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

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

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 $\rightarrow$ 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*:

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

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 ispuni *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 utvrdimo 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 iščeznu (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:

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

Pojedinačno učenje, grupno učenje i nešto između

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

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

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

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

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 $\sigma/\sqrt{n}$ 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

- optimizacijski postupci koji koriste čitav skup uzoraka za učenje nazivaju se *grupni* (engl. *batch*)
- optimizacijski postupci koji koriste jedan po jedan uzorak iz skupa uzoraka za učenje nazivaju se *pojedinačni* (engl. *online*) ili stohastički (engl. *stohastic*)
- optimizacijske 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 $\rightarrow$ precizniji gradijent (ispodlinearno 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 grupa 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) $\rightarrow$ 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: robusno i radi i s malim veličinama grupa
- 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 uzoraka 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 uzoraka u skupu za učenje i tim se redosljedom 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 redosljedom
  - 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

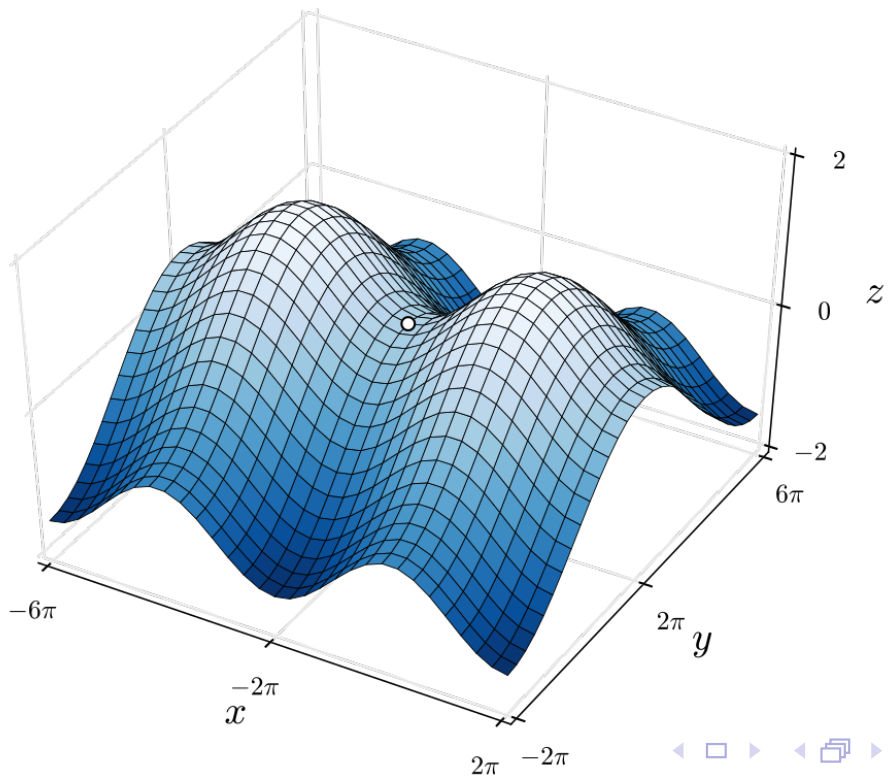
Funkcija gubitka kod neuronskih mreža nije konveksna: ima mnoštvo lokalnih minimuma

- 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 $\rightarrow$ 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

