Primjena grafova za regularizaciju neuronskih mreža

Židov, Ivan

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike
Smjer: Matematika i računarstvo

Ivan Židov

Primjena grafova za regularizaciju neuronskih mreža

Diplomski rad

Osijek, 2020.

Sveučilište J. J. Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike
Smjer: Matematika i računarstvo

Ivan Židov

Primjena grafova za regularizaciju neuronskih mreža

Diplomski rad

Mentor: doc. dr. sc. Domagoj Ševerdija

Osijek, 2020.

Sadržaj

Uvod

Jedan od glavnih izazova u strojnom učenju je osigurati da model radi dobro ne samo na podacima za učenje nego i na novim podacima. Tehnika kojom se to postiže naziva se regularizacija te ona smanjuje grešku na skupu za testiranje, uz moguće povećanje greške na skupu za učenje.

Graf-regularizacija je posebna tehnika unutar šire grane neuronskih mreža na grafovima (**eng. Graph Neural Networks**). Glavna ideja je trenirati model neuronske mreže tako da je međusobni odnos podataka (graf) uključen u postupak treniranja. Takvim pristupom iskorištavamo i označene i neoznačene podatke.

U ovom radu ćemo proučavati utjecaj graf-regularizacije u svrhu određivanje kategorije kojoj pojedini dokument pripada. Između dokumenata postoje odnosi koji čine prirodni graf. Više o podacima ćemo govoriti u kasnijem poglavlju.

Općenito, graf-regularizacija pomoću paketa Neural Structured Learning (NLS) obično se sastoji od sljedećih koraka:

1. Kreiranje skupa za treniranje načinjenog od ulaznog grafa i odgovarajućih karakteristika. Vrhovi grafa odgovaraju karakteristikama promatranih dokumentima dok bridovi odgovaraju sličnosti između vrhova. Konačni skup za treniranje će sadržavati karakteristike susjednih vrhova kao i karakteristike danog vrha.

2. Kreiranje neuronske mreže koja će se koristiti kao temeljni model. Model ćemo definirati pomoću paketa *Keras*.

3. Osnovni model se "omota" pomoću klase *GraphRegularization* koja je definirana u NSL paketu. To će proširiti funkcionalnost *Keras* modela koji će koristiti graf-regularizaciju prilikom treniranja jer će računati funkciju troška za graf.

4. Treniranje i evaluacija *Keras* modela.

1 Osnove strojnog učenja

Dubinsko učenje (eng. Deep Learning) je posebna vrsta stojnog učenja (eng. Machine Learning). Da bi razumjeli dubinsko učenje, moramo prvo razumjeti osnovne koncepte strojnog učenja. Strojno učenje je algoritam koji može učiti iz podataka. Većina algoritama može se podijeliti u 2 vrste: nadzirano učenje i nenadzirano učenje.

Nadzirano učenje sadrži bazu podataka, koja ima mnogo podataka koji opisuju svaki od primjera i svaki primjer ima neki opis. Taj opis obično vrši čovjek. Učenje bez nadzora prima bazu podataka koja je nestrukturirana. Program mora sam shvatiti kako naučiti svojstva o strukturi podataka te samostalno urediti bazu. Nadzirano učenje je puno lakše i postoje jako dobri algoritmi, no ponekad je skupo ili teško skupljati podatke za svaki primjer. S druge strane, kreiranje baze za učenje bez nadzora je relativno jeftino, ali je programski puno kompleksnije i rjeđe se viđa u praksi.

Puno problema može biti riješeno pomoću strojnog učenja. Najčešći problemi su:

1. **Klasifikacija** - za određen n-dimenzonalan ulaz traži se kojoj od k kategorija pripada, odnosno tražimo funkciju $f : \mathbb{R}^n \to \{1, ..., k\}, n \in \mathbb{N}$. Npr. na temelju slike ručno napisanog broja traži se kojoj kategoriji iz 0,1,...9 pripada.

2. **Regresija** - na temelju inputa traži se neki realan broj, odnosno traži se funkcija $f : \mathbb{R}^n \to \mathbb{R}$ koja će za dani input vratiti realan broj. Npr. na temelju podataka o kvadraturi kuće, broju soba i broju kupaonica, traži se procjena vrijednosti kuće.

3. **Transkripcija** - računalo prima nestrukturiran input i mora od njega napraviti strukturiran tekst. Na primjer, na temelju zvučnog zapisa govora generiraju se titlovi.

4. **Prevođenje** - računalo prima tekst iz jednog jezika i prevodi ga u drugi.

5. **Prepoznavanje anomalija** - na temelju dosadašnjeg uzorka ponašanja, procjenjuje se je li nova akcija neuobičajena. Na primjer, na temelju dosadašnjih platnih navika, procjenjuje se je li nova transakcija uobičajena ili se možda radi o prijevara.

6. **Generiranje podataka** - od algoritma se traži da na temelju nekoliko primjera generiraju nove slične primjere. Koristi se u izradi igara, gdje se na temelju nekoliko slika generira stil umjetnika, jer je crtanje rukom dugotrajno.

7. **Nadopunjavanje podataka koji nedostaju** - za input $\mathbf{x} \in \mathbb{R}$ nedostaje neka vrijednost x_i i nju treba procijeniti. Koristi se u medicini, jer nisu svi podaci dostupni (npr. jer su skupi, podatak zahtjeva invazivan zahvat,...) pa moramo procijeniti kakvi će biti podaci.

8. **Uklanjanje smetnji** - dobijemo neki podatak $\tilde{\mathbf{x}} \in \mathbb{R}^n$ koji ima neku smetnju i od programa se traži da ukloni tu smetnju. Npr. ako imamo sliku auta koji je u magli, tražimo od algoritma da ukloni tu maglu.

1.1 Linearna regresija

Linearna regresija je najjednostavniji oblik regresije. Cilj je konstruirati sustav koji će za vektor $\mathbf{x} \in \mathbb{R}^n$ napraviti procjenu za $y \in \mathbb{R}$. U našem slučaju y će linearno ovisiti o $\mathbf{x}$. Ako je $\hat{y}$ vrijednost koju predviđa naš model tada $\hat{y}$ mora biti što sličniji y. Za procjenu $\hat{y}$ računamo kao

$$\hat{y} = \boldsymbol{\omega}^\top \mathbf{x} = \sum_{i=1}^{n} \omega_i x_i = \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_n x_n$$

gdje je $\boldsymbol{\omega} \in \mathbb{R}^n$ vektor parametara. Parametri su vrijednosti koje kontroliraju ponašanje sustava. U ovom slučaju ω_i je koeficijent koji množimo s karakteristikom x_i prije sumiranja doprinosa svih karakteristika.

Odabirom vektora parametara ω aproksimiramo y s $\hat{y}$, zanima nas koliko je dobra aproksimacija. Jedan od načina za računanje preciznosti je određivanje srednje kvadratne udaljenosti (eng. *mean square error - MSE*). Neka je zadano m test podataka na kojima ćemo mjeriti preciznost. Tada je $\hat{\boldsymbol{y}}^{(test)} \in \mathbb{R}^m$ vektor svih procjena modela na zadanim test podacima, a MSE je dana s:

$$MSE_{test} = \frac{1}{m} \sum_i (\hat{\boldsymbol{y}}_i^{(test)} - \boldsymbol{y}_i^{(test)})^2 = \frac{1}{m} \|\hat{\boldsymbol{y}}^{(test)} - \boldsymbol{y}^{(test)}\|_2^2 \tag{1.1}$$

Jasno je vidljivo da će (1.1) biti jednaka 0 ako je $\hat{\boldsymbol{y}}^{(test)} = \boldsymbol{y}^{(test)}$.

Sada treba pronaći algoritam koji će minimizirati udaljenost (1.1) s obzirom na parametre $\boldsymbol{\omega}$. Neka su $\boldsymbol{X}^{(train)} \in \mathbb{R}^{m \times n}$ i $\boldsymbol{y}^{(train)} \in \mathbb{R}^m$ podaci za treniranje.

