
DEEP GRAPH LIBRARY - RAPORT TECHNICZNY

Zawalska Justyna

Wydział Informatyki, Elektroniki i Telekomunikacji
Akademia Górniczo-Hutnicza
Kraków
jzawalska@student.agh.edu.pl

Gędlek Paweł

Wydział Informatyki, Elektroniki i Telekomunikacji
Akademia Górniczo-Hutnicza
Kraków
gedlek@student.agh.edu.pl

Pęczek Paweł

Wydział Informatyki, Elektroniki i Telekomunikacji
Akademia Górniczo-Hutnicza
Kraków
peczek@student.agh.edu.pl

ABSTRAKT

Przetwarzanie grafów stanowi obecnie istotną część całości przetwarzania danych. Ma to związek z elastycznością grafu jako modelu przedstawiania wiedzy. W związku z rosnącym zapotrzebowaniem na narzędzia do wydajnego i skutecznego pozyskiwania informacji ze struktur grafowych widoczna jest tendencja do poszukiwania nowych metod udoskonalających obecny stan wiedzy z tego zakresu. Jednym z nowatorskich sposobów przetwarzania grafów jest wykorzystanie sieci neuronowych. Jako skutek upowszechnienia tej idei powstają biblioteki pozwalające na zastosowanie osiągnięć z tej dziedziny do rozwiązania praktycznych problemów na szeroką skalę. W ramach tego dokumentu przedstawiamy owoce eksploracji biblioteki Deep Graph Library (DGL) plasującej się w wyżej wymienionej kategorii. Naszym wkładem jest zbadanie jej możliwości i ograniczeń poprzez wykonanie serii eksperymentów będących próbą zastosowania DGL do rozwiązania praktycznych problemów z zakresu przetwarzania grafów.

1 Struktura raportu

Organizacja niniejszego dokumentu jest następująca. Sekcja 2 zawiera opis istoty działania biblioteki DGL [1]. W sekcji 3 znajduje się opis istotnych elementów struktury biblioteki i jej ocena. Sekcja 4 prezentuje wymagania techniczne. W dalszej kolejności, sekcja 5 zawiera podsumowanie eksperymentów, które zostały przeprowadzone z wykorzystaniem biblioteki DGL [1], ze szczególnym uwzględnieniem aspektów związanych z użytecznością różnych sposobów reprezentacji grafów oraz analizą jakości rozwiązań zaproponowanych przez twórców DGL [1]. Całość raportu dopełniają wnioski wyciągnięte z przeprowadzonych badań, które są zaprezentowane w sekcji 6.

2 Wstęp

W naszym projekcie użyliśmy biblioteki DGL w wersji 0.43, można więc założyć, że w pewnym stopniu niektóre jej elementy są jeszcze w fazie eksperymentalnej. Świadczy o tym chociażby fakt, że w tej wersji API próbkowania w heterografach zostało wprowadzone jako experimental, a także dopiero od tej wersji został dodany pełny support dla biblioteki Tensorflow.

Twórcy biblioteki DGL postanowili poddać ją testom wydajnościowym oraz skalowania. Używając funkcji tensorowych z backendowych frameworków takich jak PyTorch, MXNet, czy Tensorflow DGL w bardzo dużym stopniu optymalizuje zużycia pamięciowe i obliczeniowe, dzięki swoim własnym kernelom. Poniżej prezentujemy porównanie do innego podobnego pakietu PyTorch Geometric (PyG). W tabeli zostały umieszczone czasy w sekundach dla 200 epok oraz zużycie pamięci w GB.

Dataset	Model	Accuracy	Time		Memory	
			PyG	DGL	PyG	DGL
Cora	GCN	81.31 ± 0.88	0.478	0.666	1.1	1.1
	GAT	83.98 ± 0.52	1.608	1.399	1.2	1.1
CiteSeer	GCN	70.98 ± 0.68	0.490	0.674	1.1	1.1
	GAT	69.96 ± 0.53	0.674	1.399	1.3	1.1
PubMed	GCN	79.00 ± 0.41	0.491	0.690	1.1	1.1
	GAT	77.65 ± 0.32	1.946	1.393	1.6	1.1
Reddit	GCN	93.46 ± 0.06	OOM	28.6	OOM	11.7
Reddit-S	GCN	N/A	29.12	9.44	15.7	3.6

