FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Prieskum techník grafových neurónových sietí
Student:	Barbara Bobeničová
Vedoucí:	Ing. Miroslav Čepek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Seznamte se s metodami grafových neuronových sítí pro klasifikaci uzlů a grafů. Prozkoumejte dostupné implementace knihoven pro grafové neuronové sítě a vizualizace grafů. Po dohodě s vedoucím práce vyberte vhodná data (např. z repozitáře OGB nebo Papers with Code). Porovnejte výsledky vybraných metod GNN nad datasety a porovnejte dosažené výsledky s dostupnými state of the art výsledky.

Elektronicky schválil/a Ing. Karel Klouda, Ph.D. dne 17. února 2022 v Praze.

Bakalárska práca

PRIESKUM TECHNÍK GRAFOVÝCH NEURÓNOVÝCH SIETÍ

Barbara Bobeničová

Fakulta informačních technologií
Katedra aplikované matematiky
Vedúci: Ing. Miroslav Čepek, Ph.D.
16. februára 2023

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam ukážok kódu

Predovšetkým chcem poďakovať môjmu vedúcemu, Ing. Miroslavovi Čepkovi, Ph.D., ktorý mi bol po celú dobu písania tejto práce veľmi nápomocný. Ďalej by som chcela poďakovať mojej rodine, ktorá ma podporovala po celú dobu štúdia, aj keď to nešlo vždy ako po masle. Tiež by som chcela poďakovať Tomášovi Bilákovi za veľkú podporu počas štúdia a pri písaní tejto práce. A všetkým kamarátom, ktorí ma vždy počas štúdia podržali a podporovali.

Vyhlásenie

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona

V Praze dne 16. februára 2023 .....

Abstrakt

Táto práca sa venuje rozboru metód grafových neurónových sietí pre klasifikáciu vrcholov a grafov. Skúma súčasné knižnice na prácu s grafovými neurónovými sieťami ako StellarGraph, PyTorch Geometric a DGL. Na vybraných datasetoch z Open Graph Benchmark sú otestované a porovnané grafové algoritmy Graph Convolutional Networks, GraphSAGE a Graph Attention Networks. Dosiahnuté výsledky sú porovnané so state of the art výsledkami.

Klíčová slova grafové neurónové siete, klasifikácia vrcholov, StellarGraph, PyTorch Geometric, DGL, GraphSAGE, GCN, GAT, Open Graph Benchmark

Abstract

This thesis is dedicated to the analysis of graph neural network methods for the nodes and graph classification. Explores current libraries for working with graph neural networks such as StellarGraph, PyTorch Geometric and DGL. The graph algorithms Graph Convolutional Networks, GraphSAGE and Graph Attention Networks are tested and compared on selected datasets from the Open Graph Benchmark. The achieved results are compared with the state of the art results.

Keywords graph neural network, node classification, StellarGraph, PyTorch Geometric, DGL, GraphSAGE, GCN, GAT, Open Graph Benchmark

Zoznam skratiek

CNN	Convolutional neural network
CPU	Central processing unit
DGL	Deep Graph Library
GAT	Graph Attention Networks
GCN	Graph Convolutional Networks
GNN	Graph neural network
GPU	Graphics processing unit
OGB	The Open Graph Benchmark
PyG	PyTorch Geometric

Úvod

Neurónové siete sa v posledných rokoch tešia veľkej popularite. Používajú sa celkom bežne na riešenie najrôznejších problémov a obyčajný človek o tom nemusí ani vedieť. Bežný človek obvykle nevie na čo sa neurónové siete využívajú, ani ako fungujú, ale určite už pojem neurónové siete počul.

Zjednodušene by sa dalo povedať, že je to model, ktorý natrénujeme na veľkom množstve dát a vďaka týmto trénovacím dátam sa môžeme pokúsiť niečo predikovať. Využitie je veľmi široké, od predikovania obsahu na obrázku, rozpoznávanie textu, až po rôzne znalostné systémy. Neurónové siete sú široký pojem a preto som sa rozhodla zamerať konkrétne na grafové neurónové siete.

V mojej práci sa budem venovať metódam grafových neurónových sietí pre klasifikáciu uzlov a grafov. Preskúmam rôzne dostupné implementácie knihovní pre grafové neurónové siete a vizualizáciu grafov. Otestujem vybrané metódy na vybraných datasetoch a porovnám mnou dosiahnuté výsledky s dostupnými state of the art výsledkami.

Kapitola 1
Základné pojmy

Prvá kapitola je venovaná oboznámeniu čitateľa s problematikou strojového učenia a základnými pojmami, ktoré budú v práci ďalej využívané.

1.1 Graf

Definície v tejto podkapitole sú podľa Zhou a spol. [1]. Pod pojmom graf, označíme G, rozumieme dátovú štruktúru definovanú dvoma množinami, $G = (V, E)$, kde V reprezentuje množinu vrcholov a E reprezentuje množinu hrán. Grafy obvykle rozdeľujeme do niekoľkých kategórií:

Orientované/Neorientované grafy.
V prípade, že je hrana definovaná usporiadanou dvojicou vrcholov grafu G jedná sa o orientovanú hranu a ak sú všetky hrany orientované jedná sa o orientovaný graf, v prípade, že sa jedná o neusporiadanú dvojicu hovoríme o neorientovanej hrane, neorientovanom grafe.

Homogénne/Heterogénne grafy.
Vrcholy a hrany sú pri homogénnych grafoch rovnakého typu. Pri heterogénnych grafoch sú vrcholy a hrany rozdielnych typov.

Statické/Dynamické grafy.
Ak sa vstupné vlastnosti alebo topológia grafu môžu meniť v priebehu času, považujeme graf za dynamický, inak hovoríme o statickom grafe.

1.2 Strojové učenie

Strojové učenie nám umožňuje riešiť úlohy, ktoré by bolo príliš zložité riešiť s pevne napísaným programom. Miesto toho využíva pri riešení úloh schopnosť učiť sa z dát. Pri zisťovaní úspešnosti algoritmu strojového učenia sa zvyčajne zameriavame na to, ako úspešný bol model na dátach, ktoré pred tým nevidel. To znamená na dátach, na ktorých nebol naučený/natrénovaný a boli oddelené od dát na ktorých prebiehalo učenie.

Najčastejšie môžeme algoritmy strojového učenia rozdeliť na supervizované a nesupervizované učenie. Pri supervizovanom učení sa snažíme zo známych dát/hodnôt nejakého atribútu tento atribút pomocou modelu predikovať, respektíve zistiť na čom daný atribút závisí. Pri nesupervizovanom učení obsahuje dataset veľa atribútov, v ktorých sa je potrebné zorientovať a nájsť v datasete skrytú štruktúru. [2, Kapitola 5]

1.3 Neurónové siete

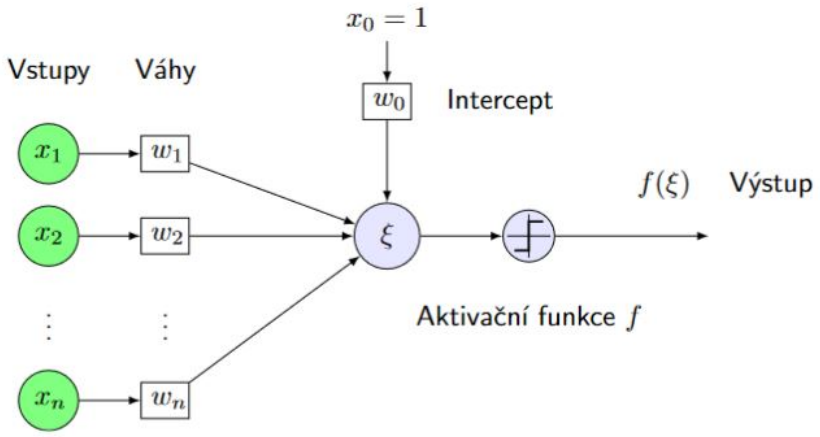
Neurónová sieť je podobná nervovej sústave v ľudskom tele. Pozostáva z elementov, takzvaných neurónov alebo perceptrónov, ktoré spolu dokážu spolupracovať na riešení rôznych úloh. Funkciu neurónovej siete by sme mohli prirovnať k funkcii mozgu, ktorý dostáva a odosiela informácie, na základe ktorých človek vykonáva určité činnosti. [3]

Neurón pozostáva zo 4 častí (Obr. 1.1). Výstup neurónu dostaneme pomocou vnútorného potenciálu, na ktorý aplikujeme *aktivačnú funkciu f*. Pomocou vstupných hodnôt a ich váh si spočítame vnútorný potenciál,

$$\xi = w_0 + \sum_{i=1}^{n} w_i xi.$$

Aplikovaním *aktivačnej funkcie* f určíme výstup neurónu, teda či je neurón aktivovaný alebo nie

$$f(x) = \begin{cases} 1 & \text{keď } \xi \geq 0, \\ 0 & \text{keď } \xi < 0. \end{cases}$$



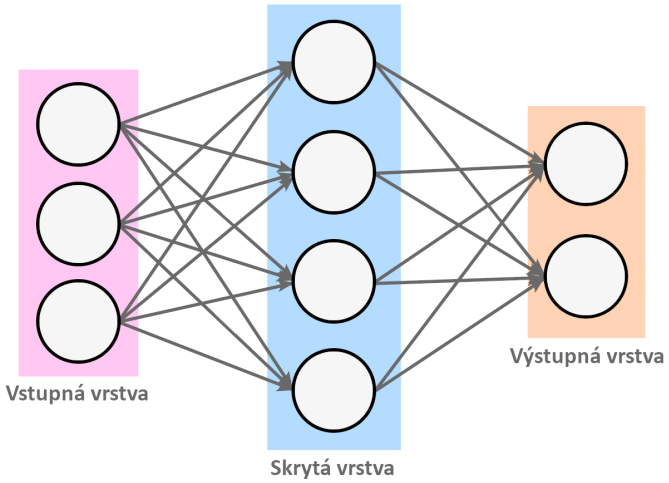
Obr. 1.1 Model umelého neurónu [4].

Výstup jedného neurónu môže byť vstupom ďalšieho neurónu, avšak neuróny v neurónovej sieti sú pospájané do acyklického grafu. Model neurónovej siete je väčšinou tvorený rôznymi vrstvami neurónov, pričom najčastejší typ je úplne prepojená vrstva. Pri tomto type vrstvy sú neuróny v rámci susedných vrstiev úplne párovo prepojené ale neuróny v rámci jednej vrstvy prepojené nie sú. Ukážka modelu takejto viacvrstvovej neurónovej siete je na obrázku 1.2.