Ideja je da se minimizira MSE_{train} na bazi podataka $(\boldsymbol{X}^{(train)}, \boldsymbol{y}^{(train)})$ te provjeri točnost na $(\boldsymbol{X}^{(test)}, \boldsymbol{y}^{(test)})$. Jedan od načina za minimizaciju je traženje minimuma, odnosno gleda se gdje je gradijent od MSE_{train} jednak 0:

$$\nabla_{\boldsymbol{\omega}} MSE_{train} = 0$$

$$\Rightarrow \nabla_{\boldsymbol{\omega}} \frac{1}{m} \|\hat{\boldsymbol{y}}^{(train)} - \boldsymbol{y}^{(train)}\|_2^2 = 0$$

$$\Rightarrow \frac{1}{m} \nabla_{\boldsymbol{\omega}} \|\boldsymbol{X}^{(train)}\boldsymbol{\omega} - \boldsymbol{y}^{(train)}\|_2^2 = 0$$

$$\Rightarrow \nabla_{\boldsymbol{\omega}} (\boldsymbol{X}^{(train)}\boldsymbol{\omega} - \boldsymbol{y}^{(train)})^\top (\boldsymbol{X}^{(train)}\boldsymbol{\omega} - \boldsymbol{y}^{(train)}) = 0$$

$$\Rightarrow \nabla_{\boldsymbol{\omega}} (\boldsymbol{\omega}^\top \boldsymbol{X}^{(train)\top} \boldsymbol{X}^{(train)} \boldsymbol{\omega} - 2\boldsymbol{\omega}^\top \boldsymbol{X}^{(train)\top} \boldsymbol{y}^{(train)} + \boldsymbol{y}^{(train)\top} \boldsymbol{y}^{(train)}) = 0$$

$$\Rightarrow 2\boldsymbol{X}^{(train)\top} \boldsymbol{X}^{(train)} \boldsymbol{\omega} - 2\boldsymbol{X}^{(train)\top} \boldsymbol{y}^{(train)} = 0$$

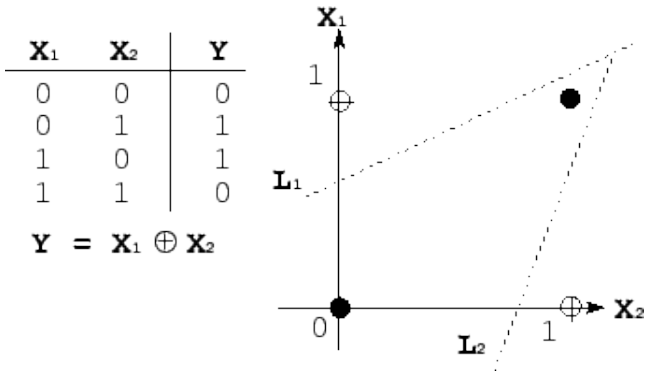
$$\Rightarrow \boldsymbol{\omega} = (\boldsymbol{X}^{(train)\top} \boldsymbol{X}^{(train)})^{-1} \boldsymbol{X}^{(train)\top} \boldsymbol{y}^{(train)} \tag{1.2}$$

Sustav jednadžbi čije je rješenje dano s (1.2) naziva se normalne jednadžbe. Detaljnije o ovome pogledati u [5]. Obično se termin linearne regresije izvodi s jednim dodatnim parametrom b (eng. bias). Tada je model

$$\hat{y} = \boldsymbol{\omega}^\top \mathbf{x} + b = \sum_{i=1}^{n} \omega_i x_i + b = \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_n x_n + b$$

i dalje linearan, ali se može pomicati po y-osi.

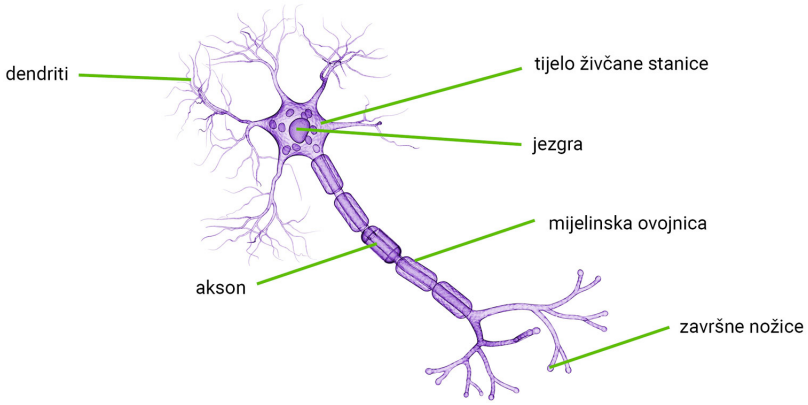
Linearna regresija je jako ograničena što je vidljivo na jednostavnom primjeru logičkog operatora isključivo-ILI (*eng. exclusive-OR; XOR*, oznaka: $\oplus$). Na Slici 1 vidimo da nije moguće odvojiti dvije klase (0 - crne točke, 1 - bijele točke) samo pomoću jednog pravca. Da bi se riješio ovaj problem moramo koristiti barem dva pravca ili uvesti nelinearnost. Jedan od načina uvođenja nelinearnosti je korištenje neuronskih mreža, o kojima ćemo govoriti u nastavku.



Slika 1: Logički operator isključivo-ILI

1.2 Neuron

Neuron ili živčana stanica je osnovni element živčanog sustava (više o neuronima u [10]). Sastoji se od 3 glavna elementa: tijelo s jezgrom, kratki ogranci (dendriti) i dugi ogranak (akson). Najlakše objašnjeno, neuron prima signal kroz dendrite, obrađuje informacije u jezgri te ih šalje u sljedeći neuron pomoću aksona.



Slika 2: Građa neurona

Na sličan način funkcionira i umjetni neuron. To je funkcija koja prima neke podatke (dendriti) i nakon računanja šalje podatke dalje (akson). Najveći problem je pronaći tu funkciju f. Na primjer, ako želimo napraviti klasifikator koji za neki ulazni podatak $\mathbf{x}$ mora odrediti klasu y, tada tražimo funkciju koja mapira $\mathbf{Y} =$

$f(\mathbf{X};\boldsymbol{W})$ gdje su $\boldsymbol{W}$ takvi parametri pomoću kojih je ta aproksimacija najbolja moguća.

Jedan neuron zapravo radi sljedeće: izračunava sumu ulaznih podataka koji ovise o parametrima $\boldsymbol{W}$ i još se dodaje neki prag (eng. bias). Nakon toga neuron odlučuje da li će poslati signal dalje ili ne.

$$Y = \sum_j (\boldsymbol{W_j x_j}) + b$$

Kako je $Y \in \langle -\infty, +\infty \rangle$ neuron i dalje ne zna treba li poslati signal ili ne, odnosno treba li se aktivirati ili ne. U tu se svrhu uvode aktivacijske funkcije.

1.3 Aktivacijske funkcije

Aktivacijske funkcije važan su dio umjetnih neuronskih mreža zato što uvode nelinearnost u modele. U nastavku dajemo pregled najčešće korištenih.

Step funkcija

Najjednostavnija aktivacijska funkcija je step funkcija. Ako je Y veći od neke težine t, onda neka bude neuron aktiviran i neka pošalje signal. Ako bude manje od t, onda nije aktiviran. Aktivacijska funkcija A bi za $t = 0$ izgledala ovako:

$$A(x) = \begin{cases} 1, x \geq 0 \\ 0, \text{inače} \end{cases}$$

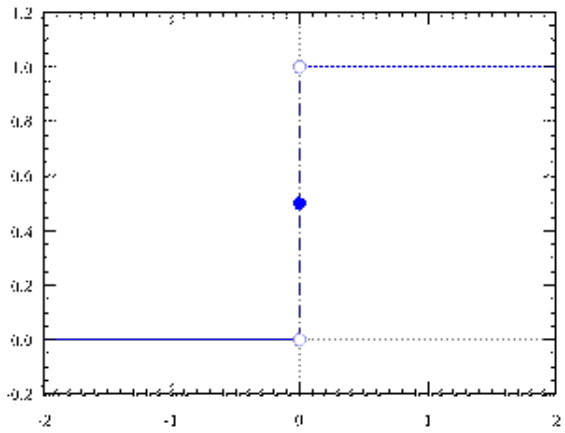
No takva funkcija se nije pokazala dobra u praksi. Jedan od problema je to što je gotovo nemoguće izgraditi klasifikator koji radi za više klasa.

Linearna funkcija

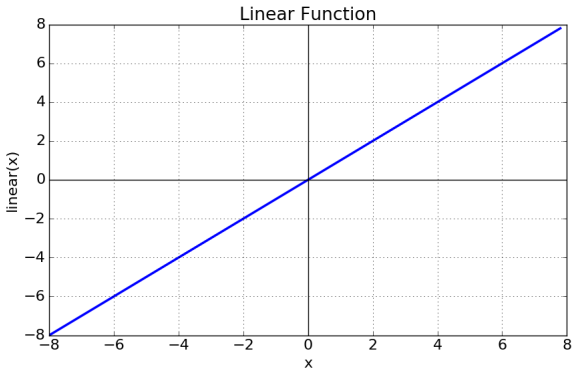
Linearna funkcija je funkcija oblika $A(x) = cx$, gdje je $c \in \mathbb{R}$. U našem slučaju promatramo $A(Y) = cY$. Na taj način nemamo binarnu aktivaciju i možemo povezivati više neurona.

Problemi koji se javljaju s linarnom aktivacijskom funkcijom su ti što ako komponiramo te funkcije u konačnici opet dobivamo novu linearnu funkciju. Kako je nelinearnost svojstvo koje omogućava naprednije zakonitosti, moramo pronaći bolju funkciju.

Sigmoidalna funkcija



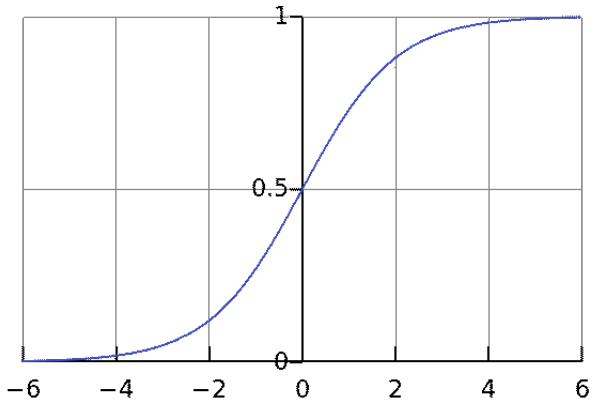
Slika 3: Primjer step funkcije



Slika 4: Primjer linearne funkcije

Sigmoidalna funkcija je funkcija oblika $A(Y) = \frac{1}{1+e^{-Y}}$. Slična je step funkciji, ali je glatka. To nam omogućuje svojstvo nelinearnosti te su njezine kompozicije također nelinearne. Još jedno važno svojstvo koje ima je to što je njezin izlaz uvijek između 0 i 1. Obično se označava s σ.

