Klasifikacija uz graf neuronske mreže

Klaić, Marko

Sveučilište Josipa Jurja Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike

Sveučilišni prijediplomski studij Matematika i računarstvo

Klasifikacija uz graf neuronske mreže

Završni rad

Mentor:
doc. dr. sc. Domagoj Ševerdija

Student:
Marko Klaić

Komentor:
dr. sc. Jurica Maltar

Osijek, 2024

Sadržaj

1. Uvod

Objekte stvarnog svijeta često definiramo u smislu njihovih veza s drugim stvarima, a kako bismo prikazali te odnose, prirodno je da koristimo grafove. Grafovi se koriste u mnogim područjima, uključujući informatiku i društvene znanosti. Posljednjih godina je došlo do velikog razvoja umjetne inteligencije te primjene tehnika dubokog učenja. Međutim, iako se duboko učenje razvilo u mnogim područjima, kao što je prepoznavanje slike ili obrada prirodnog jezika, i dalje su postojali izazovi pri rukovanju strukturiranim podacima, posebno podacima strukturiranim u obliku grafa. Za rješavanje takvih problema pojavila se revolucionarna tehnika kojom se bavimo u ovom radu, tehnika nazvana *graf neuronske mreže* (eng. graph neural network, abbr. GNN). Graf neuronske mreže omogućuju strojevima bolje razumijevanje i obradu podataka strukturiranih u obliku grafa. Koriste se u raznim područjima, kao što su klasifikacija teksta, klasifikacija slike, interakcijama između čovjeka i objekta, geolokaciji korisnika, predviđanju prometa, istraživanju strukture grafa molekula ili spojeva, sustavu preporuka i mnogim drugim. U ovom radu ćemo se baviti klasifikacijom uz pomoć graf neuronskih mreža.

2. Opis grafa

Grafovi su sveprisutna struktura podataka i univerzalni jezik za opisivanje složenih sustava. Općenito, možemo reći da je to skup objekata, odnosno čvorova, zajedno s njihovim interakcijama, odnosno bridovima. Matematički, graf $G = (V, E)$ je definiran skupom čvorova V i skupom bridova E između tih čvorova. Brid koji ide od čvora u do čvora v označavamo s (u, v). Graf može imati usmjerene ili neusmjerene bridove. Prikladan način za predstavljanje grafova jest matrica susjedstva $A \in \mathbb{R}^{(|V|*|V|)}$. Tada svaki čvor indeksira određeni redak i stupac u matrici susjedstva. Na taj način možemo prisutnost brida između dva čvora predstaviti jedinicom, a odsutnost brida nulom. Odnosno, ako postoji brid između čvora u i v onda vrijedi $A[u, v] = 1$, a u protivnom vrijedi $A[u, v] = 0$. U slučaju da matrica susjedstva sadrži samo neusmjerene bridove, ona će biti simetrična, a ako je graf usmjeren to ne mora nužno biti slučaj. Graf može imati i težinske bridove, tada su unosi u matrici A proizvoljne realne vrijednosti, a ne vrijednosti iz skupa $\{0, 1\}$.
Također imamo podjelu i na homogene, odnosno heterogene grafove. U homogenim grafovima čvorovi i bridovi imaju iste tipove, dok u heterogenim grafovima imaju različite tipove. Osim toga, postoji i podjela na statičke i dinamičke grafove. Kada se ulazne značajke ili topologija grafa mijenjaju s vremenom, graf se smatra dinamičkim grafom.

2.1. Višerelacijski grafovi

Postoje primjeri, kao što je npr. ispitivanje lijekova, u kojim nam pomoći mogu i višerelacijski grafovi. Kod ispitivanja lijekova bismo možda željeli da različiti bridovi odgovaraju različitim nuspojavama koje se mogu pojaviti kada uzmete par lijekova u isto vrijeme. U tom slučaju, proširit ćemo notaciju i ubacit ćemo tip brida koji koristimo, pa ćemo imati $(u, \tau, v) \in E$, te definiramo matricu susjedstva A_τ. Graf možemo sažeti matricom susjedstva $A \in \mathbb{R}^{(|V|*|R|*|V|)}$ pri čemu je R skup relacija. Skupinu višerelacijskih grafova dijelimo na heterogene i multipleksne grafove.

U heterogenim grafovima skup čvorova možemo podijeliti u disjunktne skupove
$\nu = \nu_1 \cup \nu_2 \cup \nu_3 \cup ... \cup \nu_k$ gdje je $\nu_i \cap \nu_j = \emptyset$, $\forall i \neq j$.
Bridovi u heterogenim grafovima općenito zadovoljavaju ograničenja prema tipovima čvorova, pri čemu je najčešće ograničenje da određeni bridovi povezuju samo čvorove određenih tipova.

U multipleksnim grafovima se čvorovi mreže repliciraju preko slojeva, a svaki sloj predstavlja određeni aspekt veze između čvorova. Kao primjer možemo uzeti multipleks transportnu mrežu, pri čemu svaki čvor može predstavljati grad, a svaki sloj može predstavljati drugačiji način prijevoza (npr. putovanje zrakoplovom ili vlakom). Unutarslojni bridovi predstavljaju gradove koji su povezani različitim načinima prijevoza, dok međuslojni bridovi predstavljaju mogućnost promjene načina prijevoza unutar određenog grada.

2.2. Informacije o podacima

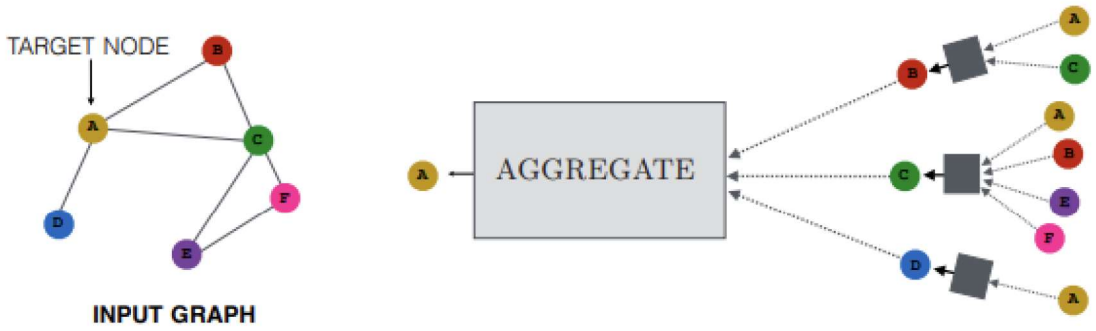
Na kraju, u mnogim slučajevima također imamo atribute ili informacije o podacima koji su povezani s grafom. Najčešće su to atributi na razini čvora koje predstavljamo pomoću matrice s realnim vrijednostima $X \in \mathbb{R}^{|V|*m}$ gdje pretpostavljamo da je redoslijed čvorova konzistentan s redoslijedom u matrici susjedstva.

