

Duboko učenje

Optimizacija parametara modela.

doc.dr.sc. Marko Čupić

Fakultet elektrotehnike i računarstva
Sveučilište u Zagrebu
Akademska godina 2017./2018.

5. travnja 2019.

Layout

1 Uvod

- Gdje smo
- Razlika između učenja i optimizacije

2 Izazovi

3 Osnovni algoritmi

- Stohastički gradijentni spust

4 Inicijalizacija parametara

5 Algoritmi s adaptivnim stopama učenja

6 Meta-algoritmi

Optimizacija kod dubokih modela

Razmatramo optimizaciju parametara kod neuronskih mreža

- najčešće postupci koji se temelje na iterativnom korigiranju parametara oslanjajući se na gradijent
- razlika u odnosu na klasični optimizacijski problem
 - želimo optimirati mjeru P koja često može biti netraktabilna
 - umjesto nje optimiramo mjeru $J(\Theta)$ u nadi da ćemo indirektno optimirati i mjeru P
- Primjerice:

$$P = J^*(\Theta) = \mathbb{E}_{(\mathbf{x}, y) \sim p_{data}} L(f(\mathbf{x}; \Theta), y) \quad (1)$$

$$J(\Theta) = \mathbb{E}_{(\mathbf{x}, y) \sim \hat{p}_{data}} L(f(\mathbf{x}; \Theta), y) \quad (2)$$

gdje je p_{data} distribucija koja je generirala podatke, \hat{p}_{data} empirijska distribucija izračunata na temelju skupa za učenje, $f(\mathbf{x}; \Theta)$ predviđen izlaz za uzorak \mathbf{x} , L funkcija gubitka

Layout

1 Uvod

- Gdje smo
- Razlika između učenja i optimizacije

2 Izazovi

3 Osnovni algoritmi

- Stohastički gradijentni spust

4 Inicijalizacija parametara

5 Algoritmi s adaptivnim stopama učenja

6 Meta-algoritmi

Minimizacija empirijskog rizika

- cilj strojnog učenja je minimizirati očekivanu pogrešku generalizacije danu s (1) (*rizik*)
- kada bismo je znali, imali bismo klasičan optimizacijski problem
- ne znamo p_{data} već imamo samo skup uzoraka za učenje → problem strojnog učenja
- problem strojnog učenja možemo svesti na optimizacijski problem tako da procjenimo pravu distribuciju na temelju empirijske distribucije \hat{p}_{data} određene uzorcima za učenje
- minimiziramo *empirijski rizik*:

$$\mathbb{E}_{(\mathbf{x}, y) \sim \hat{p}_{data}} L(f(\mathbf{x}; \Theta), y) = \frac{1}{m} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}; \Theta), y^{(i)}) \quad (3)$$

Minimizacija empirijskog rizika

- nadamo se da će smanjenjem empirijskog rizika padati i očekivani gubitak uzet po svim uzorcima iz izvorne distribucije P_{data}
- opasnost: ovaj postupak sklon je *prenaučenosti* (engl. *overfitting*)
 - modeli dovoljnog kapaciteta mogu "zapamtiti" skup uzoraka za učenje a nemati dobru generalizaciju
- minimizacija empirijskog rizika ponekad nije izvediva
 - gradijentni spust i funkcija gubitka 0-1 (nema korisnih derivacija)
- stoga u praksi optimiramo mjeru koja je još različitija od očekivanog gubitka na izvornoj distribuciji

Nadomjesna funkcija gubitka.

- često željenu funkciju gubitka (npr. pogrešku klasifikacije) ne možemo optimirati efikasno
- rješenje je optimirati nadomjesnu funkciju gubitka (engl. *surrogate loss function*) koja ima povoljnija svojstva
 - primjerice, negativnu log-izglednost točnog razreda uzorka uzimamo kao zamjenu za 0-1-gubitak
 - dopušta modelu da procijeni uvjetne vjerojatnosti razreda s obzirom na dani uzorak
 - ako model to može raditi dobro, onda može odabirati i razrede na način koji minimizira očekivanu pogrešku klasifikacije
 - uočite: ova nadomjesna funkcija može i više od izvorne; kad izvorna dosegne 0, nadomjesna i dalje nastavlja padati još neko vrijeme jer utvrđuje uvjetne vjerojatnosti i gradi robusniji klasifikator

Rano zaustavljanje.

- klasični optimizacijski postupak zaustavlja se tek kada dođe u lokalni optimum (*stagnacija*)
- algoritmi strojnog učenja staju prije toga, kada se ispunи *uvjet ranog zaustavljanja*
 - primjerice, optimiramo negativnu log-izglednost, ali računamo izvornu funkciju gubitka (0-1-gubitak) nad skupom za provjeru (engl. *validation set*) i stajemo kada utvrđimo prenaučenost
 - u tom trenutku gradijenti nadomjesne funkcije gubitka mogu i dalje biti veliki (nismo u lokalnom optimumu)
 - kod klasičnog optimizacijskog problema, kažemo da je algoritam konvergirao kada gradijenti isčeznu (postanu bliski nuli)

Pojedinačno učenje, grupno učenje i nešto između

- kod algoritama strojnog učenja, funkcija gubitka tipično je suma gubitaka po svakom od uzoraka iz skupa za učenje
- primjerice, kod problema maksimalne izglednosti, promatramo vjerojatnost da smo uz modeliranu distribuciju određenu parametrima Θ "izvukli" skup uzoraka za učenje s pripadnim oznakama razreda:

$$p_{\text{skup za učenje}} = \prod_{i=1}^m p_{\text{model}}(\mathbf{x}^{(i)}, y^{(i)}; \Theta)$$

Pojedinačno učenje, grupno učenje i nešto između

- u log-prostoru, logaritam umnoška je suma logaritama pa imamo:

$$\log(p_{\text{skup za učenje}}) = \sum_{i=1}^m \log p_{\text{model}}(\mathbf{x}^{(i)}, y^{(i)}; \Theta)$$

odnosno problem se pretvara u:

$$\Theta_{ML} = \operatorname{argmax}_{\Theta} \sum_{i=1}^m \log p_{\text{model}}(\mathbf{x}^{(i)}, y^{(i)}; \Theta) \quad (4)$$

Pojedinačno učenje, grupno učenje i nešto između

- maksimizacija sume u izrazu (4) ekvivalentna je maksimizaciji očekivanja nad empirijskom distribucijom definiranom nad skupom za učenje:

$$J(\Theta) = \mathbb{E}_{(\mathbf{x}, y) \sim \hat{p}_{data}} \log p_{\text{model}}(\mathbf{x}, y; \Theta) \quad (5)$$

- Mnoga svojstva od $J(\Theta)$ odgovaraju očekivanjima nad skupom za učenje; npr. gradijent:

$$\nabla_{\Theta} J(\Theta) = \mathbb{E}_{(\mathbf{x}, y) \sim \hat{p}_{data}} \nabla_{\Theta} \log p_{\text{model}}(\mathbf{x}, y; \Theta) \quad (6)$$

Pojedinačno učenje, grupno učenje i nešto između

- izračun gradijenta prema izrazu (6) zahtjeva prolaz kroz čitav skup uzoraka za učenje: skupo!
- očekivanje možemo aproksimirati slučajnim uzorkovanjem podskupa iz skupa za učenje i potom uzimanjem srednje vrijednosti nad tim uzorcima
- prisjetite se distribucije uzorkovanja i standardne pogreške (standardnog odstupanja računate statistike nad uzorcima; <http://stattrek.com/sampling/sampling-distribution.aspx>)

Pojedinačno učenje, grupno učenje i nešto između

- kod uzorkovanja, standardna pogreška srednje vrijednosti izračunate na temelju n uzoraka je σ/\sqrt{n} gdje je σ standardno odstupanje nad čitavim skupom
- uzmemo li uzorak veličine $n = 100$ ili $n = 10000$, posljednji daje za faktor 10 manju pogrešku ali zahtjeva 100 puta više izračuna - ne isplati se
- algoritmi često rade dobro (ponekad i uz bržu konvergenciju) i s "lošijim" aproksimacijama gradijenta koji češće dobivaju i mogu koristiti za korekcije parametara
- dodatno, skup uzoraka za učenje često sadrži redundancije - slične podatke koji daju slične gradijente - ne moramo ih sve uzeti u obzir

Pojedinačno učenje, grupno učenje i nešto između

- optimacijski postupci koji koriste čitav skup uzoraka za učenje nazivaju se *grupni* (engl. *batch*)
- optimacijski postupci koji koriste jedan po jedan uzorak iz skupa uzoraka za učenje nazivaju se *pojedinačni* (engl. *online*) ili stohastički (engl. *stochastic*)
- optimacijske postupke koji u jednom koraku koriste podskup skupa uzoraka za učenje nazivamo postupcima s mini-grupama (engl. *minibatch*), gdje je broj uzoraka određen parametrom koji nazivamo veličina grupe (engl. *batch size*).

Pojedinačno učenje, grupno učenje i nešto između

Utjecaj veličine grupe:

- veća grupa → precizniji gradijent (ispodlinearo poboljšanje!)
- na višeprocesorskim sustavima, veličina grupe ne bi smjela biti premala (loša uporaba jezgri)
- ako se svi uzorci grupe trebaju obraditi paralelno (GPU?), potrošnja memorije skalira se s veličinom grupe - može biti ograničenje!
- neke arhitekture sklopovlja (tipa GPU) najbolje rade s veličinama grupe koje su potencije broja 2; tipično 32 do 256; ponekad 16 za velike modele
- male veličine grupe mogu ponuditi regularizacijski efekt (dodaju šum zbog nepreciznog gradijenta) → poboljšava generalizaciju (najbolje: 1; treba malu stopu učenja; dugotrajno učenje)

Pojedinačno učenje, grupno učenje i nešto između

Algoritmi vs. veličina grupe:

- različiti algoritmi zahtjevaju različite veličine grupe (pitanje je koju informaciju koriste i koliko se ona precizno može odrediti iz uzorka/grupe)
- gradijentni spust treba samo gradijent: robustno i radi i s malim veličinama grupe
- algoritmi koji trebaju informacije drugog reda (npr. Hesseovu matricu): trebaju veće grupe (ali ih obično NE koristimo s modelima koji imaju puno parametara)

Pojedinačno učenje, grupno učenje i nešto između

Kako raditi odabir uzorka za pojedine grupe?

- uzorci za grupu trebali bi biti odabrani slučajnim uzorkovanjem kako bi bili nezavisni (želimo odrediti nepristranu procjenu gradijenta)
- dvije uzastopne procjene gradijenta također bi trebale biti nezavisne: za svaku bismo trebali nanovo raditi uzorkovanje
- mnogi skupovi podataka imaju uzorke koji su poredani po nekom kriteriju (zavisni)
- često se kao (učinkovito) rješenje napravi početno permutiranje uzorka u skupu za učenje i tim se redoslijedom dalje grade mini-grupe koje se potom neprestano iskorištavaju

Pojedinačno učenje, grupno učenje i nešto između

Važno svojstvo algoritma s mini-grupama:

- gradijent koji koristi odgovara gradijentu prave pogreške generalizacije (izraz (1); samo ako se uzorci ne ponavljaju)
- u praksi: napravi se permutacija skupa uzoraka za učenje i koristi ga se više puta
 - u prvom prolazu uzorci za mini-grupe su nezavisni pa daju gradijent prave generalizacijske pogreške
 - u sljedećem prolazu uzorci više nisu nezavisni: gradijent se ponavlja za već viđene uzorke već viđenim redoslijedom
 - ako je izvedivo: periodički ponovno permutirati skup uzoraka za učenje

Loše-kondicioniran H

Prisutan i kod učenja neuronskih mreža

- kako se manifestira: čak i mali koraci u smjeru (minus)gradijenta povećavaju funkciju gubitka
- može se pokazati rastavom funkcije gubitka u Taylorov red da pomak iznosa $-\epsilon \mathbf{g}$ funkciji dodaje iznos:

$$\frac{1}{2}\epsilon^2 \mathbf{g}^T H \mathbf{g} - \epsilon \mathbf{g}^T \mathbf{g}$$

- problem nastupa ako je $\frac{1}{2}\epsilon^2 \mathbf{g}^T H \mathbf{g}$ veća od $\epsilon \mathbf{g}^T \mathbf{g}$
- motriti normu gradijenta ($\mathbf{g}^T \mathbf{g}$) te član $\mathbf{g}^T H \mathbf{g}$ tijekom učenja
- norma gradijenta često ne pada značajno tijekom učenja, ali $\mathbf{g}^T H \mathbf{g}$ može rasti za više redova veličine što treba kompenzirati vrlo malim stopama učenje - spor napredak!

Lokalni minimumi

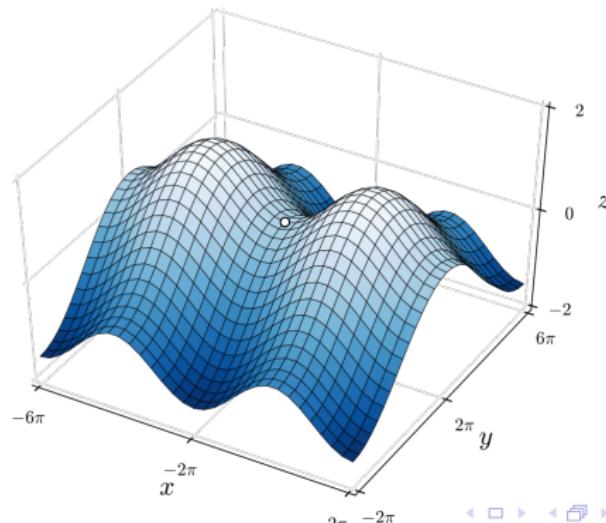
Funkcija gubitka kod neuronskih mreža nije konveksna: ima mnoštvo lokalnih minimuma

- čini se da nije toliko značajan problem kod većih modela
- od kuda dolaze lokalni optimumi? Imaju ih svi koji imaju problem identifikacije modela (model se može identificirati ako uz dovoljan skup podataka možemo jednoznačno odrediti parametre modela)
 - simetričnost prostora težina (engl. *weight space symmetry*)
 - kod ReLU i maxout neurona ulaze i pomak možemo skalirati s α a izlaz s $\frac{1}{\alpha}$ (za bilo koji $\alpha \neq 0$)
- svi prethodno navedeni su jednaki s obzirom na funkciju gubitka
- ako model ima lokalnih minimuma kod kojih je funkcija gubitka bitno veća od iznosa u globalnom minimumu, i ako takvih ima mnogo → problem! (ima li ih mnogo?)

Platoi, sedla i drugo

Kod višedimenzijskih nekonveksnih funkcija, mnogo više od lokalnih minimuma ima točaka koje su sedla

- gradijent u sedlu je također nula

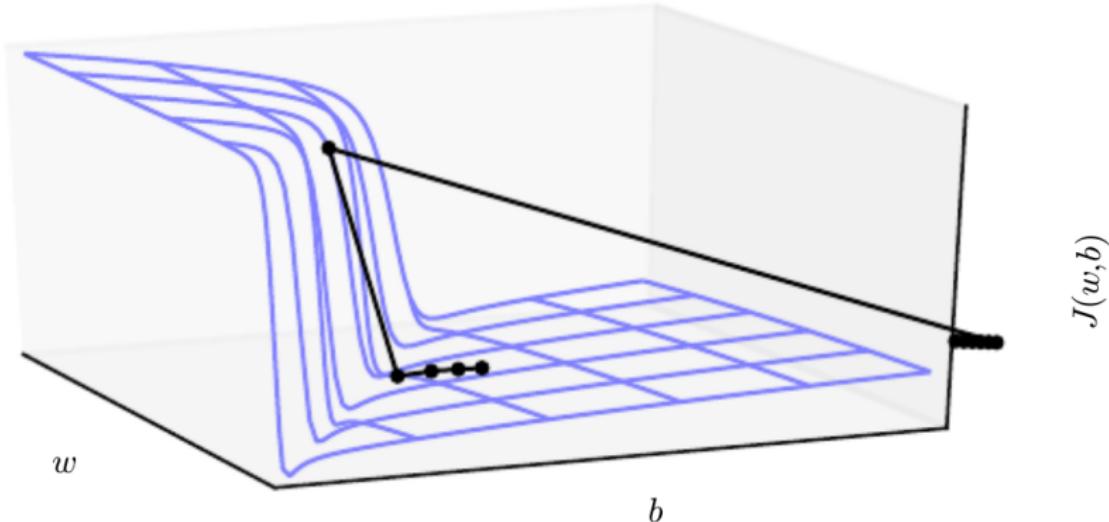


Platoi, sedla i drugo

- za postupke koji koriste informaciju prvog reda (gradijent), sedla često nisu problem, posebice stohastički algoritmi
- za postupke koji koriste informaciju drugog reda (H), sedla jesu problem (primjerice, Newtonova metoda koja upravo traži točke u kojima je gradijent 0)
 - jedan od razloga zašto nisu uspjele zamijeniti gradijentne metode kod učenja neuronskih mreža
- uz lokalne minimume i sedla, moguća su i široka područja na kojima je funkcija gubitka konstantna (gradijent i H su nula)
→ to je problem!

Litice i eksplodirajući gradijenti

Neuronske mreže s mnogo slojeva mogu imati strme regije u parametarskom prostoru: ažuriranje proporcionalno gradijentu može postupak optimizacije odbaciti daleko u jednom koraku.



Litice i eksplodirajući gradijenti

- na litici gradijent "eksplodira" (magnutida mu značajno poraste)
- problem se može riješiti jednostavnom heuristikom:
odsijecanjem gradijenta (engl. *gradient clipping*) - kada je gradijent prevelik, smanji se stopa učenja kako bi napravljeni korak ostao razumno mali

Dugotrajne ovisnosti

- problem kada imamo dubok izračunski graf u kojem se ponavljaju operacije (kod povratnih neuronskih mreža): radimo opetovano množenje s istom matricom W što je nakon t koraka ekvivalentno množenju s W^t
- malo teorije (matrice)
 - Neka za matricu W možemo napraviti dijagonalizaciju i to svojstvenu-dekompoziciju:

$$W = V \operatorname{diag}(\lambda) V^{-1}$$

tada je:

$$W^t = V \operatorname{diag}(\lambda)^t V^{-1}$$

- svojstvene vrijednosti λ_i koje nisu bliske 1 ili će eksplodirati ili će nestati

Dugotrajne ovisnosti

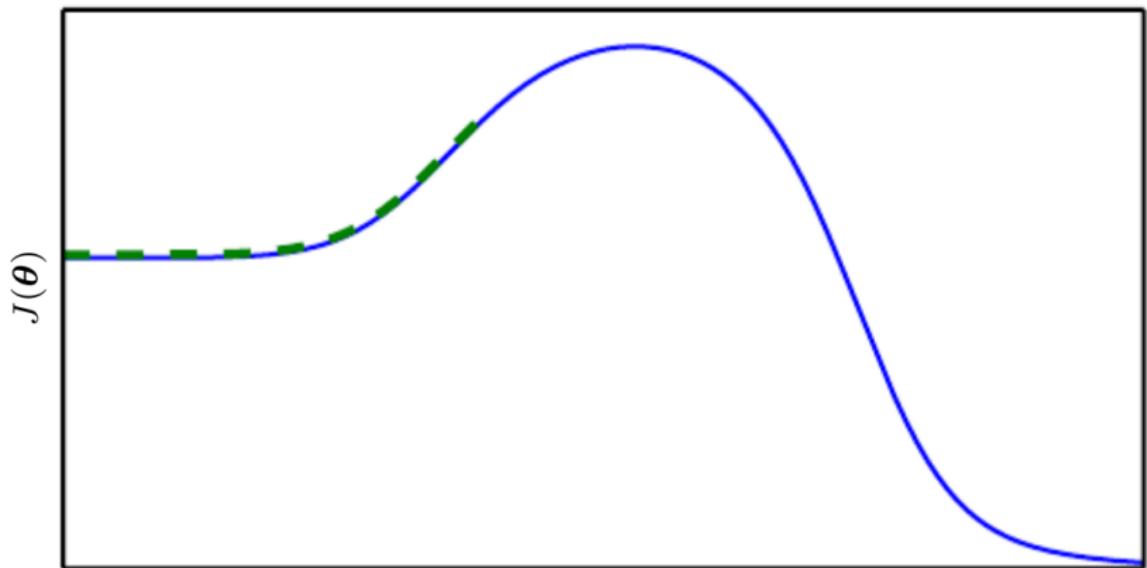
- gradijenti u takvom grafu se također skaliraju s $\text{diag}(\lambda)^t$:
 - mogu eksplodirati (engl. *exploding gradient problem*)
 - mogu nestati (engl. *vanishing gradient problem*)

Neprecizni gradijenti

- optimizacijski algoritmi prepostavljaju da možemo točno odrediti gradijent (a po potrebi i H); u praksi: uzorkujemo i aproksimiramo
- funkcija gubitka koju minimiziramo može biti netraktabilna → gradijent je također → aproksimiramo
- optimizacijski algoritmi za neuronske mreže uzimaju u obzir te činjenice

Nesklad lokalne i globalne strukture

Pogledajte sliku: što kaže "lokalna" informacija - kamo dalje, a gdje je globalni minimum?



Nesklad lokalne i globalne strukture

- gradijentni spust je algoritam koji radi male lokalne korake - za njega opisano je problematično
- postupak učenja često traje dugo jer algoritam u širokim lukovima zaobilazi područja visokog iznosa funkcije gubitka (trajektorija je dugačka)
- lokalna struktura može davati "pogrešan" naputak: lokalno poboljšavanje udaljava nas od globalnog optimuma
- lokalna struktura može biti bez informacije (plato)
- općenito otvoreno pitanje
- može se "zaobići" tako da pronađemo dobre početne vrijednosti parametara od kojih postoji povezan put prema globalnom optimumu: fokus će biti na tome!

Layout

1 Uvod

- Gdje smo
- Razlika između učenja i optimizacije

2 Izazovi

3 Osnovni algoritmi

- Stohastički gradijentni spust

4 Inicijalizacija parametara

5 Algoritmi s adaptivnim stopama učenja

6 Meta-algoritmi

Stohastički gradijentni spust

Temeljni optimizacijski algoritam (engl. *Stochastic Gradient Descent*, SGD)

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k .

Require: Initial parameter θ

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Apply update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

end while

Stohastički gradijentni spust

- kod grupne inačice gradijentnog spusta, kako optimizacija napreduje, norma gradijenta postaje sve manja i manja
 - u (lokalnom/globalnom) optimumu gradijent postaje 0
 - algoritam može koristiti fiksnu stopu učenja
- SGD pak u svakom koraku procjenjuje iznos gradijenta
 - stoga uvijek postoji šum
 - što je uzorak na temelju kojeg radimo procjenu manji, šum je veći
 - u (lokalnom/globalnom) procijenjeni gradijent stoga ne pada na nulu

Stohastički gradijentni spust

- Da bi optimizacija SGD-om konvergirala, stoga je potrebno koristiti promjenjivu stopu učenja
 - kroz vrijeme, stopu je potrebno smanjivati čime se guši utjecaj šuma u procjeni gradijenta
- Može se pokazati da je dovoljan uvjet za konvergenciju da vrijedi:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty$$

$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

Stohastički gradijentni spust

Praktično rješenje:

- stopa se linearno smanjuje od ϵ_0 u koraku 0 do ϵ_τ gdje je τ korak nakon kojeg smanjivanje prestaje
- za $k > \tau$, stopa ostaje na posljednjoj vrijednosti i dalje je ne mijenjamo
- u koraku $k \leq \tau$ vrijedi:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau$$

gdje je $\alpha = \frac{k}{\tau}$

Stohastički gradijentni spust

Kako odabrati ϵ_0 , ϵ_τ i τ : iskustvena pravila

- τ postaviti tako da odgovara nekoliko stotina prolaza kroz čitav skup uzoraka za učenje
- ϵ_τ postaviti na 1% od ϵ_0
- kako odabrati ϵ_0 ?
 - prevelik: postupak će oscilirati, biti nestabilan ili čak divergirati
 - mala: postupak će napredovati vrlo sporo
 - premala: postupak može i zaglaviti rano u vrlo lošem lokalnom minimumu
 - iskustvo: postaviti na nešto veću od one koja u prvih 100 iteracija daje najbolje ponašanje (treba se igrati!)

Stohastički gradijentni spust

- kod SGD s mini-grupama odnosno pojedinačnog, količina izračuna po svakom ažuriranju parametara nije funkcija veličine skupa za učenje
- omogućava konvergenciju čak i kod velikih skupova
- kod jako velikih skupova omogućava konvergenciju prema konačnoj pogrešci na skupu za validaciju (do neke tolerancije) i prije no što obradi čitav skup uzoraka za učenje

Uporaba restarta kod promjene stope učenja

- Loshchilov, Hutter: *SGDR: Stochastic Gradient Descent With Warm Restarts* (ICLR 2017)
- Ugraditi plan restarta na promjenu stope učenja
- Dobiveni state-of-the-art rezultati na CIFAR-10 i CIFAR-100
- Učenje ostvariti kao niz pokretanja algoritma; indeks i je redni broj pokretanja
- T_i je broj epoha (prolazaka kroz čitav skup za učenje) koje radimo u i -tom pokretanju
- T_{cur} je brojač odrađenih epoha u pokretanju (povećava se sa svakom minigrupom - decimalan broj)
- t je broj trenutne iteracije (inkrement +1 za svaku odrađenu minigrupu)

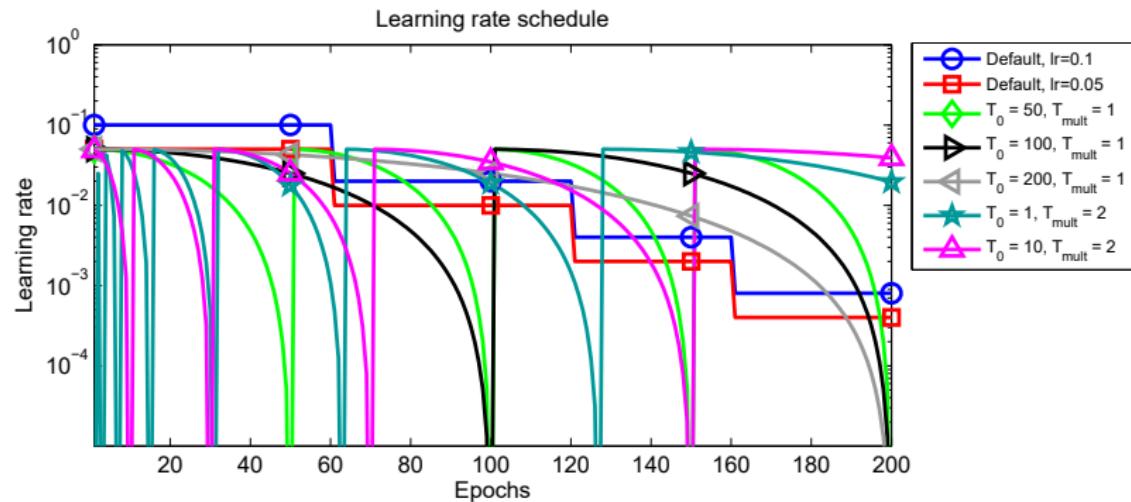
Uporaba restarta kod promjene stope učenja

- Loshchilov, Hutter: *SGDR: Stochastic Gradient Descent With Warm Restarts* (ICLR 2017)
- U i -tom pokretanju u iteraciji t (epohe od pokretanja: T_{cur}) koristi se stopa učenja

$$\eta_t = \eta_{min}^i + \frac{1}{2} (\eta_{max}^i - \eta_{min}^i) \cdot \left(1 + \cos \left(\frac{T_{cur}}{T_i} \cdot \pi \right) \right)$$

- T_i može biti fiksan, ili uz eksponencijalno rastući uz faktor T_{mult} (npr. 2)
- η_{min}^i i η_{max}^i se također s povećanjem i mogu smanjivati
 - niz novih hiperparametara...

Uporaba restarta kod promjene stope učenja



Učenje s momentom

Uporaba momenta koristi se kao metoda za ubrzavanje učenja:

- kod malih ali konzistentnih gradijenata
- kod šumovite procjene gradijenata

Postupak akumulira eksponencijalno umanjujući prosjek prethodnih gradijenata i nastavlja u tom smjeru.

- uvodimo novu varijablu v koja predstavlja eksponencijalno umanjujući prosjek prethodnih (minus) gradijenata
- uvodimo parametar $\alpha \in [0, 1)$ koji govori koliko je relevantan prethodno izračunati prosjek (brzina umanjivanja)
- parametre ažuriramo izračunatim prosjekom
- početna vrijednost je v je 0 (to je vektor jednake dimenzionalnosti kao i θ)

Učenje s momentom

Uporaba momenta koristi se kao metoda za ubrzavanje učenja:

$$v \leftarrow \alpha \cdot v - \epsilon \cdot \nabla_{\theta} \left(\frac{1}{m} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}; \Theta), y^{(i)}) \right) \quad (7)$$

$$\theta \leftarrow \theta + v \quad (8)$$

Što je veći α u odnosu na ϵ , to je veći utjecaj prethodnih gradijenata na smjer ažuriranja.

Korekcija u koraku k ovisi o poravnanju slijeda prethodno izračunatih gradijenata: bit će maksimalna kada su gradijenti poravnnati.

Učenje s momentom

Ako algoritam neprestano dobiva isti gradijent (g):

$$v_0 = -\epsilon \cdot g$$

$$v_1 = -\alpha\epsilon \cdot g - \epsilon \cdot g = -\epsilon \cdot g(\alpha + 1)$$

$$v_2 = -\alpha^2\epsilon \cdot g - \alpha\epsilon \cdot g - \epsilon \cdot g = -\epsilon \cdot g(\alpha^2 + \alpha + 1)$$

$$v_3 = -\alpha^3\epsilon \cdot g - \alpha^2\epsilon \cdot g - \alpha\epsilon \cdot g - \epsilon \cdot g = -\epsilon \cdot g(\alpha^3 + \alpha^2 + \alpha + 1)$$

v_k će biti jednak $-\epsilon \cdot g$ puta suma k -članog geometrijskog reda $(1, \alpha)$; u limesu ta je suma jednaka $\frac{1}{1-\alpha}$, pa će se prosječni gradijent uz moment stabilizirati na:

$$-\frac{\epsilon \cdot g}{1 - \alpha}$$

odnosno "duljina" korak ažuriranja će biti: $\frac{\epsilon \cdot \|g\|}{1 - \alpha}$

Učenje s momentom

Kako je maksimalni korak određen izrazom $\frac{\epsilon \cdot \|g\|}{1-\alpha}$, na α možemo gledati kao na parametar koji određuje za koji će moći faktor povećati korak:

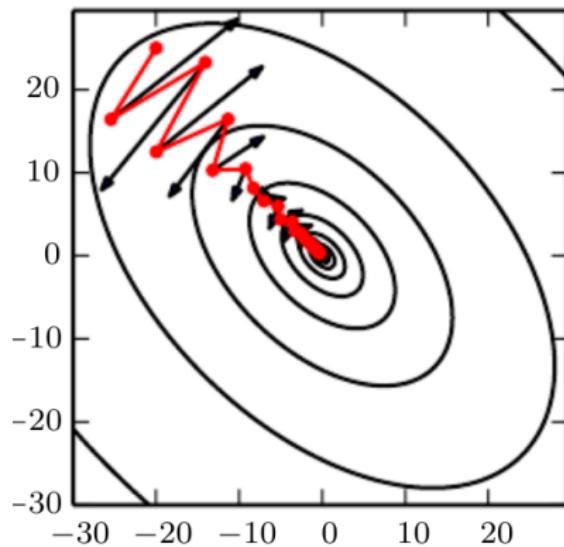
- Neka je $\alpha = 0.99$: $\frac{\epsilon \cdot \|g\|}{1-\alpha} = \frac{\epsilon \cdot \|g\|}{1-0.99} = \frac{\epsilon \cdot \|g\|}{0.01} = 100 \cdot \epsilon \cdot \|g\|$
- Neka je $\alpha = 0.9$: $\frac{\epsilon \cdot \|g\|}{1-\alpha} = \frac{\epsilon \cdot \|g\|}{1-0.9} = \frac{\epsilon \cdot \|g\|}{0.1} = 10 \cdot \epsilon \cdot \|g\|$
- Neka je $\alpha = 0.8$: $\frac{\epsilon \cdot \|g\|}{1-\alpha} = \frac{\epsilon \cdot \|g\|}{1-0.8} = \frac{\epsilon \cdot \|g\|}{0.2} = 5 \cdot \epsilon \cdot \|g\|$
- Neka je $\alpha = 0.5$: $\frac{\epsilon \cdot \|g\|}{1-\alpha} = \frac{\epsilon \cdot \|g\|}{1-0.5} = \frac{\epsilon \cdot \|g\|}{0.5} = 2 \cdot \epsilon \cdot \|g\|$

Učenje s momentom

Parametar α :

- Tipične vrijednosti: 0.5, 0.9, 0.99
- može ga se mijenjati kroz postupak učenja
- početno manje vrijednosti a kasnije podignuti na veće

Učenje s momentom



(crno: bez momenta; crveno: s momentom)

Učenje s momentom

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ , momentum parameter α .

Require: Initial parameter θ , initial velocity v .

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $y^{(i)}$.

 Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), y^{(i)})$

 Compute velocity update: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \mathbf{g}$

 Apply update: $\theta \leftarrow \theta + \mathbf{v}$

end while

Nesterov moment

Modifikacija osnovnog algoritma učenja s momentom:

$$v \leftarrow \alpha \cdot v - \epsilon \cdot \nabla_{\theta} \left(\frac{1}{m} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}; \theta + \alpha v), y^{(i)}) \right) \quad (9)$$

$$\theta \leftarrow \theta + v \quad (10)$$

Gradijent računa ne u trenutnim parametrima θ već u parametrima koji odgovaraju $\theta + \alpha v$. Potom računa prosječni gradijent i radi korekciju izvornih parametara.

Nesterov moment

Algorithm 8.3 Stochastic gradient descent (SGD) with Nesterov momentum

Require: Learning rate ϵ , momentum parameter α .

Require: Initial parameter θ , initial velocity v .

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding labels $\mathbf{y}^{(i)}$.

 Apply interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$

 Compute gradient (at interim point): $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \tilde{\theta}), \mathbf{y}^{(i)})$

 Compute velocity update: $v \leftarrow \alpha v - \epsilon g$

 Apply update: $\theta \leftarrow \theta + v$

end while

Algoritmi za učenje dubokih modela

Algoritmi za učenje dubokih modela:

- iterativni su: korisnik mora ponuditi početnu vrijednost
- loš odabir parametara može onemogućiti algoritam da pronađe prihvatljivo rješenje
- u nekim slučajevima može doći čak i do nestabilnosti i raspada (dijeljenja s nulom i slični problemi)
- ako algoritam konvergira, početno rješenje određuje brzinu i konačno stanje
- različiti parametri koji imaju jednak iznos funkcije gubitka mogu imati potpuno različite generalizacijske pogreške: početni parametri mogu stoga utjecati na konačnu generalizacijsku pogrešku

Algoritmi za učenje dubokih modela

Što želimo:

- pomake (*bias*) postaviti na fiksne heuristički određene vrijednosti
- početne vrijednosti težina moraju razbiti "simetričnost" među jedinicama; u suprotnom algoritam učenja će ih prilagođavati identično ako su spojene na iste ulaze
→ napraviti inicijalizaciju iz slučajnog izvora (Gaussova ili uniformna distribucija): skala je bitna
 - velika: veća šansa da se razbije simetrija i pozitivno djelovanje na pojačavanje signala, ali jedinice dovodimo u zasićenje (sigmoidalna prijenosna funkcija) - problem su gradijenti (0 ili mogu eksplodirati); kod povratnih veza moguće kaotično ponašanje
 - mala: velika šansa za nastanak simetrije

Algoritmi za učenje dubokih modela

Želimo osigurati da je varijanca (σ^2) ulaznih podataka jednaka varijanci transformiranih podataka na izlazu neurona. Stoga se težine izvlače iz normalne distribucije sa srednjom vrijednosti 0 i varijancom oblika:

$$\frac{1}{n}$$

gdje n može biti broj ulaza neurona (n_{in} - čuva varijancu izlaznih podataka), broj izlaza neurona (n_{out} - čuva varijancu gradijenata u povratku) ili njihova aritmetička sredina ($(n_{in} + n_{out})$ - kompromis).

Algoritmi za učenje dubokih modela

Ako umjesto iz normalne distribucije težine izvlačimo iz uniformne, tada će varijanca biti očuvana ako izvlačenje radimo prema:

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{n_{in} + n_{out}}}, \sqrt{\frac{6}{n_{in} + n_{out}}}\right)$$

Algoritmi za učenje dubokih modela

Rezultat da varijanca treba biti $\frac{1}{n}$ vrijedi u pretpostavku da su neuroni sigmoidalni. ReLU neuroni su na pola ulaznog prostora isključeni pa da bi varijanca transformiranih podataka ostala jednaka onoj ulaznih podataka, težine trebamo izvlačiti iz normalne distribucije s varijancom $\frac{2}{n}$.

Često korištena je *Xavier* inicijalizacija:

$$W_{i,j} \sim N(0, \frac{2}{n_{in} + n_{out}}) \text{ ili } W_{i,j} \sim N(0, \frac{1}{n_{in}})$$

Vidi: [http://andyljones.tumblr.com/post/110998971763/
an-explanation-of-xavier-initialization](http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization)

Algoritmi za učenje dubokih modela

TensorFlow nudi `tf.contrib.layers.variance_scaling_initializer` koji radi inicijalizaciju težina uz varijancu oblika:

$$\frac{factor}{n}$$

pri čemu su *factor*, *mode* (FAN_IN, FAN_OUT, FAN_AVG) te željena distribucija parametri.

Xavier-inicijalizaciju dobivamo odabirom: *factor*=1, *mode*=FAN_AVG, *uniform*=False.

Vidi: https://www.tensorflow.org/api_docs/python/tf/contrib/layers/variance_scaling_initializer

Algoritmi za učenje dubokih modela

Postoji i `tf.contrib.layers.xavier_initializer` koji je izravna implementacija Xavier-inicijalizacije i omogućava biranje normalne ili uniformne distribucije pa koristi

$$W_{i,j} \sim N(0, \frac{2}{n_{in} + n_{out}})$$

ili

$$W_{i,j} \sim U\left(-\sqrt{\frac{6}{n_{in} + n_{out}}}, \sqrt{\frac{6}{n_{in} + n_{out}}}\right)$$

Vidi: https://www.tensorflow.org/api_docs/python/tf/contrib/layers/xavier_initializer

Algoritmi za učenje dubokih modela

Određivanje pomaka (*bias*):

- za neurone izlaznog sloja, pretpostavimo da su težine zanemarive pa je izlaz određen samo pomakom; želimo pomake uz koje statistika izlaza odgovara statistici izlaza u skupu uzoraka za učenje.
- paziti da jedinice ne dovedemo u zasićenje (primjerice, ReLU su uz $b = 0$ u zasićenju) → postaviti na 0.1 (ali ne u svim scenarijima inicijalizacije težina)
- ponekad izlaz jednog neurona izravno omogućuje ili onemogućuje drugi neuron: početno b postaviti tako da ovaj prvi ima visok izlaz kako bi drugi neuron mogao učiti: u suprotnom je tu ali je mrtav

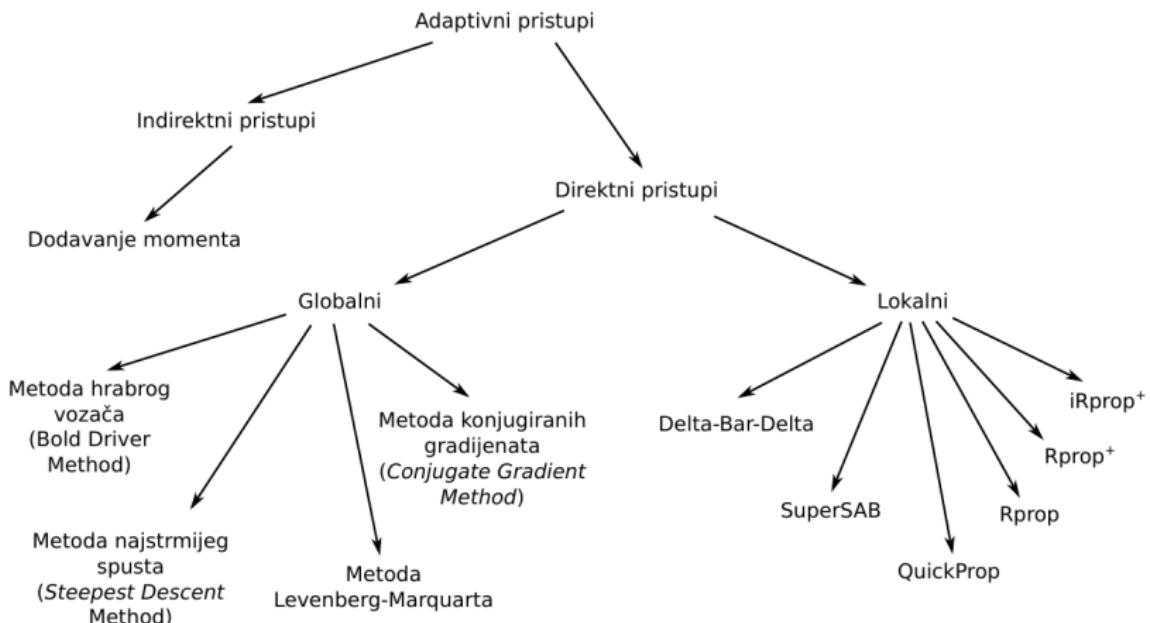
Složeniji postupci inicijalizacije mogu uključivati *predtreniranje* (nenadzirano) ili treniranje mreže da nad istim skupom podataka

Algoritmi s adaptivnim stopama učenja

Stara ideja: umjesto predodređenog načina promjene stope, adaptivno upravljanje stopom učenja!

- algoritmi koji upravljaju globalnom stopom učenja
- algoritmi koji imaju po jednu stopu učenja pridruženu svakom od parametara
 - primjerice Resilient Backpropagation (RProp) i inačice (Riedmiller, Braun; 1993.)

Algoritmi s adaptivnim stopama učenja



(iz knjige za NENR)

Algoritmi s adaptivnim stopama učenja

Danas nešto noviji skup algoritama:

- AdaGrad (Duchi et al., 2011.)
- RMSProp (Hinton, 2012.)
- Adam (Kingma i Ba, 2014.)

AdaGrad

- adaptira stope učenja parametara skalirajući ih inverzno proporcionalno kvadratnom korijenu sume svih kvadriranih vrijednosti pripadne parcijalne derivacije funkcije gubitka kroz povijest
- parametri koji imaju velike parcijalne derivacije brzo će početi koristiti male stope učenja
- parametri koji imaju male parcijalne derivacijeugo vremena će koristiti razumno velike stope učenja
- skaliranje stope je konzistentno sve jače i jače - nema mogućnosti oporavka (suma kvadrata sa svakim novim članom raste)

Problematičan za ne-konveksne probleme (ANN to jesu)

AdaGrad

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter θ

Require: Small constant δ , perhaps 10^{-7} , for numerical stability

Initialize gradient accumulation variable $r = \mathbf{0}$

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Accumulate squared gradient: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$

 Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$. (Division and square root applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while

RMSProp

- mijenja akumuliranje gradijenata tako da dodaje eksponencijalno prigušenje starih vrijednosti - novije vrijednosti imaju veći utjecaj
⇒ ako komponenta gradijenta u nekom trenutku padne, relativno brzo će pasti i njeno prigušenje pa će se korekcije ponovno povećati
- AdaGrad kontinuirano smanjuje gradijente - može biti prebrzo!

