DUBOKO UČENJE

⇒ razvoj umjetne inteligencije:

a) ekspertni sustavi (isključivo pravila)

b) strojno učenje (učenje slobodnih parametara)

c) reprezentacijsko učenje (učenje značajki i vrijednosti parametara)

⇒ teškoća obrade često ovisi o reprezentaciji podataka:

- premalo značajki - nema dovoljno informacije
- previše značajki - opasnost od prenaučenosti

⇒ znamo da je teško izabrati reprezentaciju:

↳ kako problem prikazati u računalu?

↳ drugim riječima, **teško je odabrati značajke**

⇒ IDEJA DUBOKOG UČENJA:

↳ **istovremeno uči značajke i klasifikacijsko pravilo**

⇒ **plitki modeli**: (1990-2006)

↳ modeli strogog učenja (npr. SVM)

↳ optimizacija neke konveksne funkcije

↳ kako se optimiraju konveksne funkciju, imamo garanciju konvergencije (dobro svojstvo koje je usporilo popularnost dubokih modela)

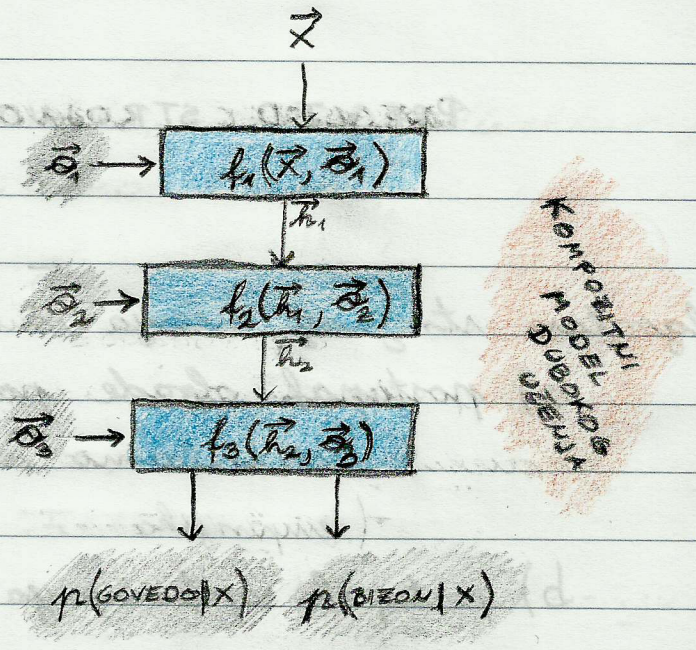
⇒ **duboki modeli:**

↳ nema garancije uspjeha

↳ nove tehnike modeliranja i učenja su dale da u praksi lokalni optimumi nisu veliki problem

↳ velki skupovi podataka

↳ potrebna velika procesna moć



h_1, h_2 ⇒ latentne značajke (one su slobodne, na njih ne djeluje gubitak)

⇒ **zašto su duboki modeli bili bolji od plitkih modela?**

↳ pa, teško je naći model koj dobro generalizira u velikom skupu plitkih modela

↳ tu nam pomaže **kompozitna struktura** dubokih modela jer svaki dio dubokog modela može naučiti nešto (npr. jedan dio uči rogove, drugi oči, treći noge, ...)

PREGLED STROJNOG UČENJA

⇒ algoritam strojnog učenja obuhvaća dva postupka:

a) postupak obrade podataka (MODEL)

↳ trebamo znati mjeriti performanse modela

(empirijska i generalizacijska pogreška)

b) postupak optimiranja slobodnih parametara (METODA)

⇒ algoritam strojnog učenja definiran je:

a) MODELOM

b) GUBITKOM

c) METODOM OPTIMIZACIJE

⇒ s obzirom na dostupnost podataka na učenje, razlikujemo:

a) nadzirano učenje

↳ tipični zadaci: klasifikacija, regresija

b) nenadzirano učenje

↳ tipični zadaci: grupiranje, smanjenje dimenzionalnosti, procjena gustoće

c) podržano učenje

⇒ specifičnost strojnog učenja u odnosu na ostale optimizacijske postupke je da performansu optimiramo posredno

(optimizacijska metoda ne vidi gubitak na ispitnom skupu)

⇒ često ne optimiramo empirijsku pogrešku direktno jer ona nije uvijek derivabilna (npr. kod klasifikacije)

↳ stoga, optimiramo gubitak koji je u nekoj vezi s empirijskom pogreškom

⇒ DETALJNIJE O MODELU:

- **kapacitet**: opisuje sposobnost prilagođavanju podacima
  - ↳ najčešće proporcionalan broju stupnjeva slobode
  - ↳ može se mjeriti (VC dimenzija), ali su te mjere bespomoćne za velik broj dimenzija
  - ↳ prisjeti se pojmova i znoj ih objasniti:
    - a) PODNAUČENOST (nedovoljan kapacitet)
    - b) PRENAUČENOST (prevelik kapacitet)
  - ↳ u praksi, radi ovako:
    - nađi optimalni model bez regularizacije za „train" skup podataka za učenje te ga onda pritegni regularizacijom
- **regularizacijske metode**: načini na koje u model ugrađujemo preferenciju za jednostavnost
  - ↳ regularizacijom nastojimo poboljšati generalizaciju
  - ↳ regularizacija smanjuje varijancu i povećava pristranost (to vodi k jednostavnijim modelima)

↓

pogledaj varijancu i pristranost te njihov utjecaj na generalizacijsku pogrešku na slajdovima!

(četri slike s izohipsama)

LOGISTIČKA REGRESIJA

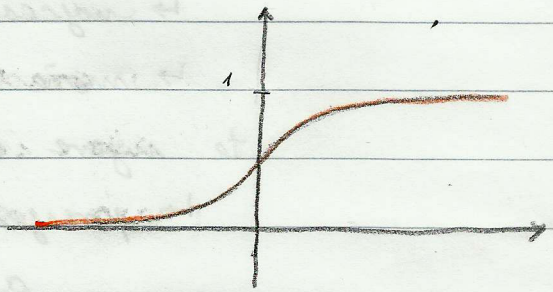
⇒ model koj služi za klasifikaciju (ne regresiju)

⇒ **izlaz modela:** (model)

$$\hat{Y}_i = \sigma(\vec{w}^T \vec{X}_i + b)$$

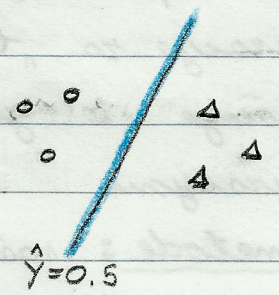
$$\sigma(\lambda) = \frac{1}{1 + e^{-\lambda}}$$

$\vec{w}^T \vec{X}_i + b$ ⇒ klasifikacijska mjera (što je to veće, to smo sigurniji da podatak ima klasu Y_i)



⇒ neka imamo skup podataka za učenje $D = \{(X, Y)\}$ s

- na sredini između razreda, model daje 0,5



⇒ **gubitak:**

$$\mathcal{L}(\vec{w}, b | D) = \sum_i - \log P(Y = y_i | \vec{X}_i, \vec{w}, b)$$

⇒ **optimizacijski postupak:** $\theta_{i+1} = \theta_i - \delta_i \cdot \frac{\partial \mathcal{L}}{\partial \theta_i}$

⇒ zadatak laboratorijske vježbe:

↳ treba implementirati logističku regresiju

↳ napomena: oznaka '_' je zapravo '^' u programu

↳ pseudokod:

```
X, Y_ = create-dataset()
alg = LogReg()
for i in range(nepochs):
    Y = alg.fwd(X)
    dW, db = alg.bwd(Y, Y_)   # backprop!
    # grad_check(X, Y, Y_, dW, db)
    alg.gd(dW, db)            # ažuriraj parametre
    alg.report()              # ispiši trenutni gubitak
```

⇒ VIŠEKLASNA LOGISTIČKA REGRESIJA:

↳ model:

$$P(Y = c_j \mid \vec{x}) = softmax(j, W\vec{x} + \vec{b})$$

$$softmax(j, \vec{s}) = \frac{e^{s_j}}{\sum_k e^{s_k}}$$

↳ gubitak:

$$\mathcal{L}(W, \vec{b} \mid D) = \sum_i -\log P(Y = y_i \mid \vec{x}_i)$$

BINARNA LOGISTIČKA REGRESIJA

⇒ **MODEL:**

$$\hat{Y}_i = \sigma(s_i)$$
$$s_i = \vec{w}^T \vec{x}_i + b$$

⇒ **GUBITAK:**

$$\mathcal{L} = -\sum_{i=1}^{N} \log \underbrace{P(Y = y_i \mid \vec{x}_i)}$$

vjerojatnost da je predikcija modela jednaka traženoj oznaci

↳ primjetit da je gubitak uvijek pozitivan jer je $0 \leq P(Y = y_i \mid \vec{x}_i) \leq 1$, pa je $0 \geq \log P(Y = y_i \mid \vec{x}_i) > -\infty$

⇒ **OPTIMIZACIJSKI POSTUPAK:**

$$\mathcal{L}(\vec{w}, b) = \mathcal{L}_p(\vec{w}, b) + \mathcal{L}_n(\vec{w}, b)$$

gubitak pozitivnih — gubitak negativnih

↳ ovaka ju olakšamo jer je:

$$P(Y = y_i \mid \vec{x}_i) = \begin{cases} \hat{y}_i, & y_i = 1 \\ 1 - \hat{y}_i, & y_i = 0 \end{cases}$$

∴ pa, izvedimo gradijentni spust za logističku regresiju ova razreda

↳ pogledajmo gubitak pozitivnih:

$$\mathcal{L}_p = -\sum_{i=1}^{N_p} \log \sigma(s_i) = -\sum_{i=1}^{N_p} \log\left(\frac{1}{1+e^{-s_i}}\right) =$$

$$= -\sum_{i=1}^{N_p} \log\left(\frac{e^{s_i}}{e^{s_i}+1}\right) = \sum_{i=1}^{N_p} \left[\log(e^{s_i}+1) - \log(e^{s_i})\right]$$

$$= \sum_{i=1}^{N_p} \left[\log(e^{s_i}+1) - e^{s_i}\right]$$

↳ sada, odredimo gubitak negativnih:

$$\mathcal{L}_N = -\sum_{i=1}^{N_N} \log\left(1 - \sigma(s_i)\right) = -\sum_{i=1}^{N_N} \log\left(\frac{1+e^{-s_i}}{1+e^{-s_i}} - \frac{1}{1+e^{-s_i}}\right) =$$

$$= -\sum_{i=1}^{N_N} \log\left(\frac{e^{-s_i}}{1+e^{-s_i}}\right) = -\sum_{i=1}^{N_N} \log\left(\frac{1}{1+e^{s_i}}\right) = \sum_{i=1}^{N_N} \log\left(1+e^{s_i}\right)$$

↳ sada odredimo gradijente:

$$\frac{\partial \mathcal{L}}{\partial \vec{w}} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}}{\partial s_i} \cdot \frac{\partial s_i}{\partial \vec{w}}$$

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}}{\partial s_i} \cdot \frac{\partial s_i}{\partial b}$$

→ rečeno vektor jer je funkcija skalarna (pa je Jakobijan 1×D)

↓

ponovlja se, pa kako bi uštedili memoriju, izračunamo ovo samo jednom!

$$\frac{\partial \mathcal{L}}{\partial s_i} = \begin{cases} \frac{\partial \mathcal{L}_p}{\partial s_i}, & y=1 \\ \frac{\partial \mathcal{L}_N}{\partial s_i}, & y=0 \end{cases}$$

↳ gradijent gubitka pozitivnih:

$$\frac{\partial \mathcal{L}_P}{\partial s_i} = \frac{\partial(\log(1+e^{s_i}) - s_i)}{\partial s_i} = \frac{1}{1+e^{s_i}} \cdot e^{s_i} - 1 = \hat{y}_i - 1$$

↳ gradijent gub-itka negativnih:

$$\frac{\partial \mathcal{L}_N}{\partial s_i} = \hat{y}_i$$

↳ sada još trebamo $\frac{\partial s_i}{\partial \vec{w}}$ i $\frac{\partial s_i}{\partial b}$:

$$\frac{\partial s_i}{\partial \vec{w}} = \vec{x}_i^T \qquad \frac{\partial s_i}{\partial b} = 1$$

↳ to je to! Sada imamo izveden postupak učenja gradijentnim spustom!

- NAPOMENA: sada još zbog optimizacije računanja gradijenata trebamo malo ovo srediti i staviti bijepo u matricu da s jednim matričnim množenjem izračunamo sve gradijente za sve podatke jednim matričnim množenjem

⇒ OPTIMIZACIJA POSTUPKA OPTIMIZACIJE:

↳ rapišimo $\frac{\partial \mathcal{L}}{\partial s_i}$ ovako:

$$\frac{\partial \mathcal{L}}{\partial s_i} = \hat{y}_i - [\![y_i = 1]\!]$$

↳ stvorimo derivacije u vektor:

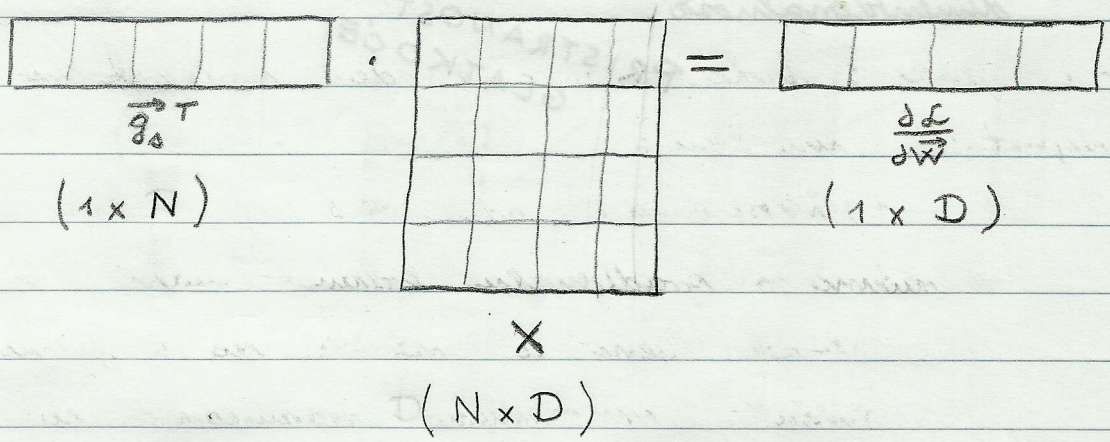
$$\vec{g}_s = \left[\frac{\partial \mathcal{L}}{\partial s_1} \cdots \frac{\partial \mathcal{L}}{\partial s_N} \right]^T$$

↳ primjetimo sada da vrijedi sljedeće:

$$\frac{\partial \mathcal{L}}{\partial \vec{W}} = \sum \frac{\partial \mathcal{L}}{\partial s_i} \cdot \frac{\partial s_i}{\partial \vec{W}} = \vec{g}_s^{\,T} \cdot X$$

↑ OBIČNO STAVI $\frac{1}{N}$ OVDJE

⇓ matrica svih danih primjera!



↳ na pomak parametra „b":

$$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}}{\partial s_i} \cdot 1 = \sum_{i=1}^{N} \vec{g}_s$$

↑ OBIČNO STAVI $\frac{1}{N}$ OVDJE

(ZA VIŠEBRAZREDNU IZVEDI SAM!)

IZAZOVI STROJNOG UČENJA

⇒ bavimo se problemima UI - ravnine
↳ dimenzionalnost su oko 10^5 i te zbog toga trebamo jako puno primjera za učenje

⇒ pristranost kakvu imaju k-NN, jezgrene metode su pristranosti o glatkoći i nisu dobri za ogromne količine podataka
↳ vidi kakva je ta pristranost na slajdovima i zašto ona nije dobra za takve visoke dimenzionalnosti

PRISTRANOST GLATKOĆE

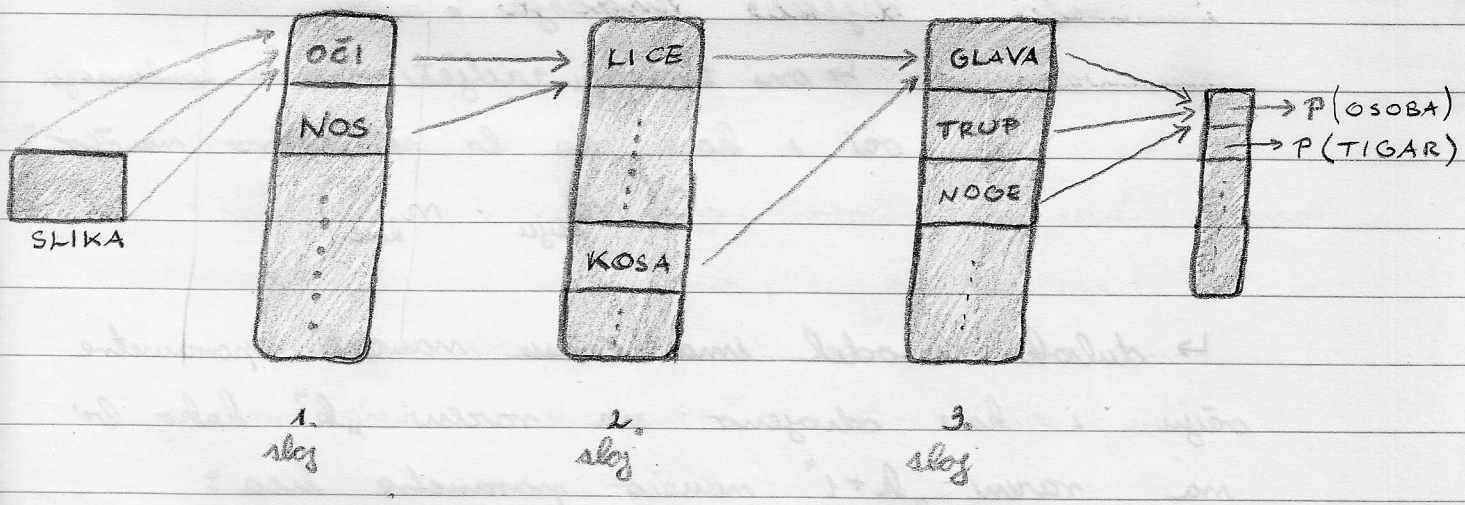
DUBOKI (KOMPOZICIJSKI) MODELI

⇒ u dubokim modelima, nalazimo „kompozicijsku" prostranost:

↳ podaci su generirani kompozicijom značajki kroz više razina hijerarhije

⇓

osoba ima glavu,
glava ima lice,
lice ima oči,
oko ima šarenicu



⇒ **pretpostavka dubokih modela:**

- stvari u prirodi su **artikulirane** (složene su kompozicijski)

⇒ intuitivno objašnjenje dubokih modela:

↳ za razliku od plitkih modela strojnog učenja, duboki modeli mogu naučiti nešto na jednoj osobi što će pomoći u raspoznavanju druge osobe

- npr. osoba s plavim očima i crvenom kosom te osoba s smeđim očima i crvenom kosom (il čak nekom narančastom kosom)

⇓

za plitke modele, te osobe su nešto potpuno različito jer nema kompozitne prestranosti

↳ ti modeli ne mogu učiti dijeliće značajki!

