technische universität
dortmund

fakultät für informatik

Master-Thesis

Convolutional Neural Networks auf Graphrepräsentationen von Bildern

Matthias Fey

24. Februar 2017

Gutachter:
Prof. Dr. Heinrich Müller
M.Sc. Jan Eric Lenssen

Lehrstuhl Informatik VII
Graphische Systeme
TU Dortmund

fakultät für
informatik

Inhaltsverzeichnis

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 Merkmalextraktion (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

2.1. Motivation

2.2. Aufbau der Arbeit

3. Grundlagen

3.1. Notationen

diag: $\mathbb{R}^n \to \mathbb{R}^{n\times n}$ 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 $\text{diag}^{-1}: \mathbb{R}^{n\times n} \to \mathbb{R}^n$ liefert zu einer beliebigen Diagonalmatrix $\mathbf{D}$ dessen Diagonalvektor $\mathbf{d}$. brauch ich glaub ich nicht mehr

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

Identitätsmatrix $\mathbf{I}$

awda

3.2. Graphentheorie

Graph Tupel $G = (\mathcal{V}, \mathcal{E})$
$\mathcal{V} = \{v_i\}_{i=1}^n$
$|\mathcal{V}| = n < \infty$
Merkmalsfunktion $f_G: \mathcal{V} \to \mathbb{R}^m$
wenn nicht explizit aufgeführt, dann bla bla wir $f_G: \mathcal{V} \to \mathbb{R}$
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 $w: \mathcal{V} \times \mathcal{V} \to \mathbb{R}_+$
ungewichtet: $w: \mathcal{V} \times \mathcal{V} \to \{0, 1\}$
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

$$w(u, v) = w(v, u) \tag{3.1}$$

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 ist eindeutig definiert durch $\mathbf{A}$ und f_G.

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

$$\deg(v) = \sum_{v \sim u} 1 \tag{3.2}$$

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

$$d(v_i) = \sum_{j=1}^{n} \mathbf{A}_{ij} \tag{3.3}$$

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

$$\mathbf{D} = \operatorname{diag}\Big([d(v_1), \dots, d(v_n)]^\top \Big) \tag{3.4}$$

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 $p \colon \mathcal{V} \to \mathbb{R}^2$ zugeordnet, die die Position eines Knotens auf der Ebene eindeutig definiert.

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

immt noch
ganz, ge-
auch
ere Kno-

nd graph
onenten,
eren

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 $s \colon \mathcal{V} \times \mathcal{V} \to \mathbb{N}$ 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.

4. Graphrepräsentationen von Bildern

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

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.

6.1.1. Eigenwerte, Eigenvektoren und Eigenfunktionen

$\mathbf{Mv} = \lambda\mathbf{v}$
Zu einem Eigenwert λ gibt es unendlich viele (skalierte) Eigenvektoren $\mathbf{v}$ Eigenvektor $\mathbf{v}$ ist dann eindeutig definiert mit $\|\mathbf{c}\|_2 = 1$ Wenn $\mathbf{M}$ symmetrisch ist und $\mathbf{v}_1$ und $\mathbf{v}_2$ zwei unterschiedliche Eigenvektoren, dann sind $\mathbf{v}_1 \perp \mathbf{v}_2$ Jede symmetrische Matrix hat n Eigenwerte mit $\lambda_0 \leq \cdots \lambda_{n-1}$

6.1.2. Der Laplacian und seine Eigenwerte

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

Der Laplacian $\mathbf{L}$ eines Graphen G ist definiert als $\mathbf{L} = \mathbf{D} - \mathbf{A}$ [2]. Der normalisierte Laplacian $\mathbf{L}_{\text{norm}}$ ist definiert als $\mathbf{L}_{\text{norm}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{L}\mathbf{D}^{-\frac{1}{2}}$ [2]. Es gilt die Konvention, dass $\left(\mathbf{D}^{-\frac{1}{2}}\right)_{ii} = 0$ falls $\mathbf{D}_{ii} = 0$ Für verbundene Graphen gilt weiterhin $\mathbf{L}_{\text{norm}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ [2]. Wenn G k-regulär ist, dann gilt $\mathbf{L}_{\text{norm}} = \mathbf{I} - \frac{1}{k}\mathbf{A}$. [2]

$\mathbf{L}$ und $\mathbf{L}_{\text{norm}}$ sind keine ähnlichen Matrizen. Insbesondere sind ihre Eigenvektoren unterschiedlich. Die Nutzung des $\mathbf{L}$ oder $\mathbf{L}_{\text{norm}}$ ist damit abhängig von dem Problem, welches man betrachtet. [1].