Bolji za ne-konveksne probleme

RMSProp

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ , decay rate ρ .

Require: Initial parameter θ

Require: Small constant δ , usually 10^{-6} , used to stabilize division by small numbers.

Initialize accumulation variables $r = 0$

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Accumulate squared gradient: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$

 Compute parameter update: $\Delta\theta = -\frac{\epsilon}{\sqrt{\delta+r}} \odot \mathbf{g}$. ($\frac{1}{\sqrt{\delta+r}}$ applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while

RMSProp + Nesterov momentum

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ , decay rate ρ , momentum coefficient α .

Require: Initial parameter θ , initial velocity v .

Initialize accumulation variable $r = 0$

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{x^{(1)}, \dots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.

 Compute interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$

 Compute gradient: $g \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(x^{(i)}; \tilde{\theta}), y^{(i)})$

 Accumulate gradient: $r \leftarrow \rho r + (1 - \rho)g \odot g$

 Compute velocity update: $v \leftarrow \alpha v - \frac{\epsilon}{\sqrt{r}} \odot g$. ($\frac{1}{\sqrt{r}}$ applied element-wise)

 Apply update: $\theta \leftarrow \theta + v$

end while

Adam

- Adam - od Adaptive Moments
- prati uprosjećeno kretanje gradijenta i kvadrata gradijenta uz eksponencijalno zaboravljanje
- moment je implicitno ugrađen u praćenje kretanja gradijenta
- korekcija se radi prema uprosjećenom gradijentu a ne izračunatom (efekt momenta)

Dosta robusna implementacija koja radi sa širokim skupom parametara.

Adam

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1]$.
(Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default:
 10^{-8})

Require: Initial parameters θ

Initialize 1st and 2nd moment variables $s = \mathbf{0}$, $r = \mathbf{0}$

Initialize time step $t = 0$

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with
 corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

$t \leftarrow t + 1$

 Update biased first moment estimate: $\hat{s} \leftarrow \rho_1 s + (1 - \rho_1) \mathbf{g}$

 Update biased second moment estimate: $\hat{r} \leftarrow \rho_2 r + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$

 Correct bias in first moment: $s \leftarrow \frac{\hat{s}}{1 - \rho_1^t}$

 Correct bias in second moment: $r \leftarrow \frac{\hat{r}}{1 - \rho_2^t}$

 Compute update: $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$ (operations applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while

Adam

- Oprez: za osnovnu inačicu ADAM-a, iako je često puno brži od SGD-a neki radovi pokazuju da algoritam lošije generalizira od SGD-a
- Napravljen niz modifikacija, npr.
 - ADAMAX: drugi moment ne prati kvadrat članova gradijenta već maksimume
 - $r \leftarrow \max(\rho_2 \cdot r, |g|)$; nema korekcije *biasa*
 - u ažuriranju ne dijeli s korijenom već samom vrijednošću
 - NADAM: kombinira ADAM i Nesterov moment
 - AMSGrad: ažuriranje direktno prema eksponencijalno umanjujućem prosjeku kvadrata u nekim primjerima onemogućava konvergenciju
 - prati eksponencijalno umanjujući prosjek kvadrata
 - ažurura prema $\hat{r}_t = \max(\hat{r}_{t-1}, r_t)$

Koji algoritam koristiti

- SGD
- SGD s momentom
- RMSProp
- RMSProp s momentom
- Adam

Nema konsenzusa: koristite koji znate :-)

PS. Adam radi poprilično dobro...

Lokalna normalizacija odziva neurona

- LRN, engl. *Local Response Normalization*
- Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton:
"ImageNet Classification with Deep Convolutional Neural Networks" (2012.)
- osnovna ideja: natjecanje među neuronima može poboljšati rad mreže jer će dovesti do specijalizacije svakog od neurona
- pogledajmo npr. prvi konvolucijski sloj neuronske mreže: imamo N jezgri koje računaju N mapa značajki nad ulaznom slikom
 - svaka mapa značajki je određenih dimenzija $w \times h$ i želimo da se jezgra koja je generira specijalizira za značajku koja je drugačija od svih ostalih značajki

Lokalna normalizacija odziva neurona

- pogledajmo npr. prvi konvolucijski sloj neuronske mreže: imamo N jezgri koje računaju N mapa značajki nad ulaznom slikom
 - neka su mape značajki linearne poredane (znamo koja je prva, druga, treća, ...)
 - promotrimo sada odziv neurona u i -toj mapi značajki na položaju (x, y) (nastao primjenom i -te jezgre i potom propuštanjem kroz ReLU); označimo ga s $a_{x,y}^i$
 - želimo da se taj neuron natječe s n neurona koji su na istom položaju (x, y) u susjednih n mapa značajki (zato smo trebali poredak): gledamo $n/2$ mapa prije promatrane i $n/2$ mapa nakon promatrane

Lokalna normalizacija odziva neurona

- pogledajmo npr. prvi konvolucijski sloj neuronske mreže: imamo N jezgri koje računaju N mapa značajki nad ulaznom slikom
 - radimo normalizaciju odziva promatranog neurona tako da vrijednost dobivenu na njegovu izlazu dijelimo sa sumom kvadrata izlaza čitavog susjedstva, prema izrazu:

$$b_{x,y}^i = \frac{a_{x,y}^i}{\left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} \left(a_{x,y}^j \right)^2 \right)^\beta}$$

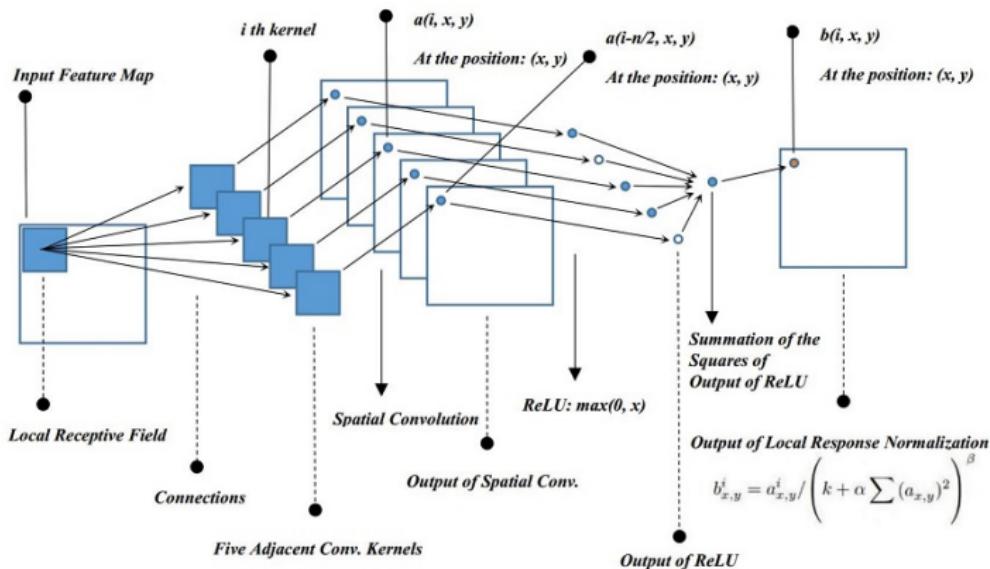
- $b_{x,y}^i$ postavljamo kao "stvarni" izlaz promatranog neurona
- npr. $\beta = 0.75$, $\alpha = 10^{-4}$, $k = 2$, $n = 5$ (hiperparametri)

Lokalna normalizacija odziva neurona

$$b_{x,y}^i = \frac{a_{x,y}^i}{\left(k + \alpha \sum_{j=\max(0, i-n/2)}^{\min(N-1, i+n/2)} (a_{x,y}^j)^2 \right)^\beta}$$

- ako je izlaz $a_{x,y}^i$ malen s obzirom na susjedstvo, normalizacija će ga još potisnuti
- kod dva izlaza koja su sumjerljiva, normalizacija će naglasiti njihovu razliku (potiče specijalizaciju neurona)
- implementacijski, ovo može biti izvedeno kao jedan novi *normalizacijski* sloj mreže čiji su ulazi izlazi prethodnog sloja a izlazi njihove normalizirane vrijednosti

Lokalna normalizacija odziva neurona



(<http://yeephycho.github.io/2016/08/03/Normalizations-in-neural-networks/>)

Normalizacija nad grupom

- Sergey Ioffe, Christian Szegedy (2015.): *Batch Normalization*
- kod dubokih modela, istovremena korekcija svih težina u svim slojevima stvara problem jer izlaz jednog sloja predstavlja ulaz za sljedeći
 - mala promjena na izlazima ranih slojeva može imati velik utjecaj na krajnji izlaz
- Npr. dvoslojna mreža: $y = F_2(F_1(x, \Theta_1), \Theta_2)$ gdje su Θ_1 i Θ_2 parametri
 - promjenom Θ_1 mijenja se ulaz za $F_2(\cdot, \Theta_2)$: gradijent nije izračunat uz te vrijednosti pa ovaj drugi sloj mora naučiti Θ_2 koji dobro pogađaju željeni izlaz i istovremeno ih mijenjati svaki puta kada se Θ_1 mijenja

Normalizacija nad grupom

- Opažanje: praksa pokazuje da neuronske mreže lakše uče nad podatcima koji su normalizirani (preslikani u podatke čija je srednja vrijednost 0 te varijanca 1)
- Primjerice: učite neuronsku mrežu da za sve ulaze veće od 1000 kaže 1, inače 0
 - mreža najprije mora naučiti da porast iznosa ulaza, tako dugo dok nije 1000, nema utjecaja na klasifikaciju
 - tek kad nauči taj prag, mreža može učiti klasificirati prema razlici ulaznog uzorka i tog praga
 - potrebno je vrijeme da mreža tako nešto nauči
 - ako od svih uzoraka oduzmemo srednju vrijednost, pomoći ćemo algoritmu učenja

Normalizacija nad grupom

- Npr. dvoslojna mreža: $y = F_2(F_1(x, \Theta_1), \Theta_2)$ gdje su Θ_1 i Θ_2 parametri
 - ovaj efekt promjene ulaznih podataka za drugi sloj promjenom parametara Θ_1 naziva se *covariate shift*.
 - htjeli bismo na neki način iz mreže izbaciti (ili barem kako ublažiti) ovaj efekt
 - htjeli bismo da izlaz svakog sloja bude ponovno normaliziran, pa sljedeći sloj ima "stabilne" ulaze nad kojima uči, što je lakše
 - normalizaciju bismo mogli raditi naizmjence s gradijentnim spustom, ali praksa pokazuje da to nije dobro
 \Rightarrow želimo rješenje koje je "razumno" računski jeftino i derivabilno tako da se može izravno uključiti u gradijentni spust

Normalizacija nad grupom

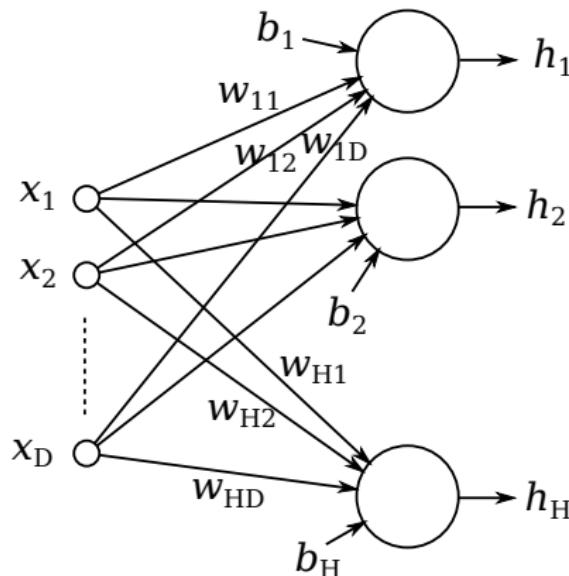
- *Normalizacija nad grupom*: normalizacija izlaza svakog neurona radi se na temelju izlaza koje taj neuron ima kada se na ulaz mreže dovede svaki od uzoraka za učenje iz minigrupe
- znači, ne normaliziraju se:
 - odzivi neurona međusobno unutar istog sloja
 - odzivi neurona u lokalnom susjedstvu
 - ...
- normalizacija ide po ulaznim uzorcima

Normalizacija nad grupom

- Početna ideja: neka smo u minigrupu izvukli uzorke $\{x_1, \dots, x_m\}$; promatrajmo izlaz jednog konkretnog neurona
 - stavimo na ulaz mreže uzorak x_1 , izračunamo izlaze mreže, "snimimo" izlaz promatranog neurona: o_1
 - stavimo na ulaz mreže uzorak x_2 , izračunamo izlaze mreže, "snimimo" izlaz promatranog neurona: o_2
 - ...
 - stavimo na ulaz mreže uzorak x_m , izračunamo izlaze mreže, "snimimo" izlaz promatranog neurona: o_m
- za skup brojeva o_1, \dots, o_m izračunamo srednju vrijednost i standardno odstupanje pa skup preslikamo u normalizirani $\hat{o}_1, \dots, \hat{o}_m$: to postaju izlazi neurona za svaki od ulaznih uzoraka
- na kraju možemo još primijeniti afinu transformaciju nad svakim izlazom nezavisno (više kasnije)

Normalizacija nad grupom: u neuronskoj mreži

Gledajmo jedan sloj višeslojne unaprijedne neuronske mreže:



Normalizacija nad grupom: u neuronskoj mreži

- neka je x vektor-redak koji predstavlja D -dimenzijski ulaz sloja:
$$[\begin{array}{cccc} x_1 & x_2 & \cdots & x_D \end{array}]$$
- težine sloja možemo organizirati u matricu W dimenzija $D \times H$ a pragove u matricu/vektor b dimenzija $1 \times H$:

$$W = \begin{bmatrix} w_{11} & \cdots & w_{H1} \\ \vdots & \ddots & \vdots \\ w_{1D} & \cdots & w_{HD} \end{bmatrix} \quad b = [\begin{array}{ccc} b_1 & \cdots & b_H \end{array}]$$

pa je izlaz $h = xW + b$ (vektor-redak, $1 \times H$):

$$h = [\begin{array}{cccc} x_1 & x_2 & \cdots & x_D \end{array}] \cdot \begin{bmatrix} w_{11} & \cdots & w_{H1} \\ \vdots & \ddots & \vdots \\ w_{1D} & \cdots & w_{HD} \end{bmatrix} + [\begin{array}{ccc} b_1 & \cdots & b_H \end{array}]$$

Normalizacija nad grupom: u neuronskoj mreži

- Pogledajmo sada situaciju gdje imamo mini-grupu od N uzoraka. Dovođenjem svakog od uzoraka iz minigrupe na ulaz mreže, prethodni sloj (koji je ulaz za naš promatrani) će generirati aktivacije neurona koje su ulaz za promatrani sloj (možda je najjednostavnije promatrati sam početak neuronske mreže gdje su te aktivacije upravo sam ulazni podatak)
- Te aktivacije sada možemo pohraniti u matricu X dimenzija $N \times D$:

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1D} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{ND} \end{bmatrix}$$

Normalizacija nad grupom: u neuronskoj mreži

- Sada odjednom možemo izračunati sve izlaze neurona za svaki od ulaznih podataka:

$$h = X \cdot W + b_{\downarrow}$$

da bi izraz bio korektan, vektor-redak b dimenzija $1 \times H$ treba "prema dolje" (zato strelica) iskopirati u matricu dimenzija $N \times H$ ili koristiti operator zbrajanja koji sam zna napraviti to širenje (broadcast)

- ovime množimo matricu $N \times D$ s matricom $D \times H$ (dobivamo $N \times H$) + dodajemo $N \times H$: rezultat je matrica koja u retku i sadrži izlaze svih neurona za i -ti ulazni uzorak/podatak; ima N redaka i H stupaca

Normalizacija nad grupom: u neuronskoj mreži

- Dobivamo matricu h

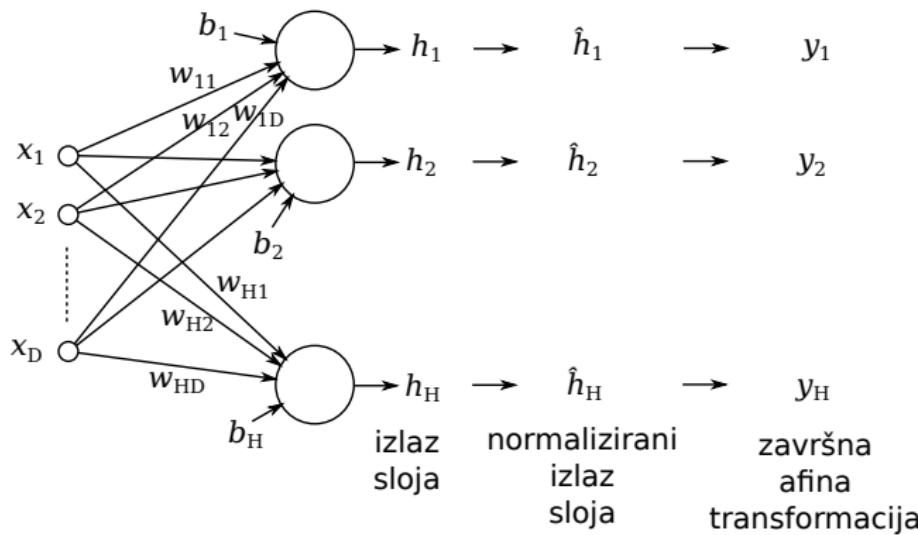
$$h = \begin{bmatrix} h_{11} & \cdots & h_{1H} \\ \vdots & \ddots & \vdots \\ h_{N1} & \cdots & x_{NH} \end{bmatrix}$$

(dimenzije $N \times H$)

- U mreži bismo svaki element još propustili kroz nelinearnost (npr. $\text{ReLU}(.)$); za sada ostanimo ovdje

Normalizacija nad grupom: u neuronskoj mreži

Normalizacija kroz sličicu...



Normalizacija nad grupom: u neuronskoj mreži

- Nakon što smo dobili odzive svih neurona za sve ulaze iz mini-grupe, krećemo u normalizaciju
- Promatramo odzive l -tog neurona za sve ulazne podatke: oni su u l -tom stupcu matrice h
 - Računamo srednju vrijednost tog odziva: $\mu_l = \frac{1}{N} \sum_{p=1}^N h_{p,l}$
 - Računamo standardno odstupanje tog odziva:
$$\sigma_l^2 = \frac{1}{N} \sum_{p=1}^N (h_{p,l} - \mu_l)^2$$
 - Ovo radimo za svaki neuron, tj. za $l \in \{1, \dots, H\}$ pa rezultate možemo spremiti u dva H -dimenzijska vektor-retka: μ i σ^2
 - Potom računamo matricu \hat{h} (dimenzija $N \times H$) koja sadrži normalizirane odzive (normalizacija ide po stupcima):

$$\hat{h}_{k,l} = \frac{h_{k,l} - \mu_l}{\sqrt{\sigma_l^2 + \epsilon}}$$

Normalizacija nad grupom: u neuronskoj mreži

- Nakon što smo dobili normalizirane odzive (matricu \hat{h}), radimo završni korak: izlaze svakog neurona skaliramo i dodajemo fiksni pomak:

$$y_{k,I} = \gamma_I \cdot \hat{h}_{k,I} + \beta_I$$

tj. radimo afinu transformaciju; uočiti, svaki neuron ima svoj γ i β .

- Možemo kraće pisati:

$$y = \gamma_{\downarrow} \odot \hat{h} + \beta_{\downarrow}$$

gdje \odot označava množenje odgovarajućih elemenata na istim položajima (*elementwise*)

Normalizacija nad grupom: u neuronskoj mreži

- Zašto posljednji korak: čista normalizacija umanjuje ekspresivnu moć mreže; ponovno skaliranje i pomak mogu rekonstruirati početnu distribuciju (ako je to potrebno)
 - Ako je početna distribucija imala određenu srednju vrijednost i odstupanje, normalizacija ih uklanja
 - Novo skaliranje i pomak mogu to poništiti (ako je potrebno), a omogućavaju puno lakše upravljanje distribucijom (imamo dva nova parametra koja izravno određuju srednju vrijednost i odstupanje)
- Parametre γ_I i β_I ćemo također učiti gradijentnim spustom (čitava transformacija je derivabilna!)

Normalizacija nad grupom: u neuronskoj mreži

- (Ioffe, Szegedy) su normalizirali izlaze prije nelinearnosti
- Stoga bismo dalje izlaze propustili kroz nelinearnost (npr. ReLU):

$$a = \text{ReLU}(y)$$

gdje je $\text{ReLU}(\cdot)$ primijenjen na svaki element matrice zasebno, pa generira matricu a iste dimenzionalnosti kao i matrica y :

$$a_{k,l} = \text{ReLU}(y_{k,l}) \quad k \in 1, \dots, N, \quad l \in 1, \dots, H$$

- opisanom transformacijom ReLU na ulaz dobiva normalizirane podatke (dok to postupak učenja ciljano ne promijeni promjenom γ_l i β_l ; početne vrijednosti su $\gamma_l = 1$ i $\beta_l = 0$)

Normalizacija nad grupom: u neuronskoj mreži

Ako se radi normalizacija, tada neuroni ne trebaju prag b - normalizacija će ga ionako ukloniti. Dovoljno je računati $h = X \cdot W$.

U Pythonu - unaprijedni dio:

```
mu = 1/N*np.sum(h, axis=0) # Size (H,)  
sigma2 = 1/N*np.sum((h-mu)**2, axis=0) # Size (H,)  
hath = (h-mu)*(sigma2+epsilon)**(-1./2.)  
y = gamma*hath+beta
```

Normalizacija nad grupom: u neuronskoj mreži

Izvod gradijenata za normalizaciju nad grupom (za znatiželjne):

- Pogledati izvorni rad: Sergey Ioffe, Christian Szegedy (2015): *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*
- Puno lakše štivo s detaljnijim (čitaj: "lakše se da shvatiti što se događa gdje") izvodom:
<http://cthorey.github.io./backpropagation/>

Normalizacija nad grupom: zaključno

U fazi učenja, algoritam radi s mini-grupama veličine N .

- Unaprijedni prolaz ne možemo organizirati kao N slijednih prolaza kroz mrežu (što bi bila jednostavna implementacija)
- Umjesto toga, računamo sloj po sloj: pogledamo odziv prvog sloja za sve podatke iz minigrupe; normaliziramo odzive; to je sada N novih ulaza za sljedeći sloj - izračunamo, normaliziramo; sljedeći sloj, sljedeći sloj, ... (gdje god radimo normalizaciju)
- Algoritam uči sve "uobičajene" parametre mreže (θ) te parametre afine transformacije (γ i β) za svaki neuron kod kojeg se obavlja normalizacija

Normalizacija nad grupom: zaključno

Da bismo došli do mreže spremne za eksplotaciju, jednom kad je učenje gotovo:

- Želimo mrežu koja može procesirati uzorak po uzorak
- Nemamo mini-grupe, pa ne možemo računati smislenu srednju vrijednost i standardno odstupanje
- Mogli bismo koristiti taj samo jedan uzorak no tada je srednja vrijednost baš on a varijanca je nula, što nije reprezentativno
- Umjesto toga, koristimo podatke možemo izračunati nad čitavim skupom za učenje

Normalizacija nad grupom: zaključno

Izračun konačne srednje vrijednosti i odstupanja. Evo postupka:

- Čitavu populaciju podijelimo u više mini-grupa veličine m
- Za svaku mini-grupu izračunamo srednju vrijednost i standardno odstupanje
- Kao konačnu srednju vrijednost i standardno odstupanje uzmemmo prosječne vrijednosti dobivene temeljem minigrupa:

$$E[h] \leftarrow E_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$Var[h] \leftarrow \frac{m}{m-1} E_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

Normalizacija nad grupom: zaključno

U fazi iskorištavanja mreže:

- Koristimo:

$$\begin{aligned}y &= \gamma \hat{h} + \beta \\&= \gamma \frac{h - E[h]}{\sqrt{Var[h] + \epsilon}} + \beta \\&= \frac{\gamma}{\sqrt{Var[h] + \epsilon}} \cdot h + \left(\beta - \frac{E[h]}{\sqrt{Var[h] + \epsilon}} \right)\end{aligned}$$

što je obična linearna transformacija.

Normalizacija nad grupom: zaključno

Prednosti normalizacije:

- Mreža lakše uči (u manjem broju iteracija)
- Mogu se koristiti veće stope učenja
- Povećana je robusnost na loše odabранe parametre (npr. prevelika skala parametara)
- Pokazuje efekte regularizacije pa je umanjena potreba za drugim tehnikama koje služe regularizaciji

Normalizacija nad grupom: zaključno

Još par napomena:

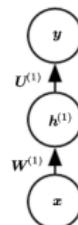
- u praksi, tijekom učenja se koriste pokretni prosjeci srednje vrijednosti i varijance (*moving average*)
- za najbolje rezultate posljednje epohe učenja mogu se provesti sa "smrznutom" populacijskom statistikom
- za najveću brzinu zaključivanja normalizacija nad grupom može se fuzionirati s prethodnom konvolucijom
- alternativno objašnjenje zašto normalizacija nad grupom ubrzava optimizaciju: <https://arxiv.org/abs/1805.11604>

Nadzirano predtreniranje

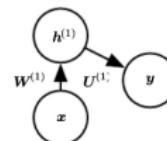
Ako je model jako dubok i problem jako težak:

- može se dogoditi da mrežu ne možemo kvalitetno učiti
- ideje:
 - trenirati jednostavniji model; zatim ga proširiti na složeniji
 - trenirati model da riješava jednostavniji zadatak; potom krenuti na teži zadatak
- opisane strategije zbirno nazivamo *predtreniranje*
- primjer iz knjige Deep Learning (slika 8.7) je na sljedećem slideu

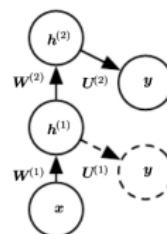
Nadzirano predtreniranje



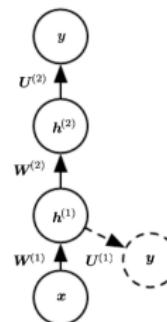
(a)



(b)



(c)



(d)

Nadzirano predtreniranje

- ① učimo mrežu s jednim skrivenom slojem; kad konvergira, zapamtimo izlaze skrivenih neurona
- ② odbacimo težine izlaznog sloja, dodamo novi skriveni sloj i učimo ga; ulazi su zapamćeni izlazi prethodnog sloja (model je plitak!)
- ③ ponavljamo prethodni korak po potrebi

Na kraju možemo napraviti i fino podešavanje tijekom kojeg algoritam učenja pokrećemo nad cjelovitom mrežom pa se istovremeno mogu podešavati težine kroz sve slojeve.

Pristup *Knowledge Distillation* (Hinton & Dean, 2014.)

- ① razmatramo klasifikacijski problem
- ② opažanje: učenje mreže i uporaba mreže su dva različita scenarija a do sada smo ih tretirali jednako
 - ① pri učenju: možemo si dopustiti puno izračuna i veći utrošak vremena
 - ② pri uporabi: naglasak je na brzini - želimo malo izračuna i malu potrošnju memorije

Pristup *Knowledge Distillation* (Hinton & Dean, 2014.)

Opažanja:

- ① velike modele (široke, duboke) možemo naučiti da rade praktički bez greške i da su vrlo sigurni u svoj izlaz
- ② primjer MNIST-a: ako je ulaz "2", odgovarajući izlaz će biti praktički 1, ostali praktički nula
 - "2" je slično "3" i "7", nije baš slično "4" i "5"
 - ova informacija praktički je izgubljena na izlazima, ali je puno vidljivija na logitima (*dark-knowledge!*)
- ③ slično: BMW je sličniji kamionu za smeće no mrkvi: ta će informacija efektivno biti izgubljena na izlazima
- ④ *dark-knowledge* može pomoći da kvalitetnije naučimo modele koji se teže uče (uži, duboki, manje parametara!)

Pristup *Knowledge Distillation* (Hinton & Dean, 2014.)

- ➊ ideja: najprije istreniramo veću mrežu (ili čak ansambl mreža)
 - sporo i puno parametara
 - danas se vjeruje da rezidualni model odgovara eksponencijalno velikom ansamblu plitkih modela
(<https://arxiv.org/abs/1605.06431>, primjer 1)
- ➋ takvu istreniranu mrežu nazivamo *mrežom učiteljem*
- ➌ sada želimo istrenirati novu dublju užu *mrežu učenika* (manje parametara, bolja generalizacija)

Pristup *Knowledge Distillation* (Hinton & Dean, 2014.)

- pri treniranju mreže učenika uz izlazne labele želimo ponuditi dodatnu informaciju na temelju mreže učitelja koja bi olakšala učenje (*dark-knowledge*)
- primijetiti: izlazni sloj mreže učitelja i mreže učenika su kompatibilni (jednaki broj neurona)
- jedna mogućnost: za svaki ulazni uzorak pogledati logite mreže učitelja pa minimizirati srednje kvadratno odstupanje razlike logita mreže učitelja i mreže učenika
 - više informacije, jači signal za korekciju, bolja generalizacija!

Pristup *Knowledge Distillation* (Hinton & Dean, 2014.)

- ➊ (Hinton & Dean, 2014.) koriste analogan pristup: želimo "omekšati" izlaze (da nisu gotovo 0 ili 1 već da se bolje vide odnosi među razredima čime ćemo dobiti pristup *dark-knowledge-u*)
 - ➌ uvodimo parametar $\tau > 1$
 - ➋ neka je $\mathbf{P}_T^\tau = \text{softmax}\left(\frac{\mathbf{a}_T}{\tau}\right)$ omekšani izlaz mreže učitelja a $\mathbf{P}_S^\tau = \text{softmax}\left(\frac{\mathbf{a}_S}{\tau}\right)$ omekšani izlaz mreže učenika za isti ulazni uzorak
- ➋ funkciju gubitka sada definiramo kao:

$$L_{KD}(\mathbf{W}_S) = H(\mathbf{y}_{\text{true}}, \mathbf{P}_S) + \lambda \cdot H(\mathbf{P}_T^\tau, \mathbf{P}_S^\tau)$$

gdje H označava unakrsnu entropiju

- ➌ λ je parametar koji regulira odnos između ove dvije komponente

Pristup *FitNets* (2015.)

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, Yoshua Bengio: *FitNets: Hints for Thin Deep Nets* (2015.; <https://arxiv.org/abs/1412.6550>)

- dodatna nadogradnja prethodne ideje
 - ① najprije istreniramo duboku širu mrežu
 - ② zatim ta mreža potom postaje *učitelj* dubokoj mreži
 - mreža učitelj davat će *hintove* mreži učeniku pri učenju
 - odaberemo jedan od slojeva (npr. srednji) s kojeg će se čitati *hintovi*

Pristup FitNets (2015.)

- ① sada krećemo u treniranje dubljeg ali tanjeg modela
 - ① u prvom koraku biramo jedan od slojeva duboke mreže učenika (npr. srednji)
 - ① treniramo duboki model do tog sloja da na temelju njega možemo rekonstruirati hintove koje daje mreža učitelj za iste ulaze
 - ② funkcija gubitka je:

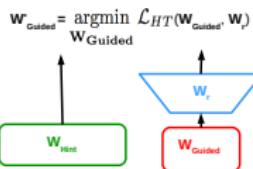
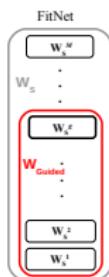
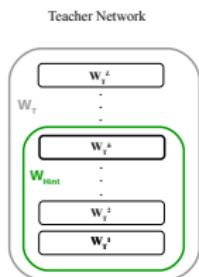
$$L_{HT}(\mathbf{W}_{guided}, \mathbf{W}_r) = \frac{1}{2} \|u_h(x; \mathbf{W}_{hint}) - r(v_g(x; \mathbf{W}_{guided}); \mathbf{W}_r)\|^2$$

Pristup *FitNets* (2015.)

- ➊ sada krećemo u treniranje dubljeg ali tanjeg modela
 - ➊ u drugom koraku odbacujemo ekstra težine potrebne za regresiju hintova i treniramo čitav model destilacijom
 - ➊ prvi korak u određenom smislu radi predtreniranje težina dijela duboke mreže
 - ➋ drugi korak radi treniranje čitavog modela ali ponovno uzimajući dodatne informacije od mreže učitelja

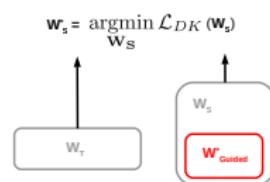
GoogLeNet

<https://arxiv.org/pdf/1412.6550.pdf>



(a) Teacher and Student Networks

(b) Hints Training



(c) Knowledge Distillation

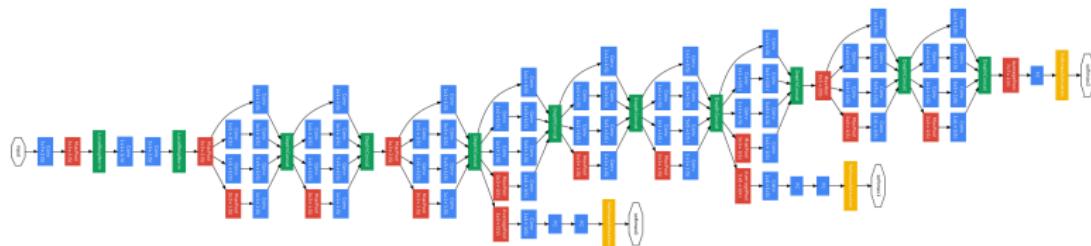
Izgradnja dubokih modela koje je lakše trenirati

Da bismo dobili kvalitetnije naučene duboke modele, umjesto modificiranja i razvoja novih algoritama učenja isplati se razmisliti o arhitekturama dubokih modela koje na napростo laganje učiti:

- ① primjer 1: dodavanje preskočnih veza smanjuje efektivnu udaljenost između nižih slojeva i izlaza: dobiva se "jači" gradijent za treniranje
- ② primjer 2: (GoogLeNet, 2014.) uz pojedine unutarnje slojeve dodati neurone na kojima se također zahtjeva ispravna reprodukcija izlaza
 - ① ovime jači signal pogreške dolazi i do ranijih slojeva
 - ② kad je mreža naučena, ovi se neuroni mogu odbaciti

GoogLeNet

<https://ai.google/research/pubs/pub43022>



Pitanja

?

Regularizacija dubokih modela

Josip Krapac i Siniša Šegvić

- Regularizacija
- Regularizacija normom vektora parametara modela
- Regularizacija generiranjem podataka i unošenjem šuma
- Regularizacija ranim zaustavljanjem
- Polunadzirano i više zadaćno učenje, dijeljenje parametara
- Regularizacija baggingom i dropout

Regularizacija: pregled

Glavni izazov u strojnom učenju: osigurati da model radi dobro ne samo na podacima za učenje nego i na novim podacima.

Tehnike regularizacije: smanjenje greške na skupu za testiranje, uz moguće povećanje greške na skupu za učenje.

Duboki modeli omogućavaju primjenu raznih tehnika regularizacije.

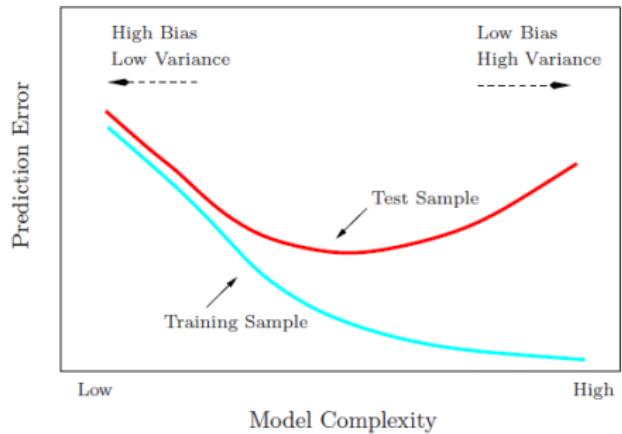
Jedan od najvažnijih otvorenih izazova: vrlo aktivno područje istraživanja.

Regularizacija: pregled

Tehnike regularizacije u strojnom učenju povećavaju pristranosti i smanjuju varijancu modela.

Krajnji cilj: otežavanje **prenaučenosti**.

U praksi: najbolju generalizacijsku pogrešku postiže **model vrlo velikog kapaciteta** na koji su primjenjene **odgovarajuće tehnike regularizacije**.



Pregled

- Regularizacija
- Regularizacija normom vektora parametara modela
- Regularizacija generiranjem podataka i unošenjem šuma
- Regularizacija ranim zaustavljanjem
- Polunadzirano i više zadaćno učenje, dijeljenje parametara
- Regularizacija baggingom i dropout

Regularizacija normom vektora parametara modela

Jedna od najstarijih metoda regularizacije: modificirati gubitka $J(\Theta; \mathbf{X}, \mathbf{y})$ dodavanjem norme vektora parametara $\Omega(\Theta)$:

$$\tilde{J}(\Theta; \mathbf{X}, \mathbf{y}) = J(\Theta; \mathbf{X}, \mathbf{y}) + \alpha \Omega(\Theta)$$

$\alpha \in [0, \infty]$ određuje relativni doprinos regularizatora Ω .

Minimizacija regularizirane funkcije gubitka \tilde{J} smanjuje i J i Ω .

Obično regulariziramo samo **težine**, tj. Ω ne djeluje na **pomak**.

Odabirom funkcije Ω **preferiramo** određene klase modela.

Intuicija: regularizator povlači vektor težina \mathbf{w} prema ishodištu.

Regularizacija L_2 normom vektora parametara modela

Promotrimo funkciju cilja regulariziranu s $\Omega(\Theta) = \frac{1}{2} \|w\|_2^2$

$$\tilde{J}(w; X, y) = J(w; X, y) + \frac{\alpha}{2} \|w\|_2^2$$

$$\nabla_w \tilde{J}(w; X, y) = \nabla_w J(w; X, y) + \alpha w$$

Korak gradijentnog spusta sada je ("weight decay"):

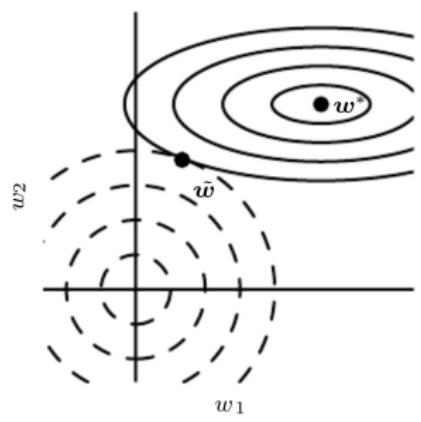
$$w^{t+1} = w^t - \epsilon \alpha w^t - \epsilon \nabla_w J(w; X, y)|_{w=w^t}$$

$$w^{t+1} = (1 - \epsilon \alpha) w^t - \epsilon \nabla_w J(w; X, y)|_{w=w^t}$$

Regularizacija L_2 normom vektora parametara modela

Razvijmo ne-regulariziranu funkciju gubitka J u Taylorov red oko minimuma $\mathbf{w}^* = \arg \min_{\mathbf{w}} J(\mathbf{w})$:

$$\widehat{J}(\mathbf{w}) = J(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^\top \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$



Regularizacija L_2 normom vektora parametara modela

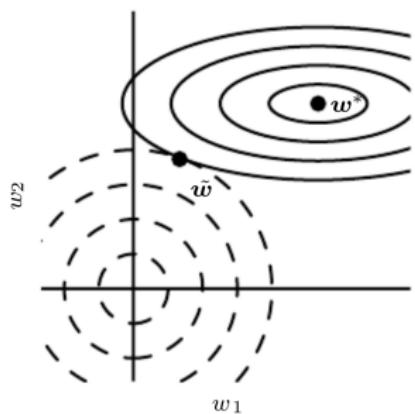
Dodajmo regularizacijski član i pogledajmo gdje se pomakne minimum

$$\nabla_{\tilde{w}} \left(\hat{J}(\tilde{w}) + \frac{\alpha}{2} \|\tilde{w}\|_2^2 \right) = 0$$

$$\nabla_{\tilde{w}} \hat{J}(\tilde{w}) + \alpha \tilde{w} = 0$$

$$H(\tilde{w} - w^*) + \alpha \tilde{w} = 0$$

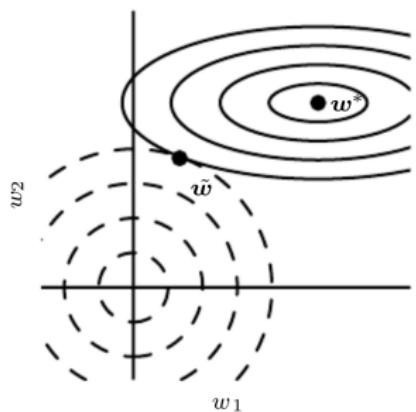
$$\tilde{w} = (H + \alpha I)^{-1} H w^*$$



Regularizacija L_2 normom vektora parametara modela

Pogledajmo što se dešava kada α raste. Uvid je lakši u prostoru razapetom svojstvenim vektorima (\mathbf{Q}) matrice $\mathbf{H} = \mathbf{Q}\Lambda\mathbf{Q}^\top$

$$\begin{aligned}\tilde{\mathbf{w}} &= (\mathbf{Q}\Lambda\mathbf{Q} + \alpha\mathbf{I})^{-1}\mathbf{Q}\Lambda\mathbf{Q}\mathbf{w}^* \\ &= [\mathbf{Q}(\Lambda + \alpha\mathbf{I})\mathbf{Q}^\top]^{-1}\mathbf{Q}\Lambda\mathbf{Q}^\top\mathbf{w}^* \\ &= \mathbf{Q}(\Lambda + \alpha\mathbf{I})^{-1}\Lambda\mathbf{Q}^\top\mathbf{w}^* \\ &= \mathbf{Q} \cdot \text{diag}\left(\frac{\lambda_i}{\lambda_i + \alpha}\right) \cdot \mathbf{Q}^\top\mathbf{w}^*\end{aligned}$$



Regularizacija L_2 normom vektora parametara modela

Pogledajmo doprinos projekcije \mathbf{w}^* na i -ti svojstveni vektor \mathbf{H} :

$$\tilde{\mathbf{w}}^{(i)} = \mathbf{q}_i \frac{\lambda_i}{\lambda_i + \alpha} \mathbf{q}_i^\top \mathbf{w}^*$$

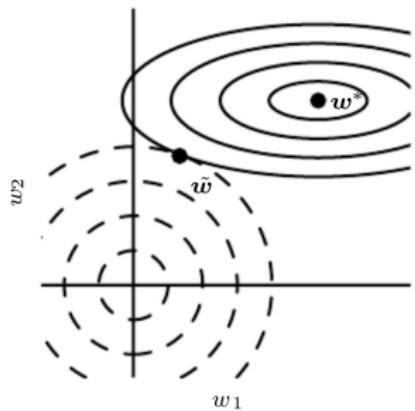
Vidimo da $\tilde{\mathbf{w}}^{(i)}$ postaje tim različitiji od \mathbf{w}_i^* što je λ_i manji, odnosno što je funkcija cilja manje strma u smjeru \mathbf{q}_i

Regularizacija L_2 normom vektora parametara modela

Za slučaj dijagonalne matrice \mathbf{H} ($\mathbf{Q} = \mathbf{I}$):

$$\tilde{\mathbf{w}} = \text{diag}\left(\frac{\lambda_i}{\lambda_i + \alpha}\right) \cdot \mathbf{w}^*$$

$$\tilde{w}_i = \frac{\lambda_i}{\lambda_i + \alpha} w_i^*$$



Vidimo da \tilde{w}_i postaje tim različitiji od w_i što je λ_i manji, odnosno što se \hat{J} manje mijenja po odgovarajućoj osi koordinatnog sustava

Regularizacija identificira parametre koji ne utječu na funkciju cilja i priteže ih prema ishodištu

Regularizacija L_2 normom vektora parametara modela

Pogledajmo utjecaj L_2 regularizacije na linearu regresiju sa srednjom kvadratnom pogreškom odstupanja kao funkcijom cilja.