↳ oni trebaju vidjeti svaku kombinaciju oči i kose da bi se dobro naučio:

$$O(n_{očiju} \cdot n_{kose})$$

↳ duboki model ima šansu naučiti parametre očiju i kose odvojeno na razini „k" kako bi na razini „k+1" naučio parametre lica:

$$O(n_{očiju}) + O(n_{kose}) + O(n_{lica})$$

↳ kako je gornji zbroj tipično manji od onog gore umnoška, zaključujemo da bi duboki modeli trebali bolje kapitalizirati podatke (bolje uče iz manje količine podataka)

⇒ **dodatno objašnjenje:**

↳ kompozitni modeli pretpostavljaju da pojave iz stvarnog svijeta leže u puno manjoj dimenzionalnosti od one koju su ovi sposobni opisati

– ovo se naziva MNOGOSTRUKOST

↳ npr. „smislene" slike 200 × 200 su sigurno jako mali podskup svih mogućih slika te određene veličine

↳ stoga, ovi načinimo veliki model i od njega želimo da se on nauči manju dimenzionalnost vrlo dobro – i to upravo onu u kojoj leže dani podaci

⇒ **SAŽETAK:**

• duboki modeli mogu raditi s jako puno značajki (milijarda značajki nije problem), ali oni dobro generaliziraju ako su dani podaci artikulirani (kompozitne strukture) //

DUBOKE UNAPRIJEDNE MREŽE

⇒ Unaprijedna mreža il višeslojni perceptron:
- osnovni duboki model
- moramo ga dobro poznavati jako se puno koristi u praksi

⇒ cilj je aproksimirati $\vec{y} = f^*(\vec{x})$ parametarskim modelom:

$$\hat{\vec{y}} = f(\vec{x}, \Theta)$$

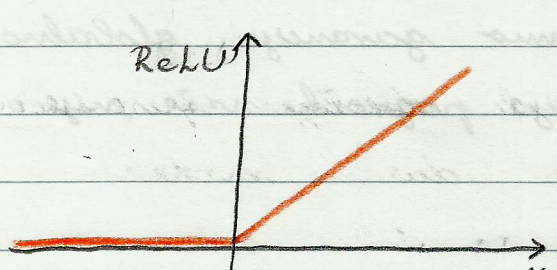
⇒ mrežu ćemo zapisivati pomoću ugniježđenih skrivenih slojeva što su zapravo izlazi skrivenih slojeva
↳ oznake: $\vec{h}_L$, $\vec{h}_{L-1}$, ...
↳ one su latentne varijable jer na njih gubitak ne djeluje (one se mijenjaju proizvoljno)

⇒ hiper-parametri:
- broj slojeva: L
- širina l-tog sloja: $\vec{h}^l \in R^{d_l}$

⇒ propagacija kroz mrežu:

$$h_i^l = f(\vec{h}^{l-1}) = g(\underbrace{\vec{w}_{l,i}^T \vec{h}^{l-1} + b_{l,i}}_{a_i^l})$$

⇒ u ranim danima „g" je bio sigmoida, no danas se pokazalo da je bolje koristiti nerazičnu funkciju, pa obično koristimo ReLU funkciju:



⇒ često želimo preslikati značajke u viši prostor zbog linearne razdvojivosti ∇

⇒ u dubokom učenju, mi učimo i značajke (tj. parametre funkcije preslikavanja):

↳ problem s time je što gubimo jednostavnost učenja modela jer pogreška više nije konveksna

(na slajdovima je prikazano kako se odredituje dimenzionalnost u kojoj je XOR problem linearno razdvojiv)

↳ linearan model ima dobro svojstvo da daje rješenje u zatvorenoj formi, al nije dovoljnog kapaciteta za XOR problem

↳ stoga, dodaje se nelinearan sloj s aktivacijom: $ReLU(x) = \max(0, x)$

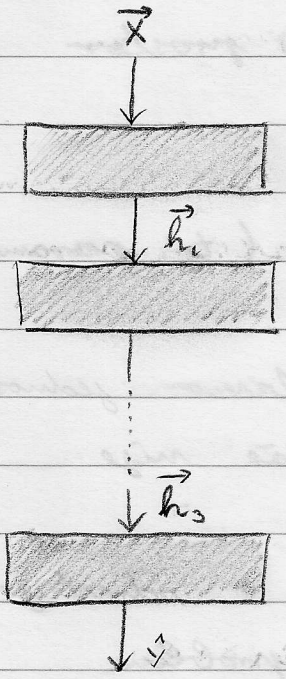
(na vektor se primjenjuje „element-wise": $g(\vec{x})_i = g(x_i)$)

⇒ dodavanjem ReLU sloja, problem je postao rješiv, no to smo platili ovako:

a) nemamo rješenje u zatvorenoj formi

b) nemamo garanciju globalnog optimuma jer funkcija pogreške više nije konveksna

ZAD: Odredi jednadžbe mreže 2, 5, 10, 5, 2 i broj parametara!



$$\vec{h}_1 = f(W_1 \vec{x} + \vec{b}_1)$$

$$\vec{h}_2 = f(W_2 \vec{h}_1 + \vec{b}_2)$$

$$\vec{h}_3 = f(W_3 \vec{h}_2 + \vec{b}_3)$$

⇒ prvi broj ovdje tretiramo kao broj neurona u ulaznom sloju (dimenzionalnost vektora $\vec{x}$)

sloj	#
1	15
2	5·10 + 10
3	10·5 + 5
4	5·2 + 2

GUBITAK DUBOKIH MODELA

⇒ **gubitak 0-1:**

$$\mathcal{L}_{01} = \begin{cases} 0, & \text{ako } \hat{Y} = Y \\ 1, & \text{inače} \end{cases}$$

↳ nije derivabilna jer malom promjenom parametara, gubitak 0-1 se vjerojatno ne mijenja, pa ne znamo u kojem smjeru mijenjati parametre

↳ može se optimirati, al je to kombinatorna optimizacija što je pretežko na velikim modelima koji su nama zanimljivi

⇒ **gubitak kao negativna log-izglednost:**

$$\mathcal{L}(Y, \hat{Y}) = -\sum \log P(\hat{Y}_i = Y_i \mid f(\vec{X}_i, \Theta))$$

↳ ovo je bolje jer sada na izlazu imamo vjerojatnost dobre oznake, pa što je ta vjerojatnost veća to je gubitak manji

↳ sada ćemo moći ovo optimizirati gradijentnim spustom jer malom promjenom parametra dobivamo varijabilan gubitak ☺

⇒ kako bi mogli dobiti vjerojatnost $P(\hat{Y}=y \mid f(\vec{x} \mid \Theta))$, trebamo pretpostaviti razdiobe:

a) za regresiju je normalna (Gaussova) razdioba

b) za klasifikaciju je kategorička (generalizirana Bernoullijeva razdioba - „multinullijeva") razdioba

⇒ UNAKRSNA ENTROPIJA:

↳ mjera razlike između distribucije stvarnih oznaka Y i distribucije našeg modela $P(\hat{Y}=y \mid f(\vec{x} \mid \Theta))$

$$H_i(P_i, Q_i) = -E_{P_i} \log Q_i = -\sum^{C} P(c) \log Q_i(c)$$

$$H_i(P_i, Q_i) = -\log Q(c_i)$$

⇒ kako bi dobili vjerojatnosni gubitak, trebamo natjerati naš model da on generira vjerojatnosti na izlazu i to tako da se one zbrajaju u 1

↳ izlaz mora biti kategorička razdioba preko C razreda!

$$P(c) = \begin{cases} 0, & c = c_i \\ 1, & c = c_i \end{cases}$$

ovo je ako imamo „one-hot-coding"

3 ← 0 0 0 1 0 0 0 0 0 0

0 1 2 3 4 5 6 7 8 9

na slici je znamenka 3

PROBABILISTIČKI IZLAZ DUBOKIH MODELA

⇒ na izlaz modela stavljamo **softmax** iz dva razloga:

a) jer ima „dobar" gradijent

b) može se interpretirati prema Bayesovom pravilu pa nas štiti teorija vjerojatnosti

(vidi slajd 26 za dodatno objašnjenje)

⇒ sa softmax-om na izlazu, dobivamo:

$$L(y, \hat{y}) = -\log \text{softmax}(\vec{z})_y = \log \sum_j \exp(\vec{z}_j) - \vec{z}_y$$

↳ ovaj gubitak se ponaša kao gornja granica gubitka 0-1, pa smo uspješno zamijenili intuitivni gubitak klasifikacije (gubitak 0-1) sa boljim, probabilističkim gubitkom //

⇒ svojstva softmax-a:

• primjet da on zapravo vraća položaj najveće vjerojatnosti, pa bi se zapravo trebao zvat:

soft-arg-max

• ako baš želimo najveću vrijednost u determinističkom načinu, često koristimo trik:

$$\text{softmax}(\vec{z}) \cdot \vec{z}^T$$

⇒ **stabilna softmax:**

↳ koristi svojstvo da dodavanjem konstante argumentu ne mijenjamo distribuciju na izlazu

↳ ovo se često koristi kako bi se spriječio overflow zbog velikog broja u eksponenti

* stoga napišemo računamo:

$$\text{softmax}(\vec{z}) \Rightarrow \text{softmax}(\vec{z} - \max(\vec{z}))$$

⇒ primjer da je negativna log-vjerodostojnost super izbor za gubitak bez obzira na vrijednost koja se nalazi ispod

⇒ s druge strane, kvadratni gubitak je loš iz dva razloga:

a) kažnjavamo različito 0.33 (0.33) 0.33 i 0.66 (0.33) 0 što ne želimo

b) na slajdu 31, vidimo da je gradijent jako mali ako je z velik ili malen zbog rasćenja sigmoide, bez obzira je li predikcija dobra ili ne, pa mreža ne može ništa naučiti na takvom primjeru (to je zbog člana umnoška: $(1 - \sigma(z))\sigma(z)$)

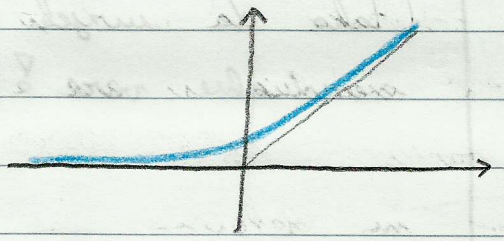
ŽGLOBNICA KAO AKTIVACIJSKA FUNKCIJA

⇒ funkcija ReLU je super, al ima nedostataka:

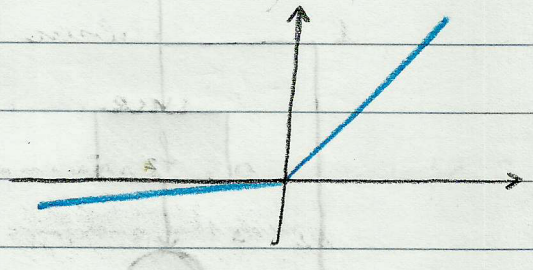
a) moramo gradijent za nulu poredno definirati

b) u negativnom stanju ne propušta signal unaprijed i gradijent unatrag, pa ako su sve aktivacije po primjerima negativne (rijetko u praksi), tada više ne možemo učiti

↳ postoje generalizacije:



Soft Plus: $g(x) = \log(1+e^x)$

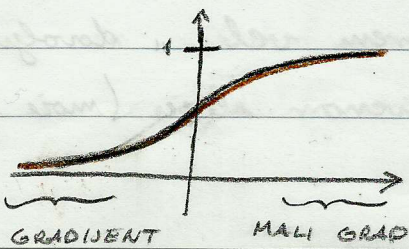


Leaky ReLU: $\max(x, 0) + \lambda \max(0, x)$

↳ no, generalizacije usporavaju primjenu aktivacije, pa se najčešće koristi **normalizacija po grupi** (vid slajdove)

c) srednja vrijednost aktivacij po grupi nije jednaka nuli

⇒ ReLU je bolja od sigmoide u unaprijednim modelima jer sigmoida ulrija gradijente na krajevima (kada je aktivacija velika ili mala):

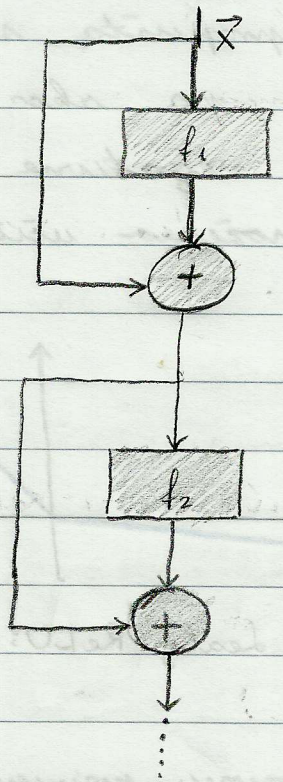


STRUKTURE MREŽE

⇒ najjednostavnija mreža: lanac

↳ danas je default preskočna i rezidualna veza!:

↳ rezidualna veza:



ideja je da gradijent što bolje dolazi do ranih slojeva u jako dubokim modelima (tako da uvijek koristi rezidualne veze!)

⇒ TEOREM O UNIVERZALNOJ APROKSIMACIJI:

• rašta uopće duboki modeli kada ovaj teorem tvrdi da je jedan skriveni sloj dovoljan za „gotovo sve"

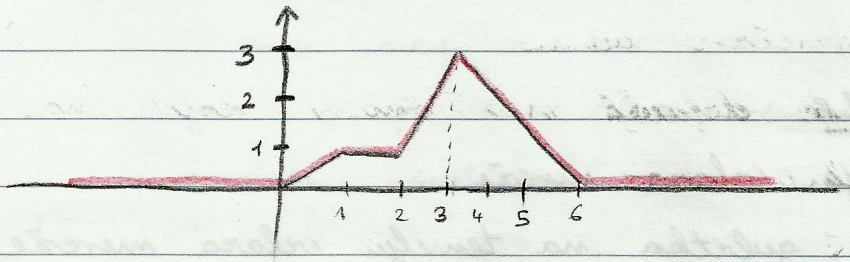
↳ pa, nekoliko razloga:

a) ne znamo kakvu funkciju aproksimiramo

b) teorem ništa ne govori o generalizaciji

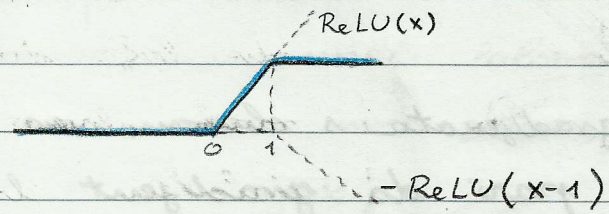
c) teorem vel „dovoljno mnogo" neurona u skrivenom sloju (može biti neizvodivo puno)

ZAD: Konstruirati dvorazinski model koji koristi linearne transformacije plus zglobnicu da bi aproksimirao sljedeću funkciju:



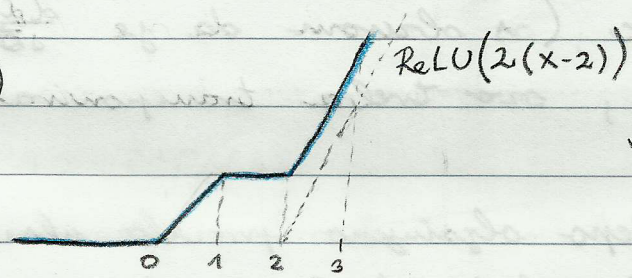
↳ ovo rješavamo tako da slažemo „dio-po-dio":

a)



$$Y_a(x) = ReLU(x) - ReLU(x-1)$$

b)



$$Y_b(x) = Y_a(x) + ReLU(2(x-2))$$

c) analogno (vidi na slajdu)

↳ ovaj zadatak pokazuje teorem o univercalnoj aproksimaciji:

- sve možemo prikazati linearnim funkcijama uz proizvoljno malu grešku, ako tih funkcija ima dovoljno puno
- no, vidi grafičko objašnjenje na slajdu zašto mi vjerujemo da duboki model bolje rade (isti broj aktivacija, plitki vs. duboki, crtkano je duboki)

NE ZNAMO SA SIGURNOŠĆU ZAŠTO ONI BOLJE RADE

(duboki model su na granici spoznaje (trenutno))

RAČUNANJE GRADIJENTA GUBITKA PO PARAMETRIMA

⇒ **prolaz unaprijed:**

- računanje izlaza mreže
- računanje gubitka na temelju izlaza mreže

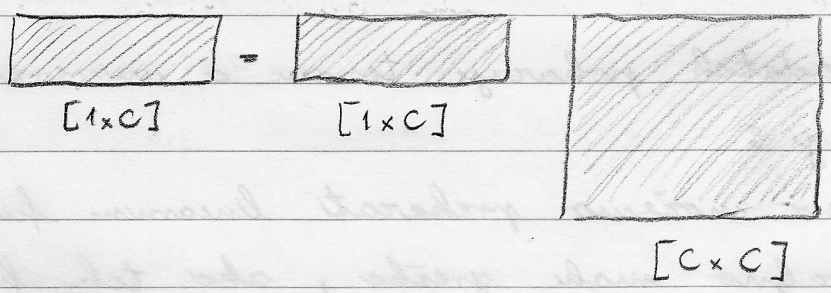
⇒ **prolaz unatrag:**

- računanje gradijenata s obzirom na parametre
- NAPOMENA: par. da gradijent bude vektor stupac (s obzirom da je $\frac{d\Delta i}{d\vec{B}}$ vektor redak, ovo treba transponirati)

⇒ na slajdu 51 je lijepo objašnjeno pravilo ulančavanja s derivacijama vektorskih funkcija:

↳ to valja i vizualizirati:

$$\frac{\partial \vec{Z}}{\partial \vec{X}} = \frac{\partial \vec{Z}}{\partial \vec{Y}} \cdot \frac{\partial \vec{Y}}{\partial \vec{X}}$$



↳ razmislimo da je $\frac{\partial \vec{Y}}{\partial \vec{X}}$ zapravo $\frac{\partial \vec{\Delta} i}{\partial \vec{B}}$, te tada dobivamo jedinstvenu matricu jer je:

$$\Delta_{ij} = \vec{W}_{j:}^{T} \vec{h}_{(l-1)i} + b_j$$

↳ e sad, kada imamo $\frac{\partial \vec{Y}}{\partial X}$ (gdje je X matrica), onda je ovo tenzor trećeg reda

- to nastojimo izbjeći kada je god takvo nešto moguće, a pokazuje se da je moguće za dobar dio formulacija naših gubitaka

↓

primjer za to je onaj trik iz multe laboratorijske vježbe za multinomijalnu logističku regresiju

⇒ BACKPROPAGATION:

↳ u svakom koraku nam treba:

- parcijalno gubitka prethodnog sloja
- parcijalno gubitka za sljedeći sloj

↳ u praksi, izračunaj parcijalno gubitka po svim slojevima i tada radi backprop (to je najefikasnije)

RAČUNSKI GRAFOVI

⇒ danas, vrlo rijetko radimo gradijente na ruke (jer postoje programski računski okviri koj mogu računat gradijente ako naš problem izrazimo RAČUNSKIM GRAFOM

⇒ računski graf radajemo programskim kôdom
↳ taj kôd možeš vidjet na na slajdu 57. i 58

BACK PROPAGATION

⇒ algoritam za računanje gradijenata u dubokom modelu

⇒ **uvod:**

↳ pretpostavimo da je na ulazu samo jedan podatak (radi jednostavnosti)

↳ pretpostavimo klasifikaciju, pa je gubitak:

$$J = \mathcal{L}_{MLE}(\vec{\hat{y}}, y) + \lambda \Omega(\vec{\Theta})$$

$$\mathcal{L}_{MLE} = -\log(\hat{y}_y) \rightarrow \text{vjerojatnost točne klase } P(\hat{y} = y_i | \vec{x}_i)$$

↳ izlaz modela i oznake:

$$\vec{\hat{y}} = \vec{h}^{(l)} = \text{softmax}(\vec{s}^{(l)})$$

$$\vec{s}^{(l)} = W^{(l)} \vec{h}^{(l-1)} + \vec{b}^{(l)}$$

$$h_j^{(l-1)} = f(s_j^{(l-1)})$$

ALGORITAM:

$$\vec{g}^T \leftarrow \frac{\partial \mathcal{L}}{\partial \vec{\hat{y}}}$$

// samo jedan element vektora $\neq 0$

za $k = l$:

$$\vec{g}^T \leftarrow \frac{\partial \mathcal{L}}{\partial \vec{s}^{(l)}} = \frac{\partial \mathcal{L}}{\partial \hat{y}_y} \cdot \frac{\partial \hat{y}_y}{\partial \vec{s}^{(l)}} = \vec{\hat{y}} - \vec{y}_{OH}$$

$$\nabla_{b^{(k)}} J \leftarrow \vec{g} + \lambda \nabla_{b^{(k)}} \Omega(\vec{\Theta})$$

// obično se regularizira (uvedi sam)

$$\nabla_{W^{(k)}} J \leftarrow \frac{\partial \mathcal{L}}{\partial W^{(k)}} = \left[\frac{\partial \mathcal{L}}{\partial W_{ru}^{(k)}}\right]$$

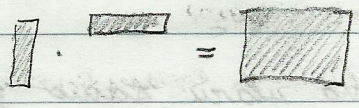
matrica, pa odredimo gradijent samo za jedan redak (lakše je tako):

$$\frac{\partial \mathcal{L}}{\partial \vec{W}_{T:}^{(k)}} = \sum_v \frac{\partial \mathcal{L}}{\partial s_v^{(k)}} \cdot \frac{\partial s_v^{(k)}}{\partial \vec{W}_{T:}} = \frac{\partial \mathcal{L}}{\partial s_r^{(k)}} \cdot \frac{\partial s_r^{(k)}}{\partial \vec{W}_{r:}^{(k)}} = g_r \cdot \vec{h}^{(k-1)^T}$$

pojednostavi se jer samo neki ostaje, pa se suma pojednostavi ☺

↳ sada se lako vidi da ove gradijente po retcima možemo odjednom izračunati tako da ove retke naslažemo (jer imamo vanjski produkt):

$$\frac{\partial \mathcal{L}}{\partial W^{(k)}} = \vec{g} \cdot \vec{h}^{(k-1)^T}$$



BITNO! SHVATI OVO!

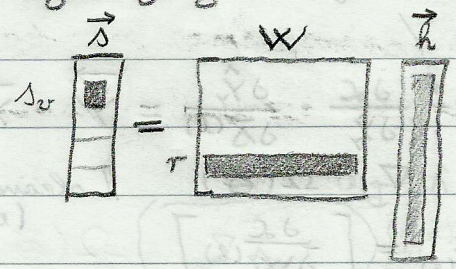
$$\frac{\partial \mathcal{L}}{\partial W^{(k)}} = \vec{g} \cdot \vec{h}^{(k-1)^T}$$

$$\vec{g}^T \leftarrow \frac{\partial \mathcal{L}}{\partial \vec{h}^{(k-1)}} = \frac{\partial \mathcal{L}}{\partial \vec{s}^{(k)}} \cdot \frac{\partial \vec{s}^{(k)}}{\partial \vec{h}^{(k-1)}} = \vec{g}^T W^{(k)}$$ // update za sljedeći sloj

⇒ NAPOMENA: ovo je izvod za radnj sloj (u knjizi i na slajdu je to izostavljeno - stavlja radnj sloj u isti koš što nije dobro)

⇒ NAPOMENA: objašnjenje nestanka sume:

$$\frac{\partial s_v^{(k)}}{\partial \vec{W}_{r:}^{(k)}}$$



↳ s_v ne ovisi o $\vec{W}_{r:}$ jer je $s_v = \vec{W}_{v:}^T \cdot \vec{h}$

↳ ovo je gradijent vezan uz „r"-tu klasu

⇒ NAPOMENA: da imamo više podataka, „$\vec{g}$" bi bila matrica s $\vec{g}$ u retcima, te bi $\vec{h}$ bila matrica (po stupcima slagana)

KONVOLUCIJSKI DUBOKI MODELI

⇒ potpuno povezani duboki modeli nemaju neki osjećaj o slici (npr. lokalnost značajki)

↳ primjerice, da se slike isprermiješaju na potpuno isti način, potpuno povezani duboki model bi svejedno naučio nešto; još uvijek dovoljno dobro (vjerojatno jednako dobro)

↳ to je zato jer on uči veze između svaka dva piksela slike, a nemaju ideju o lokalnost značajki

⇓

stoga, modimo modele koji to imaju:

KONVOLUCIJSKI MODELI

⇒ konvolucijski modeli mogu naučiti značajke po lokalnosti (npr. rogari gorcda se vjerojatno ne nalaze u donjem lijevom kutu slike već su vjerojatno negdje u gornjoj polovici)

⇒ KONVOLUCIJSKI MODELI:

- specijalizirani za podatke s topologijom rešetke (npr. govor, slika, prijenos podataka, 3D model, ...)
- ima najmanje jedan konvolucijski sloj umjesto potpuno povezanog sloja

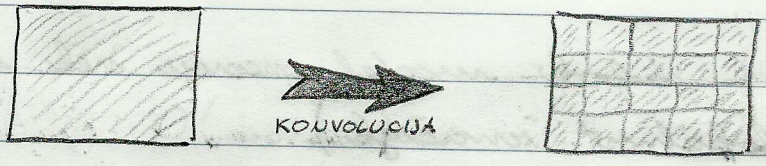
↓

tada je to konvolucijski model

⇒ **konvolucija**:

↳ engl. „extremelly complex"

↳ operacija koja modelira lokalne značajke



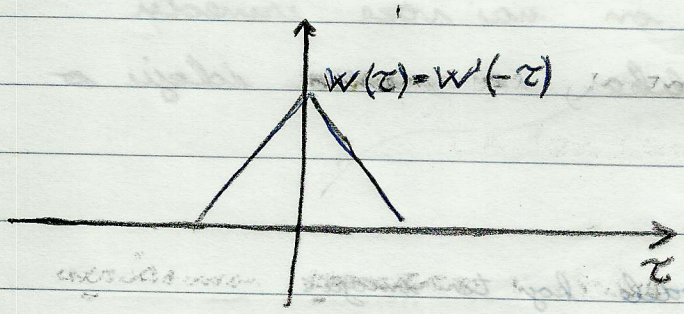
slika mapa značajki

$$h(t) = \int_{-\infty}^{\infty} x(\tau)\, w(t-\tau)\, d\tau$$

$x \Rightarrow$ ulaz

$w \Rightarrow$ jezgra

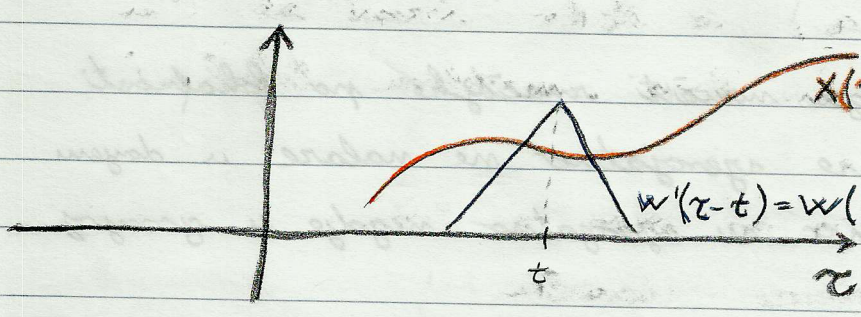
$h \Rightarrow$ mapa značajki



$w'(t) = w(-t)$

$w(t-\tau) = w'(\tau - t)$

$w'(\tau - t)$ je samo pomaknuti $w'(-\tau)$



ovo je sada lako: izmnoži i integriraj ☺

↳ ovo je intuitivno objašnjenje 1D konvolucije

⇒ kraća oznaka: $h(t) = (x * w)(t)$

⇒ no, računala su diskretna i ograničena, pa mi trebamo ograničenu sumu umjesto beskonačnih integrala (matematička konvolucija nam nije izvediva):

$$\boxed{S(i,j) = (I * K)(i,j) = \sum_{m=m_{MIN}}^{m_{MAX}} \sum_{n=n_{min}}^{n_{MAX}} I(i+m, j+n)\, K(m,n)}$$

↳ iskonstil smo komutativnost da skratimo petlju jer je jezgra K puno manjih dimenzija od same slike

↳ ovo je dvodimenzionalna konvolucija (naziva se KROS-KORELACIJA)

kros-korelacija nauči iste vrijednosti samo na obrnutim mjestima

⇒ IDEJA: jezgrom K šeći po slici i računaj konvoluciju (koju ćemo računati kao skalarni produkt jer je suma ograničena) te tako obradi sliku u jednostavniju sliku koja „modelira lokalnost"

⇓

motivacija za ovo je teorija obrade signala

(VIDI SLAJD 12.)

⇒ veza je linearna, pa svaki konvolucijski model možemo prikazati potpuno povezanim modelom (al uz PUNO, PUNO više značajki)

VIDI SLAJD 15 i 16 za objašnjenje

⇒ prostranost koju konvolucija uvodi je dobra jer:

a) modelira lokalnost značajki

b) drastično smanjuje broj parametara - rijetkost

(lakše učenje, manje potrebnih podataka)

⇒ umjesto $O(m \cdot n)$ imamo $O(k \cdot n)$ gdje je „k" širina jezgre //

$k \ll m$

⇒ UOČI: jezgra smanjuje receptivno polje značajke, no ako male jezgre kaskadiramo u više slojeva, možemo proširit to receptivno polje

(VIZUALIZACIJA NA SLAJDU 20.)

⇒ dodatna poveranost: dijeljenje težina (parametara) unutar značajki jezgre

manja prostorna, ali ista računska složenost!

↳ model je stoga još puno, puno manji:

- imamo samo „k" parametara za konvolucijski sloj (iste težine na cijelu sliku pri prolasku konvolucije)

⇒ EKVIVARIJANTNOST:

- pomak objekta na slici radi i pomak u mapi značajki

⇓

ekvivarijantnost na translaciju

(BITNO SVOJSTVO ZA SLIKE!)

(dobra prostranost)

⇒ no, konvolucija nije ekvivarijantna s obzirom na neke druge transformacije ulaza:
npr. rotacija, skaliranje

⇓

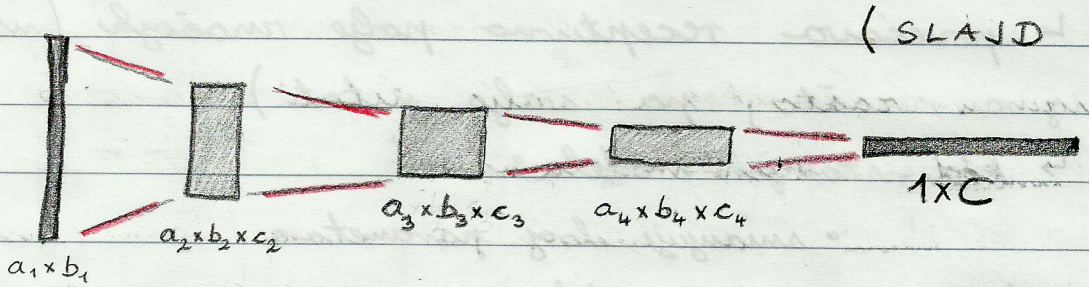
bitno je da si sjetan toga (na slikama gdje čovjek stoji ne možeš naučiti nešto o čovjeku koji leži ☹)

↳ bitno je detektirati prisutnost značajke, a ne točnu lokaciju ☺

⇒ SLOJ SAŽIMANJA ☺

- potreban jer npr. sliku pretvaramo u vektor

(SLAJD 24)



podaci bogati prostornom informacijom ⇒ podaci bogati semantičkom informacijom

- sažimanje se radi na različite načine:
  - maksimalna vrijednost
  - srednja vrijednost (rijeđe)
  - neka matematička norma
- veća regija sažimanja ⇒ veća invarijantnost (na veće pomake)

⇒ obično radimo sažimanje unutar neke mape značajki
↳ no, možemo raditi i sažimanje preko različitih mapa značajki (BITNO!)
• takvo sažimanje ide preko semantičke dimenzije (a ne preko prostora), pa možemo ostvariti ekvivarijantnost na rotaciju
↳ to efektivno govori da samo želimo lokalizirati pojavljivanje neke značajke, a ne njezin položaj

↳ može voditi podnaučenosti, jer smanjuje kapacitet modela (obično ne koristimo u modernim modelima)
NO, PAZI!

⇒ ponekad veličina sažimanja nije fiksna:
↳ omogućava procesiranje ulaza različitih veličina
↳ povećava receptivno polje značajki (nisam siguran zašto, pa valja pitati)
↳ kod starijih modela:
• smanjuje broj parametara
↳ kod modernih modela:
• isto smanjuje, ali ne drastično (kljućna je prostranost koja se uvodi)

⇒ za dobro ponavljanje i „fact-check", vidi slajd 32!

⇒ teško je smisliti nešto što bi uvelo ekvivarijantnost na rotaciju ili skaliranje, a da se pri tome ne izgubi puno na memoriji i na brzini učenja
↳ „vanilla" konvolucijski model super radi na puno primjera (teško se boriti protiv njega)

⇒ <u>ako je konvolucijski model podnaučen?</u>

↳ možda nisu slika lokalne interakcije važne - povećaj receptivna polja (obično tako da model produbiš)

↳ možda je problem u slogu sažimanja (vidi slajdove za razgovaranje problema)

⇒ nadopuna (engl. „padding"):

↳ daje neograničenu dubinu mreže (ovo se koristi)

↳ ber nje, reprezentacija se smanjuje s dubinom, pa sve ograničeno veličinom ulaza

⇒ konvolucija ne implementirana ni, već je ona implementirana na grafičkim procesorima:

↳ koristi CUDA nn knjižnicu ▽:

↳ POUKA: koristi Tensorflow ▽:

UČENJE KONVOLUCIJSKIH DUBOKIH MODELA

⇒ isto ćemo pogledati učenje prolazom unatrag:

↳ unaprijedni prolaz:

- ulaz: $\vec{x}$
- rezultat: $P(y \mid \vec{x})$

↳ unatražni prolaz:

- ulaz: željeni rezultat modela
- izlaz: gradijenti parametara sloja $\partial L / \partial \vec{\theta}_\ell$

⇒ dobivene gradijente $\partial L / \partial \vec{\theta}_\ell$ dalje rabimo optimizatorom (npr. gradijentni spust)

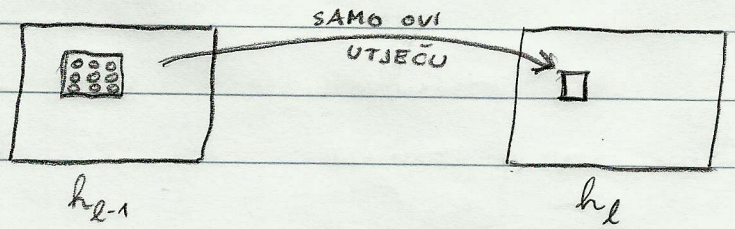
⇒ **dimenzionalnosti:** (posebno za potpuno povezane i posebno za konvolucijske) VIDI SLAJDOVE!

↳ $\partial L / \partial \vec{h}_\ell$ ⇒ isto kao $\vec{h}_\ell$ (samo transponirano)

↳ $\partial L / \partial W_\ell$ ⇒ isto kao W_ℓ

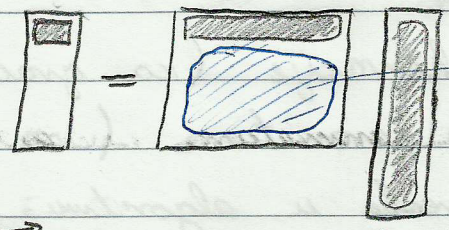
↳ $\partial L / \partial \mathrm{vec}(W_\ell)$ ⇒ vektorizirana matrica koja olakšava puno deriviranje (ulančavanje) - izbjegava tenzore viših redova!

