad  $d_v$  eines Knoten  $v$  ist dann definiert als

### 3. Spektrale Graphentheorie

$$d_v = \sum_u w(u, v). \quad (3.3)$$

Dann gilt

$$\mathcal{L} = \begin{cases} 1 - \frac{w(v,v)}{d_v}, & \text{wenn } u = v \text{ und } d_v \neq 0, \\ -\frac{w(u,v)}{\sqrt{d_u d_v}}, & \text{wenn } u \text{ und } v \text{ adjazent,} \\ 0, & \text{sonst.} \end{cases} \quad (3.4)$$

Bemerke, dass hier Schleifen nicht explizit ausgeschlossen werden!

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

$$w(x, v^*) = w(x, u) + w(x, v) \quad (3.5)$$

und

$$w(v^*, v^*) = w(u, u) + w(v, v) + 2w(u, v) \quad (3.6)$$

Mit  $\lambda_G := \lambda_1$  für einen Graphen  $G$ , gilt für einen Graphen  $H$  der aus  $G$  verkleinert wurde

$$\lambda_G \leq \lambda_H \quad (3.7)$$

### 3.1. Spectral Graph Domain

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

Eine beliebige Funktion  $f : V \rightarrow \mathbb{R}$  kann als ein Vektor in  $\mathbb{R}^n$  gesehen werden. Dies impliziert eine Ordnung auf den Knoten. Wir schreiben  $f \in \mathbb{R}^n$  für Funktionen auf den Knoten eines Graphen und  $f(m)$  für den Wert des  $m$ ten Knoten.

Dann gilt für eine beliebige Funktion  $f \in \mathbb{R}^n$

$$\mathcal{L}f(x) = \sum_y w(x, y) \cdot (f(x) - f(y)) \quad (3.8)$$

wobei die Summe über  $x, y$  die Summierung über alle Knoten  $y$  beschreibt, die adjazent zu  $x$  sind.

Angenommen  $G$  ist als ein reguläres Gitter definiert der Breite und Höhe  $M$ . Dann hat ein Knoten  $v_{x,y}$  genau 4 Nachbarn mit Kantengewicht  $\frac{1}{(\delta w)^2}$ , bei dem  $\delta w$  die euklidische Distanz zwischen zwei Gitterpunkten beschreibt.

Für eine Funktion  $f : M \times M \rightarrow \mathbb{R}$  gilt dann:

$$\mathcal{L}f(x, y) = \frac{4f(x, y) - f(x+1, y) - f(x-1, y) - f(x, y+1) - f(x, y-1)}{(\delta w)^2} \quad (3.9)$$

Damit kann ein Signal  $f$  mit der Multiplikation mit  $\mathcal{L}$  als eine Weiterpropagation von  $f$  unter der Berücksichtigung der lokalen Nachbarn verstanden werden (*5-point Stencil*, d.h.  $\mathcal{L}f \approx -\nabla^2 f$ ).

## 3.2. Diskrete Fourier Transformation

$\mathcal{L}$  besitzt genau  $n$  orthogonal zueinander stehende Eigenvektoren  $\{u_l\}_{l=1}^n \in \mathbb{R}^n$ . Eigenvektoren  $u_i$  sind auf 1 normiert, d.h.  $\|u_i\|_2 = 1$ . Diese werden auch *Graph Fourier Modes* genannt. Diesen sind Eigenwerte  $\{\lambda_l\}_{l=1}^n \in \mathbb{R}$  zugeordnet, die die „Frequenzen“ bzw. das Spektrum des Graphen beschreiben oder visuell betrachtet die Ausdehnung des Raumes, den die Eigenvektoren aufspannen. Bemerke dass  $\lambda_0 = 0$ , da für den Eigenvektor  $\vec{u}_0 = (1, 1, \dots, 1)^T$  gilt, dass  $\mathcal{L}\vec{u}_0 = 0$ .  $\mathcal{L}$  ist diagonalisierbar über  $\mathcal{L} = U\Lambda U^T$ , wobei  $U = [u_1, \dots, u_n] \in \mathbb{R}^{n \times n}$  die *Fourier Basis* und  $\Lambda = \text{diag}([\lambda_0, \dots, \lambda_n]) \in \mathbb{R}^{n \times n}$ . Die *Fourier Transformation* eines Signals  $x \in \mathbb{R}^n$  ist dann definiert als  $\hat{x} = U^T x$  und die Inverse als  $x = U\hat{x}$ .