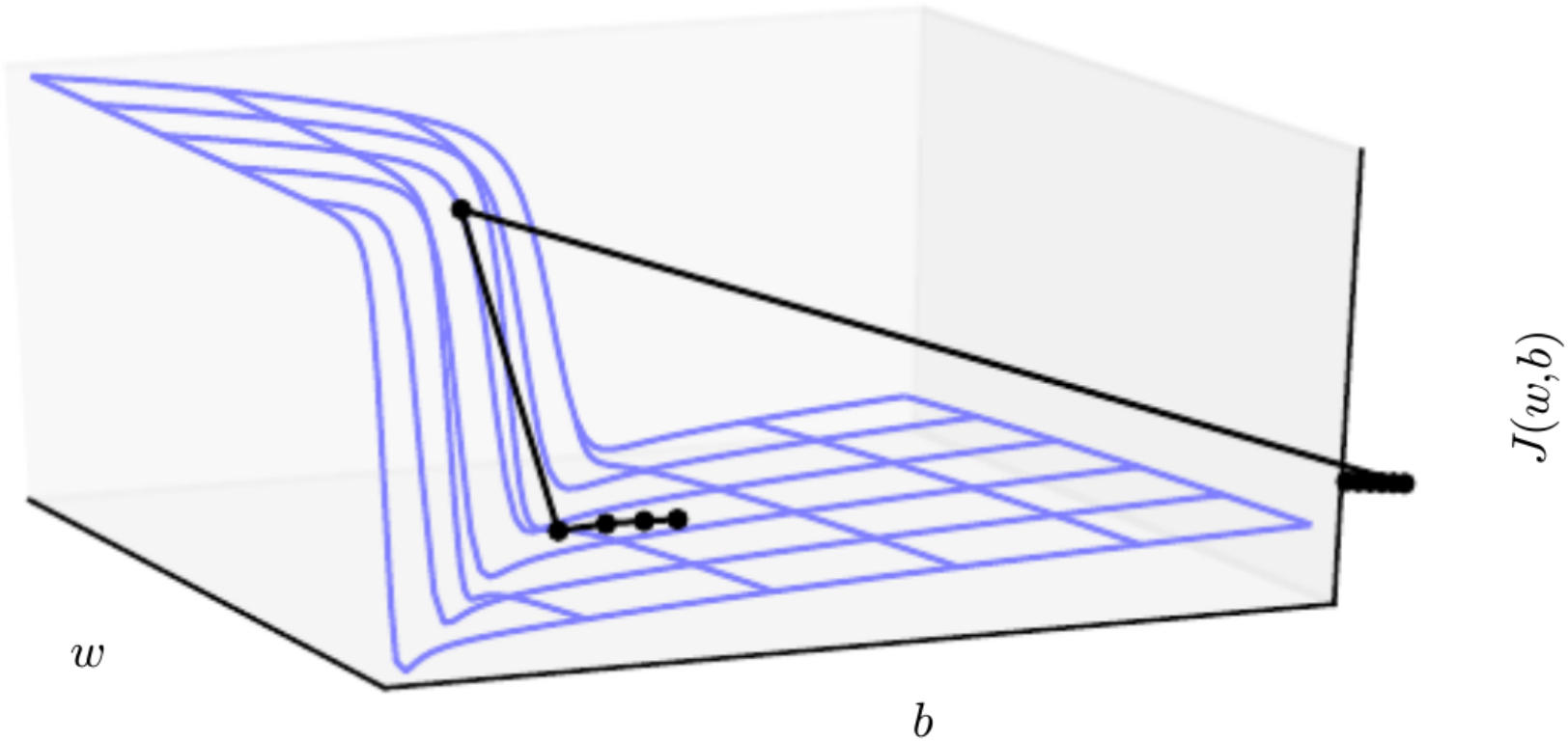


Platoi, sedla i drugo

- 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) $\rightarrow$ to je problem!

Litice i eksplodirajući gradijenti

Neuronske mreže s mnogo slojeva mogu imati strme regije u parametarkom 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:

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

tada je:

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

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

Dugotrajne ovisnosti

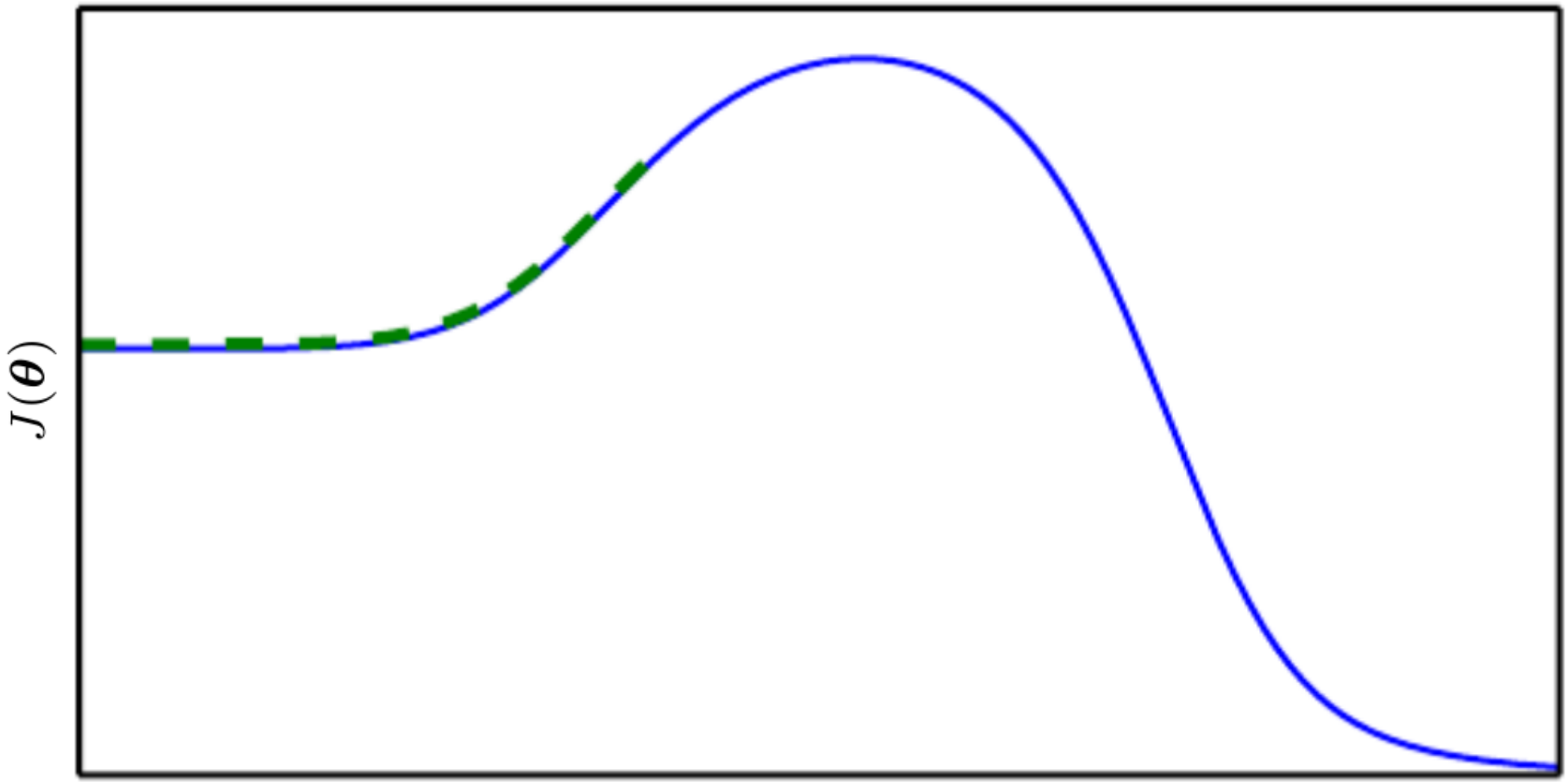
- gradijenti u takvom grafu se također skaliraju s $diag(\lambda)^t$:
  - mogu eksplodirati (engl. *exploding gradient problem*)
  - mogu nestati (engl. *vanishing gradient problem*)

