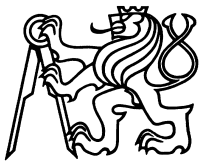
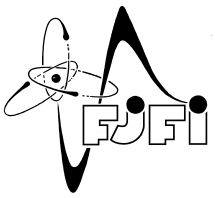


České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská



Analýza kombinování poziční a strukturální informace v grafových neuronových sítích

Analysis of positional and structural information in graph neural networks

Bakalářská práce

Author: **Adéla Schwanzerová**

Supervisor: **Ing. Pavel Procházka, Ph.D.**

Language advisor: **Mgr. Hana Čápová**

Academic year: 2022/2023

České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská

Katedra: matematiky

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Adéla Schwanzerová
Studijní program:	Aplikace přírodních věd
Studijní obor:	Matematické inženýrství
Studijní zaměření:	Aplikované matematicko-stochastické metody
Název práce (česky):	Analýza kombinování poziční a strukturální informace v grafových neuronových sítích
Název práce (anglicky):	Analysis of positional and structural information in graph neural networks

Pokyny pro vypracování:

1) Seznámení se s aktuálními technikami trénování grafických neuronových sítí (GNN).
2) Seznámení se s metodami redukce složitosti trénování GNN v [1] a referencovaných zdrojích.
3) Seznámení se s implementací numerických simulací v [1].
4) Experimentální vyhodnocení metody implementace GNN kombinující poziční a strukturální informaci, kde poziční informace (predikce modelu) je dána zjednodušenou GNN podle [1] a strukturální informace je dána predikcí GNN modelu aplikovanou na klastru uzlů [1].
5) Porovnání vlastností výsledné metody výkonost-paměťová náročnost s původní GNN. Provedení experimentu pro různé nastavené GNN a různé parametry zjednodušení grafu.

Doporučená literatura:

1) P. Procházka, M.Mareš, M. Dědič, Scalable Graph Size Reduction for Efficient GNN Application. In 'CEUR Workshop Proceedings', 2022.

2) W. L. Hamilton, R. Ying, J. Leskovec, Inductive Representation Learning on Large Graphs. In 'Advances in neural information processing systems', Neural Information Processing Systems Foundation, 2017, 1012-1034.

3) J. Bai, Y. Ren, J. Zhang, Ripple Walk Training: A Subgraph-based training framework for Large and Deep Graph Neural Network. In 'arXiv:2002.07206', arxiv.org, 2020, 1-8.

4) H. Chen, B. Perozzi, Y. Hu, S. Skiena, HARP: Hierarchical Representation Learning for Networks. In 'Proceedings of the AAAI Conference on Artificial Intelligence' 32, 1, 2018.

5) Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. S. Yu, A Comprehensive Survey on Graph Neural Networks. In 'IEEE Transactions on Neural Networks and Learning Systems', IEEE computational intelligence society, 2020, 1-22.

6) T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks. In 'Proceedings of the 5th International Conference on learning representations', ICLR, 2016, 1-14.

Jméno a pracoviště vedoucího bakalářské práce:

Ing. Pavel Procházka, Ph.D.
Cisco Systems, s.r.o., Karlovo nám. 2097/10, 120 00 Praha 2 - Nové Město,

Jméno a pracoviště konzultanta:

Datum zadání bakalářské práce: 31.10.2022

Datum odevzdání bakalářské práce: 2.8.2023

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 31.10.2022

.....
garant oboru

.....
vedoucí katedry

.....
děkan

Poděkování:
Chtěla bych zde poděkovat svoji rodině, která mě po celou dobu studia podporovala. Dále svému manželovi, který mi byl oporou a měl pochopení pro dny a večery strávených nad psaním této práce. Zejména bych však chtěla poděkovat svému školiteli, který mi byl vždy nápomocen, a v případě potřeby mě nasměroval správným směrem nebo mi poradil jiný přístup k řešení problémů, se kterými jsem si v průběhu nevěděla rady. Zároveň bych chtěla poděkovat za jeho trpělivost a ochotu. V neposlední řadě děkuji také společnosti Cisco Systems, díky které mohla celá práce vzniknout.

Čestné prohlášení:
Prohlašuji, že jsem tuto práci vypracovala samostatně a uvedla jsem všechnu použitou literaturu.

V Praze dne 5. ledna 2023 Adéla Schwanzerová

Analýza kombinování poziční a strukturální informace v grafových neuronových sítích

Autor: Adéla Schwanzerová

Obor: Matematické inženýrství
Zaměření: Aplikované matematicko-stochastické metody

Druh práce: Bakalářská práce

*Vedoucí práce:*Ing. Pavel Procházka, Ph.D.,
Cisco Systems, s.r.o.
Karlovo nám. 2097/10
120 00 Praha 2 - Nové Město

Abstrakt:

Tato práce se zabývá hledáním optimálního propojení poziční a strukturální kompozice grafu za využití grafických neuronových sítí k predikci jednotlivých tříd. Experimenty jsou prováděny na datasetech Cora a Karate klub. Po teoretické části, ve které byly vysvětleny matematické zákonitosti GNN a jejich možných základních rozdílů, byly představeny implementované metody. Těmi jsou GCN, GAT, GraphSAGE a ChebConv. Povedlo se tak najít taková nastavení GNN, která překonala výsledky lineární regrese při grafu zredukovaném až na téměř 40%. Ovšem stále je redukce grafu na úkor přesnosti.

Klíčová slova: ChebConv, GAT, GCN, GNN, GraphSAGE, Poziční GNN, Redukce, Strojové učení, Strukturální GNN

Title:

Analysis of positional and structural information in graph neural networks

Author: Adéla Schwanzerová

Abstract:

This work is about searching for optimal combination of positional and structural compozition of graph with usage of graph neural network for class prediction. Experiments are done on datasets Cora and Karate Club. The theoretical part contains the mathematical background of several types of GNN and introduces the main differences between them. In the practical part, the implemented methods are explained, such as GCN, GAT, GraphSAGE and ChebConv. There were found such settings of GNN that overreached results of linear regresson on reduced graph on almost 40%. The probability rise to its maximum, but the reduction of graph is still at the expennse of accuracy.

Key words: Computation reduction, GAT, GCN, GNN, GraphSAGE, ChebConv, Machine learning, Position-GNN, Reduction, Structural-GNN

Obsah

1 Úvod

1.1 Využití GNN

Grafové neuronové sítě (zn. GNN) se v poslední době těší vyššímu zájmu [1] a zároveň se objevuje tendence je implementovat v mnohých praktických odvětvích, kde jsou data uložena ve formě grafu. To se již v několika případech stalo. Poněvadž právě díky přítomnosti grafu se, na rozdíl od klasických neuronových sítí, mohou do výsledků promítnout vazby mezi jednotlivými entitami a jejich jednotlivé vlastnosti. Využití GNN tak můžeme rozčlenit do několika sekcí.

Jednou obsáhlou skupinou je kombinační optimalizace, která se využívá v energetice, logistice, financích nebo při rozvržení hardwaru. Což znamená, že GNN lze aplikovat na klasické problémy jako je problém obchodního cestujícího, maximální řez nebo minimální kostra. Toho využil například tým Google Brain, který přes GNN optimalizoval oblast a výkon čipů při vývoji hardwaru Google TPU [2] [3].

Další soubor různých aplikací je tzv. počítačové vidění, kdy je obraz rozložen do grafu, na základě kterého se dají vyhodnotit vzdálenosti, velikosti a rozpoznávání objektů, nebo na základě videí rozeznat lokaci či vytvořit 3D rekonstrukci místa. Tím se mimo jiné zabývá i firma MagicLeap [2].

Dále tu máme různé fyzikální a chemické úlohy. Například ukládání obnovitelné energie, kterou se zabývá Facebook a CMU v projektu Open Catalyst. GNN se zde využívá pro hledání nových katalyzátorů, čímž se vyhnuli nákladným simulacím. DeepMind zase aplikovalo GNN na simulaci komplexních dynamických systémů různých částic, čímž objevují další vzorce chování jako je formování skla. A Fermilab v CERNu díky GNN hledá nové částice [2] [4]. Obecně se GNN ve fyzice využívá k predikcím komplexních systémů a jejich interakci i několik tisíc kroků dopředu. V chemii zase k objevení nových chemických struktur či jejich bližšímu ohledání [5].

Obrovskou podsekcí této skupiny jsou léčiva. Typickým příkladem je výzkum ze Standfordu, kde se předvídají účinky různých kombinací léčiv. Zejména těch, které nejsou tak časté, a tedy není možné všechny experimentálně vyhodnotit. Na podobném principu funguje i jejich celkový vývoj, kdy díky strojovému učení přejdou do experimentální části pouze slibnější vzorky. Tyto grafy, obsahující interakce mezi metaboliky, proteiny či mRNA, se využívají na proteinové inženýrství, predikci vlastností molekul, identifikaci cíle, opakované použití léků a mnoho dalších směrů. Například na MIT určují, zda mají některé molekuly antibiotické vlastnosti [6] [2] [7].

Další skupinou je například doporučovací systém. Grafové sítě jsou natrénovány na vztahy mezi jednotlivými produkty, weby, místy či filmy (v popisu se zaměříme na produkt) a dokáží doporučit nejvhodnější zboží na základě předchozích interakcí. Toho využívá například Amazon, Alibaba, Pinterests či UberEats. Každá z těchto společností (snad až na UberEats, který využívá pouze lehkou obměnu modelu GraphSage [8] díky malému množství dat vázaných na danou lokaci) má svůj vlastní model GNN. A to zejména kvůli velkým grafům obsahujících až miliardy uzlů, které se v čase mění, a je proto nutné snižovat náročnost celého procesu [2] [7] [5].

Je tedy zřejmé, že GNN se rychle vyvíjí a má slibné výsledky v mnoha oborech. Velice často se totiž setkáváme s daty propojenými grafem. I to je důvod, proč se s ní setkáváme častěji a postupně nahrazuje jiné metody jako je mean-pooling, max-pooling, vizuální, nebo anotační vnoření - numerická reprezentace uzlu (embeddings), propagaci značení (label propagation), propagaci strojového učení (zn. MLP) a další.

1.2 Limitace GNN

Ačkoli je GNN velmi silným nástrojem, naráží na několik problémů, které s touto metodou přichází. Jedním je například přetrénování, jež se zpravidla koriguje validačními daty. Dále se mezi nevýhody dá

zařadit fakt, že neuronové sítě jsou do jisté míry černou skříňkou. A ačkoli víme, na jakém principu fungují, tak ani její programátoři nejsou schopni popsat všechny dílčí kroky rozhodování do posledního detailu. Neuronové sítě jsou navíc poměrně náročné na vývoj, a proto mnoho společností upřednostní jednodušší algoritmus, na jehož vývoji stráví méně času. Další limitací ale mohou být i malá data, na kterých se jiné metody mohou chovat lépe [9].

Nevýhodou grafových neuronových sítí je i využívání střední a maximální vrstvy zejména v konvolučních sítích. Tento jev vzniká kvůli učení na několika jednotkách vrstev, na kterých se průměr a maximum počítají. Tím může vzniknout nepřesnost celé metody. Ovšem velkým problémem je výpočetní náročnost. I když se počítá na grafech, tak se výpočetní náročnost zvyšuje s každou další iterací a aktualizací vah [10]. Díky tomu se objevují vyšší požadavky jak na paměť, tak na čas. Proto je nutné složitost GNN redukovat. Cesty, kterými se k této otázce přistupuje, a zavedení některých pojmů GNN naleznete v kapitole 2 Strojové učení a 3 Teorie GNN. Některé základní přístupy, které se tímto problémem zabývají, najdete blíže popsány v kapitole 4 Druhy GNN.

1.3 Cíl a datasety

Cílem této práce je seznámit se s metodami redukce náročnosti grafových neuronových sítí a najít optimální poziční a lokální parametry GNN upravené dle článku [11] (článek je podrobněji popsán v sekci 5.2). K testům bylo použito několik druhů známých implementacích GNN jako například GCN, GAT, GraphSage (všechny popsány dále) na datasetech Cora a Karate Club.

Dataset Cora [12] sestavil institut Ifremer. Jedná se o 2708 vědeckých publikací rozdělených do sedmi kategorií. Ty jsou navzájem provázány pomocí 5429 citací.

Karate Club [13], přesněji Zacharův Karate klub, je malý dataset, používaný především ke studijním účelům a popisování struktury skupin, vytvořený v roce 1977. Graf ukazuje provázání 34 členů karate klubu podle toho, s kým komunikují i mimo klub. Tím se vytvořilo 78 in interakcí. Původní rozdělení do dvou skupin pak vzniklo neshodami správce a instruktora, čímž se klub rozpadl. Nové kluby pak tvoří vždy ještě další dvě skupiny, čímž můžeme členy rozdělit do 4 komunit.

V sekci Strojové učení jsou představeny základní funkce strojového učení. V kapitole 3 jsou hlouběji popsány různé metody GNN. Konkrétní používané metody a přístupy naleznete v sekci 4 Druhy GNN. V části Redukce složitosti jsou vysvětleny různé přístupy k redukování náročnosti GNN v rámci výzkumu a v poslední sekci jsou představeny a popsány výsledky práce.

2 Strojové učení

K lepšímu pochopení GNN je dobré si první přiblížit strojové učení. Strojové učení je druh umělé inteligence, které na základě již známých dat dokáže předpovídat hodnoty neznámých entit, nebo pomoci k jejich predikci.

Jak již název napovídá, strojové učení se učí v průběhu řešení dané optimalizační úlohy. Přístupů k učení je několik, ale dají se rozdělit do čtyř základních skupin.

V první řadě učení s učitelem poskytuje programátorovi přímý přístup k výstupním značením. Ten pak v trénovací fázi specificky určí, které hodnoty mají být určeny pro zadané data. Tak jsou známy všechny vstupní hodnoty a předem určeny výstupní data. Na základě těchto znalostí se pak neuronová síť snaží přiřadit výsledky k dalším situacím.

Učení bez učitele naopak nemá v trénovacích datech provázána vstupní a výstupní data. Neuronová síť je tak nucená vlastní oragnizaci dat již od začátku. Předem je však stanoveno, která data budou trénovací a která se budou vyhodnocovat.

Kombinace těchto metod pracuje jak s provázanými vstupní a výstupními daty, tak s daty nezorganizovanými. Čímž se programu umožní vytvářet vlastní vyhodnocení dat s větší informační hodnotou. Převyšovat by však měl počet známých dat. Zároveň však musíme mít na vědomí riziko přeučení.

Posledním druhem strojového učení je tzv. zpětnovazebné učení. To se využívá k vícekrokovému procesu s přesně definovanými pravidly. Algoritmus pak má dokončit zadaný úkol, k němuž má daných několik mezikroků, avšak ve většině času se musí rozhodnout sám, jakým směrem pokračovat [14].

