Mreža dugog kratkoročnog pamćenja

Kovač, Marin

Sveučilište J. J. Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike
Sveučilišni prijediplomski studij Matematika i računarstvo

Marin Kovač

Mreža dugog kratkoročnog pamćenja

Završni rad

Osijek, 2023.

Sveučilište J. J. Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike
Sveučilišni prijediplomski studij Matematika i računarstvo

Marin Kovač

Mreža dugog kratkoročnog pamćenja

Završni rad

Mentor: izv. prof. dr. sc. Domagoj Matijević
Komentor: Antonio Jovanović

Osijek, 2023.

Sažetak: U ovom ćemo radu upoznati neuronske mreže, rekurentne neuronske mreže te mreže dugog kratkoročnog pamćenja. Najprije ćemo navesti osnove pojmove iz strojnog učenja. Zatim ćemo proći kroz osnove neuronskih mreže, definirati pojmove i notaciju mreža, te pojasniti proces treniranja neuronskih mreža. Nakon toga, opisat ćemo rekurentne mreže i predstaviti problematiku koja se pojavljuje kod običnih rekurentnih mreža. Predstavit ćemo mrežu dugog kratkoročnog pamćenja kao rješenje spomenute problematike kod jednostavnih rekurentnih mreža. Na kraju ćemo ukratko proći kroz različite izvedbe mreža dugog kratkoročnog pamćenja.

Ključne riječi: strojno učenje, neuronska mreža, rekurentna neuronska mreža, mreža dugog kratkoročnog pamćenja

Long short-term memory networks

Abstract: In this paper, we will define neural networks, recurrent neural networks and long short-term memory networks. Firstly, we will introduce fundamental concepts of neural networks, define terms and notations related to networks and explain the process of traiining. After that, we will describe recurrent networks and present the problem that occurs with simple recurrent networks. Then, we will long short-term memory network as a solution to mentioned problem with classic RNNs. Finally, we will briefly go through different modifications of the LSTM.

Key words: machine learning, neural network, recurrent neural network, long short-term memory network

Sadržaj

Uvod **1**

1. Osnove neuronskih mreža **2**

1.1. Neuron 2

1.2. Aktivacijske funkcije 3

1.3. Arhitektura neuronske mreže 4

1.4. Funkcije gubitka 5

1.5. Treniranje modela 6

1.6. Algoritam povratne propagacije 7

2. Rekurentna neuronska mreža **10**

2.1. Arhitektura RNN 10

2.2. Treniranje RNN 11

3. LSTM **14**

3.1. Arhitektura LSTM ćelije 14

3.2. Rješenje problema treniranja 15

3.3. Varijante LSTM-a 17

Literatura **20**

Uvod

Cilj strojnog učenja je rješavanje problema na način da se računalima pomogne "pronaći" algoritam za rješavanje problema bez pružanja eksplicitno do kraja razrađenog algoritma za problem. Matematičke osnove strojnog učenja proizlaze iz optimizacije. Po glavnoj podjeli postoje dva načina učenja:

- **Nadgledano učenje** Tip učenja kroz koji algoritam uči iz labeliranih podataka (podaci s ulazno-izlaznim parom). Koristi se za probleme kao što su klasifikacija i regresija.
- **Nenadgledano učenje** Tip učenja kroz koji algoritam uči neki obrazac/uzorak/šablon iz nelabeliranih podataka tražeći strukture i odnose. Koristi se za probleme kao što su klasteriranje i redukcija dimenzije.

O tome što i kako će naš "algoritam" za učenje raditi ovisi o vrsti problema kojom se bavimo. Neki primjeri problema iz strojnog učenja:

- **Klasifikacija slika** Razvoj modela za klasifikaciju različitog vrsta cvijeća iz slika.
- **Analiza sentimenata** Razvoj modela za sentimentalnu analizu koji određuje je li dani tekst pozitivan, neutralan ili negativan.
- **Otkrivanje spama** Izrada modela koji može klasificirati e-mail-ove kao spam.
- **Prevođenje jezika** Razvoj sustava koji prevodi tekst iz jednog jezika u drugi.
- **Prepoznavanje radnje** Izgradnja modela koji može prepoznavati i klasificirati radnje u video klipu.

Naš "algoritam" koji uči zovemo **model**. Za gore navedene probleme najčešće se koriste umjetne neuronske mreže i pokazale su se kao najbolje rješenje za mnoge probleme. Umjetne neuronske mreže inspirirane su radom ljudskog mozga.

Kroz ovaj papir, ukoliko nije naglašeno drukčije, bavit ćemo se **nadgledanim učenjem.**

1. Osnove neuronskih mreža

Princip rada neuronskih mreža zasnovan je na vrlo pojednostavljenom načinu "računanja" ljudskog mozga. Prvu takvu mrežu (**perceptron**) definirao je 1957. godine znanstvenik Rosenblatt [9]. Iako se taj koncept pojavio davne 1957., procvat te tehnologije dogodio se tek u 21. stoljeću zbog velikog unaprijeđenja brzine računala koji je omogućio brže i bolje učenje neuronskih mreža pa samim time i olakšao rješavanje problema u strojnom učenju.

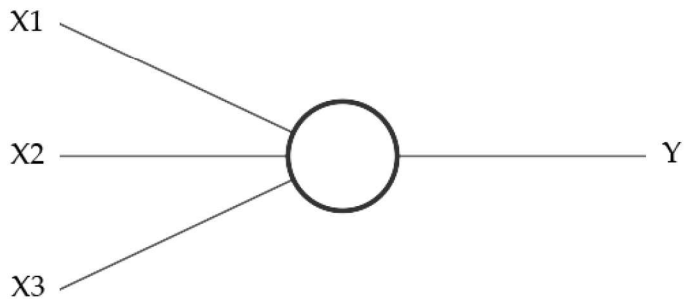
1.1. Neuron

Neuronska mreža sastoji se od umjetnih neurona i njihovih veza. Izlaz neurona y računa se kao težinska suma ulaza x_i, uvećana za bias b, na koju dijeluje aktivacijska funkcija $a : \mathbb{R} \to \mathbb{R}$.

$$y = a\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{1}$$

gdje su w_i težine pripadnih ulaza x_i.

Primjer 1.1. *Za primjer ćemo uzeti vrlo jednostavan neuron (ranije spomenuti preceptron). Rosenblatt je predložio jako jednostavno pravilo za računanje izlaza. Izlaz će biti* 0 *ili* 1*, što*



Slika 1: Perceptron

je određeno ocjenjivanjem sume $\sum_j w_j x_j$ *nekim pragom. Baš kao i težine, prag je parametar neurona.*

$$y = \begin{cases} 0 & ako \ \sum_j w_j x_j \leq \ prag \\ 1 & ako \ \sum_j w_j x_j > \ prag \end{cases} \tag{2}$$

Ovaj neuron je jednostavan matematički model. Način na koji se o njemu može razmišljati je kao objekt koji radi odluku ocjenjivajući određene zavisnosti.