Wir schreiben $\mathcal{L}$ wenn die Wahl des Laplacian (unnormalisiert, normalisiert) irrelevant ist.

Jede Reihen- und Spaltensumme von $\mathcal{L}$ ist 0, d.h. $\sum_i \mathcal{L}_{ij} = 0$ und $\sum_i \mathcal{L}_{ji} = 0$ für alle $i \in \{1, \ldots, n\}$. $\mathcal{L} \in \mathbb{R}^{n \times n}$ hat genau n Eigenwerte $\{\lambda_i\}_{i-0}^{n-1}$. $\mathcal{L}$ ist eine symmetrische reelle Matrix, d.h. insbesondere liegen ihre Eigenwerte λ_i in $\mathbb{R}_+$.
Anzahl der Eigenvektoren gleich Null ist die Anzahl an Komponenten, die ein Graph besitzt. $0 = \lambda_0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_{n-1}$ wenn Graph verbunden.
$\lambda_0 = 0$, da $\mathbf{u}_0 = [1, \ldots, 1]^\top$ Eigenvektor von $\mathcal{L}$

das gilt nat lich nur im gewichteten also weg?

was gilt der alles im gew teten?

stimmt das $\mathbf{L}_{\text{norm}}$?

positive sem finitheit

Lambda Or ring

quelle

quelle

$\mathbf{\Lambda} = \text{diag}([\lambda_0, \ldots, \lambda_{n-1}])$

la upper
l

Für einen Graphen G definieren wir $\lambda_G := \lambda_1$ und $\lambda_{\max} := \lambda_{n-1}$

Für $\mathbf{L}_{\text{norm}}$ gilt $\lambda_{\max} \leq 2$

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

$$w(x, v^*) = w(x, u) + w(x, v) \tag{6.1}$$

$$w(v^*, v^*) = w(u, u) + w(v, v) + 2w(u, v) \tag{6.2}$$

Für einen Graphen G, gilt für einen Graphen H der aus G verkleinert wurde

$$\lambda_G \leq \lambda_H \tag{6.3}$$

6.2. 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 $f : V \to \mathbb{R}$ 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 $f \in \mathbb{R}^n$

$$\mathcal{L}f(x) = \sum_{x\ y} w(x, y) \cdot (f(x) - f(y)) \tag{6.4}$$

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 δw die euklidsche Distanz zwischen zwei Gitterpunkten beschreibt.

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

$$\mathcal{L}f(x, y) = \frac{4f(x, y) - f(x + 1, y) - f(x - 1, y) - f(x, y + 1) - f(x, y - 1)}{(\delta w)^2} \tag{6.5}$$

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. $\mathcal{L}f \approx -\nabla^2 f$).

6.3. 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 Eigenwete $\{\lambda_l\}_{l=1}^n \in \mathbb{R}$ 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 $\vec{u_0} = (1, 1, \ldots, 1)^T$ gilt, dass $\mathcal{L}\vec{u_0} = 0$. $\mathcal{L}$ ist diagonalisierbar über $\mathcal{L} = U\Lambda U^T$, wobei $U = [u_1, \ldots, u_n] \in \mathbb{R}^{n\times n}$ die *Fourier Basis* und $\Lambda = \text{diag}([\lambda_0, \ldots, \lambda_n]) \in \mathbb{R}^{n\times n}$. Die *Fourier Transformation* eines Signals $x \in \mathbb{R}^n$ ist dann definiert als $\hat{x} = U^T x$ und die Inverse als $x = U\hat{x}$.

6.4. 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.

