technische universität
dortmund

fakultät für informatik

Master-Thesis

Convolutional Neural Networks auf Graphrepräsentationen von Bildern

Matthias Fey

8. Mai 2017

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

Lehrstuhl Informatik VII
Graphische Systeme
TU Dortmund

fakultät für
informatik

Inhaltsverzeichnis

1. Einleitung

1.1. Motiviation

1.2. Aufbau der Arbeit

2. Grundlagen

2.1. Mathematische Notationen

Vektoren, Matrizen und Tensoren insbesondere dünnbesetzte Matrizen

2.2. Graphentheorie

2.3. Neuronale Netze

2.3.1. Convolutional Neural Networks

Was ist ein CNN? Wie ist der Faltungsoperator definiert? Was ist ein Filter? Was ist eine Filtergröße? Aktivierungsfunktionen? Dynamische Eingabegröße? (evtl. erst später)

3. Bildrepräsentationen durch Graphen

3.1. Eingebettete Graphen

3.1.1. Kantengewichte

3.2. Gitter

3.3. Superpixel

3.3.1. Verfahren

SLIC

Quickshift

Weitere Verfahren

3.3.2. Adjazenzmatrixbestimmung

3.3.3. Merkmalsextraktion

4. Stand der Forschung

5. Räumliches Lernen auf Graphen

5.1. Räumliche Graphentheorie

5.1.1. Färbung von Knoten

5.1.2. Isomorphie und kanonische Ordnung

5.2. Räumliche Faltung

5.2.1. Knotenauswahl

5.2.2. Nachbarschaftsgruppierung

5.2.3. Normalisierung

5.3. Erweiterung auf eingebettete Graphen

5.4. Netzarchitektur

6. Spektrales Lernen auf Graphen

6.1. Spektrale Graphentheorie

6.1.1. Eigenwerte und Eigenvektoren reell symmetrischer Matrizen

6.1.2. Laplace-Matrix

Visuelle Interpretation

Eigenschaften

6.2. Spektraler Faltungsoperator

6.2.1. Graph-Fourier-Transformation

6.2.2. Polynomielle Approximation

Tschebyschow-Polynome

6.3. Graph Convolutional Networks

6.3.1. Faltungsoperator

6.3.2. Weisfeiler-Lehman Analogie

6.3.3. Erweiterung auf eingebettete Graphen

B-Spline-Kurven

Faltungsoperator

6.4. Pooling auf Graphen

6.4.1. Clustering von Knoten

Normalized Cut

6.4.2. Vergröberung von Graphen

6.4.3. Erweiterung auf eingebettete Graphen

6.5. Netzarchitektur

7. Implementierung

7.1. Verwendete Technologien

7.2. Vorverarbeitungsschritt

8. Evaluation

8.1. Datensätze

8.1.1. MNIST

8.1.2. Cifar10

8.1.3. PascalVOC

8.2. Metriken

8.3. Parameterwahl

Vorstellung aller Parameter Superpixelalgorithmen Parameterwahl

8.4. Merkmalsselektion

8.5. Ergebnisse

8.5.1. MNIST

Gitter

10.000 Steps mit Bach Size 64
Learning-Rate: 0.001
Dropout: 0.5

Verfahren	Netzarchitektur	Dauer	Genauigkeit
klassisch	C5-32-P4-C5-64-P4-FC1024	0.18s	99.189
klassisch	C32-C32-P4-C64-C64-P4-FC1024	0.25s	99.139
Chebyshev	C25-32-P4-C25-64-P4-FC1024	0.91s	98.888
GCN	C32-C32-P4-C64-C64-P4-FC1024	0.22s	96.675
E-GCN	C32-C32-P4-C64-C64-P4-FC1024	0.77s	99.145

Superpixel

Verfahren	Netzarchitektur	Dauer	Genauigkeit
SLIC E-GCN	C32-P2-C64-P2-C128-P2-C256-P2-FC128	1.12s	96.124

8.5.2. Cifar10

8.5.3. PascalVOC

Learning-Rate: 0.1
Dropout: 0.5
nur 2000 Steps trainiert!

Verfahren	Netzarchitektur	Dauer	Genauigkeit
SLIC E-GCN	C32-C32-P2-C64-C64-P2-C128-C128-P2-C256-C256-P2-C512-C512-P2-FC256-FC128	29s	60.546

8.5.4. Vergleich mit anderen Implementierungen

8.6. Laufzeitanalyse

8.6.1. Vergleich mit anderen Implementierungen

8.7. Räumlicher Ansatz vs. spektraler Ansatz

Schwächen und Vorteile beider Ansätze

9. Zusammenfassung

10. Ausblick

10.1. Weitere Anwendungsgebiete

z.B. Segmentierung

10.2. Augmentierung von Graphen

10.3. Spatial-Pyramid-Pooling

10.4. Attention-Algorithmus

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 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) 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

11. Einleitung

11.1. Motivation

Generell Lernen auf Graphen, die in der Ebene oder im Raum eingebettet sind, z.B. Straßennetze, Karten.

11.2. Aufbau der Arbeit

12. Grundlagen

12.1. 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 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 $f\colon \mathbb{R} \to \mathbb{R}$ auch elementweise auf Vektoren, Matrizen bzw. Tensoren anwenden dürfen.

Identitätsmatrix $\mathbf{I}$

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

$$\langle \mathbf{x}, \mathbf{y}\rangle = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i \tag{12.1}$$

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

hadamard $\odot$

12.2. 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 $w\colon \mathcal{V} \times \mathcal{V} \to \mathbb{R}_+$
ungewichtet: $w\colon \mathcal{V} \times \mathcal{V} \to \{0, 1\}$
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

$$w(u, v) = w(v, u) \tag{12.2}$$

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.

$$\deg(v) = |\{u \colon (v, u) \in \mathcal{E}\}| \tag{12.3}$$

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

$$d(v_i) = \sum_{j=1}^{n} \mathbf{A}_{ij} \tag{12.4}$$

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

$$\mathbf{D} = \operatorname{diag}\Big([d(v_1), \ldots, d(v_n)]^\top\Big) \tag{12.5}$$

Umschreibung in Sparse Matrix/Tensor

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

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 $\big(v_{x(1)}, v_{x(2)}, \ldots, v_{x(k)}\big)$, sodass $v_{x(i)} \sim v_{x(i+1)}$ für alle $1 \leq i < k$ mit Länge k, wobei $x \colon \{1, \ldots, n\} \to \{1, \ldots, n\}$ 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 $s \colon \mathcal{V} \times \mathcal{V} \to \mathbb{N}$ 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 $[\mathbf{A}_1, \ldots, \mathbf{A}_m] \in {\mathbb{R}_+}^{m \times n \times n}$ schreiben. Graphen mit Mehrfachkanten können ebenso als eine Menge von Graphen mit gleicher Knotenmenge betrachtet werden.

s stimmt noch
ht ganz, ge-
n ja auch
ehrere Kno-
n

s sind graph
mponenten,
finieren

lierte knoten

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

12.3. 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:

$$\begin{pmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 1 \\ 7 & 8 & 9 & 1 \end{pmatrix} * \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 12 & 16 & 11 \\ 24 & 28 & 17 \end{pmatrix} \tag{12.6}$$