Vstupná vrstva obsahuje vstupné dáta neurónovej siete a nepočíta sa do počtu vrstiev neurónovej siete. V prípade, že hovoríme o 1-vrstvovej neurónovej sieti, tak neobsahuje žiadnu skrytú vrstvu a vstup je rovno prepojený s výstupnou vrstvou. V skrytých vrstvách prebiehajú jednotlivé výpočty a plnia rolu atribútov pre ďalšie vrstvy. Neurónová sieť môže obsahovať viac skrytých vrstiev a čím viac ich obsahuje, tým je sieť hlbšia. Hodnoty vypočítané v skrytých vrstvách sú nakoniec prevedené vo výstupnej vrstve na reálne hodnoty použiteľné k predikcii. [4, 5]

Voľbou vhodných aktivačných funkcií je možné dosiahnuť aby bola neurónová sieť ako funkcia parametrov skoro všade diferencovateľná. Príkladom takejto aktivačnej funkcie pre skryté vrstvy, ktorá je často využívaná, je orezaná lineárna funkcia, `ReLU` (*rectified linear unit*)

$$f(\xi) = max(0, \xi) = \begin{cases} \xi & \text{pre } \xi \geq 0, \\ 0 & \text{pre } \xi < 0. \end{cases}$$



Obr. 1.2 Model viacvrstvovej neurónovej siete.

Táto funkcia nie je diferencovateľná v 0, ale je dôležité, že má v kladnom obore nenulovú deriváciu. [4, 5]

1.4 Grafové neurónové siete

V tejto podkapitole čerpám z vedeckého článku od Zhoua a spol. [1]. Grafové neurónové siete (GNN) sú typ neurónových sietí, reprezentované pomocou grafovej štruktúry. V učebnej fáze zvyčajne rozlišujeme tri úrovne:

- **Úlohy na úrovni vrcholov** zahŕňajú klasifikáciu vrcholov, regresiu vrcholov a zhlukovanie vrcholov. Cieľom klasifikácie je rozdeliť vrcholy do niekoľkých tried. Regresia predikuje spojitú hodnotu pre každý vrchol. Zhlukovanie rozdelí vrcholy do niekoľkých disjunktných skupín, pričom podobné vrcholy budú v rovnakej skupine.
- **Úlohy na úrovni hrán** sa zameriavajú na klasifikáciu hrán a predikcie spojitosti. To vyžaduje aby mal model klasifikované typy hrán alebo predikovať či sa medzi dvoma danými vrcholmi nachádza hrana.
- **Úlohy na úrovni grafov** zahŕňajú klasifikáciu grafov, regresiu grafov a párovanie grafov. Všetky tieto úlohy potrebujú model aby sa naučili reprezentáciu grafov.

Z pohľadu supervizovania (*supervision*) môžme učebnú fázu grafu rozdeliť na tri rozdielne trénovacie nastavenia:

- **Supervizované nastavenie** poskytuje na trénovanie označené (*labeled*) dáta.
- **Čiastočne supervizované nastavenie** poskytuje na trénovanie malú časť označených dát a veľkú časť neoznačených (*unlabeled*) dát.
- **Nesupervizované nastavenie** poskytuje iba neoznačené dáta.

Typický model grafovej neurónovej siete je obvykle postavený z kombinácie výpočtových modulov. Často používané výpočtové moduly sú:

- **Propagation modul.** Tento modul sa používa na šírenie informácií medzi vrcholmi, takže agregované informácie môžu zachytávať informácie o vlastnostiach a topologické informácie.

- **Sampling modul.** Používa sa pri veľkých grafoch na podporu predávania informácii medzi vrcholmi, vrstvami, či podgrafmi. Obvykle sa kombinuje s *propagation* modulom.
- **Pooling modul.** Zatiaľ čo *propagation* modul a *sampling* modul pomáhajú s prenosom informácii medzi vrstvami, *pooling* modul slúži na extrakciu informácii vyššej úrovne.

1.5 Konvolučné neurónové siete

V tejto podkapitole sú uvedené informácie o konvolúcii a konvolučných neurónových sieťach (CNN) podľa Goodfellowa a spol. [2, Kapitola 9]. CNN, sú typom neurónových sietí, ktoré sa špecializujú na spracovanie dát, ktoré majú topologické usporiadanie v tvare mriežky. Príkladom takéhoto usporiadania sú napríklad obrázky, ktorých dáta môžu byť uložené ako pixely v 2D mriežke.

CNN využívajú matematickú funkciu, konvolúciu, ktorá je špeciálnym druhom lineárnej operácie. Konvolúciu používajú namiesto všeobecného násobenia matíc aspoň v jednej zo svojich vrstiev. Definujeme ju rovnicou

$$s(t) = (x * w)(t)$$

Typicky sa operácia konvolúcie označuje znakom $*$. Prvý argument je vstup konvolúcie, v našom prípade x. Druhý argument je jadro w. Posledný parameter t označuje časový index. Výstup, $s(t)$, je niekedy označovaný aj ako mapa atribútov (*feature map*). V oblasti strojového učenia je väčšinou vstupom viacrozmerné pole, ktoré obsahuje dáta. Jadrom je viacrozmerné pole parametrov.

Veľa knižníc, ktoré využívajú strojové učenie používajú vzájomnú koreláciu (*cross-correlation*) a nazývajú to konvolúcia. Rozdiel je v tom, že pri vzájomnej korelácii neplatí komutatívnosť, keďže sa jadro neotáča.

Konvolúcia zlepšuje systém strojového učenia využitím 3 dôležitých myšlienok:

Riedka konektivita.
Pri používaní typickej neurónovej siete každá výstupná jednotka interaguje s každou vstupnou jednotkou, avšak CNN využívajú riedku konektivitu, alebo inak povedané riedku interakciu (*sparse connectivity/interactions*). Dosiahne sa to tak, že jadro bude menšie ako vstup. To znamená, že nám stačí ukladať menší počet parametrov, vďaka čomu sa zmenšujú pamäťové nároky modelu, zlepšuje sa jeho štatistická efektivita a na výpočet výstupu nám stačí menej operácií.

Zdieľanie parametrov.
Tradičné neurónové siete využívajú každý prvok z matice váh iba raz, pri výpočte výstupu vrstvy. Vďaka zdieľaniu parametrov (*parameter sharing*) môžeme využívať rovnaké parametre vo viacerých funkciách modelu. Pri CNN to znamená, že namiesto toho aby sme samostatne učili množinu parametrov pre každú polohu, naučíme len jednu množinu. Keďže je hodnota váhy aplikovanej na jeden vstup previazaná s hodnotou váhy aplikovateľnej inde, môžeme tiež povedať, že sieť má zviazané váhy (*tied weights*).

Ekvivariantné reprezentácie
V prípade konvolúcie môže konkrétna forma zdieľania parametrov spôsobiť, že vrstva bude mať vlastnosť ekvivariancie (*equivariance*). Môžeme povedať, že funkcia je ekvivariantná ak platí, že ak sa zmení vstup, výstup sa zmení rovnakým spôsobom.

Vrstva CNN obvykle prechádza tromi fázami. V prvej fáze vykoná vrstva paralelne niekoľko konvolúcií aby vytvorila množinu lineárnych aktivácií. V druhej fáze každá lineárna aktivácia prebieha nelineárnou aktivačnou funkciou. V tretej fáze sa výstup vrstvy upraví pomocou *pooling* funkcie. *Pooling* funkcia nahradí výstup siete na určitom mieste sumárnou štatistikou okolitých výstupov.

Kapitola 2
Grafové algoritmy

Táto kapitola sa zameriava na priblíženie troch grafových algoritmov, ktoré sú v práci porovnávané a testované. Konkrétne sa jedná o algoritmy Graph Convolutional Networks (GCN), GraphSAGE a Graph Attention Networks (GAT).

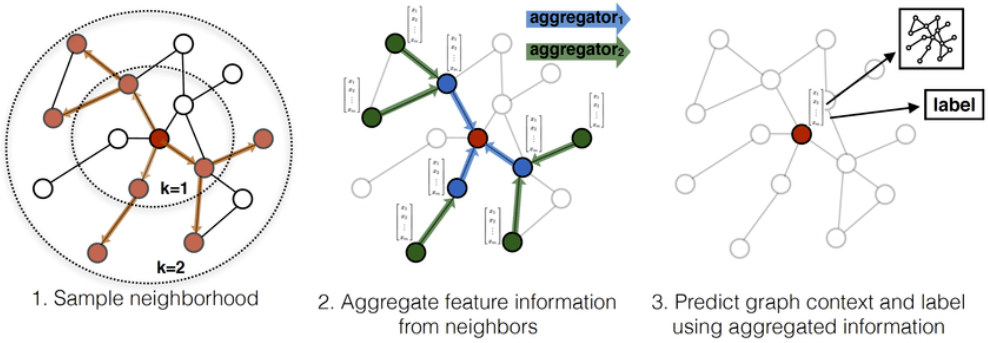
2.1 Graph Convolutional Networks

GCN sú typom CNN reprezentované formou grafovej štruktúry. Fungujú na princípe toho, že najprv operácia grafovej konvolúcie použije rovnakú lineárnu transformáciu na všetkých susedov vrcholu, potom sa použije *mean pooling* a nakoniec nelinearita. Použitím viacerých grafových konvolučných vrstiev za sebou sa môžu GCN naučiť reprezentáciu vrcholov daného grafu. Využije na to informácie zo vzdialenejších susedov vrcholu. GCN a rôzne ich varianty majú využitie napríklad pri čiastočne supervizovanej klasifikácii vrcholov, *inductive* mapovaní vrcholov (*node embedding*), predikcii prepojenia (*link prediction*). [6]

2.2 GraphSAGE

Informácie v tejto podkapitole sú čerpané z vedeckého článku od Hamiltona a spol. [7], kde algoritmus GraphSAGE (SAmple and aggreGatE) predstavujú. GraphSAGE slúži na mapovanie vrcholov v grafe a je rozšírením GCN frameworku. Pracuje na princípe využívania atribútov jednotlivých vrcholov a naučenia mapovacej funkcie aby ju bolo možné zovšeobecniť aj na ostatné vrcholy, na ktorých neprebiehalo učenie. Týmto sa GraphSAGE líši od ostatných mapovacích prístupov, ktoré využívajú faktorizáciu matíc. Vďaka tomu, že sa algoritmus učí pomocou atribútov jednotlivých vrcholov sa tak súčasne učí aj topologické usporiadanie vrcholov v grafe a rozloženie susedov spolu s ich atribútmi. Tento algoritmus môže byť použitý aj pri grafoch kde vrcholy nemajú žiadne atribúty. Namiesto trénovania mapovacieho vektora pre každý vrchol sa používa natrénovaná množina agregačných funkcií. Tieto funkcie sa učia agregovať informácie o atribútoch susedov jednotlivých vrcholov (Obr. 2.1). Každá agregačná funkcia agreguje informácie z odlišného počtu vrcholov pri prechádzaní do hĺbky od daného vrcholu.

Hlavná idea algoritmu spočíva v tom, že sa učí ako agregovať informácie z okolitých susedov nejakého určitého vrcholu, napríklad informácie o stupňoch vrcholov, alebo textové atribúty vrcholov.