Neprecizni gradijenti

- optimizacijski algoritmi pretpostavljaju da možemo točno odrediti gradijent (a po potrebi i H); u praksi: uzorkujemo i aproksimiramo
- funkcija gubitka koju minimiziramo može biti netraktabilna $\rightarrow$ gradijent je također $\rightarrow$ 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

Stohastički gradijentni spust

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

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

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  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: Stohastic 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)

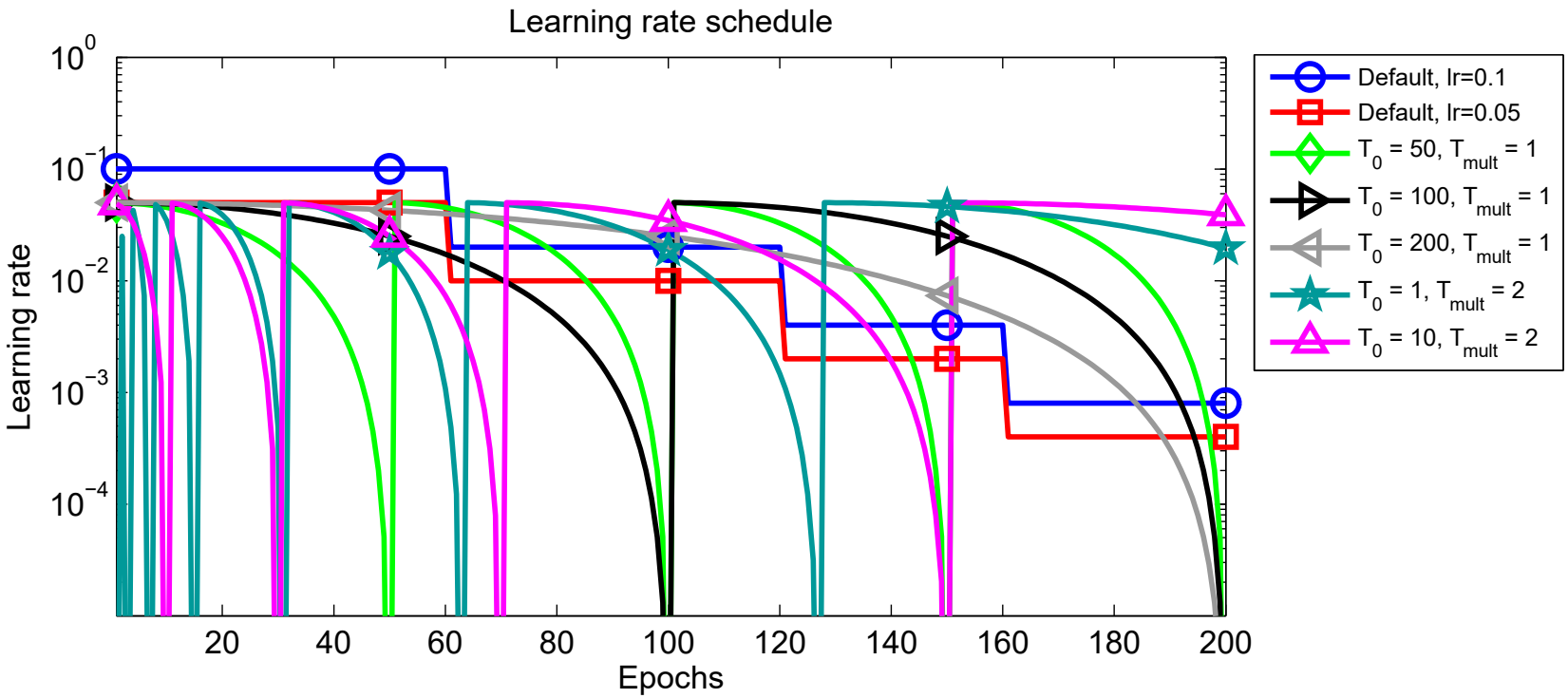
Uporaba restarta kod promjene stope učenja

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

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

- 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:

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

$$\theta \leftarrow \theta + v \tag{8}$$

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 poravnati.