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

$$(f * g)(x, y) = \sum_{x_i \in [x, x+1] y_i \in [y, y+1]} f(x_i, y_i) g(x - x_i, y - y_i) \tag{12.7}$$

13. Graphrepräsentationen von Bildern

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

13.1. Kantengewichte

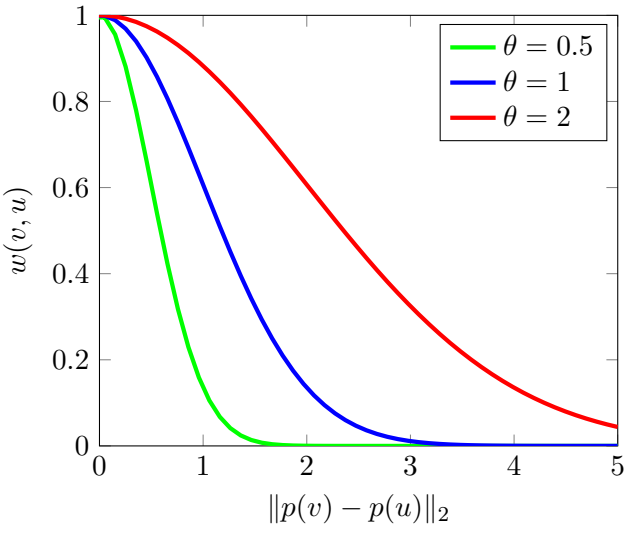
Kantengewichte werden ermittelt aus der euklidischen Distanz der Zentren zweier adjazenter Regionen $\|p(v) - p(u)\|_2$. Distanz entspricht aber nicht der üblichen Bedeutung von Kantengewichten auf Graphen. Je höher das Gewicht, desto ähnlicher bzw. näher sind zwei Knoten.

Ein üblicher Weg die Distanz zweier Knoten zueinander als Kantengewicht darzustellen ist über einen gewichteten *Gaussian-Kernel* [1]

$$w(v,u) = \exp\left(-\frac{\|p(v) - p(u)\|_2^2}{2\theta^2}\right) \tag{13.1}$$

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

Die Wahl von θ ist dabei abhängig von der Ausdehnung der Distanzen eines Graphen.



d graphen

13.2. Effiziente Adjazenzgenerierung

Segmentierungsmaske $S \in \mathbb{N}^{N \times M}$ mit Höhe N und Breite M.

1. schneide oben und unten Pixellinie ab: S_{y_1} und S_{y_2}
2. schneide links und rechts Pixellinie ab: S_{x_1} und S_{x_2}

weitermachen

14. Räumliche Graphentheorie

Isomorphismus, Automorphismus, Canonical Labeling
Labeling / Node Partitions

14.1. Patchy-SAN

15. Spektrale Graphentheorie

15.1. 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.

15.2. Probleme

Rotationsinvariant

15.3. Eigenwerte und Eigenvektoren reell symmetrischer Matrizen

$\mathbf{M}\mathbf{u} = \lambda\mathbf{u}$
Zu einem Eigenwert λ 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 $\mathbf{\Lambda} = \operatorname{diag}([\lambda_1, \ldots, \lambda_n])$.

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

$$\mathbf{M} = \mathbf{MUU}^\top = \mathbf{U\Lambda U}^\top \tag{15.1}$$

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

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

$$\mathbf{M}^k = \left(\mathbf{U\Lambda U}^\top\right)^k = \mathbf{U\Lambda}^k\mathbf{U}^\top \tag{15.2}$$

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

$$\left(\mathbf{U\Lambda U}^\top\right)^k = \mathbf{U\Lambda U}^\top\mathbf{U\Lambda U}^\top \prod_{i=1}^{k-2} \mathbf{U\Lambda U}^\top = \mathbf{U\Lambda}^2\mathbf{U}^\top \prod_{i=1}^{k-2} \mathbf{U\Lambda U}^\top = \mathbf{U\Lambda}^k\mathbf{U}^\top \tag{15.3}$$

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

$$\sum_{j=1}^{n} |\mathbf{M}_{ij}| \leq |\mathbf{M}|_{ii} \tag{15.4}$$

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

$$\mathbf{x}^\top \mathbf{M} \mathbf{x} \geq 0 \tag{15.5}$$

15.4. Der Laplacian und seine Eigenwerte

Der *kombinatorische Laplacian* $\mathbf{L}$ eines Graphen G ist definiert als $\mathbf{L} = \mathbf{D} - \mathbf{A}$ [3]. $\mathbf{L}$ ist eine symmetrische Matrix.

Der *normalisierte Laplacian* $\tilde{\mathbf{L}}$ ist definiert als $\tilde{\mathbf{L}} = \mathbf{D}^{-\frac{1}{2}} \mathbf{L} \mathbf{D}^{-\frac{1}{2}}$ [3]. Es gilt die Konvention, dass $\left(\mathbf{D}^{-\frac{1}{2}}\right)_{ii} = 0$ falls $\mathbf{D}_{ii} = 0$ (d.h. v_i ist isolierter Knoten)

Für verbundene Graphen gilt damit $\tilde{\mathbf{L}} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}} \mathbf{A} \mathbf{D}^{-\frac{1}{2}}$ [3]. 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. [2].

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

15.4.1. Visuelle Interpretation des Laplacian

- diskrete Analogie des ∇^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 euklischen Raum

$$\nabla^2 f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_k^2} \tag{15.6}$$

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

Laplace-Operator

Sei $f\colon \mathcal{V} \to \mathbb{R}$ 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.

diskreter Laplace-Operator

Dann gilt für $\mathcal{L}$, dass

$$(\mathcal{L}\mathbf{f})_i = \sum_{\substack{j=0 \\ j \neq i}}^{n} -\mathcal{L}_{ij}(\mathbf{f}_i - \mathbf{f}_j) \tag{15.7}$$

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

$$(\mathbf{L}\mathbf{f})_{x,y} = \frac{4\mathbf{f}_{x,y} - \mathbf{f}_{x+1,y} - \mathbf{f}_{x-1,y} - \mathbf{f}_{x,y+1} - \mathbf{f}_{x,y-1}}{h^2} \tag{15.8}$$

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

$$\mathcal{L}f \approx -\nabla^2 f \tag{15.9}$$

vertauschen, e
beispiel auf git
ter, dann anal
zu graphen git
ter

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

$$\mathbf{x} = \sum_i c_i \mathbf{u}_i \tag{15.10}$$

und berechnen dann

$$\mathcal{L}^k \mathbf{x} = \sum_i c_i \mathcal{L}^k \mathbf{u}_i = \sum_i = c_i \lambda_i \mathcal{L}^{k-1} \mathbf{u}_i = \sum_i c_i \lambda_i^k \mathbf{u}_i \tag{15.11}$$

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.

15.4.2. Eigenschaften

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

nur kombinato
risch ist schwa
diagonaldomi-
nant

$\mathcal{L} \in \mathbb{R}^{n \times n}$ 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 $i \in \{1, \ldots, n\}$.Insbesondere gilt $\lambda_1 = 0$, da $\mathbf{u}_1 = \frac{1}{\sqrt{n}}[1, \ldots, 1]^\top \in \mathbb{R}^n$ Eigenvektor von $\mathcal{L}$ mit $\mathcal{L}\mathbf{u}_1 = 0$.

t nur für kom- natorischen!

elle

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

s sagt λ_2 aus?

elle

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

Für $\mathcal{L}^k$ mit $k \in \mathbb{N}$ gilt $\left(\mathcal{L}^k\right)_{ij} = 0$ genau dann, wenn $s(v_i, v_j) > k$ [2]. Damit beschreibt $\mathcal{L}^k$ bildlich gesprochen die Menge an Knoten, die maximal k Kanten entfernt liegen.