↳ $\partial h_\ell / \partial h_{\ell-1}$ ⇒ tenzor četvrtog reda jer su oba tenzora drugog reda (rijedak tenzor zbog konvolucije) → ovo je za konvoluciju!



(za potpuno povezan sloj je drugačije)

↳ $\partial h_\ell / \partial W_\ell$ ⇒ u potpuno povezanom sloju, ovo je rijedak tenzor trećeg reda (to smo već pokazali, ali evo ponovno!)



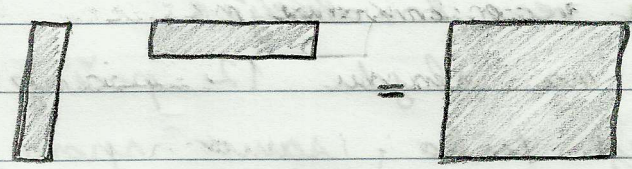
→ o ovima ovaj prvi element u $\vec{h}_\ell$ uopće ne ovisi (daje rijedak tenzor)

$\vec{h}_\ell$ W $\vec{h}_{\ell-1}$

↳ za konvolucijski sloj, ovo je tenzor četvrtog reda, ali ga je računalno (u programu) lagano dobiti - puno lakše nego za njega napisati jednadžbu

⇒ na slajdu 7 je objašnjeno kako doći do smanjenja broja množenja onim trikom s vanjskim produktom poznatim od prije:

↳ za MNIST, vanjski produkt:



$[784 \times 1] \cdot [1 \times 784] \rightarrow [784 \times 784]$

⇒ PROGRAMSKO OSTVARENJE:

↳ sloj mora pamtiti:

a) parametre

b) ulazne aktivacije (za gradijente)

c) gradijente parametara (za učenje)

↳ na slajdu 9 su greške u algoritmu:

- linija 3 treba biti: $p = X$ # input
- između linija 5 i 6 treba biti: $p = q$

(primjeti da je kôd puno lakši od samih matematičkih izraza)

⇒ na slajdu 10 je izvod gradijenata uz konvoluciju 1x1 (2D tenzor) jer nam je ovo puno jednostavnije:

- jednostavnost jer „q_r" ovisi samo o „p_r"

↳ kod većih konvolucija, ovo nije tako jednostavno jer „q_r" ovisi o susjedstvu „p_r" (imamo tenzor 4. reda) ako ne iskoristimo „one" trikove)

↳ izvod za veće konvolucije:

- na slajdu 12 počinje izvod (nije teško, samo sporo prijest to i osiguraj da shvatiš dimenzionalnost)

$$\boxed{q_{i+u,j+v} = p_{ij} \cdot W_{-u,-v}}$$

BITNO ZA SHVAĆANJE SLAJDA 15!

↳ na element dolje desno, utječe jezgra sa svojim elementima gore lijevo (zarotirana jezgra djeluje)

↳ zbog obratne jergre, uvodimo TRANSPONIRANU KONVOLUCIJU:

$$\vec{X} *^T W = pad(X, k-1) * rot_\pi(W)$$

ovdje je možda greška
(možda treba biti $(k-1)\cdot 2$,
no mje prevše batno
ra samo shvaćanje)

ZAD: Promotrimo konvolucijsku mrežu s dvije konvolucije 3×3 (W_0 i W_1), bez nadopunjavanja, korak 1, ReLU funkcijom, globalno sažimanje maksimumom te potpuno povezanim slojem s dvije dimenzije na izlazu sa funkcijom softmax. Izračunajte parametre: $W_{010} = -1$ $W_{011} = 1$, ostali od W_0 su nula

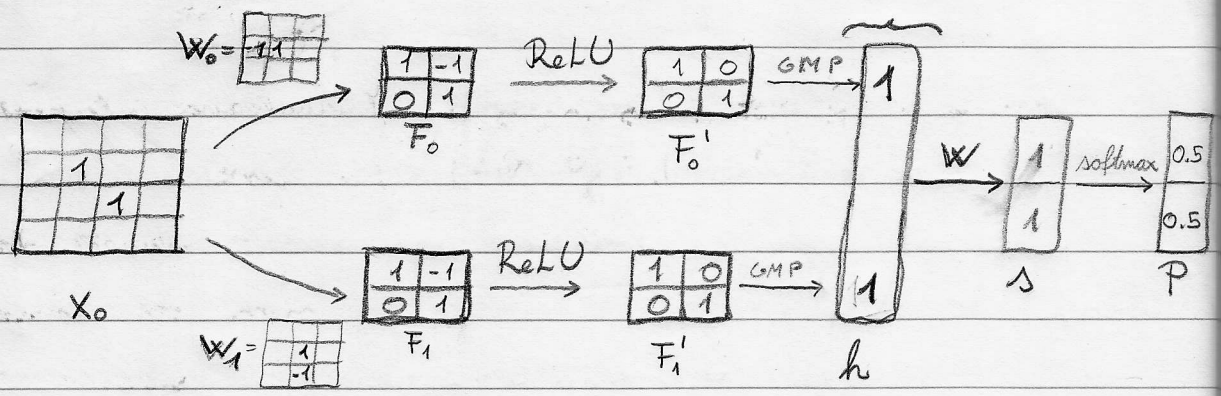
$W_{021} = -1$ $W_{011} = 1$, ostali od W_1 su nula

$W_{00} = W_{11} = 1$, ostali od W su 0

Na ulazu je 4×4 tenzor X_0 te je za njega oznaka $Y_0 = 1$. Za taj podatak je $X_{11} = X_{22} = 1$, a ostali su element jednaki nuli. Izračunajte gradijente negativne log-izglednosti po parametrima.

RJEŠENJE:

↳ skicirajmo:



ulaz u potpuno povezan sloj (gledamo kao vektor)

varijable jezgre dalje su iste mape (slučajno)

globalno sažimanje maksimumom (Global Max Pool)

↳ one-hot kôd primjera: $Y_0^{OH} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

↳ sada računamo gradijente ulančavanjem:

$$\frac{\partial \mathcal{L}}{\partial s} = (P - Y_0^{OH})^T = [0.5 \quad -0.5]$$

⇒ interpretacija:

$\vec{s} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ → ovaj treba rasti da bi gubitak rastao (jer je njegov gradijent pozitivan) (ni narovno idemo u suprotnom smjeru) (analogno za drugi član)

$$\frac{\partial \mathcal{L}}{\partial h} = \frac{\partial \mathcal{L}}{\partial s} \cdot W = [0.5 \quad -0.5]$$

$$\frac{\partial \mathcal{L}}{\partial F_0'} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0 \end{bmatrix} \qquad \frac{\partial \mathcal{L}}{\partial F_0} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial F_1'} = \begin{bmatrix} -0.5 & 0 \\ 0 & 0 \end{bmatrix} \qquad \frac{\partial \mathcal{L}}{\partial F_1} = \begin{bmatrix} -0.5 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial W_0} = X_0 * \frac{\partial \mathcal{L}}{\partial F_0} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}$$

ako W_0 dodamo ovo, gubitak će rasti, pa ni oduzimamo

$$\frac{\partial \mathcal{L}}{\partial W_1} = X_0 * \frac{\partial \mathcal{L}}{F_1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.5 & 0 \\ 0 & 0 & -0.5 \end{bmatrix}$$

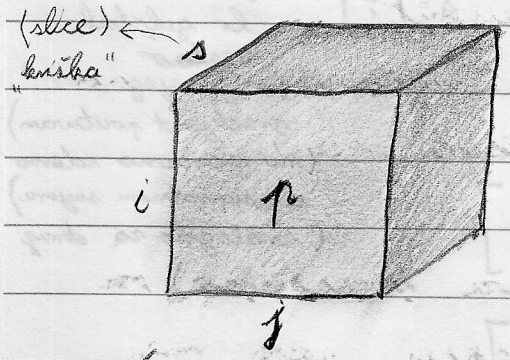
analogna interpretacija kao i gore

↳ primjet da kada ovog oduzmemo od W_1, dobivamo W_1 koji je sve sličnji podatku

⇒ NAPOMENA: ako je ulaz tenzor 3. reda (to su „p" i „q"), tada W mora biti tenzor 4. reda (konvolucijska jezgra)

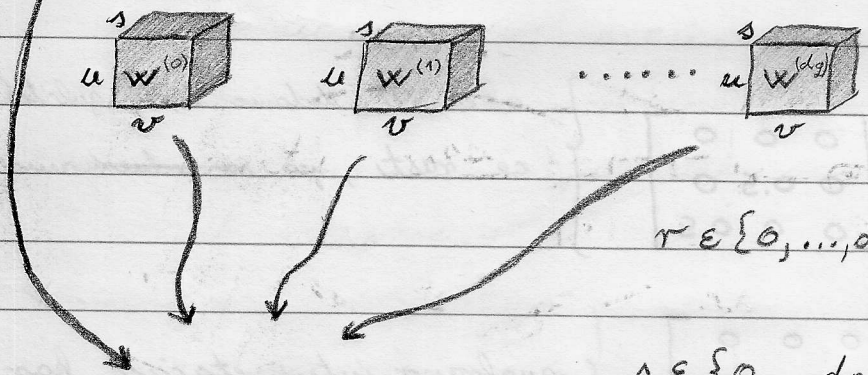
↓

moramo imati više malih kvadrića koje zetamo po jednom velikom kvadru



njih je d_q („r" je iterator)

d_q ⇒ dimenzionalitet vektora „q"



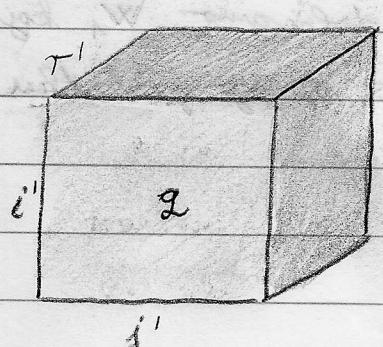
⇒ svi ovi tvore tenzor jezgre W

$r \in \{0, \dots, d_q - 1\}$ ⇒ „r" je indeks koj traži od 0 do $(d_q - 1)$

$s \in \{0, \dots, d_p - 1\}$ ⇒ „s" je indeks koj traži od 0 do $(d_p - 1)$

$i \in \{0, \dots, H - 1\}$ ⇒ „i" je indeks koj traži od 0 do $(H - 1)$

$j \in \{0, \dots, W - 1\}$ ⇒ „j" je indeks koj traži od 0 do $(W - 1)$



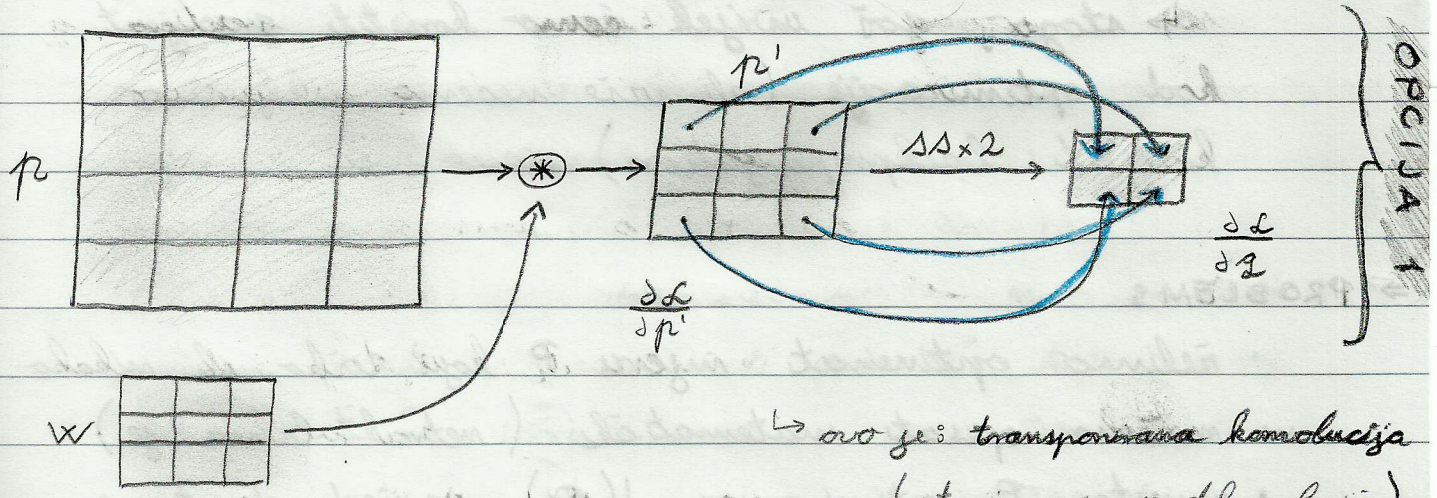
$u, v \in \{0, \dots, k - 1\}$

(ovo je 2D konvolucija sa 3D tenzorima jer su s iste u p i $W^{(r)}$)

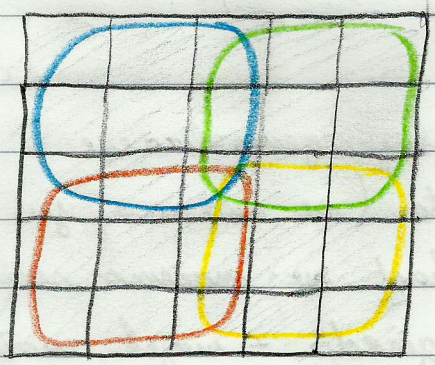
KORAČNOST KONVOLUCIJE

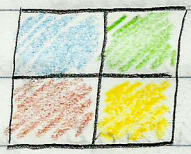
⇒ preskakanje u postupku obične konvolucije
↳ izostavlja svaki „s"-ti element
↳ lako je uočiti da time ništa ne gubimo
(vidi rešenje sa „ $\times s$ " na slajdu 23)

⇒ sa korakom s, otvaraju se 2 nova postupka za propagaciju unatrag u konvolucijskim slojevima
(VIDI SLAJD 24)



↳ ovo je: **transponirana konvolucija**
(stari naziv je dekonvolucija)





↓
služi za napuhavanje malih tenzora
(npr. male slike)
↳ koristi se u GAN-ovima
(generativnim modelima)

OPTIMIZACIJA PARAMETARA MODELA

⇒ do sada smo uvijek modificirali parametre tako da izračunamo gradijente i backpropagationom modificiramo težine

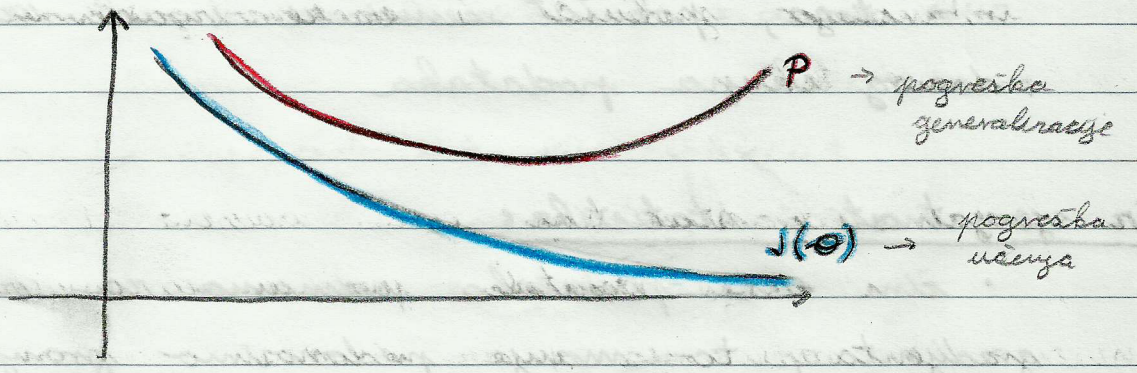
⇒ „black-box" algoritmi (npr. genetski algoritam) ne rade dobro na dubokim modela jer su dimenzionalnost prostora pretraživanja puno prevelike (milijuni parametara)
↳ stoga, još uvijek ćemo koristiti gradijent kod optimizacije (ali više nećemo isključivo koristiti backpropagation)

⇒ PROBLEMS
- želimo optimizirat mjeru P koju teško il nikako ne možemo opisat matematički (netraktabilna je)
- umjesto P optimiziramo $J(\theta)$ u nadi da time indirektno optimiziramo P

⇒ učenje: cilj je minimizirati pogrešku generalizacije
↳ ne može se potpuno optimizirati (ne možemo ju izravno niti ju optimizirat jer nikad ne možemo sarnat sve moguće podatke (npr. slike goveda - uvijek možemo uslikati novu sliku goveda))

⇒ <u>optimizacija</u>: minimiziraj / maksimiziraj funkciju koja ti je dana (sjeti se APR-a)

⇒ ako je učenjem definiran P, a optimizacijom $J(\theta)$, u opasnosti smo od prenaučenosti



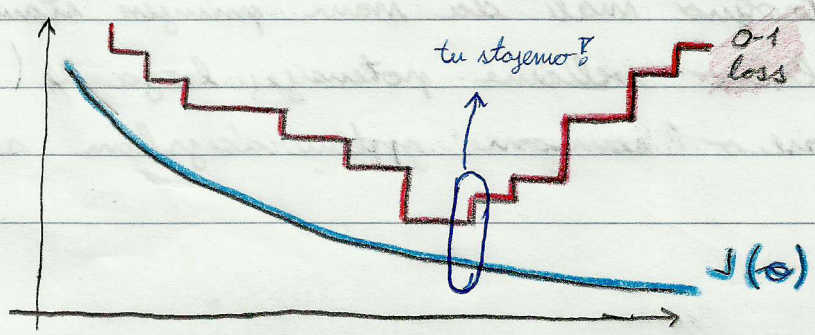
⇒ prisjeti se zašto ne možemo optimirati 0-1 gubitak:

↳ jer su gradijent svugdje 0 ili nisu definirani

↳ nadomještamo derivabilnim gubitkom

- čini model robusnijim jer definira postotke sigurnosti modela koje nastoji maksimizirati, a ne samo točnu klasifikaciju i tamo staje (što radi gubitak 0-1)

⇒ na ovom predmetu, optimiramo $J(\theta)$, ali radimo validaciju gubitkom 0-1 i kada on počne rasti onda stajemo s optimizacijom



⇒ da bi izračunali gradijent, moramo izračunati očekivani gradijent na svim slikama i tek onda imamo egzaktni gradijent (slajd 11 s dvije formule)

↳ to je skupo te nećemo moći s velikim skupovima podataka ovo provesti!