Na primjer; želimo organizirati zabavu, ali ne znamo je li ovaj vikend pogodan za takav događaj. Izlaz bi nam bio 0 za ne treba organizirati ili 1 za treba organizirati, dok bi nam ulazi bili vremenska pogodnost, dolazi li najbolji prijatelj i imamo li sredstva za organizaciju zabave.

1.2. Aktivacijske funkcije

Nakon računanja težinske sume ulaza, neuron djeluje aktivacijskom funkcijom $a : \mathbb{R} \to \mathbb{R}$ kako bi proizveo izlaz y. U strojnom učenju gotovo uvijek rješavamo problem optimizacije pa je poželjno da aktivacijske funkcije budu neprekidne i diferencijabilne. Neke od aktivacijskih funkcija koje se najčešće koriste:

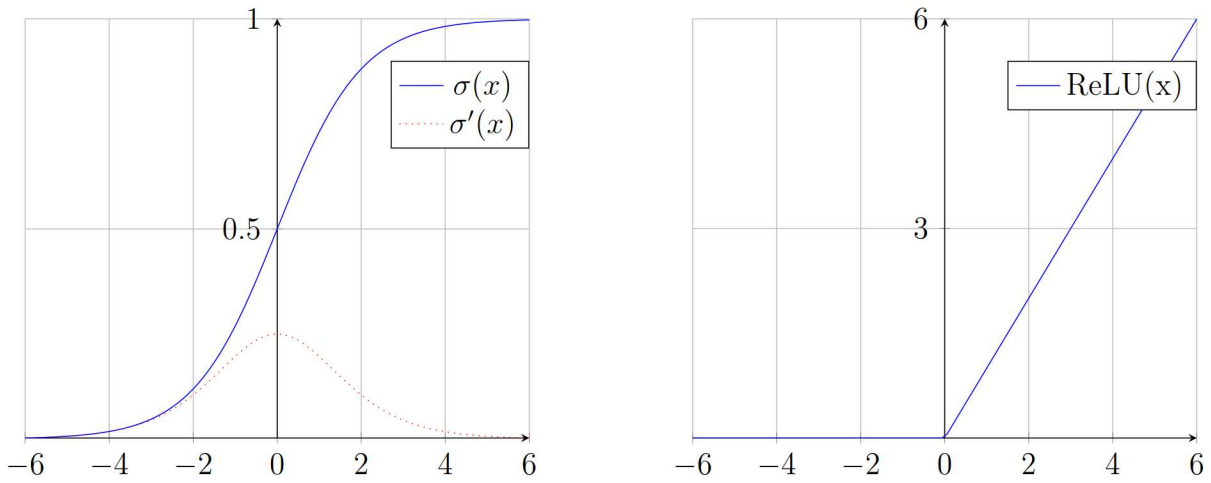
- **Identiteta**, zadana s
$$\mathrm{id}(x) = x, \tag{3}$$
gotovo uvijek se koristi samo na ulaznom sloju.

- **ReLU funkcija**, zadana s
$$\mathrm{ReLU}(x) = \max(0, x), \tag{4}$$
je po dijelovima linearna i diferencijabilna svuda osim u $x = 0$. Druga derivacija joj je 0 na cijeloj domeni pa ju to čini beskorisnom u optimizacijskim metodama koje koriste drugu derivaciju.

- **Softplus funkcija**, zadana s
$$\zeta(x) = \log(1 + \exp(x)) \tag{5}$$
je glatka verzija ReLU funkcije. U praksi pokazalo da ReLU daje bolje rezultate pa se softplus vrlo rijetko koristi.

- **Sigmoidna funkcija**, zadana s
$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{6}$$
je najkorištenija aktivacijska funkcija i koristi se u problemu klasifikacije s dvije klase jer vraća vrijednosti od 0 do 1. Diferencijabilna je i njena derivacija glasi:
$$\sigma'(x) = \frac{1}{1 + \exp(-x)} \tag{7}$$

- **Tangens hiperbolni**, zadana s
$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} = 2\sigma(2x) - 1, \tag{8}$$
je skalirana i translatirana sigmoidna funkcija pa to znači da je i diferencijabilna i njena derivacija glasi:
$$\tanh'(x) = 1 - \tanh^2(x), \tag{9}$$



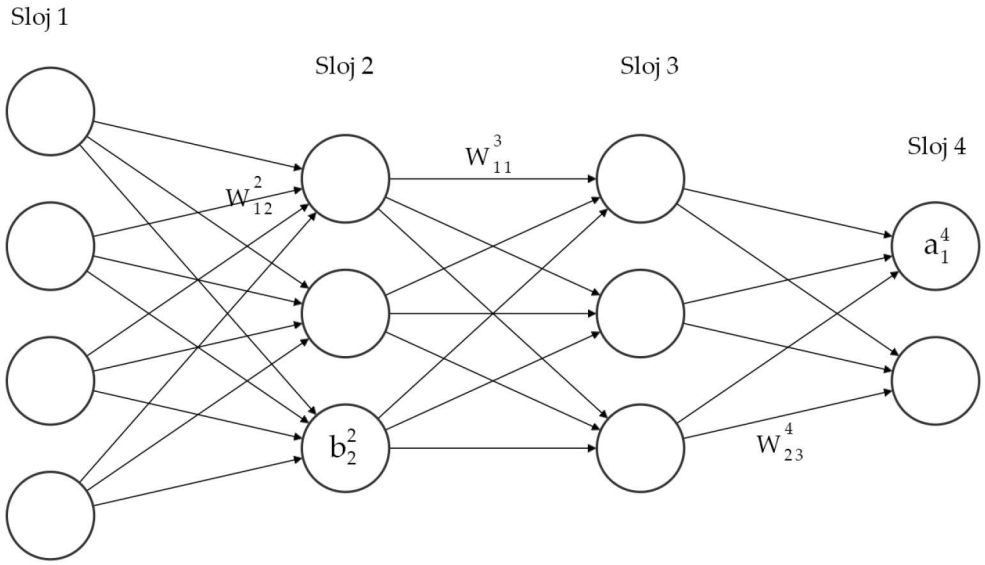
(a) Sigmoidna funkcija

(b) ReLU funkcija

Slika 2: Grafovi aktivacijskih funkcija

1.3. Arhitektura neuronske mreže

Feedforward neuronska mreža organizirana je u slojeve: ulazni sloj, skriveni slojevi i izlazni sloj. U feedforward neuronskim mrežama nema petlji i informacije uvijek putuju prema naprijed. Drugim riječima, izlaz jednog sloja služi kao ulaz sljedećeg sloja. Slojevi se sastoje od neurona što znači da je izlaz sloja zapravo vektor izlaza svih neurona tog sloja, dok je ulaz svakog neurona zapravo izlaz prijašnjeg sloja.



Slika 3: neuronska mreza

Prije svega trebamo se upoznati s notacijom. Koristit ćemo $\mathbf{w^l_{jk}}$ za označavanje težina koje spajaju k-ti neuron u $(l-1)$-om sloju s j-tim neuronom u l-tom sloju. Slično kao i za

težine uvest ćemo notaciju za biase i aktivacije. $\mathbf{b_j^l}$ koristimo za bias j-tog neurona u l-tom sloju, te $\mathbf{a_j^l}$ za aktivacije j-tog neurona u l-tom sloju.