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

$$w(x, v^*) = w(x, u) + w(x, v) \tag{15.12}$$

$$w(v^*, v^*) = w(u, u) + w(v, v) + 2w(u, v) \tag{15.13}$$

Für einen Graphen G gilt für einen Graphen H, der aus G verkleinert wurde [3],

$$\lambda_G \leq \lambda_H \tag{15.14}$$

15.5. Faltung mittels Graph-Fourier-Transformation

15.5.1. Kontinuierliche Fourier-Transformation

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

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

s verstehe ich cht

Die inverse Fourier-Transformation

$$f(t) = \frac{1}{2\pi} \int \hat{f}(\omega) \exp(\mathrm{i}\omega t)\, \mathrm{d}\omega \tag{15.15}$$

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

15.5.2. Graph-Fourier-Transformation

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

Eigenwerte beschreiben die Frequenzen des Graphen.

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

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

on weiter en definiert, er in grundla- n unterbrin- n?

$$\hat{f}(\lambda_i) = \langle \mathbf{f}, \mathbf{u}_i \rangle \tag{15.16}$$

bzw. als Vektor

$$\hat{\mathbf{f}} = \mathbf{U}^\top \mathbf{f} \tag{15.17}$$

Die inverse Transformation ist dann definiert als

$$f(v_i) = \sum_{j=1}^{n} \hat{f}(\lambda_j)(\mathbf{u}_j)_i \tag{15.18}$$

bzw. als Vektor

$$\mathbf{f} = \mathbf{U}\hat{\mathbf{f}} \tag{15.19}$$

parseval relation?

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. quelle

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 beschreiben die Signale zu den gegebenen Frequenzen. mmh

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 ω close to zero (low frequencies), the associated complex exponential eigenfunctions are smooth, slowly oscillating functions, whereas for ω 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 u0 associated with the eigenvalue 0 is constant and equal to 1 at each vertex. The graph Laplacian eigenvectors associated with low frequencies λ_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.

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

15.5.3. 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

$$\hat{f}_{\text{out}}(\omega) = \hat{f}_{\text{in}}(\omega)\hat{g}(\omega) \tag{15.20}$$

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

$$f_{\text{out}}(t) = (f_{\text{in}} \star g)(t) \tag{15.21}$$

Wir können das spektrale Graphfilterung analog definieren mit

$$\hat{f}_{\text{out}}(\lambda_i) = \hat{f}_{\text{in}}(\lambda_i)\hat{g}(\lambda_i) \tag{15.22}$$

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

$$\mathbf{f}_{\text{out}} = \mathbf{f}_{\text{in}} \star \mathbf{g} = \mathbf{U}\Big(\mathbf{U}^\top \mathbf{f}_{\text{in}} \odot \mathbf{U}^\top \mathbf{g}\Big) \tag{15.23}$$

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

$$\hat{\mathbf{f}}_{\text{out}} = \hat{g}(\mathbf{\Lambda})\hat{\mathbf{f}}_{\text{in}} = \hat{g}(\mathbf{\Lambda})\mathbf{U}^\top \mathbf{f}_{\text{in}} \tag{15.24}$$

bzw.

$$\mathbf{f}_{\text{out}} = \mathbf{U}\hat{g}(\mathbf{\Lambda})\mathbf{U}^\top \mathbf{f}_{\text{in}} \tag{15.25}$$

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

15.6. Polynomielle Approximation

was bedeutet das?

- not localized in Space
-

was bedeutet k-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.

$$\hat{g}'(\lambda_i) = \sum_{j=0}^{k} c_j \lambda_i^j \tag{15.26}$$

bzw.

$$\hat{g}'(\mathbf{\Lambda}) = \sum_{j=0}^{k} c_j \mathbf{\Lambda}^j \tag{15.27}$$

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:

$$\hat{g}'(\mathcal{L}) =: \mathbf{U}\hat{g}'(\mathbf{\Lambda})\mathbf{U}^\top = \mathbf{U}\sum_{j=0}^{k} c_j \mathbf{\Lambda}^j \mathbf{U}^\top = \sum_{j=0}^{k} c_j \mathbf{U}\mathbf{\Lambda}^j\mathbf{U}^\top = \sum_{j=0}^{k} c_j \mathcal{L}^j \tag{15.28}$$

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

bereits weiter oben definiert

Dann ist die Faltung definiert als

$$\mathbf{f}_{\text{out}} = \hat{g}'(\mathcal{L})\mathbf{f}_{\text{in}} \tag{15.29}$$

bzw.

$$f_{\text{out}}(v_i) = \sum_{j=0} f_{\text{in}}(v_j)\left(\hat{g}'(\mathcal{L})\right)_{ij} \tag{15.30}$$

Wir erinnern uns, dass $\left(\mathbf{L}^l\right)_{ij} = 0$, 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.

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

definieren

$$f_{\text{out}}(v_i) = f_{\text{in}}(v_i)\left(\hat{g}'(\mathcal{L})\right)_{ii} + \sum_{v_j \in \mathcal{N}(v_i,k)} f_{\text{in}}(v_j)\left(\hat{g}'(\mathcal{L})\right)_{ij} \tag{15.31}$$

15.6.1. 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 $T_n(x)\colon \mathbb{R} \to \mathbb{R}$ mit dem rekursiven Zusammenhang

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x) \tag{15.32}$$

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

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

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

$$\hat{g}'(\mathbf{\Lambda}) = \sum_{i=0}^{k} c_i \mathbf{\Lambda}^i = \sum_{i=0}^{k} c'_i T_i\left(\tilde{\mathbf{\Lambda}}\right) \tag{15.33}$$

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

$$\hat{g}'(\mathcal{L}) = \sum_{i=0}^{k} c'_i T_i\left(\tilde{\mathcal{L}}\right) \tag{15.34}$$

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

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

1. berechnne $\bar{\mathbf{f}}_i := T_i\left(\tilde{\mathcal{L}}\right)\mathbf{f}_{\text{in}}$ für alle $i \in \{0, 1, \ldots, k\}$ 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 $\mathbf{f}_{\text{out}} = \left[\bar{\mathbf{f}}_0, \bar{\mathbf{f}}_1, \ldots, \bar{\mathbf{f}}_k\right]\mathbf{c}'$, wobei $\mathbf{c}' = [c'_0, \ldots, c'_k] \in \mathbb{R}^{k+1}$

Laufzeit

- anstatt $\mathcal{L}^k$ zu berechnen mit Komplexität $\mathcal{O}\left(n^2\right)$ 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 $\mathcal{O}(k|\mathcal{E}|)$ [2, 5]
- für den zweiten Schritt gilt $\mathcal{O}(kn)$, mit $n \leq |\mathcal{E}|$ bedingt damit nur Schritt Eins die Laufzeit