3. Graf neuronske mreže

U ovom poglavlju uvodimo pojam graf neuronskih mreža. Pojava graf neuronskih mreža donijela je velike promjene u strojnom učenju, zbog raznih mogućnosti i velike sposobnosti rješavanja složenih problema. Graf neuronske mreže su vrsta modela dubokog učenja. Učinkovite su u raznim zadacima, od klasifikacije grafova i čvorova, pa sve do klasteriranja.

3.1. Prenošenje neuronskih poruka

Prenošenje neuronskih poruka je glavni koncept u graf neuronskim mrežama. To je koncept koji stvara razmjenu poruka između čvorova. Odnosno, čvorovi skupljaju informacije jedni od drugih, te ih agregiraju u nekom obliku kako bi obnovili svoje prikaze. Prolaskom kroz sve veći broj iteracija prenošenja poruka, dobivamo sve više informacija o grafu i o svakom čvoru. Odnosno, s matematičkim notacijama možemo reći da tijekom svake iteracije prenošenja poruka dobivamo ugradnju $h_u^{(t)}, \forall u \in V$ uz pomoć informacija koje dobivamo iz susjedstva čvora u, odnosno iz $N(u)$.



Slika 1: Prikupljanje podataka za čvor A. Na slici možemo vidjet kako čvor A prikuplja podatke od susjeda. Model prikuplja podatke susjeda od čvora A, dok su ti podaci došli nakon što su susjedi čvora A već pokupili podatke od svojih susjeda. U ovom slučaju imamo dva sloja prilikom prikupljanja poruka. Možemo primijetiti kako dobivamo formu stabla prilikom prikupljanja podataka oko čvora.

Ovaj proces sa slike možemo izraziti ovako:

$$h_u^{(t+1)} = UPDATE^{(t)}(h_u^{(t)}, AGGREGATE^{(t)}(h_v^{(t)}, \forall v \in N(u))$$

$$= UPDATE^{(t)}(h_u^{(t)}, m_{N(u)}^{(t)})$$

pri čemu je $m_{N(u)}$ poruka koju smo prikupili iz susjedstva čvora u, dok su $UPDATE$ i $AGGREGATE$ proizvoljne diferencijabilne funkcije. Funkcija $AGGREGATE$ u svakoj iteraciji za ulaz uzima skup ugradnji čvorova iz susjedstva od u te na osnovu informacija koje ima od susjeda, stvara poruku $m_{N(u)}^{(t)}$. Nakon što $AGGREGATE$ funkcija završi svoj posao, onda dolazimo do funkcije $UPDATE$ čiji je zadatak stvoriti novu ugradnju čvora u, tj. $h_u^{(t)}$ na osnovu kombiniranja poruke koju je stvorila funkcija $AGGREGATE$ te prošle ugradnje čvora u, tj. $h_u^{(t-1)}$. U prvom koraku ovog procesa, odnosno koraku kada imamo da je $t = 0$

vrijedi da je $h_u^{(0)} = x_u, \forall u \in V$, tj. za sve čvorove su ugradnje postavljene na ulazne značajke. Kada završimo sa svih T iteracija, ugradnja (eng. embedding) svakog čvora bit će:

$$z_u = h_u^{(T)}, \forall u \in V.$$

Povećanjem se broja iteracija u prenošenju poruka povećava i broj informacija koje će pojedinačni čvor prihvatiti. Te informacije zapravo dolaze u dva oblika. Prvi oblik nazivamo strukturalni. On može poslije određenog broja iteracija prenošenja poruka, npr. t, spremiti informaciju u ugradnju $h_u^{(t)}$ koja govori kojeg su stupnja svi čvorovi u $t - hop$ susjedstvu čvora u, pri čemu $t - hop$ susjedstvom čvora u smatramo sve one čvorove do kojih možemo doći od čvora u putem duljine t u grafu. Drugi oblik informacije odnosi se na značajke. Općenito, prenošenje poruka u GNN-u možemo zapisati kao

$$h_u^{(t)} = \sigma(W_{self}^{(t)} h_u^{(t-1)} + W_{neigh}^{(t)} \sum_{v \in N(u)} h_v^{(t-1)} + b^{(t)})$$

gdje su $W_{self}^{(t)}, W_{neigh}^{(t)} \in \mathbb{R}^{d^{(t)} * d^{(t-1)}}$ matrice parametara koje se mogu trenirati, dok je σ aktivacijska funkcija, najčešće su to funkcije $tanh$ ili $ReLU$. $ReLU$ je danas najčešće korištena aktivacijska funkcija, ali u praksi $tanh$ može biti bolja opcija od $ReLU$ iz više razloga, kao što je npr. raspon izlaznih vrijednosti (tanh ima vrijednosti u rasponu $(-1, 1)$, dok $ReLU$ ima samo pozitivne vrijednosti) te glatkoća koja nam može biti bitna tijekom izračunavanja optizimacije i gradijenta. Parametar pristranosti, odnosno $b^{(t)} \in \mathbb{R}^{d^{(t)}}$ opcionalni je dio funkcije, ali je gotovo uvijek koristan. On nam pomaže pomicati aktivacijsku funkciju lijevo ili desno, što može biti korisno za učenje. Ovaj pomak je ključan jer bi bez njega aktivacijska funkcija uvijek prolazila kroz ishodište $(0, 0)$, što možda nije optimalno za učenje složenih odnosa u podacima.

