Finden von Vertex Covers auf Graphen mithilfe eines neuronalen Netzes

Nina Hammer 29. September 2024

1 Einleitung

Im 2018 veröffentlichten Paper *Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search* [3] wird ein Vorgehen beschrieben, NP-harte Probleme auf Graphen mithilfe von neuronalen Netzen zu lösen. Hierzu wird eine Kombination aus einem Graph Convolutuional Network und einem Greedy Suchalgorithmus verwendet. Das dort entstandene Verfahren konnte für 100% aller Testgraphen ein Maximum Independent Set erzeugen. Das neuronale Netz aus diesem Paper erzeugt für jeden Knoten eine Wahrscheinlichkeit, mit der dieser Knoten Teil des Maximum Independent Sets ist. Aus diesen Wahrscheinlichkeiten wird anschließend ein Independent Set erzeugt.

Basierend auf diesem Vorgehen wurde im Rahmen dieser Projektarbeit versucht ein neuronales Netz zu trainieren, welches ein möglichst kleines Vertex Cover auf einem Graphen erkennt. Verwendet wird hierzu ein Multi Layer Perzeptron (MLP), welches mit Embeddings zu allen Knoten der Trainingsgraphen trainiert wird. Dieser Ansatz hat sich leider nicht als erfolgreich erwiesen.

2 Vorgehen im Paper [3]

Im Paper wird ein Graph Convolutional Network (GCN) darauf trainiert, ein Maximal Independent Set (MIS) zu finden. Als Aktivierungsfunktion wird ReLU für alle Layer, bis auf die letzte verwendet. Hier kommt Sigmoid zum Einsatz, um eine Wahrscheinlichkeitsverteilung zu erhalten. Die erste Layer H^0 wird mit Einsen initialisiert. Die darauf folgenden Layer werden mit folgender Formel berechnet:

$$H^{l+1} = \sigma(H^l\theta_0^l + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}H^l\theta_1^l)$$

l bezeichnet die aktuell betrachtete Layer. θ_0^l und θ_1^l stellen zu trainierende Gewichtungen dar, A beschreibt die Adjazenzmatrix, D die Degree Matrix des Graphen. Der Loss wird während des Trainings über die binäre Kreuzentropiefunktion berechnet. Dieses neuronale Netzwerk liefert als Ergebnis einen Vektor über alle Knoten des Graphen zurück. Dieser Vektor enthält für jeden Knoten eine Wahrscheinlichkeit, mit welcher er Teil einer idealen Lösung für MIS ist. Das MIS wird daraus über einen Greedy Algorithmus berechnet, der zunächst den Knoten mit der höchsten Wahrscheinlichkeit zur Lösungsmenge hinzufügt. Daraufhin wird dieser Knoten, sowie alle seine Nachbarn markiert. Anschließend wird der Knoten mit der höchsten Wahrscheinlichkeit, der noch nicht markiert ist zur Lösungsmenge hinzugefügt. Daraufhin wird auch dieser Knoten und alle seine Nachbarn markiert. Dieses Verfahren wird so lange wiederholt, bis alle Knoten markiert sind.

Dieses Verfahren kommt an seine Grenzen, wenn es mehr als eine ideale Lösung gibt. Mit dem oben beschriebenen Verfahren werden verschiedene Lösungen gemischt und die vom neuronalen Netz generierte Wahrscheinlichkeitsverteilung wird unbrauchbar. Die Autoren beheben dieses Problem, indem mehrere Wahrscheinlichkeitsverteilungen generiert werden. Es wird dazu allerdings ein neues Verfahren zum Erstellen der Lösungsmenge gebraucht. Hier wird nun zufällig eine der generierten Wahrscheinlichkeitsverteilungen ausgewählt und auf dieser der oben beschriebene Greedy Algorithmus so lange ausgeführt, bis der Knoten, der aktuell die höchste Wahrscheinlichkeit hat bereits markiert wurde. Die so entstandene unvollständige Lösung wird in eine Queue eingefügt. Das wird nun mit den restlichen Wahrscheinlichkeitsverteilungen wiederholt. Im nächsten Durchgang wird zufällig eine unvollständige Lösung ausgewählt und weiter bearbeitet. Das geschieht so lange, bis für jede Wahrscheinlichkeitsverteilung eine vollständige Lösungsmenge generiert wurde. So entsteht eine Baumstruktur, mit einer leeren Lösungsmenge als Wurzel und vollständigen Lösungen als Blätter des Baums. Die übrigen Knoten dieses Baumes stellen unvollständige Lösungen dar.