6.4.1. Faltung in CNNs

In der Funktionalanalysis beschreibt die *Faltung* einen mathematischen Operator, der für zwei Funktion f und g eine dirtte Funktion $f * g$ liefert. Die Faltung kann als ein Produkt von Funktionen vertanden 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 3×4 und unsere Filtergröße 2×2.

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

$$\begin{pmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 12 & 16 & 11 \\ 24 & 28 & 17 \end{pmatrix} \tag{6.6}$$

$f : 3 \times 4 \to \mathbb{R}$ und $g : 2\ times2 \to \mathbb{R}$, dann ist $*$ definiert als

$$(f * g)(x, y) = \sum_{x_i \in [x,x+1] y_i \in [y,y+1]} f(x_i, y_i) g(x - x_i, y - y_i) \tag{6.7}$$

6.4.2. 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

$$x *_G g = U \cdot (U^T \cdot x \odot \hat{g}) \tag{6.8}$$

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

$$x *_G g = U \begin{pmatrix} \hat{g}(\lambda_0) & \cdots & 0 \\ 0 & \cdots & \hat{g}(\lambda_n) \end{pmatrix} U^T x = U \hat{g}(\Lambda) U^T x \tag{6.9}$$

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.

6.4.3. Offene Fragen

- Wie erklärt sich noch einmal der normalisierte Laplacian?
- 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

6.5. Chebyshev Polynome

6.6. Probleme

Rotationsinvariant

6.7. Pfadlänge

wenn $d_G(m, n) > k$, dann $(L^k)_{m,n} = 0$ (normalisiert sowie unnormalisiert (siehe Wavelet Lemma 5.4))

6.8. Polynomielle Approxmiation

- bisheriger Ansatz skaliert nicht gut für große Graphen
- schneller Algorithmus zur Approximation des Filters notwendig ⇒ Polynome niedriger Ordnung
- Größe des Filters soll unahängig zu den Daten sein
- approximiere $g(\mathcal{L})$ durch Polynom, dass rekursiv durch $\mathcal{L}$ berechnet werden kann

Beweisidee: $\tilde{\mathbf{L}}^k = \left(\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top\right)^k = \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^\top$ [5].
Das sieht man leicht: $\mathbf{U}\left(\sum_i \mathbf{\Lambda}^k\right)\mathbf{U}^\top = \sum_i \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^\top = \sum_i \mathbf{L}^k$

Diese Polynome formen eine Orthogonalbasis Polynome formen eine Orthogonalbasis für $L^2\left([-1,1], \frac{d_x}{\sqrt{1-x^2}}\right)$, auch *Hilbertraum* genannt
Spektrale Filter, die repräsentiert werden durch ein Polynom vom Grad k sind *k-lokalisiert.*

stable recur
property

die Grundid
der Polynor
alsierung m
hierhin + ü
gang zu Ch
hev

6.8.1. Tschebyschow-Polynome

- bisher Filterung eines Signals $\mathbf{x}$ zu $\mathbf{y} = \mathbf{U}g_\theta(\mathbf{\Lambda})\mathbf{U}^\top\mathbf{x} \approx \mathbf{U}g'_{\theta'}(\mathbf{\Lambda})\mathbf{U}^\top\mathbf{x} = g'_{\theta'}(\mathbf{L})\mathbf{x}$
- $g_\theta(\mathbf{\Lambda})$ kann über ein Polynom $k$ten Grades $g'_{\theta'}(\mathbf{\Lambda})$ approximiert werden, $\theta' \in \mathbb{R}^k$
- Aber: Filterung ist sehr teuer aufgrund der Multiplikation der dichten Matrix $\mathbf{U}$, d.h. $\mathcal{O}(n^2)$
- Lösung: Parametrisiere $g_\theta(\mathbf{L})$ als eine polynomielle Funktion, die rekursiv aus $\mathbf{L}$ berechnet werden kann.
- Warum sollte das effizienter sein? $\mathbf{L}$ 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 $T_n(x)\colon \mathbb{R} \to \mathbb{R}$ mit dem rekursiven Zusammenhang