Těmto skupinám se lze v rámci strojového učení věnovat více směry. Ať už pomocí lineární, či logistické regrese, shlukování, rozhodovacích stromů, nebo neuronových sítí. Ty se liší mimo jiné i výběrem parametrů, jako jsou aktivační a ztrátové funkce, nebo vhodným algoritmem optimalizace.

2.1 Modelování funkce

Strojové učení je v podstatě optimalizační úloha, kterou se snažíme co nejlépe modelovat. Tyto modely je pak nutné optimalizovat pomocí různého nastavování parametrů. Mezi známé modely patří například rozhodovací stromy, logistické regrese, nebo třeba právě neuronové sítě.

2.2 Algoritmus optimalizace

Optimalizační algoritmy se používají například ke změnám vah a stupně učení tak, aby se zpřesnila predikce a minimalizoval se například čas. Takových optimalizačních algoritmů je několik. Patří mezi ně například Gradientní sestup (Gradient descent), Stochastický gradientní sestup (Stochastic gradient descent), Hybnost (Momentum) nebo Adagrad (adaptivní gradientní algoritmus) [15]. Zde se ovšem budeme setkávat především s funkcí Adam.

2.2.1 Adam

Jako optimalizační algoritmus často nacházíme funkci Adam. Adam propojuje RMSprop [16] a stochastický gradient s hybností. Stejně jako RMSprop i Adam využívá ke škálování rychlosti učení druhé mocniny gradientů a zároveň pomocí klouzavého průměru gradientu využívá hybnost. Rychlost učení pro váhy GNN je přizpůsobována z odhadů prvního a druhého momentu gradientu [17].

2.3 Ztrátová funkce - LOSS

Dále se v metodách strojového učení setkáváme se ztrátovou funkcí, kterou se snažíme minimalizovat. Ztrátové funkce používáme především k optimalizaci modelu během trénování. Takže čím menší je ztráta, tím je trénovaný model lepší. Během trénování model přiřazuje entitám trénovacího datasetu různé hodnoty. Cílem je upravit váhy tak, aby hodnoty byly co nejblíže skutečnosti. Tato část trénování se označuje učení.

V podobném smyslu se vykládá i funkce nákladů, která se může počítat i jako průměr všech ztrátových funkcí.

Mezi ztrátové funkce můžeme zařadit například faktorizaci grafu. Její aplikace je zahrnuta v rovnici

$$\mathcal{L} = \frac{1}{|\mathbf{Y}_O|} \sum_{i \in \mathbf{Y}_O} loss(y_i, h_i^{(L)}), \tag{1}$$

kde $\mathbf{Y}_O$ je matice značení pro známé uzly, $h_i^{(L)}$ i-tý řádek $H^{(L)}$ a y_i je i-tý řádek $\mathbf{Y}_O$.

2.3.1 Funkce křížové entropie

Funkce křížové entropie (cross-entropy loss function) se používá zejména pro optimalizaci modelů zabývajících se klasifikací. Entropie náhodné proměnné X je úroveň nejistoty, kterou máme k možnému výsledku. Pro pravděpodobnostní distribuci $p(x)$ a náhodnou proměnnou X definujeme entropii jako

$$\mathbf{K}(x) = \begin{cases} -\int_x p(x) \log p(x), & \text{pokud } X \text{ je spojité} \\ -\sum_x p(x) \log p(x), & \text{pokud } X \text{ je diskrétní} \end{cases} \tag{2}$$

Mínus se používá kvůli tomu, že $p(x)$ je pravděpodobnostní distribucí nabývající hodnot mezi 0 a 1, tudíž její logaritmus vychází záporný, ale námi požadovaný výsledek musí nabývat kladných hodnot.

Čím má **K** menší hodnotu, tím méně nejistoty přináší do pravděpodobnostního rozdělení. Z toho důvodu je zřejmé, že dokonalý model má ztrátovou funkci nulovou.

Křížovou entropii tak definujeme

$$\mathcal{L} = -\sum_{i=1}^{n} t_i \log(p_i), \tag{3}$$

pro n tříd, kde t_i je skutečné značení a p_i je pravděpodobnost softmax pro i-tou třídu [18].

2.4 Aktivační (přenosové) funkce- ACT

Aktivační, neboli přenosová funkce přidává nelinearitu do jednotlivých vrstev, čímž umožňuje řešit i komplexnější problémy. Aplikuje se na každou vrstvu po vynásobení matic (viz rovnice (8) a (10)). Aktivační funkce tak rozhoduje o tom, jestli neuron pošle dál a s jakou silou. Několik takových často používaných funkcí je popsáno níže.

2.4.1 ReLU

Funkce ReLU neboli usměrněná lineární jednotka (Rectified linear unit, někdy také rectifier) nabývá nulových hodnot pro záporné proměnné x a x pro nezáporné. Tedy

$$f(x) = max(0, x). \tag{4}$$

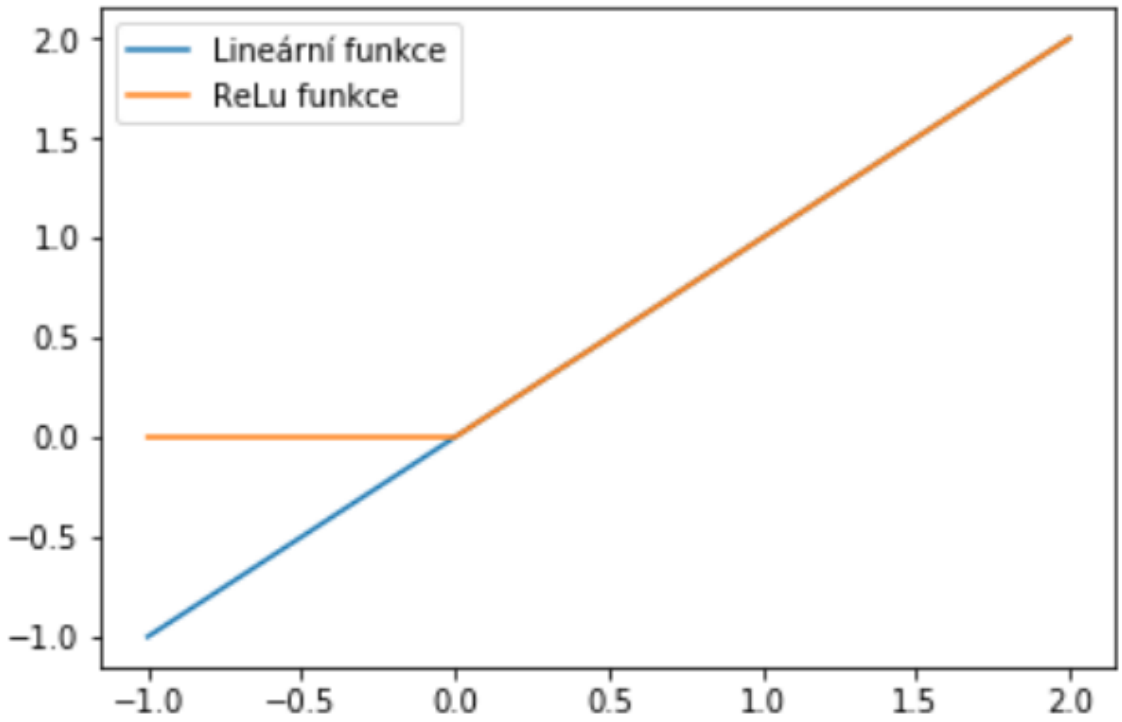
Mezi hlavní benefity ReLU patří zejména efektivní implementace a lepší propagace gradientů. Avšak největší problém může tvořit tzv. umírající ReLU (dying ReLU). To je forma mizejícího gradientu, kdy se deaktivuje vliv velkého počtu uzlů, čímž může docházet k nepřesným výsledkům [19].

2.4.2 Sigmoid

Aktivační funkce sigmoid, nebo také logistická funkce, transformuje vstupní hodnoty na čísla mezi nulou a jedničkou a je definována funkcí

$$S(x) = \frac{1}{1 + e^{-x}}. \tag{5}$$

Tato funkce v posledních letech klesá na popularitě kvůli nízkým, téměř nulovým hodnotám gradientu, čímž způsobuje velmi malý zpětný tok. Zároveň však může výrazně zpomalit trénovací čas.[20] [21]. Na druhou stranu se v některých implementacích objevuje jako aktivační funkce jednotlivých vrstev. Na poslední vrstvu se však již používá funkce jiná, například funkce softmax.



Obrázek 1: Funkce ReLU

2.4.3 Tangens hyperbolický

Další hojně používanou funkcí je hyperbolický tangens (tanh) s definičním oborem $(-1, 1)$. Hlavní výhodou této funkce je především diferencovatelnost. S touto funkcí se tak nejčastěji setkáváme při klasifikaci dvou tříd. Její definiční funkce má tvar

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1, \tag{6}$$

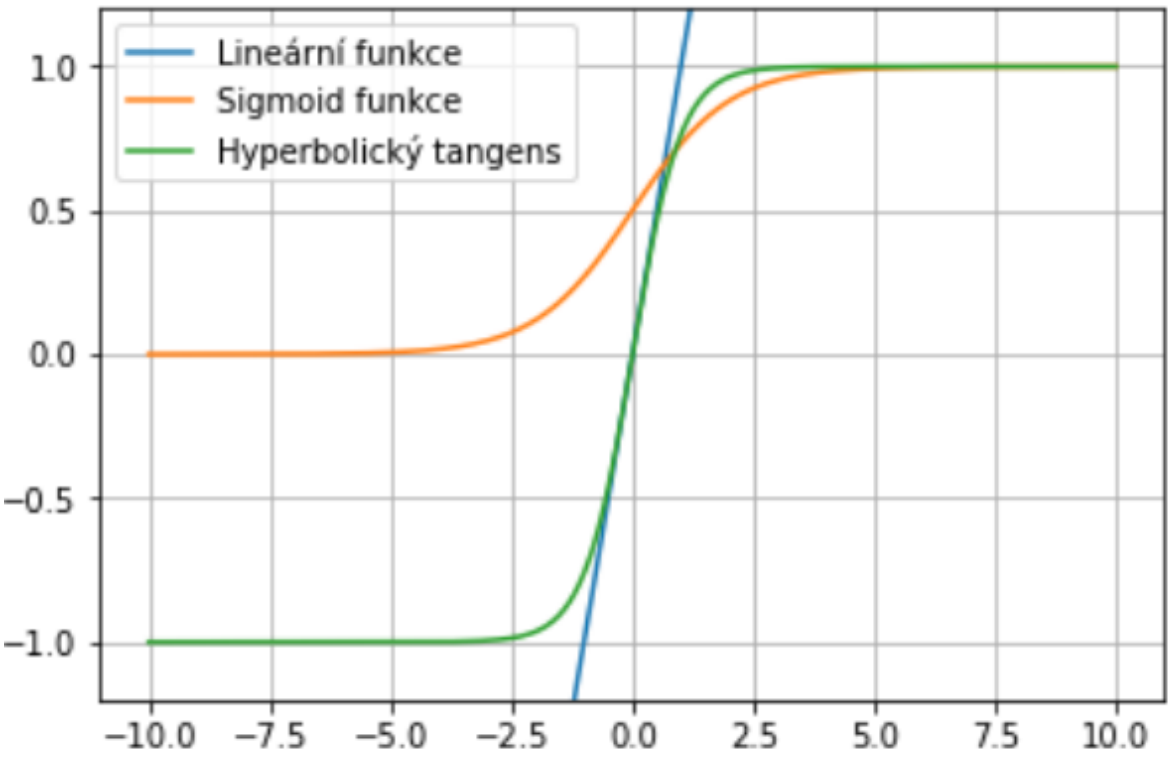
což je vlastně škálovaná sigmoid funkce, takže se opět můžeme potýkat s problémem mizejícího gradientu.

2.4.4 Softmax

Funkce softmax, neboli normalizovaná exponenciální funkce, vrací z vektoru čísel takový vektor, jehož hodnoty se nachází mezi 0 a 1 a jejich celkový součet je roven jedné. Funkce softmax se zpravidla aplikuje na poslední vrstvu GNN. Díky čemuž získáváme normalizovanou pravděpodobnost k příslušnosti jednotlivých tříd. Funkce je definována rovnicí

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{7}$$

kde $\vec{z}$ je vektor [22].



Obrázek 2: Funkce sigmoid a hyperbolický tangens

3 Teorie GNN

Neuronové sítě jsou, jak již bylo zmíněno, druhem strojového učení. Neuronová síť se skládá z tzv. neuronů (uzlů), které jsou propojeny a vzájemně si předávají informace. Využívají se například ke kompresi a rozpoznávání obrazů, či zvuků, popřípadě k předvídání dat. Hlavním rozdílem neuronových sítí například od lineárních regresních modelů jsou aktivační funkce.

3.1 Rozdělení GNN

GNN se dá rozdělit do tří základních skupin v závislosti na požadovaných výstupních datech. Výstupní data mohou záviset na uzlech, hranách a celkovém grafu.

U uzlů se setkáváme s jejich klasifikací, kde se rozpoznávají jednotlivé třídy uzlů v závislosti na vztazích a chování jejich okolí. Potom také regresí, kde se pro každý uzel predikuje spojitá hodnota. Či se setkáme se shlukováním, kde se podobné uzly rozdělují do disjunktních skupin.

Úlohy zkoumající hrany zjišťují existenci dalších hran, které reprezentují vztah mezi uzly, či jejich klasifikaci.

Na grafech se vyhodnocuje jak jejich klasifikace, regrese tak jejich shoda s dalšími grafy. U všech těchto typů je cílem naučit model reprezentaci grafu [23].

Dále se dají metody GNN dělit na transduktivní a induktivní. Transduktivní metoda je trénována na specifických případech, na základě kterých vyhodnotí závěr. Zatímco induktivní přístup vychází z obecných pravidel, ze kterých vyvodí závěr.

Induktivní učení tak vychází z trénování na označených datech a následně se model aplikuje na testovací data, s jejichž existencí předtím nepočítal. U transduktivního učení známe existenci všech uzlů již na začátku. Tudíž, ačkoli během trénování neznáme označení všech dat, víme jejich polohu a vazby s ostatními vrcholy. Proto je nutné transduktivní GNN trénovat pokaždé, když se graf jakkoli rozšíří [24].

Tato práce se zaměřuje zejména na učení se na uzlech. Konkrétněji jejich klasifikaci, kde je cílem najít či vymyslet takovou metodu, která bude mít nejoptimálnější výsledky.