Rješenje u ne-regulariziranom slučaju: $\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$

Rješenje u regulariziranom slučaju: $\mathbf{w} = (\mathbf{X}^\top \mathbf{X} + \alpha \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$

Matrica $\mathbf{X}^\top \mathbf{X}$ je proporcionalna kovarijacijskoj matrici $\frac{1}{n} \mathbf{X}^\top \mathbf{X}$.

Efekt regularizacije: prividno povećavanje varijance podataka. Kao da smo oko svakog podatka \mathbf{x} generirali nove podatke izvlačenjem iz normalne distribucije s srednjom vrijednosti koja odgovara podatku \mathbf{x} i varijancom $\alpha \mathbf{I}$.

Regularizacija L_1 normom vektora parametara modela

L_1 regularizator: $\Omega(\Theta) = \sum_i |w_i| = \|\mathbf{w}\|_1$.

Promotrimo gradijent regularizirane funkcije cilja:

$$\nabla_{\mathbf{w}} \tilde{J}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{X}, \mathbf{y}) + \alpha \text{sgn}(\mathbf{w})$$

Korak gradijentnog spusta je:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \epsilon \alpha \text{sgn}(\mathbf{w}^t) - \epsilon \nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{X}, \mathbf{y})|_{\mathbf{w}=\mathbf{w}^t}$$

Doprinos regularizacije ovisi samo o predznaku \mathbf{w} .

Regularizacija L_1 normom vektora parametara modela

Isto kao i za L_2 normu: promatramo rastav u Taylorov red u minimumu ne-regularizirane funkcije gubitka, dodamo regularizacijski član i promatramo gdje se pomakne minimum.

Dodatno, prepostavljamo da je matrica $\mathbf{H} = \boldsymbol{\Lambda}$ dijagonalna:

$$\tilde{\mathbf{w}} = \arg \min_{\mathbf{w}} \sum_i \left(\frac{1}{2} \lambda_i (w_i - w_i^*)^2 + \alpha |w_i| \right)$$

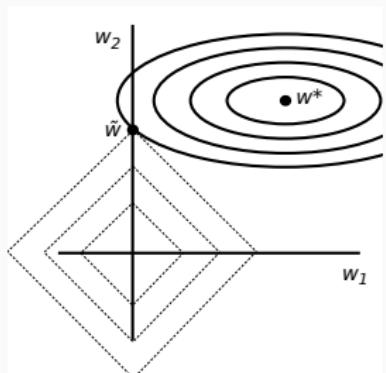
Za ovaj optimizacijski problem postoji analitičko rješenje:

$$\tilde{w}_i = \text{sgn}(w_i^*) \max \left\{ |w_i^*| - \frac{\alpha}{\lambda_i}, 0 \right\}$$

Regularizacija L_1 normom vektora parametara modela

Rješenje L1-regulariziranog kvadratnog problema:

$$\tilde{w}_i = \text{sgn}(w_i^*) \max \left\{ |w_i^*| - \frac{\alpha}{\lambda_i}, 0 \right\}$$

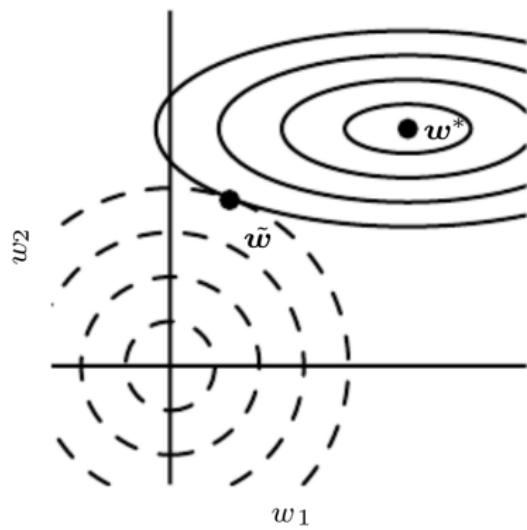


L_1 regularizacija vodi na **rijetke modele**: modeli za koje su neke vrijednosti parametara 0.

L_1 regularizacija istovremeno uči model i obavlja selekciju/eliminaciju značajki.

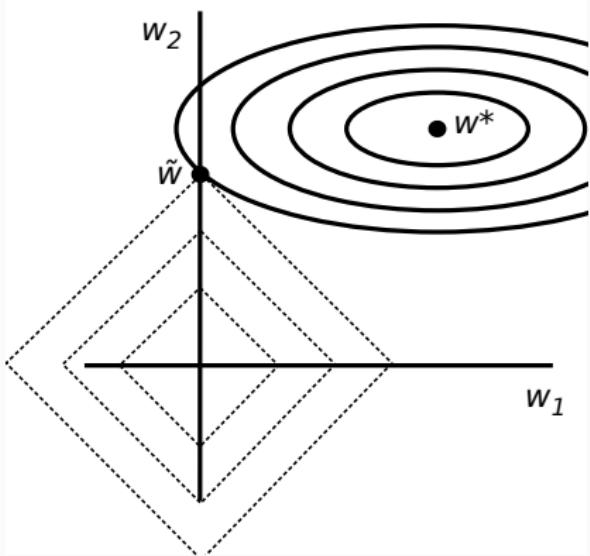
Usporedba L_1 i L_2 regularizacije

L_2



$$\tilde{w}_i = \frac{\lambda_i}{\lambda_i + \alpha} w_i^*$$

L_1



$$\tilde{w}_i = \operatorname{sgn}(w_i^*) \max \left\{ |w_i^*| - \frac{\alpha}{\lambda_i}, 0 \right\}$$

Obje regularizacije guraju w_i prema ishodištu, i to tim jače što je w_i manje važan za podatkovni gubitak (tj. tim jače što je λ_i - manji)

Norme vektora parametara kao ograničenja optimizacijskog postupka

Alternativni pogled: regularizacija osigurava da parametri budu unutar kugle $\Omega(\Theta) < k$.

Radijus k ovisan je o parametru α : veći α znači manji k i obratno.

Oblik kugle ovisi o korištenoj normi (npr. L_1 , L_2)

Regularizaciju možemo izraziti i direktno preko k . Modifikacija optimizacijskog postupka:

- napravimo korak gradijentnog spusta i
- projiciramo ažurirani Θ na najbližu točku koja zadovoljava ograničenje $\Omega(\Theta) < k$.

Norme vektora parametara kao ograničenja optimizacijskog postupka

Ova formulacija ne mijenja funkciju gubitka.

- Promjena funkcije gubitka može uzrokovati da optimizacijski postupak zapne u dijelu prostora koji odgovara malim vrijednostima parametara Θ .
- U ovakvoj formulaciji to se izbjegava budući se projekcija obavlja tek kada vektor parametara naraste dovoljno da izđe iz kugle.

Ova formulacija omogućava stabilniji optimizacijski postupak:

- kod korištenja velikih koraka učenja moguće je da postupak počne divergirati (vektor parametara počinje rasti) zbog pozitivne povratne veze.
- U ovakvoj formulaciji nekontroliran rast nije moguć.

Pregled

- Regularizacija
- Regularizacija norme vektora parametara modela
- Regularizacija generiranjem podataka i unošenjem šuma
- Regularizacija ranim zaustavljanjem
- Polunadzirano i više zadaćno učenje, dijeljenje parametara
- Regularizacija baggingom i dropout

Regularizacija generiranjem podataka

Podatci za učenje mogu se promatrati kao odličan regularizator.

Neki problemi dozvoljavaju jednostavno generiranje podataka za učenje modifikacijom postojećih podataka za učenje.

Ova tehnika se pokazala jako uspješnom kod raspoznavanja slika (translatiranje, skaliranje, rotacija) i za raspoznavanje govora.

Pretpostavka: modifikacija ne utječe na ishod predikcije.

Danas je modificiranje podataka popularno za učenje reprezentacija sa zamjenskim gubitkom na neoznačenim podatcima [kolesnikov19cvpr]

- npr. model se uči da pogodi parametar modifikacije (rotaciju)...
- ...ili da razlikuje modificirani podatak od drugih podataka itd.

Regularizacija unošenjem šuma

Unošenje šuma u model također ima regularizacijski efekt.

Šum možemo primijeniti na:

- ulaz modela (podatke iz skupa za učenje),
- reprezentacije u skrivenim slojevima,
- parametre modela,
- oznake u skupu za učenje (eng. label smoothing).
 - Ako pretpostavimo da je vjerojatnost točnog označavanja $1 - \epsilon$ onda one-hot oznak transformiramo:

$$[0, 0 \dots 1, 0, \dots 0] \rightarrow \left[\frac{\epsilon}{k}, \frac{\epsilon}{k}, \dots, 1 - \frac{k-1}{k}\epsilon, \frac{\epsilon}{k}, \dots, \frac{\epsilon}{k} \right]$$

Pregled

- Regularizacija
- Regularizacija norme vektora parametara modela
- Regularizacija generiranjem podataka i unošenjem šuma
- **Regularizacija ranim zaustavljanjem**
- Polunadzirano i višezadaćno učenje, dijeljenje parametara
- Regularizacija baggingom i dropout

Regularizacija ranim zaustavljanjem

Broj epoha je hiper-parametar koji možemo efikasno, inkrementalno validirati u jednoj epizodi učenja

Jedino što moramo promjeniti u odnosu na osnovnu proceduru:

- povremeno evaluirati model na skupu za validaciju
- čuvati parametre koji postižu najbolju validacijsku točnost
- usporavanje učenja uslijed evaluiranja možemo ublažiti:
 - evaluacijom na drugom procesoru
 - smanjenjem skupa za evaluaciju
 - rjeđom validacijom

Nedostatak: moramo imati skup za validaciju, što znači da prije konačnog testiranja trebamo ponoviti učenje na trainval skupu

Regularizacija ranim zaustavljanjem

Algorithm 7.1 The early stopping meta-algorithm for determining the best amount of time to train. This meta-algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

Let n be the number of steps between evaluations.

Let p be the “patience,” the number of times to observe worsening validation set error before giving up.

Let θ_o be the initial parameters.

$\theta \leftarrow \theta_o$

$i \leftarrow 0$

$j \leftarrow 0$

$v \leftarrow \infty$

$\theta^* \leftarrow \theta$

$i^* \leftarrow i$

while $j < p$ **do**

 Update θ by running the training algorithm for n steps.

$i \leftarrow i + n$

$v' \leftarrow \text{ValidationSetError}(\theta)$

if $v' < v$ **then**

$j \leftarrow 0$

$\theta^* \leftarrow \theta$

$i^* \leftarrow i$

$v \leftarrow v'$

else

$j \leftarrow j + 1$

end if

end while

Best parameters are θ^* , best number of training steps is i^*

Zaustavljanje učenja konačnog modela prema broju iteracija

Algorithm 7.2 A meta-algorithm for using early stopping to determine how long to train, then retraining on all the data.

Let $\mathbf{X}^{(\text{train})}$ and $\mathbf{y}^{(\text{train})}$ be the training set.

Split $\mathbf{X}^{(\text{train})}$ and $\mathbf{y}^{(\text{train})}$ into $(\mathbf{X}^{(\text{subtrain})}, \mathbf{X}^{(\text{valid})})$ and $(\mathbf{y}^{(\text{subtrain})}, \mathbf{y}^{(\text{valid})})$ respectively.

Run early stopping (Algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\mathbf{X}^{(\text{subtrain})}$ and $\mathbf{y}^{(\text{subtrain})}$ for training data and $\mathbf{X}^{(\text{valid})}$ and $\mathbf{y}^{(\text{valid})}$ for validation data. This returns i^* , the optimal number of steps.

Set $\boldsymbol{\theta}$ to random values again.

Train on $\mathbf{X}^{(\text{train})}$ and $\mathbf{y}^{(\text{train})}$ for i^* steps.

Zaustavljanje učenja konačnog modela prema validacijskoj grešci

Algorithm 7.3 Meta-algorithm using early stopping to determine at what objective value we start to overfit, then continue training until that value is reached.

Let $\mathbf{X}^{(\text{train})}$ and $\mathbf{y}^{(\text{train})}$ be the training set.

Split $\mathbf{X}^{(\text{train})}$ and $\mathbf{y}^{(\text{train})}$ into $(\mathbf{X}^{(\text{subtrain})}, \mathbf{X}^{(\text{valid})})$ and $(\mathbf{y}^{(\text{subtrain})}, \mathbf{y}^{(\text{valid})})$ respectively.

Run early stopping (Algorithm 7.1) starting from random $\boldsymbol{\theta}$ using $\mathbf{X}^{(\text{subtrain})}$ and $\mathbf{y}^{(\text{subtrain})}$ for training data and $\mathbf{X}^{(\text{valid})}$ and $\mathbf{y}^{(\text{valid})}$ for validation data. This updates $\boldsymbol{\theta}$.

$$\epsilon \leftarrow J(\boldsymbol{\theta}, \mathbf{X}^{(\text{subtrain})}, \mathbf{y}^{(\text{subtrain})})$$

while $J(\boldsymbol{\theta}, \mathbf{X}^{(\text{valid})}, \mathbf{y}^{(\text{valid})}) > \epsilon$ **do**

 Train on $\mathbf{X}^{(\text{train})}$ and $\mathbf{y}^{(\text{train})}$ for n steps.

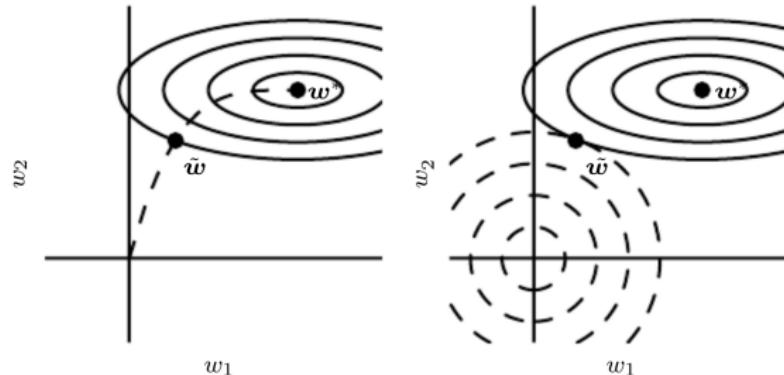
end while

Teoretska analiza ranog zaustavljanja

Pretpostavimo da prođemo kroz τ iteracija postupka gradijentnog spusta, s korakom ϵ , i da je gradijent u svakom koraku ogradien.

Tada je prostor parametara koji možemo dosegnuti iz početne vrijednosti parametra Θ_0 omeđen.

Veličina kugle ovisi o $\tau\epsilon$: što je ta vrijednost veća, to je kugla veća.



Pregled

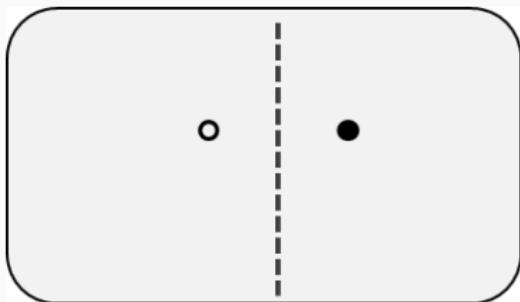
- Regularizacija
- Regularizacija norme vektora parametara modela
- Regularizacija generiranjem podataka i unošenjem šuma
- Regularizacija ranim zaustavljanjem
- Polunadzirano i više zadaćno učenje, dijeljenje parametara
- Regularizacija baggingom i dropout

polunadzirano učenje kao regularizacija

Polunadzirano učenje (eng. semi-supervised learning) je učenje $P(\mathbf{y}|\mathbf{x})$ koristeći i označene podatke (uzorke iz $P(\mathbf{x}, \mathbf{y})$) i neoznačene podatke (uzorke iz $P(\mathbf{x})$).

Možemo konstruirati model koji minimizira i nadzirani gubitak ($-\log P(\mathbf{y}|\mathbf{x})$) i nенадзирани губитак (npr. $-\log P(\mathbf{x})$).

Dodatni signal za učenje može dovesti do boljeg semantičkog sadržaja dijeljenih značajki i poboljšati generalizaciju.

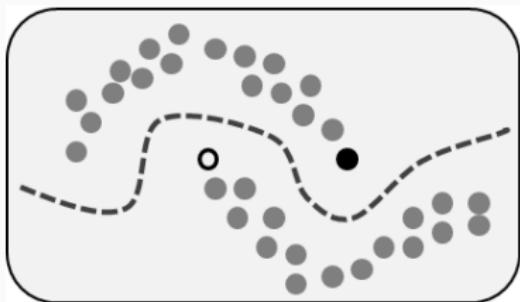


polunadzirano učenje kao regularizacija

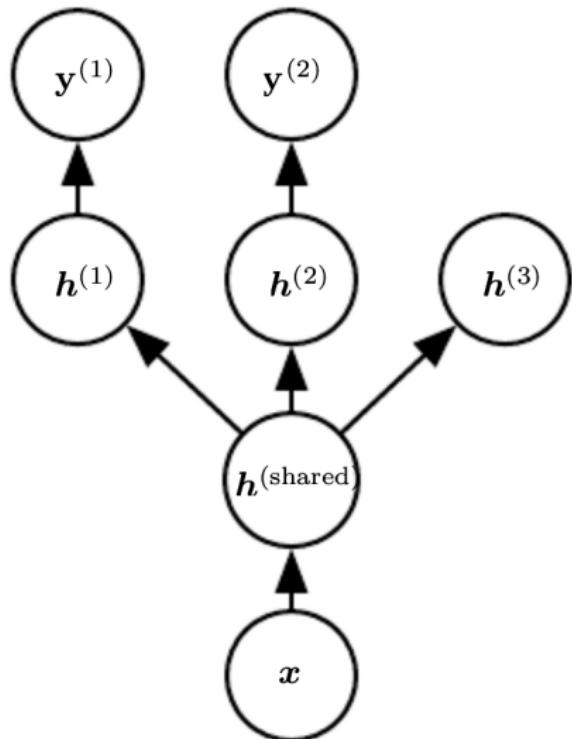
Polunadzirano učenje (eng. semi-supervised learning) je učenje $P(\mathbf{y}|\mathbf{x})$ koristeći i označene podatke (uzorke iz $P(\mathbf{x}, \mathbf{y})$) i neoznačene podatke (uzorke iz $P(\mathbf{x})$).

Možemo konstruirati model koji minimizira i nadzirani gubitak ($-\log P(\mathbf{y}|\mathbf{x})$) i nенадзирани gubitak (npr. $-\log P(\mathbf{x})$).

Dodatni signal za učenje može dovesti do boljeg semantičkog sadržaja dijeljenih značajki i poboljšati generalizaciju.



Višezadačno učenje kao regularizacija



Npr:

$y^{(1)} \dots p(c|x)$

$y^{(2)} \dots p(x)$

$h^{(3)}$... latentna reprezentacija
(za zamjenski gubitak)

Regularizacija vezanjem i dijeljenjem parametara

Vezanje parametara: ne znamo kakve parametre trebamo, ali znamo da postoji veza između dva problema

$$\Omega(\mathbf{w}^{(A)}, \mathbf{w}^{(B)}) = \|\mathbf{w}^{(A)} - \mathbf{w}^{(B)}\|_2^2$$

Dijeljenje parametara: u slučajevima kad želimo da skupovi parametara budu isti. Prednost u odnosu na vezanje parametara: trebamo čuvati samo jedan skup parametara.

Rijetke reprezentacije kao regularizator

- L_1 regularizacija težina \rightarrow rijetki modeli:

$$\begin{bmatrix} 18 \\ 5 \\ 15 \\ -9 \\ -3 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 0 & -2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 & 0 \\ 0 & 5 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & -4 \\ 1 & 0 & 0 & 0 & -5 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ -2 \\ -5 \\ 1 \\ 4 \end{bmatrix}$$
$$\mathbf{y} \in \mathbb{R}^m \qquad \qquad \mathbf{A} \in \mathbb{R}^{m \times n} \qquad \qquad \mathbf{x} \in \mathbb{R}^n$$

$$\tilde{J}(\boldsymbol{\Theta}; \mathbf{X}, \mathbf{y}) = J(\boldsymbol{\Theta}; \mathbf{X}, \mathbf{y}) + \alpha \|\boldsymbol{\Theta}\|_1$$

Rijetke reprezentacije kao regularizator

- L_1 regularizacija aktivacija \rightarrow rijetke reprezentacije:

$$\begin{bmatrix} -14 \\ 1 \\ 19 \\ 2 \\ 23 \end{bmatrix} = \begin{bmatrix} 3 & -1 & 2 & -5 & 4 & 1 \\ 4 & 2 & -3 & -1 & 1 & 3 \\ -1 & 5 & 4 & 2 & -3 & -2 \\ 3 & 1 & 2 & -3 & 0 & -3 \\ -5 & 4 & -2 & 2 & -5 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ -3 \\ 0 \end{bmatrix}$$
$$\mathbf{y} \in \mathbb{R}^m \qquad \qquad \mathbf{B} \in \mathbb{R}^{m \times n} \qquad \qquad \mathbf{h} \in \mathbb{R}^n$$

$$\tilde{J}(\Theta; X, y) = J(\Theta; X, y) + \alpha \|\mathbf{h}\|_1$$

- Regularizacija
- Regularizacija norme vektora parametara modela
- Regularizacija generiranjem podataka i unošenjem šuma
- Regularizacija ranim zaustavljanjem
- Polunadzirano i više zadaćno učenje, dijeljenje parametara
- Regularizacija baggingom i dropout

Bagging

Bagging (ili **bootstrap aggregating**) [breiman96ml]: jednostavna ali efikasna metoda za poboljšanje generalizacijske točnosti

Neka je zadan skup za učenje $\mathbf{X} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$.

Iz \mathbf{X} uzorkujemo M podskupova $\mathbf{X}_m = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N'}$

- uzorci se mogu ponavljati (uzorkovanje sa zamjenom)

Na svakom uzorku učimo jednu instancu modela: $f(\mathbf{x}, \Theta_m)$, te iz dobivenih instanci formiramo **ansambl**

Zaključivanje provodimo usrednjavanjem predikcija članova ansambla:

$$\hat{\mathbf{y}} = \frac{1}{M} \sum_{m=1}^M f(\mathbf{x}, \Theta_m)$$

Regularizacijski efekt bagginga

Prepostavimo da smo naučili M skalarnih regresijskih modela:

- opišimo pogrešku i-tog modela varijablom ϵ_i
- pogreška ansambla regresora tada je: $E_a = \frac{1}{M} \sum_i \epsilon_i$

Prepostavimo zajedničku distribuciju grešaka modela: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$

- dijagonalni elementi kovarijacijske matrice: $\mathbb{E}[\epsilon_i^2] = \sigma_{i,i} = v$,
- ne-dijagonalni elementi su: $\mathbb{E}[\epsilon_i \epsilon_j] = \sigma_{i,j} = c$.

Regularizacijski efekt bagginga (2)

Očekivanje kvadrata pogreške ansambla E_a^2 tada je:

$$\begin{aligned}\mathbb{E} \left[\left(\frac{1}{M} \sum_i \epsilon_i \right)^2 \right] &= \frac{1}{M^2} \mathbb{E} \left[\sum_i \left(\epsilon_i^2 + \sum_{i \neq j} \epsilon_i \epsilon_j \right) \right] \\ &= \frac{1}{M} v + \frac{M-1}{M} c\end{aligned}$$

Dva ekstremna slučaja:

1. Greške pojedinih regresora su potpuno korelirane $v = c$: ansambl tada grijesi kao i pojedini modeli.
2. Greške pojedinih regresora su potpuno nekorelirane $c = 0$: greška ansambla tada opada linearno s brojem regresora M .

Bagging: ilustracija

Original dataset



First resampled dataset



First ensemble member



Second resampled dataset



Second ensemble member



Usrednjavanje modela

Bitno je da modeli daju drugačije predikcije, učenje iz raznih podskupova je samo jedan način da se to omogući.

U dubokim modelima različitost možemo postići:

- različitim hiper-parametrima modela i postupka učenja
- različitim slučajnim inicijalizacijama
- različitim rasporedom podataka u mini-grupama

Ti efekti su dovoljni da naučeni modeli rade različite greške.

Usrednjavanje modela u praksi radi jako dobro: to je način kako se dobivaju najbolji rezultati (i pobjeđuje na natjecanjima).

Tehnika regularizacije dubokih modela:

- sroдna ansambliranju, računski manje zahtjevna

Intuicija: efikasni ansambl velikog broja različitih modela.

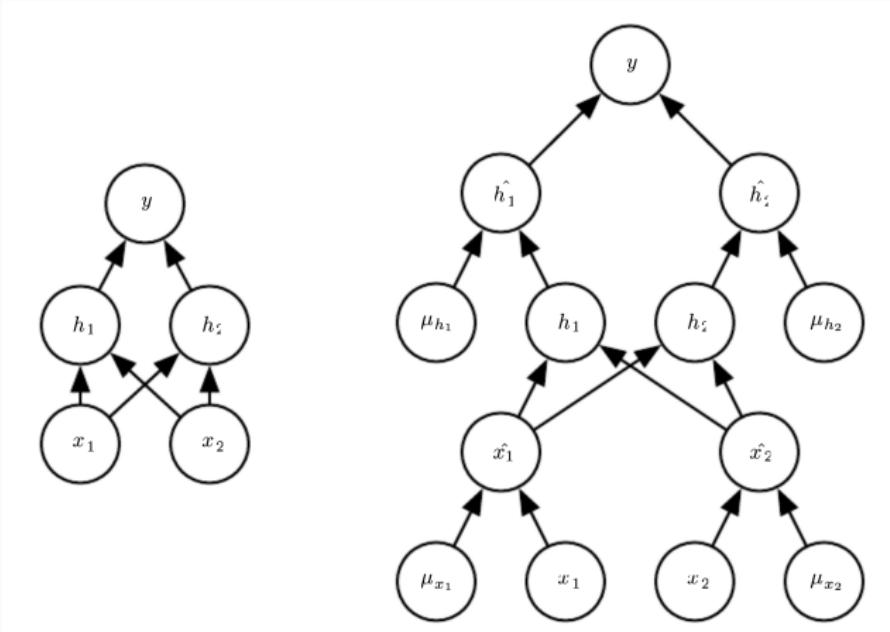
Kako dobiti veliki broj različitih modela?

- iz početnog modela formirati podmodele gašenjem/uklanjanjem aktivacija: latentnih i ulaznih značajki
- u praksi, odabir provodimo množenjem vektora aktivacija sloja \mathbf{x} s binarnom slučajnom maskom $\boldsymbol{\mu} \sim B(1, p)^N$:

$$\mathbf{x}' = \mathbf{x} \odot \boldsymbol{\mu}$$

- tipično, za ulazne značajke $p = 0.8$, a za skrivene značajke $p = 0.5$

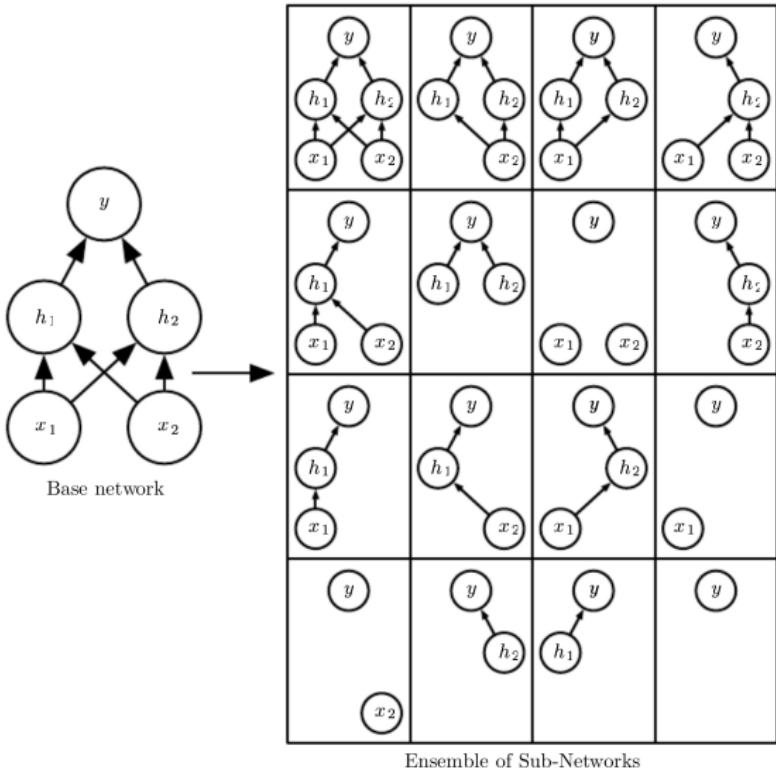
Dropout kao ansambl 2^N modela



$$\boldsymbol{\mu} = [\mu_{x_1}, \mu_{x_2}, \mu_{h_1}, \mu_{h_2}]$$

$$p(\boldsymbol{\mu}) = p(\mu_{x_1})p(\mu_{x_2})p(\mu_{h_1})p(\mu_{h_2})$$

Dropout kao ansambl 2^N modela (2)

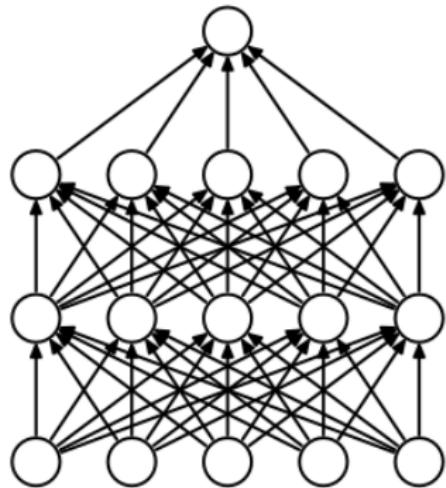


U praksi, $N > 100$: većina podmodela ima put od ulaza do izlaza

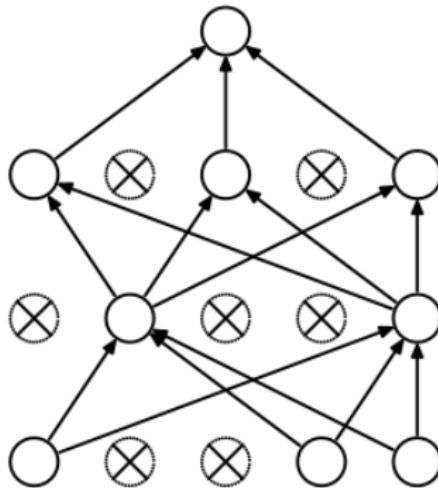
Dropout u fazi učenja

Dropout pristup: zasebno učiti dijelove istog modela

- formalno, minimiziramo: $\mathbb{E}_{\mu} J(\Theta, \mu) = \sum_{\mu} J(\Theta, \mu)p(\mu)$
- eksponencijalnu složenost možemo primiriti uzorkovanjem μ u svakoj iteraciji učenja (nepristrana procjena gradijenta)



(a) Standard Neural Net



(b) After applying dropout.

Dropout u fazi ispitivanja

U slučaju bagginga, predikcija ansambla za podatak \mathbf{x} je:

$$\frac{1}{M} \sum_m p(\mathbf{y}|\mathbf{x}, \Theta_m)$$

Kod dropoute ansambliramo podmodele:

$$p(\mathbf{y}|\mathbf{x}, \Theta) = \sum_{\boldsymbol{\mu}} p(\boldsymbol{\mu}) p(\mathbf{y}|\mathbf{x}, \boldsymbol{\mu}) \quad (1)$$

- prikazani izraz računa **očekivanje** preko distribucije maski $p(\boldsymbol{\mu})$ koje definiraju podmodele
- broj članova u izrazu je 2^N gdje je N broj aktivacija.

Dropout u fazi ispitivanja: uzorkovanje pod-modela

U praksi možemo raditi s M podmodela ($M \ll 2^N$)

Uzorkujemo $\mu_m \sim p(\mu)$ i određujemo izlaze podmodela $m \in 1..M$:

$$\hat{y}_m = p(y|x, \mu_m)$$

Konačna predikcija je srednja vrijednost izlaza podmodela:

$$\hat{y} = \frac{1}{M} \sum_{m=1}^M \hat{y}_m$$

- problem: za ovo trebamo M unaprijednih prolaza...
- prednost: možemo izračunati varijancu predikcije...

Dropout u fazi ispitivanja: skaliranje težina

Ako aritmetičku sredinu zamjenimo geometrijskom:

$$\tilde{p}(\mathbf{y}|\mathbf{x}, \Theta) = \left(\prod_{\mu} p(\mathbf{y}|\mathbf{x}, \Theta_{\mu}) \right)^{2^{-n}} \dots$$

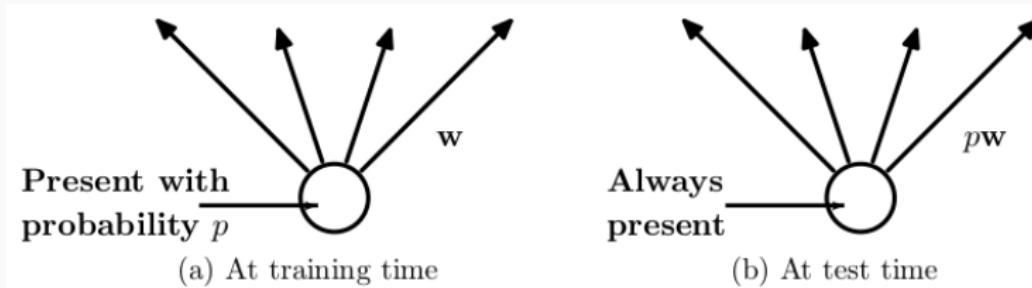
... te re-normaliziramo predikcije na distribuciju preko izlaza:

$$\hat{p}(\mathbf{y}|\mathbf{x}, \Theta) = \frac{\tilde{p}(\mathbf{y}|\mathbf{x}, \Theta)}{\sum_{c=1}^C \tilde{p}(y_c = 1|\mathbf{x}, \Theta)}$$

Tada dropout u fazi ispitivanja možemo aproksimirati **skaliranjem težina** modela.

Dropout u fazi ispitivanja

Izlazne težine aktivacije (= stupci matrice \mathbf{W}) množimo s vjerojatnošću njene prisutnosti tijekom učenja: $w'_{h_i} = p(\mu_{h_i})w_{h_i}$



[srivastava15jmlr]

- intuicija: osigurati da očekivani ulaz čvora u fazi ispitivanja bude jednak očekivanom ulazu čvora u fazi učenja
- prednost: jedan umesto M unaprijednih prolaza.

Druga mogućnost: skalirati aktivacije prilikom učenja: $h'_i = h_i / p(\mu_{h_i})$

Dropout: prednosti

Moguća primjena na razne tipove modela (konvolucijski, povratni) bez modifikacije funkcije cilja.

Jednostavna kombinacija s ostalim metodama regularizacije.

Računski znatno manje zahtjevan od bagginga

U usporedbi s ne-regulariziranim modelom:

- složenije učenje (više iteracija, npr. $2\times$)
- jednako brzo zaključivanje

Dropout: kada?

Dropout efektivno smanjuje kapacitet modela; najbolji rezultati postižu se povećanjem modela i duljim treniranjem. Za jako velike skupove podataka nije praktičan.

Kod konvolucijskih modela mogu se izbacivati:

- pojedinačne aktivacije
- cijele mape značajki (drop_channel)

Normalizacija nad grupom (eng. batch norm) također ima i regularizacijski efekt: dodavanje aditivnog i multiplikativnog šuma koji ovisi o podacima u mini-grupi. Popularna alternativa dropoutu.

Primjer: skaliranje težina u višerazrednoj logističkoj regresiji

Nema skrivenih slojeva \Rightarrow dropout samo na ulaznim značajkama x .

Za višerazrednu logističku regresiju: geometrijska sredina predikcija eksponencijalnog broja pod-modela **ekvivalentna** skaliranju težina

Višerazredna logistička regresija:

$$p(\mathbf{y} = y | \mathbf{x}, \Theta) = \text{softmax} (\mathbf{W} \cdot \mathbf{x} + \mathbf{b})_y$$

Razred pod-modela definiran binarnim vektorom μ :

$$p(\mathbf{y} = y | \mathbf{x}, \Theta_\mu) = \text{softmax} (\mathbf{W} \cdot (\boldsymbol{\mu} \odot \mathbf{x}) + \mathbf{b})_y$$

Primjer: skaliranje težina u višerazrednoj logističkoj regresiji

Dokažimo da predikciju eksponencijalno velikog dropout-ansambla uz $p=0.5$ uistinu možemo dobiti skaliranjem težina!

$$\begin{aligned}\tilde{p}(\mathbf{y} = \mathbf{y}|\mathbf{x}, \Theta) &= \left(\prod_{\boldsymbol{\mu} \in \{0,1\}^n} p(\mathbf{y} = \mathbf{y}|\mathbf{x}, \Theta_{\boldsymbol{\mu}}) \right)^{2^{-n}} \\ &= \frac{\left(\prod_{\boldsymbol{\mu} \in \{0,1\}^n} \exp(\mathbf{W}_{y,:}(\boldsymbol{\mu} \odot \mathbf{x}) + b_y) \right)^{2^{-n}}}{\left(\prod_{\boldsymbol{\mu} \in \{0,1\}^n} \sum_{y'} \exp(\mathbf{W}_{y',:}(\boldsymbol{\mu} \odot \mathbf{x}) + b_{y'}) \right)^{2^{-n}}} \\ &= C \cdot \left(\prod_{\boldsymbol{\mu} \in \{0,1\}^n} \exp(\mathbf{W}_{y,:}(\boldsymbol{\mu} \odot \mathbf{x}) + b_y) \right)^{2^{-n}} \\ &= C \cdot \exp \left(2^{-n} \sum_{\boldsymbol{\mu} \in \{0,1\}^n} \mathbf{W}_{y,:}(\boldsymbol{\mu} \odot \mathbf{x}) + b_y \right)\end{aligned}$$

Primjer: skaliranje težina u višerazrednoj logističkoj regresiji

$$\begin{aligned}\tilde{p}(\mathbf{y} = y | \mathbf{x}, \Theta) &= C \cdot \exp \left(2^{-n} \sum_{\boldsymbol{\mu} \in \{0,1\}^n} \mathbf{W}_{y,:} (\boldsymbol{\mu} \odot \mathbf{x}) + b_y \right) \\ &= C \cdot \exp \left(\mathbf{W}_{y,:} \left(2^{-n} \sum_{\boldsymbol{\mu} \in \{0,1\}^n} \boldsymbol{\mu} \right) \odot \mathbf{x} + 2^{-n} \sum_{\boldsymbol{\mu} \in \{0,1\}^n} b_y \right) \\ &= C \cdot \exp (\mathbf{W}_{y,:} (\mathbf{0.5} \odot \mathbf{x}) + b_y) \\ &= C \cdot \exp ((\mathbf{W}_{y,:} \odot \mathbf{0.5}) \cdot \mathbf{x} + b_y)\end{aligned}$$

Lako je pokazati da bi se drugi izbori $p(\boldsymbol{\mu})$ postigli drugačijim iteriranjem kroz podmodele. Tada bismo imali:

$$\tilde{p}(\mathbf{y} = y | \mathbf{x}, \Theta) = C \cdot \exp \left(\left(\mathbf{W}^\top \odot p(\boldsymbol{\mu}) \right)_{y,:}^\top \mathbf{x} + b_y \right)$$

Zadatak

Pretpostavke:

- 3-dimenzionalni ulaz: $\mathbf{x} \in \mathbb{R}^3$.
- dvije klase na izlazu $\mathbf{y} \in \mathbb{R}^2$.
- model: višerazredna logistička regresija

$$\mathbf{y} = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b})$$

- ulaz modela smo regularizirali dropoutom (bez mijenjanja aktivacija) uz $p(\mu_{x_1}) = 0.2, p(\mu_{x_2}) = 0.5, p(\mu_{x_3}) = 0.7$.

Zadatak: izračunajmo ulaz u softmax (eng. logits) za podatak $\mathbf{x} = [1, 1, 1]$, ako su parametri modela nakon učenja bili:

$$\mathbf{W} = \begin{bmatrix} -0.1 & 0.4 & -0.6 \\ 0.2 & -0.3 & 0.5 \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} 0.2 \\ -0.2 \end{bmatrix}$$

Jednostavni povratni modeli

Martin Tutek, Petra Bevandić, Josip Šarić, Siniša Šegvić
2023.

Uvod

U prethodnim predavanjima...

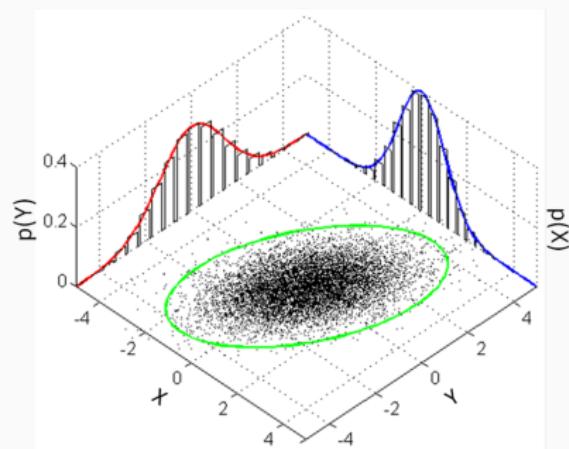
...bavili smo se modelima za podatke s:

- fiksnom dimenzionalnošću

- eksplicitnom međuvisnošću



primjeri slika iz CIFAR-10

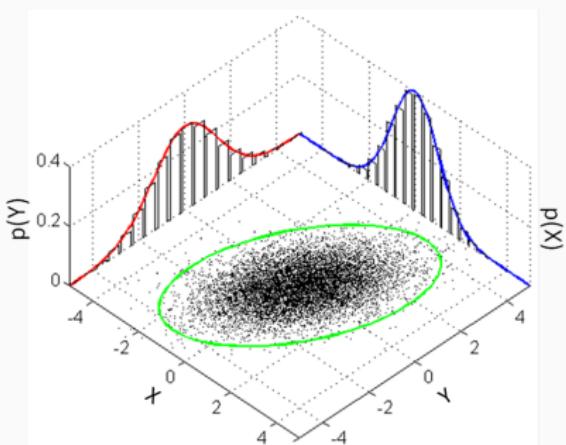


multivarijatna normalna distribucija

U prethodnim predavanjima...

