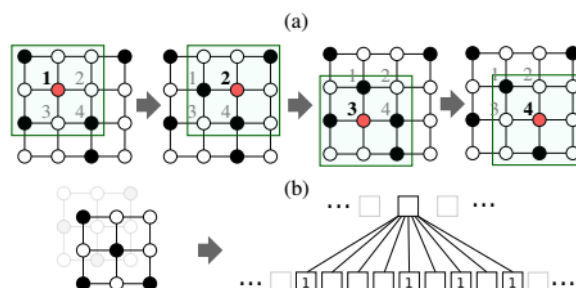


CNNs auf Graphen

9. Februar 2017

1 Einleitung

- Anwendungsfälle:
 1. Aus einer Menge von Graphen soll eine Funktion für Klassifizierungs- oder Regressionsprobleme gelernt werden, die auf nicht bekannte Graphen angewendet werden kann
 2. lerne Graph-Repräsentationen, um auf Graph-Eigenschaften (fehlende Kanten, Knoteigenschaften) unbekannter Graphen zu schließen
- Graphrepräsentation:
 - Graphen können gerichtet oder ungerichtet sein
 - Graphen können zyklisch sein
 - Graphen können mehrere unterschiedliche Kantentypen besitzen (mehrere Perceptive-Field-Layer)
 - Graphen können mehrere diskrete oder kontinuierliche Werte an ihren Knoten haben
- Methode berechnet lokal verbundene Nachbarschaften der Graphen und benutzt sie als die *Receptive Fields* des CNN
- die Methode kann für Graphen mit gewichteten Kanten erweitert werden



- Idee: repräsentiere Bilder als Graph

- ein Bild kann als Graph repräsentiert werden, indem die Knoten jeweils einen Pixel repräsentieren und es eine Kante zwischen zwei Knoten gibt, wenn deren Pixel benachbart sind
- die lokale Nachbarschaft eines Pixels wird repräsentiert als ein Quadrat um den Punkt (hier 3×3)
- Aus der Nachbarschaft kann ein Merkmal ermittelt werden
- üblicherweise gibt es keine räumliche Anordnung einer Graph-Repräsentation
- Probleme:
 1. Welche Nachbarschaften um welche Knoten und in welcher Reihenfolge bilden die Receptive Fields?
 2. Wie können die einzelnen Nachbarschafts-Graphen in einem Vektor repräsentiert werden (Normalisierung)?
- Verfahren:
 1. bestimme eine Knoten-Auswahl inklusive Reihenfolge
 2. bestimme den Nachbarschafts-Graphen um diesen Knoten mit genau k Knoten
 3. normalisiere die Nachbarschafts-Graphen
 4. füttere sie in ein CNN

2 Grundlagen

- Graph $G = (V, E)$ mit $V = \{v_1, \dots, v_n\}$ und $E \subseteq V \times V$, wobei n Anzahl der Knoten und m Anzahl der Kanten
- Adjazenzmatrix A mit Größe $n \times n$, wobei $A_{i,j} = 1$, falls eine Kante von v_i nach v_j existiert (sonst 0) $\Rightarrow v_i$ und v_j sind adjazent
- ein Weg ist eine Sequenz von Knoten, bei der benachbarte Knoten adjazent sind
- $d(u, v)$ beschreibt die minimale Distanz zwischen von u nach v
- $N_1(v)$ beschreibt die 1-Nachbarschaft um einen Knoten, d.h. alle Knoten die adjazent sind zu v

2.1 Beschriftung und Partitionierung

- eine Graph-Beschriftung $l : V \rightarrow S$ bildet einen Knoten auf eine sortierbare Einheit ab
- induziert ein *Ranking* $r : V \rightarrow \{1, \dots, |V|\}$ mit $r(u) < r(v)$ genau dann, wenn $l(u) > l(v)$
- falls l injektiv, dann gibt es eine totale Ordnung der Knoten in G und eine eindeutige Adjazenzmatrix A^l , bei der die Knoten die Position $r(v)$ haben
- eine Graph-Beschriftung induziert eine Partitionierung $\{V_1, \dots, V_k\}$ mit $u, v \in V_i$ falls $l(u) = l(v)$