3.2 Historie

První implementovanou neuronovou síť zabývající se grafy popsal Sperdutim a spol. (1997) [25]. Tehdy pomocí modelu rekurzivní neuronové sítě zpracoval stromovou strukturu dat, tedy acyklický graf, čímž započaly první výzkumy grafových neuronových sítí. Od té doby se začali objevovat další studie směřující k GNN. První neuronová síť označená jako GNN vznikla až v roce 2005 od Goriho a spol. [26] a navazovali další [27] [28] [29]. V těchto původních metodách se informace napříč sousedními uzly šířily iterativně, což vedlo k výpočetně náročným operacím, a první výzkumy se zaměřovaly právě na zlepšení tohoto postupu. Postupně se GNN vyvinula až do dnešní podoby, která je dále blíže popsána.

GNN se v průběhu let začala dále dělit do dvou základních skupin[30]: Rekurentní GNN (RecGNN) (do té řadíme zejména ty nejstarší metody [26] [27] [28]) a konvoluční GNN (ConvGNN). ConvGNN začaly vyvíjet po boku s RecGNN po úspěchu konvolučních neuronových sítí. Tato skupina se dělí ještě na prostorové GNN a spektrální GNN. Spektrální GNN byla dlouhou dobu přehlížena, ale v posledních letech se její popularita taktéž zvedá. Cílem aktuálních výzkumů tak nyní častokrát bývá i vzájemná kombinace těchto dvou přístupů.

Rekurentní GNN se opírá o Banachovu větu v pevném bodě, která říká

Definice 3.1. *Nechť (P, d) je neprázdný úplný metrický prostor $\mathbf{A} : P \longrightarrow P$. Pak existuje právě jeden prvek $x \in P$ takový, že $\mathbb{A}x = x$ [31].*

Tedy pokud budeme aplikovat mapování $\mathbf{A}$ na x k-krát, potom x^k bude téměř rovno x^{k-1}.

Prostorové konvoluční sítě se trochu podobají metodě CNN, která se využívá převážně ke klasifikaci obrázků [32]. Zde se využívá konvoluce okolních uzlů do jednoho, který je specifikován počtem uzlů a vah.

Spektrální GNN se opírá o teorii zpracování grafových signálů, aproximací a zjednodušení grafové konvoluce [30]. Celkově má tato skupina GNN metod silný matematický základ, který bude představen v následujících podkapitolách této sekce.

3.3 Zavedení grafu

Jak již bylo zmíněno, GNN je neuronová síť stavěná na grafech. Proto si nyní zavedeme několik základních pojmů, které budeme ve spojení s grafy využívat.

Definice 3.2 (Graf)**.** *Nechť nesměrový graph je označen jako $G = (V, E)$, kde V je množina vrcholů, nebo-li uzlů a E je množina hran. Nechť $v_i \in V$ značí uzel a $e_{ij} = (v_i, v_j) \in E$ značí hranu mezi v_i a v_j. Okolí uzlu v definujeme jako $N(v) = \{u \in V \mid (v, u) \in E\}$. Matice propojení $\mathbf{A}$ je matice $n \times n$, kde $A_{ij} = 1$, jestliže $e_{ij} \in E$ a $A_{ij} = 0$, jestliže $e_{ij} \notin E$. Graf může mít vlastnosti (features) uzlů $\mathbf{X} \in \mathbf{R}^{n \times d}$, kde $x_v \in \mathbf{R}^d$ reprezentuje vlastnosti vektoru uzlu v. Kdežto graf může mít vlastnosti hran $\mathbf{X}^c$, kde $\mathbf{X}^c \in \mathbf{R}^{m \times c}$ je matice funkcí hran, kde $x^e_{v,u} \in \mathbf{R}^c$ reprezentuje vektor vlastností hrany (v, u) [29].*

GNN pracuje jak na směrových, tak nesměrových grafech. V $\mathbf{X}$ můžou být zahrnuté různé atributy, texty, obrázky, stupně uzlů, značení různých skupin a tak dále [33]. Pokud se mluví o směrovém grafu, myslí se tím, že hrana $e_{ij} = (v_i, v_j) \in E$ vede z v_i do v_j ale ne naopak, pokud není řečeno jinak.

Dva uzly spojeny hranou nazveme svými sousedy. Každý uzel tak získává své sousedství, nebo-li okolí. To tvoří všechny uzly, které jsou s daným uzlem spojeny ($N(v) = \{u : \{v, u\} \in E\}$).

Stupně uzlů (vrcholů) značíme $deg(v)$ a myslíme tím velikost sousedství uzlu v, tj. $deg(v) = |\{u : \{v, u\} \in E\}|$.

Váhový graf je takový graf $G = (V, E)$, který má ke každé hraně přidělenou váhu, tedy číslo, popisující například cenu, vzdálenost, kapacitu atd. S takovým typem grafu se můžeme setkat i při řešení klasického problému obchodního cestujícího [34].

3.4 Rozdělení dat grafu

K aplikaci GNN je nutné vrcholy každého grafu rozdělit alespoň do tří základních skupin. Primárně se data dělí na trénovací, validační a testovací. Avšak můžeme pracovat i se čtvrtou skupinou dat, která nespadá do žádné z předešlých kategorií, pouze doplňuje strukturu grafu. Trénovací data jsou taková, jejichž označení známe. Na nich se trénuje celá GNN, která své výsledky koriguje podle daného značení. Validační data se využívají po každém běhu trénovacích dat. Opět známe jejich značení, ale jedná se o uzly, které nebyly využity v trénovací sadě. Zabraňují fixnímu naučení značení, tzv. přeučení. Tato data také zpřesňují predikci GNN a korigují počet běhů na základě hodnot ztrátové funkce. V neposlední řadě používáme testovací data, která se použijí až po trénování celé GNN. Může se jednat o neoznačená data, jejichž predikce nás zajímá, nebo o data, jejichž značení známe, ale trénovaná GNN se s jejich značením nesetkala. Opět se tedy jedná o data, která nebyla ani v trénovací ani ve validační sadě. V takovém případě po natrénování GNN spustíme predikci na testovacích datech. Výsledky porovnáme s reálným značením, čímž získáváme přesnost trénovaného modelu.

3.5 Princip GNN

GNN funguje ve čtyřech základních krocích

1. Předzpracování vrstev
2. Zpracování vrstev
3. Předání informací napříč vrstvami
4. Konečné zprocesování

V prvním a posledním kroku se využívá MLP (vícevrstvý perceptron). Ty se tak mohou optimalizovat zvlášť. Druhý krok můžeme dále rozdělit na Lineární část, Normování shluků (BN), Funkci odpadnutí (DROPOUT), Aktivační funkci (ACT) a Agregační funkci (AGG). Agregační funkce může být maximum, průměr, nebo součet. Pomocí těchto funkcí můžeme zpracování vrstev zapsat rovnicí

$$\mathbf{h}_v^{(k+1)} = AGG\left(\left\{ACT(DROPOUT(BN(\mathbf{W}^{(k)}\mathbf{h}_u^{(k)} + \mathbf{b}^{(k)}))),\ u \in N(v)\right\}\right), \tag{8}$$

kde $\mathbf{h}_v^{(k)}$ je k-tá vrstva uzlu v. U normování shluků se setkáváme s tím, že se v některých modelech vynechává [35].

Po zpracování mohou být vrstvy přímo spojeny do vícevrstvé GNN [36] [37]. Případně toto spojení může být přeskočeno například díky zbytkovému připojení SKIP-SUM [38] nebo propojením vnoření ve všech předchozích vrstvách díky SKIP-CAT [39].

Předání informací napříč vrstvami konfigurujeme pomocí [35]

1. Velikosti shluků (Batch size)
2. Míry učení

3. Optimalizace

4. Trénování

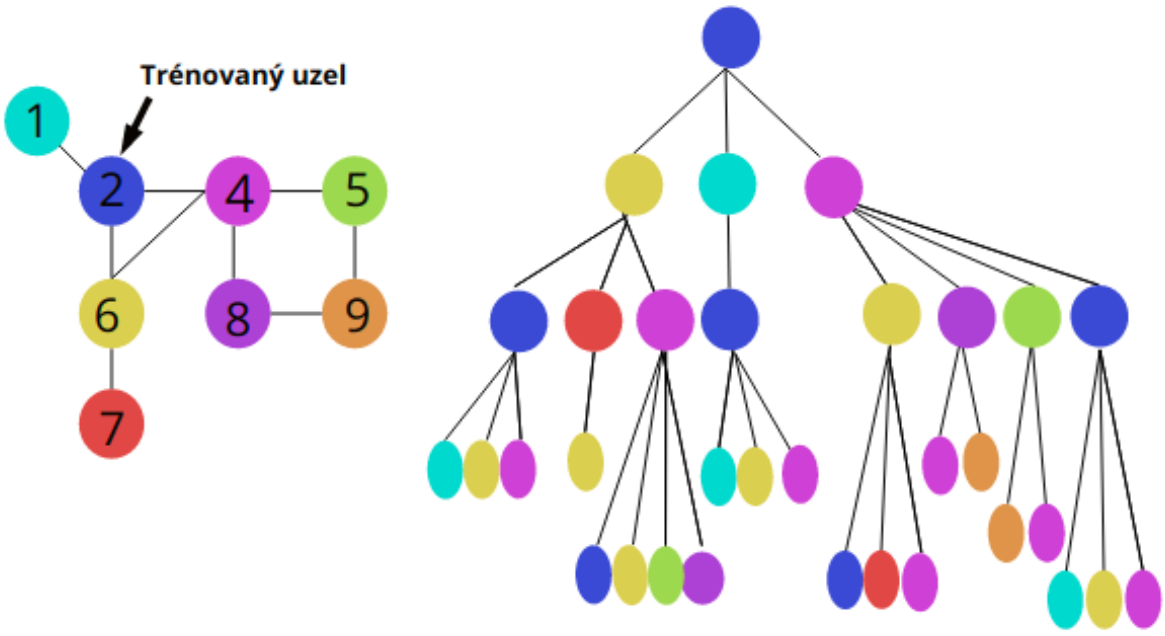
Trénování probíhá v několika vrstvách a na každou takovou vrstvu se aplikuje ztrátová funkce. Trénování můžeme dělit na kontrolované a nekontrolované povahy [33]. Mezi nekontrolovatelné ztrátové funkce patří například náhodné procházky, faktorizace grafu či blízkost uzlů v grafu [33]. Po několika bězích na trénovacích a validačních datech se GNN spustí na datech testovacích (takovéto rozdělení dat popisujeme v sekci „Tvorba grafu z reálných dat"). Na základě porovnání výsledků posledního běhu se skutečnými hodnotami může model vyhodnotit přesnost metody.

V otázce klasifikace uzlů počet vrstev určuje, jak hluboko se v grafu dostáváme od predikovaného uzlu. Předpokládá se, že každý uzel je nejvíce ovlivněn svými sousedními uzly (pokud matice **X** neříká jinak) a každý o krok vzdálenější uzel jej ovlivňuje o konstantu méně.

Při vyhodnocování jednotlivých uzlů postupuje GNN hlouběji do grafu pro získání relevantních informací o okolí predikovaného uzlu. V praxi se většinou setkáváme maximálně s třívrstvou GNN. Počet vrstev značí maximální hloubku zkoumaného okolí od klasifikovaného uzlu. V k-vrstvém GNN tedy půjdeme do uzlu vzdáleného k hran od původního uzlu ve všech směrech.

Na obrázku [3] je znázorněná třívrstvá GNN, která ilustruje, jak významně narůstá objem dat zpracovávaných pro každý uzel s přibývající vrstvou, tak vrstvení GNN.

Pro klasifikaci uzlu posbírá třívrstvá GNN informace z uzlů v nejvzdálenější vrstvě. Tak agreguje vnoření uzlů v druhé vrstvě, porovná výsledky se známými hodnotami a přidá další informace, které jsou k dispozici. Následně dopočítá hodnoty další vrstvy a nakonec vytvoří predikci pro trénovaný uzel. Tyto výsledky jsou v každém běhu porovnávány s dostupnými klasifikacemi trénovaných uzlů a tudíž vyhodnocena přesnost běhu.



Obrázek 3: agregace sousedních uzlů

3.6 Klasická GNN

Původní přístup k GNN se časem lišil a základní rovnice byla díky rozdílným přístupům a matematickým optimalizacím výrazně upravována. Princip, který se využívá, popisuje rovnice (8). Avšak původní klasická metoda GNN, od které se vyvinuly modifikované implementace, byla vytvořena zobecněním rovnice (8):

Definice 3.3 (GNN)**.** *Nechť h_v^n označuje n-tou vrstvu uzlu v, σ nelineární funkcí. Matice $\mathbf{W}_k$ a $\mathbf{B}_k$, jsou trénovací matice, kde matice $\mathbf{W}_k$ je váhová matice pro agregaci sousedů a matice $\mathbf{B}_k$ je váhová matice pro transformování skrytých vektorů daného uzlu. Pak vztah, mezi jednotlivými vrstvami zapíšeme ve tvaru [33]*

$$\begin{aligned} h_v^0 &= \mathbf{x}_v \\ h_v^k &= \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{h_u^{k-1}}{|N(v)|} + \mathbf{B}_k h_v^{k-1}\right),\ \forall k > 0. \end{aligned} \tag{9}$$

Takový vztah by se pro každý uzel zvlášť implementoval obtížně. Proto se využívá maticová podoba pomocí vztahu

$$\mathbf{H}^{(k+1)} = \sigma\left(\mathbf{H}^{(k)}\mathbf{W}_0^{(k)} + \tilde{\mathbf{A}}\mathbf{H}^{(k)}\mathbf{W}_1^{(k)}\right),\ \text{kde } \tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} \text{ a } \mathbf{D}_{ii} = \sum_j \mathbf{A}_{i,j}. \tag{10}$$

$\mathbf{A}$ je matice sousedností, $\mathbf{H}^{(k)}$ matice k-té vrstvy a σ je aktivační funkcí. V této rovnici $\mathbf{H}^{(k)}\mathbf{W}_0^{(k)}$ představuje vlastní transformaci a $\tilde{\mathbf{A}}\mathbf{H}^{(k)}\mathbf{W}_1^{(k)}$ představuje onu agregaci sousedů.

Funkce σ může být například funkce tanh nebo ReLU [36] [37]. Matice $\mathbf{B}_k$ bývá často nulová nebo v dalších implementacích vypouštěna [36]. Pokud ovšem nemáme vlastní váhovou matici, můžeme ji nahradit například maticí vytvořenou přes rovnoměrný Xavier inicializátor [40].

3.7 Rovnoměrný Xavier inicializátor

Rovnoměrný Xavier inicializátor, někdy nazývaný též rovnoměrný Glorot inicializátor, je inicializační schéma pro neuronové sítě. Zadefinujeme jej vztahem:

Definice 3.4 (Xavier)**.** *$\mathbf{W}_{ij}$ je v každé vrstvě definováno jako náhodné číslo s rovnoměrnou pravděpodobností U*

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right], \tag{11}$$

kde n je velikost předchozí vrstvy (počet uzlů) ve $\mathbf{W}$.

