Expressives und Effizientes Deep Learning auf Graph-Strukturierten Daten[1]

Matthias Fey[2]

Abstract: Dise Arbeit stellt neue Ansätze für das maschinelle Lernen auf unregelmäßig strukturierten Eingabedaten wie Graphen, Punktwolken und Mannigfaltigkeiten vor. Insbesondere löst sich diese Arbeit von der Beschränkung konventioneller Deep Learning-Techniken auf reguläre Daten und bietet Lösungen für den Entwurf, die Implementierung und die Skalierung von Deep Learning-Methodiken auf Graphen an, bekannt als *Graph Neural Networks (GNNs).* Wir analysieren die Beziehung von GNNs zu traditionellen Verfahren des Deep Learnings und der Graph-Theorie, entwickeln skalierungsfähige GNN-Lösungen und stellen die *PyG*-Bibliothek für die effziente Realisierung von GNNs vor.

1 Einleitung

Unsere Welt ist reich an Strukturen, aufgebaut aus Objekten, ihren Beziehungen und Hierarchien. So zerlegen sich beispielsweise Fahrkarten in Straßen und Kreuzungen, Webseiten im Internet referenzieren sich über Hyperlinks, und selbst chemische Verbindungen lassen sich durch eine Menge von Atomen und deren Wechselwirkungen beschreiben. Ein geeigneter Weg, solche strukturierten Informationen darzustellen, ist in Form eines *Graphen*, welcher durch eine Menge von *Knoten* und deren *Kanten* beschrieben wird.

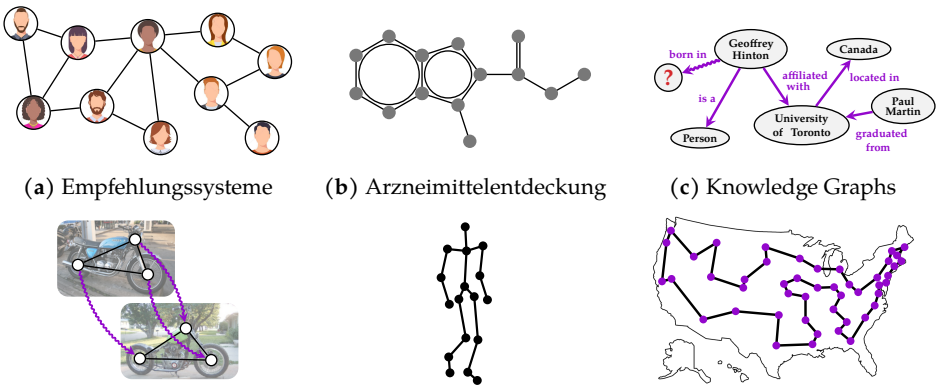
Traditionell basieren Modelle des maschinellen Lernens insbesondere auf dem Ansatz des *Feature Engineerings*, bei dem Merkmale durch entwickelte Verfahren von Fachleuten extrahiert werden und zum Training eines Modells verwendet werden. In den letzten Jahren wurde dieses Verfahren größtenteils durch einen datengesteuerten *Representation Learning*-Ansatz mittels *Deep Learning* ersetzt, bei dem das Modell eigenständig in der Lage ist, die Rohdaten entsprechend der zugrunde liegenden Problemstellung zu transformieren.

Obwohl Graphen in unserer Welt allgegenwärtig sind, können die meisten gegenwärtigen Methoden des maschinellen Lernens solch reichhaltige strukturelle Darstellungen jedoch nicht verarbeiten, da insbesondere erwartet wird, dass Merkmale durch Vektoren fester Größe beschrieben werden. Daher sind existierende Modelle in der Lage nur eine kleine Teilmenge aller potenziellen Anwendungen zu verarbeiten, siehe Abb. 1.

[1] Englischer Titel der Dissertation: “On the Power of Message Passing for Learning on Graph-Structured Data”

[2] Technische Universität Dortmund, Lehrstuhl für Computergraphik, Otto-Hahn-Straße 16, 44227 Dortmund, Deutschland matthias.fey@udo.edu

CC BY SA



(**a**) Empfehlungssysteme (**b**) Arzneimittelentdeckung (**c**) Knowledge Graphs

(**d**) Graph Matching (**e**) Motion Capturing (**f**) Kombinatorische Optimierung

Abb. 1: **Beispielhafte Anwendungen für maschinelles Lernen auf Graphen.**

Diese Beobachtung erfordert die Entwicklung neuartiger neuronaler Netze, welche unter dem Namen *Graph Neural Networks (GNNs)* gruppiert werden [Ha20]. Trotz ihres reichhaltigen Potenzials bringen GNNs auch neue Herausforderungen mit sich. Da die meisten Bausteine neuronaler Netze eine feste Struktur als gegeben voraussetzen (insbesondere aufgrund ihrer effektiven Parallisierung auf der GPU), ist die Erweiterung und Verallgemeinerung ihrer Konzepte auf beliebig strukturierte Daten sowohl theoretisch als auch implementierungstechnisch schwierig.