Slična njoj je i tangens hiperbolna funkcija $f(x) = tg(x) = 2\sigma(2x) - 1$, ali ona daje vrijednosti između -1 i 1.

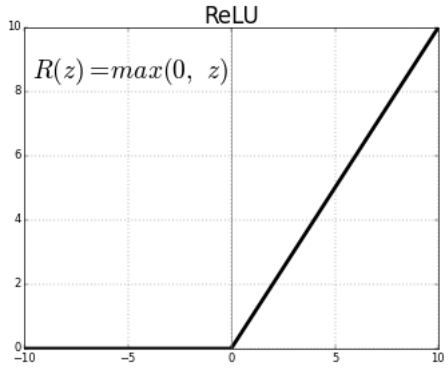


Slika 5: Sigmoidalna funkcija

ReLu funkcija

Rectified linear unit (kraće ReLu) je vrsta aktivacijske funkcije definirana formulom:

$$A(Y) = \begin{cases} 0, \text{ ako je Y} < 0 \\ \text{Y, ako je Y} \geq 0 \end{cases}$$

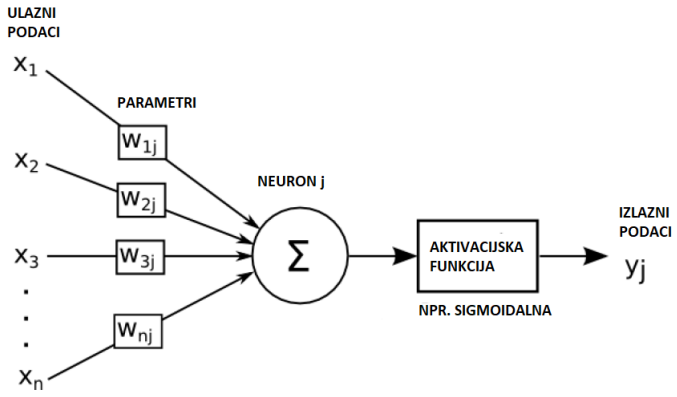


Slika 6: ReLu funkcija

Iako je naizgled vrlo jednostavna, pokazala se jako korisnom u praksi zbog svoje nelinearnosti i gradijenta koji je uvijek 1 ili 0.

1.4 Struktura neuronskih mreža

Umjetni neuron možemo shematski prikazati kao na Slici 7 gdje vidimo da prvo računamo $\sum = \langle w_j, x \rangle + b_j$ te na tu sumu primjenjujemo aktivacijsku funkciju. Te se izračunata vrijednost prosljeđuje do sljedećeg neurona ili je baš ta vrijednost traženo predviđanje.

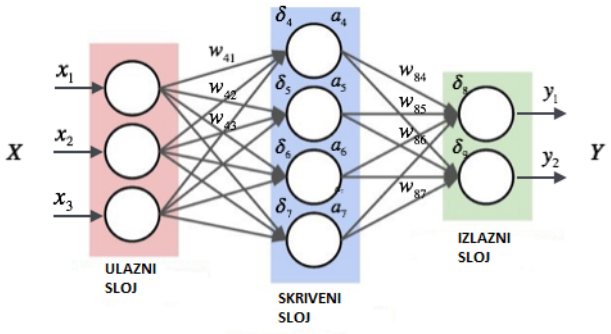


Slika 7: Umjetni neuron

Sama neuronska mreža sastoji se od više neurona koji su raspoređeni u više slojeva. Postoje 3 osnovne vrste slojeva: ulazni sloj, skriveni sloj i izlazni sloj. Ulazni sloj je prvi sloj mreže i on obrađuje ulazne podatke iz sustava. Izlazni sloj je posljednji sloj mreže u kojemu su izlazi neurona zapravo predviđanja rješenja sustava. Svi ostali slojevi su skriveni.

Na Slici 8 prikazan je primjer strukture neuronske mreže, gdje su:

- $x_1, x_2, x_3 \in \mathbb{R}$ ulazi
- $\delta_i = w_{i2}x_2 + w_{i2}x_2 + w_{i3}x_3$ za $i = 4, 5, 6, 7$
- $a_i = \sigma(\delta_i)$ za $i = 4, 5, 6, 7$
- $\delta_i = w_{i4}a_4 + w_{i5}a_5 + w_{i6}x_6 + w_{i7}a_7$ za $i = 8, 9$
- $y_i = \sigma(\delta_i)$ za $i = 8, 9$



Slika 8: Primjer strukture neuronske mreže

1.5 Funkcije pogreške

Najveća razlika između linearnih modela i neuronskih mreža je to što svojstvo nelinearnosti, kod neuronskih mreža, uzrokuje nekonveksnost funkcija pogreške. To je razlog zbog kojeg je teže pronaći točku globalnog minumuma (ukoliko uopće postoji). Da bi se riješio taj problem, koriste se iterativne metode koje postepeno smanjuju funkcije pogreške. Metode se temelje na izračunima gradijenata i ne garantiraju konvergenciju prema globalnom minimumu funkcije, odnosno moguće je zapeti u lokalnom minimumu. Važno je da se svi početni parametri $\boldsymbol{W}$ postave na male nasumične vrijednosti, a pragovi mogu biti 0 ili male pozitivne vrijednosti.

Nakon što se postave početni parametri i početni prag treba izabrati funkciju pogreške i izračunati pogrešku. Jednu od najkorištenijih funkcija pogreške smo već spomenuli. To je srednja kvadratna udaljenost (MSE) koja se računa tako što za svaki primjer računamo razliku između predviđanja modela i stvarne vrijednosti, taj broj kvadriramo te od svih tih vrijednosti uzmemo srednju vrijednost.

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_i (\hat{\boldsymbol{y}}_i^{(test)} - \boldsymbol{y}_i^{(test)})^2$$

Često se još koristi i unakrsna entropija (eng. *cross entropy*). U slučaju binarnih klasifikatora ima oblik:

$$J(\boldsymbol{\theta}) = \frac{-1}{m} \sum_i (\boldsymbol{y}_i \log(\hat{\boldsymbol{y}}_i) + (1 - \boldsymbol{y}_i) \log(1 - \hat{\boldsymbol{y}}_i))$$

1.6 Izlazi neurona

Izbor funkcije troška usko je vezan sa izborom izlaza neurona. Ti izlazi mogu biti na samom kraju mreže ili u skrivenom sloju. Nabrojat i objasnit ćemo najvažnije.

Linearni izlaz za Gaussovu distribuciju
Za dani vektor svojstava $\boldsymbol{h}$, neuron daje izlaz oblika

$$\hat{\boldsymbol{y}} = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}.$$

Linearni izlazi se često koriste za određivanje aritmetičke sredine uvjetovane Gaussove distribucije $p(\boldsymbol{y}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$. Maksimiziranje logaritamske vjerodostojnosti tada se svodi na minimiziranje srednje kvadratne pogreške.

Sigmoidalni izlaz za Bernoullijevu distribuciju
Mnogi zadaci zahtijevaju predviđanje vrijednosti binarne varijable y. Klasifikacija problema s dvije klase je jedan od tih problema. Princip maksimalne vjerodostojnosti nam nalaže da definiramo Bernoullijevu distribuciju za y uz uvjet $\boldsymbol{x}$ (vidi više u [3]). Kako je Bernoullijeva distribucija definirana sa samo jednim brojem, neuronska mreža mora predvidjeti samo $P(y = 1|\boldsymbol{x})$. To predviđanje ima smisla samo ako leži u intervalu $\langle 0, 1\rangle$.

Da bismo postigli takva predviđanja, koristimo sigmoidalan izlaz oblika

$$\hat{\boldsymbol{y}} = \sigma(\boldsymbol{w}^\top \boldsymbol{h} + \boldsymbol{b}).$$

Možemo na sigmoidalan izlaz gledati kao kompoziciju sigmoidalne funkcije i linearne funkcije. Prvo koristimo linearan izlaz za računanje $z = \boldsymbol{w}^\top \boldsymbol{h} + \boldsymbol{b}$, a potom sigmoidalnu aktivacijsku funkciju za pretvaranje z u vjerojatnost.

Softmax izlaz za multinomnu distribuciju
Kada želimo prikazati distribuciju nad diskretnim varijablama s n mogućih vrijednosti, korisno je koristiti softmax funkciju. To je na neki način generalizacija sigmoidalne funkcije. Softmax se često koristi kao izlazni neuron za klasifikatore, jer daje distribuciju za n različitih klasa, odnosno daje nam vektor $\hat{\boldsymbol{y}}$ gdje su $\hat{y}_i = P(y = i|\boldsymbol{x}), i = 1, \ldots, n$. Bitno nam je da svi $\hat{y}_i$ imaju vrijednost između 0 i 1 te da je suma svih $\hat{y}_i$ jednaka 1. Tako dobivamo dobro definiranu distribuciju.

