DEEP GRAPH LIBRARY - RAPORT TECHNICZNY

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

Gędłek 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 cześć całości przetwarzania danych. Ma to związek z elasycznością 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 powstawają biblioteki pozwalające na zastosowanie osiągnięć z tej dziedziny do rozwiąznia 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

TODO + napisać jakiej wersji DGL dotyczy! + oprzeć się na GH - jest tam wiele informacji o wydajności, skalowaniu itp - warto to wrzucić. Warto wrzucić ogólne konkluzje z notebooka 00_DGLIntroduction.ipynb odnośnie sposobu reprezentacji grafu itp.

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

Główne cechy DGL:

- integracja z wybranymi platformami do uczenia głębokiego - brak kosztów migracji,

- osiągnie 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 rowziązywania pewnego problemu. Podobną architekturę zobaczyć można w przypakdu 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 rowziązanie 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 podstawe działania mechanizmu funkcji (w tym udf - user defined functions) operujących na elementach grafu.
- model_zoo - zawierający zestaw predefiniowanych modeli sieci neuronowych opertych na publikacjach naukowych.
- nn - który zawiera implementacje najważniejszych dla grafowych sieci neuronowych operacji (dla wspieranych frameworków bazowych).
- runtime - gdzie umieszczono oniekty 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, ktróry 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 głownych obiektów udostępnianych przez DGL [1] jest stosunkowo prosty i czytelny. Biblioteka udostępnia klasy reprezentujący graf (DGLGraph będąca klasą bazową oraz DGLHeteroGraph dla grafów z typowanymi wierzchołkami pozwalająca na uzyskanie podobnego gfrafu wiedzy jak w grafowych bazach danych). Za pomocą interfejsu tych klas dokonać można szeregu operacji włącznie z transformacjami, próbkowaniem i konwersja grafu do i z różnych formatów. Na pochwałę zasługuje wprowadzenie kownersji 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 wykonananany 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 podejscia zaprezentowanego w [10] w warunkach posiadania jedynie niewielkiej ilosci etykiet. Rozwiązanie należy uznać z 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, kóre 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 stosunkwo 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 rowzoju (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 $C = C_1, ..., C_k$ 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 podstwie 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 wartw 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.

6 Wnioski

TODO

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*