Diese Arbeit baut auf dem allgemeinen Konzept von GNNs auf, analysiert ihre Fähigkeiten und entwirft neue Methodiken um ihre Leistung sowohl im Allgemeinen als auch in aufgabenspezifischen Anwendungen zu steigern. Insbesondere beschäftigt sich diese Arbeit auch mit dem Entwurf effizienter und skalierbarer GNN-Architekturen, um deren Anwendung in ressourcen-beschränkten Umgebungen zu ermöglichen.

Beiträge. Diese Arbeit fasst die folgenden Beiträge aus [Fe22] zusammen:

- Wir führen GNNs als neuronales *Message Passing*-Framework ein und entwickeln *SplineCNNs* [Fe18] als Schnittstelle zwischen GNNs und CNNs. SplineCNNs verbessern frühere Arbeiten, indem sie GNNs mit einem anisotropen Filter ausstatten, welcher für das Lernen auf Graphen, Superpixeln, Punktwolken und Mannigfaltigkeiten genutzt werden kann.
- Wir analysieren die Expressivität von GNNs aus theoretischer Sicht, indem wir sie mit der *Weisfeiler-Lehman (WL)* Graph-Isomorphie-Heuristik in Beziehung setzen [Mo19]. Die theoretischen Resultate zeigen, dass GNNs die gleiche Ausdrucksstärke wie die WL-Heuristik besitzen, um nicht-isomorphe (Teil-)Graphen zu unterscheiden. Basierend auf diesen Erkenntnissen entwickeln wir *k-GNNs*, eine Verallgemeinerung von GNNs mit nachweislich höherer Expressivität, welche Message Passing zwischen Teilgraph-Strukturen anstatt individueller Knoten erlaubt.

- Wir identifizieren inhärente Mängel von GNNs und zeigen, wie diese in der Praxis überwunden werden können [Fe19, FYW20, Fe20]. Insbesondere entwickeln wir ein Framework namens *GNNAutoScale (GAS)* [Fe21], welches in der Lage ist GNNs effizient auf riesige Datenmengen zu skalieren. Während alternative Lösungen für die Skalierbarkeit die Ausdrucksstärke von GNNs abschwächen, ist GAS nachweislich dazu in der Lage, die ursprüngliche Expressivität des GNN zu erhalten.
- Abschließend entwickeln wir die *PyTorch Geometric (PyG)* Bibliothek [FL19] für die effiziente Realisierung von Deep Learning-Verfahren auf unregelmäßig strukturierten Eingabedaten wie Graphen, Punktwolken und Mannigfaltigkeiten. PyG verfügt über eine allgemeine Message Passing-Schnittstelle, welche auf spärliche Graphen ausgelegt ist und alle Komponenten effizient parallelisiert. Alle Erkenntnisse dieser Arbeit sowie verwandte Forschungsarbeiten sind in der PyG-Bibliothek integriert, um das tiefe maschinelle Lernen auf Graphen auf breiter Basis einsetzbar zu machen. Bis heute wurden etwa 2000 Forschungsarbeiten unter Verwendung von PyG entwickelt. Die zukünftige Entwicklung wird von einem Team aus Kern-Entwicklern vorangetrieben und von mehr als 400 externen Mitwirkenden aus der ganzen Welt unterstützt. PyG wird derzeit von vielen verschiedenen Unternehmen, darunter Spotify, Netflix und AstraZeneca, erfolgreich in der Produktion eingesetzt.

2 Graph Neural Networks

Sei $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ein Graph mit Knotenmerkmalen $\mathbf{x}_v$ für alle $v \in \mathcal{V}$. Graph Neural Networks werden üblicherweise über ein neuronales *Message Passing* Framework definiert, welches benachbarte Knotenrepräsentationen iterativ aggregiert [Ha20]. Nach L Iterationen erfasst die Repräsentation eines Knotens sowohl Graph-Struktur als auch Merkmalsinformationen innerhalb seiner L-tiefen Nachbarschaft. Formal ist die ℓ-te Schicht eines GNNs definiert als

$$\mathbf{h}_v^{(\ell)} = f_\theta^{(\ell)}(\mathbf{h}_v^{(\ell-1)}, \{\mathbf{h}_w^{(\ell-1)} : w \in \mathcal{N}(v)\}) \tag{1}$$