3 Aufbau MLP

Im Rahmen dieses Projektes wurde versucht dieses Verfahren mithilfe eines einfachen MLPs zu implementieren. MLPs bekommen als Eingabedaten Vektoren. Daher ist es für ein MLP schwierig die Struktur eines Graphen zu erfassen. Um dieses Problem zu lösen, wurde zu allen Knoten der Eingabegraphen mithilfe von `node2vec` ein Embedding generiert [1]. `node2vec` berechnet für jeden Knoten v eines Graphen einen Vektor aus Werten, welche die strukturellen

Eigenschaften des Graphen beschreiben. Das wird erreicht, indem ausgehend von v eine bestimmte Anzahl von zufälligen Wegen durch den Graphen generiert wird. `node2vec` generiert für jeden Knoten 64 Werte, die aus 200 Wegen von je 30 Kanten Länge generiert werden. Aus diesen Wegen wird anschließend ein Vektor aus Zahlen für v generiert. Dieser Vektor dient nun als Eingabe für das MLP. Das MLP besteht aus insgesamt drei Layern. Die erste Layer hat eine Inputdimension von 64, entsprechend den 64 Werten, die `node2vec` für jeden Knoten berechnet hat. Die erste Hidden Layer hat 128 Input Features, die Output Layer hat eine Inputdimension von 64 und eine Outputdimension von 1, entsprechend der Wahrscheinlichkeit, dass der betrachtete Knoten Teil des minimalen Vertex Covers ist. Als Aktivierungsfunktion wird für alle Layer mit Ausnahme der letzten ReLU verwendet. Die letzte Layer nutzt Sigmoid, um eine Wahrscheinlichkeitsverteilung zu erhalten.

Als Trainingsgraphen wurden zunächst einige Beispielgraphen des PACE Wettbewerbs 2019 [4] verwendet. Hieraus konnten allerdings nur ca. 30 Graphen verwendet werden, für die anderen Graphen dauerte es zu lange die Embeddings zu berechnen. Für diese 30 Graphen wurde mithilfe des Tools `vc-solver` [2], welches der Sieger der PACE 2019 ist, ein minimales Vertex Cover für diese 30 Graphen erstellt. Aus den Embeddings und dem berechneten Vertex Cover als Labels wurde im nächsten Schritt ein Dataframe erstellt, welches für das Training des neuronalen Netzes eingelesen wurde. Trainiert wurde für 200 Epochen bei einer Lernrate von 1^{-3}. Als Lossfuktion wurde die binäre Kreuzentropie verwendet. Für diese Trainingdsaten lernte das MLP schnell die Labels auswendig.

Als Testgraphen wurden mithilfe von `networkit` [5] 100 G(n, p) Graphen mit jeweils 150 Knoten und einer Kantenwahrscheinlichkeit von 5% generiert. Das neuronale Netz schaffte es allerdings nicht, für diese Graphen gültige Vertex Cover zu generieren. Der in Kapitel 4 beschriebene Greedy Algorithmus lieferte für die Wahrscheinlichkeitsverteilung, die vom MLP berechnet wurde durchweg schlechtere Ergebnisse, als der Greedy Algorithmus auf Basis des Grades der Knoten.

Das liegt vermutlich an zwei Faktoren: zum einen wurde mit sehr wenigen Trainingsgraphen trainiert, zum anderen sind diese Trainingsgraphen grundlegend anders, als die Testgraphen. Es braucht also mehr Trainingsgraphen, sowie Testgraphen, die von der Struktur ähnlich zu den Trainingsgraphen sind. Außerdem ist ein G(n, p) Graph relativ gleichförmig strukturiert, da von jedem Knoten zu jedem anderen Knoten mit derselben Wahrscheinlichkeit eine Kante generiert wird. G(n, p) Graphen sind daher für diese Aufgabe ohnehin nicht gut geeignet.

Als neue Trainings- und Testgraphen wurden daher aus dem Graphen `road-germany-osm` aus dem Network Repository [6] zufällig 1000 Teilgraphen mit jeweils 150 Knoten herausgeschnitten. Im nächsten Schritt wurden für diese Graphen Vertex Cover und Embeddings berechnet, daraus ein Dataframe generiert und damit das MLP trainiert. Allerdings schien das Netzwerk hier nicht zu lernen. Vor dem Beginn des Trainings wurde für die erste Epoche ein Loss von ca. 0.69 berechnet, nach 200 Epochen sank der Loss auf 0,58. Hier stagniert der Loss, ein niedriger Loss lässt sich auch mit mehr Epochen nicht erreichen.

Eine Erklärung für dieses Verhalten könnte sein, dass die Strukturdaten, die von `node2vec` erfasst werden, nicht ausreichen, um ein möglichst kleines Vertex Cover finden zu können. Ein Versuch dieses Problem zu lösen besteht darin, statt Embeddings zu jedem Knoten, andere Features für jeden Knoten zu verwenden. Hier wurde mithilfe der Python Bibliothek `networkx` für jeden Knoten der Grad, der Clusterkoeffizienten, die Betweenness Centrality, die Closeness Centrality und der Page Rank betrachtet. Mit diesen Features ergaben sich allerdings dieselben Probleme wie mit Embeddings.