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 ćemo faktor povećati korak:

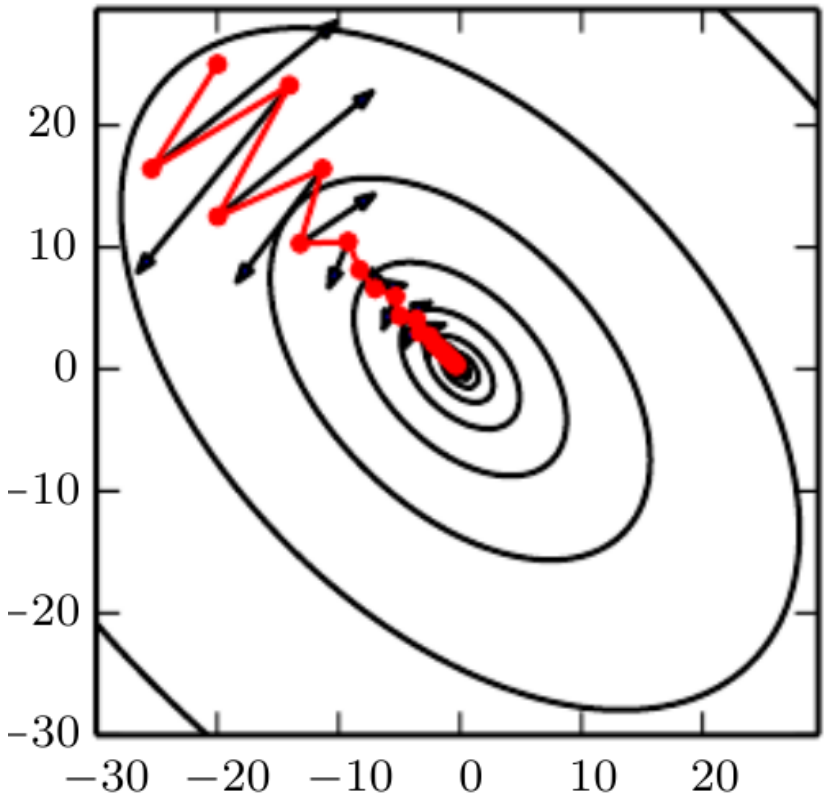
- Neka je $\alpha = 0.99$: $\frac{\epsilon \cdot \|g\|}{1-\alpha} = \frac{\epsilon \cdot \|g\|}{1-0.99} = \frac{\epsilon \cdot \|g\|}{0.01} = 100 \cdot \epsilon \cdot \|g\|$
- Neka je $\alpha = 0.9$: $\frac{\epsilon \cdot \|g\|}{1-\alpha} = \frac{\epsilon \cdot \|g\|}{1-0.9} = \frac{\epsilon \cdot \|g\|}{0.1} = 10 \cdot \epsilon \cdot \|g\|$
- 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 $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Nesterov moment

Modifikacija osnovnog algoritma učenja s momentom:

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

$$\theta \leftarrow \theta + v \tag{10}$$

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 $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding labels $\boldsymbol{y}^{(i)}$.
Apply interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient (at interim point): $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
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 (*biase*) 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
  $\rightarrow$ 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:

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

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

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

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) $\rightarrow$ 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

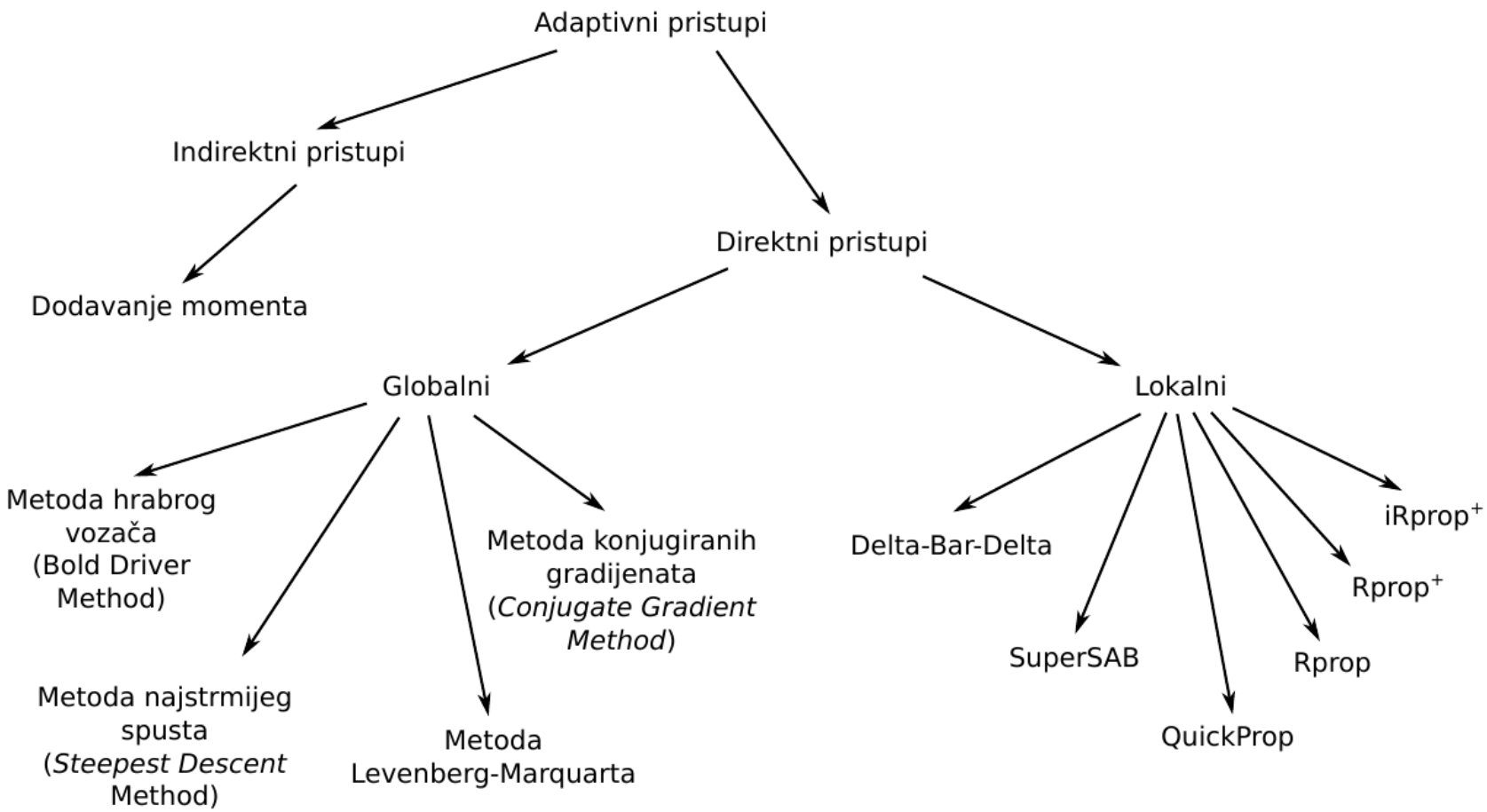
Složeniji postupci inicijalizacije mogu uključivati *predtreniranje* (nenadzirano) ili treniranje mreže da nad istim skupom podataka