$$= \textsc{Update}_\theta^{(\ell)}\Big(\mathbf{h}_v^{(\ell-1)}, \bigoplus_{w \in \mathcal{N}(v)} \textsc{Message}_\theta^{(\ell)}\big(\mathbf{h}_w^{(\ell-1)}, \mathbf{h}_v^{(\ell-1)}\big)\Big), \tag{2}$$

wobei $\mathcal{N}(v) = \{w : (w, v) \in \mathcal{E}\}$ die Nachbarschaft um $v \in \mathcal{V}$ definiert. Es gilt $\mathbf{h}_v^{(0)} = \mathbf{x}_v$. Graph Neural Networks lernen damit eine Knotenrepräsentation $\mathbf{h}_v^{(L)} \in \mathbb{R}^D$ für alle $v \in \mathcal{V}$, welche für die Vorhersage auf Knotenebene, Kantenebene oder Graphebene benutzt werden kann. Die meisten GNN-Operatoren $f_\theta^{(\ell)}$ können in differenzierbare und parametrisierte $\textsc{Message}_\theta^{(\ell)}$ und $\textsc{Update}_\theta^{(\ell)}$ Funktionen und permutations-invariante Aggregationen $\bigoplus$ zerlegt werden [Fe22]. $\textsc{Message}$, $\bigoplus$ und $\textsc{Update}$ können je nach Aufgabenstellung auf unterschiedliche Weise gewählt werden.

2.1 Message Passing mit kontinuierlichen B-Spline Filtern

Graph Neural Networks nutzen sehr ähnliche Konzepte der Faltung auf regulären Domänen für das Erlernen von Repräsentationen. Insbesondere teilen sie die Prinzipien der *Lokalität* — traditionelle CNNs führen eine lernbare Transformation lokaler Patches durch, während GNNs eine lernbare Transformation lokaler Nachbarschaften durchführen. Darüber hinaus nutzen beide das Prinzip der *Gewichtsteilung*, welches zu Translations- und Permutationsäquivarianz im Bild- bzw. Graphbereich führt.

Im Vergleich zu CNNs sind häufig verwendete GNN-Operatoren jedoch auch restriktiver. Insbesondere fehlt den meisten GNN-Operatoren ein anisotroper Filter, der die Richtung der Kanten (z.B. unten rechts oder oben links) miteinbezieht. Als Lösung für dieses Problem wurde der *SplineCNN*-Operator [Fe18] entwickelt. SplineCNNs bezeichnen eine Variante von tiefen GNNs, die den traditionellen CNN-Faltungsoperator durch eine kontinuierliche Filter-Formulierung verallgemeinern und damit insbesondere CNNs auf diskreten Eingaben entsprechen.

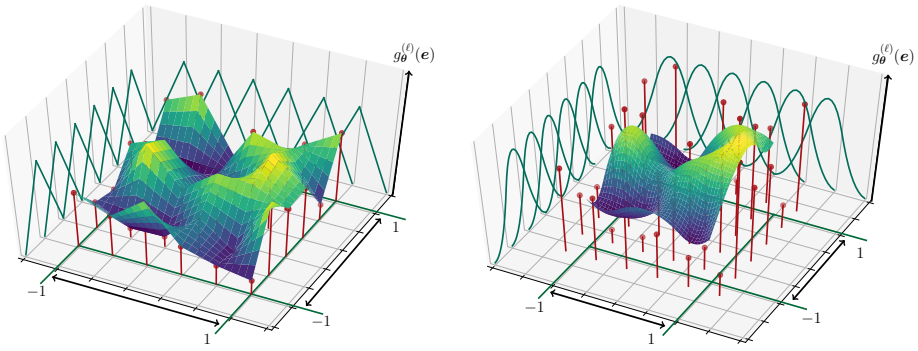
SplineCNNs nutzen mehr-dimensionale Kantenmerkmale, um das Lernen anisotroper Filter-Funktionen zu ermöglichen. Beispielsweise sind bei Bildern oder Meshes zusätzliche Informationen wie die relativen Positionen der Knoten in den Eingabedaten vorhanden. Daher erwartet SplineCNN als zusätzliche Eingabe D-dimensionale Kantenmerkmale $\mathbf{e}_{w,v}$ für jede Kante $(w, v) \in \mathcal{E}$, die als Richtungsdeskriptoren verstanden werden können.