...bavili smo se modelima za podatke s:

- fiksnom dimenzionalnošću
- eksplicitnom međuvisnošću



multivarijatna normalna distribucija



primjeri slika iz CIFAR-10

U prethodnim predavanjima...

Fiksna ulazna dimenzionalnost:

- svi uzorci multivariatne normalne distribucije (vidi Lab #1) imaju istu dimenzionalnost: $x_i \in \mathbb{R}^d, \forall i$
- svaka slika iz CIFAR-10 ima istu dimenzionalnost 32x32x3 (WxHxC)
- **kako osigurati:** izrezivanjem, nadopunjavanjem, sažimanjem, uvećavanjem (interpolacijom)

Međuovisnost:

- općenita normalna razdioba $\mathcal{N}(\mu, \Sigma)$:
 - komponente nisu dekorelirane: $P(x, y) \neq P(x)P(y)$
 - eksplicitnu međuovisnost opisuje parametar Σ
- slike: bliski pikseli implicitno su jače korelirani od dalekih
- **kako osigurati:** primjenom konvolucijskih slojeva i pozicijskog kodiranja kod transformera

Obrada prirodnog jezika

Prevedite ovu rečenicu na engleski jezik:

Duboko učenje je super. → *Deep learning is great.*

Rješenje problema zahtijeva sljedeće korake:

- segmentirati rečenicu (slijed slova, riječi ili slogova) na smislene komponente
- “razumjeti” da “duboko učenje” nije učenje na velikim dubinama (npr. na dnu Atlantika) nego grana računarstva
- shvatiti značenje izvorne rečenice bez da nam je netko rekao o kojem se jeziku radi
- generirati engleski prijevod s istim značenjem.

Tekstni podatci su čudni:

- **Nemaju** fiksnu dimenzionalnost (duljina rečenice može varirati)
- **Nemaju** eksplisitne obrasce međuovisnosti.
 - daleke ovisnosti: **tko** je prema povijesnim izvorima otkrio Ameriku?

Obrada prirodnog jezika

Prevedite ovu rečenicu na vijetnamski jezik:

最寄りのインターネットカフェはどこですか

→ *Chợ internet ở đâu*

→ *Where is the nearest internet shop*

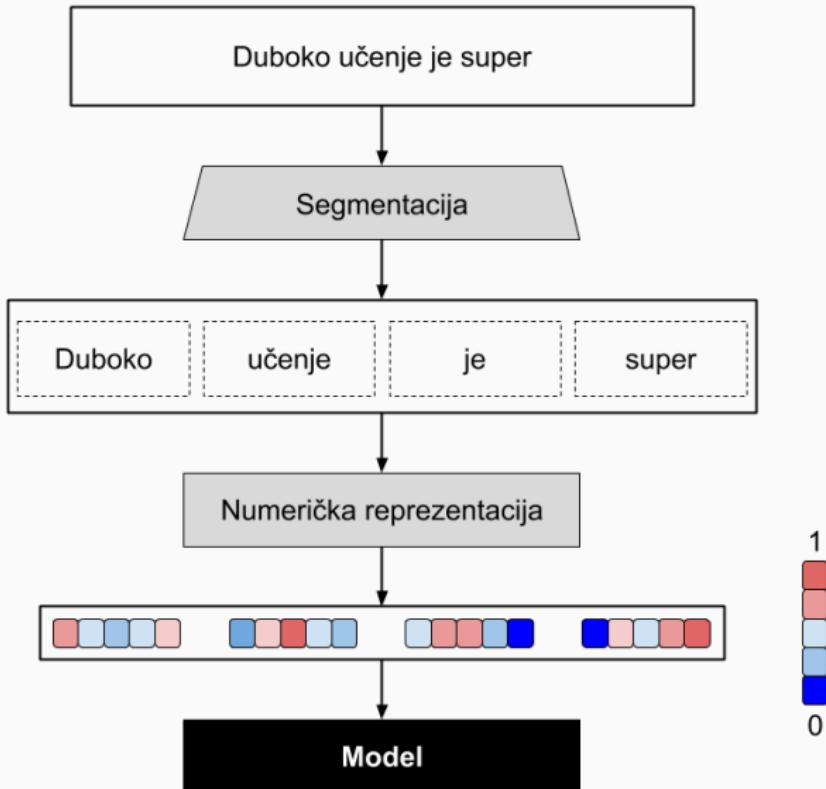
Jezik računalima izgleda otprilike kao što nama izgleda japanski (ali samo za nas koji ne razumijemo japanski)

Značenja i međusobna sličnost jezičnih jedinica su nam nepoznati.

Prije obrade tekstnih podataka algoritmima strojnog učenja možemo rijeći prvo prevesti u vektorske reprezentacije.

Reprezentacije dijelova rečenica

Obrada teksta



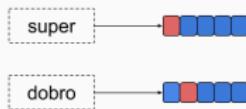
Ovo predavanje:

Segmentacija teksta:

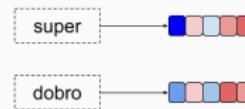
- tekst možemo segmentirati na **riječi**, slova ili podriječi¹

Numerička reprezentacija:

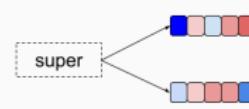
- svakoj riječi dodijeliti **gustu** visokodimenzionalnu reprezentaciju
- alternative: jednojedinične (\rightarrow vreće riječi) i višeprototipne.



(a)



(b)



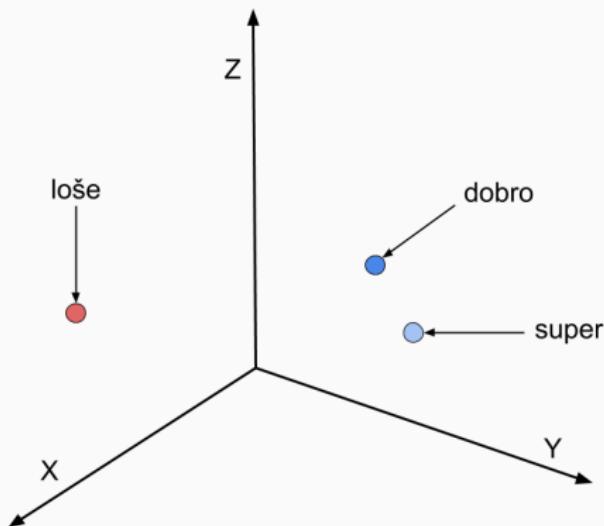
(c)

Jednojedinične (a), gусте (b) и вишеједиџитне (c) репрезентације.

¹<https://github.com/google/sentencepiece>

Guste reprezentacije riječi

Udaljenost u prostoru reprezentacija trebala bi odgovarati semantičkoj ili sintaktičkoj sličnosti među riječima:



- kolika je udaljenost između jednojediničnih reprezentacija?
- kako izračunati udaljenost između višeprototipnih reprezentacija?

Guste reprezentacije riječi

U praksi, dimenzionalnost reprezentacija puno je veća nego na slici.

Moramo se osloniti na intuicije iz niskodimenzionalnih prostora, iako one mogu biti varljive.

- If you are not used to thinking about hyper-planes in high-dimensional spaces, now is the time to learn.
- To deal with hyper-planes in a 14-dimensional space, visualize a 3-D space and say “fourteen” to yourself very loudly. **Everyone does it.**
 - But remember that going from 13-D to 14-D creates as much extra complexity as going from 2-D to 3-D.

Predavanja "Neural Networks for Machine Learning", Geoffrey Hinton

Reprezentacije riječi

Početne reprezentacije riječi mogu se prednaučiti na nekom pomoćnom zadatku:

- pomoćni zadatci pogađaju skrivene riječi u zadanim lokalnim kontekstima na velikim korpusima stvarnog jezika (Wikipedia, Common crawl²)
- modeli: word2vec³ (CBOW, Skip-gram), GloVe⁴, FastText⁵
- učenje optimira matricu ugrađivanja čiji retci sadrže vektorske reprezentacije odgovarajućih riječi.

U ovom kolegiju nećemo razmatrati pred-učenje reprezentacija:

- umjesto toga, koristit ćemo ili slučajne inicijalizacije ili već prednaučene reprezentacije.

²<https://commoncrawl.org/>

³<https://en.wikipedia.org/wiki/Word2vec>

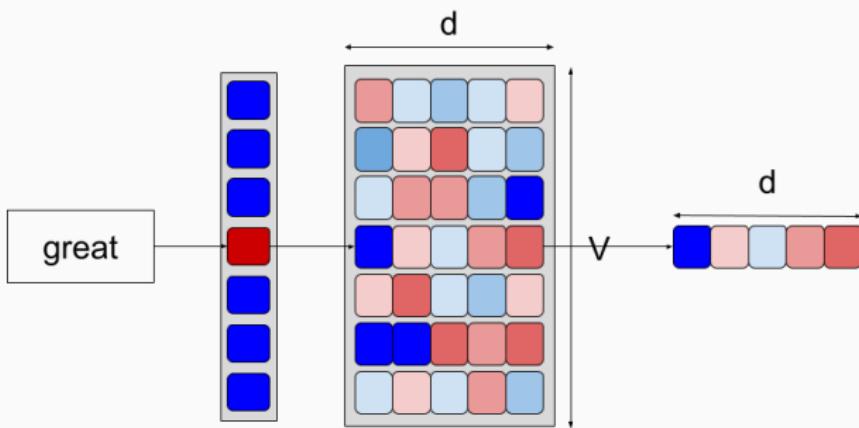
⁴<https://nlp.stanford.edu/projects/glove/>

⁵<https://fasttext.cc/>

Matrice ugrađivanja u praksi

Reprezentacije riječi x_i dobivamo množenjem retčanog jednojediničnog koda riječi e_i s matricom ugrađivanja $E \in \mathbb{R}^{V \times d}$:

$$x_i = e_i \cdot E \quad e_i = \underbrace{[0, \dots, 0]}_V, \underbrace{1, 0, \dots, 0}_V$$



Reprezentacije riječi: hiperparametri

Veličina rječnika: V

- broj jedinstvenih riječi koje model prepoznaže
- ovisi o dostupnoj memoriji i važnosti riječi⁶
- u praksi obično uzimamo V najčešćih riječi iz korpusa
- preostale riječi obično i) izostavljamo ili ii) mijenjamo simbolom $\langle\text{UNK}\rangle$

Dimenzionalnost ugrađivanja: d

- uobičajeni izbor $d = 300$ pokazuje dobra svojstva na različitim zadatcima⁷
- u praksi, izbor će ovisiti i o dostupnoj memoriji, ciljanoj generalizacijskoj moći, te dostupnosti prednaučenih ugrađivanja.

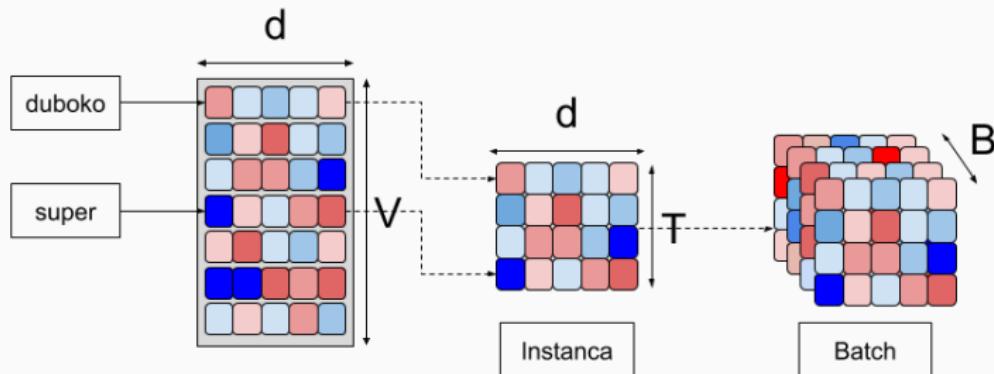
⁶ rijetke i neinformativne riječi obično se izostavljaju

⁷ loši izbori mogu dovesti do podnaučenosti ili prenaučenosti

Predstavljanje teksta

Odlomke i rečenice predstavljamo ulančavanjem reprezentacija riječi.

Duljinu teksta označavamo brojem riječi T (vremenska dimenzija).



Ovaj pristup mora riješiti nekoliko problema: možete li ih prepoznati?

Predstavljanje teksta

Problem: vremenska dimenzija mijenja se od rečenice do rečenice

- rečenice grupe za učenje imat će različite duljine
- u praksi, ovo rješavamo nadopunjavanjem i odsijecanjem.

Cilj 1: model obrađuje svaku riječ na isti način

- ovo ograničava dimenzionalnost modela i smanjuje prenaučenost.

Cilj 2: model treba biti osjetljiv na redoslijed riječi

- u suprotnom ne bismo mogli razlikovati rečenice sastavljene od istih riječi:
 - *Dog eats cat* i *Cat eats dog.*

Oba cilja možemo postići **povratnim modelima**

- međutim, prvo ćemo pogledati alternative za postizanje ovih ciljeva.

Predstavljanje teksta: vremensko agregiranje sažimanjem

Ideja: predstaviti tekst sažimanjem reprezentacija riječi (eng. *mean/average pooling*)

$$r_\mu(\text{text}) = \frac{1}{T} \sum_t^T x^{(t)}$$

- ovakvo agregiranje gubi informaciju o položaju riječi u tekstu, ali ne zahtijeva **parametre**

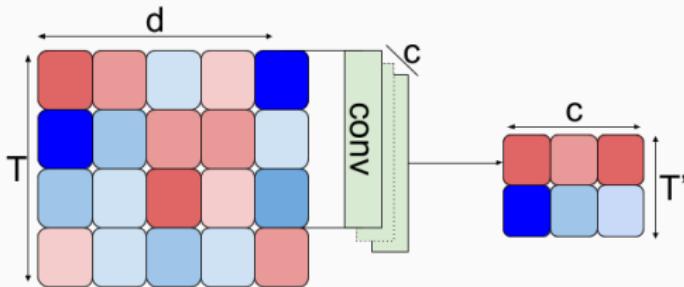
Varijante ovog pristupa: sažimanje zbrajanjem, **otežano sažimanje**

$$r_w(\text{text}) = \sum_t^T w_t x^{(t)} \quad w_t = f(x^{(t)}, \text{text})$$

- otežano sažimanje množi reprezentacije riječi faktorom koji ovisi o tekstu i o riječi (eg. IDF)

Predstavljanje teksta: 1D convolution

Konvolucije zadržavaju informaciju o redoslijedu riječi:



I dalje trebamo sažimanje ako nam treba predikcija za cijeli odlomak.

Jednostavne konvolucijske arhitekture propuštaju *globalni* kontekst:

- može se ublažiti povećanjem dubine...
- ... ili dilatiranim konvolucijama ili vremenskim sažimanjem.

Predstavljanje teksta: sažetak

Obrada prirodnog teksta (NLP) složen je zadatak:

- segmentirati tekst u riječi, **podriječi** ili slova
- detektirati i ispraviti tipografske greške
- istu misao možemo izraziti na različne načine

Prepostavke u okviru ovih predavanja:

- tekst segmentiran na riječi
- reprezentacije riječi prednaučene ili slučajno inicijalizirane
- svakoj riječi dodjelujemo točno *jedno* ugrađivanje

Pristupi za klasifikaciju teksta varijabilne duljine:

- konvolucije i sažimanja
- povratni modeli

Obični povratni modeli

Povratni modeli: motivacija

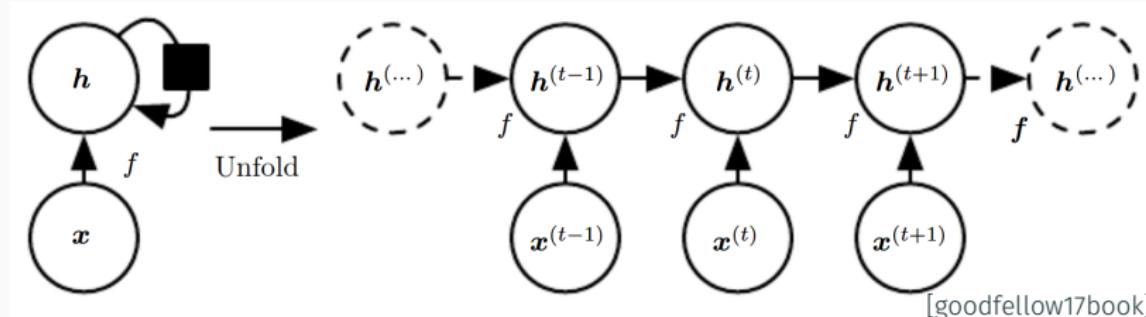
Tražimo prikladni model za slijedne podatke:

- proizvoljna duljina
- broj parametara ne ovisi o duljini slijeda
- model je osjetljiv na redoslijed podataka

Inspirirani **dinamičkim sustavima**:

- model f ažurira skriveno stanje $h^{(t)}$ s obzirom na ulaz $x^{(t)}$:

$$h^{(t)} = f(x^{(t)}, h^{(t-1)})$$



[goodfellow17book]

Povratni modeli: formulacija

Povratni model f ekvivariantan je s obzirom na vrijeme (položaj u slijedu):

$$h^{(t)} = f(x^{(t)}, h^{(t-1)})$$

Jednostavni povratni model imaju jednu nelinearnost po ulazu:

$$h^{(t)} = g(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

- W_{hh}, W_{xh}, b_h - parametri povratne afine transformacije
- $a^{(t)}$ - linearna povratna mjera
- g - nelinearnost (sigmoide, tanh,...)

Povratni modeli: intuicija

Stanje $h^{(t)}$ predstavlja **viđeni** dio slijeda:

- ako je model dobro naučen, h će kodirati značenje teksta

Matrica W_{xh} projicira ulaz u prostor reprezentacije stanja

- ovo bi trebalo odbaciti nepotrebne informacije

Matrica W_{hh} modelira evoluciju stanja:

- ona opisuje utjecaj vremena i ulaza i odbacuje nepotrebne informacije iz stanja

$$h^{(t)} = g(W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h)$$

Dimenzionalnosti ulaza i stanja **mogu se razlikovati**:

$$h \in \mathbb{R}^h \quad x \in \mathbb{R}^d$$

Povratni modeli: izlazni sloj

Stanje $h^{(t)}$ predstavlja svu viđenu informaciju:

- osim značenja, stanje pamti i neke dodatne informacije
 - npr. dvomislene riječi ili navode osoba ili lokacija
- neke od ovih informacija mogu biti **nebitne** za predikciju.

Izlazni sloj projicira stanje na prostor predikcija:

$$o^{(t)} = W_{hy}h^{(t)} + b_o \quad (1)$$

- ovaj korak *filtrira* nevažne informacije
- vektor $o^{(t)}$ sadrži logite predikcija u trenutku t
- [!!] RNN ćelija Pytorcha ne sadrži izlaznu projekciju – morate je sami dodati.

Povratni modeli: formulacija

Jednostavni povratni model definiran je sljedećim jednadžbama

1. ažuriranje skrivenog stanja:

$$h^{(t)} = \tanh(W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h)$$

2. projekcija u prostor izlaznih logita:

$$o^{(t)} = W_{hy}h^{(t)} + b_o$$

U praksi, stanje obično aktiviramo hiperbolnim tangensom:

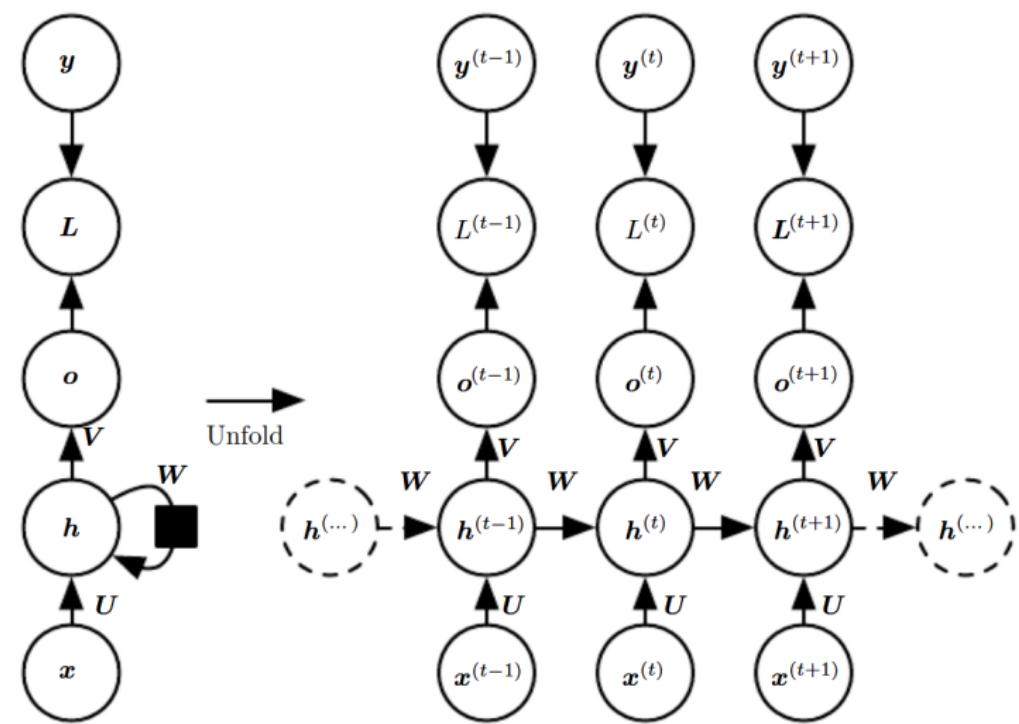
- preostale mogućnosti su sigmoida i zglobnica.

Dimenzije parametara: $W_{hh} \in \mathbb{R}^{h \times h}$, $W_{xh} \in \mathbb{R}^{h \times d}$, $W_{hy} \in \mathbb{R}^{y \times h}$

- d i y označavaju ulaznu i izlaznu dimenzionalnost

[!!] Notacija iz knjige: $W := W_{hh}$, $U := W_{xh}$, $V := W_{hy}$, $b := b_h$, $c := b_o$

Povratni modeli: vizualizacija



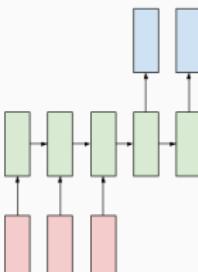
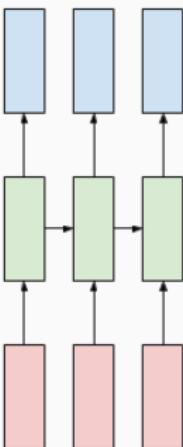
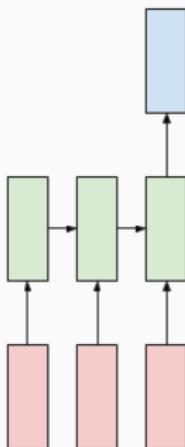
Razmotrani povratni model s ulazima $x^{(t)}$, gubitkom $L^{(t)}$ i izlazima $o^{(t)}$.

Povratni modeli: zadatci

Povratni model može generirati izlaz u svakom trenutku t – ali treba li nam to?

- konfiguracija izlaza modela ovisi o zadatku!

Pokušajte smisliti zadatke koji zahtijevaju samo jedan izlaz **(a)** po jedan izlaz za svaki ulaz **(b)**, ili promjenljivi broj izlaza ali više od jednog **(c)**.

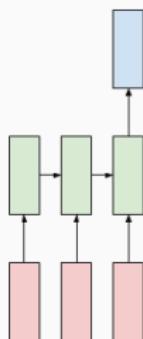


(a)

(b)

(c)

Klasifikacija slijeda: temeljni problem u razumijevanju jezika.



Klasifikacija slijeda

Potrebno je odrediti jednu izlaznu varijablu za cijeli ulazni niz

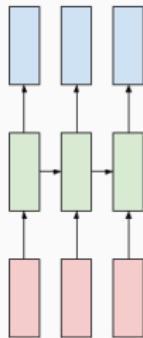
Različiti klasifikacijski problemi:

- analiza sentimenta,
- kategorizacija dokumenata,
- određivanje glazbenog žanra, ...

Izlazni sloj prima samo **posljednje** latentno stanje.

Jezični zadatci

"Označavanje" slijeda: zahtijevamo predikciju u svakom ulazu.



"Označavanje"
slijeda

Potrebno odrediti slijed varijabli: svaka
predikcija odgovara točno jednom ulazu.

Različiti zadatci gусте предикције:

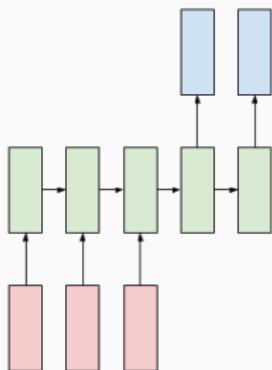
- предикција врсте ријечи,
- распознавање именованих ентитета,
- саџимање текста екстракцијом,
- детекција оквира видеа, ...

Predikcije često generiramo s malim vremenskim pomakom u
odnosu na odgovarajuće ulaze.

Naveli smo navodnike jer riječ *označavanje* obično koristimo kad
radnju vrše osobe (bolje: označavanje slijeda → gusta predikcija).

Jezični zadatci

Zadatci oblika slijed-u-slijed (*seq2seq*) podrazumijevaju složenije oblike preslikavanja ulaznog slijeda u izlazni.



slijed-u-slijed

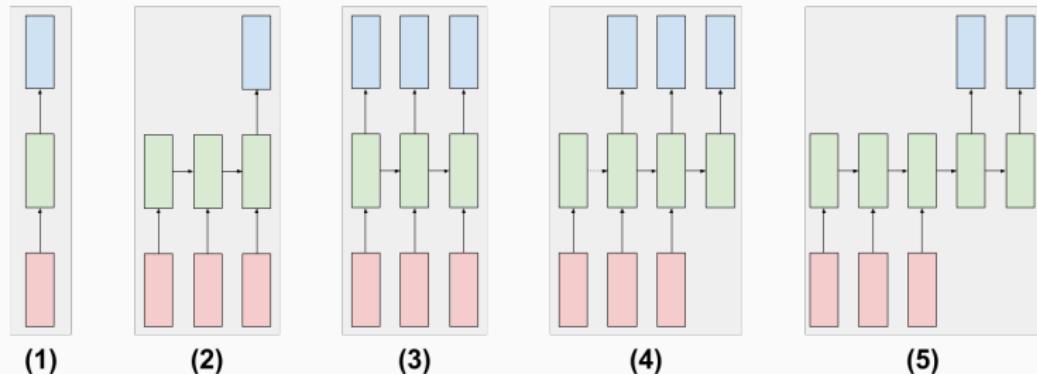
Zadatak je ulazni slijed preslikati u ciljni slijed nepoznate duljine

- strojno prevođenje,
- apstraktivno sažimanje teksta, ...

Razmotrite probleme koji bi se ovdje mogli javiti:

- kako započeti prijevod?
- bismo li uvijek trebali prediktirati najvjerojatniju riječ?
- kako odlučiti kada zaključiti predikciju?

Jezični zadatci



Smjernice za prepoznavanje zadataka:

- kad god broj izlaza odgovara broju ulaza, zadatak odgovara gustoj predikciji, neovisno o tome postoji li pomak **(4)** ili ne **(3)**
- zadatke oblika slijed-u-slijed **(5)** često razdvajamo na apsorbiranje ulaza (gdje ne provodimo predikciju) i proizvođenje izlaza (gdje generiramo čitavi izlazni slijed)

Analiza: klasifikacija slijeda

Analiza sentimenta

Klasifikacijski problem: odrediti sentiment ulaznog teksta

- binarna formulacija: pozitivan/negativan
- kategorička formulacija: brojčana ocjena [1,10]
 - ovo bi se moglo formulirati i kao regresija...
- skupovi podataka: IMDB⁸, YELP⁹
- pozitivan primjer iz IMDB:

... was the story that blew me away. hurray for Takahisa

Naša analiza pretpostavlja binarnu formulaciju i zanemaruje (značajne) probleme predobrade teksta.

⁸<https://ai.stanford.edu/~amaas/data/sentiment/>

⁹<https://www.yelp.com/dataset/>

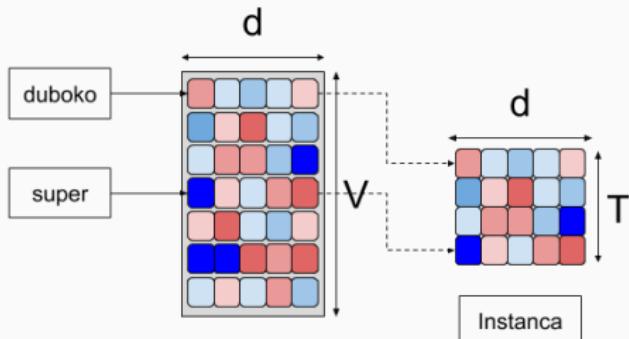
Analiza sentimenta: uvod

Ciljna varijabla:

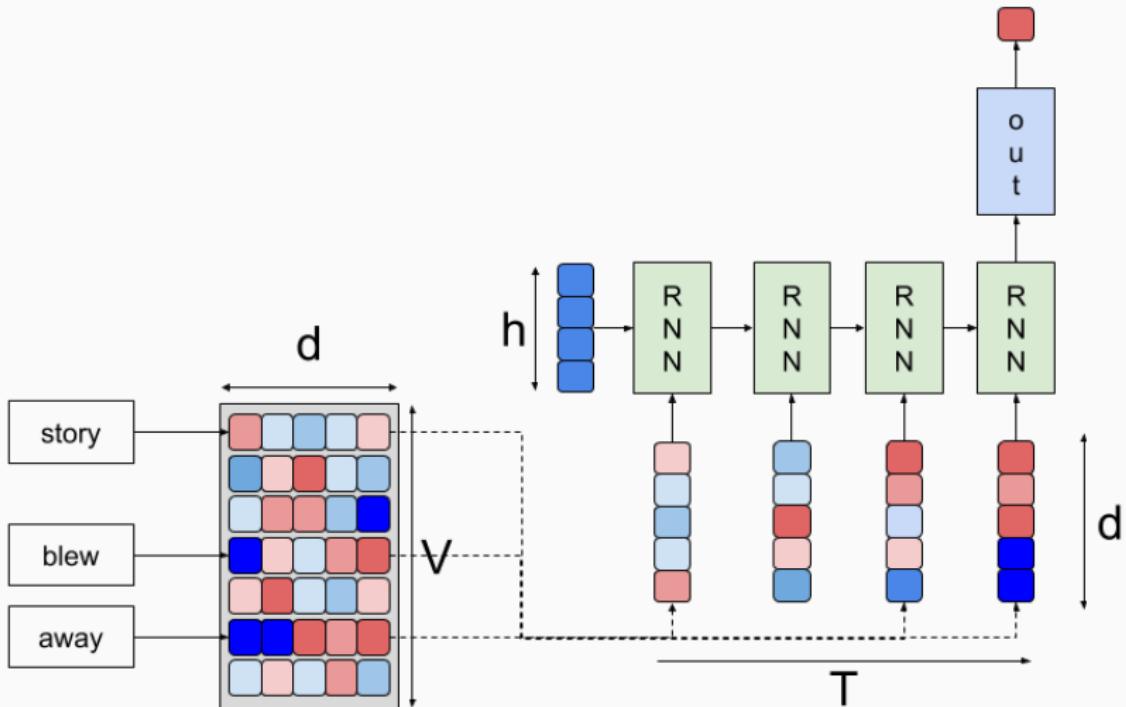
- Sentiment: {pozitivan, negativan}
- Vrijednosti varijable predstavljamo indeksima: {0, 1}

Ulagne varijable:

- slijed reprezentacija riječi, podriječi ili slova
- moramo odabrati veličinu vokabulara i svaku riječ zamijeniti s odgovarajućim ugrađivanjem ili simbolom <UNK>



Analiza sentimenta: skica



Analiza sentimenta: pseudokod

Algoritam 1: Plain RNN for sequence classification

Single output RNN ($X, W_{hh}, W_{xh}, W_{hy}, b_h, b_o$)

inputs: A sequence of vectors X , parameters $W_{...}$, $b_{...}$

output: Logits o

$h_t \leftarrow \text{zero_init};$

foreach x in X **do**

$a_t \leftarrow W_{hh}h_t + W_{xh}x + b_h;$

$h_t \leftarrow \tanh(a_t);$

end

$o \leftarrow W_{hy}h_t + b_y;$

return o ;

Ovaj algoritam možemo prilagoditi za gustu predikciju tako da generiranje izlaznih elemenata jednostavno pomaknemo u petlju.

Analiza: gusta predikcija

Raspoznavanje vrste riječi

Gusta predikcija vrste riječi za svaku riječ ulaza: raspoznavanje

- višerazredni izlaz: imenica, pridjev, glagol, ...

Skupovi: Penn Treebank (PTB) (paywall), Universal Dependencies¹⁰ (UD)

- PTB taxonomy: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Primjer rečenice iz korpusa SETimes (dio UD) i odgovarajuće oznake:

U Bruxellesu izrazili su zabilještjili.
ADP PROPN VERB AUX NOUN .

¹⁰<https://universaldependencies.org/>

Raspoznavanje vrste riječi: uvod

Ciljne varijable:

- vrste riječi: $\{ADP, PROPN, \dots\}$
- koristimo indekse razreda: $\{1, \dots, Y\}$

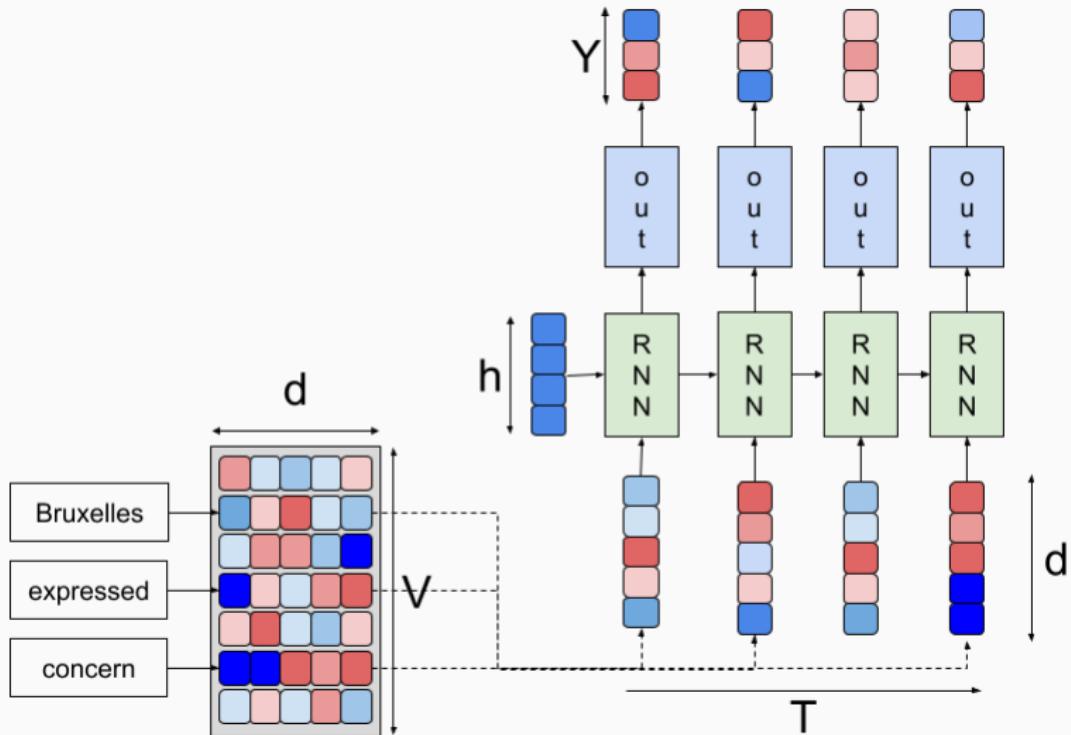
Ulazne varijable:

- slijed reprezentacija riječi, podriječi ili slova
- moramo odabrati veličinu vokabulara i svaku riječ zamijeniti s odgovarajućim ugrađivanjem ili simbolom **<UNK>**

Vrlo slično klasifikaciji slijeda

- važna razlika: dimenzija ciljne varijable (2 vs N)
- koristimo guste predikcije umjesto predikcija koje obuhvaćaju cijeli slijed

Part-of-speech tagging: top-level sketch



Raspoznavanje vrste riječi: pseudokod

Algoritam 2: Pseudokod RNNa za višerazrednu predikciju na razini simbola

Dense sequence prediction RNN ($X, W_{hh}, W_{xh}, W_{hy}, b_h, b_o$)

inputs: A sequence of vectors X , parameters $W_{...}, b_{...}$

output: A sequence of logits \hat{y}

$h_t \leftarrow \text{zero_init};$

$o \leftarrow [];$

foreach x in X **do**

$a_t \leftarrow W_{hh}h_t + W_{xh}x + b_h;$

$h_t \leftarrow \tanh(a_t);$

$o_t \leftarrow W_{hy}h_t + b_y;$

$o.append(o_t)$

end

return o ;

Povratni modeli: sažetak

Povratni model djeluje kao dinamički sustav:

- ulazna informacija ugrađuje se u latentno stanje korak po korak
- latentno stanje iterativno se ažurira s obzirom na staro stanje i tekući ulaz

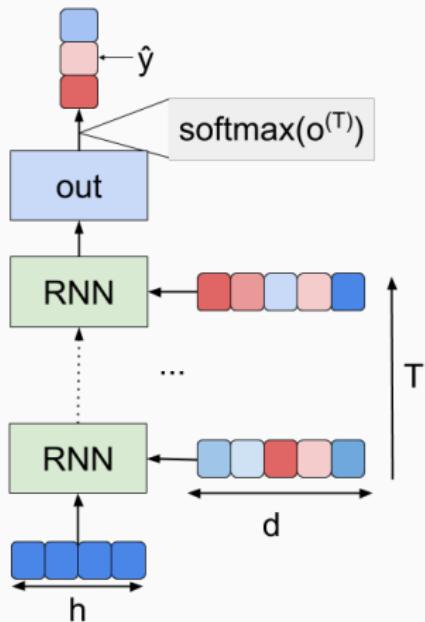
Stanje najčešće ne sadržava predikcije:

- predikcije izražavamo projiciranjem stanja
- broj izlaznih projekcija ovisi o vrsti zadatka (samo jedna, jedna po ulazu ili varijabilan broj)
- ako ima više izlaza, parametri izlaznih projekcija se **dijele**

Tri glavna zadatka raspoznavanja slijedova su **raspoznavanje slijeda, gusta predikcija i prevođenje iz slijeda u slijed.**

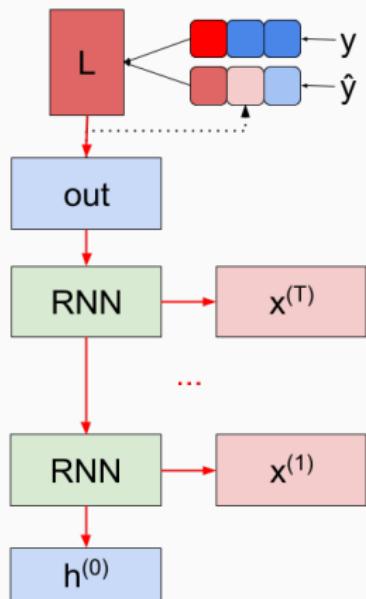
Učenje povratnih modela

Povratni model možemo razmotati u klasični potpuno povezani model s odvojenim ulazima i dijeljenim parametrima:

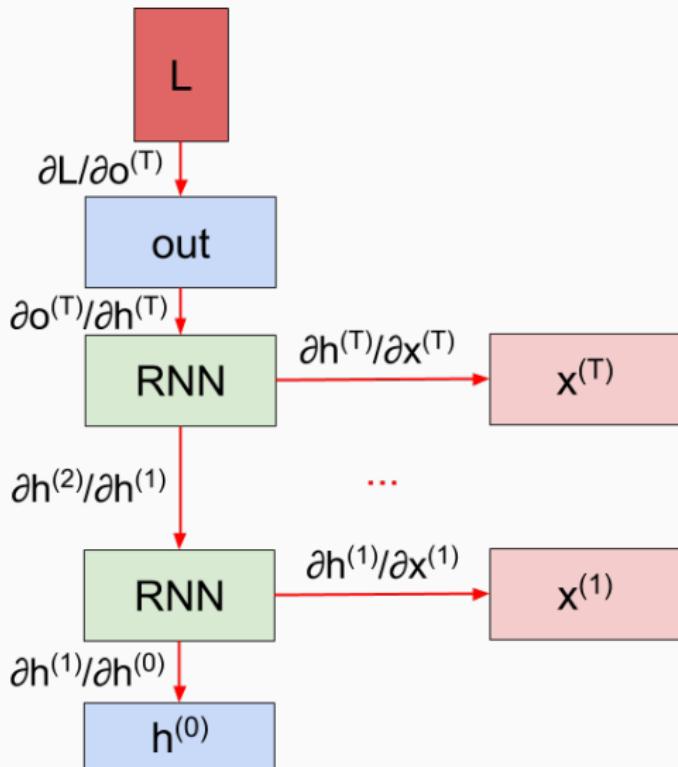


Primjer: klasifikacija niza

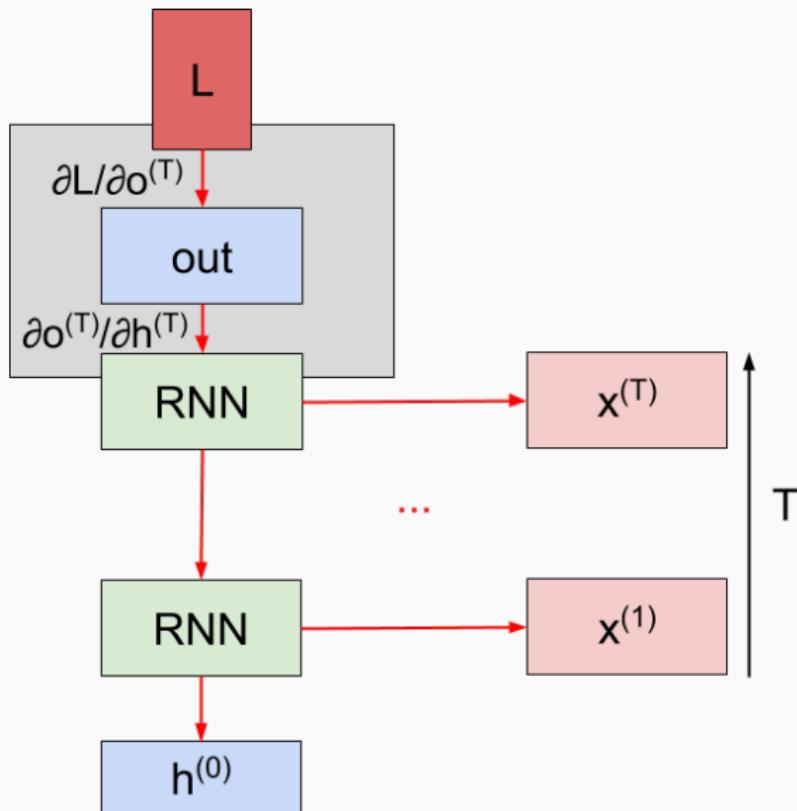
Model za prediktiranje na razini cijelog niza možemo učiti standardnim backpropom:



Primjer: klasifikacija niza (2)



Povratno učenje: prolaz kroz izlazni sloj



Povratno učenje: prolaz kroz izlazni sloj (2)

Predikcije dobivamo aktiviranjem izlaza (podsjetnik):

$$\begin{aligned}o^{(T)} &= W_{hy} h^{(T)} + b_y \\ \hat{y} &= \text{softmax}(o^{(T)})\end{aligned}$$

Prepostavljamo unakrsnu entropiju kao standardni gubitak:

$$\mathbb{L} = - \sum_j y_j \cdot \log \hat{y}_j$$

Gradijenti s obzirom na izlazne aktivacije:

$$\frac{\partial \mathbb{L}}{\partial o^{(T)}} = (\underbrace{\text{softmax}(o^{(T)}) - y}_{\hat{y}})^\top$$

Povratno učenje: prolaz kroz izlazni sloj (3)

Jednadžbe predikcije izlaza (podsjetnik):

$$o^{(T)} = W_{hy} h^{(T)} + b_y$$

Gradijenti s obzirom na parametre izlaznog sloja W_{hy} i b_y :

$$\frac{\partial \mathbb{L}}{\partial W_{hy}} = \frac{\partial \mathbb{L}}{\partial o^{(T)}} \frac{\partial o^{(T)}}{\partial W_{hy}} = \dots = (\hat{y} - y) \cdot (h^{(T)})^\top$$

$$\frac{\partial \mathbb{L}}{\partial b_y} = \frac{\partial \mathbb{L}}{\partial o^{(T)}} \frac{\partial o^{(T)}}{\partial b_y} = (\hat{y} - y)^\top$$

Gradient s obzirom na posljednje stanje:

$$\frac{\partial \mathbb{L}}{\partial h^{(T)}} = \frac{\partial \mathbb{L}}{\partial o^{(T)}} \frac{\partial o^{(T)}}{\partial h^{(T)}} = (\hat{y} - y)^\top W_{hy}$$

Povratno učenje: ažuriranje stanja

Stanje u trenutku t ovisi o stanjima u svim prethodnim trenutcima:

$$h^{(t)} = f(h^{(t-1)}, \dots, h^{(0)})$$

Unatražni prolaz u koraku $t = T$ (posljednji korak, podsjetnik)

$$\frac{\partial \mathbb{L}}{\partial h^{(T)}} = \frac{\partial \mathbb{L}}{\partial o^{(T)}} \frac{\partial o^{(T)}}{\partial h^{(T)}} = \frac{\partial \mathbb{L}}{\partial o^{(T)}} W_{hy} = (\hat{y} - y)^\top W_{hy}$$

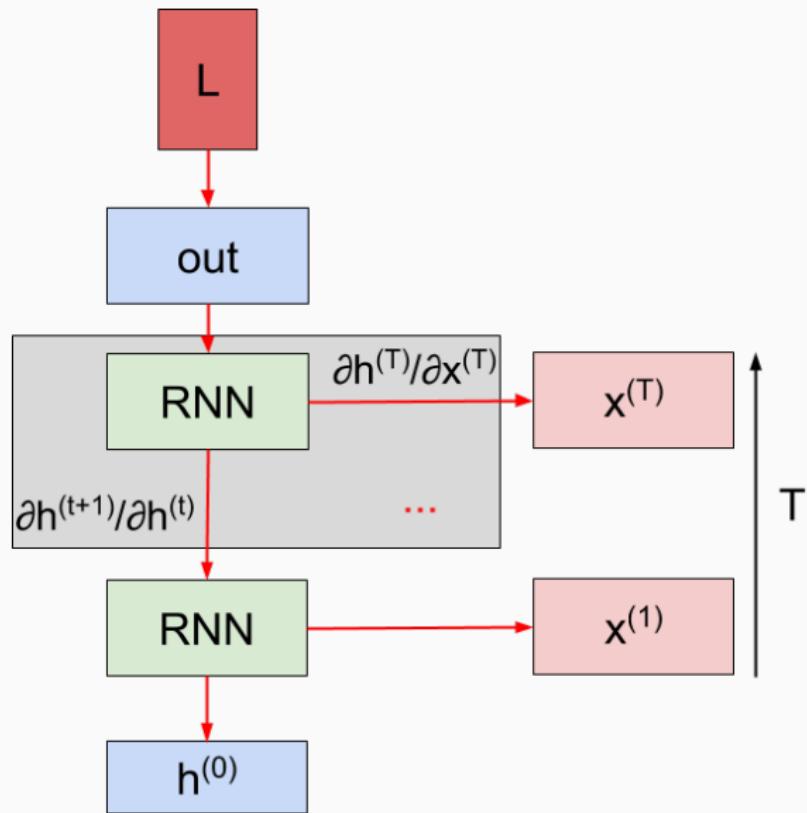
Za $t < T$ (preostale vremenske korake)

$$\frac{\partial \mathbb{L}}{\partial h^{(t)}} = \frac{\partial \mathbb{L}}{\partial h^{(T)}} \frac{\partial h^{(T)}}{\partial h^{(T-1)}} \cdots \frac{\partial h^{(t+1)}}{\partial h^{(t)}}$$

Općenito, do svakog stanja mogu doći gradijenti sljedeće dvije vrste:

1. gradijenti iz **odgovarajućih** predikcija (odozgo)
(samo kod guste predikcije)
2. gradijenti iz **budućih** skrivenih stanja (zdesna)

Povratno učenje: prolaz kroz ažuriranje stanja (2)



Povratno učenje: prolaz kroz ažuriranje stanja (3)

Jednadžba ažuriranje stanja (podsjetnik):

$$h^{(t)} = \tanh(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

Treba nam derivacija hiperbolnog tangensa:

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

$$\frac{d\tanh(x)}{dx} = 1 - \tanh^2(x)$$

Hiperbolni tangens je sigmoida čiji izlaz je *rastegnut* preko $[-1, 1]$

$$\tanh(x) = 2\sigma(2x) - 1$$

Povratno učenje: prolaz kroz ažuriranje stanja (4)

Jednadžba ažuriranja stanja (podsjetnik):

$$h^{(t)} = \tanh(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

Gradijent s obzirom na predaktivaciju $a^{(t)}$:

$$\frac{\partial \mathbb{L}}{\partial a^{(t)}} = \frac{\partial \mathbb{L}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial a^{(t)}} = \frac{\partial \mathbb{L}}{\partial h^{(t)}} \frac{\partial \tanh}{\partial a^{(t)}} = \dots = \frac{\partial \mathbb{L}}{\partial h^{(t)}} \odot (1 - h^{(t)^2})$$

Gradijenti s obzirom na parametre povratne ćelije (W_{hh} , W_{xh} , b_h):

$$\frac{\partial \mathbb{L}}{\partial W_{hh}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial W_{hh}} = \dots = \left(\frac{\partial \mathbb{L}}{\partial a^{(t)}} \right)^\top \left(h^{(t-1)} \right)^\top$$

$$\frac{\partial \mathbb{L}}{\partial W_{xh}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial W_{xh}} = \dots = \left(\frac{\partial \mathbb{L}}{\partial a^{(t)}} \right)^\top \left(x^{(t)} \right)^\top$$

$$\frac{\partial \mathbb{L}}{\partial b_h} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial b_h} = \frac{\partial \mathbb{L}}{\partial a^{(t)}}$$

povratno učenje: gradijenti s obzirom na ulaze

Jednadžba ažuriranja stanja (podsjetnik):

$$h^{(t)} = \tanh(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

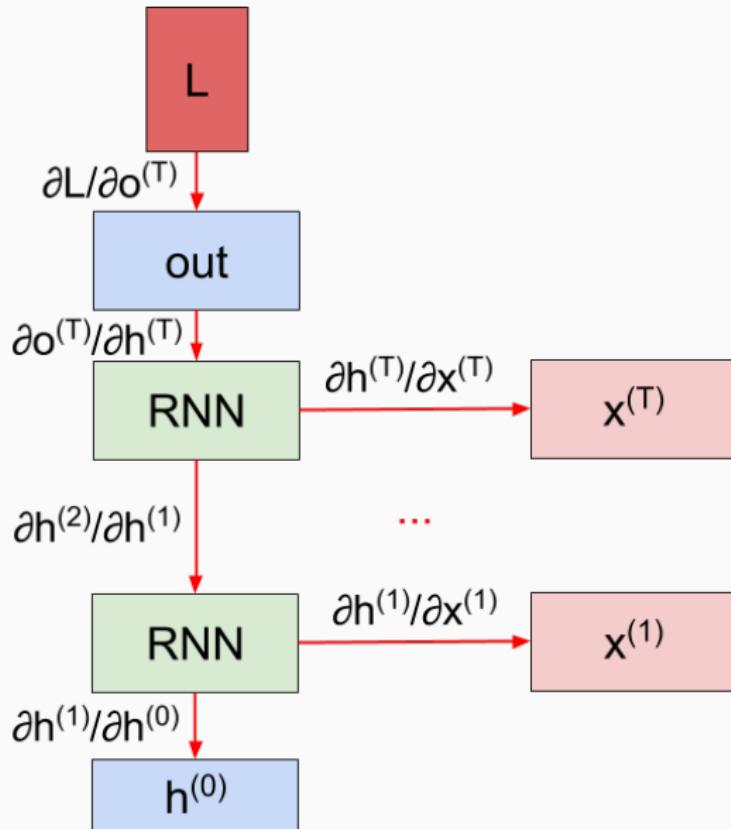
1. gradijent s obzirom na prethodno stanje:

$$\frac{\partial \mathbb{L}}{\partial h^{(t-1)}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial h^{(t-1)}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} W_{hh}$$

2. gradijenti s obzirom na ulaz (razmislite zašto!):

$$\frac{\partial \mathbb{L}}{\partial x^{(t)}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial x^{(t)}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} W_{xh}$$

povratno učenje: klasifikacija slijeda, sažetak



povratno učenje: klasifikacija slijeda, sažetak (2)

Gradijenti gubitka dolaze do parametara ažuriranja stanja kroz svako skriveno stanje:

$$\frac{\partial \mathbb{L}}{\partial W_{hh}} = \frac{\partial \mathbb{L}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial W_{hh}} = \underbrace{\left(\frac{\partial \mathbb{L}}{\partial a^{(t)}} \right)^\top}_{\forall t \in \{1, 2, \dots, T\}} \left(h^{(t-1)} \right)^\top$$

Ti gradijenti **akumuliraju** se kroz vrijeme (jednako za W_{xh} !):

$$\frac{\partial \mathbb{L}}{\partial W_{hh}} = \sum_{t=1}^T \frac{\partial \mathbb{L}}{\partial W_{hh}} = \sum_{t=1}^T \left(\frac{\partial \mathbb{L}}{\partial a^{(t)}} \right)^\top \left(h^{(t-1)} \right)^\top$$

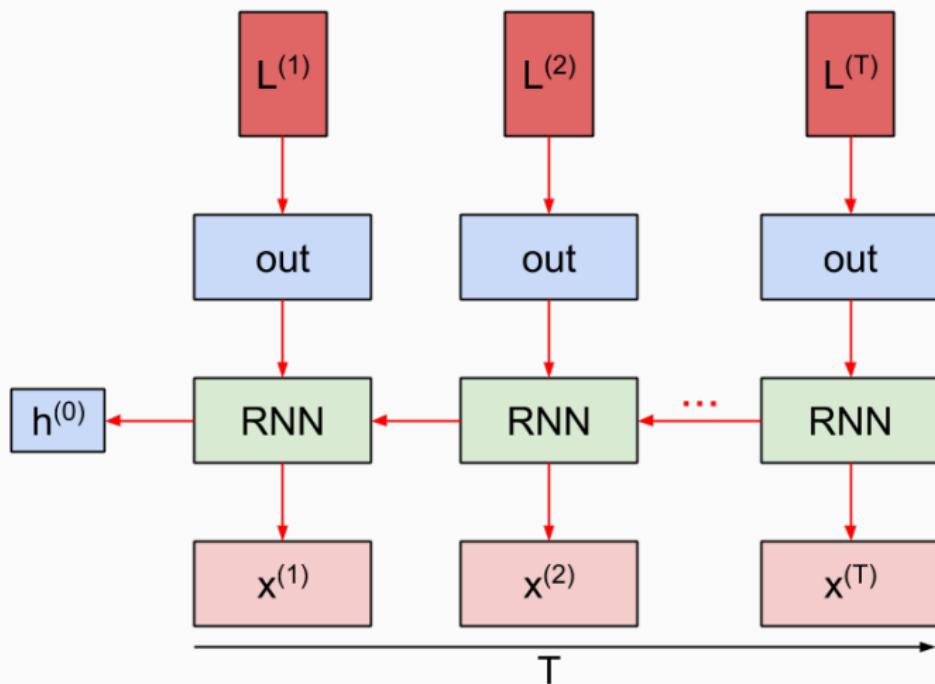
Optimizacijski korak koristi akumulirani gradijent.

Motivacija za računanje gradijenata s obzirom na ulaze ($x^{(t)}$):

1. učenje (ugađanje) ugrađivanja riječi
2. ulaz može odgovarati izlazu drugog povratnog modula

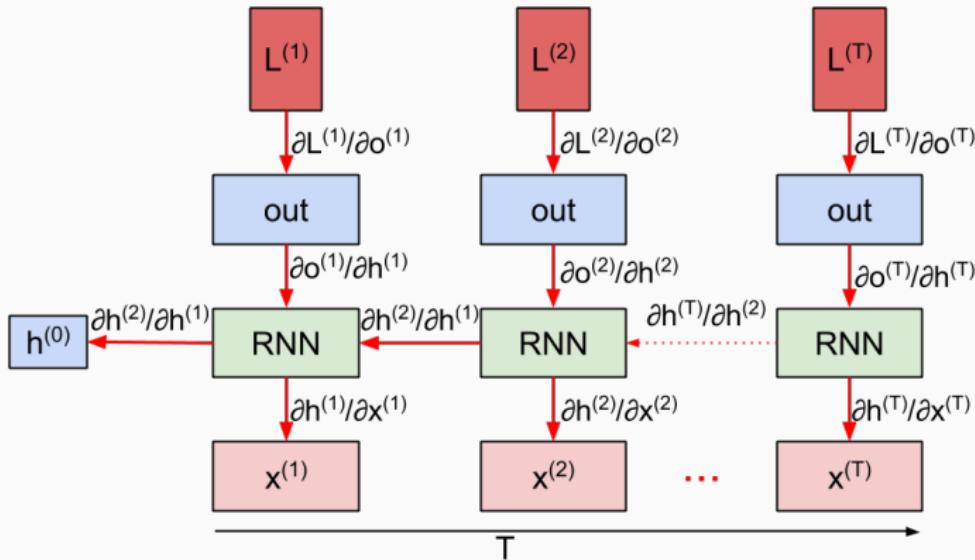
Povratno učenje: gusta predikcija - gubitak

Gusta predikcija implicira gubitak u **svakom** vremenskom koraku



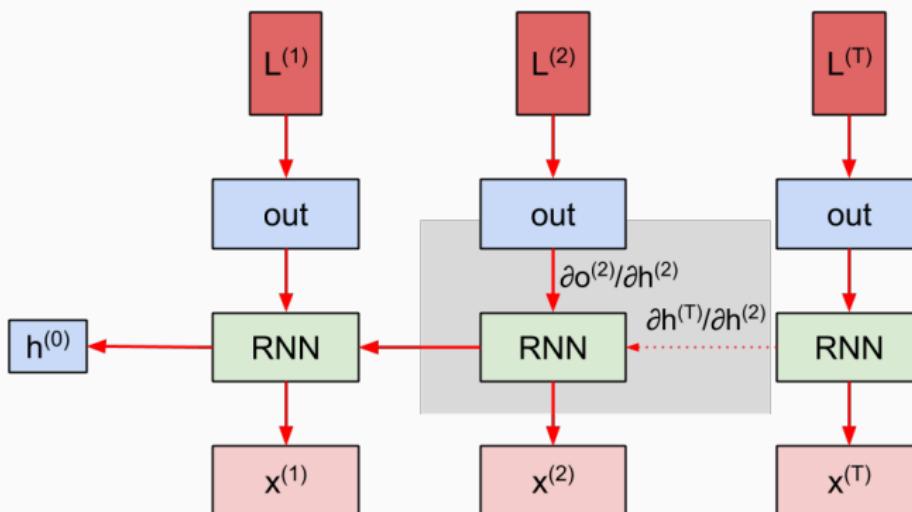
Povratno učenje: gusta predikcija - propagiranje gradijenta

U gustom slučaju moramo akumulirati doprinose članova gubitka iz svih budućih koraka:



Povratno učenje: gusta predikcija - gradijenti

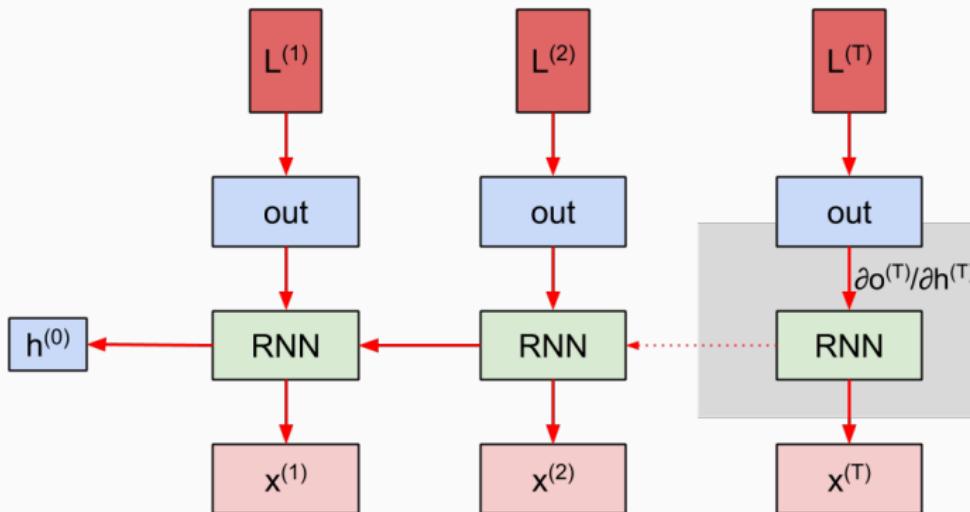
Na ažuriranje stanja utječu tekući gubitak (vertikalna ovisnost) i svi budući članovi gubitka (vodoravna ovisnost):



Povratno učenje: gusta predikcija - gradijenti (2)

Na ažuriranje stanja utječu tekući gubitak (vertikalna ovisnost) i svi budući članovi gubitka (vodoravna ovisnost):

- ... osim u posljednjem vremenskom koraku kad ne trebamo nikakav doprinos iz budućnosti



Povratno učenje: gusta predikcija - gradijenti (3)

Gubitak odgovara **zbroju** vremenskih komponenata $\mathbb{L} = \sum_t \mathbb{L}^{(t)}$

- ako gradijente računamo *ručno* trebamo posebno paziti na $\frac{\partial \mathbb{L}}{\partial h^{(t)}}$

Posljednji vremenski korak ($t = T$) isti kao i u klasifikacijskom slučaju:

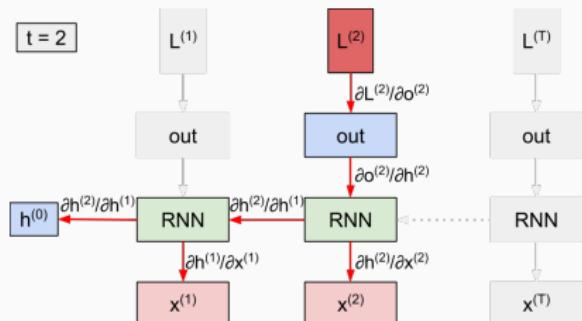
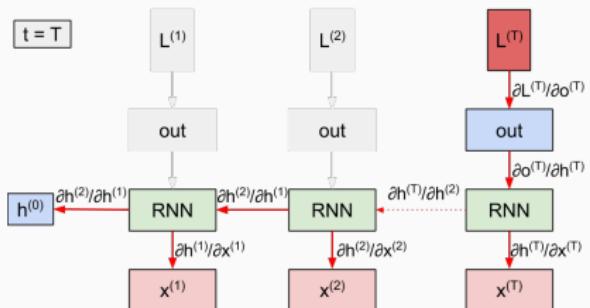
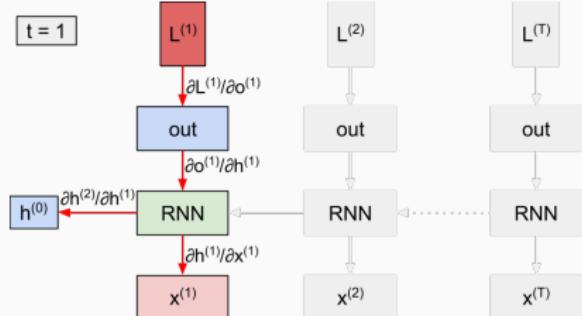
$$\frac{\partial \mathbb{L}}{\partial h^{(t)}} = \frac{\partial \mathbb{L}^{(t)}}{\partial h^{(t)}} = \frac{\partial \mathbb{L}^{(t)}}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} = \frac{\partial \mathbb{L}^{(t)}}{\partial o^{(t)}} W_{hy}$$

Preostali vremenski koraci ($t < T$) zahtijevaju posebnu pažnju:

$$\frac{\partial \mathbb{L}}{\partial h^{(t)}} = \underbrace{\frac{\partial \mathbb{L}^{(t)}}{\partial h^{(t)}}}_{\text{tekući gubitak}} + \underbrace{\frac{\partial \mathbb{L}^{(t^* > t)}}{\partial h^{(t)}}}_{\text{budući gubitci}}$$

$$\frac{\partial \mathbb{L}^{(t^* > t)}}{\partial h^{(t)}} = \sum_{t^* > t}^T \frac{\partial \mathbb{L}^{(t^*)}}{\partial h^{(t)}} = \sum_{t^* > t}^T \frac{\partial \mathbb{L}^{(t^*)}}{\partial h^{(t^*)}} \cdots \frac{\partial h^{(t+1)}}{\partial h^{(t)}}$$

Povratno učenje: gusta predikcija - intuicija



Svaka povratna ćelija prima gradijente odozgo (trenutak t) i zdesna (budući gubitak)

U praksi možemo provesti:

- ili T unutražnih prolaza (s `retain_graph=True`)
- ili 1 unutražni prolaz kroz ukupni gubitak

Povratno učenje: gusta predikcija (sažetak)

Računanje gradijenata povratnog modela često nazivamo širenjem unatrag kroz vrijeme (BPTT):

- širenje unatrag kroz vrijeme odgovara standardnom backpropom kroz razmotrani model
- gusto predikciju možemo interpretirati kao združeno optimizirani višezadačni klasifikacijski problem
- povratnu ćeliju možemo promatrati kao sloj običnog potpuno povezanog modela

Zbog dijeljenih parametara i akumulacije gradijenata, učenje može biti **nestabilno**

- česte pojave nestajućih i eksplodirajućih gradijenata
- prikazana osnovna formulacija rijetko se koristi u praksi

Praktični savjeti

Dimenzionalnost latentnog stanja h :

- što veća - to bolje, ostali hiperparametri mogu biti važniji
- dubina modela također može biti važnija (sljedeće predavanje)

Duljina slijeda T :

- obični povratni modeli **kratkoročno** pamte ($T \approx 20$ za tekst)
- često podrezujemo ulazne slijedove na zadanu duljinu
- prag ovisi o podatcima i zadatku

Dobar početak: Adam, podrezivanje gradijenta (čak i za LSTM)

Alternative povratnim modelima **postoje**:

- SVM za jednostavnu klasifikaciju teksta:
<https://github.com/mesnilgr/nbsvm>
- **pažnja** može biti sve što nam treba
- zvuk, govor: (dilatirane) konvolucije, pažnja

Pitanja?

Literatura (**pored knjige**):

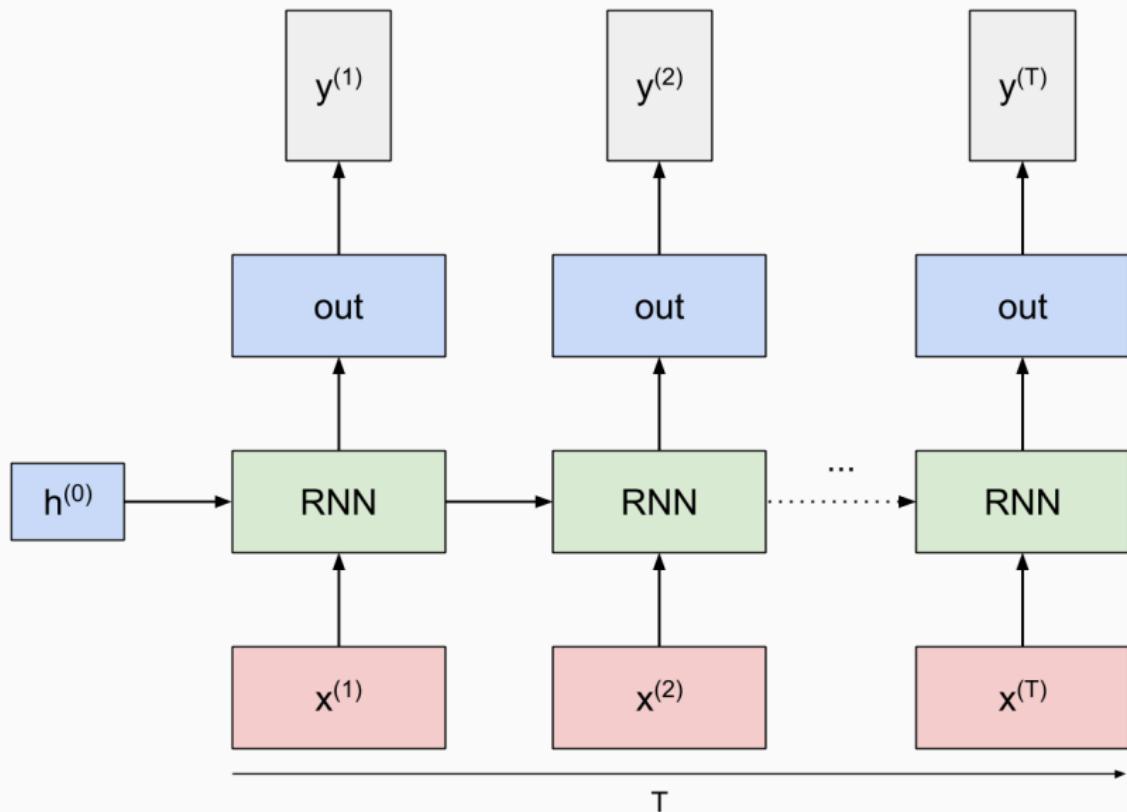
1. Peter's notes: Implementing a NN / RNN from scratch
<http://peterroelants.github.io/>
2. Andrej Karpathy: Unreasonable Effectiveness of Recurrent Neural Networks <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
3. Christopher Olah: Understanding LSTM's <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
4. CS224d: Deep Learning for Natural Language Processing
<http://cs224d.stanford.edu/syllabus.html>

Napredni povratni modeli i pažnja

Martin Tutek, Petra Bevandić, Josip Šarić, Siniša Šegvić
2023.

Ponavljanje

Obični povratni model (RNN označava povratnu čeliju)



Osnovna povratna čelija

Ažuriranje skrivenog stanja:

$$h^{(t)} = \tanh(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

Projiciranje izlaza:

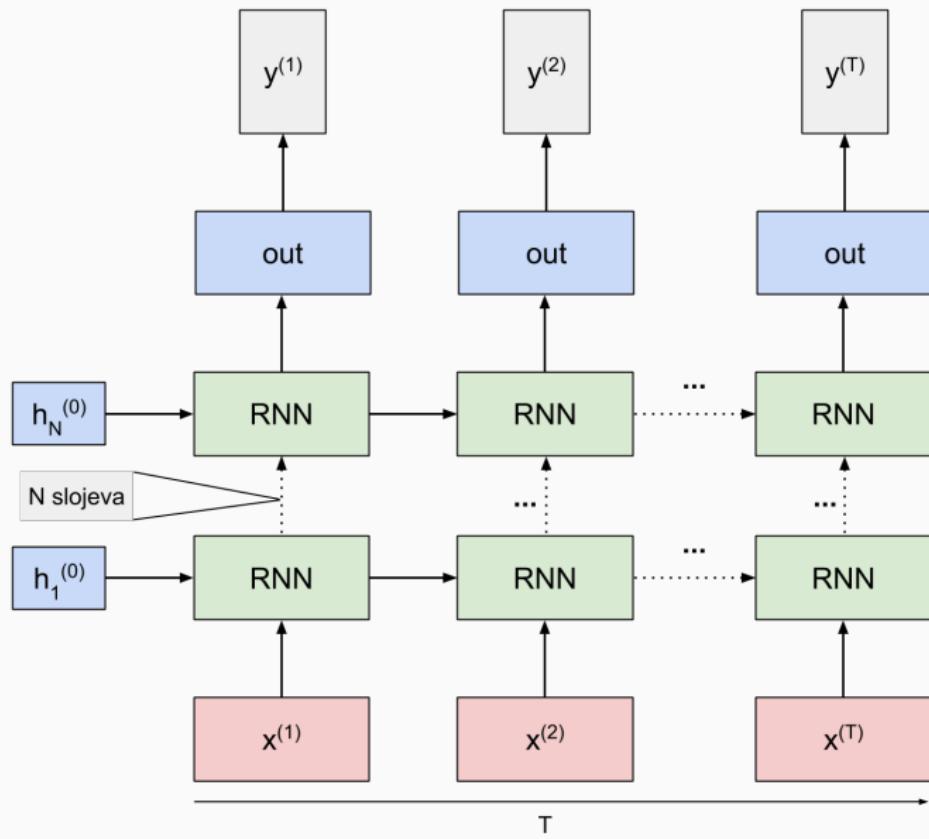
$$o^{(t)} = W_{hy}h^{(t)} + b_o \quad (1)$$

Obični povratni model obrađuje cijeli slijed *jednim* slojem:

- često nedovoljno za učenje složenih ovisnosti među elementima slijeda
- može se poboljšati dodavanjem latentnih slojeva između ulaza i predikcija

Duboki povratni modeli

Duboki (višerazinski) povratni modeli



Duboki povratni modeli

Možete li vidjeti problem na prethodnoj slici?

- $x^{(t)}$ i $h^{(t)}$ mogu imati različitu dimenzionalnost
- dimenzionalnost $W_{xh} \in \mathbb{R}^{h \times h}$ može se mijenjati preko slojeva

Sloj $n = 1$:

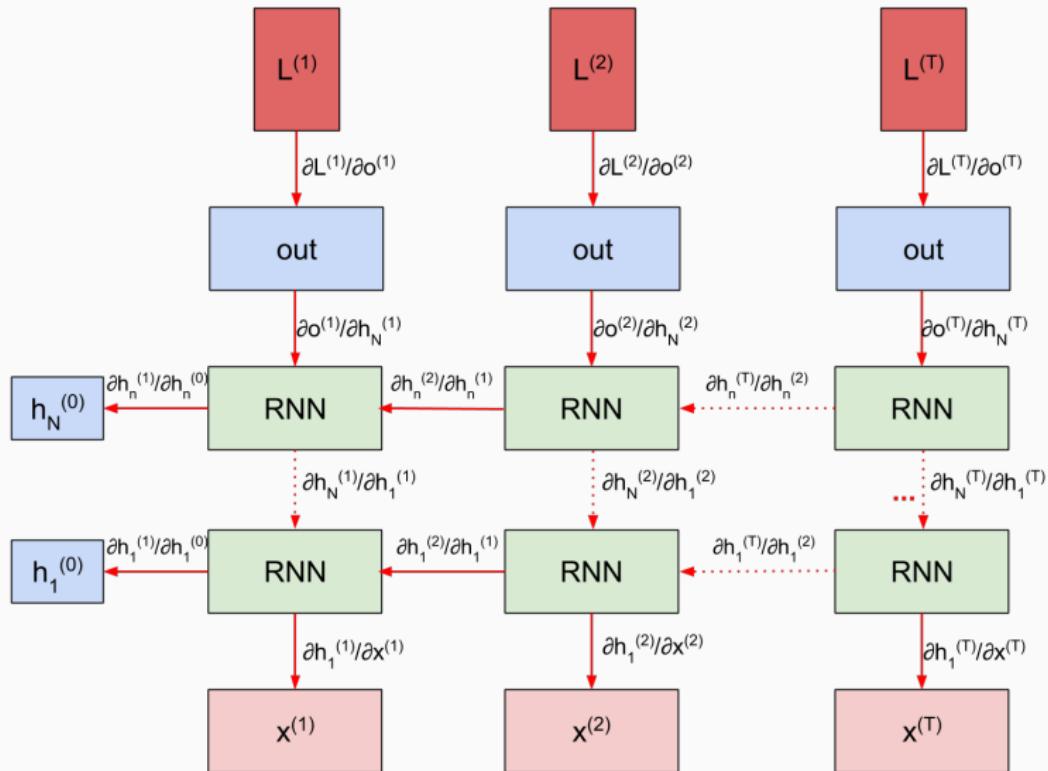
$$h_n^{(t)} = \tanh(\underbrace{W_{nhh}h_n^{(t-1)} + W_{nxh}x^{(t)} + b_{nh}}_{a_n^{(t)}}) \quad (2)$$

Sloj $n > 1$:

$$h_n^{(t)} = \tanh(\underbrace{W_{nhh}h_n^{(t-1)} + W_{nxh}h_{n-1}^{(t)} + b_{nh}}_{a_n^{(t)}}) \quad (3)$$

[!!] Ovu izmjenu ne morate raditi ručno: okviri nude povratne čelije koje automatski prilagođavaju dimenzionalnost parametara.

Duboki povratni modeli: backprop



Duboki povratni modeli: sažetak

Povratni modeli duboki su kroz slojeve (vertikalno) i vrijeme (horizontalno):

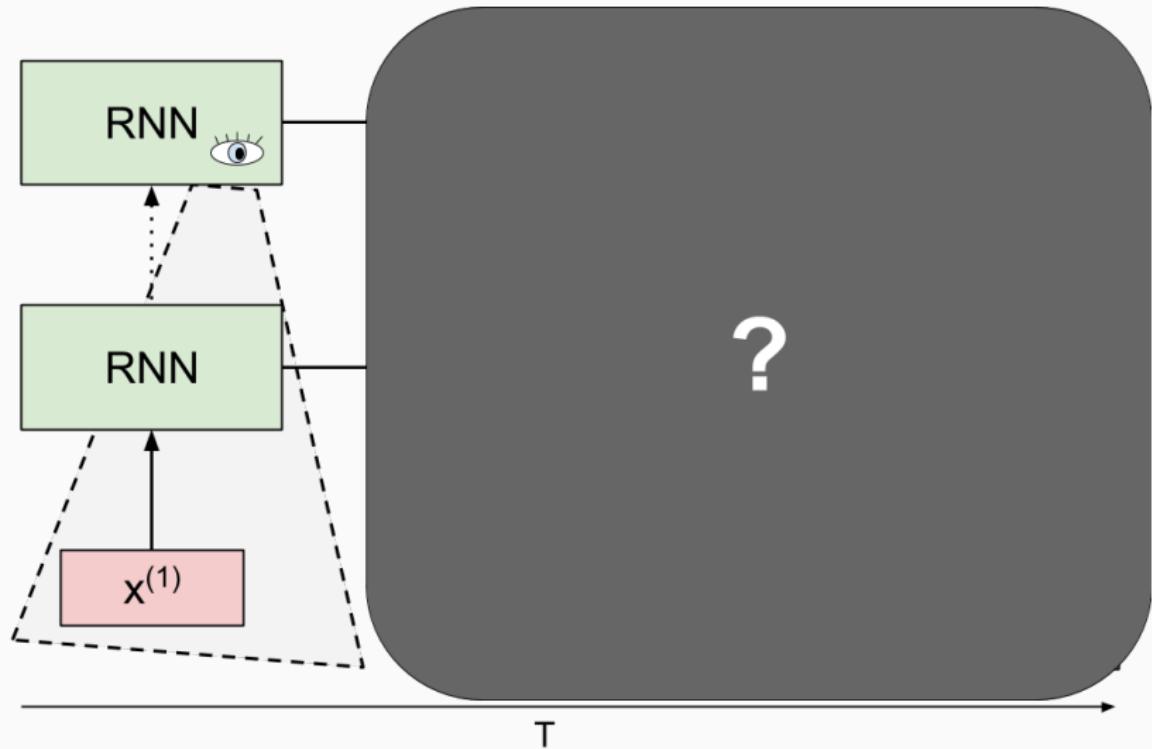
- praktične konfiguracije imaju 4 do 8 slojeva ovisno o količini podataka za učenje
- više od 8 povratnih slojeva ne dovodi do značajno bolje generalizacije (čak i za napredne ćelije)

Dimenzionalnost ulaza tipično je različita od dimenzionalnosti skrivenih slojeva:

- to najčešće ne komplicira programsku izvedbu
- slojeve konfiguriramo zadavanjem argumenata konstruktora odabrane povratne ćelije.

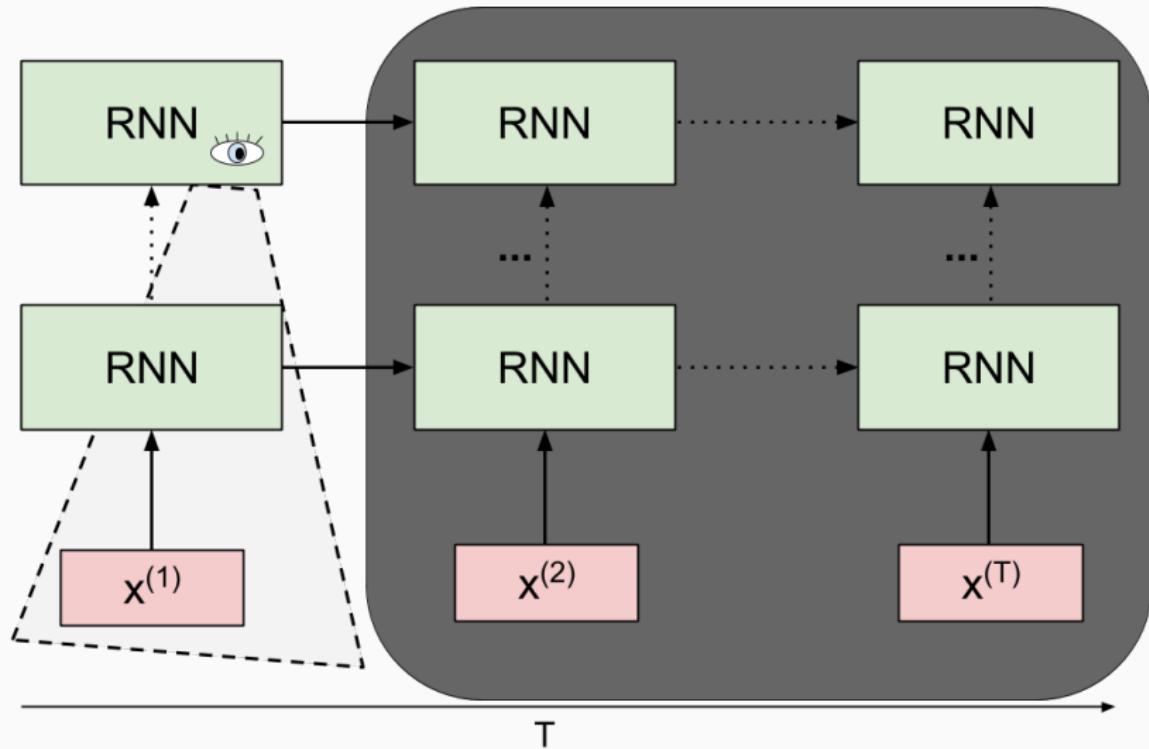
Problemi

Koliko je **receptivno polje** povratne ćelije?



Problemi

Koliko je receptivno polje povratne ćelije?



Problemi: receptivno polje

Povratna ćelija (u bilo kojem sloju) u trenutku t vidi samo $x^{(t)} \leq t$:

- predikcija u trenutku t određena je samo s do tada viđenim ulazima!
- ako zadatak ne prepostavlja skrivanje *budućeg* konteksta, htjeli bismo omogućiti modelu da vidi cijeli slijed prije donošenja odluke.