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
  $\rightarrow$ 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 derivacije dugo 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 $\boldsymbol{\theta}$

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

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

while stopping criterion not met **do**

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

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

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

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

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

RMSProp

- mijenja akumuliranje gradijenata tako da dodaje eksponencijalno prigušenje starih vrijednosti - novije vrijednosti imaju veći utjecaj
  $\Rightarrow$ 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 $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers.
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute parameter update: $\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

RMSProp + Nesterov moment

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$
Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$
Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
end while

Adam

- Adam - od Adaptive Moments
- prati uprosječeno kretanje gradijenta i kvadrata gradijenta uz eksponencijalno zaboravljanje
- moment je implicitno ugrađen u praćenje kretanja gradijenta
- korekcija se radi prema uprosječenom 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 $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  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
    - $\mathbf{r} \leftarrow \max(\rho_2 \cdot \mathbf{r}, |g|)$; 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 $\hat{\mathbf{r}}_t = \max(\hat{\mathbf{r}}_{t-1}, \mathbf{r}_t)$

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 linearno 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 $a^i_{x,y}$
  - 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:

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

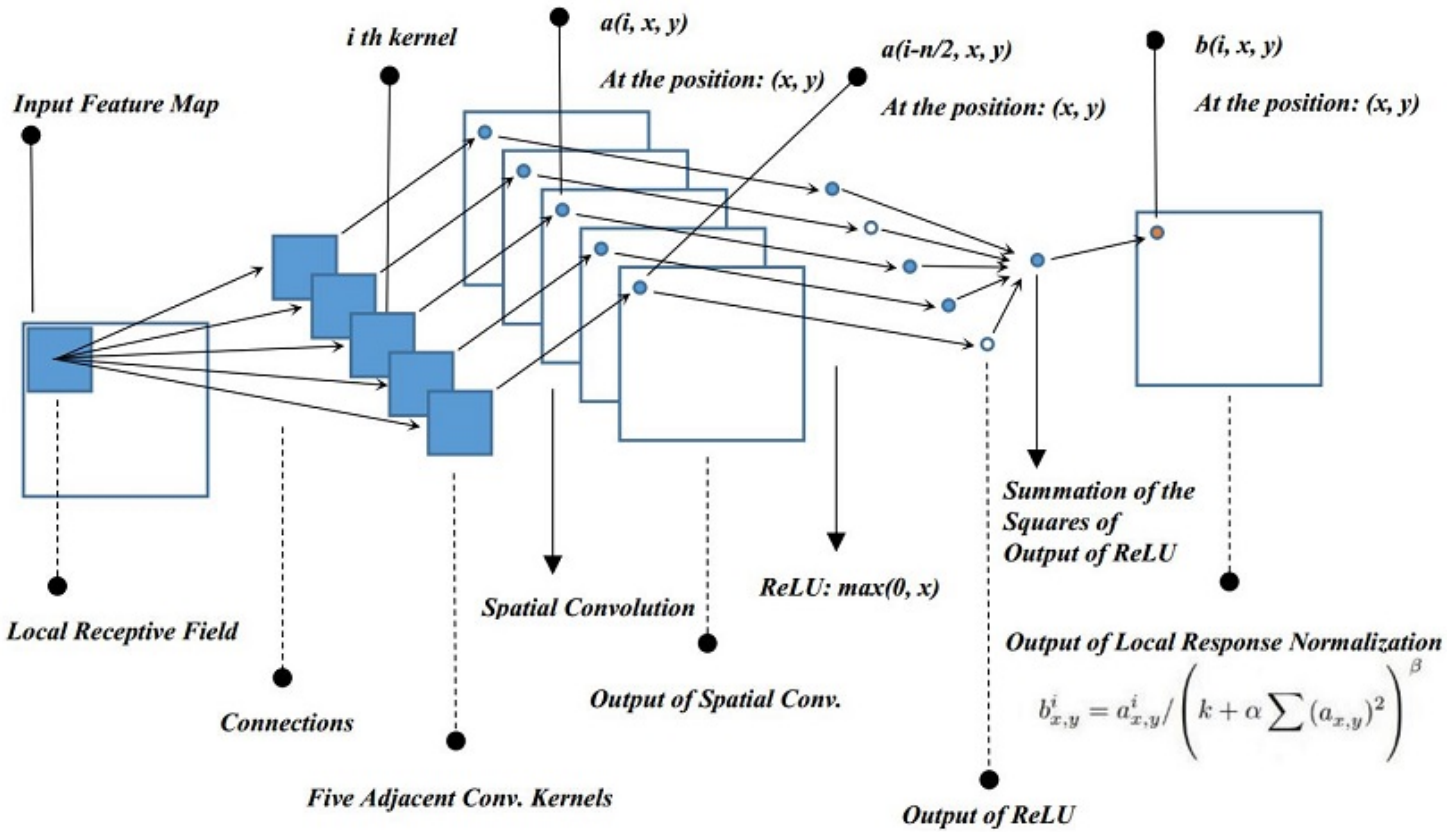
  - $b^i_{x,y}$ postavljamo kao "stvarni" izlaz promatranog neurona
  - npr. $\beta = 0.75$, $\alpha = 10^{-4}$, $k = 2$, $n = 5$ (hiperparametri)