3.1.1. Prenošenje poruka sa samostalnom petljom

U teoriji grafova, samopetlja (eng. self-loop) je brid koji povezuje vrh sam sa sobom. Dodavanje samopetlje u ulazni graf čime pojednostavljujemo prenošenje poruka zato što agregiramo informacije od susjeda čvora kao i samog čvora može biti korisno iz nekoliko razloga. Prvi razlog je implicitno ažuriranje - korak ažuriranja je implicitno definiran kroz metodu agregacije. Takav pristup modelu može napraviti model jednostavnijim te spriječiti prenaučenost. Zatim, drugim razlogom smatramo dijeljenje parametara - dodavanje samopetlje je zapravo jednako kao da smo podijelili informacije između matrica koje su odgovorne za informacije o sebi (W_{self}) i onih koje su odgovorne za informacije o susjedima (W_{neigh}). Zbog tog možemo smanjiti broj parametara koji se mogu učiti u modelu i poboljšati učinkovitost računanja. Dok trećim razlogom smatramo regularizaciju - samopetlje mogu pružiti učinke regularizacije uključivanjem informacija tog čvora prilikom procesa agregacije. Odnosno, regularizacija može poboljšati mogućnost generalizacije modela te spriječiti prenaučenost. Ali, treba znati da je cijena koju plaćamo s ovim pristupom gubitak ekspresivnosti modela.

U slučaju da koristimo samopetlju, prenošenje poruka definiramo kao

$$h_u^{(t)} = AGGREGATE(\{h_v^{(t-1)}, \forall v \in N(u) \cup \{u\}\}),$$

3.2. Generalizirana agregacija susjedstva

U ovom ćemo poglavlju pokazati kako možemo poboljšati GNN model osvrnuvši se na funkciju $AGGREGATE$.

3.2.1. Normalizacija susjedstva

Prvi način agregacije je zbroj svih ugradnji čvorova iz susjedstva. No, takva agregacija osjetljiva je na stupanj susjedstva. Umjesto zbroja, uzmimo prosjek:

$$m_{N(u)} = \frac{\sum_{v \in N(u)} h_v}{|N(u)|}.$$

Još jedan dobar način za riješiti ovaj problem je simetrično normalizirana agregacija, odnosno:

$$m_{N(u)} = \sum_{v \in N(u)} \frac{h_v}{\sqrt{|N(u)||N(v)|}}.$$

Također, možemo izdvojiti kako jedan od najpopularnijih i najraširenijih modela graf neuronskih mreža, *graf konvolucijska mreža* (eng. graph convolutional network, abbr. GCN), koristi samopetlju te simetrično normaliziranu agregaciju. U njoj je prenošenje poruka definirano kao:

$$h_u^{(t)} = \sigma(W^{(t)} \sum_{v \in N(u) \cup \{u\}} \frac{h_v}{\sqrt{|N(u)||N(v)|}}).$$

3.3. Skupni agregatori

Drugi se pristup agregaciji odnosi na *skupne agregatore* (eng. set aggregators). Radi se o funkciji na skupu gdje imamo skup ugradnji $\{h_v, \forall v \in N(u)\}$ koje trebamo preslikati u vektor $m_{N(u)}$.

Udruživanje skupa

Udruživanje skupa, kao jedan od pristupa definiranja funkcije agregacije, temelji se na teoriji permutacijski invarijantnih neuronskih mreža, odnosno teoriji u kojoj izlaz funkcije ostaje isti bez obzira na redoslijed kojim su elementi skupa raspoređeni. Primjer takve funkcije, koju inače nazivamo aproksimator funkcije univerzalnog skupa, izgleda ovako:

$$m_{N(u)} = MLP_\theta(\sum_{v \in N(u)} MLP_\phi(h_v)),$$

pri čemu je MLP_θ proizvoljno duboki višeslojni perceptron (eng. multilayer perceptron) parametriziran nekim parametrima θ koji se mogu trenirati. Višeslojni je perceptron neuronska mreža koja se sastoji od potpuno povezanih neurona s nelinearnom aktivacijskom funkcijom, organiziranu u najmanje tri sloja, poznatu po tome što može razlikovati podatke koji nisu linearno separabilni. Kao rezultat ove funkcije dobivamo to da svaka funkcija koja je permutacijski invarijantna i koja preslikava skup ugradnji u jednu ugradnju se može aproksimirati do proizvoljne točnosti uz pomoć modela koji slijedi gornju jednadžbu.

Janossyevo udruživanje

Drugi pristup ovom problemu zove se Janossyevo udruživanje. Sposoban je predstaviti bilo koju permutacijski invarijantnu funkciju, što ga čini vrlo svestranim pristupom. Odnosno, ovo udruživanje može dobro predstaviti odnose i obrasce unutar skupova. Sa svojom fleksibilnošću i učinkovitošću, Janossy udruživanje služi kao vrijedan alat za modeliranje i analizu permutacijski invarijantnih funkcija, nudeći širok spektar pomoći u različitim domenama.

Označimo s $\pi_i \in \Pi$ permutacijsku funkciju koja mapira skup $\{h_v, \forall v \in N(u)\}$ na određeni niz $(h_{v_1}, h_{v_2}, ..., h_{v_{|N(u)|}})_{\pi_i}$. Drugim riječima, π_i uzima neuređeni skup susjednih ugradnji i smješta te ugradnje u niz temeljen na nekom proizvoljnom poretku. Nakon toga se agregacija odrađuje kao:

$$m_{N(u)} = MLP_\theta(\frac{1}{|\Pi|}\sum_{\pi\in\Pi}\rho_\phi(h_{v_1}, h_{v_2}, ..., h_{v_{|N(u)|}})_{\pi_i}),$$

gdje Π označava skup permutacija, a ρ_ϕ je permutacijski osjetljiva funkcija. Za funkciju ρ_ϕ često koristimo long short-term memory (abbr. LSTM) rekurentnu mrežu zato što je on poznat kao jaka arhitektura neuronske mreže za obradu sekvencijalnih podataka i dugoročnih ovisnosti. LSTM je učinkovit u dugotrajnom pamćenju informacija, pa se koristi za zadatke kao što su jezično modeliranje te predviđanje vremenskih serija. Janossy udruživanje u praksi koristi jedan od ova dva pristupa: upotrebljava kanonski poredak čvorova u skupu susjedstva, npr. redanje čvorova silaznim redoslijedom prema njihovom stupnju ili uzorkuje nasumični podskup mogućih permutacija tijekom svake primjene agregatora i zbraja samo taj nasumični podskup.