## 3.3. Faltung

Wir suchen einen Operator  $x *_G g$ , der eine Faltung zweier Eingangssignale  $x, g$  zu einem Ausgangssignal umleitet.  $x$  beschreibt dabei die Knotenattribute und  $g$  die Gewichte.

### 3.3.1. 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:

$$\begin{pmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 12 & 16 & 11 \\ 24 & 28 & 17 \end{pmatrix} \quad (3.10)$$

### 3. Spektrale Graphentheorie

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

$$(f * g)(x, y) = \sum_{x_i \in [x, x+1] y_i \in [y, y+1]} f(x_i, y_i) g(x - x_i, y - y_i) \quad (3.11)$$

#### 3.3.2. Faltung auf Graphen

Da wir keinen Translationsoperator auf der Domäne der Knoten  $x$  beschreiben können, müssen wir unseren Faltungsoperator in der Fourier-Domäne beschreiben. Dafür wandeln wir unsere Knotenmenge  $x$  zuerst in  $\hat{x}$  um.

Wir definieren  $*_G$  in der Fouier-Domäne als

$$x *_G g = U \cdot (U^T \cdot x \odot \hat{g}) \quad (3.12)$$

wobei  $\odot(A, B) = (a_{ij} \cdot b_{ij})$  die elementweise Multiplikation bzw. das *Hadamard-Produkt*.

Das Hadamard-Produkt löst sich auf, wenn  $\hat{g}$  als eine Diagonalmatrix repräsentiert wird. Dann gilt

$$x *_G g = U \begin{pmatrix} \hat{g}(\lambda_0) & \cdots & 0 \\ 0 & \cdots & \hat{g}(\lambda_n) \end{pmatrix} U^T x = U \hat{g}(\Lambda) U^T x \quad (3.13)$$

Dann beschreibt  $\hat{g}(\Lambda) = \text{diag}(\theta)$  eine Gewichtsfunktion mit  $n$  Variablen,  $\theta \in \mathbb{R}^n$ . Damit ist die Faltung bzw. die Gewichtung abhängig von der Input-Größe  $n$ , was extrem schlecht ist.

#### 3.3.3. Offene Fragen

- Warum wird  $\hat{g}$  als Diagonalmatrix repräsentiert?
- Wie kommt die Convolution zustande mit dem  $*$  Operator?
- Was passiert bei gerichteten Graphen???? Wir haben keinen symmetrischen und insbesondere keinen positiv definiten

#### 3.3.4. Beispiel

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

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \quad (3.14)$$

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

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

$$\mathcal{L} = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix} \quad (3.15)$$

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

$$\mathcal{L} \cdot \vec{u} = \lambda \cdot \vec{u} \quad (3.16)$$

Wir erhalten 3 Eigenvektoren und Eigenwerte mit

$$\lambda_0 = 0, \vec{u}_0 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.58 \\ 0.58 \\ 0.58 \end{pmatrix}, \lambda_1 = 1, \vec{u}_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \approx \begin{pmatrix} -0.71 \\ 0 \\ 0.71 \end{pmatrix}, \lambda_2 = 3, \vec{u}_2 = \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.41 \\ -0.82 \\ 0.41 \end{pmatrix} \quad (3.17)$$

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

$$U \approx \begin{pmatrix} 0.58 & -0.71 & 0.41 \\ 0.58 & 0 & -0.82 \\ 0.58 & 0.71 & 0.41 \end{pmatrix}, \Lambda = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix}, U^T \approx \begin{pmatrix} 0.58 & 0.58 & 0.58 \\ -0.71 & 0 & 0.71 \\ 0.41 & -0.82 & 0.41 \end{pmatrix} \quad (3.18)$$

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  $x = (100, 10, 1)^T$ .