15.7. Graph Convolutional Networks

- aus [7]
- 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 3×3 keine negativen Auswirkungen haben
- viele kleine Faltungsebenen ohne Pooling propagieren die Informationen weit entfernterer Knoten weiter
- kann das Problem des Overfitting reduzieren für Graphen mit hohem Grad

VGG Paper, oder he et al, deep residual learning

Annahme: $\lambda_{\max} \approx 2$ Es gilt $0 \leq \lambda_0 \leq \cdots \leq \lambda n - 1 \leq 2$ [3] (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 begründet, woher kommt diese Zahl

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

$$g_\theta \star \mathbf{x} = \mathbf{U} g_\theta(\mathbf{\Lambda}) \mathbf{U}^\top \mathbf{x} \approx g'_{\theta'}(\mathbf{L})\mathbf{x} = \sum_{i=0}^{k-1} \theta'_i T_i(\mathbf{L} - \mathbf{I})\mathbf{x} = \theta'_0 \mathbf{x} + \theta'_1 (\mathbf{L} - \mathbf{I})\mathbf{x} \tag{15.35}$$

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

$$g_\theta \star \mathbf{x} \approx \theta'_0 \mathbf{x} - \theta'_1 \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{x} \tag{15.36}$$

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

$$g_\theta \star \mathbf{x} \approx \theta\left(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\right)\mathbf{x} \tag{15.37}$$

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

warum?

warum

$$g_\theta \star \mathbf{x} \approx \theta\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\right)\mathbf{x} \tag{15.38}$$

$$H^{(l+1)} = f(H^{(l)}, A) \tag{15.39}$$

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)}) \tag{15.40}$$

$$D_{ii} = \sum_j A_{ij} \tag{15.41}$$

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

$$D^x = \begin{pmatrix} d_{11} & 0 & \cdots & 0 \\ 0 & d_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{nn} \end{pmatrix}^x = \begin{pmatrix} d_{11}^x & 0 & \cdots & 0 \\ 0 & d_{22}^x & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_{nn}^x \end{pmatrix} \tag{15.42}$$

15.8. Weisfeiler Lehman Analogie

15.9. 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 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.

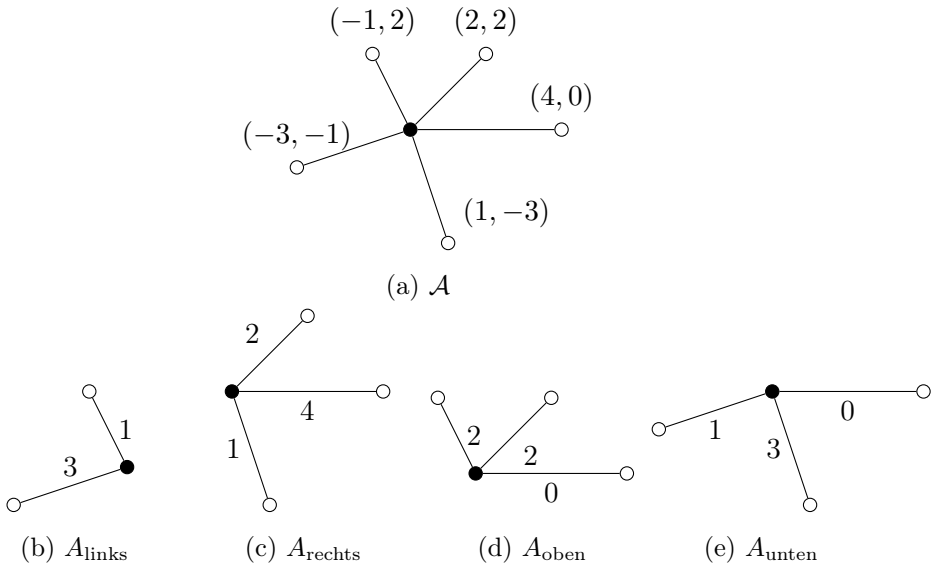
$\mathcal{A} \in \mathbb{R}^{m \times n \times n}$ kann zu einer zweidimensionalen Matrix $A \in \mathbb{R}^{m \cdot n \times n}$ geglättet werden. Dann ist $A \cdot H^{(l)} \in \mathbb{R}^{m \cdot n \times d}$. Reshape zu $\mathbb{R}^{n \times m \cdot d}$ und Gewichtsmatrix $G \in \mathbb{R}^{m \cdot d \times x}$.

$$H^{(l+1)} = f(H^{(l)}, \tilde{\mathcal{A}}) = \sigma\left(\frac{1}{|\tilde{\mathcal{A}}|} \sum_{\tilde{A}_i \in \tilde{\mathcal{A}}} \tilde{D}_i^{-\frac{1}{2}} \tilde{A}_i \tilde{D}_i^{-\frac{1}{2}} H^{(l)} W_i^{(l)}\right) \tag{15.43}$$

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

15.9.1. 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 gehen. Anstatt diese lokal im Knoten zu speichern, bietet es sich eher



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

Abbildung 15.1.: Aufteilung einer Adjazenzmatrix in vier räumlich eingebettete Bereiche.

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 15.1). 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 15.2).

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

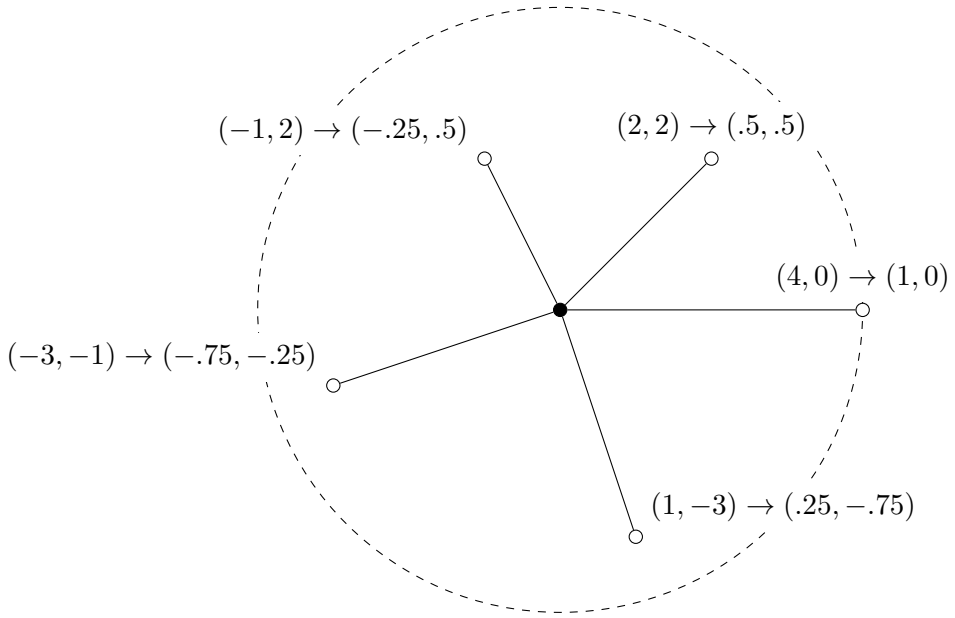


Abbildung 15.2.: Abbildung der lokalen Nachbarschaftsknoten auf den Einheitskreis.

$$\tilde{A}_{ij} = \begin{cases} 1, & \text{falls } i = j, \\ (a_{ij} + 1)^{-1}, & \text{falls } a_{ij} \neq 0, \\ 0, & \text{sonst.} \end{cases} \tag{15.44}$$

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.