S tim notacijama dobijamo formulu:

$$a_j^l = f(\sum_k w_{jk}^l a_k^{l-1} + b_j^l), \tag{10}$$

gdje je suma težinska suma neurona iz $(l-1)$-og sloja.

Kako bi napisali tu formulu u matričnoj formi definiramo matricu težina w^l za svaki sloj. Slično definiramo vektor biasa b^l i vektor aktivacija a^l te posrednu vrijednost $z^l := w^l a^{l-1} + b^l$. Sada naša jednadžba glasi:

$$a^l = f(w^l a^{l-1} + b^l) \tag{11}$$

ili

$$a^l = f(z^l) \tag{12}$$

koja nam daje puno lakši pogled na to kako se odnose aktivacije između slojeva.

Definicija 1.1. *Neuronska mreža je model koji ćemo formalno definirati kao funkciju* $m : \mathbb{R}^{n_{input}} \to \mathbb{R}^{n_{output}}$*, gdje je* n_{input} *dimenzija ulaznog prostora, a* n_{output} *dimenzija izlaznog prostora. Parametri funkcije su sve težine i biasi.*

Sada smo upoznati s konceptom, notacijom i definicijom neuronske mreže, ali mreža nam je beskorisna ukoliko ne možemo efikasno "naučiti" mrežu kako riješiti naš problem.

1.4. Funkcije gubitka

Definicija 1.2. *Funkcija gubitka* $J : Y \times Y \to R_+$*, gdje je* Y *ulazni prostor koji predstavlja moguće izlazne vrijednosti* $\hat{y}$ *i ciljane vrijednosti* y*. Ona prima par* $(\hat{y}, y)$ *te kvantificira njihovu neusklađenost.*

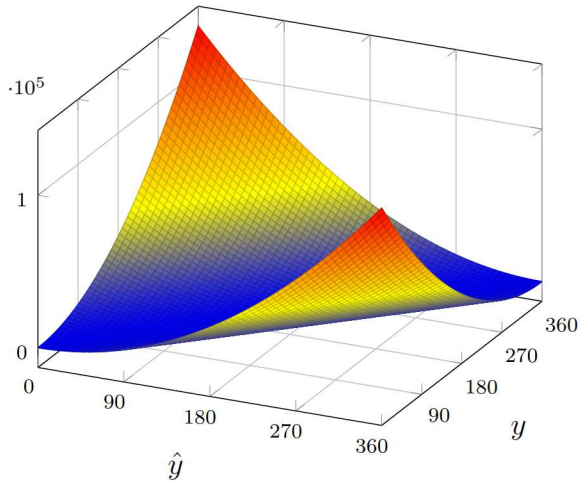
Često korištene funkcije gubitka:

- **MSE**, zadana s
$$J(\hat{y}, y) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{13}$$
koja mjeri kvadratnu razliku između predviđane i ciljane vrijednosti.

- **Cross-entropy**, zadana s
$$J(\hat{y}, y) = -\sum_{i=1}^{n} y_i \log(\hat{y}_i) \tag{14}$$
koja se često koristi u klasifikacijskim problemima.

- **Hinge**, zadana s
$$J(\hat{y}, y) = \max(0, 1 - \hat{y} \cdot y) \tag{15}$$
koja se često koristi u SVM[1] i binarnoj klasifikacij.

[1]Support vector machine - tip nadziranog učenja koji se koristi za rješavanje klasifikacije i regresije



Slika 4: MSE nad jednodimenzionalnim izlazom

1.5. Treniranje modela

Kako je funkcija gubitka računa numeričku vrijednost koja opisuje neusklađenost našeg izlaza s željenim, treniranje modela svodimo na problem minimizacije funkcije gubitka. Izlaz našeg modela ovisi o parametrima (težinama i biasima), što znači da ćemo minimum funkcije gubitka tražiti namještanjem težina i biasa.

Uzmimo neka je naša funkcija gubitka:

$$J(w,b) = \frac{1}{2n}\sum_{x} ||y - a^L||^2 \tag{16}$$

gdje je L broj slojeva u mreži.

Sada kada imamo funkciju gubitka, jedino što nam preostaje je naći optimizacijsku metodu. Kako su naša funkcija gubitka i model derivabilni, možemo odabrati neku od iterativnih optimizacijskih metoda. Dobar izbor je metoda gradijentnog spusta. Ideja je u kretati se u suprotnom smjeru od gradijenta jer je to smjer najstrmijeg pada funkcije. Dokaz konvergencije ove metode može se naći u [10]. Naši koraci su opisani formulama:

$$w^{i+1} = w^i - \lambda \nabla_w J(w^i, b^i) \tag{17}$$

$$b^{i+1} = b^i - \lambda \nabla_b J(w^i, b^i) \tag{18}$$

gdje je λ skalar koji zovemo duljina koraka i on određuje brzinu "spuštanja". Ovaj proces ponavljamo sve dok se dovoljno ne približimo točki lokalnog minimuma. Jednu iteraciju ovog algoritma zovemo korak.

Kako često u problemima koje neuronske mreže rješavaju imamo jako puno podataka, vrlo je skupo računati gradijent za cijeli skup u svakoj iteraciji. Iz tog razloga koriste se modifikacije gradijente metode:

- **Stohastički gradijentni spust** ili SGD, koji aproksimira gradijent koristeći jedan slučajan podatak.
- **Mini-batch gradijentni spust** koji podjeli podatke u disjunktne skupove, te za svaki skup napravi korak. Jedan takav prolazak kroz sve podatke nazivamo **epoha**.

U praksi se često koriste dodatne optimizacije (L1 regularizacija, L2 regularizacija, dropout, umjetno proširenje podataka). Trenutno ručno računamo duljinu koraka što može dovesti do jako sporog učenja ili ne postizanja konvergencije u minimum. U tu svrhu se koriste algoritmi kao što su **Adam**[2].

1.6. Algoritam povratne propagacije

Upoznati smo s algoritmom za učenje težina i biasa pomoću gradijentne metode. Naime, nismo upoznati s načinom na koji računamo gradijent mreže. Brzi algoritam koji računa gradijent zove se **povratna propagacija**. Algoritam je originalno nastao 1970-ih, ali je dobio pažnju tek 1990-ih. Danas je to glavni mehanizam neuronskih mreža.

Ovaj nam je algoritam potreban jer je naivno računanje gradijenta preko pravila ulančavanja jako skupo zbog toga što bi za svaki neuron morali proči svim putevima iz izlaznog sloja do tog neurona. Kako bi riješili taj problem, uvodimo posrednu vrijednost δ_j^l koju ćemo zvati greška j-tog neurona u l-tom sloju.

$$\delta_j^l := \frac{\partial J}{\partial z_j^l} \tag{19}$$

Dokazati ćemo četiri fundimentalne formule na kojima je algoritam zasnovan. Intuitivno pojašnjenje nalazi se u [7].

Propozicija 1.1 (Greška izlaznog sloja)**.**

$$\delta^L = \nabla_a J \odot f'(z^L) \tag{20}$$

Dokaz. Iz definicije:

$$\delta_j^L = \frac{\partial J}{\partial z_j^L}$$

Primjenom pravila o ulančavanju dobijamo:

$$\delta_j^L = \sum_k \frac{\partial J}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L}$$

Kako a_k ovisi o z_j samo kada $k = j$, $\partial a_k^L / \partial z_j^L$ iščezava kada $k \neq j$ pa dobijemo:

$$\delta_j^L = \frac{\partial J}{\partial a_j^L} f'(z_j^L)$$

a to je upravo (20). □

Propozicija 1.2 (Greška sloja)**.**

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot f'(z^l) \tag{21}$$

Dokaz. Kako bi došli do te formule, trebamo napisati δ_j^l u terminima δ_k^{l+1} Pa idemo primjeniti pravilo o ulančavanju:

$$\delta_j^l = \frac{\partial J}{\partial z_j^l}$$

[2]Adam je skraćenica od Adaptive Moment Estimation

$$= \sum_k \frac{\partial J}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_k^l}$$

$$\delta_j^l = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_k^l} \tag{22}$$

Sada još trebamo raspisati $\frac{\partial z_k^{l+1}}{\partial z_k^l}$:

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1}$$

$$(z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1}) \setminus \frac{\partial}{\partial z_k^l}$$

$$\frac{\partial z_k^{l+1}}{\partial z_k^l} = w_{kj}^{l+1} f'(z_j^l)$$

Vratimo to nazad u (22) i dobijemo:

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} f'(z_j^l)$$

što je upravo (21). □

Propozicija 1.3 (Parcijalna derivacija biasa)**.**

$$\frac{\partial J}{\partial b_j^l} = \delta_j^l \tag{23}$$

Dokaz. Po definicij:

$$\delta_j^L = \frac{\partial J}{\partial z_j^L}$$

Primjenom pravila o ulančavanju dobijamo:

$$\delta_j^L = \sum_k \frac{\partial J}{\partial b_k^l} \frac{\partial b_k^l}{\partial z_j^l}$$

Kako b_k^l ovisi samo o z_k^l, sve sume iščezavaju osim kada $k = j$:

$$\delta_j^l = \frac{\partial J}{\partial b_j^l} \frac{\partial b_j^l}{\partial z_j^l}$$

Iz definicije $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$ lako se vidi da je:

$$\frac{\partial b_j^l}{\partial z_j^l} = 1$$

pa nam to daje upravo (23). □

Propozicija 1.4 (Parcijalna derivacija težine)**.**

$$\frac{\partial J}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{24}$$

Dokaz. Analogno kao i u prijašnjem dokazu primjenjujemo pravilo o ulančavanju i odmah se riješimo sume jer w^l_{jk} ovisi samo o z^l_j pa dobijamo:

$$\frac{\delta J}{\delta w^l_{jk}} = \frac{\delta J}{\delta z^l_j} \frac{\delta z^l_j}{\delta w^l_{jk}} \tag{25}$$

Deriviranjem definicije z^l_j dobijamo:

$$(z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j) \setminus \frac{\delta}{\delta w^l_{jk}}$$

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

i po definicij je δ^l_j jednak $\frac{\delta J}{\delta z^l_j}$ pa uvrštavanjem u (25) dobijamo (24). □

Te jednadžbe nam pružaju način za izračunati gradijent funkcije gubitka. Sada kada ih imamo, možemo napisati cijeli algoritam koristeći povratnu propagaciju i gradijentnu metodu:

Algorithm 1 Treniranje

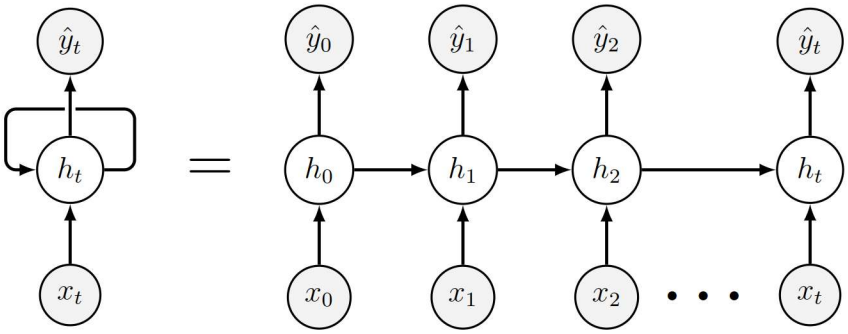
Require: ulaz x
Require: labele y
Require: tezine w^l_{jk}
Require: biase b^l_j
Require: aktivacijske funkcije f^l
Require: funkcija gubitka J
Require: duljina koraka η
Require: $a^l, biaseb$
  $a^1 \leftarrow x$
  for $l = 2, 3, ..., L$ **do** (propagacija unaprijed)
    $z^l \leftarrow w^l a^{l-1} + b^l$
    $a^l \leftarrow f(z^l)$
  end for
  $\delta^L \leftarrow \nabla a \odot f'(z^L)$
  for $l = L-1, L-2, ..., 2$ **do** (povratna propagacija)
    $\delta^l \leftarrow ((w^l + 1)^T \delta^{l+1}) \odot f'(z^l)$
  end for
  for $l = L, L-1, ..., 2$ **do** (gradijentna metoda)
    $w^l \leftarrow w^l - \eta a^{l-1} \delta^l$
    $b^l \leftarrow b^l - \eta \delta^l$
  end for

2. Rekurentna neuronska mreža

Do sada smo pričali o feedforward neuronskim mrežama koje kao ulaz u mrežu dobijaju vanjske podatke. Što kada bi uz podatke htjeli koristiti i informacije o povijesti stanja iz našeg modela? Recimo da želimo modelom predviđati temperaturu nekog mjesta za sljedeći sat i kao ulaz mu dajemo temperature po satima mjerene zadnjih par dana. U ovom slučaju, korištenje feedforward neuronske mreže nema smisla jer feedforward neuronska mreža tretira svaki ulaz kao nezavisan. Kako bismo postigli da naša neuronska mreža zapamti informacije o prijašnjim temperaturama model bi nam bio puno smisleniji. Tu dolaze **rekurentne neuronske mreže** koje čuvaju neko skriveno stanje o prošlosti i koriste ga u sljedećem predviđanju.

Rekurentna neuronska mreža dizajnirana je za baratanje s nizovima podataka uvodeči petlje u mreži kako bi uhvatila zavisnosti.

2.1. Arhitektura RNN



Slika 5: RNN Arhitektura

Skriveno stanje h_t na svakom koraku računa se težinskom sumom prijašnjeg stanja h_{t-1}, trenutnog ulaza x_t i biasa b_h na koju djelujemo aktivacijskom funkcijuom, dok se izlaz y_t računa sumom težinske vrijednosti skrivenog stanja h_t i biasa b_y. Skriveno stanje h_t je to što rekurentnim mrežama omogućava hvatanje zavisnosti između podataka u nizu podataka.

Ponašanje rekurentnih mreža može se opisati sljedećim formulama:

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h) \tag{26}$$

$$y_t = g(W_{hy}h_t + b_y) \tag{27}$$

gdje su:

t : vrijeme ulaza (indeks)
x_t : ulaz u vremenu t
h_t : skriveno stanje u vremenu t
y_t : izlaz u vremenu t
W_{hh} : težina skrivenog stanja
W_{xh} : težina između ulaza i prijašnjeg skrivenog stanja
W_{hy} : težine između skrivenog stanja i izlaza
b_h : bias za skriveno stanje
b_y : bias za izlaz
f : aktivacijska funkcija
g : aktivacijska funkcija na izlazu

Glavna razlika između feedforward i rekurentne arhitekture je ta što feedforward neuronska mreža svoje težine koristi samo jednom (između slojeva) dok u rekurentnoj arhitekturi koristimo iste težine za cijelu duljinu ulaznog niza. Zbog tog je potrebno prilagoditi algoritam povratne propagacije rekurentnoj arhitekturi. Također, rekurentna arhitektura je vrlo prilagodljiva u smislu ulaza i izlaza pa po vrsti problema koji rješavamo možemo birati:

- **više u više**; niz x_t na ulazu i niz y_t na izlazu
- **više u jedan**; niz x_t na ulazu i zadnje stanje y na izlazu
- **jedan u više**; kroz svaki korak primi isti ili proizvoljni x na ulazu i vrati niz y_t na izlazu
- **jedan u jedan**; isto kao u jedan u više osim što vrati samo zadnji y
- **izlaz različite duljine od ulaza**; za ovo se najčešće koristi encoder-decoder struktura

2.2. Treniranje RNN

Isto kao i kod feedforward mreža, potreban nam je algoritam za treniranje RNN. Rekurentna mreža ima petlju, međutim ako pogledamo desnu stranu slike (5), iz "raspetljane" mreže vidimo da je to zapravo feedforward mreža u kojoj se ponavljaju parametri. Razlika je u tome što se kod velikih nizova pojavljuju računski i optimizacijski problemi koje ćemo u detalje proći nakon što dobijemo ideju o povratnoj propagaciji kroz vrijeme. Pretpostavimo da računamo gradijent za jedan par (x, y) te definirajmo posrednu vrijednost

$$z_t = W_{hy} h_t + b_y$$

i uzmimo da nam je aktivacijska funkcija na izlazu

$$g(z_t) = \text{softmax}(z_t) = \hat{y}_t$$

te funkcija gubitka

$$J(\hat{y}, y) = -\sum_{t=1}^{T} y_t \log(\hat{y}_t)$$

Kako je naša aktivacijska funkcija softmax, pretpostavit ćemo da je y oblika one hot encoded vektora[3], što znači da mu je zbroj komponenti jednak upravo 1. Iz definicije funkcije gubitka:

$$J(\hat{y}, y) = -\sum_j y_j \log(\hat{y}_j)$$

deriviranjem dobijamo izraz:

$$\begin{aligned}
\frac{\partial J}{\partial z_i} &= -\sum_{j=i} y_j \frac{\partial \log(\hat{y}_j)}{\partial z_i} - \sum_{j\neq i} y_j \frac{\partial \log(\hat{y}_j)}{\partial z_i} \\
&= -\sum_{j=i} y_j \frac{1}{\hat{y}_j}\frac{\partial \hat{y}_j}{\partial z_i} - \sum_{j\neq i} y_j \frac{1}{\hat{y}_j}\frac{\partial \hat{y}_j}{\partial z_i} \\
&= -y_i \frac{1}{\hat{y}_j}\hat{y}_j(1-\hat{y}_i) - \sum_{j\neq i} y_j \frac{1}{\hat{y}_j}(-\hat{y}_j\hat{y}_i) \\
&= -y_i(1-\hat{y}_i) - \sum_{j\neq i} y_j \frac{1}{\hat{y}_j}(-\hat{y}_j\hat{y}_i) \\
&= -y_i + y_i\hat{y}_i + \sum_{i\neq j} y_i\hat{y}_i \\
&= \hat{y}_i(\sum_j y_j) - y_i
\end{aligned} \tag{28}$$

Pa iz pretpostavke da je $\sum_j y_j = 1$ dobijamo:

$$\frac{\partial J}{\partial z_t} = \hat{y}_t - y_t \tag{29}$$

Sada kada imamo formulu za derivaciju funkcije gubitka s obzirom na težinsku sumu, uočimo da u svakom vremenu t koristimo istu težinu W_{hy} i zapišimo parcijalnu derivaciju funkcije gubitka s obzirom na težine:

$$\begin{aligned}
\frac{\partial J}{\partial W_{hy}} &= \sum_t^T \frac{\partial J_t}{\partial W_{hy}} \\
&= \sum_t^T \frac{\partial J_t}{\partial z_t}\frac{\partial z_t}{\partial W_{hy}}
\end{aligned}$$

Lagano je za uočiti da $\frac{\partial z_t}{\partial W_{hy}} = h_t$ pa uz (29) dobijamo:

$$\frac{\partial J}{\partial W_{hy}} = \sum_t^T (\hat{y}_t - y_t) \odot h_t \tag{30}$$

Slično dobijemo i parcijalnu derivaciju po biasu izlaza:

$$\frac{\partial J}{\partial b_y} = \sum_t^T \frac{\partial J_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial z_t}{\partial b_y}$$

[3]Vektor duljine broja opcija koji je sastavljen od nula i jedne jedinice

$$\frac{\partial J}{\partial b_y} = \sum_{t}^{T} (\hat{y}_t - y_t) \tag{31}$$

Za sada smo izračunali parametre koji utječu na izlaz i nismo naišli na izraz koji može biti problematičan. Sljedeće što trebamo je gradijent za parametre skrivenog stanja, pa idemo prvo naći derivaciju težine skrivenog stanja u terminima $(t+1)$-og koraka:

$$\frac{\partial J_{t+1}}{\partial W_{hh}} = \frac{\partial J_{t+1}}{\partial \hat{y}_{t+1}} \frac{\partial \hat{y}_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial W_{hh}}$$

Kako naš h_{t+1} parcijalno ovisi o nekom h_k, formulu još možemo raspisati:

$$\frac{\partial J_{t+1}}{\partial W_{hh}} = \sum_{k=1}^{t+1} \frac{\partial J_{t+1}}{\partial \hat{y}_{t+1}} \frac{\partial \hat{y}_{t+1}}{\partial h_{t+1}} \frac{\partial h_{t+1}}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}$$

Primjenjujući pravilo o ulančavanju na izraz $\frac{\partial h_{t+1}}{\partial h_k}$ dobijamo:

$$\frac{\partial J_{t+1}}{\partial W_{hh}} = \sum_{k=1}^{t+1} \frac{\partial J_{t+1}}{\partial \hat{y}_{t+1}} \frac{\partial \hat{y}_{t+1}}{\partial h_{t+1}} (\prod_{j=k}^{t} \frac{\partial h_{j+1}}{\partial h_j}) \frac{\partial h_k}{\partial W_{hh}}$$

Još nam je preostalo agregirati derivaciju kroz sve korake iz čega dobijamo formulu:

$$\frac{\partial J}{\partial W_{hh}} = \sum_{t}^{T} \sum_{k=1}^{t+1} \frac{\partial J_{t+1}}{\partial \hat{y}_{t+1}} \frac{\partial \hat{y}_{t+1}}{\partial h_{t+1}} (\prod_{j=k}^{t} \frac{\partial h_{j+1}}{\partial h_j}) \frac{\partial h_k}{\partial W_{hh}} \tag{32}$$

Vrlo slično se računa $\frac{\partial J}{\partial W_{xh}}$ i nije teško za izračunati $\frac{\partial J}{\partial b_h}$ pa čemo taj račun preskočiti. Naime, u formuli (32) naišli smo na izraz koji bi nam mogao predstavljati problem. Problem se pojavljuje kod velikih nizova zbog velikog broja množenja parcijalnih derivacija što možemo promatrati na izrazu:

$$\prod_{j=k}^{t} \frac{\partial h_{j+1}}{\partial h_j} = \frac{\partial h_{t+1}}{\partial h_k}$$

Pa zapišimo to u obliku:

$$\prod_{j=k}^{t} f_h'(W_{xh} x_{j+1} + W_{hh} h_j + b_h) W_{hh}$$

Iz tog se vidi da ovisno o broju koraka možemo imati jako puno množenja matrice W_{hh} pa ovisno o funkciji naš izraz može eksplodirati ili iščeznuti. Zbog demonstracije, idemo uzeti da je naša funkcija f identiteta.

$$f(x) = id(x) = x$$

pa iz tog slijedi:

$$\frac{\partial h_{t+1}}{\partial h_k} = \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \dots \frac{\partial h_{k+1}}{\partial h_k}$$

$$\frac{\partial h_{t+1}}{\partial h_k} = W_{hh}^{t-k}$$

Pretpostavimo još da je matrica W_{hh} simetrična pa se može zapisati u obliku:

$$W_{hh} = Q \Lambda Q^T$$

gdje je Q ortogonalna, a Λ dijagonalna sa svojstvenim vrijednostima od W_{hh} na dijagonali. Uvrštavanjem nazad u gornju formulu dobijamo:

$$\frac{\partial h_{t+1}}{\partial h_k} = Q\Lambda^{t-k}Q^T$$

Pa za velik broj koraka t sve vrijednosti matrice Λ^{t-k} manje od 1 iščezavaju, dok veće od 1 eksplodiraju. To znači da će u tom slučaju i vrijednosti našeg gradijenta iščeznuti ili eksplodirati jer ovise o matrici Λ^{t-k}. Postoje razne metode kojima se izbjegava problem eksplodirajućeg/nestajućeg gradijenta od kojih je najpoznatija:

- **metoda izrezivanja gradijenta**, u kojoj na gradijent dijelujemo s funkcijom

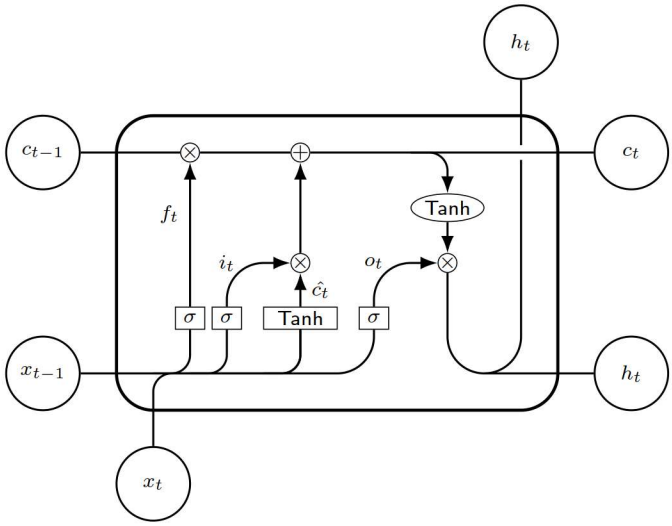
$$f(\nabla, \text{prag}) = \nabla \max(q, \frac{\text{prag}}{\|\nabla\|})$$

  čime ne mijenjamo njegov smjer nego samo "jakost". Kako je $\frac{\partial h_{t+1}}{\partial h_k}$ dio koji uzrokuje eksploziju/nestajanje, dovoljno je djelovati samo na taj dio.

3. LSTM

Jedno rješenje problema ponudili su Hochreiter i Schmidhuber [4], a to je zamjena standardnog neurona s LSTM ćelijom. Uvođenjem LSTM ćelije radimo zamjenu množenja s zbrajanjem kod računanja novog skrivenog sloja. Uz skriveno stanje h_t, sada uvodimo i stanje ćelije c_t koje služi kao dugoročna memorija kombiniranjem "pamćenja" i "zaboravljanja" prijašnjih informacija.

3.1. Arhitektura LSTM ćelije



Slika 6: LSTM čelija

Ključan dio arhitekture LSTM-a je gornji protok od c_{t-1} do c_t koji se ponaša kao pokretna traka za informacije. LSTM briše i dodaje informacije u stanje ćelije strukturama zvanim sklopovi (gates). Sklopovi su sastavljeni od sigmoidnog sloja i množenja. Izlaz sigmoidnog

sloja su vrijednosti u intervalu $(0, 1)$ koje određuju koliko informacija treba "propustiti". LSTM ima 3 takva sklopa.

Prvi korak u LSTM-u je određivanje koliko informacija iz stanja ćelije ćemo odbaciti. Drugim riječima, koliko će se dugoročnog stanja iz c_{t-1} zaboraviti u c_t. Odluku radi sklop zaboravljanja (forget gate). On gleda na h_{t-1} i x_t te vraća broj za svaku komponentu od c_{t-1}.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \tag{33}$$

Sljedeći korak je određivanje koje informacije ćemo spremiti u c_t i sastoji se od dva dijela. Prvi dio je ulazni sklop (input gate) koji odlučuje koje vrijednosti ćemo "ažurirati". Zatim ide tanh sloj koji predstavlja ("kandidate") ulaznih vrijednosti za c_t izračunatih iz h_{t-1} i x_t.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \tag{34}$$

$$\hat{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \tag{35}$$

Kombiniranjem tih dviju vrijednosti ažuriramo c_t.

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t \tag{36}$$

Preostaje nam odrediti izlaz. Izlaz će biti "filtrirano" stanje ćelije. Prvo izlaznim sklopom (output gate) odredimo dijelove c_t koje ćemo "propustit".

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{37}$$

Zatim "gurnemo" vrijednosti od c_t u interval $(-1, 1)$ aktivacijom tanh i izračunamo izlaz (skriveno stanje).

$$h_t = o_t \odot \tanh(c_t) \tag{38}$$

Često se u računu kod težinske sume h_{t-1} i x_t koriste različite težine, ali smo zbog pojednostavljenja koristili zajedničku težinu.

Umjesto množenja kojim se računaju vrijednosti skrivenih stanja u standardnoj rekurentnoj mreži, stanja ćelija LSTM arhitekture računaju se zbrajanjem što rezultira boljim ponašanjem gradijenta kroz vrijeme.

Cijela LSTM arhitektura opisuje se pomoću već spomenutih 6 formula:

$$\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\
\hat{c}_t &= \tanh(W_c[h_{t-1}, x_t] + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \hat{c}_t \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned} \tag{39}$$

3.2. Rješenje problema treniranja

Već smo spomenuli da je LSTM dan kao rješenje problema nestajućeg gradijenta kod treniranja rekurentnih mreža. Kao i kod standarnih rekurentnih mreža, LSTM na izlazu daje vektor vrijednosti po koracima, dok su informacije u c_t dugoročne zavisnosti u nizu podataka. Računanje gradijenta težina uz pretpostavku da su težine dijeljene u svim koracima daje nam:

$$\frac{\partial J}{\partial W} = \sum_{t}^{T} \frac{\partial J}{\partial W}$$

Idemo raspisati derivaciju po W u terminima koraka t:

$$\frac{\partial J_t}{\partial W} = \frac{\partial J_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}\frac{\partial c_t}{\partial W}$$

$$= \frac{\partial J_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}(\prod_{j=k+1}^{t}\frac{\partial c_j}{\partial c_{j-1}})\frac{\partial c_k}{\partial W}$$

U ovom računu imamo sličan izraz kao i kod rekurentnih mreža:

$$\prod_{j=k+1}^{t}\frac{\partial c_j}{\partial c_{j-1}}$$

Takav produkt koji ovisi o broju koraka u rekurentnim mrežama dovodi do iščezavanja gradijenta, ali mi ćemo pokazati kako to u slučaju LSTM-a nije problem. Idemo za početak izderivirati izraz $\frac{\partial c_j}{\partial c_{j-1}}$:

$$\frac{\partial c_j}{\partial c_{j-1}} = \frac{\partial}{\partial c_{j-1}}(f_j \odot c_{j-1} + i_j \odot \hat{c_j})$$

$$\frac{\partial c_j}{\partial c_{j-1}} = \frac{\partial f_j}{\partial c_{j-1}}c_{j-1} + \frac{\partial c_{j-1}}{\partial c_{j-1}}f_j + \frac{\partial i_j}{\partial c_{j-1}}\hat{c_j} + \frac{\partial \hat{c_j}}{\partial c_{j-1}}i_j$$

Izračunajmo sad linearne članove jednadžbe:

$$\frac{\partial f_j}{\partial c_{j-1}}c_{j-1} = \sigma'(W_f[h_{j-1}, x_j] + b_f)W_f\frac{\partial h_j}{\partial c_{j-1}}c_{j-1}$$

$$= \sigma'(W_f[h_{j-1}, x_j] + b_f)W_f(o_{j-1} \odot \tanh'(c_{j-1}))c_{j-1}$$

Analogno se dobiju i druge dvije parcijalne derivacije:

$$\frac{\partial i_j}{\partial c_{j-1}}\hat{c_j} = \sigma'(W_i[h_{j-1}, x_j] + b_i)W_i(o_{j-1} \odot \tanh'(c_{j-1}))\hat{c_j}$$

$$\frac{\partial \hat{c_j}}{\partial c_{j-1}}i_j = \sigma'(W_c[h_{j-1}, x_j] + b_c)W_c(o_{j-1} \odot \tanh'(c_{j-1}))i_j$$

Uvedimo nove oznake za linearne članove:

$$A_j = \sigma'(W_f[h_{j-1}, x_j] + b_f)W_f(o_{j-1} \odot \tanh'(c_{j-1}))c_{j-1}$$

$$B_j = \frac{\partial c_{j-1}}{\partial c_{j-1}}f_j = f_j$$

$$C_j = \sigma'(W_i[h_{j-1}, x_j] + b_i)W_i(o_{j-1} \odot \tanh'(c_{j-1}))\hat{c_j}$$

$$D_j = \sigma'(W_c[h_{j-1}, x_j] + b_c)W_c(o_{j-1} \odot \tanh'(c_{j-1}))i_j$$

Uvrštavanjem nazad u jednadžbu za gradijent dobijamo:

$$\frac{\partial J_t}{\partial W} = \frac{\partial J_t}{\partial h_t}\frac{\partial h_t}{\partial c_t}(\prod_{j=k+1}^{t}(A_j + B_j + C_j + D_j))\frac{\partial c_k}{\partial W}$$

U standardnim rekurentnim mrežama, parcijalna derivacija $\frac{\partial h_{j+1}}{\partial h_j}$ će na posljetku poprimiti vrijednosti uvijek veće od 1 ili uvijek manje od 1 te je to ono što dovodi do problema eksplozije/iščezavanja. Ono što je drukčije kod LSTM-a je to što $\frac{\partial c_j}{\partial c_{j-1}}$ u bilo kojem koraku može

poprimiti vrijednosti i manje i veće od 1, pa ako proširimo broj koraka u beskonačnost nema garancije da ćemo završiti u 0. Ako krenemo konvergirati u 0, linearni članovi se uvijek mogu podesiti tako da nam gradijent ode bliže 1, što spriječava iščezavanje gradijenta. Za primjer uočimo da naš produkt sadrži B_j što je zapravo naš sklop zaboravljanja f_j pa to omogućuje mreži kontrolu nad vrijednosti gradijenta ažuriranjem parametara sklopa zaboravljanja.

Primjedba 3.1. *Linearni članovi u spomenutom produktu ne moraju se u svakom koraku "slagati" što znači da je vjerojatnost da gradijent iščezne vrlo mala te mreža uči koje vrijednosti će iščeznuti.*

Primjedba 3.2. *Važno je uočiti da LSTM ne rješava problem eksplodirajućeg gradijenta, pa se često uz LSTM koristi i gradient clipping.*

3.3. Varijante LSTM-a

Arhitektura koju smo objasnili je standardna LSTM arhitektura, ali nisu sve LSTM izvedbe iste. Naprotiv, skoro svaki rad koristi izmjenjenu verziju standardnog LSTM-a. Iako su promjene male, vrijedi istaknuti neke od najpoznatijih.

Jednu poznatu verziju "peephole connections" uveli su Gers i Schmidhuber [2] dodavajući stanja ćelija kod računanja vrijednosti sklopova:

$$f_t = \sigma(W_f[C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i[C_{t-1}, h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o[C_t, h_{t-1}, x_t] + b_o)$$

Mnogi papiri umjesto dodavanja stanja ćelije kod računanja svih sklopova dodaju samo na neke sklopove.

Još jedna varijacija je povezivanje sklopa zaboravljanja i sklopa ulaza na način da umjesto odvojenog biranja što će se zaboraviti a što dodati, odluku sklopovi donose zajedno. Drugim riječima, zaboravljamo samo kada dodajemo nove informacije.

$$c_t = f_t \odot c_{t-1} + (1 - f_t) \odot \hat{c}_t$$

Najpoznatija verzija LSTM-a je takozvani **Gated recurrent unit** koju je uveo Cho i suradnici [1]. On kombinira sklop zaboravljanja i sklop ulaza u jedan sklop ažuriranja. Osim toga, kombinira stanje ćelije i skriveno stanje te radi još neke promjene. Rezultat toga je puno jednostavniji model od standardnog LSTM modela. GRU model se može opisati sljedećim jednadžbama:

$$z_t = \sigma(W_z[h_{t-1}, x_t])$$
$$r_t = \sigma(W_r[h_{t-1}, x_t])$$
$$\hat{h_t} = \tanh(W[r_t h_{t-1}, x_t])$$
$$h_t = (1 - z_t)h_{t-1} + z_t\hat{h_t}$$

Ovo su samo neke od mnogih varijanti LSTM-a. Ovisno o problemu neke varijante se pokazuju kao bolje, ali nema čistog "pobjednika". Greff i suradnici [3] su pokazali kako su sve varijante približno jednake, dok su Jozefowicz i suradnici [6] testirali više desetaka tisuća RNN arhitektura te pokazali da neke rade bolje od LSTM-a na određenim zadacima.

Primjer 3.1. *Za primjer ćemo kontriurati LSTM i jednostavni RNN model kao rješenje problema Named Entity Recognition-a. NER problem svodi se na određivanje oznake za riječi u rečenici. Moguće oznake su: B-geo, I-art, B-org, I-nat, I-geo, B-art, I-per, I-gpe, B-nat, O, B-per, B-eve, I-tim, B-gpe, B-tim, I-eve, I-org. Za treniranje koristit ćemo korpus za NER* [4].

	Sentence #	Word	POS	Tag
10	NaN	war	NN	O
11	NaN	in	IN	O
12	NaN	Iraq	NNP	B-geo
13	NaN	and	CC	O
14	NaN	demand	VB	O
15	NaN	the	DT	O
16	NaN	withdrawal	NN	O
17	NaN	of	IN	O
18	NaN	British	JJ	B-gpe
19	NaN	troops	NNS	O

Slika 7: Retci iz korpusa za NER

Kako bi mogli trenirati model trebamo podatke pretvoriti u nešto mreži čitljivo. Prvo ćemo razdvojiti naše retke u rečenice (nizove parova riječi i oznaka). Zatim kreiramo riječnik za riječi i oznake te svim riječima/oznakama dodijeljujemo odgovarajući cijeli broj. Kako svi podaci u batchu moraju biti istih duljina[5]*, a naše rečenice su različitih duljina, koristit ćemo takozvani sequence padding* [6] *kako bi postavili sve podatke na istu duljinu.*

Sada nam samo preostaje definiranje i treniranje modela. Naš model će se sastojati od embedding sloja koji naše cijele brojeve pretvara u float vektore smislenije za model. Nakon njega ide glavni (rekurentni) sloj. Zadnji je gusti sloj "umotan" u sloj koji omogućava neovisno primjenjivanje našeg rekurentnog sloja na svaku riječ. Definicije tih modela izgledaju ovako:

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 104, 50)	175890
simple_rnn_1 (SimpleRNN)	(None, 104, 64)	7360
time_distributed_6 (TimeDistributed)	(None, 104, 17)	1105

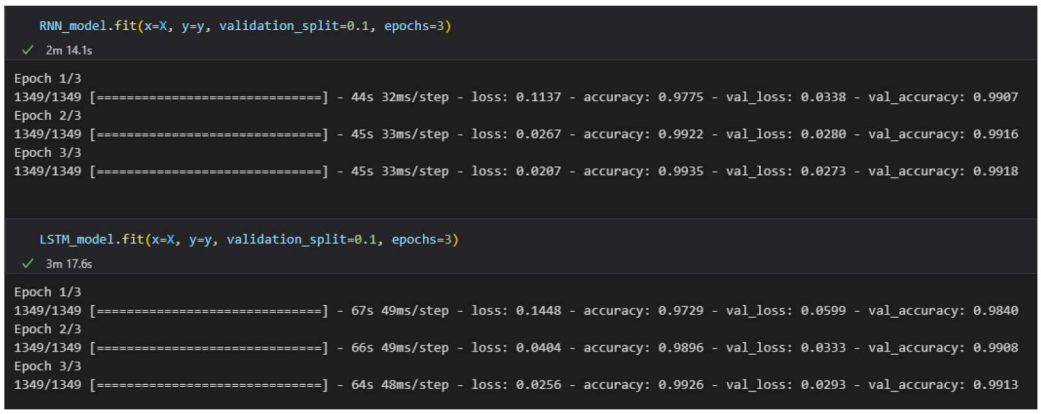
Total params: 1767365 (6.74 MB)
Trainable params: 1767365 (6.74 MB)
Non-trainable params: 0 (0.00 Byte)

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 104, 50)	1758900
lstm_5 (LSTM)	(None, 104, 64)	29440
time_distributed_7 (TimeDistributed)	(None, 104, 17)	1105

Total params: 1789445 (6.83 MB)
Trainable params: 1789445 (6.83 MB)
Non-trainable params: 0 (0.00 Byte)

Jedino što preostaje je treniranje modela nad podacima koje smo pripremili:

[4]https://www.kaggle.com/datasets/abhinavwalia95/entity-annotated-corpus
[5]Drugo rješenje bilo bi konstrutiranje batcheva po veličini rečenica.
[6]"Punjenje" podatka s vrijednostima koje minimalno utječu na treniranje.



Slika 9: Rezultati treniranja

Kao što vidimo, oba modela su izrazito dobra u rješavanju problema NER-a te postižu preciznost veću od 99%.

Primjedba 3.3. *Iako se jednostavan RNN pokazao jednako dobar kao i LSTM kod problema NER-a, LSTM ima puno veći značaj u težim problemima (npr. prevođenje teksta) gdje model mora "hvatati" daleke zavisnosti između podataka.*

Literatura

[1] K. Cho i sur., *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014.

[2] F. Gers, J. Schmidhuber, *Recurrent nets that time and count*, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, Como, Italy, 2000.

[3] K. Greff i sur., *LSTM: A Search Space Odyssey*, IEEE Transactions on Neural Networks and Learning Systems (Volume: 28, Issue: 10), 2017.

[4] S. Hochreiter, J. Schmidhuber, *Long Short-Term Memory*, Neural Computation (1997) 9(8):1735-1780, Massachusetts Institute of Technology, 1997.

[5] A. Jovanović, *Primjena rekurentnih neuronskih mreža u analizi programskog koda*, Sveučilište J.J.Strossmayera u Osijeku, Odjel za matematiku, Osijek, Hrvatska, 2021.

[6] R. Jozefowicz i sur., *An Emperical Exploration of Recurrent Network Architectures*, ICML'15: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, 2015.

[7] M. Nielsen i sur., *http://neuralnetworksanddeeplearning.com/*, 2019. (2.9.2023.)

[8] R. Pascanu, T. Mikolov, Y. Bengio, *On the difficulty of training recurrent neural networks*, Proceedings of the 30th International Conference on Machine Learning, PMLR 28(3):1310-1318, 2013.

[9] F. Rosenblatt *The perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review, 65(6), 386-408., 1958.

[10] R. Scitovski, N. Truhar, Z. Tomljanović, *Metode optimizacije*, Sveučilište J.J.Strossmayera u Osijeku, Odjel za matematiku, Osijek, Hrvatska, 2014.

[11] P. J. Werbos, *Backpropagation through time: What it does and how to do it*, Proceedings of the IEEE (Volume: 78, Issue: 10), National Science Foundation, Washington D.C., USA, 1990.