Primjena dubokog učenja u obradi zvuka

Vitez, Petra

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Petra Vitez

PRIMJENA DUBOKOG UČENJA U OBRADI ZVUKA

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, rujan, 2018.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem obitelji i prijateljima na podršci tijekom studiranja, te mentoru izv. prof. dr. sc. Saši Singeru na pomoći pri izradi diplomskog rada.

Sadržaj

Uvod

U početcima razvoja, polje umjetne inteligencije je rješavalo probleme koji su bili intelektualno zahtjevni za čovjeka, no prilično jednostavni za računalo — probleme koji su se mogli opisati skupom matematičkih pravila. No, pravi izazov u polju umjetne inteligencije i strojnog učenja je rješavanje problema koje ljudi lako rješavaju, međutim, teško formalno opišu način na koji rješavaju problem. Radi se o stvarima koje ljudima dolaze prirodno i intuitivno, i baziraju se na ljudskom dugoročnom pamćenju, kao što je prepoznavanje izgovorenih riječi ili prepoznavanje što se nalazi na slikama.

Duboko učenje je moderna grana strojnog učenja koja se bavi takvim problemima i koristi modele tzv. dubokih neuronskih mreža — skup međusobno povezanih jednostavnih elemenata koje nazivamo neuroni, po uzoru na ljudski mozak, s većim brojem tzv. skrivenih slojeva.

Ovaj rad je podijeljen u tri djela. U prvom djelu dajemo osnovni uvid u neuronske mreže. Objašnjavamo pojam umjetnog neurona i vrste umjetnih neurona te objašnjavamo na koji način se informacije iz prethodnih slojeva prenose u sljedeće slojeve. Objasnit ćemo pojam učenja neuronske mreže i algoritme učenja. U drugom djelu rada naglasak je na posebnoj vrsti neuronskih mreža pod nazivom rekurentne neuronske mreže. Takve mreže vjerno prikazuju rad ljudskog mozga i dobro rješavaju probleme, poput obrade zvuka i prepoznavanja govora. U trećem djelu dajemo implementaciju programskog rješenja. Cilj programa je što točnije klasificirati zvukove koji se često mogu čuti u gradskim sredinama, poput bušenja i trube u autu. Pritom koristimo biblioteku otvorenog koda za strojno učenje *Tensorflow*.

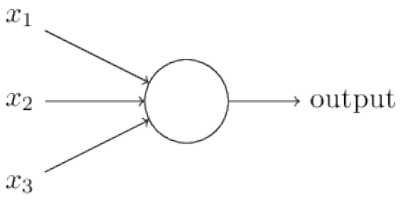
Poglavlje 1

Uvod u neuronske mreže

1.1 Perceptroni

Umjetne neuronske mreže su razvijene kao matematički modeli koji bi trebali obrađivati informacije na način sličan onome u ljudskom mozgu. Osnovna struktura umjetnih neuronskih mreža je mreža malih jedinica za obrađivanje, točnije, čvorova koji su povezani vezama s nekom težinom. Po uzoru na ljudski mozak, čvorovi tako predstavljaju neurone, dok veze sa svojim težinama predstavljaju jačinu sinapsi između neurona. Mreža je aktivirana kada neki ili svi čvorovi primaju neki ulazni podatak, i ta aktivacija se širi cijelom mrežom.

Objasnit ćemo sada pojam umjetnog neurona pod nazivom *perceptron*. Perceptroni su razvijeni 1950-ih i 1960-ih godina od strane znanstvenika Franka Rosenblatta, a funkcioniraju na način da, kao ulaz, primaju nekoliko binarnih varijabli x_1, x_2, ..., te daju binarni izlaz, što možemo vidjeti na slici 1.1



Slika 1.1: Perceptron

Rosenblatt je predložio jednostavno pravilo kako bi se izračunao izlaz perceptrona. Uveo je tzv. *težine* u perceptronu, realne brojeve w_1, w_2, ..., koji predstavljaju "važnost"

pojedinih ulaza za izlaz. Rad perceptrona možemo zapisati na način:

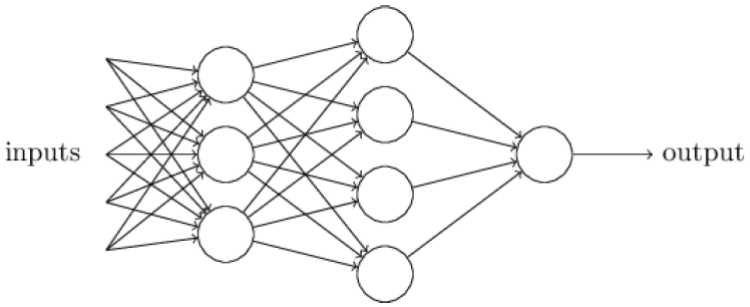
$$izlaz = \begin{cases} 0, & \text{za} \quad \sum_j w_j x_j \leq \theta, \\ 1, & \text{za} \quad \sum_j w_j x_j > \theta, \end{cases} \tag{1.1}$$

gdje vrijednost $\theta \in \mathbb{R}$ nazivamo *granična vrijednost* (eng. *threshold value*).

Pogledajmo sada primjer, kako bi bilo jasnije na koji način perceptron odlučuje. Neka, primjerice, odlučujemo o odlasku u restoran na temelju tri faktora:

1. Je li hrana u restoranu po našem ukusu?
2. Jesu li cijene prihvatljive?
3. Je li restoran u blizini?

Ta tri faktora možemo reprezentirati odgovarajućim binarnim varijablama x_1, x_2, x_3. Tako je, primjerice, $x_1 = 1$ ako vrijedi 1. faktor i $x_1 = 0$ ako ne vrijedi, i slično za ostale varijable. Obzirom da perceptrone koristimo za modeliranje problema odlučivanja, još jedan način da to radimo je preko težina. Tako bi npr. zadavanje težine w_2 kao najveće, značilo da nam je 2. faktor od presudne važnosti prilikom odabira. Na kraju, biramo i graničnu vrijednost θ. Pretpostavimo da su težine $w_1 = 2$, $w_2 = 6$, $w_3 = 2$ i $\theta = 2$. Ako uvrstimo te brojeve u (1.1), vidimo da perceptron modelira problem odlučivanja na način da je izlaz jednak 0 kada 2. faktor ne vrijedi i 1, kada vrijedi. Ako bismo odabrali graničnu vrijednost $\theta = 3$, tada bi perceptron kao izlaz imao 1 kada vrijedi 2. faktor ili ako istovremeno vrijede 1. faktor i 3. faktor.



Slika 1.2: Mreža perceptrona

Na slici 1.2 prikazana je mreža perceptrona. Prvi stupac perceptrona nazivat ćemo *prvim slojem* perceptrona. Svaki od perceptrona u nekom sloju radi odluku i daje izlaz na temelju rezultata prethodnog sloja. Na slici 1.2 vidimo kako je izlaz jednog perceptrona

korišten kao ulaz u nekoliko drugih perceptrona. Tako svaki sljedeći sloj radi sve kompleksnije odluke i na apstraktnijoj razini nego prethodni sloj. Ulazni podaci se ne promatraju kao perceptroni, već kao posebne jedinice koje su jednostavno definirane na način da kao izlaz daju varijable $x_1, x_2, \ldots$.

Što se tiče granične vrijednosti θ, praksa je promatrati vrijednost $b = -\theta$, koja se naziva *pristranost* perceptrona (eng. *bias*). Sada rad perceptrona možemo zapisati kao:

$$izlaz = \begin{cases} 0, & \text{za} \quad w \cdot x + b \leq 0, \\ 1, & \text{za} \quad w \cdot x + b > 0, \end{cases}$$

gdje je $w \cdot x$ skalarni produkt vektora w i x. Intuitivno, na pristranost perceptrona možemo gledati kao na mjeru koja govori koliko je lako dobiti 1 kao izlaz perceptrona. Ako je pristranost jako velika, vrlo je lako dobiti 1 kao izlaz perceptrona, i obratno, ako je vrlo mala, tada će biti teško dobiti 1 kao izlaz perceptrona.

1.2 Sigmoidni neuroni

Pretpostavimo da imamo mrežu perceptrona koju želimo naučiti da rješava neki problem odlučivanja, sličan prošlom primjeru. Jedan od standardnih problema koji rješavaju neuronske mreže, i koji će nam nekad služiti kao ogledni primjer u ovom radu, je problem klasifikacije znamenaka. Tako, u praksi, neuronske mreže služe u klasifikaciji slika rukom pisanih znamenki, ili pak zvučnih zapisa izgovorenih znamenaka. Želimo da neuronska mreža mijenja težine i pristranosti, tako da je izlaz neuronske mreže ispravno klasificirana znamenka. Pretpostavimo da napravimo neku malu promjenu u težini ili pristranosti. Ono što želimo je da ta mala promjena u težini ili pristranosti uzrokuje jednako malu promjenu u izlazu mreže. Npr., ako mreža krivo klasificira znamenku 9 kao znamenku 8, mogli bismo mijenjanjem težina i pristranosti dobivati sve bolji i bolji izlaz, i tada bismo rekli da mreža uči. Problem je što se to ne događa ako koristimo perceptrone. Mala promjena bi mogla uzrokovati da se izlaz perceptrona kompletno promijeni, npr. s 0 u 1. Ta promjena bi mogla dalje utjecati na ponašanje mreže na način da su ostale znamenke potpuno krivo klasificirane, što je onda teško dalje kontrolirati.

Zbog ovakvih razloga uvodi se pojam drugačijeg umjetnog neurona, zvanog *sigmoidni neuron*. Oni su slični perceptronima, ali modificirani na način da male promjene u težini ili pristranosti daju i male promjene u izlazu tih neurona. Sigmoidni neuron, jednako kao i perceptron, prima ulaze $x_1, x_2, \ldots$, ali, umjesto binarnih ulaza, on može primati bilo koji realni broj između 0 i 1. Jednako kao i perceptron, ima težine $w_1, w_2, \ldots$ i pristranost b. Međutim, izlaz sigmoidnog neurona nije 0 ili 1. Umjesto toga, izlaz je $\sigma(w \cdot x + b)$, gdje je $\sigma : [0, 1] \to (0, 1)$ *sigmoidna funkcija*:

$$\sigma(z) = \frac{1}{1 + e^z}$$

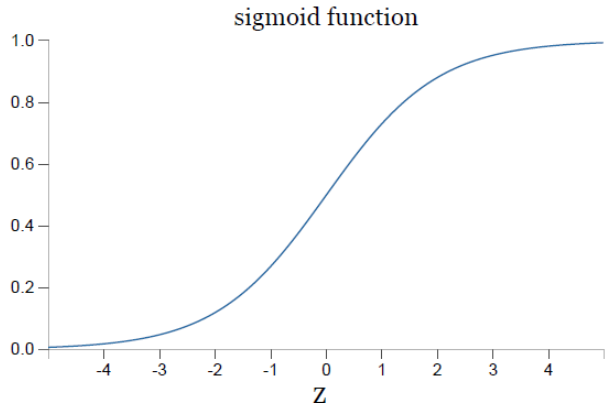
i σ nazivamo *funkcijom aktivacije*. Preciznije, ako sigmoidni neuron ima ulaz x_1, x_2, ..., težine w_1, w_2, ... i pristranost b, tada je izlaz:

$$\sigma(z) = \frac{1}{1 + \exp(-\sum_j w_j x_j - b)}.$$

Sigmoidna funkcija se ponekad naziva logističkom funkcijom, i u skladu s time spominjemo logističke neurone.

Kako bismo razumjeli sličnost s perceptronima, pretpostavimo da za $z = w \cdot x + b$ vrijedi $z \to +\infty$. Tada je $e^{-z} \approx 0$ i $\sigma(z) \approx 1$. Drugim riječima, kada je z veliki pozitivan broj, izlaz sigmoidnog neurona je približno 1, jednako kao što bi bilo u slučaju perceptrona. Pretpostavimo sada da $z \to -\infty$. Tada $e^{-z} \to \infty$ i $\sigma(z) \approx 0$, i ponovno se aproksimira vrijednost perceptrona.

Na slici 1.3 vidimo oblik sigmoidne funkcije. Ona je zaglađenje tzv. step funkcije $step : \mathbb{R} \to \mathbb{R}$, prikazane na slici 1.4. Kada bi funkcija aktivacije bila step funkcija, tada bismo umjesto sigmoidnog neurona imali perceptron.

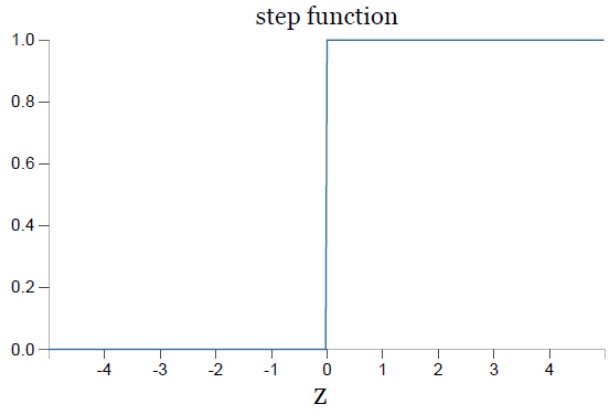


Slika 1.3: Sigmoidna funkcija [8]

Ako imamo problem neke binarne klasifikacije, dakle želimo odrediti pripada li ulazni podatak prvoj ili drugoj od dvije klase c_1 i c_2, izlaz $\sigma(z)$, za $z = w \cdot x + b$, možemo interpretirati kao vjerojatnost da ulazni podatak pripada prvoj ili drugoj klasi:

$$p(c_1|x) = y = \sigma(z),$$
$$p(c_2|x) = 1 - y.$$

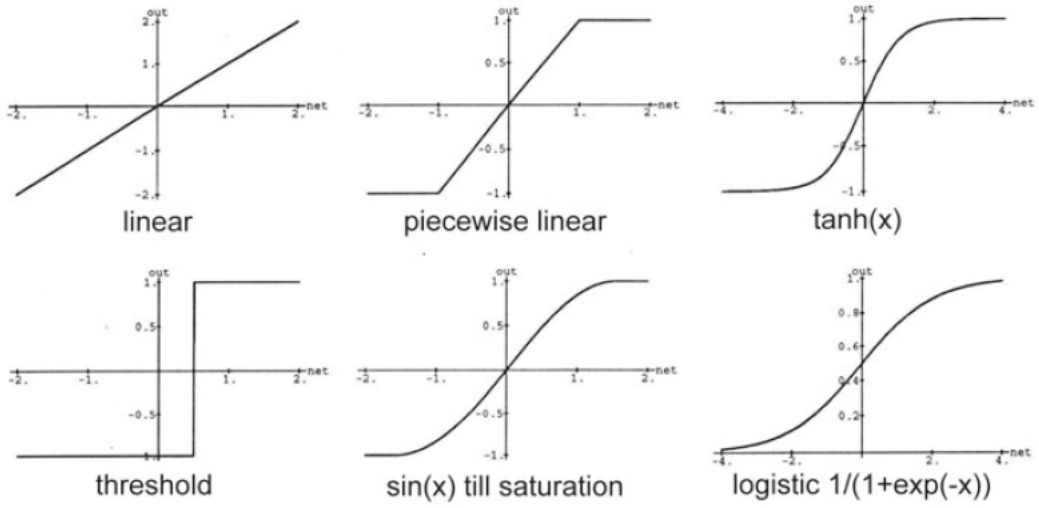
Primjerice, ako u praksi želimo da izlaz odgovara informaciji koja nam govori "ova znamenka je 9" ili "ova znamenka nije 9", jednostavnije bi bilo da je izlaz 0 ili 1, kao kod



Slika 1.4: Step funkcija

perceptrona. Međutim, to se može riješiti odlukom da interpretiramo izlaz na način da nam svaki izlaz manji od 0.5 ukazuje da znamenka nije 9, dok svaki veći od 0.5 ukazuje da znamenka je 9.