3 Lernen von Graphen

3.1 Knotenauswahl

- Auswahl an Knoten, für die ein Receptive Field erstellt werden soll
- Sortierung soll dem Verfahren von Bildern nahekomen, d.h. Knoten mit ähnlichen strukturellen Merkmalen sollen auch in der Vektorrepräsentation nah beieinanderliegen
- Graph-Beschreibung l – Metriken:
 - **Betweenness centrality**:
 - * $g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$
 - * σ_{st} beschreibt die Anzahl an kürzesten Pfaden von s nach t ist und σ_{st} die Anzahl dieser Pfade, die durch v gehen
 - **Eigenvector centrality**:
 - * *Google's PageRank* ist eine Variante der Eigenvector centrality
 - * $G = (V, E)$ mit Adjazenzmatrix A , sodass $a_{v,t} = 1$, falls eine Kante von v nach t existiert
 - * relative Centrality von v : $x_v = \frac{1}{\lambda} \sum_{t \in N(v)} x_t = \frac{1}{\lambda} \sum_{t \in G} a_{v,t} x_t$
 - * kann als Eigenwertproblem formuliert werden: $Ax = \lambda x$
 - * zusätzliche Einschränkung: alle Werte des Eigenvektors x sollen nicht-negativ sein \Rightarrow bestimme größten Eigenwert $\lambda \Rightarrow$ eindeutig
 - **Degree centrality**:
 - * Grad der Knoten, d.h. Anzahl adjazenter Knoten (gewichtet: Auswärtsgrad – Einwärtsgrad)
 - **Closeness centrality**:
 - * durchschnittliche Länge zwischen dem Knoten und allen anderen Knoten
 - * je zentraler ein Knoten ist, umso näher sind alle anderen Knoten
 - * $C(x) = \frac{1}{\sum_y d(y,x)}$
 - * kann sich für gerichtete Graphen stark unterscheiden (hohe Closeness für ausgehende Kanten, geringe Closeness für eingehende Kanten)
 - *Weisfeiler-Lehman Algorithmus*
 - *Page-Rank*
-
- eventuell werden diese Metriken gar nicht benötigt, da wir ja eine räumliche Struktur unseres Graphen besitzen, dann kann ähnlich von links nach rechts und von oben nach unten analog zu CNNs auf Bildern vorgegangen werden
- Gegeben: Graph-Beschreibung l , Abstand s , Anzahl w an Receptive Fields

1. sortiere die Knoten auf Basis von l

es werden anscheinend mehrere Metriken benutzt, wie werden diese kombiniert?

2. iteriere über die sortierte Knotenmenge mit Abständen s , bis w Knoten ausgewählt wurden
3. werden weniger als w Knoten gefunden, so werden *all-zero* Receptive Fields erstellt und diese angehängt (*padding purposes*)

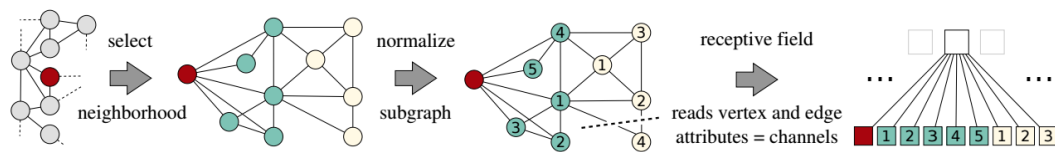
3.2 Nachbarschaftssuche

- Bemerkung: wahrscheinlich falsche Angabe in Algorithmus 2 $\Rightarrow |L| < |V|$, d.h. Abbruch, wenn keine Knoten mehr eingesammelt werden können
 - Gegeben: Knoten v , Größe k des Receptive Fields
1. setze initiale Knotenmenge N auf v
 2. wiederhole bis $|N| > k$:
 - a) berechne für alle Knoten i in N die Nachbarschaften $N_1(i)$ und füge sie zu N hinzu
- Bemerkungen:
 - im Allgemein gilt $|N| \neq k$
 - hier wird das Gewicht der Kanten nicht berücksichtigt, man kann den Algorithmus aber dementsprechend verändern (finde die ersten k Knoten, mit dem kürzesten Pfad zu v)
 - \Rightarrow das ist dann wichtig, wenn das Gewicht etwas über die räumliche Anordnung aussagt