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x) \tag{6.10}$$

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 $\tilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{\max}}\mathbf{\Lambda} - \mathbf{I} \in [-1,1]^{n\times n}$. $\lambda_{\max}$ ist der Wert des größten Eigenvektors von $\mathbf{L}$.
Dann ist $T_k\left(\tilde{\mathbf{\Lambda}}\right) \in [-1,1]^{n\times n}$

$$g_\theta(\mathbf{\Lambda}) \approx g'_{\theta'}(\mathbf{\Lambda}) = \sum_{i=0}^{k-1} \theta'_i T_i\Big(\tilde{\mathbf{\Lambda}}\Big) \tag{6.11}$$

sidee nach-
n

Es zeigt sich, dass

$$\mathbf{U} g'_{\theta'}(\mathbf{\Lambda}) \mathbf{U}^\top = g'_{\theta'}(\mathbf{L}) \tag{6.12}$$

wobei $g'_{\theta'}(\mathbf{L}) = \sum_{i=0}^{k-1} \theta_i T_i\Big(\tilde{\mathbf{L}}\Big)$ mit $\tilde{\mathbf{L}} = \mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{U}^\top = \frac{2}{\lambda_{\max}}\mathbf{L} - \mathbf{I}$
Jetzt lässt sich $y = g'_{\theta'}(\mathbf{L})\mathbf{x} = \sum_{i=0}^{k-1} \theta'_i T_i\Big(\tilde{\mathbf{L}}\Big)\mathbf{x}$ sehr schnell berechnen:

1. berechnne $\overline{\mathbf{x}}_i := T_k\Big(\tilde{\mathbf{L}}\Big)\mathbf{x}$ für alle $i \in \{0, 1, \ldots, k-1\}$ mit Hilfe von Rekursion:
   a) $\overline{\mathbf{x}}_0 = \mathbf{x}$
   b) $\overline{\mathbf{x}}_1 = \tilde{\mathbf{L}}\mathbf{x}$
   c) $\overline{\mathbf{x}}_i = 2\tilde{\mathbf{L}}\overline{\mathbf{x}}_{i-1} - \overline{\mathbf{x}}_{i-2}$
2. berechne $\mathbf{y} = [\overline{\mathbf{x}}_0, \overline{\mathbf{x}}_1, \ldots, \overline{\mathbf{x}}_{k-1}]\theta'$

Laufzeit

- anstatt $\mathbf{L}^k$ zu berechnen mit Komplexität $\mathcal{O}\big(n^2\big)$ haben wir nur noch k Multiplikationen mit der Matrix $\tilde{\mathbf{L}}$
- da $\tilde{\mathbf{L}}$ 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 $\mathcal{O}(k|\mathcal{E}|)$ [1, 4]
- für den zweiten Schritt gilt $\mathcal{O}(kn)$, mit $n \leq |\mathcal{E}|$ bedingt damit nur Schritt Eins die Laufzeit

6.9. Graph Convolutional Networks

- aus [5]
- Idee: setze $k = 2$ für alle Faltungsebenen
- Begründung: Faltung über $i < k = 2$ 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 3×3 keine negativen Auswirkungen haben
- viele kleine Faltungsebenen ohne Pooling propagieren die Informationen weit entfernterer Knoten weiter

Paper,
ne et al,
esidual
ng

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

Annahme: $\lambda_{\max} \approx 2$ Es gilt $0 \leq \lambda_0 \leq \cdots \leq \lambda n - 1 \leq 2$ [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 $\tilde{\mathbf{L}} = \mathbf{L} - \mathbf{I}$.

ist das begr
det, woher
kommt dies
Zahl

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

$$g_\theta \star \mathbf{x} = \mathbf{U} g_\theta(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{x} \approx g'_{\theta'}(\mathbf{L})\mathbf{x} = \sum_{i=0}^{k-1} \theta'_i T_i(\mathbf{L} - \mathbf{I})\mathbf{x} = \theta'_0 \mathbf{x} + \theta'_1 (\mathbf{L} - \mathbf{I})\mathbf{x} \tag{6.13}$$

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

$$g_\theta \star \mathbf{x} \approx \theta'_0 \mathbf{x} - \theta'_1 \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{x} \tag{6.14}$$

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

$$g_\theta \star \mathbf{x} \approx \theta\left(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)\mathbf{x} \tag{6.15}$$