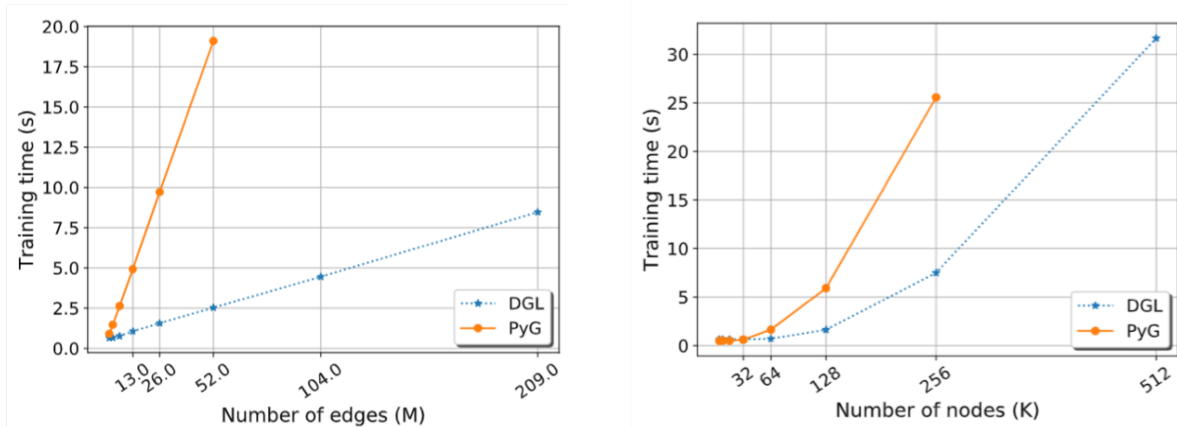


Figure 1: Porównanie czasów dla obu bibliotek

W kwestii skalowalności biblioteka DGL wygląda naprawdę dobrze na tle konkurencji. W dużym stopniu wykorzystuje możliwości obliczeń na wielu GPU zarówno na jednej maszynie jak i na klastrach obliczeniowych, w celu przyspieszenia szybkości treningu.

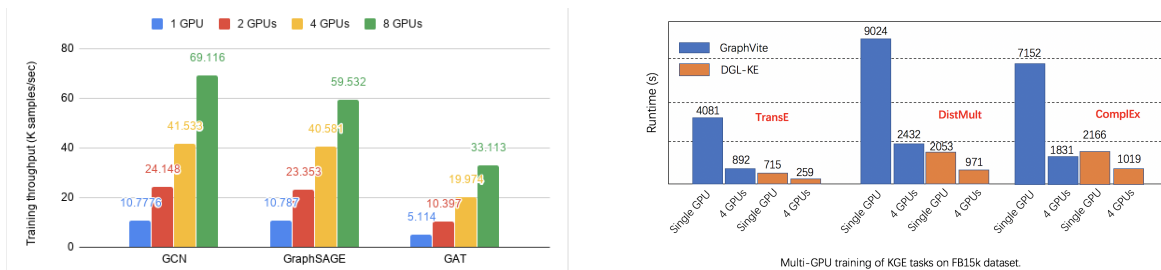


Figure 2: Skalowalność DGL na wielu GPU oraz porównanie skalowalności na tle innej biblioteki

DGL zapewnia szeroki zakres konwersji pomiędzy strukturami danych a grafami. Graf można wygenerować:

- za pomocą: biblioteki NetworkX
- z tensora pochodzącego z biblioteki PyTorch
- z tablicy NumPy
- z macierzy adiacencji
- z listy krawędzi

Oprócz tego warto wspomnieć, że zarówno wierzchołkom jak i krawędziom można nadawać odpowiednie właściwości, a mianowicie w przypadku wierzchołkom mogą być to odpowiednie features używane do uczenia modelu, natomiast w przypadku krawędzi mogą to być też wagi.

DGL jest uważana za łatwą w użyciu, wydajną i skalowalną bibliotekę języka Python do głębokiego uczenia na strukturze grafu.