3.3.1. Pažnja susjedstva

Posljednji pristup koji ćemo predstaviti u ovom radu za poboljšavanje *AGGREGATE* funkcije jest tzv. pažnja susjedstva (eng. neighborhood attention), odnosno dodjeljivanje pažnje susjedima. U ovoj strategiji dajemo težinu pažnje svakom susjedu ovisno o utjecaju. Prvi GNN model koji je koristio ovaj pristup je Graph Attention Network (GAT), koji je izgledao ovako:

$$m_{N(u)} = \sum_{v\in N(u)} \alpha_{u,v} h_v,$$

gdje $\alpha_{u,v}$ označava pažnju susjeda $v \in N(u)$ prilikom agregiracije informacija na čvoru u. Pažnja se definira kao

$$\alpha_{u,v} = \frac{exp(a^T[Wh_u \oplus Wh_v])}{\sum_{v'\in N(u)} exp(a^T[Wh_u \oplus Wh'_v])},$$

gdje je a vektor pažnje koji se može trenirati, W je matrica koja se može trenirati, a $\oplus$ označava operaciju konkatenacije. Ovaj mehanizam nam može pomoći ako imamo neko znanje o podacima od prije, odnosno ako znamo da neki susjedi nam mogu biti korisniji od drugih pa im možemo dati veću težinu (značenje).

3.4. Generalizirane metode ažuriranja

Nakon što smo rekli ponešto o $AGGREGATE$ funkciji, njenim problemima i rješenjima, vrijeme je da kažemo nešto i o drugoj funkciji koja je bitna za graf neuronske mreže, a to je funkcija $UPDATE$. Prvo ćemo se dotaknuti problema koji je čest u GNN modelima, a to je problem prekomjernog izglađivanja (eng. over-smoothing problem). Kao što smo već spomenuli, ključni koncept cijelog modela graf neuronske mreže je prenošenje neuronskih poruka, međutim dodavanjem prevelikog broja slojeva u modelu (čime želimo dobiti što veći broj informacija o čvorovima, kako bi mogli na kraju imati što točnije rezultate), možemo zapravo napraviti suprotnu stvar od onog što smo htjeli, te nam čvorovi mogu postati previše slični jedni drugima, zbog čega dolazimo u problem koji smo malo prije spomenuli, problem prekomjernog izglađivanja.

Kako otkriti ovaj problem?

Ovdje ćemo izdvojiti dvije najpoznatije metrike, a to su MAD, te njeno proširenje, odnosno MADGap.

MAD izračunava srednju prosječnu udaljenost između ugradnji čvorova u grafu i koristi ju da pokaže da je izglađivanje prirodan rezultat dodavanja više slojeva GNN modelu.

MADGap procjenjuje sličnost prikaza u klasama čvorova. Možemo primijetiti da dva čvora s bliskom topološkom udaljenošću, odnosno čvorovi koji mogu doći jedan do drugog u nekoliko skokova vjerojatnije pripadaju istoj klasi, i obrnuto. Tako razlikujemo ulogu između udaljenih i susjednih čvorova, te računamo vrijednosti MADGap-a između udaljenih i susjednih čvorova kako bismo procijenili glatkoću prikaza grafa.

Kako ga riješiti?

Problem možemo riješiti umetanjem slojeva nelinearne neuronske mreže s napredovanjem unutar svakog GNN sloja. Neuronska mreža s napredovanjem je neuronska mreža čiji je tok jednosmjeran, čije informacije teku prema naprijed, od ulaznih čvorova, kroz skrivene čvorove pa sve do izlaznih čvorova, bez ikakvih ciklusa ili petlji. Ovo je dosta jednostavna, ali moćna tehnika.

3.4.1. Konkatenacija i preskakanje veza

Kao što smo već napomenuli, prekomjerno izglađivanje je ključni problem u GNN-ovima. U tim slučajevima, ažurirani će prikazi čvorova $h_u^{(t+1)}$ jako ovisiti o dolazećim porukama agregiranima od strane susjeda $m_{N(u)}$ na račun prikaza čvorova iz prethodnih slojeva $h_u^{(t)}$. Jedno od najjednostavnijih ažuriranja veze za preskakanje koristi konkatenaciju za očuvanje više informacija na razini čvora tijekom prosljeđivanja poruke:

$$UPDATE_{concat}(h_u, m_{N(u)}) = [UPDATE_{base}(h_u, m_{N(u))} \oplus h_u],$$

gdje je $UPDATE_{base}$ osnovna funkcija ažuriranja. U ovom ažuriranju jednostavno konkateniramo izlaz osnovne funkcije ažuriranja s prikazom prethodnog sloja čvora. Time potičemo razdvajanja informacija koje dolaze od susjeda od trenutnog prikaza svakog čvora. Uz ovo, možemo izdvojiti još i metodu linearne interpolacije. U ovoj metodi, konačni ažurirani prikaz je linearna interpolacija između prethodnog prikaza i prikaza koji je ažuriran na temelju informacija o susjedstvu. U praksi su ove metode najkorisnije za klasifikaciju čvorova s umjereno dubokim GNN-ovima.

3.4.2. Zatvorena ažuriranja

Jedan od načina na koji možemo promatrati algoritam za prenošenje neuronskih poruka je da funkcija agregacije prima zapažanje susjeda, koje se zatim koristi za ažuriranje skrivenog stanja svakog čvora. Tada možemo izravno primijeniti metode koje se koriste za ažuriranje skrivenog stanja arhitekture rekurentnih neuronskih mreža (eng. recurrent neural network, abbr. RNN) na temelju opažanja. Jedna od prvih GNN arhitektura definira funkciju ažuriranja kao

$$h_u^{(k)} = GRU(h_u^{(t-1)}, m_{N(u)}^{(t)}),$$

gdje GRU označava jednadžbu ažuriranja ćelije zatvorene rekurentne jedinice (GRU). Općenito, bilo koja funkcija ažuriranja definirana za RNN može se koristiti u kontekstu GNN-a. Ova zatvorena ažuriranja vrlo su učinkovita u olakšavanju dubokih GNN arhitektura i sprječavanju prekomjernog izglađivanja.