$\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ hat nun Eigenwerte im Bereich $[0, 2]$. Wiederholte Anwendungen dieses Operators können daher zu numerischen Instabilitäten und dann zu explodierenden oder verschwindenen 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 $\tilde{\mathbf{A}}$.

warum?

warum

$$g_\theta \star \mathbf{x} \approx \theta\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\right)\mathbf{x} \tag{6.16}$$

$$H^{(l+1)} = f(H^{(l)}, A) \tag{6.17}$$

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)}) \tag{6.18}$$

$$D_{ii} = \sum_j A_{ij} \tag{6.19}$$

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

$$D^x = \begin{pmatrix} d_{11} & 0 & \cdots & 0 \\ 0 & d_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{nn} \end{pmatrix}^x = \begin{pmatrix} d_{11}^x & 0 & \cdots & 0 \\ 0 & d_{22}^x & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{nn}^x \end{pmatrix} \tag{6.20}$$

6.10. 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 gewichte 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 $\mathcal{A} = \{A_1, A_2, \ldots, A_m\}$ 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 $G \in \mathbb{R}^{m \cdot d \times x}$.

$$H^{(l+1)} = f(H^{(l)}, \tilde{\mathcal{A}}) = \sigma\left(\frac{1}{|\tilde{\mathcal{A}}|} \sum_{\tilde{A}_i \in \tilde{\mathcal{A}}} \tilde{D}_i^{-\frac{1}{2}} \tilde{A}_i \tilde{D}_i^{-\frac{1}{2}} H^{(l)} W_i^{(l)}\right) \tag{6.21}$$

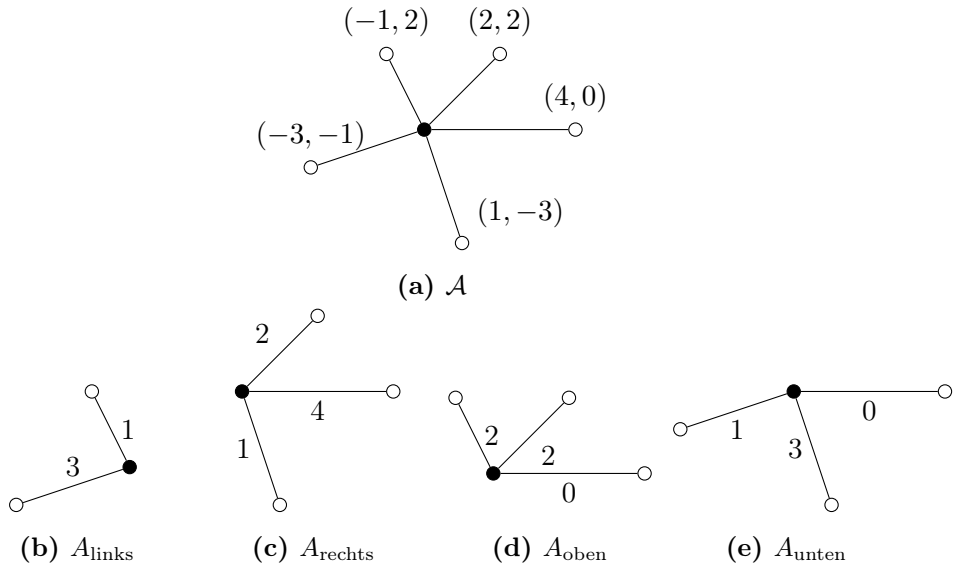
$\sigma(\cdot)$ kennzeichnet eine Aktivierungsfunktion wie zum Beispiel $\mathrm{ReLU}(\cdot) = \max(0, \cdot)$.

6.10.1. 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 euklische Distanz zwischen zwei benacharten 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_{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



(a) $\mathcal{A}$ **(b)** A_{links} **(c)** A_{rechts} **(d)** A_{oben} **(e)** A_{unten}

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

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

$$\tilde{A}_{ij} = \begin{cases} 1, & \text{falls } i = j, \\ (a_{ij} + 1)^{-1}, & \text{falls } a_{ij} \neq 0, \\ 0, & \text{sonst.} \end{cases} \tag{6.22}$$

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.