Prvo izračunamo linearni izlaz $z = \boldsymbol{W}^\top \boldsymbol{h} + \boldsymbol{b}$ koji daje nenormalizirane log vjerojatnosti, gdje su $z_i = \log P(y = i|\boldsymbol{x})$ za svaki i. Softmax funkcija je dana sa:

$$softmax(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}}.$$

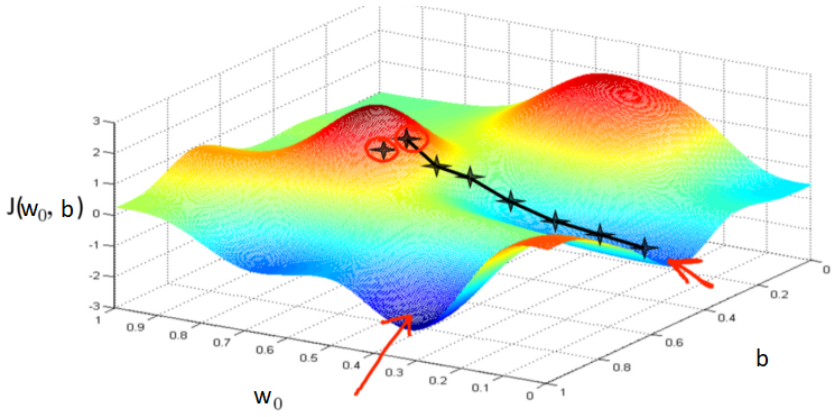
1.7 Učenje neuronskih mreža

U uvodnom dijelu rekli smo da neuronske mreže imaju sposobnost učenja, a u ovom dijelu objasniti ćemo kako se to postiže. Kako bi neuronska mreža mogla rješavati stvarne probleme, potrebno je podesiti njezine parametre. Da bi se to postiglo koriste se dvije važne tehnike: algoritam propagacije pogreške unatrag (eng. *Backpropagation*) i gradijentnu metodu (eng. *Gradient Descent*).

Gradijentna metoda

Gradijentna metoda je optimizacijska metoda koja kod neuronskih mreža minimizira funkciju pogreške. Da bi bolje shvatili kako funkcionira, zamislimo da se nalazimo pri samom vrhu brežuljka. Naš cilj je spustiti se u najnižu točku. Pretpostavimo da je oko nas gusta magla i vidimo samo 5 metara oko sebe. Moramo odlučiti u kojem se smjeru želimo kretati da se spustimo s brežuljka. To ćemo napraviti tako što pogledamo oko sebe i odredimo u kojem smjeru je najveći nagib, odnosno gdje je najstrmije. U tome smjeru se krećemo tih 5 metara. Nakon što smo se pomakli na novu točku, ponovo gledamo oko sebe i krećemo se u smjeru gdje je najveća nizbrdica. Postupak ponavljamo sve dok ne dođemo do takve točke iz koje se više ne možemo spuštati nizbrdo već se samo penjati uzbrdo. Postupak je ilustriran na Slici 9.

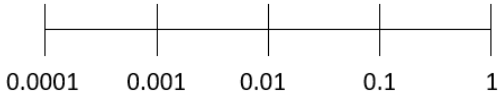
Na sličan način funkcionira gradijentna metoda. Kako se početni parametri sustava postave na neke nasumične vrijednosti, vrijednost funkcije pogreške u početku je jako velika (vrh brežuljka). U toj točci računamo gradijent funkcije troška koji nam govori u kojem je smjeru minimum (gdje je najstrmije). Tada s unaprijed određenim koeficijentom učenja α (5 metara), ažuriramo parametre tako da napreduju prema minimumu. Taj postupak ponavljamo sve dok se krećemo prema minimumu.



Slika 9: Spuštanje gradijenata

$$\boldsymbol{\theta} \leftarrow Prethodno\, zadano$$
$$Ponavljati\, dok\, konvergira\, \{$$
$$\boldsymbol{\theta_j} \leftarrow \boldsymbol{\theta_j} - \alpha \frac{\partial}{\partial \boldsymbol{\theta_j}} J(\boldsymbol{\theta})$$
$$\}$$

Koeficijent učenja (eng. *learning rate*) α je koeficijent brzine konvergiranja. Ako je α prevelik, nećemo doći do minimuma već ćemo divergirati, a ako je α premalen, konvergirat će jako sporo prema minimumu. Da bi pronašli odgovarajući α možemo se koristiti sljedećom skalom. Krećemo s $\alpha = 0.01$ te ako nam model presporo uči, povećamo ga na 0.1. Analogno ako nam model krene divergirati, smanjimo koeficijent na 0.001.



Slika 10: Skala za koeficijent učenja

Koeficijent učenja učenja jedan je od parametara koje moramo unaprijed odrediti te će on značajno utjecati na proces treniranja. Sve parametre koje je potrebno postaviti prije procesa treniranja, nazivamo **hiperparametri**.

Za korištenje gradijentne metode potrebni su nam gradijenti. No to nije trivijalno izračunati za neuronske mreže. Zato se koristi algoritam propagacije pogreške unazad.

Algoritam propagacije pogreške unatrag

Jednostavno rečeno, algoritam propagacije pogreške unatrag je metoda za računanje gradijenta funkcije pogreške u neuronskim mrežama.

Krećemo od početnog izračuna vrijednosti funkcije pogreške. Znamo kako ju derivirati u odnosu na prethodni sloj. Ako znamo derivaciju od svakog sloja, možemo se vraćati unatrag i popraviti parametre u svakom sloju. Pogledajmo rad prvo na primjeru.

Neka je naša mreža zadana s 3 neurona gdje su težine w_1, w_2 i w_3 te pragovi b_1, b_2 i b_3. Neka je $a^{(n)}$ aktivacija posljednjeg neurona te $a^{(L-1)}$ aktivacija predposljednjeg neurona. Želimo da nam $a^{(L)}$ bude jednak stvarnoj vrijednosti y. Pogreška za jedan primjer je

$$J_0(w_1, w_2, w_3, b_1, b_2, b_3) = (a^{(L)} - y)^2.$$

Jedan od načina na koji možemo definirati $a^{(L)}$ je sa $a^{(L)} = \sigma(z^{(L)})$, gdje je $z^{(L)} = w^{(L)}a^{(L-1)} + b^{(L)}$. Želimo izračunti $\frac{\partial J_0(\boldsymbol{\theta})}{\partial w^{(L)}}$, a za to su nam potrebne tehnike složenog deriviranja.

$$\frac{\partial J_0(\boldsymbol{\theta})}{\partial w^{(L)}} = \frac{\partial J_0(\boldsymbol{\theta})}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial w^{(L)}} \tag{1.3}$$

Pogledajmo pojedine parcijalne derivacije vezane za naš primjer:

$$\begin{aligned} \frac{\partial J_0(\boldsymbol{\theta})}{\partial a^{(L)}} &= 2(a^{(L)} - y) \\ \frac{\partial a^{(L)}}{\partial z^{(L)}} &= \sigma'(z^{(L)}) \\ \frac{\partial z^{(L)}}{\partial w^{(L)}} &= a^{(L-1)} \end{aligned}$$

Jednadžba 1.3 nam daje parcijalnu derivaciju samo za jedan primjer. Mi moramo izračunati za sve primjere i uzeti prosječnu vrijednost kako bi model popravili za sve primjere.

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w^{(L)}} = \frac{1}{n}\sum_{i=1}^{n}(2(a_i^{(L)} - y_i)\cdot\sigma'(z_i^{(L)})\cdot a_i^{(L-1)})$$
$$\frac{\partial J(\boldsymbol{\theta})}{\partial b^{(L)}} = \frac{1}{n}\sum_{i=1}^{n}(2(a_i^{(L)} - y_i)\cdot\sigma'(z_i^{(L)}))$$
$$\frac{\partial J(\boldsymbol{\theta})}{\partial a^{(L-1)}} = \frac{1}{n}\sum_{i=1}^{n}(2(a_i^{(L)} - y_i)\cdot\sigma'(z_i^{(L)})\cdot w_i^{(L)})$$

Sada kad smo izračunali derivaciju za zadnji sloj, izračunajmo ju za predzanji tako što koristimo informacije iz zadnjeg sloja. Postupak ponavljamo sve dok ne izračunamo sve derivacije.

$$\frac{\partial J(\boldsymbol{\theta})}{\partial w^{(L-1)}} = \frac{\partial J(\boldsymbol{\theta})}{\partial a^{(L-1)}}\cdot\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}}\cdot\frac{\partial z^{(L-1)}}{\partial w^{(L-1)}} \tag{1.4}$$

Vidimo da su nam potrebne derivacije aktivacijskih funkcija. Izdvojimo neke od njih:

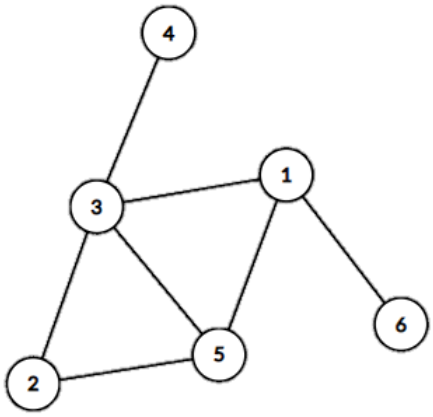
Aktivacijska funkcija	Formula	Derivacija
Linearna	$f(x) = cx + b$	$f(x)' = c$
Sigmoidalna	$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f(x)' = f(x)(1 - f(x))$
Tangens hiperbolni	$f(x) = tg(x)$	$f(x)' = 1 - f(x)^2$
ReLu	$f(x) = \begin{cases} 0, \text{ ako je x} < 0 \\ \text{x, ako je x} \geq 0 \end{cases}$	$f(x) = \begin{cases} 0, \text{ ako je x} < 0 \\ 1, \text{ ako je x} \geq 0 \end{cases}$

Tablica 1: Aktivacijske funkcije i njihove derivacije

2 Osnove teorije grafova

U ovom dijelu ćemo spomenuti samo osnovne definicije koje su nam potrebne u daljnjem dijelu rada.