## 3.4. Chebyshev Polynome

## 3.5. Probleme

Rotationsinvariant



## 4. Graph Convolutional Networks

$$H^{(l+1)} = f(H^{(l)}, A) \quad (4.1)$$

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)}) \quad (4.2)$$

$$D_{ii} = \sum_j A_{ij} \quad (4.3)$$

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

$$D^x = \begin{pmatrix} d_{11} & 0 & \cdots & 0 \\ 0 & d_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{nn} \end{pmatrix}^x = \begin{pmatrix} d_{11}^x & 0 & \cdots & 0 \\ 0 & d_{22}^x & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{nn}^x \end{pmatrix} \quad (4.4)$$

### 4.1. 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  $n$ .

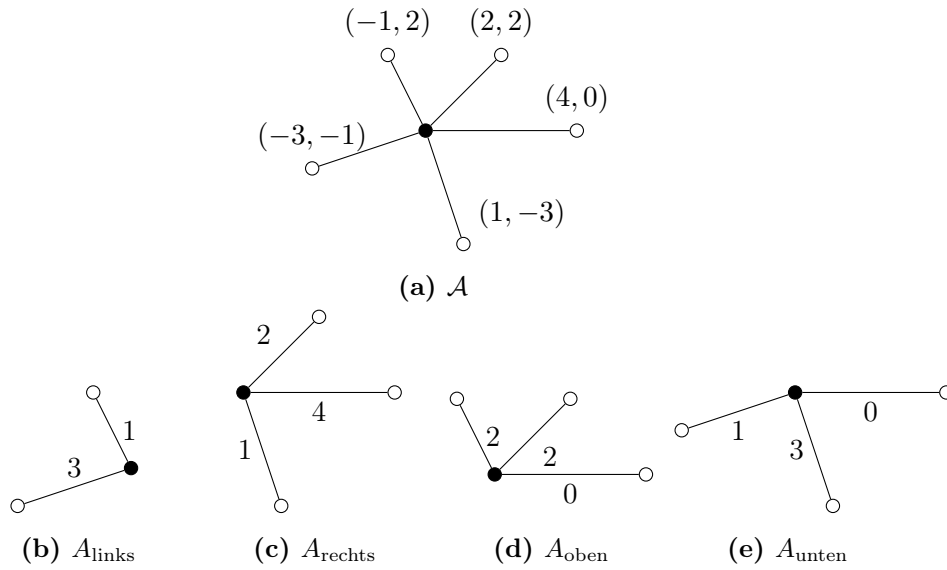
$\mathcal{A} \in \mathbb{R}^{m \times n \times n}$  kann zu einer zweidimensionalen Matrix  $A \in \mathbb{R}^{m \cdot n \times n}$  geglättet werden.

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  $G \in \mathbb{R}^{m \cdot d \times x}$ .

$$H^{(l+1)} = f(H^{(l)}, \tilde{\mathcal{A}}) = \sigma \left( \frac{1}{|\tilde{\mathcal{A}}|} \sum_{\tilde{A}_i \in \tilde{\mathcal{A}}} \tilde{D}_i^{-\frac{1}{2}} \tilde{A}_i \tilde{D}_i^{-\frac{1}{2}} H^{(l)} W_i^{(l)} \right) \quad (4.5)$$