Lokalna normalizacija odziva neurona

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

- ako je izlaz $a^i_{x,y}$ 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

  $\rightarrow$ 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 $F_2(., \Theta_2)$: 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 jako 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

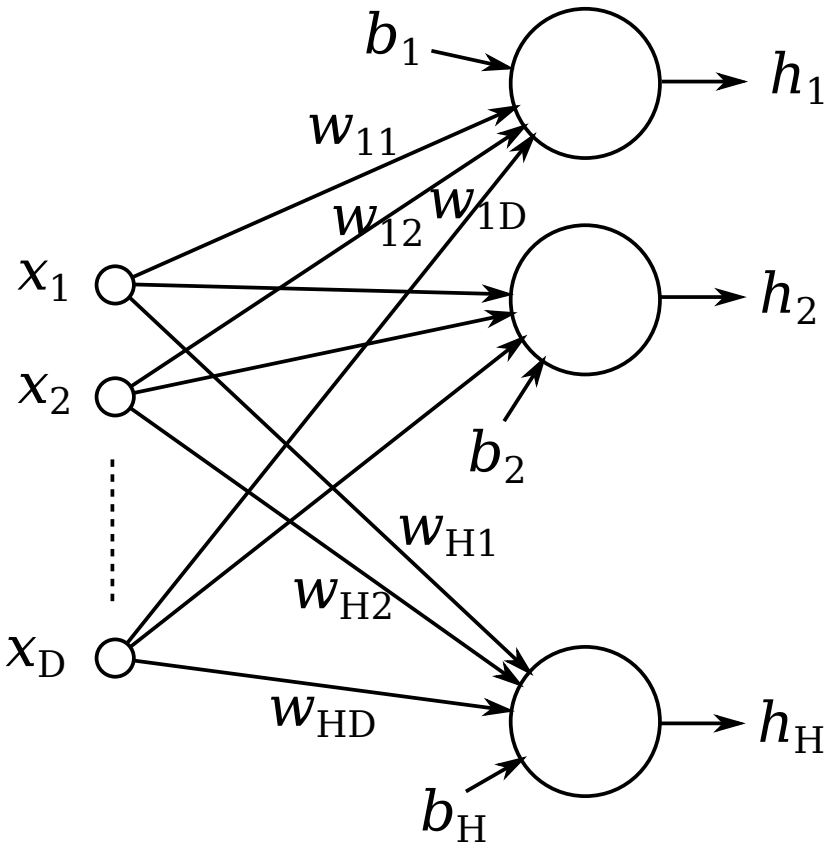
- *Normalizacija nad grupom*: normalizacija izlaza svakog neurona radi se na temelju izlaza koje taj neuron ima kada se na ulaz mreže dovede svaki od uzoraka za učenje iz minigrupe
- znači, ne normaliziraju se:
  - odzivi neurona međusobno unutar istog sloja
  - odzivi neurona u lokalnom susjedstvu
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

Normalizacija nad grupom

- Početna ideja: neka smo u minigrupu izvukli uzorke $\{x_1, \ldots, x_m\}$; 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 $o_1, \ldots, o_m$ izračunamo srednju vrijednost i standardno odstupanje pa skup preslikamo u normalizirani $\hat{o}_1, \ldots, \hat{o}_m$: 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:



D je dimenzionalnost ulaza, H je broj neurona

Normalizacija nad grupom: u neuronskoj mreži

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

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

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

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

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

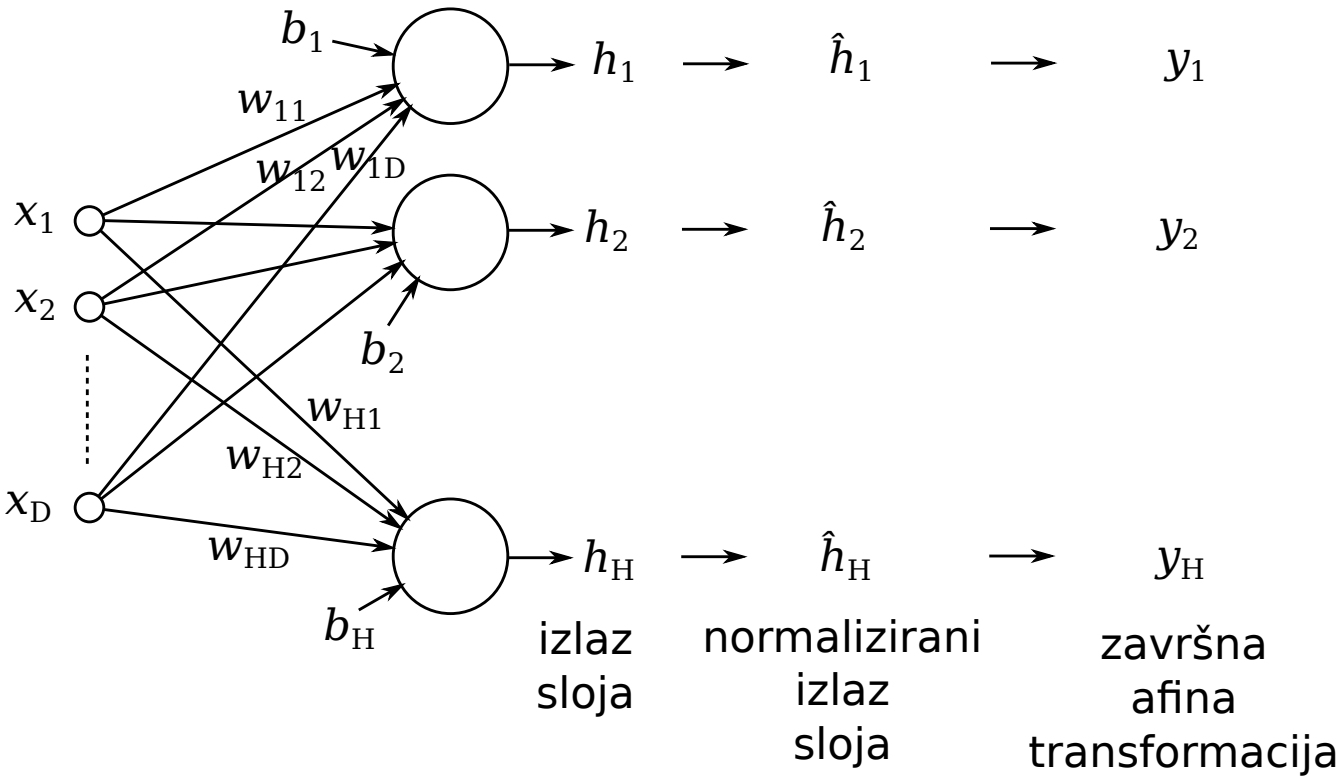
- Dobivamo matricu h

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

  (dimenzije $N \times H$)
- U mreži bismo svaki element još propustili kroz nelinearnost (npr. 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 $l \in \{1, \ldots, H\}$ 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:

$$y_{k,l} = \gamma_l \cdot \hat{h}_{k,l} + \beta_l$$

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_l i β_l 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):

$$a = ReLU(y)$$

gdje je ReLU(.) primijenjen na svaki element matrice zasebno, pa generira matricu a iste dimenzionalnosti kao i matrica y:

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

- 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 eksploataciju, 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 uzmemo prosječne vrijednosti dobivene temeljem minigrupa:

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

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

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 odabrane 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:

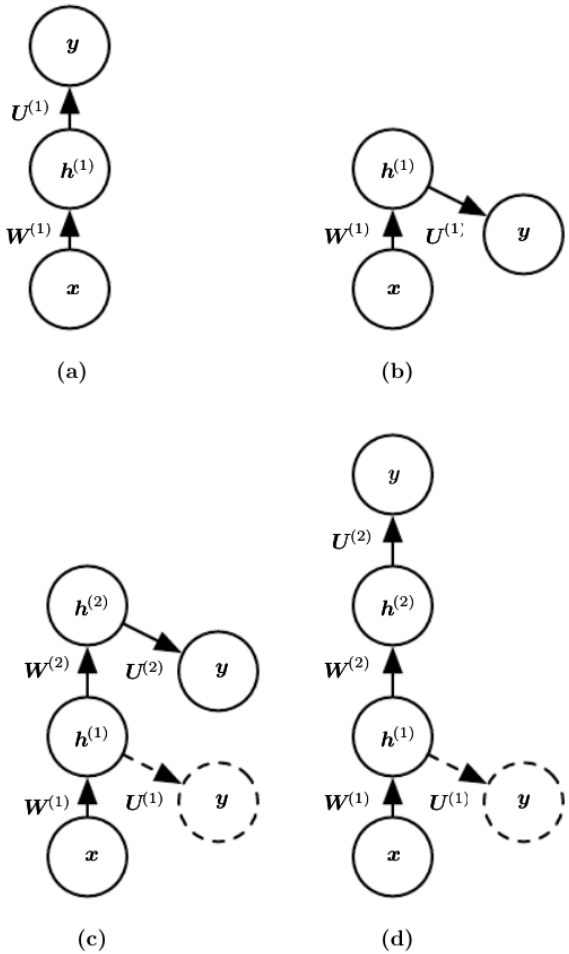
- u praksi, tijekom učenja se koriste pokretni prosjeci srednje vrijednosti i varijance (*moving average*)
- za najbolje rezultate posljednje epohe učenja mogu se provesti sa "smrznutom" populacijskom statistikom
- za najveću brzinu zaključivanja normalnizacija nad grupom može se fuzionirati s prethodnom konvolucijom
- alternativno objašnjenje zašto normalnizacija 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 rješava 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



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.)