Der Spline-basierte Faltungsoperator nutzt dann eine *kontinuierliche* Filter-Formulierung $g_\theta^{(\ell)} \colon \mathbb{R}^D \to \mathbb{R}^{F \times F'}$, welcher die spezifische und anisotrope Transformation für jeden Nachbarsknoten $w \in \mathcal{N}(v)$ definiert, gegeben $\mathbf{e}_{w,v} \in \mathcal{E}$:

$$\mathbf{h}_v^{(\ell)} = \frac{1}{|\mathcal{N}(v)|} \sum_{w \in \mathcal{N}(v) \cup \{v\}} g_\theta^{(\ell)}(\mathbf{e}_{w,v}) \cdot \mathbf{h}_w^{(\ell-1)}. \tag{3}$$

Damit beschreiben die Kantenmerkmale $\mathbf{e}_{w,v}$ *wie* Information aggregiert werden soll, und Knotenmerkmale $\mathbf{h}_w^{(\ell-1)}$ *welche* Information aggregiert werden soll. Insbesondere verallgemeinert dieses Schema traditionalle CNNs auf graph-strukturierten Daten, da eine Transformations-Matrix über $g_\theta^{(\ell)}$ für *jeden beliebigen* Abtastpunkt $\mathbf{e}_{w,v}$ bestimmt werden kann, nicht nur an diskreten stationären Punkten.

Die konkrete Definition der Filter-Funktion $g_\theta^{(\ell)}$ nutzt das Konzept der *B-Spline Basisfunktionen*, welche durch eine konstante Anzahl an tranierbaren Parametern gesteuert wird (siehe [Fe18] für eine formale Definition von $g_\theta^{(\ell)}$). Das Konzept von *B-Spline Basisfunktionen* erweist sich als vorteilhaft für die effiziente Berechnung und Skalierbarkeit des SplineCNN-Operators, denn durch die lokale Unterstützungseigenschaft von B-Spline Basisfunktionen ist die Auswertung von $g_\theta^{(\ell)}$ *unabhängig* von der tatsächlichen Anzahl an Parametern θ. Abb. 2 visualiert die spline-basierte Filter-Formulierung für unterschiedliche



(a) Lineare B-Spline Basisfunktion (b) Quadratische B-Spline Basisfunktion

Abb. 2: **Beispiele für spline-basierte Filter mit Basisgrad (a)** 1 **und (b)** 2 **für Kantendimensionalität** $D = 2$**.** Die Höhe der roten Punkte beschreiben die trainierbaren Parameter θ. Für die finale Ausgabe werden sie mit den Elementen der B-Spline Tensorproduktbasis multipliziert [Fe18].

B-Spline Basisgrade. Der interessierte Leser sei auf [Fe18] für zusätzliche Information zur effizienten GPU-Parallelisierung dieses Schemas verwiesen.

Beziehung zu CNNs. SplineCNNs können als direkte Verallgemeinerung von CNNs für *ungerade* Filtergrößen in jeder Dimension verstanden werden. Auf einem D-dimensionalem Gitter-Graphen mit diagonalen, horizontalen und vertikalen Kanten entspricht $g_\theta^{(\ell)}$ mit B-Spline Basisgrad 1 und Filtergröße 3×3 der diskreten Faltung eines Bildes mit Filtergröße 3×3 (siehe [Fe22] für einen formalen Beweis). Dies gilt auch für weitere Filtergrößen, wenn die Nachbarschaften des Gitter-Graphen entsprechend modifiziert werden.

Anwendungen. SplineCNNs können auf unterschiedlichen Domänen angewendet werden. Wir haben SplineCNNs auf drei Domänen evaluiert: (1) Superpixel-Bilder, (2) Graphen und (3) Meshes (siehe [Fe22]). Beispielsweise können SplineCNNs geometrische Merkmale direkt aus den Mesh-Rohdaten lernen, während ähnliche Ansätze auf handgefertigte Merkmalsdeskriptoren als Eingabeknotenmerkmale angewiesen sind, was sowohl zu einer besseren Laufzeit und besseren Ergebnissen führt. In allen drei Domänen konnten SplineCNNs bisherige Ergebnisse steigern.

2.2 Maximal Expressive Graph Neural Networks

GNNs sind ein leistungsfähiges Werkzeug für die Aggregation von Merkmalen in graphstrukturierten Daten. Im Idealfall ist ein maximal expressives GNN in der Lage, isomorphe Graphen auf dieselbe Repräsentation abzubilden, während nicht-isomorphe Graphen auf unterschiedliche Repräsentation abgebildet werden. Eine solche Fähigkeit setzt jedoch voraus, dass GNNs das Problem der Graph-Isomorphie lösen können.