15.10. Pooling-Ebene

eragePooling

15.10.1. 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* [**Luxburg**]. brauch ich evt garnicht

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

Defferrard et al. [5] benutzen für die Pooling-Ebene eines Netzes auf Graphen die *Vergröberungsphase* des mehrstufigen Cluster-Algorithmus *Graclus* [4]. 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 [4].

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 [5].

Nach der spektralen Graphentheorie [3] gilt für Kanten eines Graphen $G = (\mathcal{V}, \mathcal{E})$ nach Verschmelzung von u und v zu v^*

$$w_{xv^*} = w_{xu} + w_{xv} \tag{15.45}$$

$$w_{v^*v^*} = w_{uu} + w_{vv} + 2w_{uv} \tag{15.46}$$

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 λ_G, λ_H jeweils die ersten Eigenvektoren λ_1 von G respektive H [3].

Übertragung auf planare Graphen

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

r die Kanten, wichte werden u berechnet!

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

15.10.2. 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:** $v^* := \|u, v\|_2$
- **Average-Pooling:**

das nicht das iche wie L2?

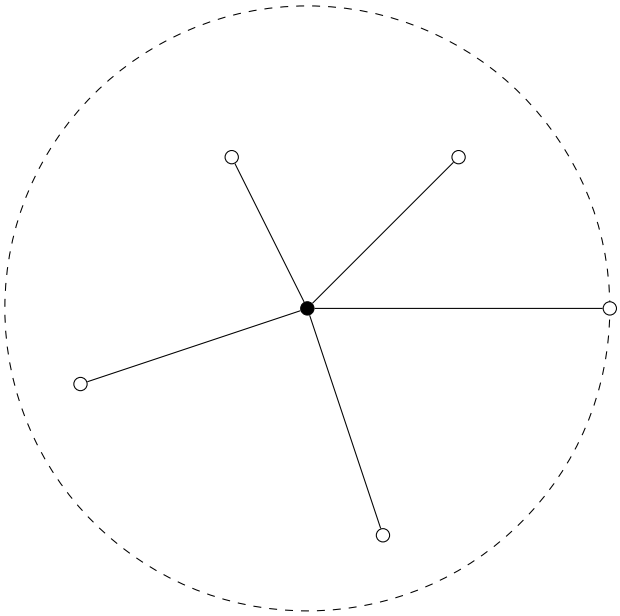
15.10.3. Informationen

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 durch 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 sehr gut.

fekt geht rch das Ave- gePooling eh rloren und nst weiß ich cht wie gut s überhaupt (siehe Email)

15.11. Partitionierung



15.11.1. Grundlagen

- ungerichtete Distanzadjazenzmatrix $\mathbf{A}_{\text{dist}} \in {\mathbb{R}_+}^{N \times N} = (a)_{ij}$
- (lokal) normalisierte ungerichtete Distanzadjazenzmatrix $\tilde{\mathbf{A}}_{\text{dist}} \in {\mathbb{R}_+}^{N \times N} = (\tilde{a})_{ij}$
- gerichtete Winkeladjazenzmatrix $\mathbf{A}_{\text{rad}} \in {\mathbb{R}_+}^{N \times N} = (\alpha)_{ij}$
- Input-Featurematrix $\mathbf{F} \in \mathbb{R}^{N \times X} = (f)_{ij}$
- Output-Featurematrix $\mathbf{F'} \in \mathbb{R}^{N \times Y} = (f')_{ij}$
- Anzahl Partitionen $P \in \mathbb{N}$
- Gewichtstensor $\mathbf{W} \in \mathbb{R}^{(P+1) \times X \times Y} = (w)_{ijw}$

15.11.2. Faltung

$$f'_{iy} = \sum_{x=1}^{X} \tilde{a}_{ii} \cdot f_{ix} \cdot w_{(P+1)xy} \sum_{n=1, n \neq i}^{N} \tilde{a}_{in} \cdot f_{nx} \cdot b_P^K(\alpha_{in}, x, y) \tag{15.47}$$

wobei b_P^K eine B-Spline-Kurve.

a_{ii} kann raus, immer 1

Es ist anzumerken, dass im Summanden der betrachtete Knoten übersprungen wird, da für diesen ein Wert in der Winkelmatrix keinen Sinn ergibt. Er wird daher in der Faltung jeweils einzeln mit einem Gewicht multipliziert und dazuaddiert.

15.11.3. B-Spline-Kurven

$b_P^K :]0, 2\pi] \times \{1, \ldots, X\} \times \{1, \ldots, Y\} \to \mathbb{R}$ ist eine B-Spline-Kurve der Ordnung $K \in \mathbb{N}$ auf den Kantenwinkeln des Graphen. Bemerke, dass wir 0 für Winkel aussschließen und stattdessen den Winkel 2π benutzen, so dass wir nicht mit der Bedeutung von 0 bei Adjazenzmatrizen in die Quere kommen.

$$b_P^K(\alpha, x, y) = \sum_{p=1}^{P} w_{pxy} \cdot e_p^K(\alpha) \tag{15.48}$$

wobei die Basisfunktion e_p^K rekursiv über K definiert ist mit Initialisierung

$$e_p^1(\alpha) = \begin{cases} 1, & \text{wenn } \alpha \in]t(p-1), t(p)], \\ 0, & \text{sonst} \end{cases} \tag{15.49}$$

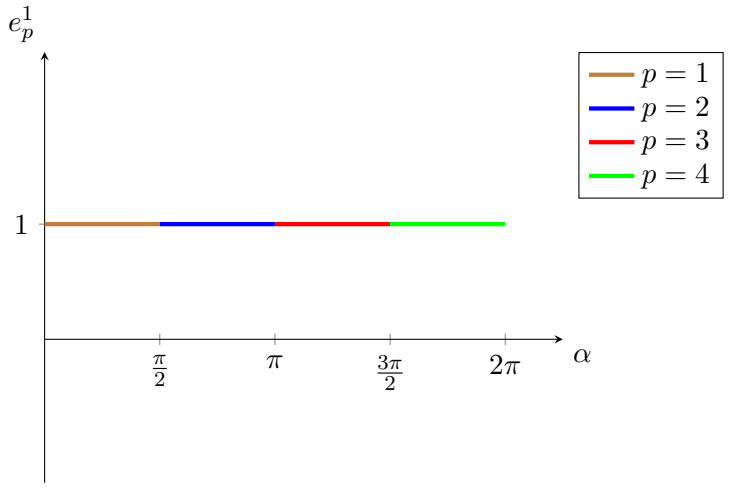
und Rekursionsschritt

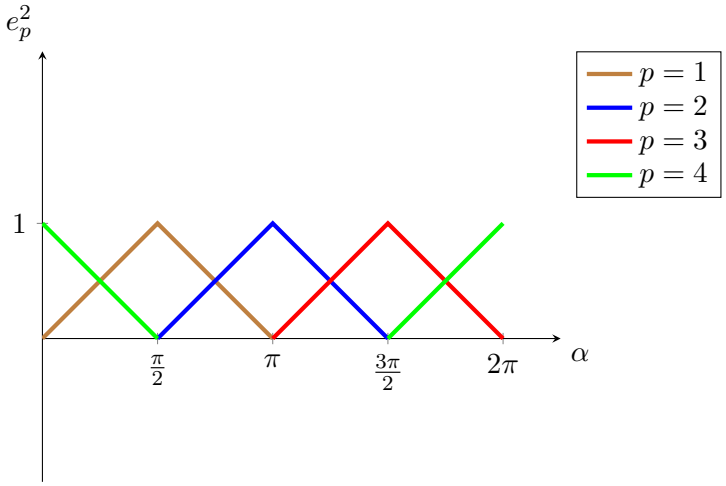
$$e_p^k(\alpha) = \frac{\alpha - t(p-1)}{t(p+k-2) - t(p-1)} e_p^{k-1}(\alpha) + \frac{t(i(p+k-1)) - \alpha}{t(p+k-1) - t(p)} e_{i(p+1)}^{k-1}(\alpha) \tag{15.50}$$