1.3 Funkcije aktivacije



Slika 1.5: Funkcije aktivacije za neuronsku mrežu [8]

Na slici 1.5 su prikazane moguće funkcije aktivacije za neuronske mreže. Najčešći odabir su već spomenuta sigmoidna funkcija i funkcija tangens hiperbolni:

$$\text{th}(x) = \frac{e^{2x} - 1}{e^{2x} + 1}.$$

Te dvije funkcije su povezane sljedećom linearnom transformacijom:

$$\text{th}(x) = 2\sigma(2x) - 1.$$

To znači da svaka funkcija koja je izračunata neuronskom mrežom s funkcijom aktivacije th, može biti izračunata drugom mrežom sa sigmoidnom funkcijom aktivacije. Zato smatramo da su te dvije funkcije ekvivalentne kao funkcije aktivacije.

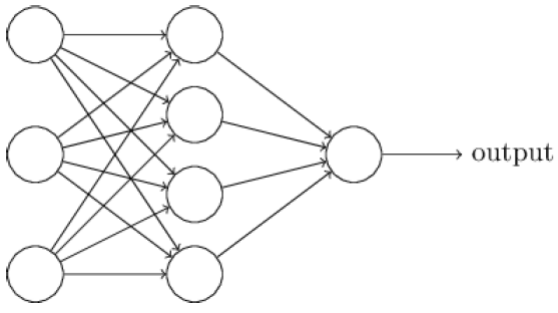
Važno svojstvo ovih dviju funkcija aktivacije je da su obje diferencijabilne

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)),$$
$$\text{th}'(z) = 1 - \text{th}^2(z),$$

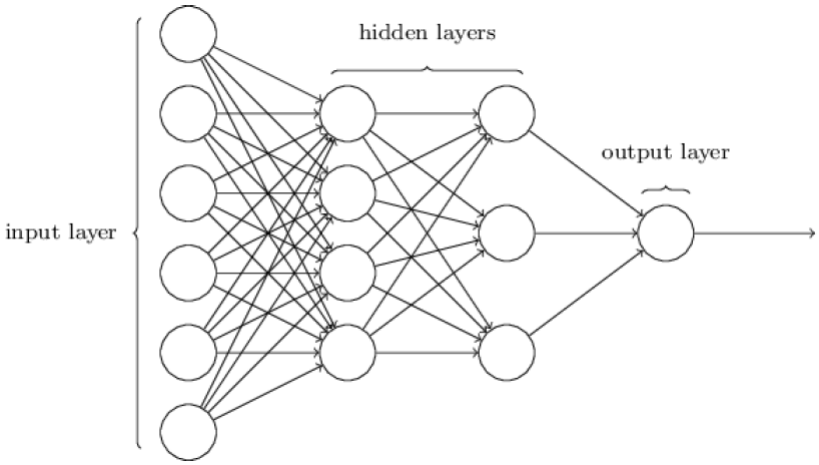
a kasnije ćemo vidjeti zašto nam je to od važnosti.

1.4 Arhitektura neuronske mreže

Objasnimo sada ukratko terminologiju neuronskih mreža. Neka je mreža kao na slici 1.6. Najljeviji sloj se naziva *ulazni sloj* s ulaznim neuronima, dok se najdesniji naziva *izlazni sloj* s izlaznim neuronima. Srednji sloj se naziva *skriveni sloj*. Pojam skriveni sloj samo znači da se ne radi ni o ulaznom ni o izlaznom sloju. Mreža na slici 1.6 ima samo jedan skriveni sloj, no neuronske mreže mogu imati i više skrivenih slojeva, tzv. *duboke neuronske mreže*, kao na slici 1.7.



Slika 1.6: Neuronska mreža [8]



Slika 1.7: Duboka neuronska mreža [8]

Do sada smo spominjali neuronske mreže gdje je izlaz jednog sloja korišten kao ulaz u drugi sloj neurona. Takve mreže nazivaju se *mreže bez povratnih veza*. U slučaju takvih mreža, nema situacija u kojoj ulaz funkcije aktivacije ovisi o izlazu, kao kod rekurentnih neuronskih mreža, što ćemo vidjeti kasnije. Sada možemo dati formalnu definiciju mreže bez povratnih veza.

Definicija 1.4.1. *Mreža bez povratnih veza se sastoji od sedmorke* $F = (N, \to, w, b, f, I, O)$, *gdje je* N *konačan skup čvorova (neurona), a* $(N, \to)$ *aciklički graf s bridovima u* $N \times N$. *Pišemo* $i \to j$ *ako je* i*-ti neuron povezan s* j*-tim u ovom grafu. Svaka veza* $i \to j$ *ima težinu* $w_{ij} \in \mathbb{R}$. *Neuroni bez prethodnika se nazivaju ulaznim neuronima i tvore skup* I. *Svi ostali neuroni se nazivaju neuronima za izračunavanje. Neprazni podskup neurona za izračunavanje se naziva skupom izlaznih neurona, i označavamo ga s* O. *Svi neuroni za izračunavanje koji nisu izlazni neuroni nazivaju se skrivenim neuronima. Svaki neuron za izračunavanje* i *ima pristranost* $b_i \in \mathbb{R}$ *i funkciju aktivacije* $f_i : \mathbb{R} \to \mathbb{R}$. *Bez smanjenja općenitosti, pretpostavljamo* $N \subset \mathbb{N}$ *te da su ulazni neuroni* $1, \ldots, m$.

Mreža s m *ulaza i* n *izlaza izračunava funkciju* $g : \mathbb{R}^m \to \mathbb{R}^n$,

$$g(x_1, \ldots, x_m) = (y_{i_1}, \ldots, y_{i_n}),$$

gdje su $i_1, \ldots, i_n$ *izlazni neuroni i* y_i *je definirano za svaki neuron* i *s*

$$y_i = \begin{cases} x_i & \text{ako je } i \text{ ulazni neuron}, \\ f_i(\sum_{i \to j} w_{ji} y_j + b_i), & \text{inače}. \end{cases}$$

Izraz $\sum_{i \to j} w_{ji} y_j + b_i$ *je aktivacija* i*-tog neurona.*

1.5 Nadzirano učenje i klasifikacija uzoraka

Zadatak nadziranog učenja sastoji se od *skupa za treniranje* S, od parova (x, z), gdje je x element ulaznog prostora $\mathcal{X}$, a z element ciljnog prostora $\mathcal{Z}$, te *testnog skupa* S' disjunktnog sa S. Uglavnom ćemo elemente skupa S nazivati *primjercima treniranja*.

Cilj je koristiti skup za treniranje za minimiziranje neke mjere greške E, definirane na testnom skupu za određeni zadatak. Za neuronske mreže je česti pristup prilikom rješavanja minimizacije greške, postupno prilagođavanje parametara algoritma učenja (koje ćemo objasniti malo kasnije), da bi se optimizirala tzv. *funkcija gubitka* (eng. *cost function*) na skupu za treniranje, koja je usko povezana s E.

Klasifikacija uzoraka, ili prepoznavanje uzoraka, je jedan od najviše istraživanih problema strojnog učenja. Klasifikator uzoraka $h : \mathcal{X} \to \mathcal{Z}$ je funkcija, čija domena je skup vektora, a kodomena skup oznaka. Mjera pogreške za h je tzv. *učestalost pogreške za klasifikaciju* $E^{class}(h, S')$ na testnom skupu S':

$$E^{class}(h, S') = \frac{1}{|S'|} \sum_{(x,z)\in S'} \begin{cases} 0, & \text{za} \quad h(x) = z, \\ 1, & \text{za} \quad h(x) \neq z, \end{cases}$$

tj. koliko puta je ulaznom elementu pridjeljena kriva oznaka.

Klasifikatori koji kao izlaz imaju oznaku klase se ponekad nazivaju *diskriminativnim funkcijama*. Oni direktno računaju vjerojatnosti $p(c_k|x)$, gdje je c_k k-ta klasa. U nekim slučajevima je poželjno prvo računati uvjetne vjerojatnosti $p(x|c_k)$ i zatim iskoristiti Bayesovo pravilo:

$$p(c_k, x) = \frac{p(x|c_k)p(c_k)}{p(x)},$$

gdje je

$$p(x) = \sum_k p(x|c_k)p(c_k).$$

Drugi pristup su tzv. *vjerojatnosne klasifikacije*, gdje su prvo određene uvjetne vjerojatnosti $p(c_k|x)$ od k klasa, za dani ulazni uzorak x, i zatim je najvjerojatnija izabrana kao izlaz klasifikatora: $h(x) = \arg\max_k p(c_k|x)$.

1.6 Gradijentni spust

Na koji način možemo učiti neuronsku mrežu da rješava probleme, npr. ispravno klasificira objekte? Prvo što trebamo je skup podataka na kojem će neuronska mreža učiti — skup podataka za treniranje. Uzmimo primjer ispravnog klasificiranja znamenaka. Tada

bi skup podataka za treniranje sadržavao slike rukom pisanih znamenaka, ili pak zvučne zapise, ako se radi o problemu ispravnog klasificiranja izgovorenih znamenki. Kada bismo testirali našu mrežu, tražili bismo da ispravno klasificira objekte koji nisu u našem skupu za treniranje, već u skupu podataka za testiranje. Ovo je u skladu s intuicijom, jer i mozak radi na način da prepoznaje dosad neviđene objekte na temelju prije viđenih.

Koristit ćemo notaciju x da označimo ulaz u neuronsku mrežu tijekom treniranja. Željeni izlaz označit ćemo s $y = y(x)$. U primjeru sa znamenkama, y bi bio 10-dimenzionalni vektor. Za dani ulaz x, gdje se radi o znamenci 6, vektor $y(x) = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)^\intercal$ je željeni izlaz. Želimo naći algoritam koji će nam naći težine i pristranosti, tako da izlaz neuronske mreže aproksimira $y(x)$ za sve ulaze x tijekom treniranja. Da prikažemo koliko dobro postižemo ovaj cilj, definiramo funkciju gubitka C:

$$C(w, b) \equiv \frac{1}{n} \sum_x \|y(x) - a(x)\|^2. \tag{1.2}$$

Ovdje w predstavlja sve težine u mreži, b sve pristranosti, n je ukupan broj ulaza tijekom treniranja, a je vektor koji predstavlja izlaz neuronske mreže za ulaz x.

Funkcija gubitka definirana u (1.2) je poznata i kao *srednja kvadratna pogreška* (eng. *mean squared error – MSE*). Primjećujemo da je ona nenegativna i teži nuli kada je $y(x)$ približno jednak izlazu $a(x)$, za sve ulaze x tijekom treniranja. Možemo sada zaključiti kako je naš algoritam dobar ako nađe težine i pristranosti tako da je funkcija gubitka približno jednaka nuli. Cilj treniranja naše neuronske mreže je minimizirati funkciju gubitka $C(w, b)$, pronalaženjem težina i pristranosti. U tu svrhu koristimo algoritam poznat kao *gradijentni spust* (eng. *gradient descent*).

Pretpostavimo sada da želimo minimizirati neku funkciju od m varijabli $C(v_1, \ldots, v_m)$. Ono što želimo naći je gdje C postiže globalni minimum. Za pomak ΔC vrijedi:

$$\Delta C = \nabla C \cdot \Delta v. \tag{1.3}$$

Na koji način sada biramo pomak Δv tako da je ΔC negativan? Pretpostavimo da biramo:

$$\Delta v = -\eta \nabla C, \tag{1.4}$$

gdje je $\eta > 0$ parametar koji se naziva *stopa učenja* (eng. *learning rate*).

Kombinirajući jednadžbe (1.3) i (1.4) vidimo da vrijedi

$$\Delta C = -\eta \|\nabla C\|^2. \tag{1.5}$$

Jer je $\|\nabla C\|^2 \geq 0$, iz (1.5) vidimo da je $\Delta C \leq 0$, pa će C uvijek padati ako mijenjamo v u skladu s (1.4). Zato ćemo ponavljati sljedeće pravilo:

$$v \to v' = v - \eta \nabla C.$$

Ovo pravilo zapravo definira algoritam gradijentnog spusta. Tako u više navrata mijenjamo v, u svrhu traženja minimuma funkcije C. Ideja primjene gradijentnog spusta za učenje neuronskih mreža leži u tome da koristimo gradijentni spust za traženje težina w_k i pristranosti b_l koje minimiziraju funkciju gubitka. Izrazimo ponovno pravilo gradijentnog spusta u terminima w_k i b_l:

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k},$$
$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

Ovo pravilo se primjenjuje u više navrata, u nadi da ćemo naći minimum funkcije C.

Funkcija gubitka ima oblik $C = \frac{1}{n}\sum_x C_x$, tj. ona je prosjek svih funkcija gubitaka za svaki primjerak treniranja x. Zato gradijent ∇C računamo kao:

$$\nabla C = \frac{1}{n}\sum_x \nabla C_x.$$

No, problem nastaje kada je broj ulaza za treniranje vrlo velik, tada učenje neuronske mreže u praksi može trajati vrlo dugo. U tu svrhu koristi se tzv. *stohastički gradijentni spust*, koji može ubrzati učenje. Ideja je procijeniti gradijent ∇C, računajući ∇C_x za mali skup ulaznih podataka za treniranje. Uzimajući prosjek po tom manjem skupu možemo dobiti dobru aproksimaciju ∇C, i ova metoda ubrzava algoritam gradijentnog spusta te tako i učenje neuronske mreže. Stohastički gradijentni spust funkcionira na način da se nasumično biraju male serije podataka za treniranje $X_1, \ldots, X_m$ (eng. *mini-batch*). Ako je veličina tog manjeg skupa m dovoljno velika, možemo očekivati sljedeće:

$$\frac{\sum_{j=1}^m \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C,$$

tj.

$$\nabla C \approx \frac{1}{m}\sum_{j=1}^m \nabla C_{X_j}.$$

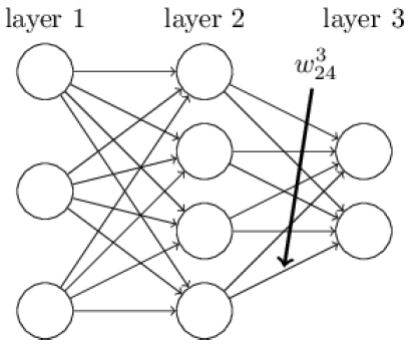
Sada, u terminima težina i pristranosti, imamo:

$$w_k \to w_k' = w_k - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial w_k},$$
$$b_l \to b_l' = b_l - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial b_l}.$$

Nakon toga se nasumično bira novi manji skup za treniranje i ponavljamo pravilo. Kada smo iscrpili ulazne podatke za treniranje, kažemo da smo završili jednu *epohu* treniranja, i u tom trenutku započinjemo novu.

1.7 Algoritam s propagiranjem greške unatrag

Objasnili smo kako neuronske mreže mijenjaju težine i pristranosti koristeći algoritam gradijentnog spusta, međutim, nismo diskutirali kako računati gradijent funkcije gubitka. U ovom poglavlju objašnjavamo algoritam za računanje tog gradijenta, tzv. *algoritam s propagiranjem greške unatrag* (eng. *backpropagation algorithm*). Koristit ćemo notaciju w^l_{jk} da označimo težinu za vezu k-tog neurona u $(l-1)$-om sloju, s j-tim neuronom u l-tom sloju. Na sljedećoj slici 1.8 označena je veza četvrtog neurona u drugom sloju s drugim neuronom u trećem sloju:



Slika 1.8: Veza četvrtog neurona u drugom sloju s drugim neuronom u trećem sloju [8]

Slično, s b^l_j označavamo pristranost j-tog neurona u l-tom sloju, i s a^l_j aktivaciju j-tog neurona u l-tom sloju. Koristeći ovu notaciju, sljedećom jednadžbom je prikazano kako je aktivacija a^l_j povezana s aktivacijom u prethodnom, $(l-1)$-om sloju:

$$a^l_j = \sigma\Big(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\Big). \tag{1.6}$$

Jednaždbu (1.6) možemo vektorski zapisati kao

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

Veličinu $z^l = w^l \cdot a^{l-1} + b^l$ nazivamo *težinski ulaz* neurona u sloju l.

Cilj algoritma s propagiranjem greške unatrag je izračunati parcijalne derivacije $\frac{\partial C}{\partial w}$ i $\frac{\partial C}{\partial b}$ funkcije gubitka C za svaku težinu w i pristranost b u mreži. Za algoritam s propagiranjem greške unatrag potrebne su nam dvije pretpostavke o funkciji gubitka. Prva pretpostavka je da se funkcija gubitka C može zapisati kao $C = \frac{1}{n}\sum_x C_x$ za svaki primjerak treniranja x. To je slučaj za kvadratnu funkciju gubitka, kod koje je gubitak za pojedini primjerak treniranja $C_x = \frac{1}{2}\|y - a^L\|^2$, gdje je L broj slojeva, a a^L oznaka za aktivaciju u posljednjem

sloju. Razlog zašto nam treba ova pretpostavka je zbog toga što algoritam s propagiranjem greške unatrag omogućava računanje parcijalnih derivacija $\frac{\partial C}{\partial w}$ i $\frac{\partial C}{\partial b}$ za pojedini primjerak treniranja. S ovime na umu, možemo pretpostaviti da je primjerak treniranja x fiksiran pa možemo pisati C, umjesto C_x. Druga pretpostavka o funkciji gubitka je da se ona može zapisati kao funkcija izlaza neuronske mreže. Kvadratna funkcija gubitka zadovoljava taj zahtjev, jer se kvadratna funkcija za pojedini primjerak treniranja x može zapisati kao

$$C = \frac{1}{2}\|y - a^L\|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

i tako je zapisana kao funkcija aktivacija.

Algoritam s propagiranjem greške unatrag kroz mrežu služi za računanje parcijalnih derivacija $\frac{\partial C}{\partial w_{jk}^l}$ i $\frac{\partial C}{\partial b_j^l}$. No, za računanje tih vrijednosti, prvo uvodimo jednu prijelaznu vrijednost δ_j^l, koju nazivamo *greška* u j-tom neuronu l-tog sloja. Algoritam s propagiranjem greške unatrag će nam dati grešku δ_j^l i zatim povezati grešku s $\frac{\partial C}{\partial w_{jk}^l}$ i $\frac{\partial C}{\partial b_j^l}$. Da bismo shvatili kako je greška δ_j^l definirana, pretpostavimo da za ulaz u j-ti neuron u l-tom sloju, umjesto izlaza $\sigma(z_j^l)$, neuron daje izlaz $\sigma(z_j^l + \Delta z_j^l)$. Ova promjena se širi kroz ostale slojeve u mreži, uzrokujući da se cjelokupni gubitak promijeni za $\frac{\partial C}{\partial z_j^l}\Delta z_j^l$. Pretpostavimo da $\frac{\partial C}{\partial z_j^l}$ ima veliku vrijednost (bilo pozitivnu ili negativnu). Tada se gubitak može smanjiti ako je Δz_j^l suprotnog predznaka od onog od $\frac{\partial C}{\partial z_j^l}$. Ako je pak, $\frac{\partial C}{\partial z_j^l}$ približno jednako nuli, tada se gubitak ne može previše popraviti mijenjanjem težinskog ulaza z_j^l. Intuitivno sada vidimo da ima smisla govoriti o $\frac{\partial C}{\partial z_j^l}$ kao o mjeri greške u neuronu, pa zato definiramo

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}.$$

Algoritam s propagiranjem greške unatrag je baziran na četiri fundamentalne jednadžbe koje daju način za računanje greške δ^l i gradijenta funkcije gubitka:

1. Jednadžba za grešku u izlaznom sloju

$$\delta_j^L = \frac{\partial C}{\partial a_j^L}\,\sigma'(z_j^L). \tag{1.7}$$

Jednadžbu možemo zapisati kao

$$\delta^L = \nabla_a C \odot \sigma'(z^L),$$

gdje $\odot$ označava Hadamardov (ili točkovni) produkt matrica.

2. Jednadžba za grešku δ^l, u terminima greške u sljedećem sloju δ^{l+1}, je

$$\delta^l = ((w^{l+1})^\intercal \delta^{l+1}) \odot \sigma'(z^l). \tag{1.8}$$

   Kombinirajući ovu jednaždbu s (1.7), možemo izračunati grešku δ^l za svaki sloj u mreži, propagirajući unatrag, tj. smanjujući l.

3. Jednadžba

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j,$$

   koju možemo zapisati kao

$$\frac{\partial C}{\partial b} = \delta.$$

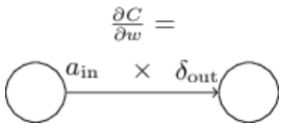
4. Jednadžba

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j,$$

   koju možemo zapisati kao

$$\frac{\partial C}{\partial w} = a_{in}\delta_{out},$$

   gdje je a_{in} aktivacija ulaznog neurona, a δ_{out} je greška izlaznog neurona, povezanih vezom s težinom w, kao što je prikazano na slici 1.9.



Slika 1.9: [8]

   Ako je aktivacija $a_{in} \approx 0$, tada je i $\frac{\partial C}{\partial w} \approx 0$, i u tom slučaju kažemo da *težina sporo uči*.

Pogledajmo sada jednadžbu (1.7). U slučaju kada vrijedi $\sigma(z^L_j) \approx 0$ ili $\sigma(z^L_j) \approx 1$, tada je $\sigma'(z^L_j) \approx 0$. Tada težina u posljednjem sloju sporo uči i kažemo da je izlazni neuron *zasićen*. Ako pogledamo jednadžbu (1.8), vidimo da je veća vjerojatnost da greška δ^l_j bude mala ako je neuron blizu zasićenja.

Sada ćemo dati algoritam s propagiranjem greške unatrag kroz mrežu:

1. **Ulaz** x: Postavi odgovarajuću aktivaciju a^1 za prvi sloj.
2. **Propagiranje unaprijed**: Za svaki $l = 2, 3, \ldots, L$, izračunaj $z^l = w^l a^{l-1} + b^l$ i aktivacije $a^l = \sigma(z^l)$.
3. **Izlazna greška** δ^L: Izračunaj $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Propagiranje greške unatrag**: Za svaki $l = L-1, L-2, \ldots, 2$, izračunaj grešku $\delta^l = ((w^{l+1})^\intercal \delta^{l+1}) \odot \sigma'(z^l)$.
5. **Izlaz**: Gradijent funkcije gubitka je dan s $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ i $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

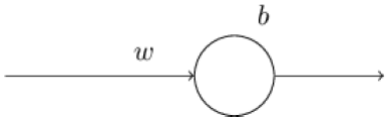
Algoritam s propagiranjem greške unatrag računa gradijent funkcije gubitka za jedan primjerak treniranja $C = C_x$. Česta je praksa kombinirati algoritam s propagiranjem greške unatrag s, npr., stohastičkim gradijentom spusta, u kojem računamo gradijent za mnoge primjerke treniranja. Za dani skup od m primjeraka treniranja, sljedeći algoritam primjenjuje algoritam gradijentnog spusta baziran na tom manjem skupu:

1. **Ulaz je skup primjeraka treniranja**.
2. **Za svaki primjerak treniranja** x: Postavi odgovarajuću aktivaciju $a^{x,1}$ i radi sljedeće korake:
   - **Propagiranje unaprijed**: Za svaki $l = 2, 3, \ldots, L$, izračunaj $z^{x,l} = w^{x,l} a^{x,l-1} + b^l$ i $a^{x,l} = \sigma(z^{x,l})$.
   - **Izlazna greška** $\delta^{x,L}$: Izračunaj $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.
   - **Propagiranje greške unatrag**: Za svaki $l = L-1, L-2, \ldots, 2$, izračunaj $\delta^{x,l} = ((w^{l+1})^\intercal \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.

Za neuronsku mrežu s W težina, računanje gradijenta izvršava se u vremenu $O(W^2)$, dok se propagiranje unatrag izvršava u vremenu $O(W)$.

1.8 Pogreška unakrsne entropije

U idealnom slučaju očekujemo da će neuronska mreža brzo učiti na pogreškama. No, je li to stvarno slučaj? Promotrimo jedan jednostavan primjer — neuron sa samo jednim ulazom, kao na slici 1.10. Trenirat ćemo neuron da nauči nešto vrlo jednostavno: za ulaz 1, kao izlaz ima 0.



Slika 1.10: Neuron sa samo jednim ulazom [8]

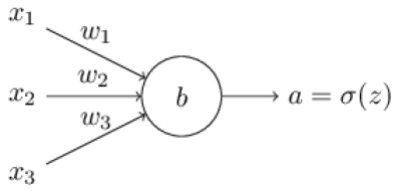
Kako bismo shvatili izvor problema, uzmimo da neuron uči mijenjajući težinu i pristranost, brzinom koja je određena parcijalnim derivacijama funkcije gubitka $\frac{\partial C}{\partial w}$ i $\frac{\partial C}{\partial b}$. Koristimo kvadratnu funkciju gubitka $C = \frac{(y-a)^2}{2}$, gdje je a izlaz neurona, $x = 1$ je ulaz, a $y = 0$ je željeni izlaz. Kako bismo ovo napisali u terminima težine i pristranosti, sjetimo se da je $a = \sigma(z)$, gdje je $z = wx + b$. Koristeći lančano pravilo, dobivamo

$$\frac{\partial C}{\partial w} = (a - y)\sigma'(z)x = a\sigma'(z), \tag{1.9}$$

$$\frac{\partial C}{\partial b} = (a - y)\sigma'(z) = a\sigma'(z), \tag{1.10}$$

za $x = 1$ i $y = 0$. Kada je izlaz neurona bliže 1, tj. vrijedi $\sigma(z) \approx 1$, tada je $\sigma'(z) \approx 0$, pa je $\frac{\partial C}{\partial w} \approx 0$ i $\frac{\partial C}{\partial b} \approx 0$, što je izvor usporavanja učenja. Isto se generalno događa za neuronske mreže, a ne samo za ovaj jednostavan primjer.

Kako sada pristupamo problemu usporavanju učenja? Problem se može riješiti zamjenom kvadratne funkcije gubitka, s funkcijom pod nazivom *pogreška unakrsne entropije* (eng. *cross entropy*). Pretpostavimo da treniramo neuron s ulaznim varijablama $x_1, x_2, \ldots$, odgovarajućim težinama $w_1, w_2, \ldots$, i pristranošću b, kao na slici 1.11.



Slika 1.11: Neuron [8]

Definiramo pogrešku unakrsne entropije ovog neurona kao funkciju gubitka

$$C = -\frac{1}{n}\sum_x [\, y \ln a + (1 - y)\ln(1 - a)\,],$$

gdje je n ukupan broj primjeraka treniranja, x su svi primjerci treniranja i y je željeni izlaz.

Pogledajmo u kojem smislu se pogreška unakrsne entropije može smatrati funkcijom gubitka. Prvo, možemo uočiti kako je funkcija nenegativna, tj. $C > 0$. Svi članovi sume su negativni, obzirom da su brojevi pod logaritmom između 0 i 1, te je predznak minus ispred sume. Drugo, ako je izlaz neurona blizu željenom izlazu, za sve primjerke treniranja x, tada je pogreška unakrsne entropije blizu nuli. Pretpostavimo da je $y = 0$ i $a \approx 0$ za neki ulaz x. Ako to uvrstimo u jednadžbu, vidimo da nam ostaje član $-\ln(1-a) \approx 0$. Slično je kada je $y = 1$ i $a \approx 1$. Dakle, ako je izlaz blizu željenom izlazu, tada je doprinos gubitku malen. Pogreška unakrsne entropije je pozitivna i bliže je nuli, kako neuron postaje bolji u izračunavanju željenog izlaza y, za sve primjerke treniranja x. No, za razliku od kvadratne funkcije gubitka, pogreška unakrsne entropije ima svojstvo da izbjegava problem usporavanja učenja. Izračunajmo parcijalne derivacije pogreške unakrsne entropije obzirom na težine i napravimo supstituciju $a = \sigma(z)$ te primijenimo lančano pravilo:

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right)\frac{\partial \sigma}{\partial w_j},$$

$$\frac{\partial C}{\partial w_j} = -\frac{1}{n}\sum_x \left(\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right)\sigma'(z)x_j.$$

Na kraju dobivamo:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x \frac{\sigma'(z)x_j}{\sigma(z)(1-\sigma(z))}(\sigma(z)-y).$$

Koristeći definiciju funkcije σ, $\sigma(z) = \frac{1}{1+e^{-z}}$, lako se pokaže $\sigma'(z) = \sigma(z)(1-\sigma(z))$. Sada dobivamo:

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j\,(\sigma(z)-y).$$

Ova jednadžba nam govori kojom brzinom je učenje težina kontrolirano sa $\sigma(z) - y$, tj. greškom u izlazu. Što je greška veća, to će neuron brže učiti. Također, ne moramo više brinuti za $\sigma'(z)$ i da će ta vrijednost biti mala, kao u slučaju kvadratne funkcije gubitka. Slično, u slučaju pristranosti dobivamo

$$\frac{\partial C}{\partial b} = \frac{1}{n}\sum_x (\sigma(z)-y).$$

Do sada smo govorili o pogrešci unakrsne entropije za samo jedan neuron, no možemo govoriti o njoj i u slučaju višeslojnih neuronskih mreža. Pretpostavimo da su y_1, y_2, ... željeni izlazi neurona, dok su a_1, a_2, ... izlazne vrijednosti. Tada definiramo pogrešku unakrsne entropije kao

$$C = -\frac{1}{n}\sum_x \sum_j [\, y_j \ln a_j^L + (1-y_j)\ln(1-a_j^L)\,].$$

Analogno kao i prije, lako bi se pokazalo da, koristeći C, izbjegavamo usporavanje učenja u slučaju višeslojnih neuronskih mreža.

Pogledajmo sada što je uopće moglo motivirati pogrešku unakrsne entropije. Pretpostavimo da smo došli do usporavanja učenja i da je $\sigma'(z)$ izvor usporavanja, kao u (1.9) i (1.10). Zanima nas je li moguće izabrati funkciju gubitka tako da $\sigma'(z)$ iščezne. U tom slučaju bi funkcija gubitka $C = C_x$, za jedan primjerak treniranja x, zadovoljavala

$$\frac{\partial C}{\partial w_j} = x_j(a - y),$$

$$\frac{\partial C}{\partial b} = (a - y). \tag{1.11}$$

Koristeći lančano pravilo imamo

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial a}\,\sigma'(z) \tag{1.12}$$

i, koristeći $\sigma'(z) = \sigma(z)(1 - \sigma(z)) = a(1 - a)$, jednadžba (1.12) postaje

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial a}\,a(1 - a). \tag{1.13}$$

Koristeći (1.11) i (1.13), dobivamo

$$\frac{\partial C}{\partial a} = \frac{a - y}{a(1 - a)}.$$

Ako integriramo posljednji izraz, dobivamo

$$C = -\frac{1}{n}\sum_x [\, y \ln a + (1 - y)\ln(1 - a)\,] + c,$$

gdje je c neka konstanta.

1.9 Softmax

Još jedan pristup problemu usporavanja učenja poznat je pod nazivom *softmax* sloj neurona. Ideja softmaxa je definirati novi tip izlaznog sloja neurona za neuronsku mrežu. U softmax sloju primjenjujemo tzv. *softmax* funkciju na $z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L$. Prema toj funkciji, za aktivaciju a_j^L j-tog izlaznog neurona vrijedi

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}.$$

U mnogim problemima pogodno je interpretirati izlaznu aktivaciju a_j^L kao procjenu neuronske mreže da je ispravan izlaz jednak j, tj. da ulaz pripada nekoj j-toj klasi c_j:

$$p(c_j|x) = a_j^L.$$

Ako, primjerice, imamo $K = 5$ klasa i ispravna klasa ulaza je c_2, tada je *ciljni vektor* $t = (0, 1, 0, 0, 0)$. Sada možemo pisati:

$$p(t|x) = \prod_{j=1}^{K} a_j^{t_j}.$$

Kako bismo razumjeli kako softmax slojevi rješavaju problem usporavanja učenja, definiramo *log-vjerojatnosnu funkciju gubitka* (eng. *log-likelihood*). Označimo ponovno x kao ulaz treniranja za neuronsku mrežu i y kao željezni izlaz. Tada je log-vjerojatnosna funkcija gubitka definirana kao

$$C = -\ln ay^L.$$

Ako je mreža "sigurna" da je ulaz u mrežu j, tada će procijenjena vrijednost za vjerojatnost a_j^L biti blizu 1, pa će gubitak $-\ln a_j^L$ biti malen. U suprotnom, vjerojatnost a_j^L će biti manja, a gubitak $-\ln a_j^L$ veći. Vidimo da se log-vjerojatnosna funkcija ponaša kao funkcija gubitka. Lako se pokaže da vrijedi

$$\frac{\partial C}{\partial b_j^L} = a_j^L - y_j,$$

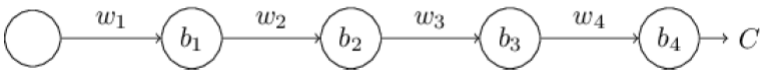
$$\frac{\partial C}{\partial w_{jk}^L} = a_k^{L-1}(a_j^L - y_j).$$

Ove jednadžbe su analogne onima u poglavlju o pogrešci unakrsne entropije (1.9) i (1.10), te, kao u prošlom poglavlju, ove jednadžbe osiguravaju da neće doći do problema usporavanja učenja. Korisno je razmišljati o softmax sloju s log-vjerojatnosnom funkcijom gubitka, kao o sigmoidnom izlaznom sloju s pogreškom unakrsne entropije kao funkcijom gubitka. Generalno, oba pristupa funkcioniraju dobro, no softmax sloj s log-vjerojatnosnom funkcijom je poželjno koristiti u problemima kada na izlazne aktivacije želimo gledati kao na vjerojatnosti, što može biti slučaj u primjerima gdje su klase disjunktne, kao što je npr. klasifikacija znamenaka.

1.10 Problem nestajućih gradijenata

Različiti slojevi u dubokim neuronskih mrežama uče različitim brzinama. Specijalno, kasniji slojevi uče dobro, dok prijašnji slojevi često zapinju tijekom treniranja. No, što je zapravo razlog tome? Je li to slučajnost ili kasniji skriveni slojevi uče brže od prijašnjih?

Problem kada neuroni u prijašnjim slojevima uče mnogo sporije nego neuroni u sljedećim slojevima, poznat je pod nazivom *problem nestajućih gradijenata*. No, zašto uopće dolazi do tog problema? Kako bismo dobili uvid u problem, uzmimo kao primjer duboku neuronsku mrežu u kojoj svaki skriveni sloj ima samo jedan neuron. Na slici 1.12 je primjer takve mreže s tri skrivena sloja, gdje su $w_1, \ldots, w_4$ težine, $b_1, \ldots, b_4$ pristranosti i C je neka funkcija gubitka.



Slika 1.12: Primjer duboke neuronske mreže [8]

Znamo da je izlaz a_j j-tog neurona jednak $\sigma(z_j)$, gdje je $z_j = w_j a_{j-1} + b_j$. Pogledajmo sada izraz $\frac{\partial C}{\partial b_1}$ za prvi skriveni neuron i pokažimo da vrijedi:

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)\, w_2 \sigma'(z_2)\, w_3 \sigma'(z_3)\, w_4 \sigma'(z_4)\, \frac{\partial C}{\partial a_4}.$$

Pretpostavimo da napravimo malu promjenu Δb_1 u pristranosti b_1. To će izazvati seriju promjena u ostatku neuronske mreže. Prvo će izazvati promjenu Δa_1 u izlazu prvog neurona, što će izazvati promjenu Δz_2 u težinskom ulazu drugog skrivenog neurona, a zatim promjenu Δa_2, itd., sve do promjene u funkciji gubitka ΔC:

$$\frac{\partial C}{\partial b_1} \approx \frac{\Delta C}{\Delta b_1}.$$

Ovo nam sugerira da možemo dobiti izraz za gradijent $\frac{\partial C}{\partial b_1}$, ako po redu pratimo efekt kojeg svaka od prije spomenutih promjena ima. Tako imamo

$$\Delta a_1 \approx \frac{\partial \sigma(w_1 a_0 + b_1)}{\partial b_1}\, \Delta b_1 = \sigma'(z_1)\, \Delta b_1. \tag{1.14}$$

Promjena Δa_1 uzrokuje promjenu u $z_2 = w_2 a_1 + b_2$ za drugi skriveni neuron

$$\Delta z_2 \approx \frac{\partial z_2}{\partial a_1}\, \Delta a_1 = w_2 \Delta a_1. \tag{1.15}$$

Kombinirajući jednadžbe (1.14) i (1.15), dobivamo

$$\Delta z_2 \approx \sigma'(z_1)\, w_2\, \Delta b_1.$$

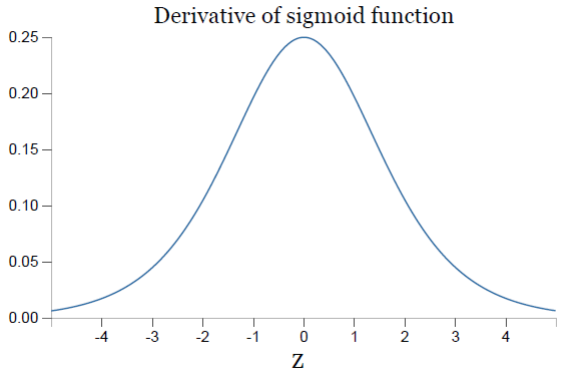
Nastavljajući tako, došli bismo do promjene u funkciji gubitka

$$\Delta C \approx \sigma'(z_1)\, w_2\sigma'(z_2)\, w_3\sigma'(z_3)\, w_4\sigma'(z_4)\, \frac{\partial C}{\partial a_4}\, \Delta b_1. \tag{1.16}$$

Podijelimo li izraz u (1.16) s Δb_1, dobivamo

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)\, w_2\sigma'(z_2)\, w_3\sigma'(z_3)\, w_4\sigma'(z_4)\, \frac{\partial C}{\partial a_4}. \tag{1.17}$$

Kako bismo razumjeli kako se svaki od faktora $w_j\sigma'(z_j)$ u izrazu (1.17) ponaša, pogledajmo sada oblik derivacije sigmoidne funkcije na slici 1.13.



Slika 1.13: Derivacija sigmoidne funkcije [8]

Vrijedi $\sigma'(z) = \frac{e^x}{(1+e^x)^2}$, a maksimum se postiže u 0 i jednak je $\sigma'(0) = \frac{1}{4}$. Standardni pristup kod inicijalizacije težina je prema normalnoj razdiobi s očekivanjem 0 i varijancom 1, tako da težine zadovoljavaju $|w_j| < 1$, pa će vrijediti $|w_j\sigma'(z_j)| < \frac{1}{4}$. Množenjem takvih izraza umnožak će se eksponencijalno smanjivati.

Usporedimo sada izraz za $\frac{\partial C}{\partial b_1}$ u (1.17), s npr. onim za $\frac{\partial C}{\partial b_3}$:

$$\frac{\partial C}{\partial b_3} = \sigma'(z_3)\, w_4\sigma'(z_4)\, \frac{\partial C}{\partial a_4}.$$

Vidimo da $\frac{\partial C}{\partial b_1}$ sadrži u produktu dva člana više, oblika $w_j\sigma'(z_j)$, za koje smo malo prije objasnili da vrijedi $|w_j\sigma'(z_j)| < \frac{1}{4}$, pa će tako izraz $\frac{\partial C}{\partial b_1}$ biti barem 16 puta manji od $\frac{\partial C}{\partial b_3}$. Ovo nam daje generalnu intuiciju zašto dolazi do problema nestajućih gradijenata, te zašto sljedeći slojevi uče bolje od prethodnih u neuronskoj mreži.

Ako se pak, težine odaberu tako da su velike, npr. $w_j = 100$, i ako biramo pristranosti tako da vrijedi $b_j = -100a_j - 1$, tada je $z_j = 0$ i $\sigma'(z_j) = \frac{1}{4}$, tako da su svi članovi $w_j\sigma'(z_j)$ u produktu jednaki 25. Gradijent bi na taj način rastao eksponencijalno, ako bismo išli unatrag kroz mrežu, i došli bismo do tzv. *problema eksplodirajućeg gradijenta*.

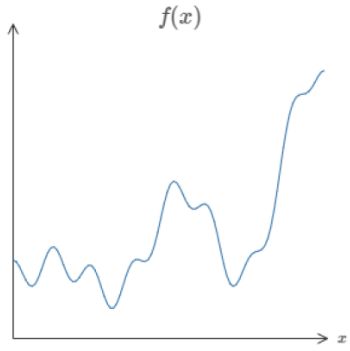
Generalni problem učenja neuronskih mreža je to što je gradijent u prvim slojevima umnožak članova u svim sljedećim slojevima, i kada je mnogo slojeva, dolazi do nestabilne situacije pa neuronska mreža pati od *problema nestabilnih gradijenata*. Kao ogledni primjer, dali smo neuronsku mrežu s jednim skrivenim neuronom u skrivenom sloju, no isti problem se javlja i kod kompleksnijih neuronskih mreža s više neurona u skrivenim slojevima.

1.11 Univerzalnost neuronskih mreža

U radu Kurta Hornika, Maxwella Stinchombea i Halberta Whitea [5] iz 1989. godine, pokazano je kako standardne višeslojne mreže bez povratnih veza s barem jednim skrivenim neuronom, koristeći proizvoljne funkcije aktivacije, mogu aproksimirati svaku Borel izmjerivu funkciju s jednog konačnodimenzionalnog prostora u drugi, i zato se smatraju univerzalnim aproksimatorima.

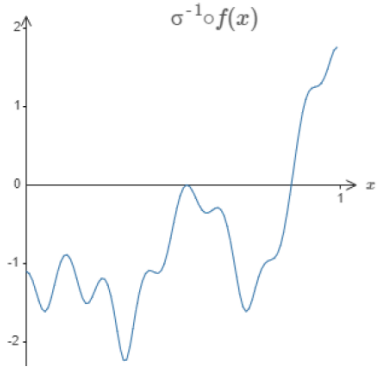
Rad ne govori da neuronska mreža može zbilja izračunavati svaku funkciju, ali dobivamo dovoljno dobru aproksimaciju, koju možemo poboljšavati dodavanjem skrivenih neurona. Pretpostavimo da imamo funkciju $f(x)$ koju želimo izračunati unutar neke željene točnosti $\epsilon \in \mathbb{R}$, $\epsilon > 0$. Korištenjem dovoljnog broja skrivenih neurona, možemo uvijek naći neuronsku mrežu čiji izlaz $g(x)$ zadovoljava $|f(x) - g(x)| < \epsilon$, za sve ulaze x.

Za ilustraciju, uzmimo sada problem konstruiranja neuronske mreže koja aproksimira funkciju f, kao na slici 1.14.



Slika 1.14: Primjer funkcije [8]

Ono što računa skriveni neuron je $\sigma(wx + b)$. Možemo li, znajući ovo, imati kontrolu nad izlazom neuronske mreže? Rješenje je konstruirati neuronsku mrežu čiji skriveni sloj ima težinski izlaz jednak $\sigma^{-1}f(x)$, gdje je σ^{-1} inverz funkcije aktivacije, tj. želimo da je težinski izlaz skrivenog sloja kao na slici 1.15. Ako to postignemo, tada je izlaz neuronske mreže dovoljno dobra aproksimacija za funkciju f.



Slika 1.15: Primjer težinskog izlaza skrivenog sloja [8]

1.12 Treniranje neuronske mreže

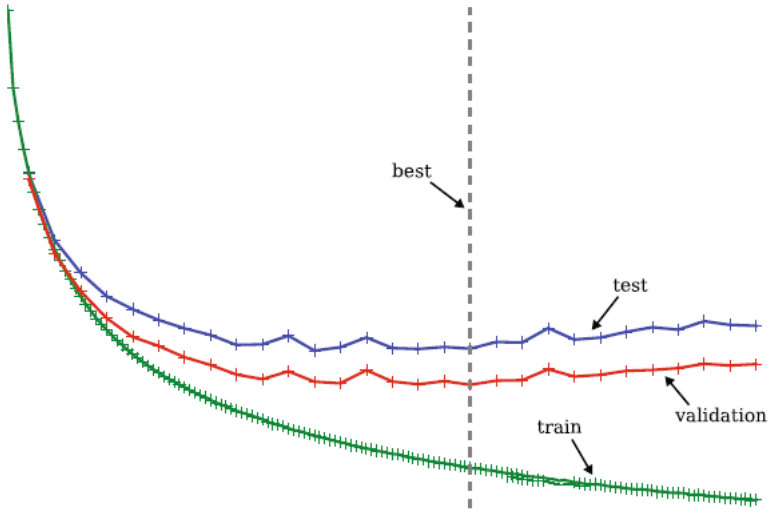
Zasada smo diskutirali kako mreža može učiti mijenjanjem težina i pristranosti, što nazivamo *treniranje mreže*, i algoritam gradijentnog spusta. No, kako bismo osigurali da je treniranje mreže efikasno i dovoljno brzo, te da se dobro ponaša u slučaju dosad neviđenih podataka, trebamo se osvrnuti na neke probleme i pristupe.

Generalizacija

Iako je funkcija gubitka definirana za skup podataka za treniranje, pravi cilj je optimizirati performanse na testnom skupu dosad neviđenih podataka. Problem koji se bavi pitanjem prenosi li se performansa sa skupa za treniranje na skup za testiranje, naziva se *generalizacija*. Uglavnom, vrijedi: čim je veći skup podataka za treniranje, tim je bolja generalizacija.

Ponekad se iz skupa podataka za treniranje izvlači tzv. *validacijski skup*. Kriteriji zaustavljanja treniranja se provjeravaju na validacijskom skupu, umjesto na testnom skupu. ”Najbolje” težine su isto odabrane iz validacijskog skupa; biraju se one koje minimiziraju grešku koja određuje performansu na testnom skupu. Greška se ocjenjuje na validacijskom skupu u pravilnim intervalima, i treniranje prestaje kada se greška prestaje smanjivati, nakon određenog broja ocjenjivanja.

Tijekom treniranja, greška u početku uglavnom pada na svim skupovima podataka. No, nakon određene točke, počne rasti na testnom skupu i validacijskom skupu, dok se, s druge strane, smanjuje na skupu za treniranje, što je prikazano na slici 1.16. Takvo ponašanje nazivamo *prenaučenost* (eng. *overfitting*). U tom slučaju, mreža će raditi dobro na skupu podataka za treniranje, ali neće uspjeti generalizirati za dosad neviđene podatke. Crtkana linija prikazuje najbolje performanse na validacijskom skupu.



Slika 1.16: Prenaučenost na skupu podataka za treniranje [3]

Reprezentacija ulaza

Biranje pogodne reprezentacije ulaznih podataka je ključni dio zadaća strojnog učenja. Kod neuronskih mreža, praksa je da komponente ulaznog vektora imaju očekivanje 0, a standardnu devijaciju 1, pa računamo:

$$m_i = \frac{1}{|S|} \sum_{x \in S} x_i$$

i standardnu devijaciju:

$$\sigma_i = \sqrt{\frac{1}{|S|} \sum_{x \in S} (x_i - m_i)^2},$$

a zatim se izračuna ulazni vektor $\hat{x}$, čije su komponente:

$$\hat{x}_i = \frac{x_i - m_i}{\sigma_i}.$$

Ovaj pristup ne utječe na informacije u skupu podataka za treniranje, ali poboljšava performanse, stavljajući ulazne vrijednosti u raspon pogodniji za klasične funkcije aktivacije.

Inicijalizacija težina

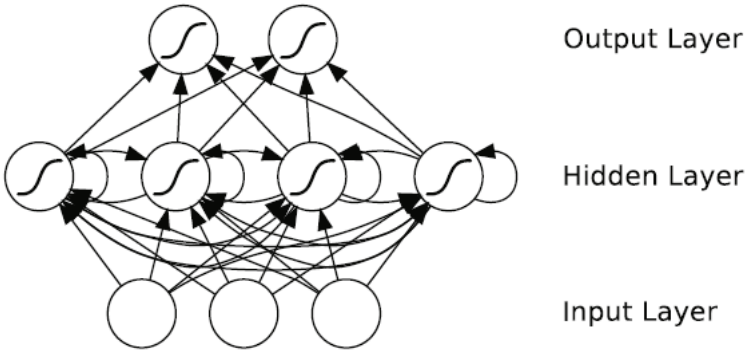
Algoritmi gradijentnog spusta za neuronske mreže često zahtijevaju male i nasumične inicijalne vrijednosti za težine. Zato su one često inicijalizirane prema normalnoj razdiobi s očekivanjem 0 i standardnom devijacijom 1.

Poglavlje 2

Rekurentne neuronske mreže

U prvom poglavlju spominjali smo višeslojne mreže bez povratnih veza. S druge strane, postoje modeli neuronskih mreža u kojima su povratne veze moguće. Takvi modeli nazivaju se *rekurentne neuronske mreže*. Nadalje ćemo kraće pisati *RNN mreža*. Glavna ideja takvih modela su neuroni koji su "aktivirani" u ograničenom vremenskom periodu, prije nego postanu "prigušeni". Ta aktivacija može stimulirati ostale neurone, koji se mogu aktivirati malo kasnije i biti aktivirani ograničeni vremenski period. Povratne veze ne predstavljaju problem u ovom modelu, jer izlaz neurona djeluje na ulaz nakon nekog vremena, a ne trenutačno.

Algoritmi učenja rekurentnih neuronskih mreža su generalno sporiji od onih za mreže bez povratnih veza. Međutim, one su od interesa, jer puno vjernije prikazuju kako ljudski mozak radi i stoga rješavaju probleme, poput obrade zvuka i prepoznavanja govora, puno bolje od mreža bez povratnih veza. Na slici 2.1 je prikazano kako izgleda rekurentna neuronska mreža sa sigmoidnom funkcijom aktivacije.



Slika 2.1: Rekurentna neuronska mreža [3]

2.1 Definicija RNN mreže

U slučaju RNN mreža, promatramo stabla, gdje svaki neprazni čvor ima najviše k sljedbenika, od kojih neki mogu biti prazno stablo. Stablo s oznakama u nekom skupu Σ, gdje je Σ neki konačan skup ili realan vektorski prostor, je ili prazno stablo, što označavamo s $\perp$, ili se sastoji od korijena, koji je označen nekom oznakom $l \in \Sigma$, i k podstabala $t_1, \ldots, t_k$, od kojih neka mogu biti prazna. U drugom slučaju, cijelo stablo označavamo s $l(t_1, \ldots, t_k)$. Skup tako definiranih stabala označit ćemo sa Σ_k^*.

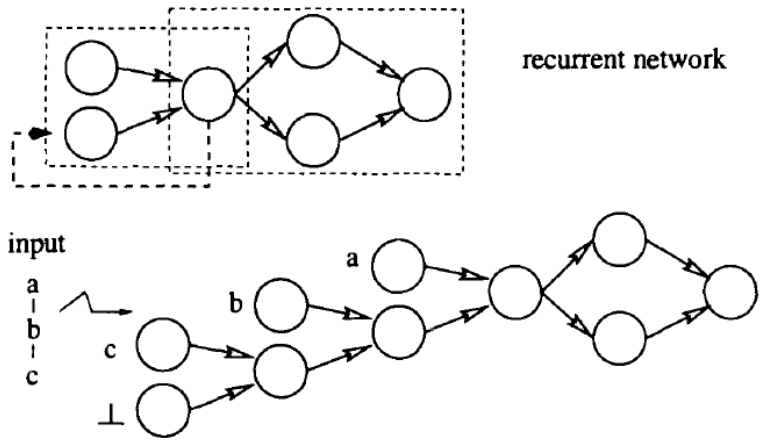
Definicija 2.1.1. *Neka je R neki skup. Svako preslikavanje $g : \Sigma \times R^k \to R$ i element $y \in R$ induciraju preslikavanje $\tilde{g}_y : \Sigma_k^* \to R$, koje je rekurzivno definirano na sljedeći način:*

$$\tilde{g}_y(\perp) = y,$$
$$\tilde{g}_y(l(t_1, \ldots, t_k)) = g(l, \tilde{g}_y(t_1), \ldots, \tilde{g}_y(t_k)).$$

Koristeći ovu definiciju (uz $\Sigma = \mathbb{R}^m$ i $R = \mathbb{R}^l$), možemo formalno definirati rekurentne neuronske mreže. One su posebni slučaj, za $k = 1$, tzv. preklopljenih ili zamotanih mreža (eng. *folded network*, v. [4]).

Definicija 2.1.2. *Rekurentna neuronska mreža se sastoji od dvije mreže bez povratnih veza koje, redom, izračunavaju $g : \mathbb{R}^{m+l} \to \mathbb{R}^l$ i $h : \mathbb{R}^l \to \mathbb{R}^n$, i tzv. početnog konteksta $y \in \mathbb{R}^l$. RNN mreža izračunava preslikavanje $h \circ \tilde{g}_y : (\mathbb{R}^m)_1^* \to \mathbb{R}^n$.*

Ulazni neuroni $m+1, \ldots, m+l$ funkcije g se nazivaju kontekstni neuroni, a g se naziva rekurzivnim dijelom mreže, dok je h dio mreže bez povratnih veza. Ulazni neuroni RNN mreže su neuroni $1, \ldots, m$ za g, a izlazni neuroni mreže su izlazni neuroni za h (slika 2.2).

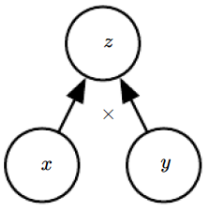


Slika 2.2: Rekurentna neuronska mreža [4]

2.2 Odmotani graf izračunavanja

Graf izračunavanja

Graf izračunavanja je način na koji možemo formalizirati strukturu nekog izračunavanja. Koristimo čvor u grafu kako bismo prikazali varijablu. Varijabla može biti skalar, vektor ili matrica. Uvodimo pojam operacije — funkcije jedne ili više varijabli. Bez smanjenja općenitosti, definiramo da operacija vraća samo jednu izlaznu varijablu. Ako je varijabla y izračunata primjenom operacije na varijablu x, tada u grafu izračunavanja povlačimo usmjereni brid od x do y, s imenom operacije u izlaznom čvoru, kao na slici 2.3.



Slika 2.3: Graf koristi operaciju × kako bi izračunao $z = xy$ [2]

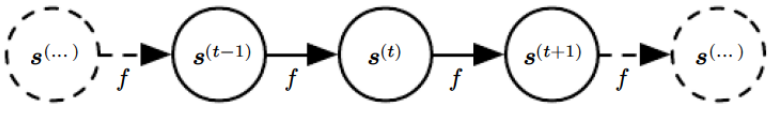
Pogledajmo sada klasični oblik jednadžbe dinamičkog sustava:

$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \theta), \tag{2.1}$$

gdje se $\boldsymbol{s}^{(t)}$ naziva stanje sustava u trenutku t. Za konačan broj vremenskih koraka τ, graf se može "odmotati" tako da primijenimo definiciju $\tau - 1$ puta. Ako jednadžbu (2.1) odmotamo unatrag $\tau = 3$ koraka, dobivamo

$$\boldsymbol{s}^{(3)} = f(\boldsymbol{s}^{(2)}; \theta) = f(f(\boldsymbol{s}^{(1)}; \theta); \theta).$$

Jednadžba (2.1) i njezino odmotavanje se može prikazati usmjerenim acikličkim grafom izračunavanja, kao na slici 2.4. Svaki čvor predstavlja stanje u nekom vremenu t i funkcija f preslikava stanje u koraku t u stanje u koraku $t + 1$.

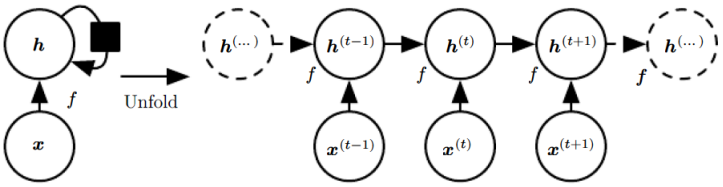


Slika 2.4: Sustav opisan jednadžbom (2.1), prikazan kao odmotani graf izračunavanja [2]

Mnoge rekurentne neuronske mreže koriste jednadžbu oblika (2.2) za definiranje vrijednosti u skrivenim neuronima

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \theta). \tag{2.2}$$

Na slici 2.5 vidimo rekurentnu neuronsku mrežu bez izlaza. Ova mreža samo obrađuje informaciju ulaza $\boldsymbol{x}$, tako da je pripoji stanju $\boldsymbol{h}$, koje se propagira unaprijed kroz vrijeme; crni kvadrat označava odgodu od jednog vremenskog koraka. Istu mrežu vidimo i kao odmotani graf izračunavanja, gdje je svaki čvor povezan s jednim vremenskim korakom. Odmotavanje je pridruživanje sklopa na lijevoj strani slike grafu izračunavanja na desnoj strani slike.



Slika 2.5: Odmotana rekurentna neuronska mreža [2]

Na slici 2.6 vidimo graf izračunavanja za računanje funkcije gubitka tijekom treniranja rekurentne neuronske mreže, koja preslikava ulazni niz $\boldsymbol{x}$ u niz izlaznih vrijednosti $\boldsymbol{o}$, iste duljine. Ovdje je funkcija gubitka označena s L, a ne s C (u dijelu literature se *cost function* spominje kao *loss function*). Funkcija gubitka L mjeri koliko je izlaz $\boldsymbol{o}$ "daleko" od željene vrijednosti $\boldsymbol{y}$. Veze između ulaza i skrivenih slojeva su parametrizirane matricom težina $\boldsymbol{U}$, veze između skrivenih slojeva matricom težina $\boldsymbol{W}$, a između skrivenih i izlaznih slojeva matricom težina $\boldsymbol{V}$.

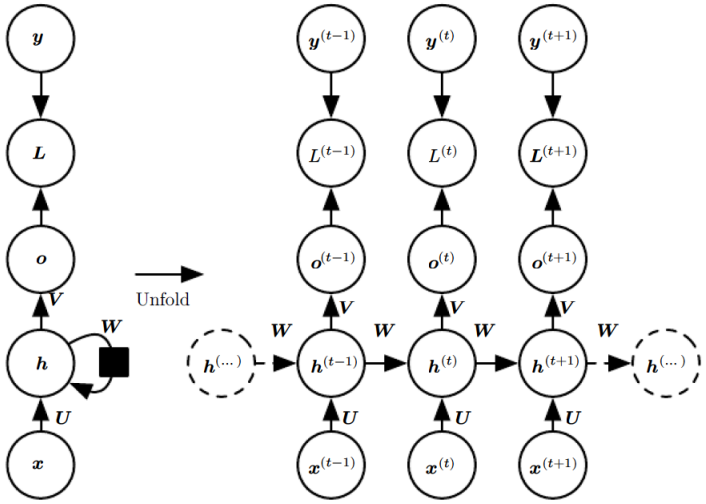
2.3 Prolaz unaprijed i prolaz unatrag

Prolaz unaprijed

Pretpostavimo da je niz $\boldsymbol{x}$, duljine T, ulaz u RNN mrežu, koja ima I ulaznih neurona, H skrivenih neurona i K izlaznih neurona. Neka je x_i^t vrijednost ulaza i u vremenu t i neka su z_j^t i a_j^t, redom, ulaz mreže u j-ti neuron u vremenu t i aktivacija j-tog neurona u vremenu t. Za skrivene neurone i aktivacije vrijedi:

$$z_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} a_{h'}^{t-1} + b, \tag{2.3}$$

$$a_h^t = \sigma(z_h^t). \tag{2.4}$$



Slika 2.6: Graf izračunavanja za računanje funkcije gubitka RNN [2]

Cijeli niz skrivenih aktivacija može se izračunati tako da počnemo od $t = 1$ i rekurzivno primjenjujemo (2.3) i (2.4), te povećavamo t u svakom koraku. Ovo zahtijeva da početne vrijednosti a_i^0 budu odabrane, što označava stanje mreže prije nego primi bilo kakve informacije. Obično se te vrijednosti inicijaliziraju na nulu.

Za izlazne neurone vrijedi:

$$z_k^t = \sum_{h=1}^{H} w_{hk} a_h^t + b'. \tag{2.5}$$

U (2.3) i (2.5), b i b' označavaju pristranosti, a za težine vrijedi $w_{ih} \in \boldsymbol{U}$, $w_{hh'} \in \boldsymbol{W}$ i $w_{hk} \in \boldsymbol{V}$.

Prolaz unatrag

U slučaju RNN mreža, funkcija gubitka C ovisi o aktivacijama skrivenih slojeva, ne samo preko njihovih utjecaja na izlazni sloj, nego i na skriveni sloj u sljedećem koraku:

$$\delta_h^t = \sigma'(z_h^t)\left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_h'^{t+1} w_{hh'}\right), \tag{2.6}$$

gdje je

$$\delta_j^t = \frac{\partial C}{\partial z_j^t}.$$

Cijeli niz δ članova se može izračunati tako da počnemo od $t = T$ i rekurzivno primjenjujemo (2.6), smanjujući t u svakom koraku. Primijetimo da je $\delta_j^{T+1} = 0$, za svaki j, jer nema greške nakon kraja niza.

Konačno, imajući na umu da su iste težine korištene u svakom koraku, sumiramo po cijelom nizu, kako bismo dobili:

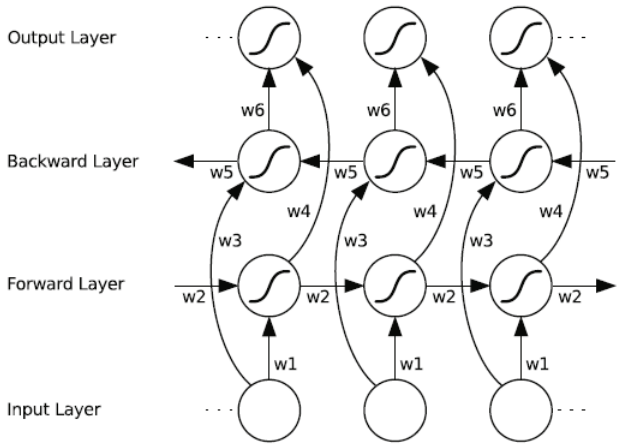
$$\frac{\partial C}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial C}{\partial z_j^t} \frac{\partial z_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t a_i^t. \tag{2.7}$$

Ovako opisani algoritam naziva se *propagiranje unatrag kroz vrijeme*.

2.4 Dvosmjerne mreže

Za mnoge zadatke označavanja nizova korisno je imati pristup budućem, kao i prošlom kontekstu. Primjerice, kada klasificiramo neko slovo u riječi, korisno je znati slova koja dolaze prije i koja dolaze poslije tog slova.

Dvosmjerne neuronske mreže predstavljaju svaku od sekvenci treniranja unaprijed i unatrag kao dva rekurentna skrivena sloja, gdje je svaki povezan s izlaznim slojem. "Odmotanu" dvosmjeru RNN mrežu možemo vidjeti na slici 2.7. Ovakva struktura osigurava izlaznom sloju kompletni prošli i budući kontekst, za svako mjesto u ulaznom nizu. Prolaz unaprijed u dvosmjernim mrežama za skrivene slojeve je isti kao u jednosmjernim, s razlikom da izlazni sloj nije ažuriran tako dugo, dok oba skrivena sloja nisu obradili cijeli ulazni niz.



Slika 2.7: "Odmotana" dvosmjerna rekurentna neuronska mreža [3]

Algoritam za prolaz unaprijed, u slučaju dvosmjernih RNN, ima oblik:

za $t = 1$ do T **radi**:
Prolaz unaprijed za prvi skriveni sloj, pohrani aktivacije u svakom koraku;

za $t = T$ do 1 **radi**:
Prolaz unaprijed za drugi skriveni sloj, pohrani aktivacije u svakom koraku;

za sve t, u bilo kojem redoslijedu **radi**:
Prolaz unaprijed za izlazni sloj, koristeći aktivacije od oba skrivena sloja.

Slično, algoritam za prolaz unatrag je:

za sve t, u bilo kojem redoslijedu **radi**:
Prolaz unatrag za izlazni sloj, pohrani δ u svakom koraku;

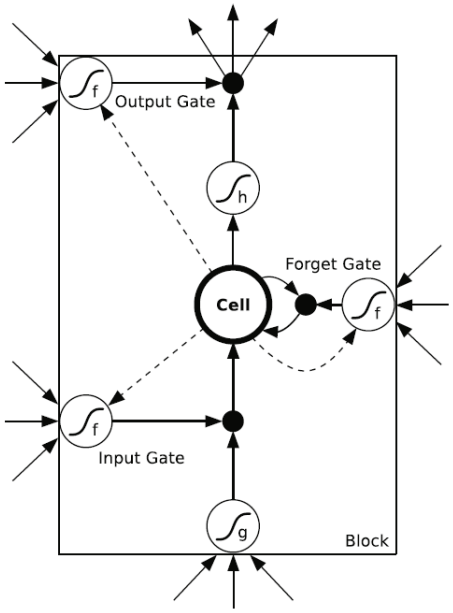
za $t = T$ do 1 **radi**:
Prolaz unatrag kroz vrijeme za prvi skriveni sloj, koristeći δ iz izlaznog sloja;

za $t = 1$ do T **radi**:
Prolaz unatrag kroz vrijeme za drugi skriveni sloj, koristeći δ iz izlaznog sloja.

2.5 Long Short-Term Memory

Važno svojstvo rekurentnih neuronskih mreža je korištenje konteksta. No, za standardne RNN mreže, raspon konteksta koji se u praksi može dohvatiti je prilično ograničen. Problem je već spomenuti *problem nestabilnih gradijenata*, obrađen u odjeljku 1.10. Utjecaj danog ulaza na skrivene slojeve, pa i na izlazni sloj, vodi do nestajanja ili eksplodiranja gradijenata. Jedan od pristupa koji rješava taj problem je tzv. *Long Short-Term Memory* (LSTM) arhitektura neuronske mreže.

LSTM arhitektura se sastoji povezanih podmreža koje se nazivaju *memorijskim blokovima*. Svaki blok se sastoji od jedne memorijske ćelije i tri tzv. *propusnice — ulazna propusnica*, *izlazna propusnica* i *propusnica zaboravljanja*. Ćelija i propusnice su neuroni. Na slici 2.8 vidimo LSTM memorijski blok s jednom ćelijom. Tri propusnice "skupljaju" aktivacije unutar i izvan bloka, i kontroliraju aktivaciju ćelije preko umnoška (crni krug). Ulazna i izlazna propusnica množe, redom, ulaze i izlaze ćelije, dok propusnica zaboravljanja služi za množenje s prijašnjim stanjem ćelije. Funkcija aktivacije f za propusnicu je, generalno, sigmoidna funkcija, tako da su aktivacije propusnica između 0 ("zatvorena propusnica") i 1 ("otvorena propusnica"). Funkcije aktivacije g i h, za ulaz i izlaz ćelije, su obično th ili sigmoidna funkcija, s time da je h nekad i identiteta. Veze od ćelije prema propusnicama su označene crtkanom vezom (eng. *peephole connections*) i one imaju težinu. Ostale veze u bloku nemaju težinu. Izlaz iz bloka u ostatak mreže proizlazi iz umnoška izlazne propusnice.

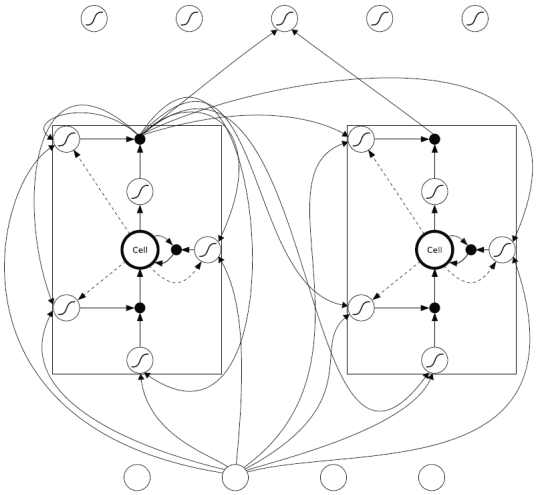


Slika 2.8: LSTM memorijski blok s jednom ćelijom [3]

Propusnice dopuštaju LSTM memorijskim blokovima da pohranjuju i pristupaju informacijama kroz duži vremenski period i smanjuju problem nestajućih gradijenata. Tako dugo dok je ulazna propusnica zatvorena (tj. ima aktivaciju blizu 0), aktivacija ćelije neće biti pregažena novim ulazom u mrežu i može biti dostupna mreži mnogo kasnije u nizu, otvaranjem izlazne propusnice. Uloga propusnice zaboravljanja je da se omogući ćeliji da se na neki način ”resetira”, tj. da ”zaboravi” prijašnje ulaze. Na slici 2.9 vidimo mrežu s četiri ulazna neurona, skriveni sloj s dva LSTM memorijska bloka i pet izlaznih neurona.

LSTM mreže se koriste u područjima učenja beskontekstnih jezika, predviđanja strukture proteina, prepoznavanje govora, prepoznavanje rukopisa, tj. rješavaju probleme koji zahtijevaju duži raspon konteksta.

U nastavku dajemo jednadžbe za aktivacije (prolaz unaprijed) i za propagiranje unatrag kroz vrijeme (prolaz unatrag) za LSTM skriveni sloj unutar rekurentne mreže. Jednadžbe će biti dane za jedan memorijski blok. Težina veze između i-tog i j-tog neurona je označena s w_{ij}, ulaz u j-ti neuron u koraku t je označen sa z_j^t i aktivacija j-tog neurona u koraku t je označena s a_j^t. Oznake ι, ϕ i ω će, redom, označavati ulaznu propusnicu, izlaznu propusnicu i propusnicu zaboravljanja bloka. S c ćemo označiti jednu od N memorijskih ćelija. Veze od ćelije c do ulazne propusnice, izlazne propusnice i propusnice zaboravljanja su, redom, označene s $w_{c\iota}$, $w_{c\phi}$, $w_{c\omega}$. Stanje ćelije c u vremenu t je s_c^t (aktivacija ćelije). f je funkcija aktivacije za propusnice, dok su g i h funkcije aktivacije za ulaz i izlaz ćelije.



Slika 2.9: LSTM mreža [3]

Neka je I broj ulaza, K je broj izlaza i H je broj ćelija u skrivenom sloju. Primjetimo da su samo izlazi ćelije a_c^t povezani s ostalim blokovima u skrivenom sloju. Ostale aktivacije, kao što su stanje, ulazi u ćeliju i aktivacije propusnica, su vidljive samo unutar bloka. Indeks h koristimo da označimo izlaze ćelije ostalih blokova u skrivenom sloju. Kao slučaju standardnih RNN mreža, prolaz unaprijed se računa za ulazni niz duljine T, počinje u $t = 1$ i rekurzivno se primijenjuju jednadžbe kako se povećava t, dok se pripagiranje unatrag kroz vrijeme računa počevši od $t = T$ i rekurzivno se primijenjuju jednadžbe kako se t smanjuje. Prisjetimo se da je $\delta_j = \frac{\partial C}{\partial z_j^t}$, gdje je C funkcija gubitka tijekom treniranja mreže.

Prolaz unaprijed

Ulazne propusnice:

$$z_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} a_h^{t-1} + \sum_{c=1}^{N} w_{c\iota} s_c^{t-1},$$

$$a_\iota^t = f(z_\iota^t).$$

Propusnice zaboravljanja:

$$z_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} a_h^{t-1} + \sum_{c=1}^{N} w_{c\phi} s_c^{t-1},$$

$$a_\phi^t = f(z_\phi^t).$$

Ćelije:

$$z_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} a_h^{t-1},$$
$$s_c^t = a_\phi^t s_c^{t-1} + a_\iota^t g(z_c^t).$$

Izlazne propusnice:

$$z_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} a_h^{t-1} + \sum_{c=1}^{N} w_{c\omega} s_c^{t-1},$$
$$a_\omega^t = f(z_\omega^t).$$

Izlaz ćelije:

$$a_c^t = a_\omega^t h(s_c^t).$$

Prolaz unatrag

Definiramo

$$\epsilon_c^t = \frac{\partial C}{\partial a_c^t} \quad \text{i} \quad \epsilon_s^t = \frac{\partial C}{\partial s_c^t}.$$

Izlaz ćelije:

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck} \delta_k^t + \sum_{h=1}^{H} w_{ch} \delta_h^{t+1}.$$

Izlazne propusnice:

$$\delta_\omega^t = f'(z_\omega^t) \sum_{c=1}^{N} h(s_c^t) \epsilon_c^t.$$

Stanja:

$$\epsilon_s^t = a_\omega^t h'(s_c^t) \epsilon_c^t + a_\phi^{t+1} \epsilon_s^{t+1} + w_{c\iota} \delta_\iota^{t+1} + w_{c\phi} \delta_\phi^{t+1} + w_{c\omega} \delta_\omega^t.$$

Ćelije:

$$\delta_c^t = a_\iota^t g'(z_c^t) \epsilon_s^t.$$

Propusnice zaboravljanja:

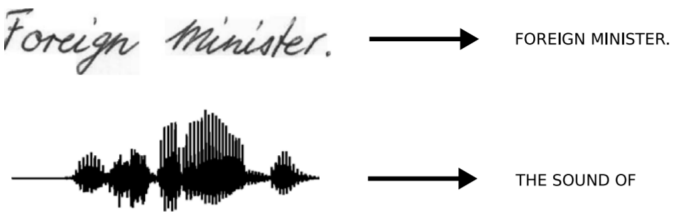
$$\delta_\phi^t = f'(z_\phi^t) \sum_{c=1}^{N} s_c^{t-1} \epsilon_s^t.$$

Ulazne propusnice:

$$\delta_\iota^t = f'(z_\iota^t) \sum_{c=1}^{N} g(z_c^t) \epsilon_s^t.$$

2.6 Označavanje nizova

U strojnom učenju, pojam *označavanje nizova* (eng. *sequence labelling*) obuhvaća sve zadaće gdje je niz podataka pretvoren u nizove diskretnih vrijednosti. Poznati primjeri uključuju prepoznavanje govora i rukom pisanih rečenica. U primjeru prepoznavanja govora, ulaz (govorni signal) je proizveden kontinuiranim kretanjem vokalnog trakta, dok su oznake (nizovi riječi) međusobno ograničene pravilima sintakse i gramatike. Cilj označavanja nizova je pridijeliti nizove oznaka, koje pripadaju fiksiranom alfabetu, nizovima ulaznih podataka. Na slici 2.10 je prikazano kako je za niz ulaznih podataka dan niz diskretnih oznaka.



Slika 2.10: Označavanje nizova [3]

Ako je pretpostavka da su nizovi nezavisni i jednako distribuirani, koristimo osnovni model klasifikacije uzoraka, opisan u odjeljku 1.5, s nizovima, umjesto uzoraka. U praksi, ovaj pristup možda nije potpuno opravdan (niz može predstavljati naizmjenični dijalog, ili linije rukom pisanog teksta); no, ovaj pristup zapravo odgovara, tako dugo dok su granice za nizove razumno odabrane. Zato nadalje pretpostavljamo da je svaki izlazni niz najviše jednako dug kao i odgovarajući ulazni niz. S tim restrikcijama možemo formalizirati zadatak označavanja nizova.

Neka je S skup podataka za treniranje. Ulazni prostor $\mathcal{X} = (\mathbb{R}^m)^*$ je skup svih nizova, ciljni prostor $\mathcal{Z} = \mathcal{L}^*$ je skup svih nizova nekog (konačnog) *alfabeta oznaka* $\mathcal{L}$. Elemente skupa $\mathcal{L}^*$ nazivamo *nizovima oznaka*. Svaki element skupa S je par nizova $(\boldsymbol{x}, \boldsymbol{z})$. Ciljni niz $\boldsymbol{z} = (z_1, z_2, \ldots, z_U)$ je najviše jednako dug kao ulazni niz $\boldsymbol{x} = (x_1, x_2, \ldots, x_T)$, tj. $|\boldsymbol{z}| \leq |\boldsymbol{x}|$. Zadatak je koristiti skup S za treniranje nekog algoritma $h : \mathcal{X} \to \mathcal{Z}$, za označavanje nizova u testnom skupu S', što je moguće točnije.

Klasifikacija nizova

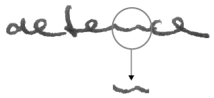
Najrestriktivniji slučaj je kada su nizovi oznaka ograničeni na duljinu jedan, što nazivamo *klasifikacijom nizova*. Tada je svaki ulazni niz pridijeljen jednoj klasi. Primjeri takvih problema uključuju prepoznavanje jedne izgovorene riječi ili rukom napisanog slova. Ako

su ulazni nizovi fiksirane veličine, mogu biti predstavljeni ulaznim vektorom i standardna klasifikacija uzoraka se može primijeniti. Očita mjera greške je postotak krivih klasifikacija E^{seq}, pod nazivom *učestalost pogrešaka nizova*:

$$E^{seq}(h, S') = \frac{100}{|S'|} \sum_{(\boldsymbol{x},z)\in S'} \begin{cases} 0, & \text{za} \quad h(\boldsymbol{x}) = \mathbf{z}, \\ 1, & \text{za} \quad h(\boldsymbol{x}) \neq \mathbf{z}. \end{cases}$$

Klasifikacija segmenata

Pod *klasifikacijom segmenata* mislimo na zadatke gdje se ciljni nizovi sastoje od više oznaka, dok se pozicije ulaznih segmenata, kojima su oznake pridijeljene, unaprijed znaju. Primjenjuje se u obradi prirodnog jezika i bioinformatici, gdje su ulazi diskretne vrijednosti pa se mogu trivijalno segmentirati. Važnost kod klasifikacije segmenata ima kontekst, s obje strane segmenta. Tako na slici 2.11 možemo lako vidjeti kako se radi o riječi "defence", no segment koji sadrži slovo "n" bi mogao, sam po sebi, biti višeznačan; nismo nužno sigurni radi li se slovu "n", slovu "u", ili pak o nekom drugom znaku.



Slika 2.11: Važnost konteksta u klasifikaciji segmenata [3]

To predstavlja problem kod standardne klasifikacije uzoraka, koja je dizajnirana na način da obradi jedan ulaz po vremenskoj jedinici. Jednostavno rješenje je skupiti podatke s obje strane segmenta u tzv. *vremenske prozore* i koristiti ih kao ulazne uzorke. No, ovaj pristup i dalje pati od problema da je opseg korisnog konteksta nepoznat, pa time i veličina vremenskih prozora. Mjera greške je postotak krivo klasificiranih segmenata E^{seg}, pod nazivom *učestalost pogrešaka segmenata*:

$$E^{seg}(h, S') = \frac{1}{Z} \sum_{(\boldsymbol{x},z)\in S'} HD(h(\boldsymbol{x}), z),$$

gdje je $Z = \sum_{(x,z)\in S'} |z|$, a $HD(\boldsymbol{p}, \boldsymbol{q})$ je *Hammingova udaljenost* nizova $\boldsymbol{p}$ i $\boldsymbol{q}$, jednake duljine, i jednaka je broju pozicija na kojima su odgovarajući simboli različiti.

Temporalna klasifikacija

U slučaju *temporalne klasifikacije*, ništa se ne može pretpostaviti o označavanju nizova, osim da je njihova duljina manja ili jednaka onoj ulaznih nizova. Mogu čak biti prazni.

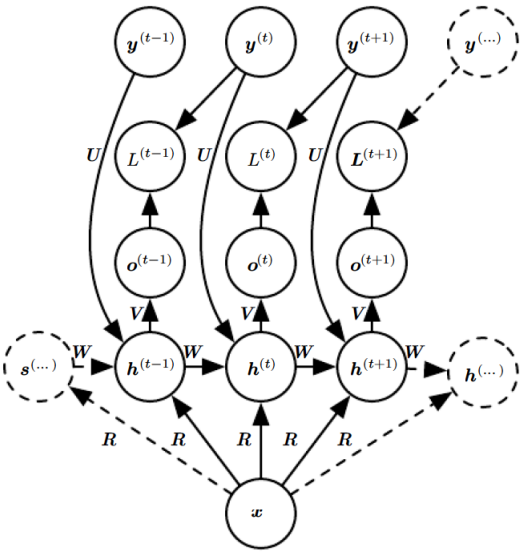
Razlika od klasifikacije segmenata leži u tome da temporalna klasifikacija počiva na algoritmu koji odlučuje gdje u ulaznom nizu treba biti napravljena klasifikacija. Mjerimo ukupan broj supstitucija, ubacivanja i izbacivanja, koja su potrebna da jedan niz pretvore u drugi, i tako dobivamo *učestalost pogrešaka oznaka* E^{lab}:

$$E^{lab}(h, S') = \frac{1}{Z} \sum_{(\boldsymbol{x},\boldsymbol{z})\in S'} ED(h(\boldsymbol{x}), \boldsymbol{z}),$$

gdje je $ED(\boldsymbol{p}, \boldsymbol{q})$ (eng. *edit distance*) udaljenost nizova $\boldsymbol{p}$ i $\boldsymbol{q}$, koja nam daje minimalni potrebni broj supstitucija, ubacivanja i izbacivanja da bismo niz $\boldsymbol{p}$ pretvorili u niz $\boldsymbol{q}$, i može se izračunati u vremenu $O(|\boldsymbol{p}|\,|\boldsymbol{q}|)$.

2.7 Modeliranje nizova koristeći RNN

Ranije smo diskutirali RNN mreže koje primaju ulazni niz vektora $\boldsymbol{x}^{(t)}$, za $t = 1, \ldots, \tau$. Druga opcija je uzeti jedan vektor $\boldsymbol{x}$, kao ulaz u RNN mrežu, koja generira niz $\boldsymbol{y}$ vektora. Kada je $\boldsymbol{x}$ fiksirani vektor, tada veze između $\boldsymbol{x}$ i skrivenih slojeva možemo parametrizirati matricom $\boldsymbol{R}$, kao na slici 2.12. Umnožak $\boldsymbol{x}^\intercal \boldsymbol{R}$ je dodan kao dodatni ulaz skrivenim slojevima u svakom vremenskom koraku. Ovaj model je pogodan za zadatke poput opisivanja slika, gdje je jedna slika ulaz u model koji, kao izlaz, daje niz riječi koje opisuju tu sliku.

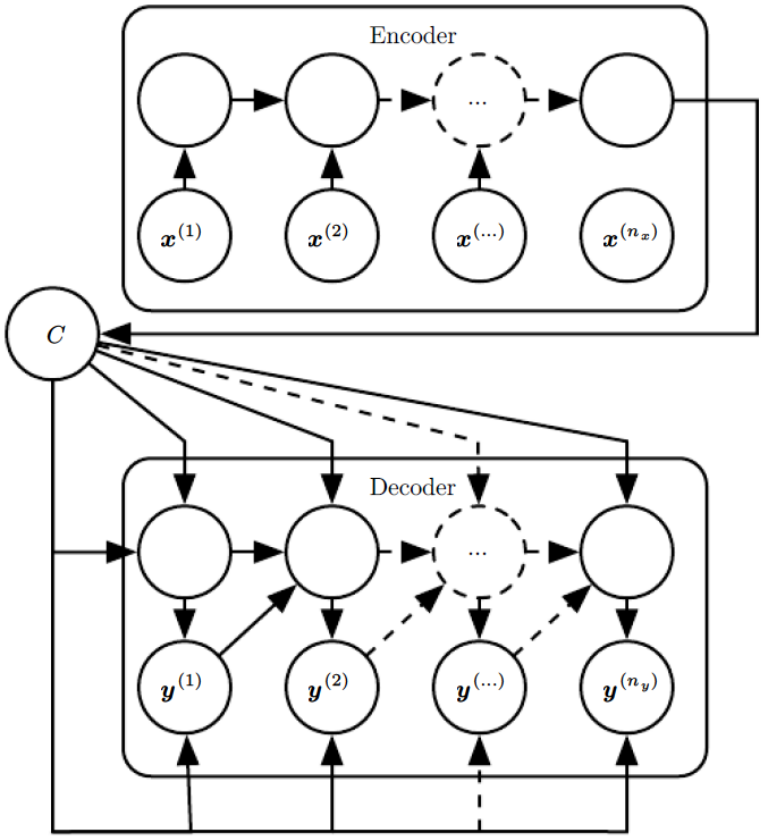


Slika 2.12: RNN mreža preslikava fiksni vektor $\boldsymbol{x}$, u distribuciju nad nizom vektora $\boldsymbol{y}$ [2]

Enkoder-Dekoder Arhitektura

Ovdje diskutiramo kako možemo trenirati RNN da preslikava ulazni niz u izlazni, koji nije nužno iste duljine. To je potrebno u mnogim primjenama, kao što je prepoznavanje govora, ili strojno prevođenje.

Enkoder-Dekoder arhitektura se sastoji od enkoder RNN mreže, koja čita ulazni niz, i dekoder RNN mreže, koja generira izlazni niz (ili računa vjerojatnost danog izlaznog niza). Posljednje skriveno stanje enkodera služi za računanje kontekstne varijable C_v, koja služi kao ulaz u dekoder. Arhitektura je prikazana na slici 2.13.



Slika 2.13: Enkoder-Dekoder arhitektura [2]

Enkoder obrađuje ulazni niz $\boldsymbol{X} = (x^{(1)}, \ldots, x^{(n_x)})$ i, kao funkciju posljednjeg skrivenog neurona, daje izlaz C_v. Dekoder ovisi o tom ulaznom vektoru fiksirane veličine i, kao izlaz, daje niz $\boldsymbol{Y} = (y^{(1)}, \ldots, y^{(n_y)})$, kao u 2.12. Enkoder i dekoder maksimiziraju prosjek od $\log p(\boldsymbol{Y}|\boldsymbol{X})$, po svim parovima $\boldsymbol{x}$ i $\boldsymbol{y}$ u skupu za treniranje.

Poglavlje 3

Implementacija programskog rješenja

3.1 Skup podataka

Skup podataka sadrži 8732 označena zvučna zapisa u WAV formatu, preuzetih s [9]. Sastoji se od deset različitih klasa zvukova: klima uređaj (*air_conditioner*), truba u autu (*car_horn*), djeca koja se igraju (*children_playing*), lavež psa (*dog_bark*), bušilica (*drilling*), rad motora (*enginge_idling*), pucanj iz pištolja (*gun_shot*), udaranje čekića (*jackhammer*), sirena (*siren*), i zvukovi ulice (*street_music*). U imenu datoteke, drugi broj označava klasu kojoj datoteka pripada:

0 = air_conditioner

1 = car_horn

2 = children_playing

3 = dog_bark

4 = drilling

5 = engine_idling

6 = gun_shot

7 = jackhammer

8 = siren

9 = street_music

Skup podataka je već podijeljen u deset mapa, kako bismo primijenili metodu unakrsne validacije.

Unakrsna validacija

Unakrsna validacija je statistička metoda koja se koristi u polju strojnog učenja kako bi procijenila koliko je dobar model strojnog učenja. U metodi unakrsne validacije bira se parametar k, koji određuje na koliko dijelova će se skup podataka podijeliti i organizirati u k mapa (eng. *k-fold cross validation*). Unakrsna validacija se koristi kako bi se procijenilo koliko dobro model strojnog učenja radi na dosad neviđenim podacima. Generalno je manje pristrana i pouzdanija metoda od obične podjele podataka na skup za testiranje i skup za treniranje. Procedura se sastoji od sljedećih koraka:

- nasumično se promiješa skup podataka;
- skup podataka se podijeli u k grupa;
- za svaku grupu:
  - uzmi grupu kao skup podataka za testiranje;
  - uzmi preostale grupe za skup podataka za treniranje;
  - izračunaj točnost na skupu podataka za testiranje i pamti taj podatak;
- računaj prosjek točnosti po svim grupama.

Nema formalnog pravila za odabir broja k, no u polju strojnog učenja, generalno se uzima $k = 10$. Naš skup podataka već je podijeljen metodom unakrsne validacije, gdje je $k = 10$.

3.2 Odabir značajki

U strojnom učenju i prepoznavanju uzoraka, značajka je mjerljivo svojstvo. Odabir informativnih i neovisnih značajki ključan je korak za učinkovite algoritme u prepoznavanju uzoraka. U prepoznavanju uzoraka i strojnom učenju, vektor značajka je n-dimenzionalni vektor numeričkih značajki koje predstavljaju neki objekt. Mnogi algoritmi u strojnom učenju zahtijevaju numerički prikaz objekata. Početni skup sirovih značajki može biti prevelik za upravljanje. Stoga se, u mnogim primjenama strojnog učenja i prepoznavanja uzoraka, preliminarni korak sastoji od odabira podskupa značajki ili izgradnje novog i smanjenog skupa značajki, za olakšavanje učenja. U prepoznavanju govora, značajke za prepoznavanje fonema mogu uključivati omjere šuma, duljinu zvuka i mnoge druge.

Kako bi se postupak ekstrakcije značajki iz zvučnih zapisa olakšao, definirana je pomoćna metoda *extract_feature*, preuzeta sa [6]. Ova metoda je sve što nam je potrebno za pretvoriti čisti zvučni isječak u informativne značajke (uz klasičnu oznaku za svaki zvučni isječak). Oznaka klase svakog zvučnog isječka nalazi se u nazivu datoteke. Na primjer, ako je naziv datoteke *108041-9-0-4.wav*, onda će oznaka klase biti 9.

U obradi zvuka koristi se *mel-frequency cepstrum (MFC)* —- reprezentacija kratkotrajnog spektra snage zvuka. *Mel-frequency cepstral coefficients (MFCCs)* su koeficijenti koji zajedno čine *MFC*. Izvedeni su iz cepstralne reprezentacije zvučnog zapisa. Cepstrum je rezultat inverza Fourierove tranformacije logaritma spektra signala. Više o tome se može naći na [14], [13] i [12]. *MFCC* se koriste kao značajke u prepoznavanju govora.

3.3 Model

Za kreiranje modela rekurentne neuronske mreže koristit ćemo *TensorFlow* — biblioteku otvorenog koda za programski jezik *Python*. Tensorflow omogućava kreiranje modela neuronske mreže bez da sami moramo implementirati određene funkcije, kao što je npr. algoritam s propagiranjem greške unatrag.

Dosad smo upotrebljavali varijable za upravljanje podacima, no postoji osnovnija struktura — *placeholder*. Placeholder je jednostavno varijabla, kojoj ćemo naknadno dodijeliti vrijednost. To nam omogućuje stvaranje mjesta u memoriji gdje ćemo kasnije pohraniti vrijednosti. U `x` i `y` placeholderima će se držati ulazni i ciljni podaci:

```
x = tf.placeholder("float", [None, n_steps, n_input])
y = tf.placeholder("float", [None, n_classes])
```

Za svaku LSTM ćeliju koju inicijaliziramo, trebamo dati broj jedinica u ćeliji. Previsoki broj može dovesti do prenaučenosti, dok premali broj može dovesti do lošijih rezultata. Na nama je da odaberemo broj:

```
cell = rnn_cell.LSTMCell(n_hidden,state_is_tuple=True)
```

Kada koristimo Tensorflow, prvo se definiraju sve funkcije, a kasnije se u Tensorflow sesiji izvrše. Definiramo težine i pristranosti, množimo izlaz s težinama, te dodamo pristranost.

```
weight = tf.Variable(tf.random_normal([n_hidden, n_classes]))
bias = tf.Variable(tf.random_normal([n_classes]))

def RNN(x, weight, bias):
cell = rnn_cell.LSTMCell(n_hidden,state_is_tuple = True)
cell = rnn_cell.MultiRNNCell([cell] * 2)
output, state = tf.nn.dynamic_rnn(cell, x, dtype = tf.float32)
output = tf.transpose(output, [1, 0, 2])
last = tf.gather(output, int(output.get_shape()[0]) - 1)
return tf.nn.softmax(tf.matmul(last, weight) + bias)
```

Nakon množenja izlaza s težinama i dodavanja pristranosti, dobivamo matricu s različitim vrijednostima za svaku klasu. Ono što nas zanima je vjerojatnost da niz pripada određenoj klasi, zato računamo softmax aktivaciju, koja nam daje te vjerojatnosti.

```
prediction = RNN(x, weight, bias)
```

Sljedeći korak je računanje gubitka, tj. pogreške unakrsne entropije, koju nastojimo minimizirati:

```
loss_f = -tf.reduce_sum(y * tf.log(prediction))

optimizer = tf.train.AdamOptimizer(learning_rate =
                        learning_rate).minimize(loss_f)
```

Kao algoritam za učenje koristi se inačica stohastičkog gradijenta spusta — *Adam* (eng. *adaptive moment estimation*). Više o Adam algoritmu se može naći u radu [7], gdje je demonstrirano da je Adam vrlo učinkovit u području strojnog učenja, kada se radi o modelima dubokih neuronskih mreža i velikim skupovima podataka.

Potrebno: veličina koraka ϵ (prijedlog: 0.001);
Potrebno: parametri pomičnog prosjeka ρ_1 i ρ_2 (prijedlog: 0.9 i 0.999, respektivno);
Potrebno: mala konstanta δ (prijedlog 10^{-8});
Potrebno: parametar θ;
Inicijaliziraj varijable prvog i drugog momenta $s = 0$ i $r = 0$;
Inicijaliziraj vremenski takt na $t = 0$;
while *dok nisu zadovoljeni kriteriji zaustavljanja* **do**
  uzmi uzorak od m primjeraka treniranja iz skupa za treniranje $x^{(1)}, \ldots, x^{(m)}$ i ciljem $y^{(i)}$;
  računaj gradijent: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$;
  $t \leftarrow t + 1$;
  $s = \rho_1 s + (1 - \rho_1) g$;
  $r = \rho_2 r + (1 - \rho_2) g \odot g$;
  $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$;
  $\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$;
  $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$;
  $\theta \leftarrow \theta + \Delta\theta$;
end

Algorithm 1: Adam algoritam

Točnost (*accuracy*) nam daje koliki je postotak nizova ispravno klasificiran:

```
correct_pred = tf.equal(tf.argmax(prediction,1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

Nakon što smo dizajnirali model, sada ga treba izvršiti. Pokrenemo sesiju, inicijaliziramo sve varijable koje smo definirali i započnemo treniranje.

```
with tf.Session() as session:
session.run(init)
```

Nakon svakih 100 koraka ispisuje se točnost na skupu podataka za treniranje i vrijednost pogreške unakrsne entropije. Nakon svakih 1000 koraka ispisujemo tzv. *matricu konfuzije* — matricu koja nam omogućuje vizualizaciju performanse algoritma nadziranog učenja [10]. Svaki redak matrice reprezentira predviđene oznake klase, dok svaki stupac predstavlja ispravne oznake. Zato, ako bismo imali savršeni model, matrica konfuzije bila bi dijagonalna, jer bi svaka oznaka bila ispravno predviđena.

3.4 Rezultati

U sljedećoj tablici prikazana je točnost za svaku od 10 grupa, dobivenih metodom unakrsne validacije na prije opisani način:

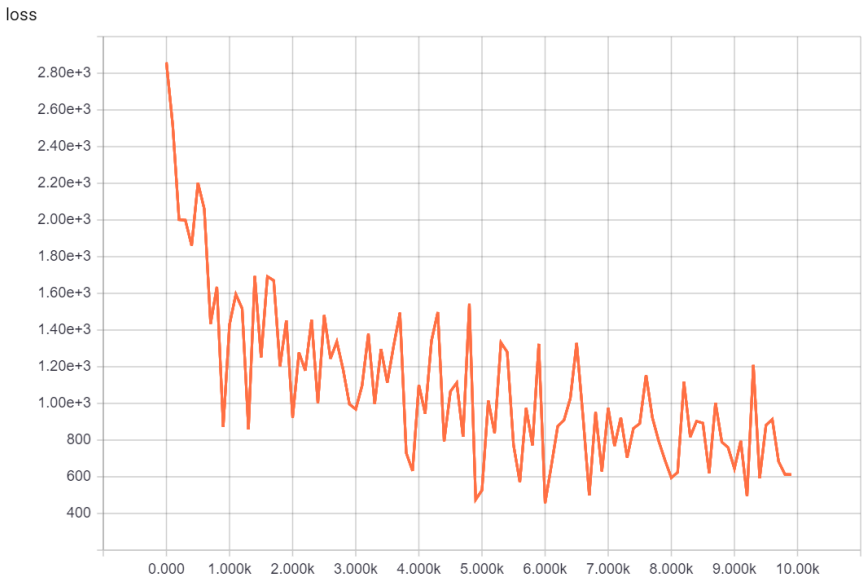
grupa	točnost (%)
fold1	40.4
fold2	39.5
fold3	35.8
fold4	42.9
fold5	45.4
fold6	43.0
fold7	39.8
fold8	51.0
fold9	50.5
fold10	53.3

Prosječna točnost po svim grupa je **44.1%**.

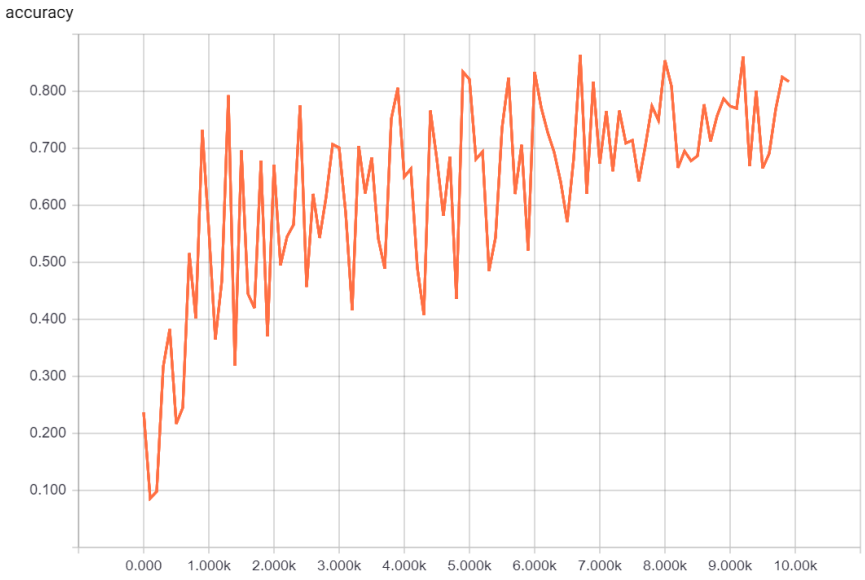
Sljedeća tablica prikazuje matricu konfuzije, za slučaj kada je deseta grupa služila kao skup podataka za testiranje. Vidimo da je, u tom slučaju, najbolja točnost.

	0	1	2	3	4	5	6	7	8	9
0	346	15	20	4	72	3	0	36	57	21
1	5	58	3	2	1	54	0	0	14	12
2	18	24	416	70	21	51	30	27	130	59
3	0	3	100	280	73	1	28	10	58	0
4	7	2	1	44	308	0	8	211	11	69
5	45	25	14	30	33	425	2	3	21	21
6	0	0	0	0	0	0	0	1	0	1
7	84	3	0	0	42	34	0	277	1	22
8	77	1	22	25	36	14	2	2	255	29
9	118	4	116	60	37	58	2	71	6	466

Na kraju, koristeći alat za vizualizaciju *TensorBoard* (koji dolazi zajedno s Tensorflowom), prilažemo slike koje pokazuje kako se mijenjala pogreška unakrsne entropije (slika 3.1) i točnost (slika 3.2), kroz treniranje mreže, gdje se treniralo na grupama 1–9:



Slika 3.1: Pogreška unakrsne entropije



Slika 3.2: Točnost

Bibliografija

[1] M. Čupić, http://www.zemris.fer.hr/~ssegvic/du/du3optimization.pdf, posjećeno 1. 9. 2018.

[2] I. Goodfellow, Y. Bengio i A. Courville, *Deep Learning*, MIT Press, 2016, http://www.deeplearningbook.org.

[3] A. Graves, *Supervised Sequence Labeling with Recurrent Neural Networks*, Springer-Verlag Berlin Heidelberg, 2012.

[4] B. Hammer, *Learning with Recurrent Neural Networks*, Springer-Verlag London, 2000.

[5] K. Hornik, M. Stinchcombe i H. White, *Multilayer feedforward networks are universal approximators*, Neural Networks **2** (1989), br. 5, 359–366.

[6] KDnuggets, https://www.kdnuggets.com/2016/09/urban-sound-classification-neural-networks-tensorflow.html, posjećeno 1. 9. 2018.

[7] D. P. Kingma i J. Ba, *Adam: a Method for Stochastic Optimization*, ICLR 2015 (2015), 1–15, http://arxiv.org/abs/1412.6980.

[8] M. A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015, http://neuralnetworksanddeeplearning.com/.

[9] J. Salamon, C. Jacoby i J. P. Bello, https://urbansounddataset.weebly.com/urbansound8k.html, posjećeno 1. 9. 2018.

[10] Wikipedia contributors, *Confusion matrix — Wikipedia, The Free Encyclopedia*, 2018, https://en.wikipedia.org/w/index.php?title=Confusion_matrix&oldid=849067221, posjećeno 1. 9. 2018.

[11] _____, *Feature (machine learning) — Wikipedia, The Free Encyclopedia*, 2018, https://en.wikipedia.org/w/index.php?title=Feature_(machine_learning)&oldid=856849809, posjećeno 1. 9. 2018.

[12] _____, *Mel-frequency cepstrum — Wikipedia, The Free Encyclopedia*, 2018, https://en.wikipedia.org/w/index.php?title=Mel-frequency_cepstrum&oldid=852631751, posjećeno 1. 9. 2018.

[13] _____, *Mel scale — Wikipedia, The Free Encyclopedia*, 2018, https://en.wikipedia.org/w/index.php?title=Mel_scale&oldid=856366148, posjećeno 1. 9. 2018.

[14] _____, *Spectral density — Wikipedia, The Free Encyclopedia*, 2018, https://en.wikipedia.org/w/index.php?title=Spectral_density&oldid=851825921, posjećeno 1. 9. 2018.

Sažetak

U ovom radu objašnjavamo pojam neuronskih mreža i promatramo njihov matematički aspekt. Dajemo generalni uvid u arhitekturu neuronskih mreža, te način na koji neuronska mreža uči i kako se zatim ponaša na dosad neviđenim podacima.

Zatim, dajemo naglasak na rekurentne neuronske mreže i kako one rješavaju problem klasifikacije nizova. Objašnjavamo posebnu arhitekturu rekurentnih neuronskih mreža, tzv. Long-Short Term Memory arhitekturu.

Na kraju, pokazujemo kako smo iskoristili programski jezik *Python* i biblioteku otvorenog koda *Tensorflow*, za izradu modela rekurentne neuronske mreže, koji se sastoji od dvije Long-Short Term Memory jedinice, kako bismo riješili problem klasifikacije gradskih zvukova, na *UrbanSound 8k* skupu podataka. Koristimo metodu unakrsne validacije i, kao rezultate, prikazujemo prosječnu točnost koju dobivamo na testnim skupovima podataka, te matricu konfuzije za jedan testni skup podataka.

Summary

In this thesis we explain the concept of neural networks and observe their mathematical aspect. We give a general insight into the architecture of neural networks, the way the neural network learns, and how it behaves on previosly unseen data.

Then, we give a special accent to recurrent neural networks and how they solve the problem of sequence classification. We explain a special architecture of recurrent neural networks, the so-called Long-Short Term Memory architecture.

In the end, we present how we used Python programming language and the open source library Tensorflow, to create a recurrent neuronal network model that consists of two Long-Short Term Memory units, to solve the problem of urban sound classification, on the *Urban Sound 8k* data set. We use the cross-validation method and, as results, we give the average accuracy obtained for test datasets, and the confusion matrix for one test dataset.

Životopis

Rođena sam 1. srpnja 1992. godine u Varaždinu, a odrasla u Zbelavi. Godine 1999. upisujem se u VII. osnovnu školu Varaždin, a nakon završetka osnovne škole, 2007. godine upisujem se u Prvu gimnaziju u Varaždinu. Nakon završetka srednje škole, 2011. godine upisujem Preddiplomski sveučilišni studij Matematika na Prirodoslovno–matematičkom fakultetu u Zagrebu i 2016. godine sam stekla titulu prvostupnice matematike. Iste godine upisujem Diplomski sveučilišni studij Računarstvo i matematika.