Wir können die Expressivität eines GNN bezüglich der Fähigkeit nicht-isomorphe Graph-Strukturen zu unterscheiden theoretisch analysieren, indem wir sie mit dem *Weisfeiler-Lehman (WL)*-Test [WL68] in Verbindung bringen, eine Heuristik, die den Graph-Isomorphismus-Test für eine breite Klasse von Graphen erfolgreich löst. Der WL-Test funktioniert wie folgt: Bei einer anfänglichen Färbung $c_v^{(0)}$ der Knoten zweier Graphen erhalten zwei Knoten mit der gleichen Färbung unterschiedliche Färbungen, wenn die Anzahl der identisch gefärbten Nachbarn nicht gleich ist:

$$c_v^{(\ell)} = \textsc{Hash}\left(c_v^{(\ell-1)}, \left\{c_w^{(\ell-1)} : w \in \mathcal{N}(v)\right\}\right). \tag{4}$$

Um zu testen, ob zwei Graphen $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ und $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ nicht-isomorph sind, wird Gleichung (4) "parallel" auf beiden Graphen bis zur Konvergenz augeführt und deren Farb-Histogramme miteinander verglichen.

Interessanterweise weist eine einzelne Iteration des WL-Tests große Ähnlichkeiten mit dem in GNNs verwendeten Message Passing-Schema auf, bei dem die Hash-Funktion durch differenzierbare und trainierbare Message- und Update-Funktionen ersetzt wird.

Das erste theoretische Resultat in [Mo19] zeigt, dass ein GNN bezüglich der Unterscheidung nicht-isomorpher (Teil-)Graph nicht expressiver ist als der WL-Test:

Theorem 1 *Sei $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ein Graph. Dann gilt*

$$c_v^{(\ell)} = c_w^{(\ell)} \Rightarrow \mathbf{h}_v^{(\ell)} = \mathbf{h}_w^{(\ell)} \tag{5}$$

für alle $v, w \in \mathcal{V}$, für alle $\ell \geq 1$ und für alle Parameter θ, d.h. ein GNN ist höchstens so expressiv *wie der WL-Test.*

Wir können jedoch auch zeigen, dass bei geeigneter Parameter-Initialisierung ein GNN die gleiche Expressivität wie der WL-Test aufweist, was die Äquivalenz vervollständigt:

Theorem 2 *Sei $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ein Graph. Dann existiert eine Sequenz von Parametern $(\theta^{(1)}, \dots \theta^{(\ell)})$, so dass*

$$\mathbf{h}_v^{(\ell)} = \mathbf{h}_w^{(\ell)} \iff c_v^{(\ell)} = c_w^{(\ell)} \tag{6}$$

für alle $v, w \in \mathcal{V}$ und für alle $\ell \geq 1$, d.h. ein GNN kann so expressiv *sein wie der WL-Test.*

In Anbetracht der obigen Ergebnisse aus [Mo19] können GNNs als eine Erweiterung des WL-Tests betrachtet werden, welche die gleiche Expressivität haben, aber flexibler sind in ihrer Fähigkeit sich an die gegebene Datenverteilung anzupassen. Dadurch können GNNs nicht nur verschiedene Strukturen unterscheiden, sondern auch lernen, ähnliche Graph-Strukturen auf ähnliche Repräsentation abzubilden und Abhängigkeiten zwischen Graph-Strukturen zu erfassen.

***k*-dimensionale GNNs.** Da die Fähigkeiten des WL-Tests vollständig charakterisiert sind [Fe22], können diese Ergebnisse auch auf GNNs übertragen werden. So wird zum

Beispiel ersichtlich, dass sowohl der WL-Test als auch GNNs Schwierigkeiten darin haben, reguläre Graphen voneinander zu unterscheiden. Daher können GNNs nachweislich expressiver gestaltet werden, indem sie auf Grundlage des k-*WL*-Tests modelliert werden [Mo19]. Die sogenannten k-*GNNs* aggregrieren dabei Informationen nicht zwischen einzelnen Knoten, sondern zwischen Teilgraph-Strukturen (siehe [Fe22] für eine formale Definition). Dadurch kann diese "höhere Form" des Message Passings strukturelle Informationen erfassen, die auf der Knotenebene nicht sichtbar sind. Wir haben k-GNNs auf anspruchsvollen Benchmark-Datensätzen evaluiert, einschließlich molekularer Graph-Datensätze, in denen wir ihre bessere Leistung im Vergleich zu herkömmlichen GNNs zeigen.