wobei $t \colon \mathbb{N} \to \mathbb{R}$ mit $t(p) = 2\pi \frac{p}{P}$ und $i(p) = \mathrm{mod}(p-1, P) + 1$. Es ist anzumerken, dass wir t und i dabei für den Rekursionsschritt über die Grenze P hinaus definieren. Das hilft uns, die B-Spline-Kurve *kreisförmig* abzuschließen.

Je größer K gesetzt wird, umso mehr Anteile anderer benachbarter Stützpunkte fließen in die Berechnung mit ein. Die Größe von K wird deshalb auch oft *lokale Kontrollierbarkeit* genannt.

Beispiel mit $P = 4$





Effiziente Berechnung für $K = 1$

Eigentlich ist nicht viel zu tun. Wir suchen lediglich eine effiziente Implementierung für $e_p^1(\alpha)$, die auch von TensorFlow verstanden wird und ableitbar ist. Es zeigt sich, dass $e_p^1(\alpha) = 1$ genau dann, wenn $\alpha > 2\pi\frac{p-1}{P}$ und $\alpha \leq 2\pi\frac{p}{P}$. Vereinfact damit

$$0 < \frac{P}{2\pi}\alpha + p + 1 \leq 1. \tag{15.51}$$

Es bleibt die Ungleichungsüberprüfung.

Effiziente Berechnung für $K = 2$

Wir können für $K = 2$ die Basis-Berechnung durch

$$e_p^2(\alpha) = \begin{cases} \max\left(\frac{P}{2\pi}\alpha - p + 1, 0\right), & \text{wenn } \alpha \leq t(p), \\ \max\left(-\frac{P}{2\pi}\alpha + p + 1, 0\right), & \text{sonst} \end{cases} \tag{15.52}$$

vereinfachen.

Beweis. Aufsteigende Gerade der Dreiecksfunktion für p, $1 \leq p \leq P$, ist definiert durch

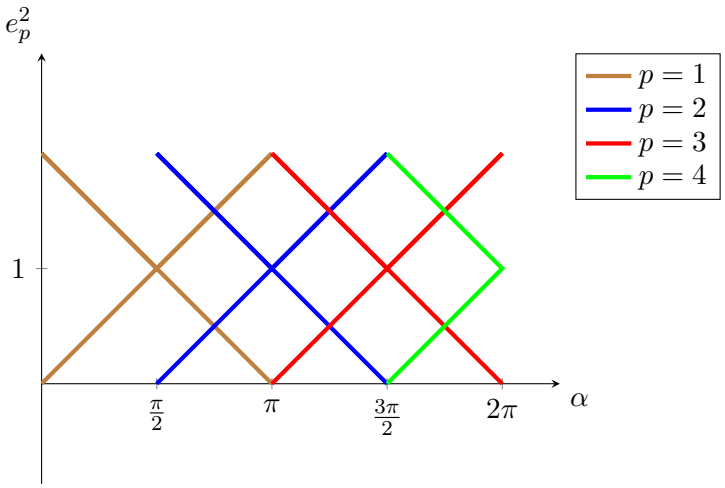
$$\frac{\alpha - t(p-1)}{t(p) - t(p-1)} = \frac{P(\alpha - t(p-1))}{2\pi} = \frac{P\alpha - 2\pi(p-1)}{2\pi} = \frac{P}{2\pi}\alpha - p + 1. \tag{15.53}$$

$\max\left(\frac{P}{2\pi}\alpha - p + 1, 0\right)$ beschreibt damit die linke Seite der Dreiecksfunktion. Analog für absteigende Gerade. □

Die Fallunterscheidung ist unnötig, wir können uns einfach immer für das Minimum der beiden entscheiden. Die Grafik zeigt dies ziemlich eindeutig. Das ergibt letztendlich

$$e_p^2(\alpha) = \min\left(\max\left(\frac{P}{2\pi}\alpha - p + 1, 0\right), \max\left(-\frac{P}{2\pi}\alpha + p + 1, 0\right)\right). \tag{15.54}$$

Wir haben bisher noch nicht den *Kreis* geschlossen mit unserer Formel.



Das können wir aber leicht tun, indem wir unsere absteigenden Geraden um 2π nach links verschieben und daraus wiederum das Maximum von 0 ziehen. Dann sind diese Geraden außer für $p = P$ im Gültigkeitsbereich der Funktion allesamt 0. Wir können demnach aus unserer bisherigen Formel und der verschobenen Gerade das Maximum ziehen. Wir erhalten

$$e_p^2(\alpha) = \max\Bigg(\min\Bigg(\max\Bigg(\frac{P}{2\pi}\alpha - p + 1, 0 \Bigg), \max\Bigg(-\frac{P}{2\pi}\alpha + p + 1, 0 \Bigg) \Bigg), \max\Bigg(-\frac{P}{2\pi}(\alpha + 2\pi) + p + 1, 0 \Bigg) \Bigg). \tag{15.55}$$

15.11.4. Tensorimplementierung

$$\mathbf{F}' = \left(\tilde{\mathbf{A}}_{\text{dist}} \right)_{ii} \cdot \mathbf{F} \cdot \mathbf{W}_{P+1} + \sum_{p=1}^{P} \tilde{\mathbf{A}}_{\text{dist}} \odot e_p^K(\mathbf{A}_{\text{rad}}) \cdot \mathbf{F} \cdot \mathbf{W}_p \tag{15.56}$$

mit $e_p^K(0) = 0$. Elementweise Multiplikation mit dünnbesetzten Matrizen `sparse_multiply` ist in TensorFlow nicht implementiert. Es kann aber intern auf den Daten von dünnbesetzten Matrizen eine elementweise Multiplikation angewendet werden. So kann die Faltung ohne Preprocessing implementiert werden.

oll bei 0 an-
ngen

kann raus,
immer 1

15.12. Diskreter geometrischer Kotangens-Laplacian

$$w(v_i, v_j) = \frac{\cot(\alpha(v_i, v_j)) + \cot(\beta(v_i, v_j))}{2} \tag{15.57}$$

und Grad eines Knoten ist das *Voronoi-Gebiet*

$$d(v_i) = a(v_i) \tag{15.58}$$

also die Fläche des Voronoi-Gebietes um einen Knoten [6].