Normalizovaná váhová distribuce je opět náhodné číslo s rovnoměrným pravděpodobnostním rozdělením U

$$W_{ij} \sim U\left[-\sqrt{\frac{6}{n+m}}, \sqrt{\frac{6}{n+m}}\right], \tag{12}$$

kde n je opět velikost (počet uzlů) předchozí vrstvy a m je velikost výstupu (počet uzlů) z aktuální vrstvy [40].

Rovnoměrný Xavier inicializátor se využívá zejména kvůli potřebě vytvářet signál ve směru k predikci a současně ve zpětném směru pro šíření gradientů. Zároveň není žádoucí, aby byl tento signál extrémně velký, ale ani extrémně malý. Proto se rozptyl výstupů každé vrstvy rovná rozptylu jejich vstupů a přechody před a po protékání touto vrstvou mají stejný rozptyl i v opačném směru.

3.8 Ustálení pojmů

V předchozích kapitolách bylo zavedeno několik značení, která se budou často opakovat. Ta shrnuje a doplňuje tabulka 1. Zároveň obsahuje některé české názvosloví, které je často zaužívané v anglickém jazyce.

G	graf	shluk	batch
V	vrcholy / uzly	vnoření	embedding
v_i	uzel i	přeplnění	overfitting
E	hrany	nákladová funkce	cost function
e_{ij}	hrana mezi uzly i a j	propagace značení	label propagation
$\mathbf{A}$	matice sousednosti grafu	matice sousednosti	adjacency matrix
n	celkový počet vrcholů	křížová entropie	loss entropy
c	počet tříd	uhlazení	smoothing
$N(v)$	okolí uzlu v	graf spojitosti	connection map
$\mathbf{X}$	matice vlastností (features)	přeučení	overlearning
x_v	vektor vlastností uzlu v	MLP	multi-layer perceptron
$\mathbf{H}^{(k)}$	k-tá vrstva GNN celého grafu	AGG	agregační funkce
h_v^k	k-tá vrstva GNN vrcholu v	ACT	aktivační funkce
$\mathbf{W}$	váhová matice	$DROPOUT$	funkce odpadnutí
$\mathbf{D}_{ii}$	$\mathbf{D}_{ii} = \sum_j \mathbf{A}_{i,j}$	BN	normoví shluků
$\mathbf{Y} \in \mathbb{R}^{n \times c}$	matice značení (labels)	$LOSS$	ztrátová funkce
$\mathbf{Y}_O$	matice značení pro označené uzly	GNN	grafové neurnové sítě
$\mathbf{Y}_V$	validační matice značení	GCN	Grafové konvoluční sítě
$\tilde{\mathbf{A}}$	$\mathbf{A} + \mathbf{I}$		
$\hat{\mathbf{A}}$	$\mathbf{D}^{-\frac{1}{2}}\tilde{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}$	C&S	Propagace korekcí

Tabulka 1: Pojmy a značení

4 Druhy GNN

Nyní se blíže podíváme na prostorové a spektrální GNN. V praxi se mnohokrát setkáme s jejich kombinací a jejich vzájemné rozlišení není často jednoduché. Proto je dále nebudeme pro jednoduchost rozlišovat. Mezi základní implementace těchto GNN se řadí například GCN, GraphSage, FastGCN a GAT [36], [8], [37], [41], na které navazují další [42], [43].

4.1 Grafová konvoluční síť

Grafová konvoluční síť (GCN) [36], jak už název napovídá, je podtřídou GNN [44], ve které se využívá konvoluce. Hlavní myšlenkou je generovat vkládání uzlu na základě lokálního sousedství v síti. Každý uzel tak definuje svůj vlastní graf s jeho sousedy. Tyto výpočetní grafy dělíme do několika vrstev (viz obrázek [3]). Důležité je zjistit, jak moc rozdílné jsou souhrnné informace napříč jednotlivými vrstvami. Základní rovnice GCN má pouze lehkou odchylku od rovnice GNN (9) a zapisuje se ve tvaru [45]

$$h_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v) \bigcup v} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}} + \mathbf{B}_k h_v^{k-1}\right),\ \forall k > 0. \tag{13}$$

Hlavním rozdílem je především jmenovatel ve zlomku, který normalizuje přes každý sousední uzel. Zároveň používá stejnou transformační matici pro svoje vkládání a vkládání sousedů. Ekvivalentně s (10) můžeme rovnici zapsat efektivněji pomocí maticového zápisu

$$\mathbf{H}^{(k+1)} = \sigma\left(\mathbf{D}^{-\frac{1}{2}}\tilde{\mathbf{A}}\mathbf{D}^{-\frac{1}{2}}\mathbf{H}^{(k)}\mathbf{W}_k\right), \text{ kde } \tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I} \text{ a } \mathbf{D}_{ii} = \sum_j \mathbf{A}_{i,j}. \tag{14}$$

Matice $\mathbf{A}$ představuje matici propojení a $\mathbf{I}$ je jednotková matice.

4.2 GraphSAGE

Hlavním rozdíl metody GraphSAGE [8] je v tom, že nebereme (vážený) průměr sousedních uzlů, ale mapujeme set vektorů do jednoho vektoru.

Rovnici (9) tedy upravíme do tvaru [46]:

$$\mathbf{h}_v^k = \sigma\left([\mathbf{A}_k\, AGG(\{\mathbf{h}_u^{k-1}, \forall u \in N(v)\}),\, \mathbf{B}_k\mathbf{h}_v^{(k-1)}]\right). \tag{15}$$

V rovnici je znázorněná zobecněná agregace, která se v mnohých implementacích liší. Zároveň obsahuje vnoření (embedding) sebe samé a sousedních uzlů.

Nejzákladnější a asi nejjednodušší varianty agregace jsou:

$$\begin{aligned} \text{Průměr: } AGG &= \sum_{u \in N(v)} \frac{\mathbf{h}^{k-1}}{|N(v)|} \\ \text{Sdružování (pooling): } AGG &= \gamma\left(\{\mathbf{Q}\mathbf{h}_u^{k-1}, \forall u \in N(v)\}\right) \\ \text{LSTM: } AGG &= LSTM([\mathbf{h}_u^{k-1}, \forall u \in \pi(N(v))]) \end{aligned} \tag{16}$$

Agregace průměru bere vážený průměr okolí. Sdružování transformuje sousední vektory a aplikuje symetrickou vektorovou funkci, kde funkce γ aplikuje elementární funkci průměru nebo maxima. Funkce $LSTM$ (z anglického Long Short-Term memory, tedy dlouhá krátkodobá paměť) [47] je aplikovaná na náhodnou permutaci sousedů [46].

4.3 Sítě zaměřené na grafy

Sítě zaměřené na grafy (Graph attention networks - GAT) [37] se zaměřují zejména na jednodušší agregaci sousedních uzlů a implicitní definování váhového faktoru α_{vu}, který se definuje jako

$$\alpha_{vu} = \frac{1}{|N(v)|}, \tag{17}$$

kde $N(v)$ představuje agregovanou informaci napříč sousedními uzly z uzlu v. V GCN je α_{vu} definována explicitně v závislosti na strukturních vlastnostech grafu a počítá s tím, že všechny uzly v okolí uzlu v jsou stejně důležité. Naopak GAT pro jednotlivé sousedy specifikuje rozdílnou důležitost. Proto lze hlavní myšlenku popsat jako snahu vypočítat vnoření (embedding) h_v^k pro každý uzel tak, aby uzly ovlivňovaly $N(v)$, ale zároveň pro různé uzly specifikovaly rozdílné váhy.

Definice 4.1. *Nechť a je mechanismus pozornosti, který počítá koeficient pozornosti e_{vu} z uzlu u vzhledem k uzlu v.*

$$e_{vu} = a\left(\mathbf{W}_k\mathbf{h}_u^{k-1},\, \mathbf{W}_k\mathbf{h}_v^{k-1}\right) \tag{18}$$

Normlaizační koeficient α_{vu}, využívající softmax funkci, která umožní porovnávat rozdílné sousedství, je popsána rovnicí

$$\alpha_{vu} = \frac{exp(e_{vu})}{\sum_{k \in N(v)} exp(e_{vk})}. \tag{19}$$

Potom

$$\mathbf{h}_v^k = \sigma(\sum_{u \in N(v)} \alpha_{vu} \mathbf{W}_k \mathbf{h}_u^{k-1}). \tag{20}$$

Mechanizmus pozornosti a může používat jednoduchou jednovrstvou neuronovou síť, nebo může mít odhadnuté parametry. Parametry a jsou trénovány společně s váhovými maticemi.

Tento mechanizmus pozornosti stabilizuje vlastní pozornost učícího se procesu vícehlavou pozorností. To znamená, že operátor pozornosti je v každé vrstvě R-krát nezávisle replikován s různými parametry a výstupy jsou agregovány sčítáním nebo spojováním.

Výpočetní čas tak lze zkrátit díky paralelnímu výpočtu koeficientů pozornosti napříč hranami grafu a agregací napříč všemi uzly. Zároveň počet parametrů nezávisí na velikosti grafu, čímž se redukuje náročnost na paměť. Výpočet není závislý na celkové struktuře grafu, což vede k tomu, že můžeme využít pouze nejbližší okolí sítě, a tak snížit výpočetní náročnost vyhodnocováním predikce pouze na potřebném podgrafu [33].

4.4 Bránová grafová neuronová síť

Hlavní myšlenkou bránové GNN (GatedGNN) je, že uzly agregují informace („zprávy") z jejich sousedů, za použití GNN [44].

Klasické GCN a GraphSAGE se implementuje v mnohých článcích v různých obměnách ve dvou či třech vrstvách. Zejména kvůli náročnosti na paměť a čas (viz dále). Proto se bránová grafová neuronová síť snaží využít rekurentní neuronové sítě, díky čemuž by se vyhnula přeučen9 kvůli mnoha parametrům a mizícím nebo extrémně velkým gradientům během zpětné propagace.

Jedním z nápadů je sdílení parametrů napříč vrstvami, tedy agregace sousedů obdobně jako u GraphSAGE. Dalším je rekurentní aktualizace stavu za pomoci RNN.

Matematicky vyjádříme předávání informací od sousedů v jednom kroku k následovně:

$$\mathbf{m}_v^k = \mathbf{W} \sum_{u \in N(v)} \mathbf{h}_u^{k-1}, \tag{21}$$

kde tato agregační funkce nezávisí na k.

Aktualizace stavu se provádí přes bránovou rekurentní jednotku (GRU z anglického Gated Recurent Unit) [48], kde nový stav uzlu závisí na starém stavu a informacích od jeho sousedů

$$\mathbf{h}_v^k = GRU(\mathbf{h}_v^{k-1}, \mathbf{m}_v^k). \tag{22}$$

Díky těmto úpravám je možné modelovat více než 20 vrstev, nebo pracovat s menším množstvím použité výpočetní paměti.

4.5 Poziční GNN

Zajímavou myšlenku, která inspirovala tuto práci, představili v článku [49]. Zde nepracují pouze se strukturálním rozvržením grafu tak, jak je to běžné ve většina grafových neuronových sítí, ale také s pozicí jednotlivých uzlů. Tím se nám otevírá celá nová třída GNN. Tento konkrétní model nese zkratku P-GNN.

Velice zjednodušeně se v prvním kroku zvolí základní set uzlů. Každý další uzel si dopočítá vzdálenost k těmto uzlům. Učení pak probíhá přes nelineární agregační schéma za využití vážených vzdáleností. Jednou z výhod této metody je například začlenění vlastností uzlů do grafového učení.

Úvaha, která vedla k sestavení tohoto modelu je založena na tom, že pokud máme dva uzly, které si jsou v grafu velice vzdáleny, ale mají podobnou strukturu sousedů, pak budou v klasické GNN označeny stejně. Tedy za předpokladu, že zanedbáme vlastnosti uzlů. Protože v některých grafech, jako je například PPI dataset [50], jsou informace o pozicicích uloženy právě zde.

Definice pozičního a strukturálního vnoření uzlů jsou následující:

Definice 4.2. *Vnoření uzlů* $\mathbf{z}_i = f_p(v_i), \forall v_i \in \mathcal{V}$ *je poziční, jestliže existuje funkce* $g_p(\cdot,\cdot)$ *jako* $d_{sp}(v_i, v_j) = g_p(\mathbf{z}_i, \mathbf{z}_j$*, kde* $d_{sp}(\cdot,\cdot)$ *je nejkratší vzdálenost v G.*

Definice 4.3. *Vnoření uzlů* $\mathbf{z}_i = f_{s_q}(v_i), \forall v_i \in \mathcal{V}$ *je strukturální, jestliže se jedná o funkci sítě sousedství uzlu* v_i *až o q krocích. Specificky* $\mathbf{z}_i = g_s(N_1(v_i), \ldots, N_q(v_i))$*, kde* $N_k(v_i)$ *je set uzlů (sousedství) vzdálených k kroků od uzlu* v_i*.* g_s *může být jakákoli funkce.*

Jinými slovy, vnoření uzlů bývá ve většině GNN počítáno na základě shromáždění informací z uzlů vzdálených q kroků, ale už se nezajímáme o rozdělení zbylého grafu. Proto jsou strukturální. Na druhou stranu metody založené na náhodném krokování, jako například Node2Vec a DeepWalk jsou poziční, neboť předpokládají, že uzly, které si jsou blíže, budou mít bližší prostor vnoření.

Výstup vnoření P-GNN je právě poziční. Informace potřebné k provázání uzlů na základě strukturálního rozdělení jsou obsaženy v každé dimenzi výstupních dat, aby se propojily také uzly z jiných částí grafu. Problematické je ale plnění vstupních dat do další vrstvy. V tomto případně totiž nelze využít výstup předchozí vrstvy, kvůli tomu, že dimenze $\mathbf{z}_i$ může být obměňována. Navíc pozice uzlů je závislá na výběru základních uzlů. Proto se využívá výpočtu, který využívá informace skrze základní set uzlů. Čímž se do modelu dostává také strukturální složka, díky jejíž agregaci můžeme pokračovat do další vrstvy.

Nejkratší q kroková vzdálenost je zavedena pomocí

$$d_{sp}^{q}(v,u) = \begin{cases} d_{sp}(v,u), & \text{jestliže } d_{sp}(v,u) \leq q, \\ \infty, & \text{jinak,} \end{cases} \tag{23}$$

kde d_{sp} je nejkratší vzdálenost mezi uzly u a v. Pomocí matice sousednosti lze rovnou vyčíst jednokrokové vzdálenosti. Dále se ještě vzdálenosti přeškálují pomocí

$$s(v,u) = \frac{1}{d_{sp}^{q}(v,u) + 1} \tag{24}$$

na rozsah $(0, 1)$.