3 Skalierbare Graph Neural Networks

Eine der Herausforderungen, die die breite Einführung von GNNs in industriellen und sozialen Anwendungen verhindern, ist die Schwierigkeit, sie auf große Graphen zu skalieren. Die Anwendung des stochastischen Mini-Batch-Trainings in GNNs ist besonders rechenintensiv, da die Repräsentation eines Knotens rekursiv von allen Repräsentationen seiner Nachbarn abhängt, ein Problem das durch den exponentiellen Wachstum als *Nachbarschaftsexplosion* bekannt ist. Daher sind Skalierungstechniken für die Anwendung von GNNs auf großen Graphen unverzichtbar. Dies funktioniert in der Regel über das *Sampling* von Knoten und Kanten, bei dem der ursprüngliche Teilgraph um einen Knoten auf eine geeignete Anzahl von Knoten und Kanten reduziert wird. Jedoch führt das Sampling dazu, dass GNNs ihre ursprüngliche Expressivität verlieren, da wertvolle strukturelle Informationen innerhalb der Teilgraphen verloren gehen können (siehe [Fe21] für einen formalen Beweis).

Daher stellt sich die Frage, ob es eine skalierbare GNN-Technik gibt, die nachweislich dazu in der Lage ist, die Expressivität des ursprünglichen GNNs zu erhalten. Eine solche Lösung ist in dem *GNNAutoscale (GAS)* Framework [Fe21] beschrieben. GAS erreicht dies, indem es Teilbäume der ursprünglichen Berechnung durch die Verwendung historischer Repräsentationen aus früheren Trainingsiterationen approximiert.

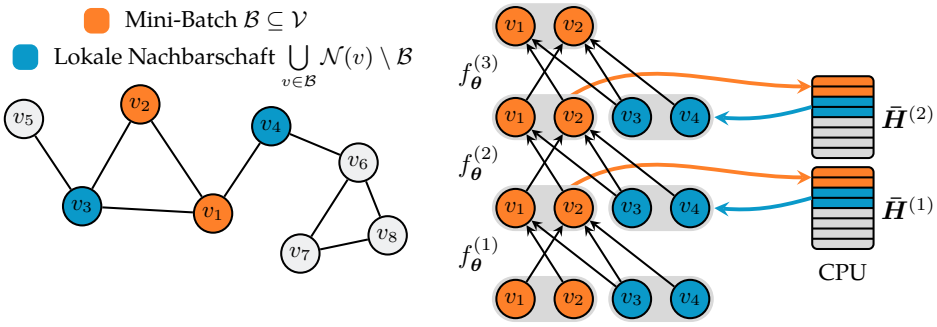
Sei $\mathbf{h}_v^{(\ell)}$ eine Repräsentation des Knotens $v \in \mathcal{B}$ in der ℓ-ten Schicht eines Mini-Batches $\mathcal{B} \subseteq \mathcal{V}$. Basierend auf Gleichung (1) kann die Ausführung einer Schicht im GNN dann wie folgt formuliert werden:

$$\mathbf{h}_v^{(\ell+1)} = f_\theta^{(\ell+1)}\left(\mathbf{h}_v^{(\ell)}, \left\{\mathbf{h}_w^{(\ell)} : w \in \mathcal{N}(v)\right\}\right) \tag{7}$$

$$= f_\theta^{(\ell+1)}\left(\mathbf{h}_v^{(\ell)}, \left\{\mathbf{h}_w^{(\ell)} : w \in \mathcal{N}(v) \cap \mathcal{B}\right\} \cup \left\{\mathbf{h}_w^{(\ell)} : w \in \mathcal{N}(v) \setminus \mathcal{B}\right\}\right) \tag{8}$$

$$\tilde{\mathbf{h}}_v^{(\ell+1)} \approx f_\theta^{(\ell+1)}\left(\mathbf{h}_v^{(\ell)}, \left\{\mathbf{h}_w^{(\ell)} : w \in \mathcal{N}(v) \cap \mathcal{B}\right\} \cup \underbrace{\left\{\bar{\mathbf{h}}_w^{(\ell)} : w \in \mathcal{N}(v) \setminus \mathcal{B}\right\}}_{\text{Historische Repräsentationen}}\right). \tag{9}$$

Gleichung (8) separiert die Nachbarschaft in *zwei* Teile: (1) die Nachbarschaft $\mathcal{N}(v) \cap \mathcal{B}$, die ebenso Teil des aktuellen Mini-Batches $\mathcal{B}$ ist, und (2) Nachbarschaft $\mathcal{N}(v) \setminus \mathcal{B}$, die nicht im



