
DEEP GRAPH LIBRARY - RAPORT TECHNICZNY

Zawalska Justyna

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

Gędleć 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. W dalszej kolejności, sekcja 4 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 5.

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.

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 reprezentujący 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 Opis eksperymentów

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

4.1 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 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