Obr. 2.1 Ilustrácia vzorového a agregovaného prístupu GraphSAGE. [7]

Algoritmus 1: GraphSAGE embedding generation

Input: Graf $G(V, E)$; vstupné parametre $\{x_v, \forall v \in V\}$; hĺbka K; matice váh $W^{\mathrm{k}}, \forall k \in \{1, ..., K\}$; nelinearita σ; diferencovateľné agregačné funkcie $\mathrm{AGGREGATE}_k$, $\forall k \in \{1, ..., K\}$; funkcia susedov $N : v \rightarrow 2^V$

Output: Vektorová reprezentácia $z_v \forall v \in V$

$h_v^0 \leftarrow x_v, \forall v \in V$;
for $k = 1...K$ **do**
  for $v \in V$ **do**
    $h_{N(v)}^k \leftarrow \mathrm{AGGREGATE}_k(\{h_u^{k-1}, \forall u \in N(v)\})$;
    $h_v^k \leftarrow \sigma(\mathrm{W}^k \cdot \mathrm{CONCAT}(h_v^{k-1}, h_{N(v)}^k))$;
  end
  $h_v^k \leftarrow h_v^k / \parallel h_v^k \parallel_2, \forall v \in V$
end
$z_v \leftarrow h_v^K, \forall v \in V$;

2.2.1 Embedding generation algoritmus

Embedding generation algoritmus alebo *forward propagation* algoritmus (Algoritmus 1) predpokladá, že model už je natrénovaný, parametre sú fixné a že je natrénovaných K agregačných funkcií, ktoré agregujú informácie zo susedných vrcholov a matíc váh W^{k}, $\forall k \in \{1, ..., K\}$, ktoré sa používajú na šírenie informácií medzi jednotlivými vrstvami grafu.

Hlavná myšlienka Algoritmu 1 spočíva v tom, že vrchol v každej iterácii agreguje informácie zo susedných vrcholov. Vďaka tomu vrcholy získavajú každou ďalšou iteráciou viac a viac informácií zo vzdialenejších častí grafu.

Algoritmus 1 popisuje *embedding* proces v prípade, keď je na vstupe daný celý graf $G = (V, E)$ s vlastnosťami všetkých vrcholov $x_v, \forall v \in V$. Každý krok vonkajšieho cyklu prebieha tak, že k označuje súčasný krok a h^k označuje reprezentáciu vrcholov v tomto kroku. Najprv každý vrchol $v \in V$ agreguje reprezentáciu vrcholov, ktoré sú v jeho bezprostrednom susedstve $\{h_u^{k-1}, \forall u \in N(v)\}$, do jedného vektora $h_{N(v)}^{k-1}$. GraphSAGE ďalej zreťazí súčasnú reprezentáciu vrcholov h_v^{k-1}, s agregovaným vektorom susedov $h_{N(v)}^{k-1}$. Tento zreťazený vektor vedie cez plne prepojenú vrstvu s nelineárnou aktivačnou funkciou σ, ktorá transformuje reprezentácie, ktoré sa majú použiť v ďalšom kroku. Pre zjednodušenie zápisu sa finálna reprezentácia výstupu pri hĺbke K zapisuje ako $z_v \equiv h_v^K, \forall v \in V$.

Pri použití agregácie na reprezentáciu susedných vrcholov je na výber niekoľko rôznych agregačných architektúr (v Algoritme 1 označené ako AGGREGATE). Na rozdiel od strojového

učenia cez N-D mriežky, susedia vrcholov nemajú prirodzené usporiadanie a teda agregačná funkcia v Algoritme 1 musí pracovať s neusporiadanou množinou vrcholov. V ideálnom prípade by bola agregačná funkcia symetrická a súčasne by bola stále trénovateľná a zachovala si vysokú reprezentačnú schopnosť. Vlastnosť symetrie pri agregačnej funkcii zaisťuje, že neurónová sieť môže byť natrénovaná a aplikovaná na ľubovoľne usporiadanú množinu atribútov vrcholov susedných vrcholov. Príklady agregačných funkcií, ktoré by sa dali použiť:

Mean aggregator.
Táto agregačná funkcia využíva priemer vektorov v $h_u^{k-1}, \forall u \in N(v)$.

LSTM aggregator
V porovnaní s *mean aggergator* má výhodu vo väčšej vyjadrovacej schopnosti. *LSTM aggregator* nie je prirodzene symetrický, pretože vstupy spracúva sekvenčným spôsobom.

Pooling aggregator
Pooling aggregator je symetrický aj trénovateľný. Každý vektor susedov je nezávisle vedený cez plne prepojenú neurónovú sieť a po tejto transformácii je na agregované informácie pre množinu susedov použitá operácia *max-pooling*

$$AGGREGATE_k^{pool} = max(\{\sigma(W_{pool} h_{u_i}^k + b), \forall u_i \in N(v)\}), \tag{2.1}$$

kde *max* označuje elementárny operátor *max* a σ označuje nelineárnu aktivačnú funkciu [7].

2.3 Graph Attention Networks

Veličković a spol. [8] vo vedeckom článku predstavujú *attention-based* architektúru na klasifikáciu vrcholov na grafovo štrukturovaných dátach. Základom je vypočítanie skrytej reprezentácie každého vrcholu v grafe postupným prechádzaním všetkých susedov podľa *self-attention* stratégie. *Attention* architektúra podľa autorov vykazuje niekoľko zaujímavých vlastností:

- je efektívna, najmä vďaka paralelizovateľnosti nad množinou párov vrchol-susedia
- je ju možné aplikovať na vrcholy s rôznym stupňom dosadením ľubovoľných váh susedom
- je priamo použiteľná na problémy *inductive* učenia

Na dáta, ktoré sa dajú reprezentovať štruktúrou v podobe mriežky, ako napríklad klasifikácia obrázkov, sémantická segmentácia alebo strojový preklad je možné aplikovať riešenie pomocou CNN, ako bolo zmienené v podkapitole 1.5. Avšak dáta, ktoré nemôžeme reprezentovať pomocou mriežkovej štruktúry tiež obsahujú veľa zaujímavých úloh, ktoré by sa dali riešiť. Napríklad 3D *meshes*, štruktúra zložená z polygónov, sociálne siete, telekomunikačné siete alebo biologické siete. Takéto dáta môžu byť reprezentované formou grafov a zaoberá sa nimi GAT.

Attention mechanizmus sa štandardne používa pri riešení sekvenčných úloh. Jednou z výhod tohto mechanizmu je to, že sa sústredí na najpodstatnejšie časti vstupu pri rozhodovaní, vďaka čomu sa vie vysporiadať s premenlivou veľkosťou vstupov. Ak *attention* mechanizmus počíta reprezentáciu iba jednej sekvencie, hovorí sa o *self-attention* alebo *intra-attention*.

GAT architektúra sa skladá z *attentional* grafových vrstiev. Vstupom tejto vrstvy je množina atribútov vrcholov $\mathbf{h} = \{\overrightarrow{h}_1, \overrightarrow{h}_2, ..., \overrightarrow{h}_N\}, \overrightarrow{h}_i \in \mathbb{R}^F$, kde N je počet vrcholov a F je počet atribútov jednotlivých vrcholov. Výstupom vrstvy je nová množina atribútov vrcholov, $\mathbf{h'} = \{\overrightarrow{h'}_1, \overrightarrow{h'}_2, ..., \overrightarrow{h'}_N\}, \overrightarrow{h'}_i \in \mathbb{R}^{F'}$, ktorá môže mať rozdielnu mohutnosť F'. Na to aby bolo možné transformovať vstupné atribúty na atribúty vyššej triedy je potrebná aspoň jedna učiaca lineárna transformácia. V počiatočnom kroku je zdieľaná lineárna transformácia, ktorá je parametrizovaná maticou váh, $\mathbf{W} \in \mathbb{R}^{F' \times F}$, použitá na všetky vrcholy. V ďalšom kroku sa na

vrcholoch vykoná zdieľaný *attentional* mechanizmus $a : \mathbb{R}^{F'} \times \mathbb{R}^{F} \to \mathbb{R}$, ktorý vypočíta *attention* koeficienty

$$e_{ij} = a(W\overrightarrow{h}_i, W\overrightarrow{h}_j),$$

čo ukazuje aké dôležité sú atribúty vrcholu j, pre vrchol i. V prípade použitia *masked attention* sa vypočíta e_{ij} iba pre vrcholy $j \in \mathcal{N}_i$, kde $\mathcal{N}_i$ patrí do susedstva vrcholu i v danom grafe. Aby sa koeficienty medzi jednotlivými vrcholmi jednoduchšie porovnávali môžeme ich normalizovať naprieč všetkými j použitím funkcie softmax

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}.$$

Takto normalizované *attention* koeficienty sa používajú na počítanie lineárnej kombinácie atribútov, z ktorých vzniknú výstupné atribúty pre každý vrchol.

Attentional grafová vrstva rieši pomocou neurónových sietí niekoľko problémov, ktoré mali predchádzajúce prístupy pri modelovaní grafovo štrukturovaných dát. Napríklad, výpočtovo je veľmi efektívna vďaka tomu, že *self-attention* vrstva môže byť paralelizovateľná naprieč hranami a výpočet výstupných atribútov môže byť paralelizovateľný naprieč všetkými vrcholmi. Taktiež má model väčšiu kapacitu, čo je spôsobené tým, že model umožňuje priradiť vrcholom v rovnakej hĺbke od nejakého vrcholu rôzne dôležitosti. *Attention* mechanizmus je používaný zdieľaným spôsobom na všetky hrany grafu a teda nezávisí na priamom prístupe ku globálnej štruktúre grafu alebo atribútom všetkých vrcholov. To má za následok, že nie je potrebné aby bol graf neorientovaný a súčasne môže byť táto technika priamo použitá na *inductive* učenie, čo zahŕňa aj prípady, kedy bol model vyhodnotený na grafe, ktorý nebol vôbec použitý pri trénovaní. [8]

Kapitola 3
Vizualizácie grafov

Vizualizácia grafov je netriviálna úloha, pretože pri nich nejde len o vlastnosti prvkov hrán a vrcholov ale aj o ich štruktúru.

Existuje na ňu veľa špecializovaných nástrojov, napríklad `Gephi`. Jedná sa o *open source* softvér na analýzu grafov a sietí. Vizualizácie v gephi sú interaktívne a je v nich možné preskúmavať rôzne druhy sietí, komplexných systémov alebo grafov [9].

Sú aj vizualizácie, ktoré fungujú priamo v pythone, ale nie sú až tak interaktívne ako napríklad gephi. Takýmto python balíčkom je aj `Networkx`, ktorý poskytuje základnú štruktúru dát pre reprezentáciu rôznych druhov grafov. Na obrázku 6.3 je ukážka použitia Networkx na datasete ogbn-arxiv. Networkx vrcholy môžu byť reprezentované pomocou akéhokoľvek hašovacieho python objektu a ako hrany môže obsahovať ľubovoľné dáta. Z tohto dôvodu je možné povedať, že je veľmi flexibilný [10].