↳ stoga, pokušat ćemo ze izvući uzorkovanjem danog skupa podataka

⇒ <u>iz vjerojatnosti i statistike:</u>

- čim više podataka uzimamo za izračun gradijenta, to manje pridonosimo promjeni samog gradijenta (slajd 13 za dodatno objašnjenje)

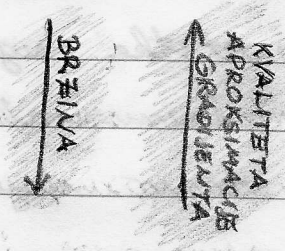
⇓

stoga, uzorkuj dataset!

⇒ PODJELA ALGORITMA UČENJA:

- batch (egzaktni)
- batch size
- stohastic

BRZINA ↓



KVALITETA APROKSIMACIJE GRADIJENTA ↑

hrv

grupni

mini-grupni

pojedinačni

⇒ primjeti da veličine grupe želimo da budu:

- dovoljno velike da se iskoriste svi procesori GPU-a
- dovoljno mali da nam primjeri stanu u memoriju
- obično veličine potencije broja 2 (npr. 256),

ali ovisi o samom optimizacijskom algoritmu

⇒ GENERALNA NAPOMENA: bolja aproksimacija gradijenta može biti popraćena većom stopom učenja

⇒ u dubokim modelima se tipično ne koristimo druge derivacije jer su Hesseove matrice prevelike (gradijent u 10000 dimenzija = Hesse 10000 x 10000)

↳ dodatno: neotporni su na sedlaste točke

⇒ IZBOR PODATAKA ZA MINI-GRUPE:

↳ svaki put ispermutiraj podatke i zatim ga slijedno „slice"-aj (prvih tisuću, drugih tisuću, ...)

↳ ako želimo najbolje rezultate i permutacija nije skupa, permutiraj između svaka dva „slice"-a ⇒ tako osiguramo da onaj koj je izabran u jednom „slice"-u ponovno može biti izabran u nekom drugom (TO JE DOBRO!)

⇓

ako ovo ne možemo, barem permutiraj kada pročeš sve „slice"-ve (nakon epohe) kako ne bi mreža stalno dobivala isti slijed gradijenata u svakoj epohi //

⇒ IZAZOVI GRADIJENTNE OPTIMIZACIJE:

a) sedlaste točke i lokalni minimumi:

- nisu tipični problemi, ali danas su sve manji problem zbog min-grupnog učenja

b) iščezavajući i eksplodirajući gradijenti:

- ako su gradijenti preveliki, radimo prevelike pomake → rješenje: odsjecanje gradijenata (engl. gradient clipping)
- iščezavanje gradijenata nije problem ako koristimo ReLU

c) eksplodiranje kod prolaza unaprijed i unatrag

- ako imamo puno slojeva, to je puno puta množenje sa sličnim matricama „W" → ovo se može napisati matematički:

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

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

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

⇓

velike vrijednosti na dijagonali idu u beskonačnost, a male iščezavaju u nulu!

STOHASTIČKI GRADIJENTNI SPUST

(ustvari se misli na mini-batch inačicu)

⇒ izvrćem gradijenta smo naučili:

↳ algoritam backpropagation

⇒ vrlo je važan izbor stope učenja:

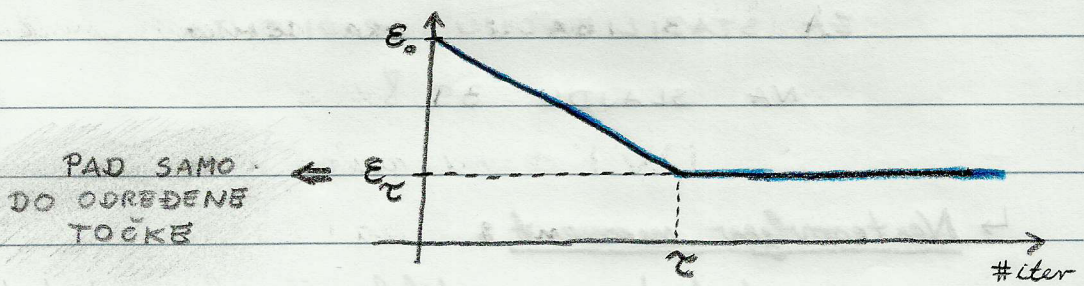
↳ matematičke ograde:

$$\sum_{k=1}^{\infty} \varepsilon_k \to \infty \,, \quad \sum_{k=1}^{\infty} \varepsilon_k^2 < \infty$$

ε_k je stopa učenja u koraku „k"

↳ stopa učenja kroz iteracije pada i to se u praksi pokazalo dobro:

- linearni pad stope učenja:



↳ kako izabrat „ε_0"?

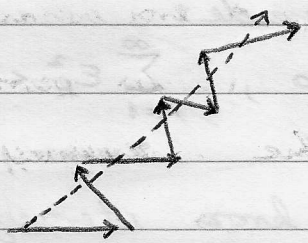
- pa, malo veći od onog koji daje monotonu konvergenciju u prvih 100 iteracija (iskustveno)

⇒ svi navedeni izvori i dalje su problem kod stohastičnog gradijentnog spusta //

⇒ UČENJE S MOMENTOM:

↳ ako učimo dobivam da gradijent pokazuje u istom (sličnom) smjeru, to se sigurnije osjećam da on pokazuje prema optimumu i mogu raditi veće korake nego inače

↳ dodatno, moment pomaže izravnati kretanje po prostoru tako da dodamo neko umanjeni gradijent iz prethodnog koraka: $\alpha \cdot \vec{v}$, $\alpha \in [0, 1]$



} izravnanje puta ☺

⇓

VIDI MATEMATIČKO OBJAŠNJENJE ZA STABILIZACIJU GRADIJENTA NA SLAJDU 39 ☺

↳ **Nesterovljev moment**:

- dodatna modifikacija koja dodatno poboljšava ponašanje učenja s momentom
- ne radimo sa stvarnim parametrima, već gradijent računamo s kopijom parametara kojima dodamo $\alpha \cdot \vec{v}$ (i forward-pass radimo sa tom kopijom parametara uvećanom za $\alpha \cdot \vec{v}$)
- time prije računanja gradijenta unaprijed malo izmijenimo parametre (UBRZANJE ☺)

⇒ INICIJALIZACIJA PARAMETARA

↳ težine želimo takve da narušiju simetričnost modela (da prvi i radnj neurone ne uče isto!)

↳ biramo slučajno težine svakog neurona:

za sigmoidalne neurone:
- iz normalne distribucije sa varijancom: $\frac{1}{n}$
  - n_{in} → broj ulaza neurona (čuva varijancu na podacima)
  - n_{out} → broj izlaza neurona (čuva varijancu gradijenata)

za ReLU neurone:
- iz normalne distribucije s varijancom: $\frac{2}{n}$

OVO IMAŠ IMPLEMENTIRANO U TENSORFLOW-u! (vidi slajdove)

↳ *inicijalizacija biasa*:
- varijable za varijable mreže, al često nešto bliu nule!

ALGORITMI S ADAPTIVNIM STOPAMA UČENJA

⇒ tri algoritma koji nam sljedeći poprvlja prethodnog:

- Ada Grad (2011.)
- RMSProp (2012.)
- Adam (danas se koristi)

⇒ **ALGORITAM Ada Grad:**

$$\begin{bmatrix} \frac{\partial L}{\partial W_1} \\ \frac{\partial L}{\partial W_2} \\ \frac{\partial L}{\partial W_3} \end{bmatrix} \begin{bmatrix} 100 \\ 10 \\ 0.2 \end{bmatrix} \xrightarrow{\text{iter}} \begin{bmatrix} 200 \\ 10 \\ -0.4 \end{bmatrix}$$

} velike vrijednosti, pa ne „W_1" i „W_2" brzo počnu koristit male stope učenja (dajemo im karmu)

↳ **problem:** nikada ne raborављaš velike vrijednosti (suma stalno raste), pa u nekim dimenzijama algoritam može stati mijenjati težine

↳ ažuriranje parametara:

$$\Delta\Theta \leftarrow -\frac{\varepsilon}{\delta + \sqrt{r}} \odot g$$

$\odot$ ⇒ „element-wise" množenje

$\delta = 10^{-7}$ ⇒ za numeričku stabilnost

⇒ ALGORITAM RMSProp:

↳ rješava problem iz AdaGrada tako da daje eksponencijalno gušenje na vrijednosti koje smo vidjeli dalje u prošlosti

- tako daje više značajnosti aktualnim vrijednostima, a manje starim vrijednostima

$$\vec{r} \leftarrow \rho \vec{r} + (1-\rho)\vec{g} \odot \vec{g}$$

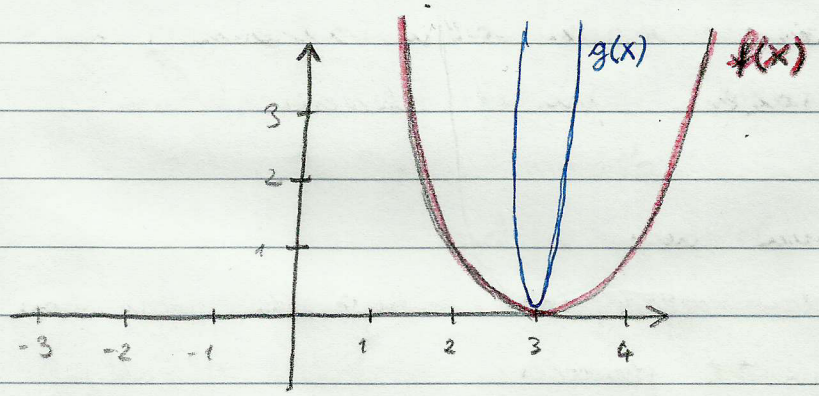
↳ krajnosti:

$\rho = 1$ ⇒ bitne su samo stare vrijednosti

$\rho = 0$ ⇒ bitan je samo gradijent

⇒ rekapitulacija: zašto želimo adaptivni gradijent?

↳ zato jer veliki gradijent ne znači da se jako dobro ići u tom smjeru



$$f(x) = (x-3)^2$$

$$g(x) = 10^{70}(x-3)^2$$

↳ ako smo u $x = 2$, obje funkcije zahtijevaju pomak od 1, a imaju drastično različit gradijent (algoritmu optimizacije se moraju i time moći nositi)

⇒ **ALGORITAM Adam:**

↳ koristi moment kako bi procjenio gradijente i norme gradijenata i tako bi izbjegao krivudanje po prostoru

(procjene su označene s točkicom: $\dot{s}$, $\dot{r}$)

↳ Adam nije najbrži, ali je jako robusan jer koristi dvije dobre metode da pare na pomake

- dobar ra „first-try", pa onda kada vidiš ponašanje napadaš sa stohastičkim gradijentnim spustom

↳ ne uvodi puno dodatnih računanja ra memoriju:

- treba dva dodatna vektora iste veličine kao i vektor parametara jer su sve operacije element-wise

META - ALGORITMI ZA DODATNA POBOLJŠANJA

⇒ LOKALNA NORMALIZACIJA ODZIVA NEURONA

↳ želimo da neuroni kroz varijable mape značajki (na istim pozicijama) više varijable stvore

• želimo da nisu korelirani, tj. da oni daju varijable vrijednosti

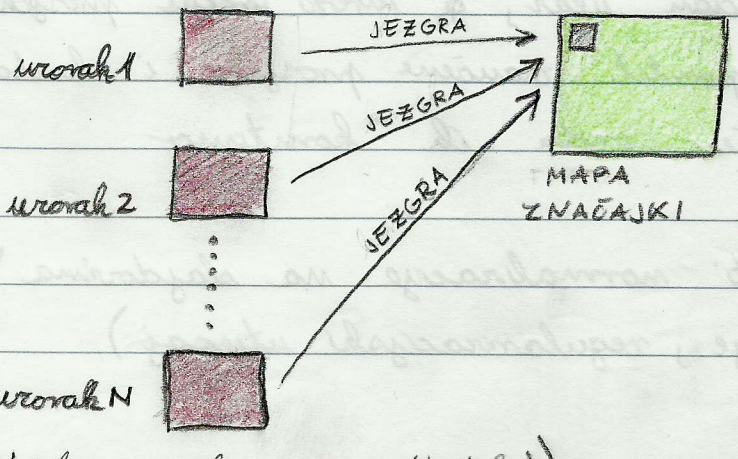
↳ ideja je da pojačamo učenje nekih kako ovi neki davali isto

(VIDI OKVIRNO IZRAZ NA SLAJDU ▽)

⇒ NORMALIZACIJA NAD GRUPOM (jako učinkovita)

↳ problem: mala promjena parametara bliže ulaza mreže daje i malu promjenu na izlazu, a promjena bliže ulaza se jako propagira i generira drastičnu promjenu na izlazu dubokog modela

• želimo model robusniji na to ▽



(N → broj uzoraka u grupi (batchu))

} za svak neuron, pratimo kako se njegova vrijednost mijenja s promjenom ulaza (pamtimo N brojeva za svaki neuron)

↳ od svakog neurona oduzmemo prosjek tih N zapamćenih vrijednosti za neuron

• dobivamo:

a) biasi su umrtvljeni jer smo oduzeli aritmetičku sredinu

b) „izlazi" svakog neurona su centrirani oko nule

⇓

no, to nisu stvarni izlazi, nego su to normalizacijski odzivi koje ćemo tek koristiti za normalizaciju

↳ kako bi napravili normalizaciju još trebamo afinu transformaciju jer nismo sigurni je li dobro da su normalizacijski odzivi oko nule

• dodatna dva parametra (VIDI SLAJD) koja se isto mogu učiti jer je funkcija po njima i dalje derivabilna (njih isto šaljemo "iza" backpropagationom)

↳ PROBLEM: kada idemo u fazu iskorištavanja mreže, imamo samo jedan ulaz, a mreži treba prosjek za izlaz, pa treba pamtiti naučene prosjeke i kada eksploatiramo mrežu onda ih koristimo

↳ vidi prednosti normalizacije na slajdovima!
(brže učenje, regularizacijski utjecaj)

⇒ **NADZIRANO PRETRENIRANJE:**

↳ kada je model dubok, učenje je jako teško problem (prostor je jako velik)

↳ **ideja:** uči mrežu na lakšem problemu kako bi ona došla u neko smisleno područje, ratim dodaj jedan sloj (povećaj kapacitet) i ponovno nauči mrežu i dodaj sloj itd.

REGULARIZACIJA DUBOKIH NEURONSKIH MREŽA

⇒ **REGULARIZACIJA**: svaka tehnika koja ne poboljšava rezultate na skupu za učenje u nadi da poboljšamo generalizaciju modela (na neki način sneta optimizatoru)

↳ smanjuje varijancu (osjetljivost modela o skupu za testiranje koji mu je dan) i povećava pristranost (gura model u pravom smjeru)

⇒ u praksi, najbolji model dobivamo s prevelikim kapacitetom pritegnutim s regularizacijom

⇒ **REGULARIZACIJA NORMOM VEKTORA PARAMETARA**:

↳ najstarija metoda regularizacije

meka norma

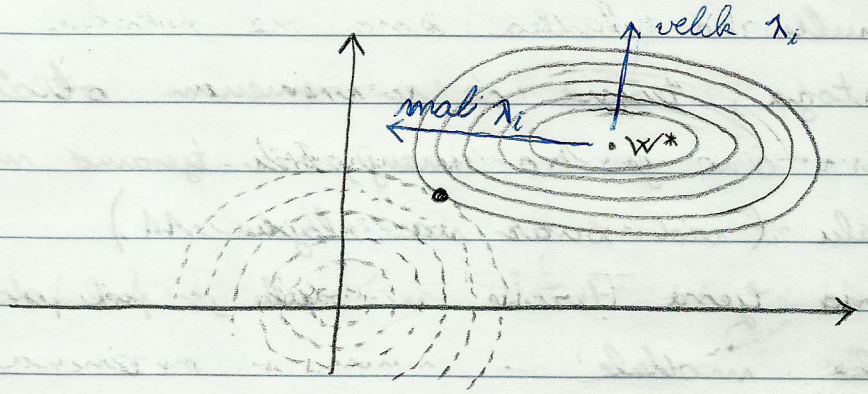
$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \lambda \Omega(\theta)$$

↳ intuicija: teške može podatke, pa ako su one velike i malo promjenjene podatke onda dobivamo veliku promjenu vrlara

$\lambda \in [0, \infty\rangle$ ⇒ određuje se unakrsnom provjerom (ponavljanje eksperimenta ☹)

↳ vizualizacija L2 norme:

• razvijemo neregularizirani funkciju gubitka J u Taylorov red oko minimuma W^* za dane podatke (prva derivacija nestaje zbog minimuma):



$$\tilde{\vec{w}}_i = \vec{q}_i \cdot \frac{\lambda_i}{\lambda_i + \alpha} \vec{q}_i^{\,T} \vec{w}^*$$

ako je H jedinična, $\vec{q}_i$ nestaje

⇒ ako je λ_i velik, jako smo sigurni o poziciji na kojoj jesmo (W^* je dolar), pa se pomičemo malo zbog regularizacije

↳ mala λ_i znači da se podaci rasipaju više, pa i više regulariziramo u tom smjeru

↳ utjecaj L2 regularizacije na linearnu regresiju:

$$\vec{y} = X\vec{w} \quad / \cdot X^T$$

$$X^T\vec{y} = X^T X \vec{w} \Rightarrow \vec{w} = (X^T X)^{-1} X^T \vec{y}$$

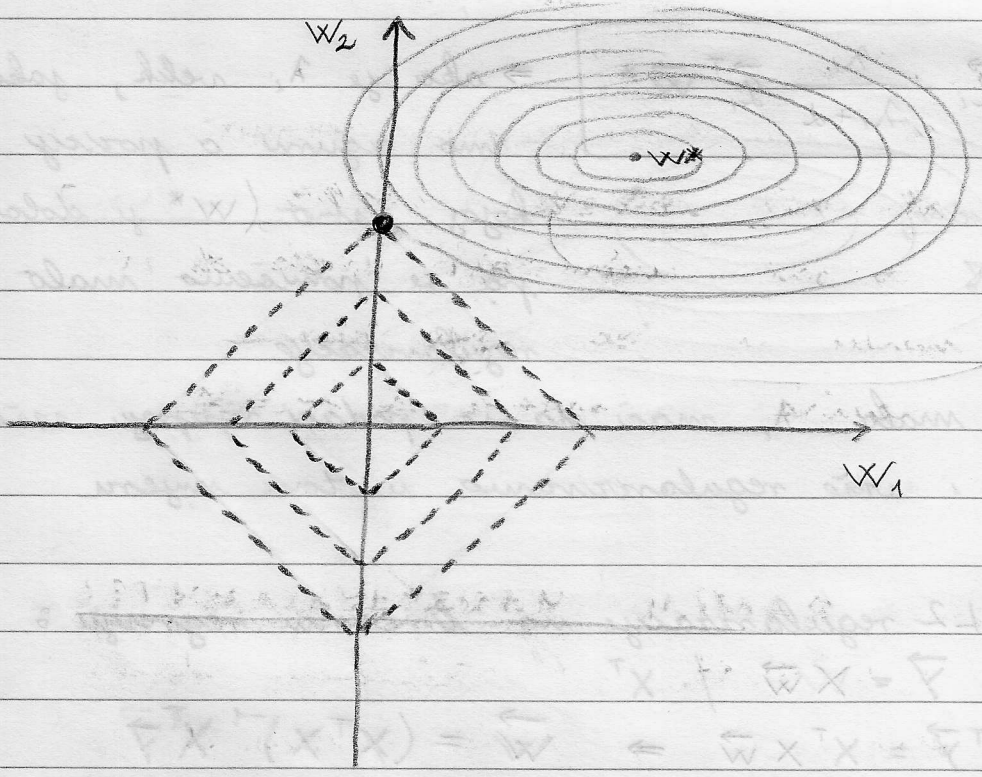
regularizacija

$$\vec{w} = (X^T X + \alpha I)^{-1} X^T \vec{y}$$

smanjujemo varijance, pa se ovaj puta bolje definira

↳ regularizacija L1 normom:

- isto kao i L2 norma, no sada dodajemo $\lambda \operatorname{sgn}(w)$, pa nema nikakvog mehanizma raščitke od toga da težine odu u nulu
- stoga, težine će s vremenom otići u nulu - čim je λ_i manji, brže tjeramo model u nulu (vidi crvar na slajdu 11)
- ona tjera težine u nulu, pa dođe rijetke modele



- radi isto što i L2, al vuče te parametre koje više regularizirama jako u nulu

⇒ NORME VEKTORA PARAMETARA KAO OGRANIČENJA:

↳ definira se područje u kojem je rješenje prihvatljivo s nekom odredenom normom

↳ ovo nije loše jer radi isto kao i obična regularizacija, a izbjegava lokalne minimume nešto bolje (nisam uguran našto...)