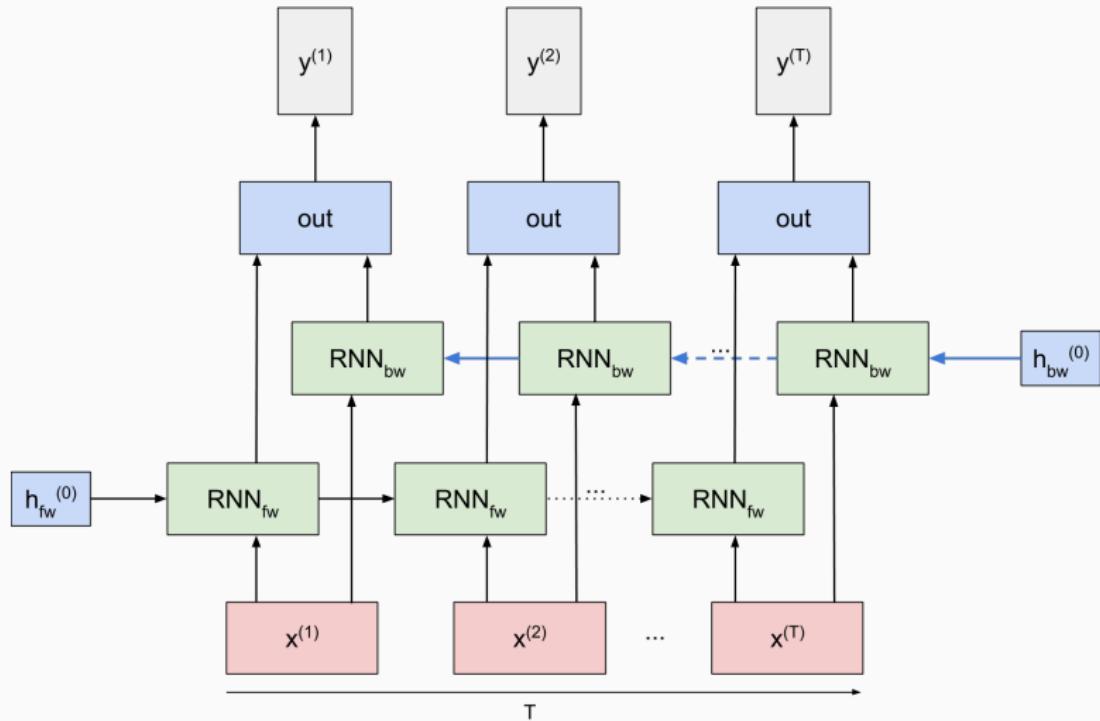
Idea: ako ciljno stanje $h^{(t)}$ ćelije koja gleda s lijeva na desno vidi $x^{(t)} \leq t$, tada će ćelija koja gleda u suprotnom smjeru vidjeti sve preostale ulaze $x^{(t)} > t$

- zajedno, te dvije ćelije vide cijeli ulazni niz

Bidirekcionalni povratni modeli

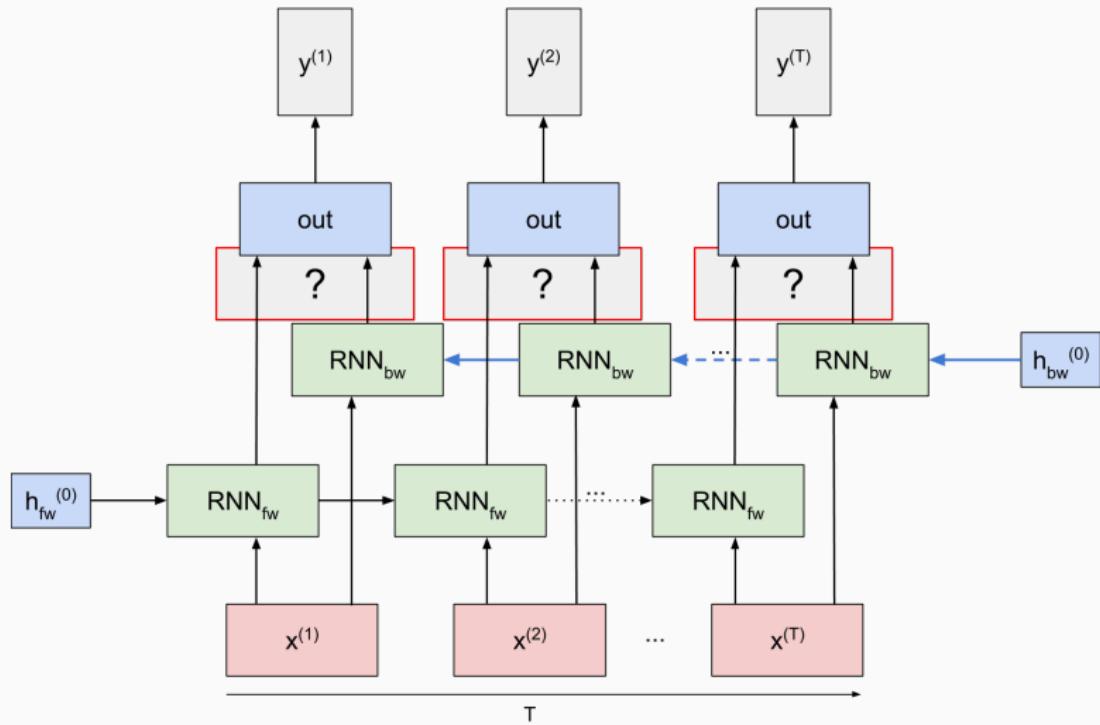
Dvosmjerni povratni modeli

Dodajemo **nezavisan** povratni model (\overleftarrow{RNN}) koji gleda u **suprotnom smjeru** s obzirom na originalni model (\overrightarrow{RNN})



Dvosmjerni povratni modeli: agregiranje stanja

Kako agregirati izlaze povratnih ćelija prije predikcije?



Dvosmjerni povratni modeli: detalji

Dvosmjerni povratni model (BiRNN) sastoji se od dvija odvojena povratna modela koji funkciraju u suprotnim smjerovima:

- \overrightarrow{RNN} čita s lijeva na desno
- \overleftarrow{RNN} čita s desna na lijevo

Kako agregirati skrivena stanja?

1. konkateniranjem:

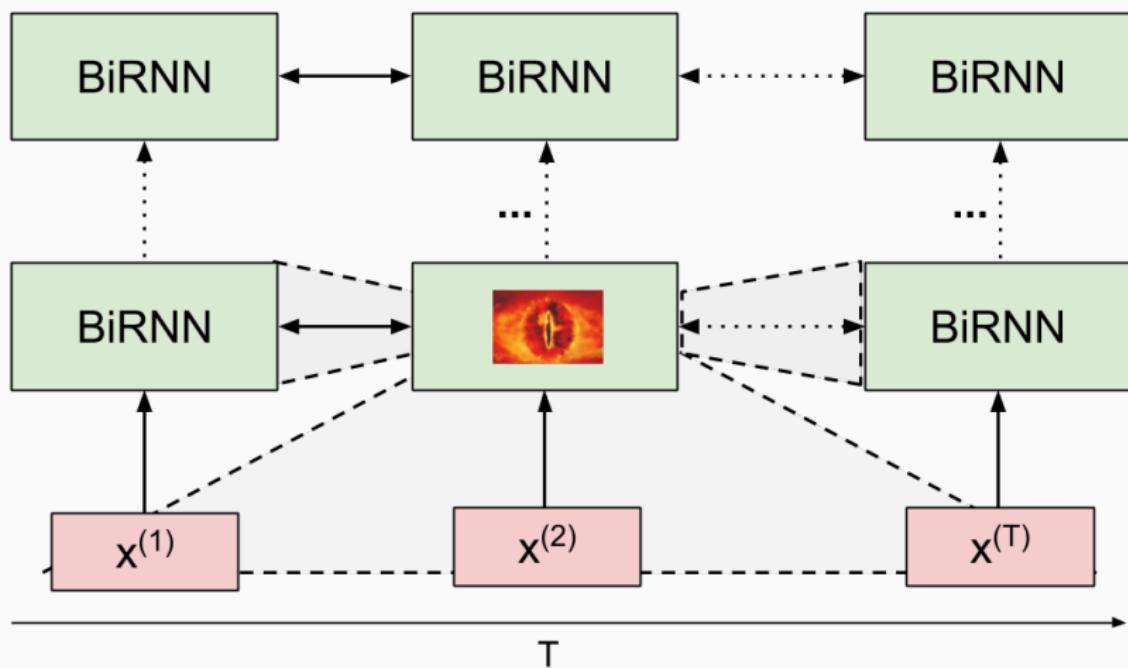
$$h^{(t)} = [\overrightarrow{h}^{(t)}, \overleftarrow{h}^{(t)}]$$

- ovo udvostručuje dimenzionalnost sljedećeg sloja
- podrazumijevani izbor u postojećim okvirima

2. usrednjavanjem

3. proizvoljnom (parametriziranom) funkcijom

Dvosmjerni povratni modeli: receptivno polje



Dvosmjerni povratni modeli: sažetak

Dvosmjerni modeli sastoje se od dva povratna modela koji napreduju u suprotnim smjerovima:

- konkateniranje stanja sljedećem sloju omogućava pregled svih ulaza

Konkatenacija povećava dimenzionalnost sljedećih slojeva:

- podrazumijevano ponašanje
- alternative: usrednjavanje, sažmanje + projekcija, ...

Važno je razmotriti **dozvoljava** li zadatak pristup cijelokupnom ulazu (prognoziranje vs gusta predikcija).

Učenje povratnih modela

Nestajući i eksplodirajući gradijenti

Gradijenti povratnih modela podložni numeričkoj nestabilnosti:

- uzrokovani dijeljenjem parametara u uzastopnim operacijama
- preciznije: uzastopno množenje s W_{hh}

Podsjetnik:

$$h^{(t)} = \tanh(W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h)$$

Prvo razmatramo *skalarni* kontekst:

$$h^{(t)} = \tanh(\underbrace{w_{hh}h^{(t-1)} + w_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

- Vrijedi: $w_{hh}, w_{xh}, b_h, h, x \in \mathbb{R}$

Gradijenti u skalarnom slučaju

$$h^{(t)} = \tanh(\underbrace{w_{hh}h^{(t-1)} + w_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

Razmatramo gradijent između susjednih stanja:

$$\begin{aligned}\frac{\partial h^{(t)}}{\partial h^{(t-1)}} &= \frac{\partial h^{(t)}}{\partial a^{(t)}} \frac{\partial a^{(t)}}{\partial h^{(t-1)}} \\ &= \frac{\partial \tanh(a^{(t)})}{\partial a^{(t)}} w_{hh} \\ &= (1 - \tanh^2(a^{(t)})) w_{hh}\end{aligned}$$

$\tanh = th$
 $\frac{dth(x)}{dx} = 1 - th^2(x)$

Derivacija hiperbolnog tangensa ograničena na jedinični interval:

$$\tanh(x) \in (-1, 1)$$

$$\frac{\partial \tanh(x)}{x} = (1 - \tanh^2(x)) \in (0, 1)$$

Gradijenti u skalarnom slučaju (2)

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = (1 - th^2(a^{(t)}))w_{hh}$$

Primijenimo supstituciju:

$$\gamma_t = \partial \tanh(x) / \partial x \Big|_{a^{(t)}} < 1$$

Slično: $\gamma_{\sigma t} = \partial \sigma(x) / \partial x \Big|_{a^{(t)}} < 1/4$

$$\begin{aligned}\frac{\partial h^{(t)}}{\partial h^{(t-1)}} &= \gamma_t w_{hh} \\ \frac{\partial h^{(T)}}{\partial h^{(t_0)}} &= \prod_{t_0}^T \gamma_t w_{hh} \\ \frac{\partial h^{(T)}}{\partial h^{(t_0)}} &= (\bar{\gamma} w_{hh})^{T-t_0}\end{aligned}\quad \begin{array}{l} \nearrow t \rightarrow T \\ \searrow \bar{\gamma}, w_{hh} \text{ ne ovisi o } t \end{array}$$

Gradijenti u skalarnom slučaju (3)

$$\frac{\partial h^{(T)}}{\partial h^{(t_0)}} = (\bar{\gamma} w_{hh})^{T-t_0}$$

Kod dugih slijedova imamo: $T - t_0 \gg 0$

- numerička stabilnost gradijenta ovisi o $\bar{\gamma} w_{hh}$:

$$(\bar{\gamma} w_{hh})^{T-t_0} \rightarrow \begin{cases} \infty & \text{if } \bar{\gamma} w_{hh} > 1 \text{ (eksplodira)} \\ 0 & \text{if } \bar{\gamma} w_{hh} < 1 \text{ (nestaje)} \\ 1 & \text{if } \bar{\gamma} w_{hh} \approx 1 \text{ (stabilan)} \end{cases}$$

Ako pretpostavimo $\bar{\gamma} = 1$, tada gore navedeni uvjet primjenjujemo na **parametar w_{hh}** .

Nastavljamo s analizom u kontekstu vektorskog skrivenog stanja.

Gradijenti u vektorskom slučaju: spektralna norma

Razmatramo svojstva spektralne norme kvadratne matrice A:

- norma produkta manja je ili jednaka produktu normi (vrijedi za sve matrične norme):

$$\|AB\| \leq \|A\| \|B\|$$

- spektralna norma odgovara najvećoj singularnoj vrijednosti
 - ili, ekvivalentno, korijenu najveće svojstvene vrijednosti $A^T A$
- spektralna norma inducirana je L2-normom:

$$\|Ax\| \leq \|A\| \|x\|$$

Gradijenti u vektorskom slučaju: jedan korak

Ažuriranje skrivenog stanja (podsjetnik):

$$h^{(t)} = \tanh(\underbrace{W_{hh}h^{(t-1)} + W_{xh}x^{(t)} + b_h}_{a^{(t)}})$$

Gledamo gradijent između dva uzastopna stanja:

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \frac{\partial h^{(t)}}{\partial a^{(t)}} W_{hh}$$

Postavljamo gornju ogragu gradijenta:

$$\left\| \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \right\| \leq \left\| \frac{\partial h^{(t)}}{\partial a^{(t)}} \right\| \|W_{hh}\| \leq \gamma_{\max} \lambda_1$$

- λ_1 ... najveća singularna vrijednost W_{hh}
- $\gamma_{\max} = \max\left(\frac{\partial \tanh(a^{(t)})}{\partial a^{(t)}}\right)$... gornja ograda gradijenta aktivacije

Gradijenti u vektorskom slučaju: svi koraci

Razmatamo prethodnu jednadžbu kroz vrijeme:

$$\frac{\partial h^{(T)}}{\partial h^{(t_0)}} \leq (\gamma_{\max} \lambda_1)^{T-t_0}$$

Za dugačke slijedove ($T - t_0 \gg 0$) numerička stabilnost radijenta ovisi o $\gamma_{\max} \lambda_1$:

$$(\gamma_{\max} \lambda_1)^{T-t_0} \rightarrow \begin{cases} \infty & \text{if } \gamma_{\max} \lambda_1 > 1 \text{ (eksplodira)} \\ 0 & \text{if } \gamma_{\max} \lambda_1 < 1 \text{ (nestaje)} \\ 1 & \text{if } \gamma_{\max} \lambda_1 \approx 1 \text{ (stabilan)} \end{cases}$$

- Matrica W_{hh} mora zadovoljiti **stroge uvjete** ako želimo stabilnu optimizaciju
- za detaljniju analizu preporučamo pogledati [pascanu13icml]:
Razvan Pascanu, Tomás Mikolov, Yoshua Bengio: On the difficulty of training recurrent neural networks. ICML 2013.

Povratni modeli konzistentno podbacuju na dugim slijedovima:

- problem nastaje zbog numeričke nestabilnosti gradijenata uslijed uzastopnog množenja s W_{hh} :
[bengio94tnn] Y Bengio, PY Simard, P Frasconi: Learning long-term dependencies with gradient descent is difficult, IEEE TNN 1994.

Simptome možemo ublažiti:

- osiguravanjem umjerenih singularnih vrijednosti povratne veze
M Arjovsky, A Shah, Y Bengio: Unitary Evolution Recurrent Neural Networks. ICML 2016
- izostavljanjem matričnog množenja iz povratne veze.

Rješenje: razdvojiti zadatke povratne veze

- W_{hh} i W_{xh} sprežu filtriranje informacija, pamćenje ulaza i projekciju novih elemenata u skriveno stanje
- skriveno stanje h spreže projekciju izlaza i pamćenje informacija za buduće izlaze.

Povratna čelija s dugoročnom memorijom (LSTM: Long short-term memory)

LSTM: notacija

Sada skriveno stanje $h^{(t)}$ koristimo samo za računanje izlaza.

Uvodimo stanje ćelije $c^{(t)}$ koji samo pamti viđenu informaciju.

Uvodimo doprinos stanju ćelije $\hat{c}^{(t)}$ s obzirom na ulaz u trenutku t .

Uvodimo logičke vektore $f^{(t)}$ i $i^{(t)}$:

- $f^{(t)}$ nazivamo propusnicom (vratima) zaboravljanja
- $i^{(t)}$ nazivamo propusnicom (vratima) ulaza

Izbacujemo matrično množenje iz povratne jednadžbe stanja ćelije:

$$c^{(t)} = c^{(t-1)} + \hat{c}^{(t)}$$

$$\frac{\partial c^{(t)}}{\partial c^{(t-1)}} = \mathbb{I}$$

[hochreiter96nips] LSTM can solve hard long time lag problems

LSTM: jednadžbe

$$c^{(t)} = c^{(t-1)} + \hat{c}^{(t)}$$

LSTM čelija **zaboravlja** dio informacije iz prethodnog stanja:

$$f^{(t)} = \sigma(W_{fhh} h^{(t-1)} + W_{fxh} x^{(t)} + b_{fh}) = \sigma(a_f^{(t)})$$

- svaka vrata imaju svoj vlastiti skup parametara W_{hh}, W_{xh}, b_h .

LSTM čelija propušta **samo podskup** ulaza:

$$i^{(t)} = \sigma(W_{ihh} h^{(t-1)} + W_{ixh} x^{(t)} + b_{ih}) = \sigma(a_i^{(t)})$$

- povratni put stanja čelije upotpunjavamo vratima f i i :

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

Hadamardov produkt (\odot): tenzorsko množenje po elementima

$$a \odot b = \begin{pmatrix} a_0 b_0 \\ \dots \\ a_i b_i \end{pmatrix}$$

Svrha vrata: filtriranje informacije ($\sigma : \mathbb{R} \rightarrow (0, 1)$).

Sigmoidna funkcija može se probabilistički interpretirati kao **dio informacije koji želimo zadržati**.

Ograničavanje $f^{(t)}$ i $i^{(t)}$ na jedinični interval $(0, 1)$ *eliminira eksplodirajuće gradijente*

- u teoriji, kod domena sigmoide je otvorena ($\sigma(x) < 1 \quad \forall x$)
- u praksi imamo podljev izraza $\exp(-x)$ zbog konačne preciznosti

LSTM: detalji

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

Izraz za računanje doprinosa stanju ćelije \hat{c} :

$$\hat{c}^{(t)} = \tanh(W_{chh}h^{(t-1)} + W_{cxh}x^{(t)} + b_{ch}) = \tanh(a_c^{(t)})$$

- određujemo ga kao afinu transformaciju **skrivenog stanja** i ulaza.

Naša notacija malo je **različita** nego u knjizi

- u knjizi: $s^{(t)} := c^{(t)}$; $g^{(t)} := i^{(t)}$; $q^{(t)} := o^{(t)}$
- umjesto estetske supstitucije $\hat{c}^{(t)}$, knjiga ima razmotani izraz:

$$s^{(t)} = f^{(t)}s^{(t-1)} + g^{(t)} \left(\tanh(Wh^{(t-1)} + Ux^{(t)} + b_s) \right)$$

LSTM: skriveno stanje

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

Skriveno stanje računamo kao funkciju stanja ćelije:

$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)})$$

Logički vektor $o^{(t)}$ nazivamo *izlaznim vratima*:

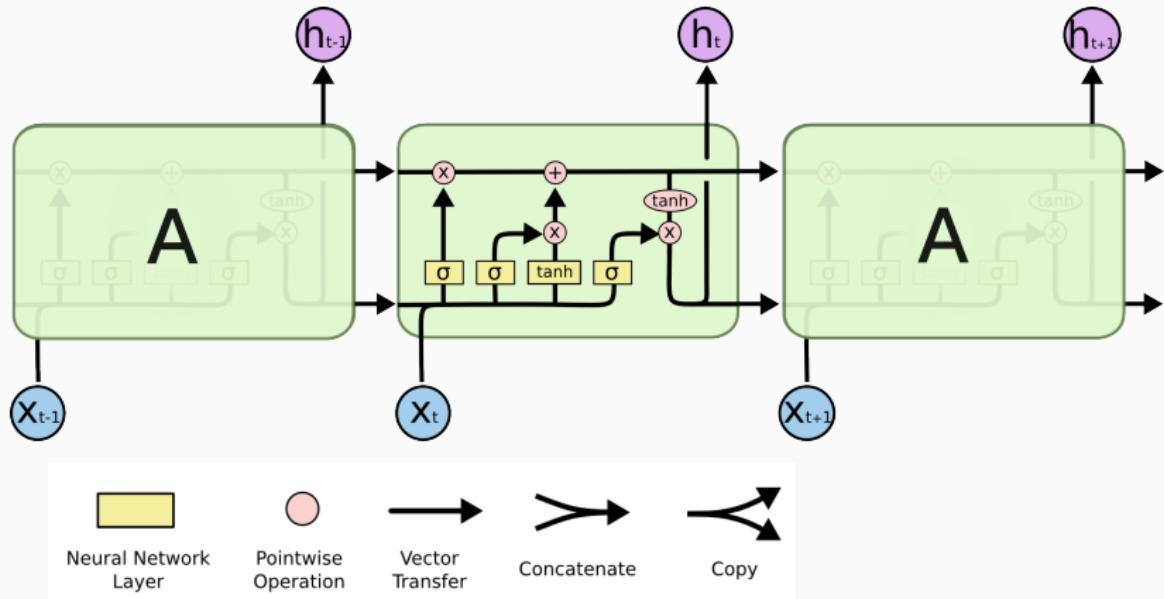
$$o^{(t)} = \sigma(W_{ohh}h^{(t-1)} + W_{oxh}x^{(t)} + b_{oh}) = \sigma(a_o^{(t)})$$

Sažetak:

- Razdvojili smo stanje ćelije ("memoriju") $c^{(t)}$ od skrivenog stanja $h^{(t)}$ iz kojeg se računa izlaz
- arhitektura ne dopušta lako mijenjanje stanja ćelije
- imamo **4×** više parametara

LSTM: vizualizacija

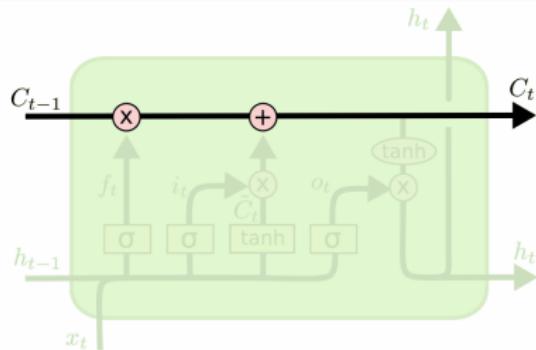
Prikazujemo nekoliko slika s [bloga] Christophera Olaha



Pitanje: koji je redoslijed vrata na skici?

LSTM: vizualizacija stanja ćelije

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

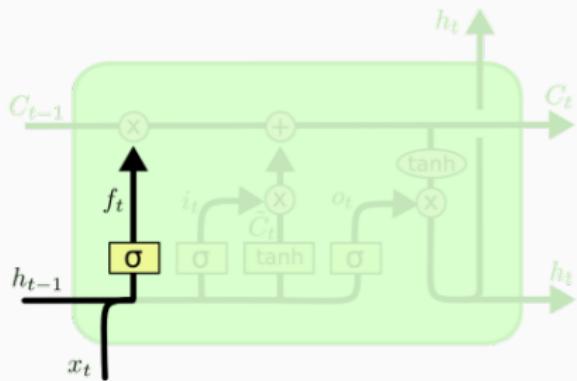


Stanje ćelije mijenjamo množenjem po elementima i zbrajanjem - informacijski tok je jednostavan

- Stanje ćelije je **teško** izmijeniti: svaku promjenu moraju *podržati* dvoja vrata.

LSTM: vizualizacija vrata zaboravljanja

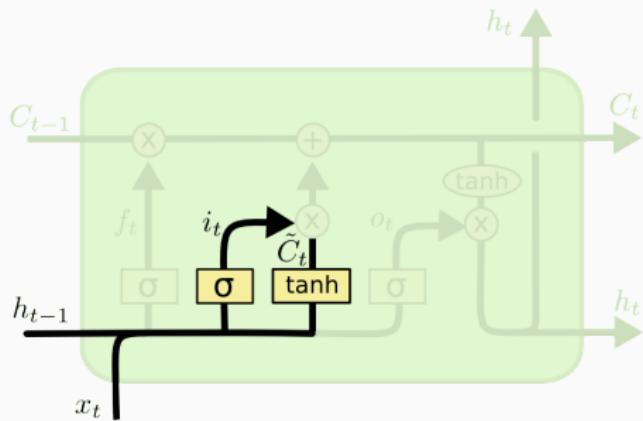
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$



$$f^{(t)} = \sigma(W_{fhh}h^{(t-1)} + W_{fxh}x^{(t)} + b_{fh})$$

LSTM: vizualizacija ulaznih vrata

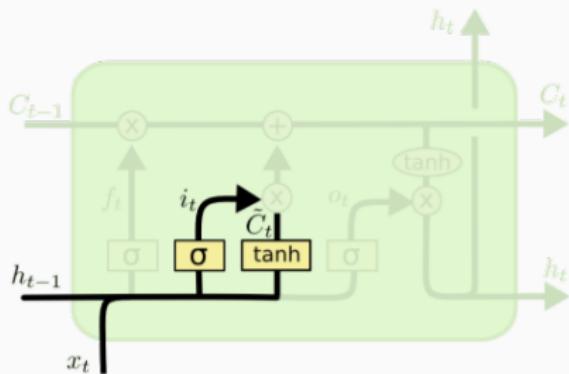
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$



$$i^{(t)} = \sigma(W_{ihh}h^{(t-1)} + W_{ixh}x^{(t)} + b_{ih})$$

$$\hat{c}^{(t)} = \tanh(W_{chh}h^{(t-1)} + W_{cxh}x^{(t)} + b_{ch})$$

LSTM: vizualizacija ulaznih vrata (2)



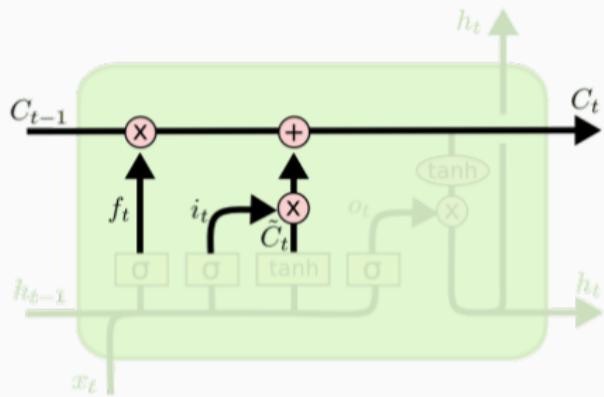
$$\hat{c}^{(t)} = \tanh(W_{chh}h^{(t-1)} + W_{cxh}x^{(t)} + b_{ch})$$

Prema literaturi, doprinos stanju $\hat{c}^{(t)}$ može biti aktiviran sigmoidom ili hiperbolnim tangensom

- PyTorch i Tensorflow koriste \tanh

LSTM: vizualizacija ažuriranja stanja

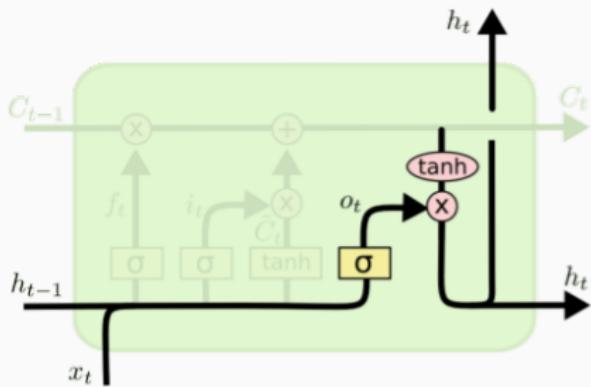
$$C^{(t)} = f^{(t)} \odot C^{(t-1)} + i^{(t)} \odot \hat{C}^{(t)}$$



LSTM: vizualizacija izlaznih vrata

$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)})$$

$$o^{(t)} = \sigma(W_{ohh}h^{(t-1)} + W_{oxh}x^{(t)} + b_{oh})$$



LSTM: sažetak

Obične povratne modele nije lako naučiti

- često susrećemo eksplodirajuće i nestajuće gradijente zbog uzastopnog množenja s W_{hh}
- skriveno stanje mora pamtiti informacije i voziti izlaz
- zbog toga se ovi modeli loše ponašaju na dugim slijedovima.

Zbog toga smo uveli ćeliju s dugoročnim pamćenjem (LSTM)

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)})$$

- izostavili smo matrično množenje iz unaprijednog i povratnog puta
- uveli smo troja vrata za filtriranje informacija
- raspregnuta **odgovornost** olakšava pritisak na stanje ćelije.

LSTM: backprop

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i \odot \hat{c}^t$$

Razmatramo gradijent za povratnu vezu:

$$\frac{\partial c^{(t)}}{\partial c^{(t-1)}} = f^{(t)} = \sigma(a_f^{(t)}) \in (0, 1)$$

Pravilo ulančavanja daje:

$$\frac{\partial c^{(T)}}{\partial c^{(t_0)}} = \prod_{t=t_0}^T f^{(t)} \leq 1$$

Čini se da ova jednadžba ne dozvoljava **eksplodirajuće** gradijente!

- *Je li uistinu tako?*
- LSTM čelija ima dvojno skriveno stanje ($c^{(t)}, h^{(t)}$)

LSTM: backprop (2)

$$h^{(t)} = o^{(t)} \odot \tanh(c^{(t)})$$

Pogledajmo izlazna vrata:

$$o^{(t)} = \sigma(a_o^{(t)}) = \sigma(W_{ohh}h^{(t-1)} + W_{oxh}x^{(t)} + b_{oh})$$

Primjećujemo sličan oblik kao u običnoj povratnoj ćeliji:

$$h^{(t)} = \sigma(W_{ohh} h^{(t-1)} + W_{oxh}x^{(t)} + b_{oh}) \odot \tanh(c^{(t)})$$

$$\frac{\partial h^{(t)}}{\partial h^{(t-1)}} = \frac{\partial h^{(t)}}{\partial a_o^{(t)}} \frac{\partial a_o^{(t)}}{\partial h^{(t-1)}} = \frac{\partial h^{(t)}}{\partial a_o^{(t)}} W_{ohh} = \dots$$

Zbog toga, eksplodirajući gradijent je **ipak moguć** pri unutražnom prolazu kroz $h^{(t)}$

- međutim, to se rijetko događa u praksi

LSTM varijante: čelija sa špijunkom

Uključiti stanje čelije $c^{(t-1)}$ u jednadžbu za računanje vrata:

$$f^{(t)} = \sigma(\underbrace{W_{fch}c^{(t-1)} + W_{fhh}h^{(t-1)} + W_{fxh}x^{(t)} + b_{fh}}_{a_f^{*(t)}})$$

Ova ideja može se primijeniti na sva vrata.

- **Prednost:** dodatna informacija može **pomoći** [gers00ijcnn]
- **Nedostatak:** veći broj parametara
- **Nedostatak:** još jedan put kroz koji može dovesti do eksplodirajućeg gradijenta.

LSTM varijante: spojena vrata (eng. fused gates)

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \hat{c}^{(t)}$$

Ideja: ako smo neku informaciju zaboravili, trebamo je zamijeniti

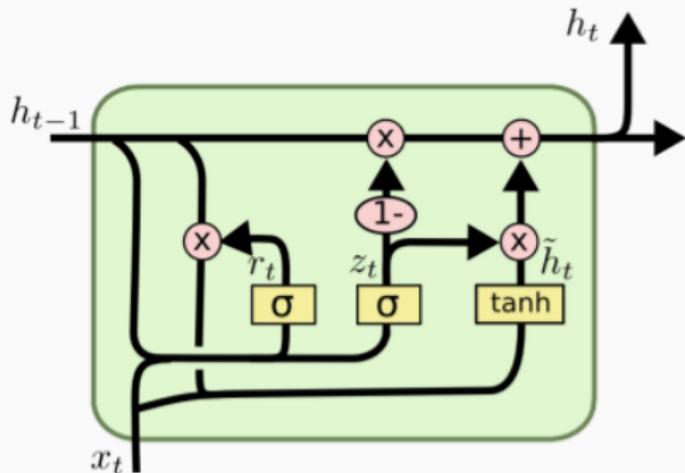
$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + (1 - f^{(t)}) \odot \hat{c}^{(t)}$$

- **Prednost:** 25% manje parametara
- **Nedostatak:** radi lošije od standardnog LSTM-a (više parametara pomaže)
- **Nedostatak:** smanjuje izražajnost modela (nemogućnost akumulacije)

LSTM varijante: propusna povratna čelija

Propusna čelija (eng. gated recurrent unit): pojednostavljeni LSTM

- izvrsno radi iako ne izdvaja stanje čelije iz latentnog stanja
- ovo sugerira da su naše intucije u vezi LSTM-ova nepotpune.



LSTM varijante: propusna povratna čelija (2)

GRU čelije imaju dvoja vrata: $r^{(t)}$ and $u^{(t)}$:

- $u^{(t)}$ nazivamo vratima *ažuriranja* (eng. update gate)

$$u^{(t)} = \sigma \left(W_{uh} h^{(t-1)} + W_{uxh} x^{(t)} + b_{uh} \right) \quad (4)$$

- $r^{(t)}$ nazivamo vratima *resetiranja* (eng. reset gate)

$$r^{(t)} = \sigma \left(W_{rh} h^{(t-1)} + W_{rxh} x^{(t)} + b_{rh} \right) \quad (5)$$

Važan međurezultat: privremeno stanje $\hat{h}^{(t)}$

$$\hat{h}^{(t)} = \sigma \left(W_{hh} (r^{(t)} \odot h^{(t-1)}) + W_{xh} x^{(t)} + b_h \right) \quad (6)$$

Povratno stanje $h^{(t)}$ ponovo ima višestruke odgovornosti:

$$h^{(t)} = u^{(t)} h^{(t-1)} + (1 - u^{(t)}) \hat{h}^{(t)} \quad (7)$$

LSTM varijante: sažetak

Eksplodirajući i nestajući gradijenti mogu se pojaviti i u LSTM-ovima

- međutim, oni se javljaju znatno rjeđe u praksi

Uspjeh LSTM-ova doveo je do razvoja više varijanti.

1. LSTM sa špijunkom:

- stanje čelije koristi se u jednadžbi ažuriranja
- ima smisla jer LSTM-ovi ionako ne uspijevaju u potpunosti izbjegći numeričke probleme s gradijentom

2. LSTM sa spojenim vratima

- spajanje vrata zaboravljanja s vratima ulaza dovodi do veće učinkovitosti i manjeg broja parametara

3. Propusna povratna čelija (GRU)

- spojena vrata i izmjenjena semantika stanja
- slična generalizacijska moć kao i LSTM-ovi unatoč spregnutom stanju

Analiza: slijed-u-slijed

Strojno prevođenje

Želimo naučiti generirati prijevod danog ulaznog slijeda:

- model cilja izlazne slijedove **unknown length**.
- prediktiramo kategoričku distribuciju preko izlaznog rječnika u svakom izlaznom elementu.

Skupovi podataka: WMT, IWSLT (povremeno se ažuriraju):

- <https://www.statmt.org/wmt15/translation-task.html>
- <https://sites.google.com/site/iwsltevaluation2015/mt-track>

Primjer ulaza i izlaza za jedan primjer WMT-14 en-de skupa:

Parliament Does Not Support Amendment Freeing Tymoshenko
Keine befreiende Novelle für Tymoshenko durch das Parlament

Strojno prevođenje: pretpostavke

Ciljne varijable:

- slijed riječi ciljanog jezika: $\{keine, befreiende, Novelle, \dots\}$
- ciljne varijable pretvaramo u indekse: $\{0, \dots, V_{out}\}$
- biramo veličinu ciljnog vokabulara.

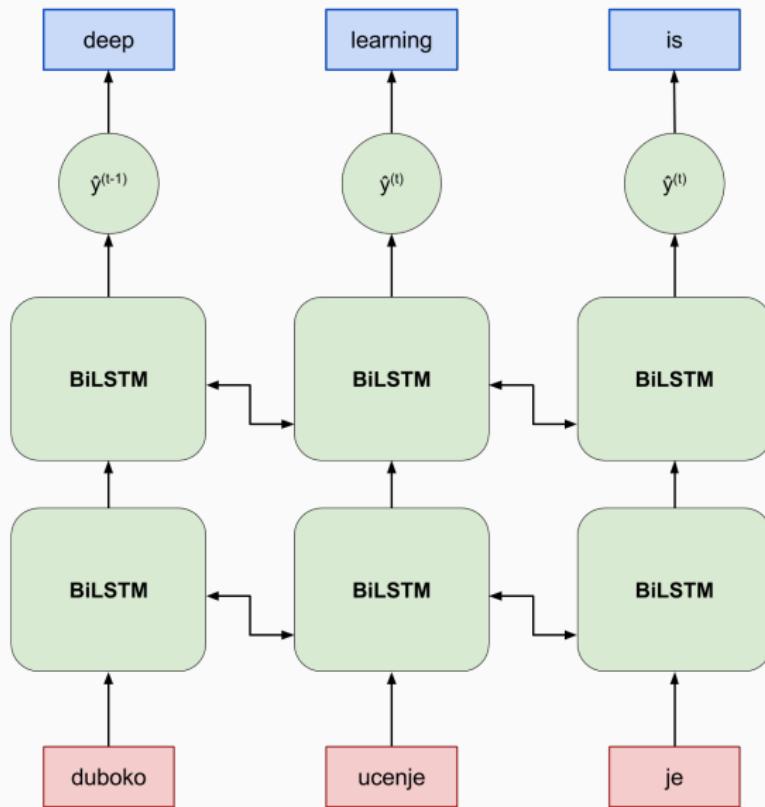
Ulazne varijable:

- biramo veličinu ulaznog vokabulara
- ulazne riječi pretvaramo u indekse koji odgovaraju indeksima u matrici ugrađivanja

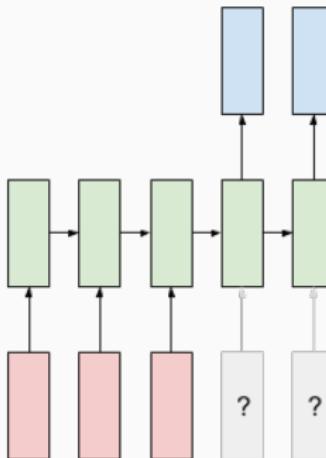
Ovaj pristup je sličan gustoj predikciji (npr. predikciji vrste riječi), ali:

1. možemo započeti predikciju tek nakon što smo vidjeli cijeli ulaz
2. ne znamo broj izlaznih simbola
3. nije jasno o kojim ulazima ovisi tekući izlaz

Strojno prevodenje: naivno rješenje koje ne funkcioniра



Slijed u slijed: prepostavke



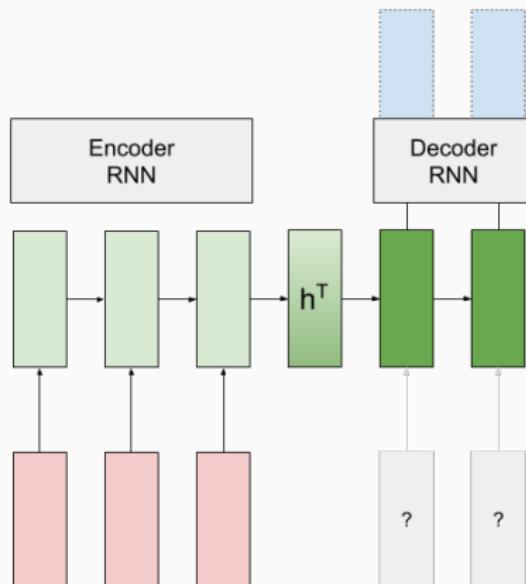
Započinjemo formalizacijom problema slijed-u-slijed (gore).

Nakon toga, prikazat ćemo neka konkretna rješenja.

Slijed u slijed: formalizacija

Predviđamo rješenje s dva modula:

1. **koder** ("čitač"): čita ulazni slijed i gradi skrivenu reprezentaciju izrečenog
2. **dekoder** ("pisač") generira prijevod na temelju skrivene reprezentacije



Slijed u slijed: formalizacija (2)

Posljednje stanje kodera određuje prvo stanje dekodera:

$$h_{dec}^{(0)} = f(h_{enc}^{(T)})$$

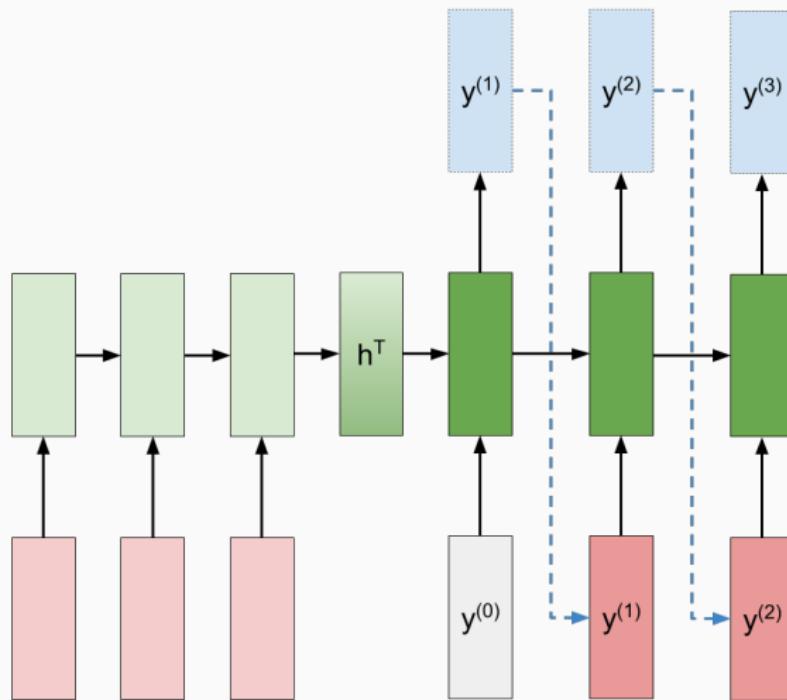
- u praksi često imamo: $h_{dec}^{(0)} = h_{enc}^{(T)}$
- f može biti bilo kakva parametrizirana glatka funkcija
- pitanje: kada bi to bilo **potrebno**?

Koder **ne prima** gubitak izravno

- gradijenti prvo moraju proći kroz cijeli **dekoder**

Problem: što ulazi u povratne ćelije dekodera?

Slijed u slijed: ulazi dekodera



Povratne ćelije dekodera primaju izlaze iz prethodnog koraka obrade.

Slijed u slijed: generiranje izlaza

Ulaz dekodera u trenutcima $t > 0$ sadrži najizgledniji izlaz iz prethodnog koraka:

- što se zbiva u koraku $t = 0$?
- na ulaz dekodera u $t = 0$ dovodimo poseban simbol koji označava početak slijeda (**<sos>**)

Kako znamo da je izlaz upotpunjeno?

- model označava kraj prijevoda predkcijom posebnog simbola koji označava kraj slijeda (**<eos>**)
- simbol **<eos>** dodajemo na **kraj** svakog ciljnog niza
- generiranje prijevoda zaustavljamo kada dobijemo simbol **<eos>** ili kada duljina slijeda premaši maksimalnu duljinu

Slijed u slijed: generiranje izlaza (2)

Prevođenje slijeda nije lako naučiti, a posebno u ranim fazama optimizacije.

Zato često koristimo **forsiranje učitelja** (engl. teacher forcing) gdje ulaze dekodera postavljamo stohastički na:

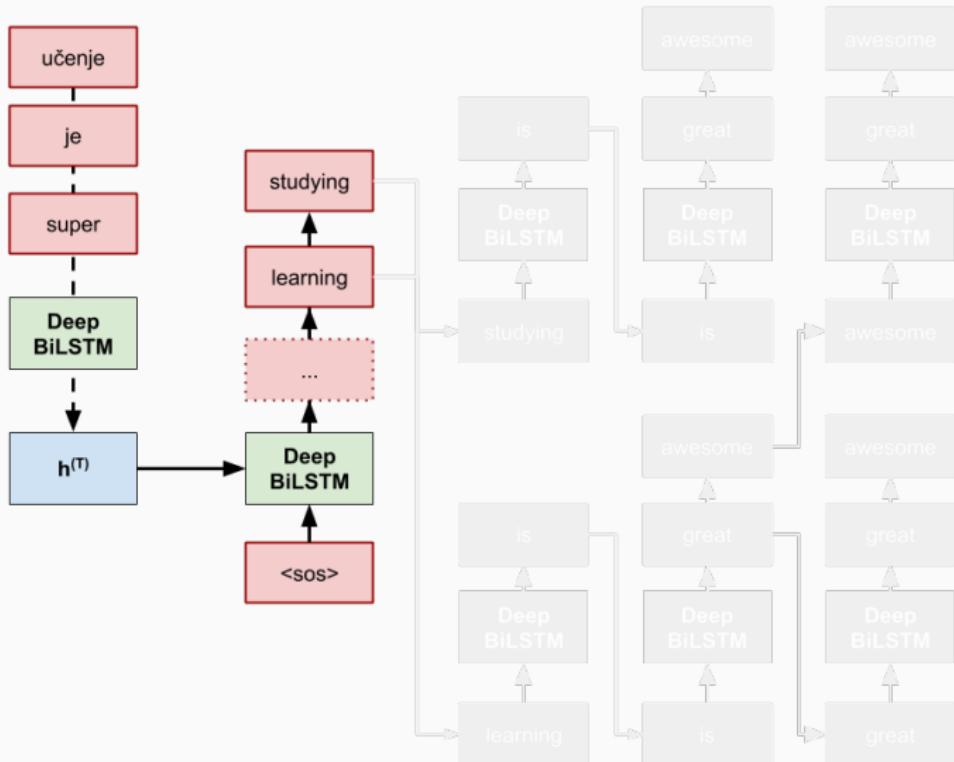
- **točne simbole** prethodnog koraka u p primjeraka za učenje
- **predikcije** prethodnog koraka u $1 - p$ primjera za učenje
- $p \in [0, 1]$ je hiper-parametar koji započinje s $p = 1$ i može se smanjiti kad učenje uznapreduje.

