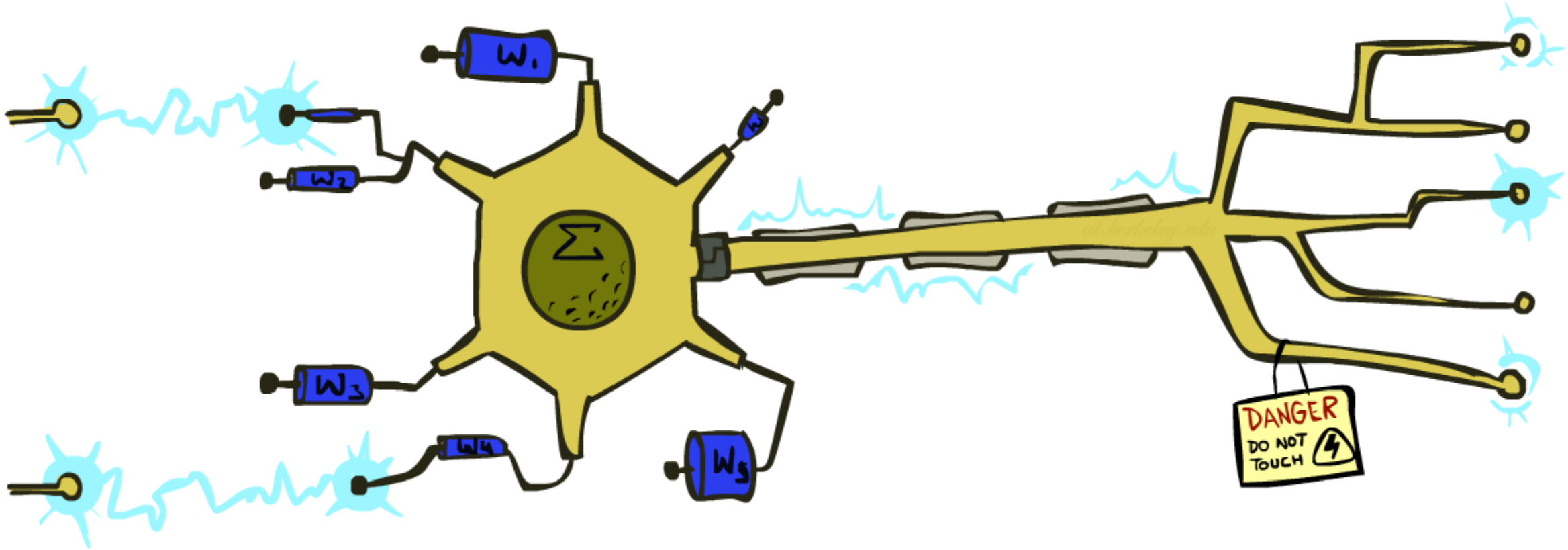
Osnovi Računarske Inteligencije

Perceptron

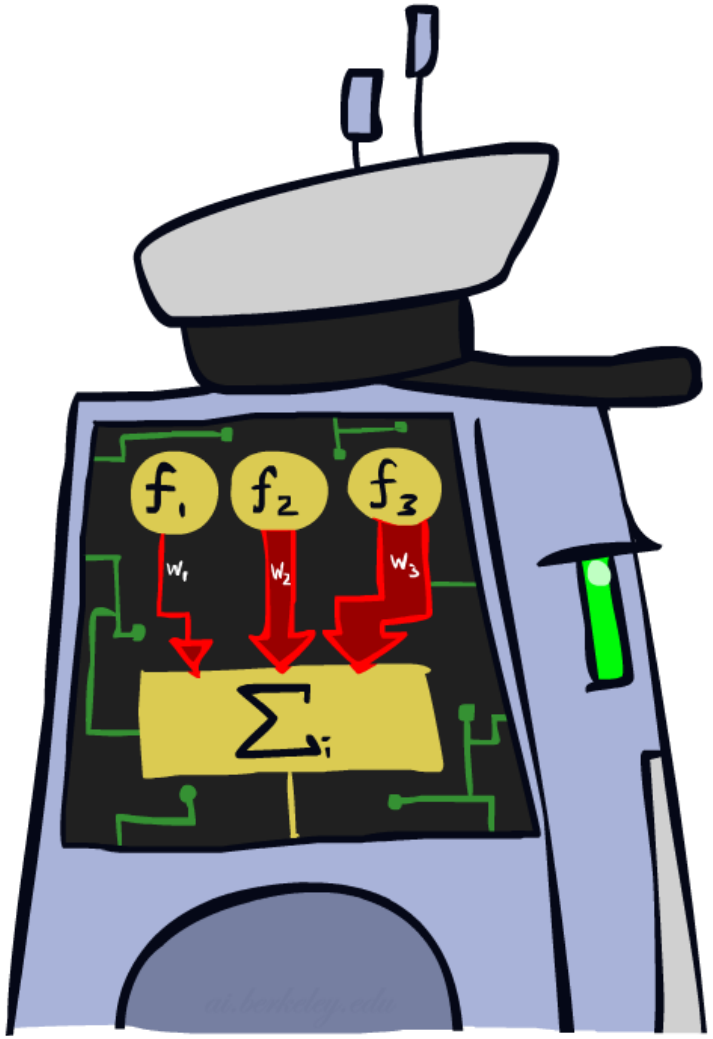


Predavač: Aleksandar Kovačević

Slajdovi preuzeti sa kursa CS188, University of California, Berkeley

http://ai.berkeley.edu/

Linearni Klasifikatori



Vektori Osobina (*Feature Vectors*)

x $\qquad$ $f(x)$ $\qquad$ y

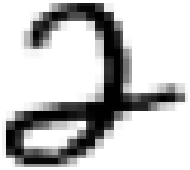
```
Hello,

Do you want free printr
cartriges?  Why pay more
when you can get them
ABSOLUTELY FREE!  Just
```

$\Rightarrow$

```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

$\Rightarrow$

SPAM
or
+



$\Rightarrow$

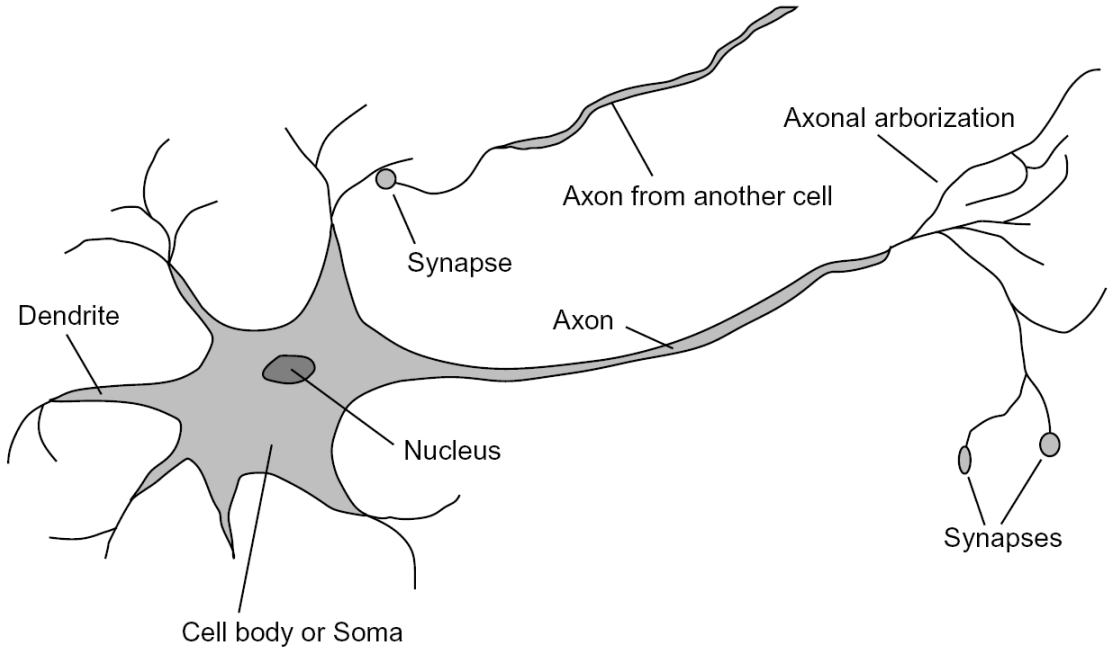
```
PIXEL-7,12   : 1
PIXEL-7,13   : 0
...
NUM_LOOPS    : 1
...