Zřetězením ($CONCAT$) nebo součinem se pak může dojít ke spojení pozičních informací a vlastností. Například přes funkci F

$$F(v, u, \mathbf{h}_v, \mathbf{h_u}) = s(v,u) CONCAT(\mathbf{h}_v, \mathbf{h}_u), \tag{25}$$

kde $\mathbf{h}_v$ a $\mathbf{h}_u$ jsou strukturální informace pro jednotlivé uzly.

5 Redukce složitosti

Ač metoda grafových neuronových sítí dosahuje skvělých výsledků, přináší s sebou mimo jiné i několik nevýhodných vlastností, které se projevují zejména na větších grafech. Trénování velkých grafů

potřebuje při klasické GNN metodě velké množství paměti. To zejména z toho důvodu, že při klasické GCN je nutné uchovávat v paměti celý graf najednou. Tak se například s metodou GCN u OGB datasetu [51] dostáváme na GPU 33 GB paměti, což je pro komerční použití nepraktické. Další překážkou GNN na velkých grafech je výpočetní doba, která se s rozšiřujícím se grafem zvětšuje. Tím se značně prodražuje provoz GNN a není v takovém případě únosné jej pravidelně využívat.

Složitost některých známých modelů jsou vyjádřeny v následující tabulce, kde L představuje počet vrstev, N počet uzlů, $||A||_0$ je počet nenulových čísel v matici propojení a F nese počet vlastností, přičemž F uvažujeme konstantní pro všechny vrstvy. Dále b představuje velikost skupiny (batch) a r počet vybraných sousedů pro každý uzel. Ve výpočtu je zanedbána paměť potřebná k ukládání grafu a podgrafů. Jejich velikost bývá zpravidla fixní a proto tato část nebývá velkým problémem [52].

	GCN [36]	GraphSAGE [8]	Cluster-GCN [52]
časová nároč.	$O(L\|\|A\|\|_0F + LNF^2)$	$O(r^LNF^2)$	$O(L\|\|A\|\|_0F + LNF^2)$
nároč. na paměť	$O(LNF + LF^2)$	$O(br^LF + LF^2)$	$O(bLF + LF^2)$

Tabulka 2: Tabulka složitosti [52]. L = počet vrstev, N = počet uzlů, $||A||_0$ = počet nenulových čísel v matici propojení, F = počet vlastností, b = velikost skupiny, r = počet sousedů, $|E|$ = počet hran v grafu.

2

5.1 Rozdíly v přístupech k redukci

V literatuře a dostupných modifikací GNN lze najít mnoho přístupů, jak přistupovat ke snížení časové a výpočetní náročnosti. Po menší analýze jsme pro jednoduchost tyto přístupy rozdělili do několika základních směrů. Ve skutečnosti se ale pouze zřídka setkáváme s aplikací pouze jednoho směru. Většinou se jedná o jejich kombinaci či propojení.

Rozdíly v přístupech redukce jsou

- předtrénování - využívá opakovaně základních matematických výpočtů tak, aby nebylo nutné je počítat znovu. V [53] se tak věnují správnému výběru trénovacích podgrafů tak, aby se zachovali strukturální a sémantické vlastnosti dat.

- grafová komprese - počáteční graf se uměle zmenší. Nadále se pak celá metoda pouští pouze na redukovaném grafu. Graf se může zjednodušit například podle jeho vlastností [54] nebo náhodně například pomocí metody Monte Carlo. Díky tomu metoda HAG [54] dokáže snížit počet agregací $1,5 - 6,3x$ v závislosti na grafu a tím snížit časovou náročnost.

- samplování (batchování) - nejdříve je zprocesován pouze určitý počet uzlů. Ať už náhodně vybraných či nalezených sofistikovanější metodou. Až se znalostí těchto dat se pokračuje i na zbylém grafu. Jedna z možností tvorby shluků je metoda Monte Carlo kterou využívají ve FastGCN [41]. Náhodné vybírání podgrafů pak bylo aplikováno např. v metodě Ripple walk [55], která znatelně prokazuje snížení náročnosti jak na čas, tak na paměť. V neposlední řadě stojí za zmínku i klastrová GCN [52], která je více popsána v sekci **??**.

- dekompozice grafu - vytvoří se několik menších podgrafů, na které se postupně pouští GNN, a na závěr se propojí zpátky do původního grafu. Na tuto metodu se velice často u známějších metod využívá Metis [56], který graf dokáže efektivně rozdělit.

- spojení několika přístupů nebo spojení s jinou metodou strojového učení - asi nejčastější přístup, kdy se využije více z výše zmíněných metod nebo se k GNN přidá jiná metoda jako například propagace značení (label propagation) [57].

Z poslední sekce můžeme namátkou vybrat několik příkladů. Například PCGCN- Částečně centrované zprocesování pro zrychlení grafové konvoluční sítě (PCGCN: Partition-centric processing for accelerating graph convolutional network) [58] využívá kombinaci dekompozice grafu a předtrénování. Díky tomu dosáhli zrychlení celého procesu až 8.8 krát vzhledem k GCN. GRIP- architektura akcelerátoru grafové neuronové sítě (GRIP- A grah neural network accelerator architecure) [59] rozděluje jednotlivé kroky do několika strojů a tím počítá více operací najednou. Jedná se o metodu, která je mimo jiné postavená i na změně hardwaru. Při použití metody GRIP tak stačí využít výkon necelých 5 W. Zároveň se zde využívá předtrénování. Dekompozici za použití vylepšeného Metisu a samplování implikovali v DistDGL: trénink distribuované grafové neuronové sítě v grafu miliardového měřítka (DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs) [60]. Tak bylo možné bez omezení kvality výsledků snížit čas jednotlivých běhů na 13 sekund pro grafy obsahující 100 milionů uzlů a 3 miliardy hran za použití 16 strojů.

5.2 Redukce grafu

V článku o škálování redukce grafové sítě kvůli efektivnějšímu využití aplikace GNN [61] si stanovili za cíl maximalizovat výsledky GNN a zároveň co nejvíce minimalizovat její náročnost.

Hlavní myšlenkou článku je zredukovat graf, na kterém je neuronová síť trénována, tím, že spojí několik uzlů a jejich hrany i vlastnosti do nového uzlu. Tím se zmenší paměť potřebná k uložení grafu a zrychlí se učící proces, neboť probíhá na menším grafu. Tyto výhody jsou ovšem na úkor přesnosti výsledků.

Slučování dvou uzlů do jednoho pro graf $G_i = (V_i, E_i, X_i, Y_i)$ a hranu $e = (v_X, v_Y), e \in E_i$ budou uzly $v_X \in V_i, v_Y \in V_i$ odstraněny a nahrazeny novými uzly a hranami

$$V_{i+1} = V_i \setminus \{v_X, v_Y\} \cup \{v_{XY}\}, \tag{26}$$

kde v_{XY} je nový uzel a pro nové hrany platí

$$E_{i+1} = E_i \setminus \cup\{(v_X, *)\}_{v \in N_i(v_x, v_y)}, \tag{27}$$

kde $([v_X, v_Y], *) = (v_X, *) \cup (V_Y, *)$. Sousedství nově vzniknuvšího uzlu pak můžeme zapsat následovně: $\mathbf{N}_i(v_X, v_Y) = \mathbf{N}_i(v_X) \cup \mathbf{N}_i(v_Y) \setminus \{v_X, v_Y\}$.

Sloučením několika uzlů musí být sloučeny i vlastnosti daných uzlů. Zároveň musí být zachována vlastnost, díky které mají všechny vlastnosti uzlů stejné dimenze. Tento problém byl vyřešen jednoduchým váhovým průměrem

$$x_{XY} = a(x_X, x_Y) = \frac{w_X x_X + w_Y x_Y}{w_X + w_Y}, \tag{28}$$

kde a je agregace a váhy w_A jsou počtu uzlů z původního grafu.

Podobným způsobem se agregovalo také značení uzlů v_X, v_Y. Ty byly přeznačeny

$$p_{yXY} = \frac{w_X p_{yX} + w_Y p_{yY}}{w_X + w_Y}, \tag{29}$$

kde tentokrát w_A značí počet trénovacích uzlů spojených do v_A a p_y je předchozí distribuce značení.

Další úkol, který bylo nutné vyřešit, je rozdělení predikcí do původních uzlů. Procházka a spol. zde využili toho, že sloučené uzly by měli mít v ideálním případě stejné značení, a tak tohoto předpokladu využili.

Tímto způsobem se vytvořil seznam hran, které se odstraní, čím se vytvoří zjednodušený model. Ovšem právě díky předpokladu, že sloučené uzly budou mít vždy stejné značení dochází nevyhnutelně k určitému počtu špatných predikcí, které je třeba minimalizovat. K tomu pomáhá mimo jiné právě pořadí odstraňovaných hran, které se budou rušit v přesně daném pořadí. Toto pořadí se počítá přes KL-divergenci a křížovou entropii na predikovaných uzlech.

Pro rychlejší postup a zmenšení výpočetního času se však uzly mohou spojovat i po více kusech najednou. Tím se zmenší jednak počet redukovaných grafů, tak velikost hierarchického stromu.

Při srovnání tohoto modelu spolu s logistickou regresí bylo dosaženo lepších výsledků právě tímto modelem na několika základních grafech.

6 Technická část

K celé práci byl použit programovací jazyk Python 3. V tomto jazyce se k programování neuronových sítí využívá buď balíček TensorFlow nebo PyTorch. Dále bylo využito balíčku NetworkX, který je určen na práci s grafy, díky čemuž mohlo být vykresleno několik obrázků a grafů. Ovšem právě díky většímu rozmachu GNN se v roce 2019 postupně začal vyvíjet Pytorch Geometric (zkr. PyG) [62] a především v posledním roce se přidávali zajímavé funkce, které výrazně zjednodušují implementace jak už známých GNN, tak tvorbu vlastních.

6.1 TensorFlow vs PyTorch

Pojďme se ale podívat, jaký je vlastně rozdíl mezi TesorFlow a PyTorch, neboť se s nimi setkáme pokaždé, když budeme hledat originální implementace modelů. Na první pohled by se mohlo zdát, že jediným rozdílem je délka zápisu. PyTorch umožňuje kód zapsat s pomocí několika funkcí na pár řádcích, zatímco v TensorFlow se rozepisuje každý krok zvlášť. Tento rozdíl je sice pravdivý a pro mnohé zásadní, ale skrývá za sebou několik dalších vlastností.

PyTorch vyvinuli v Meta Platforms (Facebook) a širší veřejnosti byl představen v roce 2016. Jeho velkou výhodou je bezesporu možnost kontroly kódu před jeho úplným dopsáním (po částech), díky čemuž rychle roste na popularitě. Využívá především knihovny Torch. Na rozdíl od TensorFlow, který pracuje na statických grafech, PyTorch pracuje na grafech dynamických a v minulosti měl výrazně méně vestavěných funkcí.

TensorFlow byl naopak vyvinut společností Google a uveřejněn v roce 2015. Jedná se o balíček vystavěný na knihovně Theano. Velkou oblibu si získal především ve výzkumu [63].

Obecně se udává, že je výrazně jednodušší se naučit v PyTorch. Ovšem jestliže člověk zná podrobněji matematiku schovanou za neuronovými sítěmi, má u TensorFlow větší přehled, co se ve kterém kroku děje a může upravovat i jemnější nuance.

6.2 Další knihovny používané k GNN

Další veřejně dostupnou knihovnou, která se specializuje na GNN je PyTorch Geometric (PyG [62]). Široké využití získala díky snadnému načítaní dat malých shluků, práci s vícečetnou GPU, distributivní možnosti učení a spoustě implementovaných metod. Jak již název napovídá je vystavěná za použití balíčku PyTorch, čímž činí PyTorch silnějším v rychlé implementaci známých, často používaných metod.

Mezi metody, které lze jednoduše využít pomocí PyG existují i metody, které byly implementovány v rámci této práce. Jedná se konkrétně o GCN, GAT, Cluster GCN, ale i jiné metody, které jsou mezi programátory četně využívány. Mezi ty známější se může řadit například Node2Vec, Graph SAINT nebo ChebConv.

Další knihovnou určenou jak pro PyTorch tak pro TensorFlow je Deep graph library (DGL [64]), která taktéž dobře škáluje na velkých grafech. Opět umožňuje použít vícečetnou GPU a distribuované školící infrastruktury. Mezi nabízené implementace se řadí například GAT nebo GCN.

DGL a PyG neumožňují pouze snazší implementaci populárních metod, ale i rychlejší a efektivnější hledání sousedů nebo dělení grafu. Rozdíl mezi oběma skupinami není veliký. V současné době má PyG zařazených více známých přístupů, ale co se týče dalších aspektů, tak záleží spíše na preferenci jednotlivých programátorů.

Dále stojí za zmínku třeba GeometricFlux.jl, který je vytvořen pro jazyk Julia. Jedná se o doplňující balíček Pytorch GNN opět určený pro práci v PyTorch nebo JAX, který na druhou stranu rozšiřuje TensorFlow.

Díky těmto přístupům se implementace základních GNN stává jednodušší, což vede k ještě větší popularizaci neuronových sítí a jejich použití i na menších projektech. Ovšem v této práci se budeme držet zejména balíčku PyG.

Co se týče redukce časové náročnosti je mnohem výhodnější používání knihovny TensorFlow. Z různých měření Chiang a spol. [52] zjistili, že Tensorflow oproti PyTorch nabízí výraznou časovou redukci v každém běhu. Pokusy vytvořené na grafu Amazon(334 863 uzlů a 925 872 hran) vykazovaly 3.5 krát nižší čas (2.53 s ku 8.81 s) potřebný ke každému běhu při používání TensorFlow oproti PyTorch při 128 skrytých jednotkách. Při 512 skrytých jednotkách byl časový rozdíl více než šestinásobný (40.08 s ku 7.13 s). Tento jev byl později potvrzen i námi provedenými experimenty v rámci implementace GAT. Na druhou stranu, vzhledem k tomu, že práce je testována na poměrně malém grafu, nebylo třeba v takové míře snižovat náročnost na paměť na čas. Proto jsme dále využívali předimplementovaných modelů z PyG.

7 Praktická část

Cílem práce bylo najít ideální kombinaci přístupu redukování grafu popsané v sekci 5.2 za využití poziční a strukturální GNN. Tedy zkombinovat oba dva přístupy a najít ideální nastavení parametrů.

K dispozici jsme měli zdrojový kód k metodě redukce grafu popsané v sekci 5.2, ze kterého se vycházelo. Proto bylo nutné se se zdrojovým kódem nejdříve dobře seznámit.