Slijed u slijed: zaključivanje

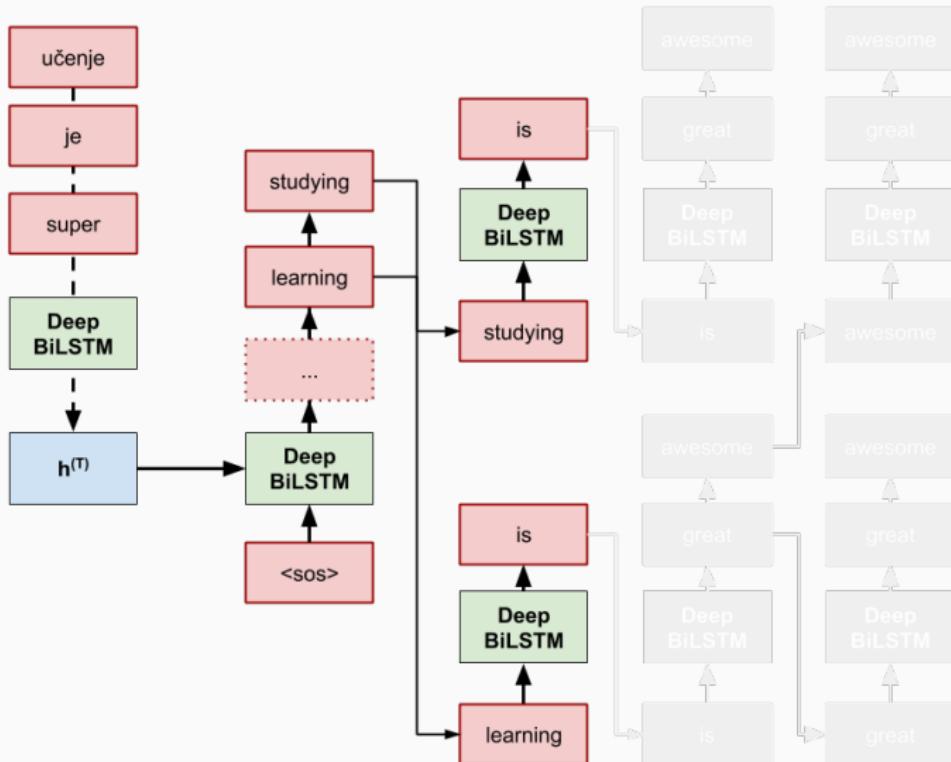
Pristupi za generiranje izlaza:

1. odabratи najvjerojatniju riječ u svakom koraku:
 - pohlepni pristup, može biti suboptimalan
 - moguće je da najbolji prijevod ne sadrži najvjerojatniju riječ u svakom koraku
 - nije dobro za uzorkovanje jer proizvodi **determinističke sljedove**
2. uzorkovanje s vjerojatnosnim težinama (roulette wheel selection):
 - uvodi slučajnost u postupak prevodenja i ohrabruje **raznolikost** izlaza
 - nije jasno je li nam prihvatljivo da model može izabrati lošu riječ s vjerojatnošću koja je **veća od nule**.
3. fokusirano uzorkovanje pretraživanjem zrakom:
 - razmatramo k najboljih prijevoda u svakom koraku
 - hiperparametar k označava širinu zrake

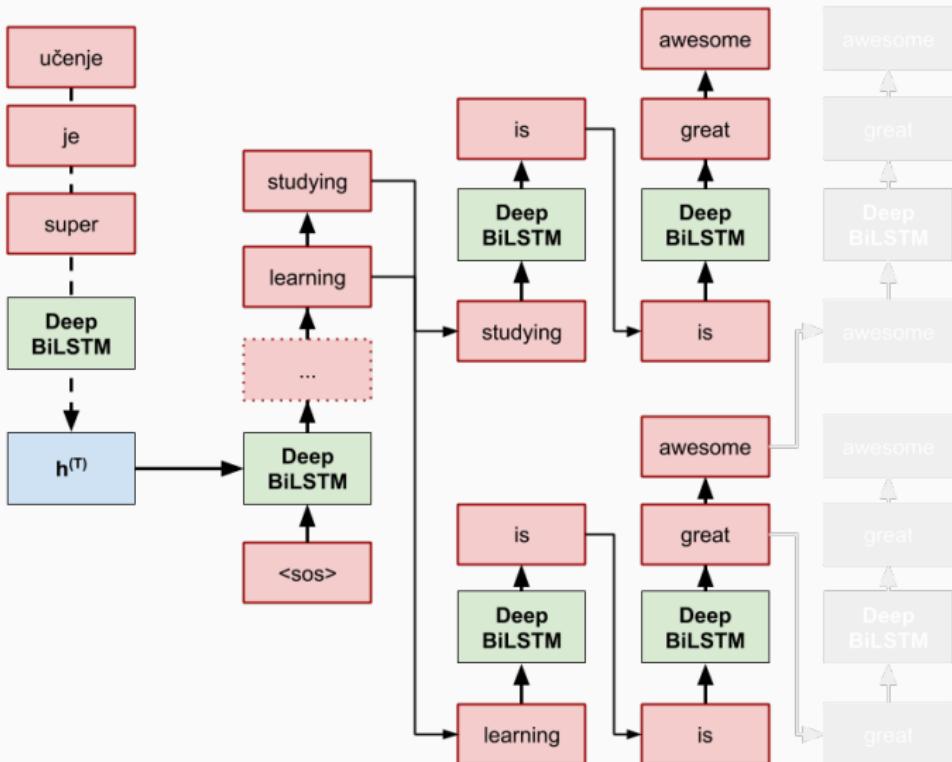
Slijed u slijed: pretraživanje zrakom



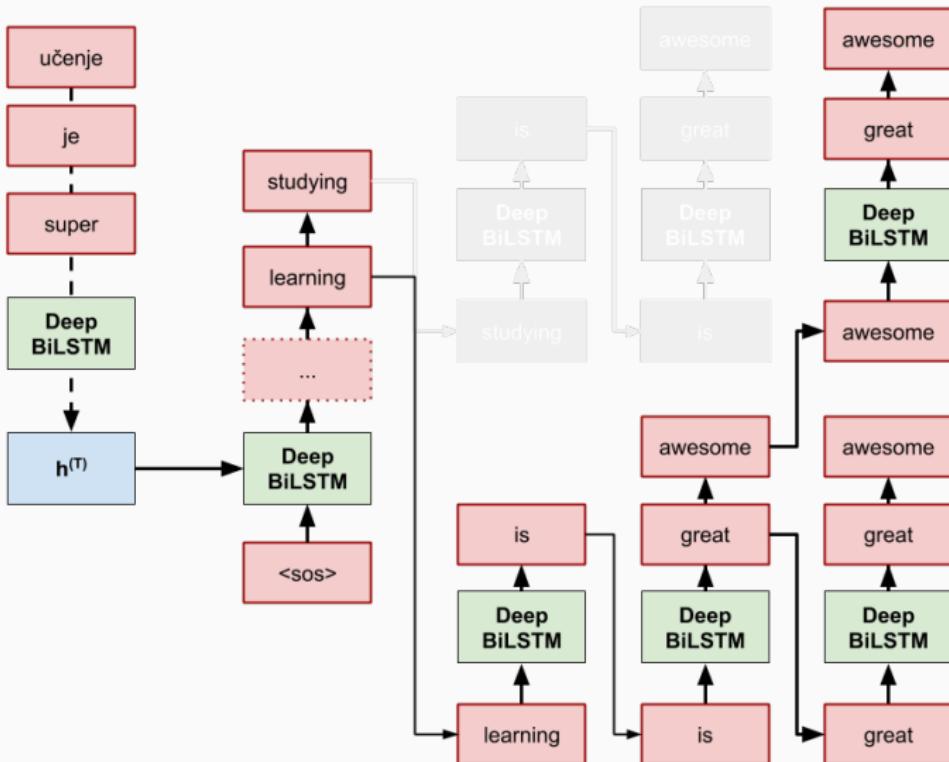
Slijed u slijed: pretraživanje zrakom (2)



Slijed u slijed: pretraživanje zrakom (3)



Slijed u slijed: pretraživanje zrakom (4)



Slijed u slijed: sažetak

Složenost prevođenja proizlazi iz varijabilne duljine ciljnog slijeda:

- umjesto "jednostavne" klasifikacije iz konteksta, model mora naučiti prediktirati cijele slijedove.

Ovom problemu pristupamo dekompozicijom na i) čitanje ulaznog slijeda i ii) generiranje izlaznog slijeda:

- koder i dekoder imaju odvojene parametre
- ti parametri združeno se uče s kraja na kraj
- isti pristup prikladan i za multimodalno prevođenje: jezik -> slika, slika -> jezik

Slijed u slijed: sažetak (2)

Rana faza učenja je posebno problematična:

- može se olakšati **forsiranjem učitelja** ("učenjem prema šalabahteru") u nekom udjelu ulaznih primjera

Generiranje prijevoda je teško:

- želimo maksimizirati vjerojatnost **slijeda** umjesto vjerojatnost pojedinačnih dijelova

Primjer:

- Il est difficile à dire → He is difficult to say (greedy)
- Il est difficile à dire → It's hard to say. (optimal)

Tom problemu možemo pristupiti **pretraživanjem zrakom**

- u svakom koraku generiranja prijevoda pratimo *k* najvjerojatnijih slijedova.

Slijed u slijed: sažetak (3)

Slabost: višejezično prevođenje zahtijeva kvadratno mnogo modela

- potrebno naučiti po jedan model za svaki par jezika

Ovom problemu možemo pristupiti prikladnim zagrijavanjem **generativnog** modela:

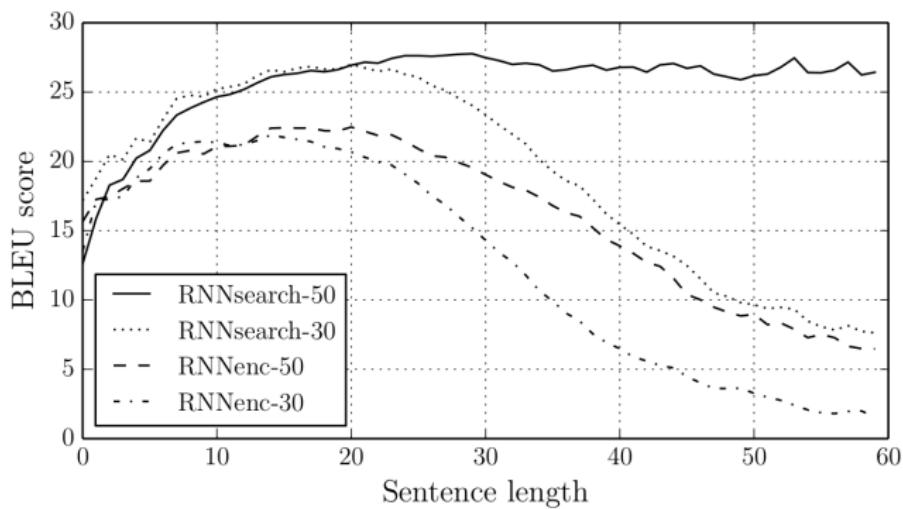
- "I wish to translate from English to Croatian. If the English sentence is 'Learning is great', then the translation is ...".

Pažnja (ili pozornost)

Pažnja

Uspješnost strojnog prevođenja za rečenice različite duljine:

- RNNsearch modeli [bahdanau14iclr] koriste pažnju.



Čak i najbolje povratne čelije znatno lošije prevode duge rečenice:

- to sugerira da povratni modeli imaju slabo pamćenje.

Pažnja: ideja

Motivacija (<https://distill.pub/2016/augmented-rnns/>):

- *"When I'm translating a sentence, I pay special attention to the word I'm presently translating. When I'm transcribing an audio recording, I listen carefully to the segment I'm actively writing down. And if you ask me to describe the room I'm sitting in, I'll glance around at the objects I'm describing as I do so."*

Naše skrivene reprezentacije **nisu** savršene (ograničena veličina).

Ako ćelija ne može zapamtiti sve, može li barem zaključiti gdje se tražena informacija može naći?

- označimo tekuću reprezentaciju dekodera kao **upit**
- označimo reprezentacije kodera kao **ključeve** (memoriju)
- pronađimo *sličnost* između upita i ključeva
- aggregirajmo prethodne reprezentacije **otežanim sažimanjem** gdje težine odgovaraju sličnosti.

Pažnja: osnovna formulacija

Funkcija attn vraća skalarnu sličnost između dva vektora:

$$a^{(t_{\text{dec}}, t_{\text{enc}})} = \text{attn}(q^{(t_{\text{dec}})}, k^{(t_{\text{enc}})}), \quad a \in \mathbb{R}, q \in \mathbb{R}^{d_q}, k \in \mathbb{R}^{d_k}.$$

- pri tome su $q^{(t_{\text{dec}})} = h_{\text{dec}}^{(t_{\text{dec}})}$ i $k^{(t_{\text{enc}})} = h_{\text{enc}}^{(t_{\text{enc}})}$ skrivena stanja modela

Treba nam sličnost između upita i *svih* ključeva:

$$a = \text{attn}(q, K), \quad a \in \mathbb{R}^T, K = [k^{(1)}, \dots, k^{(T)}].$$

Mjeru sličnosti normaliziramo na vjerojatnosnu distribuciju:

$$\alpha = \text{softmax}(a).$$

Izlaz pažnje je linearna kombinacija skrivenih stanja kodera:

$$\text{out}_{\text{attn}} = \sum_t \alpha_t k^{(t)}.$$

Pažnja: osnovna formulacija (2)

Izlaz pažnje je linearne kombinacije skrivenih stanja kodera:

- rezultat konkateniramo sa stanjem dekodera neposredno prije generiranja izlaza (**pažnja se ne koristi u povratnim ćelijama**)

$$h_{dec}^{*(t)} = [h_{dec}^{(t)}; out_{attn}] .$$

Kako formulirati funkciju sličnosti attn?

1. Diferencijabilni modul, npr. Bahdanauova funkcija s parametrima W_1 (matrica) i w_2 (vektor):

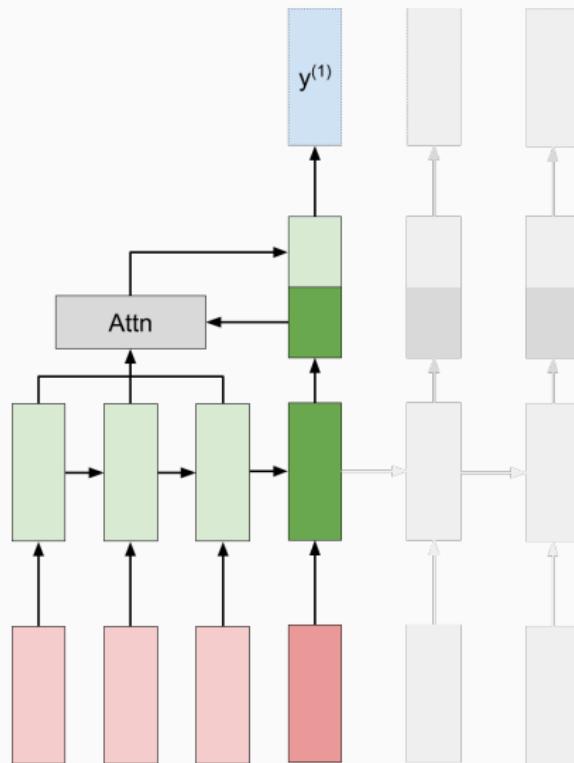
$$a^{(t)} = w_2^\top \cdot \tanh(W_1 \cdot [q^{(t)}; k^{(t)}]) .$$

2. Skalarni produkt (uvjet: $\dim(q) = \dim(k)$):

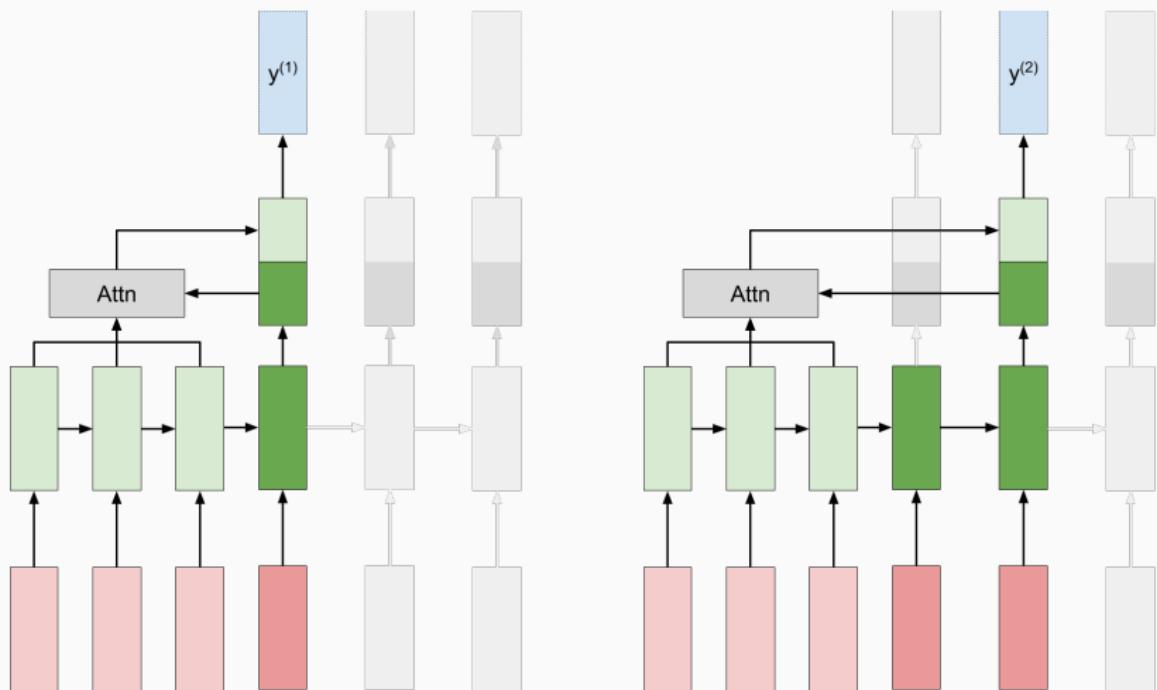
$$a^{(t)} = \frac{q^{(t)\top} \cdot k^{(t)}}{\sqrt{\dim(k)}} .$$

- skaliranje s k čuva varijancu pod pretp. $q_i, k_i \sim \mathcal{N}(0, 1)$, $q \perp k$
- pomoć: $\text{var}(q_i \cdot k_i) = 1$, $\text{var}(aX) = a^2 \text{var}(X)$

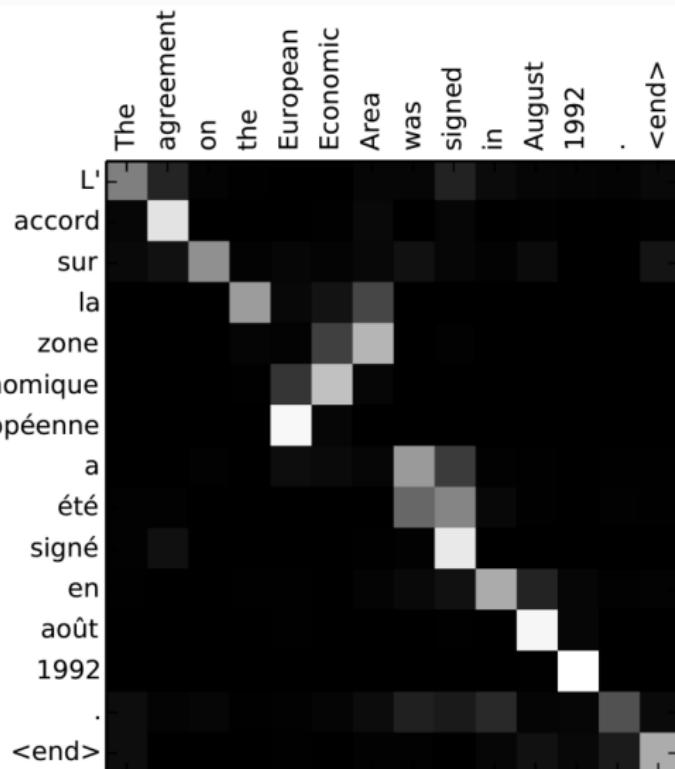
Pažnja: vizualizacija



Pažnja: vizualizacija (2)



Pažnja: vizualizacija sličnosti



Sličnost između skrivenih stanja kodera i dekodera za slučaj prijevoda iz francuskog u engleski jezik.

Pažnja: proširena formulacija

Uvodimo razliku između **ključeva i vrijednosti**:

$$k^{(t)} = f_k(h_{enc}^{(t)}), \quad v^{(t)} = f_v(h_{enc}^{(t)}) .$$

Funkcije f_k i f_v transformiraju skrivena stanja u **ključeve i vrijednosti**.

U praksi, f_k i f_v su projekcije:

$$k^{(t)} = W_k h_{enc}^{(t)}, \quad v^{(t)} = W_v h_{enc}^{(t)} .$$

Proširena pažnja:

$$\alpha = \text{softmax}(\text{attn}(q, K)),$$

$$out_{\text{attn}} = \sum_t \alpha_t v^{(t)} .$$

Ova formulacija može biti korisna i izvan prevođenja iz slijeda u slijed

Pažnja: sažetak

Naši najbolji povratni modeli i dalje se muče s dugim rečenicama

Stoga uvodimo **pažnju** za modeliranje dalekih međuovisnosti

- izlaz pažnje je težinski zbroj skrivenih stanja (ili njihovih projekcija)
- težine modeliraju **sličnost** ključeva i upita
 - značaj informacije ovisi o trenutnoj potrebi

U povratnim modelima za prevođenje sljedova, **upit** je trenutno skriveno stanje dekodera, a ključevi su skrivena stanja **kodera**

- proširene varijante mogu se primijeniti na klasifikaciju sljedova i gusto predviđanje

Pažnja: sažetak (2)

Načini definiranja sličnosti:

- Bahdanauova pažnja: diferencijabilni modul prima konkatenaciju upita i ključa
- pažnja skalarnim produktom: *izravna usporedba* (projiciranog upita s (projiciranim) ključem)

Pažnja se koristi u praktički svim modernim povratnim modelima.

Pažnja je kritična komponenta suvremenih pristupa dubokog učenja.

Pažnja kao samostalna operacija

Pažnja u klasifikaciji slijedova

Ako upit dolazi iz iste reprezentacije kao i ključevi, onda se pažnja $\text{attn}(k_i, K)$ može približiti jednojediničnom vektoru e_i :

- može se izbjegići naučenim upitim
- izgleda da računalni vid ne pati od ovog problema

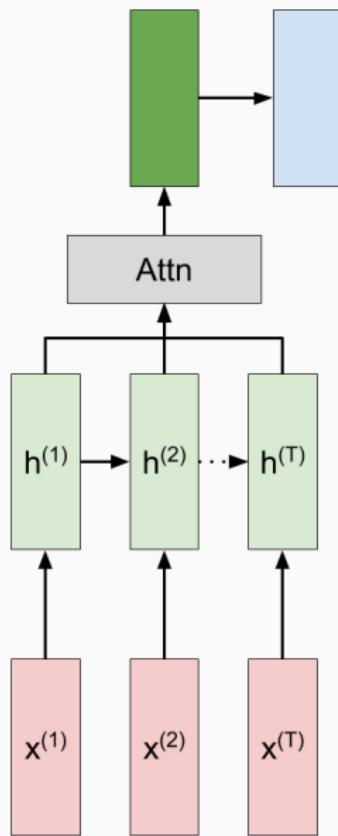
Pažnja s naučenim upitima q_ϕ :

$$\hat{\alpha} = \text{softmax}(\text{attn}(q_{phi}, K)),$$

$$out_{attn} = \sum_t^T \hat{\alpha}_t v^{(t)}.$$

Intuitivno, naučeni upiti odgovaraju apstraktnim konceptima kao što su *formalni tekst, slang, nogomet, bliski istok*, itd.

Pažnja u klasifikaciji slijeda: vizualizacija



Attention Is All You Need

Ashish Vaswani*

Google Brain

avaswani@google.com

Noam Shazeer*

Google Brain

noam@google.com

Niki Parmar*

Google Research

nikip@google.com

Jakob Uszkoreit*

Google Research

usz@google.com

Llion Jones*

Google Research

llion@google.com

Aidan N. Gomez* †

University of Toronto

aidan@cs.toronto.edu

Lukasz Kaiser*

Google Brain

lukaszkaiser@google.com

Illia Polosukhin* ‡

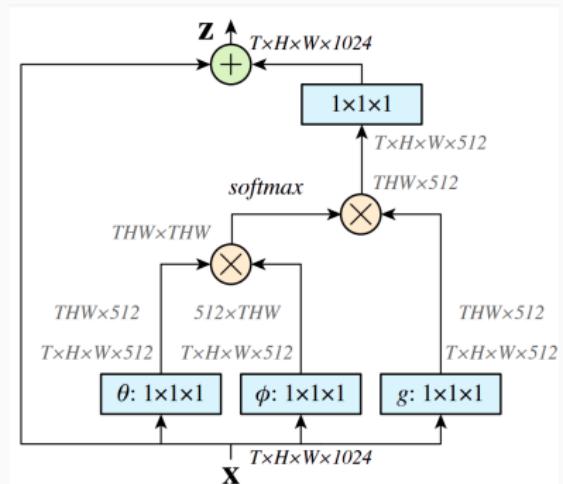
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to

Pažnja u računalnom vidu (klasifikacija videa)

Neki algoritmi računalnog vida modeliraju **daleke** međuovisnosti proširenom pažnjom bez naučenih upita



[wang18cvpr]

Ulaz: apstraktna reprezentacija X

- tenzor 4. reda $T \times H \times W \times 1024$
- gledamo ga kao $THW \times 1024$.
- H - visina, W - širina, vrijeme

Izlaz: reprezentacija Z s poboljšanim dalekim vezama

Ulaz X projiciramo na upite (θ), ključeve (ϕ) i vrijednosti (g).

Svaka značajka $x_i \in R^{1024}$ istovremeno je i upit i ključ.

Matrica sličnosti A ($THW \times THW$) uspoređuje upite s ključevima.

Pažnja u računalnom vidu (detalji)

Matricu A dobivamo matričnim množenjem:

- lako je umetnuti i drukčije formulacije sličnosti.
- tu bi dobro došlo i normaliziranje varijance ($\sqrt{1024}$)

$$\begin{aligned} A &= (W_\theta X^\top)^\top \cdot (W_\phi X^\top), \\ &= (X W_\theta^\top) \cdot (W_\phi X^\top). \end{aligned}$$

Matricu težina α dobivamo aktiviranjem redaka softmaksom.

- α_{ij} odražavan sličnost upita $W_\theta x_i$ i vrijednosti $W_g x_j$

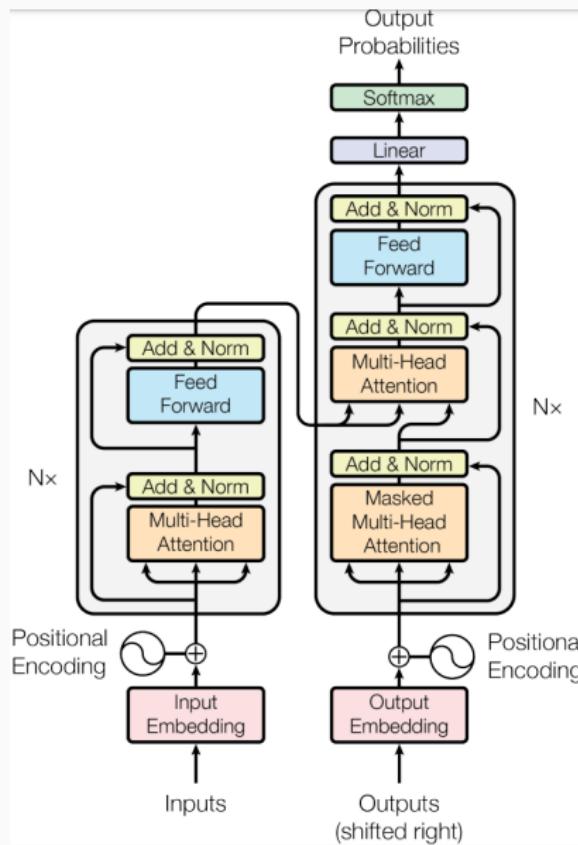
$$\alpha = \text{softmax}(A, \text{axis} = 1).$$

‘ Izlazi $Z = \{z_i\}$ su linearne kombinacije vrijednosti $V = g(x_i)$:

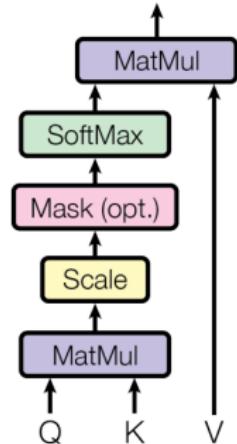
- naravno, težine odgovaraju elementima matrice α

$$z_i = \sum_j \alpha_{ij} \cdot g(x_j).$$

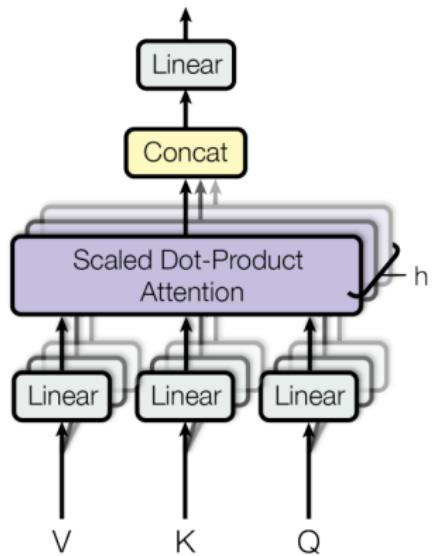
Pažnja za prevođenje slijedova



Scaled Dot-Product Attention



Multi-Head Attention



$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

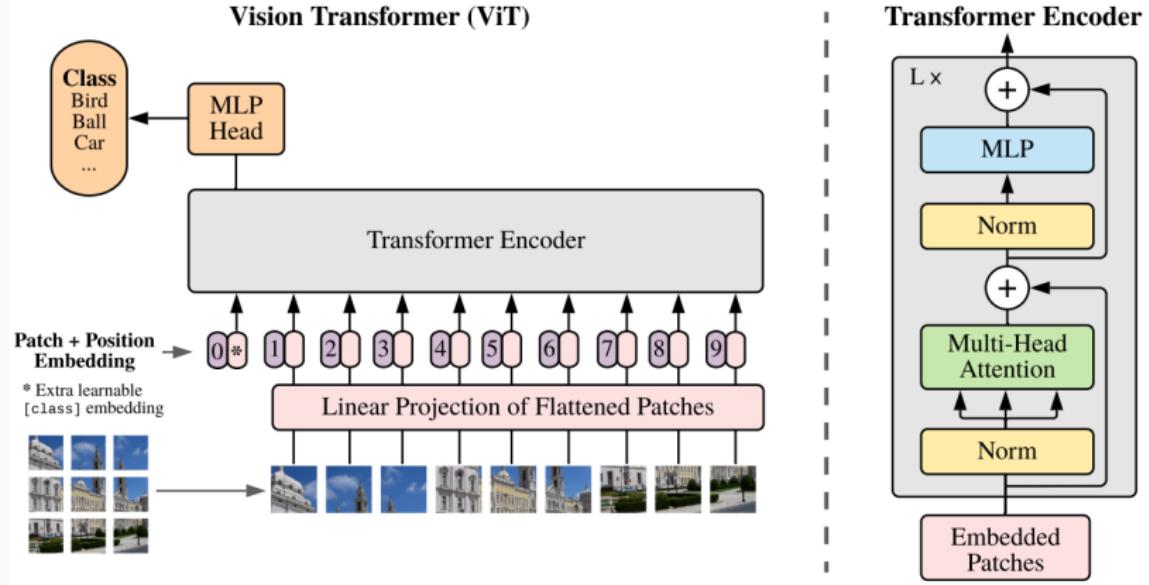
gdje $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

$$\text{MaskedAttention}(Q, K, V) = \text{softmax} \left(M + \frac{QK^\top}{\sqrt{d_k}} \right) V$$

$$\text{PE}_{\text{pos},2i} = \sin \left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}} \right),$$

$$\text{PE}_{\text{pos},2i+1} = \cos \left(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}} \right) \quad (8)$$

Gledanje pažnjom



Transformeri uče brže od povratnih modela

- nema potrebe za propagiranjem stanja
- proslijedivanje izlaza na ulaz tijekom učenja izbjegava teacher forcing

Nedostatci Vaswanijeve arhitekture:

- prikladna samo za zadatke prevođenja
- višejezično prevođenje traži kvadratno mnogo modela
- zahtijeva označene podatke (prijevode)

Generativno modeliranje teksta

Ideja: autoregresijsko pogađanje sljedećeg simbola

$$L(U) = \sum_i \log P(u_i | u_{i-k}, \dots, u_{i-1} | \Theta)$$

Arhitektura: Vaswanijev dekoder [radford18openai]!

$$\begin{aligned} h_0 &= U \cdot W_e + W_p \\ h_l &= \text{transformer_block}(h_{l-1}) \forall l \in [1, L] \\ P(u) &= \text{softmax}(h_n \cdot W_e^\top) \end{aligned} \tag{9}$$

Pitanja?

Knjiga

- relevantna poglavlja: 10.1, 10.2, 10.3, 10.4, 10.5, 10.7, 10.10, 10.11

Učenje sličnosti

metrička ugrađivanja složenih podataka

Siniša Šegvić
UniZg-FER

PLAN

- motivacija za učenje sličnosti
- sijamsko učenje
- trojni gubitak
- detalji izvedbe i vrednovanje
- primjene: stereoskopija, samonadziranje

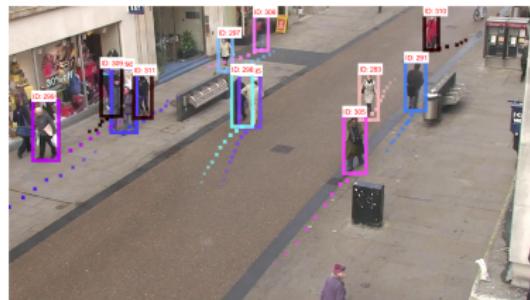
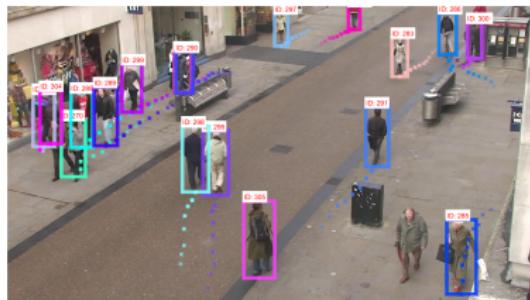
Uvod : GRANICE KLASIFIKACIJSKE PARADIGME

Kada klasifikacijski modeli mogu postati nepraktični:

- razredi ne postoje ili nisu poznati
- nepraktično veliki broj razreda

Primjeri primjena:

- stereoskopska korespondencija
- samonadzirano učenje
- praćenje, asocijativno pretraživanje, biometrijska verifikacija



[bicanic19fusion]

Uvod : GRANICE KLASIFIKACIJSKE PARADIGME (2)

Za takve primjene najpraktičnije prediktirati sličnost primjera

Lijevi par: različit. Desni par: sličan



[chopra05cvpr]

Ideja: ugraditi podatke u prikladni vektorski prostor

- standardne metrike (L_2 , ...) modeliraju sličnost među podatcima
- kratki naziv: metrička ugrađivanja

Uvod : METRIČKA UGRAĐIVANJA

Prije nego što nastavimo dalje, moramo se zapitati:

- zašto ne bismo mjerili sličnost u originalnom prostoru podataka?

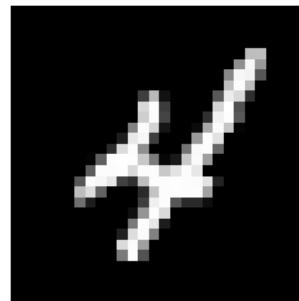
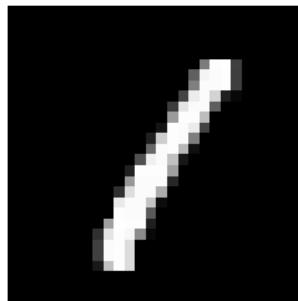
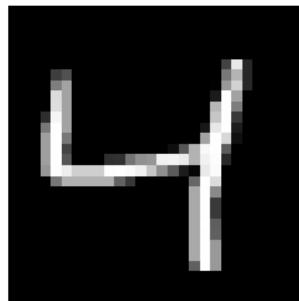
Odgovori:

- zato što udaljenosti u visokodimenzionalnim vektorskim prostorima nemaju smisla (prokletstvo dimenzionalnosti)
- zato što vektorske reprezentacije složenih podataka tipično nisu pogodne za njihovu usporedbu

Uvod : METRIČKA UGRAĐIVANJA - PRIMJER

Promotrimo udaljenosti između znamenki skupa MNIST

- L2 udaljenost između prve i druge znamenke: 121.4
- L2 udaljenost između prve i treće znamenke: 133.2
- L2 udaljenost između druge i treće znamenke: 114.9



[lecun98pieee]

Zaključak: ne postoji jaka veza između udaljenosti u originalnom prostoru i semantičke sličnosti!

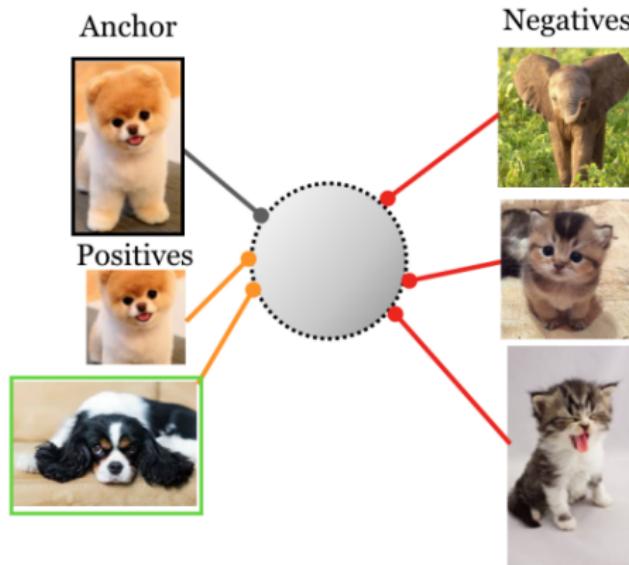
Uvod: METRIČKA UGRAĐIVANJA - PRIMJER (2)

```
import torch
import torchvision
mnist = torchvision.datasets.MNIST('data', download=True)
print(mnist.targets[:10])
# tensor([5, 0, 4, 1, 9, 2, 1, 3, 1, 4])
print(torch.sqrt(torch.sum((mnist.data[2]-mnist.data[3])**2,
                           dtype=torch.float)))
# tensor(121.4)
print(torch.sqrt(torch.sum((mnist.data[2]-mnist.data[9])**2,
                           dtype=torch.float)))
# tensor(133.2)
print(torch.sqrt(torch.sum((mnist.data[3]-mnist.data[9])**2,
                           dtype=torch.float)))
# tensor(114.9)
import cv2
cv2.imwrite('m2.png', mnist.data[2].numpy())
cv2.imwrite('m3.png', mnist.data[3].numpy())
cv2.imwrite('m9.png', mnist.data[9].numpy())
```

Uvod : METRIČKA UGRAĐIVANJA - CILJ

Ugraditi podatke u (relativno) niskodimenzionalni prostor gdje će standardna metrika modelirati sličnost među podatcima

Ako reprezentacije normiramo (tj. smjestimo ih na hipersferu), udaljenost možemo mjeriti skalarnim produktom



Uvod : METRIČKA UGRAĐIVANJA - PREDNOSTI

Podatke možemo asocirati iako u trenutku učenja nismo vidjeli sve razrede

Modelu je teže prenaučiti se:

- klasifikacija: $O(N)$ podataka za učenje
- sličnost: $O(N^2)$, $O(N^3)$ ili $O(N^\beta)$ podataka za učenje

Vrlo korisne reprezentacije mogu se naučiti i u samonadziranom kontekstu gdje zahtijevamo da podatak bude sličan perturbiranom sebi a različit od ostalih podataka; npr. SimCLR [chen20icml].

Metrička ugrađivanja mogu ponekad pomoći i u klasičnim nadziranim zadatcima [khosla20neurips].

METRIKE : POJMOVI

Razmatramo skup X te preslikavanje $d : X \times X \rightarrow R$.

Kažemo da je d metrika, a (X, d) - metrički prostor akko:

1. $d(a, b) \geq 0 \quad \forall a, b \in X$ (pozitivnost),
2. $d(a, b) = 0 \iff a = b \quad \forall a, b \in X$ (strogost),
3. $d(a, b) = d(b, a) \quad \forall a, b \in X$ (simetričnost),
4. $d(a, b) \leq d(a, c) + d(c, b) \quad \forall a, b, c \in X$ (nejednakost trokuta).

Ova definicija dobro se uklapa u koncept sličnosti podataka.

Aksiomi su redundantni: npr. pozitivnost i simetričnost slijede iz strogosti i nejednakosti trokuta.