```

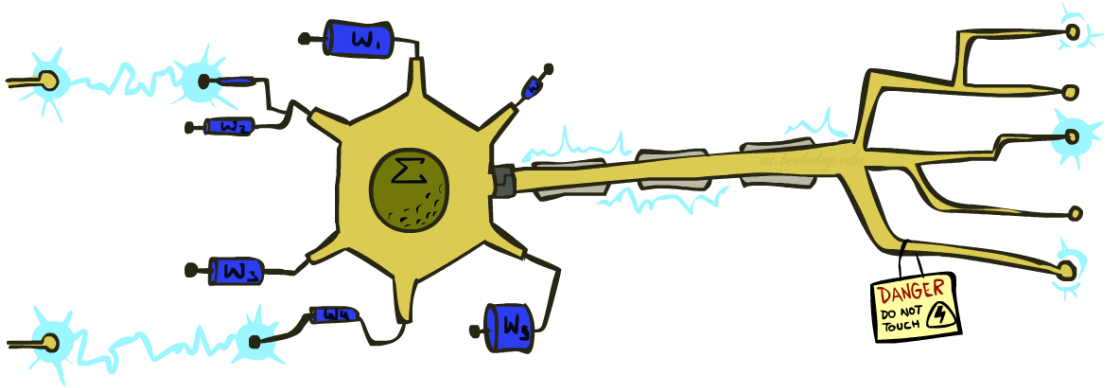
$\Rightarrow$

“2”

Inspiracija iz Biologije

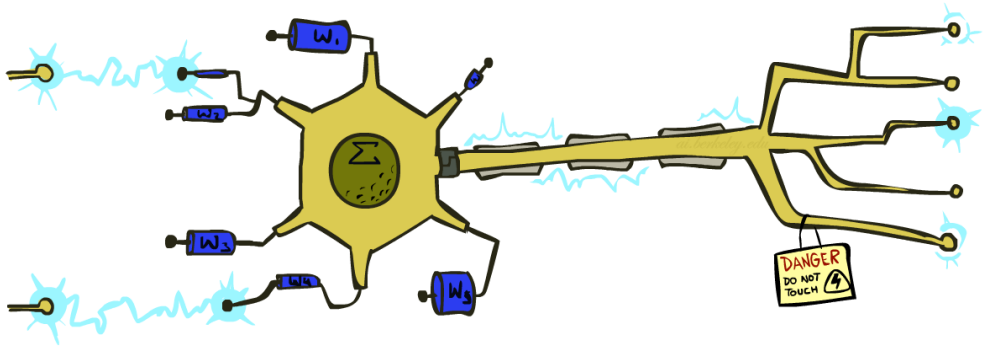
- Perceptron se može posmatrati kao vrlo uprošćen ljudski neuron





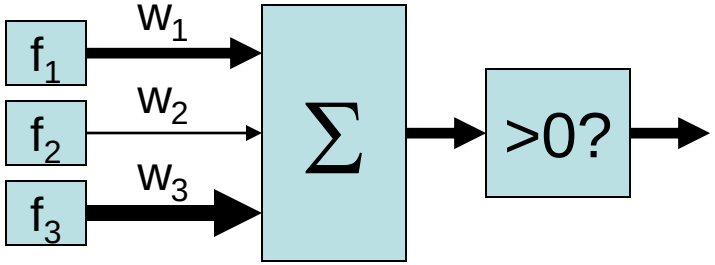
Linearni Klasifikatori

- Ulazi su vrednosti osobina
- Svaka osobini dodljena je težina
- Suma težina i vrednosti je aktivacija



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- Kod preceptrona:
- Ako je aktivacija:
  - Pozitivna, izlaz je +1
  - Negativna, izlaz je -1

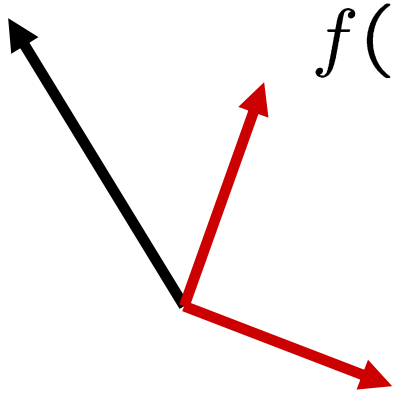


Težine

- Ako imamo dve klase: poredimo vektor osobina sa vektorom težina (gledamo ugao između njih)
- Učenje: Određivanje vektora težina na osnovu obučavajućeg skupa

w

```
# free       : 4
YOUR_NAME    :-1
MISSPELLED   : 1
FROM_FRIEND  :-3
...
```



$f(x_1)$

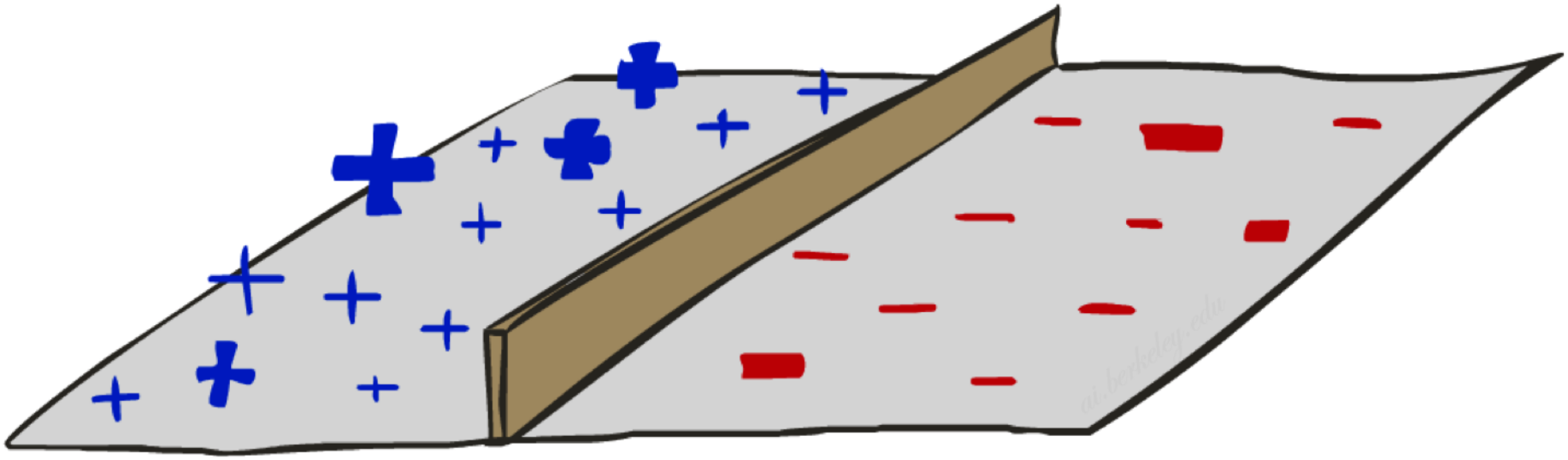
```
# free       : 2
YOUR_NAME    : 0
MISSPELLED   : 2
FROM_FRIEND  : 0
...
```

$f(x_2)$

```
# free       : 0
YOUR_NAME    : 1
MISSPELLED   : 1
FROM_FRIEND  : 1
...
```

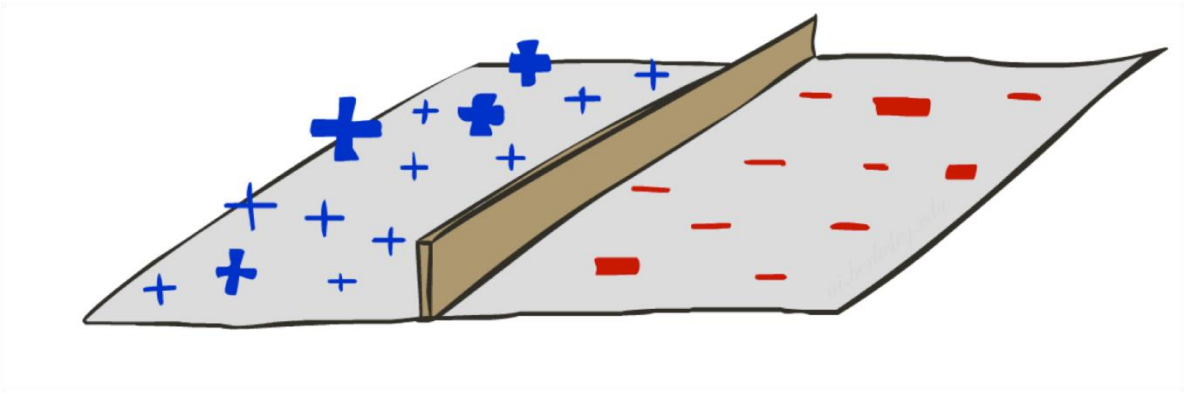
Skalarni proizvod $w \cdot f$ > 0 tj. ugao između w i f je manji od 90 stepeni znači da je predikcija za f pozitivna klasa

Granica Odluke (*Decision Boundary*)



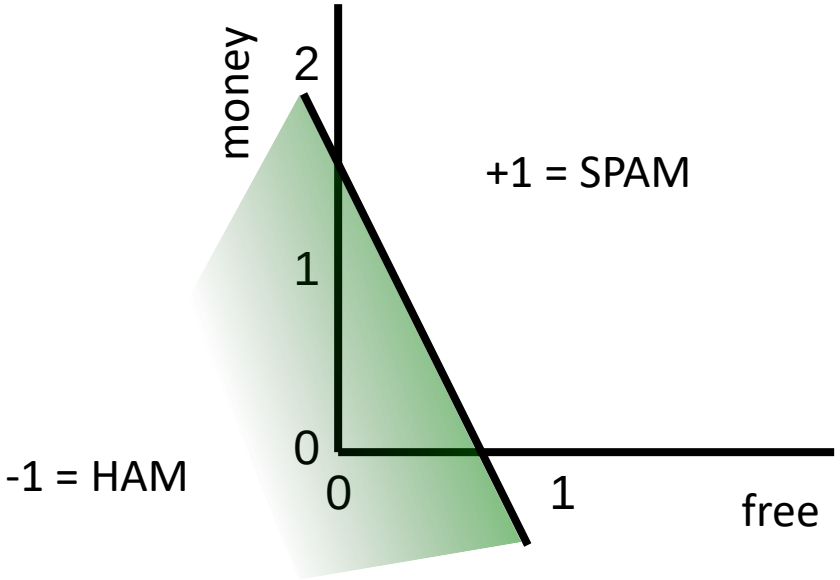
Binarna Granica Odluke

- U prostoru vektora osobina
  - Primeri iz ob. skupa su tačke
  - Svaki vektor težina je hiper-ravan
  - Jednoj strani odgovara pozitivna klasa Y=+1
  - Drugoj negativna Y=-1



$$w$$

```
BIAS  : -3
free  :  4
money :  2
...