Do této metody bylo také vkládáno několik různých druhů GNN. Konkrétněji GCN, SAGE a GAT se třemi hlavami. V minulosti byly různé formy GNN implementovány především pomocí Pytorch.

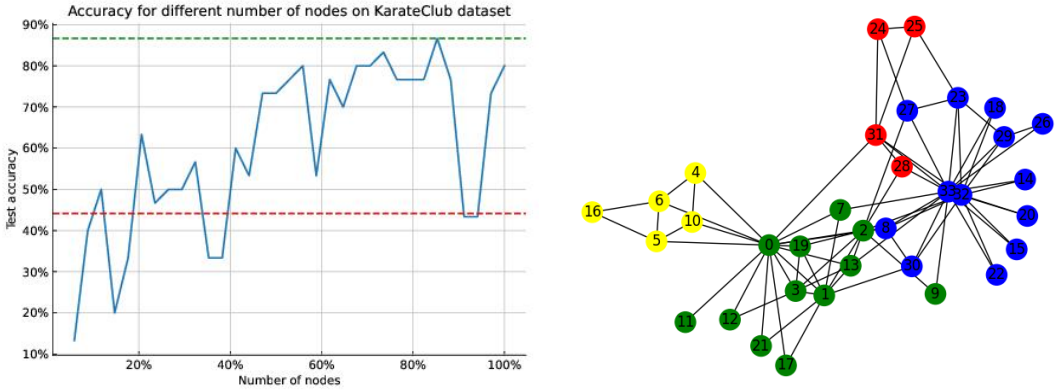
7.1 Redukce grafu a GNN na Karate Clubu

Abychom si dokázali lépe vizualizovat, jak škálování redukce GNN funguje, rozhodli jsme se ukázat si jednotlivé kroky na grafu Karate Clubu. Tento graf byl zvolen především díky dobrému členění na skupiny, ale také kvůli malému počtu uzlů, čímž docílíme vyšší přehlednosti.

Pro tuto vizualizaci jsme využili nastavení se slibnými výsledky z testování na datasetu Cora (viz dále). Následně bylo vybráno i několik jednotek dalších náhodných nastavení, které ale nesli obdobné výsledky. Proto si ukážeme krokování po jednom uzlu s neuronovou sítí GCN.

V první řadě je nutno podotknout, že vyhodnocování výsledků bylo poměrně chaotické a obsahovalo několik propadů i u více plného grafu. Nejlepších výsledků se také nedosáhlo při plném grafu ale

u 82% načtených uzlů (zbylé uzly byly spojovány k sobě dle popisu v sekci Redukce grafu 5.2). Maximální dosažená přesnost je vyznačena zeleně. Červenou barvou je vyznačen výsledek lineární regrese na tomto datasetu. Vše je vyobrazeno na obrázku 4 spolu se správným rozvržením skupin. Každá skupina je vykreslena jinou barvou.

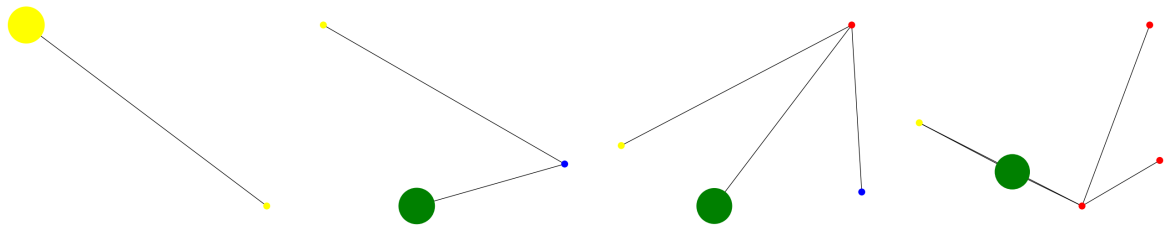


Obrázek 4: Zacharův karate klub- vyhodnocení a ukázka grafu

Na obrázku 5 jsou vyobrazené čtyři nejmenší kroky rozkládání grafu. Každá barva určuje náležitost do dané skupiny a velikost uzlů počet sloučených uzlů do daného uzlu. Tedy čím větší bod, tím více uzlů se pod ním skrývá. Z toho vidíme, že přesnost v maximálně zredukovaném grafu může být maximálně tak velká, kolik žlutých uzlů je celkem v celém datasetu.

Krásně ukázané jak reaguje graf na přidání jednoho uzlu ukazují obrázky 6. Zde lze vidět, jak se ze z velkého modrého uzlu odtrhne červený uzel, který se následně umístí nahoře v pravé části. Spolu s oddělením uzlu se také změní počet hran spojených se modrým uzlem a převážou se na daný červený. Zde si můžeme uvědomit například fakt, že v modrém uzlu byl schován také uzel jiné skupiny, který byl ale v předchozím kroku označen stejně. Tím byla snížena přesnost modelu.

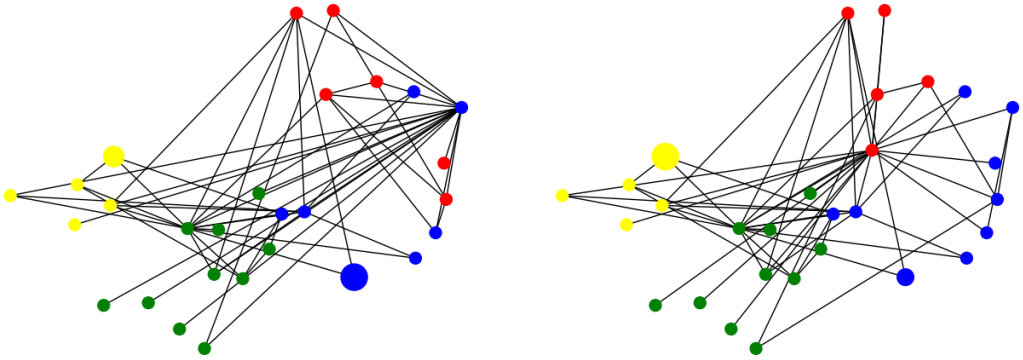
V posledních obrázcích 7 Pak můžeme porovnat predikci GCN a reálná data grafu. Vidíme především zaměnění červeného a modrého uzlu a špatnou predikci zelené části grafu. Ovšem z obrázku 4 vidíme, že úplný graf nenabízí maximální dosaženou přesnost. Tu vidíme na předchozím obrázku vpravo.



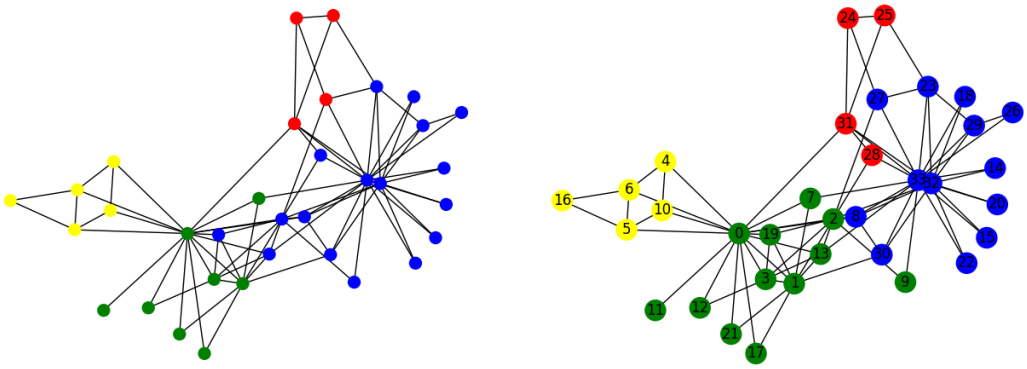
Obrázek 5: čtyři nejmenší grafy

Během testování bylo zjištěno, že není možné mít v trénovací sadě datasetu uzly pouze jedné třídy, neboť by model označil všechny další uzly touto třídou (jak lze odhadnout i z obrázků 4). Tento jev nebyl zrovna žádoucí, protože je tím pádem nutné, aby v trénovací sadě podgrafu byl obsažen vždy alespoň jeden uzel z každé třídy.

Dále bylo zajímavým úkazem pozorovat, které informace nám do grafu přináší strukturální a které poziční výpočty grafu. Závěrečné predikce je pak počítána vynásobením těchto dvou přístupů. Struktu-



Obrázek 6: odtržení červeného uzlu od zeleného



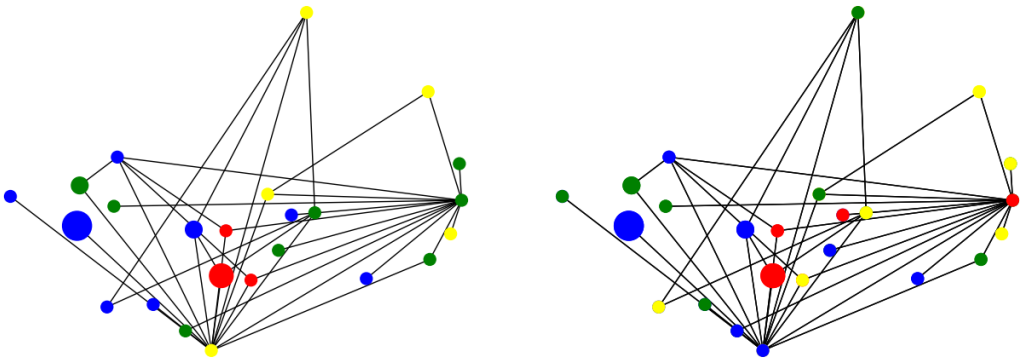
Obrázek 7: Porovnání výsledků

rální výpočet byl prováděn pomocí label refinement a pracuje s celým rozložením grafu. Poziční část pak počítá klasická GCN na zredukovaném grafu.

Label refinement tak pracuje neustále s celým grafem a ne pouze se shluky několika uzlů. Na obrázku 8 vidíme, že několik uzlů bylo značeno odlišně, což vede k chybám v celkových výsledcích. Při bližším prozkoumání pak bylo také zjištěno, že pokud se ve shluku vyskytuje pouze jeden druh uzlů, pak dokáží oba přístupy identifikovat tuto skupiny správně. Nachází-li se tam ale více druhů, pak je strukturální schéma dokáže rozeznat a s vyšší pravděpodobností ohodnotit správně vice uzlů, než poziční schéma, které převezme pro všechny značení stejnou skupinu. Problém tak pak nastává zejména v případě, kdy se ve skupině objevuje více tříd, s čímž si klasický součin výsledků nedokáže nejlépe poradit. Tím získávají i méně početné třídy větší sílu při určování výsledků.

7.2 Hledání optimálního nastavení

Této části bylo věnováno poměrně dost prostoru. Na datasetu Cora bylo experimentálně zkoušeno několik desítek základních nastavení na modelu redukce grafu z 5.2. Pracovalo se se čtyřmi základními neuronovými sítěmi. Konkrétně s GCN, GraphSAGE, tříhlavou GAT a ChebConv. Dále se přenastavoval poměr trénovacích a testovacích dat a počet uzlů, které se sjednocovali v jednotlivých krocích redukce grafu.



Obrázek 8: Poziční vs strukturální rozdělení

Výsledky byly porovnávány navzájem mezi sebou, s lineární regresí a dále s sebou samými v rámci jednotlivých redukcí grafu.

Všechny metody, pokud není uvedeno jinak, používají optimalizační algoritmus Adam, míru učení s parametrem 0.01 a aktivační funkci ReLU. Pro ztrátovou funkci se využívá funkce křížové entropie.

7.2.1 GCN

První implementovanou metodou byla klasická grafová konvoluční síť (GCN) [36]. Tato metoda byla implementována pomocí balíčku PyG

Nastavení bylo ponecháno v doporučené podobě, tudíž je možné, že by se experimentálně dalo pro konkrétní graf najít ještě vhodnějšího primárního nastavení hodnot. Regularizace L2 byla nastavena na hodnotu $5e-4$, počet skrytých jednotek na 16 a míra odpadnutí na 0.5. Jako optimalizační algoritmus slouží funkce Adam se stupněm učení 0.01 a snahou minimalizovat ztrátovou funkci.

Všechny metody byly implementovány víceméně stejným způsobem za použití PyG funkce. Proto si jejich implementaci ukážeme pouze zde, u vzorového modelu GNN.

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

class GCN(torch.nn.Module):
    def __init__(self, pocet_vlastnosti, pocet_trid, skryte_jednotky = 16):
        super().__init__()
        self.conv1 = GCNConv(pocet vlastnosti, skryte_jednotky) #prvni vrstva
        self.conv2 = GCNConv(skryte_jednotky, pocet_trid)

    def forward(self, x, hrana_index):
        x = self.conv1(x, hrana_index) #prvni vrstva
        x = x.relu() #relu(D*A*D*X*W) = H[0]
        x = F.dropout(x, p=0.5, training=self.training) #p = mira odpoadnuti
        x = self.conv2(x, hrana_index) #druha vrstva
        return x.log_softmax(dim=-1)
```

Listing 1: GCN- nastavení vrstvy [65]

Vidíme, že implementace pomocí PyG je vcelku jednoduchá a především záměnou funkce *GCNConv* za jinou (např. SAGEConv pro funkci GraphSAGE), můžeme přecházet mezi jednotlivými neuronovými sítěmi.

Pro lepší představu, jak se přenáší matematické rovnice do kódu ovšem přináší TensorFlow. Především z toho důvodu, že jednotlivé kroky opravdu musíme popsat zvlášť a neexistuje předimeplementovaná funkce pro jednotlivé vrstvy GCN. Ukážeme si tedy jak taková implementace na základě článku [36] vypadá. Tím si ukotvíme základy GNN a jejich implementace. Tato část může velmi pomoci k bližšímu pochopení toho, jak psát i další neuronové sítě.

