Otkrivanje osobnih podataka u strukturiranim izvorima podataka primjenom grafovske neuronske mreže

Črepinko, Marko

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:539418

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Marko Črepinko

OTKRIVANJE OSOBNIH PODATAKA U STRUKTURIRANIM IZVORIMA PODATAKA PRIMJENOM GRAFOVSKE NEURONSKE MREŽE

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Marko Črepinko

Matični broj: 45072/16–R

Studij: Baze podataka i baze znanja

OTKRIVANJE OSOBNIH PODATAKA U STRUKTURIRANIM IZVORIMA PODATAKA PRIMJENOM GRAFOVSKE NEURONSKE MREŽE

DIPLOMSKI RAD

Mentor:
Prof. dr. sc. Kornelije Rabuzin

Varaždin, rujan 2022.

Marko Črepinko

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Zahvala:

Iskoristio bih ovu priliku da zahvalim mome mentoru, Prof. dr. sc. Korneliju Rabuzinu, na svim prilikama, iskustvu, znanju i savjetima koje mi je omogućio i prenio tijekom godina studiranja, a posebno na svom strpljenju kojeg je imao za mene tijekom izrade ovog rada.

Veliku zahvalnost dugujem i mojim kolegama, iz tvrtke Legit Software d.o.o., Ana-Mariji Petric, Luciji Marković i Josipu Punišu, na stečenom iskustvu, znanju i prilikama za profesionalni razvoj, koje tako nesebično dijele samnom, a posebno im hvala na savjetima i razumijevanju koje su imali za mene tijekom izrade ovog rada.

I na kraju, veliko hvala prijateljima i obitelji koji su uvijek čvrsto stajali uz mene tijekom godina studiranja, posebno Ivani, Mihaelu i Krešimiru na svoj inspiraciji i vjeri koju su imali u mene - te mojim roditeljima, na svoj ljubavi i trudu kojeg ulažu u mene.

Hvala Vam svima od srca!

Lose your dreams and you might lose your mind.

~ Mick Jagger

Sažetak

U ovome će radu biti predstavljena implementacija modela dubokog učenja na grafovima, nazvanog *TICLS*, za problem klasifikacije osobnih podataka u tabličnim strukturama. Zadaci implementiranog rješenja zahtijevati će transformaciju tabličnih podataka u grafovsku strukturu, nad kojom će potom biti izvršen zadatak klasifikacije čvorova. Rad također predstavlja teorijske osnove strojnog i dubokog učenja, dubokog učenja na grafovima i područja obrade prirodnog jezika. Na kraju su prikazani rezultati i performanse implementiranog modela *TICLS*.

Ključne riječi: duboko učenje, DGL, graf, tablica, GNN, GDPR, osobni podatak

Sadržaj

1. Uvod

Ime, prezime, OIB, ... - svi nas ovi podaci, samostalno ili združeno s ostalim podacima, jednoznačno identificiraju kao originalne, jedinstvene i neponovljive jedinke. No, vodimo li uopće računa o tome gdje sve i kome dajemo na korištenje naše osobne podatke u digitalnom svijetu: na društvenim mrežama, *on-line* forumima, prilikom ispunjavanja obrazaca za prijavu za sudjelovanje u nagradnim igrama, itd.? Kakav god da je odgovor na ovo pitanje, brojne su države svijeta shvatile potrebu za vraćanjem kontrole nad osobnim podacima natrag u ruke njihovim *vlasnicima*. Jedan takav primjer je i *Opća uredba za zaštitu podataka, OUZP*, (engl. *General Data Protection Regulation, GDPR*), koja je na snazi od 2015. godine na području zemalja članica EU. *OUZP* definira osobne podatke na sljedeći način [1, str. 33].

„... pojedinac čiji se identitet može utvrditi jest osoba koja se može identificirati izravno ili neizravno, osobito uz pomoć identifikatora kao što su ime, identifikacijski broj, podaci o lokaciji, mrežni identifikator ili uz pomoć jednog ili više čimbenika svojstvenih za fizički, fiziološki, genetski, mentalni, ekonomski, kulturni ili socijalni identitet tog pojedinca.”

Dodatno, *OUZP* ne dopušta obradu osobnih podataka bez da je korisnik pristao na njihovu obradu, dajući pritom *privolu* da se slaže s jasno definiranim razlozima obrade jasno definiranog skupa osobnih podataka [1, str. 33–34] . No, vrlo je bitno i tzv. *pravo na zaborav*, tj. pravo na brisanje osobnih podataka, a uključuje i pravo na izmjenu osobnih podataka [1, str. 43]. Stoga, gledano iz perspektive krajnjeg korisnika, ali i neke organizacije koja obrađuje njegove podatke, obje bi strane u bilo kojem trenutku trebale znati, ili barem biti u stanju pronaći, gdje se nalaze osobni podaci korisnika i koji su to podaci koje organizacija obrađuje, tj. smije obrađivati - jer *OUZP* ne dopušta obradu podataka za koju korisnik nije dao privolu (ili je istu povukao) ili za koju (više) ne postoje validni razlozi obrade podataka. Dakle, organizacija treba biti u mogućnosti u svakom trenutku djelovati nad osobnim podacima svojih korisnika, a kako bi to mogla ispoštovati, mora raspolagati informacijama o tome gdje se u podatkovnim kolekcijama (pr. bazama podataka, kolekcijama dokumenata, itd.) nalaze pojedini korisnikovi podaci. To je osobito težak problem za organizacije poput telekoma, banaka i sl., čiji se korisnici broje u stotinama tisuća – pratiti svu tu količinu podataka ručno jednostavno je nemoguće, a kako se većina podataka danas obrađuje pomoću računala, jasno je da računala možemo iskoristiti za rješenje problema pronalaska osobnih podataka. Stoga se tema ovoga rada fokusira na jedan dio automatizacije procesa pronalaska i općenito upravljanja osobnim podacima – *na dio prepoznavanja vrsta osobnih podataka u strukturiranim izvorima podataka, kao što su tablice*.

Kako bismo razumjeli predstavljeno rješenje, moramo se najprije zapitati kako uopće računalo možemo iskoristiti za takav zadatak. Mi, ljudi, intuitivno možemo prepoznati većinu vrsta osobnih podataka, poput *e-mail* adresa, osobnih imena i prezimena, adresa stanovanja, datuma, itd. Neke od tih vrsta osobnih podataka, koji su opisani dobro poznatim uzorcima, poput *e-mail* adresa, računalo može znatno lakše detektirati, pr. upotrebom regularnih izraza, od onih s visoko nepredvidivom strukturom, poput osobnih imena ili adresa. Intuitivno možemo zaključiti kako će takvi podaci, pogotovo kada dolaze iz različitih geografskih i govornih područja, sadržavati nepredvidivu količinu varijacija. Rješenje predstavljeno u ovom radu, osloniti će se na upotrebu metoda umjetne inteligencije i područja obrade i razumijevanja prirodnog jezika - interdisciplinarna područja računale znanosti, filozofije i lingvistike, koja mogu ponuditi rješenja, kako računalo *naučiti*, temeljem našeg znanja o različitim vrstama osobnih podataka, da iste prepoznaje i ispravno klasificira, tj. da ispravno prepoznaje vrste pojedinih osobnih podatka.

Za izradu programskog rješenja, koristit će se metode dubokog učenja na grafovima (engl. *Deep Learning on Graphs*). Glavni razlog tome jest nastojanje da se podatke tablične strukture transformira u graf, tj. u strukturu povezanih čvorova, pa bi podaci slogova (ćelija) tablice predstavljali čvorove grafa. Ideja koja leži iza ovog nastojanja jest pokušati računalo naučiti širem kontekstu pojedinog podatka iz tablice, a kojeg čine metapodaci tablice, poput naziva atributa - ali i podaci okolnih atributa. Posebni će fokus pritom biti stavljen na učenje značajki adresa i datuma, čiji se cjeloviti zapis često u tablicama može pronaći *rastranširan* na vrijednosti većeg broja atributa. Jedan takav primjer mogu predstavljati dimenzijske tablice u skladištima podataka s *modelom zvijezda,* poput datumske dimenzije, koja može s većim brojem atributa opisivati neki podatak o datumu, poput atributa koji pojedinačno opisuju dan, mjesec i godinu pojedinog datuma. Ili pak dimenzija koje pobliže opisuju korisnike, narudžbe, lokacije, itd. - gdje podatak o adresi može biti razlomljen na više atributa, poput naziva ulice, poštanskog broja, države, grada, itd. [2] .

U nastavku slijedi kratki pregled korištenih tehnologija, potom teorijska razrada područja dubokog učenja i dubokog učenja na grafovima i obrade prirodnog jezika. Potom će biti predstavljena priprema podataka te konačno implementirano rješenje i dobiveni rezultati učenja modela grafovske neuronske mreže.

2. Korištene tehnologije

Programsko rješenje predstavljeno u ovome radu, implementirano je u programskom jeziku *Python*. Za potrebe manipulacije i pripreme podataka, korištena je *Pythonova* knjićnica *pandas* [3], te osobito korisna struktura *DataFrame* podataka, koju ova knjižnica implementira [4]. Za potrebe generiranja sintetskih skupova podataka, korištenih za trening implementiranog modela dubokog učenja na grafovima, korištena je *Pythonova* knjižnica *faker* [5]. Za implementaciju spomenutog modela dubokog učenja na grafovima, korišten je *Pythonov* okvir za razvoj rješenja za duboko učenje na grafovima *PyTorch Geometric* [6], a iskorišten je i *Pythonov* okvir za razvoj rješenja s područja *tradicionalnog* dubokog učenja *PyTorh* [7], kao i okvir za duboko učenje *PyTorch Lightning* [8], koji uvelike pojednostavnjuje i automatizira brojne zadatke s kojima se susrećemo prilikom korištenja okvira *PyTorch*, odnosno *PyTorch Geometric.* Za obradu i prikaz rezultata izrade modela dubokog učenja, iskorišten je alat *TensorBoard* [9]. Za potrebe izvršenja zadataka iz domene obrade prirodnog jezika, poput kodiranja teksta u vektorsku reprezentaciju, iskorišten je *Pythonov* okvir za razvoj rješenja u području obrade prirodnog jezika *flair* [10]. Rješenje je instalirano i pogonjeno unutar *Docker* kontenjera [11], čija će konfiguracijska datoteka biti također priložena uz programsko rješenje. Programsko je rješenje razvijeno na operacijskom sustavu *Manjaro Linux [12]*.

3. Duboko učenje i NLP

U ovome će poglavlju biti predstavljeni osnovni koncepti dubokog učenja, dubokog učenja na grafovima i područja obrade prirodnog jezika

3.1. Strojno učenje

Već od samog osmišljanja prvog računala, ljudi se pitaju je li moguće računala učiniti inteligentnima – da uče poput nas, stječu nova iskustva i snalaze se u novim situacijama i prilikama [13]. Ta je radoznalost utkala temelje pojmu umjetne inteligencije (eng. *artificial intelligence, AI*), a kojeg možemo opisati kao mogućnost digitalnog računala ili računalom upravljanog robota, da obavlja zadatke karakteristične inteligentnim bićima – poput sposobnosti rasuđivanja, otkrivanja novih značenja, generaliziranja ili učenja iz prošlih iskustava [14]. Kako bi omogućio otkrivanje sposobnosti računala da razmišlja, Alan Turing je još davne 1950. godine osmislio *Turingov test*, kojeg računalo može uspješno *položiti* ako (ljudski) ispitivač ne može temeljem dobivenih odgovora od računala, na postavljena mu pitanja, odgonetnuti je li na pitanja odgovorilo računalo ili čovjek. Kako bi računalo ostvarilo taj cilj, trebalo bi raspolagati sljedećim sposobnostima:

- ***obrađivanja prirodnog jezika –*** kako bi uspješno komuniciralo na ljudskom jeziku,
- ***reprezentacije znanja*** – kako bi uspješno pohranilo ono što zna, čuje i vidi,
- ***automatiziranog rezoniranja*** – kako bi uspješno odgovaralo na pitanja i stvaralo nove zaključke,
- ***strojnog učenja*** – kako bi se uspješno prilagodilo novim okruženjima i prilikama te prepoznalo i ekstrapoliralo uzorke [15].

Ovakav opis inteligentnog stroja naslućuje nam kakav potencijal ima njegova realizacija pri pomoći ljudima u obavljanju svakodnevnih zadataka. U današnje se doba pametnih uređaja, interneta stvari, opće digitalizacije i umreženosti, ne moramo previše osvrtati oko sebe kako bismo otkrili na koje nam sve načine umjetna inteligencija pomaže u svakodnevnom životu. Neke od primjena uključuju automatizaciju rutinskih poslova, razumijevanje govora i slika, pomoć pri donošenju medicinskih dijagnoza, itd. [13]; pa možemo razumjeti kako zadatak prepoznavanja vrsta osobnih podataka čini izvrsnu priliku za primjenu neke od metoda umjetne inteligencije za razvoj rješenja sposobnog za njegovo uspješno izvršavanje. Za čovjeka, ovaj zadatak ne predstavlja veliki izazov – pogledom u određenu tablicu u bazi podataka ili pak neki dokument, možemo lako uočiti i razlučiti nečije ime i prezime od broja telefona. No, taj proces

predstavlja jedan od bitnih izazova za umjetnu inteligenciju, budući da ga mi (ljudi) obavljamo intuitivno jer raspolažemo sposobnošću razumijevanja ljudskog jezika [13]. Također je razumljivo i da se ovaj problem teško može formalizirati nekim konačnim, iscrpnim skupom jednostavnih matematičkih pravila – obrazaca u samom sadržaju tekstualnih podataka, npr. adresa, ali i mnogobrojna preklapanja, je jednostavno previše za izradu nekog jednostavnog algoritma baziranog na odabiru ispravnog pravila. Stoga se u ovome slučaju možemo pokušati osloniti na mogućnost *učenja* računala kako da taj posao obavlja što točnije i efikasnije, umjesto nas.

Algoritam strojnog učenja možemo definirati kao računalni program koji uči temeljem iskustva *E* s obzirom na neki razred zadataka *T* i mjere uspješnosti *P*, ako se njegova uspješnost u obavljanju zadatka *T*, mjerena mjerom *P*, poboljšava s iskustvom *E*. Takav je algoritam zapravo sposoban učiti temeljem podataka. Zadaci strojnog učenja obično opisuju način na koji algoritam obrađuje primjere podataka, koji se predstavljaju *n*-dimenizionalnim vektorima značajki (eng. *feature vector*) $x \in R^n$, gdje svaka komponenta $x_i$vektora predstavlja jednu značajku. Dimenzionalnost tog vektora predstavlja broj značajki kojima je neki primjer opisan [13]. U nastavku će se rada pojmovi *vektor značajki*, *ulazni vektor*, *ulazni primjer* i *primjer* koristiti kao sinonimi. Algoritam strojnog učenja gotovo nikada neće *učiti* temeljem samo jednog primjera, već temeljem skupa *n*-dimenzionalnih vektora značajki, kojeg još nazivamo i ulaznim prostorom [16]. Dakle, vektori u tom ulaznom prostoru sadrže određene uzorke i karakteristike, opisane pojedinim značajkama, a temeljem kojih nastojimo *naučiti* algoritam da ih uspješno prepoznaje u budućim primjerima. Jedan takav ulazni prostor mogu predstavljati i zapisi neke tablice u bazi podataka, gdje svaki redak tablice predstavlja jedan primjer (vektor) čije su značajke zapravo vrijednosti pojedinih atributa te tablice. Neki od zadataka koje možemo riješiti primjenom algoritama strojnog učenja uključuju:

- **klasifikacija** – algoritmom se nastoji predvidjeti kojoj klasi (razredu) $c_i \in \{c_1, \ldots, c_k\}$ pripada ulazni vektor $x \in R^n$. Podatke svake od klasa opisuje različit skup karakteristika, a, kolokvijalno rečeno, zadatak je algoritma naučiti njihove razlike. Skup klasa je općenito konačan, a nazivi se klasa za potrebe učenja mogu kodirati u cjelobrojne vrijednosti, koje npr. mogu predstavljati njihove pozicije u redoslijedu njihova navođenja. Jedan primjer klasifikacije predstavlja prepoznavanje objekata na slici temeljem vrijednosti pojedinih piksela (npr. auto, čovjek, prometni znak), kada je različitim objektima pridružena druga klasa,
- **regresiju –** u kojem se algoritmom nastoji predvidjeti (kontinuirana) numerička vrijednost temeljem ulaznog podatka, a koja zapravo predstavlja specifičan oblik klasifikacije. Jedan primjer regresije može predstavljati predviđanje cijene nekretnine temeljem njezinih karakteristika, karakteristika susjedstva, itd.,

- **strojno prevođenje –** u kojem algoritam niz simbola jednog jezika treba pretvoriti u odgovarajući niz simbola drugog jezika (npr. prevesti tekst s hrvatskog na engleski jezik),
- **sintetizaciju podataka i uzorkovanje –** gdje se algoritmom nastoje naučiti karakteristike nekog skupa podataka kako bi se izgenerirao skup novih primjera po uzoru na postojeće,
- **nadomještanjem nedostajućih vrijednosti u ulaznom skupu podataka –** gdje se algoritmom nastoji predvidjeti neka od nedostajućih komponenti ulaznog vektora značajki$x \in R^n$,
- i mnoge druge

Za ovaj je rad osobito bitno detaljnije pojasniti zadatak ***klasifikacije***, u kojem se algoritmom nastoji predvidjeti kojoj od *K* klasa pripada neki ulazni vektor $x \in R^n$ – formalno, algoritmom se nastoji producirati funkcija $h: R^n \rightarrow \{0, \ldots, K-1\}$, tj. ako vrijedi da je $y = f(x)$, tada y predstavlja klasu kojoj pripada vektor$x \in R^n$, a koja je opisana nekom brojevnom, obično cjelobrojnom, vrijednošću. [13].

Kako bismo algoritmom strojnog učenja postigli što veću razinu točnosti za određeni zadatak, potrebno je razumjeti kako će *izgledati* iskustvo samog algoritma tijekom učenja. Temeljem tog iskustva, algoritme strojnog učenja možemo podijeliti na tri općenite vrste: ***nadzirano učenje*** (engl. *supervisied learning*), ***nenadzirano učenje*** (engl. *unsupervisied learning*), te ***podržano učenje*** (engl*. reinforcement learning).* Prilikom nenadziranog učenja, algoritam nastoji samostalno razlučiti bitne od nebitnih karakteristika vektora ulaznog skupa, bez ikakve povratne informacije, pa temeljem naučenih razlika diskriminirati vektore ulaznog skupa. Jedan takav primjer predstavlja skupina algoritma **klasteriranja**, kojima je zadatak vektore ulaznog skupa podijeliti u klastere (grupe) najsličnijih elemenata. Primjer takvog algoritma može predstavljati sustav računalnog vida koji fotografije na kojima se nalaze mačke dijeli u jednu grupu, a ostale fotografije u drugu [13][15]. Za razliku od nenadziranog učenja, podržanim se učenjem algoritmu daje povratna informacija s obzirom na dobiveni rezultat u obliku nagrada i kazni. Takav se algoritam obično opisuje kao agent koji djeluje u nekom okruženju konačnim skupom akcija, a njegov je zadatak naučiti temeljem kojih je akcija, za dano stanje okruženja reprezentiranog kao ulazni vektor značajki, dobio nagradu. Ovdje se problem svodi na odabir optimalne akcije te je ovakav način učenja primjenjiv kod problema redoslijednog donošenja odluka, uz dugoročni cilj maksimizacije prosječne dobiti, u područjima poput igranja video igara, robotike, upravljanja resursima ili pak kod logističkih problema [17].