Główne cechy DGL:

- integracja z wybranymi platformami do uczenia głębokiego - brak kosztów migracji,
- osiągnięcie wysokiej wydajności poprzez automatyczne grupowanie obliczeń i szybkie mnożenie macierzy rzadkich,
- dobra skalowalność do wykresów z dziesiątkami milionów wierzchołków,
- przejrzyste interfejsy dostępu do wierzchołków oraz krawędzi grafu, możliwość manipulacji strukturą grafu,
- przekazywanie wiadomości (message passing) - zarówno niskopoziomowo np. przekazywanie informacji wzdłuż krawędzi i otrzymywanie ich na wybranych węzłach, jak i wysokopoziomowo np. aktualizowanie cech całego grafu.

3 Struktura biblioteki

Źródła DGL [2] podzielone są na następujące moduły:

- backend - odpowiedzialny za przygotowanie wspólnego API dla wywołań metod frameworków, które mogą być użyte jako baza dla działania DGL [1], takich jak: PyTorch [3], Tensorflow [4], MXNet [5], czy numpy [6]. Jest to klasyczne rozwiązanie dla bibliotek będących wysokopoziomowymi wrapperami na grupę bibliotek niższego poziomu stanowiących implementację zbioru wspólnych operacji dla rozwiązywania pewnego problemu. Podobną architekturę zobaczyć można w przypadku biblioteki Keras [7]. Z poziomu biblioteki DGL [1] wysokopoziomowe operacje wykonywane są za pomocą wspólnego API dla wszystkich obsługiwanych bibliotek bazowych.
- contrib - w którym znajdują się rozszerzenia przygotowane przez społeczność.
- data - gdzie umieszczone są obiekty pośredniczące w dostępie do popularnych źródeł danych - ułatwia to szybkie rozpoczęcie eksperymentów, gdyż takie rozwiązanie pozwala na pominięcie w dużej części przypadków narzutu czasowego na przygotowanie wygodnej warstwy dostępu do danych i konwersji na odpowiedni format, który jest akceptowany przez bibliotekę.
- function - w którym umieszczono kod odpowiedzialny za podstawę działania mechanizmu funkcji (w tym udf - user defined functions) operujących na elementach grafu.
- model_zoo - zawierający zestaw predefiniowanych modeli sieci neuronowych opartych na publikacjach naukowych.
- nn - który zawiera implementacje najważniejszych dla grafowych sieci neuronowych operacji (dla wspieranych frameworków bazowych).
- runtime - gdzie umieszczono obiekty sterujące wykonaniem m. in. funkcji na elementach grafu.
- sampling - zawierający kod pozwalający na próbkowanie elementów grafu.

Od strony strukturalnej biblioteka prezentuje bardzo wysoki poziom przejrzystości, który świadczy o dobrej organizacji źródeł. Niestety przyznać należy, że nie wszystko zostało przygotowane w sposób ułatwiający korzystanie. Podczas instalacji pakietu z pip zauważyć można, że zależności do bibliotek bazowych nie zostają zainstalowane. Wymaga to wiedzy odnośnie wersji tych bibliotek, które są wymagane do poprawnej pracy DGL [1]. Co więcej podczas testów wystąpiły problemy ze współpracą ze współdzielonymi bibliotekami CUDA [8], który udało się rozwiązać przez zainstalowanie (w teorii) nieodpowiedniej wersji pakietu DGL [1].

Na poziomie struktury interfejs głównych obiektów udostępnianych przez DGL [1] jest stosunkowo prosty i czytelny. Biblioteka udostępnia klasy reprezentujące graf (DGLGraph będąca klasą bazową oraz DGLHeteroGraph dla grafów z typowanymi wierzchołkami pozwalająca na uzyskanie podobnego ggrafu wiedzy jak w grafowych bazach danych). Za pomocą interfejsu tych klas dokonać można szeregu operacji włącznie z transformacjami, próbkowaniem i konwersją grafu do i z różnych formatów. Na pochwałę zasługuje wprowadzenie konwersji na i z obiektów biblioteki NetworkX [9]. Na ogół udostępnione metody są zaprojektowane w sposób bardzo przemyślany. Niestety nie ma to zastosowania w każdym przypadku. Przykładem może być brak możliwości stworzenia obiektu grafu wprost z gęstej macierzy sąsiedztwa, czy konieczność stworzenia pustego obiektu celem użycia metody konwersji z macierzy rzadkiej. Rzeczą wprost niezrozumiałą jest jednak błąd który pojawia się w konwersji z i rzadką macierz sąsiedztwa. W tym wypadku odtworzenie grafu z jego własnej macierzy sąsiedztwa powoduje zamianę kierunku krawędzi na przeciwny.