Neusmjeren graf (*eng. undirected graph*) je uređen par skupova $G = (V, E)$. Elemente skupa V nazivamo **vrhovima** (ili čvorovima ili točkama; *eng. vertex, node*), dok elemente skupa E nazivamo **bridovima** (ili vezama ili lukovima; *eng. edge*). Svaki brid je par elemenata iz V, tj. $e = \{u, v\}$ gdje su $u, v \in V$. Vrhovi u i v čine krajeve brida te za brid e kažemo da povezuje vrhove u i v. Za dva vrha u i v kažemo da su **susjedni** ako postoji brid koji ih povezuje. Ukoliko brid povezuje dva ista vrha, njega nazivamo **petljom**. Ukoliko postoji više bridova koji povezuju iste čvorove, kažemo da u grafu postoji **višestruki brid**.

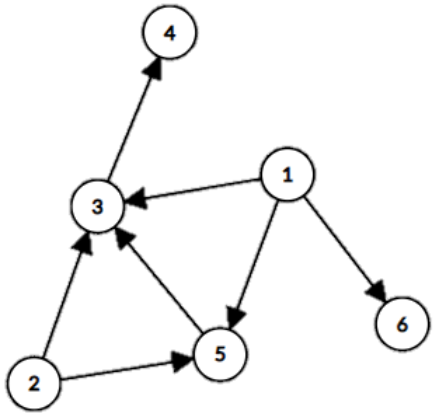


$$V = \{1,2,3,4,5,6\}$$

$$E = \{\{1,3\},\{1,5\},\{1,6\},\{2,5\},\{2,3\},\{3,4\},\{3,5\}\}$$

Slika 11: Primjer neusmjerenog grafa

Usmjereni graf (*eng. directed graph*) se razlikuje od neusmjerenog grafa po tome što je brid u usmjerenom grafu uređen par vrhova, tj. za brid $e = (u, v)$ kažemo da povezuje vrh u s vrhom v, dok obrat ne mora nužno vrijediti. Tada vrh u nazivamo **izvorište** (*eng. source*), a vrh v nazivamo **odredište** (*eng. target*).



Slika 12: Primjer usmjerenog grafa

Jednostavan graf je konačan usmjeren ili neusmjeren graf koji ne sadrži petlje ili višestruke bridove. Za jednostavan graf $G = (V, E)$ definiramo **stupanj vrha** (*eng. degree*) $v \in V$ kao:

- **Neusmjereni graf:**

$$deg_G(v) = |\{u \in V : \{u, v\} \in E\}|$$

- **Usmjereni graf:**
  Ulazni stupanj vrha:

$$deg_G^{IN}(v) = |\{u \in V : (u, v) \in E\}|$$

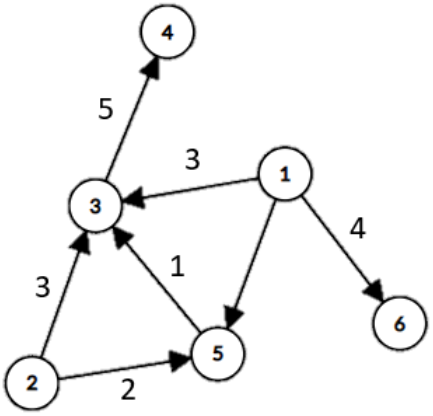
  Izlazni stupanj vrha:

$$deg_G^{OUT}(v) = |\{u \in V : (v, u) \in E\}|$$

  Stupanj vrha:

$$deg_G(v) = deg_G^{OUT}(v) - deg_G^{IN}(v)$$

Težinski grafovi (*eng. weighted graphs*) definiraju se kao uređena trojka $G = (V, E, W)$. Skup V je skup vrhova, E skup bridova i W matrica težina bridova. Razlikuju se od bestežinskih po tome što svaki brid ima pridruženu **težinu** (*eng. weight*). Ponekad se umjesto pojma težina koristi pojam **cijena** (*eng. cost*). Oni, također, mogu biti usmjereni i neusmjerni. Bestežinske grafove (dakle one opisane do ovog dijela) možemo zamisliti kao težinske kojima su svi bridovi iste težine.



vrh	deg_G^{IN}	deg_G^{OUT}	deg_G
1	0	3	3
2	0	2	2
3	3	1	-2
4	1	0	-1
5	2	1	-1
6	1	0	-1

$$V = \{1,2,3,4,5,6\}$$

$$E = \{(1,3), (1,5), (1,6), (2,5), (2,3), (3,4), (5,3)\}$$

Slika 13: Primjer usmjerenog težinskog grafa i stupnjevi vrhova

Podgraf grafa $G = (V, E)$ je graf $G' = (V', E')$, gdje je V' podskup skupa V, a E' podskup skupa E s time da oba vrha svakog brida iz E' se moraju nalaziti u V'. Kažemo da je G **nadgraf** od G' te pišemo $G \subseteq G'$.

Grafove se može računalno implementirati pomoću **matrice susjedstva** kao $A_G[u, v] = |\{(u, v) \in E : u, v \in V\}|$ ili pomoću **matrice incidencije**

$$I_G[v, e] = \begin{cases} 0, & v \text{ nije kraj od } e \\ 1, & v \text{ je kraj od } e \\ 2, & v \text{ je petlja} \end{cases} \tag{2.1}$$

3 Cora skup podataka

Cora dataset je jedan od osnovnih skupova podataka za učenje na grafovima. Ovaj skup podataka se sastoji od znanstvenih radova i njihovih međusobnih citata. Svaki vrh grafa predstavlja znanstveni rad, a svaki brid predstavlja referencu na drugi znanstveni rad. Zadatak je svakom znanstvenom radu pridružiti jednu od 7 kategorija, tj. imamo "multi-class" klasifikacijski problem.

Izvorno, podaci u skupu podataka tvore usmjeren graf. Međutim, u svrhu ovog primjera razmatramo neusmjerenu verziju ovog grafa. Dakle, ako znanstveni rad A citira znanstveni rad B, također smatramo da je znanstveni rad B citirao A. Iako to nije nužno istina, u ovom primjeru citate smatramo aproksimacijom sličnosti između vrhova, što je obično komutativno svojstvo.

Cora dataset se sastoji od dvije datoteke:

1. Značajke ("cora.content")

   Svaki redak u ulazu sadrži značajke:

   (a) **ID**: Jedinstveni identifikator znanstvenog rada.

   (b) **Words**: Riječi koje se nalaze u znanstvenom radu. Rječnik za Cora dataset sadrži 1433 jedinstvenih riječi. Dakle, svaki vektor je dimenzije 1433 te se na i-toj poziciji nalazi 0 ili 1 koji pokazuje dali se riječ na i-toj poziciji pojavljuje u danom tekstu, dakle to je oblika $\{0,1\}^{1433}$. Takvu reprezentaciju nazivamo "multi-hot" reprezentacija.

   (c) **Subject**: Kategorija kojoj pripada znanstveni rad.

ID	w_0	w_1	w_2	$\cdots$	w_1430	w_1431	w_1432	subject
1128927	0	1	0	$\cdots$	0	0	0	Rule_Learning
1152711	0	0	0	$\cdots$	0	0	0	Neural_Networks
137868	0	0	0	$\cdots$	0	1	0	Probabilistic_Methods
1129494	0	0	0	$\cdots$	0	0	0	Neural_Networks
346292	0	0	0	$\cdots$	0	0	1	Neural_Networks

Tablica 2: Primjeri podataka iz "cora.content" datoteke

2. Graf citiranja ("cora.cites")

Svaki redak sadrži:

(a) **target**: ID rada kojega se citira.

(b) **source**: ID rada koji citira.