1. 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)
2. takvu istreniranu mrežu nazivamo *mrežom učiteljem*
3. 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.)

1. (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)
   1. uvodimo parametar $\tau > 1$
   2. neka je $\mathbf{P}_T^\tau = softmax\left(\frac{\mathbf{a_T}}{\tau}\right)$ omekšani izlaz mreže učitelja a $\mathbf{P}_S^\tau = softmax\left(\frac{\mathbf{a_S}}{\tau}\right)$ omekšani izlaz mreže učenika za isti ulazni uzorak
2. funkciju gubitka sada definiramo kao:

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

   gdje H označava unakrsnu entropiju
3. 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:

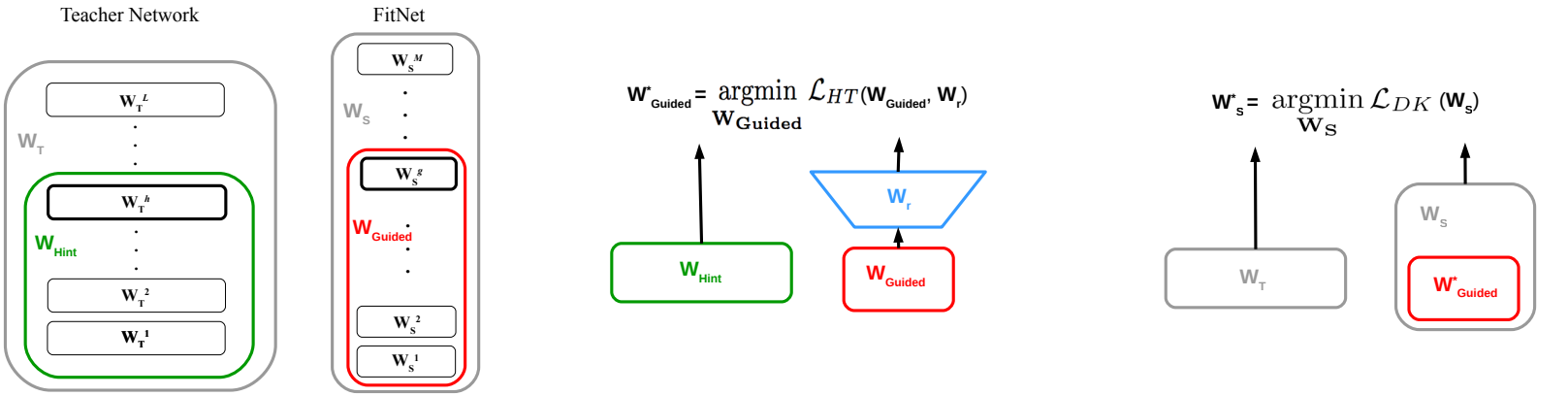
$$L_{HT}(\mathbf{W}_{guided}, \mathbf{W}_r) = \frac{1}{2}\|u_h(x; \mathbf{W}_{hint}) - r(v_g(x; \mathbf{W}_{guided}); \mathbf{W}_r)\|^2$$

Pristup *FitNets* (2015.)

1. sada krećemo u treniranje dubljeg ali tanjeg modela
   1. u drugom koraku odbacujemo ekstra težine potrebne za regresiju hintova i treniramo čitav model destilacijom
      1. prvi korak u određenom smislu radi predtreniranje težina dijela duboke mreže
      2. 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

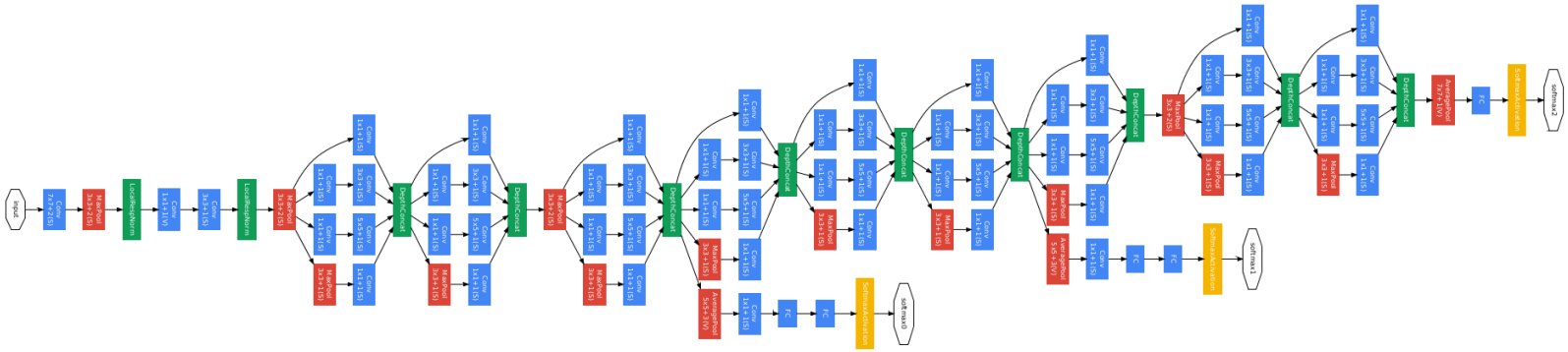
Izgradnja dubokih modela koje je lakše trenirati

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

?