```

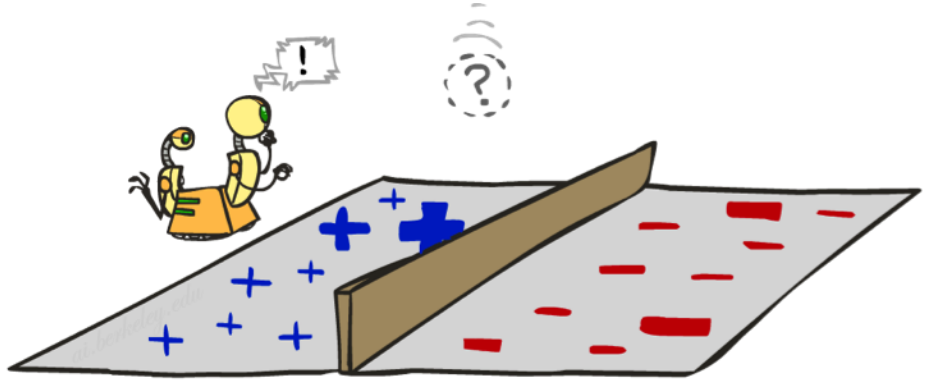


$$f \cdot w = 0$$

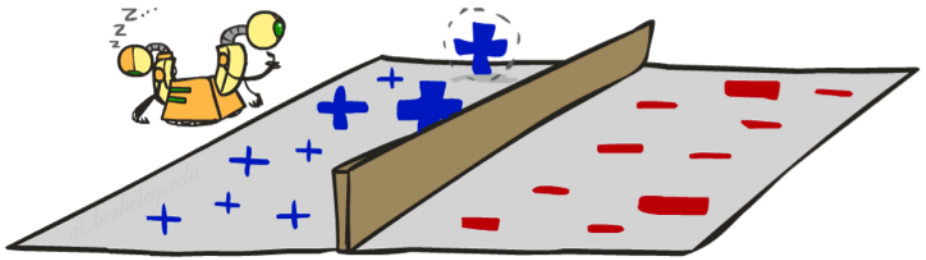
Promene (Korekcije) Težina

Obučvavanje Binarnog Perceptrona

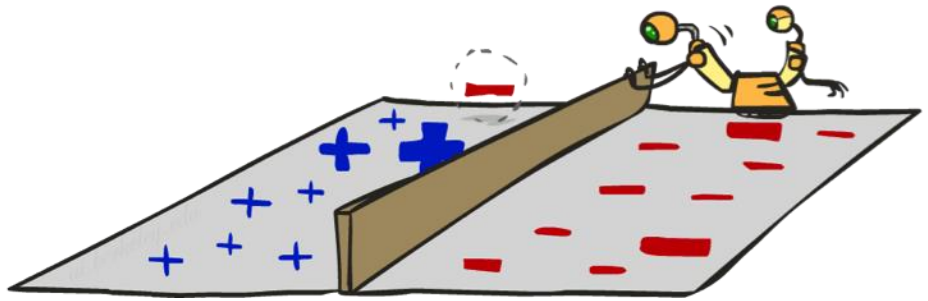
- Na početku sve težine su = 0 (ili slučajno odabrane vrednosti)
- Za svaki primer iz ob skupa radimo:
  - Klasifikujemo ga pomoću trenutnih težina



  - Ako je klasifikacija tačna (tj. y=y*) onda nema promene težina!
  - y* je prava klasa primera



  - Ako je pogrešna: menjamo vektor težina



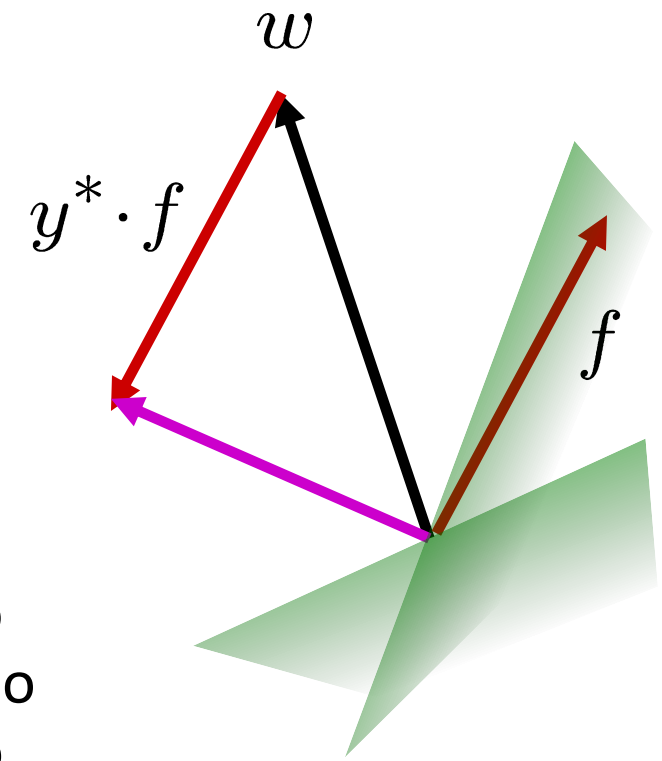
Obučvavanje Binarnog Perceptrona

- Na početku sve težine su = 0
- Za svaki primer iz ob skupa radimo:
  - Klasifikujemo ga pomoću trenutnih težina

$$y = \begin{cases} +1 & \text{if} \quad w \cdot f(x) \geq 0 \\ -1 & \text{if} \quad w \cdot f(x) < 0 \end{cases}$$

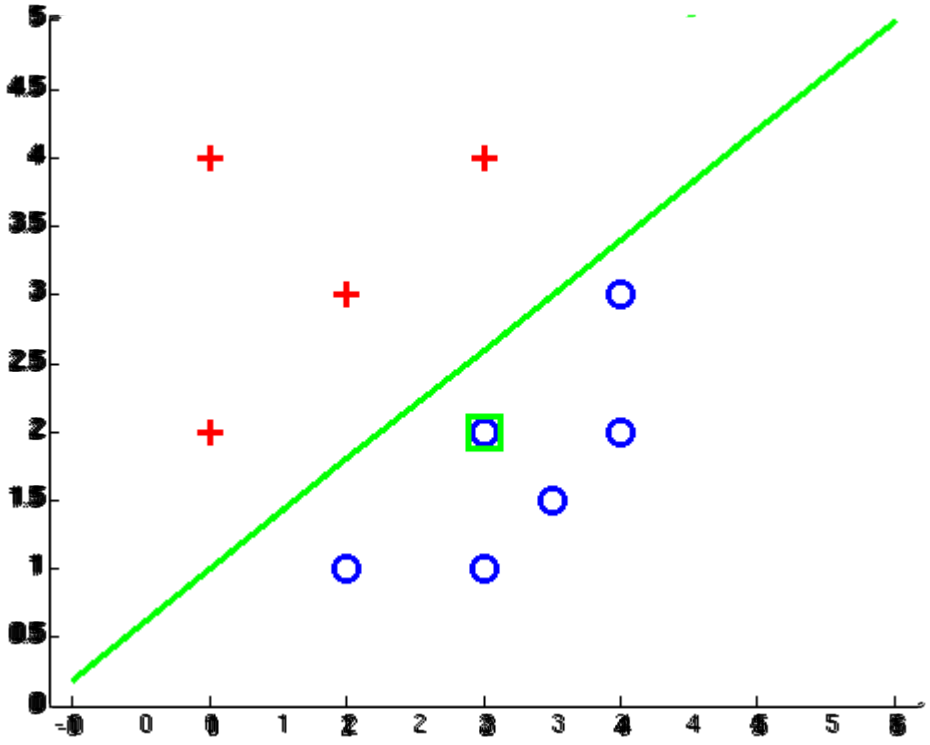
  - Ako je klasifikacija tačna (tj. y=y*) onda nema promene težina!
  - Ako je pogrešna: menjamo vektor težina tako što na njega dodajemo ili od njega oduzimamo vektor osobina tog primera. To je realizovano kroz formulu dole. Npr. ako je y*= -1 biće oduzimanje f.

$$w = w + y^* \cdot f$$



Primer

- Pimeri su linearno razdvojivi po klasi



Više-klasna Granica Odluke

- Ako imamo više klasa:
  - Imamo vektor težina za svaku klasu:

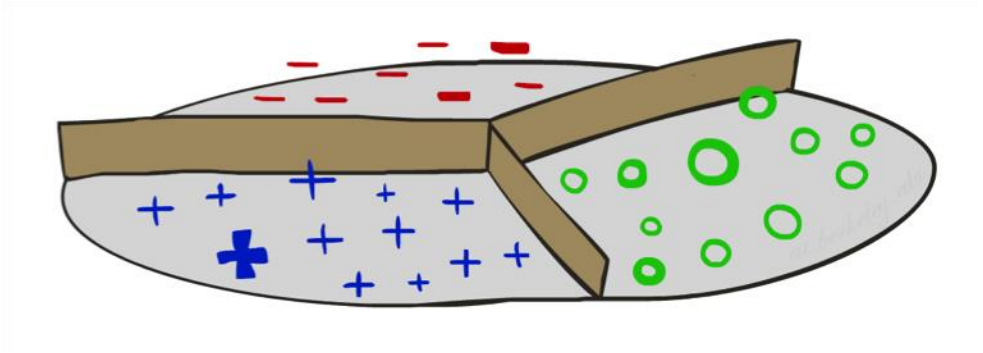
$$w_y$$

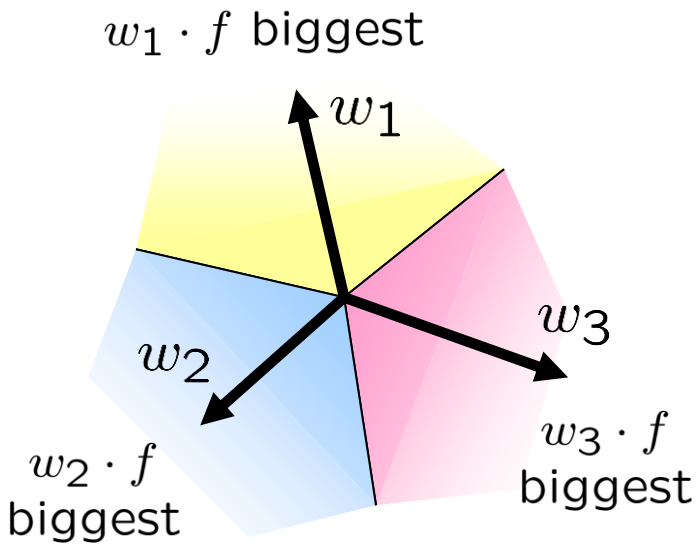
  - Za svaku klasu računamo aktivaciju:

$$w_y \cdot f(x)$$

  - Predikcija sa najvećom aktivacijom je klasa koju vraćamo

$$y = \arg\max_y \; w_y \cdot f(x)$$





Obučavanje Više-klasnog Perceptrona

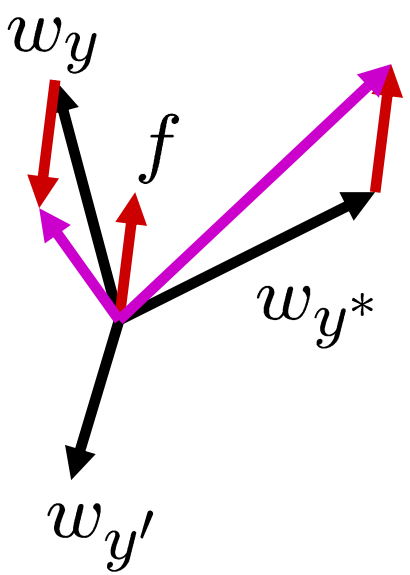
- Na početku sve težine su = 0
- Idemo redom po primerima iz ob. skupa
- Određujemo klasu pomoću formule:

$$y = \arg\max_y \; w_y \cdot f(x)$$

- Ako je tačna, ništa ne menjamo!