3.3 Normalisierung

- Aus einem Nachbarschaftsgraphen soll ein Receptive Field konstruiert werden
- Problem: Nachbarschaftssuche liefert uns nicht genau k Knoten, sind nicht sortiert
- wenn weniger als k Knoten, dann füge *Dummy-Knoten* hinzu
- Vorgehen:
 1. bestimme Distanz zum Wurzelknoten
 2. führe Centrality-Berechnungen auf den Knoten gleicher Distanzen aus
 3. Canonicalization (breche gleiche Centrality auf) \Rightarrow hier wird **nauty** benutzt
 4. nehme, die ersten k Knoten
- Knoten werden anhand eines Graph-Labelings l sortiert
 - ein Receptive Field für die Knoten (Größe k) und ein Receptive Field für die Kanten (Größe $k \times k$)
 - jedes Knoten- oder Kantenattribut wird in einem Receptive Field abgespeichert (z.B. Farbe)
- Gegeben: Menge von Graphen \mathcal{G} mit k Knoten, Distanzmetriken für $k \times k$ Matrizen d_A und Graphen d_G für k Knoten

- d_A , z.B. *Hamming-Abstand*: $d_A(x, y) = |\{j \in \{1, \dots, N\} | x_j \neq y_j\}|$
- Beispiel: 12345 und 13344 $\rightarrow 2$
- d_G : z.B. *Edit distance*
- Optimierungsproblem über l : $\min_l \sum_{G \in \mathcal{G}} \sum_{G' \in \mathcal{G}} (d_A(A^l(G), A^l(G')) - d_g(G, G'))$
- \Rightarrow für beliebige Graphen G und G' soll die Ähnlichkeit dieser Graphen gleich der Ähnlichkeit der Graphen im Vektorraum sein (basierend auf den Adjazenzmatrizen der Graphen)
- \Rightarrow Problem ist NP-schwer
- Alternative: wähle aus einer Menge von Labelings die beste zu einer gegebenen Menge von Graphen
 - $\{(G_1, G'_1), \dots, (G_N, G'_N)\}$ eine zufällige Auswahl an Graphpaaren von \mathcal{G}
 - wähle das Labeling l so, dass $\sum_{i=1}^N \frac{d_A(A^l(G_i), A^l(G'_i))}{N}$ minimal
- Labelings werden nur berechnet für Knoten gleicher Distanz zum Startknoten v
- Labelings sind im Allgemeinen nicht injektiv \Rightarrow sortiere anhand lexikographischer maximaler Adjazenzmatrizen



4 Auswertung

- CNNs mit Bildern können identisch über CNNs mit Graphen dargestellt werden
- Methode funktioniert teilweise deutlich besser als State-of-the-Art Graph-Kerne (z.B. bei Klasifizierungsproblemen)

5 Zukünftige Arbeiten

- gewichtete Kanten (oder allgemeiner Graphen mit Kanteneigenschaften)
- Graphen auf andere Netze übertragen, z.B. RNNs
- kombiniere unterschiedliche Receptive Field-Größen

6 Dünnbesetzte Matrizen

Aus Speichereffizienz-Gründen lohnt es sich mit *dünnbesetzte Matrizen* (engl. *Sparse Matrices*) für Adjazenzmatrizen zu rechnen. Eine dünnbesetzte Matrix ist eine Matrix, bei der so viele Einträge aus Nullen bestehen, dass man nach Möglichkeiten sucht, dies insbesondere hinsichtlich Algorithmen sowie Speicherung auszunutzen. In der Regel gelten Matrizen als dünnbesetzt, wenn eine Matrix mit n^2 Einträgen nicht mehr als n oder $n \cdot \log n$ Einträge ungleich Null besitzt. Eine Matrix die nicht dünnbesetzt ist, heißt *dicht* (engl. *dense*).

Dünnbesetzte Matrizen erlauben einigen Operationen, optimierter bzw. schneller zu sein, da nicht alle Werte der Matrix betrachtet werden müssen. Es gibt jedoch auch Operationen, die nicht auf dünnbesetzten Matrizen definiert sind, so dass sie vorher in eine normale Dense-Matrix überführt werden müssen.