Obr. 3.1 Vizualizácia podgrafov knižnicou Networkx na datasete ogbn-arxiv.

Ďalšou možnosťou na vytváranie vizualizácií grafov je `Neo4j Bloom`. Jedná sa o grafovú databázu, pri ktorej je možné sa na grafy dotazovať. Tiež je možné animovaný graf posúvať a približovať, aby bolo lepšie vidieť nejaký konkrétnu časť, alebo upravovať vrcholy, vzťahy alebo vlastnosti [11].

Kapitola 4
Dostupné knižnice

Štvrtá kapitola sa zameriava na Python knižnice na spracovanie grafových neurónových sietí, konkrétne na StellarGraph, PyTorch Geometric a Deep Graph Library. Rozoberá základné informácie o daných knižniciach. A tiež popisuje ako sa jednotlivé knižnice používajú pri vytváraní grafovej neurónovej siete a jej trénovaní, čo je doplnené aj ukážkami kódu.

4.1 StellarGraph

StellarGraph je Python knižnica [12], ktorá využíva strojové učenie na spracovanie grafov a sietí. Ponúka *state of the art* algoritmy pre spracovanie grafov strojovým učením, čím uľahčuje objavovanie vzorov a odpovedanie na otázky ohľadom grafovo štrukturovaných dát. Vďaka tomu ju môžeme použiť pri rôznych úlohách strojového učenia, napríklad:

- **Učenie reprezentácie vrcholov a hrán aby mohli byť použité na vizualizáciu a rôzne iné úlohy strojového učenia,**
- **Klasifikácia a odvodenie atribútov pre vrcholy alebo hrany,**
- **Klasifikácia celých grafov,**
- **Predikcia prepojenia,**
- **Interpretácia klasifikácie vrcholov.**

Grafovo štrukturované dáta reprezentujú vrcholy a vzťahy medzi nimi ako hrany (alebo prepojenia) a môžu obsahovať dáta spojené s obomi ako atribúty. Napríklad graf, ktorý zobrazuje ľudí ako vrcholy a priateľstvo medzi ľudmi ako hrany môže obsahovať dáta ako vek ľudí a dátum, kedy sa určitý ľudia stali priateľmi. StellarGraph podporuje analýzu rôznych druhov grafov, napríklad:

- **homogénne (s vrcholmi a hranami jedného typu),**
- **heterogénne (s viac ako jedným typom vrcholov a/alebo hrán),**
- **znalostné grafy (extrémne heterogénne grafy s tisíckami typov hrán),**
- **grafy s dátami alebo bez dát spojených s vrcholmi,**
- **grafy s váhami hrán**

StellarGraph je postavený na TensorFlow2 a Keras high-level API, rovnako ako aj na Pandas a NumPy. Vďaka tomu je užívateľsky prívetivý, modulárny a rozšíriteľný. Základné StellarGraph algoritmy na strojové učenie grafov využívajú štandardné Keras vrstvy a scikit-learn, takže sú ľahko upraviteľné. [12]

V ukážke kódu 1 je príklad vytvorenia dvojvrstvovej grafovej konvolučnej siete.
Generátor zakóduje informácie potrebné pre vytvorenie vstupov modelu. V ukážke je použitý `FullBatchNodeGenerator` generátor pre klasifikáciu vrcholov. Parameter `method` špecifikuje metódu, ktorú chceme používať a generátor poskytne vhodné dáta konkrétne pre túto metódu, v tomto prípade pre GCN. Vrstvy budeme vytvárať pomocou triedy GCN. Parameter `layer_sizes` definuje počet skrytých vrstiev a ich veľkosť. Parameter `activations` určuje aktivačnú funkciu, ktorá bude aplikovaná na výstup každej vrstvy. Parameter `dropout` určuje hodnotu dropout-u pre vstup každej vrstvy, v tomto prípade je to 50%.

```
from stellargraph.mapper import FullBatchNodeGenerator
from stellargraph.layer import GCN

generator = FullBatchNodeGenerator(G, method="gcn")

gcn = GCN(
    layer_sizes=[16, 16], activations=["relu", "relu"],
    generator=generator, dropout=0.5
)
```

Ukážka kódu 1 Použitie knižnice StellarGraph na vytvorenie dvojvrstvovej GCN

Trénovanie modelu grafovej konvolučnej siete je ukázané v ukážke kódu 2.
Metóda `flow` vytvorí zo vstupných vrcholov a ich označení objekt, ktorý môže byť použitý na trénovanie modelu na nich. Pomocou metódy `GCN.in_out_tensors` sa sprístupní vstupný a výstupný *tensor* modelu GCN pre predikciu vrcholov. Keras model sa vytvorí pomocou vstupného *tensoru* `x_inp` a predikcií z poslednej *dense* vrstvy. *Dense* vrstva funguje ako lineárna vrstva, na ktorú je aplikovaná nelineárna aktivačná funkcia, v tomto prípade `softmax`. Podľa typu úlohy sa vyberie stratová funkcia *loss function*, ktorú model použije. Samotné trénovanie modelu prebieha prostredníctvom metódy `fit`.

4.1.1 Algoritmy

Príklady niektorých algoritmov na spracovanie grafov strojovým učením, ktoré v súčasnosti obsahuje knižnica StellarGraph [12]:

GraphSAGE
Podporuje supervizované aj nesupervizované učenie reprezentácie, klasifikáciu a regresiu vrcholov a predikovanie prepojenia v homogénnych sieťach. Súčasná implementácia podporuje viaceré agregačné metódy ako napríklad mean, maxpool, meanpool a attentional aggregator.

Graph ATtention Network (GAT)
GAT algoritmus v prípade homogénnych grafov podporuje učenie reprezentácie a klasifikáciu vrcholov. Tiež existujú verzie graph attention vrstvy, ktoré podporujú riedke aj husté matice susednosti.

Graph Convolutional Network (GCN)
GCN algoritmus podporuje učenie reprezentácie a klasifikáciu vrcholov pri homogénnych grafoch. Existujú verzie konvolučnej vrstvy grafu, ktoré podporujú riedke aj husté matice susednosti.

```
train_gen = generator.flow(train_subjects.index, train_targets)

x_inp, x_out = gcn.in_out_tensors()

predictions = layers.Dense(
    units=train_targets.shape[1],
    activation="softmax"
)(x_out)

model = Model(inputs=x_inp, outputs=predictions)
model.compile(
    optimizer=optimizers.Adam(lr=0.01),
    loss=losses.categorical_crossentropy,
    metrics=["acc"],
)

val_gen = generator.flow(val_subjects.index, val_targets)

history = model.fit(
    train_gen,
    epochs=200,
    validation_data=val_gen,
    verbose=2,
    shuffle=False,
)
```

Ukážka kódu 2 Trénovanie modelu s knižnicou StellarGraph

4.2 PyTorch Geometric

PyTorch Geometric (PyG) je knižnica na jednoduché písanie a trénovanie grafových neurónových sietí, postavená na PyTorch. Pozostáva z rôznych metód pre hlboké učenie na grafoch a iných nepravidelných štruktúrach, tiež známych ako geometrické hlboké učenie. Ďalšie informácie v tejto podkapitole pochádzajú z dokumentácie knižnice [13].

Graf vytvára dvojice medzi objektmi, konkrétne medzi hranami a vrcholmi. Hrany sa vždy indexujú od nuly do počtu vrcholov mínus jeden, vďaka čomu bude finálna reprezentácia dát čo najkompaktnejšia.

Väčšina datasetov, ktoré sa používajú pri reálnych úlohách sú uložené ako heterogénne grafy. Takéto grafy obsahujú rôzne typy informácií spojené s vrcholmi a hranami. *Tensor* atribútov jednotlivých vrcholov alebo hrán nemôže obsahovať atribúty všetkých vrcholov alebo hrán z celého grafu. Je to z dôvodu možných rozdielov v dimenzionalite a dátových typoch, keďže *tensor* môže obsahovať iba prvky rovnakého dátového typu.

V ukážke kódu 3 je vidieť použitie PyG na vytvorenie dvojvrstvovej grafovej konvolučnej siete. V konštruktore sú zadefinované dve `GCNConv` vrstvy, teda vrstvy grafovej konvolučnej siete, ktoré sú počas behu volané vo funkcii `forward`. Parametrami prvej vrstvy sú počet atribútov vrcholu a potom počet dimenzií skrytej vrstvy. Parametrami druhej vrstvy je najprv počet dimenzií skrytej vrstvy a potom počet tried daného datasetu. Aby nedošlo k preučeniu modelu, ktoré môže vzniknúť pridávaním ďalších vrstiev je použitý `Dropout`. Slúži na regularizáciu a náhodne vynuluje niektoré neuróny.

V ukážke kódu 4 je zobrazené trénovanie grafovej konvolučnej siete. Trénovanie prebieha na

```
import torch.nn.functional as F
from torch_geometric.nn import GCNConv

class GNN(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.conv1 = GCNConv(in_channels, 64)
        self.conv2 = GCNConv(64, out_channels)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = F.dropout(x, p = 0.5, training=self.training)
        x = self.conv2(x, edge_index)
        return F.log_softmax(x, dim=1)

model = GNN(dataset.num_features, dataset.num_classes)
```

Ukážka kódu 3 Použitie knižnice PyG na vytvorenie dvojvrstvovej GCN

trénovacích vrcholoch a v cykle beží pre vopred určený počet epoch. Ako optimalizačný algoritmus je v ukážke použtý `Adam`, ktorý iteratívne na základe trénovacích dát aktualizuje váhy siete. `Optimizer.zero_grad` nastaví gradient všetkých optimalizovaných tensorov na nulu. Ďalej sa použije vhodná stratová funkcia, v tomto prípade `nll_loss`.

```
optimizer = torch.optim.Adam(
    model.parameters(),
    lr=0.01,
    weight_decay=5e-4
    )

model.train()

optimizer.zero_grad()
out = model(data)
loss = F.nll_loss(out[data.train_mask], data.y[data.train_mask])
loss.backward()
optimizer.step()
```

Ukážka kódu 4 Trénovanie modelu s knižnicou PyG

4.3 DGL

Deep Graph Library (DGL) je Python balíček, ktorý vznikol pre uľahčenie implementácie modelov patriacich do rodiny grafových neurónových sietí. V súčastnosti podporuje niektoré existujúce DL frameworky, ako napríklad PyTorch, MXNet a TensorFlow.