⇒ REGULARIZACIJA GENERIRANJEM PODATAKA:

↳ najbolja regularizacija je dodavanje još podataka, no dobivanje podataka i njihovo označavanje može biti skupo

↳ npr. ako na slikama prepoznajemo aute, možemo skalirati, rotirati i zrcaliti slike, pa dobijemo još slika automobila (nešto drugačijih)

↳ drugi način je **dodavanje šuma** na različite podatke i mjesta u model

⇒ REGULARIZACIJA RANIM ZAUSTAVLJANJEM:

↳ dok god greška na validaciji pada, uči, a kada više ne pada - prestani se ☺

↳ dobro jer nemamo dodatnih parametara koje učimo unakrsnom provjerom (ne moramo ponavljati eksperiment treniranja koj može trajati nekoliko dana...)

↳ za učenje, evaluiraj na drugom procesoru!

↳ nedostatak: smanjuje se skup za učenje

(možeš naći optimum i onda pobrenuti učenje na svim podacima za optimalni broj epoha)

⇒ **DJELOMIČNO-NADZIRANO UČENJE:** (dosta novo?!)

↳ može služit kao metoda regularizacije

↳ dio podataka nema oznaku

↳ ideja: računaju se dva gubitka:

- na označenom skupu
- na neoznačenom skupu

↳ dijeljenje parametara prilikom optimizacije ova dva gubitka osigurava regularizaciju

- neoznačen podac pokazuje kako bi otprilike trebala ić granica (tamo gdje je „rupa" u podacima)

⇒ **REGULARIZACIJA VEZANJEM PARAMETARA:**

↳ ideja: ne znamo točno rješenje dva problema, al smatramo da su njihova rješenja slična

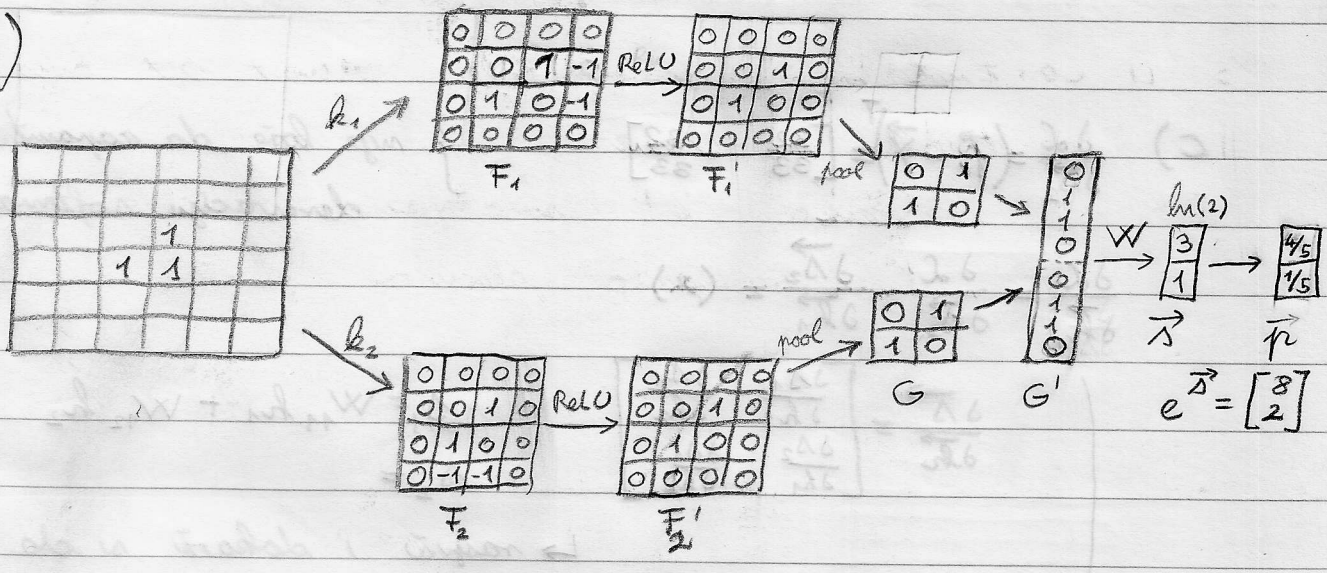
PONAVLJANJE ZA MEĐUISPIT

(međuispit je 2016)

(1) • pretpostavlja se default: pomak na poolingu (2)
• pretpostavlja se kvadratna slika
• flatten se radi po redovima

a)
↳ veće nakon poola: 2×2
↳ prije poola su: 4×4
↳ ako je konvolucija 3×3 bez nadopunjavanja, slika je 6×6 (gubi se po dva retka)

b)



(2) a) $\vec{s}_1 = W_1 \vec{x} + \vec{b}_1$

$\vec{h}_1 = ReLU(\vec{s}_1)$

$\vec{s}_2 = W_2 \vec{h}_1 + \vec{b}_2$

$\vec{p} = softmax(\vec{s}_2)$

b) $\vec{s}_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ -2 \end{bmatrix}$

$\vec{h}_1 = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$ $\quad \vec{s}_2 = \begin{bmatrix} \ln 2 & 0 \\ 0 & -\ln 2 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 5\ln 2 \\ 0 \end{bmatrix}$

$e^{\vec{s}_2} = \begin{bmatrix} 32 \\ 1 \end{bmatrix} \Rightarrow \vec{p} = \begin{bmatrix} 32/33 \\ 1/33 \end{bmatrix}$

c) $\frac{\partial \mathcal{L}}{\partial \vec{s}_2} = (\vec{p} - \vec{y})^T = \begin{bmatrix} \frac{32}{33} & -\frac{32}{33} \end{bmatrix}$ } nije loše da zapamtiš ten derivaciju softmaxa

$\frac{\partial \mathcal{L}}{\partial \vec{h}_1} = \frac{\partial \mathcal{L}}{\partial \vec{s}_2} \cdot \frac{\partial \vec{s}_2}{\partial \vec{h}_1} = (*)$

$\frac{\partial \vec{s}}{\partial \vec{h}} = \begin{bmatrix} \frac{\partial s_1}{\partial h_1} & \frac{\partial s_1}{\partial h_2} \\ \frac{\partial s_2}{\partial h_1} & \frac{\partial s_2}{\partial h_2} \end{bmatrix}$

$s_1 = W_{11} h_1 + W_{12} h_2$

$s_2 =$

↳ raspiši i dokaži si da je ova derivacija W, a ne W^T

W_2

$\frac{\partial \mathcal{L}}{\partial \vec{h}_1} = \begin{bmatrix} \frac{32}{33} & -\frac{32}{33} \end{bmatrix} \cdot \begin{bmatrix} \ln 2 & 0 \\ 0 & -\ln 2 \end{bmatrix} = \begin{bmatrix} \frac{32}{33}\ln 2 & -\frac{32}{33}\ln 2 \end{bmatrix}$

ovi su $[1\ 0]$ jer je $\begin{bmatrix} 5>0 \\ -2<0 \end{bmatrix}$

$$\frac{\partial \mathcal{L}}{\partial \vec{s}_1} = \frac{\partial \mathcal{L}}{\partial \vec{h}_1} \cdot \frac{\partial \vec{h}_1}{\partial \vec{s}_1} = \begin{bmatrix} \frac{32}{33}\ln 2 & -\frac{32}{33}\ln 2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \frac{32}{33}\ln 2 & 0 \end{bmatrix}$$

ovi su nula jer

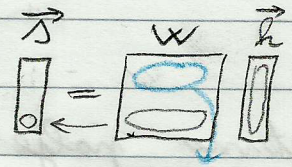
ovi ne ovise o $\vec{s}_1$

(UVIJEK DIAGONALNA MATRICA)

po prvom retku matrice W_1

$$\frac{\partial \mathcal{L}}{\partial W_{11:}} = \underbrace{\frac{\partial \mathcal{L}}{\partial \vec{s}_1} \cdot \frac{\partial \vec{s}_1}{\partial W_{11:}}} = \underset{2\times 1}{\begin{bmatrix} \frac{32}{33}\ln 2 & 0 \end{bmatrix}} \underset{2\times 2}{\begin{bmatrix} x_1 & x_2 \\ 0 & 0 \end{bmatrix}} = \frac{32}{33}\ln 2 \cdot \begin{bmatrix} 5 & 2 \end{bmatrix}$$

s_2 ne ovisi o $W_{11:}$:



O TIMA NE OVISI!

zbog neovisnost može i ovako:

$$\frac{\partial \mathcal{L}}{\partial s_{11}} \cdot \frac{\partial s_{11}}{\partial W_{12:}}$$

$$\frac{\partial \mathcal{L}}{\partial W_{12:}} = \frac{\partial \mathcal{L}}{\partial \vec{s}_1} \frac{\partial \vec{s}_1}{\partial W_{12:}} = \frac{\partial \mathcal{L}}{\partial s_2} \cdot \frac{\partial s_2}{\partial W_{12:}} = 0 \cdot \begin{bmatrix} x_1 & x_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

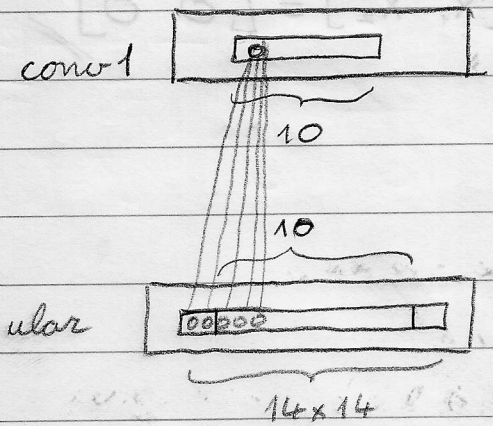
③

bias nisu navedeni

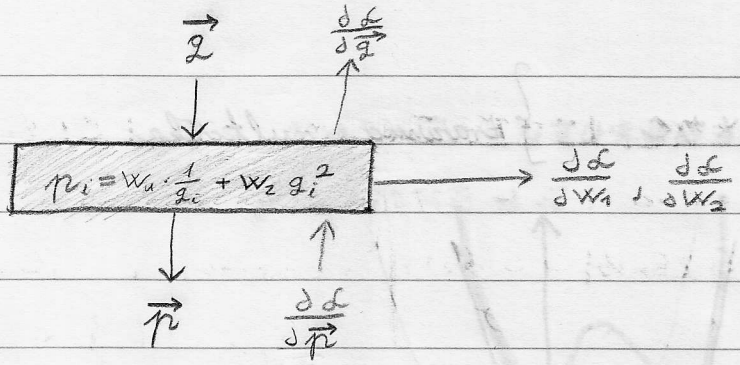
sloj	izlazna dim	n_param	c) fc_replacement n_param	receptivno polje značaj druge konvolucije
ulaz	28 x 28	0	∅	14 x 14
conv 1	28x28x16	5·5·16(+16)	(28·28)·(28·28·16)	10 x 10 (jer je kaže 2x2)
pool 1	14x14 x 16	∅	∅	5 x 5 (jer je conv 5x5)
conv 2	14 x 14 x 32	(5·5·16)·32(+32)	(14·14·16)·(14·14·32)	
pool 2	7x7 x 32	∅	∅	
flatten	1568	∅	∅	
FC1	512	1568·512	1568·512	
FC2	2	512·2	512·2	
softmax	2	∅	∅	

bias nisu navedeni

- receptivno polje s obzirom na ulaz:
  - najlakše je da idemo od kaja prema početku



(4)



$$\frac{\partial \mathcal{L}}{\partial \vec{z}} = \frac{\partial \mathcal{L}}{\partial \vec{r}} \cdot \frac{\partial \vec{r}}{\partial \vec{z}} = \frac{\partial \mathcal{L}}{\partial \vec{r}} \odot \left[\frac{\partial r_i}{\partial z_i}\right]$$

$[1 \times D] \quad [1 \times D] \quad [D \times D]$

$$\frac{\partial r_i}{\partial z_j} = \begin{cases} -\frac{w_1}{z_i^2} + 2 w_2 z_i, & i = j \\ 0, & i \neq j \end{cases}$$

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial \vec{r}} \cdot \frac{\partial \vec{r}}{\partial w_i} = \left\langle \frac{\partial \mathcal{L}}{\partial \vec{r}}, \frac{\partial \vec{r}}{\partial w_i} \right\rangle$$

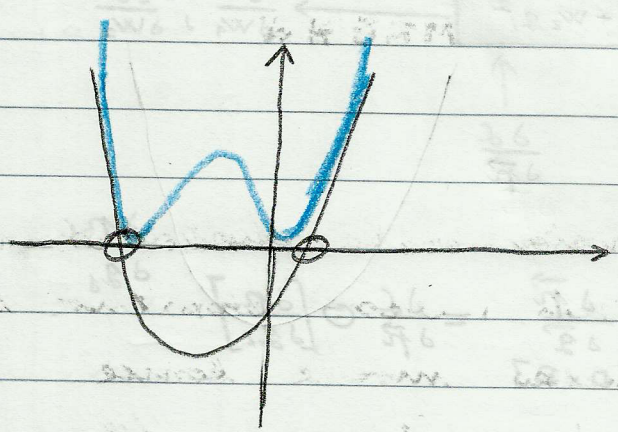
$[1 \times D] \quad [D \times 1]$

} skalarni produkt (samo druga oznaka)

$$\frac{\partial r_i}{\partial w_1} = \frac{1}{z_i} \qquad \frac{\partial r_i}{\partial w_2} = z_i^2$$

```
class Layer7:
    def __init__(self):
        ...                    # ne traži se?
    def forward(self, z):
        self.z = z
        return self.w1 / z + self.w2 * z**2
    def backward_inputs(self, grads):
        return grads * (-self.w1 / self.z**2 + 2 * self.w2 * self.z)
    def backward_parameters(self, grads):
        return [[self.w1, np.dot(grads, 1/self.z)],
                [self.w2, np.dot(grads, self.z**2)]]
```

(6) $f(x) = ax^2 + bx + c$ } tražimo multiple



$$X_0 = \arg\min\left[f^2(x)\right]$$

```
import tensorflow as tf
a = tf.Constant(1.0)        # kako to?
b = tf.Constant(1.0)
c = tf.Constant(1.0)
X = tf.Variable(0.0)        # ili neki inicijalizator
f = a * X ** 2 + b * X + c
L = f ** 2
sgd = tf.train.GradientDescentOptimizer(0.01);
train = sgd.minimize(L)
sess = tf.Session()
sess.run(tf.GlobalVariablesInitializer())
for i in range(1000):
    print(sess.run([train, X, L]))
```

POVRATNE NEURONSKE MREŽE

⇒ kod slika, postoj velika eksplicitna međuovisnost susjednih piksela (vrlo vjerojatno su slični) ; to konvolucijske mreže koriste

↳ dodatno, slike su fiksne dimenzije (ako nisu, svodimo ih na neku razjednačenu veličinu)

⇒ no, što je s npr. **tekstom**?

↳ kako rad Google Translate?

↳ e pa, to je motivacija za povratne neuronske mreže

⇒ jezik (slova) izgleda računalu onako kako nama izgleda japanski (nama koji ne znamo japanski)

↳ stoga, pametno treba prikazati podatke kako bi ih računalo znalo prikazati

↳ <u>prva ideja</u>:

ONE-HOT + BAG-OF-WORDS (vidi slajdove)

(ovo je bila state-of-the-art do 2011.)