Um zu verifizieren, dass das Problem an den Trainingsdaten liegt, wurde zusätzlich ein GCN mit Vorbild des GCNs aud dem Paper mit denselben Trainingsgraphen trainiert. Auch dieses Netzwerk macht beim Training keine Fortschritte.

4 Aufbau Suche nach Vertex Covers

Zum Verifizieren der Vertex Cover, die vom MLP generiert wurden, werden die Wahrscheinlichkeiten für jeden Knoten in eine Datei geschrieben. Diese wird von einer Rust Funktion eingelesen, und gemeinsam mit dem dazu gehörigen Graphen, dem Namen des Graphen und dem Grad zu jedem Knoten des Graphen in eine Datenstruktur geschrieben. Anschließend wird mithilfe eines Greedy Algorithmus auf Basis der Grade der Knoten ein Vertex Cover generiert. Hierzu werden die Knoten absteigend nach Grad sortiert, und so lange Knoten zur Lösungsmenge hinzugefügt, bis sich ein gültiges Vertex Cover ergibt.

Eine zweite Lösung wird mit einem ähnlichen Greedy Algorithmus berechnet, indem die Knoten absteigend nach Wahrscheinlichkeit sortiert werden. Dann werden wieder in Reihenfolge so lange Knoten zur Lösungsmenge hinzugefüht, bis ein gültiges Vertex Cover entsteht.

Die größe der Vertex Cover, die nach beiden Verfahren berechnet wurde, wird anschließend ausgegeben, sodass beurteilt werden kann, welches Verfahren das kleinere Vertex Cover findet.

5 Fazit

Das fehlgeschlagene Training von sowohl GCN, als auch MLP legt nahe, dass die verwendeten Trainingsgraphen überhaupt nicht geeignet sind. Alternativ könnte es ein Problem darstellen, dass für jeden Graphen nur ein Vertex Cover generiert wurde. Dieses Vertex Cover ist zwar ein minimales, es kann allerdings noch weitere genauso große Vertex Cover geben, die das neuronale Netz nicht kennt. Diese werden im Training als falsch erkannt und das Netzwerk lernt so nicht. Auch werden nicht, wie im Paper beschrieben mehrere Lösungen generiert, wodurch mehrere Lösungen gemischt werden könnten.

Um diese Probleme zu lösen, müsste bestenfalls mit allen idealen Vertex Covern und mehreren Lösungen trainiert werden. Auch könnte es sich lohnen andere Trainingsgraphen zu verwenden. Die hier verwendeten Trainingsgraphen sind alle Teilgraphen von einem großen Graphen, welcher das deutsche Straßennetz repräsentiert. Eventuell sind auch diese Graphen zu homogen, um aus ihnen etwas lernen zu können.

Literatur

[1] Aditya Grover und Jure Leskovec. *Node2vec: Scalable Feature Learning for Networks*. 3. Juli 2016. arXiv: 1607.00653 [cs, stat]. URL: http://arxiv.org/abs/1607.00653 (besucht am 29. 09. 2024). Vorveröffentlichung (siehe S. 2).

[2] Demian Hespe u. a. "WeGotYouCovered: The Winning Solver from the PACE 2019 Challenge, Vertex Cover Track". In: *2020 Proceedings of the SIAM Workshop on Combinatorial Scientific Computing*. SIAM, 2020, S. 1–11 (siehe S. 3).

[3] Zhuwen Li, Qifeng Chen und Vladlen Koltun. *Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search*. 24. Okt. 2018. arXiv: 1810.10659 [cs, stat]. URL: http://arxiv.org/abs/1810.10659 (besucht am 10. 07. 2024). Vorveröffentlichung (siehe S. 1).

[4] M. Ayaz Dzulfikar, Johannes K. Fichte und Markus Hecher. *PACE2019: Track 1 - Vertex Cover Instances*. Zenodo, 29. Juli 2019. DOI: 10.5281/ZENODO.3354609. URL: https://zenodo.org/record/3354609 (besucht am 29. 09. 2024) (siehe S. 3).

[5] *NetworKit Graph Generators*. URL: https://networkit.github.io/dev-docs/notebooks/Generators.html (besucht am 16. 01. 2024) (siehe S. 3).

[6] Ryan A. Rossi und Nesreen K. Ahmed. "The Network Data Repository with Interactive Graph Analytics and Visualization". In: *AAAI*. 2015. URL: https://networkrepository.com (siehe S. 4).