DGL označuje vrcholy v grafe pomocou ID vrcholu, kde má každý vrchol pridelené unikátne

celé číslo (*integer*). Každá hrana v grafe je reprezentovaná pomocou dvojice čísel, ktoré označujú vrcholy na oboch jej koncoch. DGL priradí každej hrane podľa poradia, v akom boli pridávané do grafu unikátne celé číslo, ID hrany. Značenie ID vrcholov a ID hrán začína vždy od čísla 0. V DGL sú všetky hrany orientované a označenie (u, v) znamená, že hrana ide z vrcholu u do vrcholu v.

Na prácu s viac vrcholmi DGL používa 1-D pole celočíselného typu (*integer tensor*) obsahujúce ID vrcholov a využíva techniky ako PyTorch tensor, TensorFlow Tensor alebo MXNet ndarray. V DGL sa tento formát nazýva *node-tensor*. Na prácu s viacerými hranami sa používa dvojica *node-tensors* (U, V), kde $(U[i], V[i])$ označuje hranu z $U[i]$ do $V[i]$.

Pri DGL grafoch môžu byť atribúty hrán a vrcholov iba numerického typu. Atribúty vrcholov musia mať unikátny názov, rovnako ako aj atribúty hrán.

Príklad použitia DGL na vytvorenie dvojvrstvovej grafovej konvolučnej siete je zobrazený v ukážke kódu 5. Pomocou agregácie informácií zo susedných vrcholov vie každá vrstva vypočítať nové reprezentácie vrcholov. Vytváranie dvojvrstvovej GNN je podobné ako prostredníctvom PyTorch Geometric. V konštruktore sú zadefinované dve vrstvy grafovej konvolučnej siete, ktoré sú volané vo funkcii `forward` počas behu. Parameter `in_feats` je počet atribútov vrcholu, parameter `h_feats` je počet dimenzií skrytej vrstvy a parameter `num_classes` je počet tried datasetu. Po vytvorení prvej vrstvy sa použije aktivačná funkcia `relu` a potom sa vytvorí druhá vrstva.

```
import torch.nn.functional as F
from dgl.nn import GraphConv

class GCN(nn.Module):
    def __init__(self, in_feats, h_feats, num_classes):
        super(GCN, self).__init__()
        self.conv1 = GraphConv(in_feats, h_feats)
        self.conv2 = GraphConv(h_feats, num_classes)

    def forward(self, g, in_feat):
        h = self.conv1(g, in_feat)
        h = F.relu(h)
        h = self.conv2(g, h)
        return h

model = GCN(g.ndata['feat'].shape[1], 16, dataset.num_classes)
```

Ukážka kódu 5 Použitie knižnice DGL na vytvorenie dvojvrstvovej GCN

V ukážke kódu 6 je priblížené ako prebieha trénovanie grafovej konvolučnej siete. Trénovanie prebieha podobne ako pri iných PyTorch neurónových sieťach. Aj pri DGL je použitý optimalizačný algoritmus `Adam`. Pomocou funkcie `argmax` sa vypočítajú predikcie. Ako stratová funkcia je v tomto prípade použitá funkcia `cross_entropy`, dôležité je, že stratová funkcia sa používa iba na vrcholoch z trénovacej množiny.

```
    def train(g, model):
        optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

        logits = model(g, features)

        pred = logits.argmax(1)

        loss = F.cross_entropy(logits[train_mask], labels[train_mask])

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    model = GCN(g.ndata['feat'].shape[1], 16, dataset.num_classes)
    train(g, model)
```

Ukážka kódu 6 Trénovanie modelu s knižnicou DGL

Kapitola 5
Datasety

V tejto kapitole sú popísané dáta, ktoré boli používané pri porovnávaní výsledkov vybraných metód. Vybrané boli datasety na klasifikáciu vrcholov `ogbn-arxiv` a `ogbn-products`. Obidva používané datasety pochádzajú z OGB (*The Open Graph Benchmark*) [14], čo je zbierka veľkých, realistických, rôznorodých datasetov na testovanie strojového učenie na grafoch.

5.1 Dataset ogbn-arxiv

Dataset `ogbn-arxiv` sa radí svojou veľkosťou podľa škálovania medzi malé datasety. Obsahuje 169 343 vrcholov a 1 166 243 hrán. Je to orientovaný graf, ktorý predstavuje sieť citácií medzi všetkými arXiv vedeckými článkami zameranými na počítačové vedy indexovanými pomocou *Microsoft academic graph* (MAG) [15]. Každý vrchol reprezentuje jeden vedecký článok z arXiv a každá orientovaná hrana reprezentuje citovanie jedného článku druhým. Každý článok obsahuje atribúty vrcholu, ktoré sú reprezentované vektorom so 128 dimenziami, získaný spriemerovaním namapovaných slov z nadpisu a abstraktu. Mapovanie slov z MAG korpusu prebieha pomocou `WORD2VEC` modelu [16].

Úlohou pri tomto datasete je napredikovať 40 kategórií článkov arXiv o počítačových vedách, ako napríklad umelá inteligencia, strojové učenie, operačné systémy, programovacie jazyky, alebo multiagentné systémy [17].

5.2 Dataset ogbn-products

Dataset `ogbn-products` podľa veľkosti patrí medzi stredne veľké datasety. Obsahuje 2 449 029 vrcholov a 61 859 140 hrán. Jedná sa neorientovaný graf, ktorý nie je váhovo ohodnotený a reprezentuje Amazon produkty a ich vzájomný vzťah v jednotlivých nákupoch [18]. Vrcholy reprezentujú jednotlivé produkty a hrana medzi dvomi produktami znamená, že tieto dva produkty boli kúpené spolu. Atribúty vrcholu sú generované pomocou metódy *bag-of-words* z popisu produktu, na ktoré sa ešte použije *Principal Component Analysis* aby sa zredukovala veľkosti dimenzie na 100. Úlohou je predikovať kategóriu produktu, pričom sú kategórie rozdelené na 47 hlavných skupín.

Kapitola 6

Experimenty

Táto kapitola popisuje experimenty s použitými metódami grafových neurónových sietí na klasifikáciu vrcholov, ktoré boli na datasetoch vyskúšané. Testovanie prebiehalo na vybraných knižniciach a grafových algoritmoch, kvôli obmedzeniu v prostriedkoch nebolo možné otestovať všetky knižnice so všetkými algoritmami opísanými v teoretickej časti.

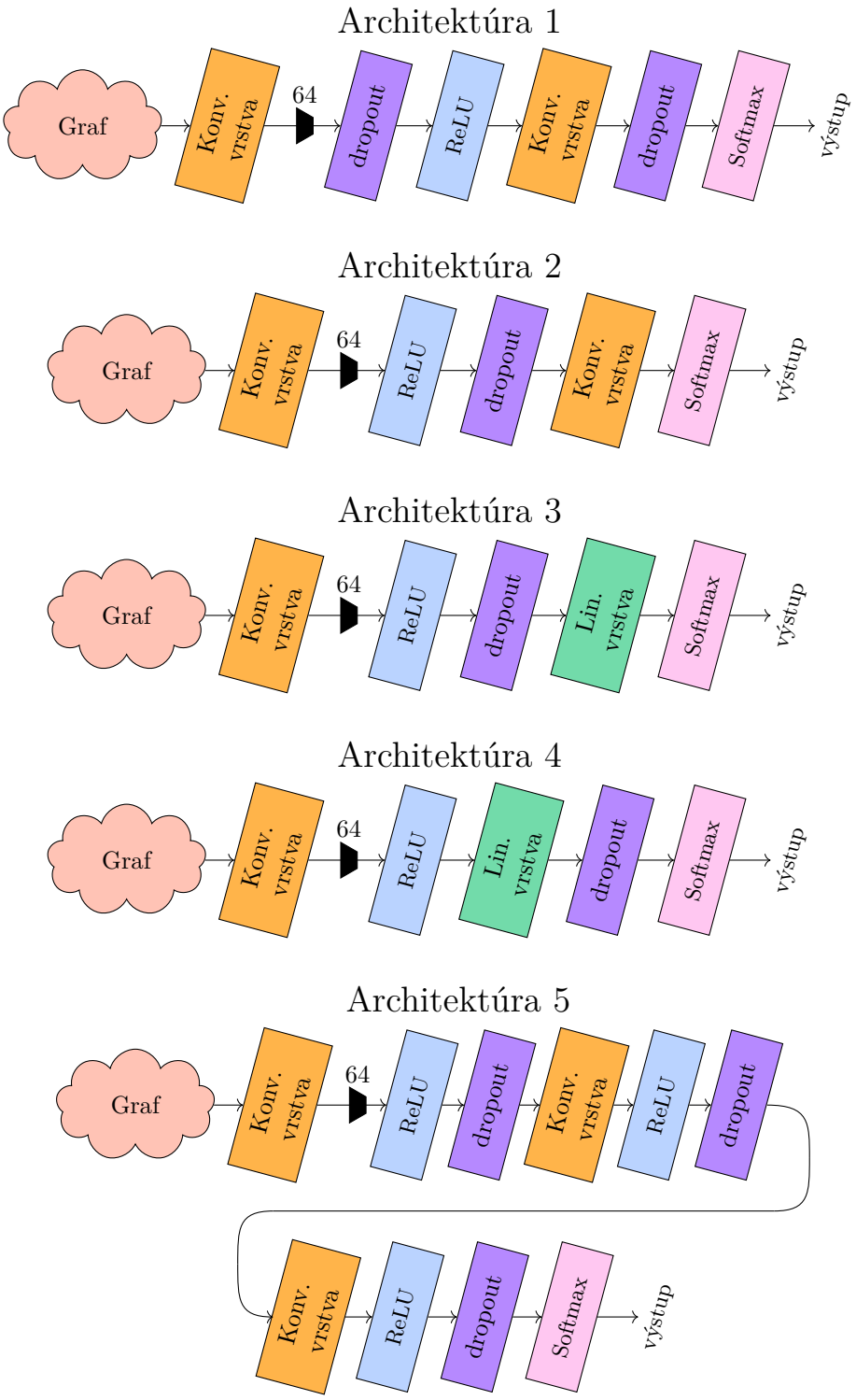
Testovanie s knižnicou PyG prebiehalo na všetkých algoritmoch spomínaných v kapitole 2, GraphSAGE, GCN a GAT. Knižnica StellarGraph bola testovaná iba s algoritmom GraphSAGE. Knižnica DGL nebola testovaná z dôvodu, že jej architektúra je podobná ako pri iných PyTorch sieťach.

V prvej podkapitole sú priblížené jednotlivé architektúry, na ktorých boli metódy skúšané. Druhá podkapitola je venovaná datasetu `ogbn-arxiv` a výsledkom trénovania v súhrnných tabuľkách. V tretej podkapitole je analyzovaný dataset `ogbn-products` a výsledky dosiahnuté s jednotlivými metódami na ňom. Vo štvrtej podkapitole sú prehľadové tabuľky s dosiahnutými výsledkami a porovnanie dosiahnutých výsledkov s dostupnými *state of the art* výsledkami.