Biblioteka DGL [1] prezentuje się zatem jako twór dość dobrze przemyślany, jednak wciąż posiadający sporo istotnych mankamentów utrudniających podstawowe wykorzystanie. Jednocześnie widać dość duże zainteresowanie społeczności i intensywny rozwój, co bardzo dobrze rokuje na przyszłość.

4 Wymagania techniczne

- DGL w wersji **0.4.x** współpracuje z następującymi systemami operacyjnymi:
 - Ubuntu 16.04
 - macOS X
 - Windows 10
- Rekomendowana wersja języka to Python 3.5 lub nowszy (biblioteka nie była testowana dla wersji niższych i może nie być z nimi kompatybilna).
- Zalecana instalacja przy użyciu systemu zarządzania pakietami *pip* lub *conda*. Istnieje także możliwość instalacji poprzez pobranie plików źródłowych z GitHub [2].

5 Opis eksperymentów

Jako podstawa dla niniejszego raportu został wykonany szereg eksperymentów prezentujących praktyczną użyteczność biblioteki DGL [1] przy rozwiązywaniu problemów z zakresu przetwarzania grafów.

5.1 Częściowo nadzorowana klasyfikacja wierzchołków

W ramach eksperymentu, na podstawie prostej symulacji rozprzestrzeniania się wirusa zostało sprawdzone w jaki sposób użyć biblioteki DGL [1] do klasyfikacji wierzchołków przy użyciu podejścia zaprezentowanego w [10] w warunkach posiadania jedynie niewielkiej ilości etykiet. Rozwiązanie należy uznać za przydatne i dość istotne - zwłaszcza w obliczu globalnej pandemii. System badający kontakty międzyludzkie (na przykład na podstawie danych z urządzeń mobilnych), który potrafiłby określić potencjalnych chorych na bazie przeprowadzonych testów byłby bardzo cenny w wyżej wymienionych okolicznościach.

Symulacja obejmowała umieszczenie osobników w przestrzeni 2D i udostępnienie im możliwości ruchu w dowolnym kierunku o ograniczoną w ramach jednego kroku symulacji odległość. Osobniki będące w danym kroku symulacji w tym samym punkcie przestrzeni mogą wchodzić ze sobą w interakcje, które powodują ich wzajemną ekspozycję w skali $e_{o_1, o_2, x, y, t} \in [0.0; 1.0]$. Dla każdej pary osobników w danym punkcie przestrzeni i w danym punkcie czasu symulacji $(k_{o_1, o_2, x, y, t})$ zostaje wylosowana wartość $e_{o_1, o_2, x, y, t}$. Wirus, którego rozprzestrzenianie się odzwierciedla symulacja cechuje się współczynnikiem przenośności $a \in [0.0; 1.0]$ obrazującym jego zaraźliwość. Prawdopodobieństwo transmisji wirusa $P(k_{x, y, t})$ obliczne jest ze wzoru $P(k_{o_1, o_2, x, y, t}) = e_{o_1, o_2, x, y, t} * a$.