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

## 4. Graph Convolutional Networks



**Abbildung 4.1.:** Aufteilung einer Adjazenzmatrix in vier räumlich eingebettete Bereiche.

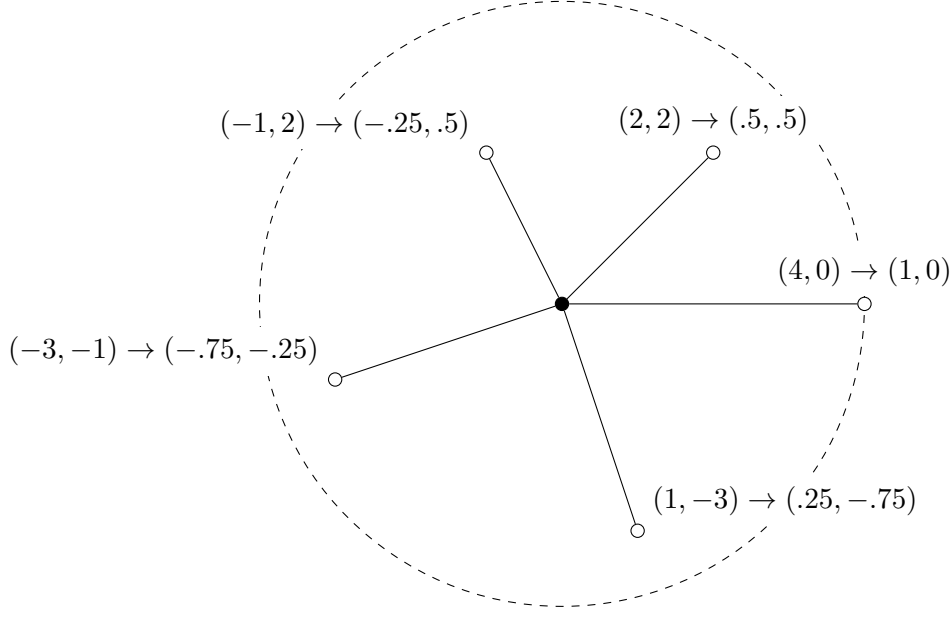
### 4.1.1. Übertragung auf räumlich eingebettete Graphen

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 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 4.1). 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





**Abbildung 4.2.:** Abbildung der lokalen Nachbarschaftsknoten auf den Einheitskreis.

der längsten Distanz zum Wurzelknoten genau auf dem Einheitskreis liegt (vgl. Abbildung 4.2).

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

$$\tilde{A}_{ij} = \begin{cases} 1, & \text{falls } i = j, \\ (a_{ij} + 1)^{-1}, & \text{falls } a_{ij} \neq 0, \\ 0, & \text{sonst.} \end{cases} \quad (4.6)$$

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

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

## 4.2. Tschebyschow-Polynome

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

$$T_n(x) = 2x \cdot T_{n-1}(x) - T_{n-2}(x) \quad (4.7)$$

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

## 4.3. Pooling-Ebene

### 4.3.1. 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!!

brauch ich  
garnicht

Defferrard et al. [2] benutzen für die Pooling-Ebene eines Netzes auf Graphen die *Vergrößerungsphase* des mehrstufigen Cluster-Algorithmus *Graculus* [3]. 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 [3].

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*).

erklären

Graculus 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 Graculus nur sehr wenige Singleton-Knoten generiert [2].

#### 4. Graph Convolutional Networks

Nach der spektralen Graphentheorie [1] gilt für Kanten eines Graphen  $G = (\mathcal{V}, \mathcal{E})$  nach Verschmelzung von  $u$  und  $v$  zu  $v^*$

$$w_{xv^*} = w_{xu} + w_{xv} \quad (4.8)$$

$$w_{v^*v^*} = w_{uu} + w_{vv} + 2w_{uv} \quad (4.9)$$

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  $H$  [1].

#### Übertragung auf planare Graphen

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

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

Die Kanten,  
die nicht  
berechnet werden  
müssen!

#### 4.3.2. 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:**  $v^* := \|u, v\|_2$

## A. Weitere Informationen



# Symbolverzeichnis

$\mathbb{N}$  Menge der natürlichen Zahlen. 3

$\mathbb{R}^+$  Menge der positiven reellen Zahlen inklusive Null. 11

$\mathbb{R}$  Menge der reellen Zahlen. 3, 11





# Abbildungsverzeichnis

4.1. Aufteilung einer Adjazenzmatrix in vier räumlich eingebettete Bereiche. . .	12
4.2. Abbildung der lokalen Nachbarschaftsknoten auf den Einheitskreis. . . . .	13



## List of Algorithms



# Literaturverzeichnis

- [1] CHUNG, F. R. K.: *Spectral Graph Theory*. American Mathematical Society, 1997.
- [2] DEFFERRARD, M., X. BRESSON und P. VANDERGHEYNST: *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*. CoRR, 2016.
- [3] DHILLON, I. S., Y. GUAN und B. KULIS: *Weighted Graph Cuts Without Eigenvectors: A Multiscale Approach*. IEEE, Seiten 1944–1957, 2007.



# Eidesstattliche Versicherung

---

Name, Vorname

---

Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit\* mit dem Titel

---

---

---

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

---

Ort, Datum

---

Unterschrift

\*Nichtzutreffendes bitte streichen

## Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG - )

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

---

Ort, Datum

---

Unterschrift