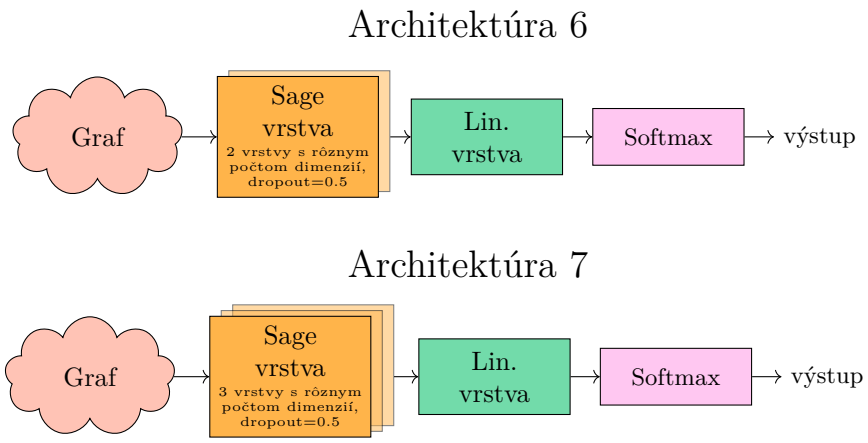
6.1 Popis architektúr

Pre testovanie bolo v počiatočnej fáze vybratých dvanásť architektúr. Testovanie bolo nakoniec zúžené na sedem architektúr, keďže obmedzenie v prostriedkoch nedovoľovalo vybrať viac architektúr. Architektúry jedna až päť sú testované na knižnici PyG s metódami GraphSAGE, GCN a GAT. Šiesta a siedma architektúra je testovaná na knižnici StellarGraph s metódou GraphSAGE. Pre knižnicu StellarGraph boli testované iba dve architektúry, keďže pri nej nie je možné ovplyvňovať rozmiestnenie vrstiev, aktivačných funkcií alebo napríklad *dropout-u*, tak ako pri PyG knižnici.

Na obrázku 6.1 sú ilustračne zobrazené jednotlivé architektúry pre knižnicu PyG. Vstupom je vždy jeden z dvoch datasetov, označený ako graf. Konvolučné vrstvy na obrázkoch sú zástupnými názvami namiesto konkrétnych grafových konvolučných vrstiev, ktoré boli využívané podla typu algoritmu. Potom nasleduje znázornenie počtu dimenzií skrytých vrstiev, pri testovaní boli využívané rôzne veľkosti dimenzií. Pri všetkých architektúrach bola ako aktivačná funkcia využívaná `ReLU`. Obrázok 6.2 ilustruje jednotlivé architektúry pre knižnicu StellarGraph. Vstupom je jeden z dvoch datasetov, nasledujú dve *sage* vrstvy v architektúre 6, resp. tri *sage* vrstvy v architektúre 7. Každá vrstva vykoná *dropout* s hodnotou parametru `p=0,5`.



Obr. 6.1 Zobrazenie vybraných architektúr pre knižnicu PyG.



Obr. 6.2 Zobrazenie vybraných architektúr pre knižnicu StellarGraph.

6.2 Arxiv

V tejto podkapitole sú popísané výsledky testovania vybraných knižníc a algoritmov s jednotlivými architektúrami na datasete `ogbn-arxiv`. Pri testovaní vznikli rozdiely medzi testovaním knižnice PyG a StellarGraph. Pre experimenty s knižnicou PyG, dáta uvedené v tabuľkách boli testované s rovnakými parametrami pri všetkých architektúrach aby sa výsledky medzi sebou lepšie porovnávali. Počet dimenzií skrytých vrstiev bol `hidden_dim=128`, hodnota *dropout-u* bola `p=0,2` a počet *epochs* na ktorých prebiehalo trénovanie bol `epochs=150`. Pri testovaní bolo vidieť, že počet *epoch* ovplyvňuje výsledky iba do nejakého počtu, kedy sa výsledky zlepšujú, potom tam už boli minimálne rozdiely. Aj z tohto dôvodu bol nakoniec stanovený ako ideálny počet *epoch* na trénovanie 150. Čas, ktorý sa v tabuľkách uvádza, je dĺžka času trénovania a vyhodnotenia modelu na procesore počítača (CPU). Tento údaj je ale iba orientačný, nakoľko čas môže byť ovplyvnený aj tým, ako inak bol počítač počas trénovania využívaný a vyťažený. Trénovanie tiež prebiehalo prostredníctvom externej grafiky (GPU), kedy boli všetky časy, okrem architektúry 5, kratšie ako 1 minúta. Z dôvodu, aby boli časové rozdiely dĺžky trénovania lepšie viditeľné sú v tabuľke uvedené časy trénovania na CPU.

6.2.1 PyG - GraphSAGE

Výsledky testovania datasetu `ogbn-arxiv` s knižnicou PyG a algoritmom GraphSAGE sú zobrazené v tabuľke 6.1. Všetky testované architektúry majú približne rovnaké presnosti na testovacích dátach. Ak sa ale pozrieme aj na časy, aj keď sú iba orientačné, tak najlepšie dopadli architektúra 3 a 4. Okrem testovania, ktoré je uvedené v tabuľke, prebiehalo testovanie aj s inými hodnotami parametrov. Napríklad pri zmene počtu dimenzií skrytých vrstiev na `hidden_dim=64` a *dropout-u* `p=0,5` vyšla presnosť testovacieho modelu pri skoro všetkých architektúrach pod 50 %, iba pri architektúre 4 vyšla 51 %.

Na architektúre 3 boli testované kombinácie rôzneho počtu *epoch*, na ktorých trénovanie prebiehalo, rôzneho počtu dimenzií skrytých vrstiev a *dropout-u*. Pri `epochs=200`, `hidden_dim=64` a `p=0,5` bola presnosť na testovacích dátach 51,69 %. Pri `epochs=150`, `hidden_dim=256` a `p=0,5` bola presnosť na testovacích dátach 54,12 % a aj keď je presnosť o trochu vyššia ako pri rovnakej architektúre v tabuľke, tak čas bol dvakrát dlhší. A pri `epochs=150`, `hidden_dim=512` a `p=0,5` bola presnosť na testovacích dátach 55,25 % a čas približne 14 minút.

Architektúra	1	2	3	4	5
Presnosť	53,48 %	53,78 %	53,41 %	53,58 %	51,22 %
Čas (CPU)	26 min 56 s	25 min 11 s	7 min 59 s	7 min 35 s	48 min 4 s

Tabuľka 6.1 Porovnanie testovaní jednotlivých architektúr s knižnicou PyG a algoritmom GraphSAGE na datasete ogbn-arxiv.

6.2.2 PyG - GCN

Výsledky testovania datasetu `ogbn-arxiv` s knižnicou PyG a algoritmom GCN sú zobrazené v tabuľke 6.2. Najlepšie, čo sa týka presnosti na trénovacích dátach dopadli architektúry 3 a 4, stále to ale nie je smerodajný rozdiel oproti ostatným architektúram. Testovanie prebiehalo aj s inými hodnotami parametrov. Pri všetkých architektúrach a hodnotách parametrov `epochs=150`, `hidden_dim=64` a `p=0,5` bola presnosť na testovacích dátach približne rovnaká ako pri výsledkoch v tabuľke ale čas bol kratší približne o 9 minút a pri architektúre 5 až o 20 minút. Pri `epochs=150`, `hidden_dim=128` a `p=0,5`, teda jediný zmenený parameter je hodnota *dropout-u*, bola presnosť aj čas skoro rovnaká ako vo výsledkoch v tabuľke.

Architektúra	1	2	3	4	5
Presnosť	51,37 %	51,55 %	53,22 %	53,25 %	48,94 %
Čas (CPU)	25 min 54 s	24 min 28 s	24 min 11 s	23 min 49 s	48 min 10 s

Tabuľka 6.2 Porovnanie testovaní jednotlivých architektúr s knižnicou PyG a algoritmom GCN na datasete ogbn-arxiv.

6.2.3 PyG - GAT

Výsledky testovania datasetu `ogbn-arxiv` s knižnicou PyG a algoritmom GCN sú zobrazené v tabuľke 6.3. Opäť by sa dalo povedať, že najlepšie, čo sa presnosti týka, dopadla architektúra 3 a 4, ale znovu sa nejedná o veľký, smerodajný, rozdiel. Pri testovaní s inými hodnotami parametrov, `epochs=150`, `hidden_dim=64` a `p=0,5` majú všetky architektúry, okrem 5, presnosť na testovacích dátach okolo 53 % ale čas je približne o 10 minút kratší ako pri údajoch z tabuľky. Architektúra 5, má presnosť iba 49,75 % a čas skoro 31 minút. Architektúra 5 má ako jediná aj lepší výsledok pri zmene parametrov na, `epochs=150`, `hidden_dim=128` a `p=0,5`, kde má presnosť 54,28 %. Opäť si ale treba uvedomiť, že pri ďalšom spustení by už až tak úspešná byť nemusela.

Architektúra	1	2	3	4	5
Presnosť	53,14 %	53,22 %	54,15 %	54,16 %	51,77 %
Čas (CPU)	27 min 32 s	29 min 37 s	25 min 9 s	27 min 39 s	54 min 1 s

Tabuľka 6.3 Porovnanie testovaní jednotlivých architektúr s knižnicou PyG a algoritmom GAT na datasete ogbn-arxiv.

6.2.4 StellarGraph - GraphSAGE

Výsledky testovania datasetu `ogbn-arxiv` s knižnicou StellarGraph a algoritmom GraphSAGE sú zobrazené v tabuľke 6.4. Testovanie architektúr 6 a 7 z tabuľky prebiehalo s parametrami `layer_size=[128, 128]`, alebo `layer_size=[128, 128, 128]`, podľa počtu vrstiev, s dimenziou skrytých vrstiev 128. Spracovanie prebiehalo po várkach (*batch*), `batch_size=1000`. Hodnota *dropout-u* bola `p=0,5`, počet *epoch* bol `epochs=20` a výber počtu krokov do hĺbky od vrcholu pre každú vrstvu, `hops=[10, 5]` pre dve vrstvy a `hops=[10, 5, 5]` pre tri vrstvy.

Testovanie prebiehalo aj so zmenou hodnoty parametra `layer_size=[64, 64]`, kedy bola presnosť trénovacích dát 55,52 %, teda skoro rovnaká ako pri vyššej dimenzii v tabuľke. Pri `layer_size=[64, 64, 64]` bola presnosť 53,15 %. Na príklade testovania čo prebiehalo na `layer_size=[32, 32]` pri `epochs=20`, kde bola presnosť 53,24 % a pri `epochs=100` bola presnosť skoro rovnaká, 53,32 %, je vidieť, že pri StellarGraph a algoritme GraphSAGE stačí na trénovanie aj menší počet *epoch*.