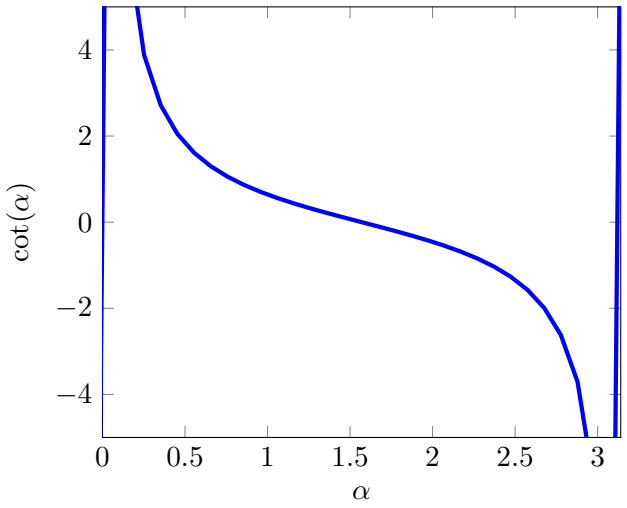
Winkel α und β beschreiben die Winkel, deren Kantenenden von v_i und v_j aufgespannt werden. Damit ist $\mathbf{L}$ symmetrisch, denn in der Rückrichtung vertauschen sich nur die Werte von α und β. Da α und β stets im Interval $(0, \pi)$ ist der Kotangenz eindeutig definiert. Der Winkel befindet sich aufjedenfall in diesem Intervall!

Diese Matrix ist aber weiterhin rotationsinvariant, so dass sie eigentlich uns keinen Nutzen bringt.

A beschreibt die Lage der Knoten zu ihren benachbarten Knoten? Das Voronoi-Gebiet gibt an, wie viele Knoten in näherer Umgebung vorhanden sind. Je größer das Gebiet, umso isolierter ist der Knoten. Beinhaltet also auch eine Art Abstand (aber zu allen Knoten, nicht nur zu einem).

Grafik

15.12.1. Intuition



Angenommen wir haben eine Kante und zwei Winkel mit $\alpha = \beta = \frac{\pi}{2}$. Dann ist $\cot\left(\frac{\pi}{2}\right) = 0$ und wir haben $w(v_i, v_j) = 0$.

DAS IST EXTREM KACK

Wenn unsere Winkel sehr spitz sind, dann ist das Gewicht der Kante sehr hoch Wenn unsere Winkel sehr stumpf sind, dann ist das Gewicht der Kante niedrig und ggf. negativ.

ES GIBT AUC NEGATIVE GEWICHTE????

15.13. Beispiel

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

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{15.59}$$

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

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

$$\mathcal{L} = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix} \tag{15.60}$$

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

$$\mathcal{L} \cdot \vec{u} = \lambda \cdot \vec{u} \tag{15.61}$$

Wir erhalten 3 Eigenvektoren und Eigenwerte mit

$$\lambda_0 = 0, \vec{u}_0 = \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.58 \\ 0.58 \\ 0.58 \end{pmatrix}, \lambda_1 = 1, \vec{u}_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \approx \begin{pmatrix} -0.71 \\ 0 \\ 0.71 \end{pmatrix}, \lambda_2 = 3, \vec{u}_2 = \frac{1}{\sqrt{6}} \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.41 \\ -0.82 \\ 0.41 \end{pmatrix} \tag{15.62}$$

Dann sind U, Λ und U^T definiert als

$$U \approx \begin{pmatrix} 0.58 & -0.71 & 0.41 \\ 0.58 & 0 & -0.82 \\ 0.58 & 0.71 & 0.41 \end{pmatrix}, \Lambda = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix}, U^T \approx \begin{pmatrix} 0.58 & 0.58 & 0.58 \\ -0.71 & 0 & 0.71 \\ 0.41 & -0.82 & 0.41 \end{pmatrix} \tag{15.63}$$

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

$$\tilde{\mathbf{\Lambda}} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -\frac{1}{3} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{15.64}$$

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

$$g_\theta(\mathbf{\Lambda}) = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, g_{\theta prime} = \Big(wd\Big) \tag{15.65}$$

16. Implementierung

Hier ein paar Gedanken zu:

- Sparse-Tensoren
- dynamischem Input
- Daten I/O (wir betrachten dynamische Adjazenzmatrizen, die nicht so einfach gebatcht werden können)
- Implementierung der Convolutions
- Verweis auf Datensätze mit cite auf MNIST, Cifar10, PascalVOC
- Verweis/Cite auf TensorFlow
- Distortions
- Preprocessing vs. Postprocessing

16.1. Multilabel

PascalVOC hat mehrere Label pro Bild. Multilabel-Problem über Sigmoid Loss Function, d.h. jedem möglichen Label wird ein Wert zwischen 0 und 1 zugeordnet. Dann kann über einen Threshold, der vorher festgelegt wird oder eindeutig über den Testdatensatz ermittelt wird, passieren.

17. Auswertung

Netzstrukturen vorstellen

Accuracy/Loss der unterschiedlichen Tests gegenüber klassischen CNNs der gleichen Struktur

17.1. MNIST

Trainingsbilder: 55.000

- 10.000 Steps mit Batch Size 64 (ungefähr 12 Epochen)
- Learning Rate 0.001
- klassisches Convolution Neural Network nachgebildet mit Gridgraphen
- Conv1: 5×5, $1 \to 32$
- MaxPool1: Size 2, Stride 2
- Conv2: 5×5, $32 \to 64$
- MaxPool2: Size 2, Stride 2
- FC1: 1024
- Dropout: 0.5
- FC2: 10

17.1.1. Auswertung

- **2D Conv > Max:** 0.18s pro Batch, Accuracy: 99.189, Cost: 0.03458
- **2D Conv > 2D Conv > Max:** 0.25s pro Batch, Accuracy: 99.139, Cost: 0.03062
- **Chebyshev $k = 25$ GCNN:** 0.91s pro Batch, Accuracy: 98.888, Cost: 0.04329
- $k = 1$ **GCNN:** 0.22s pro Batch, Accuracy: 96.765, Cost: 0.10596
- **Partitioned GCNN:**
  - Conv > Max: 0.45s pro Batch, Accuracy: 98.998, Cost: 0.03198
  - Conv > Conv > Max: 2.87s pro Batch, Accuracy: 99.189, Cost: 0.02704

17.1.2. SLIC

- keine lokale Normierung
- Stddev: 1
- 4 Level
- Graphkonnektivität: 1
- Anzahl Segmente: 100
- Compactness: 10
- Maximum Iterations: 10
- Sigma: 0
- Anzahl Partitionen: 8
- Features: Area, Bbox height, bbox width, Mean Color = 4 Features
- **Aufbau**: Conv zu 32, Pool2, Conv zu 64, Pool2, Conv zu 128, Pool2, Conv zu 256, Pool2, AveragePool, FC210
- Meiste zeit wird durch Partitionierung verschwendet.
- **Ergebnisse**: 0.79s pro Batch, Accuracy: 0,79497, Loss: 0.62814
- enttäuschend!