Z technicznego punktu widzenia wykonanie eksperymentu było stosunkowo proste, jako że biblioteka udostępnia dość wysoki poziom abstrakcji nad dość skomplikowanymi operacjami matematycznymi wykonywanymi w ramach trenowania grafowych sieci konwolucyjnych [10]. Zgodnie z propozycją zawartą w [10] wykorzystano ciągłą (co do wartości) macierz sąsiedztwa dla reprezentowania grafu celem odzwierciedlenia wartości $e_{o_1, o_2, x, y, t}$ już na poziomie struktury grafu, tak aby sieć neuronowa była uczona w kontekście głęboko osadzonym w strukturze projektu. Uproszczone rozwiązanie nie zakładało jednak wstrzykiwania do modelu sieci neuronowej informacji o ewolucji grafu w czasie, gdyż wymagałoby to sporej ilości pracy czysto koncepcyjnej nad architekturą modelu sieci neuronowej, która była poza zakresem działań w ramach projektu. Z racji przyjęcia zaprezentowanego sposobu przedstawienia problemu najbardziej właściwa metoda reprezentacji grafu to macierz sąsiedztwa (w przypadku dużych grafów w wersji rzadkiej), a w dalszej kolejności lista krawędzi (wraz z ich wagami) pozwalająca w sposób stosunkowo łatwy (jednak wymagający kosztu $O(E)$) wygenerować rzadką macierz sąsiedztwa. Lista wierzchołków może również być przekształcona w macierz sąsiedztwa dodatkowym kosztem $O(E)$ dla uzyskania macierzy rzadkiej. Etykiety węzłów wygodnie jest przechowywać w liście opisującej cechy każdego z nich, niezależnie od reprezentacji (informacja ta może być dołączana naturalnie w przypadku listy wierzchołków).

Wspomniany już wysoki poziom abstrakcji biblioteki DGL [1] umożliwił sprawne wykonanie eksperymentu i jest cechą, którą należy rozpatrywać pozytywnie. Należy jednak pamiętać, że zbyt duży poziom abstrakcji w ogólnym przypadku powodować może utratę kontroli nad przebiegiem eksperymentu i przynieść skutki odwrotne do zamierzonych. Pod tym względem obawiać się można problemów analogicznych do tych, które występowały w bibliotece Keras [7]

na początkowych etapach jej rozwoju (jak i wówczas kiedy była już oficjalnie częścią TensorFlow [4]). Problemy te objawiały się m. in. pojawiającymi się zakleszczeniami, czy wyciekami pamięci. Zauważyć jednak należy, że w przypadku DGL [1] z wykorzystaniem biblioteki PyTorch [3] możliwa jest manualna kontrola procesu uczenia, która może niwelować szanse wystąpienia w/w problemów. Pomimo przedstawionych obaw. w ogólności działanie DGL [1] w czasie eksperymentu ocenić należy bardzo dobrze.

5.2 Wykrywanie wspólnot (Community Detection)

Wykrywanie wspólnot w sieci połączeń polega na odnajdywaniu podgrup odznaczających się wspólnymi cechami. Wspólnoty obejmują podgrafy sieci z wysoką liczbą połączeń pomiędzy wierzchołkami tego samego podgrafu i relatywnie mniejszą liczbą połączeń z wierzchołkami innych podgrafów. Struktura sieci nie wykazuje jednorodności, a raczej cechuje ją podział na klasy z wyższym prawdopodobieństwem połączeń między węzłami tej samej klasy niż między węzłami różnych klas [11]. Dla danej sieci reprezentowanej przez graf $G = (V, E)$, gdzie V jest zbiorem wierzchołków, a E zbiorem krawędzi, struktura wspólnoty to taki podział wierzchołków V na zbiory $C = C_1, \dots, C_k$ tak, że każdy $C_i, 1 \leq i \leq k$ wykazuje strukturę wspólnoty reprezentujące grupy węzłów.

Do przykładów takich sieci można zaliczyć m.in. sieci społecznościowe, gdzie połączenia pokazują powiązania między osobami, systemy rekomendujące, w których połączenia pokazują interakcje między klientami a produktami i procesy analizy chemicznej, w której związki są modelowane jako połączenia atomów i wiązań [12]. Zrozumienie struktury sieci połączeń i wykrywanie wspólnot ma kluczowe znaczenie dla wydobycia przydatnych informacji. Dzięki poznaniu zależności pomiędzy poszczególnymi komponentami można wykorzystać posiadaną wiedzę np. w marketingu docelowym czy naukach o naturze.

Biorąc pod uwagę, że interakcje między klientami a produktami są dobrym przykładem sieci umożliwiających wykrywanie wspólnot eksperyment został przeprowadzony na zbiorze danych Polbooks [13]. Zbiór ten reprezentuje sklasyfikowany zestaw książek o tematyce polityki USA. Został on stworzony na podstawie danych z serwisu Amazon.com zgodnie z powiązaniem: „klienci, którzy kupili tę książkę, również kupili te książki”. Składa się on z 105 książek i 441 powiązań pomiędzy nimi.