```
import tensorflow as tf

def vrstva(self):
        random_tensor = 1-self.drop  #drop = mira odpadnuti = 0,5
        x = self.vlastnosti
        random_tensor += tf.random_uniform(self.shape) #self.shape = tf.int32
        dropout_mask = tf.cast( tf.floor( random_tensor), dtype = tf.bool) #tenzor ve tvaru [false, true, true, false, ...]
        pre_out = tf.sparse_retain(x, dropout_mask)
        x = pre_out*1./(1-self.drop)  #nahodne nastavi prvky na 0, kvuli preplneni
        pre_sup = tf.sparse_tensor_dense_matmul(x, self.var['vahy_0']) #X*W[0]
        vrstva = tf.sparse_tensor_dense_matmul( self.adj, pre_sup) #D^(-1/2)*A*D^(-1/2)*X*W[0]
        vrstva = tf.nn.relu(vrstva) #relu(D*A*D*X*W)= H[0]]

        x = tf.nn.dropout(vrstva, 1-self.drop) #nahodne nastavi prvky na 0, kvuli preplneni
        pre_sup = tf.matmul(x,self.var['vahy_1']) #H[0]*W[1]
        x = tf.sparse_tensor_dense_matmul(self.adj, pre_sup) #H[1]=DAD*H[0]*W[1]
        vysledek = tf.nn.log_softmax(x)
        return vysledek
```

Listing 2: GCN- nastavení vrstvy [65]

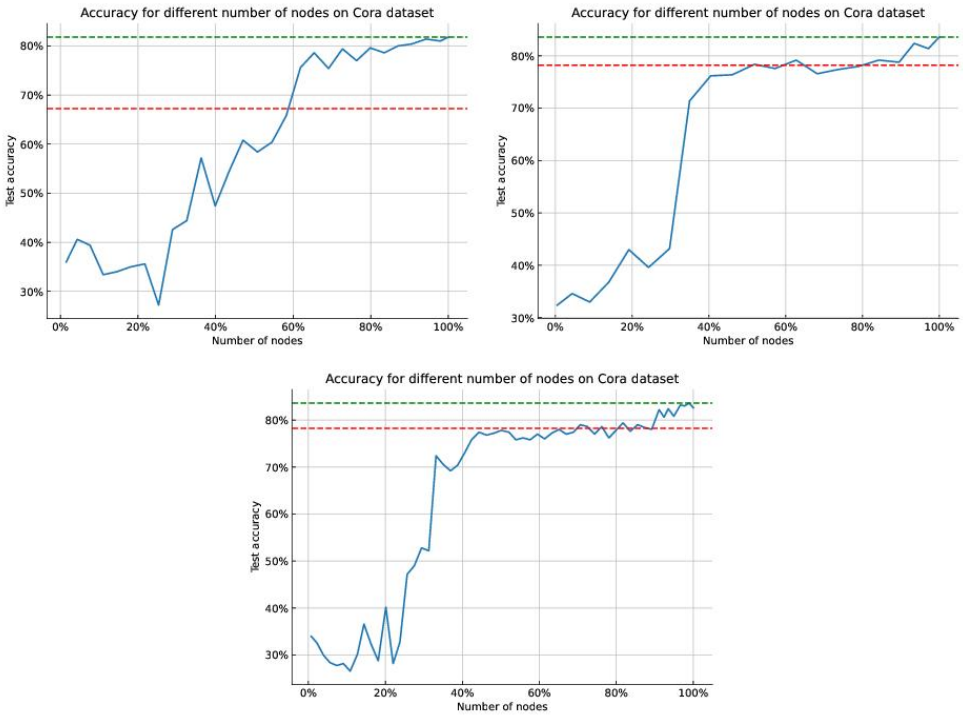
Můžeme si tak zde ukázat i několik triků, které se v GNN využívají. Jedná se především o přidání několika náhodným prvkům hodnotu 0. Tento trik se aplikuje především na matici vlastností, aby velká míra informací nepoškodila dobré výsledky. Metoda se následně nechává několikrát běžet střídavě na trénovacích a validačních datech. Během těchto běhů si hlídáme hodnoty ztrátových funkcí validačních dat a v případě, že hodnota ztrátové funkce je vyšší než průměr předchozích hodnot, bývá trénovací fáze ukončena, z toho důvodu, aby se model vyhnul přetrénovaní. V opačném případě cyklus s tímto nastavením proběhne 200x.

My ovšem dále využívali implementace v Torchi, který je využit v námi používaném článku abychom se vyhnuli zbytečným migracím. V několika hodnotách jsme dosáhli velice slibných výsledků již při výrazně zredukovaném grafu. Jako vzorový příklad budiž nastavení s poměrem trénovaných jednotek na 50%. Shlukování grafu bylo prováděno po 200, 150, 100 a po 50 uzlech a trénování proběhlo v 10 bězích.

Dle obrázků 9 můžeme vidět, že již u grafu zredukovaných na cca 40% původních uzlů se dostáváme na podobnou úroveň přesnosti odhadů, jako u referenční logistické regrese (značeno červeně). Zároveň nárůst přesnosti skokově stoupá pokaždé s podobnou velikostí redukce a další zvětšování datasetu již přesnost zlepšuje pouze pozvolna. Nejlepších výsledků však dosahuje vždy až téměř plném datasetu (maximální dosažená hodnota je vyznačena zeleně).

7.2.2 GAT

Další implementovanou metodou byla metoda GAT, která byla popsána již v sekci 4.3 Sítě zaměřené na grafy. Jedná se tedy o jeden ze základních přístupů GNN. Vzhledem k tomu, že na GAT navazovalo již několik novějších implementací, nebyla tato implementace prvoplánová. K jejímu zařazení se přistoupilo po zjištění, že vykazuje skvělé výsledky na velkých grafech i v dnešní době, a tudíž i dnes skrývá velký potenciál. Proto jsme tyto její vlastnosti chtěli vyzkoušet i v našich podmínkách.



Obrázek 9: GCN: multistep = 200, 150, 50

Implementace proběhla obdobně, jako implementace GCN. Pro GAT je ovšem, na rozdíl od GCN, potřeba nastavit i počet hlav. Ten jsme nastavili na 3 a míra odpadnutí nabývá hodnot 0.6. Dále byla jen změněna funkce GCNConv na GATConv, což je opět implementace z PyG.

Výše bylo zmíněno, že implementace v TensorFlow dokáže částečně snížit výpočetní náročnost. Tento jev jsme si potvrdili právě implementací GAT.

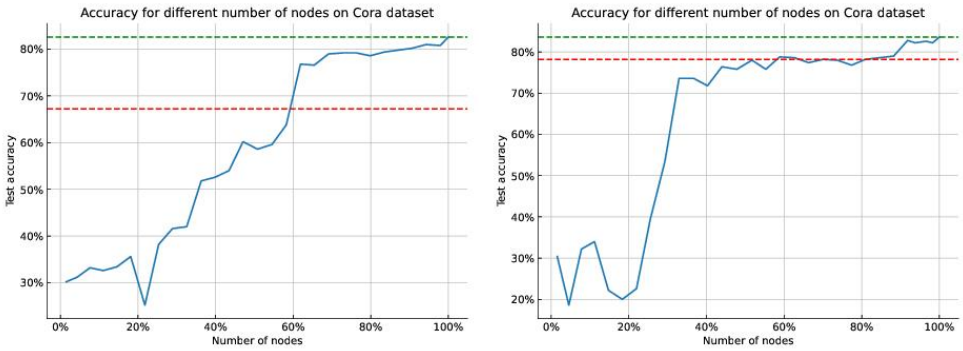
Samotná implementace pomocí dostupného zdrojového kódu [66] proběhla intuitivně bez větších problémů. Tato metoda běží za použití balíčku TensorFlow. Vychází ze základní implementace GCN s přidání koeficinetu pozornosti, který je určen:

```
import tensorflow as tf

if in_drop != 0.0:
    seq = tf.nn.dropout(seq, 1.0 - in_drop)

seq_fts = tf.layers.conv1d(seq, out_sz, 1, use_bias=False)

# nejjednodussi mozna vlastni pozornost
f_1 = tf.layers.conv1d(seq_fts, 1, 1)
f_2 = tf.layers.conv1d(seq_fts, 1, 1)
logits = f_1 + tf.transpose(f_2, [0, 2, 1])
coefs = tf.nn.softmax(tf.nn.leaky_relu(logits) + bias_mat)

vals = tf.matmul(coefs, seq_fts)
ret = tf.contrib.layers.bias_add(vals)