- Ako je pogrešna: smanjujemo aktivaciju pogrešnog odgovora, povećavamo aktivaciju tačnog odgovora

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$

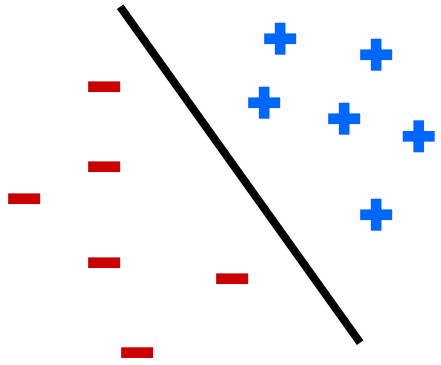


- Pogrešan odgovor je vektor težina zbog koga je primer dobio pogrešnu klasu (u argmax).
- Tačan odgovor je vektor težina koji odgovara pravoj klasi primera.

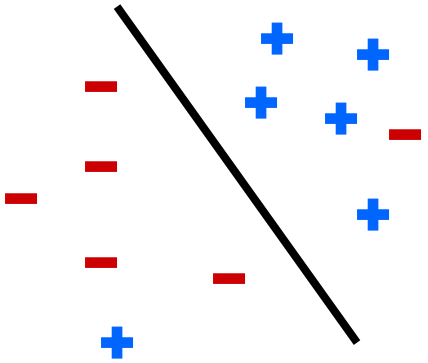
Karakteristike Perceptrona

- Može linearno da razdvoji dve ili više klasa
- Postoji teorijski dokaz da ako su podaci linearno separabili preceptron algoritam će sigurno da pronađe rešenje tj. da konvergira

Separable

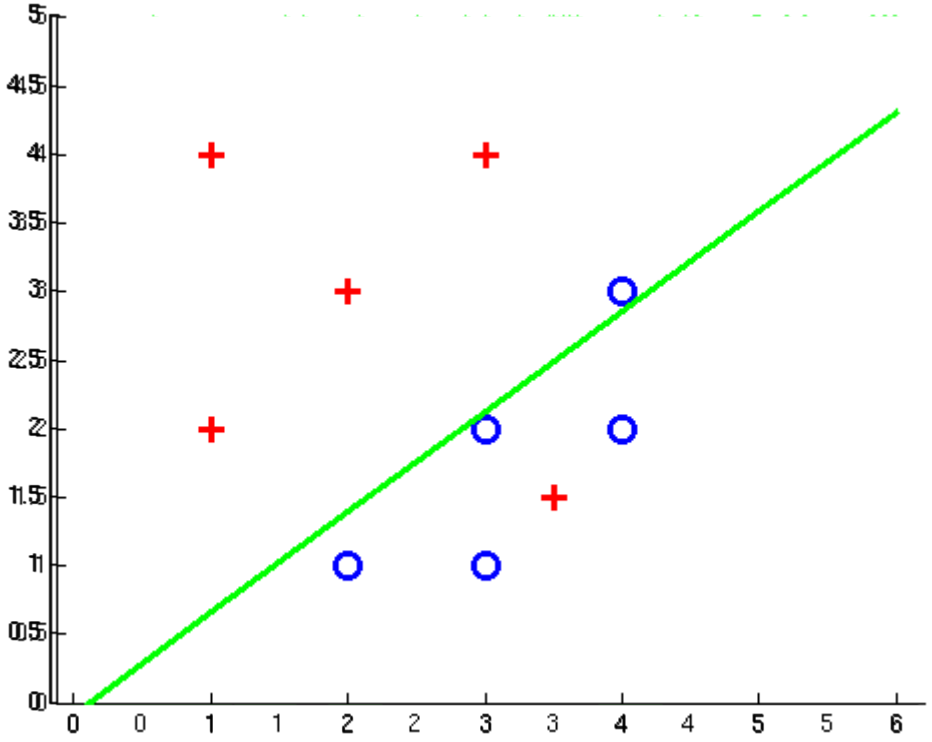


Non-Separable



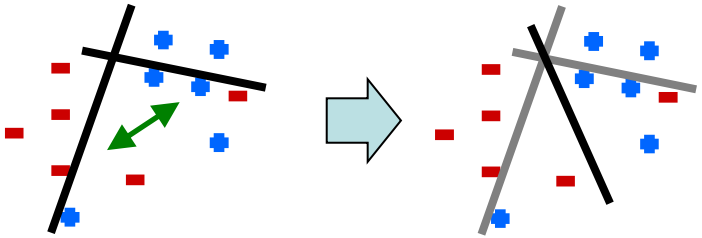
Primer

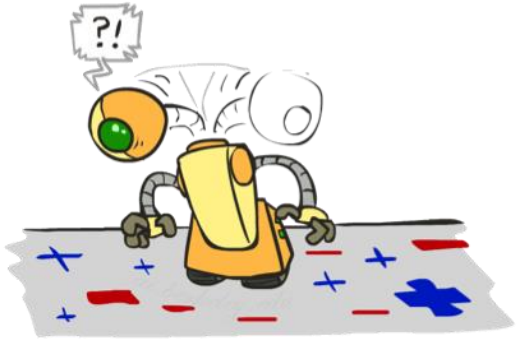
- Linearno ne-separabilni podaci



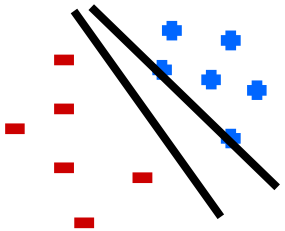
Problemi sa Perceptronom

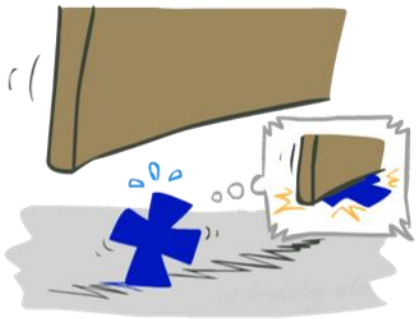
- Šum: ako su podaci skoro linearno razdvojivi možemo dobiti večito „šetanje“ vektora težina tj. divergenciju



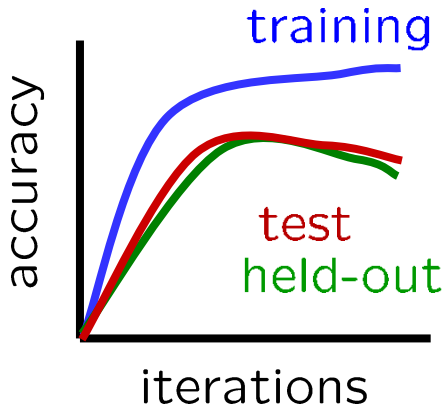


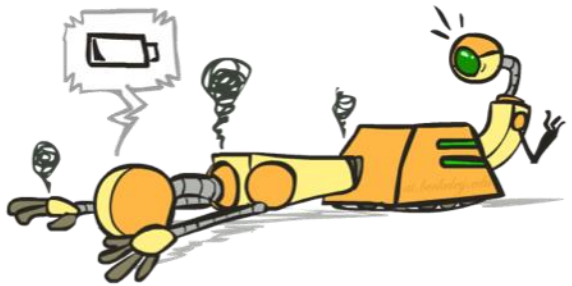