3.4.3. Preskakanje veza znanja

Dosad smo pretpostavljali da je prikaz čvora z_u jednak ugradnji čvora konačnog sloja u GNN-u:

$$z_u = h_u^{(T)}, \forall u \in V$$

Ovu pretpostavku koriste mnogi pristupi GNN-u. Način na koji možemo bolje prikazati konačni čvor je taj da iskoristitimo prikaze na svakom sloju prolaska poruka, umjesto samo korištenja izlaza konačnog sloja. Ako koristimo ovaj pristup, konačni prikazi čvorova z_u će izgledati ovako:

$$z_u = f_{JK}(h_u^{(0)} \oplus h_u^{(1)} \oplus ... \oplus h_u^{(T)}), \forall u \in V$$

gdje je f_{JK} proizvoljna diferencijabilna funkcija. U mnogim primjenama se kao f_{JK} koristi funkcija identiteta, ali se mogu koristiti i druge funkcije.

3.5. Multi-relacijski GNN

Do sad smo se bavili, i pretpostavljali da imamo jednostavne grafove. Međutim, grafovi mogu biti multirelacijski ili na neki drugi način heterogeni. U ovom poglavlju ćemo reći ponešto o tom kako riješiti takve slučajeve.

Prvi pristup za rješavanje ovog problema je poznat kao pristup konvolucijske mreže relacijskog grafa (eng. Relational Graph Convolutional Network, abbr. RCGN). U ovom pristupu povećavamo funkciju agregacije kako bi se prilagodili višestrukim tipovima odnosa uz određivanje zasebne matrice transformacije po tipu odnosa:

$$m_{N(u)} = \sum_{\tau \in R} \sum_{v \in N_\tau(u)} \frac{W_\tau h_v}{f_n(N(u), N(v))},$$

gdje je f_n normalizacijska funkcija koja može ovisiti i o susjedu čvora u kao i o susjedu v koji se agregira. Ovaj je pristup analogan osnovnom GNN pristupu s normalizacijom, ali u njemu zasebno agregiramo informacije preko različitih vrsta bridova.

3.6. Graf udruživanje

Dosad je predstavljeno kako izgleda ugradnja čvorova $z_u, \forall u \in V$, dok ćemo sada pogledati kako naučiti ugrađivanje z_G za cijeli graf G. Ovaj zadatak se često naziva udruživanje grafova, budući da je naš cilj udružiti ugradnje čvorova kako bismo naučili ugradnju cijelog grafa. Kao i kod $AGGREGATE$ operatora, ovaj problem možemo gledati kao problem učenja preko skupova. Želimo dizajnirati funkciju udruživanja f_p, koja mapira set ugrađivanja čvorova $\{z_1, ..., z_{|V|}\}$ u ugrađivanje z_G koje predstavlja cijeli graf. U praksi postoje dva pristupa koji se najčešće primjenjuju za učenje ugrađivanja na razini grafa putem udruživanja skupova. Prvi pristup je jednostavno uzeti zbroj (ili srednju vrijednost) ugrađenih čvorova:

$$z_G = \frac{\sum_{v \in V} z_u}{f_n(|V|)},$$

gdje je f_n neka normalizirajuća funkcija. Drugi popularni pristup koji se temelji na skupovima koristi kombinaciju LSTM-ova i pozornosti na skup ugradnji čvorova. U tom pristupu prolazimo kroz niz agregacija temeljenih na pažnji.

Pristupi ogrubljivanja grafova

Jedno ograničenje pristupa udruživanju skupova je da ne iskorištavaju strukturu grafa. Jedna od popularnih strategija kako bi iskoristili topologiju grafa u fazi udruživanja je izvođenje grupiranja ili ogrubljivanja grafova kao sredstva za udruživanje prikaza čvorova. Pretpostavljamo da imamo neku funkciju klasteriranja:

$$f_c \rightarrow G \times \mathbb{R}^{|V| \times d} \rightarrow \mathbb{R}^{+|V| \times c},$$

koja preslikava sve čvorove u grafu u neki od c klastera. Konkretno, pretpostavljamo da ova funkcija kao izlaz daje matricu dodjele $S = f_c(G, Z)$, gdje $S[u, i] \in \mathbb{R}^+$ označava snagu povezanosti između čvora u i klastera i.

Ključna ideja pristupa ogrubljivanju grafa je da koristimo matricu S. Konkretno, koristimo matricu dodjele S za izračunavanje nove grube matrice susjedstva

$$A^{new} = S^T A S \in \mathbb{R}^{+c \times c}$$

i novi set značajki čvora

$$X^{new} = S^T X \in \mathbb{R}^{c \times d}$$

Nova matrica susjedstva sada predstavlja snagu povezanosti između klastera u grafu, a nova matrica značajki predstavlja agregirane ugradnje za sve čvorove dodijeljene svakom klasteru. Sada možemo pokrenuti GNN na ovom ogrubljenom grafu te ponavljati proces ogrubljivanja za nekoliko iteracija gdje se graf smanjuje svakim korakom. Konačni prikaz grafa tada se izračunava skupom udruživanja preko ugradnji čvorova u dovoljno grubom grafu.