6.11. Pooling-Ebene

6.11.1. 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

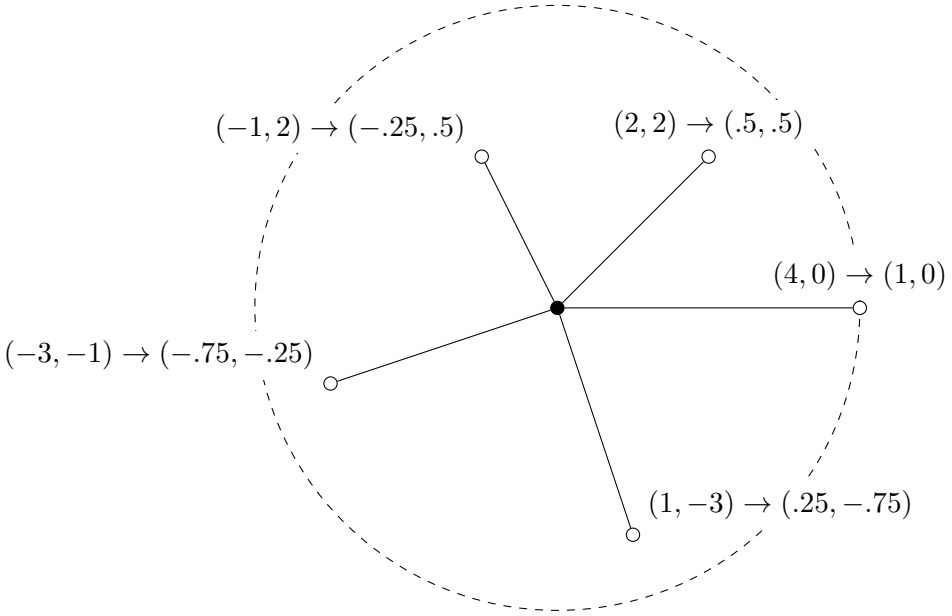


Abbildung 6.2.: Abbildung der lokalen Nachbarschaftsknoten auf den Einheitskreis.

- 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* [?].

h ich evtl
cht

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

Defferrard et al. [4] benutzen für die Pooling-Ebene eines Netzes auf Graphen die *Vergröberungsphase* des mehrstufigen Cluster-Algorithmus *Graclus* [3]. Dabei wird der initiale Graph G_0 sukzessive in kleinere Graphen $G_1, G_2, \ldots, G_m$ mit $|\mathcal{V}_0| > |\mathcal{V}_1| > \cdots > |\mathcal{V}_m|$ 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 initialisert 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

Graclus 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 Graclus nur sehr wenige Singleton-Knoten generiert [4].

Nach der spektralen Graphentheorie [2] gilt für Kanten eines Graphen $G = (\mathcal{V}, \mathcal{E})$ nach Verschmelzung von u und v zu v^*

$$w_{xv^*} = w_{xu} + w_{xv} \tag{6.23}$$

$$w_{v^*v^*} = w_{uu} + w_{vv} + 2w_{uv} \tag{6.24}$$

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 λ_G, λ_H jeweils die ersten Eigenvektoren λ_1 von G respektive H [2].

Übertragung auf planare Graphen

- Mittelwert der Positionen wird gebildet (auch gewichtet über d_i?)
- $\mathcal{E}_{v^*} = \mathcal{E}_u \cup \mathcal{E}_v$

nur die Kan
Gewichte w
neu berechn

6.11.2. Pooling-Operation

Anhand eines kleineren, vergröberten 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:** $v^* := \|u, v\|_2$

6.12. Beispiel

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

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{6.25}$$

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

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

$$\mathcal{L} = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix} \tag{6.26}$$

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

$$\mathcal{L} \cdot \vec{u} = \lambda \cdot \vec{u} \tag{6.27}$$

Wir erhalten 3 Eigenvektoren und Eigenwerte mit

$$\lambda_0 = 0, \vec{u}_0 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.58 \\ 0.58 \\ 0.58 \end{pmatrix}, \lambda_1 = 1, \vec{u}_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \approx \begin{pmatrix} -0.71 \\ 0 \\ 0.71 \end{pmatrix}, \lambda_2 = 3, \vec{u}_2 = \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.41 \\ -0.82 \\ 0.41 \end{pmatrix} \tag{6.28}$$