Zadaniem eksperymentu było sprawdzenie czy biblioteka DGL, przy użyciu grafowych sieci konwolucyjnych i uczenia częściowo nadzorowanego, nadaje się do wykrywania wspólnot w danych o strukturze grafowej. Aby to zweryfikować stworzono dwa rozwiązania: pierwsze z małym podzbiorem zbioru Polbooks (12 książek i 25 połączeń między nimi, zachowując proporcje odpowiednich kategorii) oraz drugie z całością danych. Pierwszym krokiem było ustalenie jakie parametry GCN będą odpowiednie dla pierwszego rozwiązania, żeby zastosować je w większej skali (ze względu na to, że łatwiej obserwować trenowanie danych na mniejszej ich ilości). Drugim było zastosowanie wytworzonego modelu dla całości danych.

Niestety można mieć mieszane odczucia jeśli chodzi o korzystanie z tej biblioteki w celu wykonania tego zadania. Z jednej strony nie było problemów z integracją biblioteki PyTorch, konwersją grafu do i z obiektu biblioteki NetworkX oraz obsługą danych grafowych. Z drugiej jednak napotkano dość duże problemy z dobraniem prawidłowych parametrów modelu sieci, szczególnie dotyczących liczby warstw ukrytych oraz ich rozmiarów. Resumując, wydaje się że DGL to dobra biblioteka do realizacji zadania jakim jest wykrywanie wspólnot lecz metody jakie są przez nią udostępniane zmuszają do spędzenia dużej ilości czasu na dostrajaniu parametrów sieci, co nadaje to zastosowanie bardziej badawczo-naukowe niż gotowe do użycia w produkcji.

5.3 Wykorzystanie Heterografów w uczeniu maszynowym

Według publikacji 'A Survey of Heterogeneous Information Network Analysis' [14] ważnym elementem dzisiejszych systemów informatycznych jest fakt, że składają się z wielu komponentów stale komunikujących się między sobą, a większość współczesnych naukowców modeluje je za pomocą sieci homogenicznych. Tego typu sieci nie pozwalają na rozróżnianie różnych typów obiektów (wierzchołków w grafie) czy też różnych rodzajów połączeń (krawędzi w grafie) w sieciach. Dlatego też coraz więcej badaczy rozpatruje bardziej skomplikowane połączenia jako sieci heterogeniczne. Sieci informacyjne zbudowane z heterografów mogą zawierać bogatszą strukturę oraz informacje semantyczne, które zapewniają wiele możliwości rozwoju analizy specyficznego rodzaju danych.

Biblioteka DGL wspiera obliczenia na grafowych sieciach neuronowych z użyciem klasy heterogenicznych grafów wraz z odpowiednim API. W jej ramach możemy tworzyć heterografy między innymi z połączenia kilku grafów, grafów dwudzielnych czy też listy relacji. Możliwa jest również konwersja w drugą stronę na sieć NetworkX lub na homograf.

Ciekawymi przykładami grafów heterogenicznych, które zostały poruszone w ramach dokumentacji DGL [1] niech będą:

- Citation graph (graf cytowań) - Opierając się na datasetcie stworzonym przez The Association for Computing Machinery, który zawiera informację o publikacjach, ich autorów, miejsca, gdzie publikacje się ukazały, a także co ważne publikacje, które cytują inne. Wszystkie te informacje mogą zostać zapisane i zwizualizowane za pomocą grafów heterogenicznych. Przykładowo możemy wyróżnić następujące relacje: (P) publikacja napisana przez (A) autora, (P) publikacja opublikowana na łamach danego (V) czasopisma, oraz (P) publikacja cytująca inne.
- Recommender systems (system rekomendacji) - innym przykładem ciekawym z punktu widzenia chociażby grafowych baz danych może być system rekomendacji, gdzie występują interakcje pomiędzy użytkownikami i pewnymi elementami systemu. Jako przykład posłużymy się ratingami filmów ocenianymi przez użytkowników. Tego typu interakcje można reprezentować za pomocą heterografów - wierzchołki dwóch typów (użytkownicy i filmy) oraz krawędzie dwóch rodzajów (ocena pozytywna lub negatywna).
- Knowledge graph (graf wiedzy) - do reprezentowania relacji pomiędzy poszczególnymi osobami oraz ich rolami, a także elementów które je ze sobą łączą.