U praksi najčešće učimo pseudo-metriku koja relaksira strogost

- teško osigurati $d(a, b) \neq 0 \quad \forall a \neq b$, zahtijevamo samo $d(a, a) = 0$

METRIKE: STANDARDNI IZBORI

Euklidska metrika:

$$d_E(a, b) = \sqrt{(a - b)^\top (a - b)} \sim (a - b)^\top (a - b)$$

Ako su podatci normirani, skalarni produkt odgovara **kosinusnoj sličnosti** te inducira isto **rangiranje** kao i Euklidska metrika:

$$\begin{aligned} d_E(a, b) &\sim (a - b)^\top (a - b) \\ &= a^\top a - 2 \cdot a^\top b + b^\top b = 2 - 2 \cdot a^\top b \\ &\sim -a^\top b \end{aligned}$$

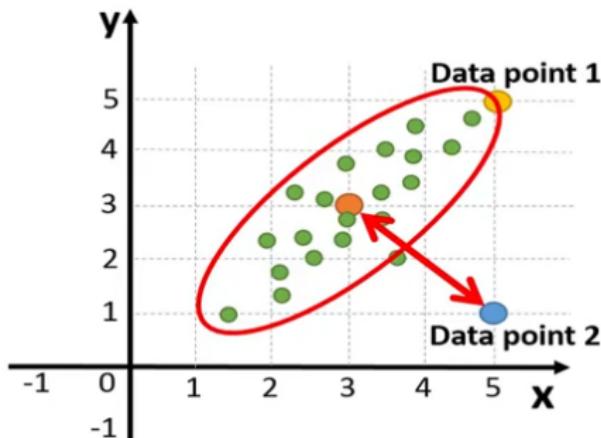
Mahalanobisova metrika (M odgovara inverznoj kovarijanci podataka):

$$d_M(a, b) \sim (a - b)^\top \cdot M \cdot (a - b)$$

METRIKE : MAHALANOBIS

Mahalanobisova izohipsa označena je crvenom bojom

- Po Mahalanobisu, žuti podatak puno je bliži narančastom od plavog podatka
- možemo hipotetizirati da plavi podatak ne pripada zelenima
- Euklidska metrika ne podržava takvo zaključivanje!



METRIKE: MAHALANOBIS (2)

Matrica M je realna i simetrična \Rightarrow može se dijagonalizirati: $M = W^\top W$

Korištenje Mahalanobisove metrike možemo interpretirati kao plitko ugrađivanje u Euklidski metrički prostor W :

$$\begin{aligned} d_M(a, b) &\sim (a - b)^\top \cdot M \cdot (a - b) \\ &\sim (W \cdot (a - b))^\top \cdot (W \cdot (a - b)) \\ &\sim (W \cdot a - W \cdot b)^\top \cdot (W \cdot a - W \cdot b) \\ &\sim d_E(Wa, Wb) \end{aligned}$$

Postoje analogni plitki pristupi koji uzimaju u obzir informaciju o pripadnosti podataka simboličkim razredima (Fisher LDA)

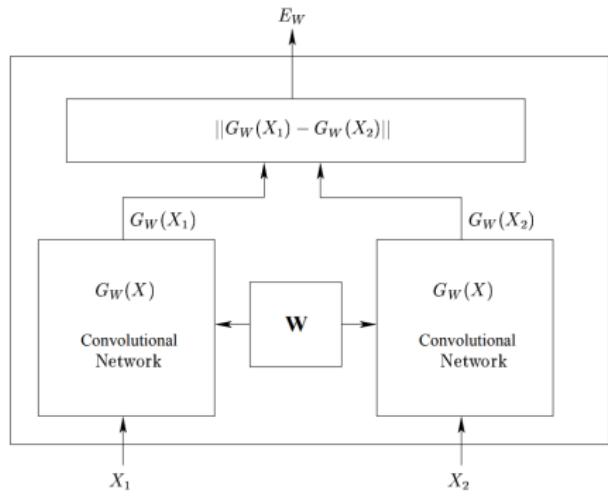
Logični korak dalje: zamijeniti W **dubokim modelom** f_θ

- ono što je logično danas, nije bilo logično 2005...

SIJAMSKO UČENJE : IDEJA

Naučiti model G_W koji ugrađuje podatke X u prostor gdje euklidska metrika E_W odražava sličnost među podatcima

Sijamsko učenje: parovi podataka prolaze kroz dva primjerka modela.



[chopra05cvpr]

Primjeri dijele parametre W , a gradijenti odgovarajućih parametara dviju grana se akumuliraju.

SIJAMSKO UČENJE : GUBITAK

Sijamsko učenje koristi neku od varijanti kontrastnog gubitka

Kontrastni gubitak ovisi o tome jesmo li na ulaze sijamskih modela doveli primjerke istog razreda

$$L(\theta) = \sum_{y_q=y_p} L_{\text{pos}}(\theta|x_q, x_p) + \sum_{y_q \neq y_n} L_{\text{neg}}(\theta|x_q, x_n)$$

Gubitak L_{pos} tjera primjerke istih razreda da se približe:

$$L_{\text{pos}}(\theta|x_q, x_p) = \|f_\theta(x_q) - f_\theta(x_p)\|^2$$

Gubitak L_{neg} tjera različite primjerke da se udalje [hadsell06cvpr]:

$$L_{\text{neg}}(\theta|x_q, x_n) = [\max(0, m - \|f_\theta(x_q) - f_\theta(x_n)\|)]^2$$

SIJAMSKO UČENJE : GRADIJENTI

Pogledajmo gradijente gubitaka L_{pos} i L_{neg} s obzirom na metrička ugrađivanja $f_q = f_\theta(x_q)$:

$$\frac{\partial L_{\text{pos}}}{\partial f_p} = 2 \cdot (f_p - f_q)$$

$$\frac{\partial L_{\text{neg}}}{\partial f_n} = 2 \cdot \underbrace{\max(0, m - \|f_q - f_n\|)}_{\text{iznos}} \cdot \underbrace{\frac{f_q - f_n}{\|f_q - f_n\|}}_{\text{smjer}}$$

Ovi gradijenti potiču približavanje ugrađivanja pozitivnih parova te udaljavanje negativnih parova sve dok oni ne postanu udaljeniji od m .

Gradijent $\frac{\partial L_{\text{neg}}}{\partial f_n}$ je vektor:

- usmjeren je od negativa prema sidru: $\frac{f_q - f_n}{\|f_q - f_n\|}$
- iznos opada kako udaljenost raste prema m : $\max(0, m - \|f_q - f_n\|)$

SIJAMSKO UČENJE : GRADIJENTI (2)

Domaći rad:

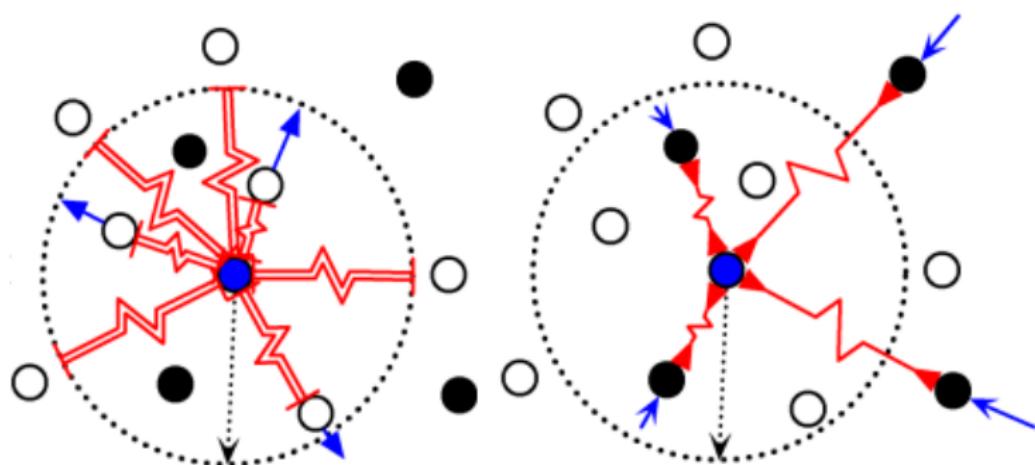
- izvesti gradijente kontrastnog gubitka $\frac{\partial L_{\text{pos}}}{\partial f_p}$ i $\frac{\partial L_{\text{neg}}}{\partial f_n}$.
- usporediti gradijent $\frac{\partial L_{\text{neg}}}{\partial f_n}$ s gradijentom alternativne formulacije:

$$L_{\text{neg2}}(\theta|x_q, x_p) = \max(0, m^2 - \|f_\theta(x_q) - f_\theta(x_p)\|^2)$$

SIJAMSKO UČENJE : GRADIENTI (3)

Dinamiku kontrastnog gubitka možemo ilustrirati sustavom mehaničkih opruga (sila opruge proporcionalna je udaljenosti)

- crni i bijeli krugovi predstavljaju pozitivne i negativne primjere s obzirom na plavi podatak
- negativi izvan radijusa m ne osjećaju odbijanje plavog podatka



[hadsell06cvpr]

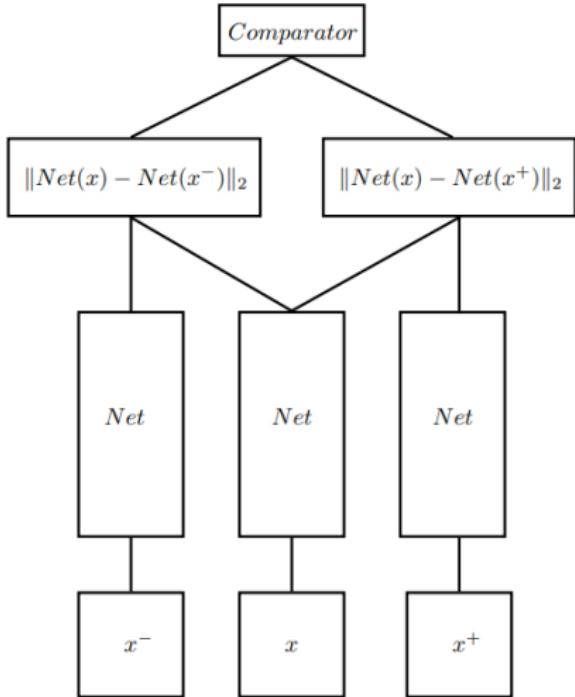
TROJNO UČENJE: IDEJA

U sijamskom učenju jedan te isti podatak uspoređujemo i s negativnim i s pozitivnim primjerima.

Pri tome ugrađivanje promatranog primjera trebamo izračunati dva puta

Taj problem adresira trojno učenje:

- referentno ugrađivanje
uspoređujemo s pozitivnim i
negativnim primjerom



[hoffer15iclrw]

Tri primjerka modela dijele parametre i ravnopravno doprinose gradijentima gubitka

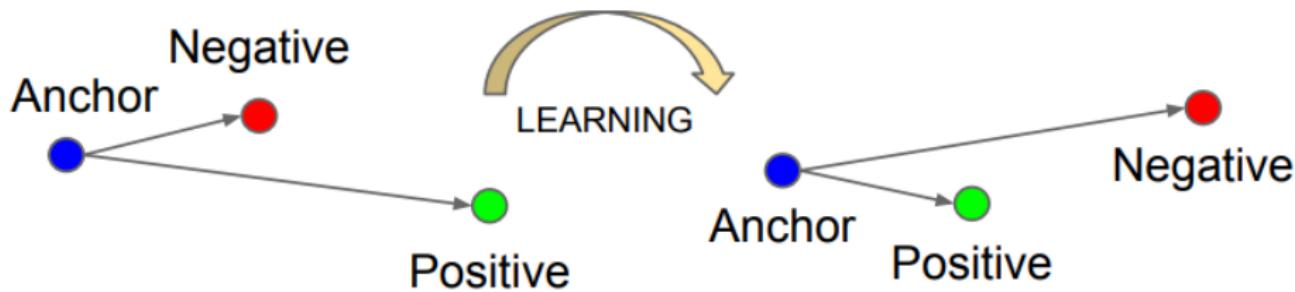
TROJNO UČENJE : GUBITAK

Trojno učenje tipično koristi trojni gubitak

Trojni gubitak spaja obje komponente kontrastnog gubitka u jedan izraz:

$$L(\theta) = \sum_i \max(0, \|f_\theta(x_{ia}) - f_\theta(x_{ip})\| - \|f_\theta(x_{ia}) - f_\theta(x_{in})\| + \alpha)$$

Trojni gubitak privlači referentni i pozitivan podatak te odbija referentni i negativan podatak:



[schroff15cvpr]

TROJNO UČENJE : GRADIJENTI

Pogledajmo gradijente trojnog gubitka s obzirom na ugrađivanja

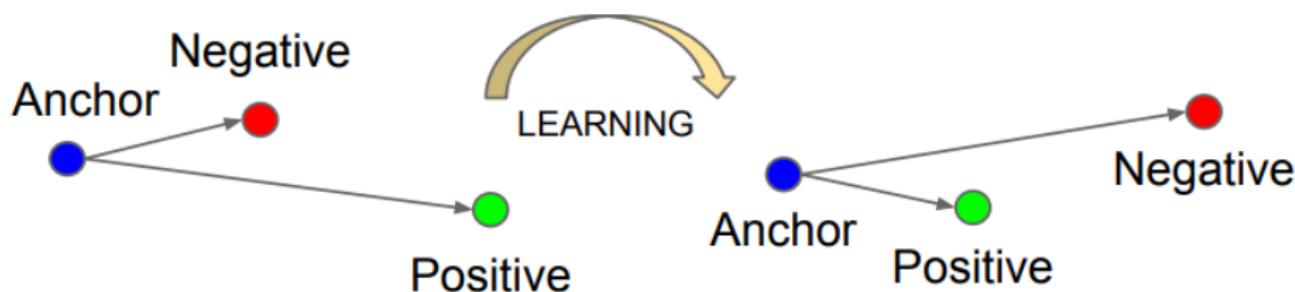
$$f_a = f_\theta(x_{ia}), f_p = f_\theta(x_{ip}) \text{ i } f_n = f_\theta(x_{in}):$$

$$\frac{\partial L}{\partial f_p} = [\|f_a - f_p\| + \alpha] > \|f_a - f_n\| \cdot \frac{f_p - f_a}{\|f_p - f_a\|}$$

$$\frac{\partial L}{\partial f_n} = [\|f_a - f_p\| + \alpha] > \|f_a - f_n\| \cdot \frac{f_a - f_n}{\|f_a - f_n\|}$$

Ti gradijenti testiraju je li udaljenost od f_p do f_a manja od $\|f_n - f_a\| - \alpha$.

- ako nije, gradijenti potiču približavanje f_p i udaljavanje f_n :



DETALJI: MEKA ZGLOBNICA

Klasični trojni gubitak zanemaruje trojke kod kojih je negativ dalji od pozitiva za više od margine.

Prednost tog pristupa jest onemogućavanje prenaučenosti: kada se podaci dovoljno dobro rasporede --- učenje prestaje

Međutim, s druge strane, takav ziheraški pristup može dovesti do lošije generalizacije

Stoga ponekad možemo poboljšati generalizaciju na način da čvrstu zglobnicu $\text{ReLU}(x) = [x]_+$ zamijenimo njenom mekom varijantom:

$$\text{softplus}(x) = \ln(1 + e^x)$$

DETALJI: VERZIJA SA SKALARnim PRODUKTOM

Dogovorimo skraćenu notaciju ugrađivanja s obzirom na x_p , x_p , x_n :

$$f_a = f_\theta(x_{ia})$$

$$f_p = f_\theta(x_{ip})$$

$$f_n = f_\theta(x_{in})$$

Ako su latentne reprezentacije normirane, $\|f_a\| = \|f_p\| = \|f_n\| = 1$, gubitak možemo izraziti i kroz kosinusnu sličnost:

$$L(\theta) = \sum_i \max(0, f_\theta(x_{ia})^\top f_\theta(x_{in}) - f_\theta(x_{ia})^\top f_\theta(x_{ip}) + \alpha)$$

DETALJI: FORMIRANJE TROJKI

Pojednostavljeni izraz za osnovni trojni gubitak [hermans17arxiv]:

$$L_3(\theta) = \sum_{y_a=y_p \neq y_n} [\alpha + D_{ap} - D_{an}]_+$$

Formiranje trojki za učenje vrlo je važan izvedbeni detalj

- glavni problem je u učinkovitosti učenja
- broj svih trojki raste s $O(N \cdot \overline{N_p} \cdot \overline{N_n})$



[schroff15cvpr]

DETALJI: FORMIRANJE TROJKI (2)

Ponekad koristimo čvrsti trojni gubitak na slučajnim grupama (eng. batch-hard [hermans17arxiv]):

- referentni podatak a povezujemo s najtežim pozitivom i najtežim negativom [schroff15cvpr].

$$L_{BH}(\theta) = \sum_a [\alpha + \max_{y_a=y_p} D_{ap} - \min_{y_a \neq y_p} D_{an}]_+$$

Ponekad trojke formiramo unaprijed [zbontar15cvpr]:

- svaki primjer koristimo kao referentni (sidro)
- ako je praktično, tražimo teške pozitive i negative s obzirom na ažurni skup parametara [schroff15cvpr]

DETALJI: N PAROVA

Pretpostavimo da u grupi imamo N parova podataka [sohn16neurips]:

- svi parovi dijele sidro: \mathbf{x}_a
- samo jedan par je pozitivan: $(\mathbf{x}_a, \mathbf{x}_p)$
- svi preostali parovi su negativni (ima ih $N - 1$): $(\mathbf{x}_a, \mathbf{x}_{ni})$

Gubitak definiramo tako da raste kad je sidro slično negativima a pada kad je sidro slično pozitivima:

$$\mathcal{L}_{N\text{-pairs}}(\mathbf{x}_a, \mathbf{x}_p, \{\mathbf{x}_{ni}\}) = \log \left(1 + \sum_{i=1}^{N-1} e^{f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_{ni}) - f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_p)} \right)$$

Za $n = 2$, $\mathcal{L}_{N\text{-pairs}}$ vrlo je sličan trojnom gubitku sa skalarnim produktom.

DETALJI: N PAROVA (2)

Nakon nekoliko jednostavnih koraka $\mathcal{L}_{\text{N-pairs}}$ svodimo na sljedeći oblik:

$$\begin{aligned}\mathcal{L}_{\text{N-pairs}}(\mathbf{x}_a, \mathbf{x}_p, \{\mathbf{x}_{ni}\}) &= \\ &= -\log \frac{\exp(f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_p))}{\exp(f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_p)) + \sum_{i=1}^{N-1} \exp(f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_{ni}))} \\ &= -\log \frac{\exp(f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_p))}{\sum_{i=1}^N \exp(f_\theta(\mathbf{x}_a)^\top f_\theta(\mathbf{x}_i^{\text{all}}))}\end{aligned}$$

Vidimo da je $\mathcal{L}_{\text{N-pairs}}$ ekvivalentan standardnoj unakrsnoj entropiji nad softmaksom vektora sličnosti parova podataka.

Gubitak n parova ekvivalentan je infoNCE gubitku [vandenoord18arxiv]:

- infoNCE dolazi do iste jednadžbe optimiranjem zajedničke informacije.

DETALJI: N PAROVA (3)

Gubitak n parova može se poopćiti i na slučaj kad imamo više pozitivnih i više negativnih primjera u grupi $\{\mathbf{x}_i, y_i\}_{i=1}^B$ (eng. soft nearest neighbours) [frosst19icml]:

$$\mathcal{L}_{\text{snn}} = -\frac{1}{B} \sum_{i=1}^B \log \frac{1}{|\{y_j = y_i, j \neq i\}|} \frac{\sum_{j \neq i, y_j = y_i} e^{f_\theta(\mathbf{x}_i)^\top f_\theta(\mathbf{x}_j)/\tau}}{\sum_{y_i \neq y_k} e^{f_\theta(\mathbf{x}_i)^\top f_\theta(\mathbf{x}_k)/\tau}}$$

- hiper-parametar τ (temperatura) modulira entropiju izlaza.

Stroža varijanta tog gubitka zahtjeva da **svaki** pozitiv iz grupe bude sličniji od negativa [khosla20neurips]:

$$\mathcal{L}_{\text{sum-out}} = -\frac{1}{B} \sum_{i=1}^B \frac{1}{|\{y_j = y_i, j \neq i\}|} \sum_{j \neq i, y_j = y_i} \log \frac{e^{f_\theta(\mathbf{x}_i)^\top f_\theta(\mathbf{x}_j)/\tau}}{\sum_{y_i \neq y_k} e^{f_\theta(\mathbf{x}_i)^\top f_\theta(\mathbf{x}_k)/\tau}}$$

- ova varijanta generalizira bolje iako $\mathcal{L}_{\text{snn}} > \mathcal{L}_{\text{sum-out}}$ (Jensen)

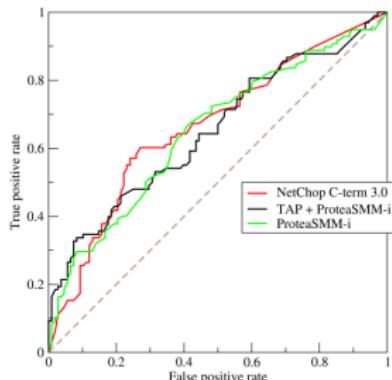
DETALJI: VREDNOVANJE

Naučene mjere sličnosti induciraju rangiranje:

- fiksiramo podatak x_a , sortiramo sve druge podatke prema padajućoj sličnosti: $s_{ai} = -d(f_\theta(x_a), f_\theta(x_i))$
- ako rangiranje savršeno generalizira (to obično nije slučaj) onda su svi pozitivi sortirani prije negativa

Kvalitetu rangiranja mjerimo površinom ispod krivulja PR ili ROC

- veća površina \Rightarrow bolji model



AUROC odgovara vjerojatnosti da slučajni pozitiv rangira ispred slučajnog negativa.

Nepogodno za nebalansirane probleme:

- za takve probleme preferiramo AUPR.

DETALJI: VREDNOVANJE (2)

Evo kako formirati krivulje za podatak x_a :

- postavljamo prag na svaki indeks i : podatci koji su sličniji od s_{ai} su pozitivi (ima ih P_i), a ostali su negativi (ima ih N_i);
- uz pomoć oznaka dobivamo brojnosti
 - točnih pozitiva - TP_i ,
 - lažnih pozitiva - FP_i ,
 - lažnih negativa - FN_i ,
 - i točnih negativa - TN_i ;
- sada možemo (i dalje za taj isti prag i) odrediti relevantne metrike:
 - odziv $R_i = TPR_i = TP_i / (TP_i + FN_i)$
 - preciznost $P_i = TP_i / (TP_i + FP_i)$
 - udio lažnih pozitiva = $FPR_i = FP_i / (TN_i + FP_i)$
- krivulju preciznosti i odziva (PR) čine točke (R_i, P_i)
- krivulju ROC čine točke (FPR_i, TPR_i)

DETALJI: ZADATAK

Zadani su podatci x_1 do x_5 .

Poznato je da su identiteti podataka redom $Y=[1, 0, 0, 1, 1]$

Poznato je da udaljenosti od podatka x_1 iznose:

$$d(x_1, X) = [0.0, 5.0, 2.0, 3.0, 1.0]$$

Odredite površinu ispod P-R krivulje (skraćeno AUPR) za predikcije modela u podatku x_1

Napomena: AUPR često nazivamo i prosječnom preciznošću (eng. average precision, AP)

DETALJI: ZADATAK - RJEŠENJE

Rangiranje podataka je: $[x_5^{(1)}, x_3^{(0)}, x_4^{(1)}, x_2^{(0)}]$

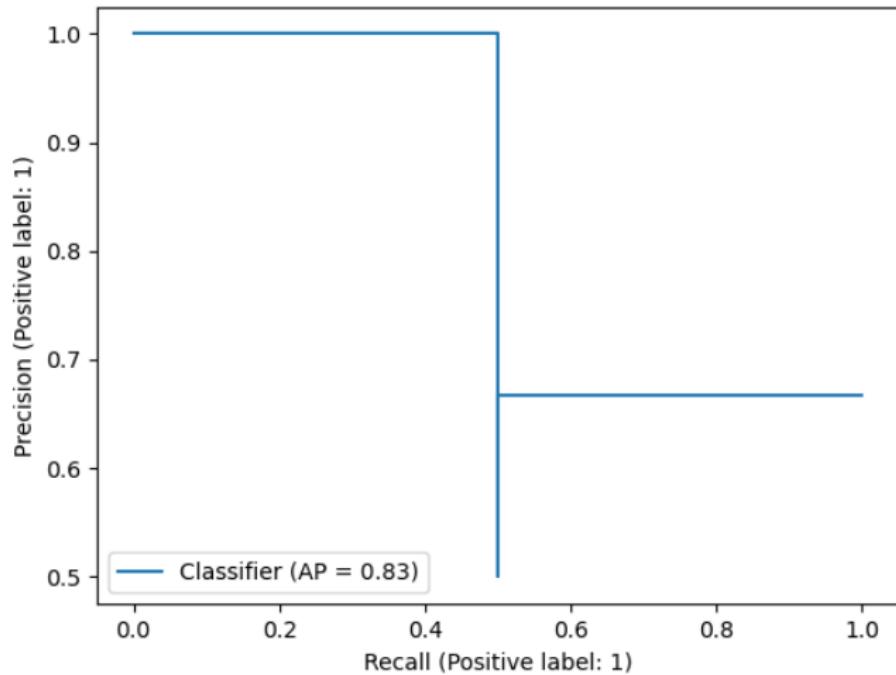
Postavljamo prag na svaki podatak i mjerimo preciznost $P=TP/(TP+FP)$ i odziv $R=TP/(TP+FN)$:

pozitivne predikcije	TP	FP	FN	P	R
x_5, x_3, x_4, x_2	2	2	0	0.5	1.0
x_5, x_3, x_4	2	1	0	0.7	1.0
x_5, x_3	1	1	1	0.5	0.5
x_5	1	0	1	1.0	0.5

Za niti jedan prag nemamo $R=0$. Zato dogovorno dodajemo točku ($R=0$, P =preciznost za najmanji R).

Ako za isti R imamo više P -ova — smijemo izabrati bolji.

DETALJI: ZADATAK - GRAF



$$\text{Rješenje: AUPR} = (0.5-0) \times 1 + (1-0.5) \times 0.66 = 0.83$$

DETALJI: ZADATAK - KOD

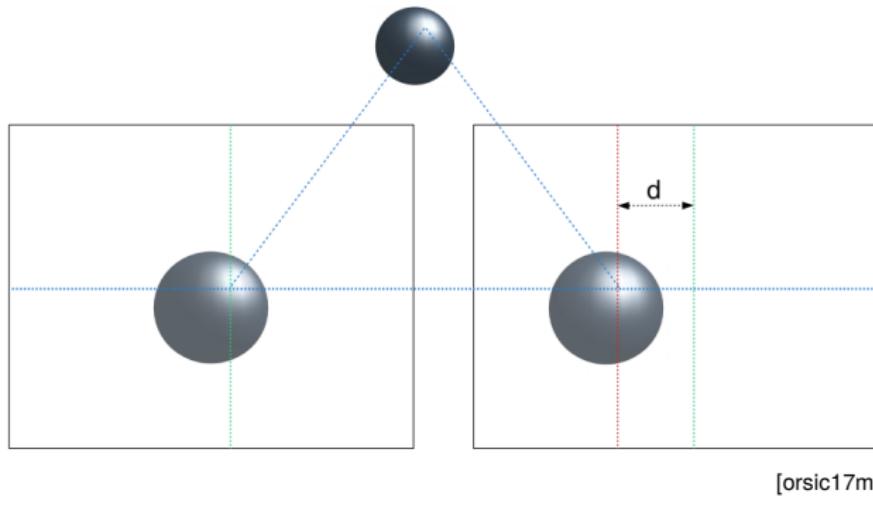
```
import numpy as np
from sklearn.metrics import average_precision_score
from sklearn.metrics import PrecisionRecallDisplay
import matplotlib.pyplot as plt

y_true = np.array([0, 0, 1, 1])
y_scores = np.array([-5, -2, -3, -1])
print(average_precision_score(y_true, y_scores))

PrecisionRecallDisplay.from_predictions(y_true, y_scores)
plt.show()
```

STEREO : ZADATAK

Za svaki piksel lijeve slike tražimo korespondentni piksel u desnoj slici:



[orsic17ms]

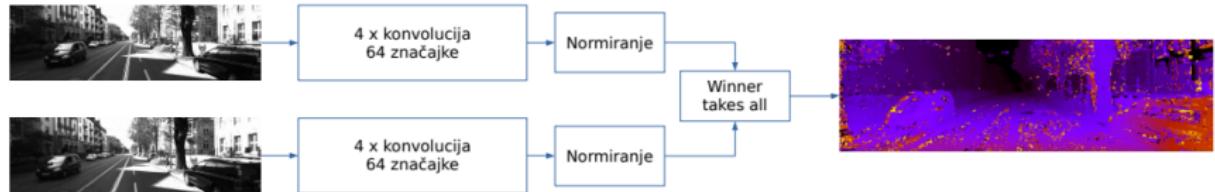
Pretpostavljamo kalibrirani slučaj: korespondencija je u istom retku

- tražimo gusto polje horizontalnih pomaka (dispariteta)
- ako znamo disparitet, širinu vidnog polja kamere i udaljenost među kamerama - možemo odrediti dubinu tog dijela scene u metrima

STEREO : IDEJA

Ugraditi piksele obje slike u metrički prostor konvolucijskim modelom

$$f_{\theta} : \mathbb{R}^{3 \times H \times W} \rightarrow \mathbb{R}^{F \times H \times W}, F=64 \text{ [zbontar15cvpr].}$$



[orsic17ms]

Formirati gusti volumen cijene V oblika $D \times H \times W$:

- $V_{ijd} = \text{cost}(f_{\theta}(I_L)_{i,j}, f_{\theta}(I_R)_{i,j+d})$

U svakom pikselu odrediti najbolji disparitet (winner takes all):

- $D_{ij} = \arg \min_d V_{ijd}$

STEREO : DETALJI

Trojni gubitak izražavamo skalarnim produktom nad normiranim metričkim ugrađivanjima isječaka 9×9 :

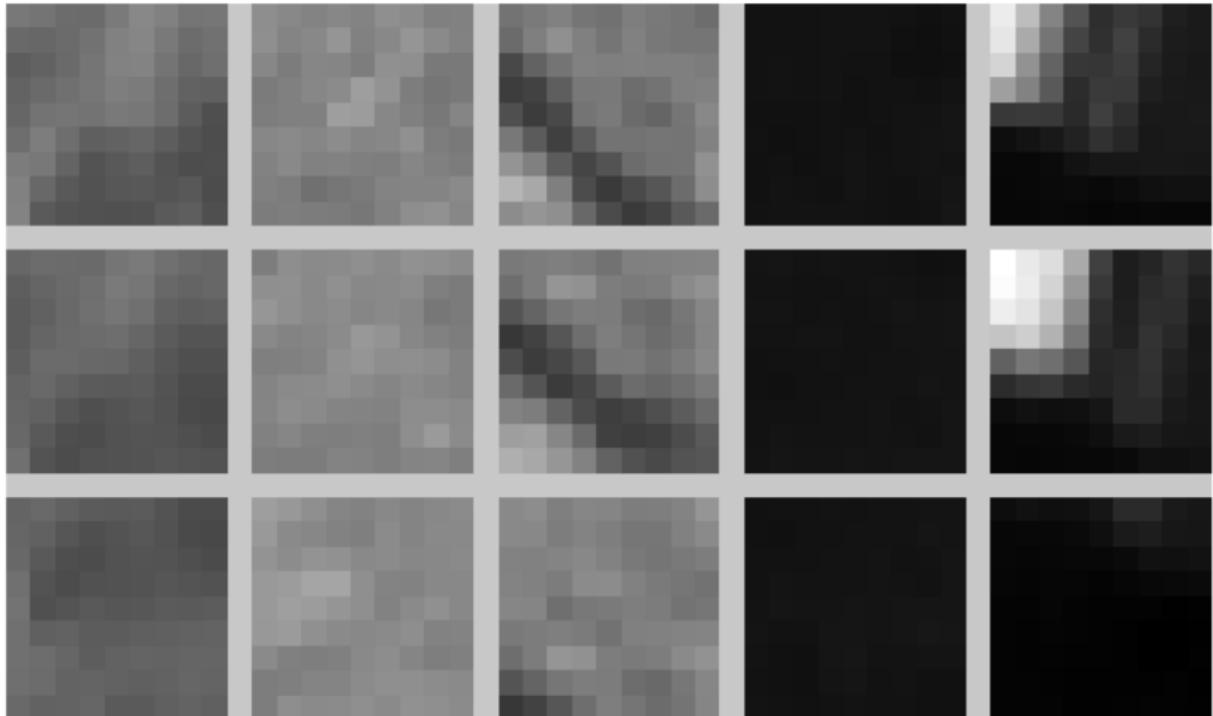
- tri primjerka modela f_θ dijele parametre
- svaki primjerak pamti aktivacije i računa gradijente
- ukupni gradijent za svaki parametar dobivamo agregacijom doprinosu primjeraka modela

Model f_θ je ekvivariantan: $4 \times \text{conv}3 \times 3$ bez sažimanja:

- ulazni podatci za učenje imaju dimenzije $128 \times 3 \times 9 \times 9$
- pri učenju ne koristimo nadopunjavanje (pri zaključivanju - da)
- ugrađivanja imaju 64 dimenzije, $f_\theta : \mathbb{R}^{3 \times 9 \times 9} \rightarrow \mathbb{R}^{64}$
- pikseli slika se normiraju na $N(0, \mathbf{I})$ pri učenju i zaključivanju
- zaključivanje primjenjujemo na cjelokupne slike $2 \times 3 \times H \times W$

STEREO : TROJKE

Stupci prikazuju trojke za učenje:

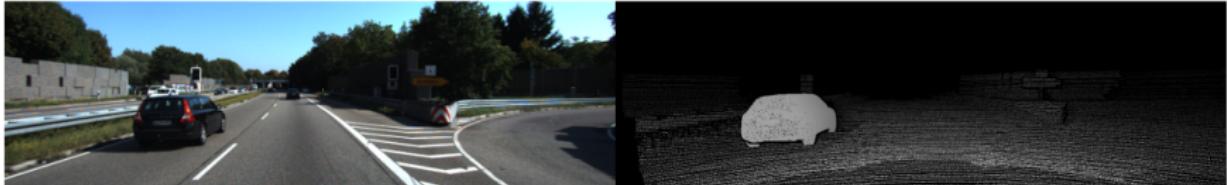


[orsic17ms]

STEREO : EKSPERIMENTI

Skup podataka KITTI [geiger13ijrr]:

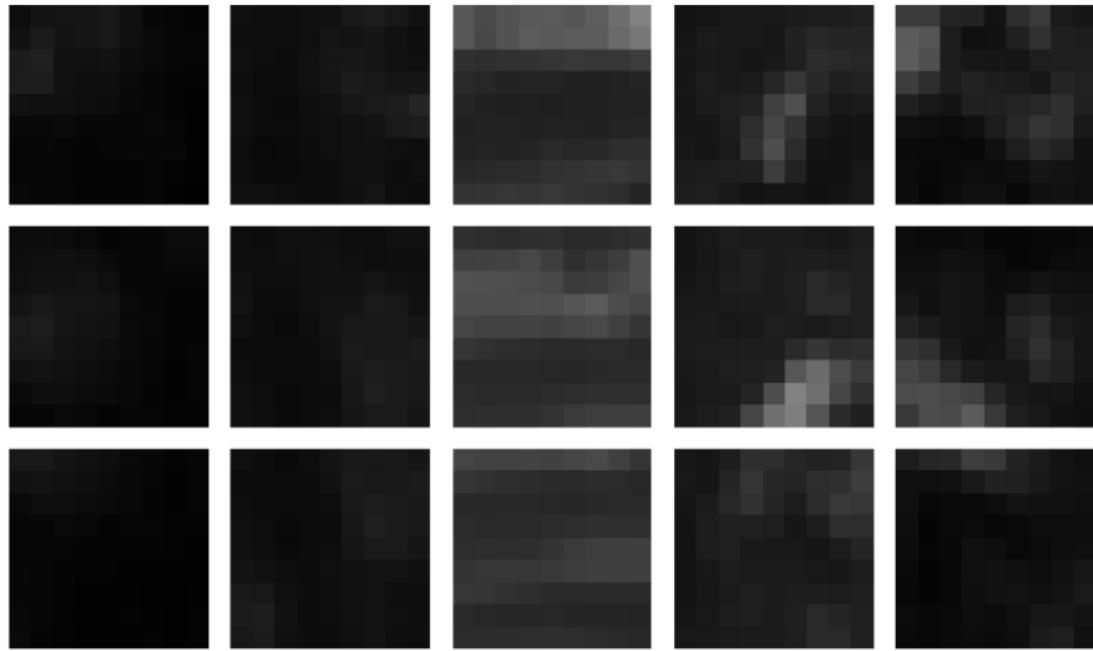
- 200 rektificiranih slika 1382 x 512
- skup za učenje: 80% slika (ostatak - skup za validaciju)
- točni dispariteti izmjereni LIDAR-om u 30% piksela
- gusti dispariteti na automobilima dobiveni fitanjem CAD modela



[orsic17ms]

STEREO : GREŠKE

Mnogi pikseli nemaju korespondenciju zbog "stereoskopske sjene":



[orsic17ms]

Retci prikazuju i) sidro, ii) korespondenciju, iii) najsličniji negativ.

STEREO : TOČNOST

Stereoskopske metode na KITTI-ju uspoređujemo prema postotku točnih dispariteta s tolerancijom ± 3 piksela.

Eksperimentalna točnost je solidna, iako lošija od stanja tehnike:

Model	Točnost - treniranje	Točnost - testiranje
Sive ulazne slike	84.99%	82.32%
Sive ulazne slike - BN	74.51%	71.61%
Ulagne slike u boji	85.50%	82.84%
Ulagne slike u boji - BN	74.40%	71.33%

[orsic17ms]

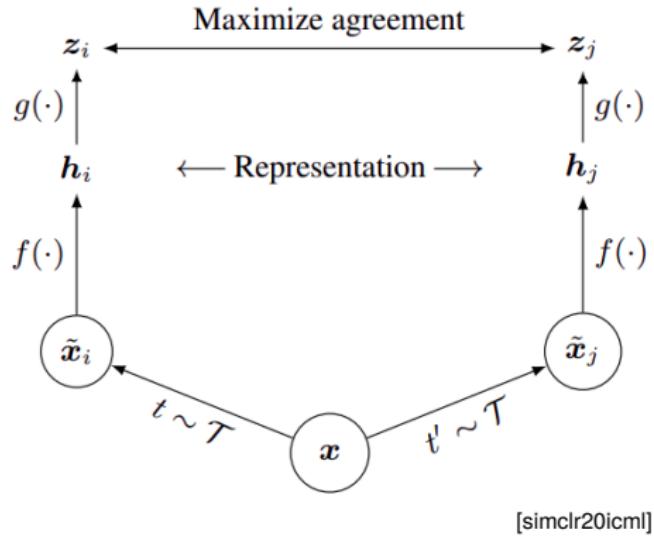
Velika prednost ovakvih pristupa: otpornost na prenaučenost.

Sastavna komponenta novijih pristupa koji rješavaju preostale izazove:

- učenje na neoznačenom videu [liu20cvpr]
- popunjavanje područja bez korespondencija analizom konteksta

SAMONADZIRANJE : ZADATAK

Naučiti korisne reprezentacije bez korištenja semantičkih oznaka.



- f - konvolucijska okosnica
 - npr. ResNet-50
- g - projekcijski modul
 - npr. $2 \times \text{FC}(128)$
- $t, t' \in \mathcal{T}$ - slučajne perturbacije
 - npr. izrezivanje, rastresanje boje, zaglađivanje

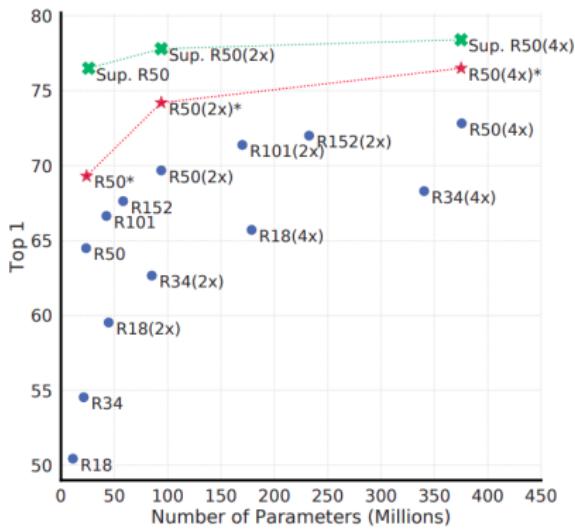
Prednosti: možemo učiti bez oznaka, bolji prijenos učenja!

Nedostatci: dugotrajno učenje, velike grupe, trivijalna rješenja?

SAMONADZIRANJE : SimCLR

SimCLR: Simple Contrastive Learning of visual Representations

- gubitak N parova: za svaki pozitivni par imamo $2(B-1)$ negativa
- zaključivanje: odbaciti g , koristiti f za prijenos učenja
- evaluacija: nad f naučiti višerazrednu logističku regresiju
- ostale primjene: predtreniranje + ugađanje, polunadzirano učenje



- veći modeli i veće grupe (4096) bolje uče
- SimCLRv2 + linear gotovo jednako dobar kao i nadzirano učenje
- za najbolje rezultate (*) trebamo $10 \times$ duže učiti

SAMONADZIRANJE : SIMCLR (2)

Algorithm 1 SimCLR's main learning algorithm.

```
input: batch size  $N$ , constant  $\tau$ , structure of  $f, g, \mathcal{T}$ .  
for sampled minibatch  $\{\mathbf{x}_k\}_{k=1}^N$  do  
    for all  $k \in \{1, \dots, N\}$  do  
        draw two augmentation functions  $t \sim \mathcal{T}, t' \sim \mathcal{T}$   
        # the first augmentation  
         $\tilde{\mathbf{x}}_{2k-1} = t(\mathbf{x}_k)$   
         $\mathbf{h}_{2k-1} = f(\tilde{\mathbf{x}}_{2k-1})$  # representation  
         $\mathbf{z}_{2k-1} = g(\mathbf{h}_{2k-1})$  # projection  
        # the second augmentation  
         $\tilde{\mathbf{x}}_{2k} = t'(\mathbf{x}_k)$   
         $\mathbf{h}_{2k} = f(\tilde{\mathbf{x}}_{2k})$  # representation  
         $\mathbf{z}_{2k} = g(\mathbf{h}_{2k})$  # projection  
    end for  
    for all  $i \in \{1, \dots, 2N\}$  and  $j \in \{1, \dots, 2N\}$  do  
         $s_{i,j} = \mathbf{z}_i^\top \mathbf{z}_j / (\|\mathbf{z}_i\| \|\mathbf{z}_j\|)$  # pairwise similarity  
    end for  
    define  $\ell(i, j)$  as  $\ell(i, j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$   
 $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^N [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$   
    update networks  $f$  and  $g$  to minimize  $\mathcal{L}$   
end for  
return encoder network  $f(\cdot)$ , and throw away  $g(\cdot)$ 
```

[simclr20icml]

Gubitak ima 2B članova:

- svaki član je $\ell_{NP}(i, j)$
 - i : sidro
 - j : pozitiv
 - $k \neq i$: negativi
- $\ell_{NP}(i, j)$ je izražen s obzirom na kosinusne sličnosti s_{ij} i s_{ik}
- negativi su svi preostali podatci minigrupe
 - $k \in [1..2N], k \neq i$

ZAKLJUČAK

Kvantificiranje sličnosti jedan od temeljnih zadataka strojnog učenja

Metrička ugrađivanja primjenjujemo kad klasifikacija nije praktična

- broj razreda prevelik ili unaprijed nepoznat
- malo ili nimalo označenih podataka za učenje

Danas metrička ugrađivanja tipično ostvarujemo dubokim modelima koje učimo s prikladnim kontrastnim gubitcima.

Primjene:

- praćenje osoba, verifikacija lica, stereo
- učenje s malenim brojem označenih primjeraka (few-shot)
- samonadzirano učenje
- detekcija izvandistribucijskih podataka.