• PROBLEMI:

1) velična vokabulara zbog one-hot
2) nemamo sličnost riječi
3) gubimo pojam redoslijeda riječi

⇒ DISTRIBUIRANE REPREZENTACIJE RIJEČI:

↳ rješavamo probleme 1. i 2. vrste riječi

↳ „ponavat ćete riječ po njenom društvu"

↳ IDEJA:

- svaka riječ je vektor
- ako su dvije riječi slične, imat će slične vektore

(vektorski prikaz riječi vidi na slajdu 13)

↳ još uvijek imamo veliki vokabular, a to rješavamo tako da smanjimo **matricu susjedstva** (vidi slajdove okvirno - ima još načina)

↳ u svakom reduciranom prostoru uspijevamo **dobivati semantičke sličnosti**:

- npr. udaljenost muž - žena je ista kao i ujak - ujna

↳ kako bi dobili semantičku sličnost, možemo koristiti i klasifikacijske modele

- damo logističkoj regresiji puno ispravnih primjera rečenice s riječi „QUICK" i tako naučimo koje riječi su uz riječ „QUICK" daju ispravnu rečenicu (time dobivamo semantičku sličnost)

↳ vidimo da za semantičku sličnost ne trebamo duboke modele - plitki su dovoljni

⇒ rješavanje problema 3:

↳ za to, trebamo povratne neuronske mreže

⇒ kada bi ovo probali riješiti sa klasičnim dubokim neuronskim mrežama, zapeli bi zbog:

a) fiksnog broja riječi u rečenici

b) prevelikog broja parametara

c) nemamo poredak riječi zbog toga što je svaki neuron povezan sa svakim

↳ prva dva problema možemo riješiti dijeljenjem parametara (matrica U) - ne kažem zašto to ne mijenja funkciju mreže ako je U pametno postavljen za naš vokabular

↳ treći problem rješavamo povratnim težinama spremljenim u matrici W:

$$h^{(t)} = \left(U x^{(t)} + W \cdot h^{(t-1)} \right) = \underbrace{[U; W] \cdot [x^{(t)}; h^{(t-1)}]}_{\text{ako konkatenirano}}$$

U i W, dobivamo isto kao i neuronske mreže prije

↳ mana ovakvog pristupa je da neka riječ ima samo za riječi prije (a ne nakon), ali to ćemo popraviti kasnije //

⇒ pogledajmo što se zapravo događa u svakom koraku:

- gledaj slajd 41 ▽
- želimo zapravo rečenicu su jedan vektor

(rečenica je kontekst varijabilne duljine)

$$h^{(t)} = f\left(\underbrace{W \cdot h^{(t-1)}}_{\text{PRETHODNI KONTEKST}} + \underbrace{U \cdot x^{(t)}}_{\text{NOVA INFORMACIJA}} + b\right)$$

- što je vektor koji pamtimo veći, to manje informacije gubimo kroz vrijeme
- napomena: matrica W je kvadratna, pa pari da li ju treba transponirat ili ne jer se neće javit greška u matričnom množenja
- PROBLEM:
  - ↳ zadnja riječ ima najveći utjecaj, a riječi prema početku se gube
- ako prevodimo i prevedena rečenica ima više slova, na $X^{(t)}$ dajemo nul-vektor, sve dok ne generiramo neku terminacijsku oznaku na izlazu
- hiperparametri:
  1) veličina skrivenog sloja H (što više, to bolje)
  2) broj vremenskih koraka
  3) algoritam optimizacije (najčešće Adam)
- na slajdu 46 vidi sve formule koje definiraju našu povratnu neuronsku mrežu

⇒ treniranje (učenje) povratnih neuronskih mreža:

↳ naziva se propagacija unatrag kroz vrijeme:

· jedina komplikacija je da $h^{(1)}$ utječe na $L^{(1)}$ i na sve ostale, $L^{(2)}$ na sve nakon i na $L^{(2)}$, itd.

↳ koristimo za „f" sljedeću funkciju

$$\boxed{\tanh(x) = 2\sigma(2x) - 1}$$

(ima izlaz od −1 do 1 što može biti korisno)
(vidi derivaciju ove funkcije na slajdovima)

↳ ova propagacija nije teška, samo se sjeti što je $h^{(t)}$ i da to nije nikakvo osobito proširenje na obične duboke modele – samo imamo ekstra ulaz, pa po njemu isto radimo parcijalnu derivaciju
(vidi slajdove za izračun i gledaj skice mreže paralelno s time)

↳ <u>problem</u>: gradijent na parametre se akumuliraju, pa za puno vremenskih koraka (veliki T) imamo eksplodirajuće gradijente

(vidi formulu na slajdu 66)

RJEŠENJE

GRADIENT CLIPPING

NAPREDNIJE POVRATNE NEURONSKE MREŽE

⇒ prošli smo preko obične (plitke) povratne neuronske mreže

↳ jedna povratna ćelija nam nije dosta, pa ćemo pogledat naprednije (dublje) povratne neuronske mreže

↳ na slajdu 5 pogledaj dobru sliku jedne povratne ćelije naše (jednoslojna neuronska mreža)

↳ nije teško shvatiti backpropagation kod RNN-ova, al proučit ćemo to!

⇒ DUBOKA POVRATNA NEURONSKA MREŽA:

- samo naslijepimo jedan RNN na drugi (imamo dvije dimenzije mreže - dubina i vremenska ovisnost)
- u praksi 4 do 8 slojeva je dovoljno (kasnije se ne uvlače neke nove informacije)
- receptivno polje:
  - prvi korak RNN ćelije vidi samo prvi ulaz
  - loše jer prva riječ nema kontekst

"BAR"
↙ ↘
kafić? blok čokolade?

DVOSMJERNA DUBOKA POVRATNA NEURONSKA MREŽA

↳ rješenje: spojamo dvije mreže, pa jedna ide unaprijed, a druga unatrag (rezultate tih mreža konkateniramo)

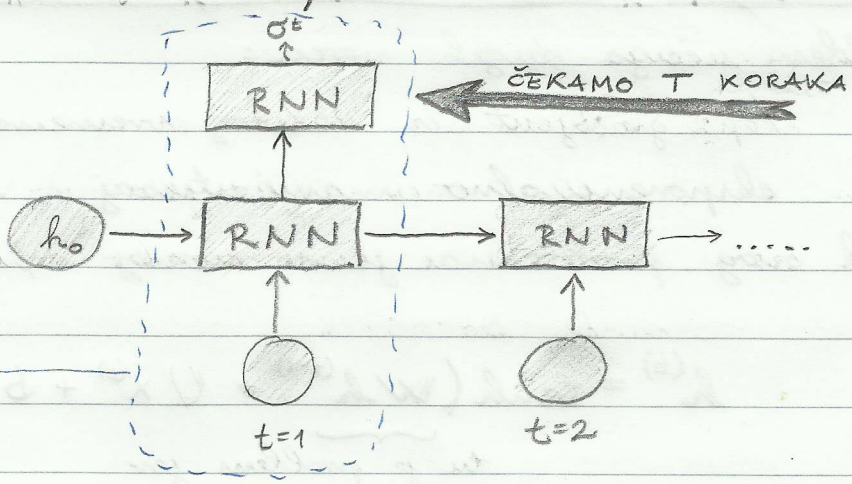
⇒ DVOSMJERNA POVRATNA NEURONSKA MREŽA:

↳ rješava problem „slijepog" prvog ulaza

↳ konkatenacija bolja od uprosječavanja jer se uprosječavanjem može gubiti informacija

↳ **veća složenost:**

• trebamo čekati da mreža iz drugog smjera dođe do početka da bi mogli obraditi prvi element



ovo radi jednosmjerna duboka povratna neuronska mreža

• na svakoj razini čekamo T koraka, pa je složenost T·D

↳ prije je složenost bila T+D-1

↳ **nestajući i eksplodirajući gradijent:**

$W\delta \gg 1 \Rightarrow$ eksplodiranje

$W\delta \to 0^+ \Rightarrow$ nestajanje

(ovo je na slajdu 30 uvedeno sa skalarima radi jednostavnosti, a na slajdu 31 je isto u matričnom prostoru, pa imamo svojstvene vrijednosti, ali je isti princip)

↳ zašto ne ReLU?

- u unaprijednom prolazu množenu dobiti jako velike brojeve
- stoga, koristi il sigmoidu il već spomenuti tangens hiperbolni

⇒ ĆELIJA S DUGOROČNOM MEMORIJOM (LSTM):

↳ problem učenja dugih veza:

- gradijent iz dalekog vremena ima eksponencijalno manji utjecaj

↳ uzrok ovog problema je u samoj definiciji:

$$h^{(t)} = \tanh\left(\underbrace{W h^{(t-1)}}_{\text{tu je problem jer } h^{(t)} \text{ množimo sa } W \text{ u svakom koraku}} + U x^{(t)} + b\right)$$

↳ sada, jedno stanje pokraj $h^{(t)}$ ima i $s^{(t)}$ (definirano je s dva vektora)

delegiranje uloga:

- $h^{(t)}$ samo računa izlaz
- $s^{(t)}$ samo pamti informacije (ono se ne dira, barem ne jednostavno - mreža mora imati dobar razlog da ga mijenja)

↳ dva nova vektora:

služe za interpolaciju:

$f^{(t)}$ - koliko zaboravljamo prethodne informacije

$g^{(t)}$ - koliko utječu novi izlazi

$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right) = \sigma\left(a_f^{(t)}\right)$$

$$g^{(t)} = \sigma\left(W_g h^{(t-1)} + U_g x^{(t)} + b_g\right) = \sigma\left(a_g^{(t)}\right)$$

↳ LSTM: „Long Short Term Memory"

↳ mreža ima više parametara jer učimo i vektore $f^{(t)}$ i $g^{(t)}$ (3 puta više parametara)

$$\boxed{s^{(t)} = f^{(t)} \odot s^{(t-1)} + g^{(t)} \odot \hat{s}^{(t)}}$$

↓

način „update"-anja skrivenog stanja

(vidi što je $\hat{s}^{(t)}$ na slajdovima)

↳ još trebamo jednu propusnicu

- propusnica - za memorije u računanje izlaza: oznaka $q^{(t)}$
- računa se analogno $f^{(t)}$ i $g^{(t)}$, pa sada imamo 4 puta više parametara ☹

$$\boxed{h^{(t)} = q^{(t)} \odot \tanh\left(s^{(t)}\right)}$$

↳ ukratko, ideja svega ovoga je otežati prestup skrivenom stanju - mreža treba naučit nešto stvarno dobro da jako izmjeni skriveno stanje

- to smo realizirali pomoću 4 RNN-a umjesto jednog ☹

↳ kod LSTM-a, problem nestajanja gradijenata i dalje postoji, ali se dulje informacije prenose
- gradijenti ne nestanu tako brzo kao kod običnih dubokih RNN-ova

⇒ ŠPIJUNKE:

↳ propusnice ne vide stanje ćelije, pa bi im bilo dobro to omogućiti

↳ engl. „peepholes"

↳ obično se uključi s nekim booleanom u postojećim radnim okvirima za duboko učenje prilikom instanciranja samog LSTM-a

⇒ PROPUSNA POVRATNA ĆELIJA:

↳ dosta složenije i ima puno formula (samo okvirno treba znati)

↳ sada pokušavamo brisati stanje tako da dodamo propusnicu resetiranja

↳ rad dosta dobro!

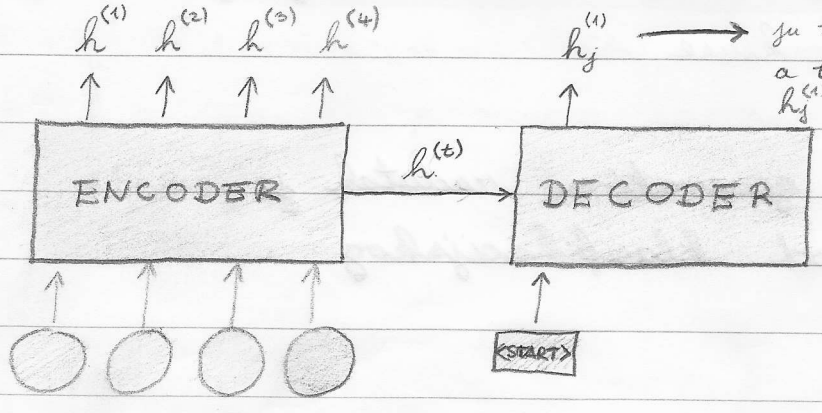
⇒ vratimo se na primjer strojnog prevođenja:

↳ SLIJED-U-SLIJED:

- imamo dvije mreže - jedna koja čita i jedna koja prevodi (mreža koja prevodi ne može biti dvosmjerna, pa je mala greška na slajdovima)
- na slajdu 57 pogledaj kako mreža generira izlazni niz (imaju 3 popularna načina)
  - ručno generirat niz koji ima najveću vjerojatnost
  - ručno gledat cijeli niz jer bi se inače najčešće generirala najčešća riječ jer to minimizira gubitak (najčešće bi se generiralo „the")

⇒ PAŽNJA (engl. „attention"):

↳ ideja je da se pamti „ideja" teksta i sam pokaz informacija koje smo pročitali jer tako i ljudi pamte informacije dok piše il čitaju članak



on traži neku informaciju, pa ju tražimo u ENCODERU (pročitanom nizu) a to radimo tako da množimo $h_j^{(1)}$ sa svima $h^{(i)}$ od encodera

↳ uzme najveći dobiveni iznos jer to daje najveću vjerojatnost

↳ nadamo se da će DECODER naučiti tu riječ u sličnom kontekstu kao ENCODER

$h_j^{(1)} \cdot h^{(i)} = a^t$

⇒ slajd 61 prikazuje interpretaciju ∇!

DOBRO SVOJSTVO

GENERATIVNE MREŽE

⇒ opet krenimo s motivacijom:

↳ pretpostavimo klasifikacijski radatak: slike banana i jabuka

↳ možemo radatak postavit kao generativni:

- nauči mrežu da generira umjetne slike jabuka i banana

↳ trenirama mrežu na označenim slikama i damo joj ulaz: „GENERIRAJ JABUKU", a ona uz pomoć nekog slučajnog šuma i naučenih značajki generira sliku jabuke

OVO JE POČETNA IDEJA KOJU NEĆEMO KORISTITI

⇒ kod klasifikacijskog radatka je dovoljno naučiti neke diskriminativne značajke klasa da bi ih mogli razlikovati

⇒ kod generativnog radatka, potrebno je naučiti sve značajke neke klase kako bi umjetno mogli generirati sliku te klase

⇓

stoga, generativni radatak je teži od klasifikacijskog

⇒ PRIMJENA GENERATIVNIH MODELA:

a) predtreniranje za diskriminativni problem

↳ na nelabeliranom skupu, naučimo mrežu da za danu ulaznu sliku generira sličnu takvu na izlazu (da bi to mreža mogla, ona je morala izvući van bitne značajke i jabuka i banana)

↳ sada, na sloj generativnog modela koj je pohranio bitne značajke stavimo potpuno povezan model (tada potpuno povezan sloj ima na ulazu samo bitne značajke)

b) umjetna inteligencija

↳ ako stroj može generirati sliku banane, onda vrlo vjerojatno ima jako dobro znanje o modelu banane (pa čak i 3D model?!)

⇒ PROBLEMI GENERATIVNIH MODELA:

• glavni problem je **evaluacija**:

„Što je uvjerljiva slika?"

⇒ modeliva se distribucija više varijabli: $p(\vec{x})$

↳ kod diskriminativnih modela, modelirali smo samo vjerojatnost klase uz primjer: $p(c|\vec{x})$

⇒ procjene distribucija u praksi ne želimo direktno nego trebamo uzorkovati

BITNI ALATI ZA GENERATIVNE MODELE

⇒ Markovljev lanac } OVO NAJČEŠĆE KOMBINIRAMO
⇒ Monte Carlo } (vid slajdove!)

⇒ Gibbsovo uzorkovanje:
- nasumične se uzorkuju pojedine varijable
- mora biti poznata uvjetna vjerojatnost

⇒ BOLTZMANOV STROJ:
- minimizira energiju (što god to bilo)
- neuroni su stohastički i binarni
- energetska funkcija:
  ↳ ovisi o stanjima neurona i težina

⇒ Hopfieldova mreža: (samo primjer)
- binarni neuron: sgn za aktivaciju $\{-1, 1\}$
- no, ovdje neuroni nisu stohastički
- to je obična neuronska mreža sa sgn za aktivacijsku funkciju
- zbog sgn:
  - otpornost na šum (teško promijeniti predznak)
  - otpornost na gubitak jednog neurona
  - gubitak informacije
- učenje: ako dva neurona pale zajedno, pojačaj njihovu vezu

- energetska funkcija: $E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} w_{ji} x_j x_i$

- promjena energija je uvijek negativna (ne treba računati izvod)
- problem je mali kapacitet:
  - ↳ rješenje je „raboravljanje" - smanjujemo povezanost neurona
    - ↳ to je skup lažnih stanja - njih želimo raboraviti i ponovno naučiti i time povećati kapacitet (napravimo mjesta za bitne stvari (značajke) tako što raboravimo lažna stanja)
- ova mreža ide u prvi lokalni optimum (vrlo vjerojatno nije globalni)

BOLTZMANOV STROJ

⇒ stohastička verzija Hopfieldove mreže

↳ stohastičnost omogućava bijeg iz lokalnog minimuma (ideja kao kod mutacije genetskih algoritama)

⇒ vjerojatnost je definirana pomoću energetske funkcije:

↳ IDEJA IZ FIZIKE:

- stanja više energije su manje vjerojatna

$$P(\vec{x}) \sim e^{-\frac{E(\vec{x})}{kT}} \qquad \sum_{\vec{x}} P(\vec{x}) = 1$$

⇒ razlika energija stanja 0 i 1 daje vjerojatnost stanja neurona:

- ako se T ignorira, dobivamo da je vjerojatnost za stanje 1 razvoj sigmoida:

$$x_j = \begin{cases} 1, & \text{s vjerojatnošću } \dfrac{1}{1 + e^{-\sum_{i \neq j} w_{ij} x_i}} \\ 0, & \text{inače} \end{cases}$$

⇒ samo treba riješiti ideju:

- iz energije, izvući smo vjerojatnost da je stanje neurona 1 ∇

$$\frac{\partial}{\partial W_{ij}} \ln\left[\prod_{m=1}^{N} P(\vec{x}^{(m)}; W)\right] = \dots$$

⇒ **o gradijentu:**

↳ prvi član:

- Empirijska korelacija
- ista ideja: ako su neuroni istog stanja, povećaj veru između njih (korelacija)

↳ drugi član:

- korelacija prema modelu
- „očekivanje (prema modelu - što je verovatno da model generiše (kako model vidi))"
- no, on je definisan preko svih vrednosti, a to nam je nedostižno (nije moguće imati sve moguće slike jabuke)

↳ **ideja:** proceni pomoću Monte Carlo metode

↓

ideja učenja:

- pamti ono što je u skupu za učenje (PRVI ČLAN), a zaboravljaj sve (DRUGI ČLAN)

<PRVI ČLAN> − <DRUGI ČLAN>

↓ „wake" faza (budna faza) (tjera model prema skupu za učenje)

↓ „sleep" faza (faza spavanja) (malo tjera model od skupa za učenje)

VIDI SLAJD SA SLIKOM DISTRIBUCIJA

⇒ **skriveni neuroni**:

- pomažu mreži da bolje radi (dublji model)
- oznaka: h
- postoji i gradijent za te neurone i analogan je kao i za vidljive neurone opisan na prethodnoj stranici
- oni bi trebali čuvati značajke koje su bitne za ono što generiramo

↳ vidljivi sloj štrebla skup za učenje, a skriveni uči bitne značajke

⇒ NAPOMENA:

↳ Boltzmannov stroj je potpuno povezan po defaultu (i među slojevima i među neuronima istog sloja)

OGRANIČENI BOLTZMANOV STROJ

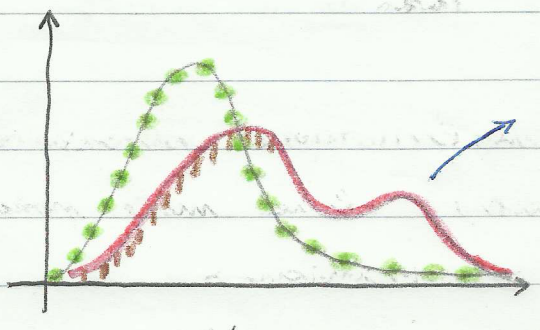
⇒ sve je isto s idejom, samo sada nemamo povezanost unutar slojeva
↳ ograničenje na veze

⇒ $p(\vec{x} \mid W)$ i $p(\vec{y} \mid W)$ su sada lakše za maksimizirati (vidi slajdove samo okvirno)

⇒ problem y i dalje **faza spavanja** (negativna faza):
↳ zahtjeva puno uzorkovanja (beskonačno?!)
↳ no, zbog ograničenih veza možemo uvesti modifikaciju negativnog fazi:

CONTRASTIVE DIVERGENCE

↳ aproksimacija negativne faze
jer nejdemo do termalnog ekvilibrija
(ne uzorkujemo beskonačno)



BELIEF NETS

⇒ duboko povezani rijetki model

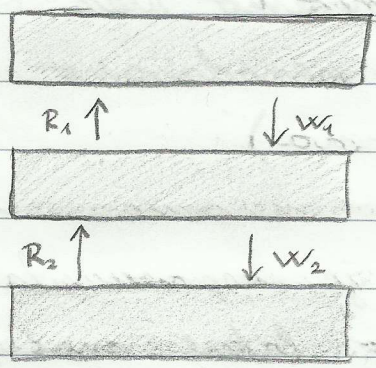
⇒ ista ima stohastičke neurone

⇒ ista maksimizira vjerojatnost vezana za treniranje

⇒ razlika:

↳ nema negativne faze ∇

↳ sleep faza služi za korekciju



kada ideš dolje, korigiraj težine za gore, a kada ideš gore, korigiraj težine za dolje

BEZ IZVODA: ovo maksimizira vjerojatnost skupa za učenje

⇒ pohlepno treniranje sloj po sloj:

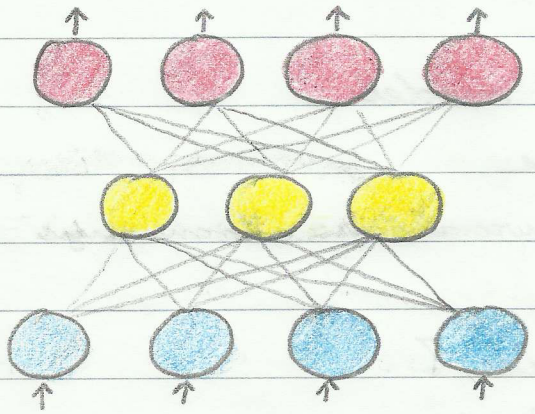
- treniraj jedan Boltzmannov stroj, zatim kada si zadovoljan, „stackaj" drugi na prvi i treniraj taj drugi itd.

⇒ ove mreže ne maksimiziraju vjerojatnost (ne može se matematički potvrditi), ali se može pokazati da one poboljšaju donju granicu:

$$p_2(\vec{x}) = L + \text{„nešto pozitivno"}$$

⇒ ovo je stack Boltzmannovih strojeva

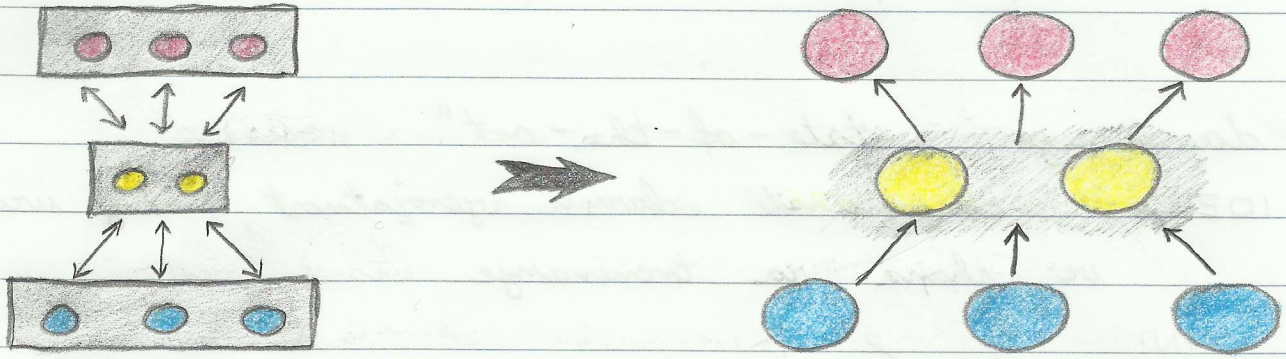
AUTOENKODERI



⇒ ideja je da na izlazu kopiramo ulaz
↳ nova, radimo nenadzirano učenje
(kao i kod RBM-ova)

⇒ ograničenje skrivenog sloja je zapravo sila koja tjera autoenkoder da nauči samo bitne značajke
↳ to ograničenje zove se regularizacija
↳ najčešće metode ograničavanja / regularizacije:
1) manji broj elemenata u skrivenom sloju
(time dobivamo KLASIČNI AUTOENKODER)
2) veranje težina ulaznog i izlaznog sloja
3) dodaje se šum u ulazne podatke
(time se dobiva DENOISING AUTOENKODER)
4) ako se ulaz malo promjeni, ne mijenjaj skriveni sloj (time dobivamo CONTRACTIVE AUTOENKODER)
↳ gledamo veličinu promjene ulaza (derivacija)
5) rijetkost broj očekivane aktivacija skrivenih neurona
(time dobivamo RIJETKE / SPARSE AUTOENKODERE)

⇒ NAPOMENA: primjeti da je autoenkoder zapravo autoenkoder zapravo jednoslojni RBM

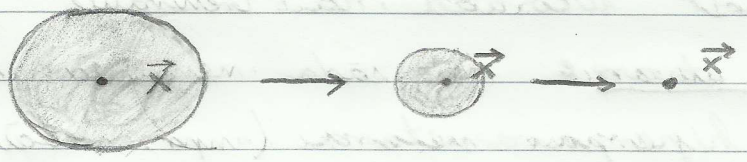


→ ideja sažimajućeg (contractive) autoenkodera:

↳ mala promjena ulaza ne daje promjenu stanja

↳ prostor „sažmu" oko svoraka za učenje

(„što je približno $\vec{x}$ je zapravo $\vec{x}$")



↳ to radimo tako da imamo regularizacijski član koji smanjuje derivacije:

– ov koji imaju male derivacije su vrlo vjerojatno beskorisni i možemo ih urbaditi

⇒ autoenkodere možemo ulančati (duboki autoenkoderi)

↳ povećamo kapacitet

VARIJACIJSKI AUTOENKODERI

⇒ danas je „state-of-the-art"

⇒ IDEJA: maksimizirati izlaznu vjerojatnost svakog uzorka iz skupa za treniranje

↓

da bi to mogli, tražimo i varijable $\vec{z}$ skrivenog sloja koje omogućuju tu maksimizaciju

⇒ primjeti da do sada nemamo nikakve veze s klasičnim autoenkoderom već samo nekako želimo maksimizirati izlaznu vjerojatnost uzoraka za treniranje

↳ <u>dobra stvar</u>: za sada nemamo nikakvih hiperparametara (npr. veličina skrivenog sloja)

ENKODER

$$p(\vec{x}) = \int p(\vec{x}|\vec{z};\theta)\, p(\vec{z})\, d\vec{z}$$

↓

da bi ovo maksimizirali, trebamo znati $p(\vec{z})$

↳ stoga, pretpostavljamo normalnu razdiobu latentnih značajki i neovisne su:

$$\boxed{p(z) = N(0,1)}$$

DEKODER

$p_\theta(x|z) = N(\mu_x(z), \sigma_x(z))$

↓

pretpostavljamo i normalnu razdiobu našeg dekodera (očekujemo normalnu razdiobu na izlazu autoenkodera)

⇒ sada imamo sve za varijacijski autoenkoder:

↳ maksimiziramo $\log(p(x))$

(ako to ranije nismo znali, vidi slajdove)

$$\log(p(x)) = L + D_{KL}(q(z|x) \| p(z|x)) \geq L$$

neka donja varijacijska granica

nešto pozitivno

povećali smo donju varijacijsku granicu (nadamo se da smo sve povećali jer smo donju granicu povećali)

⇒ INTERPRETACIJA VARIJACIJSKOG AUTOENKODERA:

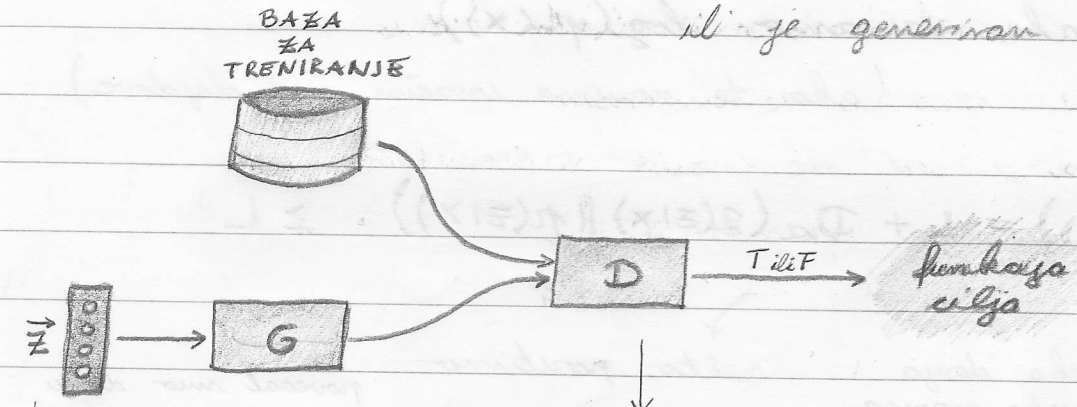
↳ enkoder uči koliko šuma σ_z smije dodat, a da je izlaz još uvijek dobar

⇒ nedostatak autoenkodera je da ne maksimiziramo $p(\vec{x})$ već maksimiziramo donju varijacijsku granicu

GENERATIVE ADVERSARIAL NETWORKS

⇒ dvije suprotstavljene mreže (MINIMAX IGRA):

- GENERATOR (G) - pokušava vjerno generirati uzorke
- DISKRIMINATOR (D) - pokušava prepoznati je l' dani uzorak iz skupa za učenje il' je generiran od generatora



z ... skriveni vektor iz neke željene distribucije

on u stvari radi nadzirano učenje jer znamo je l' slika od generatora il' iz skupa za treniranje

⇒ diskriminator na izlazu daje vjerojatnost da je uzorak došao iz skupa za treniranje

⇒ ako generator nije lic dobar, šalje mu se povratna informacija (diskriminator (učitelj) mora biti dobar da bi generator (učenik) bio dobar)

⇒ treniva se naizmjenično, stohastičkom gradijentnom spustom

⇒ <u>optimalan diskriminator</u>:

$$D_G = \frac{p_{data}}{p_{data} + p_g}$$

$> 0.5 \Rightarrow$ PRAVI UZORAK!

$< 0.5 \Rightarrow$ „FAKE" UZORAK!

↳ procjenjuje razliku između stvarne distribucije i distribucije generatora (ako je generator optimalan, ovo je stalno 0.5, pa bacamo novčić na svakoj odluci)

⇒ problem: <u>mode collapse</u> - želimo generirati slike automobila, a generator stalno generira slike čolba što je auto (diskriminator je nadovoljan), ali ne želimo samo čolbore, već želimo raznolikost auta

⇒ GAN direktno minimizira funkciju cilja

- VAN to nije mogao (želeli smo maksimizirati $p(\vec{x})$, a naravno smo maksimizirali donju varijacijsku granicu)

⇒ NESTABILNOST GAN-a:

↳ jedna od komponenti (diskriminator ili generator) je predobra i guši učenje druge

↓

rješenje je **Feature Matching**

⇒ MINIBATCH DISCRIMINATION:

↳ rješava mode-collapse

↳ ugradnja mehanizma u diskriminator kojim on može raspoznati mode-collapse unutar jedne grupe za učenje

DODATNO O REGULARIZACIJI

⇒ ideja je smanjiti varijancu i povećati pristranost
- povećava performanse na testu, a na trainu je čak moguće pogoršanje

⇒ L2 regularizacija ima isti utjecaj kao i da smo raširili podatke
↳ pogledajmo primjer linearne regresije:

$$\vec{\hat{y}} = X\vec{w}$$



$$J = (\vec{\hat{y}} - \vec{y})^2 = (X\vec{w} - \vec{y})^T (X\vec{w} - \vec{y})$$

$$J = \vec{w}^T X^T X \vec{w} - 2\vec{y}^T X\vec{w} + \vec{y}^T\vec{y} \quad / \frac{d}{d\vec{w}}$$

IZVOD DERIVACIJE KVADRATNE FORME!

$$\frac{dJ}{d\vec{w}}$$

$$\vec{w}' = X\vec{w} \Rightarrow \frac{d\vec{w}'^T\vec{w}'}{d\vec{w}} = \frac{d\vec{w}'^T\vec{w}'}{d\vec{w}'} \cdot \frac{d\vec{w}'}{d\vec{w}} = 2\vec{w}'^T \cdot X$$

$$= 2\vec{w}^T X^T X$$

$$\frac{dJ}{d\vec{w}} = 2\vec{w}^T X^T X - 2\vec{y}^T X = 0 \quad / \cdot 2$$

$$\vec{w} X^T X - X^T\vec{y} = 0$$

$$\vec{\hat{w}} = (X^T X)^{-1} X^T \vec{y}$$

OKRENI!

↳ sada, napravimo ovo za L_2 regularizirani gubitak:

$$J_R = (\vec{\hat{y}} - \vec{y})^2 + \alpha \vec{w}^T \vec{w} \quad \Big/ \frac{\partial}{\partial \vec{w}}$$

$$\frac{\partial J_R}{\partial \vec{w}} = 2\vec{w}^T X^T X - 2\vec{y}^T X + 2\alpha \vec{w}^T = 0 \quad /:2$$

$$\vec{\hat{w}}(X^T X + 2\alpha I) - \vec{y}^T X = 0$$

$$\vec{\hat{w}} = \left[(X^T X + \alpha I)^{-1}\right]^T \vec{y}^T X$$

↳ $X^T X$ je simetrična pa je $(X^T X)^T = X^T X$

$$\vec{\hat{w}} = (X^T X + \alpha I)^{-1} \vec{y}^T X$$

↓

zapravo podacima dajemo šum!

⇒ djelomično nadzirano učenje:

- minimiziraj dva gubitka:
  - a) nadzirani: $-\log P(Y|X)$
  - b) nenadzirani: $-\log P(X)$
- značajke naučene na nenadziranom dijelu mogu pomoći klasifikatoru u nadziranom dijelu (GAN-ovi ovo primjenjuju)

⇒ bagging :

- uzorkuj podatke u M skupova
- nauči M modela i na kraju napravi glasanje da bi dovio odluku (ansambl)
  ↳ npr. usrednji odluku od svih
- zašto ovo radi?
  ↳ pa, pogledajmo:

$$E_a = \frac{1}{M} \sum_i \varepsilon_i$$

E_a ↓ pogreška ansambla

OČEKIVANJE POGREŠKE ANSAMBLA

$$E(E_a) = E\left(\frac{1}{M} \sum_i \varepsilon_i\right)^2 = \frac{1}{M^2}\left[E\left(\sum_i \varepsilon_i^2\right) + E\left(2 \cdot \sum_{i<j} e_i e_j\right)\right]$$

$$= \frac{1}{M^2}\left[\sum_i E(\varepsilon_i^2) + 2 \sum_{i<j} E(e_i e_j)\right] = (*)$$

↳ pretpostavka: dijagonalni elementi su jednaki „v", a nedijagonalni su jednaki „c"

$$(*) = \frac{1}{M^2}\left[M v \cdot 2 + 2 \cdot \frac{M(M-1)}{2} \cdot c\right] =$$

$$= \frac{1}{M} v + \frac{M-1}{M} c$$

↳ pogledaj ekstremne slučajeve na slajdu 35
(što se dešava kada je $v = c$ (HINT: bolje je $\frac{1}{5}$))

⇒ dropout:

- kao bagging velikog broja modela, al bez da učimo taj veliki broj modela
- iz početnog modela formiramo nove gašenjem nekih aktivacija

↓

time dobivamo varijacije modele (odnosno, dobivamo ansambl pod-mreža)

- dobivamo eksponencijalan broj modela u ovisnosti o broju čvorova na kojem gasimo težine

⇒ danas, najpopularniji je **batch-norm**!

ZAD: Zadatak s droputom na slajdu 51 ▽

$$W = \begin{bmatrix} -0.1 & 0.2 \\ 0.4 & -0.3 \\ -0.6 & 0.5 \end{bmatrix} \quad b^T = [0.2 \ \ -0.2] \quad x = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$p(\mu_{x_1}) = 0.2 \qquad p(\mu_{x_2}) = 0.5 \qquad p(\mu_{x_3}) = 0.7$$

$$Y = softmax(W^T \vec{x} + \vec{b})$$