Dann sind U, Λ und U^T definiert als

$$U \approx \begin{pmatrix} 0.58 & -0.71 & 0.41 \\ 0.58 & 0 & -0.82 \\ 0.58 & 0.71 & 0.41 \end{pmatrix}, \Lambda = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix}, U^T \approx \begin{pmatrix} 0.58 & 0.58 & 0.58 \\ -0.71 & 0 & 0.71 \\ 0.41 & -0.82 & 0.41 \end{pmatrix} \tag{6.29}$$

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 $\tilde{\mathbf{\Lambda}}$ definiert als

$$\tilde{\mathbf{\Lambda}} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -\frac{1}{3} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.30}$$

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

$$g_\theta(\mathbf{\Lambda}) = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, g_{\theta prime} = \Big(wd\Big) \tag{6.31}$$

A. Weitere Informationen

Symbolverzeichnis

deg Gradfunktion der Knoten eines Graphen G mit $\deg\colon \mathcal{V} \to \mathbb{N}$. 4, 20

$\mathbf{L_{norm}}$ Laplacian, normalisiert. 7, 8

$\mathbb{N}$ Menge der natürlichen Zahlen. 4, 20

$\mathbb{R}_+$ Menge der positiven reellen Zahlen inklusive Null. 3, 4, 7, 14, 20

$\mathbb{R}$ Menge der reellen Zahlen. 3, 4, 7, 11, 13, 14, 20

$\mathcal{E}$ Kantenmenge eines Graphen G mit $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. 3, 20

$\mathcal{V}$ Knotenmenge $\{v_i\}_{i=1}^n$ eines Graphen G. 3, 4, 20

$\perp$ Orthogonalität. 7

$\sim$ Adjazenzrelation zweiter Knoten eines Graphen G mit $u \sim v$ genau dann, wenn u und v adjazent. 3, 4, 20

diag Diagonalfunktion. 4, 8

d gewichtete Gradfunktion der Knoten eines Graphen G mit $d\colon \mathcal{V} \to \mathbb{R}_+$. 4, 20

p Positionsfunktion auf den Knoten $\mathcal{V}$ mit $p\colon \mathcal{V} \to \mathbb{R}^2$. 4, 20

s kürzeste Distanzfunktion mit $s\colon \mathcal{V} \times \mathcal{V} \to \mathbb{N}$. 4, 20

w Gewichtsfunktion der Kanten eines Graph G mit $w\colon \mathcal{V} \times \mathcal{V} \to \mathbb{R}_+$. 3, 4, 8, 20

$\mathbf{A}$ Adjazentmatrix eines Graphen G. 4, 7

$\mathbf{D}$ gewichtete Gradmatrix. 4, 7

$\mathbf{I}$ Identitätsmatrix. 3, 7

$\mathbf{L}$ Laplacian, unnormalisiert. 7, 8

$\mathbf{\Lambda}$ Diagonalmatrix der Eigenwerte des Laplacian. 8

$\mathcal{L}$ Laplacian, normalisiert oder unnormalisiert. 7

G Graph. 3, 4, 7, 8, 20

Abbildungsverzeichnis

List of Algorithms

Literaturverzeichnis

[1] D. K. Hammond and P. Vandergheynst and R. Gribonval: *Wavelets on graphs via spectral graph theory.* Applied and Computational Harmonic Analysis, Seiten 129–150, 2011.

[2] F. R. K. Chung: *Spectral Graph Theory.* American Mathematical Society, 1997.

[3] I. S. Dhillon and Y. Guan and B. Kulis: *Weighted Graph Cuts Without Eigenvectors: A Multilvel Approach.* IEEE, Seiten 1944–1957, 2007.

[4] M. Defferrard and X. Bresson and P. Vandergheynst: *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering.* CoRR, 2016.

[5] T. N. Kipf and M. Welling: *Semi-Supervised Classification with Graph Convolutional Networks.* CoRR, 2016.

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