17.2. PascalVOC

erster Test: 17 s Preprocess, 12s Training auf BatchSize 64 loss = 0.2, acc = 0.55 SpeedTest mit anderen Implementierungen (z.B. Coarsening, RegionProps)

17.3. Schwächen

- TensorFlow hat keine gute Anbindung an dynamischen Input (den wir hier aber brauchen)
- Augmentierung nicht mehr einfach möglich
  - Graphen können nicht ohne weiteres gedreht werden
  - Augmentierung der Form-Features oder der Farbe verändert auch den Graphen
  - Graphgenerierung im Postprocessing schwierig, weil langsam

- Jeder Superpixelalgorithmus und jeder Datensatz (verschiedene Bildauflösungen/anders Anwendungsgebiet) erfordert eigentlich Neuberechnung der geeignesteten Form Features
- PCA nicht möglich, weil zu teuer (alle Form Features zu berechnen ist utopisch und wiederspricht der Grundidee)
- viel langsamer als ausgereifte Bildimplementierungen
- abhängig von der Anzahl der Form-Features und der Größe des Graphen nicht unbedingt speichereffizienter als ein einzelnes Bild, wir haben aber in der Tat kleinere Convolutions.
- es gibt Superpixelalgorithmen, die bei einer Menge von Bildern unterschiedliche Parameter brauchen für geeignete Superpixelrepräsentationen. Diese Algorithmen sind zum Lernen ungeeignet (z.B. Felzenszwalb)

17.3.1. Übergang zum FC-Layer

Der Übergang zum Fully-Connected Layer kann durchaus als eine Schwäche im Vergleich zu klassischen CNN auf Bildern angesehen werden. Pixel und daraus entstehende Receptive-Fields haben auch im späteren Verlauf der Layer eine Ordnung. Diese Ordnung ist auf Graphen und auch auf eingebetten Graphen nicht oder nur bedingt gegeben und man kann diese nicht ausnutzen. (siehe Email) Die Lösung für dieses Problem besteht darin, so viele Layer vor dem FC-Layer zu stacken, sodass die Receptive-Field-Größe sehr klein ist und als Merkmale für das gesamte Bild aufgefasst werden können.

18. Ausblick

18.1. Attention Algorithmus

If the graphs are of different sizes, you need to somehow represent them as fixed-size vectors before the fully connected layers (same principle as varying-length sentences being represented as fixed-size vectors by RNNs). One way is to use an attention mechanism such as equation 7 in https://arxiv.org/abs/1511.05493.

Anderen Algorithmus vorstellen, der das ähnlich macht. SRR oder so.

Ausblick weiterhin erweiterhin auf 2,5d oder 3D Graphen. Das sollte einfach möglich sein.

german bib

glossary aufräumen

no build warnings

A. Weitere Informationen

Symbolverzeichnis

T Tschebyschow-Polynom. 28

deg Gradfunktion der Knoten eines Graphen G mit $\deg\colon \mathcal{V} \to \mathbb{N}$. 16, 49

λ Eigenwert. 21–28

$\langle \cdot, \cdot \rangle$ Skalarprodukt. 15

$\mathbf{A}_{\mathsf{dist}}$ Distanzadjazentmatrix eines eingebetten Graphen G. 35

$\mathbf{A}_{\mathsf{rad}}$ Winkeladjazentmatrix eines eingebetten Graphen G. 35, 38

$\mathbf{F}'$ Output-Featurematrix eines Graphen. 35, 38

$\tilde{\mathbf{A}}_{\mathsf{dist}}$ normalisierte Distanzadjazentmatrix eines eingebetten Graphen G. 35, 38

$\mathbb{N}$ Menge der natürlichen Zahlen. 16, 18, 21, 24, 35, 36, 49, 50

$\mathbb{R}_+$ Menge der positiven reellen Zahlen inklusive Null. 15, 16, 22, 24, 30, 35, 50

$\mathbb{R}$ Menge der reellen Zahlen. 15, 16, 18, 21, 23, 24, 27, 28, 30, 35, 36, 50

$\mathcal{E}$ Kantenmenge eines Graphen G mit $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. 15, 16, 28, 49

$\mathcal{O}$ O-Notation. 28

$\mathcal{V}$ Knotenmenge $\{v_i\}_{i=1}^n$ eines Graphen G. 15, 16, 23, 24, 49, 50

$\tilde{\mathcal{L}}$ reskalierter Laplacian $\mathcal{L}$. 28

diag Diagonalfunktion. 16, 21, 26

$\odot$ Hadamard-Produkt. 15, 26, 38

$\perp$ Orthogonalität. 21

$\sim$ Adjazenzrelation zweiter Knoten eines Graphen G mit $u \sim v$ genau dann, wenn u und v adjazent. 15, 16, 18, 49

mod Modulo. 36

a Voronoi-Gebiet über der Knotenmenge. 39

d gewichtete Gradfunktion der Knoten eines Graphen G mit $d\colon \mathcal{V} \to \mathbb{R}_+$. 16, 39, 50

p Positionsfunktion auf den Knoten $\mathcal{V}$ mit $p\colon \mathcal{V} \to \mathbb{R}^2$. 16, 18, 50

s kürzeste Distanzfunktion mit $s\colon \mathcal{V} \times \mathcal{V} \to \mathbb{N}$. 16, 24, 27, 50

w Gewichtsfunktion der Kanten eines Graph G mit $w\colon \mathcal{V} \times \mathcal{V} \to \mathbb{R}_+$. 15, 16, 18, 24, 39, 50

$\mathbf{A}$ Adjazentmatrix eines Graphen G. 16, 22, 39

$\mathbf{D}$ gewichtete Gradmatrix. 16, 22

$\mathbf{F}$ Featurematrix eines Graphen. 35, 38

$\mathbf{I}$ Identitätsmatrix. 15, 21, 22, 28

$\mathbf{L}$ Laplacian, unnormalisiert. 22–24, 27, 39

$\mathbf{U}$ Eigenvektormatrix. 21, 25–28

$\mathbf{\Lambda}$ Diagonalmatrix der Eigenwerte des Laplacian. 21, 26–28

$\mathcal{L}$ Laplacian, normalisiert oder unnormalisiert. 22–25, 27, 28, 49

$\tilde{\mathbf{L}}$ Laplacian, normalisiert. 22, 24

$\tilde{\mathbf{\Lambda}}$ reskalierte Diagonalmatrix der Eigenwerte des Laplacian. 28

G Graph. 15, 16, 22, 24, 49, 50

$\mathbf{u}$ Eigenvektor mit $\|\mathbf{u}\|_2 = 1$. 21, 23–25, 50

Abbildungsverzeichnis

Literatur

[1] D.I. Shuman and S.K. Narang and P. Frossard and A. Ortega and P. Vandergheynst. “Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Data Domains”. In: *CoRR* (2012).

[2] D.K. Hammond and P. Vandergheynst and R. Gribonval. “Wavelets on graphs via spectral graph theory”. In: *Applied and Computational Harmonic Analysis* (2011), S. 129–150.

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

[4] I.S. Dhillon and Y. Guan and B. Kulis. “Weighted Graph Cuts Without Eigenvectors: A Multilvel Approach”. In: *IEEE* (2007), S. 1944–1957.

[5] M. Defferrard and X. Bresson and P. Vandergheynst. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering”. In: *CoRR* (2016).

[6] M. Reuter and S. Biasotti and D. Giorgi and G. Patane and M. Spagnuolo. “Discrete Laplace-Beltrami operators for shape analysis and segmentation”. In: *Computers & Graphics* (2009), S. 381–390.

[7] T.N. Kipf and M. Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *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