Architektúra	6	7
Presnosť	55,54 %	55,24 %
Čas (CPU)	9 min 30 s	62 min 17 s

Tabuľka 6.4 Porovnanie testovaní jednotlivých architektúr s knižnicou StellarGraph a algoritmom GraphSAGE na datasete ogbn-arxiv.

6.3 Products

Táto podkapitola popisuje výsledky testovania vybraných knižníc a algoritmov s jednotlivými architektúrami na datasete `ogbn-products`. Pri testovaní boli obmieňané hodnoty parametrov, ale pre lepšie porovnávanie jednotlivých algoritmov navzájom sú všetky dáta v tabuľke testované s rovnakými hodnotami. Počet dimenzií skrytých vrstiev `hidden_dim=64`, hodnota *dropout-u* je `dropout=0,2`, veľkosť várky, `batch_size=1024`, výber počtu krokov do hĺbky od vrcholu pre každú vrstvu, `hops = [15, 10]`, pre dve vrstvy a `hops=[15, 10, 5]` pre tri vrstvy. Počet *epoch* bol vybraný ako najvhodnejší pre `epochs=70`, pri testovaní na vyššom počte *epochs* bolo vidno, že od hranice 70 *epoch* sa už presnosť model zlepšovala iba o desatiny percenta.

Trénovanie na tomto datasete, s knižnicou PyG, vzhľadom na jeho veľkosť prebiehalo vo várkach, okrem GCN, keďže veľkosť GPU bola obmedzená a nedovoľovala trénovanie celého grafu naraz. V prípade trénovania a testovania modelu na CPU bez rozdelenia na várky dosahoval čas viac 10 hodín, ako je to aj v prípade údajov v tabuľke k algoritmu GCN. Algoritmus GCN nie je možné trénovať na várkach, pretože táto možnosť nie je v algoritme implementovaná [6]. Kvôli tomu bol algoritmus GCN trénovaný iba na CPU a preto sú časové údaje v tabuľke o toľko vyššie a pohybujú sa v hodinách. V prípade algoritmov GraphSAGE a GAT a rozdelenia na várky s možnosťou trénovania na GPU bol priemerný čas trénovania a testovania modelu, okrem architektúry s tromi vrstvami, 15 minút. Samozrejme rovnako ako pri datasete arxiv, aj tu platí, že údaje o čase sú iba orientačné.

6.3.1 PyG - GraphSAGE

Výsledky testovania datasetu `ogbn-products` s knižnicou PyG a algoritmom GraphSAGE sú zobrazené v tabuľke 6.5. Z výsledkov testovania, ktoré sú v tabuľke je možné povedať, že architektúra 1 a 2 dosahujú trochu vyššiu presnosť. Ak by sme brali do úvahy aj čas testovania a trénovania modelu, tak by ako najlepšia vyšla architektúra 2.

Okrem testovania, ktoré je v tabuľke, prebiehalo testovanie na architektúre 1 aj na iných hodnotách parametrov a to konkrétne `hidden_dim=64` a `dropout=0,5`, kedy bola výsledná presnosť na testovacom modeli približne rovnaká, 75,30 % s časom 30 minút. Pri testovaní rovnakej architektúry s parametrami `hidden_dim=256` a `dropout=0,2` bola presnosť testovacích dát 76,85 %, čo je minimálny rozdiel oproti výsledkom v tabuľke, ale čas bol výrazne vyšší ako pri veľkosti dimenzie 64. Aj z tohto dôvodu bola vybraná veľkosť dimenzie skrytých vrstiev porovnávaná v tabuľkách 64.

Pri experimentovaní tiež prebehlo testovanie s porovnávanými architektúrami na CPU bez várok (*batches*). Pri architektúre 1 bola presnosť testovacích dát 69,40 % a čas trénovania a testovania modelu 590 minút. Pri architektúre 2 bola presnosť 70,18 % a čas 610 minút. Pri architektúre 3 bola presnosť 64,65 % a čas 142 minút. Pri architektúre 4 bola presnosť 65,21 % a čas 127 minút. Architektúra 5 týmto spôsobom nebola testovaná, keďže na základe predchádzajúcich dát bolo zrejmé, že čas trénovania a testovania modelu by bol pri troch *sage* vrstvách veľmi vysoký.

Pri porovnaní trénovaní grafu na vybraných architektúrach na CPU v celku a na GPU s várkami je jasne vidieť, že v prípade rozdelenia grafu várkami na podgrafy je možné dosiahnuť vyššiu presnosť testovacích dát.

Architektúra	1	2	3	4	5
Presnosť	75,86 %	75,78 %	69,99 %	71,76 %	70,15 %
Čas (GPU)	21 min 19 s	12 min 34 s	17 min 28 s	12 min 28 s	45 min 11 s

Tabuľka 6.5 Porovnanie testovaní jednotlivých architektúr s knižnicou PyG a algoritmom GraphSAGE na datasete ogbn-products.

6.3.2 PyG - GCN

Výsledky testovania datasetu `ogbn-products` s knižnicou PyG a algoritmom GCN sú zobrazené v tabuľke 6.6. Trénovanie na všetkých architektúrach prebiehalo na CPU, keďže pri GCN nie je možné trénovať dáta vo várkach. Najlepšie výsledky presnosti na testovacích dátach dosahuje architektúra 1 a 2, ale beh týchto architektúr bol oveľa dlhší ako pri architektúre 3 a 4. Testovanie s inými hodnotami parametrov kvôli dĺžke času trénovania a testovania modelu neprebiehalo.

Architektúra	1	2	3	4	5
Presnosť	68,42 %	68,97 %	61,70 %	62,13 %	61,62 %
Čas (CPU)	667 min 12 s	712 min 17 s	481 min 46 s	501 min 18 s	1 122 min 25 s

Tabuľka 6.6 Porovnanie testovaní jednotlivých architektúr s knižnicou PyG a algoritmom GCN na datasete ogbn-products.

6.3.3 PyG - GAT

Výsledky testovania datasetu `ogbn-products` s knižnicou PyG a algoritmom GAT sú zobrazené v tabuľke 6.7. Opäť je možné konštatovať, že architektúra 1 a 2 dosahujú najvyššiu, podobnú presnosť a aj čas, počas ktorého prebiehalo trénovanie a testovanie modelu.

Počas experimentov pri architektúre 1 bola obmieňaná hodnota parametra `batch_size` z 1 024 na 256 a 64. Pri hodnote 256 bola presnosť 77,14 % a čas 18 minút, takže presnosť ostala podobná

ako v tabuľke, ale čas trénovania a testovania sa predĺžil. Pri hodnote 64 bola presnosť 75,84 % a čas skoro 46 minút. Z experimentov vyplýva, že znižovaním hodnoty parametra `batch_size` sa nedosiahnu lepšie výsledky.

Architektúra	1	2	3	4	5
Presnosť	77,04 %	77,39 %	69,90 %	72,15 %	65,65 %
Čas (CPU)	13 min 47 s	13 min 25 s	14 min 43 s	14 min 27 s	52 min 28 s

Tabuľka 6.7 Porovnanie testovaní jednotlivých architektúr s knižnicou PyG a algoritmom GAT na datasete ogbn-products.

6.3.4 StellarGraph - GraphSAGE

Výsledky testovania datasetu `ogbn-products` s knižnicou StellarGraph a algoritmom GraphSAGE sú zobrazené v tabuľke 6.8. Testovanie architektúr 6 a 7 z tabuľky prebiehalo s parametrami `layer_size=[64, 64]`, alebo `layer_size=[64, 64, 64]`, podľa počtu vrstiev, s dimenziou skrytých vrstiev 64. Zvyšné parametre sú konfigurované rovnako ako v podpodkapitole 6.2.4. Vyššiu presnosť v tomto prípade síce dosiahla architektúra 7, ale rozdiel v úspešnosti oproti architektúre 6 nie je až tak výrazný. Avšak rozdiel v časoch pri trénovaní a testovaní modelu medzi obidvomi architektúrami je 5-násobný.

Ďalšie testovanie prebiehalo so zmenou hodnoty parametru `layer_size` pri dvoch vrstvách. Pri `layer_size=32,32` dosiahla presnosť testovacích dát 66,41 %. Na dvoch vrstvách o veľkosti 128 dimenzií bola presnosť 70,69 %.

Architektúra	6	7
Presnosť	68,27 %	70,14 %
Čas (CPU)	21 min 58 s	115 min 40 s

Tabuľka 6.8 Porovnanie testovaní jednotlivých architektúr s knižnicou StellarGraph a algoritmom GraphSAGE na datasete ogbn-products.

6.4 Prehľad výsledkov použitých metód

V tejto podkapitole sú prehľadové tabuľky pre vybrané architektúry s knižnicami PyG a StellarGraph. Je v nich zobrazené porovnanie jednotlivých algoritmov medzi sebou. Vďaka tomu je ľahšie vidieť, ktorý algoritmus je najlepší, tj. dosahuje najvyššiu presnosť na testovacích dátach.

Z testovania vyplýva, že najlepším algoritmom na klasifikáciu vrcholov grafových neurónových sietí s knižnicou PyG je GAT. Rozdiel je ale naozaj minimálny, pretože s algoritmom GraphSAGE dosahujú skoro rovnaké výsledky. Z tohto dôvodu by som na základe mnou dosiahnutých výsledkov povedala, že algoritmami GraphSAGE a GAT na knižnici PyG sú rovnako dobré. Algoritmus GCN nedosahoval pri testovaní až také výsledky. Na datasete ogbn-products to bolo hlavne z dôvodu, ako bolo v práci už viackrát spomenuté, že nie je možné aby trénovanie prebiehalo vo várkach [6].

Pri porovnaní dosiahnutých výsledkov s algoritmom GraphSAGE na knižnici PyG a StellarGraph je vidieť, že pri menšom datasete, ogbn-arxiv, ktorý nebol na knižnici PyG trénovaný

vo várkach dosahuje lepšie výsledky StellarGraph, pri ktorom prebieha trénovanie vo várkach pri oboch datasetoch. Ale pri porovnaní knižníc na datasete ogbn-products, kde obidve trénovania prebiehali s várkami už dosahuje lepšie výsledky knižnica PyG.

Ako *state of the art* výsledky budem používať hodnoty z rebríčkov obidvoch datasetov, ktoré sú na stránkach OGB [19]. Pri datasete ogbn-arxiv je najlepšia dosiahnutá presnosť na trénovacích dátach 79,66 %, s počtom parametrov modelu 139 617 656. Z mnou testovaných algoritmov sa najvyššie v rebríčku nachádza GCN, ktorý dosiahol presnosť na testovacích dátach približne 71,74 % a bol trénovaný na 110 120 parametroch modelu. Hneď za ním sa v rebríčku nachádza algoritmus GraphSAGE s presnosťou 71,49 %, trénovaný na 218 664 parametroch. Algoritmus GAT sa v rebríčku nachádza iba v kombinácii s inými metódami, takže ho z porovnania na tomto datasete vynechám. Mnou najlepší dosiahnutý výsledok na algoritme GCN je presnosť 53,25 % a počet parametrov modelu je 21 672, čo je viac ako 5-krát menej. Pri algoritme GraphSAGE dosiahli moje dáta najvyššiu presnosť pri 43 176 parametroch modelu, čo je opäť 5-krát menej parametrov ako využíval model zo *state of the art* výsledkov.