# rezidualni spojeni
if residual:
    if seq.shape[-1] != ret.shape[-1]:
```

```
20          ret = ret + conv1d(seq, ret.shape[-1], 1) # aktivace
21      else:
22          ret = ret + seq
```

Listing 3: GAT- pozornost [66]

Slibné výsledky opět přináší například implementace na 20% trénovacích datech. V tomto případě je zobrazeno slučování 100 uzlů do jednoho v každém kroku redukce grafu. Abychom nyní prezentovali důležitost grafových neuronorývch sítí, porovnáme si výsledky na jednotlivých podgrafech za využití GAT a za využití naivního Bayese.



Obrázek 10: multistep 100 - GAT vs NAIVE

Porovnání výsledků jednoznačně dokazuje, že

7.2.3 SAGE

Dalším implementovaným modelem byl GraphSAGE popsaný v sekci 4.2. Implementace rovněž proběhla obdobně jako s GCN se stejným nastavením parametrů.

Zajímavých výsledků jsme při této metodě nalezli hned několik. Názornou ukázkou nechť je třeba nastavení trénovacích uzlů na 20%. Opět zobrazujeme (obrázek 11) shlukování uzlů po 200, 150, 100 a 50 uzlech.

Vidíme, že ve všech případech již na grafu obsahujícím pouze 60% uzlů překonáváme výsledky lineární regrese (vyznačeno červeně) a okamžitě nám přesnost začíná tíhnout k maximálnímu možnému výsledku. Pokud si tedy můžeme dovolit míti o pár procentních bodů nižší preciznost, pak lze výrazně zredukovat výpočetní náročnost právě zmenšením grafu na 60 – 80%. Grafy ukazují, že maximálních hodnot opět dosahujeme až u téměř plného grafu.

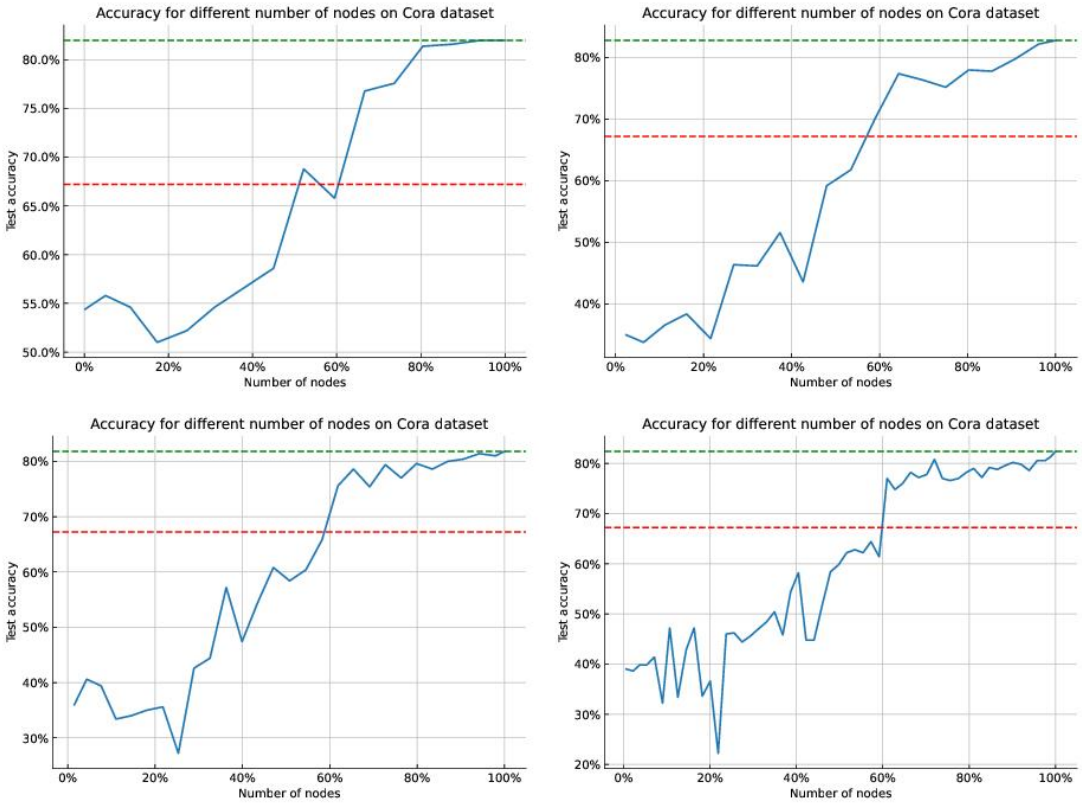
Grafy nám také ukazují, že je výhodnější shlukovat uzly po menších počtech, neboť přesnost značení pak dříve a výrazněji směřuje ke svému maximu.

7.2.4 Cebysevova konvoluce

Poslední implementovanou neuronovou sítí byla Chebišova konvoluce. Nastavení bylo identické s GCN. Pro tuto metodu je implementovaná funkce ChebConv.

U této metody jsme dosáhli nejvíce zajímavých výsledků, které umožňovali výrazné zmenšení grafu bez výrazné redukce přesnosti učení. Opět jako názornou ukázku volíme 20% trénovacích uzlů.

Podobně jako u GraphSAGE tento princip překonává základní metodu už okolo 60% grafu a skokově navyšuje přesnost výpočtů. Viz obrázky 12.



Obrázek 11: GraphSAGE: multistep = 200, 150, 100 a 50

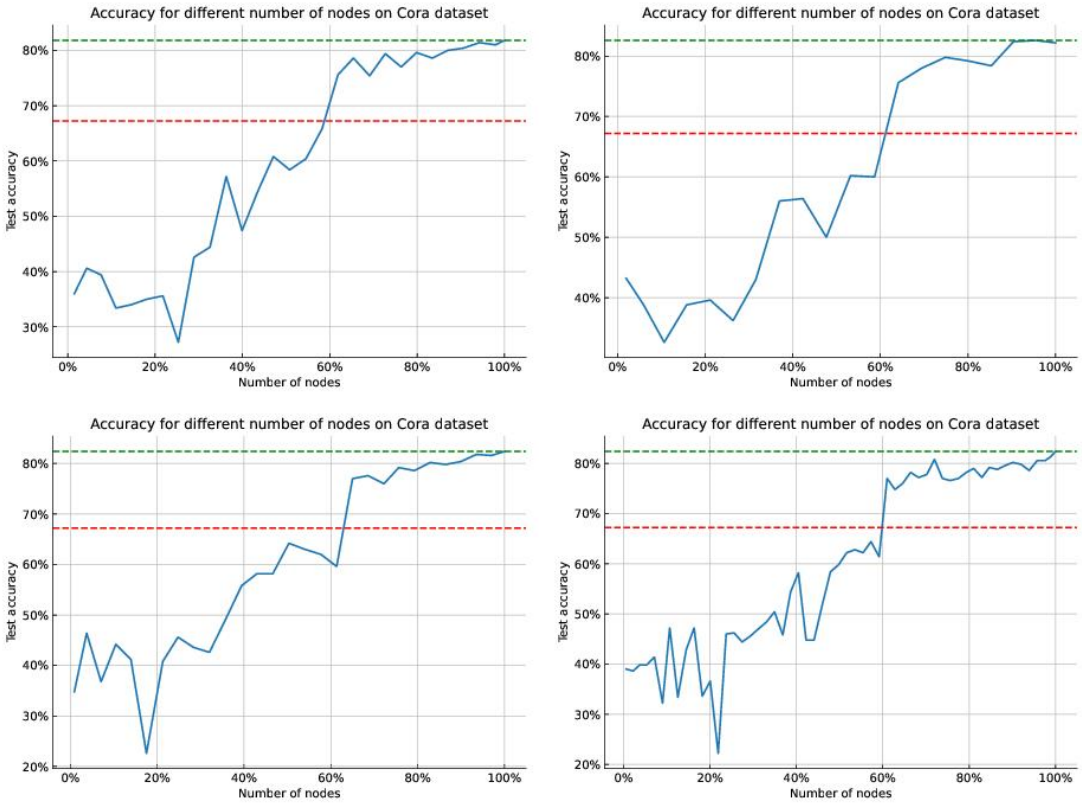
8 Závěr

Experimentálně se nám nepovedlo předčit výsledky publikované v [11], ovšem v několika desítkách případech jsme již při výrazně zredukovaném grafu schopni dosáhnout stejných, nebo dokonce lepších výsledků, než lineární regrese. Zároveň všechny pokusy ukázaly, že jakmile GNN na zredukovaném grafu překročí hodnoty lineární regrese, tak výsledky GNN skokově stoupají a blíží se svým maximálním hodnotám. Tudíž za cenu pouze malého snížení kvality výsledků dokážeme výrazně snížit časovou i výpočetní náročnost celého procesu grafických neuronových sítí. To je přínosné zejména pro velké grafy, které nelze nahrát do paměti najednou.

Reference

[1] S. Deoras, Why Graph Neural Networks Are Gaining Popularit In 2021. In analycsindiamag.com [online]. Copyright © 25. únor 2021, 2. ledna 2022 [cit. 02.01.2022]. Dostupné z: https://analyticsindiamag.com/why-graph-neural-networks-are-gaining-popularity-in-2021/

[2] S. Ivanov, Top Applications of Graph Neural Networks 2021. In: Medium – Where good ideas find you. [online]. © 14. leden 2021, 2. ledna 2022 [cit. 2022-02-01]. Dostupné z: https://medium.com/criteo-engineering/top-applications-of-graph-neural-networks-2021-c06ec82bfc18



Obrázek 12: Chebišev: multistep = 200, 150, 100 a 50

[4] J. Shlomi, P. Battaglia, J.-R. Vlimant, Graph Neural Network in Particle Physics. In 'Machine Learning: Science and Technology 2 021001', IOP Publishing, 2020.

[5] A. Jadhav, Applications of Graph Neural Networks. In Towards data science [online]. ©26.únor 2019, 2. ledna 2022 [cit. 2022-02-01]. Dostupné z: https://towardsdatascience.com/https-medium-com-aishwaryajadhav-applications-of-graph-neural-networks-1420576be574

[6] J. M. Stokes, K. Yang, K. Swanson, T. S. Jaakkola, R. Barzilay, A Deep Learning Approach to Antibiotic Discovery. Cell, Elsevier, 2020.

[7] Exciting Applications of Graph Neural Networks. Cloudera Fast Forward Blog [online]. © 30. říjen 2019, 2. ledna 2022 [cit. 2022-02-01]. Dostupné z: https://blog.fastforwardlabs.com/2019/10/30/exciting-applications-of-graph-neural-networks.html

[8] W. L. Hamilton, R. Ying, J. Leskovec, Inductive representation learning on large graphs. In 'Proceedings of the 31st International Conference on Neural Information Processing Systems', Curran Associates Inc., 2017, 1025-1035.

[9] N. Donges, 4 Reasons Why Deep Learning and Neural Networks Aren't Always the right Choice. Built In. [online]. Copyright © Built In 2022, 2. ledna 2022 [cit. 02.01.2022]. Dostupné z: https://builtin.com/data-science/disadvantages-neural-networks

[10] D. Parikh, Overview of Graph Neural Networks. OpenGenus IQ: Computing Expertise Legacy [online]. Copyright © 2022 All rights, 2. leden 2022 [cit. 02.01.2022]. Dostupné z: https://iq.opengenus.org/graph-neural-networks/

[11] P. Procházka, M.Mareš, M. Dědič, Scalable Graph Size Reduction for Efficient GNN Application. In 'CEUR Workshop Proceedings', 2022.

[12] C. Cabanes, A. Grouazel, K. von Schuckmann, M. Hamon, V. Turpin, C. Coatanoan, F. Paris, S. Guinehut, C. Boone, N. Ferry, C. de Boyer Montégut, T. Carval, G. Reverdin, S. Pouliquen, and P. Y. Le Traon, The CORA dataset: validation and diagnostics of in-situ ocean temperature and salinity measurements, In 'Ocean Science Journal', Springer nature 2013, 1-18.

[13] Zachary W. (1977). An information flow model for conflict and fission in small groups. Journal of Anthropological Research, 33, 452-473.

[14] Machine learning. In TechTarget [online]. 1.7. 2022 [cit. 01. 07. 2022]. Dostupné z: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML

[15] S. Doshi, Various Optimization Algorithms For Training Neural Network. In Towards Data science [online]. © 13. leden 2019, 2. ledna 2022 [cit. 02. 01. 2022]. Dostupné z: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

[16] Sanghvirajit, A Complete Guide to Adam and RMSprop Optimizer. Medium/ analytics vidhya [online]. © 20. únor 2021, 2. ledna 2022 [cit. 02. 01. 2022]. Dostupné z: https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be

[17] V. Bushaev, Adam - latest trends in deep learning optimization, Towards Data science [online]. © 22. říjen 2018, 2. ledna 2022 [cit. 02. 01. 2022]. Dostupné z: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c

[18] K. E. Koech, Cross-Entropy Loss Function. In Towards Data science [online]. © 2. říjen 2020, 2. ledna 2022 [cit. 02. 01. 2022]. Dostupné z: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

[19] Rectifier (neural networks). In Wikipedia the free encyklopedia [online]. © 9. listopad 2021, 2. ledna 2022 [cit. 02. 01. 2022]. Dostupné z:https://en.wikipedia.org/wiki/Rectifier_(neural_networks)

[20] S. Sharma, Activation Functions in Neural Netowrks. In Towards Data science [online]. © 6. září 2017, 2. ledna 2022 [cit. 02. 01. 2022]. Dostupné z: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[21] A. Sharma, Undertanding activation Function in Neural Networks. In Medium-The Theory of Everything [online]. © 30. březen 2017, 2. ledna 2022 [cit. 02. 01. 2022]. Dostupné z: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[22] T. Wood, Softmax Function. DeepAI: The front page of A.I. [online]. 2. ledna 2022, [cit. 02.01. 2022]. Dostupné z: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

[23] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, M. Sun, Graph neural networks: A review of methods and applications. In 'AI Open', 1, 2020, 57-81.

[24] J. Kuben, Anglická matematická terminologie / English mathematical terminologie [online]. 2. ledna 2022 [cit. 2022-02-01]. Dostupné z: http://matika.umat.feec.vutbr.cz/inovace/preklad/matematicky_slovnik.pdf

[25] A. Sperduti, S. Antonina, Supervised neural networks for the classification of structures. In 'IEEE Transactions on Neural Networks', IEEE, 1997, 714-735.

[26] M. Gori, G. Monfardini, and F. Scarselli, A new model for learning in graph domains. In 'Proc. of IJCNN, vol. 2.', IEEE, 2005, 729–734

[27] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, The graph neural network model. In 'IEEE Transactions on Neural Networks, vol. 20, no. 1', IEEE, 2009, 61–80.

[28] C. Gallicchio, A. Micheli, Graph echo state networks. In 'The 2010 International Joint Conference on Neural Networks (IJCNN)', IEEE, 2010, 1–8.

[29] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, P. S. Yu, A Comprehensive Survey on Graph Neural Networks. In 'IEEE Transactions on Neural Networks and Learning Systems', IEEE computational intelligence society, 2020, 1-22.

[30] S. Hong, An Introduction to Graph Neural Netowrk (GNN) For Analysing Structured Data. Towards data science [online]. © 5. březen 2020, 2. ledna 2022 [cit. 2022-02-01]. Dostupné z: https://towardsdatascience.com/an-introduction-to-graph-neural-network-gnn-for-analysing-structured-data-afce79f4cf

[31] Banachova věta o pevném bodě. In Wikipedia the free encyklopedia [online]. © 5. srpen 2021, 2. ledna 2022 [cit. 02. 01. 2022]. Dostupné z: https://cs.wikipedia.org/wiki/Banachova_v$\%$C4$\%$9Bta\_o\_pevn$\%$C3$\%$A9m\_bod$\%$C4$\%$9B

[32] J. Murphy, An overview of convolutional neural network architectures for deep learning. In 'Microway Inc', 2016, 1-22.

[33] J. Leskovec, Graph Neural Networks [online]. 10. říjen 2020. Dostupné z: https://web.stanford.edu/class/cs224w/slides/08-GNN.pdf

[34] Graph (discrete mathematics). In Wikipedia the free encyklopedia [online]. © 10. říjen 2021, 2. ledna 2022 [cit. 02. 01. 2022]. Dostupné z: https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)

[35] J. You, Z.Ying, J. Leskovec, Design space for graph neural networks, In 'Advances in Neural Information Processing Systems 33', NeurIPS 2020, 2020, 1-13.

[36] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks. In 'Proceedings of the 5th International Conference on learning representations', ICLR, 2016, 1-14.

[37] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks. In 'arXiv:1710.10903', arxiv.org 2017, 1-12.

[38] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In 'IEEE Computer Society Conference on Computer Vision and Pattern Recognition', IEEE, 2016, 770-778.

[39] G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger. Densely connected convolutional networks. In 'IEEE Computer Society Conference on Computer Vision and Pattern Recognition',CVPR, 2017, 4700-4708.

[40] J. Dellinger, Weight Initialization in Neural Networks: A Journey From the Basics to Kaiming, Towards Data science [online]. © 3. duben 2019, 2. ledna 2022 [cit. 02. 01. 2022]. Dostupné z: https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9

[41] J. Chen, T. Ma, C. Xiao, Fastgcn: fast learning with graph convolutional networks via importance sampling. In 'arXiv:1801.10247', arxiv.org, 2017, 1-15.

[42] D. Zügner, T. Kirschstein, M. Catasta, J. Leskovec, S. Günnemann, Language-agnostic representation learning of source code from structure and context. In 'arXiv:2103.11318', arxiv.org, 2021, 1-22.

[43] F. M. Bianchi, D. Grattarola, L. Livi, C. Alippi, Graph neural networks with convolutional arma filters. In 'IEEE Transactions on Pattern Analysis and Machine Intelligence', IEEE, 2021, 1-1.

[44] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model. In 'IEEE transactions on neural networks', 20(1), 2008, 61-80.

[46] J. Leskovec, Representation Learning on Networks. Snap.Standford.edu [online]. © 24. duben 2018, 2. ledna 2022 [cit. 2022-02-01]. Dostupné z: http://snap.stanford.edu/proj/embeddings-www/

[47] Long short-term memory, Wikipedia [online]. © 23. prosinec 2012, 2. ledna 2022, [cit. 02.01. 2022]. Dostupné z: https://en.wikipedia.org/wiki/Long_short-term_memory

[48] S. Kostadinov, Understanding GRU Networks. Towards data science, © 16. prosinec 2017, 2. ledna 2022, [cit. 02.01. 2022]. Dostupné z: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

[49] J.You, R. Ying, J.Leskovec, Position-aware graph neural networks. In 'International conference on machine learning', PMLR, 2019, 7134-7143.

[50] Protein-protein interaction, Wikipedia [online]. © 2. leden 2012, 5. leden 2023, [cit.05.01. 2023]. Dostupné z: https://en.wikipedia.org/wiki/Protein_protein_interaction

[51] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, J. Leskovec, Open graph benchmark: Datasets for machine learning on graphs. In 'arXiv:2005.00687', arxiv.org, 2020, 1-34.

[52] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, C.-J. Hsieh, Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In 'Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining', Association for Computing Machinery, 2019, 257-266.

[53] Z. Hu, Y. Dong, K. Wang, K. W. Chang, Y. Sun, Gpt-gnn: Generative pre-training of graph neural networks. In 'Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining', KDD, 2020, 1857-1867.

[54] Z. Jia, S. Lin, R. Ying, J. You, J. Leskovec, A. Aiken, Redundancy-Free Computation for Graph Neural Networks. In 'Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining', KDD, 2020, 997-1005.

[55] J. Bai, Y. Ren, J. Zhang, Ripple Walk Training: A Subgraph-based training framework for Large and Deep Graph Neural Network. In 'arXiv:2002.07206', arxiv.org, 2020, 1-8.

[56] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs. In 'SIAM Journal on scientific Computing', 20(1), 1998, 359-392.

[57] Label propagation. In neo4. graph data science [online]. 2. ledna 2022 [cit. 02. 01. 2022]. Dostupné z: https://neo4j.com/docs/graph-data-science/current/algorithms/label-propagation/

[58] C. Tian, L. Ma, Z. Yang, Y. Dai, Pcgcn: Partition-centric processing for accelerating graph convolutional network. In '2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)', IEEE, 2020, 936-945.

[59] K. Kiningham, C. Re, P. Levis, GRIP: a graph neural network accelerator architecture. In 'arXiv:2007.13828', arxiv.org, 2020, 1-14.

[60] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, G. Karypis, Distdgl: distributed graph neural network training for billion-scale graphs. In '2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)', IEEE, 2020, 36-44.

[62] M. Fey, PyG Documentation — pytorch_geometric 2.0.4 documentation. [online]. © Copyright 2022, 2. ledna 2022 [cit. 2022-02-01]. Dostupné z: https://pytorch-geometric.readthedocs.io/en/latest/

[63] Amansinghal2002, Difference between PyTorch and TensorFlow - GeeksforGeeks. In: GeeksforGeeks | A computer science portal for geeks [online]. © 22. říjen, 2020, 2. ledna 2022 [cit. 2022-02-01]. Dostupné z: https://www.geeksforgeeks.org/difference-between-pytorch-and-tensorflow/

[64] Deep Graph Library [online] 2. ledna 2022 [cit. 2022-02-01]. Dostupné z: dgl.ai

[??] Tkipf, Graph Convolutional Networks. GitHub: Where the world builds software [online]. Copyright © 26. září 2020 GitHub, Inc. 2. ledna 2022 [cit. 02.01.2022]. Dostupné z: https://github.com/tkipf/gcn

[66] PetarV, GAT: Graph Attention Networks. GitHub: Where the world builds software [online]. Copyright © 2022 GitHub, Inc. 2. ledna 2022, [cit. 02.01.2022]. Dostupné z: https://github.com/PetarV-/GAT