- Generalizacija nije baš dobra: nalazi rešenje, ali ne ostavlja mnogo „lufta“ između klasa (termin iz literature za „luft“ je margina separacije)





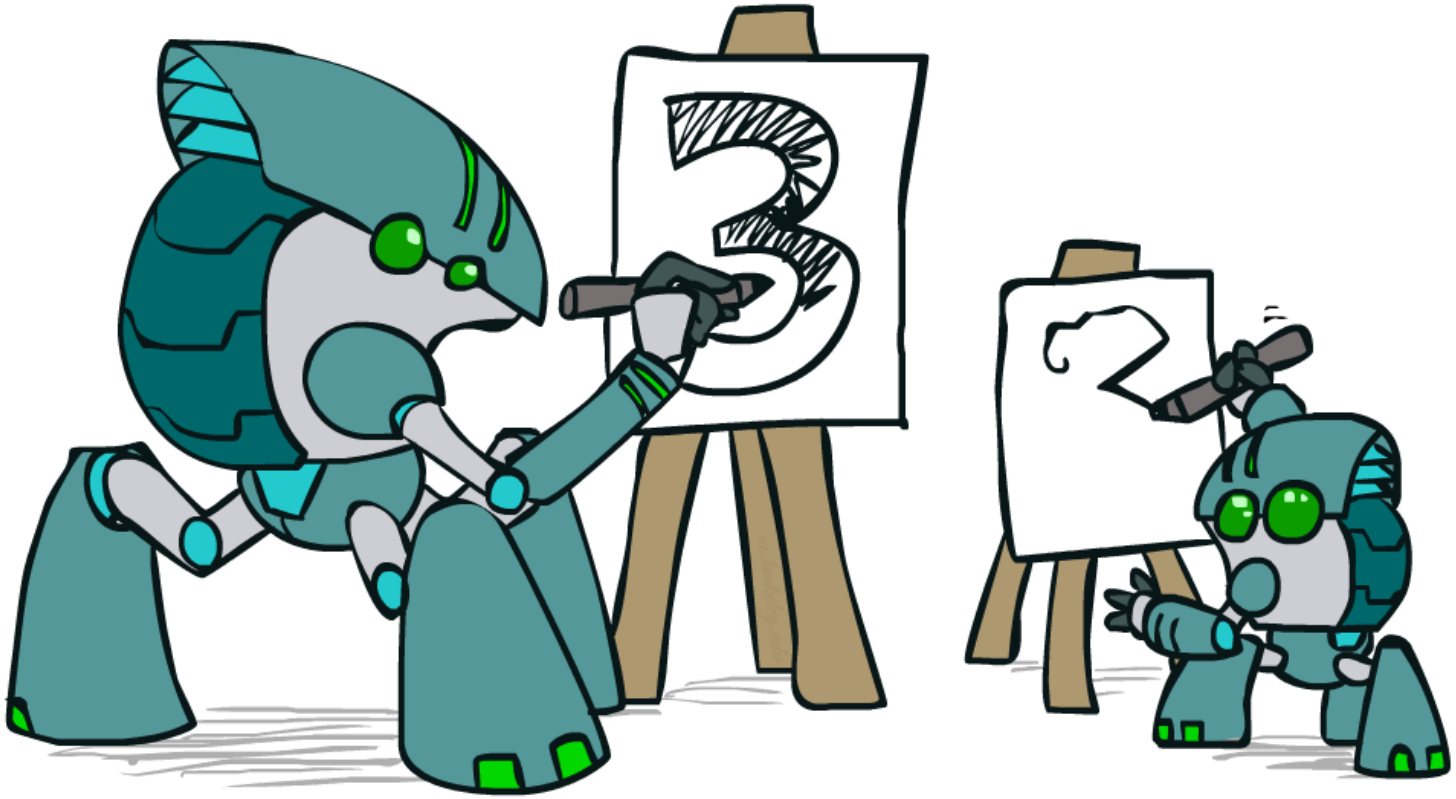
- Često se dešava da tokom iteracija tačnost modela raste do nekog momenta, pa onda krene da opada
  - Dobro je pamtiti iteraciju u kojoj je model imao najveću tačnost



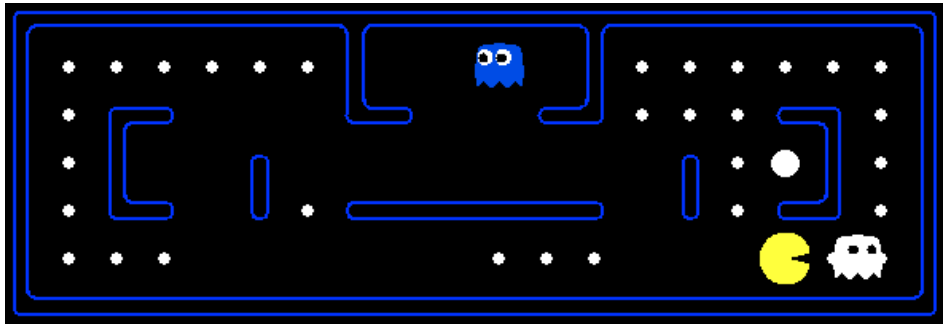


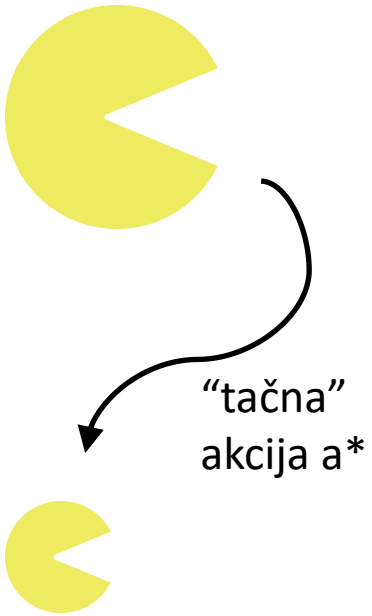
„Šegrtovanje“ - *Apprenticeship*

- Ideja kod ovog načina učenja je da učimo od eksperta tj. da mu budemo šegrt
- Ekspertovo znanje može da se reprezentuje kao klasa,
- dok se stanje sveta tj. opis problema može reprezentovati kao vektor osobina
- Klasa kaže šta bi ekspert uradio u situaciji koja je reprezentovana vektorom osobina



Primer - Pacman!



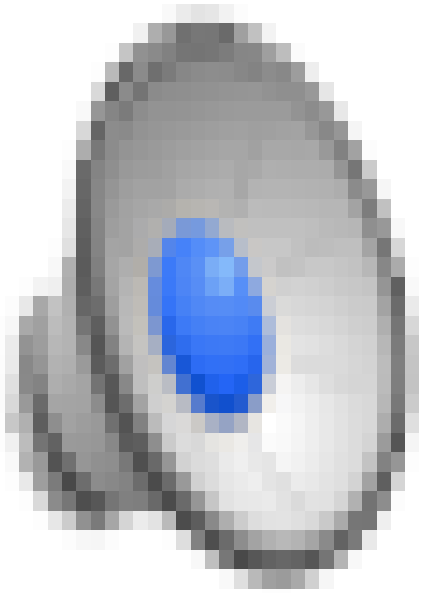


$$\forall a \neq a^*,$$
$$w \cdot f(a^*) > w \cdot f(a)$$

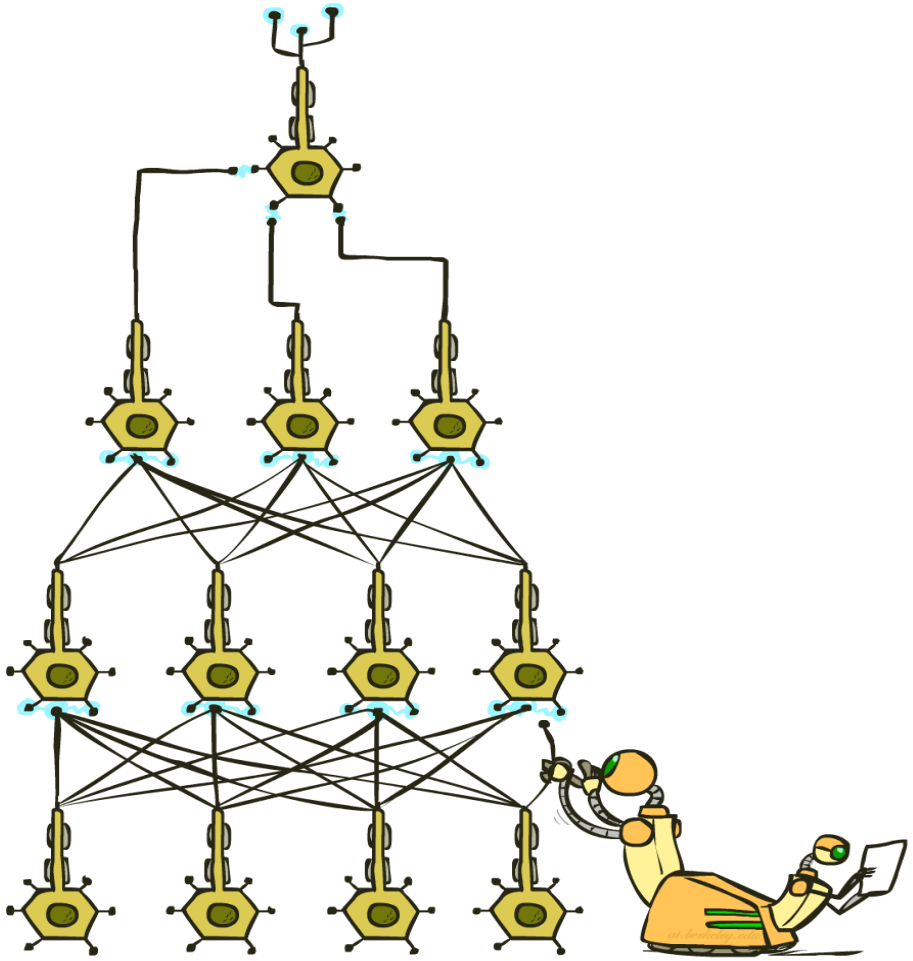
- Kandidati za klasifikator su parovi stanje-akcija (s,a)
- Ekspert svakom paru stanje-akcija (s,a) dodeljuje +1 ili -1
- , u zavisnosti da li je to akcija koju bi on uradio u stanju s
- Ceo vektor osobina je reprezentacija i stanja i akcije (npr. udaljenost duhova, broj pojedenih kuglica id.)
