Unaprijedni duboki modeli

Josip Krapac i Siniša Šegvić

Pregled

- Što su unaprijedni duboki modeli?
- Funkcija gubitka i izlazni slojevi.
- Aktivacijske funkcije u skrivenim slojevima.
- Univerzalna aproksimacija: dubina je važna.
- Backprop: izračun gradijenta kompozicije funkcije prosljeđivanjem greške unazad.

Pregled

- Što su unaprijedni duboki modeli?
- Funkcija gubitka i izlazni slojevi.
- Aktivacijske funkcije u skrivenim slojevima.
- Univerzalna aproksimacija: dubina je važna.
- Backprop: izračun gradijenta kompozicije funkcije prosljeđivanjem greške unazad.

Što su unaprijedni duboki modeli?

Duboka unaprijedna mreža (*eng. deep feedforward network*)

- osnovna formulacija dubokog modela
- mreža: sastoji se od većeg broja jednostavnijih premreženih funkcija
- osnovni oblik odgovara slijedu afinih transformacija s nelinearnom aktivacijom

Cilj:

- aproksimirati funkciju $\boldsymbol{y} = f^*(\boldsymbol{x})$ parametarskim modelom $\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{\Theta})$.
- model f mapira ulaz $\boldsymbol{x}$ u prediktirani izlaz $\hat{\boldsymbol{y}}$
- parametre $\boldsymbol{\Theta}$ združeno učimo na podatcima s kraja na kraj

Što su unaprijedni duboki modeli?

Detalji cilja:

- funkcija $y = f^*(x)$ opisuje stvarni odnos između ulaza x i izlaza y
- naš model $\hat{y} = f(x, \Theta)$ aproksimira stvarnu funkciju.
- želimo pronaći skup parametara Θ^* koji daje najbolju aproksimaciju funkcije $f^*(x) \approx f(x, \Theta^*)$.
- problem: ne znamo kako funkcija $f^*(x)$ izgleda za svaki x; znamo samo kako f izgleda na ograničenom skupu za učenje $\{(x_i, y_i)\}_{i=1}^N$.
- stalo nam je do ispravne **generalizacije**
- izbor modela mora biti prilagođen podatcima (usp. no free lunch theorem)

Što su unaprijedni duboki modeli?

Osnovna svojstva dubokih modela:

- informacija struji od ulaza prema izlazu, nema petlji.
- mogu se predstaviti kompozicijom jednostavnijih funkcija: $f(\boldsymbol{x}, \boldsymbol{\Theta}) = o(f_L(f_{L-1}(\cdots(f_1(\boldsymbol{x}, \boldsymbol{\Theta}_1)), \cdots), \boldsymbol{\Theta}_{L-1}), \boldsymbol{\Theta}_L))$,
- jednostavne funkcije f_i nazivamo slojevima
- svaki sloj ima točno jednu nelinearnu aktivaciju
- dubina modela (L): broj slojeva

Što su unaprijedni duboki modeli?

Model možemo izraziti i preko pomoćnih varijabli $\boldsymbol{h}_L, \boldsymbol{h}_{L-1}, \cdots \boldsymbol{h}_1$

$$\begin{aligned} \boldsymbol{h}_1 &= f_1(\boldsymbol{x}, \boldsymbol{\Theta}_1) \\ &\vdots \\ \boldsymbol{h}_{L-1} &= f_{L-1}(\boldsymbol{h}_{L-2}, \boldsymbol{\Theta}_{L-1}) \\ \boldsymbol{h}_L &= f_L(\boldsymbol{h}_{L-1}, \boldsymbol{\Theta}_L) \\ f(\boldsymbol{x}, \boldsymbol{\Theta}) &= o(\boldsymbol{h}_L) \end{aligned}$$

pomoćne varijable zovemo skrivenim ili latentnim značajkama

širina l-tog sloja: dimenzija značajki u l-tom sloju: $\boldsymbol{h}^l \in \mathbb{R}^{d_l}$.

Samo su ulaz $\boldsymbol{x}$ i izlaz $\boldsymbol{y}$ specificirani, model ima slobodu da iskoristi skrivene slojeve na način koji osigurava najbolju aproksimaciju funkcije.

Što su unaprijedni duboki modeli?

Osnovni oblik: lanac potpuno povezanih slojeva

- svaka f_i modelira elementarnu nelinearnu transformaciju: afino preslikavanje i nelinearnu aktivaciju $\boldsymbol{\sigma}$:

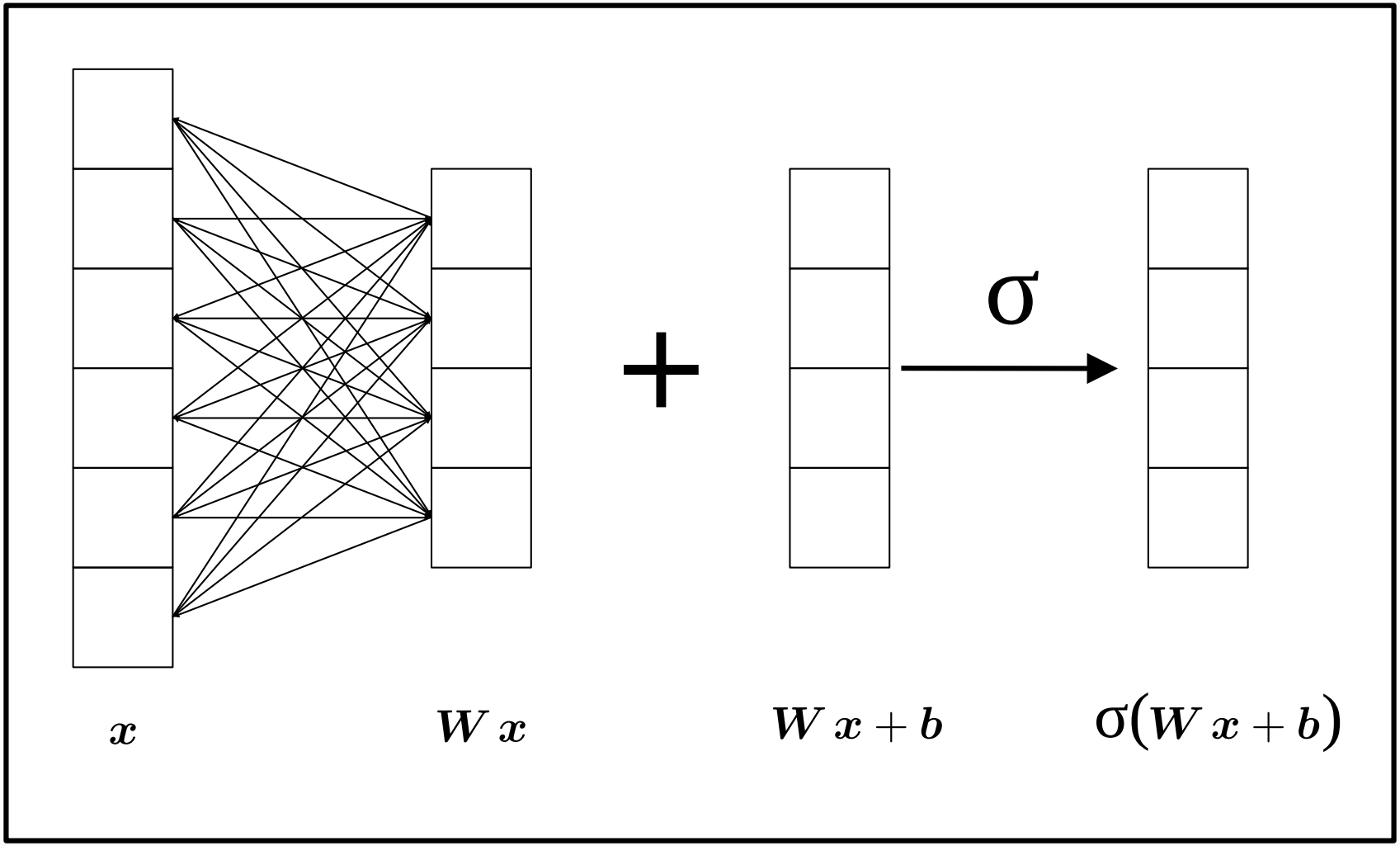
$$\mathbf{f}_k(\mathbf{h}_{k-1}) = \boldsymbol{\sigma}(\mathbf{W}_k\mathbf{h}_{k-1} + \mathbf{b}_k)$$

- Potpuno povezane slojeve moramo dobro upoznati:
  - osnova za složenije slojeve (npr. konvolucijske)
  - građevne jedinice složenijih arhitektura (npr. pažnja)

Drugi nazivi:

- (unaprijedni, duboki) **potpuno povezani model** (s afinim slojevima) (*eng. fully connected*)
- višeslojni perceptron (*eng. multi-layer perceptron*)
- (unaprijedna) umjetna neuronska mreža

Potpuno povezani sloj



$$f(x; W, b) = \sigma(W \cdot x + b)$$

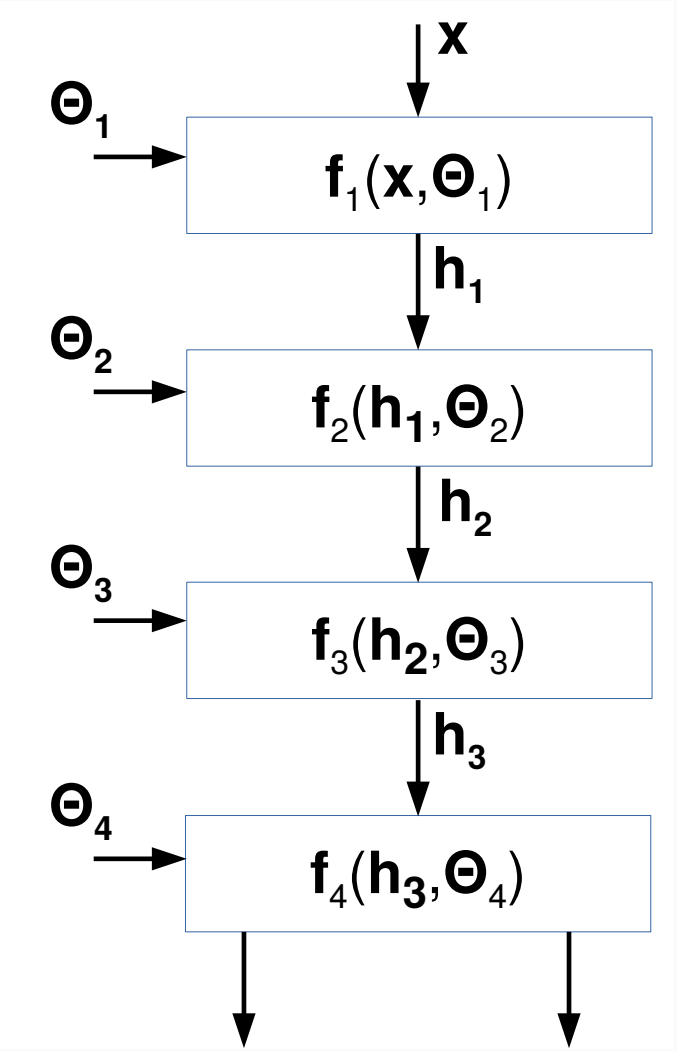
$$\sigma(s)_i = \sigma(s_i)$$

Potpuno povezani model

Zadatak: odrediti strukturu, jednadžbe i ukupan broj parametara potpuno povezanog modela za 2D podatke. ako znamo da su dimenzije slojeva: 5, 10, 5, 2.

Potpuno povezani model

Zadatak: odrediti strukturu, jednadžbe i ukupan broj parametara potpuno povezanog modela za 2D podatke. ako znamo da su dimenzije slojeva: 5, 10, 5, 2.



Što su unaprijedni duboki modeli?

Odnos prema umjetnim neuronskim mrežama:

- umjetne neuronske mreže proučavaju algoritme strojnog učenja koji su inspirirani ranim modelima ljudskog mozga
- s druge strane, duboko učenje proučava modele koji dobro generaliziraju na stvarnim podatcima

Linearni i nelinearni modeli

Pitanje: koliko dubok treba biti potpuno povezani model?

Zavodljiva ideja: L=1!

$$f(\boldsymbol{x}, \boldsymbol{\Theta} = (\boldsymbol{w}, b)) = \boldsymbol{\sigma}(\boldsymbol{w}^\top \boldsymbol{x} + b)$$

- **prednost**: uz uobičajene funkcije gubitka vodi na konveksnu optimizaciju
- **prednost**: garantirana konvergencija
- nedostatak: naš svijet nije linearan.

Linearni i nelinearni modeli

Ako jednoslojni model nije opcija, što nam preostaje?

- Rješenje: originalne značajke $\boldsymbol{x}$ mapirati nelinearnom funkcijom $\Phi(\boldsymbol{x})$ u drugi prostor u kojem klasifikaciju obavljamo linearnim modelom

$$f(\boldsymbol{x}, \Phi, \boldsymbol{\Theta} = (\boldsymbol{w}, b)) = \Phi(\boldsymbol{x})^\top \boldsymbol{w} + b$$

- Tri su dominantna načina konstrukcije funkcije $\Phi(\boldsymbol{x})$:
  - upotrijebiti generičku funkciju $\Phi(\boldsymbol{x})$,
  - ručno dizajnirati funkciju $\Phi(\boldsymbol{x})$,
  - naučiti funkciju $\Phi(\boldsymbol{x}|\boldsymbol{\Theta}_\Phi)$.

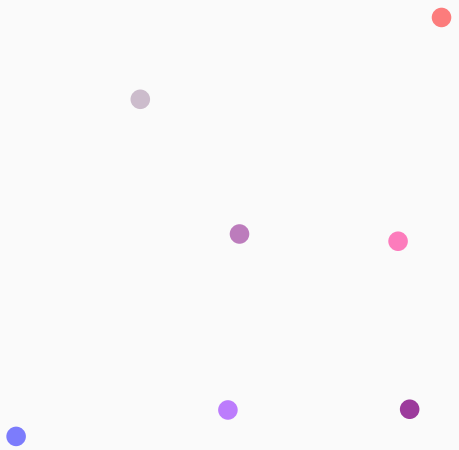
Generička funkcija za mapiranje značajki

Primjer: jezgrene funkcije.

- npr. RBF funkcija $k(\boldsymbol{x}, \cdot)$ implicitno preslikava ulaz u točku $\Phi(\boldsymbol{x})$ beskonačno-dimenzionalnog prostora.

Problem: takve funkcije pretpostavljaju lokalnu glatkoću (*eng. local smoothness prior*)

- nažalost, lokalna glatkoća nije dovoljno dobra kad je $\dim(\boldsymbol{x})=10^5$

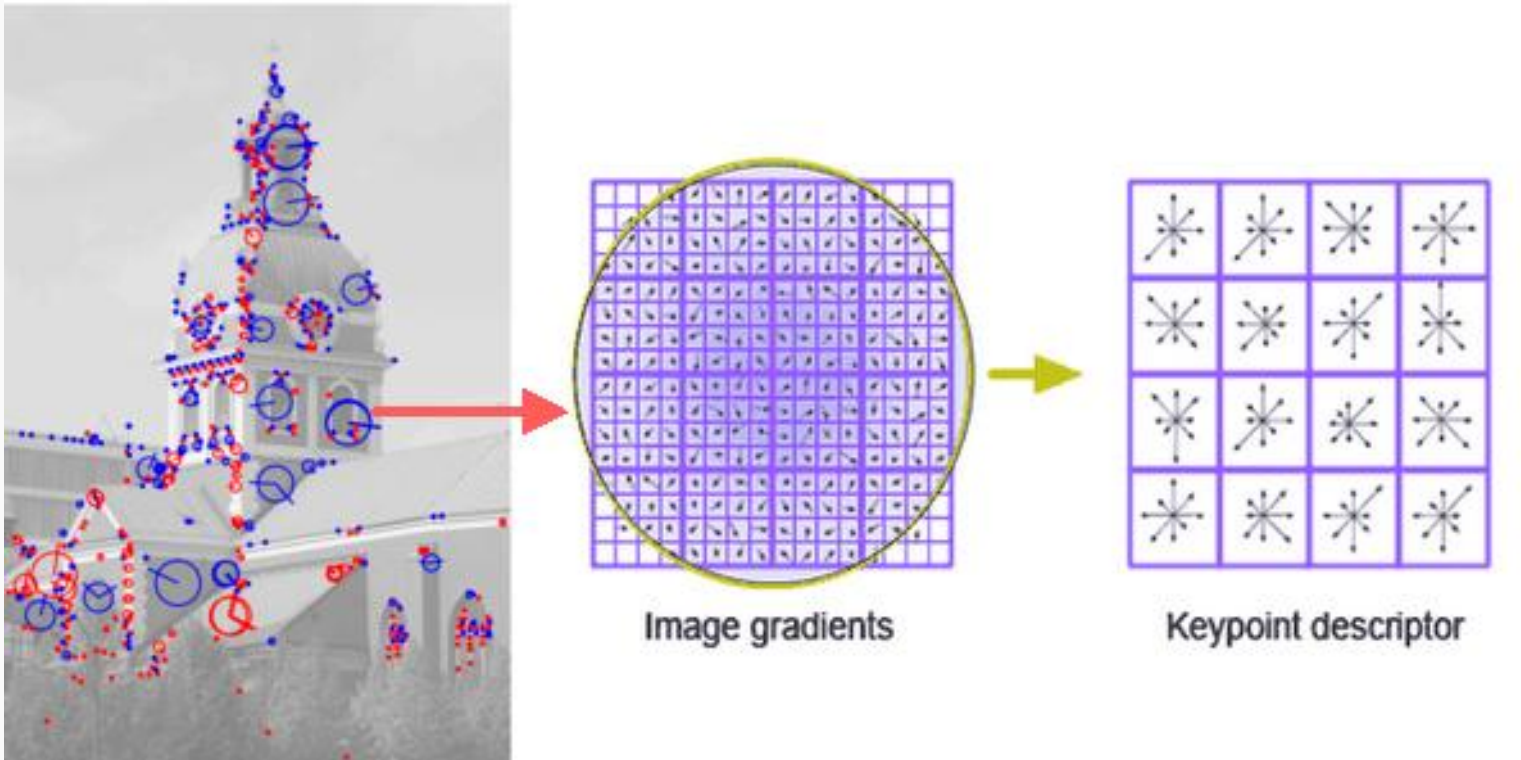


Ručni dizajn značajki

Primjer: SIFT deskriptor u računalnom vidu, vreće riječi u obradi prirodnog jezika, MFCC deskriptori u obradi govora.

Problem: zahtjeva domensko znanje, dugotrajan proces

Problem: (danas znamo) ograničeni rezultati



Učenje značajki

Jedino preostalo: naučiti funkciju $\Phi(\boldsymbol{x}|\boldsymbol{\Theta}_\Phi)$, s parametrima $\boldsymbol{\Theta}_\Phi$

Možemo probati slojeve učiti odvojeno: prvo značajke $\boldsymbol{\Theta}_\Phi$ (npr. nenadzirano), a tek onda klasifikator $\boldsymbol{w}, b$

- to bi radilo bolje od linearnog modela
- ali nije dobro skaliralo u stvarnim eksperimentima s više od dva sloja

Ostaje samo jedna opcija: učiti duboki model s kraja na kraj:

- spregnuto učenje $\boldsymbol{\Theta} = (\boldsymbol{w}, b) \cup \boldsymbol{\Theta}_\Phi$

Učenje dubokog modela s kraja na kraj

Prednosti u odnosu na generičke funkcije i ručni dizajn:

- zadajemo *klasu* funkcija $\Phi(x|\Theta_\Phi)$ umjesto specifične funkcije $\Phi(x)$
- klasa funkcija je određena strukturom modela.
- možemo imati proizvoljno mnogo slojeva (ili skoro)

Nedostatak u odnosu na generičke funkcije i ručni dizajn:

- optimizacijski problem više nije konveksan
- nema garancije za globalnu konvergenciju učenja

Učenje dubokog modela s kraja na kraj

U praksi se međutim pokazuje da nekonveksni gubitak nije problem u visokodimenzionalnom prostoru

Duboki modeli najprikladniji za podatke koji su generirani kombinacijom faktora, npr. lice se sastoji od usta, očiju, nosa...

Ako takvi faktori postoje, njihova nezavisna obrada može osigurati efikasnu reprezentaciju regija ulaznog prostora.

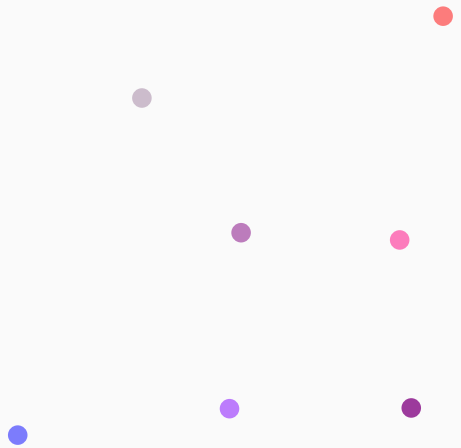
Duboki modeli mogu biti eksponencijalno učinkovitiji [delalleau11nips] od modela koji pretpostavljaju da se predikcija glatko mijenja u okolini podataka za učenje:

- plitki modeli, prototipovi (k-NN), jezgrene funkcije

Učenje dubokog modela s kraja na kraj

Primjer funkcije koja točke ravnine preslikava u RGB boju

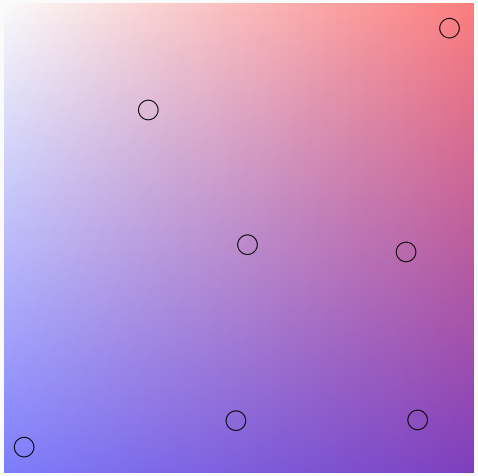
- zadan je sljedeći skup za učenje



Učenje dubokog modela s kraja na kraj

Primjer funkcije koja točke ravnine preslikava u RGB boju

- zadan je sljedeći skup za učenje
- intuitivno je jasno da generalizaciju nije lako postići

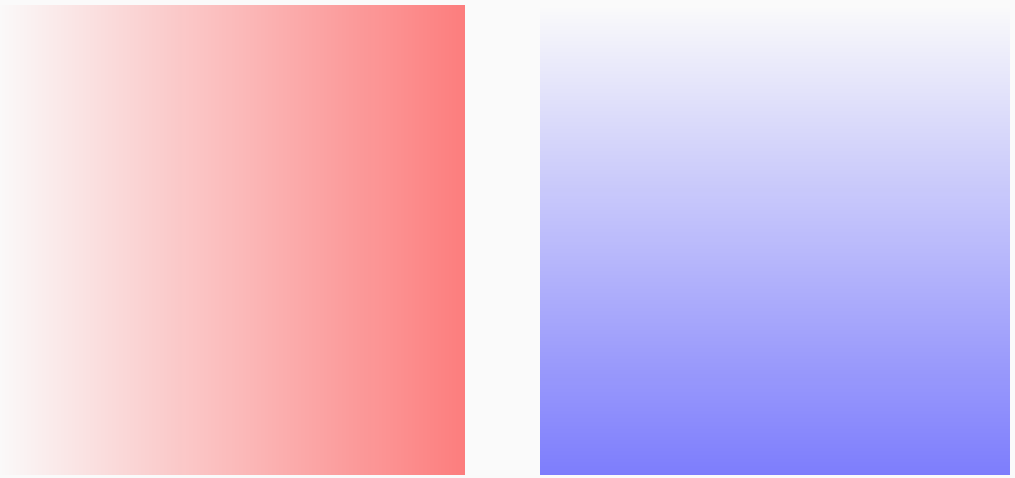


Učenje dubokog modela s kraja na kraj

Primjer funkcije koja točke ravnine preslikava u RGB boju

- zadan je sljedeći skup za učenje
- intuitivno je jasno da generalizaciju nije lako postići
- međutim, problem postaje lakši ako model izrazimo aditivnom kombinacijom dvaju nezavisnih 1D modela

$$f(x, y) = (f_R(x) + f_B(y))/2$$



Primjer: učenje funkcije XOR

- Promotrimo sljedeću funkciju dvije binarne varijable: $f^*(\boldsymbol{x}) = (x_0 \wedge \overline{x_1}) \vee (\overline{x_0} \wedge x_1)$.
- pokušajmo naučiti linearni model koji aproksimira f^*: $f(\boldsymbol{x}, \boldsymbol{\Theta} = (\boldsymbol{w}, b)) = \boldsymbol{w}^\top \boldsymbol{x} + b$
- želimo naći takve $\boldsymbol{\Theta}^* = (\boldsymbol{w}^*, b^*)$ da kvadratna greška predikcije bude minimalna:

$$J_{\mathrm{MSE}}(\boldsymbol{Y}, f(\boldsymbol{X}, \boldsymbol{\Theta})) = \frac{1}{4} \sum_{i=1}^{4} (f(\boldsymbol{x}_i, \boldsymbol{\Theta}) - y_i)^2$$

- Vidjet ćemo kasnije da takva funkcija gubitka nije osobito dobar izbor za klasifikacijske probleme, ali ovdje je pogodna jer rješenje možemo dobiti eksplicitno.

Primjer: učenje funkcije XOR

Označimo:

$$\boldsymbol{w}' = \begin{bmatrix} w_1, w_2, b \end{bmatrix}^\top, \quad \mathbf{X}' = \begin{bmatrix} x_{11} = 0, x_{12} = 0, 1 \\ x_{21} = 0, x_{22} = 1, 1 \\ x_{31} = 1, x_{32} = 0, 1 \\ x_{41} = 1, x_{42} = 1, 1 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 = 0 \\ y_2 = 1 \\ y_3 = 1 \\ y_4 = 0 \end{bmatrix}$$

Izrazimo gubitak u pogodnom obliku (za pravilo ulančavanja):

$$J_{\mathrm{MSE}}(\boldsymbol{y}, \mathbf{X}', \boldsymbol{w}') = \frac{1}{N} \left\| \mathbf{X}'\boldsymbol{w}' - \boldsymbol{y} \right\|_2^2 = \frac{\boldsymbol{q}^\top \boldsymbol{q}}{N}, \boldsymbol{q} = \mathbf{X}'\boldsymbol{w}' - \boldsymbol{y}$$

Sada možemo izračunati gradijent:

$$\begin{aligned} \nabla_{\boldsymbol{w}'} J_{\mathrm{MSE}}(\boldsymbol{y}, \mathbf{X}', \boldsymbol{w}')^\top &= \frac{\partial J}{\partial \boldsymbol{w}} = \frac{\partial J}{\partial \boldsymbol{q}} \frac{\partial \boldsymbol{q}}{\partial \boldsymbol{w}} \\ &= \frac{2 \cdot \boldsymbol{q}^\top}{N} \cdot \mathbf{X}' = \frac{2}{N} \left(\mathbf{X}'\boldsymbol{w}' - \boldsymbol{y} \right)^\top \mathbf{X}' \end{aligned}$$

Primjer: učenje funkcije XOR

Tražimo minimum funkcije J:

$$\nabla_{w'} J = \mathbf{0} \rightarrow \boldsymbol{w}' = \left(\mathbf{X}'^{\top}\mathbf{X}'\right)^{-1}\mathbf{X}'^{\top}\boldsymbol{y}$$

Rješenje: $\boldsymbol{w}^* = \mathbf{0}, b^* = 0.5$ (??!)

Zaključak: linearni model ne može riješiti XOR problem.

- Minski and Papert objavili su ovaj zaključak u knjizi: Perceptrons: An Introduction to Computational Geometry
- ovo se smatralo ograničenjem svih pristupa učenju i pridonijelo je prvoj zimi umjetne inteligencije (1974.-1980.)
- algoritam backprop izumio je Seppo Linnainmaa u okviru svog magisterija (1970)

Primjer: učenje funkcije XOR

- Uvedimo dodatni nelinearni sloj: mora biti nelinearan, inače bi cijeli model bio (opet) linearan.
- Uobičajeno se danas koristi zglobnica kao nelinearnost: $g(x) = \mathrm{ReLU}(x) = \max(0, x)$.
- Nelinearnost djeluje na svaki element vektora odvojeno: $\boldsymbol{g}(\boldsymbol{x})_i = g(x_i)$
- Sada možemo postaviti nelinearni model:

$$f(\boldsymbol{x}, \boldsymbol{\Theta}) = \boldsymbol{w}_2^\top \boldsymbol{h} + b_2,$$
$$\boldsymbol{h} = \boldsymbol{g}(\boldsymbol{W}_1^\top \boldsymbol{x} + \boldsymbol{b}_1)$$

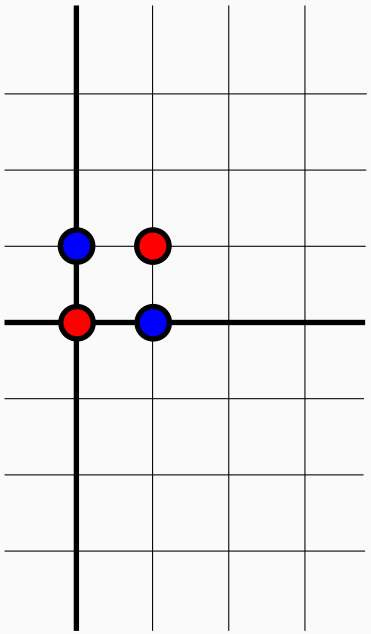
- $\boldsymbol{h}$: vektor (naučenih) značajki u skrivenom sloju
- $\boldsymbol{W}_1$, $\boldsymbol{b}_1$: naučeni parametri za računanje značajki iz podataka

Primjer: učenje funkcije XOR

$$\text{Rješenje}: \quad W_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad w_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b_2 = 0.$$

$$f(x_0, x_1) = 1 \cdot \max(x_0 + x_1, 0) - 2 \cdot \max(x_0 + x_1 - 1, 0) + 0$$

$$\mathrm{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

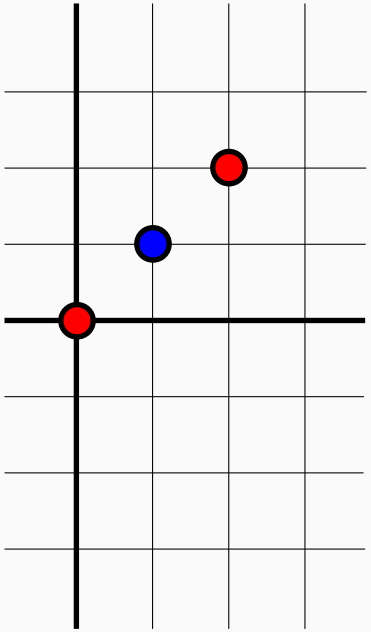


Primjer: učenje funkcije XOR

$$\text{Rješenje}: \quad W_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad w_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b_2 = 0.$$

$$f(x_0, x_1) = 1 \cdot \max(x_0 + x_1, 0) - 2 \cdot \max(x_0 + x_1 - 1, 0) + 0$$

$$W_1 X = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}$$

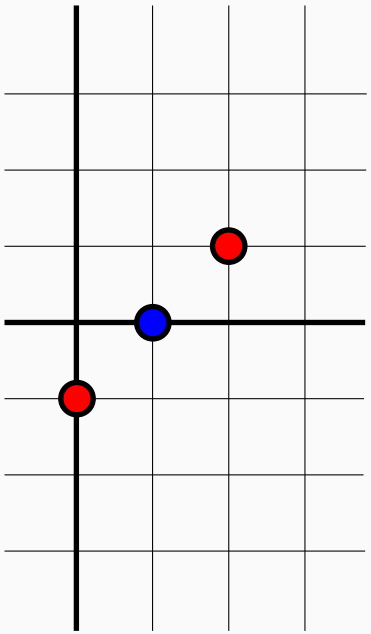


Primjer: učenje funkcije XOR

$$\text{Rješenje}: \quad W_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad w_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b_2 = 0.$$

$$f(x_0, x_1) = 1 \cdot \max(x_0 + x_1, 0) - 2 \cdot \max(x_0 + x_1 - 1, 0) + 0$$

$$W_1 X + b_1 = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

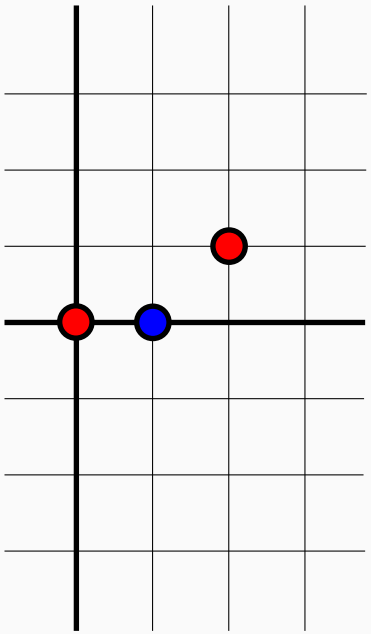


Primjer: učenje funkcije XOR

$$\text{Rješenje}: \quad W_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad w_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \quad b_2 = 0.$$

$$f(x_0, x_1) = 1 \cdot \max(x_0 + x_1, 0) - 2 \cdot \max(x_0 + x_1 - 1, 0) + 0$$

$$\mathrm{ReLU}(W_1 X + b_1) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$



Primjer: učenje funkcije XOR

Pokazali smo konačno rješenje, ali nismo pokazali kako se do njega dolazi.

U pravilu se za određivanje parametara dubokih modela koriste postupci optimizacije temeljeni na gradijentu funkcije gubitka.

Prikazano rješenje je globalni minimum funkcije gubitka,

- u generalnom slučaju gradijentni spust dovodi u lokalni minimum, ako se radi o ne-konveksnom gubitku
- u tom slučaju rješenje ovisi o inicijalizaciji.

Pregled

- Što su unaprijedni duboki modeli?
- Funkcija gubitka i izlazni slojevi.
- Aktivacijske funkcije u skrivenim slojevima.
- Univerzalna aproksimacija: dubina je važna.
- Backprop: izračun gradijenta kompozicije funkcije prosljeđivanjem greške unazad.

Učenje parametara modela – minimizacija empirijskog rizika

- Učenje modela: pronalaženje parametara $\boldsymbol{\Theta}^*$ za koje je empirijski rizik minimalan:

$$J(\mathbf{X}, \mathbf{Y}, \boldsymbol{\Theta}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\mathbf{y}_i, f(\mathbf{x}_i, \boldsymbol{\Theta})) + \lambda \Omega(\boldsymbol{\Theta})$$

$$\boldsymbol{\Theta}^* = \arg\min_{\boldsymbol{\Theta}} J(\mathbf{X}, \mathbf{Y}, \boldsymbol{\Theta})$$

- Funkcija gubitka $\ell(\mathbf{y}, \hat{\mathbf{y}})$ odražava naše "razočaranje" razlikom između predikcije modela $\hat{\mathbf{y}}$ i stvarne vrijednosti $\mathbf{y}$.
- Regularizator $\Omega(\boldsymbol{\Theta})$ kažnjava neke vrijednosti parametara modela.

0-1 gubitak

$$\ell_{01}(y, \hat{y}) = \begin{cases} 0, & \text{ako } y = \hat{y} \\ 1, & \text{inače} \end{cases}$$

- Ovaj gubitak nije derivabilan, pa je minimizacija $J(\mathbf{X}, \mathbf{Y}, \boldsymbol{\Theta})$ teška: kombinatorna optimizacija

Funkcija gubitka

Pokušajmo ne biti isključivi, oblikujmo model koji prediktira distribuciju preko mogućih izlaza $P(\hat{\boldsymbol{y}}|\boldsymbol{x};\boldsymbol{\Theta})$

Definirajmo gubitak kao negativnu log-izglednost:

$$\ell_{\mathrm{MLE}}(\boldsymbol{Y},\hat{\boldsymbol{Y}}) = -\sum \log P(\hat{\boldsymbol{y}} = \boldsymbol{y}|\boldsymbol{x};\boldsymbol{\Theta})$$

- za regresiju pretpostavljamo normalnu (Gaussovu) razdiobu, s jediničnom kovarijacijskom matricom:

$$P(\hat{\boldsymbol{y}} = \boldsymbol{y}|\boldsymbol{x};\boldsymbol{\Theta}) = \mathcal{N}(\boldsymbol{y}|\mu = f(\boldsymbol{x},\boldsymbol{\Theta}), \Sigma = I)$$

- za klasifikaciju pretpostavljamo kategoričku razdiobu (ili "generaliziranu Bernoullijevu razdiobu"):

$$P(\hat{y} = y|\boldsymbol{x};\boldsymbol{\Theta}) = f_y(\boldsymbol{x},\boldsymbol{\Theta})$$

Negativna log-izglednost kao funkcija gubitka (2)

Negativna log izglednost kategoričke razdiobe odgovara unakrsnoj entropiji između distribucije stvarnih oznaka $\boldsymbol{y}$ i distribucije izlaza našeg modela $P(\hat{\boldsymbol{y}}|\boldsymbol{x};\boldsymbol{\Theta})$.

Možemo formulirati i deterministično predviđanje uključivanjem Diracove δ-distribucije (ovo dovodi do gubitka 0 – 1)

Negativna log-izglednost kao funkcija gubitka

- Prednost prikaza funkcije gubitka preko negativne log-izglednosti je općenitost: nema potrebe dizajnirati funkcije gubitka specifične za svaki pojedinačni model.
  - definiramo distribuciju preko izlaza: $P(\hat{y}|x;\Theta)$
  - funkcija gubitka je: $\ell_{\mathrm{MLE}}(y,\hat{y}) = -\log P(\hat{y}=y|x;\Theta)$
- Za regresiju dobivamo srednju kvadratnu pogrešku:

$$\ell_{\mathrm{MSE}}(y,\hat{y}) = (y-\hat{y})^2$$

- Za klasifikaciju dobivamo:

$$\ell_{\mathrm{CE}}(y,\hat{y}) = -\log f_y(x,\Theta)$$

- Obje ove funkcije gubitka su derivabilne.

  $\Rightarrow$ mogu se učiti gradijentnim spustom

Klasifikacija: soft-max

Klasifikator mora vratiti kategoričku razdiobu preko C razreda:

- $f_i(\boldsymbol{x}, \boldsymbol{\Theta}) \in [0, 1]\ \forall i, \sum_i^C f_i(\boldsymbol{x}, \boldsymbol{\Theta}) = 1$

Promotrimo značajke u zadnjem sloju modela:

$$\boldsymbol{z} = \boldsymbol{h}^L = \boldsymbol{W}_L^\top \boldsymbol{h}^{L-1} + \boldsymbol{b}_L$$

Tada distribuciju možemo dobiti funkcijom $\mathrm{softmax}$:

$$f_i(\boldsymbol{x}, \boldsymbol{\Theta}) = P(\hat{y} = i | \boldsymbol{x}; \boldsymbol{\Theta}) = \mathrm{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- eksponenciranje garantira $f_i(\boldsymbol{x}, \boldsymbol{\Theta}) > 0\ \forall i$
- nazivnik garantira: $\sum_i^C f_i(\boldsymbol{x}, \boldsymbol{\Theta}) = 1$

Klasifikacija: softmax (objašnjenje)

Reinterpretirajmo logite $\boldsymbol{z}$ kao logaritmiranu nenormaliziranu združenu gustoću podataka i razreda:

$$z_y = \log \text{const} \cdot P(\hat{y} = y, \boldsymbol{x}; \boldsymbol{\Theta}) \ .$$

Odatle lako slijedi da softmaks odgovara aposteriornoj vjerojatnosti [grathwohl20iclr]:

$$\sum_k e^{z_k} = P(\boldsymbol{x}; \boldsymbol{\Theta}) \qquad \text{(marginalizacija preko } \hat{y}) \ ,$$

$$\begin{aligned} \text{softmax}(\mathbf{z})_y &= \frac{e^{z_j}}{\sum_k e^{z_k}} \\ &= \frac{\text{const} \cdot P(\hat{y} = y, \boldsymbol{x}; \boldsymbol{\Theta})}{\text{const} \cdot P(\boldsymbol{x}; \boldsymbol{\Theta})} \\ &= P(\hat{y} = y | \boldsymbol{x}; \boldsymbol{\Theta}) \qquad \text{(Bayesov poučak)} \ . \end{aligned}$$

Klasifikacija: stabilnost log-softmaksa

Izrazimo negativnu log-izglednost kao funkciju logita $\mathsf{z} = f_\theta(\mathsf{x})$ i indeksa točnog razreda y:

$$\ell_{\mathrm{NLL}}(\mathsf{z}, y) = -\log \mathrm{softmax}_y(\mathsf{z})$$

- problem: ako softmaks podlije (ode u nulu), gubitak $\to \infty$
- kada se to može dogoditi?

Stabilnija formulacija log-softmaksa:

$$\begin{aligned}\ell_{\mathrm{NLL}}(\mathsf{z}, y) &= -\log \mathrm{softmax}_y(\mathsf{z}) \\ &= -\log \frac{e^{z_y}}{\sum_k e^{z_k}} = \log \sum_k e^{z_k} - z_y\end{aligned}$$

- ova izvedba osjetljiva je na preljev softmaksa
- kako to možemo izbjeći?

Klasifikacija: soft-max kao izlazni sloj

Stabilna negativna log izglednost (y: indeks točnog razreda):

$$\ell_{\mathrm{NLL}}(\mathbf{z}, y) = -\log \mathrm{softmax}_y(\mathbf{z}) = \log \sum_k e^{z_k} - z_y$$

Možemo izvesti sljedeće intuitivne zaključke:

- Kada model točno predviđa ($\max_j z_j = z_y$) gubitak je ≈ 0.
- Kada model griješi, tj. kada je $\max_j z_j \neq z_y$, na gubitak najviše utječe najaktivnija (najjača) netočna predikcija.
- Takvo ponašanje je vrlo slično 0-1 gubitku: negativna log-izglednost je gornja ograda 0-1 gubitka.

Za domaći rad dokazati:

$$\frac{d\ell_{\mathrm{CE}}(\hat{\mathbf{y}}, \mathrm{softmax}(\mathbf{z}))}{dz_k} = \mathrm{softmax}(\mathbf{z})_k - [\![\hat{y} = k]\!]$$

Svojstva softmaksa

Invarijantnost na dodavanje konstante:

$$\mathrm{softmax}(z) = \mathrm{softmax}(z + c) = \mathrm{softmax}(z - \max_j z_j)$$

Bolje ime (koje se nažalost nije uvriježilo): softargmax

"Pravi softmax" izračunali bismo kao log-sum-exp:

$$\mathrm{LSE}(z) = \log \sum_i e^{z_i} = \max(z) + \log \sum_i e^{z_i - \max(z)}$$

Parametrizacije softmaxa

Iako je izlaz softmaxa C-dimenzionalan, postoji samo $C - 1$ stupnjeva slobode (izlaz je distribucija)

To znači da jedan od ulaza možemo fiksirati (npr. postaviti na 0) bez smanjenja općenitosti modela.

U praksi nema razlike između te dvije varijante, a implementacija je jednostavnija kada je ulaz C dimenzionalan.

Binarna klasifikacija: sigmoida kao izlazni sloj

Ako je $C = 2$ onda:

$$P(\hat{y} = 1|\boldsymbol{x}) = \mathrm{softmax}(z)_1 = \frac{\exp(z_1)}{\exp(z_0) + \exp(z_1)} = \frac{1}{1 + \exp(z_0 - z_1)}$$

Ako postavimo $z_0 = 0$ dobivamo:

$$P(\hat{y} = 1|\boldsymbol{x}) = \sigma(z_1)$$

$\Rightarrow$ soft-max je poopćenje sigmoide na slučaj C>2 klasa

$\Rightarrow$ kategorička razdioba je poopćenje Bernoullijeve razdiobe na slučaj C>2 ishoda.

Za domaći rad dokazati: $\frac{d\ell_{\mathrm{CE}}(y,\sigma(z))}{dz} = \sigma(z) - y$

Srednja kvadratna pogreška za klasifikacijski gubitak?

Zašto negativna (log-)izglednost kao funkcija gubitka za klasifikaciju? Zašto ne srednja kvadratna pogreška?

$$\ell_{\mathrm{MSE}}(y, \sigma(z)) = (y - \sigma(z))^2$$

Pogledajmo gradijent funkcije gubitka s obzirom na značajke u zadnjem sloju:

$$\frac{\partial \ell_{\mathrm{MSE}}(y, \sigma(z))}{\partial z} = 2(\sigma(z) - y)(1 - \sigma(z))\sigma(z)$$

Kada je sigmoida u zasićenju ($z >> 0$ ili $z << 0$) gradijent funkcije gubitka je malen, bez obzira da li je $\sigma(z)$ blizu y ili ne: model ne može naučiti iz takvog primjera.

Srednja kvadratna pogreška za klasifikacijski gubitak?

Glavni nedostatak MSE-a jest ignoriranje strukture vjerojatnosne distribucije; to se najbolje vidi u višerazrednom slučaju

Npr. pretpostavimo da imamo podatke $\mathbf{x}_1$ i $\mathbf{x}_2$ koji bi se trebali klasificirati u drugi razred:

$\mathbf{Y}_1^{OH} = \mathbf{Y}_2^{OH} = [0, 0, 1]$

Pretpostavimo dalje da za njih dobivamo sljedeće distribucije:

$P(\mathbf{Y}|\mathbf{x}_1) = [0.8, 0, 0.2]$, $P(\mathbf{Y}|\mathbf{x}_2) = [0.4, 0.4, 0.2]$

Gubitci su različiti iako su predikcije jednako pogrešne:

$L_{MSE}(\mathbf{x}_1, \mathbf{Y}_1^{OH}) = 1.28$, $L_{MSE}(\mathbf{x}_2, \mathbf{Y}_2^{OH}) = 0.96$

Za klasifikaciju, MLE je bolji gubitak od MSE.

Pregled

- Što su unaprijedni duboki modeli?
- Funkcija gubitka i izlazni slojevi.
- Aktivacijske funkcije u skrivenim slojevima.
- Univerzalna aproksimacija: dubina je važna.
- Backprop: izračun gradijenta kompozicije funkcije prosljeđivanjem greške unazad.

Primjer: učenje funkcije XOR

Uveli smo bili dodatni nelinearni sloj: mora biti nelinearan, inače bi cijeli model bio (opet) linearan.

Uobičajeno se danas koristi zblobnica kao nelinearnost: $g(x) = \mathrm{ReLU}(x) = \max(0, x)$.

Nelinearnost djeluje na svaki element vektora odvojeno: $g(\boldsymbol{x})_i = g(\boldsymbol{x}_i)$

Nelinearnost se zove aktivacijska funkcija.

Zglobnica kao aktivacijska funkcija

Zglobnica (*eng. rectified linear unit*):
$g(x) = \mathrm{ReLU}(x) = \max(0, x)$.

Prednosti:

- u aktivnom stanju propušta signal unaprijed i gradijent unatrag
- podržava propagiranje gradijente s obzirom na ulazne i izlazne aktivacije

”Nedostatak” 1: gradijent funkcije nije definiran za $x = 0$

- u implementaciji definiramo gradijent da bude jednak ili gradijentu s lijeva (0) ili gradijentu s desna (1).

Zglobnica kao aktivacijska funkcija

Nedostatak 2: u neaktivnom stanju ne propušta signal unaprijed i gradijent unatrag, ali:

- postoje bijektivne generalizacije zglobnice:
  - Leaky ReLU: $g(x, \alpha) = \mathbf{max}(0, x) + \alpha\, \mathbf{min}(0, x)$.
  - Soft Plus: $g(x) = \mathbf{log}(1 + e^x)$.
- normalizacija po grupi (*eng. batch normalization*) implicira srednju vrijednost 0 i varijancu 1

  $\Rightarrow$ u svakoj iteraciji učenja i u svakoj značajki imamo 50% aktivnih aktivacija

Nedostatak 3: srednja vrijednost aktivacija nije 0

- situaciju ponovo spašava normalizacija po grupi

Druge korištene aktivacijske funkcije

Sigmoida $\sigma(x) = (1 + \exp(-x))^{-1}$

- guši gradijent kada uđe u zasićenje (učenje prestaje, nestajući gradijent, eng. vanishing gradient),
- nema ih više u unaprijednim modelima (ima iznimki)

Tangens hiperbolni $\tanh(x) = \frac{\exp(2x)-1}{\exp(2x)+1}$ obično se ponaša bolje od sigmoide budući da sliči identitetu u dijelu oko $x = 0$, to osigurava jednostavan transfer gradijenata unatrag

- veza između $\tanh$ i σ: $\tanh(x) = 2\sigma(2x) - 1$.

Eksp.-linearna funkcija (ELU): $f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1); \text{for } x < 0 \\ x; \text{for } x \geq 0 \end{cases}$

Generalno: bilo koja nelinearna funkcija je u redu

Pregled

- Što su unaprijedni duboki modeli?
- Funkcija gubitka i izlazni slojevi.
- Aktivacijske funkcije u skrivenim slojevima.
- Univerzalna aproksimacija: dubina je važna.
- Backprop: izračun gradijenta kompozicije funkcije prosljeđivanjem greške unazad.

Teorem o univerzalnoj aproksimaciji

Teorem: model s linearnim izlaznim slojem i najmanje jednim skrivenim slojem s nepolinomnom aktivacijskom funkcijom može aproksimirati bilo koju Borel mjerljivu funkciju iz jednog konačno-dimenzionalnog prostora u drugi s proizvoljno malenom greškom (većom od 0), ako model ima dovoljno aktivacija u skrivenom sloju (ili skrivenim slojevima).

Svaka neprekinuta funkcija definirana na zatvorenom i ograničenom podskupu $\mathbb{R}^n$ je Borel mjerljiva funkcija

Ne trebamo prilagođavati aktivacijske funkcije: dovoljno je da imamo jedan skriveni sloj

Teorem o univerzalnoj aproksimaciji (caveat 1)

Teorem o reprezentacijskom kapacitetu dubokih modela: kad bi funkcija f^* bila poznata onda bi je mogli aproksimirati proizvoljno dobro.

Međutim, funkcija nije poznata, dostupni su samo podaci za učenje $(\mathcal{X}, \mathcal{Y})$.

Ovaj teorem ne govori ništa o tome da li f^* možemo naučiti iz $(\mathcal{X}, \mathcal{Y})$.

Teorem samo kaže da dovoljno ”nabildani” model može naštrebati skup za učenje

Teorem o univerzalnoj aproksimaciji (caveat 2)

Teorem ne specificira koliko aktivacija trebamo za postići zadanu grešku aproksimacije, ali postoji gornja ograda (*eng. upper bound*)

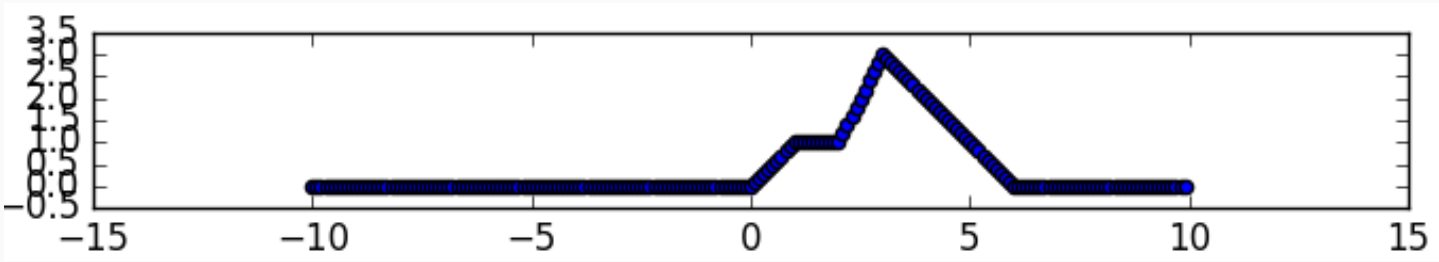
- u najgorem slučaju trebamo eksponencijalno veliki broj značajki u srednjem sloju:

$$\dim(\mathbf{h}) \sim O(a^{\dim(\mathbf{x})})$$

- svaka od tih značajki odgovara konfiguraciji ulazâ koja se mapira u izlaz koji moramo razlikovati od drugih izlaza.
  - za naučiti logičku funkciju n varijabli trebamo izračunati $O(2^n)$ minterma

Teorem o univerzalnoj aproksimaciji

Zadatak: konstruirati dvorazinski afini model aktiviran zglobnicom za aproksimaciju sljedeće funkcije $\mathbb{R} \to \mathbb{R}$:



Rješenje:

```
h10 = np.maximum(X-0,0)
h11 = np.maximum(X-1,0)
h12 = np.maximum(X-2,0)
h13 = np.maximum(X-3,0)
h14 = np.maximum(X-6,0)
h21 = 1*h10 - 1*h11 + 2*h12 - 3*h13 + 1*h14
```

Literatura: `http://neuralnetworksanddeeplearning.com/chap4.html`

Zašto onda uopće duboki modeli?
Efikasnost prikaza funkcije.

Duboki modeli mogu smanjiti potreban broj aktivacija u skrivenim slojevima za prikaz funkcije

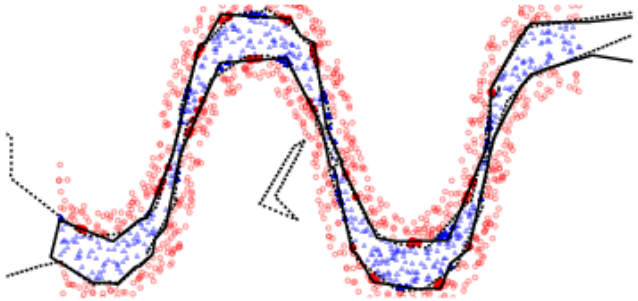
- postoje funkcije koje se mogu efikasno predstaviti dubokim modelima.

Efikasnost dubokih modela u odnosu na plitke (samo jedan skriveni sloj) može biti čak eksponencijalna u broju potrebnih aktivacija

Zašto onda uopće duboki modeli?
Efikasnost prikaza funkcije.

Modeli koje koriste zglobnicu kao aktivacijsku funkciju definiraju po dijelovima linearne funkcije nad regijama ulaznog prostora

- broj tih regija je mjera fleksibilnosti (kapaciteta) modela.
- duboki modeli imaju eksponencijalno više regija od plitkih modela s istim brojem aktivacija.



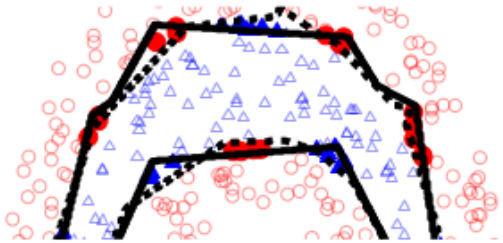
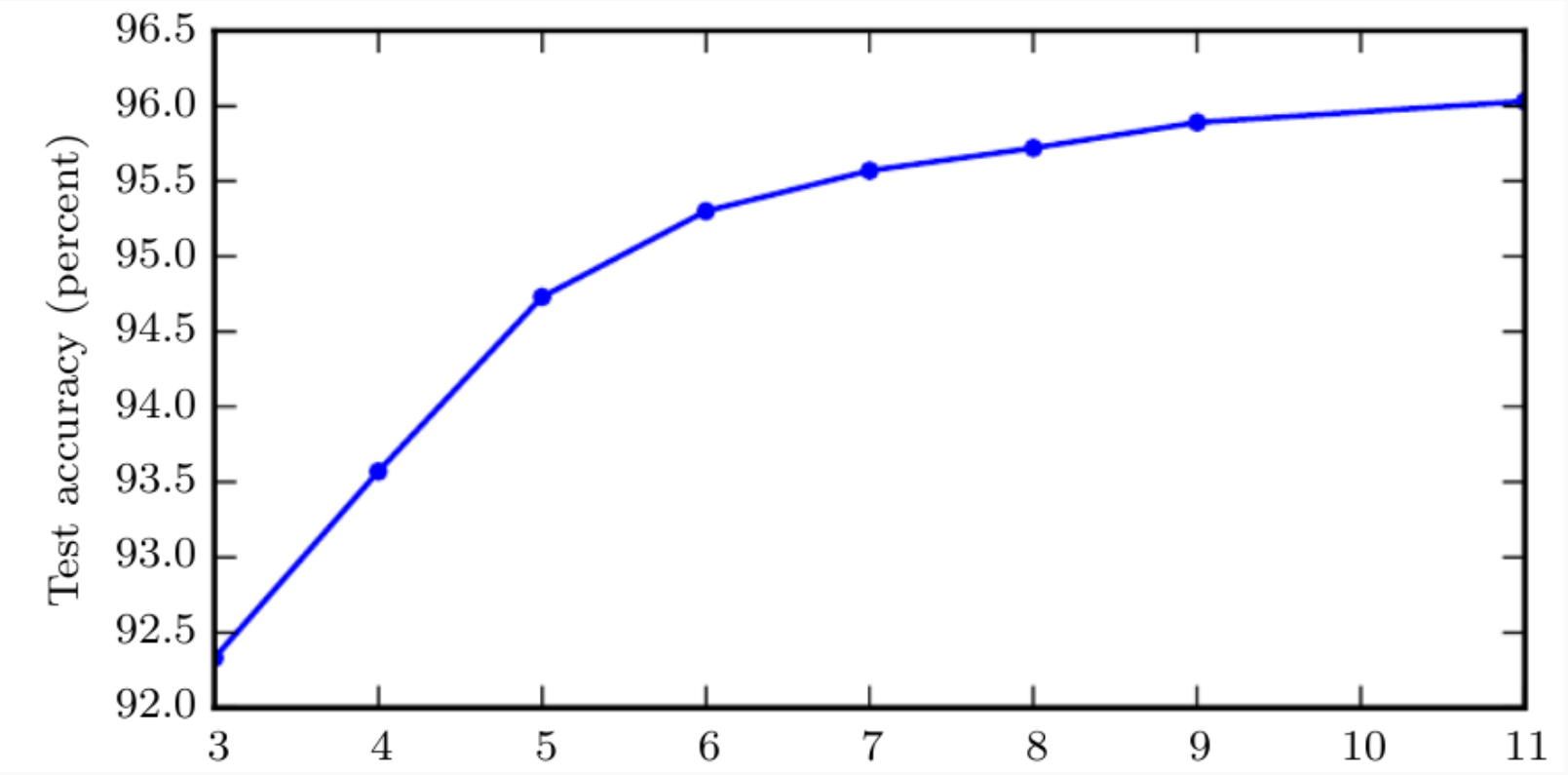


Figure 1: Binary classification using a shallow model with 20 hidden units (solid line) and a deep model with two layers of 10 units each (dashed line). The right panel shows a close-up of the left panel. Filled markers indicate errors made by the shallow model.

[montufar14nips]

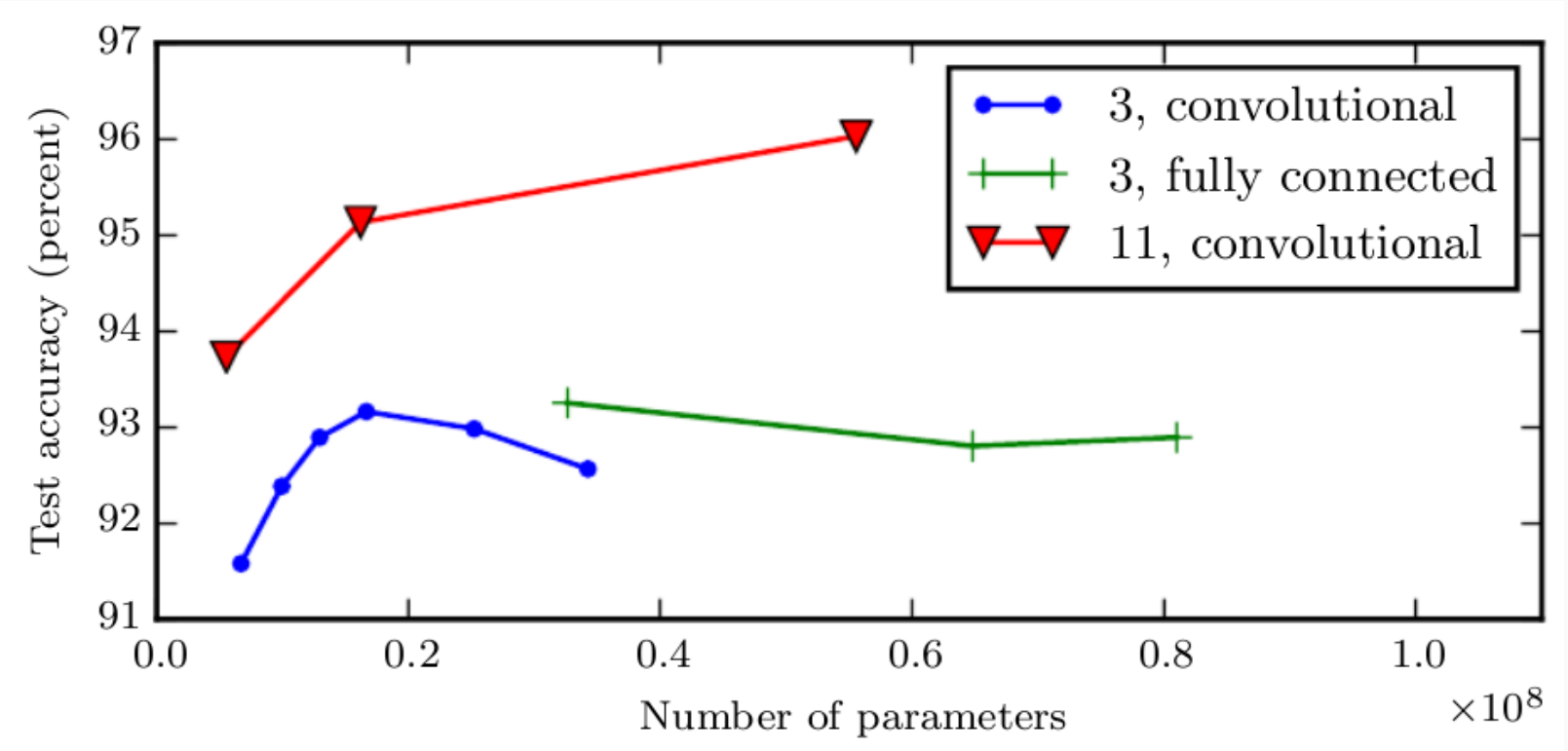


SVHN sequence transcription [goodfellow17]

Empirijski rezultati pokazuju da modeli za klasifikaciju slika postižu bolje rezultate kad povećamo dubinu

- x: dubina modela, y: točnost klasifikacije

povećanjem širine modela dobivamo značajno manja poboljšanja...



Duboki konvolucijski modeli za klasifikaciju slika generaliziraju bolje od plitkih (SVHN sequence transcription [goodfellow17])

- x: broj parametara, y: točnost klasifikacije

Produbljenje uvodi pristranost koja pogoduje generalizaciji!

- plitki modeli se prenauče već na 2e7 parametara
- duboki modeli dobro generaliziraju i sa 6e7 parametara

Pregled

- Što su unaprijedni duboki modeli?
- Funkcija gubitka i izlazni slojevi.
- Aktivacijske funkcije u skrivenim slojevima.
- Univerzalna aproksimacija: dubina je važna.
- Backprop: izračun gradijenta kompozicije funkcije prosljeđivanjem greške unazad.

Nadzirano učenje modela

Prolaz unaprijed: računanje izlaza modela $\hat{y} = f(x, \Theta)$ i funkcije gubitka $J(y, \hat{y}) = J(y, f(x, \Theta))$

Prolaz unatrag: računanje gradijenta funkcije gubitka s obzirom na parametre modela $\nabla_{\Theta} J(y, f(x, \Theta)) = (\frac{\partial J(y, f(x, \Theta))}{\partial \Theta})^{\top}$

Optimizacijski postupak: tipično varijanta gradijentnog spusta

- $\Theta' = \Theta - \delta \cdot \nabla_{\Theta} J(y, f(x, \Theta))$
- o tome ćemo pričati detaljnije neki drugi put.

Proslijeđivanje greške unatrag (*eng. backprop*) je jednostavan i računski nezahtjevan način računanja gradijenta kompozicije funkcija.

Derivacija kompozicije funkcija ulančavanjem gradijenata

Pravilo ulančavanja (*eng. chain rule*): recept za deriviranje kompozicije funkcija za koje su derivacije poznate

Za skalarne funkcije, npr. $y = g(x), z = f(y) = f(g(x))$, imamo:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = \frac{df(y)}{dy}\frac{dg(x)}{dx}$$

Za vektorske funkcije, $\boldsymbol{y} = g(\boldsymbol{x}), z = f(\boldsymbol{y}) = f(g(\boldsymbol{x}))$, dobivamo:

$$\frac{\partial z}{\partial \boldsymbol{x}} = \frac{\partial z}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \qquad \text{ili} \qquad \nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z.$$

- $\frac{\partial z}{\partial \boldsymbol{y}}$ i $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}$ su Jakobijani dimenzija $1 \times n$ odnosno $n \times m$;
- $\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$

Takav umnožak provodimo za svaki računski korak modela.

Backprop: rekurzivna primjena ulančavanja

Promotrimo duboki model f parametriziran s $\boldsymbol{\Theta}$ koji podatke $\boldsymbol{x}$ preslikava u predikcije $\hat{y}$:

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

Gradijent gubitka s obzirom na parametre l-tog sloja tada je:

$$\begin{aligned}\frac{\partial \mathcal{L}(y,\hat{y})}{\partial \boldsymbol{\Theta}^l} &= \frac{\partial \mathcal{L}(y,\hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \boldsymbol{h}^L} \frac{\partial \boldsymbol{h}^L}{\partial \boldsymbol{h}^{L-1}} \cdots \frac{\partial \boldsymbol{h}^l}{\partial \boldsymbol{\Theta}^l} \\ &= \frac{\partial \mathcal{L}(y,\hat{y})}{\partial \boldsymbol{h}^L} \frac{\partial f^L(\boldsymbol{h}^{L-1}, \boldsymbol{\Theta}^L)}{\partial \boldsymbol{h}^{L-1}} \cdots \frac{\partial f^{l+1}(\boldsymbol{h}^l, \boldsymbol{\Theta}^{l+1})}{\partial \boldsymbol{h}^l} \frac{\partial f^l(\boldsymbol{h}^{l-1}, \boldsymbol{\Theta}^l)}{\partial \boldsymbol{\Theta}^l}\end{aligned}$$

Za svaki sloj trebamo izračunati parcijalnu derivaciju izlaza:

- s obzirom na parametre (ako postoje) $\frac{\partial f^l(\boldsymbol{h}^{l-1}, \boldsymbol{\Theta}^l)}{\partial \boldsymbol{\Theta}^l}$
- s obzirom na ulaz $\frac{\partial f^l(\boldsymbol{h}^{l-1}, \boldsymbol{\Theta}^l)}{\partial \boldsymbol{h}^{l-1}}$ (samo ako nismo gotovi)
- problem: $\boldsymbol{\Theta}^l$ može biti matrica (potpuno povezani sloj)
- problem: $\boldsymbol{h}^l$ i $\boldsymbol{\Theta}^l$ mogu biti tenzori 4. reda (konv. sloj)

Gradijenti po tenzorima višeg reda (>1)

Gradijente po tenzorima *možemo* računati kao i za vektore

- prvo odredimo gradijente za vektorizirani tenzor,
- te ih na kraju presložimo u početni oblik

Pretpostavimo: $\mathbf{X} \in \mathbb{R}^{m_1} \times \mathbb{R}^{m_2} \times \cdots \mathbb{R}^{m_M}$, $\mathbf{Y} \in \mathbb{R}^{n_1} \times \mathbb{R}^{n_2} \times \cdots \mathbb{R}^{n_N}$

- Tada je $\frac{\partial \text{vec}(\mathbf{Y})}{\partial \text{vec}(\mathbf{X})}$ Jakobijan dimenzija $(n_1 n_2 \cdots n_N) \times (m_1 m_2 \cdots m_M)$.
- backprop je i u ovom slučaju samo množenje Jakobijana:

$$\frac{\partial z}{\partial \text{vec}(\mathbf{X})} = \frac{\partial z}{\partial \text{vec}(\mathbf{Y})} \frac{\partial \text{vec}(\mathbf{Y})}{\partial \text{vec}(\mathbf{X})}$$

Backprop za parametre potpuno povezanog sloja

U praksi, gradijente gubitka ipak nećemo računati po vektoriziranim težinama jer postoje efikasniji pristupi.

Razmotrimo potpuno povezani sloj ($\mathbf{h}_{k-1} \in \mathbb{R}^{D_{k-1}}$, $\mathbf{s}_k \in \mathbb{R}^{D_k}$):

$$\mathbf{s}_k = \mathbf{W}_k \cdot \mathbf{h}_{k-1} + \mathbf{b}_k \ .$$

Gradijente po vektoriziranim parametrima možemo izračunati prema osnovnom receptu u $\mathrm{O}(D_k^2 \cdot D_{k-1})$:

$$\frac{\partial \mathcal{L}}{\partial \mathrm{vec}(\mathbf{W}_k)} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_k} \cdot \frac{\partial \mathbf{s}_k}{\partial \mathrm{vec}(\mathbf{W}_k)} \ .$$

Backprop za parametre potpuno povezanog sloja

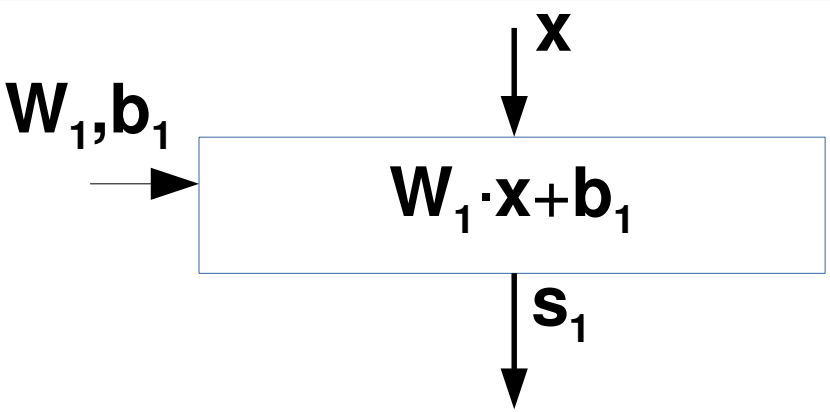
Umjesto toga, taj gradijent radije ćemo računati prema receptu iz uputa za laboratorijske vježbe u $O(D_k \cdot D_{k-1})$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_k} := \left(\frac{\partial \mathcal{L}}{\partial w_{kij}} \right)_{D_k \times D_{k-1}} = \left[\frac{\partial \mathcal{L}}{\partial \mathbf{s}_k} \right]^\top \cdot \mathbf{h}_{k-1}^\top$$

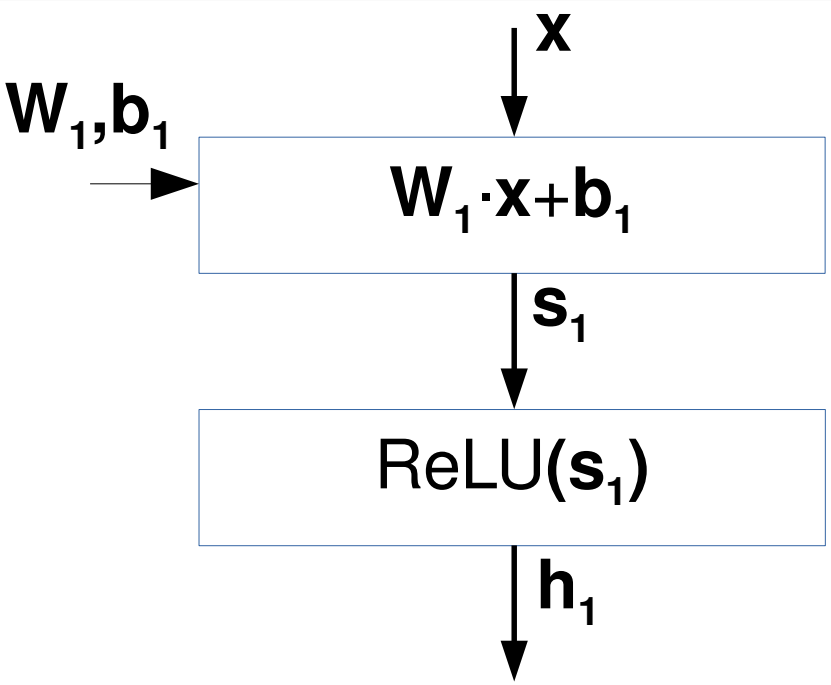
Domaći rad:

- pokazati da prikazani pristupi ekvivalentni (isti rezultat)
- razumjeti kako određujemo njihovu složenost

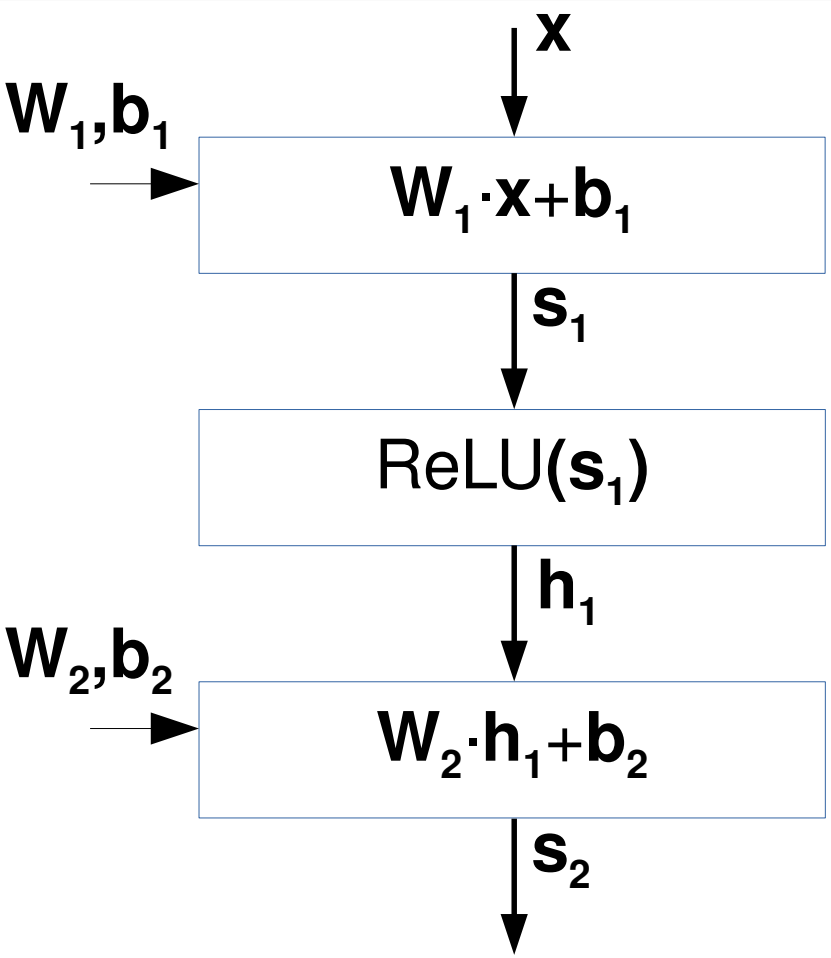
Primjer: unaprijedni prolaz



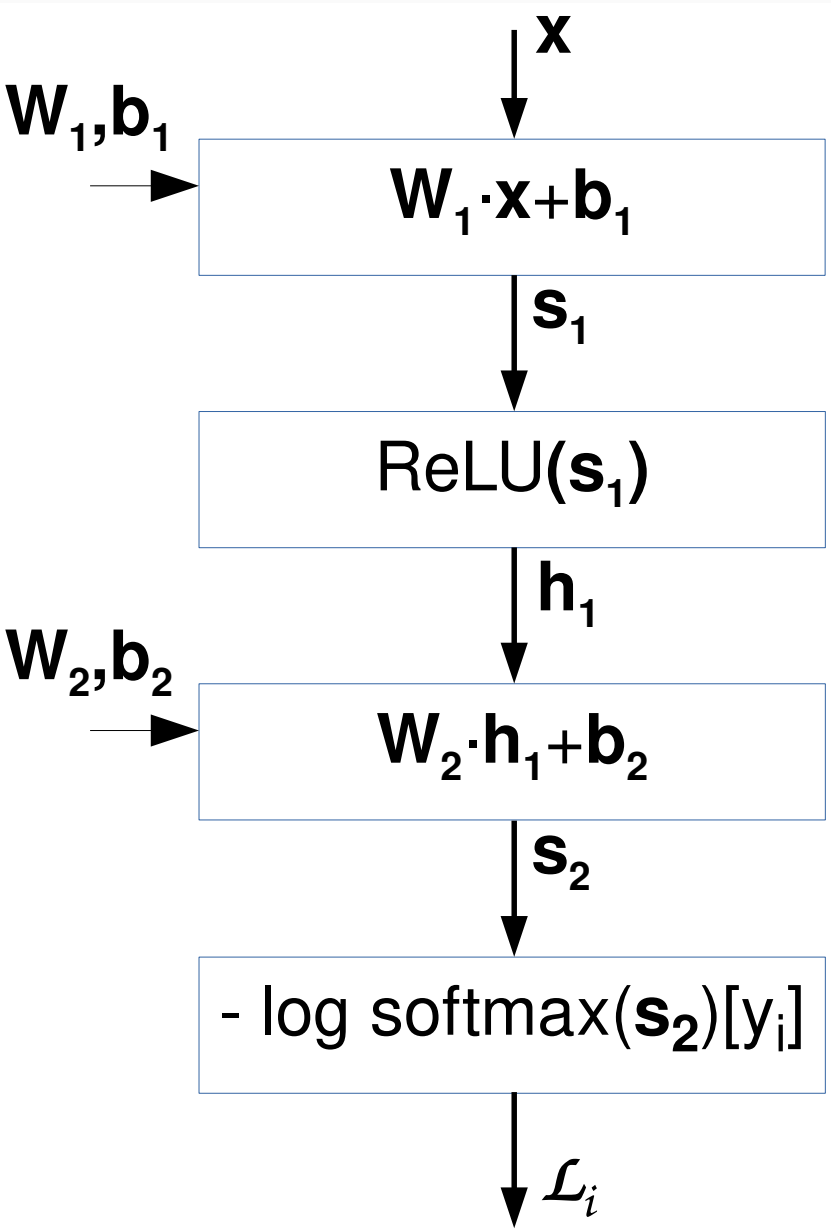
Primjer: unaprijedni prolaz



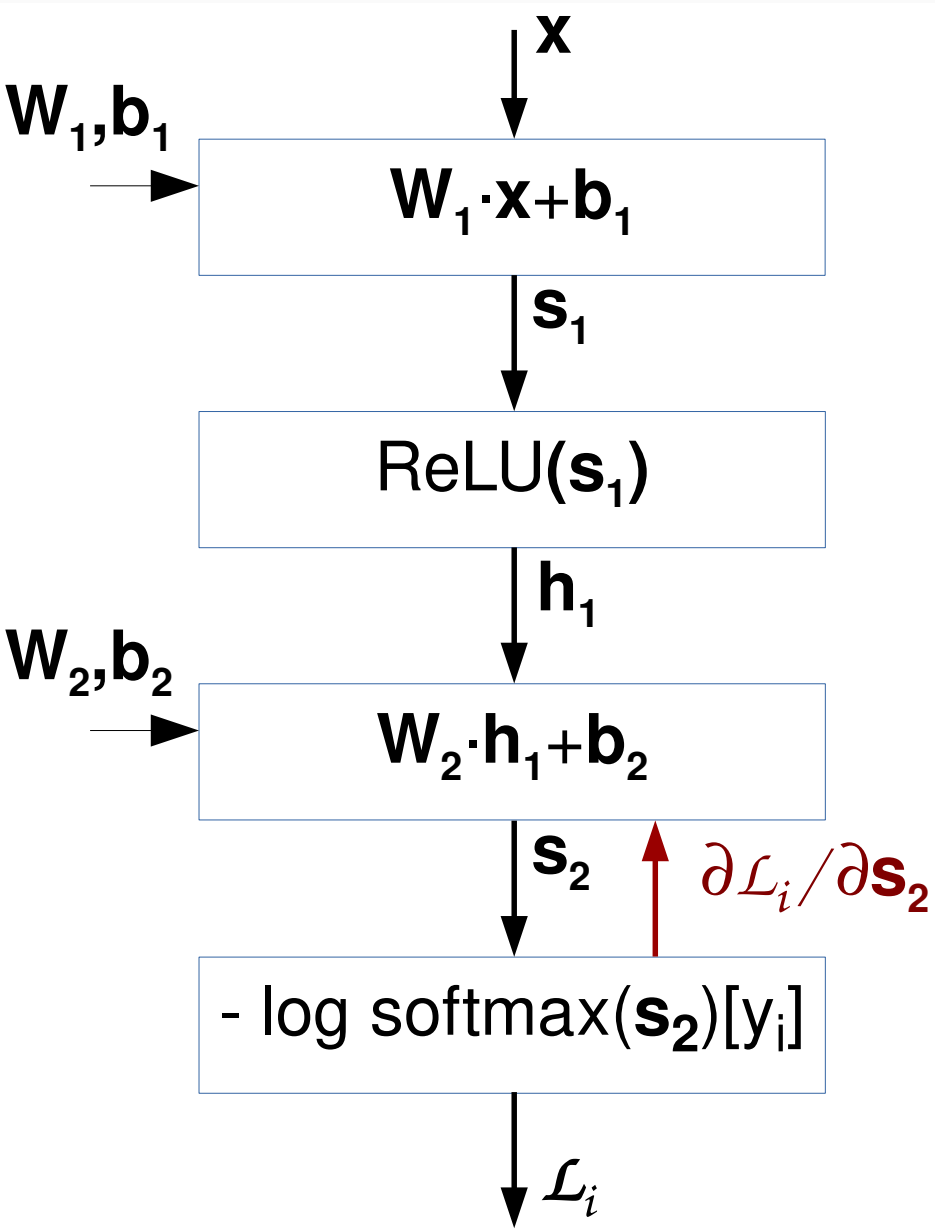
Primjer: unaprijedni prolaz



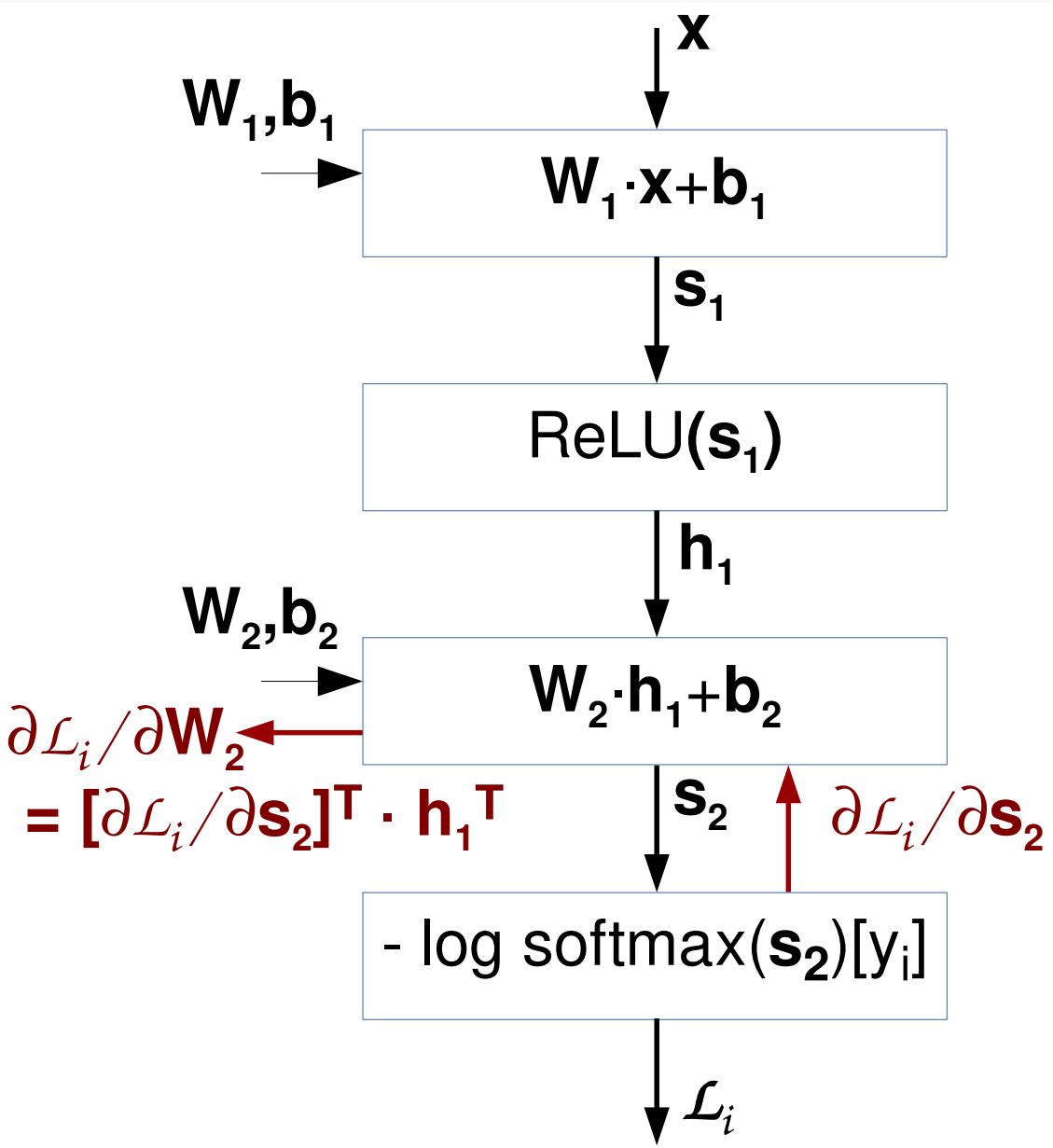
Primjer: unaprijedni prolaz



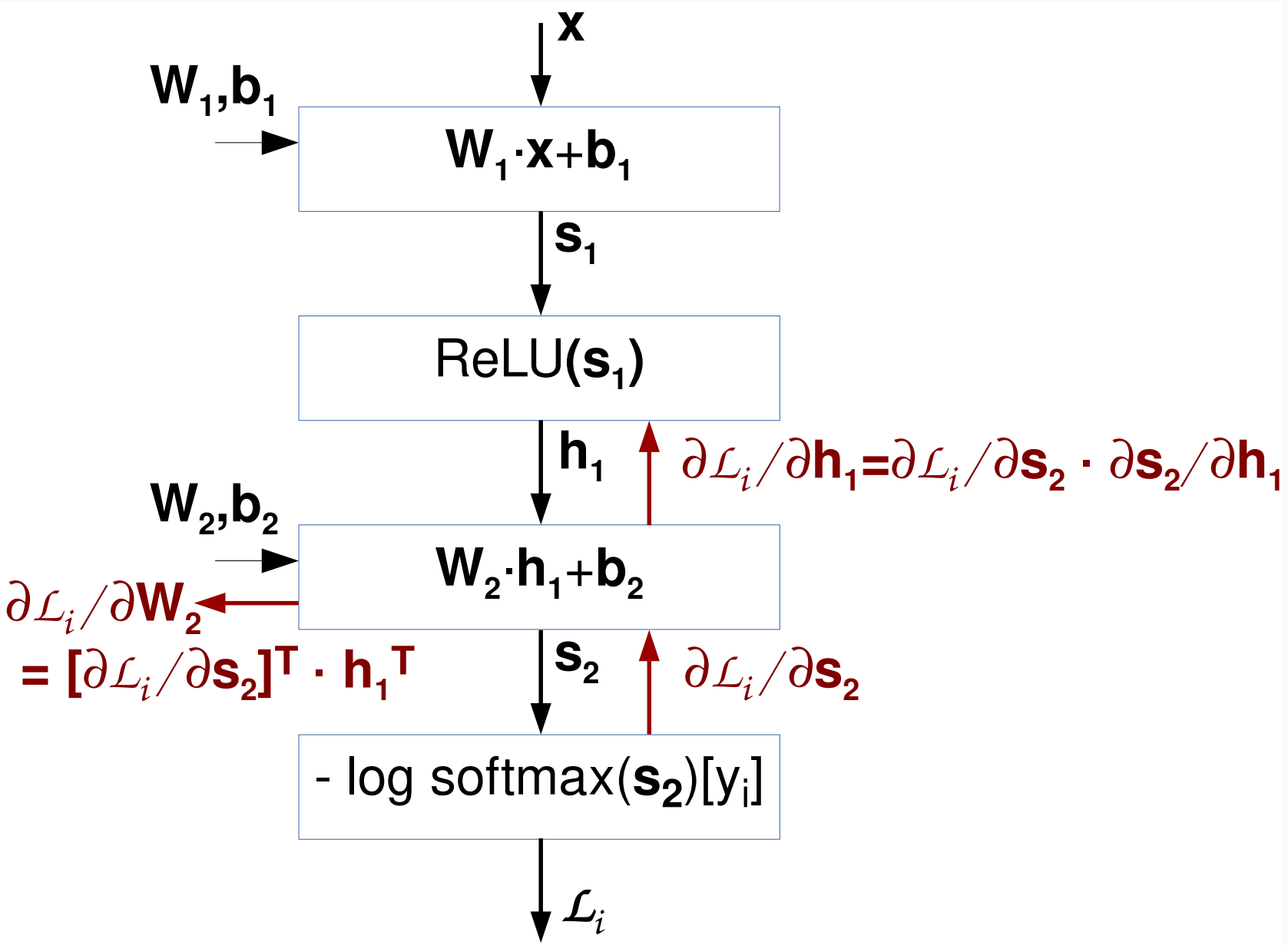
Primjer: unatražni prolaz (=dinamičko programiranje)



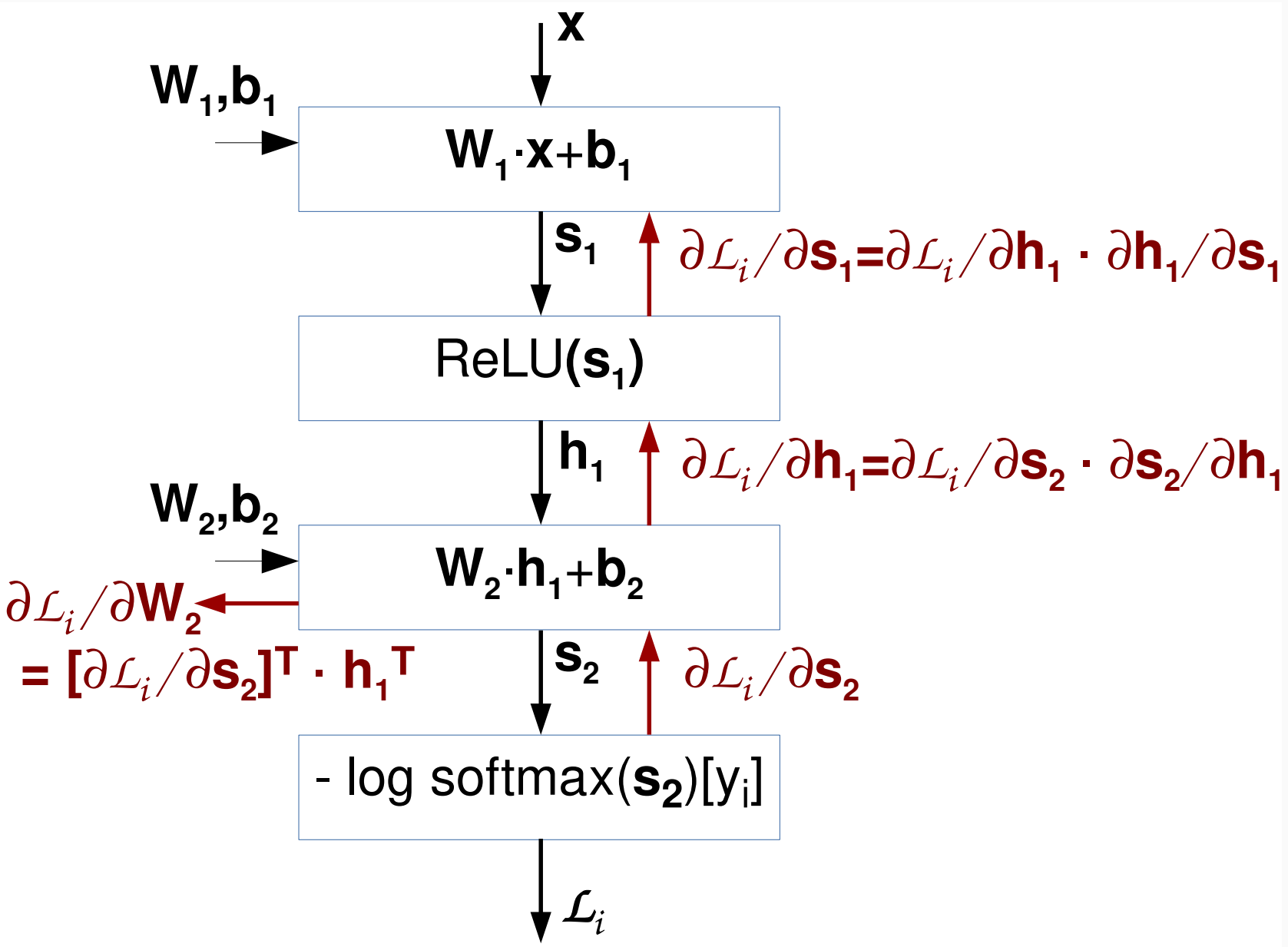
Primjer: unatražni prolaz (=dinamičko programiranje)



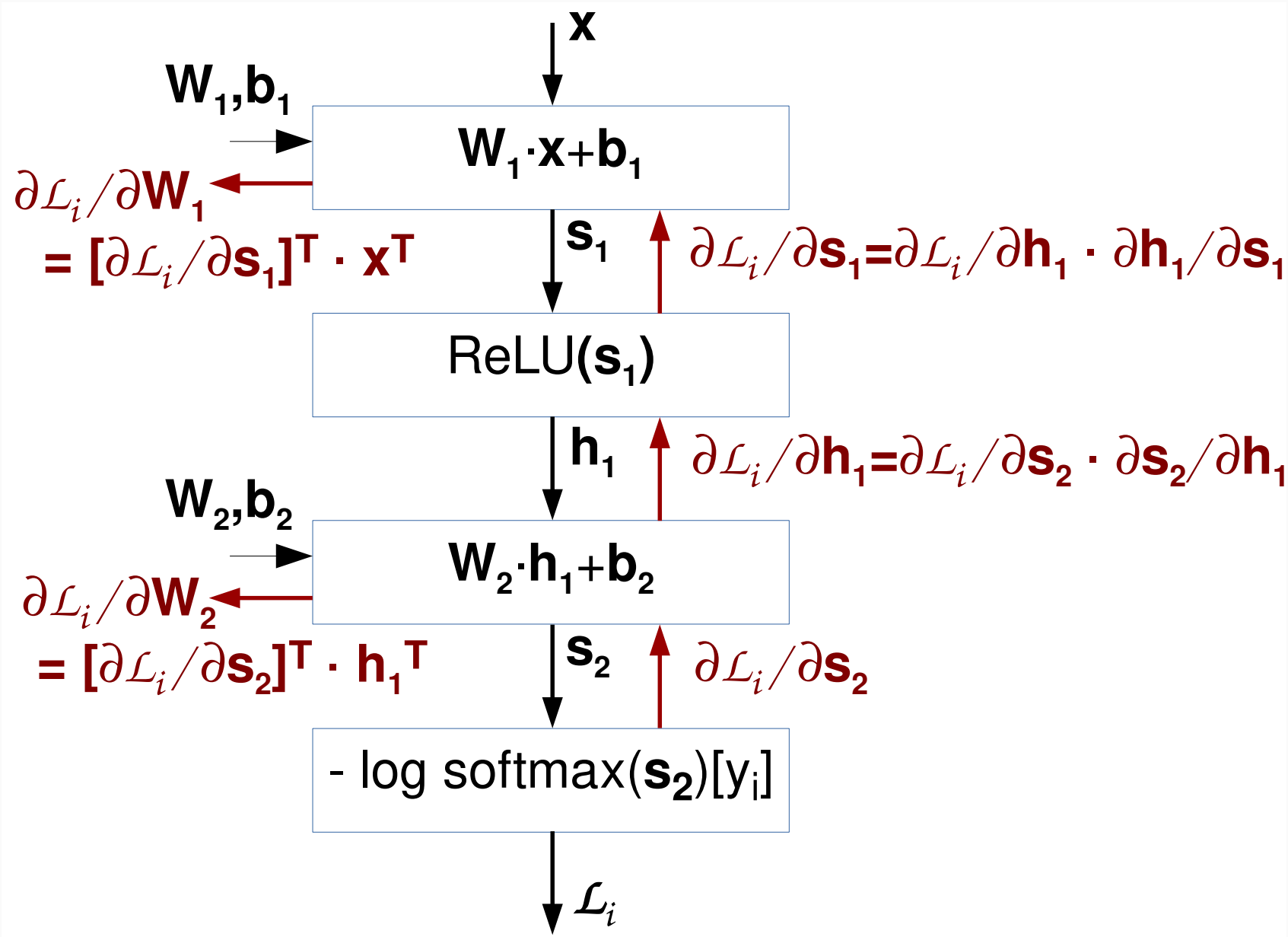
Primjer: unatražni prolaz (=dinamičko programiranje)



Primjer: unatražni prolaz (=dinamičko programiranje)



Primjer: unatražni prolaz (=dinamičko programiranje)



Automatsko računanje gradijenata

Kako bismo proveli automatsko računanje gradijenata, duboki model predstavljamo računskim grafom

Korijeni grafa predstavljaju ulazne podatke, oznake, parametre i hiperparametre

Svi ostali čvorovi predstavljaju operacije: diferencijabilne funkcije jedne ili više varijabli.

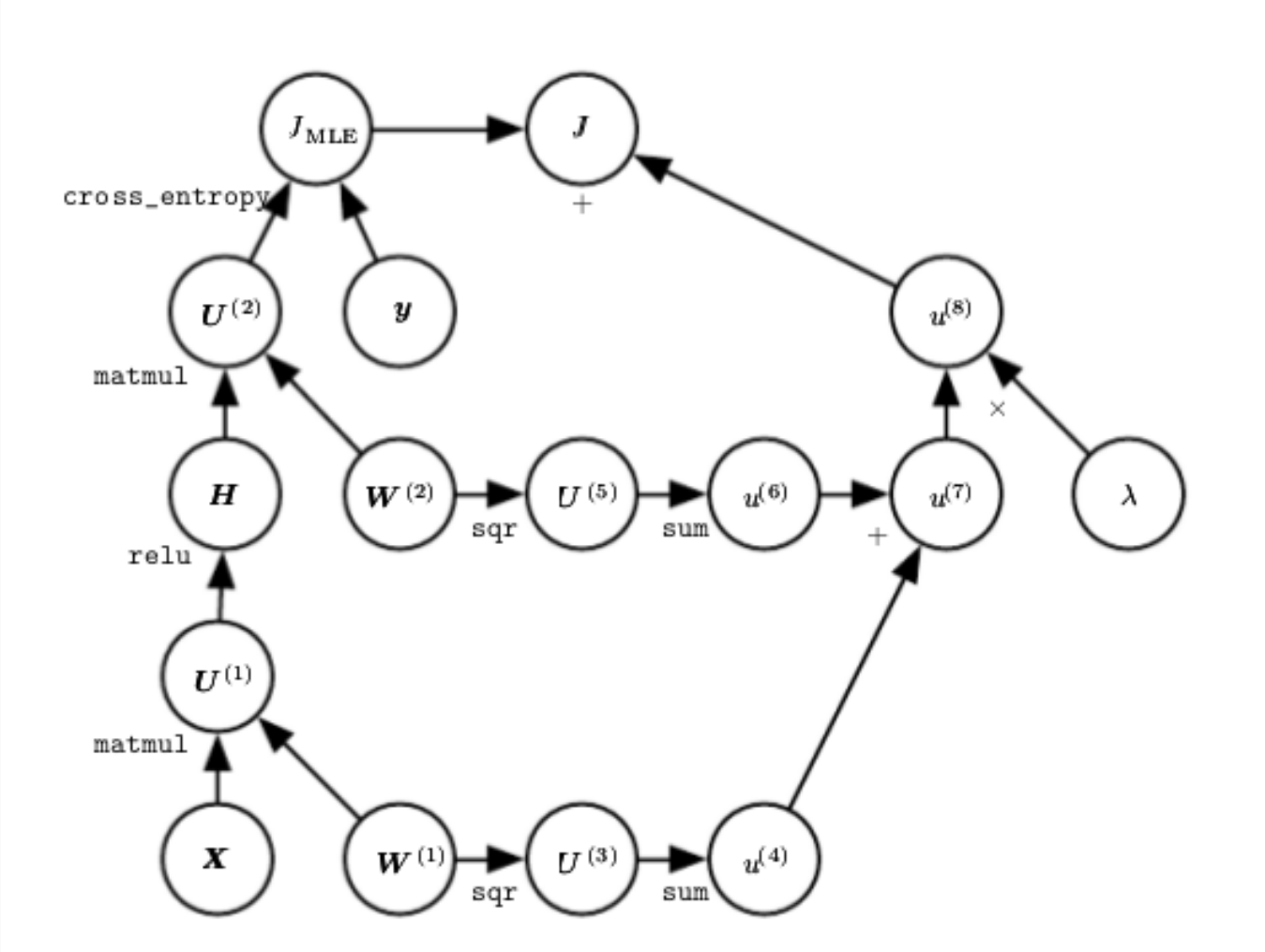
Operacije vraćaju samo jedan izlaz (radi jednostavnosti)

- izlaz može biti skalar, vektor, matrica ili tenzor
- stoga ovo ograničenje ne smanjuje općenitost.

Primjer: klasifikacijski model s dva potpuno povezana sloja i L2 regularizacijom

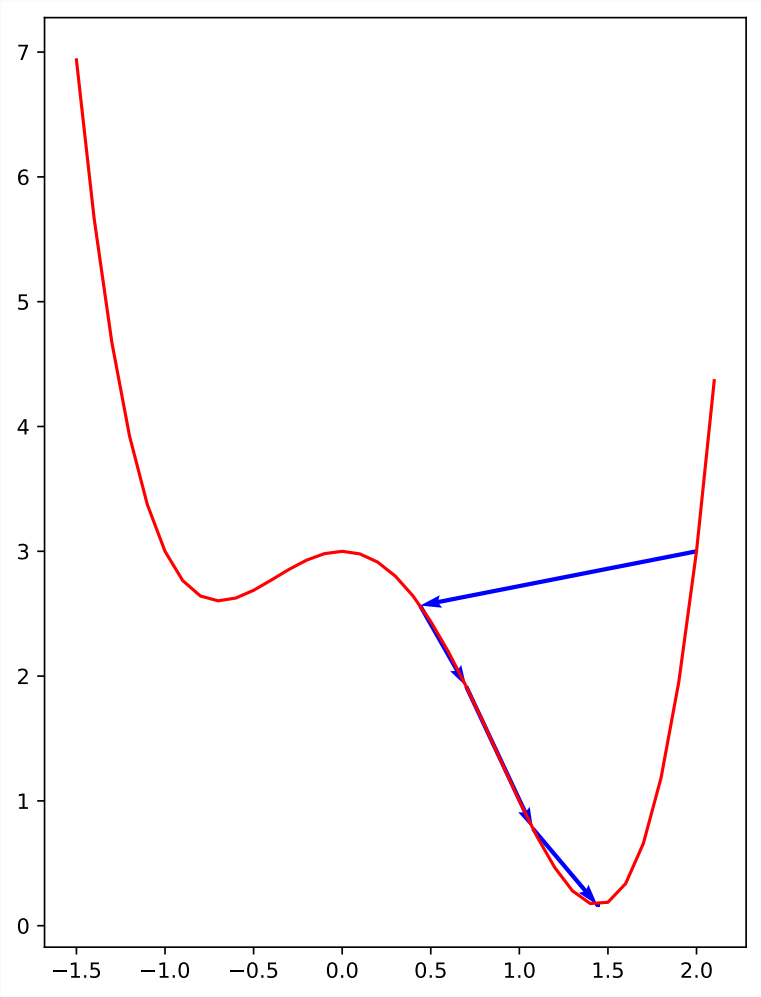
- radi jednostavnosti izostavit ćemo pomake $\mathbf{b}$

Prikaz dubokog modela računskim grafom



PyTorch hello world

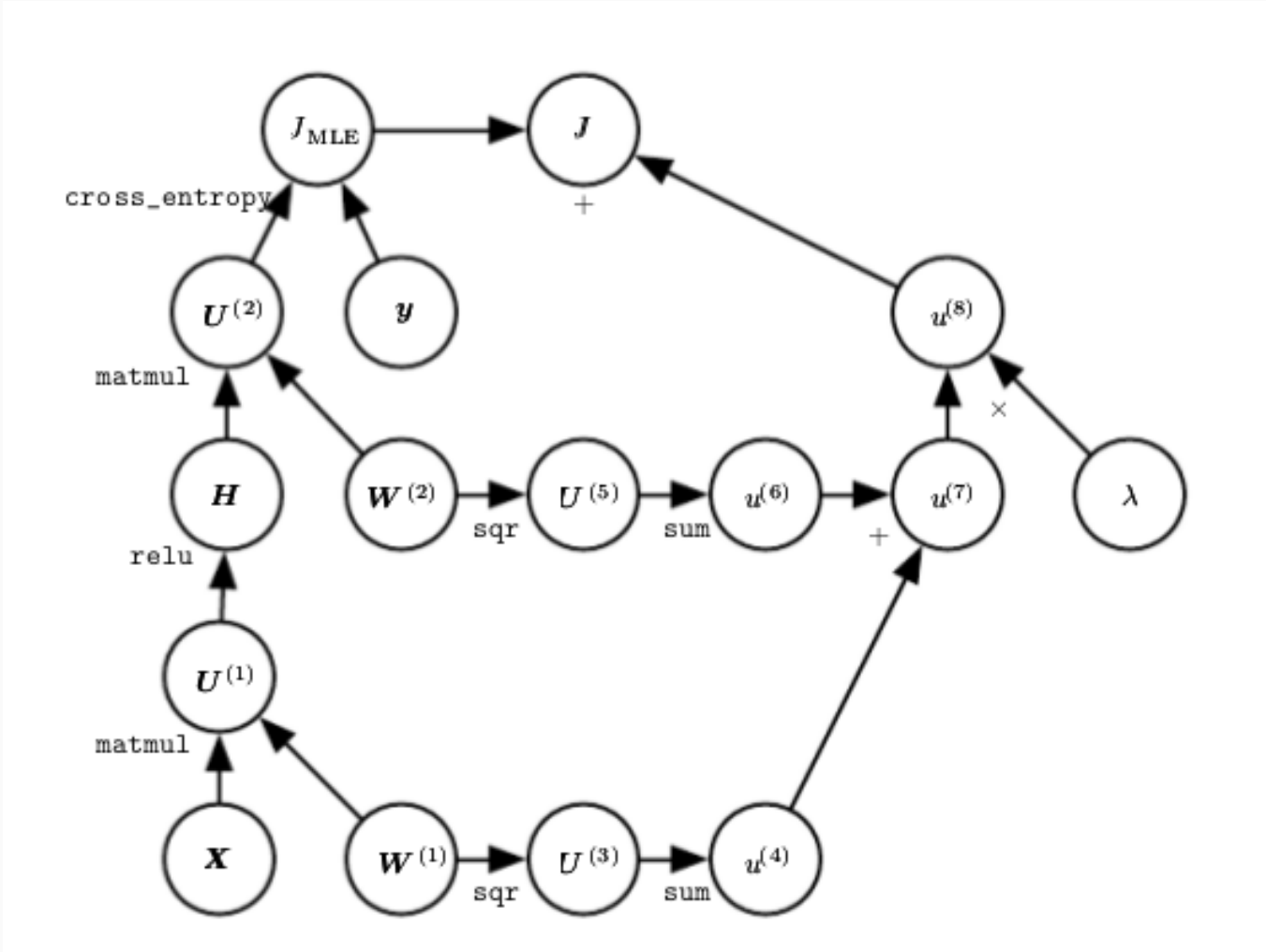
```
import torch

step = 0.13

x = torch.tensor(2.0,
      requires_grad=True)

for i in range(100):
  y = x**4 - x**3 - 2*x**2 + 3
  y.backward()
  print(x, y, x.grad)

  x.data = x - step * x.grad
  x.grad.zero_()
```



Glavni sastojak: unatražno automatsko differenciranje

Domaći rad: riješiti $x^5 - x^4 - x = -1$ gradijentnim spustom

Vratimo se na naš računski graf



Odgovarajući kôd pod PyTorchem

```
import torch, torch.nn.functional as F

# roots: input, label, parameters, hiperparameter
x = torch.tensor([1.,1.])
y = torch.tensor(0.)
W1 = torch.tensor([[0.5,0], [0,1]], requires_grad=True)
W2 = torch.tensor([1.,0.], requires_grad=True)
lambda1 = torch.tensor(0.01)

# model
h1 = torch.relu(W1 @ x)
JMLE = F.binary_cross_entropy_with_logits(W2 @ h1, y)
J = JMLE + lambda1 * (W1.pow(2).sum() + W2.pow(2).sum())

# ask autograd to compute the gradients
J.backward()
print(W1.grad)
```