3.7. Generalizacija prenošenja poruka

Sada ćemo generalizirati GNN pristup prenošenju poruka kako bi iskoristili informacije na razini bridova i grafova u svakoj fazi prenošenja poruka. Generalizirani je pristup prenošenja poruka:

$$h_{(u,v)}^{(t)} = UPDATE_{edge}(h_{(u,v)}^{(t-1)}, h_u^{(t-1)}, h_v^{(t-1)}, h_G^{(t-1)})$$

$$m_{N(u)} = AGGREGATE_{node}(\{h_{(u,v)}^{(t)} \forall v \in N(u)\})$$

$$h_u^{(t)} = UPDATE_{node}(h_u^{(t-1)}, m_{N(u)}, h_G^{(t-1)})$$

$$h_G^{(t)} = UPDATE_{graph}(h_G^{(t-1)}, \{h_u^{(t)}, \forall u \in V\}, \{h_{(u,v)}^{(t)} \forall (u,v) \in \varepsilon\})$$

Važna novost u ovom generaliziranom pristupu prenošenju poruka je da tijekom prenošenja poruka generiramo skrivene ugradnje $h_{(u,v)}^{(t)}$ za svaki brid u grafu, kao i ugradnju $h_G^{(t)}$, koja je ugradnja cijelog grafa. To omogućuje modelu prenošenja poruka da lako integrira značajke brida i značajke na razini grafa. Tim generiranjem ugradnji također činimo trivijalnim definiranje funkcije gubitka za zadatake klasifikacije na razini grafa ili na razini bridova. U ovom generaliziranom pristupu za prenošenje poruka, prvo ažuriramo ugradnje bridova na temelju ugradnji njihovih susjednih čvorova. Zatim ažuriramo ugradnje čvorova tako što agregiramo ugradnje bridova za sve njihove susjedne bridove. Ugradnja grafa se koristi u jednadžbi ažuriranja za prikaze čvorova i bridova, dok se ugradnja na razini grafa ažurira agregiranjem svih ugradnji čvorova i bridova na kraju svake iteracije.

4. Dizajniranje funkcije gubitka

U ovom poglavlju pokazat ćemo kako dizajnirati funkciju gubitka na temelju našeg tipa zadatka i načinu učenja.

4.1. Vrste zadatka

U učenju grafa, postoje tri vrste zadatka:

- **Zadaci na razini čvora.** Zadaci na razini čvora (eng. node-level task) usredotočeni su na čvorove, što uključuje klasifikaciju čvorova, regresiju čvorova, grupiranje čvorova, itd. Klasifikacija čvorova pokušava kategorizirati čvorove u nekoliko klasa, a regresija čvorova predviđa kontinuiranu vrijednost za svaki čvor. Grupiranje čvorova ima za cilj podijeliti čvorove u nekoliko nepovezanih skupina, gdje bi slični čvorovi trebali biti u istoj skupini.

- **Zadaci na razini bridova.** Zadaci na razini bridova (eng. edge-level task) su klasifikacija bridova i predviđanje veze, koji zahtijevaju od modela klasificiranje tipova bridva ili predviđanje postoji li brid između dva data čvora.

- **Zadaci na razini grafa.** Zadaci na razini grafa (eng. graph-level task) koji uključuju klasifikaciju grafa, regresiju grafa i podudaranje grafa, a za sve je potreban model za učenje prikaza grafova.

4.2. Postavke za učenje

Iz perspektive nadzora, također možemo kategorizirati zadatke učenja grafa u tri različite načina:

- **Nadzirano učenje.** Nadzirano učenje (eng. supervised learning) pruža označene podatke za treniranje.

- **Polu-nadzirano učenje.** Polu-nadzirano učenje (eng. semi-supervised learning.) daje malu količinu označenih čvorova i veliku količinu neoznačenih čvorova za treniranje. Većina zadataka klasifikacije čvorova i rubova je polunadzirana.

- **Nenadzirano učenje.** Nenadzirano učenje (eng. non-supervised learning.) pruža neoznačene podatke modelu kako bi pronašao uzorke. Jedan od tipičnih zadataka ovakvog učenja je grupiranje čvorova.

4.3. Funkcije gubitka

Funkcije gubitka su ključne komponente u nadziranom učenju. Sastoje se od mjere koju prilikom učenja pokušavamo minimizirati kako bi dobili što bolji model za naš problem. Iterativnim ažuriranjem mrežnih parametara i minimiziranjem gubitka, neuronska mreža postupno uči generirati ugradnje čvorova koje kodiraju korisne informacije o čvorovima u grafu. Označimo sa Y prostor ciljeva, te neka $P_{|Y|}$ označava $|Y|$-dimenzionalni vjerojatnosni prostor. U tipičnom problemu klasifikacije, funkcija gubitka $L : P \times Y \to \mathbb{R}^+$ preslikava ciljno predviđanje para domene u nenegativan skalar. Želimo minimizirati funkciju na skupu D, koji predstavlja skup za treniranje. Pretpostavit ćemo da su ciljne oznake predstavljene kao one-hot vektori duljine C, odnosno duljine jednake broju klasa koje imamo u našem problemu. Radi jednostavnosti zapisa, izostavljamo eksplicitnu ovisnost predviđanja o parametrima modela i ilustriramo jednadžbe gubitaka za jednu podatkovnu točku.

Kao prvu funkciju gubitka navest ćemo gubitak unakrsne entropije (eng. cross-entropy loss, abbr. CE). Ona je najčešće korištena funkcija gubitka za probleme klasifikacije. Gubitak unakrsne entropije je zapravo negativni logaritam kategoričke vjerojatnosti parametriran izlazom softmax-a. Pretpostavimo da je p transformirano softmax predviđanje našeg modela za uzorak treniranja čija prava oznaka ima one-hot kodiranje y. Kategorijalni unakrsni gubitak entropije možemo izraziti kao:

$$L_{CE}(y, p) = -\sum_{i=1}^{C} y_i log p_i.$$

Druga funkcija gubitka koju ćemo navesti je srednja kvadratna pogreška (eng. mean squared error, abbr. MSE). Srednja kvadratna pogreška se također može koristiti za grešku klasifikacije, iako je češće korištena za regresiju. Ovu funkciju gubitka možemo zapisati kao:

$$L_{MSE}(y, p) = \sum_{i=1}^{C} (y_i - p_i)^2.$$

Treća funkcija gubitka je srednja apsolutna pogreška (eng. mean absolute error, abbr. MAE). Kao i srednju kvadratnu pogrešku, često ju koristimo u zadacima regresije. Međutim, dok MSE odgovara $L2$ normi, MAE odgovara normi $L1$. Pa zato funkcija gubitka izgleda ovako:

$$L_{MAE}(y, p) = \sum_{i=1}^{C} |y_i - p_i|.$$

Te posljednja funkcija gubitka koju ćemo ovdje navesti je funkcija gubitka zglobnice (eng. hinge-loss). Funkcija gubitka zglobnice je posebno korisna za treniranje modela u problemima