- Aktivacija za svaki par (s,a) data je sa:

$$w \cdot f(s, a)$$

Demo



Osnovi Računarske Inteligencije



Uvod u Neuronske Mreže

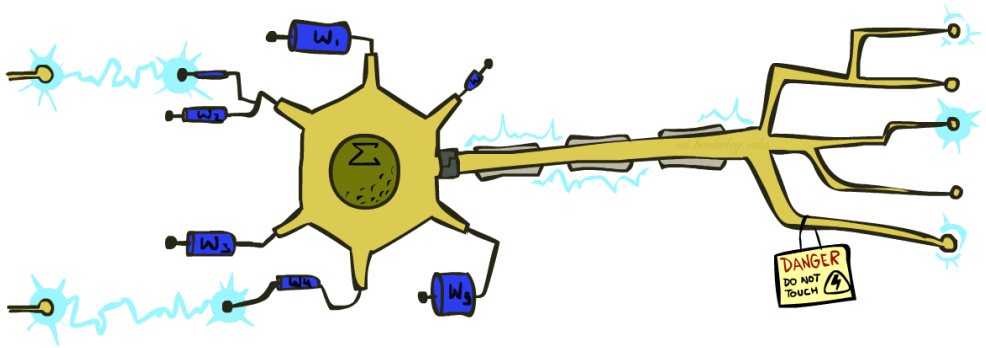
Predavač: Aleksandar Kovačević

Slajdovi preuzeti sa kursa CS188, University of California, Berkeley

http://ai.berkeley.edu/

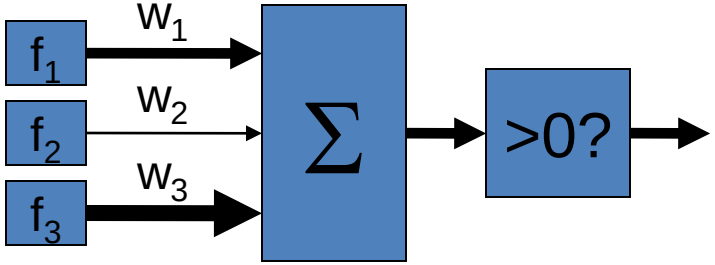
Prošli put: Perceptron

- Ulazi su vrednosti osobina
- Svakoj osobini dodeljena je težina
- Suma je aktivacija

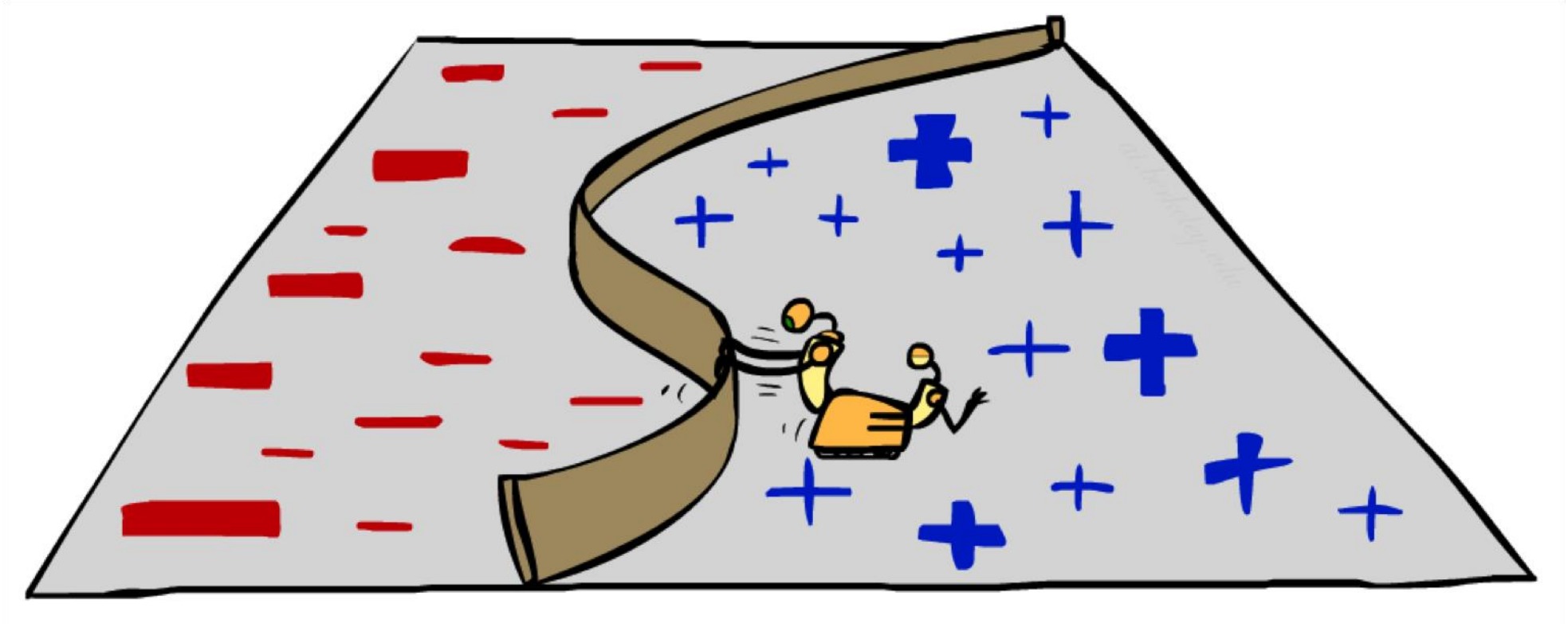


$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

- Ako je aktivacija:
  - Pozitivna, izlaz je +1
  - Negativna, izlaz je -1

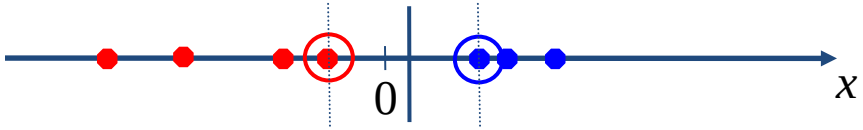


Ne-linearna Granica Odluke

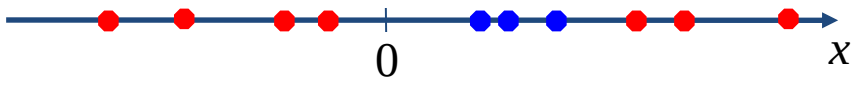


Ne-linearni Klasifikatori

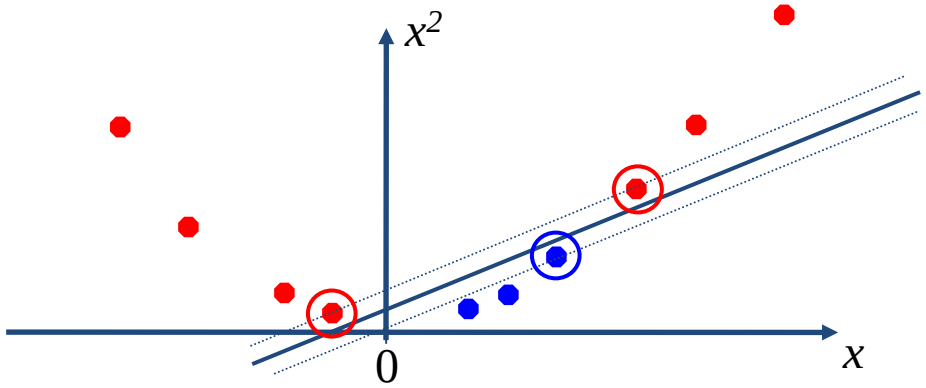
- Ako su podaci linearno razvojivi, percetron i slični modeli (SVM) rade dobro:



- Šta ćemo da uradimo ako dobijemo stvano težak klasifikacioni problem?



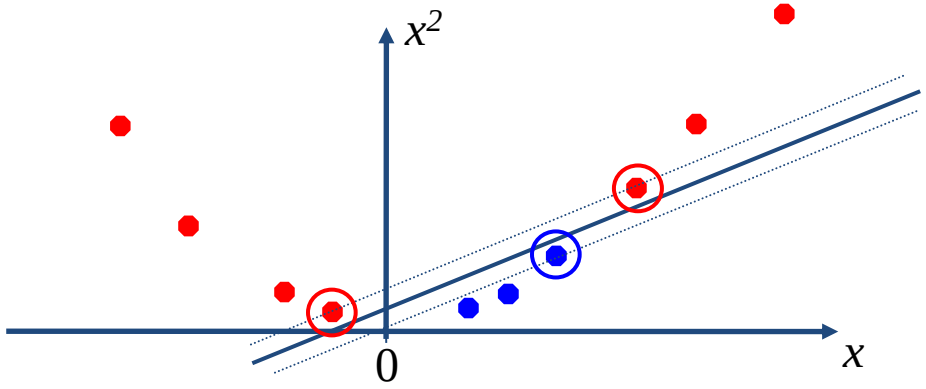
- Da li možemo da mapiramo podatke na prostor u kome su linearno razdvojivi?:



This and next slide adapted from Ray Mooney, UT

Ne-linearni Klasifikatori

- Da li možemo da mapiramo podatke na prostor u kome su linearno razdvojivi?:



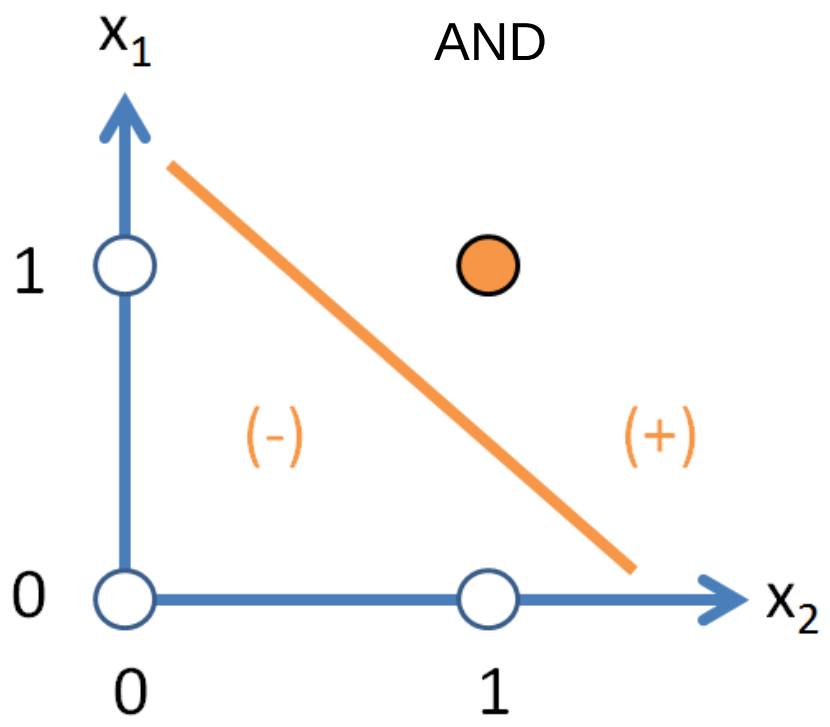
- Ovo jeste opcija i radićete je na više predmeta kasnije, ali
- nije uvek lako odrediti način mapiranja, uz to postoje i mnoge druge mane koje nećemo sad navoditi (npr. *overfitting*)

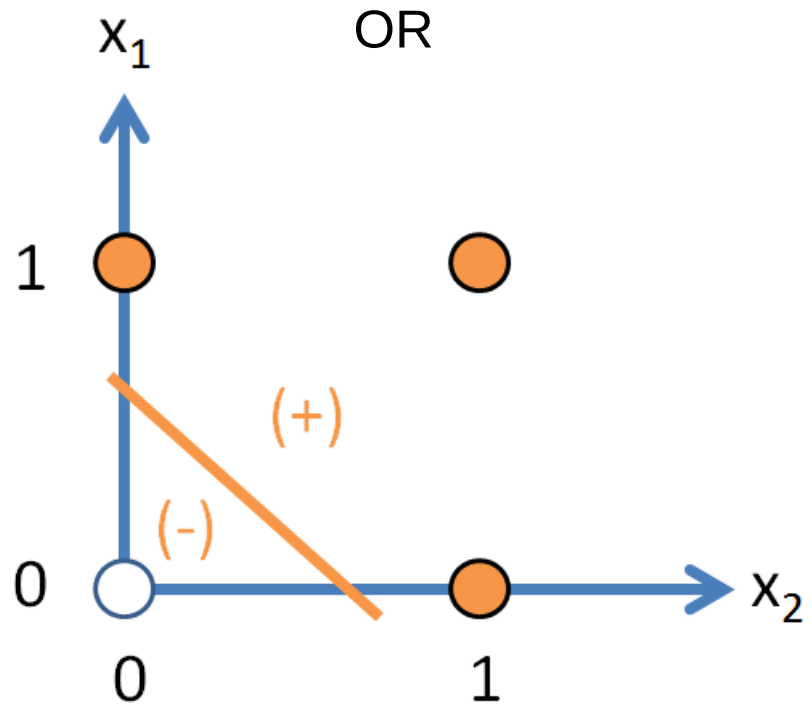
- Danas radimo Neuronske Mreže koje su idealna alternativa!

This and next slide adapted from Ray Mooney, UT

Ne-linearni Klasifikatori

- Potrebu za Neuronskim mrežama kao ne-linearnim klasifikatorima objasnićemo pomoću Perceptrona
- Konkretno realizovaćemo logičke operacije AND i OR koristeći perceptron.
- Logičko AND i OR su linearno separabilini klasifikacioni problemi.

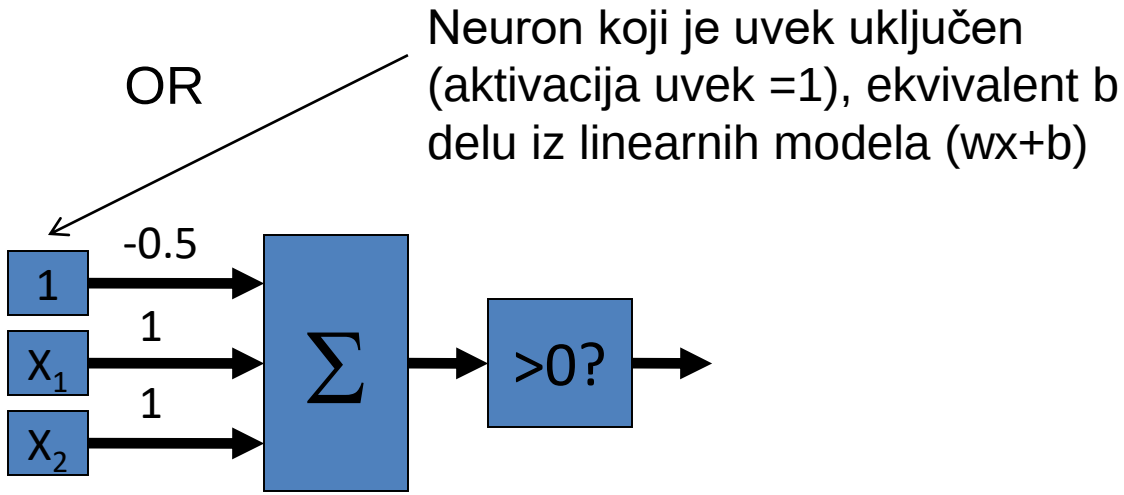


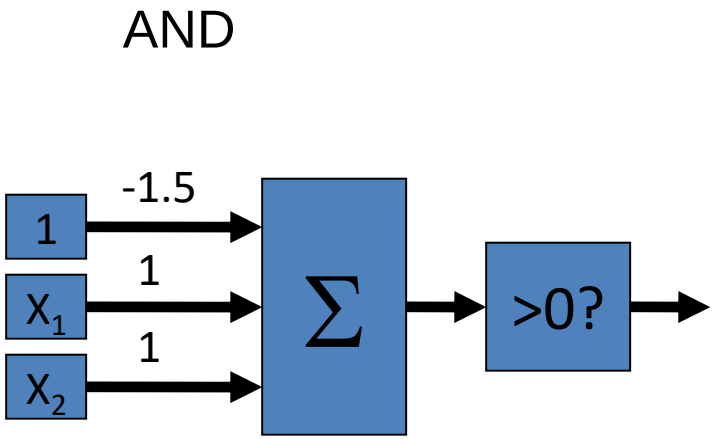


Izvor slika: http://web.cs.ucla.edu/~forns/classes/winter-2016/cs-161/week-10.html

Ne-linearni Klasifikatori

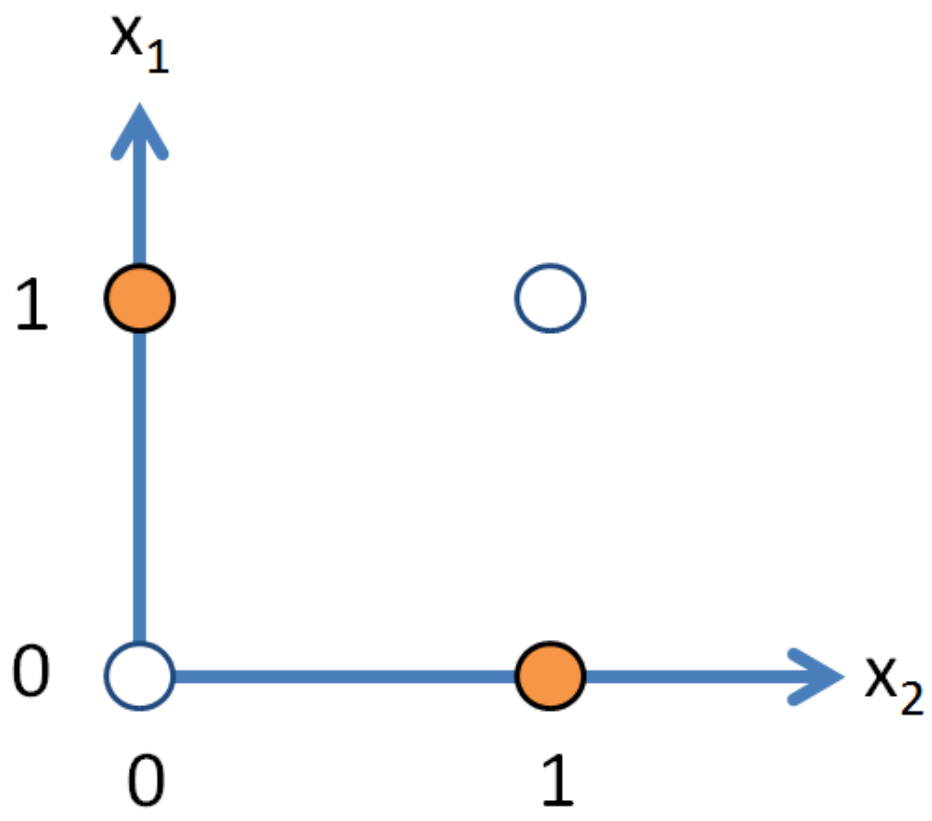
- Logičko AND i OR su linearno separabilini klasifikacioni problemi.
- Dakle svaka od ove dve operacije može se realizovati jednim perceptronom





Ne-linearni Klasifikatori

- Da li pomoću jednog perceptrona možemo da realizujemo logičku operaciju XOR?

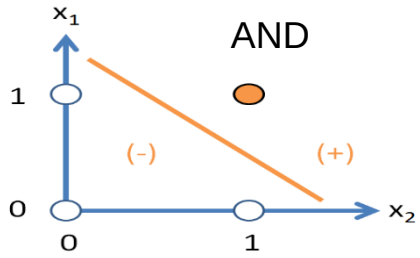


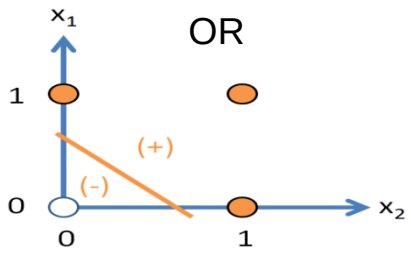
Ne-linearni Klasifikatori

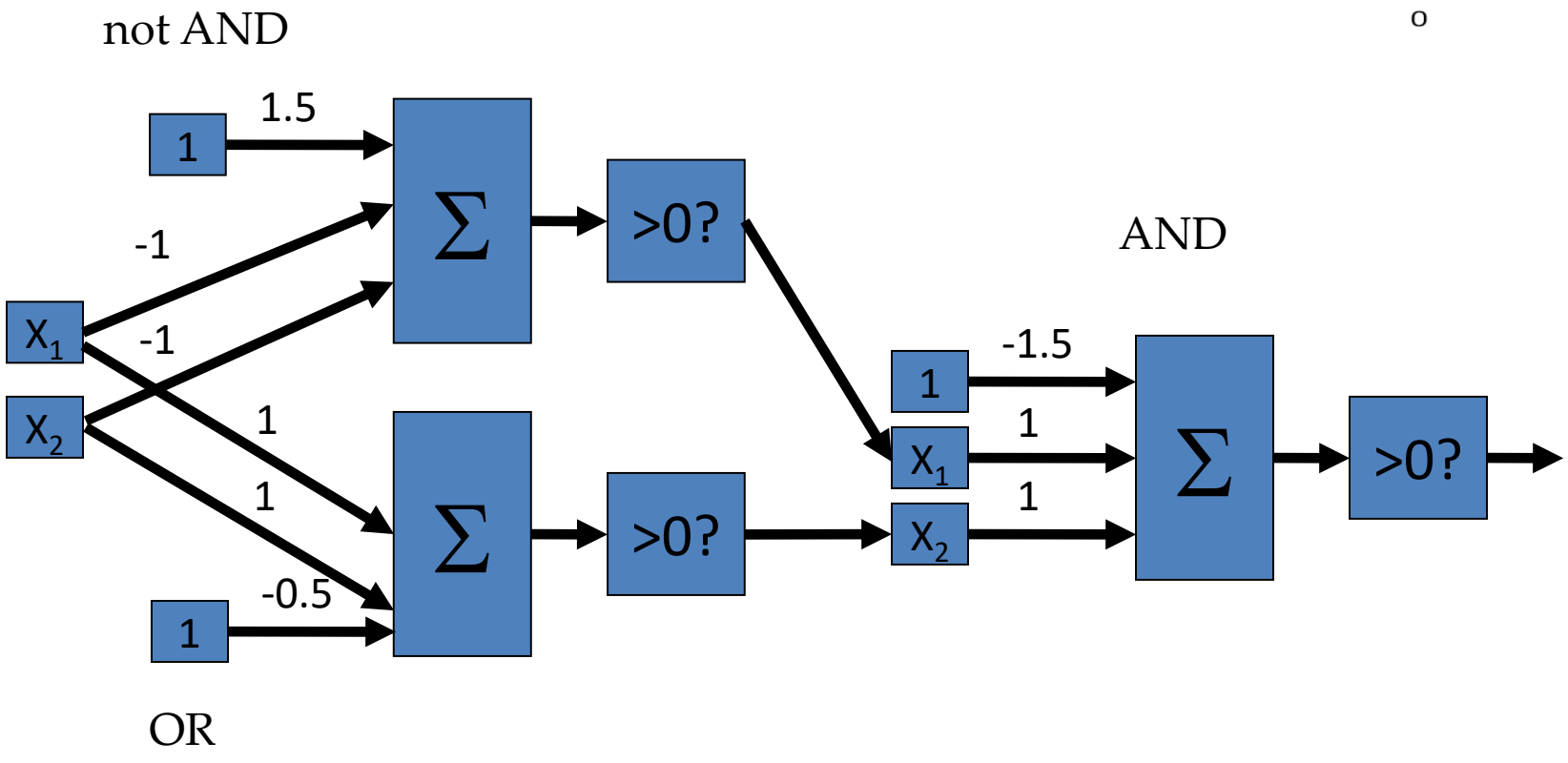
- Vidimo da XOR nije linearno separabilan problem, tako da nam jedan perceptron nije dovoljan.