Kod nadziranog se učenja algoritmu također daje povratna informacija, no u obliku oznaka (engl. *label*) pridruženih vektorima značajki ulaznog prostora, koje predstavljaju oznaku klase (kod klasifikacije) ili ciljne brojčane vrijednosti (kod regresije). Time dobivamo skup označenih primjera (engl. *labeled dataset*):

$$D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N} \subseteq X \times Y$$

pri čemu je $X = \{x | x \in R^n\}$ ulazni prostor svih *n*-dimenzionalnih vektora značajki (pri čemu *N* predstavlja broj primjera u *X*), a $Y = \{0, \ldots, K-1\}$ je skup svih oznaka primjera veličine *K*. Skup *D* zapravo predstavlja podskup *Kartezijevog* produkta skupova *X* i *Y*, a možemo ga predstaviti pomoću dvije komponente – matricom primjera *X* i vektorom njihovih oznaka *y*:

$$X = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \ldots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \ldots & x_n^{(2)} \\ x_1^{(N)} & x_2^{(N)} & \ldots & x_n^{(N)} \end{pmatrix} Y = \begin{pmatrix} y^{(1)} \\ y^{(1)} \\ \vdots \\ y^{(N)} \end{pmatrix}$$

pri čemu retci matrice *X,* dimenzije $N \times n$, predstavljaju primjere, tj. vektore značajki, a komponente vektora *Y* oznake pridružene pojedinom primjeru (retku) u *X* [16]. Ranije smo spomenuli kako je zadatak algoritma strojnog učenja kod problema klasifikacije naučiti funkciju $h: R^n \to \{0, \ldots, K-1\}$. ***Naučiti*** tu funkciju zapravo znači primjerima iz *X* dodijeliti ispravnu oznaku iz *Y*, pa funkciju *h* možemo zapisati i kao:

$$h: X \to Y$$

Tu funkciju još nazivamo i *hipotezom*. Konačno, možemo formalno objasniti što je zapravo rezultat ***učenja*** algoritma strojnog učenja. Kao što smo ranije već spomenuli, primjeri u *X* predstavljaju *n*-dimenzionalne vektore značajki, pa će funkcija preslikavanja *h* zapravo predstavljati linearnu kombinaciju parametara i značajki ulaznog vektora x, što možemo zapisati na sljedeći način:

$$h(x; \theta)$$

pri čemu je ***θ*** vektor parametara, pa kažemo da je *h* parametrizirana parmetrima ***θ [16]***. Drugim riječima, parametri iz ***θ*** određuju utjecaj svake značajke na vrijednost izlazne varijable . Konkretno, kada je ulazni prostor $X = R$, svaki će primjer $x \in R$ biti opisan samo jednom značajkom, pa hipotezu za taj slučaj možemo definirati ovako:

$$h(x) = \theta_1 x + \theta_0$$

Čime smo zapravo definirali pravac, koji dijeli ulazni prostor na dva dijela, a funkcija *h* je opisana skupom parametara $\theta = \{\theta_1, \theta_0\}$. Ti se parametri u strojnom učenju često još nazivaju i ***težinama*** (engl. *weight*), pa vektor parametara možemo alternativno označiti s ***w***, pri čemu tada vrijedi [18]:

$$h(x) = w_1 x + w_0$$

Općenito, kada je $X \in R^n$, tj. kada su vektori značajki *n*-dimenzionalni, *h* možemo definirati kao:

$$h(x) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$$

Tada rezultat *učenja* algoritma strojnog učenja predstavlja ispravno podešene parametre *modela*, pri čemu je model funkcija definirana do na parametre (θ). No, različite kombinacije parametara u θ, jednoznačno određuju različite funkcije, pa je *model* zapravo skup *H* hipoteza *h*, parametriziranih (tj. indeksiranih) s θ. Model formalno možemo zapisati kao:

$$H = \{h(x;\theta)\}_\theta$$

Stoga, strojno učenje možemo opisati kao pretraživanje skupa hipoteza *H,* kako bi se pronašla najbolja hipoteza $h \in H$ *[16]*.

Prema definiciji algoritma strojnog učenja spočetka poglavlja, još je samo mjera uspješnosti *P* ostala neobjašnjena. Ona će nam zapravo poslužiti za odabir najbolje hipoteze $h \in H$. Kod zadatka klasifikacije (regresije), najbolja hipoteza je ona koja najtočnije klasificira (daje najtočniju ciljnu vrijednost za) primjere iz ulaznog prostora. Tako se mjera koja kvantificira koliko je neka hipoteza dobra (ili loša) naziva ***empirijska pogreška***, a empirijsku pogrešku hipoteze *h* na (fiksiranom) skupu označenih primjera *D* možemo označiti kao:

$$E(h|D)$$

Tako nas kod zadatka klasifikacije zanima udio pogrešno klasificiranih primjera u ukupnom broju primjera, tj. udio primjera za koje *h* daje pogrešne oznake. Nadalje, vrijednost pogreške načinjene na pojedinačnom primjeru naziva se *funkcija gubitka* (engl. *loss function*), a govori nam koliko je model dobio na ukupnoj pogrešci na jednom primjeru [16]. Formalno ju možemo definirati ovako:

$L: Y \times Y \to R^+$

Ta funkcija kao svoj ulaz uzima dvije oznake: onu točnu i onu dobivenom kao izlaz od *h*, a kao rezultat daje nenegativan realni broj koji predstavlja iznos gubitka, pa ju kraće možemo zapisati i kao $L\big(y, h(x)\big)$ [16]. Ukupnu empirijsku pogrešku tada možemo izračunati tako da zbroj svih pojedinačnih iznosa funkcije gubitka za svaki primjer podijelimo s brojem primjera. Što nas konačno dovodi do objašnjenja postupka kako pronaći optimalnu $h' \in H$. Taj postupak zapravo predstavlja minimizaciju empirijske pogreške, pri čemu je:

$$h' \underset{h \in H}{=} E(h|D)$$

što je zapravo istovjetno nalaženju optimalnih parametara θ':

$$\theta' \underset{\theta}{=} E(\theta|D)$$

No, odabir optimalne *h*, tj. modela, podrazumijeva optimalnu sposobnost generalizacije naučenog modela. To znači da optimalnim modelom ostvarujemo najveću točnost (najmanju pogrešku) na neviđenim primjerima. Kako bismo procijenili sposobnost generalizacije modela

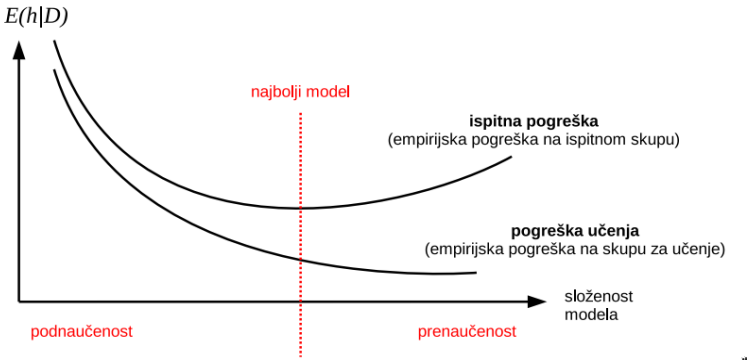
na neviđenim primjerima, skup *D* dijelimo na skup za učenje (trening) i skup za ispitivanje (testni skup):

$$D = D_{train} \cup D_{test}$$

Ti skupovi moraju biti disjunktni, a učenje modela obavlja se isključivo na skupu za učenje. Podjelom skupa *D*, a kako bismo utvrdili sposobnost generalizacije modela, računamo:

- **pogrešku učenja** (engl. *train error*) – $E(h|D_{train})$, te
- **ispitnu pogrešku** (engl. *test error*) – $E(h|D_{test})$.

Kako testni skup za model predstavlja ***neviđene*** primjere, ispitna pogreška nam ukazuje koliko će dobro model generalizirati. Pritom, osim optimalnog modela koji minimizira ispitnu pogrešku, razlikujemo još i podnaučeni i prenaučeni model. Podnaučeni model karakteriziraju vrlo visoka pogreška učenja i ispitna pogreška, a ukazuje na moguću nedovoljnu složenost modela, tj. da model nije sposoban naučiti potrebne zakonitosti u podacima skupa za učenje. Prenaučene modele karakterizira vrlo mala pogreška učenja, ali velika ispitna pogreška, a oni pak ukazuju na preveliku složenost modela, tj. model je prenaučio zakonitosti u podacima skupa za učenje pa jedino njih uspješno, u slučaju zadatka klasifikacije, klasificira. U oba je slučaja sposobnost generalizacije modela vrlo niska [16]. Odnos optimalnog, podnaučenog i prenaučenog modela, prikazan je krivuljama pogreške učenja i ispitne pogreške na grafu na slici 1.

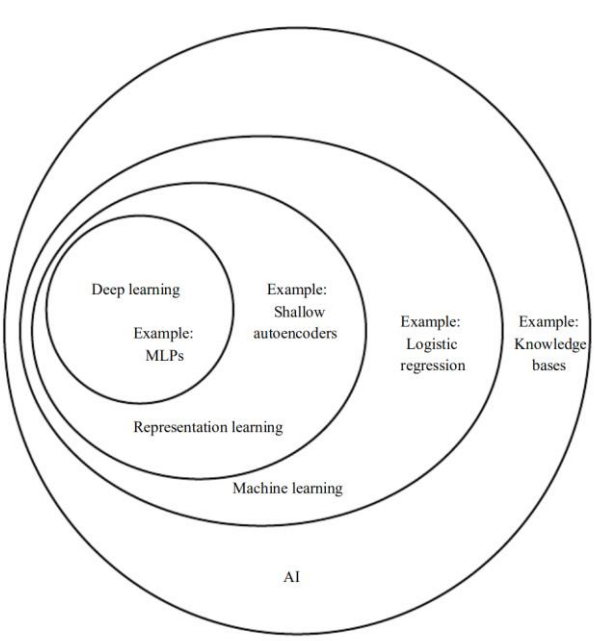


Slika 1: Odnos krivulja pogreške učenja i ispitne pogreške optimalnog, podnaučenog i prenaučenog modela. [14]

Ovime smo opisali sve potrebne osnovne koncepte strojnog učenja kako bismo mogli razumjeti što zapravo znači *duboko učenje*, budući da se ideje koraka postupka treniranja modela ne razlikuju.

3.2. Duboko učenje

Duboko učenje (engl. *Depp Learning*), predstavlja jedan podskup metoda strojnog učenja, odnosno umjetne inteligencije. Odnos pojedinih metoda umjetne inteligencije, prikazan je na slici 2.



Slika 2: Odnos područja umjetne inteligencije. [11]

Slično kao i strojno učenje, duboko učenje se temelji na predstavljanju podataka složenim reprezentacijama do kojih se dolazi slijedom naučenih nelinearnih transformacija [19]. Duboko učenje se pokazalo korisnim u mnogim programskim disciplinama, uključujući računalni vid, obradu govora i zvuka, obradu prirodnog jezika, robotiku, bioinformatiku i kemiju, video igre, tražilice, online oglašavanje i financije. Iako se oslanja da desetljećima stare matematičke koncepte, ovo je područje doživjelo svoj procvat tek pojavom snažnijeg računalnog sklopovlja. Veliki uspjeh i popularnost duguje boljim generalizacijskim sposobnostima modela dubokog učenja, spram ostalih metoda, nad novim podacima i pritom zahtijevajući manje skupove podataka za učenje. Također, ovo područje strojnog učenja donosi i veću skalabilnost s obzirom na učenje nelinearnih modela nad velikim skupovima podataka i u područjima primjene gdje je dimenzionalnost podataka vrlo velika [13].

Središnja skupina metoda dubokog učenja su *umjetne neuronske mreže* (engl. *Artificial Neural Network, ANN*). Naziv im potječe još iz 40-ih godina prošlog stoljeća i pokušaja modeliranja biološke mreže neurona računalnim sklopovima [15]. Umjetne neuronske mreže, često sadrže više *slojeva*, zbog kojih ih još nazivamo i *dubokim (umjetnim) neuronskim*

mrežama (engl. *deep artificial neural network*). Kako će se u nastavku rada pričati isključivo o umjetnim neuronskim mrežama, pojam *umjetne* se više neće navoditi.

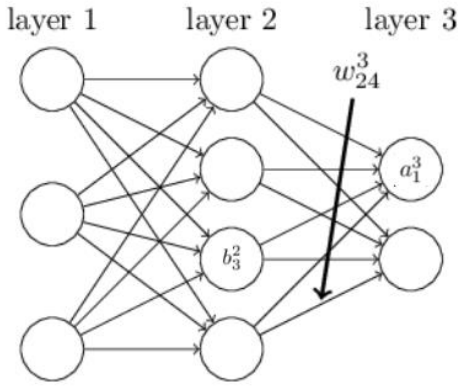
Osnovni oblik (arhitekture) neuronske mreže jest *unaprijedna neuronska mreža* (engl. *feedforward neural network, FFN*), poznata još i kao *višeslojni perceptron* (engl. *multilayer perceptron, MLP*). U nastavku će se rada za opis unaprijednih mreža koristiti engleska pokrata *FFN*. Cilj *FFN-a* jest aproksimirati neku funkciju f', npr. klasifikator $y = f'(x)$. *FFN* u tom slučaju definira mapiranje $y = f(x;\theta)$ i uči parametre θ za postizanje najtočnije aproksimacije funkcije f'. Ranije je rečeno da se područje dubokog učenja oslanja na slijed naučenih nelinearnih transformacija. U svijetu neuronskih mreža, slijedove nelinearnih transformacija dijelimo na slojeve, tj. svaki sloj *FFN* može biti predstavljen nekom drugom parametriziranom funkcijom f_θ. Formalno, slijed tih nelinearnih transformacija možemo prikazati kompozicijom više funkcija, npr. . $f(x,\theta) = \sigma\left(f_{(3)}\left(f_{(2)}\quad\left(f_{(1)}(x,\theta_1),\theta_2\right),\theta_3\right)\right)$. Primijetimo pritom usmjerenost toka informacija, tj. informacije *ulaze* u *ulazni sloj* $f_{(1)}$, potom u sloj $f_{(2)}$, itd. Obično posljednji sloj mreže, tj. modela, nazivamo *izlaznim slojem*, a sve slojeve između ulaznog i izlaznog *skrivenim slojevima*. Upravo zbog većeg broja slojeva, neuronske mreže zovemo dubokima, a broj slojeva definira *dubinu* modela. Model još možemo izraziti pomoćnih varijabli $h_L, h_{L-1}, \ldots, h_1$, za model s L slojeva, kao:

$$h1 = f_{(1)}(x,\theta_1)$$
$$\vdots$$
$$h_{L-1} = f_{(L-1)}(h_{L-2},\theta_{L-1})$$
$$h_L = f_{(L)}(h_{L-1},\theta_L)$$
$$f(x,\theta) = \sigma(h_L)$$

gdje je *σ* nelinearna aktivacijska funkcija. Te pomoćne varijable još nazivamo i *skrivenim* ili *latentnim značajkama*, dok x predstavlja značajke ulaznih vektora. Također, osim dubine, možemo definirati i *širinu* modela, kao dimenziju vektora značajki u l*-tom sloju.* Dodatno, ulazni i izlazni sloj su određeni ulazima x, odnosno izlazima y, pa model za učenje zapravo koristi samo skrivene slojeve [20]. Kako bismo proširili ranije opisani linearni model strojnog učenja, moramo osigurati da model *FFN* ostane nelinearan. To možemo učiniti na način da na skrivene značajke u $(l-1)$-om sloju također primijenimo nelinearnu aktivacijsku funkciju. To činimo za svaki element vektora skrivenih značajki individualno. Elementi vektora značajki pojedinog sloja zapravo predstavljaju neurone neuronske mreže, a aktivacijskom se funkcijom određuje izlaz pojedinog neurona. Kako je svaki neuron u mreži, osim neurona ulaznog sloja, povezan sa svim neuronima prethodnog sloja, njegov ulaz određen je linearnom kombinacijom aktivacija neurona prethodnog sloja i vektora težina veza koji povezuju dva neurona. Grafički,

FFN se može prikazati kao na slici 3 [21]. Vidimo da se mreža ukupno sastoji od 3 sloja, a oznake znače sljedeće:

- a_1^3 - aktivacija prvog neurona u trećem sloju
- w_{24}^3 - vrijednost parametra (težine) između drugog neurona u trećem sloju i četvrtog neurona u (*3-1*)-om sloju
- b_3^2 - vrijednost parametra pristranosti trećeg neurona u drugom sloju.

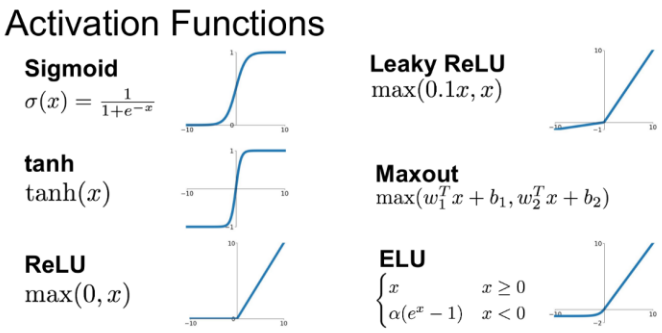


Slika 3: Unaprijedna neuronska mreža. Prema [19]

Sada možemo definirati pojedine vrijednosti. Vrijednost aktivacije pojedinog neurona definiramo na sljedeći način:

$$a_j^l = \sigma(z_j^l)$$
$$z_j^l = \sum_k w_{jk}^{l-1} \cdot a_k^{l-1} + v_j^l$$

gdje vrijednost z_l^j predstavlja vrijednost neurona prije primijenjene aktivacijske funkcije, k je broj slojeva mreže, a σ je nelinearna aktivacijska funkcija. Aktivacijskih funkcija postoji više, a definicije i grafovi krivulja nekih od najpopularnijih su prikazani na slici 4 .



Slika 4: Različite aktivacijske funkcije [20]

U programskom je rješenju korištena aktivacijska funkcija *LeakyReLU.* Njezina definicija zapravo glasi [23]:

$$LeakyReLU = \begin{cases} x, x \geq 0 \\ \frac{x}{\alpha}, x < 0 \end{cases}$$

gdje *α* predstavlja fiksni parametar koji se određuje prije treninga modela. U nastavku slijedi opis algoritma optimizacije parametara modela, no najprije ćemo uvesti neke konvencije s obzirom na prethodne definicije elemenata neuronske mreže. Pa neka tako w, w^l, b^l, a^l, z^l redom predstavljaju: vektor svih parametara modela, vektor težina koje ulaze u l-ti sloj, vektor parametara pristranosti l-tog sloja, vektor aktivacija l-tog sloja, vektor vrijednosti neurona l-tog sloja prije primijenjene aktivacijske funkcije. Ovakvi će poopćeni zapisi koristiti za lakše objašnjenje matematičkih jednadžbi. Vrijedi sljedeće:

$$z^l = w^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l)$$

Slijedi opis algoritma *gradijentni spust* (engl. *gradient descent, GD*), odnosno njegove unaprijeđene inačice *mini-batch stohastički gradijentni spust* (engl. *mini-batch stochastic gradient descent, SGD*), koji se koriste za optimizaciju parametara modela. Sva su sljedeća objašnjenja i matematički raspisi porijeklom iz [21].