(a) Mini-batch Auswahl (b) GAS-Ausführung

Abb. 3: **Mini-batch Verarbeitung von GNNs mit historischen Repräsentationen.**

aktuellen Mini-Batch enthalten ist. Die zweite Menge kann über historische Repräsentationen $\bar{\mathbf{h}}_w^{(\ell)}$ approximiert werden, welche in früheren Trainingsiterationen berechnet wurden (vgl. Gleichung (9)). Historische Repräsentation fungieren hierbei als *Offline-Speicher*. Nach jedem Trainingsschritt werden die neu berechneten Repräsentationen $\mathbf{h}_v^{(\ell+1)}$ in den Offline-Speicher geschrieben und dienen dann als historische Repräsentationen $\bar{\mathbf{h}}_w^{(\ell+1)}$ in zukünftigen Iterationen (siehe Abb. 3). Damit werden nur Knoten in $\mathcal{B}$ und ihre lokale Nachbarschaft für die Berechnung benutzt, unabhängig von der Tiefe des GNNs. Dies vermeidet das Problem der Nachbarschaftsexplosion und führt zu einem konstanten GPU-Speicherverbrauch in Bezug zur Eingabegröße.

Approximationsfehler. Die Anwendung von historischen Repräsentationen führt zu einem Approximationsfehler $\|\tilde{\mathbf{h}}_v^{(\ell)} - \mathbf{h}_v^{(\ell)}\|$ in der Ausgabe, der durch zwei Komponenten verursacht wird: (1) Der Approximationsfehler der Eingabe $\|\tilde{\mathbf{h}}_v^{(\ell-1)} - \mathbf{h}_v^{(\ell-1)}\|$ und der durch die historischen Repräsentationen induzierte Fehler $\|\bar{\mathbf{h}}_v^{(\ell-1)} - \tilde{\mathbf{h}}_v^{(\ell-1)}\|$. Nichsdestotrotz kann gezeigt werden, dass dieser Fehler beschränkt ist und daher auch weiterhin maximal expressive GNNs erlaubt:

Theorem 3 *Sei $f_\theta^{(L)}$ ein L Schichten tiefes Lipschitz-stetiges GAS-GNN. Dann existiert eine Funktion $\phi\colon \mathbb{R}^D \to \Sigma$, so dass $\phi(\tilde{\mathbf{h}}_v^{(L)}) = c_v^{(L)}$ für alle $v \in \mathcal{V}$.*

Theorem 3 zeigt, dass skalierbare GNNs, die historische Repräsentationen verwenden, auch weiterhin in der Lage sind, nicht-isomorphe Graphen (die durch den WL-Test unterscheidbar sind) zu unterscheiden. Als solches ist GAS die erste skalierbare Lösung, die nachweislich die Expressivität des zugrundeliegenden GNN erhält [Fe21].

Experimente. Es zeigt sich, dass GAS effizient auf tiefen GNNs angewendet werden kann und expressive Repräsentationen erlernen kann. Insbesondere ist GAS ≈160x schneller und verbraucht ≈10x weniger Speicher als GNNs mit vergleichbaren Sampling-Algorithmen.

Der Nutzen tiefer und expressiver GNNs zeigt sich besonders auf großen Graphdatensätzen, was den derzeitigen Stand der Technik auf diesen erheblich verbessert.

4 Effiziente Graph Neural Networks via PyTorch Geometric

Moderne Deep Learning-Softwarebibliotheken sind auf reguläre Strukturen und statische Matrix-Multiplikationen ausgerichtet. Dies macht die effiziente Realisierung von GNNs zu einer Herausforderung, da ein hoher GPU-Durchsatz bei sehr spärlichen und unregelmäßigen Daten unterschiedlicher Größe erreicht werden muss.

Um diese Ziele zu erreichen, haben wir die *PyTorch Geometric (PyG)*[3] Bibliothek entwickelt [FL19]. PyG bündelt alle Resultate aus [Fe22] und eine Vielzahl an verwandter Arbeiten und macht sie einer Breite Masse zugänglich. Dabei verbindet PyG die Tensor-zentrierte Perspektive von Deep-Learning-Frameworks mit dem spärlichen und unregelmäßigen Design-Paradigma von GNNs. Insbesondere erreicht PyG einen hohen GPU-Durchsatz durch die Nutzung eigens kreierter spärlicher GPU-Routinen.