binarne klasifikacije. Ključna ideja iza funkcije gubitka zglobnice je više "kazniti" model kada pogrešno klasificira uzorak koji je bliži granici odluke. Ovdje razmatramo višeklasno proširenje funkcije koje izgleda ovako:

$$L_{hinge}(y, p) = \sum_{i=1}^{C} max(0, \gamma + p_i - p_{j*}),$$

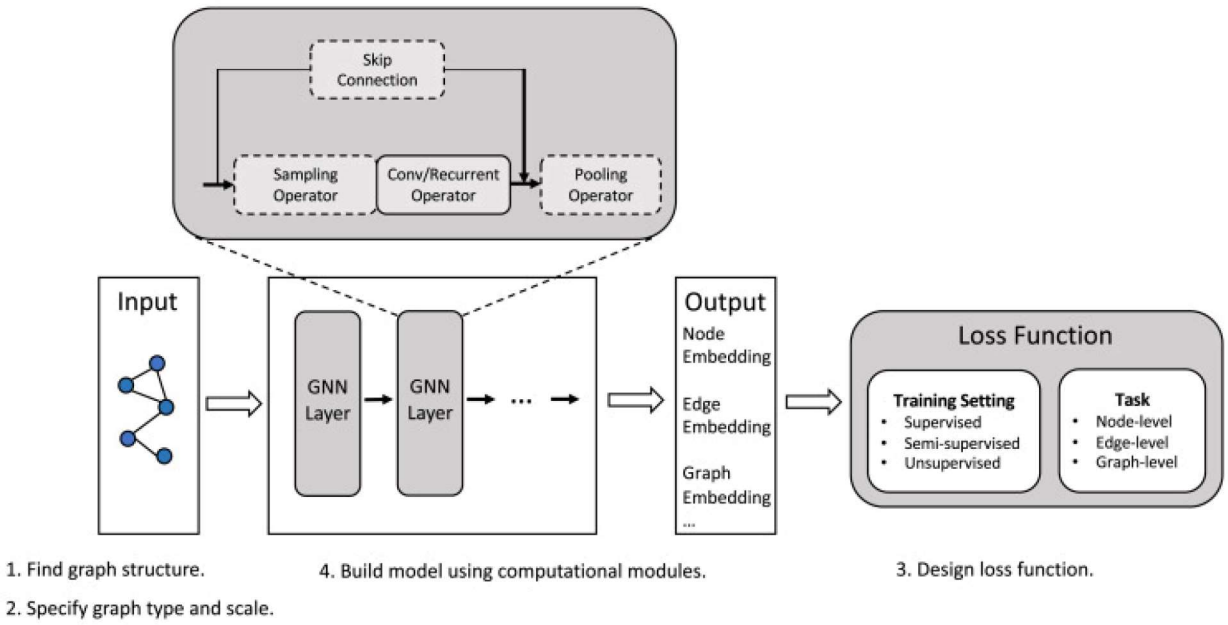
gdje je $j* = argmax_j\ y_j$ i γ je hiperparametar. U svim funkcijama gubitka smo smatrali da je p izlaz softmax aktivacijske funkcije.

5. Računalni moduli

Koristeći računalne module možemo početi graditi model. Neki često korišteni računalni moduli su:

- **Modul propagacije.** Modul propagacije koristi se za širenje informacija između čvorova tako da agregirane informacije mogu uhvatiti i karakteristike i topološke informacije. U modulima propagacije, operator konvolucije i rekurentni operator obično se koriste za prikupljanje informacija od susjeda, dok se operacija preskakanja veze koristi za prikupljanje informacija iz povijesnih prikaza čvorova i ublažavanje problema prekomjernog izglađivanja.
- **Modul uzorkovanja.** Ovaj modul se obično kombinira s modulom propagacije, a koristimo ga kada radimo sa velikim grafovima za provođenje širenja na grafovima.
- **Modul udruživanja.** Ovaj modul koristimo pri potrebi prikaza podgrafova ili grafova visoke razine kako bi dobili informacije iz čvorova.

Obično se GNN modeli grade kombinacijom svih ovih modula.



Slika 2: Arhitektura GNN modela

Na slici možemo vidjeti tipičnu arhitekturu GNN modela, gdje se konvolucijski operator, rekurentni operator, modul uzorkovanja i veza za preskakanje koriste za širenje informacija u svakom sloju, a zatim se dodaje modul udruživanja za izdvajanje informacija visoke razine.

6. Eksperimentalni dio

U eksperimentalnom dijelu smo implementirali nekoliko modela graf neuronskih mreža. Sve je rađeno u Google Colab-u koristeći PyTorch Geometric biblioteku koja služi za jednostavno pisanje i treniranje graf neuronskih mreža za širok raspon aplikacija povezanih sa strukturiranim podacima. Sastoji se od različitih metoda dubokog učenja na grafovima i drugim nepravilnim strukturama, također poznatih kao duboko geometrijsko učenje. Napravili smo po dva modela za klasifikaciju čvora i klasifikaciju grafa. U modelima za klasifikaciju čvora smo koristili CiteSeer skup podataka. CiteSeer skup podataka se sastoji od 3312 znanstvenih publikacija razvrstanih u jednu od šest klasa. Citatnu mrežu čine 4732 poveznice. Svaka objava u skupu podataka opisana je vektorom riječi s vrijednosti 0 ili 1 koji označava odsutnost ili prisutnost odgovarajuće riječi iz rječnika. Rječnik se sastoji od 3703 jedinstvene riječi. U prvom modelu smo koristili dva sloja GCNConv-a, čija je formulacija prema čvorovima dana sa:

$$x_i' = \Theta^T \sum_{j \in N(i) \cup \{i\}} \frac{e_{j,i}}{\sqrt{\hat{d}_j \hat{d}_i}} x_j,$$

pri čemu je $\hat{d}_i = 1 + \sum_{j \in N(i)} e_{j,i}$, gdje $e_{j,i}$ označava težinski brid od izvornog čvora j do ciljnog čvora i (pri čemu je zadana vrijednost jednaka 1). Koristili smo $ReLU$ aktivacijsku funkciju, te smo na kraju ispisali softmax distribuciju preko broja klasa. U treniranju svih modela smo mijenjali parametre kako bismo vidjeli razliku u kvaliteti modela s obzirom na parametre. U drugom modelu smo koristili GraphConv operator čija je formulacija prema čvorovima dana sa:

$$x_i' = W_1 x_i + W_2 \sum_{j \in N(i)} e_{j,i} * x_j,$$

gdje $e_{j,i}$ ponovo označava težinski brid od izvornog čvora j do ciljnog čvora i (pri čemu je zadana vrijednost jednaka 1). Sve ostale dijelove modela smo radili na istom principu kao i u prvom primjeru. Nakon tog smo napravili dva modela za klasifikaciju grafa. U prvom smo također koristili GCNConv, a u drugom GraphConv operator. Ali, u ovim primjerima smo imali po tri konvolucijska sloja. U primjerima za klasifikaciju čvorova smo trenirali model za 11 epoha, dok smo kod klasifikacije grafa to radili za 22 epohe. Dobiveni rezultati prikazani su u sljedećim tablicama:

GCNNode

Learning rate	Weight decay	Accuracy
0.1	0.1	0.7250
0.1	0.01	0.6910
0.1	0.001	0.5740
0.1	0.0001	0.6230
0.01	0.1	0.5650
0.01	0.01	0.6140
0.01	0.001	0.6620
0.01	0.0001	0.6670
0.001	0.1	0.6670
0.001	0.01	0.6650
0.001	0.001	0.6750
0.001	0.0001	0.6760
0.0001	0.1	0.6770
0.0001	0.01	0.6780
0.0001	0.001	0.6780
0.0001	0.0001	0.6780

GraphNode

Learning rate	Weight decay	Accuracy
0.1	0.1	0.4680
0.1	0.01	0.4020
0.1	0.001	0.4790
0.1	0.0001	0.5330
0.01	0.1	0.5410
0.01	0.01	0.5490
0.01	0.001	0.5720
0.01	0.0001	0.6330
0.001	0.1	0.6300
0.001	0.01	0.6290
0.001	0.001	0.6320
0.001	0.0001	0.6330
0.0001	0.1	0.6320
0.0001	0.01	0.6310
0.0001	0.001	0.6290
0.0001	0.0001	0.6340

GCNGraph

Learning rate	Weight decay	Accuracy
0.1	0.1	0.7368
0.1	0.01	0.7368
0.1	0.001	0.8684
0.1	0.0001	0.7368
0.01	0.1	0.7368
0.01	0.01	0.8684
0.01	0.001	0.8684
0.01	0.0001	0.8421
0.001	0.1	0.8158
0.001	0.01	0.8421
0.001	0.001	0.8684
0.001	0.0001	0.7895
0.0001	0.1	0.7895
0.0001	0.01	0.7895
0.0001	0.001	0.7895
0.0001	0.0001	0.7895

GraphGraph

Learning rate	Weight decay	Accuracy
0.1	0.1	0.7368
0.1	0.01	0.7368
0.1	0.001	0.7368
0.1	0.0001	0.7368
0.01	0.1	0.7368
0.01	0.01	0.7368
0.01	0.001	0.7368
0.01	0.0001	0.8158
0.001	0.1	0.7895
0.001	0.01	0.7895
0.001	0.001	0.7895
0.001	0.0001	0.7895
0.0001	0.1	0.7895
0.0001	0.01	0.7895
0.0001	0.001	0.7895
0.0001	0.0001	0.7895

Opširnije o ovom eksperimentalnom dijelu, te implementacija navedenog zadatka nalazi se na ovoj poveznici: `https://github.com/MarkoKlaic/Graf-neuronske-mreze`

7. Sažetak

U posljednjih nekoliko godina graf neuronske mreže dobivaju sve veću pažnju te se sukladno s tim razvijaju velikom brzinom. Graf neuronske mreže su postale moćni i praktični alati za zadatke strojnog učenja u domeni grafova. Napredak je zasnovan najviše na izražajnoj snazi, algoritmima za treniranje te na fleksibilnosti modela. U industriji su se tek nedavno krenule koristiti primjene strojnog učenja s grafovima, međutim, graf neuronske mreže su se odmah počele koristiti u različitim domenama te u najsuvremenijim modelima. U budućnosti možemo samo očekivati još veću kvalitetu te korist graf neuronskih mreža u još većem broju grana.

Ključne riječi

graf, neuronske mreže, strojno učenje, algoritmi

Summary

In the last few years, graph neural networks have been receiving more and more attention and, accordingly, are developing at a high speed. Graph neural networks have become powerful and practical tools for machine learning tasks in the graph domain. Progress is based mostly on expressive power, training algorithms and model flexibility. Machine learning applications with graphs have only recently begun to be used in industry, however, graph neural networks have immediately begun to be used in various domains and in state-of-the-art models. In the future, we can only expect even greater quality and benefit of graph neural networks in an even greater number of branches.

Keywords

graph, neural networks, machine learning, algorithms

8. Životopis

Rođen sam 04.01.2001. u Vinkovcima. Osnovnu školu sam završio u Osnovnoj školi Ruđera Boškovića u Donjoj Mahali, nakon čega sam upisao i završio Opću gimnaziju u Školskom centru Fra Martina Nedića u Orašju. Nakon toga sam upisao Sveučilišni prijediplomski studij Matematike i računarstva u Osijeku.

Literatura

[1] William. L. Hamilton, *Graph representation learning*, McGill University, 2020.

[2] *Pytorch Documentation*

[3] https://www.baeldung.com/cs/ml-gnn

[4] https://www.sciencedirect.com/science/article/pii/S2666651021000012?via%3Dihub

[5] https://medium.com/the-modern-scientist/graph-neural-networks-series-gnns-message-passing-over-smoothing

[6] https://en.wikipedia.org/wiki/Multilayer_perceptron

[7] https://arxiv.org/html/2403.17410v2

[8] https://medium.com/@kramiknakrani100/understanding-gnn-and-lstm

[9] https://medium.com/@hellorahulk/exploring-over-smoothing-in-graph-neural-networks

[10] https://medium.com/the-modern-scientist/graph-neural-networks-series-node-embedding

[11] https://medium.com/@TheDataScience-ProF/understanding-hinge-loss-in-machine-learning