Za zadatak klasifikacije, optimizacijom parametara modela zapravo nastojimo postići da vrijedi $E(w) \approx 0$, što postižemo kada će se izlazi modela ($y(x)$) najmanje razlikovati od očekivanih odnosa oznaka y_i. Kako je algoritam optimizacije iterativan, cilj je svakom iteracijom odabrati takve parametre w koji će pridonijeti dodatnom smanjenju pogreške. Stoga

možemo intuitivno razumjeti kako pogreška $E(w)$ ovisi o vrijednostima parametara modela, pa njezinu promjenu možemo zapisati na sljedeći način:

$$\Delta E(w) = \frac{\partial E(w)}{\partial w_1} \cdot \Delta w_1 + \cdots + \frac{\partial E(w)}{\partial w_n} \cdot \Delta w_n$$

Dakle, trebaju nam takvi parametri w, koji će promijenu $E(w)$ odvesti u negativnom smjeru. Vektor koji nam govori smjer najvećeg rasta neke funkcije je njezin gradijent, stoga odabrani parametri trebaju vrijednost pogreške odvesti u smjeru vektora $-\nabla E(w)$, čiji nam smjer govori smjer najvećeg pada funkcije pogreške. Kako je E funkcija parametara w, njezin gradijent možemo zapisati kao:

$$\nabla E(w) = \left(\frac{\partial E(w)}{\partial w_1}, \ldots, \frac{\partial E(w)}{\partial w_n}\right)^T$$

Konačno, ako promijene parametara možemo zapisati kao vektor promijene u w:

$$\Delta w = (\Delta w_1, \ldots, \Delta w_n)$$

tada promijenu $\Delta E(w)$možemo zapisati na sljedeći način:

$$\Delta E(w) \approx \nabla E(W) \cdot \Delta w$$

Dakle, cilj algoritma je u svakoj iteraciji pronaći takav Δw, koji će promijenu $\Delta E(w)$ odvesti u smjeru vektora $-\nabla E(w)$. Recimo da smo odabrali takav Δw da vrijedi:

$$\Delta w = -\eta \nabla E(w)$$

gdje je η neka zadana, fiksna stopa učenja, tada je zagarantirano sljedeće:

$$\Delta E(w) \approx \nabla E(W) \Delta w \approx -\eta \nabla E(w) \cdot \nabla E(w) = -\eta \|\nabla E(w)\|^2$$

pa kako uvijek vrijedi $\|\nabla E(w)\|^2 \geq 0$, uvijek će vrijediti i $\Delta E(w) \leq 0$, tj. iznos $E(w)$ će svakom iteracijom opadati, odnosno neće nikada rasti. Konačno možemo predstaviti postupak *pravila ažuriranja* (engl. *update rule*) kojim ažuriramo vrijednosti parametara modela:

$$w(t+1) = w(t) + \Delta w(t)$$
$$w(t+1) = w(t) - \eta \nabla E\big(w(t)\big)$$

pri čemu je $w(t+1)$ nova vrijednost parametara sljedećeg koraka, a $w(t)$ vrijednost parametara trenutnog koraka.

Ranije spomenuta implementacija *SGD* algoritma pojednostavnjuje *GD* algoritam, koji je računalno vrlo zahtjevan, kada se mora sprovoditi za učenje dubokih modela nad vrlo velikim brojem primjera. Stoga možemo dodatno aproksimirati vrijednost $\Delta E(w)$ u svakom koraku učenja (epohi, iteraciji) na način da algoritam *GD* primijenimo na manjem podskupu primjera za učenje. Prisjetimo se da ukupni iznos empirijske pogreške, u općem obliku, iznosi prosjek iznosa funkcije gubitka za svaki od n primjera:

$$E(w) = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, h(x_i, w)\right)$$

dakle, kako bismo izračunali $\nabla E(w)$, moramo najprije izračunati $\nabla L(w)$ za svaki pojedini primjer, pa konačno dobivamo da vrijedi:

$$\nabla E(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla L\big(y_i, h(x_i, w)\big)$$

Navođenje funkcije gubitka za svaki primjer možemo skratiti i pisati $E(w)_i$, kako bismo ukazali da računamo iznos funkcije gubitka za neki ulazni primjer $x_i \in D_{train}$. Kako *SGD* podrazumijeva da se algoritam *GD* primjenjuje na nekom podskupu skupa za učenje, potrebno je na početku svakog koraka učenja modela odabrati neki nasumični podskup primjera veličine m, $D'_{train} \subseteq D_{train}$, tako da vrijedi $m = |D'_{train}|, n = |D_{train}|, m \ll n$. Iz toga konačno slijedi :

$$\nabla E(w|D_{train}) \approx \nabla E(w|D'_{train})$$
$$\frac{1}{m}\sum_{j=1}^{m} \nabla E(w)_j \approx \frac{1}{n}\sum_{i=1}^{n} \nabla E_i(w) = \nabla E(w)$$
$$\nabla E(w) = \frac{1}{m}\sum_{j=1}^{m} \nabla E(w)_j$$

Kako bismo algoritam *GD* mogli primijeniti nad modelom *FFN*, trebamo opisati i algoritam *unazadne propagacije* (engl. *backpropagation, BP*). Cilj ovog algoritma je zapravo izračunati

parcijalne derivacije $\frac{\partial E(w)}{\partial w_{jk}^{l}}$, odnosno $\frac{\partial E(w)}{\partial b_{j}^{l}}$, za bilo koje iznose težina i pristranosti u mreži. No, najprije moramo definirati izraz δ_j^l, koji ćemo nazivati pogreškom *j*-tog neurona u *l*-tom sloju, a kasnije ćemo ga dovesti u odnos s $\frac{\partial E(w)}{\partial w_{jk}^{l}}$ i $\frac{\partial E(w)}{\partial b_{j}^{l}}$, a definiramo ga kao $\delta_j^l = \frac{\partial E(w)}{\partial z_j^l}$, pri tom je δ^lvektor pogrešaka svih neurona sloja *l*. Za opis BP algoritma, trebamo najprije definirati četiri jednadžbe:

- **BP1 –** jednadžba za izračun pogreške izlaznog sloja *L,* čije su komponente

$$\delta_j^L = \frac{\partial E(w)}{\partial a_j^L} \cdot \sigma'\left(z_j^L\right)$$

pri čemu izraz $\frac{\partial E(w)}{\partial a_j^L}$ predstavlja brzinu promijene $E(w)$, ovisno o funkciji aktivacije *j*-tog neurona izlaznog sloja, dok se izraz $\sigma'\left(z_j^L\right)$ definira kao $\sigma'\left(z_j^L\right) = \frac{\partial a_j^L}{\partial z_j^L}$ i predstavlja brzinu promijene iznosa aktivacije, ovisno o z_j^L.

- **BP2** – jednadžbu pogreške *l-tog* skrivenog sloja mreže u ovisnosti o pogrešci sljedećeg sloja:

$$\delta_j^l = \left(w^{l+1^T}\delta^{l+1}\right) \odot \sigma'\left(z_j^L\right)$$

gdje pritom $\odot$ predstavlja Hadamardov produkt matrica. Ovim izrazom možemo izračunati vektor pogreške δ^l, unatrag, za bilo koji sloj mreže, a koji nije izlazni.

- **BP3** – jednadžbu stope promijene funkcije gubitka $E(w)$, s obzirom na promjene pristranosti svakog neurona:

$$\frac{\partial E(w)}{\partial b_j^l} = \delta_j^l$$

- **BP4** - jednadžbu stope promijene funkcije gubitka $E(w)$, s obzirom na promjene pristranosti težina koje ulaze u pojedine neurone mreže:

$$\frac{\partial E(w)}{\partial w_{jk}^l} = a_{jk}^{l-1}\delta_j^l$$

Konačno, slijedi ispis potpunog algoritam učenja *FFN*, kao kombinacija algoritama *SGD* i propagacije unatrag:

1 Za korak učenja, radi:

2 Odaberi $D'_{train} \subseteq D_{train}$ veličine m

3 Za svaki vektor $x \in D'_{train}$ radi:

4 1. Izračunaj aktivacije ulaznog sloja $a^{x,l}$

5 2. Izvrši unaprijednu propagaciju za slojeve $l = 2,3,\ldots,L$:

6 2.a $z^{x,l} = w^l a^{x,l-1} + b^l$

7 2.b $a^{x,l} = \sigma(z^{x,l})$

8 3. Izračunaj grešku izlaznog sloja L:

9 $$\delta^{x,l} = \nabla_a E(w)_x \odot \sigma'(z^{x,l})$$

10 4. Izvrši unazadnu propagaciju:

11 Za svaki sloj $l = 2,3,\ldots,L-1$ izračunaj:

12 $$\delta^{x,l} = \left(\left(w^{l+1}\right)^T \delta^{x,l+1}\right) \odot \sigma'(z^{x,l})$$

13 Izvrši *GD:*

14 Za svaki sloj $l = 2,3,\ldots,L-1$ ažuriraj vektore težina i pristranosti:

15 $$w^l = w^l - \frac{\eta}{m} \sum_x \delta^{x,l} \left(a^{x,l-1}\right)^T$$

16 $$b^l = b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$$

U nastavku slijedi upoznavanje s dubokim učenjem na grafovima, prezentacija grafovskih neuronskih mreža, pregled nekih njihovih svojstava i opis arhitekture grafovske neuronske mreže, korištene u implementiranom programskom rješenju.

3.3. Duboko učenje na grafovima

Iako *tradicionalne* metode dubokog učenja ostvaruju izniman uspjeh nad podacima s topologijom rešetke, gdje se odnosi (veze) između podataka mogu vrlo lako opisati u *n*-dimenzionalnom Euklidskom prostoru, mnoga se znanstvena područja oslanjaju upravo na podatke koji se ne mogu predočiti rešetkastom topologijom. Tako se, na primjer, vrijednosti piksela u slici, koju si možemo dočarati kao pravilnu rešetku piksela, mogu vrlo lako prikazati matričnim zapisom dimenzija $n \times m \times c,$ gdje *c* može predstavljati broj bitova s kojima je neki piksel opisan, dok *n* i *m* predstavljaju koordinate pojedinog piksela na slici. Intuitivno nam je iz toga jasno kako se *zamjenom* mjesta više piksela zapravo dobiva drugačija slika. Dakle, kod podataka s rešetkastom topologijom, osim njihovih vrijednosti, bitna je i njihova *prostorna* informacija. Još jedan primjer takvih podataka su i vremenski nizovi – ili pak tekst, kojeg možemo predstaviti kao redoslijed, sekvencu (engl. *sequence*) riječi; oba ova primjera zapravo predstavljaju posebni slučaj rešetkaste topologije, gdje se rešetka sastoji od samo jednog retka. No, problem nastaje kada ovakva topologija postane *prekruta* i nedostatna za opis odnosa podataka iz određenih domena – poput društvenih mreža. U domeni društvenih mreža, odnose između pojedinih osoba modeliramo društvenim grafom, gdje pojedinci predstavljaju čvorove tog grafa, a njihovi su međuodnosi predstavljeni bridovima (vezama) grafa. No, upravo te veze između pojedinaca mogu predstavljati dodatne (kontekstualne) informacije, poput: tko je čiji *pratitelj* na nekoj digitalnoj društvenoj mreži, poput *Instagram*-a, pa te informacije možemo dodatno iskoristiti za otkrivanje društvenih grupa pojedinaca. Također možemo zamisliti da pojedince na digitalnim društvenim mrežama možemo povezati i drukčijom vrstom veze, čija će usmjerenost i samo postojanje upućivati na to je li neki pojedinac posjetio *profil* drugog pojedinca, što možemo nazvati interakcijom između pojedinaca, a pritom toj vezi može biti pridružena i vrijednost dosadašnjeg broja tih posjeta. Dakle, dva pojedinca možemo povezati s dvije različite vrste veza, od kojih će jedna upućivati na prirodu njihova odnosa, dok će druga upućivati na frekventnost njihove interakcije. Mreža prometnica se također može predstaviti nekim oblikom grafovskog zapisa, gdje bridovi grafa mogu sadržavati informaciju o duljini prometnice koja povezuje dvije lokacije, a grafom se pak mogu predstaviti i kovalentne veze atoma neke molekule [24] [25]. Pritom, trebamo imati na umu da podaci s rešetkastom topologijom također predstavljaju jedan od specijalnih oblika grafovskih struktura, tj. mogu se reprezentirati kao graf. No, za razliku od grafova, jedna bitna pretpostavka *tradicionalnih* algoritama dubokog učenja jest neovisnost pojedinih primjera, dok ista pretpostavka, kao što je i opisano, ne vrijedi za podatke reprezentirane grafovskom strukturom [26]. Neovisnost podataka predstavljenih grafovskom strukturom možemo opisati i na način da svaki čvor grafa predstavlja neki objekt ili koncept (iz stvarnog svijeta), a bridovi grafa predstavljaju veze, tj.

odnose između tih koncepata [27]. Ti koncepti pritom mogu biti različiti, kao na primjer: čovjek (korisnik), proizvod, lokacija, itd. - dok, npr., pikseli neke slike svi predstavljaju isti koncept. Stoga nam je intuitivno jasno da u grafu može postojati i više vrsta čvorova. Dakle, obradom grafovskih struktura, nastoje se iskoristiti strukturalne i semantičke informacije sadržane u izvornim podacima, kako bi se specifično domensko znanje dodatno iskoristilo za što precizniji opis odnosa između podataka, a mogli bismo nadodati i koncepata, a jedna od tehnika koja to omogućuje, naziva se *duboko učenje na grafovima*. Stoga ni ne čudi što je ovo područje doživjelo nagli porast u popularnosti u proteklih nekoliko godina, a kao jedna od najuspješnijih metoda u ovoj domeni dubokog učenja, pokazale su se upravo *grafovske neuronske mreže* (engl. *Graph Neural Networks, GNN*) [28]. U nastavku rada će se, osim pojma *grafovska neuronska mreža*, koristiti skraćeni izraz *grafovksa mreža*.

No, kako bismo bolje razumjeli rad grafovskih mreža, najprije ćemo definirati neke pojmove iz teorije grafova. ***Usmjereni graf*** $G = (V, E)$ sačinjen je od čvorova $V = \{1,2,\dots,n\}$ i bridova E ⊆ V × V, pri čemu je $e_{ij} = (i,j)\big|e_{ij} \in E$ brid grafa koji povezuje čvorove $i, j \in V$, a za koje, budući da čine dva suprotna kraja istog brida, kažemo da su *susjedni čvorovi;* pritom za neki brid $e_{ij} \in E$ vrijedi da je čvor i izvorišni, a čvor j odredišni čvor. Općenito, susjedstvo čvora $v_i \in V$ definiramo kao $N_i = \{j \in V | (i,j) \in E\}$. Stoga je usmjereni graf svaki onaj graf koji sadrži isključivo usmjerene bridove, dok je ***neusmjereni graf*** onaj graf čiji su svi bridovi neusmjereni [29]. Bridove neusmjerenog grafa $G' = (V, E')$ *ne predstavljam*o kao uređene parove, već možemo pisati $e'_{ij} = \{i,j\}\big|e'_{ij} \in E', \{i,j\} \subseteq V$. Brid e'_{ij} pritom predstavlja *dvosmjernu vezu* između čvorova i i j. Također ćemo za daljnji opis rada podrazumijevati da je svaki čvor $v_i \in V$ inicijalno reprezentiran (ulaznim) vektorom značajki $h_i^{(0)} \in R^{d_0}$, pritom oznaka (0) u eksponentu označava da se radi o inicijalnoj reprezentaciji čvora, dok d_0 označava dimenzionalnost vektora značajki kojim je neki čvor (inicijalno) opisan [30]. Taj (ulazni) vektor značajki ponekad se označava s x_i, a skup vektora značajki svih čvorova s X, kako bi se vrijednost tog vektora lakše razlikovala od vrijednosti $h_i^j \in R^d, j > 0$, koja predstavlja reprezentaciju čvora v_i u skrivenom sloju (engl. *hidden state, state embedding*) u nekoj (sljedećoj) j-toj iteraciji algoritma učenja.

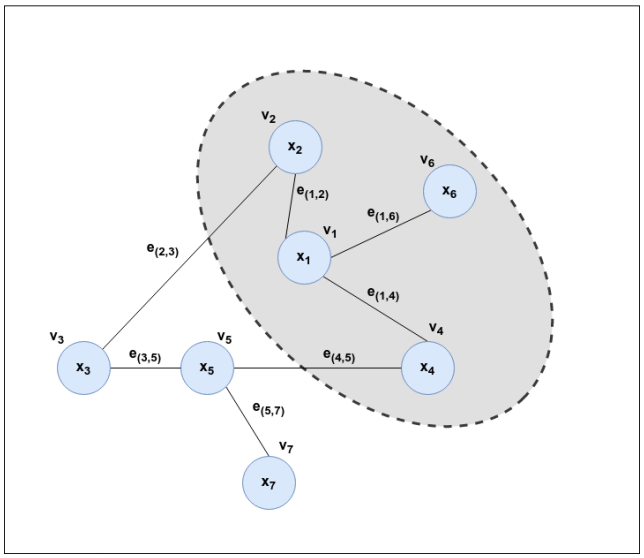
Ranije je spomenuto da graf može sadržavati više vrsta čvorova, tj. može povezivati više različitih koncepata, s više različitih vrsta bridova, tj. veza. Takav graf nazivamo ***heterogeni graf***, a možemo ga predstaviti kao graf $G = (V, E)$, gdje je pojedina vrsta čvora određena preslikavanjem $\phi: V \to A$, dok je vrsta brida (veze) određena preslikavanjem $\psi: E \to R$, pritom A predstavlja skup mogućih vrsta čvorova, dok R predstavlja skup mogućih vrsta

bridova. Kod heterogenih grafova vrijedi $|A| > 1$ ili $|R| > 1$. Specijalno, graf u kojem postoji samo jedna vrsta čvorova, odnosno bridova, naziva se još i ***homogeni graf [29]*.**

Potrebno je još definirati i podatkovnu strukturu kojom bilježimo informacije o bridovima koji povezuju pojedine čvorove grafa. Te se informacije predstavljaju matricom susjedstva $A \in R^{n \times n}$grafa $G = (V, E)$, pri čemu vrijedi $n = |V|$. Dakle, to je kvadratna matrica čiji je broj redaka i stupaca jednak broju čvorova grafa. Njezine vrijednosti definiramo na sljedeći način:

$A_{ij} = \begin{cases} 1, \{v_i, v_j\} \in E \\ 0, \{v_i, v_j\} \notin E \end{cases}$iz čega je očito da je matrica susjedstva A simetrična ako je G neusmjereni graf. Kako grafovi mogu sadržavati velik broj čvorova, možemo očekivati da će velik broj elemenata matrice A biti jednak nuli, tj. da će vrijediti $\left|\{a_{ij} \in A \middle| a_{ij} = 0\}\right| \gg |\{a_{nm} \in A | a_{nm} = 1\}|$. Takve se matrice u literaturi nazivaju *rijetke matrice* (engl. *sparse matrix*), a kako sadrže veliku količinu nepotrebnih informacija (nule na mjestu nepostojećih bridova), u samim implementacijama programskih rješenja koristimo sažeti oblik te matrice, tzv. *koordinatni format* (engl. *Coordinate list, COO*), koji elemente matrice susjedstva zapisuje kao liste koordinata [31]. U ovom će slučaju, takva matrica biti dimenzija$2 \times |V|$, gdje će prvi redak matrice sadržavati sve izvorišne čvorove bridova, a drugi sve odredišne, gdje pritom podrazumijevamo da se pozicije izvorišnih i odredišnih čvorova podudaraju u pripadajućim retcima. Ako skup čvorova grafa $G = (V, E)$ označimo s V, a rijetku matricu susjedstva označimo s A', tada za elemente matrice $a_{1i}, a_{2j} \in A$ vrijedi $a_{1i} = v_i, a_{2j} = v_j$, ako vrijedi$(v_i, v_j) \in E, \{v_i, v_j\} \subseteq V$ [32] .

Na slici 5 (prema [29] je prikazan primjer grafa, s čvorovima$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$, bridovima $E = \{e_{12}, e_{14}, e_{16}, e_{23}, e_{35}, e_{45}, e_{57}\}$, te vektorima značajki svakog čvora $X = \{x_1, x_2, x_3, x_4, x_5, x_6, x_7\}, X \subseteq R^{d_0}$. Dodatno je sivom površinom označeno susjedstvo čvora v_1, kojeg čini skup čvorova $N_1 = \{v_2, v_4, v_6\}$.



Slika 5: Primjer grafa (Autorski rad, 2022.)

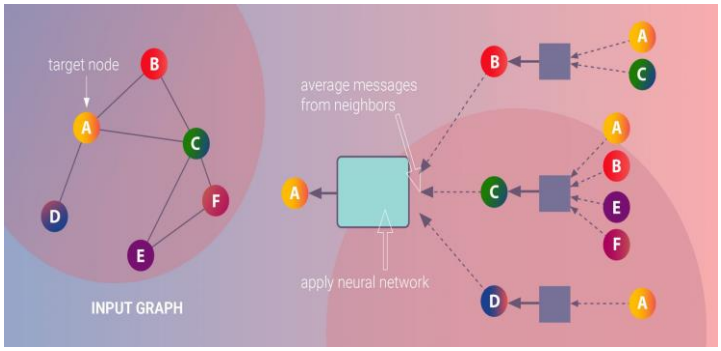
Konačno, cilj je grafovskih neuronskih mreža, odnosno pojedinih slojeva *GNN*-a, naučiti vektorsku reprezentaciju čvora grafa, te ju ažurirati svakom iteracijom algoritma učenja, temeljem agregacije vektorskih reprezentacija susjednih čvorova [30][29]. Pritom, ulaz u pojedini sloj *GNN*-a čini vektorska reprezentacija čvora $\{h_i \in R^d | i \in V\}$ i skup bridova E, dok je izlaz sloja nova vektorska reprezentacija čvora $\{h_i' \in R^{d'} | i \in V\}$, gdje je nad svakim čvorom i za dano mu susjedstvo N_i , primijenjena parametrizirana funkcija f_θ , pa vrijedi da je:

$$h_i' = f_\theta\left(h_i, AGGREGATE\left(\{h_j | j \in N_i\}\right)\right)$$

Kako postoje brojne vrste grafovskih mreža, implementacija same funkcije *AGGREGATE* varira, a neke od tipičnih su uprosječavanje po komponentama vektorskih reprezentacija čvorova, dok parametrizirana funkcija f_θ može biti *MLP* sloj s nekom aktivacijskom funkcijom na izlazu *[30]*. Dodajmo još kako se funkcijom *AGGREGATE* mogu obuhvatiti i značajke bridova između promatranog čvora i njegovih susjeda. Za samo učenje *GNN*-a, primjenjuju se već opisani algoritmi gradijentnog spusta i unazadne propagacije.

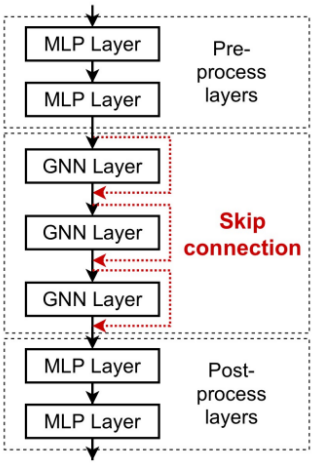
Prethodno opisani način ažuriranja vektorske reprezentacije nekog čvora temeljem vektorskih reprezentacija njegovih susjednih čvorova, naziva se *razmjena poruka* (engl. *message passing, MP*) ili *agregcija susjedstva* (engl. *neighborhood aggregation*), te predstavlja jedan od općenitih okvira za implementaciju grafovskih neuronskih mreža [33]. Također, ono što je bitno istaknuti, u grafovskoj će se mreži, baziranoj na *MP* okviru, svakim dodatnim slojem, reprezentacija nekog čvora ažurirati podacima šireg susjedstva. Tako će u prvom sloju agregacija podataka obuhvatiti reprezentacije prvih susjeda nekog čvora, dok će već u sljedećem sloju, reprezentacije prvih susjeda nekog čvora biti ažurirane

reprezentacijama njihovih prvih susjeda. Stoga će u drugom sloju, reprezentacija promatranog čvora biti ažurirana i reprezentacijama prvih susjeda njegovih prvih susjeda – i tako u nedogled [34]. Grafički je *MP* sustav prikazan ispod, na slici 6.



Slika 6: Primjer agregacije značajki susjednih čvorova [32]

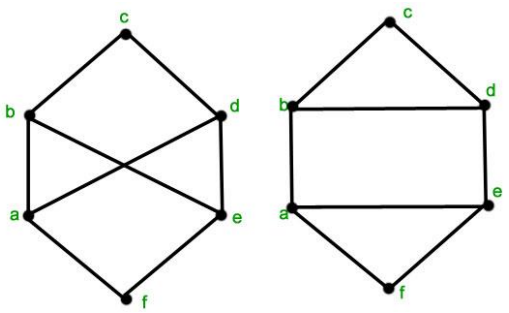
No, ovakav sustav propagacije informacija u grafovskim neuronskim mrežama zapravo sadrži jednu od njihovih najvećih mana – dodavanjem većeg broja slojeva, obično više od dva, *GNN*-ovi se susreću s tzv. problemom ***pretjeranog zaglađivanja*** (engl. *over-smoothing*). Kako se kod *GNN*-ova s većim brojem slojeva, u svakom sljedećem sloju promatrani čvor ažurira temeljem vektorskih reprezentacija sve šireg susjedstva, postepeno će svi čvorovi konvergirati istoj vrijednosti, tj. možemo reći da će se njihove reprezentacije previše *zagladiti*, odnosno ujednačiti. Stoga je većina grafovskih neuronskih mreža zapravo *plitka* s obzirom na broj slojeva [29] [35]. Jedno od obećavajućih rješenja koje se često spominje u literaturi jesu tzv. *veze s preskakanjem* (engl. *skip-connections*). One, osim što omogućuju brže učenje modela [36], omogućuju i implementaciju *dubokih GNN*-ova, tako što u odmakle slojeve mreže uvode ***svježe*** reprezentacije čvorova - bilo s ulaza prethodnih slojeva ili čak inicijalne vektorske reprezentacije čvorova [37][38]. Jedan takav primjer, prikazan je na ispod na slici 7.



Slika 7: Primjer veza s preskakanjem [36]

U ovome je radu, prilikom implementacije programskog rješenja, iskorišten koncept veza s preskakanjem, na način da su s vektorskim reprezentacijama čvorova na ulazu svakog skrivenog sloja, ulančane (engl. *concatenated*) reprezentacije čvorova s ulaza prošlog skrivenog sloja, te inicijalne vektorske reprezentacije čvorova. Naravno, ovakvim se pristupom značajno povećava broj parametara modela, no s druge se strane nastoji riješiti problem pretjeranog zaglađivanja i povećati ekspresivnost same grafovske neuronske mreže.

Još jedno svojstvo funkcije *AGGREGATE* nalaže da je ona invarijantna s obzirom na permutacije čvorova (engl. *permutation invariant*), tj. da poredak kojim se agregiraju reprezentacije čvorova ovom funkcijom ne utječe na konačni rezultat. No, ovo svojstvo dovodi u pitanje ekspresivnost *GNN*-ova, tj. njihovu sposobnost da razlikuju grafove naizgled slične strukture, broja čvorova i vektorskih reprezentacija čvorova i bridova, problem poznat kao problem izomorfizma grafova, kao graf na slici 8.



Slika 8: Primjer dvaju naizgled slična grafa [37]

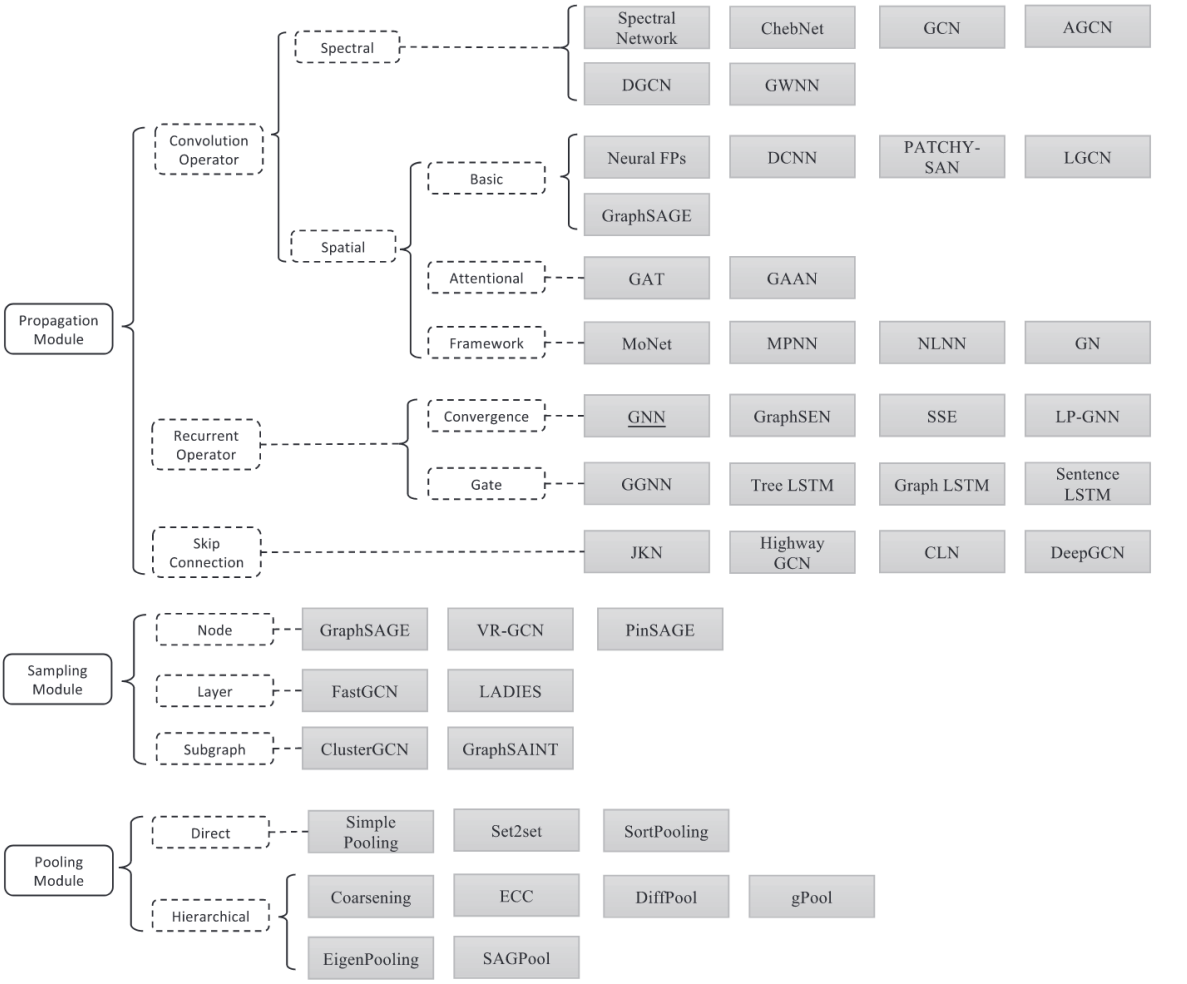
Ovaj je problem osobito bitan u domeni problema vezanih uz klasifikaciju lijekova, gdje je bitno prepoznati razlike između kemijskih spojeva vrlo slične strukture. Relacijska induktivna pristranost (engl. *Relational inductive bias*), jedno je od najvažnijih svojstava *GNN*-ova, a opisuje njihovu sposobnost učenja nad grafovskim strukturama, tj. učenja odnosa između različitih kombinacija koncepata iz stvarnog svijeta, stoga je vrlo bitno poznavati u kojim je slučajevima ovo svojstvo ograničeno [36]. Kao što je dokazano u [28], trivijalnije implementacije funkcije *AGGREGATE* doista ograničavaju ekspresivne sposobnosti *GNN*-ova s obzirom na opisani problem, no nedavna su istraživanja uspjela ponuditi rješenja ovome problemu, poput: [39], [40], [41], [42]. Čitatelja se dodatno upućuje na pregled poglavlja *5 - The Expressive Power of Graph Neural Networks* u [28], odnosno na izvore [43] i [44], gdje je dan detaljan matematički opis problema vezanih uz ekspresivnost *GNN-ova*, uz detaljna pojašnjenja postojećih rješenja.

Do sada opisani principi rada *GNN*-ova, priroda i inherentna svojstva podataka nad kojima ih primjenjujemo, te kratak uvid u neke od temeljnih izazova karakterističnih za njihove implementacije i korištenje, ukazuju na veliku raznolikost vrsta grafovskih neuronskih mreža i domena njihove primjene. Autori u [26] daju iscrpan pregled različitih primjena grafovskih neuronskih mreža, ali i njihovu usporedbu s ostalim metodama obrade grafova. Ovim su radom predstavili četiri opće kategorije grafovskih neuronskih mreža: **1)** povratne grafovske neuronske mreže (RecGNN) **2)** konvolucijske grafovske neuronske mreže (ConvGNN) **3)** grafovske autoenkodere (GAE) te **4)** spacijalno-temporalne grafovske neuronske mreže (STGNN). Slijedi pregled primjena pojedinih kategorija grafovskih neuronskih mreža:

- RecGNN - klasifikacija čvorova; klasifikacija grafova; sustavi preporuka
- ConvGNN - klasifikacija čvorova; klasifikacija grafova; primjena u istraživanju izlaznih podataka LiDAR radarskih sustava za bolje razumijevanje okoline; primjena u obradi prirodnog jezika za potrebe strojnog prevođenja, stvaranje grafa semantičke ovisnosti riječi u rečenici; sustavi preporuka
- GAE - učenje vektorskih reprezentacija mreža; generiranje grafova; učenje generativne distribucije podataka; generiranje molekularnih grafova za potrebe otkrivanja novih lijekova; sustavi preporuka
- STGNN - učenje o ljudskom djelovanju u video zapisima, tj. razumijevanje video sadržaja; problem predviđanja prometa; predviđanje manevarskih odluka vozača; parcijalno rješenje za obradu dinamičnosti grafovski struktura
- općenita primjena GNN - prirodna obrada jezika; učenje vektorskih reprezentacija grafova; predviđanje veza između čvorova grafa; klasifikacija čvorova grafa, verifikacija i razumijevanje programskog koda; predviđanje društvenog utjecaja; učenje o moždanim putevima; prepoznavanje događaja; sprečavanje napada na modele

strojnog učenja; računalni vid; modeliranje medicinskih zapisa bolesnika; sustavi preporuka; itd.

Ovo, dakako, nije konačna podjela arhitektura grafovskih neuronskih mreža – autori u [25] daju nešto širi, slikoviti prikaz, koje sve vrste *GNN*-ova postoje, a prikazane su na slici 9.



Slika 9: Prikaz raznih arhitektura grafovskih neuronskih mreža [23]

Za ovaj je rad posebno signifikantna skupina *grafovskih konvolucijskih neuronskih mreža* (engl. *Graph convolutional neural network, GCN*). S gornje slike, a i prema podijeli u [29], vidimo da se one dijele u skupine tzv. *spektralnih* i *spacijalnih* grafovskih konvolucijskih mreža, a za ovaj su rad od iznimne važnosti *grafovske neuronske mreže s mehanizmom pozornosti* (engl. *Graph Attention Neural Network, GAT*) iz skupine spacijalnih grafovskih mreža. Kako autori u [29] navode, osnovnu razliku između ovih dviju skupina konvolucijskih mreža čini ograničenost arhitektura spektralne skupine s obzirom na generaliziraju modela na nove grafovske strukture. Preciznije, važnost susjednih čvorova određena je težinama bridova, koje neće uvijek biti u skladu sa stvarnim međuutjecajem čvorova [28], a mnoge popularne

arhitekture *GNN*-ova svaki čvor tretiraju s istom *važnosti [30]*. Iz ovog nam je intuitivno jasno da se promjenom strukture grafa mijenjaju i težine veza. S druge pak strane, *GAT* mreže fokusirane su na izračunavanje neposredne važnosti koje čvorovi susjedstva imaju na promatrani čvor, tj. mehanizmom pozornosti nastoji se otkriti koji susjedni čvorovi najviše utječu na reprezentaciju promatranog čvora. *GAT* mreže, po prvi su puta predstavljene u [45]. Autori navode kako su neuronske mreže bazirane na mehanizmu pozornosti, postigle izniman uspjeh u zadacima poput učenja vektorskih reprezentacija pojedinačnih redoslijednih podatkovnih struktura (tzv. *self-attention* mehanizam (hrv. *mehanizam samo-pozornosti*)), poput teksta, te tako postale industrijski standard u domenama redoslijednih podatkovnih struktura. Pritom navode kako je jedna od najbitnijih prednosti neuronskih mreža s mehanizmom pozornosti rad s ulaznim podacima varijabilne veličine, što postižu fokusiranjem na najrelevantnije karakteristike ulaznih podataka. Dok kao tri primarna svojstva *GAT* mreža, ističu:

- komputacijska efikasnost – učenje vektorskih reprezentacija čvorova moguće je paralelizirati,
- učenje može biti primijenjeno na čvorove s proizvoljnim stupnjem, tj. proizvoljnim brojem bridova s kojima je neki čvor povezan,
- *GAT* model se može izravno primijeniti na probleme induktivnog učenja, uključujući zadatke u kojima se očekuje dobra generalizacija modela na dosad nove (neviđene) grafove.

Predstavljeno je rješenje iskorišteno na problemu klasifikacije čvorova grafa. Kao što je već opisano u uvodnom poglavlju, rješenje implementirano u ovome radu primarno će se fokusirati na podatke organizirane u tabličnu strukturu. Ovdje moramo napomenuti kako tablična struktura u ovome smislu zapravo predstavlja jednu vrstu organizacije podataka, dok se, gledano iz perspektive nad kakvim podacima će model dubokog učenja učiti – tablični podaci sami po sebi zapravo ne predstavljaju predvidivu strukturu. Nazivi atributa, redoslijed vrijednosti unutar atributa, sadržaj atributa, redoslijed atributa u zaglavlju tablice, svi zajedno, sa svojim varijacijama, predstavljaju iznimno nepredvidivu podatkovnu strukturu. Stoga će jedan od izazova prilikom izrade programskog rješenja predstavljati način na koji će se tablični podaci (i metapodaci) transformirati u grafovsku strukturu. Rješenja predstavljena u [46] i [47], iako ne koriste metode dubokog učenja na grafovima, daju dobar primjer kako se sadržaj atributa tablice može modelirati za potrebe predviđanja semantičke oznake kolone – problem koji će se nastojati riješiti programskim rješenjem predstavljenim u ovome radu. Autori sadržaj atributa modeliraju upravo kao tekstualni redoslijed, nadodajući statistička obilježja teksta, poput distribucije znakova u atributu, na vektorsku reprezentaciju samog sadržaja atributa. No intuitivno možemo zaključiti kako ovakav princip učenja modela ne uspijeva iskoristiti dodatne

informacije šireg susjedstva pojedinog sloga (ćelije) tablice, tj. susjedstva kojeg čine polja susjednih atributa. No, vratimo se na *GAT* mreže. Visoko varijabilna struktura tabličnih podataka bila je jedan od osnovnih kriterija prilikom odabira odgovarajuće arhitekture *GAT* mreže, tj. odabrana arhitektura trebala je imati sposobnost generalizacije nad novim strukturama (dobru induktivnu pristranost), tj. grafovima (onima na kojima model nije učen). Također, ne možemo ni očekivati da će svaka tablica sadržavati istu količinu podataka, no *GAT* mreže očito se mogu nositi i s ovim ograničenjem. Konačno, kako je fokus rješenja ovoga rada ispravno klasificirati vrstu osobnih podataka atributa tablica, a koji opisuju, npr., adrese ili datume, te da su pritom cjeloviti podaci *rastranširani* na vrijednosti više atributa, očekuje se da će *GAT* mreže opravdati relacijsku induktivnu pristranost grafovskih neuronskih mreža, donošenjem ispravnih zaključaka temeljem šireg susjedstva pojedinih slogova tablice. Konkretno, čovjeku je lako shvatiti da se neki cjelobrojni niz podataka, zapisan u atributu naziva *PostCode,* vrlo vjerojatno odnosi na poštanski broj, koji pak vrlo vjerojatno pripada nizu naziva ulica nekog susjednog atributa naziva *StreetName*. Intuitivno nam je jasno da čovjek može donijeti ovakav zaključak temeljem raspoznavanja samog sadržaja atributa, naziva atributa, ali i činjenice da se ova dva atributa pojavljuju zajedno u istoj tablici. Stoga se možemo nadati kako će *GAT* mreže uspješno prepoznati utjecaj susjednih podataka u ovakvim scenarijima. U ovome je radu zapravo iskorištena arhitektura *GAT* mreže predstavljena u [30], tzv. *GATv2* arhitektura, gdje autori nastoje riješiti tzv. problem ***statičke pozornosti*** (engl. *static attention*), a koji je prisutan u implementaciji izvorne *GAT* arhitekture predstavljene u [45]. Podsjetimo se, ulaz u sloj *GNN*-a određen je skupom $\{h_i \in R^d | i \in V\}$ vektrorskih reprezentacija čvorova grafa $G = (V, E)$. Izlaz *GNN* sloja predstavlja naučene reprezentacije čvorova $\{h_i' \in R^{d'} | i \in V\}$, gdje je nad svakim čvorom i njegovim susjedstvom N_i , primijenjena parametrizirana funkcija f_θ , pa vrijedi da je:

$$h_i' = f_\theta\left(h_i, AGGREGATE\left(\{h_j | j \in N_i\}\right)\right)$$

U *GAT* mreži, proširujemo opisani općeniti model *GNN*-a, definiranjem funkcije $e: R^d \times R^d \rightarrow R$za izračun *koeficijenta pozornosti* za svaki brid $(j, i) \in E$, a koji opisuje ***važnost*** utjecaja značajki čvora $j \in N_i$ na čvor i, pa vrijedi da je:

$$e(h_i, h_j) = LeakyReLU\left(a^T \cdot \left[Wh_i \parallel Wh_j\right]\right)$$

pri čemu funkcija $a: R^{d'} \times R^{d'} \rightarrow R$ definira *mehanizam pozornosti*, kao jednoslojnu *FFN* s aktivacijskom funkcijom *LeakyReLU,* parametriziranu vektorom $\vec{a} \in R^{2d'}$, potom matrica $W \in R^{d' \times d}$predstavlja naučene parametre linearne transformacije ulaznih značajki, dok $\parallel$

predstavlja operaciju ulančavanja vektora. Izračunati koeficijenti pozornosti potom se normaliziraju za sve čvorove $j \in N_i$, korištenjem *softmax* funkcije:

$$\alpha_{ij} = softmax_j\left(e\left(h_i, h_j\right)\right) = \frac{exp\left(e\left(h_i, h_j\right)\right)}{\sum_{j\prime \in N_i} exp\left(e\left(h_i, h_j{}'\right)\right)}$$

Potom se izračunavaju težinski prosjeci transformiranih značajki čvorova $j \in N_i$, na koje se dalje primijenjuje nelinearna aktivacijska funkcija σ, kako bi se dobila nova vektorska reprezentacija čvorova i:

$$h_i' = \sigma\left(\sum_{j \in N_i} \alpha_{ij} \cdot W h_j\right)$$

Dodatno se mehanizam pozornosti može proširiti mehanizmom *višestruke pozornosti* (engl. *multi-head attention*), kod kojih *K* neovisnih mehanizama izračunava novu vektorsku reprezentaciju $h_i{}'$čvora i, koje se potom ulančavaju u finalnu reprezentaciju čvora $h_i{}' \in R^{K \times d\prime}$:

$$h_i' \underset{k=1}{\overset{K}{=}} \sigma\left(\sum_{j \in N_i} \alpha_{ij}^k \cdot W^k h_j\right)$$

Ograničenje ovakve implementacije *GAT* mreže leži u definicije funkcije e kojom se izračunavaju koeficijenti pozornosti – takvu pozornost autori u [30] nazivaju ***statičkom pozornosti*** (engl. *static attention*), a karakterizira ju mogućnost da se dvama, ili više, susjednih čvorova $j \in N_i$pridruži isti koeficijent pozornosti e_{ij} s obzirom na više promatranih čvorova i. Intuitivno, problem možemo opisati kao ograničenu ekspresivnost *GAT*-a da neki čvor, koji je susjedan većem (*>1*) skupu čvorova, ima različiti koeficijent pozornosti za svaki čvor kojemu je on susjed. Drugim riječima, ako naučeni parametar a zapišemo kao $a = [a1 \parallel a2] \in R^{2d\prime}, a1, a2 \in R^{d\prime}$, tada vrijedi:

$$e\left(h_i, h_j\right) = LeakyReLU\left(a_1^T \cdot W h_i + a_2^T \cdot W h_j\right)$$

pritom vrijedi da, s obzirom na to što je skup čvorova V konačan, postoji čvor j_{max} takav da je $W h_{j_{max}}$maksimalan za sve čvorove $j \in V$.

Rješenje predstavljeno u [30], rješava ovaj problem, za definiranu *dinamičku pozornost* (engl. *dynamic attention*), jednostavnom zamjenom redoslijeda operacija u funkciji e, pa je ona redefinirana na sljedeći način

$$e(h_i, h_j) = a^T LeakyReLU(W \cdot [h1 \parallel h2])$$

Kako bismo priveli ovo poglavlje kraju i definirali problem koji će se nastojati riješiti u ovome radu, potrebno je navesti općenite razine vrsta zadataka za koje se *GNN* primjenjuju. Oni se mogu podijeliti na sljedeće 3 razine [25]:

- **razina čvorova** – uključuje: klasifikaciju čvorova (engl. *node classification)*, regresiju čvorova (engl. *node regression)*, klasteriranje čvorova (engl. *node clustering)*, itd.;
- **razina bridova** – uključuje: predviđanje postojanje bridova (veza) između čvorova (engl. *link prediction*), klasifikacija bridova (temeljem njihovih značajki) (engl. *edge classification*)
- **razina grafova** – klasifikacija grafova (engl. *graph classification*), grafovska regresija (engl. *graph regression*), podudaranje grafova (engl. *graph matching*).

Bitno je spomenuti i koje vrste učenja grafovskih neuronskih mreža postoje – dijelimo ih na *induktivno* (engl. *inductive*) i *transduktivno* (engl. *transductive*) učenje. Osnovna je razlika u tome što induktivno učenje omogućuje *GNN*-u generalizaciju nad novim problemima, tj. grafove novih struktura, uključujući i značajke čvorova i bridova, na koji model nije bio učen [28].

Ranije je spomenuto, kako će se programskim rješenjem implmenetiranim u ovome radu nastojati riješiti problem prepoznavanja vrste osobnih podataka u atributima neke tablice. Prirodno, ovaj problem pripada domeni klasifikacije podataka, pa osobne podatke možemo podijeliti u više klasa te izraditi podatkovni skup za učenje (testiranje) *GNN* modela u nadziranom okruženju. Pritom bi se traženi podatkovni skup trebao sastojati od tablica s raznim kombinacijama atributa, a svakom bi atributu bila pridružena odgovarajuća oznaku klase. No, razmislimo li malo, ovo zapravo predstavlja problem klasifikacije grafova, pri čemu bi svaki atribut trebao biti predstavljen kao graf, čiji su čvorovi podaci tog atributa. U [28] su predstavljene brojne metode klasifikacije grafova, a gotovo svima su zajedničke operacije tzv. *sažimanja* (engl. *pooling*) i *iščitavanja* (engl. *readout*), koje temeljem neke agregacijske funkcije nastoje grafu od interesa pridružiti odgovarajuću oznaku klase, temeljem *sažimanja* naučenih reprezentacija njegovih čvorova (i same strukture grafa), pa se graf često na kraju svodi i predstavlja jednim *umjetnim* čvorom. Slijedimo li taj pristup, na kraju ćemo nekako morati povezati više takvih *umjetnih čvorova*, od kojih će svaki predstavljati jedan atribut

tablice. Budući da ulaz u *GNN* slojeve predstavljaju značajke čvorova (i opcionalno bridova) i njihovi bridovi, svaku bismo tablicu trebali podijeliti na više grafova, klasificirati ih, ponovno povezati u graf *umjetnih čvorova* i svaki *umjetni čvor* nanovo klasificirati. Ovakav pristup iznimno komplicira pripremu podataka, a očit je i gubitak informacija koje se mogu ekstrahirati iz susjedstva pojedine ćelije tablice, kada podaci različitih atributa dijele puno više veza. Stoga će se za ovaj rad, odabrani problem definirati u okvirima klasifikacije čvorova. Preciznije, slogovi (ćelije) tablice će predstavljati čvorove grafa, kojima će biti pridruženi nazivi atributa, a pojedinom će slogu (čvoru) atributa biti pridružena oznaka klase osobnih podataka koja pripada tom atributu. Stoga problem koji će biti riješen ovim radom, možemo formulirati na sljedeći način:

Neka je tablica T, opisana nekim skupom atributa $\{A_1, A_2, \dots, A_n\} \subseteq T$ i skupom podataka i naziva atributa $D = \{a_{1,1}, \dots, a_{1,m}, a_{2,1}, \dots, a_{2,m}, \dots, a_{n,1}, \dots, a_{n,m}\}$, za čije elemente vrijedi$\{a_{i,2}, \dots, a_{i,m}\} \in A_i \big| \{A_i\} \subseteq T, i \in \{1,2,\dots,n\}$, odnosno neka skup $\{a_{1,1}, a_{2,1}, \dots, a_{n,1}\} \subseteq D$ predstavlja skup naziva atributa tablice T. Neka postoji proizvoljan skup $C = \{c_1, c_2, \dots, c_k\}$, čiji elementi predstavljaju oznake klasa (vrsta) osobnih podataka i neka je svakom atributu iz T pridružena jedna oznaka iz C. Tada definiramo skup oznaka $Y = \{y_{1,1}, \dots, y_{1,m}, y_{2,1}, \dots, y_{2,m}, \dots, y_{n,1}, \dots, y_{n,m} \big| y_{i,j} \in C\}$ pridružene podacima iz D, tako da pritom vrijedi da je $y_{ij} \in Y$ oznaka pridružena atributu $A_i | \{A_i\} \subseteq T, i \in 1,2,\dots,n$. Neka je nadalje tablica T, transformirana u graf $G = (V, E)$, pri čemu za skup čvorova vrijedi $V \subseteq D$, a skup Y neka predstavlja skup oznaka čvorova grafa G. Tada se traži: **a)** programsko rješenje koje će svaku tablicu T transformirati u graf $G = (V, E)$ **b)** programsko rješenje koje će čvorove grafa G najprije transformirati u početne vektorske reprezentacije, tako da vrijedi $V = \{h_{i,j}^{(0)} \big| h_{i,j}^{(0)} \in R^{d_0}\}$ **c)** programsko rješenje modela grafovske neuronske mreže koji će naučiti najbolju aproksimaciju klasifikatora $f: V \to Y$ (tj. kako preslikati čvorove grafa G na skup oznaka Y).

3.4. NLP

Kako ljudi za komunikaciju koriste ljudski jezik, u govornom ili pismenom obliku, koji sadrži iznimno kompleksne obrasce, a klasično računalo pak razumije isključivo vrlo jednostavan *jezik* brojeva, potrebno mu je omogućiti razumijevanje ljudskog jezika njegovom transformacijom u računalu razumljiv oblik. Rješenja ovome problemu donosi interdisciplinarno znanstveno područje obrade prirodnog jezika (eng. *Natural Language Processing, NLP)*, u daljnjem tekstu *NLP*, a koje predstavlja susretište lingvistike, računalne znanosti i umjetne inteligencije [48]. *GNN*-ovi su također primjenjivi na širokom skupu zadataka iz područja obrade prirodnog jezika, a za ovaj je rad najznačajniji zadatak klasifikacije teksta. Različite implementacije *GNN*-ova eksperimentiraju s raznim načinima reprezentacije tekstualnih sadržaja grafovskim strukturama [29]. No, razlog zašto nam je područje obrade prirodnog jezika zapravo zanimljivo jesu metode reprezentacije teksta nekim vektorom značajki, tj. zbog transofrmacije teksta, nestrukturiranog oblika podataka, u vektorsku reprezentaciju (engl. *embedding*), tj. strukturiranog oblika podataka [48]. Za razvoj ideje programske implementacije predstavljene u ovome radu, osobito je koristilo rješenje predstavljeno u [49], u kojem su autori predstavili novu arhitekturu grafovske neuronske mreže, nazvane *BEGNN,* za zadatak klasifikacije dokumenata. Pritom su riječi rečenica u tekstu predstavljale graf, a za inicijalizaciju značajki čvorova, autori su koristili prednaučeni jezični model *BERT,* prvi put predstavljenim u [50]. *BERT* model je baziran na popularnoj *Transformer* arhitekturi neuronskih mreža, a koja je bazirana na mehanizmu pozornosti i omogućuje učenje tekstualnih odnosa između riječi u tekstu, što *BERT* modelu daje sposobnost razumijevanja konteksta i dvosmislenosti u tekstu [51], pa tako *BERT*, npr., istu riječ preslikava u drugačiji vektor, ovisno o kontekstu rečenice, tj. kontekstu oko riječi. Ovdje nećemo ulaziti previše u dubinu oko objašnjavanja arhitekture *BERT* modela ni načina rada *Transformer*-a, no moramo napomenuti kako će *kvalitetnije* vektorske reprezentacije podataka, uroditi i boljim rezultatima učenja modela. Osim *BEGNN* mreže, vrijedi spomenuti još jedno, a kako su pokazali predstavljeni rezultati eksperimenata, vrlo uspješno rješenje za klasifikaciju tabličnih podataka temeljem oznaka atributa, tzv. model *DODUO,* koji je također baziran na *Transformer* arhitekturi [52]. Pritom, predstavljeno rješenje dopušta proizvoljan odabir jezičnog modela koji će se koristiti za samu klasifikaciju podataka, a među najboljima se opet pokazao *BERT* model. Stoga će u ovome radu također biti iskorišten *BERT* model, dostupan na [53], primjenom *flair* knjižnice programskog jezika *Python*. Preciznije, iskoristit će se klasa *TransformerDocumentEmbeddings [54]*, koja omogućuje preslikavanje niza riječi u samo jedan vektor značajki, umjesto u po jedan vektor za svaku riječ. U pozadini sustav svejedno primijenjuje *BERT* model na svaku od riječi u nizu, stvaravši pritom vektor značajki za svaku od njih, no na kraju sažima sve vektore značajki u jedan vektor, nekom od zadanih operacija. Izlazni će vektor biti dimenzije *768,* te će ta brojka predstavljati

širinu ulaznog sloja implementiranog modela u ovome radu. U nastavku slijedi opis pripreme podataka.

4. Priprema podataka

U ovom će poglavlju biti predstavljen odabrani način pretvorbe tablične strukture u graf te proces sakupljanja i generiranje podatkovnih skupova za trening i test.

4.1. Pretvorba tablične strukture u graf

id	job	company	ssn	residence_line_1	residence_city	residence_zip	username	name	sex	main_address	birthday_day	birthday_month	birthday_year
other	other	other	ssn	address	address	address	other	name	other	address	date	date	date
other	other	other	ssn	address_part	address_part	address_part	other	name	other	address	date_part	date_part	date_part
0	Manufacturing engineer	Freeman, Caldwell and Woodard	886-22-0253	3176 South Eufaula Avenue	Auburn	6684	cshaw	Jessica Johnson PhD	F	1537 Hwy 231 South Torrington AL 36420	19	5	1989
1	Administrator, sports	Hansen-Holloway	222-36-3596	425 Route 31	Alabaster	86529	denise00	Scott Johnston	M	145 Kelley Blvd Commack AL 12414	30	4	1985
2	Associate Professor	Williams-Harding	663-76-1725	70 Pleasant Valley Street	Geneseo	2884	mwolf	Crystal Rodgers	F	150 Barnum Avenue Cutoff Lisbon NY 35146	19	9	1998

Slika 10: Primjer pripreme tabličnih podataka za transformaciju u grafovsku strukturu (Autorski rad, 2022.)

Slika 10 prikazuje ispis *DataFrame* objekta koji predstavlja primjer jedne tablice s podacima, koja je u sekundarnoj memoriji bila pohranjena u formatu *CSV* datoteke. Sa slike možemo zamijetiti kako se tablica sastoji od sljedećih dijelova:

- zaglavlja s nazivima atributa tablice *{id, job, company, ssn, ...}*
- prvog retka koji sadrži oznaku klase osobnih podataka kojoj pripadaju podaci pripadnog atributa
- drugog retka koji sadrži *pod-oznaku* klase osobnih podataka kojoj pripadaju podaci pripadnog atributa, a kada ti podaci predstavljaju samo jedan sastavni dio cjelovitog podatka koji je u tablici razdijeljen na više atributa, npr. podskup atributa *{birthday_day, birthday_month, birthday_year}*
- proizvoljnog broja zapisa

Budući da implementirano rješenje predstavlja model koji aproksimira preslikavanje $f: V \rightarrow Y$, tj. funkciju koja čvorovima grafa pridružuje pripadne oznake vrsta osobnih podataka kojoj pripadaju, sada je pravo vrijeme da se predstavi odabrani skup oznaka koje mogu biti pridružene pojedinom čvoru, tj. podatku. Odabrani skup daleko je od iscrpnog, budući da, kako govori i sama definicija osobnih podataka, a prema *OZUP*, osobnim se podatkom može smatrati bilo koji podatak koji samostalno, ili združen s nekim drugim skupom podataka, može identificirati pojedinca kojem pripada. Stoga je za potrebe implementacije programskog rješenja predstavljenog u ovome radu odabran sljedeći skup oznaka vrsta osobnih podataka:

$$C = \{other, name, date, address, ssn, phone\}$$

Oznake redom označuju:

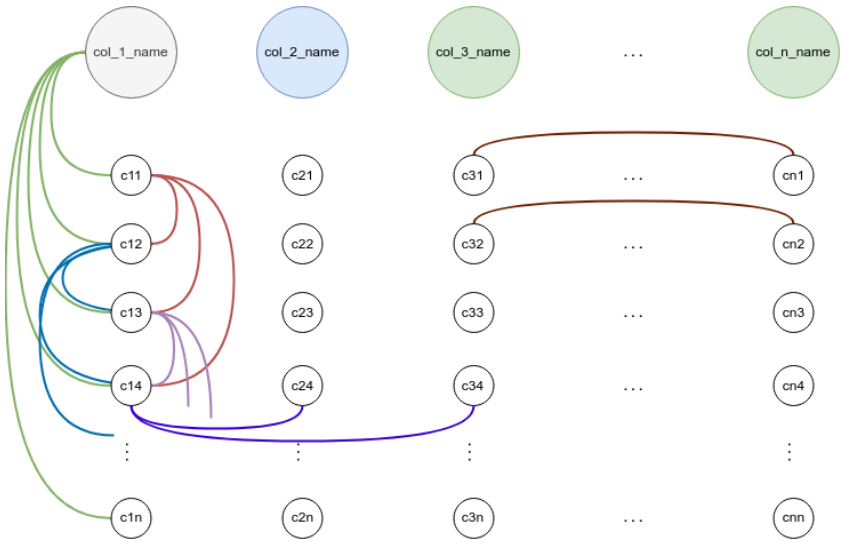
- *other* - općenita skupina podataka koja obuhvaća sve podatke koji nisu prepoznati kao osobni, dakle, predstavlja one podatke koje ne smatramo osobnim
- *name* – skupina osobnih podataka koja predstavlja osobna imena, prezimena i njihove kombinacije
- *date* – skupina osobnih podataka koja predstavlja datumske zapise
- *address* – skupina osobnih podataka koja predstavlja fizičke adrese pojedinaca
- *ssn* – skupina osobnih podataka koja predstavlja broj osiguranika socijalnog osiguranja (engl. *Social Security Number*), koji u američkom pravnom sustavu predstavlja jedinstveni osobni identifikacijski broj, kao i *OIB* u Hrvatskoj
- *phone* – skupina podataka koja predstavlja brojeve telefona, mobitela, telefaks uređaja, itd.

Na slici 10 ispisa podataka i strukture tablice možemo zamijetiti i dodatne skupine podataka koje se mogu smatrati osobnima, poput korisničkog imena, oznake spola, naziva radnog mjesta i naziva poslodavca. Naravno, skup *C* se može proširiti tim oznakama, te to može biti temelj daljnjih istraživanja vezanih uz implementirano programsko rješenje.

Pojasnimo i ulogu oznaka i *pod-oznaka*. Oznaka klase kojoj atribut pripada koristi se prilikom samog učenja modela, dakle za izračun metrika pogreške i točnosti, dok se pod-oznaka koristi prilikom pretvorbe tablične strukture u grafovsku. Svrha ovog postupka jest dodatno povezivanje rastranširanih podataka atributa iste klase osobnih podataka, koji zajedno opisuju jedan cjeloviti podatak; poput podskupa atributa *{birthday_day, birthday_month, birthday_year}* s gornjeg primjera, čije vrijednosti zajedno opisuju neki datum, ili pak podskup atributa *{residence_line_1, residence_city, residence_zip}*, čije vrijednosti zajedno pak opisuju neku adresu. Vrijedno je primijetiti da se u gornjem primjeru nalazi i atribut naziva *main_address* čiji podaci također opisuju neku adresu, no ovaj put u cijelosti. Stoga model treba biti sposoban prepoznavati različite kombinacije atributa pojedine klase i veze između njih. Sama ideja korištenja GNN-ova za klasifikaciju tabličnih podataka, potječe od nastojanja da se model nauči širem kontekstu svakog podatka, tj. da ne donosi zaključke o klasi pojedinog podatka isključivo temeljem lokalnih značajka podataka samo tog atributa kojem podaci pripadaju. Širi se kontekst nastoji ostvariti povezivanjem pojedine ćelije tablice (*uključujući i sam naziv atributa*) s njezinim susjedstvom: susjednim ćelijama promatranog atributa, te ćelijama susjednih atributa. Slika 11 prikazuje način na koji su čvorovi grafa, tj. ćelije tablice, međusobno povezani, nakon transformacije tablične strukture i graf. Autori u već spomenutom rješenju u [49], pojedine rečenice modeliraju kao graf, na način da svaka riječ rečenice predstavlja jedan čvor grafa, a povezana je s prvih *n* susjednih čvorova, tj. riječi, u

nizu. Analogno tome, niz podatka atributa tablice također možemo smatrati nizom riječi u rečenici, no taj bismo *vertikalni* pogled na povezivanje ćelija mogli proširiti povezivanjem neke ćelije s prvih *m* susjeda u retku. Pritom će se, kao što je prikazano na slici 11, smeđom bojom označenih veza, rastranširani podaci povezivati dodatnim vezama, ako su u retku međusobno udaljeni za više od *m skokova*. Nadodajmo i da će se ćelije u grafu povezivati uvijek istim redoslijedom – od gore prema dolje, unutar istog atributa; odnosno s lijeva na desno unutar istog retka, a da pritom posljednjih *n-1*, odnosno *m-1*, čvorova neće biti povezano s *n*, odnosno *m* čvorova.

Dodatni problem predstavlja činjenica što *dodatni* bridovi grafa koji povezuju rastranširane podatke, podrazumijevaju da unaprijed znamo da ti podaci zajedno tvore jedan cjeloviti podatak – no, ti su podaci dostupni jedino unutar skupa za učenje, tj. kada bi se naučeni model trebao koristiti nad stvarnim podacima, te nam informacije o dodatnim bridovima ne bi bile dostupne. Rani eksperimenti su pokazali kako model daje lošije rezultate kada je treniran s dodatnim bridovima, stoga je u konačnoj verziji skupa podataka za učenje, ta informacija izostavljena. Nepostojanje informacije o dodatnim bridovima zapravo može poslužiti kako bi se opisani problem klasifikacije čvorova proširio problemom *predviđanja postojanja bridova između dva čvora*, što također može poslužiti kao temelj budućih nadogradnji implementiranog rješenja.



Slika 11: Primjer pretvorbe tablične u grafovsku strukturu

(Autorski rad, 2022.)

4.2. Prikupljanje podataka

Potrebno je još napomenuti kako je skup za učenje i test modela u potpunosti umjetno generiran, nasumičnim generiranjem podataka iz konačnih skupova prema uniformnoj distribuciji vjerojatnosti. Prilikom tog postupka, korištena je *Python* knjižnica *faker*, a primjeri adresa, osobnih imena i naziva organizacija, dodatno su prošireni javnim skupovima podataka: [55], [56], [57], [58], [59]. Svi su podaci karakteristični za sjevernoameričko govorno područje. Ukupno je izgenerirano 200 tablica, od kojih je 40 (20%) primjera iskorišteno za skup za test. Dodatno je osigurano da se je testni skup primjera generirao podskupom imena, adresa i naziva organizacija koji nisu uključeni u skup za trening. Nazivi atributa tablica također su nasumično izgenerirani (uniformnom distribucijom vjerojatnosti), a nastojao se ostvariti balans između atributa čiji nazivi ispravno upućuju na vrstu osobnih podataka atributa, npr. atribut naziva *ime_kupca* zaista sadrži osobna imena kupaca – i atributa čiji nazivi krivo naslućuju na sadržaj podataka koje sadrže.

U nastavku slijedi opis implementiranog rješenja.

5. Implementirano rješenje

U ovome će poglavlju biti opisano cjelokupno programsko rješenje. Kako bismo ga bolje razumjeli, ponovit ćemo formulaciju zadanog problema:

Neka je tablica T, opisana nekim skupom atributa $\{A_1, A_2, \ldots, A_n\} \subseteq T$ i skupom podataka i naziva atributa $D = \{a_{1,1}, \ldots, a_{1,m}, a_{2,1}, \ldots, a_{2,m}, \ldots, a_{n,1}, \ldots, a_{n,m}\}$, za čije elemente vrijedi$\{a_{i,2}, \ldots, a_{i,m}\} \in A_i \big| \{A_i\} \subseteq T, i \in \{1,2,\ldots,n\}$, odnosno neka skup $\{a_{1,1}, a_{2,1}, \ldots, a_{n,1}\} \subseteq D$ predstavlja skup naziva atributa tablice T. Neka postoji proizvoljan skup $C = \{c_1, c_2, \ldots, c_k\}$, čiji elementi predstavljaju oznake klasa (vrsta) osobnih podataka i neka je svakom atributu iz T pridružena jedna oznaka iz C. Tada definiramo skup oznaka $Y = \{y_{1,1}, \ldots, y_{1,m}, y_{2,1}, \ldots, y_{2,m}, \ldots, y_{n,1}, \ldots, y_{n,m} \big| y_{i,j} \in C\}$ pridružene podacima iz D, tako da pritom vrijedi da je $y_{ij} \in Y$ oznaka pridružena atributu $A_i | \{A_i\} \subseteq T, i \in 1,2,\ldots,n$. Neka je nadalje tablica T, transformirana u graf $G = (V, E)$, pri čemu za skup čvorova vrijedi $V \subseteq D$, a skup Y neka predstavlja skup oznaka čvorova grafa G. Tada se traži: **a)** programsko rješenje koje će svaku tablicu T transformirati u graf $G = (V, E)$ **b)** programsko rješenje koje će čvorove grafa G najprije transformirati u početne vektorske reprezentacije, tako da vrijedi $V = \{h_{i,j}^{(0)} \big| h_{i,j}^{(0)} \in R^{d_0}\}$ **c)** programsko rješenje modela grafovske neuronske mreže koji će naučiti najbolju aproksimaciju klasifikatora $f: V \to Y$ (tj. kako preslikati čvorove grafa G na skup oznaka Y).

U prošlom smo poglavlju definirali mogući, sažeti skup oznaka vrsta osobnih podataka $C = \{other, name, date, address, ssn, phone\}$, tako da je definicija problema sada zaista kompletna.

U nastavku će rada biti predstavljeni i opisani samo oni najbitniji dijelovi cjelokupnog programskog rješenja, dok se sveukupan programski kod implementiranog rješenja može pronaći u repozitoriju na poveznici https://gitlab.com/mcrepinko/ticls.

5.1. Obrada ulaznih podataka i njihova pretvorba u graf

Dakle, u ovom je radu implementiran model grafovske neuronske mreže, nazvan *TICLS* (*Tabular Information Classifier* (hrv. *Klasifikator tabličnih informacija*)), kojem je glavni zadatak klasificirati vrste osobnih podataka u tabličnim strukturama. Osim razvoja i učenja samog modela, razvijeni su i programski moduli za: pripremu podataka, tj. transformaciju tabličnih podataka u grafove; interpretaciju rezultata modela i optimizaciju hiperparametara modela. Najprije će biti objašnjen modul *graph_creation.py*, koji služi za transformaciju podataka. Taj se modul sastoji od razreda *Grapher* i *CustomDataTransformer*.

Razred *Grapher* u svom konstruktoru prima parametre

- *labels* – listu svih oznaka osobnih podataka (skup C),
- *sample_size* – broj podataka koji će biti uzorkovan iz skupa podataka pojedinog atributa,
- *node_embedder* – razred koji služi za transformiranje tekstualnih značajki čvorova (vrijednosti ćelija tablice) u vektorski zapis, pritom je bitno napomenuti da svi podaci u tablici trebaju biti tipa znakovnog niza *string.* U ovom se slučaju ta transformacija vrši nakon učitavanja tablice u *DataFrame* objekt, u metodi *process_nodes* razreda *Grapher.* Predodređeni razred za transformaciju značajki je već ranije opisani [53].
- self_loops – indikator trebaju li se prilikom konstrukcije grafa dodati petlje za svaki čvor, koje taj čvor povezuju sa samim sobom

Najbitnije metode ovog razreda su:

- *process_nodes* – koja transformira podatke tablice u njihove vektorske reprezentacije, tj. producira vektore značajki čvorova. Osim značajki čvorova, kao povratne vrijednosti vraća oznake i pod-oznake pojedinog čvora. Ova se metoda može iskoristiti nad bilo kojom tablicom, čiji su atributi označeni oznakama klasa osobnih podataka, kao na slici 10 u prošlom poglavlju
- *construct_ordinary_graph_edges* – koja stvara listu usmjerenih bridova grafa, čiji su elementi uređeni parovi čvorova, prema zahtjevima opisanim u prošlom poglavlju o pripremi podataka
- *construct_sub_label_edges* – koja stvara dodatne bridove za moguće rastranširane podatke, ako takvi postoje, koji su također opisani u prošlom poglavlju
- *edges_to_coo_format* – metoda koja listu bridova, uređenih parovi čvorova, pretvara u matricu *COO* formata

Razred *CustomDataTransformer* u svome konstruktoru prima listu objekata *torch_geometric.transforms.BaseTransform [60]*, koji implementiraju dodatne operacije

transformacije grafova, opisanih *torch_geometric.data.Data [61]* objektima. Operacija koja se najčešće koristi u ovome rješenju pretvara usmjereni graf u neusmjereni, dodajući dodatne bridove u graf. *Grapher* i *CustomDataTransformer* razrede koristi *load_raw_csv_data* funkcija, definirana u modulu za opće zadatke *utils.py*. Spomenuta funkcija učitava skupove za trening, set i validaciju modela, sačinjenih od *CSV* datoteka. Kada datoteke učita, koristi spomenute razrede za transformaciju tabličinih podataka u grafove. Ona na svome izlazu vraća liste *torch_geometric.data.Data* objekata, koje redom odgovaraju skupu podataka za učenje, trening i validaciju modela. Objekt *torch_geometric.data.Data* zapravo predstavlja jedan graf, a sadrži proizvoljne atribute, između kojih i vektore značajki čvorova, pripadne oznake i *COO* matricu bridova.

5.2. Korištenje, trening i odabir hiperparametara modela

Sljedeći bitan po redu razred jest *Predictor*, implementiran u *predict.py*. Ovaj razred implementira metode za lakše baratanje naučenim *TICLS* modelom, koji se može učitati iz sekundarne memorijske jedinice računala. Instancu *TICLS* modela prima u konstruktoru, isto kao i instance *Grapher* i *CustomDataTransformer* razreda, koje koristi za transformaciju ulaznih podataka. Stoga se metodom *predict* može tražiti zaključak (engl. *inference*) modela o tome kojim vrstama osobnih podataka pripadaju podaci iz tablice, koja se metodi prosljeđuje kao *DataFrame* objekt. Nakon što model vrati rezultate za proslijeđene mu podatke, rezultati se interpretiraju metodom *process_results*. Kako se klasifikacija zapravo vrši na razini čvorova grafa, moguće je da će model različite čvorove grafa klasificirati kao različite vrste osobnih podataka. Trenutna implementacija ove metode, za svaki od atributa tablice konstruira rezultat klasifikacije na način da najprije za svaki čvor dohvati samo one oznake klasa za koje je model najsigurniji da ih je ispravno klasificirao. Potom, prebrojava dohvaćene oznake po klasama i za svaku klasu vraća udio čvorova te klase u ukupnom broju čvorova. Recimo da skup nekog atributa čini pet primjera podataka, tj. pet čvorova grafa, za koje je model zaključio da tri čvora pripadaju klasi *addres*, a dva klasi *name*. Tada vraćeni rezultat za taj atribut predstavlja da je model zaključio da $60\% = (3/5) * 100$ podataka tog atributa pripada klasi *address*, dok 40% podataka pripada klasi *name*. Slika 12, predstavlja primjer rezultata kakvog vraća metoda *predict*, a u ovom su to slučaju rezultati klasifikacije za tablicu prikazanu na slici 10 u prošlom poglavlju. Vidimo da je model ispravno klasificirao podatke svih atributa tablice, osim atributa *id*. Što je razumno zbog čestog preklapanja formata cjelobrojnih vrijednosti atributa koji predstavljaju identifikatore zapisa u tablicama (pr. *primarni ključ* u relacijama baza podataka) s vrijednostima dana i mjeseca u zapisu datuma. Ovakva bi se greška modela mogla otkloniti dodavanjem većeg broja primjera tablica s ovakvim formatom identifikatorskih atributa i ponovnim treniranjem modela.

```
[{'column_name': 'id', 'predictions': [{'date': 1.0}]},
 {'column_name': 'job', 'predictions': [{'other': 1.0}]},
 {'column_name': 'company', 'predictions': [{'other': 1.0}]},
 {'column_name': 'ssn', 'predictions': [{'ssn': 1.0}]},
 {'column_name': 'residence_line_1', 'predictions': [{'address': 1.0}]},
 {'column_name': 'residence_city', 'predictions': [{'address': 1.0}]},
 {'column_name': 'residence_zip', 'predictions': [{'address': 1.0}]},
 {'column_name': 'username', 'predictions': [{'other': 1.0}]},
 {'column_name': 'name', 'predictions': [{'other': 0.1818}, {'name': 0.8182}]},
 {'column_name': 'sex', 'predictions': [{'other': 1.0}]},
 {'column_name': 'main_address', 'predictions': [{'address': 1.0}]},
 {'column_name': 'birthday_day', 'predictions': [{'date': 1.0}]},
 {'column_name': 'birthday_month', 'predictions': [{'date': 1.0}]},
 {'column_name': 'birthday_year', 'predictions': [{'date': 1.0}]}]
```


Slika 12: Primjer rezultata klasifikacije podataka tablice prikazane na Slici 9, korištenjem TICLS modela (Autorski rad, 2022.)

Skripta *train.py* služi za pripremu okruženja za trening modela. Za pripremu skupova za učenje i testiranje modela, nakon što se *Grapher* razredom transformiraju tablični podaci, liste *torch_geometric.data.Data* objekata, koje predstavljaju pojedine tablice skupova transformirane u grafove, učitava u *torch_geometric.loader.DataLoader* [62] objekte. Tim se objektima skupovi podataka automatski dijele na manje podskupove (engl. *mini-batching*), nasumičnim uzorkovanjem, kojima se ubrzava učenje modela, sukladno opisu *SGD* algoritma. *DataLoader* razred prilikom instanciranja objekata, osim samih podataka, prihvaća i vrijednost veličine podskupova koje će uzorkovati za brže učenje. Nakon pripreme podataka, u skripti *train.py* priprema se instanciranje *pytorch_lightning.Trainer [63]* objekta koji upravlja postupkom učenja modela. Usto sprječava prenaučenost modela (engl. *early stopping*), prateći kontinuitet opadanja testne pogreške, koja ako se zadani broj iteracija ne promijeni, ili ako počne ponovno rasti, *Trainer* objekt će zaustaviti učenje modela i pohraniti ga na jedinicu sekundarne memorije računala.

Skripta *tune.py* služi optimizaciji hiperparametara modela. Hiperparametri određuju model, tj. utječu na njegovu složenost pa samim time i na performanse modela [16], kao što su npr. širine skrivenih slojeva, broj slojeva, parametri aktivacijskih funkcija, itd. Hiperparametri se ne uče tijekom treninga modela, već se određuju prije. Knjižnica *PyTorchLightning* uvelike olakšava izbor hiperparametara, na način da se navedu svi hiperparametri modela čija nas optimizacija interesira te rasponi i skupovi vrijednosti nad kojima ih želimo optimizirati. Automatskim se postupkom tada biraju razne kombinacije hiperparametara s kojima se instancira model za trening, a idealan rezultat optimizacije predstavljaju vrijednosti hiperparametara za koje model postiže najveću sposobnost generalizacije nad novim primjerima.

Za kraj je još bitno spomenuti datoteku *config.py*, u kojoj se definiraju svi hiperparametri modela i parametri korišteni za pripremu transformacija podataka u grafovsku strukturu.

5.3. Definicija modela *TICLS*

Za kraj je ostalo još objasniti definiciju samog *TICLS* modela, čiji je programski kod dostupan u Prilogu 1 ovom radu. Model čine ukupno tri *GATv2_CONV* [64] sloja, čija je implementacija već ranije opisana. Drugi i treći (izlazni) sloj na svoje ulaze dobivaju aktivacije iz prethodnih slojeva, ulančane s vektorima značajki ulaza u prethodni sloj i vektorima značajki s ulaza u model, kako bi se izbjeglo zaglađivanje naučenih reprezentacija čvorova. Ovo je samo jedna od brojnih tehnika implementiranja *veza s preskakanjem*. Razred *TICLS* implementiran je prema obrascu kojeg omogućuje *PyTorchLightning* knjižnica, stoga su definirane samo četiri metode:

- ***forward* –** implementira korak unaprijedne propagacije, tj. izračunavanja aktivacija pojedinih slojeva i konačno izlaznog sloja
- ***training_step*** – metoda koja implementira korak učenja modela, tj. unaprijedne propagacije, izračuna pogreške i logiranja rezultata – izračun gradijenata i ažuriranje parametara, obavlja *PyTorchLightning* u pozadini
- ***validation_step*** – implementira korak izračuna testne pogreške za jedan podskup (*mini-batch*) primjera. Prilikom problema klasifikacije s *K* klasa, preporučeno je koristiti funkciju gubitka *unakrsne entropije* (engl. *Cross-entropy loss*) [65], stoga je to funkcija gubitka koju koristi i *TICLS* model.
- ***validation_epoch_end*** – implementira korak izračuna ukupne pogreške za sve podskupove primjera izvornog testnog skupa
- ***configure_optimizer*** – metoda koja inicijalizira objekt tzv. *optimizatora* – optimizator je zapravo konkretna implementacija algoritma ažuriranja parametara modela, poput *SGD* algoritma, a konstruktor razreda optimizatora obično na ulaz prima tenzor s parametrima modela i stopu učenja. *TICLS*, za odabrani optimizator koristi tzv. *Adam* algoritam, a koji predstavlja unaprijeđenu verziju *SGD* algoritma [66][67]. Razlog odabira ovog algoritma jesu njegove dobre performanse [68].

Slijedi pregled rezultata učenja modela.

6. Rezultati

U ovome će poglavlju biti prikazani rezultati optimizacije hiperparametara modela te performanse modela za odabrani skup hiperparametara.

6.1. Priprema podataka i opis učenja modela

Za učenje modela, pripremljen je skup podataka, koji je podijeljen na skup za učenje i na testni skup, na način da su se sirovi podaci tablica, tj. datoteka, sa sintetički generiranim podacima, u *CSV* formatu, učitale sa sekundarne memorijske jedinice računala i transformirale u grafove. Za svaku je datoteku, tj. tablicu, izrađen zaseban graf, a za potrebe pretvorbe, odabrano je deset nasumičnih zapisa svakog atributa – broj nasumičnih zapisa koji će se uzorkovati iz svakog atributa, može se odabrati podešavanjem vrijednosti parametra *sample_size* konstruktora razreda *Grapher*. Kao predodređene vrijednosti, za svaki se čvor, tj. ćeliju tablice, odabralo *2* susjednih čvorova u pripadnom atributu, te *2* susjednih čvorova u pripadnom retku promatranog čvora. Taj se broj može specificira parametrom *num_p_col_neighbors* metode *construct_ordinary_graph_edges* razreda *Grapher*. Budući da su podaci pripremljeni unaprijed, transformirani u grafovske strukture, bili su pohranjeni u binarnom formatu na memorijsku jedinicu računala radi lakšeg i bržeg ponovnog iskorištavanja podataka. Prilikom pokretanja skripte za trening ili hiperoptimizaciju parametara, najprije bi se provjerilo postoje li već takvi transformirani podaci u binarnom obliku, te, ako ne bi postojali, učitale bi se *CSV* datoteke sa sirovim podacima i izvršila bi se transformacija podataka.

Proces učenja sastoji se od dvije faze – faze učenja i testne, ili validacijske faze, kada se provjerava preciznost modela, tj. pogreška, nad podacima testnog skupa. Učenje modela moglo je trajati najviše *EPOCH* (parametar u *config.*py) epoha, a svaka se epoha sastojala od onoliko iteracija, čiji je broj određen brojem podskupova na koje se podijelio skup podataka za učenje, a koji je zavisan o hiperparametru *BATCH_SIZE*. Također, *Trainer* razred u svom konstruktoru omogućuje definiranje parametra *callbacks*, kojim se mogu definirati dodatne postavke učenja, a između ostaloga i broj iteracija za koji će *Trainer* instanca, ako ne dođe do promijene iznosa greške na testnom skupu, zaustaviti učenje. Taj se parametar može definirati u konfiguracijskoj datoteci *config.py*, gdje je nazvan *patience* (hrv. *strpljenje*) i predstavlja ključ rječnika *ERALY_STOPPING_PARAMS* u konfiguracijskoj datoteci. Nakon svake faze učenja, tj. nakon što model izvrši predikcije za svaki od podskupova za učenje, pokreće se testna faza, koja može trajati najviše onoliko iteracija na koliko je podskupova podijeljen testni skup. Optimizacija hiperparametara, odvija se na način da se za svaku odabranu kombinaciju

hiperparametara istrenira model, a na kraju se odabire onaj koji rezultirao najboljim performansama.

6.2. Rezultati optimizacije hiperparametara

Optimizacijom hiperparametara modela, obuhvaćeni su sljedeći hiperparametri:

- ***BATCH_SIZE*** – koji određuje veličinu podskupova za učenje/test, algoritam optimizacije je birao iz skupa vrijednosti $\{8,16,32\}$
- ***hidden_channels_gat1*** – određuje dimenzionalnost, tj. širinu, prvoga po redu *GATv2_CONV* sloja, algoritam optimizacije je birao iz skupa vrijednosti $\{256,512\}$
- ***hidden_channels_gat2*** – određuje dimenzionalnost, tj. širinu, drugoga po redu *GATv2_CONV* sloja, algoritam optimizacije je birao iz skupa vrijednosti $\{128,256,512\}$
- ***learning_rate*** – iznos stope učenja, algoritam optimizacije je birao na intervalu $[1 \times 10^{-4}, 1 \times 10^{-1}]$

Rezultat optimizacije, polučio je sljedeće optimalne vrijednosti navedenih hiperparametara:

- **BATCH_SIZE**: 32
- **hidden_channels_gat1**: 512
- **hidden_channels_gat2**: 128
- **learning_rate**: 0.00022331378497946229

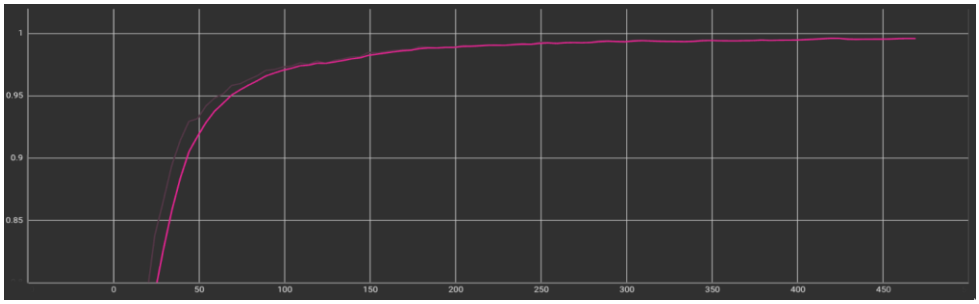
dok su vrijednosti ostalih hiperparametara, koji nisu bili obuhvaćeni optimizacijom, bile sljedeće:

- *input_channels* – širina ulaznog sloja: 768
- *heads_gat_1* – broj mehanizama pozornosti prvog *GATv2_CONV* sloja: 3
- *heads_gat_2* – broj mehanizama pozornosti drugog GATv2_CONV sloja: 1
- *heads_gat_3* – broj mehanizama pozornosti trećeg GATv2_CONV sloja: 1
- *dropout* **–** mjera odbacivanja aktivacija neurona mreže u *Dropout* sloju: 0.5
- *lrelu_neg_slope* – parametar *α LeakyReLU* aktivacijske funkcije: 0.01
- *add_self_loops* – određuje hoće li slojevi modela koristiti petlje, tj. bridove koji čvorove povezuju sa samima sobom: *True*
- use_skip_connection – određuje hoće li slojevi koristiti veze s preskakanjem: True
- *use_prev_layer_input_on_skip_conn* – ako se koriste *veze s preskakanjem*, onda ovaj hiperparametar određuje hoće li na ulaze skrivenog i izlaznog sloja ulančavati i vektori značajki s ulaza u prošli sloj (*True*) ili ne (*False*)

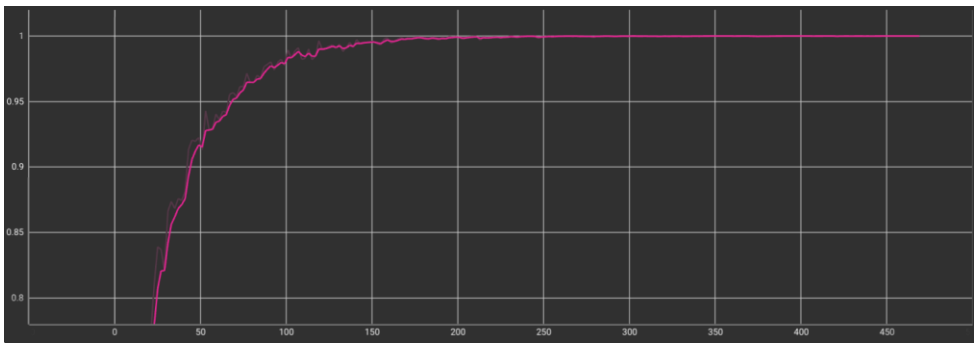
Bitno je napomenuti, kako su svi hiperparametri, koji nisu bili obuhvaćeni optimizacijom, proizvoljno odabrani - uz iznimku hiperparametra *input_channels*, koji predstavlja širinu ulaznog sloja, a ovisi o dimenzionalnosti ulaznih vektora značajki, tj. o odabranom rješenju za vektorsku reprezentaciju ulaznih podataka.

6.3. Performanse modela

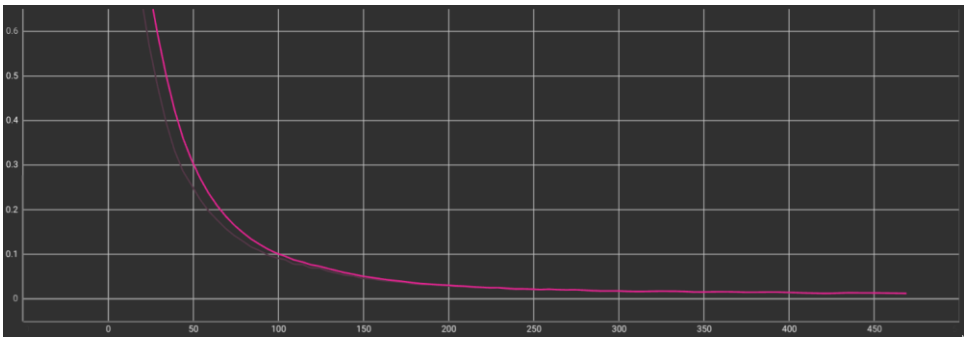
Za navedene je hiperparametre modela postignuta preciznosti (engl. *accuracy*) klasifikacije na testnom skupu podataka od *0.9961*. Preciznost se definira kao omjer ispravno klasificiranih primjera i ukupnog broja primera. Graf kretanja iznosa preciznosti klasifikacije nad testnim skupom kroz iteracije učenja, prikazan je na slici 13. Dok je iznos preciznosti na skupu za učenje iznosio *1.0*. Graf kretanja iznosa preciznosti klasifikacije nad skupom za trening kroz iteracije učenja, prikazan je na slici 14. Iznos pogreške je u posljednjoj iteraciji učenja, nad testnim skupom iznosio *0.01183*. Graf kretanja iznosa pogreške nad testnim skupom, prikazan je na slici 15. Iznos pogreške je u posljednjoj iteraciji učenja, nad skupom za trening iznosio *0.001198*. Graf kretanja iznosa pogreške nad skupom za trening, prikazan je na slici 16.



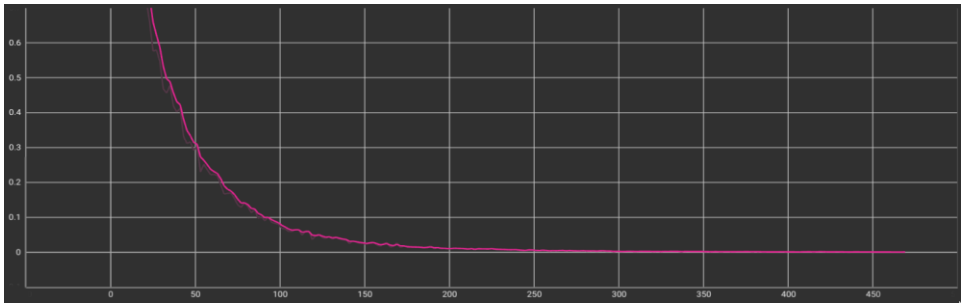
Slika 13: Graf kretanja iznosa preciznosti klasifikacije nad testnim skupom kroz iteracije učenja



Slika 14: Graf kretanja iznosa preciznosti klasifikacije nad skupom za trening kroz iteracije učenja



Slika 15: Graf kretanja iznosa pogreške nad testnim skupom



Slika 16: Graf kretanja iznosa pogreške nad skupom za trening

Trening je trajao ukupno nepunih 93 epoha (469 iteracija), od kojih se svaka epoha sastojala od 5 iteracija, a trening je ukupno trajao *9.73* minute. *Prilog 2* ovome radu sadrži tablični prikaz kretanja iznosa preciznosti i pogreške za testni skup tokom svih koraka učenja modela. Prema iznosu preciznosti nad testnim skupom, možemo zaključiti kako model vrlo dobro generalizira na *neviđenim* primjerima. No, ipak, mjera preciznosti ne govori nam ništa o eventualnoj pristranosti modela da podatke neke klase klasificira bolje od drugih, ili kolika je uopće preciznost po pojedinim klasama, ili kakav je odziv (engl. *recall*) modela po klasama, itd. Stoga dodatni eksperimenti i uvid u dodatne metrike uspješnosti klasifikacije predstavljaju temelj za daljnja istraživanja. Dodatno, za potrebe ovog rada odabran je skroman skup hiperparametara za koje se sprovela optimizacija, stoga bi se u daljnjim istraživanjim taj skup mogao dodatno proširiti. Za sada možemo zaključiti kako model *TICLS* uspješno uči razlike u pojedinim vrstama osobnih podataka, te da je mogućnost upotrebe *GNN*-ova na problemu koji je definiran u ovome radu opravdana, no potreban je daljnji uvid u performanse modela.

Dodatno, budući da se učenjem modela pokazalo da je model sposoban generalizirati na nove primjere, poput tablice prikazane na slici 10, možemo zaključiti i kako je predloženi način transformacije tablične strukture u graf pogodan za rješavanje problema definiranog u ovome radu.

Kako bi se ovi rezultati stavili u širi kontekst, u tablici 1 su prikazane vrijednosti usporedba preciznosti i pogreške dobivenih tijekom optimizacije hiperparametara za ukupno sedam različitih kombinacija hiperparametara. Prvi redak u tablici sadrži gore navedenu kombinaciju hiperparametara, a koja je polučila naučenim modelom s najboljim performansama.

U poglavlju o pripremi podataka je navedeno, kako za potrebe učenja modela, podaci generiranih tablica nisu bili transformirani u grafove koji su uključivali dodatne veze između rastranširanih podataka. Rani su eksperimenti pokazali kako učenje modela nad tako pripremljenim podacima, rezultira modelom s lošijom preciznošću. Razlog tome je činjenica što se podaci o dodatnim vezama neće nalaziti u stvarnim podacima iz realnog svijeta, tj. podacima za koje ćemo vršiti upit nad modelom, stoga te dodatne veze ne smiju biti uključene u testni skup podataka – pa budući da nisu bile uključene u testni skup podataka, model nije raspolagao tim informacijama u testnoj fazi treninga i davao je lošije rezultate. Možemo na taj nedostatak informacija gledati kao na neku značajku temeljem koje je model učen, no koja nedostaje u stvarnim podacima – takav model, prilikom predikcije nad stvarnim podacima, ne može postići performanse postignute u fazi učenja jer ne raspolaže potpunim informacijama. Ovaj scenario možemo usporediti sa slučajem kada se čovjeku, koji je položio vozački ispit za

vožnju nekog motornog vozila te poznaje prometne propise i znakove, dodijeli zadatak da pokuša prepoznati prometni znak koji oblikom nalikuje na neki čovjeku poznati prometni znak, no koji na sebi ne sadrži nikakve grafičke detalje (dodatne značajke), poput boje, oznaka, slova, itd. Naravno, čovjek ne može biti siguran u svoju pretpostavku o kojem se znaku radi, iako je naučen prepoznavati ih. No, kako bi se ipak pokazao odnos performansi modela kada je isti naučen nad verzijom skupa za trening koji je uključivao dodatne veze, odnosno skupa koji nije – nakon što je optimizacija hiperparametara polučila mogući optimalni rezultat, izvršen je jedan trening modela nad podacima skupa za trening koji su uključivali te dodatne veze. Takav skup za trening uključivao je iste podatke kao i onaj koji nije uključivao informacije o dodatnim vezama, stoga je to jedina razlika u rezultatima eksperimenata, čiji su rezultati prikazani u tablici 3. Model je u oba eksperimenta bio učen s istom kombinacijom hiperparametara koja je navedena na početku potpoglavlja. Dodatno, *Prilog 3* ovome radu sadrži tablični prikaz kretanja iznosa preciznosti i pogreške za testni skup tokom svih koraka učenja modela, a koji je treniran na skupu podataka koji je uključivao dodatne veze između rastranširanih podataka.

Tablica 1 - Odnos rezultata optimizacije hiperparametara

R.br.	**Vrijednosti hiperparametara obuhvaćenih optimizacijom**				**Rezultati**	
	BATCH_SIZE	hidden_channels_gat1	hidden_channels_gat2	learning_rate	Preciznost	Pogreška
1	**32**	**512**	**128**	**0,000223314**	**0,99608612**	**0,01182856**
2	8	256	512	0,08545361	0,99146062	0,0173204
3	32	256	256	0,004618149	0,99501866	0,01054362
4	32	512	256	0,017458424	0,99501866	0,01054362
5	16	256	512	0,060866871	0,9928838	0,01958575
6	8	512	256	0,014465338	0,99146062	0,0173204
7	32	512	512	0,0002	0,99501866	0,01054362

Tablica 2 - Usporedba rezultata učenja modela nad skupom podataka koji nije uključivao informacije o dodatnim vezama i skupa koji ih je uključivao

Dodatne veze uključene	**Ne**	**Da**
Preciznost	0,996086121	0,952677488
Gubitak	0,011828557	0,167911261

7. Zaključak

U ovome je radu predstavljen model grafovske neuronske mreže, *TICLS* (*Tabular Information Classifier*). Zadatak koji se njime nastojao riješiti jest definirati model dubokog učenja na grafovima za klasifikaciju vrsta osobnih podataka u tabličnim strukturama. Primarno je u fokusu razvoja modela bio poseban slučaj podijele cjelovitih zapisa atributa na više različitih atributa, poput razlamanja cjelovitog podatka o adresi stanovanja na individualne atribute poput: naziva *ulice, kućnog broja, grada, države*. Za svrhu učenja modela, implementiran je i način transformacije tabličnih podataka u grafovsku strukturu, na način da podaci slogova (ćelija) tablice predstavljaju čvorove grafa, za koje se potom konstruiralo susjedstvo, tj. čvorovi su se povezivali s proizvoljnim brojem susjednih čvorova u istome stupcu (atributu), odnosno retku tablice. Nakon optimizacije hiperparametara, dobiven je model koji ostvaruje visoku preciznost na skupu za učenje (0.9961), no ostalo je još uvijek puno prostora za napredak, poput: proširenja modela na zadatak predviđanja postojanja veza između komponenti cjelovitih podataka, rastranširanih na vrijednosti više atributa; validiranja performansi modela nad stvarnim skupovima podataka; proširenja skupa hiperparametara koji se optimizira; uvid u ekspresivnije mjere točnosti modela; itd. Osim implementacije samog modela i rješenja za transformaciju tablične strukture u graf, implementirana su i programska rješenja za optimizaciju hiperparametara i lakše rukovanje samim modelom nakon učenja.

Osim opisa same implementacije modela dubokog učenja na grafovima, na početku je rada dana i motivacija zašto je uopće bitno da ovakva rješenja postoje. Opisana motivacija dolazi iz područja zaštite osobnih podataka - područja koje se sve češće nastoji zakonski regulirati u sve većem broju država svijeta, pa tako i na području država članica *EU* gdje je ovo područje zaštite podataka regulirano *Općom uredbom za zaštitu podataka*. U radu je predstavljen i iscrpan pregled pristupa sličnim problemima poput zadanog, te je, osim *tradicionalnih* metoda strojnog i dubokog učenja, predstavljeno područje dubokog učenja na grafovima, za čiju se primjenu nastojala stvoriti dobra intuicija, pregled postojećih metoda te neki od glavnih izazova s kojima se možemo susresti u tom području.

Popis literature

[1] ‘Uredba (EU) 2016/679 Europskog parlamenta i Vijeća od 27. travnja 2016. o zaštiti pojedinaca u vezi s obradom osobnih podataka i o slobodnom kretanju takvih podataka te o stavljanju izvan snage Direktive 95/46/EZ (Opća uredba o zaštiti podataka)’. Feb. 13, 2022. Accessed: Sep. 13, 2022. [Online]. Available: https://eur-lex.europa.eu/legal-content/HR/TXT/PDF/?uri=CELEX:32016R0679&from=HR

[2] K. Rabuzin, ‘SPPI: Predavanje 2. Skladišta podataka’. 2021. Accessed: Sep. 13, 2022. [Online]. Available: https://elf.foi.hr/mod/resource/view.php?id=9651

[3] ‘pandas - Python Data Analysis Library’. https://pandas.pydata.org/ (accessed Sep. 14, 2020).

[4] ‘pandas - Python Data Analysis Library’. https://pandas.pydata.org/ (accessed Sep. 14, 2022).

[5] ‘Welcome to Faker’s documentation! — Faker 14.2.0 documentation’. https://faker.readthedocs.io/en/master/ (accessed Sep. 14, 2022).

[6] ‘PyG Documentation — pytorch_geometric documentation’. https://pytorch-geometric.readthedocs.io/en/latest/ (accessed Sep. 14, 2022).

[7] ‘PyTorch’. https://www.pytorch.org (accessed Sep. 14, 2022).

[8] ‘Welcome to ⚡ PyTorch Lightning — PyTorch Lightning 1.7.6 documentation’. https://pytorch-lightning.readthedocs.io/en/stable/ (accessed Sep. 14, 2022).

[9] ‘TensorBoard’, *TensorFlow*. https://www.tensorflow.org/tensorboard (accessed Sep. 14, 2022).

[10] ‘flairNLP/flair’. flair, Sep. 13, 2022. Accessed: Sep. 14, 2022. [Online]. Available: https://github.com/flairNLP/flair

[11] ‘Home - Docker’, May 10, 2022. https://www.docker.com/ (accessed Sep. 15, 2022).

[12] ‘Manjaro’. https://manjaro.org/ (accessed Sep. 15, 2022).

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org/

[14] ‘artificial intelligence - Alan Turing and the beginning of AI | Britannica’. Accessed: Sep. 06, 2022. [Online]. Available: https://www.britannica.com/technology/artificial-intelligence

[15] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th, Global ed. Prentice Hall, 2022.

[16] J. Šnajder, ‘Strojno učenje: 2. Osnovni koncepti’. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2021. Accessed: Sep. 07, 2022. [Online]. Available: https://www.fer.unizg.hr/_download/repository/SU1-2021-P02-OsnovniKoncepti.pdf

[17] E. Alpaydin, *Introduction to machine learning*, 4th ed. MIT Press, 2020.

[18] J. Šnajder, ‘Strojno učenje: 3. Regresija’. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2021. Accessed: Sep. 07, 2022. [Online]. Available: https://www.fer.unizg.hr/_download/repository/SU1-2021-P03-Regresija.pdf

[19] P. Bevandić, M. Oršić, I. Grubišić, J. Šarić, and S. Šegvić, ‘Neslužbene stranice predmeta Duboko učenje 1’, 2022. http://www.zemris.fer.hr/~ssegvic/du/ (accessed Sep. 08, 2022).

[20] J. Krapac and S. Šegvić, *Duboke unaprijedne mreže*. Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, 2022. [Online]. Available: http://www.zemris.fer.hr/~ssegvic/du/du1feedforward.pdf

[21] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. Accessed: Sep. 08, 2022. [Online]. Available: http://neuralnetworksanddeeplearning.com

[22] *Aktivacijske funkcije*. Accessed: Sep. 13, 2022. [Online]. Available: https://raw.githubusercontent.com/shruti-jadon/Data_Science_Images/main/activation_function.png

[23] B. Xu, N. Wang, T. Chen, and M. Li, ‘Empirical Evaluation of Rectified Activations in Convolutional Network’. arXiv, Nov. 27, 2015. Accessed: Sep. 13, 2022. [Online]. Available: http://arxiv.org/abs/1505.00853

[24] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, ‘Geometric deep learning: going beyond Euclidean data’, *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017, doi: 10.1109/MSP.2017.2693418.

[25] J. Zhou *et al.*, ‘Graph neural networks: A review of methods and applications’, *AI Open*, vol. 1, pp. 57–81, 2020, doi: 10.1016/j.aiopen.2021.01.001.

[26] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, ‘A Comprehensive Survey on Graph Neural Networks’, *IEEE Trans. Neural Netw. Learning Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021, doi: 10.1109/TNNLS.2020.2978386.

[27] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, ‘The Graph Neural Network Model’, *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009, doi: 10.1109/TNN.2008.2005605.

[28] L. Wu, P. Cui, J. Pei, and L. Zhao, *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer Nature Singapore, 2022.

[29] Z. Liu and J. Zhou, *Introduction to Graph Neural Networks*. Morgan & Claypool, 2020.

[30] S. Brody, U. Alon, and E. Yahav, ‘How Attentive are Graph Attention Networks?’ arXiv, Jan. 31, 2022. doi: 10.48550/arXiv.2105.14491.

[31] ‘Sparse matrix’, *Wikipedia*. Jun. 02, 2022. Accessed: Sep. 11, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Sparse_matrix&oldid=1091166610

[32] ‘Introduction by Example — pytorch_geometric documentation’. https://pytorch-geometric.readthedocs.io/en/latest/notes/introduction.html (accessed Sep. 11, 2022).

[33] ‘Creating Message Passing Networks — pytorch_geometric documentation’. https://pytorch-geometric.readthedocs.io/en/latest/notes/create_gnn.html#implementing-the-gcn-layer (accessed Sep. 12, 2022).

[34] R. Demush, ‘Gentle Introduction to Graph Neural Networks’, *Perfectial*, 2021. https://perfectial.com/blog/graph-neural-networks-and-graph-convolutional-networks/ (accessed Sep. 12, 2022).

[35] A. Aomar, ‘Over smoothing issue in graph neural network’, *Medium*, Jun. 07, 2021. https://towardsdatascience.com/over-smoothing-issue-in-graph-neural-network-bddc8fbc2472 (accessed Sep. 12, 2022).

[36] K. Xu, M. Zhang, S. Jegelka, and K. Kawaguchi, ‘Optimization of Graph Neural Networks: Implicit Acceleration by Skip Connections and More Depth’, p. 40, 2021.

[37] *CS224W: Machine Learning with Graphs | 2021 | Lecture 7.3 - Stacking layers of a GNN*, (May 04, 2021). Accessed: Sep. 12, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=ew1cnUjRgl4

[38] J. Leskovec, ‘CS224W: Machine Learning with Graphs - 7. Graph Neural Networks 2: Design Space’. Stanford University, 2021. Accessed: Sep. 12, 2022. [Online]. Available: https://web.stanford.edu/class/cs224w/slides/07-GNN2.pdf

[39] C. Morris *et al.*, ‘Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks’, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, Art. no. 01, Jul. 2019, doi: 10.1609/aaai.v33i01.33014602.

[40] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, ‘Provably Powerful Graph Networks’, in *Advances in Neural Information Processing Systems*, 2019, vol. 32. Accessed: Sep. 12, 2022. [Online]. Available: https://papers.nips.cc/paper/2019/hash/bb04af0f7ecaee4aae62035497da1387-Abstract.html

[41] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, ‘Invariant and Equivariant Graph Networks’. arXiv, Apr. 30, 2019. doi: 10.48550/arXiv.1812.09902.

[42] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, ‘How Powerful are Graph Neural Networks?’ arXiv, Feb. 22, 2019. doi: 10.48550/arXiv.1810.00826.

[43] J. Leskovec, ‘CS224W: Machine Learning with Graphs - 9. Theory of Graph Neural Networks’. Stanford University, 2021. Accessed: Sep. 12, 2022. [Online]. Available: https://web.stanford.edu/class/cs224w/slides/09-theory.pdf

[44] *CS224W: Machine Learning with Graphs | 2021 | Lecture 9.2 - Designing the Most Powerful GNNs*, (May 11, 2021). Accessed: Sep. 12, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=B5y47gWt3co

[45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, ‘Graph Attention Networks’, presented at the International Conference on Learning Representations,

Mar. 2022. Accessed: Sep. 09, 2022. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[46] P. Azunre *et al.*, 'Semantic Classification of Tabular Datasets via Character-Level Convolutional Neural Networks'. arXiv, Jan. 24, 2019. Accessed: Sep. 12, 2022. [Online]. Available: http://arxiv.org/abs/1901.08456

[47] M. Hulsebos *et al.*, 'Sherlock: A Deep Learning Approach to Semantic Data Type Detection', in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Anchorage AK USA, Jul. 2019, pp. 1500–1508. doi: 10.1145/3292500.3330993.

[48] E. Kavlakoglu, 'NLP vs. NLU vs. NLG: the differences between three natural language processing concepts'. Nov. 2020. Accessed: Sep. 06, 2022. [Online]. Available: https://www.ibm.com/blogs/watson/2020/11/nlp-vs-nlu-vs-nlg-the-differences-between-three-natural-language-processing-concepts/

[49] Y. Yang and X. Cui, 'Bert-Enhanced Text Graph Neural Network for Classification', *Entropy*, vol. 23, no. 11, Art. no. 11, Nov. 2021, doi: 10.3390/e23111536.

[50] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding'. arXiv, May 24, 2019. Accessed: Sep. 13, 2022. [Online]. Available: http://arxiv.org/abs/1810.04805

[51] B. Lutkevich, 'What is BERT (Language Model) and How Does It Work?', *SearchEnterpriseAI*, 2020. https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model (accessed Sep. 13, 2022).

[52] Y. Suhara *et al.*, 'Annotating Columns with Pre-trained Language Models', in *Proceedings of the 2022 International Conference on Management of Data*, Jun. 2022, pp. 1493–1503. doi: 10.1145/3514221.3517906.

[53] 'bert-base-uncased · Hugging Face'. https://huggingface.co/bert-base-uncased (accessed Sep. 13, 2022).

[54] 'flairNLP/flair - TransformerDocumentEmbeddings'. flair, Sep. 11, 2022. Accessed: Sep. 13, 2022. [Online]. Available: https://github.com/flairNLP/flair/blob/d4ed67bf663e4066517f00397412510d90043653/resources/docs/TUTORIAL_5_DOCUMENT_EMBEDDINGS.md

[55] 'OpenAddresses — Download Data'. https://results.openaddresses.io/ (accessed Sep. 13, 2022).

[56] 'city-of-bloomington/ccc1b422-0f89-462d-9a77-077fc2e3907f | Workspace', *data.world*. https://data.world/city-of-bloomington/ccc1b422-0f89-462d-9a77-077fc2e3907f/workspace/file?filename=bloomington-address-points-gis-data-csv-3.csv (accessed Sep. 13, 2022).

[57] ‘crowdflower/transc-names-from-handwriting | Workspace’, *data.world*. https://data.world/crowdflower/transc-names-from-handwriting/workspace/file?filename=first_and_last_names.csv (accessed Sep. 13, 2022).
[58] ‘alexandra/baby-names | Workspace’, *data.world*. https://data.world/alexandra/baby-names/workspace/file?filename=babynames-clean.csv (accessed Sep. 13, 2022).
[59] ‘7+ Million Company Dataset’. https://www.kaggle.com/datasets/peopledatalabssf/free-7-million-company-dataset (accessed Sep. 13, 2022).
[60] ‘torch_geometric.transforms — pytorch_geometric documentation’. https://pytorch-geometric.readthedocs.io/en/latest/modules/transforms.html?highlight=BaseTransform#torch_geometric.transforms.BaseTransform (accessed Sep. 13, 2022).
[61] ‘torch_geometric.data — pytorch_geometric documentation’. https://pytorch-geometric.readthedocs.io/en/latest/modules/data.html?highlight=data#torch_geometric.data.Data (accessed Sep. 13, 2022).
[62] ‘torch_geometric.loader — pytorch_geometric documentation’. https://pytorch-geometric.readthedocs.io/en/latest/modules/loader.html#torch_geometric.loader.DataLoader (accessed Sep. 13, 2022).
[63] ‘Trainer — PyTorch Lightning 1.7.5 documentation’. https://pytorch-lightning.readthedocs.io/en/stable/common/trainer.html (accessed Sep. 13, 2022).
[64] ‘torch_geometric.nn.conv.gatv2_conv — pytorch_geometric documentation’. https://pytorch-geometric.readthedocs.io/en/latest/_modules/torch_geometric/nn/conv/gatv2_conv.html (accessed Sep. 13, 2022).
[65] ‘CrossEntropyLoss — PyTorch 1.12 documentation’. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html (accessed Sep. 13, 2022).
[66] D. P. Kingma and J. Ba, ‘Adam: A Method for Stochastic Optimization’. arXiv, Jan. 29, 2017. Accessed: Sep. 13, 2022. [Online]. Available: http://arxiv.org/abs/1412.6980
[67] ‘Adam — PyTorch 1.12 documentation’. https://pytorch.org/docs/stable/generated/torch.optim.Adam.html (accessed Sep. 13, 2022).
[68] J. Brownlee, ‘Gentle Introduction to the Adam Optimization Algorithm for Deep Learning’, *Machine Learning Mastery*, Jul. 02, 2017. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (accessed Feb. 06, 2022).

Popis slika

Popis tablica

Prilozi

Prilog 1

```
class TICLS(pl.LightningModule):
    def __init__(
        self,

 input_channels: int,
        hidden_channels_gat1: int,
        hidden_channels_gat2: int,
        num_node_classes: int,
        heads_gat1: int=1,
        heads_gat2: int=1,
        heads_gat3: int=1,
        dropout: float=0.5,
        lrelu_neg_slope: float=0.01,
        learning_rate: float=0.001,
        add_self_loops: bool=True,
        use_skip_connection: bool=False,
        use_prev_layer_input_on_skip_conn: bool=False,
        hparams: Dict[str, Any]=None
    ):
        super(TICLS, self).__init__()
        torch.manual_seed(RANDOM_SEED)

        self.dropout = dropout
        self.lrelu_neg_slope = lrelu_neg_slope
        self.learning_rate = learning_rate
        self.use_skip_connection = use_skip_connection
       self.use_prev_layer_input_on_skip_conn = use_prev_layer_input_on_skip_conn

        self.criterion_train = torch.nn.CrossEntropyLoss()
        self.criterion_eval = torch.nn.CrossEntropyLoss(reduction='sum')

        gat2_input = heads_gat1*hidden_channels_gat1
        gat3_input = heads_gat2*hidden_channels_gat2

        if hparams:
```

```
36            self.save_hyperparameters({
37                "input_channels": input_channels,
38                **hparams["MODEL_PARAMS"],
39                "hparams":hparams
40            })
41
42        if use_skip_connection:
43            gat2_input += input_channels
44            if not self.use_prev_layer_input_on_skip_conn:
45                gat3_input += input_channels
46            else:
47                gat3_input += gat2_input + input_channels
48
49        self.gat1 = GATv2Conv(
50            in_channels=input_channels,
51            out_channels=hidden_channels_gat1,
52            heads=heads_gat1,
53            add_self_loops=add_self_loops
54        )
55
56        self.gat2 = GATv2Conv(
57            in_channels=gat2_input,
58            out_channels=hidden_channels_gat2,
59            heads=heads_gat2,
60            add_self_loops=add_self_loops
61        )
62
63        self.gat3 = GATv2Conv(
64            in_channels=gat3_input,
65            out_channels=num_node_classes,
66            heads=heads_gat3,
67            add_self_loops=add_self_loops
68        )
69
70
71    def forward(self, x, edge_index):
72        xx = self.gat1(x, edge_index)
73        xx = F.leaky_relu(xx, self.lrelu_neg_slope)
74        xx = F.dropout(xx, p=self.dropout, training=self.training)
75
```

```
76          if self.use_skip_connection:
77              xx = torch.cat([xx, x], dim=1)
78
79          xxx = self.gat2(xx, edge_index)
80          xxx = F.leaky_relu(xxx, self.lrelu_neg_slope)
81          xxx = F.dropout(xxx, p=self.dropout, training=self.training)
82
83          if self.use_skip_connection:
84              if not self.use_prev_layer_input_on_skip_conn:
85                  xxx = torch.cat([xxx, x], dim=1)
86              else:
87                  xxx = torch.cat([xxx, xx, x], dim=1)
88
89          xxx = self.gat3(xxx, edge_index)
90          return xxx
91
92
93      def training_step(self, batch, batch_idx):
94          logits_nc = self.forward(batch.x, batch.edge_index)
95
96          loss_nc = self.criterion_train(logits_nc, batch.y)
97
98          pred = logits_nc.argmax(dim=1)
99          correct = int((pred == batch.y).sum())
100         accuracy = correct/len(batch.y)
101
102         self.log("loss/train", loss_nc)
103         self.log("accuracy/train", accuracy)
104
105         return loss_nc
106
107
108     def validation_step(self, batch, batch_idx):
109         logits_nc = self.forward(batch.x, batch.edge_index)
110         pred = logits_nc.argmax(dim=1)  # Use the class with highest probability.
111
112         return logits_nc, pred, batch.y, batch.num_graphs
113
114
115     def validation_epoch_end(self, validation_step_outputs):
```

```
        total_loss = 0.0
        num_correct = 0
        total_nodes = 0
        total_graphs = 0

        for output, pred, labels, num_graphs in validation_step_outputs:
            total_loss += self.criterion_eval(output, labels)

            num_correct += (pred == labels).sum()
            total_nodes += pred.size(0)
            total_graphs += num_graphs

        total_loss = total_loss / total_nodes
        val_accuracy = num_correct / total_nodes

        self.log("accuracy/val", val_accuracy)
        self.log("loss/val", total_loss)


    def configure_optimizers(self):
        return torch.optim.Adam(
            self.parameters(),
            lr=self.learning_rate,
        )
```

Prilog 2

TEST/TRAIN	TEST	
ITERACIJA UČENJA	ACC	LOSS
4	0,43248531	1,25303924
9	0,75057817	1,00656009
14	0,76819074	0,83588624
19	0,78562534	0,68443274
24	0,83775127	0,56846666
29	0,86550438	0,47621065
34	0,89432484	0,39549008
39	0,91442806	0,3318851
44	0,92954987	0,28532717
49	0,93168473	0,25450975
54	0,94200319	0,22262561
59	0,94822985	0,1967935
64	0,95161003	0,1765433
69	0,95837039	0,15757032
74	0,95979363	0,14196014
79	0,96317381	0,12949646
84	0,96619821	0,11623079
89	0,97046787	0,1081176
94	0,97117949	0,09792142
99	0,97313643	0,09175126
104	0,97402596	0,08761447
109	0,9765166	0,07776877
114	0,97527128	0,07719927
119	0,97811776	0,06920558
124	0,9759829	0,06925574
129	0,9784736	0,06175972
134	0,97971892	0,05781433
139	0,98132002	0,0530448
144	0,98149794	0,05126338
149	0,98487812	0,04596842
154	0,98452234	0,04496746
159	0,98558974	0,04198702
164	0,98647928	0,03958007
169	0,9871909	0,03797791
174	0,9871909	0,03635513
179	0,98950362	0,03306827
184	0,98914784	0,03168459
189	0,98843622	0,03157786
194	0,98932576	0,03009747
199	0,98896992	0,02974171
204	0,99057108	0,02737062
209	0,99003738	0,02741188
214	0,99057108	0,02530462
219	0,99110478	0,02498339

224	0,990749	0,0239384
229	0,99057108	0,02512721
234	0,99163848	0,02190421
239	0,99199432	0,02086276
244	0,9912827	0,02194178
249	0,99306172	0,02111603
254	0,9928838	0,01994783
259	0,9918164	0,02216432
264	0,99306172	0,01973963
269	0,9928838	0,01939885
274	0,99252802	0,02053091
279	0,99306172	0,018709
284	0,99430704	0,01733272
289	0,99430704	0,01698468
294	0,99341756	0,01754536
299	0,99323964	0,01787486
304	0,99466288	0,01612261
309	0,99466288	0,01584251
314	0,99395126	0,01624941
319	0,99359542	0,01725683
324	0,99359542	0,01747966
329	0,99359542	0,0169136
334	0,99341756	0,01676648
339	0,99395126	0,01567295
344	0,9948408	0,01428172
349	0,99466288	0,01493478
354	0,99412918	0,01577149
359	0,99412918	0,01591015
364	0,99412918	0,01558097
369	0,99448496	0,01507913
374	0,99448496	0,01423112
379	0,99519658	0,0144978
384	0,99430704	0,01515984
389	0,9948408	0,01520694
394	0,9948408	0,01495134
399	0,9948408	0,01369883
404	0,9953745	0,0132919
409	0,99573028	0,01256346
414	0,99626398	0,01231743
419	0,99661982	0,01179302
424	0,99626398	0,01222131
429	0,99501866	0,01336077
434	0,99519658	0,01412214
439	0,99555242	0,01359435
444	0,99555242	0,01302255
449	0,99555242	0,01319069
454	0,99573028	0,01298618
459	0,99626398	0,01247351
464	0,99626398	0,01223959
469	0,99608612	0,01182856

Prilog 3

TEST/TRAIN	TEST	
ITERACIJA UČENJA	ACC	LOSS
4	0.43248531222343445	1.2527220249176
9	0.7496886849403381	0.9981692433357239
14	0.7633873224258423	0.8320891857147217
19	0.7669453620910645	0.6893971562385559
24	0.8005692958831787	0.5908262729644775
29	0.8144458532333374	0.5248489379882812
34	0.8361501693725586	0.46642935276031494
39	0.8441558480262756	0.4211594760417938
44	0.8731542229652405	0.37013378739356995
49	0.8701298832893372	0.3437601923942566
54	0.8916562795639038	0.30260762572288513
59	0.9025084376335144	0.2772959768772125
64	0.9046432971954346	0.2675604522228241
69	0.9231453537940979	0.23286789655685425
74	0.9202988743782043	0.2373019903898239
79	0.928838312625885	0.2142794132232666
84	0.9336416721343994	0.1983567625284195
89	0.9339975118637085	0.20390631258487701
94	0.9395125508308411	0.1842651516199112
99	0.9428927302360535	0.17721565067768097
104	0.9388009309768677	0.18899907171726227
109	0.9469845294952393	0.16631750762462616
114	0.943070650100708	0.1828610748052597
119	0.9489414691925049	0.1577586829662323
124	0.9411137104034424	0.18980926275253296
129	0.9505426287651062	0.15309949219226837
134	0.9469845294952393	0.17026269435882568
139	0.9541006684303284	0.14939820766448975
144	0.9471624493598938	0.17060932517051697
149	0.9523216485977173	0.15442323684692383
154	0.9505426287651062	0.16192053258419037
159	0.9512542486190796	0.1626734435558319
164	0.953389048576355	0.15340791642665863
169	0.9521437287330627	0.16146992146968842
174	0.9541006684303284	0.14962267875671387
179	0.953389048576355	0.16135156154632568
184	0.9549902081489563	0.15457110106945038
189	0.9526774883270264	0.16791126132011414