Im Herzen von PyG steht eine allgemeine Message Passing-Schnittstelle, um ein schnelles und sauberes Prototyping von neuen Forschungsideen zu ermöglichen. Insbesondere ermöglicht die Schnittstelle die Definition beliebiger GNN-Schichten, indem MESSAGE, $\bigoplus$ und UPDATE in anpassbare und benutzerdefinierte Funktionen zerlegt werden. Im Allgemeinen bietet die Message Passing-Formulierung aus Gleichung (1) zwei Dimensionen der Parallelisierung: (1) Der "parallele Kantenraum" für die Berechnung von MESSAGE und (2) der "parallele Knotenraum" für die Berechnung von UPDATE, welche abwechselnd in einer allgemeinen GNN-Pipeline ausgeführt werden. Wir können durch effiziente parallele `scatter_`$\bigoplus$ und `gather`-Operationen leicht zwischen beiden Räumen wechseln. Insbesondere führt eine `gather`-Operation ein paralleles Lesen unter Verwendung von E Threads auf der GPU durch, wobei jeder Thread die Merkmale von Quell- und Zielknoten aus dem Speicher liest. In ähnlicher Weise führt eine `scatter_`$\bigoplus$-Operation eine parallele Aggregation unter Verwendung von E Threads mittels atomarer Funktionen durch.

Desweiteren stellt PyG eine Vielzahl an GNN-Schichten und Sampling-Operatoren bereit, und erlaubt effizientes Mini-Batching von Graph-Strukturen. Neben GNN-Schichten stehen ebenso Pooling und Explainability-Operationen zur Verfügung. PyG erlaubt das Lernen sowohl auf homogenen als auch heterogenen Graphen. Über 100 Beispiele, Blogs, Videos und Notebooks führen praktisch in die Welt des maschinellen Lernens auf Graphen ein. Bis heute wurden etwa 2000 Forschungsarbeiten unter Verwendung von PyG entwickelt. Die zukünftige Entwicklung wird von einem Team aus Kern-Entwicklern vorangetrieben und von mehr als 400 externen Mitwirkenden aus der ganzen Welt unterstützt. PyG wird derzeit von vielen verschiedenen Unternehmen, darunter Spotify, Netflix und AstraZeneca, erfolgreich in der Produktion eingesetzt.

[3] GitHub: `https://github.com/pyg-team/pytorch_geometric`

Literaturverzeichnis

[Fe18] Fey, M.; Lenssen, J. E.; Weichert, F.; Müller, H.: SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels. In: Computer Vision and Pattern Recognition (CVPR). 2018.

[Fe19] Fey, M.: Just Jump: Dynamic Neighborhood Aggregation in Graph Neural Networks. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (RLGM). 2019.

[Fe20] Fey, M.; Lenssen, J. E.; Masci, J.; Kriege, N. M.: Deep Graph Matching Consensus. In: International Conference on Learning Representations (ICLR). 2020.

[Fe21] Fey, M.; Lenssen, J. E.; Weichert, F.; Leskovec, J.: GNNAutoScale: Scalable And Expressive Graph Neural Networks via Historical Embeddings. In: International Conference on Machine Learning (ICML). 2021.

[Fe22] Fey, M.: On the Power of Message Passing for Learning on Graph-Structured Data. Dissertation, TU Dortmund University, 2022.

[FL19] Fey, M.; Lenssen, J. E.: Fast Graph Representation Learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (RLGM). 2019.

[FYW20] Fey, M.; Yuen, J. G.; Weichert, F.: Hierarchical Inter-Message Passing for Learning on Molecular Graphs. In: ICML Workshop on Graph Representation Learning and Beyond (GRL+). 2020.

[Ha20] Hamilton, W. L.: Graph Representation Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, 14(3):1–159, 2020.

[Mo19] Morris, C.; Ritzert, M.; Fey, M.; Hamilton, W. L.; Lenssen, J. E.; Rattan, G.; Grohe, M.: Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks. In: Conference on Artificial Intelligence (AAAI). 2019.

[WL68] Weisfeiler, B.; Lehman, A. A.: A Reduction of a Graph to a Canonical Form and an Algebra Arising During this Reduction. Nauchno-Technicheskaya Informatsia, 2(9), 1968.

Matthias Fey ist ein Gründungsmitglied der Firma Kumo.ai, wo er daran arbeitet, die Anwendung von Graph Neural Networks auf Data Warehouses zu realisieren. Zuvor promovierte er am Lehrstuhl für Computergraphik an der Technischen Universität Dortmund. Sein Forschungsgebiet liegt in der Entwicklung neuer Deep-Learning-Methoden, die auf irregulären Daten wie Graphen, Punktwolken und Mannigfaltigkeiten angewendet werden können. Darüber hinaus ist er der Erfinder der PyG (PyTorch Geometric) Bibliothek, die viele der vorgeschlagenen Methoden in diesem Bereich bündelt. Er ist ebenso Mitglied des Open Graph Benchmark (OGB) Teams. Matthias studierte Informatik an der Technischen Universität Dortmund, an der er 2013 seinen Bachelor und 2017 seinen Master machte.