ID	target	source
2517	27243	59045
626	2654	463825
417	1365	1129443
4543	210871	1117476
3050	39890	714289

Tablica 3: Primjeri podataka iz "cora.cites" datoteke

3.1 Eksplorativna analiza podataka

3.1.1 Graf citiranja

ID	source
66594	5
30895	5
6910	5
1131236	5
1113934	5
⋮	⋮
107177	1
152227	1
8865	1
1131165	1
1122304	1

Tablica 4: ID rada te broj radova koje citira

ID	target
35	166
6213	76
1365	74
3229	61
114	42
⋮	⋮
264347	1
346243	1
221302	1
143476	1
851968	1

Tablica 5: ID rada te broj radova koji ga citiraju

Primjećujemo u Tablici 4 da svaki rad citira najviše 5 drugih radova, te najmanje samo jedan. Tablica 7 nam pokazuje da je 95% radova citirano do 10 puta, dok

je rad ID = 35 citiran daleko najviše puta, čak 166 puta. Ako ovaj graf zapišemo matematički kao $G = (V, E)$ to znači da je:

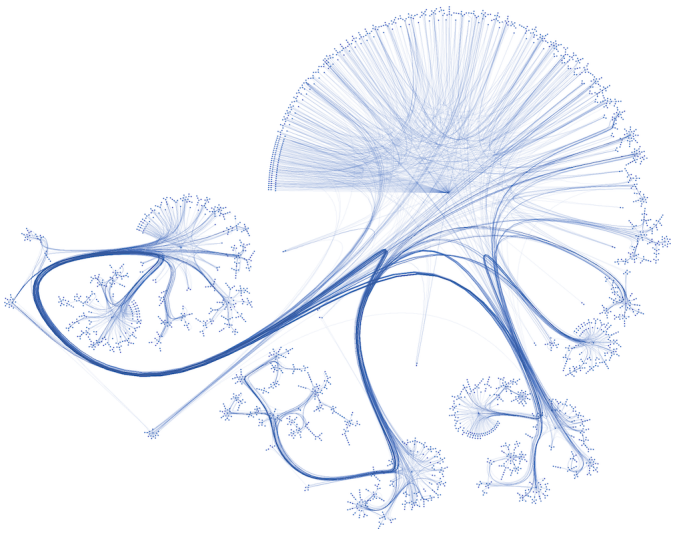
$$\max_{v \in V}\{deg_G^{OUT}(v)\} = 5 \quad \max_{v \in V}\{deg_G^{IN}(v)\} = 166$$

	Relativna frekvencija
1	0.289379
2	0.280378
3	0.208821
4	0.140414
5	0.081008

Tablica 6: Distribucija broja citiranih radova

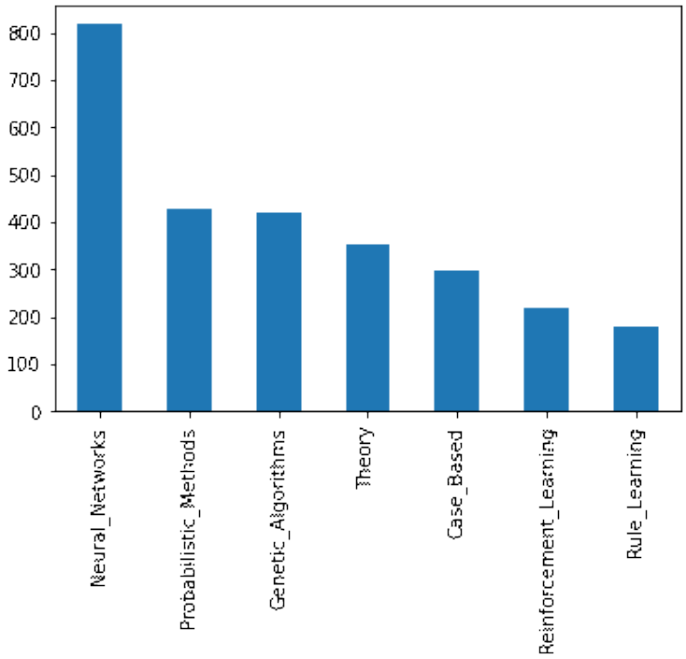
	target
count	1565.000000
mean	3.469010
std	6.485659
min	1.000000
25%	1.000000
50%	2.000000
75%	4.000000
95%	10.000000
max	166.000000

Tablica 7: Statističke značajke citiranja pojedinog rada



Slika 14: Vizualizacija međusobnih citata u datasetu (izvor [11])

3.1.2 Značajke



Slika 15: Distribucija kategorija

Kao što vidimo i na Slici 15, radovi su podijeljeni u 7 kategorija:

	Kategorija	Relativna frekvencija
1.	Neural_Networks	0.302068
2.	Probabilistic_Methods	0.157312
3.	Genetic_Algorithms	0.154357
4.	Theory	0.129616
5.	Case_Based	0.110044
6.	Reinforcement_Learning	0.080133
7.	Rule_Learning	0.066470

Najzastupljenija kategorija ima relativnu frekvenciju 30.2%, što znači da naš model mora imati preciznost veću od 30.2% inače je bolje koristiti model koji uvijek predviđa "*Neural_Networks*" kao kategoriju. Takve jednostavne modele nazivamo "baseline" modeli i oni postavljaju minimalnu granicu koju naš model mora zadovoljavati. Više o tome kako i zašto koristi "baseline" model pročitati u [6]. Kako bismo

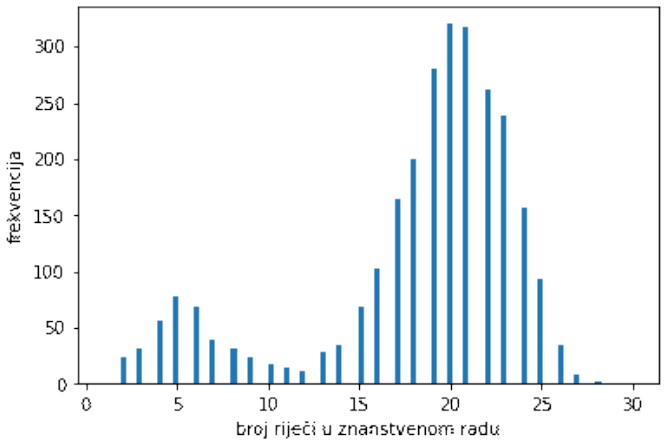
razumjeli strukturu podataka, provjerimo koliki broj riječi se nalazi u dokumentima. To ćemo dobiti tako što ćemo sumirati "multi-hot" reprezentaciju po retcima. U Tablici 9 primjećujemo da se u dokumentima najčešće pojavljuje između 17 i 22 riječi dok je maksimalno 30 riječi u jednom dokumentu.

ID	count
31336	20
1061127	17
1106406	22
13195	21
37879	23
⋮	⋮
1128975	26
1128977	21
1128978	18
117328	19
24043	19

Tablica 8: Broj riječi u radovima

count	2708.000000
mean	18.169867
std	5.756860
min	1.000000
25%	17.000000
50%	20.000000
75%	22.000000
max	30.000000

Tablica 9: Statističke značajke broja riječi u pojedinim radovima



Slika 16: Distribucija broja riječi u radovima

4 Graf regularizacija

4.1 Polu-nadzirano učenje

Polu-nadzirano učenje je moćna tehnika strojnog učenja koja često daje bolje rezultate u odnosu na klasično nadzirano učenje [4]. Takvi rezultati postižu se zahvaljujući velikom broju neoznačenih podataka koji se koriste prilikom treniranja modela na označenim podacima.

Polu-nadzirano učenje često se koristi u problemima računalnog vida, obrade prirodnog jezika i društvenih mreža, gdje je označavanje podataka skupo i dugotrajno, no postoji velika količina neoznačenih podataka [13].

Postoji više načina kako pristupiti takvoj vrsti problema. Osnovni pristup je natrenirati model na označenim podacima te ga iskoristiti za označavanje ostalih podataka. Takve podatke zovemo pseudo-oznake te se one koriste za treniranje novog modela zajedno sa skupom stvarnih oznaka. No takav pristup nije dao očekivane rezultate jer se u pseudo-oznake uvodi greška čime se točnost daljnjeg treniranja dovodi u pitanje.

Značajniji iskorak pružile su metode temeljene na grafovima koje su pružile skalabilnost, prilagodljivost i efikasnost na različitim skupovima problema. Te metode konstruiraju graf nad označenim i neoznačenim podacima, jer su grafovi prirodan način za definiranje odnosa među vrhovima. Bridovi grafa povezuju semantički slične vrhove, a težine bridova ukazuju na snagu sličnosti. Pomoću skupa označenih podataka, te vrste tehnika iterativno prikupljaju informacije susjednih vrhova koje koriste prilikom označavanja danog vrha. Ažurirano stanje vrha zatim se prosljeđuje susjednim vrhovima za njihovo ažuriranje. Takav pristup se u praksi pokazao praktičan jer brzo konvergira te je skalabilan za velike skupove podataka.

4.2 Graf regularizacija

Nastavno na korištenje grafova za polu-nadzirano učenje, u znanstvenom radu [1] predložili su novu funkciju pogreške koja koristi graf augmentaciju te može biti trenirana metodom gradijentnog spusta. Nova funkcija pogreške ima regularizacijski izraz koji podržava više vrsta neuronskih mreža koji se temelji na sličnosti vrhova u grafu. Pokazali su sljedeće:

- graf-augmentirane neuronske mreže korisne su za unaprijedne (*eng. feed-forward*), konvolucijske (*eng. convolution*) i povratne (*eng. recurrent*) mreže.