Pri datasete ogbn-products je najlepšia dosiahnutá presnosť na trénovacích dátach 90,14 %, pri počte parametrov modelu 139 633 805. Z mnou testovaných algoritmov sa najvyššie v rebríčku nachádza algoritmus GAT s *NeighborSampling*, ktorý dosiahol presnosť na testovacích dátach približne 79,45 %, pri počte parametrov 751 574. Ide o ukážkovú implementáciu z dokumentácie PyG. Na jej základe vznikli implementácie mnou vybraných architektúr.

Mnou dosiahnuté výsledky nedosahujú celkovo tak vysoké hodnoty presnosti na trénovacom modeli ako *state of the art* výsledky. Hlavným dôvodom je, že nemám dostatočné výpočtové prostriedky. To je tiež dôvod, prečo som mohla trénovať modely iba na oveľa menšom počte parametrov. Tiež by sa dalo povedať, že dataset je komplexný a pre správnu klasifikáciu vyžaduje komplexné techniky, ktoré by opäť potrebovali oveľa väčšie výpočtové prostriedky. A keďže mojou úlohou bolo demonštrovať základné techniky mám výrazne horšie výsledky.

Dataset - ogbn-arxiv			
Architektúra	PyG - GraphSAGE	PyG - GCN	PyG - GAT
1	53,48 %	51,37 %	53,14 %
2	53,78 %	51,55 %	53,22 %
3	53,41 %	53,22 %	54,15 %
4	53,58 %	53,25 %	54,16 %
5	51,22 %	48,94 %	51,77 %

Tabuľka 6.9 Prehľad testovaní vybraných architektúr s knižnicou PyG a všetkými algoritmami na datasete ogbn-arxiv.

Dataset - ogbn-arxiv	
Architektúra	StellarGraph
6	55,54 %
7	55,24 %

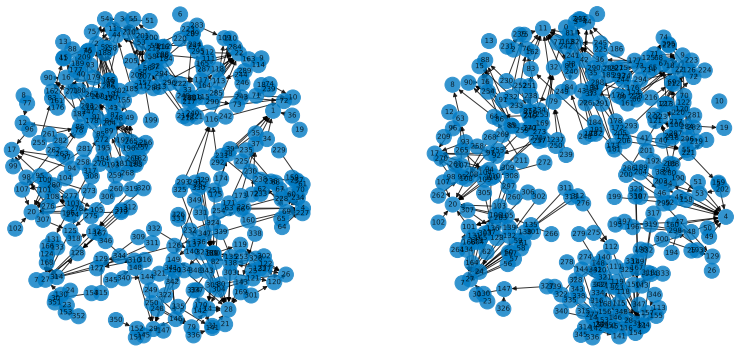
Tabuľka 6.10 Prehľad testovaní vybraných architektúr s knižnicou StellarGraph a algoritmom GraphSAGE na datasete ogbn-arxiv.

Dataset - ogbn-products			
Architektúra	PyG - GraphSAGE	PyG - GCN	PyG - GAT
1	75,86 %	68,42 %	77,04 %
2	75,78 %	68,97 %	77,39 %
3	69,99 %	61,70 %	69,90 %
4	71,76 %	62,13 %	72,15 %
5	70,15 %	61,62 %	65,65 %

Tabuľka 6.11 Prehľad testovaní vybraných architektúr s knižnicou PyG a všetkými algoritmami na datasete ogbn-products.

Dataset - ogbn-products	
Architektúra	StellarGraph
6	68,27 %
7	70,14 %

Tabuľka 6.12 Prehľad testovaní vybraných architektúr s knižnicou StellarGraph a algoritmom GraphSAGE na datasete ogbn-products.



Obr. 6.3 Vizualizácia podgrafov pomocou Networkx na väčšej várke z datasetu ogbn-arxiv.

Kapitola 7
Záver

Vo svojej bakalárskej práci som vyskúšala rôzne metódy GNN pre klasifikáciu uzlov a grafov a porovnala som mnou dosiahnuté výsledky medzi sebou. Testovala som dve knižnice, PyG a StellarGraph. V teoretickej časti práce opisujem aj knižnicu DGL, ale nezaradila som ju do testovania, pretože jej architektúra je podobná ako pri iných PyTorch sieťach. Vybrané knižnice som skúšala s tromi algoritmami, GraphSAGE, GCN a GAT. PyG som skúšala so všetkými tromi knižnicami a StellarGraph iba s algoritmom GraphSAGE.

Pri mojich experimentoch dosahovala lepšie výsledky knižnica PyG. Algoritmy GraphSAGE a GAT s ňou dosahovali približne rovnaké výsledky. GCN s nimi ale nie je možné úplne porovnávať, hlavne na väčších grafoch, keďže s ním nie je možné trénovať dáta vo várkach [6]. Taktiež som porovnala moje výsledky s dostupnými *state of the art* výsledkami a v podkapitole 6.4 som vysvetlila príčiny, prečo mnou dosiahnuté výsledky nie sú až tak vysoké, ako *state of the art* výsledky. Hlavným dôvodom je, že som nemala dostatočné výpočtové prostriedky, ktoré by boli potrebné na správnu klasifikáciu komplexnejšími technikami. Aj z tohto dôvodu bolo mojou úlohou demonštrovať základné techniky s výrazne nižšími počtami parametrov ako boli použité v *state of the art* výsledkoch.

Bibliografia

1. ZHOU, Jie; CUI, Ganqu; ZHANG, Zhengyan; YANG, Cheng; LIU, Zhiyuan; SUN, Maosong. Graph Neural Networks: A Review of Methods and Applications. *CoRR*. 2018, roč. abs/1812.08434. Dostupné z arXiv: 1812.08434.
2. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
3. ABIODUN, Oludare Isaac; JANTAN, Aman; OMOLARA, Abiodun Esther; DADA, Kemi Victoria; MOHAMED, Nachaat AbdElatif; ARSHAD, Humaira. State-of-the-art in artificial neural network applications: A survey. *Heliyon*. 2018, roč. 4, č. 11, e00938. ISSN 2405-8440. Dostupné z DOI: https://doi.org/10.1016/j.heliyon.2018.e00938.
4. KLOUDA, Karel; LOPEZ, Juan Pablo Maldonado; VAŠATA, Daniel. *BI-VZD přednáška 11* [[online prednáška]]. FIT ČVUT Courses, 2021. Dostupné tiež z: https://courses.fit.cvut.cz/BI-VZD/lectures/files/BI-VZD-11-cs-handout.pdf. [vid. 24.1.2023].
5. LI, Fei-Fei; WU, Jiajun; GAO, Ruohan. *Neural Networks Part 1: Setting up the Architecture* [online]. Stanford University CS231n: Deep Learning for Computer Vision, 2022. Dostupné tiež z: https://cs231n.github.io/neural-networks-1/#nn. [vid. 1.2.2023].
6. CHEN, Jianfei; ZHU, Jun; SONG, Le. *Stochastic Training of Graph Convolutional Networks with Variance Reduction*. arXiv, 2017. Dostupné z DOI: 10.48550/ARXIV.1710.10568.
7. HAMILTON, William L.; YING, Rex; LESKOVEC, Jure. Inductive Representation Learning on Large Graphs. *CoRR*. 2017, roč. abs/1706.02216. Dostupné z arXiv: 1706.02216.
8. VELIČKOVIĆ, Petar; CUCURULL, Guillem; CASANOVA, Arantxa; ROMERO, Adriana; LIÒ, Pietro; BENGIO, Yoshua. *Graph Attention Networks*. arXiv, 2017. Dostupné z DOI: 10.48550/ARXIV.1710.10903.
9. BASTIAN, Mathieu; HEYMANN, Sebastien; JACOMY, Mathieu. Gephi: An Open Source Software for Exploring and Manipulating Networks. *Proceedings of the International AAAI Conference on Web and Social Media*. 2009, roč. 3, s. 361–362. Dostupné z DOI: 10.1609/icwsm.v3i1.13937.
10. HAGBERG, Aric A.; SCHULT, Daniel A.; SWART, Pieter J. Exploring Network Structure, Dynamics, and Function using NetworkX. In: VAROQUAUX, Gaël; VAUGHT, Travis; MILLMAN, Jarrod (ed.). *Proceedings of the 7th Python in Science Conference*. Pasadena, CA USA, 2008, s. 11–15.
11. *Graph Visualization with Neo4j* [online]. Neo4j, Inc., 2023. Dostupné tiež z: https://neo4j.com/graph-visualization-neo4j/. [vid. 15.2.2023].
12. DATA61, CSIRO's. *StellarGraph Machine Learning Library* [https://github.com/stellargraph/stellargraph]. GitHub, 2018.

13. TEAM, PyG. *PyG Documentation — pytorchgeometricdocumentation* [https://pytorch-geometric.readthedocs.io/en/latest/]. 2023. [vid. 15.2.2023].

14. HU, Weihua; FEY, Matthias; ZITNIK, Marinka; DONG, Yuxiao; REN, Hongyu; LIU, Bowen; CATASTA, Michele; LESKOVEC, Jure. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687*. 2020.

15. WANG, Kuansan; SHEN, Zhihong; HUANG, Chiyuan; WU, Chieh-Han; DONG, Yuxiao; KANAKIA, Anshul. Microsoft Academic Graph: When experts are not enough. *Quantitative Science Studies*. 2020, roč. 1, č. 1, s. 396–413. ISSN 2641-3337. Dostupné z DOI: 10.1162/qss_a_00021.

16. MIKOLOV, Tomas; SUTSKEVER, Ilya; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey. *Distributed Representations of Words and Phrases and their Compositionality*. arXiv, 2013. Dostupné z DOI: 10.48550/ARXIV.1310.4546.

17. *Computer Science* [online]. Cornell University, 2023. Dostupné tiež z: https://arxiv.org/corr/subjectclasses. [vid. 12.2.2023].

18. BHATIA, K.; DAHIYA, K.; JAIN, H.; KAR, P.; MITTAL, A.; PRABHU, Y.; VARMA, M. *The extreme classification repository: Multi-label datasets and code*. 2016. Dostupné tiež z: http://manikvarma.org/downloads/XC/XMLRepository.html.

19. *Leaderboards for Node Property Prediction | Open Graph Benchmark* [online]. Open Graph Benchmark, [b.r.]. Dostupné tiež z: https://ogb.stanford.edu/docs/leader_nodeprop/#ogbn-arxiv. [vid. 12.2.2023].

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF