Rekurentne neuronske mreže

Pešut, Lovre

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Lovre Pešut

REKURENTNE NEURONSKE MREŽE

Diplomski rad

Voditelj rada:
prof. dr. sc. Miljenko Huzak

Zagreb, studeni 2019.

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. _____

Sandiju za članke

Sadržaj

Sadržaj **iv**

Uvod **3**

1 Osnove neuronskih mreža **4**
1.1 Perceptron 4
1.2 Unaprijedne neuronske mreže 9

2 Rekurentne neuronske mreže **20**
2.1 Uvod u rekurentne neuronske mreže 20
2.2 "Obične" rekurentne neuronske mreže 21
2.3 LSTM mreže 27
2.4 Rekurentne mreže s neprekidnim vremenom 29

3 Primjene rekurentnih neuronskih mreža **31**
3.1 Pregled primjena 31
3.2 Eksperimenti s rekurentnim neuronskim mrežama 32

Bibliografija **38**

Uvod

Ime "neuronske mreže" odmah budi asocijaciju na mozak. Unatoč tome što su neronske mreže relativno jednostavni matematički modeli, a mozak još uvijek misteriozna nakupina kemijskih spojeva i stanica, u onoj mjeru u kojoj su neuronske mreže nastale u analogiji s mozgom, možemo reći da u ovom radu pratimo jedan niz pokušaja da se, aproksimacijom i analogijom, uhvati srž funkcioniranja najmoćnijeg stroja za učenje kojem trenutno imamo pristup. Vidjet ćemo kako se drastičnim pojednostavljenjem – čak i iskrivljenjem – rada mozga došlo do sustava koji su sposobni naučiti, odnosno modelirati kompleksne uzorke.

Embrio tih pojednostavljenja je tzv. perceptron. Ono što nam je poznato o mozgu je to da neuroni ispaljuju električne impulse onda kad im napon prijeđe određeni prag. Kako bismo povukli matematičku analogiju s tim? Pretpostavimo da imamo funkciju

$$f : D \to \{0, 1\},\ D \subseteq \mathbb{R}^n,$$

odnosno funkciju koja nam daje "binarnu klasifikaciju" nekog skupa n-dimenzionalnih vektora, te da su nam poznate njezine vrijednosti na konačno mnogo točaka. Perceptron je tada model koji aproksimira tu funkciju s funkcijom $\hat{f} : \mathbb{R}^n \to \mathbb{R}$,

$$\hat{f}(x) = \begin{cases} 1 & \text{ako je } w \cdot x - \theta > 0, \\ 0 & \text{inače,} \end{cases}$$

s $w \in \mathbb{R}^n$ i $\theta \in \mathbb{R}$, gdje se w i θ izračunavaju uz pomoć algoritma koji opisujemo o poglavlju 1. Dobra vijest za perceptron je to što taj algoritam konvergira ako postoje w i θ takvi da je $f(x) = \hat{f}(x)$ za sve x čije su nam funkcijske vrijednosti poznate. Loša je vijest to što takvi w i θ postoje tek u nekim vrlo strogim okolnostima – naime, iščitavajući definiciju perceptrona, možemo vidjeti da je nužno da postoji neka hiperravnina u $\mathbb{R}^n$, dana s $w \cdot x - \theta = 0$, takva da točke s vrijednosti 0 "leže s jedne strane", a točke s vrijednosti 1 "leže s druge strane".

S obzirom na ograničenost perceptrona, može nas iznenaditi da postoji jednostavna generalizacija perceptrona koja je "univerzalni aproksimator", odnosno model koji može aproksimirati proizvoljnu neprekidnu funkciju $f : D \to \mathbb{R}$, gdje je D kompaktan podskup

$\mathbb{R}^n$. Danu funkciju aproksimiramo s

$$\hat{f}(x) = \sum_{i=1}^{k} w_i \sigma(v_i \cdot x + v_{i,0}) + w_0,$$

gdje su $w_i \in \mathbb{R}$, $v_{i,0} \in \mathbb{R}$, $v_i \in \mathbb{R}^n$, za $i = 0, \ldots, k$, a $\sigma : \mathbb{R} \to \mathbb{R}$ je sigmoidna funkcija, dana s

$$\sigma(t) = \frac{1}{1 + e^{-t}}.$$

Taj model se može konceptualizirati kao da "šaljemo" vektor $x \in \mathbb{R}^n$ u k "perceptrona" – sloj perceptrona u ovom modelu zovemo "skriveni sloj" – s *ulaznim težinama* v_i i *pragom* $v_{i,0}$, gdje se izlaz perceptrona "izglađuje" sigmoidnom funkcijom, te konačno taj izlaz množimo s *izlaznim težinama* w_i i *pragom* w_0. Unatoč teorijskoj moći ovog aproksimatora, u praksi nalaženje idealnih parametara nije jednostavan posao, budući da efektivne metode treniranja imaju tendenciju naći se u lokalnim optimumima.

U praksi također često pomaže proširiti naš model s više od jednog "sloja" perceptrona. Primjer iz prošlog paragrafa, višeslojne mreže perceptrona i druge varijacije na sada opisanu temu su dio raznovrsnog asortimana *unaprijednih neuronskih mreža*. Nije teško vidjeti kako bismo dosadašnje primjere proširili na aproksimaciju funkcije $f : \mathbb{R}^n \to \mathbb{R}^m$.

U stvarnim primjenama, doduše, često se događa to da veličina ulaza ili veličina izlaza nisu konstantne. Primjerice, rečenica od 5 riječi prevedena s hrvatskog na engleski ne mora nužno i nakon prijevoda imati 5 riječi. Ako želimo klasificirati rečenice kao pozitivne ili negativne, željeli bismo da nam ulazna rečenica bude proizvoljne duljine, dok bi izlaz mogao biti jedan realan broj (0 za negativnu rečenicu, 1 za pozitivnu). Naposlijetku, slike fiksne veličine možemo željeti opisati rečenicom proizvoljne dužine. Svi ti scenariji: preslikavanje mnogo-na-mnogo, mnogo-na-jedan, te jedan-na-mnogo, podobni su za primjenu *rekurentnih neuronskih mreža*.

Rekurentne neuronske mreže dolaze u mnogo oblika i arhitektura, ali zajedničko im je obilježje prisutnost rekurencije koja omogućavaja modelu da održi neko "sjećanje" na prošle ulazne podatke, što vodi do mogućnosti modeliranja proizvoljnih nizova. U vrlo općenitim i pojednostavljenim terminima, ako imamo konačni niz $(x^{(t)})$, $t = 1, \ldots, n$, onda ako je funkcija f_θ unaprijedna neuronska mreža, gdje su θ parametri unaprijedne neuronske mreže, tada ona vezu između ulaza i izlaza možemo zapisati $f_\theta(^{(t)}) = \hat{y}^{(t)}$. Dakle, sa zadanim parametrima unaprijedne neuronske mreže, izlaz neuronske mreže je funkcija (samo) ulaza. Rekurentna neuronska mreža g, pak, računa

$$g_\theta(x^{(1)}, h^{(0)}) = \hat{y}^{(1)}$$
$$g_\theta(x^{(t)}, h^{(t-1)}) = \hat{y}^{(t)}, \ t > 1$$

gdje je $h^{(0)}$ unaprijed zadana konstanta, a $h^{(t-1)}$, tzv. skriveno stanje u vremenu $t-1$, je funkcija $h^{(t-2)}$ i $x^{(t-1)}$. Dakle, o $h^{(t)}$ razmišljamo kao o "pamćenju" neuronske mreže nakon što je vidjela ulaz $x^{(t)}$, koje je funkcija trenutnog ulaza ($x^{(t)}$) i prijašnjeg "pamćenja" ($h^{(t-1)}$).

Najviše razmatramo "obične" rekurentne neuronske mreže, koje su analogoni već opisane unaprijedne neuronske mreže s jednim slojem "skrivenih" perceptrona, osim što su u takvom modelu perceptroni povezani i s "vremenskom vezom", tako da se aktivacije iz jednog vremenskog koraka šalju, osim u izlaz u tom vremenu, i u sljedeći vremenski korak.

Osim njih definiramo i izrazito uspješne LSTM[1] mreže, koje zamjenjuju perceptrona kao osnovne "računalne jedinice" s "LSTM ćelijama", koje imaju sofisticiraniji način održavanja pamćenja na prijašnje vremenske korake.

Na kraju, testiramo rekurentne neuronske mreže na nekoliko praktičnih zadataka. Prvo testiramo njihovu mogućnost da "zamijene nule i jedinice", odnosno da niz nula i jedinica $x^{(1)}, \ldots, x^{(t)}$ mapiraju na niz nula i jedinica $y^{(1)}, \ldots, y^{(t)}$, takav da $y^{(1)} = 0$ ako i samo ako je $x^{(1)} = 1$. Tada testiramo koliko je "perceptrona" potrebno "običnoj" rekurentnoj mreži da bi naučila ponoviti niz koji je "vidjela" prije $t = 2, \ldots, 10$ vremenskih koraka, te testiramo njezinu sposobnost da razlikuje normalne distribucije s različitom očekivanom vrijednošću i istom varijancom, ili istom očekivanom vrijednošću i različitom varijancom.

[1] Kratica od eng. *long short-term memory*

Poglavlje 1

Osnove neuronskih mreža

1.1 Perceptron

Povijest i motivacija

Godine 1943. Waren S. McCulloch i Walter H. Pitts uveli model umjetnog neurona, referirajući se na dotadašnje znanje neuroznanosti kako bi argumentirali da je težinska suma ulaza uspoređena s određenim pragom – gdje je izlaz neurona 1 ako je ta suma veća ili jednaka pragu, a 0 ako je suma manja ili jednaka pragu – razuman analogon biološkom neuronu. McCulloch i Pitts nisu pružili proceduru kojim bi se težine takvog neurona naučile, ali je njihov rad, zbog uspostavljanja veze između bioloških sustava i digitalnih računala, popraćen s velikim interesom.

U kasnim 1950. godinama, Frank Rosenblatt i nekoliko drugi istraživača su uveli model perceptrona. Neuroni u tim modelima su bili slični kao oni u modelima McCullocha i Pittsa, ali je Rosenblattov doprinos bio taj što je uveo algoritam kojim bi se perceptron trenirao, odnosno algoritam koji za dani konačan broj parova ulaza i željenih izlaza, $(x_1, y_1), \ldots, (x_n, y_n)$, pronalazi težine θ takve da za perceptron P_θ vrijedi $P_\theta(x_i) = y_i$ za sve $i = 1, \ldots, n$, pod uvjetom da takve težine postoje.

Godine 1958. u američkom časopisu The New York Timesu, u izvještaju o Rosenblattovoj konferenciji za novinare u organizaciji američke mornarice, nalazimo da je perceptron "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence"[1]

No Marvin Minsky i Seymour Papert su 1969. godine objavili knjigu Perceptrons u kojoj su potanko analizirali nedostatke perceptrona. Tek su kasnih 1980. godina istraživači

[1] Prijevod autora: embrio elektroničkog računala za koji [američka mornarica] očekuje da će biti u mogućnosti hodati, pričati, gledati, razmožavati se i biti svjestan svog postojanja.

nadišli ograničenja perceptrona uvodeći višeslojne perceptrone, odnosno unaprijedne neuronske mreže. Za više o povijesti perceptrona vidi [12].

Kao važno poglavlje u razvitku neuronskih mreža, u nastavku analiziramo perceptron i njegov algoritam za učenje.

Definicija perceptrona i algoritam za učenje

Ponavljamo, u malo širim okvirima, definiciju perceptrona iz uvoda. Podsjetimo se, perceptron koristimo onda kada želimo odrediti binarnu klasifikaciju određenog podskupa $\mathbb{R}^n$.

Definicija 1.1.1 (Perceptron)**.** *Perceptron je skup funkcija na $D \subseteq \mathbb{R}^n$, u ovisnosti o parametrima $w \in \mathbb{R}^n$ i $\theta \in \mathbb{R}$, takav da*

$$\mathrm{P}_{w,\theta}(x) = \begin{cases} 1 & \text{ako je } w \cdot x - \theta \geq 0, \\ 0 & \text{inače.} \end{cases} \tag{1.1}$$

Koordinate w zovemo težinama, a broj θ pragom.

Za nalaženje težina i praga koristimo sljedeći algoritam. Važno svojstvo tog algoritma jest da, ako za konačan niz parova $(x_1, y_1), \ldots, (x_k, y_k)$, $(x_i, y_i) \in D \times \{0, 1\}$ za $i = 1, \ldots, k$ postoje w i θ takvi da je

$$\mathrm{P}_{w,\theta}(x_i) = y_i, \forall i \in \{1, \ldots k\},$$

onda algoritam nalazi takve, ili ekvivalentne, w i θ. Da bismo lakše iskazali algoritam, uvodimo koncept proširenog vektora težina, gdje ako u 1.1.1 imamo $w = (w_1, \ldots, w_n)$, onda je prošireni vektor duljine dan s $\mathbf{w} = (w_1, \ldots, w_n, -\theta)$. Analogono proširujemo i ulazni vektor $x = (x_1, \ldots, x_n)$ na $\mathbf{x} = (x_1, \ldots, x_n, 1)$. Tada nejednakost $w \cdot x - \theta > 0$ iz 1.1 možemo zapisati kao $\mathbf{w} \cdot \mathbf{x} > 0$. Također, perceptron $\mathrm{P}_{w,\theta}$ skraćeno pišemo $\mathrm{P}_{\mathbf{w}}$.

Definicija 1.1.2 (Algoritam za učenje perceptrona)**.** *Neka je dan konačan niz parova*

$$(x_1, y_1), \ldots, (x_k, y_k),\ (x_i, y_i) \in D \times \{0, 1\},\ D \subseteq \mathbb{R}^m\,.$$

Označimo s P skup proširenih vektora onih x_i takvih da je $y_i = 1$, a s N skup proširenih vektora onih x_i takvih da je $y_i = 0$. Prošireni vektor od x_i pišemo $\mathbf{x}_i$. Algoritam za nalaženje proširenog vektora težina za 1.1.1 je sljedeći:

1. *Inicijaliziramo vektor težina $\mathbf{w}_0 = (0, \ldots, 0)$ i stavimo $t := 0$.*
2. *Provjerimo imamo li za sve x_i, $i = 1, \ldots, k$, $\mathrm{P}_{\mathbf{w}}(x_i) = y_i$. Ako da, onda je algoritam završen i izlaz je $\mathbf{w}_t$. Ako ne, onda na slučajan način izaberemo prošireni vektor $\mathbf{x} \in P \cup N$.*

a) Ako je $\mathbf{x} \in P$ *i* $\mathbf{w}_t \cdot \mathbf{x} > 0$*, onda idi na 2,*

b) Ako je $\mathbf{x} \in P$ *i* $\mathbf{w}_t \cdot \mathbf{x} \leq 0$*, onda idi na 3,*

c) Ako je $\mathbf{x} \in N$ *i* $\mathbf{w}_t \cdot \mathbf{x} < 0$*, onda idi na 2,*

d) Ako je $\mathbf{x} \in N$ *i* $\mathbf{w}_t \cdot \mathbf{x} \geq 0$*, onda idi na 4.*

3. Stavi $\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{x}$ *i* $t := t + 1$*, i idi na 2.*

4. Stavi $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{x}$ *i* $t := t + 1$*, i idi na 2.*

Vektor težina u prvom koraku također možemo inicijalizirati i na bilo koju drugi način, tj. u budućim dokazima tražimo samo $\mathbf{w}_0 \in \mathbb{R}^{m+1}$.

Primijetimo da "klasifikacijski test" nije identičan u definiciji perceptrona 1.1.1 i algoritmu za njegovo učenje 1.1.2. Naime, perceptron u sebi sadrži "test" $\mathbf{w} \cdot \mathbf{x} \geq 0$, dok algoritam ne mijenja vektor težina za klasu P ako i samo ako je $\mathbf{w}_t \cdot \mathbf{x} > 0$. U sljedećoj sekciji propozicija 1.1.5 nam daje opravdanje za tu razliku.

Linearna separabilnost i dokaz konvergencije

Kako bismo iskazali i dokazali dovoljan uvjet za završetak algoritma za učenje perceptrona 1.1.2, trebamo koncept linearne separabilnosti dva skupa točaka u n-dimenzionalnom prostoru.

Definicija 1.1.3 (Linearna separabilnost dva skupa)**.** *Skup* $A \subset \mathbb{R}^n$ *i* $B \subset \mathbb{R}^n$ *zovemo linearno separabilnim ako postoje realni brojevi* $w_1, \ldots, w_n, w_{n+1}$*, takvi da svaka točka* $(a_1, \ldots, a_n) \in A$ *zadovoljava*

$$\sum_{i=1}^{n} w_i a_i \geq w_{n+1},$$

a svaka točka $(b_1, \ldots, b_n) \in B$ *zadovoljava*

$$\sum_{i=1}^{n} w_i b_i < w_{n+1}.$$

Kažemo da vektor $w = (w_1, \ldots, w_n)$ *razdvaja A i B.*

Linearna separabilnost odgovara uvjetu perceptrona. Uvjetu algoritma za učenje perceptrona odgovara apsolutna separabilnost.

Definicija 1.1.4 (Apsolutna linearna separabilnost dva skupa)**.** *Skup* $A \subset \mathbb{R}^n$ *i* $B \subset \mathbb{R}^n$ *zovemo apsolutno linearno separabilnim ako postoje realni brojevi* $w_1, \ldots, w_n, w_{n+1}$*, takvi*

da svaka točka $(a_1, \ldots, a_n) \in A$ *zadovoljava*

$$\sum_{i=1}^{n} w_i a_i > w_{n+1},$$

a svaka točka $(b_1, \ldots, b_n) \in B$ *zadovoljava*

$$\sum_{i=1}^{n} w_i b_i < w_{n+1}.$$

Sada dokazujemo ekvivalenciju ta dva pojma u slučaju konačnog broja točaka.

Propozicija 1.1.5. *Dva konačna skupa, A i B, n-dimenzionalnih točaka su linearno separabilni ako i samo ako su apsolutno linearno separabilni.*

Dokaz. Ako su apsolutno linearno separabilni, onda su očito i linearno separabilni. Dakle, trebamo pokazati drugi smjer. Neka su A i B linearno separabilni. Tada po definiciji postoje $w_1, \ldots, w_{n+1}$ takvi da za svaki $(a_1, \ldots, a_n) \in A$ imamo

$$\sum_{i=1}^{n} w_i a_i \geq w_{n+1} \tag{1.2}$$

i za svaki $(b_1, \ldots, b_n) \in B$ imamo

$$\sum_{i=1}^{n} w_i b_i < w_{n+1}. \tag{1.3}$$

Definirajmo

$$\varepsilon = \max\left\{\left(\sum_{i=1}^{n} w_i b_i\right) - w_{n+1} \mid (b_1, \ldots, b_n) \in B\right\}.$$

Budući da je B konačan, ε postoji, a zbog 1.3 je negativan. Onda imamo i $\varepsilon < \varepsilon/2 < 0$, te stavljajući $w' = w_{n+1} + \varepsilon/2$ vrijedi

$$\sum_{i=1}^{n} w_i a_i - (w' - \varepsilon/2) \geq 0 \tag{1.4}$$

za sve $(a_1, \ldots, a_n) \in A$. Raspisujući 1.4 i uzimajući u obzir da je $-\varepsilon/2 > 0$, dobivamo

$$\sum_{i=1}^{n} w_i a_i > w'.$$

Na analogan način bismo pokazali da za sve $(b_1, \ldots, b_n) \in B$ vrijedi

$$\sum_{i=1}^{n} w_i b_i < w',$$

što znači da su A i B apsolutno linearno separabilni. □

Sada smo spremni dokazati da algoritam 1.1.2 konvergira ako su P i N konačni i linearno separabilni.

Teorem 1.1.6 (Dovoljan uvjet konvergencije algoritma za učenje perceptrona)**.** *Ako su P i N konačni i linearno separabilni, onda algoritam 1.1.2 promijeni prošireni vektor težina $\mathbf{w}_t$ konačno mnogo puta.*

Dokaz. Prvo napomenimo da bez gubitka općenitosti možemo P i N objediniti u jedan skup, $\mathcal{P}$, $\mathcal{P} = P \cup (-1)N$, gdje je $(-1)N = \{(-x_1, \ldots, -x_n) \mid (x_1, \ldots, x_n) \in N\}$. Uz tu modifikaciju u algoritmu za učenje 1.1.2 izbacujemo četvrti korak.

Nadalje, vektore u $\mathcal{P}$ možemo normirati, budući da je $\mathbf{w} \cdot \mathbf{x} > 0$ ekvivalentno s $\mathbf{w} \cdot (\alpha \mathbf{x}) > 0$, gdje je α pozitivni skalar. Analogan argument vrijedi i za vektore težina. Pošto smo pretpostavili da su P i N linearno separabilni, oni su i apsolutno linearno separabilni. Neka je $\mathbf{w}^*$ normiran vektor koji ih razdvaja.

Pretpostavimo da smo nakon $t + 1$ koraka izračunali vektor težina $\mathbf{w}_{t+1}$. Dakle, iz definicije algoritma za učenje slijedi da u vremenu t vektor $\mathbf{p}_t$ nije bio točno klasificiran, te da je

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{p}_t. \tag{1.5}$$

Kosinus kuta α između $\mathbf{w}_{t+1}$ i $\mathbf{w}^*$ je dan s

$$\cos \alpha = \frac{\mathbf{w}^* \cdot \mathbf{w}_{t+1}}{\|\mathbf{w}_{t+1}\|}, \tag{1.6}$$

gdje, koristeći 1.5, imamo

$$\mathbf{w}^* \cdot \mathbf{w}_{t+1} = \mathbf{w}^* \cdot \mathbf{w}_t + \mathbf{w}^* \cdot \mathbf{p}_t. \tag{1.7}$$

Budući da vektor $\mathbf{w}^*$ definira apsolutnu linearnu separaciju između P i N, možemo definirati $\beta = \min\{\mathbf{w}^* \cdot \mathbf{p} \mid \mathbf{p} \in \mathcal{P}\}$, gdje onda imamo $\beta > 0$. Posebno, $\mathbf{w}^* \cdot \mathbf{p}_t \geq \beta$, pa imamo i

$$\mathbf{w}^* \cdot \mathbf{w}_{t+1} \geq \mathbf{w}^* \cdot \mathbf{w}_t + \beta.$$

Nastavljajući induktivno, dobivamo

$$\mathbf{w}^* \cdot \mathbf{w}_{t+1} \geq \mathbf{w}^* \cdot \mathbf{w}_0 + (t + 1)\beta. \tag{1.8}$$

Dobili smo donju granicu brojnika (1.6). Kako bismo dobili ocjenu nazivnika (1.6), primijetimo da je

$$\begin{aligned}\|\mathbf{w}_{t+1}\|^2 &= (\mathbf{w}_t + \mathbf{p}_t)(\mathbf{w}_t + \mathbf{p}_t) \\ &= \|\mathbf{w}_t\|^2 + 2\mathbf{w}_t \cdot \mathbf{p}_t + \|\mathbf{p}_t\|^2.\end{aligned}$$

Pogrešna klasifikacija $\mathbf{p}_t$ znači upravo da je $\mathbf{w}_t \cdot \mathbf{p}_t \leq 0$, pa jer smo napomenuli da možemo smatrati da su svi vektori u $\mathcal{P}$ normirani, imamo

$$\|\mathbf{w}_{t+1}\|^2 \leq \|\mathbf{w}_t\|^2 + \|\mathbf{p}_t\|^2 \leq \|\mathbf{w}_t\|^2 + 1.$$

Opet nastavljajući induktivno dobivamo

$$\|\mathbf{w}_{t+1}\|^2 \leq \|\mathbf{w}_0\|^2 + (t+1). \tag{1.9}$$

Sada kombinirajući 1.8 i 1.9 imamo

$$\cos\alpha \geq \frac{\mathbf{w}^* \cdot \mathbf{w}_0 + (t+1)\beta}{\sqrt{\|\mathbf{w}_0\|^2 + (t+1)}}.$$

Ali,

$$\lim_{t\to\infty} \frac{\mathbf{w}^* \cdot \mathbf{w}_0 + (t+1)\beta}{\sqrt{\|\mathbf{w}_0\|^2 + (t+1)}} = \infty.$$

Dakle, kako znamo da je $\cos\alpha \leq 1$, to znači da je t ograničen, što znači da je broj promjena težinskog vektora konačan. □

Dakle, u vrlo ograničenom kontekstu perceptron može (ako zanemarimo brzinu konvergencije) raditi najbolje moguće. Više o perceptronu se može naći u [7]. U većini situacija u životu, doduše, ne nalazimo linearnu separabilnost, već je granica odluke nelinearna. U sljedećim sekcijama uvodimo algoritme kojima možemo modelirati i nelinearne veze.

1.2 Unaprijedne neuronske mreže

Višeslojni perceptron

U vrlo općenitim terminima, neuronske mreže možemo smatrati modelima koji reprezentiraju nelinearne funkcije između skupa ulaznih varijabli i skupa izlaznih varijabli. Dakle, možemo reći da se neuronska mreža sastoji od skupa *ulaznih jedinica*, skupa *računalnih jedinica* (neurona), te skupa usmjerenih *veza* između neurona. Uvjet unaprijednosti je taj

da neurone možemo označiti cijelim brojevima, tako da, ukoliko postoji veza iz jedinice i u jedinicu j, imamo $i < j$.

Svakoj jedinici pridružujemo realan broj koji je njezin *izlaz*. Ulazne jedinice su prolazni elementi ulaznih podataka, pa u slučaju n-dimenzionalnog ulaza imamo n ulaznih jedinica. Izlaz neurona, pak, je određena funkcija izlaza jedinica koje imaju vezu usmjerenu prema tom neuronu. Tu funkciju zovemo *funkcija aktivacije*, te je ona zaslužna za mogućnost da s neuronskim mrežama možemo modelirati nelinearne funkcije.

Najpoznatija varijanta neuronske mreže je takozvani višeslojni perceptron[2]. Perceptron koji se tu pojavljuje nije identičan perceptronu opisanom do sada. Naime, perceptron u prošlim sekcijama se može prikazati kao kompozicija funkcija

$$P_{w,\theta}(x) = \tau(g(x)),$$

gdje je $g(x) = w \cdot x - \theta$, a

$$\tau(z) = \begin{cases} 1 & \text{ako je } z \geq 0, \\ 0 & \text{inače.} \end{cases}$$

U nastavku dopuštamo da τ bude proizvoljna funkcija. Nju, sukladno utvrđenoj terminologiji, nazivamo aktivacijskom funkcijom perceptrona.

Dakle, višeslojni perceptron se sastoji od ulaznog sloja i k slojeva perceptrona, gdje zadnji od tih k slojeva smatramo izlaznim slojem.[3] Svaku jedinicu ulaznog sloja također možemo smatrati perceptronom s težinom 1, pragom 0 i identitetom kao aktivacijskom funkcijom.

Tako formulirano, u višeslojnom perceptronu za svaki par perceptrona iz n-tog i $n+1$-tog sloja, postoji veza koja ide iz perceptrona u n-tom sloju do perceptrona u $n+1$-tom sloju. Svaka od tih veza ima svoju težinu, a svaki perceptron svoj prag. Sažimamo ta razmatranja u sljedeću definiciju.

Definicija 1.2.1 (Višeslojni perceptron)**.** *Višeslojni perceptron je skup funkcija*

$$\mathrm{MLP}_{\mathbf{W},\mathbf{b},\tau} : \mathbb{R}^n \to \mathbb{R}^m,$$

gdje s $\mathbf{W}$ *označavamo listu* $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(k)}$*, matrica težina, takvih da*

$$\mathbf{W}^{(1)} \in \mathbb{R}^{n \times H_1}, \mathbf{W}^{(2)} \in \mathbb{R}^{H_1 \times H_2}, \ldots, \mathbf{W}^{(k)} = \mathbb{R}^{H_{k-1} \times m},$$

a H_i *označava broj perceptrona u* k*-tom sloju. S* $\mathbf{b}$ *označavamo listu*

$$\mathbf{b}^{(1)} \in \mathbb{R}^{H_1}, \ldots, \mathbf{b}^{(k-1)} \in \mathbb{R}^{H_{k-1}}, \mathbf{b}^{(k)} \in \mathbb{R}^m,$$

[2] eng. multilayer perceptron, skraćeno MLP

[3] Napominjemo da je u literaturi nomenklatura broja slojeva vrlo nekonzistentna. Tako bi se višeslojni perceptron s $k = 2$ slojeva mogao smatrati mrežom s jednim slojem, s dva sloja, ili s tri sloja, ovisno brojimo li ulazne i izlazne slojeve.

vektora pragova, dok s τ *označavamo listu* $\tau^{(1)}, \ldots, \tau^{(k)}$*, aktivacijskih funkcije,* $\tau_i : \mathbb{R} \to \mathbb{R}$ *za svaki i.*

Sada dajemo rekurzivnu definiciju funkcije $\mathrm{MLP}_{\mathbf{W},\mathbf{b},\tau}$*. Za ulaz,* $\mathbf{x} = (x_1, \ldots, x_n)$*, izlaz i-te jedinice,* $i = 1, \ldots, H_1$*, prvog sloja perceptrona je dan s*

$$h_i^{(1)} = \tau^{(1)}\left(\sum_j w_{ij}^{(1)} x_j + b_i^{(1)}\right). \tag{1.10}$$

Izlaz i-te jedinice, $i = 1, \ldots, H_\ell$*,* ℓ*-tog sloja perceptrona,* $\ell < k$*, je dan s*

$$h_i^{(\ell)} = \tau^{(\ell)}\left(\sum_j w_{ij}^{(\ell)} h_j^{(\ell-1)} + b_i^{(\ell)}\right), \tag{1.11}$$

dok je izlaz i-te jedinice, $i = 1, \ldots, m$*, zadnjeg sloja dan s*

$$y_i = \tau^{(k)}\left(\sum_j w_{ij}^{(k)} h_j^{(k-1)} + b_i^{(k)}\right), \tag{1.12}$$

gdje definiramo $\mathbf{y} = (y_1, \ldots, y_m)$*. Sa zadanim* $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(k)}$*,* $\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(k)}$*, i* $\tau^{(1)}, \ldots, \tau^{(k)}$ *nam ovako dana rekurzija definira funkciju*

$$\mathrm{MLP}_{\mathbf{W},\mathbf{b},\tau}(\mathbf{x}) := \mathbf{y}.$$

Neformalno, element $w_{ij}^{(\ell)} \in \mathbf{W}^\ell$ označava težinu veze iz i-tog perceptrona u $\ell - 1$-tom sloju perceptrona u j-ti perceptron u ℓ-tom sloju, te u $\mathbf{b}^{(\ell)} = (b_1^{(\ell)}, \ldots, b_i^{(\ell)}, \ldots b_t^{(\ell)})$, $b_i^{(\ell)}$ smatramo pragom i-tog perceptrona u ℓ-tom sloju.

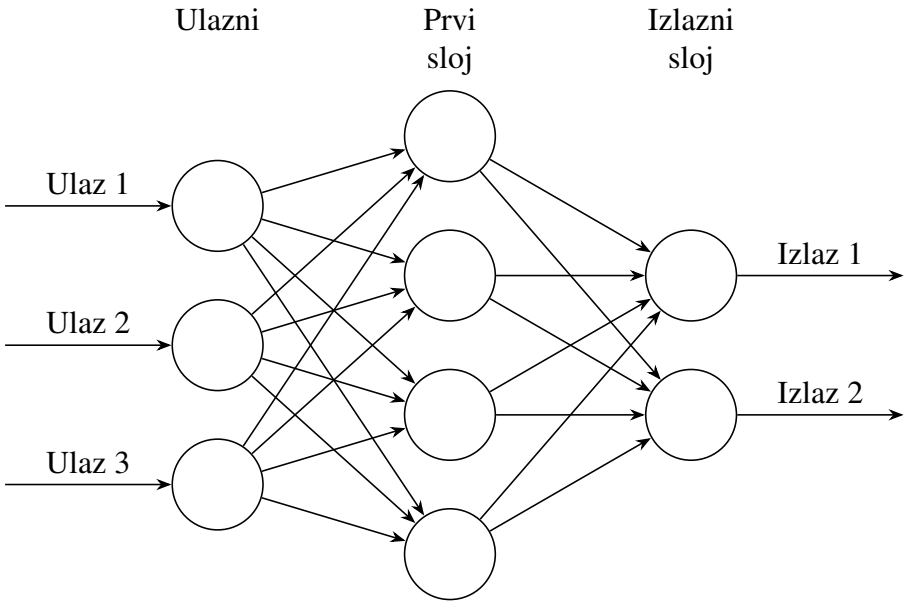
Nadovezujući se na notaciju danu u definiciji 1.2.1, primijetimo da ukoliko definiramo $\mathbf{h}^{(\ell)} = (h_1^\ell, \ldots, h_{H_\ell}^\ell)$, onda 1.10, 1.11, i 1.12 možemo redom zapisati kao

$$\begin{aligned} \mathbf{h}^{(1)} &= \tau^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right), \\ \mathbf{h}^{(\ell)} &= \tau^{(\ell)}\left(\mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right), \ \ell < k, \\ \mathbf{y} &= \tau^{(k)}\left(\mathbf{W}^{(k)}\mathbf{h}^{(k-1)} + \mathbf{b}^{(k)}\right). \end{aligned} \tag{1.13}$$

U tom zapisu aktivacijske funkcije primjenjujemo na svaki koordinatu vektora zasebno.

Na teorijskom nivou, višeslojni perceptron ovisi o matricama težina, vektorima pragova te aktivacijskim funkcijama. U praktičnim primjenama fiksiramo aktvacijske funkcije, dok težine i pragove učimo na temelju podataka. Neke od najčešće korištenih aktivacijskih funkcija su sigmoidna,

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



Slika 1.1: Dijagram neuronske mreže s dimenzijom izlaza 3, 4 perceptrona u prvom sloju, te dva perceptrona u izlaznom sloju, tj. dimenzijom izlaza 2.[4]

tangens hiperbolni,

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

te tzv. rektificirana linearna jedinica[5],

$$\mathrm{ReLU}(z) = \max(0, z).$$

Sljedeći teorem se često citira kao dokaz u prilog ekspresivne moći višeslojnih perceptrona. Možemo ga naći u [5].

Teorem 1.2.2 (Teorem univerzalne aproksimacije)**.** *Neka je $\tau : \mathbb{R} \to \mathbb{R}$ nekonstantna, ograničena i neprekidna funkcija. Neka je I_m n-dimenzionalna jedinična hiperkocka, tj. $[0,1]^m$. Prostor svih realnih neprekidnih funkcija na I_m označavamo $\mathrm{C}(I_m)$. Tada, za bilo koji $\epsilon > 0$ i za bilo koju funkciju $f \in \mathrm{C}(I_m)$, postoje cijeli broj N, realni brojevi v_i, b_i, te*

[4]LaTeX kôd za generiranje dijagrama je modifikacija kôda preuzetog s `https://tex.stackexchange.com/a/354001`.

[5]eng. rectified linear unit, skraćeno ReLU.

realni vektori $w_i \in \mathbb{R}^m$ *za* $i = 1, \ldots, m$*, takvi da za*

$$F(x) = \sum_{i=1}^{N} v_i \tau(w_i \cdot x + b_i) \tag{1.14}$$

imamo

$$|F(x) - f(x)| < \epsilon$$

za sve $x \in I_m$*.*

Primijetimo da funkcija (1.14) odgovora višeslojno perceptronom s dva sloja (gdje brojimo ulazni sloj kao nulti). Aktivacijska funkcija prvog sloja perceptrona je τ, dok je aktvacijska funkcija drugog sloja, koji se sastoji od jednog perceptrona s pragom 0, identiteta.

Koliko god da ovaj rezultat izgledao obećavajuće, ne daje nam nikakvu gornju ogradu za N, broj perceptrona u drugom sloju. Dakle, čak ni uz vrlo efikasnu proceduru za nalaženje težina i praga nas teorem 1.2.2 ne približava nužno nalaženju neuronske mreže koja aproksimira neku funkciju.

Čak ni taj nedostatak teorema univerzalne aproksimacije ne implicra nužno da su "duboke arhitekture", uz mnogo slojeva perceptrona, efektivnije od dva sloja. Bengio i LeCun u [2] daju brojne argumente za to da "većina funkcija koja ima kompaktnu reprezentaciju u dubokoj arhitekturi zahtijeva vrlo velik broj komponenta u plitkoj arhitekturi".

Jedan od temeljnih argumenata za to nalaze u činjenici da se na "plitkoj arhitekturi" može implementirati bilo koja formula propozicijske logike izraziva u disjunktivnoj normalnoj formi. U prvom sloju se izračunaju konjukcije, dok se u drugom izračunaju konjukcije. No u takvoj arhitekturi broj jedinica se može popeti do reda veličine 2^N za N-bitne ulaze. Zato se u praksi digitalni sklopovi, npr. za zbrajanje ili množenje dva broja, grade od više slojeva logičkih sklopova. [2]

Treniranje višeslojnog perceptrona gradijentim spustom i unazadnom propagacijom

Sada dolazimo do ključnog sastojka "magije" neuronskih mreža. Naime, naš model je bezvrijedan osim ako imamo smislen način određivanja težina i pragova. Prije nego što izložimo način nalaženja pogodnih težina i pragova, dopustimo si digresiju oko toga što su "pogodni" parametri.

U okviru statističkog učenja, dan nam je konačan niz ulaza i izlaza, $(x_1, y_1), \ldots, (x_n, y_n)$. Naš model proizvede konačan niz izlaza za dane ulaze $(x_1, \hat{y}_1), \ldots, (x_n, \hat{y}_n)$. Da bismo mogli reći koliko je model "daleko" od stvarne funkcije, uobičajeno imamo neku funkciju gubitka[6] L, koja nam realnim brojem izrazi koliko se dani izlaz $\hat{y}_i$ razlikuje od željenog

[6] eng. loss function

izlaza y_i, $\mathrm{L}(y_i, \hat{y}_i)$, pa onda možemo definirati ukupan gubitak kao prosjek gubitaka preko cijelog uzorka,

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^{n} \mathrm{L}(y_i, \hat{y}_i). \tag{1.15}$$

Primjerice, ako su y_i realni brojevi, onda je česta funkcija gubitka dana s kvadriranim gubitkom, tj.

$$\mathrm{L}(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2. \tag{1.16}$$

Primijetimo da zasad nismo eksplicitno napisali funkciju $\mathcal{L}$ u ovisnosti o parametrima modela. No, budući da izlazi $\hat{y}_i$ ovise samo o parametrima modela, možemo zapisati $\mathcal{L} : \Theta \to \mathbb{R}$, gdje je Θ neki prostor parametara našeg modela. Zapisivanje funkcije gubitka kao funkcije parametara je korisna ideja utoliko što, ako možemo naći gradijent funkcije gubitka, tada možemo iskoristiti "gradijentni spust" do njezinog (lokalnog) minimuma.

Naime, ako imamo funkciju $F : \mathbb{R}^n \to \mathbb{R}$ koja je diferencijabilna oko točke c, te želimo naći njezin minimum oko te točke, smislena strategija je poći u smjeru negativnog gradijenta F na c. Odnosno, ako definiramo $c_0 = c$, te stavimo $c_{n+1} = c_n - \mu_n \nabla F(c_n)$, $\mu_n \in \mathbb{R}$, tada uz određene uvjete na funkciju F i definiciju veličine koraka μ_n, možemo pokazati da, ako je c_0 "dovoljno blizu" točke lokalnog minimuma funkcije F, onda metoda gradijentnog spusta konvergira k toj točki lokalnog minimuma F. [18]

Vratimo se sada definiciji višeslojnog perceptrona 1.2.1. Primijetimo da, ako pretpostavimo da su sve aktivacijske funkcije diferencijabilne, onda koristeći činjenicu da je kompozicija diferencijabilnih funkcija diferencijabilna, nije teško indukcijom pokazati da je i višeslojni perceptron diferencijabilna funkcija – posebno, da je diferencijabilna u ovisnosti o svojim težinama i pragovima. Iz toga slijedi da je $\mathrm{L}(\mathbf{W}, \mathbf{b}) = \mathrm{L}(y_i, \mathrm{MLP}_{\mathbf{W},\mathbf{b}}(x_i))$ diferencijabilna u ovisnosti o težinama i pragovima ako je L diferencijabilna. Dakle, u ostatku ove sekcije pretpostavljamo da su L i aktivacijske funkcije $\tau^{(i)}$, $i = 1, \ldots, k$, diferencijabilne.

Naći ćemo derivacije funkcije gubitka u odnosu na težine i pragove i jedan ulaz perceptrona. Primijetimo da u slučaju kvadratnog gubitka kao u (1.15) možemo, putem linearnosti derivacije, izračunati i gubitak preko cijelog uzorka.

Da bismo našli derivacije po težinama i pragovima, prvo uvodimo Hadamardov produkt dva vektora.

Definicija 1.2.3 (Hadamardov produkt)**.** *Neka su* $\mathbf{x} \in \mathbb{R}^n$ *i* $\mathbf{y} \in \mathbb{R}^n$*, dva vektora,* $\mathbf{x} = (x_1, \ldots, x_n)$*,* $\mathbf{y} = (y_1, \ldots, y_n)$*. Tada je Hadamardov produkt ta dva vektora,* $\mathbf{x} \odot \mathbf{y}$*, dan s jednakošću*

$$\mathbf{x} \odot \mathbf{y} = (x_1 y_1, \ldots, x_n y_n),$$

gdje je operacija $x_i y_i$ *obično množenje dva realna broja.*

Također uvodimo i par korisnih oznaka. U definiciji perceptrona 1.2.1 zapisujemo

$$h_i^{(\ell)} = \tau^{(\ell)}\left(\sum_j w_{ij}^{(\ell)} h_j^{(\ell-1)} + b_i^{(\ell)}\right),$$

gdje uvodimo oznaku $z_i^{(\ell)} = \sum_j w_{ij}^{(\ell)} h_j^{(\ell-1)} + b_i^{(\ell)}$, $\ell < k$, te analogno za zadnji sloj kad je $\ell = k$. Po uzoru po prijašnju notaciju, s $\mathbf{z}^{(\ell)}$ označavamo vektor $(z_1^{(\ell)}, \ldots, z_{H_\ell}^{(\ell)})$, te prateći prijašnju konvenciju da se aktivacijska funkcija primjenjuje koordinatno, imamo $\mathbf{h}^{(\ell)} = \tau^{(\ell)}(\mathbf{z}^{(\ell)})$.

Sada za svaki $\ell = 1, \ldots, k$ i $j = 1, \ldots, H_\ell$ definiramo

$$\delta_j^\ell = \frac{\partial L}{\partial z_j^{(l)}},$$

tzv. *pogrešku* neurona j u sloju ℓ, te opet definiramo $\boldsymbol{\delta}^\ell$ kao vektor koji sadrži sve pogreške u sloju ℓ.

Naša strategija je sada sljedeća. Prvo ćemo pokazati kako izračunati $\boldsymbol{\delta}^\ell$ za $\ell = k$, te onda kako izračunati $\boldsymbol{\delta}^\ell$ za $\ell < k$. Tada ćemo pokazati kako možemo iskoristiti izračun $\boldsymbol{\delta}^\ell$ da izračunamo $\frac{\partial L}{\partial b_j^{(\ell)}}$ i $\frac{\partial L}{\partial w_{ij}^{(\ell)}}$. Pretpostavljamo da su sve aktivacijske funkcije diferencijabilne.

Propozicija 1.2.4. *Koordinate pogreške izlaznog sloja višeslojnog perceptrona, $\boldsymbol{\delta}^k$, su, ako označimo $\tau^{(k)}$ s τ, dane s*

$$\delta_j^k = \frac{\partial L}{\partial y_j}\tau'(z_j^{(k)}), \;\; j = 1, \ldots, m. \tag{1.17}$$

Primijetimo da su sve vrijednosti na desnoj strani 1.17 lako izračunljive. Naime, $z_j^{(k)}$ izračunamo pri samom računanju funkcije MLP, dok $\frac{\partial L}{\partial y_i}$ je npr. u slučaju kvadratnog gubitka rutinski izračunljivo.

Dokaz. Po definiciji imamo da je

$$\delta_j^k = \frac{\partial \mathrm{L}}{\partial z_j^{(k)}}.$$

Želimo to raspisati tako da ovisi parcijalne derivacije koje se pojavljuju ovise o y_i. Prisjetimo se da ako je $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, $g : \mathbb{R}^m \to \mathbb{R}^n$, a $f : \mathbb{R}^n \to \mathbb{R}$, tada, ako je $\mathbf{y} = g(\mathbf{x})$ i $z = f(\mathbf{y})$, po derivaciji kompozicije imamo

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{1.18}$$

Budući da je, s novom uvedenom notacijom, $y_j = \tau(z_j^{(k)})$, onda koristeći (1.18), možemo napisati

$$\delta_j^k = \frac{\partial \mathrm{L}}{\partial z_j^{(k)}} = \sum_{i=1}^{m} \frac{\partial \mathrm{L}}{\partial y_i} \frac{\partial y_i}{\partial z_j^{(k)}}.$$

No primijetimo da je $\frac{\partial y_i}{\partial z_j^{(k)}} = 0$ za $i \neq j$, pošto y_i ne ovisi ni na koji način o $z_j^{(k)}$ kad je $i \neq j$. Dakle, ostaje nam

$$\delta_j^k = \frac{\partial \mathrm{L}}{\partial y_j} \frac{\partial y_j}{\partial z_j^{(k)}}.$$

Zbog $y_j = \tau(z_j^{(k)})$ to je zapravo

$$\delta_j^k = \frac{\partial \mathrm{L}}{\partial y_j} \tau'(z_j^{(k)}),$$

što smo i željeli dobiti. □

Dakle, sada znamo izračunati pogreške zadnjeg sloja perceptrona. U nastavku ćemo pokazati kako iz izračunatih grešaka $\ell + 1$-og sloja možemo izračunati greške ℓ-tog, što onda možemo iskoristiti da izračunamo greške u svim slojevima.

Propozicija 1.2.5. *Vektor pogrešaka za ℓ-ti sloj, $\ell < k$, označavajući $\tau^{(\ell)}$ s τ, je dan s*

$$\boldsymbol{\delta}^\ell = ((\mathbf{W}^{(\ell+1)})^T \boldsymbol{\delta}^{\ell+1} \odot \tau'(\mathbf{z}^{(\ell)}).$$

Dokaz. Koristeći kompoziciju derivacija kao u propoziciji 1.2.4, imamo

$$\delta_j^\ell = \frac{\partial \mathrm{L}}{\partial z_j^{(\ell)}} = \sum_{i=1}^{H_{i+1}} \frac{\partial \mathrm{L}}{\partial z_i^{(\ell+1)}} \frac{\partial z_i^{(\ell+1)}}{\partial z_j^{(\ell)}} \tag{1.19}$$

$$= \sum_{i=1}^{H_{i+1}} \delta_j^{\ell+1} \frac{\partial z_i^{(\ell+1)}}{\partial z_j^{(\ell)}}. \tag{1.20}$$

Prisjećajući se da je

$$z_i^{(\ell+1)} = \sum_{p=1}^{H_{\ell+1}} w_{ip}^{(\ell+1)} \tau^{(\ell)}(z_j^{(\ell)}) + b_p^{(\ell+1)},$$

dobivamo

$$\frac{\partial z_i^{(\ell+1)}}{\partial z_j^{(\ell)}} = w_{ij}^{(\ell+1)} \tau(z_j^{(\ell)}).$$

Uvrštavajući dobiveno u (1.20) imamo

$$\delta_j^\ell = \sum_{i=1}^{H_{\ell+1}} w_{ij}^{(\ell+1)} \delta_i^{\ell+1} \tau'(z_j^{(\ell)}),$$

što je traženi rezultat napisan u koordinatnoj formi. □

Sada dokazujemo sljedeće dvije propozicije, koje nam daju način kako da, pomoću izračunatih pogrešaka, izračunamo derivacije funkcije gubitka u odnosu na težine i pragove.

Propozicija 1.2.6. *Derivacija funkcije gubitka u odnosu na prag j-tog u ℓ-tom sloju je dana s*

$$\frac{\partial \mathrm{L}}{\partial b_j^{(\ell)}} = \delta_j^\ell.$$

Dokaz. Koristeći opet derivaciju kompozicije (1.18) možemo raspisati

$$\delta_j^\ell = \frac{\partial \mathrm{L}}{\partial z_j^{(\ell)}} = \sum_{i=1}^{H_\ell} \frac{\partial \mathrm{L}}{\partial b_i^{(\ell)}} \frac{\partial b_i^{(\ell)}}{\partial z_j^\ell}. \tag{1.21}$$

Pošto $b_i^{(\ell)}$ ovisi o $z_j^{(\ell)}$ samo kad je $i = j$, ova suma se reducira na

$$\delta_j^\ell = \frac{\partial \mathrm{L}}{\partial z_j^{(\ell)}} = \frac{\partial \mathrm{L}}{\partial b_j^{(\ell)}} \frac{\partial b_j^{(\ell)}}{\partial z_j^\ell}. \tag{1.22}$$

Iz $z_j^{(\ell)} = \sum_{i=1}^{H_{(\ell-1)}} w_{ji}^{(\ell)} h_i^{(\ell-1)} + b_j^{(\ell)}$ dobivamo

$$b_j^{(\ell)} = z_j^{(\ell)} - \sum_{i=1}^{H_{\ell-1}} w_{ji}^{(\ell)} h_i^{(\ell-1)}$$

iz čega slijedi

$$\frac{\partial b_j^{(\ell)}}{\partial z_j^{(\ell)}} = 1. \tag{1.23}$$

Uvrštavajući sada (1.23) u (1.22) dobijemo

$$\frac{\partial \mathrm{L}}{\partial b_j^{(\ell)}} = \delta_j^\ell,$$

što smo i željeli. □

Propozicija 1.2.7. *Derivacija funkcije gubitka u odnosu na težinu između i-tog perceptrona u $\ell-1$-om sloju i j-tog perceptrona u ℓ-tom sloju, tj. $w_{ij}^{(\ell)} \in \mathbf{W}^{(\ell)}$, je dan s*

$$\frac{\partial \mathrm{L}}{w_{ij}^{(\ell)}} = h_j^{(\ell-1)} \delta_j^{\ell}.$$

Dokaz. Opet koristeći derivaciju kompozicije (1.18) imamo

$$\frac{\partial \mathrm{L}}{w_{ij}^{\ell}} = \sum_{k=1}^{H_\ell} \frac{\partial \mathrm{L}}{\partial z_k^{(\ell)}} \frac{\partial z_k^{(\ell)}}{\partial w_{ij}^{(\ell)}}. \tag{1.24}$$

Budući da $w_{ij}^{(\ell)}$ ima utjecaj jedino na $z_j^{(\ell)}$, suma se opet svodi na

$$\frac{\partial \mathrm{L}}{w_{ij}^{\ell}} = \frac{\partial \mathrm{L}}{\partial z_j^{(\ell)}} \frac{\partial z_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} = \delta_j^{\ell} \frac{\partial z_j^{(\ell)}}{\partial w_{ij}^{(\ell)}}, \tag{1.25}$$

gdje druga jednakost slijedi iz definicije δ_j^{ℓ}. Sada iz $z_j^{(\ell)} = \sum_{i=1}^{H_{(\ell-1)}} w_{ji}^{(\ell)} h_i^{(\ell-1)} + b_j^{(\ell)}$ odmah dobivamo

$$\frac{\partial z_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} = h_i^{(\ell-1)}. \tag{1.26}$$

Sada kombinirajući (1.25) i (1.26) dobivamo traženi rezultat. □

Sada, uz prije opisani gradijentni spust, imamo sve sastojke potrebne za treniranje višeslojnih perceptrona. Iskažimo, za ilustraciju, precizno algoritam učenja putem unazadne propagacije i gradijentno spusta kada imamo neki konačan broj primjera za "učenje", te želimo za sve njih simultano ažurirati težine i pragove.

U algoritmu pretpostavljamo da smo već na neki način inicijalizirali težine i pragove u našem višeslojnom perceptronu.

Definicija 1.2.8 (Algoritam za treniranje višeslojnog perceptrona)**.**

1. *Ulaz: niz $(x_1, y_1), \ldots, (x_n, y_n)$ i realni broj $\eta > 0$, te višeslojni perceptron* MLP *s k slojeva, te funkcija gubitka* L*.*

2. *Za svaki i od* 1 *do n:*

   a) *Izračunaj* $\mathrm{MLP}(x_i) = \hat{y}_i$*. Prilikom računanja, spremi vrijednosti $\mathbf{z}^{\ell}$ za $\ell = 1, \ldots, k$, te $\mathbf{h}^{(\ell)}$ za $\ell = 1, \ldots, k-1$, gdje stavi $\mathbf{h}^{(\ell),i} := \mathbf{h}^{(\ell)}$*

   b) *Izračunaj izlazne pogreške i označi ih s $\delta^{k,i} = (\nabla_y \mathrm{L}) \odot \tau^{(k)}(\mathbf{z}^{(k)})$.*

c) Za svaki $\ell = k-1, k-2, \ldots, 1$*, stavi* $\tau = \tau^{(\ell)}$*, izračunaj greške* ℓ*-tog sloja i označi ih s* $\delta^{\ell,i} = ((W^{(k+1)})^T \delta^{\ell+1,i}) \odot \tau'(\mathbf{z}^{(\ell)})$*.*

3. Za svaki $\ell = k, k-1, \ldots, 1$ *stavi*

$$\mathbf{W}^\ell := \mathbf{W}^\ell - \frac{\eta}{n} \sum_{i=1}^{n} \delta^{\ell,i} (\mathbf{h}^{(\ell),i})^T$$

i

$$\mathbf{b}^\ell := \mathbf{b}^\ell - \frac{\eta}{n} \sum_{i=1}^{n} \delta^{\ell,i}.$$

Korisnost unazadne propagacije ne staje tu, već se iste ideje mogu iskoristiti u puno širem kontekstu. Naš višeslojni perceptron je neuronska mreža specifična arhitektura, gdje imamo k slojeva perceptrona gdje su svi neuroni iz jednog sloja povezani sa svim neurona iz sljedećeg sloja. Moguće je, na potpuno analogan način, definirati razne arhitekture, recimo arhitekture gdje neuroni nisu nužno svi povezani s onima u sljedećem sloju, ili arhitekture u kojima neuroni uopće nisu poredani u slojeve.

Nije teško vidjeti kako bi se iste ideje, specifično rekurzivno računanje od izlaznih neurona te korištenje derivacije kompozicije, mogle primijeniti na bilo koju arhitekture unaprijednih neuronskih mreža. Razvoj općenitog algoritma za unazadnu propagaciju se može naći u [8].

Razlog zašto je unazadna propagacija za unaprijedne neuronske mreže važna za rekurentne neuronske mreže jest taj da najčešći način treniranja rekurentnih neuronskih mreža, tzv. *unazadna propagacija kroz vrijeme*, koristi "običnu" unazadnu propagaciju na "odrolanoj" rekurentnoj neuronskoj mreži.

Poglavlje 2

Rekurentne neuronske mreže

2.1 Uvod u rekurentne neuronske mreže

Govoreći vrlo općenito, rekurentne neuronske mreže su neuronske mreže koje u sebi sadržavaju petlju. Pošto je taj opis previše općenit, u ovom radu ćemo promatrati nekoliko osnovnih vrsta rekurentnih neuronskih mreža.

Najjednostavniji primjer koji ćemo promatrati, "obična" rekurentna neuronska mreža[1] s jednim skrivenim slojem, može se konceptualizirati kao dvoslojni perceptron, gdje u prvom skrivenom sloju imamo usmjerenu vezu iz svakog perceptrona u sve druge perceptrone u tom sloju. Ta veza pak, neformalno govoreći, ne povezuje ulazni i izlazni sloj, već povezuje uzastopne vremenske korake.

Dakle, u rekurentnim neuronskim mrežama aktivacije ne putuju samo iz smjera ulaznog do izlaznog sloja, već one dolaze i iz prošlog vremenskog koraka. To omogućava rekurentnim neuronskim mrežama da održe određeno pamćenje na povijest niza, ali praksi rekurentne neuronske mreže imaju određene probleme s učenjem dalekosežnih veza. Promotrit ćemo neke razloge zašto je tako, te razmotriti neka rješenja.

Jedno od rješenja je dano *long short-term memory* (LSTM) mrežama, modifikacijom rekurentne neuronske mreže zamijenjivanjem perceptrona s "LSTM jedinicom" koja "pamti" informacije proizvoljno dugo. To je omogućeno zbog bogate strukture "LSTM jedinice" koja ima, također istrenira, sklopove za "ulaz u jedinicu", za "zaboravljanje", te za "izlaz iz jedinice".

[1] U engleskoj literaturi se često upotrebljava naziv *vanilla reccurent network* kako bi se napravila distinkcija s sofisticiranijim arhitekturama. Kako to običava i u tim tekstovima, mi ćemo dalje u tekstu "običnu rekurentnu neuronsku mrežu" zvati samo "rekurentna neuronska mreža".

2.2 "Obične" rekurentne neuronske mreže

Rekurentna neuronska mreža je, kako smo prije spomenuli, slična višeslojnom perceptronu, te je jedina razlika to što aktivacije neurona stižu i iz prijašnjeg vremenskog koraka, te postoje odgovarajuće težine tih veza. Ovdje ćemo formalizirati i koristiti rekurentnu neuronsku mrežu s dva sloja perceptrona, prateći prijašnju konvenciju da je drugi sloj izlazni, ali bi se analogne definicije mogle dati i za više slojeva od dva. Više o "dubokim" rekurentnim neuronskim mrežama se može naći u [8].

Mogli bismo rekurentnu neuronsku mrežu formalizirati na više načina. Definirat ćemo je kao funkciju na svim konačnim nizovima realnih vektora fiksne dimenzijie. Specifično, za $n \in \mathbb{N}$, s $\mathrm{FinSeq}(\mathbb{R}^n)$ označimo skup svih konačnih nizova realnih n-dimenzionalnih vektora. Niz $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}$ označavamo s $(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)})$.

Definicija 2.2.1 (Rekurentna neuronska mreža)**.** *Rekurentna neuronska mreža s $H \in \mathbb{N}$ jedinica je skup funkcija*

$$\mathrm{RNN}_{\mathbf{W}_{yh},\mathbf{W}_{hx},\mathbf{W}_{hh},\mathbf{b}_h,\mathbf{b}_y,\tau_h,\tau_y,\mathbf{h}^{(0)}} : \mathrm{FinSeq}(\mathbb{R}^n) \to \mathrm{FinSeq}(\mathbb{R}^m),$$

gdje su

$$\mathbf{W}_{hy} \in \mathbb{R}^{m\times H}, \mathbf{W}_{hx} \in \mathbb{R}^{H\times n}, \mathbf{W}_{hh} \in \mathbb{R}^{H\times H}$$

matrice težina,

$$\mathbf{b}_h \in \mathbb{R}^H, \mathbf{b}_y \in \mathbb{R}^m,$$

vektori pragova, $\tau_h : \mathbb{R} \to \mathbb{R}$ i $\tau_y : \mathbb{R} \to \mathbb{R}$ aktivacijske funkcije, a $\mathbf{h}^{(0)} \in \mathbb{R}^H$ inicijalno stanje.

Tada funkciju $\mathrm{RNN}_{W_{yh},W_{hx},W_{hh},b_h,b_y,\tau_h,\tau_y,\mathbf{h}^{(0)}}$, za ulaz $(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t_0)})$ definiramo sljedećom petljom. Za $t = 1$ do $t = t_0$ računamo

$$\mathbf{a}^{(t)} = b_h + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{W}_{hx}\mathbf{x}^{(t)}, \tag{2.1}$$

$$\mathbf{h}^{(t)} = \tau_h(\mathbf{a}^{(t)}), \tag{2.2}$$

$$\mathbf{o}^{(t)} = \mathbf{b}_y + \mathbf{W}_{yh}\mathbf{h}^{(t)}, \tag{2.3}$$

$$\hat{\mathbf{y}}^{(t)} = \tau_y(\mathbf{o}^{(t)}), \tag{2.4}$$

gdje onda imamo

$$\mathrm{RNN}_{\mathbf{W}_{yh},\mathbf{W}_{hx},\mathbf{W}_{hh},\mathbf{b}_h,\mathbf{b}_y,\tau_h,\tau_y,\mathbf{h}^{(0)}}((\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t_0)})) := (\hat{\mathbf{y}}^{(1)}, \ldots, \hat{\mathbf{y}}^{(t_0)}). \tag{2.5}$$

Primijetimo da su matrice $\mathbf{W}_{yh}$ i $\mathbf{W}_{hx}$ igraju analognu ulogu kao matrice težina iz definicije višeslojnog perceptrona, dok je esencijalna novina matrica $\mathbf{W}_{hh}$, koja nam daje težine veza između vremena $t-1$ i t. Vektor $\mathbf{h}^{(t)}$ označava, dakle, ono što mreža "zapamti" od

niza do vremena t, a matrica $\mathbf{W}_{hh}$ daje koliku težinu dajemo određenim komponentama "pamćenja".

Napomenimo da naša definicija daje rekurentnu neuronsku mrežu koja mapira ulazni niz u izlazni niz iste duljine. Primijetimo da bismo jednostavno modifikacijom mogli prilagoditi našu mrežu da bude funkcija ulaznog niza i daje samo jedan vektor kao izlaz, onaj koji bi po definiciji 2.2.1 bio zadnji u nizu, odnosno da (2.5) modificiramo na sljedeći način:

$$\mathrm{RNN}_{\mathbf{W}_{yh},\mathbf{W}_{hx},\mathbf{W}_{hh},\mathbf{b}_h,\mathbf{b}_y,\tau_h,\tau_y,\mathbf{h}^{(0)}}((\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(t_0)})) = \hat{\mathbf{y}}^{(t_0)}. \tag{2.6}$$

Tako bismo mogli dobiti preslikavanje mnogo-na-jedan.

U nekim slučajevima, ulaz je fiksne duljine, dok je izlaz varijabilne duljine. Jedan takav primjer je labeliranje slika, gdje bismo željeli slikama dimenzija 100×100 piksela pridružiti rečenicu koja ih opisuje. Postoji više načina da to napravimo, ali primijetimo da u slučaju da su nam svi ulazi, recimo, slike dimenzije 100×100 s jednim brojem pridruženim svakom pikselu, onda to možemo prikazati kao jedan vektor, $\mathbf{x}$, fiksne duljine $100 \cdot 100$. Jedan dio način na koji bismo izveli mapiranje $\mathbf{x}$ na izlaz varijabilne duljine je taj da uvedemo nove težinsku matricu $\mathbf{R}$, koju također "učimo". Tada računamo izlaze (modificirane, za modifikacije vidi [8]) rekurentne neuronske mreže s parametrima θ, RNN_θ,

$$\begin{aligned}
\mathrm{RNN}_\theta((\mathbf{x}^T\mathbf{R})) &= \hat{\mathbf{y}}^{(1)},\\
\mathrm{RNN}_\theta((\mathbf{x}^T\mathbf{R},\mathbf{x}^T\mathbf{R})) &= \hat{\mathbf{y}}^{(2)},\\
\mathrm{RNN}_\theta((\mathbf{x}^T\mathbf{R},\mathbf{x}^T\mathbf{R},\mathbf{x}^T\mathbf{R})) &= \hat{\mathbf{y}}^{(3)}\\
&\vdots
\end{aligned}$$

sve dok za ne dobijemo $t \in \mathbb{N}$ takav da je $\hat{\mathbf{y}}^{(t)}$ jednako nekom stanju koje identificiramo kao kraj niza. Recimo, ako generiramo rečenicu, onda bi to mogla biti točka.

Zahtjevniji slučaj je onaj u kojem imamo preslikavanje mnogo-na-mnogo u kojem su i ulaz i izlaz varijabilne duljine. Primjerice, takvi slučajevi se pojavljuju u prevođenju teksta, prepoznavanju govora, i odgovaranju odgovora.

Unatoč tome što rekurentne neuronske mreže vuku korijene iz 80tih godina prošlog stoljeća, tek su 2014. Cho i suradnici, te neovisno Sutskever i suradnici, predložili arhitekturu koja je sposobna za tretiranje tog slučaja. Dapače, Sutskever su s tom arhitekturom dobili *state-of-the-art* rezultate u prevođenju teksta.[8]

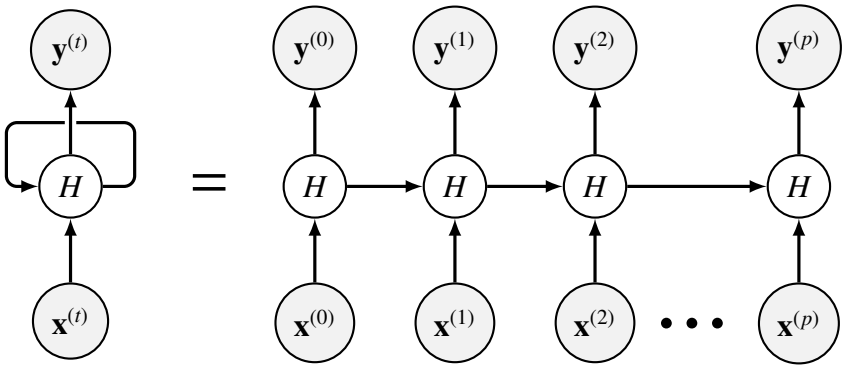
Ključna ideja je da se istreniraju dvije rekurentne neuronske mreže. Jedna rekurentna neuronska mreža, koju zovemo *ulazna*, procesuira niz. Njezin izlaz je takozvani kontekst C, koji je obično neka jednostavna funkcija njezinog zadnjeg skrivenog stanja (u našoj notaciji, za niz duljine n_x to bi bio vektor $\mathbf{h}^{(n_x)}$). Kontekst C smatramo vektorom koji na neki način daje "sažetak" ulaznog niza $\mathbf{X} = (\mathbf{x}^{(0)},\ldots,\mathbf{x}^{(n_x)})$.

Tada druga rekurentna neuronska mreža, *izlazna* mreža, uzima C koja svoj ulaz i na temelju njega generira izlaz $\mathbf{Y} = (\mathbf{y}^{(0)},\ldots,\mathbf{y}^{(n_y)})$, koji može biti varijabilne duljine. Dakle,

pošto općenito ne mora biti $n_x \neq n_y$, ovakva arhitektura se može koristiti za mnogo-na-mnogo preslikavanja s varijabilnim duljinama ulaza i izlaza.

Treniranje rekurentnih neuronskih mreža

Postoji više algoritama koji se koriste za treniranje rekurentnih neuronskih mreža, od kojih su dva najpoznatija *rekurentno učenje u stvarnom vremenu*[2] i *unazadna propagacija kroz vrijeme*. Mi se fokusiramo na drugi algoritam zato što je konceptualno jednostavniji i računski brži [11]. Unazadna propagacija je, kako smo prije ustvrdili, samo uobičajena propagacija primjenjena na 'odrolani' graf rekurentne mreže. Dijagram odrolavanja za našu mrežu možemo vidjeti na slici 2.1. Na tako "odrolanu" rekurentnu neuronsku mrežu



Slika 2.1: Lijevo je graf rekurentne neuronske mreže, desno je "odrolani" graf za tu istu neuronsku mrežu za niz duljine p.[3]

možemo primijeniti klasičnu unazadnu propagaciju, budući da nemamo petlji.

Skicirat ćemo izvođenje jednadžbi unazadne propagacije pod nekim specijalnim uvjetima. Prvo, pretpostavimo da je naša mreža "probabilistički klasifikator" svakog elementa u nizu, odnosno da je izlaz u trenutku t vektor gdje je prva komponenta vjerojatnost prve klase, druga komponenta vjerojatnost druge klase, itd.

Tada je prirodan izbor za τ_y dan s softmax funkcijom, koja je definirana kao

$$\text{softmax}(z_1, \ldots, z_n) = \left(\frac{e^{z_1}}{\sum_{i=1}^{n} e^{z_i}}, \ldots, \frac{e^{z_n}}{\sum_{i=1}^{n} e^{z_i}} \right).$$

Za prvu funkciju aktivacije τ_h, uzimamo tangens hiperbolni. Također, pretpostavljamo da je funkcija gubitka negativna log-vjerodostojnost. Recimo, ako imamo niz $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}$ sa

[2] eng. real time recurrent learning

[3] LaTeX kôd za generiranje dijagrama je modifikacija kôda preuzetog s `https://tex.stackexchange.com/a/494148`.

stvarnim klasama danima s $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(k)}$ onda je gubitak dan s

$$L(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(k)}, \hat{\mathbf{y}}^{(1)}, \ldots, \hat{\mathbf{y}}^{(k)}) = -\sum_{t=1}^{k} \log p_{\text{model}} \left(\mathbf{y}^{(t)} \mid \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}\} \right),$$

gdje $p_{\text{model}}(\mathbf{y}^{(t)} \mid \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}\})$ iščitavamo iz koordinate za klasu $\mathbf{y}^{(t)}$ u izlazu modela $\hat{\mathbf{y}}^{(t)}$. Također, s $L^{(t)}$ označavamo gubitak samo u vremenu t Tada, prvo primijetimo da, po svojstvima softmax funkcije, imamo

$$\frac{\partial L}{\partial L^{(t)}} = 1. \tag{2.7}$$

Ono što također možemo odmah izračunati jest gradijent od L u ovisnosti o izlazima $\mathbf{o}^{(t)}$ (vidi definiciju 2.2.1), gdje za svaki vremenski korak t i $i = 1, \ldots, m$, imamo

$$\begin{aligned}
(\nabla_{\mathbf{o}^{(t)}} L)_i &= \frac{\partial L}{\partial o_i^{(t)}} \\
&= \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} \\
&= \hat{y}_i^{(t)} - \mathbf{1}_{\{i=y^{(t)}\}}
\end{aligned}$$

Sada opet radimo rekurzivno, od kraja mreže. Na zadnjem vremenskom koraku $t = k$, nakon $\mathbf{o}^{(k)}$ jedini direktni prethodnik jest $\mathbf{h}^{(k)}$, tako da je njegov gradijent jednostavan

$$\nabla_{\mathbf{h}^{(k)}} L = (\mathbf{W}_{yh})^T \nabla_{\mathbf{o}^{(k)}} L.$$

Sada analogno nastavimo kretanje "niz prošlost", pa pošto za $t < k$ $\mathbf{h}^{(t)}$ ima direktne nasljednike samo $\mathbf{o}^{(t)}$ i $\mathbf{h}^{(t+1)}$, njegov gradijent je dan s

$$\begin{aligned}
\nabla_{\mathbf{h}^{(t)}} L &= \left(\frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}} \right)^T (\nabla_{\mathbf{h}^{(t+1)}} L) + \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \right)^T (\nabla_{\mathbf{o}^{(t)}} L) \\
&= \mathbf{W}_{hh}^T \operatorname{diag}\left(1 - \left(\mathbf{h}^{(t+1)} \right)^2 \right) (\nabla_{\mathbf{h}^{(t+1)}} L) + \mathbf{W}_{yh}^T (\nabla_{\mathbf{o}^{(t)}} L),
\end{aligned}$$

gdje $\operatorname{diag}\left(1 - \left(\mathbf{h}^{(t+1)} \right)^2 \right)$ označava dijagnonalni matricu u kojoj je na i-tom mjestu na dijagonali element $1 - (h_i^{(t+1)})^2$. To je jakobijan tangensa hiperbolnog asociran s i-tim 'perceptronom' u skrivenom sloju, u vremenu $t + 1$.

Sad kad smo izračunali gradijente u ovisnosti o stanjima $\mathbf{h}^{(t)}$, možemo izračunati, putem unazadne propagacije, gradijente u ovisnosti o parametrima. [8]. Da bismo iskazali jednadžbe koje daju te gradijente, uvodimo notaciju $\nabla_{\mathbf{W}^{(t)}}$, za $W \in \{\mathbf{W}_{hx}, \mathbf{W}_{hh}, \mathbf{W}_{yh}\}$, da

označimo gradijent u odnosu na $\mathbf{W}$ do vremena t. Sada imamo, gdje pod sumama po t mislimo od 1 to k, dok po i mislimo po od 1 do m, gdje je m dimenzija izlaza rekurentne neuronske mreže:

$$\begin{aligned}
\nabla_{\mathbf{b}_y} L &= \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{b}_y}\right)^T \nabla_{\mathbf{o}^{(t)}} L \\
&= \sum_t \nabla_{\mathbf{o}^{(t)}} L, \\
\nabla_{\mathbf{b}_h} L &= \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}_h^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}} L \\
&= \sum_t \operatorname{diag}\left(1 - \left(\mathbf{h}^{(t)}\right)^2\right) \nabla_{\mathbf{h}^{(t)}} L, \\
\nabla_{\mathbf{W}_{yh}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_{\mathbf{W}_{yh}} o_i^{(t)} \\
&= \sum_t (\nabla_{\mathbf{o}^{(t)}} L)(\mathbf{h}^{(t)})^T, \\
\nabla_{\mathbf{W}_{hh}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\mathbf{W}_{hh}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\mathbf{h}^{(t)}\right)^2\right) (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{h}(^{(t-1)})^T, \\
\nabla_{\mathbf{W}_{hx}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\mathbf{W}_{hx}^{(t)}} h_i^{(t)} \\
&= \sum_t \operatorname{diag}\left(1 - \left(\mathbf{h}^{(t)}\right)^2\right) (\nabla_{\mathbf{h}^{(t)}} L)(\mathbf{x}^{(t)})^T.
\end{aligned}$$

Problem iščezavanja ili eksplodiranja gradijenata

Najveći problem treniranja rekurentnih neuronskih mreža je to što gradijenti koji su propagirani kroz mnogo koraka imaju tendenciju ili "iščeznuti' (tj. otići vrlo blizu nule) ili eksplodirati (tj. postati jako veliki).

Promotrimo sljedeću rekurenciju:

$$\mathbf{h}^{(t)} = \mathbf{W}^T \mathbf{h}^{(t-1)}.$$

Ona je zapravo primjer rekurentne mreže s identitetama kao aktivacijskim funkcijama i bez ulaza, te sa svim parametrima 0 osim ne-nul matrice koja množi prošlo stanje. Pretpostavimo nadalje da je $\mathbf{W}$ simetrična matirca. Ista rekurencija se očito može napisati na

sljedeći način:

$$\mathbf{h}^{(t)} = (\mathbf{W}^t)^T \mathbf{h}^{(0)}.$$

Sada, budući da je $\mathbf{W}$ simetrična, ona ima dekompoziciju

$$\mathbf{W} = Q\Lambda Q^T,$$

gdje je Q ortogonalna a Λ je realna dijagonalna matrica koja na dijagonali ima svojstvene vrijednosti od $\mathbf{W}$. Tada, iz ortogonalnosti matrice Q slijedi

$$\mathbf{h}^{(t)} = Q\Lambda^t Q^T \mathbf{h}^{(0)}.$$

Pošto su Q^T, Q i $\mathbf{h}^{(0)}$ samo "konstante", dugoročno vrijednost niza ovisi pretežito o Λ^t. No budući da, ako dijagonalnu matricu potenciramo nekim prirodnim brojem, onda je rezultat opet dijagonalna matrica gdje je na svaki element primijenjena ta potencija. Vidimo da kako t teži u beskonačnost oni elementi dijagonale koji odgovaraju svojstvenim vrijednostima s apsolutnom vrijednosti manjom od 1 će ići u nulu, dok oni koji odgovaraju svojstvenim vrijednostima s apsolutnom vrijednosti većom od 1 ići u (pozitivnu ili negativnu) beskonačnost.

Pokazali smo postojanje fenomena iščezavanja ili eksplodiranja, u ovom slučaju za skriveno stanje $\mathbf{h}^{(t)}$, na vrlo jednostavnom primjeru, ali je empirijski utvrđeno da se to često događa prilikom treniranja rekurentnih neuronskih mreža. Naime, da bi model "naučio" dalekosežne ovisnosti, on "prirodno" mora ući u parametarski prostor gdje gradijenti teže ka nuli. Eksperimenti su pokazali da, ukoliko postoje dalekosežne ovisnosti, tada treniranje rekurentnih neuronskih mreža putem gradijentnog spusta postane nemoguće već za nizova duljine 10 ili 20. [8]

Razni načini za rješavanje tog problema su iskušani, a jedan od njih predstavljamo kasnije u ovom poglavlju: LSTM mreže.

Dvosmjerne rekurentne neuronske mreže

Već smo naveli prevođenje i prepoznavanje govora kao dvije domene za koje su rekurentne neuronske mreže posebno podobne. U primjeru prevođenja bi, u vremenskog koraku n, ulaz neuronske mreže bila n-ta riječ (odnosno, neki način na koji smo kodirali tu riječ kao broj ili vektor) koja se pojavljuje u nekom nizu. Očiti nedostatak koji se pojavljuje jest taj što je za prevođenje važno ne samo koje se riječi pojavljuju prije neke riječi, već i one koje se pojavljuju poslije. Na sličan način je to važno i za prepoznavanje govora. Iako se na prvi pogled čini da prepoznavanje govora ima temporalnu komponentu, u praksi se prepoznavanje obično obavlja nakon što je cijeli unos procesuiran.

Rješenje koje bi nam moglo prvo pasti na pamet je dodavanje nekog vremenskog perioda budućnosti koju uvijek gledamo kada predviđamo. No, zbog tada zadanog fiksnog perioda konteksta to rješenje nije idealno.

Dvosmjerne rekurentne neuronske mreže, pak, nude rješenje koje nema ograničeni fiksni period budućnosti. Pretpostavimo da imamo niz, ili skup nizova takve forme, $\mathbf{x}^{(0)}$, ..., $\mathbf{x}^{(t)}$. Kao što možemo istrenirati rekurentnu neuronsku mrežu na skupu takvih nizova, tako možemo istrenirati i rekurentnu neuronsku mrežu na nizovima $\mathbf{x}^{(t)}, \ldots, \mathbf{x}^{(0)}$, dakle, idući u suprotnom smjeru.

Dvosmjerne rekurentne neuronske mreže iskorištavaju ovu ideju tako da kombiniraju dvije takve rekurentne neuronske mreže u jednu. Preciznije, ako s $\mathbf{h}^{(t)}$ označimo skriveno stanje u trenutku t rekurentne neuronske mreže koja ide "naprijed", a s $\mathbf{g}^{(t)}$ označimo skriveno stanje u trenutku t rekurentne neuronske mreže koja ide "natrag", onda je izlaz $\mathbf{o}^{(t)}$ definiran kao neka funkcija s argumentima $\mathbf{h}^{(t)}$ i $\mathbf{g}^{(t)}$. Na taj način u bilo kojem vremenu t izlaz ovisi o cijeloj povijesti i o cijeloj budućnosti niza.

Ideja dvosmjerne rekurentne neuronske mreže se može na prirodan način generalizirati na dvo-dimenzionalan ulaz, poput slika. Tada, umjesto dvije rekurentne neuronske mreže, imamo četiri, od kojih jedna ide gore, druga ide dolje, treća ide lijevo, a četvrta ide desno. Tada, za svaku točku (i, j) mreže ulaza, "četvoro-smjerna" rekurentna neuronska mreža ima izlaz $\mathbf{o}_{i,j}$.

2.3 LSTM mreže

LTSM mrežu su 1997. uveli Hochreiter i Schmidhuber kao način rješavanja problema iščezavajućeg i eksplodirajućeg gradijenta [14]. Model je sličan običnog rekurentnoj neuronskoj mreži, ali umjesto perceptrona, u prvom sloju rekurentne neuronske mreže ima takozvanu *LTSM ćeliju*. LSTM ćelija se može smatrati kao na određeni način "podmrežom" koja pak ima svoju rekurenciju. Svaka takva podmreža ima svoj sustav *kontrolirajućih sklopova*, koji se sastoji od *ulaznog sklopa*, *sklopa za zaboravljanje*, te *izlaznog sklopa*. Ti sklopovi zajedno reguliraju tok informacija u i iz takozvanog *internog stanja ćelije*.

Prije nego što uđemo u intuiciju toga kako LSTM mreža radi, iskažimo preciznu definiciju. Napominjemo da ima više srodnih verzija LSTM mreža; primjerice, neke ne sadrže vrata za zaboravljanje, itd. Verzija koju mi predstavljamo se može naći u [19].

Definicija 2.3.1. *LSTM mreža je skup funkcija*

$$\mathrm{LTSM}_{\mathbf{W},N,b_f,\mathbf{h}^{(0)},\mathbf{s}^{(0)},\mathbf{q}^{(0)},\tau} : \mathrm{FinSeq}(\mathbb{R}^n) \to \mathrm{FinSeq}(\mathbb{R}^m), \tag{2.8}$$

koje zadovoljavaju sljedeća svojstva. Prvo, $\mathbf{W}$ *je lista matrica težina*

$$\mathbf{W}_{hh}, \mathbf{W}_{hx}, \mathbf{W}_{hq}, \mathbf{W}_{igh}, \mathbf{W}_{igx}, \mathbf{W}_{igq}, \mathbf{W}_{ih}, \mathbf{W}_{ix}, \mathbf{W}_{iq},$$
$$\mathbf{W}_{oh}, \mathbf{W}_{ox}, \mathbf{W}_{oq}, \mathbf{W}_{fh}, \mathbf{W}_{fx}, \mathbf{W}_{fq}, \mathbf{W}_{yh}, \mathbf{W}_{yq},$$

N je prirodan broj koji zovemo brojem LSTM ćelija, te imamo $b_f, \mathbf{s}^{(0)}, \mathbf{q}^{(0)} \in \mathbb{R}^N$*, funkciju* $\tau : \mathbf{R} \to \mathbf{R}$ *zovemo funkcijama aktivacije, a* $\mathbf{h}^{(0)}$ *je realni vektor dimenzije h. Uz te oznake imamo*

$$\mathbf{W}_{hh} \in \mathbb{R}^{h\times h}, \mathbf{W}_{hx} \in \mathbb{R}^{h\times n}, \mathbf{W}_{hq} \in \mathbb{R}^{h\times N},$$
$$\mathbf{W}_{igh} \in \mathbb{R}^{N\times h}, \mathbf{W}_{igx} \in \mathbb{R}^{N\times n}, \mathbf{W}_{igq} \in \mathbb{R}^{N\times N},$$
$$\mathbf{W}_{ih} \in \mathbb{R}^{N\times h}, \mathbf{W}_{ix} \in \mathbb{R}^{N\times x}, \mathbf{W}_{iq} \in \mathbb{R}^{N\times N},$$
$$\mathbf{W}_{oh} \in \mathbb{R}^{N\times h}, \mathbf{W}_{ox} \in \mathbb{R}^{N\times n}, \mathbf{W}_{oq} \in \mathbb{R}^{N\times N},$$
$$\mathbf{W}_{fh} \in \mathbb{R}^{N\times h}, \mathbf{W}_{fx} \in \mathbb{R}^{N\times n}, \mathbf{W}_{fq} \in \mathbb{R}^{N\times N},$$

i

$$\mathbf{W}_{yh} \in \mathbb{R}^{m\times h}, \mathbf{W}_{yq} \in \mathbb{R}^{m\times M}.$$

Tada funkciju $\mathrm{LTSM}_{\mathbf{W},N,b_f,\mathbf{h}^{(0)},\mathbf{s}^{(0)},\mathbf{q}^{(0)}}$ *zadajemo, za ulaz* $(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t_0)})$*, rekurzivno sa sljedećom petljom. Za* $t = 1$ *do* $t = t_0$*, radimo*

$$\mathbf{h}^{(t)} = \tanh(\mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{W}_{hx}\mathbf{x}^{(t)} + \mathbf{W}_{hq}\mathbf{q}^{(t-1)}, \tag{2.9}$$

$$\mathbf{i}_g^{(t)} = \sigma(\mathbf{W}_{igh}\mathbf{h}^{(t-1)} + \mathbf{W}_{igx}\mathbf{x}^{(t)} + \mathbf{W}_{igq}\mathbf{q}^{(t-1)}), \tag{2.10}$$

$$\mathbf{i}^{(t)} = \tanh(\mathbf{W}_{ih}\mathbf{h}^{(t-1)} + \mathbf{W}_{ix}\mathbf{x}^{(t)} + \mathbf{W}_{iq}\mathbf{q}^{(t-1)}), \tag{2.11}$$

$$\mathbf{o}^{(t)} = \sigma(\mathbf{W}_{oh}\mathbf{h}^{(t-1)} + \mathbf{W}_{ox}\mathbf{x}^{(t)} + \mathbf{W}_{oq}\mathbf{q}^{(t-1)}), \tag{2.12}$$

$$\mathbf{f}^{(t)} = \sigma(\mathbf{W}_{fh}\mathbf{h}^{(t-1)} + \mathbf{W}_{fx}\mathbf{x}^{(t)} + \mathbf{W}_{fq}\mathbf{q}^{(t-1)}), \tag{2.13}$$

$$\mathbf{s}^{(t)} = \mathbf{s}^{(t-1)} \odot \mathbf{f}^{(t)} + \mathbf{i}^{(t)} \odot \mathbf{i}_g^{(t)}, \tag{2.14}$$

$$\mathbf{q}^{(t)} = \mathbf{s}^{(t)} \odot \mathbf{o}^{(t)}, \tag{2.15}$$

$$\mathbf{z}^{(t)} = \tau(\mathbf{W}_{yh}\mathbf{h}^{(t)} + \mathbf{W}_{yq}\mathbf{q}^{(t)}). \tag{2.16}$$

Sada stavljamo

$$\mathrm{LTSM}_{\mathbf{W},N,b_f,\mathbf{h}^{(0)},\mathbf{s}^{(0)},\mathbf{q}^{(0)},g}(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t_0)}) := (\mathbf{z}^{(0)}, \ldots, \mathbf{z}^{(t_0)}). \tag{2.17}$$

Kao korak prema intuiciji o tome što se u LSTM ćeliji događa, navedimo da $\mathbf{i}_g^{(t)}$ zovemo vektorom ulaznog sklopa[4] (u vremenu t), $\mathbf{i}^{(t)}$ vektorom ulaza u memorijske ćelije, $\mathbf{o}^{(t)}$ vektorom izlaznog sklopa, $\mathbf{f}^{(t)}$ vektorom sklopa za zaboravljanje, $\mathbf{s}^{(t)}$ vektorom stanja memorije.

[4] eng. input gates

Dakle, neka je za danu LSTM mrežu j neki broj između 1 i N. Tada s $(\mathbf{i}_g^{(t)})_j$ označavamo j-tu komponentu vektora $\mathbf{i}_g^{(t)}$, te analogno i za druge gore napisane vektore. Tada, po svojstvima tangensa hiperbolnog i sigmoidne funkcije, imamo

$$(\mathbf{i}_g^{(t)})_j \in [0,1],\ (\mathbf{i}^{(t)})_j \in [-1,1],\ (\mathbf{o}^{(t)})_j \in [0,1],\ (\mathbf{f}^{(t)})_j \in [0,1].$$

Promotrimo sada jednakost (2.14), koja opisuje kako ažuriramo memorijskog stanje LSTM mreže kroz vrijeme. Prvi izraz u zbroju, $\mathbf{s}^{(t-1)} \odot \mathbf{f}^{(t)}$, opisuje što "zaboravljamo" iz prošlog stanja. Dakle, $(\mathbf{f}^{(t)})_j$ je realan broj između 0 i 1 koji opisuje u koliko mjeri "zaboravljamo" prošlo stanje j-te LSTM ćelije $(\mathbf{s}^{(t-1)})_j$, gdje $(\mathbf{f}^{(t)})_j = 0$ označava potpuni "zaborav", a $(\mathbf{f}^{(t)})_j = 1$ označava potpuno "pamćenje". Tome pribrajamo $\mathbf{i}^{(t)} \odot \mathbf{i}_g^{(t)}$, što je doprinos novog ulaza, danog s $\mathbf{i}^{(t)}$ reguliranog ulaznim sklopom $\mathbf{i}_g^{(t)}$. Dakle, za j-tu LSTM ćeliju imamo $(\mathbf{i}_g^{(t)})_j \in [0,1]$, gdje $(\mathbf{i}_g^{(t)})_j = 0$ znači da se ulaz $(\mathbf{i}^{(t)})_j$ potpuno "blokira", odnosno ne utječe na stanje $(\mathbf{s}^{(t)})_j$, dok $(\mathbf{i}_g^{(t)})_j = 1$ označava da se ulaz u potpunosti "propušta" u stanje ćelije $(\mathbf{s}^{(t)})_j$.

Kad je memorijsko stanje u vremenu t izračunato, još u (2.15) putem izlaznog sklopa na analogan način reguliramo što će izaći, te to kao i skriveno stanje $\mathbf{h}^{(t)}$ koristimo za izračun izlaza.

2.4 Rekurentne mreže s neprekidnim vremenom

Postoji još mnogo varijanti rekurentnih neuronskih mreža. Ovdje u kratkim crtama i neformalno opisujemo još jednu verziju. Naime, uveli smo rekurentne neuronske mreže kao alat koji pomaže pri modeliranju temporalnih veza, no zasad smo se osvrnuli samo na diskretne vremenske veze. Opišimo sada jedan koncept rekurentne neuronske mreže koji ne pretpostavlja diskretizirano vrijeme.

Hopfieldova rekurentna neuronska mreža je dana s funkcijama $u_1, \ldots, u_n$, $v_1, \ldots, v_n$ i $g_1, \ldots, g_n$, gdje su $u_i : \mathbb{R} \to \mathbb{R}$, $v_i : \mathbb{R} \to \mathbb{R}$ i $g_i : \mathbb{R} \to \mathbb{R}$ za svaki $i = 1, \ldots, n$, realnim konstantama $I_1, \ldots, n$ i T_{ij} gdje je $(i, j) \in \{1, \ldots n\} \times \{1, \ldots, n\}$, te pozitivnim realnim konstantama $C_1, \ldots, C_n, R_1, \ldots, R_n$, sa svojstvom da

$$C_i \frac{du_i(t)}{dt} = -\frac{u_i(t)}{R_i} + \sum_{j=1}^{n} T_{ij} v_i(t) + I_i, \tag{2.18}$$

$$v_i(t) = g_i(u_i(t)), \tag{2.19}$$

za $i = 1, \ldots, n$, gdje je n broj neurona, te $t \geq 0$. Tada $u_i(t)$ smatramo stanje i-tog neurona u trenutku t, a $v_i(t)$ izlaz u trenutku t. g_i su aktivacijske funkcije, za koje uzimamo da su sigmoidalne. Pod tim mislimo da imamo da je svaka funkcija g_i neprekidna i diferencijabilna,

te

$$\begin{aligned}
&\lim_{s\to\pm\infty} g_i(s) = \pm 1\\
&|g_i(s)| \leq 1, \forall s \in \mathbb{R},\\
&0 < g_i'(s) < g_i'(0), \forall s \in R \setminus \{0\},\\
&\lim_{s\to\pm\infty} g_i'(s) = 0.
\end{aligned}$$

Više o ovoj specifičnoj inačici rekurentne mreže, kao i o drugim inačicama s neprekidnim vremenom, se može naći u [21].

Poglavlje 3

Primjene rekurentnih neuronskih mreža

Rekurentne neuronske mreže, u svojim raznim oblicima su, kao što smo tijekom rada navodili, pogodne za razne zadatke. U ovom ćemo poglavlju prvo dati pregled nekih rezultata ostvarenih s rekurentnim neuronskim mrežama, a onda ćemo na nekim jednostavnim primjerima proučiti kako se rekurentne neuronske mreže treniraju i ponašaju.

3.1 Pregled primjena

Graves i suradnici [9] koriste dvosmjernu LSTM mrežu za prepoznavanje teksta napisanog rukom, bez ikakvih ograničenja. Navode skrivene Markovljeve lance kao do tada obećavajuć pristup tom problemu, te ih uspoređuju s, dakle, dvosmjernom LSTM mrežom. U njihovim glavnim rezultatima, skriveni Markovljevi modeli imaju 64.5% točnost u prepoznavanju riječi, dok LSTM mreža ima 74.1% točnost. LSTM mreža svoja prepoznavanja radi na razini samih slova, koje je prepoznala s točnosti 81.8%

Eck i Schmidhuber [6] koriste LSTM mreže za generiranje *Blues* glazbe. Mreža za treniranje koristi akorde i melodiju, te nakon treniranja mreži se daje inicijalna nota ili niz nota, te ona "svojevoljno" nastavlja niz.

Baccouche i suradnici [1] koriste tzv. konvolucijske neuronske mreže za preprocesiranje videa ljudske aktivnosti (npr. hodanje, pljeskanje, mahanje, itd.), te tada LSTM mrežu za njihovu klasifikaciju.

Graves i Schmidhuber [10] koriste rekurentne neuronske mreže za klasifikaciju fonema. U svom radu uspoređuj dvosmjernu LSTM mrežu (točnost 69.8% na testnom skupu), dvosmjernu rekurentnu neuronsku mrežu (točnost 69.0%), LSTM mrežu (točnost 64.6%), višeslojni perceptron (točnost 51.4%), te neke druge varijacije.

Malhotra i suradnici [16] koriste LSTM mreže za detekciju anomalija u vremenskim serijama. Demonstrirali su sposobnost LSTM mreža za detekciju anomalija na četiri skupa podataka: podaci EKG-a u kojima anomalija odgovara pred-ventrikularnoj kontrakciji,

podaci o svemirskom *shuttleu* Marotta, podaci o korištenju eleketrične energije gdje je očekivano ponašanje pet 'vrhova' koji odgovaraju danima tjedna i dva 'dna' koji odgovaraju vikendima, te podaci iz 12 različitih senzora motora pomoću kojih se detektiraju kvarovi.

Mayer i suradnici [17], motivirani potrebama minimalno invazivnih operacija srca, koriste rekurentne neuronske mreže, specifično LSTM mrežu, kako bi robote naučili vezivati čvorove.

Rekurentne neuronske mreže, obično u formi LSTM mreža, su također u znanstvenoj literaturi primijenjene i za diferencijalnu dijagnozu u zdravstvu [3], prevođenje teksta [20], prepoznavanje homologije proteina [13], i mnoge druge stvari. U puno tih slučajeva rekurentne neuronske mreže ostvaruju *state-of-the-art* rezultate, s nerijetko većom točnošću ili računalnom brzinom od trenutnih rješenja.

3.2 Eksperimenti s rekurentnim neuronskim mrežama

Naše eksperimente ćemo provoditi s modulom *keras* u programskom jeziku *Python* [4]. Keras je jedan od najpopularnijih modula za implementaciju neuronskih mreža, te se koristi kako u industriji, tako i za istraživanje.

Eksperiment 1

Prvi zadatak za koji ćemo istrenirati rekurentnu neuronsku mrežu je taj da nauči niz vrijednosti $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$, gdje su svi $x^{(i)} \in \{0, 1\}$, preslikati u niz $y^{(1)}, \ldots, y^{(t)}$ takav da je

$$y^{(t_0)} = \begin{cases} 1 & \text{ako je } x^{(t_0)} = 0, \\ 0 & \text{ako je } x^{(t_0)} = 1. \end{cases}$$

Kako bismo mogli pretpostaviti, dovoljna je vrlo jednostavna rekurentna neuronska mreža, trenirana gradijentni spustom, da "proizvoljno dobro" nauči taj uzorak. Dovoljna su samo dva neurona ($H = 2$ u definiciji 2.2.1). Tada (referirajući se oznakama na definiciju 2.2.1), uz $\tau_h = \text{ReLu}$, $\tau_y = \sigma$, imamo

$$\mathbf{W}_{hx} \approx \begin{bmatrix} 3.64 \\ 0.39 \end{bmatrix},$$

$$\mathbf{W}_{hh} \approx \begin{bmatrix} 0 & 0.14 \\ 0.39 & -1 \end{bmatrix},$$

$$b_h \approx \begin{bmatrix} 0 \\ -0.9 \end{bmatrix},$$

te

$$\mathbf{W}_{yh} \approx \begin{bmatrix} -3.57 & 0.538 \end{bmatrix} \text{ i } b_y \approx 5.1.$$

Napominjemo da smo istrenirali model samo na nizovima čija je duljina izabrana slučajno kao cijeli broj između 10 i 20, no model je robustan i na nizovima puno veće duljine. Primjerice, na slučajno generiramo primjeru duljine 20 je prosječna apsolutna pogreška bila 0.001, dok je na primjeru slučajno generiranog niza duljine 10 tisuća, apsolutna prosječna pogreška je bila 0.003.

Eksperiment 2

Definirajmo sada, za niz $x^{(1)}, \ldots, x^{(t)}$, $x^{(i)} \in \{0, 1\}$ i prirodan broj $\alpha < t$, niz $y^{(1)}, \ldots, y^{(t)}$ dan s

$$y^{(0)} = 0, \ldots, y^{(\alpha)} = 0, y^{(\alpha+1)} = x^{(1)}, \ldots, y^{(t)} = x^{(t-\alpha)}.$$

Treniranje rekurentne neuronske mreže da mapira $x^{(t)}$ na $y^{(t)}$ traži, dakle, "pamćenje" α koraka u prošlost.

Sada za α od 2 do 10 tražimo H_α, najmanji broj neurona (parametar H u definiciji 2.2.1) takav da treningom neuronske mreže na pedeset tisuća primjeraka nizova $x^{(1)}, \ldots, x^{(t)}$, slučajno odabrane duljine od 20 do 30, i odgovarajućih $y^{(1)}, \ldots, y^{(t)}$, konvergira "jako blizu" željenom mapiranju.[1] Napominjemo da naši rezultati krucijalno ovise o proceduri treniranja. Nismo koristili klasični gradijentni spust, već popularnu varijantu zvanu *Adam*, s parametrom *learning rate* jednakim 0.001, o kojoj se više detalja može naći u radu u kojem je uvedena [15]. Aktivacijske funkcije su $\tau_h = \tanh$ i $\tau_y = \sigma$. Treniranje je donekle stohastičkog karaktera, iz kojeg razloga smo za svaku vrijednost α negativan rezultat (tj. da neki broj neurona nije dovoljan) potvrdili pet puta. Sljedeća tablica pokazuje rezultate:

α	2	3	4	5	6	7	8	9	10
H_α	3	4	5	6	7	8	9	10	11

Vidimo da rekurentna neuronska mreža ne treba veliku kompleksnost kako bi "zapamtila" određeni broj koraka unazad. Kao blagu varijaciju na ovaj eskperiment smo ga kombinirali i s prvim eksperimentom, odnosno, za t_0 takv da je $\alpha < t_0 \leq t$, stavili smo $y^{(t_0)} = 1$ ako je $x^{(t_0)} = 0$ i $y^{(t_0)} = 0$ ako je $x^{(t_0)} = 1$. Tada su rezultati bili identični, odnosno ako je određen broj neurona bio dovoljan za pamćenje, onda je bio dovoljan i za to da se nakon tog pamćenja obrnu nule i jedinice.

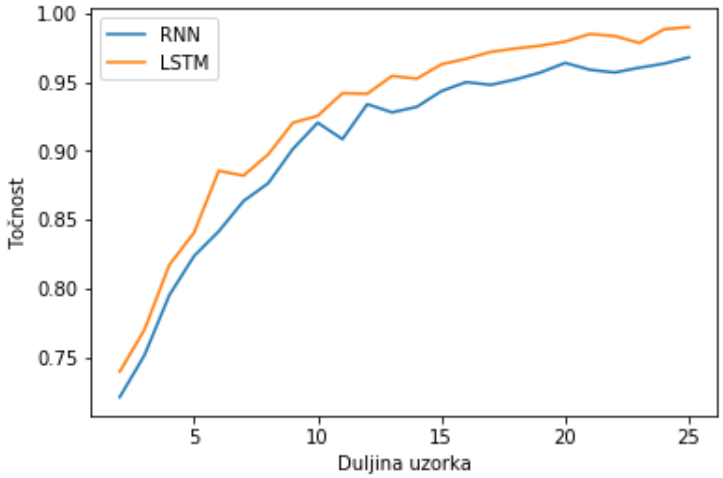
[1] Precizno bismo mogli to iskazati kao zahtjev da funkcija gubitka, binarna unakrsna entropija, padne ispod 0.01.

Eksperiment 3

Za sljedeći eksperiment smo odabrali testirati hoće li rekurentna neuronska mreža, i s kojom točnošću, moći razlikovati uzorke, slučajno odabrane duljine od 2 do 15, generirane ili iz normalne distribucije s očekivanom vrijednosti 0 i standardnom devijacijom 1, ili iz normalne distribucije s očekivanom vrijednosti 0 i standardnom devijacijom 2.

Dvije arhitekture koje smo testirali su bile rekurentna neuronska mreža, s 16 neurona i aktivacijskim funkcijama $\tau_h = \tanh$, $\tau_y = \sigma$, te LSTM mreža s 16 memorijskih ćelija. Nakon treniranja na 60 tisuća uzoraka, duljine uniformno odabrane od 2 do 15, procijenili smo točnost našeg klasifikatora tako što smo, za svaki prirodan broj n od 2 do 25, generirali 1000 uzoraka duljine n iz distribucije $\mathcal{N}(0,1)$ i 1000 uzoraka duljine n iz distribucije $\mathcal{N}(0,2)$[2], te na temelju točnosti rekurentne neuronske mreže na tim uzorcima izračunali empirijsku procjenu točnosti klasifikatora.

Napominjemo da smo pri procjeni točnosti nju testirali i na dužim nizovima od onih na kojima se meža trenirala. Za vrijeme treniranja su joj bili "dani" nizovi duljine od 2 do 15, dok je procjena točnosti napravljena na nizovima duljine od 2 do 25. Dolje priloženi graf pokazuje točnost u ovisnosti o duljini uzorka.



Slika 3.1: Empirijska procjena točnosti klasifikatora (y-os) u ovisnosti o duljini uzorka (x-os).

Vidimo da LSTM mreža na tom zadatku konzistentno ostvaruje bolji rezultat od "obične" rekurentne neuronske mreže. Napomenimo da to ne znači nužnu superiornost LSTM mreže naspram "obične rekurentne mreže", čak ni nužno na tom zadatku. Postoji mnoštvo

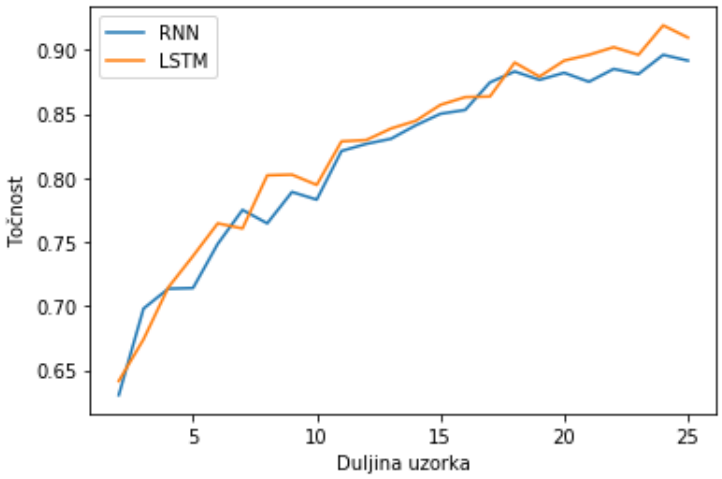
[2]Koristimo notaciju $\mathcal{N}(\mu, \sigma)$ da označimo normalnu distribuciju s očekivanom vrijednosti 0 i standardnom devijacijom σ.

hiperparametara, načina njihovog ugađanja i izabiranja kod neuronskih mreža, pa i stohastičnosti prilikom inicijaliziranja početnog stanja mreže, da bi rigorozni test superiornosti bilo teško, ako ne i nemoguće, provesti.

Eksperiment 4

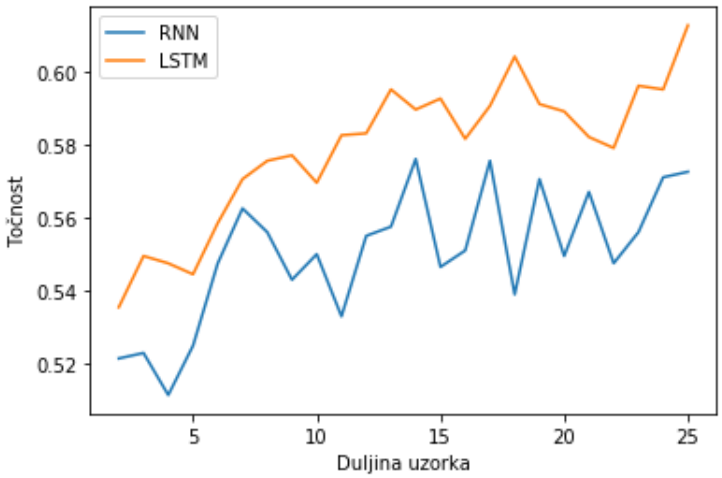
Sada ponavljamo treći eksperiment s raznim odabirima dvije distribucije. Eksperimentalni dizajn je u svim slučajevima isti: treniramo dvije arhitekture, opisane u prošloj sekciji, na 60 tisuća nizova slučajno (uniformno) odabrane duljine od 2 do 15, te tada radimo empirijsku procjenu točnosti pomoću 1000 nizova duljina od 2 do 25, za obje razmatrane distribucije.

(a) Prvo modificramo eksperiment 3 na način da, umjesto distribucija $\mathcal{N}(0,1)$ i $\mathcal{N}(0,2)$, razmatramo distribucije $\mathcal{N}(0,1)$ i $\mathcal{N}(0,1.5)$. Rezultati su vidljivi na sljedećem grafu. Vidimo da je, očekivano, prepoznavanje manje uspješno. Pri raspoznavanju distribucija $\mathcal{N}(0,1)$ i $\mathcal{N}(0,2)$ "obična" rekurentna neuronska mreža je za nizove duljine 25 imala točnost oko 95%, dok u ovom slučaju je njezina točnost ispod 90%.

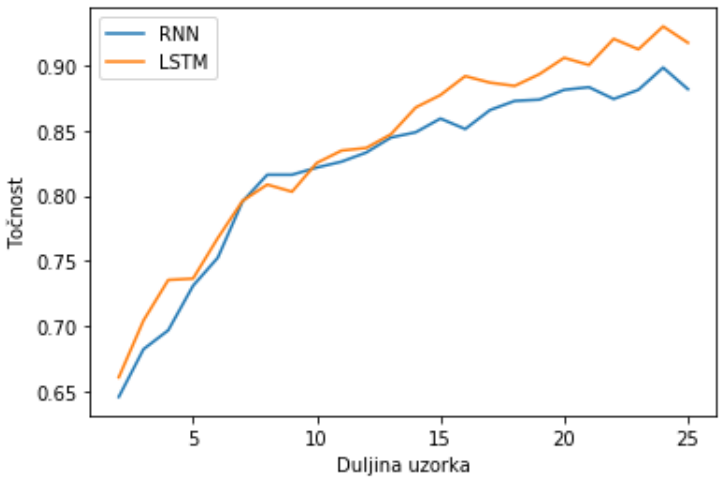


Slika 3.2: Empirijska procjena točnosti klasifikatora (y-os) u ovisnosti o duljini uzorka (x-os) za primjer (a).

(b) Sada učimo naša dva klasifikatora raspoznavanje između $\mathcal{N}(0,1)$ i $\mathcal{N}(0,1.1)$. Vidimo da i rekurentna neuronska mreža i LSTM mreža jedva raspoznaju te dvije distribucije, ali ipak rade bolje od slučajnog pogađanja.



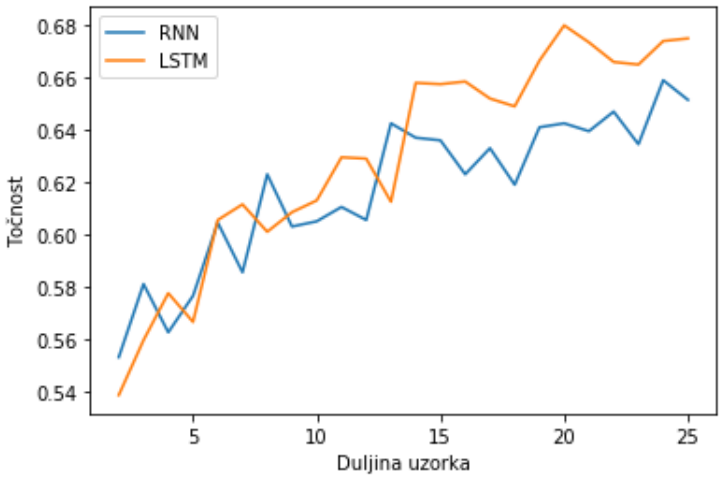
Slika 3.3: Empirijska procjena točnosti klasifikatora (y-os) u ovisnosti o duljini uzorka (x-os) za primjer (b).



Slika 3.4: Empirijska procjena točnosti klasifikatora (y-os) u ovisnosti o duljini uzorka (x-os) za primjer (c).

(c) Sada fiksiramo standardnu devijaciju na 1 a variramo očekivanu vrijednost; gledamo distribucije $\mathcal{N}(-0.3, 1)$ i $\mathcal{N}(0.3, 1)$.

(d) Opet otežavamo problem smanjenjem "razlike" između distribucija; promatramo distribucije $\mathcal{N}(-0.1, 1)$ i $\mathcal{N}(0.1, 1)$.



Slika 3.5: Empirijska procjena točnosti klasifikatora (y-os) u ovisnosti o duljini uzorka (x-os) za primjer (d).

Diskusija eksperimenata

U prvom eksperimentu smo vidjeli da rekurentna neuronska mreža sa samo dva neurona dostaje da u nizu jedinica i nula pretvori jedinice u nule, a nule u jedinice, te smo vidjeli da mreža koja je trenirana na nizovima duljine od 10 do 20 održava dobre rezultate na mnogo dužim nizovima.

Drugi eksperiment je tražio od rekurentne neuronske mreže da zapamti $\alpha = 2, \ldots, 10$ koraka u prošlost, odnosno da u vremenskom koraku α reproducira niz od njegovog početka. Vidjeli smo da broj neurona koji je bio potreban za to linearno rastao s α, pa čak je na našem uzorku uvijek bio $\alpha + 1$. Također smo provjerili može li mreža s tim brojem neurona zapamtiti niz, pa onda i obrnuti nule i jedinice kao u prvom eksperimentu.

Nakon toga, u trećem i četvrtom eksperimentu usporedili smo običnu rekurentnu neuronsku mrežu i LSTM mrežu na zadatku prepoznavanja nizova koji dolaze iz distribucija $\mathcal{N}(\mu_1, \sigma_1)$ i $\mathcal{N}(\mu_2, \sigma_2)$ gdje je $\mu_1 = \mu_2$ i $\sigma_1 \neq \sigma_2$, ili gdje je $\mu_1 \neq \mu_2$ i $\sigma_1 = \sigma_2$. Vidjeli smo da gotovo uvijek LSTM mreža radi nešto bolje od obične rekurentne mreže. U svim slučajevima rekurentne neuronske mreže rade znatno bolje od slučajnog pogađanja.

Bibliografija

[1] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia i Atilla Baskurt, *Sequential Deep Learning for Human Action Recognition*, Proceedings of the Second International Conference on Human Behavior Unterstanding (Berlin, Heidelberg), HBU'11, Springer-Verlag, 2011, str. 29–39, ISBN 978-3-642-25445-1, http://dx.doi.org/10.1007/978-3-642-25446-8_4.

[2] L. Bottou, O. Chapelle, D. Decoste, J. Weston i C.J. Lin, *Large-scale Kernel Machines*, Neural information processing series, MIT Press, 2007, ISBN 9780262026253, https://books.google.hr/books?id=MDup2gE3BwgC.

[3] Edward Choi, Mohammad Taha Bahadori, Andy Schuetz, Walter F. Stewart i Jimeng Sun, *Doctor AI: Predicting Clinical Events via Recurrent Neural Networks*, JMLR workshop and conference proceedings **56** (2015), 301–318.

[4] François Chollet, *Keras*, dostupno na https://github.com/fchollet/keras (studeni 2019.).

[5] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of Control, Signals, and Systems (MCSS) **2** (1989), br. 4, 303–314, ISSN 0932-4194, http://dx.doi.org/10.1007/BF02551274.

[6] Douglas Eck i Jürgen Schmidhuber, *Learning the Long-Term Structure of the Blues*, Artificial Neural Networks — ICANN 2002 (Berlin, Heidelberg) (José R. Dorronsoro, ur.), Springer Berlin Heidelberg, 2002, str. 284–289, ISBN 978-3-540-46084-8.

[7] J. Feldman i R. Rojas, *Neural Networks: A Systematic Introduction*, Springer Berlin Heidelberg, 2013, ISBN 9783642610684, https://books.google.hr/books?id=4rESBwAAQBAJ.

[8] I. Goodfellow, Y. Bengio i A. Courville, *Deep Learning*, Adaptive Computation and Machine Learning series, MIT Press, 2016, ISBN 9780262035613, https://books.google.hr/books?id=Np9SDQAAQBAJ.

[9] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke i Jürgen Schmidhuber, *A Novel Connectionist System for Unconstrained Handwriting Recognition*, IEEE Trans. Pattern Anal. Mach. Intell. **31** (2009), br. 5, 855–868, ISSN 0162-8828, http://dx.doi.org/10.1109/TPAMI.2008.137.

[10] Alex Graves i Jürgen Schmidhuber, *2005 Special Issue: Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures*, Neural Netw. **18** (2005), br. 5-6, 602–610, ISSN 0893-6080, http://dx.doi.org/10.1016/j.neunet.2005.06.042.

[11] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, Studies in Computational Intelligence, Springer Berlin Heidelberg, 2012, ISBN 9783642247965, https://books.google.hr/books?id=4UauNDGQWN4C.

[12] M.T. Hagan, H.B. Demuth, M.H. Beale i O. De Jesús, *Neural Network Design*, Martin Hagan, 2014, ISBN 9780971732117, https://books.google.hr/books?id=4EW9oQEACAAJ.

[13] Sepp Hochreiter, Martin Heusel i Klaus Obermayer, *Fast Model-based Protein Homology Detection Without Alignment*, Bioinformatics **23** (2007), br. 14, 1728–1736, ISSN 1367-4803, http://dx.doi.org/10.1093/bioinformatics/btm247.

[14] Sepp Hochreiter i Jürgen Schmidhuber, *Long Short-Term Memory*, Neural Comput. **9** (1997), br. 8, 1735–1780, ISSN 0899-7667, http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[15] Diederik P. Kingma i Jimmy Ba, *Adam: A Method for Stochastic Optimization*, CoRR **abs/1412.6980** (2014).

[16] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff i Puneet Agarwal, *Long Short Term Memory Networks for Anomaly Detection in Time Series*, ESANN, 2015.

[17] Hermann Georg Mayer, Faustino J. Gomez, Daan Wierstra, Istvan Nagy, Alois Knoll i Jürgen Schmidhuber, *A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks*, Advanced Robotics **22** (2006), 1521–1537.

[18] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*, Applied Optimization, Springer US, 2003, ISBN 9781402075537, https://books.google.hr/books?id=VyYLem-l3CgC.

[19] Ilya Sutskever, *Training Recurrent Neural Networks*, Disertacija, Toronto, Ont., Canada, Canada, 2013, ISBN 978-0-499-22066-0, AAINS22066.

[20] Ilya Sutskever, Oriol Vinyals i Quoc V. Le, *Sequence to Sequence Learning with Neural Networks*, Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (Cambridge, MA, USA), NIPS'14, MIT Press, 2014, str. 3104–3112, http://dl.acm.org/citation.cfm?id=2969033.2969173.

[21] Z. Yi, *Convergence Analysis of Recurrent Neural Networks*, Network Theory and Applications, Springer US, 2013, ISBN 9781475738193, https://books.google.hr/books?id=gTjnBwAAQBAJ.

Sažetak

U ovom radu prezentiramo osnovne definicije i činjenice o neuronskim mrežama, s naglaskom na rekurentne mreže. Prvo prezentiramo perceptron i njegov algoritam za učenje, te unaprijedne neuronske mreže s fokusom na višeslojni perceptron. Posebno, izvodimo algoritam unazadne propagacije za višeslojni perceptron. Proučavamo "obične" rekurentne neuronske mreže i LSTM mreže. Diskutiramo dvosmjerne rekurentne neuronske mreže i problem eksplodirajućeg/iščezavajućeg gradijenta. Naposljetku, prezentiramo neke primjene rekurentnih neuronskih mreža, te testiramo snagu rekurentnih neuronskih mreža na nekoliko jednostavnih zadataka.

Summary

In this work we discuss discuss neural networks, particularly recurrent ones. We present the perceptron and its learning algorthm, as well as feedforward neural networks with emphasis on multilayer perceptron (MLP). In particular, we derive backpropagation algorithm for the MLP. We then present vanilla recurrent neural networks (RNNs) and LSTM networks. We discuss bidirectional RNNs and vanishing/exploding gradient problem. Finally, we present some applications of RNNs, as well as conduct several experiments testing RNNs ability to perform some simple tasks.

Životopis

Rođen sam 1993. u Zadru. U Zadru sam završio osnovnu i srednju školu. Preddiplomski studij matematike završio sam u Rijeci, na Odjelu za matematiku pri Sveučilištu u Rijeci, a nakon toga sam upisao diplomski studij matematičke statistike na Prirodoslovno-matematičkom fakultetu pri Sveučilištu u Zagrebu. U slobodno vrijeme volim kuhati i jesti umake od rajčice.