- moguće je koristiti više vrsta grafova (usmjereni, neusmjereni, težinski,...)
- pomoću graf-augmentacije mogu se pronaći manji i jednostavniji modeli čiji su rezultati usporedivi s značajno većim i kompleksnijim modelima

Neka je dan trening skup $\{\boldsymbol{x_i}, y_i\}, i = 1, \ldots, N$, gdje je $\boldsymbol{x_i} \in \mathbb{R}^n, y_i \in \mathbb{R}$, koji čini graf $G = (V, E, W)$. Također, neka je V_L skup označenih podataka i V_U skup neoznačenih podataka. Za takve pretpostavke definirali su danu funkciju pogreške:

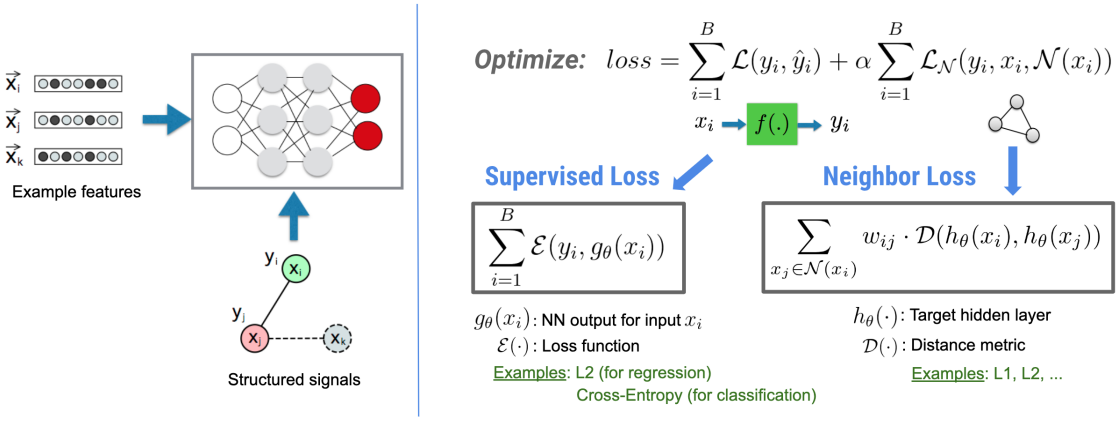
$$\begin{aligned} C_{NGM}(\theta) \quad &= \alpha_1 \sum_{(u,v)\in\mathcal{E}_{LL}} w_{uv} d(h_\theta(\boldsymbol{x_u}), h_\theta(\boldsymbol{x_v})) \\ &+\alpha_2 \sum_{(u,v)\in\mathcal{E}_{LU}} w_{uv} d(h_\theta(\boldsymbol{x_u}), h_\theta(\boldsymbol{x_v})) \\ &+\alpha_3 \sum_{(u,v)\in\mathcal{E}_{UU}} w_{uv} d(h_\theta(\boldsymbol{x_u}), h_\theta(\boldsymbol{x_v})) \\ &+ \sum_{(x_n,y_n)\in V_L} c(g_\theta(\boldsymbol{x_n}), y_n) \end{aligned} \tag{4.1}$$

gdje je:

- $\mathcal{E}_{LL}$ - skup bridova između označenih vrhova
- $\mathcal{E}_{LU}$ - skup bridova između jednog označenog i jednog neoznačenog vrha
- $\mathcal{E}_{UU}$ - skup bridova između neoznačenih vrhova
- w_{uv} - težina brida između vrhova (u, v)
- θ - parametri modela
- $h_\theta(x_u)$ - skrivena reprezentacija (može biti i ulaz) producirana u modelu za ulaz x_u
- $d(\cdot)$ - metrika za računanje udaljenosti
- $\{\alpha_1, \alpha_2, \alpha_3\}$ - hiperparametri
- $g_\theta(x_u)$ - izlaz zadnjeg sloja modela za ulaz x_u (npr. sigmoid)
- $c(\cdot)$ - funkcija pogreške za označene podatke (npr. binarna unakrsna entropija; *eng. binary cross-entropy*)

Implementaciju u Neural Structured Learning paketu možemo vidjeti na Slici 17 te se ona razlikuje od funkcije (4.1) po tome što su ovdje hiperparametri $\{\alpha_1, \alpha_2, \alpha_3\}$ jednaki, tj. $\alpha = \alpha_1 = \alpha_2 = \alpha_3$. Ako vrijede iste oznake kao i kod funkcije (4.1), možemo implementaciju zapisati na sljedeći način:

$$\begin{aligned} C_{NGM}(\theta) \quad &= \alpha \sum_{(u,v)\in E} w_{uv} d(h_\theta(\boldsymbol{x_u}), h_\theta(\boldsymbol{x_v})) \\ &+ \sum_{(x_n,y_n)\in V_L} c(g_\theta(\boldsymbol{x_n}), y_n) \end{aligned}$$



Slika 17: Neural Structured Learning implementacija (izvor [9])

5 Primjer korištenja NSL paketa

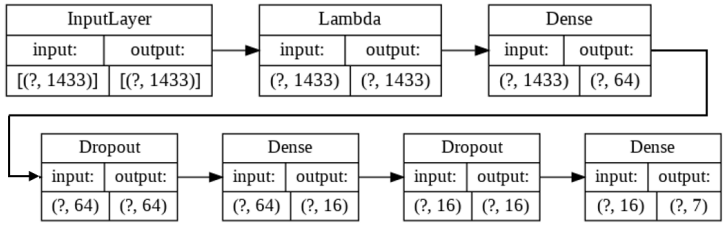
Kako bismo uspješno istrenirali model bilo je potrebno odrediti neke hiperparametre i konstante. U svrhu ovog primjera korištena je instanca *HParams* u kojoj se definira:

- **num_classes** = 7
  Broj klasa koje želimo predvidjeti. U našem slučaju 7.
- **max_seq_length** = 1433
  Ovo je maksimalna veličina rječnika koji ulazi model. U našem slučaju to je 1433 jer imamo 1433 riječi koje ulaze u model u "multi-hot" obliku $\{0,1\}^{1433}$.
- **distance_type** = nsl.configs.DistanceType.L2
  Vrsta metrike koju ćemo koristiti prilikom regularizacije uzorka s njegovim susjedima. U funkciji (4.1) je oznaka $d(\cdot)$.
- **graph_regularization_multiplier** = 0.2
  Koeficijent graf-regularizacije koji će utjecati na ukupni trošak modela. U funkciji (4.1) to je $\alpha = \alpha_1 = \alpha_2 = \alpha_3 = 0.2$.
- **num_neighbors** = 7
  Broj susjednih vrhova koji se koristi za graf-regularizaciju. Ako je $G = (V, E)$ originalni graf, uzimamo njegov podgraf $G' = (V, E')$ takav da je maksimalan supanj vrha jednak **num_neighbors**.
- **num_fc_units** = [64, 16]
  Broj slojeva neuronske mreže.
- **train_epochs** = 100
  Broj trening epoha.
- **batch_size** = 128
  Batch size korišten za treniranje i evaluaciju.
- **dropout_rate** = 0.2
  Dropout je korišten u "fully connected" slojevima.
- **eval_steps** = None
  Broj batcheva procesiranih prije prekida evaluacije. Ako je postavljen na *None*, sve instance testnog skupa biti će evaluirane.

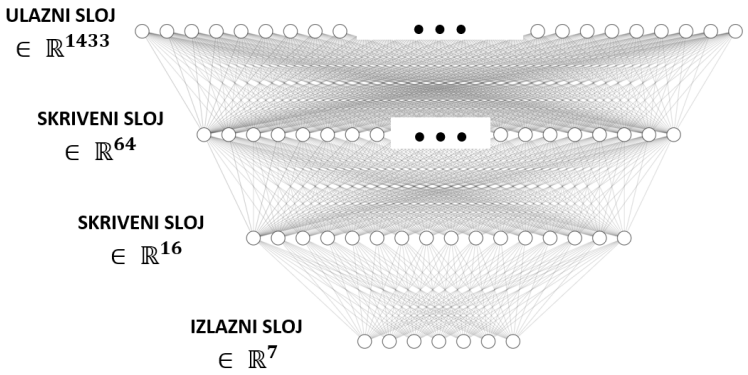
5.1 Treniranje i evaluacija osnovnog modela

Na temelju definiranih hiperparametara, kreirali smo osnovni model kao na Slici 18. Kako je ulaz u "multi-hot" obliku, svaki broj u vektoru je cijeli broj (*eng. integer*) kojega pretvaramo u broj sa pomičnim zarezom (*eng. float*) pomoću Lambda sloja. Nakon ulaza slijedi skriveni sloj s 64 neurona koji koristi *ReLu* aktivacijsku funkciju nakon kojega se dodaje *Dropout* sloj za regularizaciju. Analogno, slijedi skriveni sloj s 16 neurona, *ReLu* aktivacijskom funkcijom i *Dropout* slojem. Konačni izlaz mreže ima 7 neurona koji imaju *Softmax* aktivacijsku funkciju.

Osnovni model trenirali smo ga na trening podacim te smo testne podatke koristili za rano zaustavljanje (*eng. Early Stopping*). To je tehnika kojom se pokušava spriječiti prenaučenost tako što se prati točnost modela na testim podacima. Samo treniranje modela se prekida kada točnost na testnim podacima ne počne rasti (uz mogućnost čekanja nekoliko epoha). Prilikom prekida treniranja, model se vraća na stanje u kojemu je postigao najbolje rezultate.



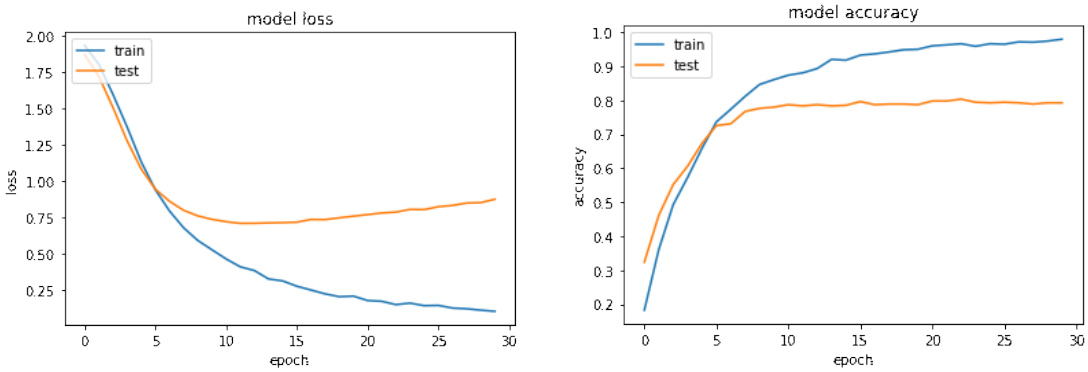
(a)



(b)

Slika 18: Arhitektura osnovnog modela

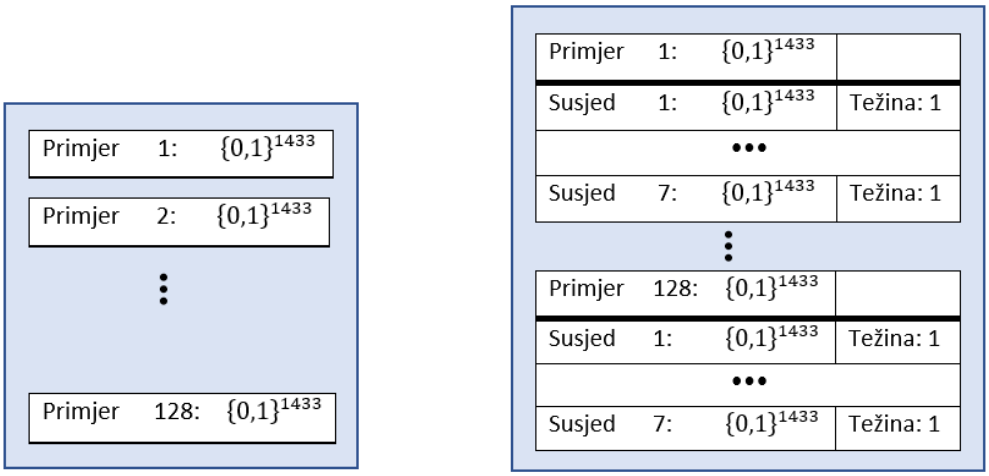
Nakon završetka treniranja, model je postigao preciznost od 77.94% na testnom skupu. Na Slici 19 vidimo promjenu preciznosti i funkcije troška na trening i test skupu.



(a) Trošak (b) Preciznost

Slika 19: Trošak i preciznost prilikom treniranja osnovnog modela

5.2 Treniranje i evaluacija graf-regulariziranog modela



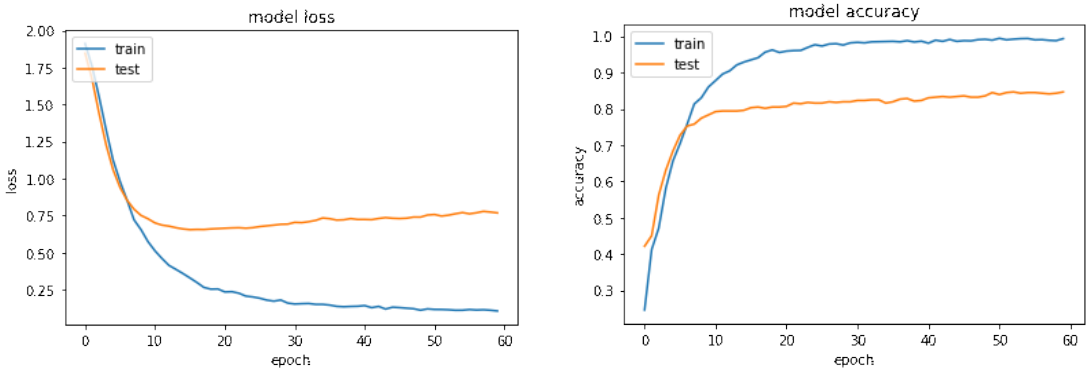
(a) Batch osnovnog modela (b) Batch graf-augmentiranog modela

Slika 20: Batch u osnovnom modelu i graf-augmentiranom modelu

Za potrebe treniranja augmentirali smo podatke, što znači da naš ulaz neće biti samo riječi koje su u znanstvenom radu i odgovarajuća oznaka, već će biti uključene

značajke susjeda. Broj susjeda smo definirali u *hparams.num_neighbors*. Kako smo ranije pokazali, postoje radovi koji imaju mali broj susjeda, pa će oni vrhovi koji imaju manje od *hparams.num_neighbors* susjeda, poprimiti 0 kao vrijednosti. Izgled ulaza prikazan je na Slici 20.

Nakon završetka treniranja model je postigao preciznost od 81.74% na testnom skupu. Na Slici 21 vidimo promjenu preciznosti i funkcije troška na trening i test skupu. To nam daje **poboljšanje od 3.8%** u odnosu na osnovni model. Međutim ta brojka može varirati zbog nasumičnosti inicijalizacije neuronske mreže i nasumičnosti odabira testnog skupa. Za ovaj konkretan primjer poboljšanje je obično između 1% i 4%. U praksi, takvo poboljšanje može činiti značajnu razliku u iskoristivosti modela.



(a) Trošak (b) Preciznost

Slika 21: Trošak i preciznost prilikom treniranja graf-regulariziranog modela

Literatura

[1] T. D. Bui, S. Ravi, V. Ramavajjala, *Neural Graph Learning: Training Neural Networks Using Graphs*, Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18), Association for Computing Machinery, New York, NY, USA, 64–71

[2] R. Diestel, *Graph theory*, Springer, Berlin, 2017

[3] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press, 2016

[4] T. Likhomanenko, Q. Xu, J. Kahn, G. Synnaeve, R Collobert, *slimIPL: Language-Model-Free Iterative Pseudo-Labeling*, arXiv preprint arXiv:2010.11524, 2020

[5] N. Truhar, *Numerička linearna algebra*, Sveučilište J.J. Strossmayera, Odjel za matematiku, Osijek, 2010, http://www.mathos.unios.hr/nla/NLA.pdf

[6] *Always start with a stupid model, no exceptions.*, https://blog.insightdatascience.com/always-start-with-a-stupid-model-no-exceptions-3a22314b9aaa

[7] *Gradient Descent: All You Need to Know*, https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

[8] *Graph Convolutional Network (GCN) on the CORA citation dataset*, https://stellargraph.readthedocs.io/en/v1.0.0rc1/demos/node-classification/gcn/gcn-cora-node-classification-example.html

[9] *Graph regularization for document classification using natural graphs*, https://www.tensorflow.org/neural_structured_learning/tutorials/graph_keras_mlp_cora

[10] *Od živčane stanice do živčanoga sustava*, https://shorturl.at/dwxO2

[11] *The Cora dataset*, https://graphsandnetworks.com/the-cora-dataset/

[12] *The Neural Structured Learning Framework*, https://www.tensorflow.org/neural_structured_learning/framework

[13] *Understanding Semi-supervised Learning*, https://jrodthoughts.medium.com/understanding-semi-supervised-learning-a6437c070c87

Sažetak

U današnje vrijeme, upotreba modela temeljenih na neuronskih mreža postala je sve češća, jer nerijetko postižu bolje rezultate od klasičnih metoda. Kako bi modeli bili primijenivi na stvarnim podacima, potrebno je spriječiti prenaučenost nad podacima za treniranje. To se postiže regularizacijom neuronskih mreža. Jedna od tehnika je i graf-regularizacija o kojoj smo govorili u ovom radu.

U prvom dijelu rada, objasnili smo osnove strojnog učenja i osnove teorije grafova, dok smo u drugom dijelu definirali graf-regularizaciju i primjenili ju na konkretnom primjeru.

Ključne riječi: strojno učenje, neuronske mreže, graf-regularizacija, graf

Summary

Nowadays, the use of models based on neural networks has become more common, as they often achieve better results than classical methods. In order for the models to be applicable to real data, it is necessary to prevent over-fitting over training data. This is achieved by regularizing neural networks. One of the techniques is graph-regularization, which we talked about in this paper.

In the first part of the paper, we explained the basics of machine learning and the basics of graph theory, while in the second part we defined graph-regularization and applied it to a concrete example.

Keywords: machine learning, neural networks, graph-regularization, graph

Životopis

Rođen sam 8. veljače 1996. godine u Čakovcu. Obrazovanje sam započeo u Osnovnoj školi Prelog, a nakon toga upisao sam Opću gimnaziju u Prelogu, gdje u trećem razredu ostvarujem pravo na sudjelovanje na državnom natjecanju iz matematike. Potom, 2014. godine upisujem Preddiplomski studij matematike na Odjelu za matematiku Sveučilišta J.J. Stossmayera u Osijeku. Akademski naziv prvostupnika matematike stječem 2018. godine s temom završnog rada *Uvod u neuronske mreže* pod mentorstvom izv.prof.dr.sc. Domagoja Matijevića. U jesen 2018. godine upisujem diplomski studij matematike, smjer Matematika i računarstvo. Tijekom diplomskog studija bavio sam se praktičnim primjenama strojnog učenja što je rezultiralo mojim zaposlenjem od lipnja 2019. u kompaniji TalentLyft (AdoptoTech d.o.o), gdje radim kao podatkovni znanstvenik (Data Scientist).