Jak już zostało wspomniane powyżej, DGL pozwala na uczenie maszynowe z użyciem grafów heterogenicznych. W naszym eksperymencie posłużyliśmy się grafem cytowań stworzonym na podstawie datasetu The Association for Computing Machinery, który zawiera ponad 2 mln różnych publikacji. Celem eksperymentu było przewidzenie z którą konferencją jest związana dana publikacja. Aby zachować charakter uczenia maszynowego częściowo nadzorowanego wybraliśmy tylko kilka spośród konferencji, aby zostały olabelowane. Do nauki modelu posłużyliśmy się relacyjnym GCN o następującym wzorze przekazywania wiadomości: $h_i^{l+1} = \sigma(\sum_{r \in R} \sum_{j \in N_r(i)} W_r^l h_j^l)$. Sieć heterogeniczna wykorzystana w tym przykładzie, trzyma w sobie wagi poszczególnych relacji, następnie specyfikujemy funkcję forward, która zawiera funkcję odpowiedzialną za agregację wag relacji oraz za ich poprawną redukcję. Tak wytrenowany model, poddajemy ewaluacji, która zwraca obiecujące rezultaty zachowując odpowiednią wydajność obliczeń.

Innym aspektem związanym z heterografami jest link prediction, czyli predykcja czy pomiędzy parą wierzchołków istnieje krawędź danego typu lub znalezienie które wierzchołki są połączone z wierzchołkiem danego typu. Eksperymentowaliśmy na zbiorze treningowym FB15K zawierającym tysiące wierzchołków i relacji pomiędzy nimi. Była to równocześnie próba przeprowadzenia obliczeń na GPU, która w bardzo dużym stopniu zoptymalizowała szybkość wykonywanych obliczeń. W rezultacie testu na dosyć ograniczonym ze względu na zasoby obliczeniowe zbiorze otrzymaliśmy rezultat na poziomie 80% skuteczności.

6 Wnioski

Reasumując jak pokazała analiza przeprowadzona w ramach tego dokumentu, biblioteka DGL jest stale rozwijającym się rozwiązaniem służącym do reprezentowania grafowych struktur danych. Mimo pewnych niedoskonałości wynikającym z niedopracowania w pełni niektórych elementów tej biblioteki, możemy ją uznać za dosyć dobry wybór do wykonania pewnych eksperymentów analizy danych. Co warto dodać wypada także lepiej na tle konkurencyjnych rozwiązań w kwestii wydajności i skalowalności. Coraz bardziej rosnące zainteresowanie społeczności naukowej tematyką przetwarzania grafów z użyciem uczenia maszynowego, może przynieść bibliotece dalszy rozwój oraz dopracowanie niektórych jej mankamentów.

Źródła

- [1] Deep Graph Library. <https://www.dgl.ai/>
- [2] Źródła Deep Graph Library. <https://github.com/dmlc/dgl/tree/0.4.x>
- [3] PyTorch. <https://pytorch.org/>
- [4] TensorFlow. <https://www.tensorflow.org/>
- [5] MXNet. <https://mxnet.apache.org/>
- [6] numpy. <https://numpy.org/>
- [7] Keras. <https://keras.io/>
- [8] CUDA. <https://developer.nvidia.com/cuda-zone>
- [9] NetworkX. <https://networkx.github.io/>
- [10] Kipf T., Welling M.: Semi-Supervised Classification with Graph Convolutional Networks, *arxiv:1609.02907*, 2016
- [11] Hric D., Darst R. K., Fortunato S.: Community detection in networks: Structural communities versus ground truth, *arXiv:1406.0146*, 2014
- [12] Khan B. S., Niazi M. A.: Network Community Detection: A Review and Visual Survey, *arXiv:1708.00977*, 2017
- [13] Polbooks dataset. <http://www-personal.umich.edu/~mejn/netdata/polbooks.zip>
- [14] Heterogenous Graph. <https://arxiv.org/pdf/1511.04854.pdf>