- Rešenje: Upotrebićemo više perceptrona!
- Perceptrone ćemo spojiti u slojeve, izlaz jednog biće ulaz u drugi.
- Takav model zove se Višeslojni Perceptron (*Multilayer Perceptron*, MLP)

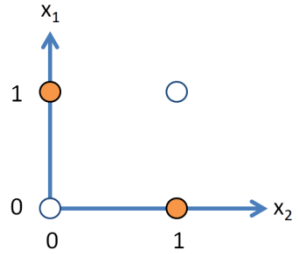
Ne-linearni Klasifikatori

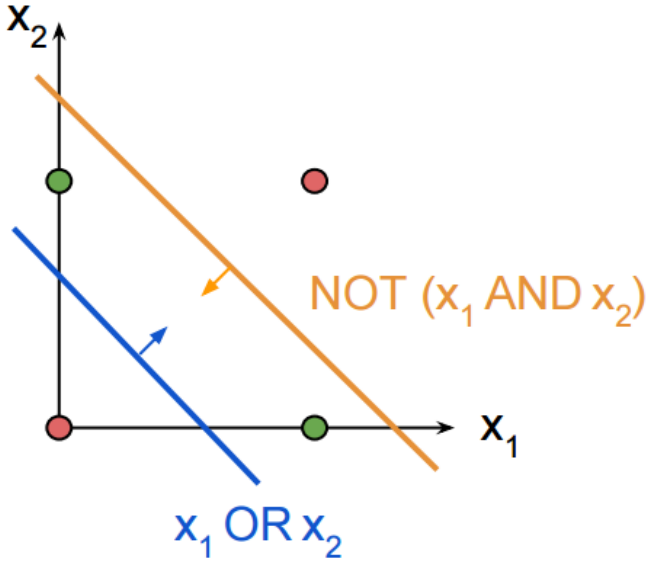
- Višeslojni perceptron za XOR problem
- XOR(x,y) = OR(x,y) AND not AND (x,y)





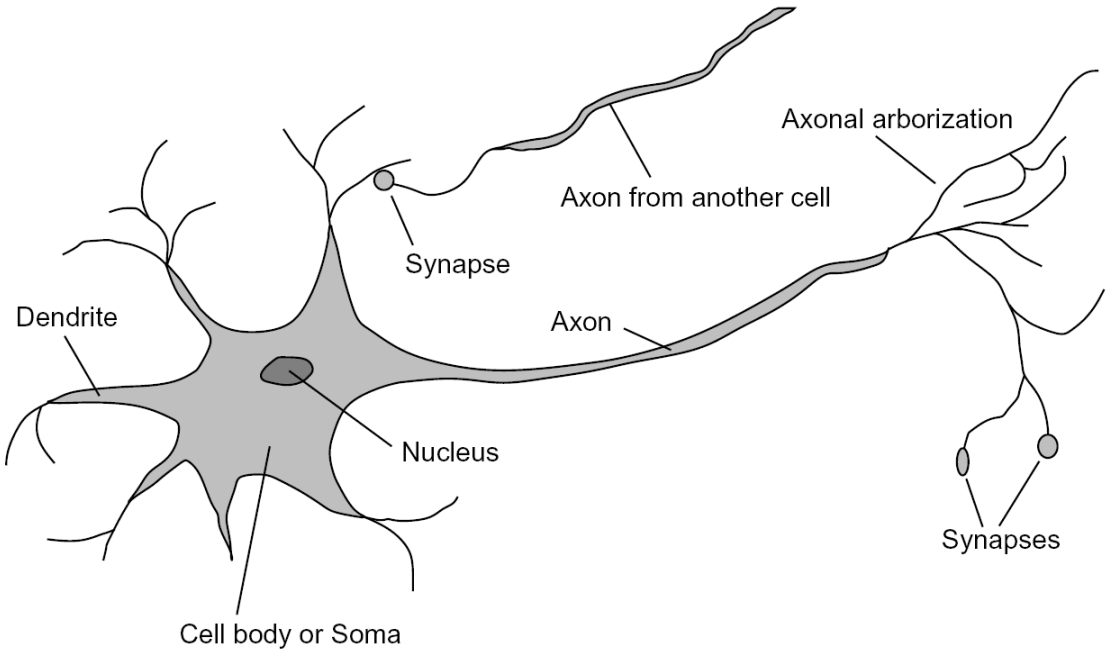


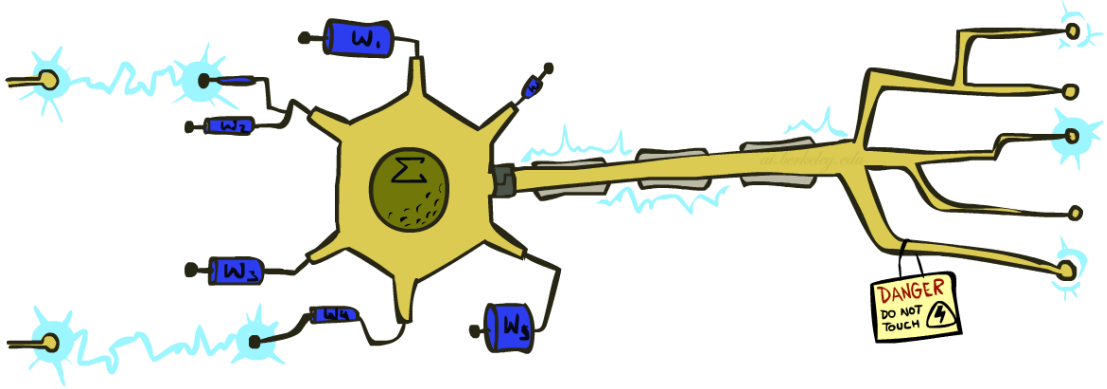




Neuronska Mreža

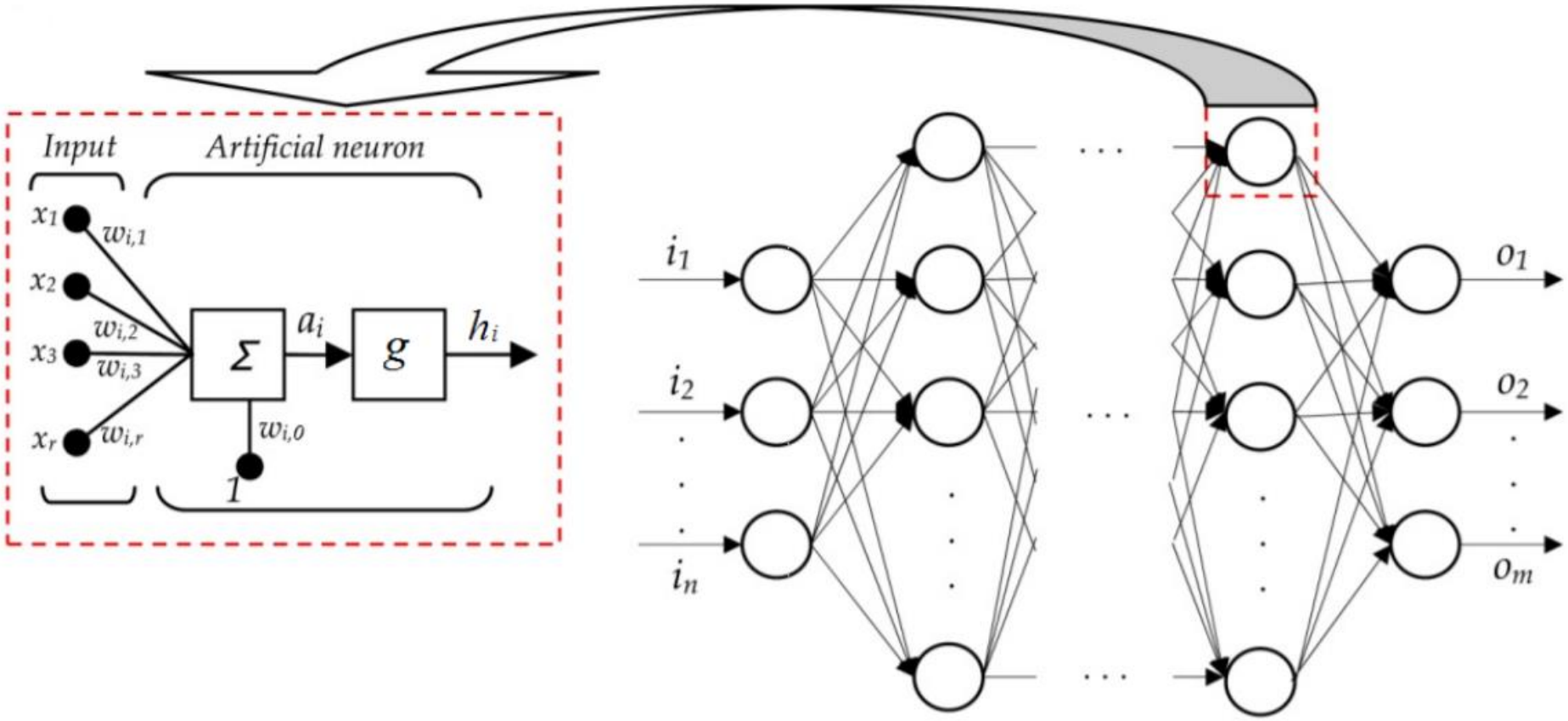
- Ako je perceptron jedan neuron, šta je onda Višeslojni Perceptron?
- Višeslojni perceptron je onda Neuronska Mreža!



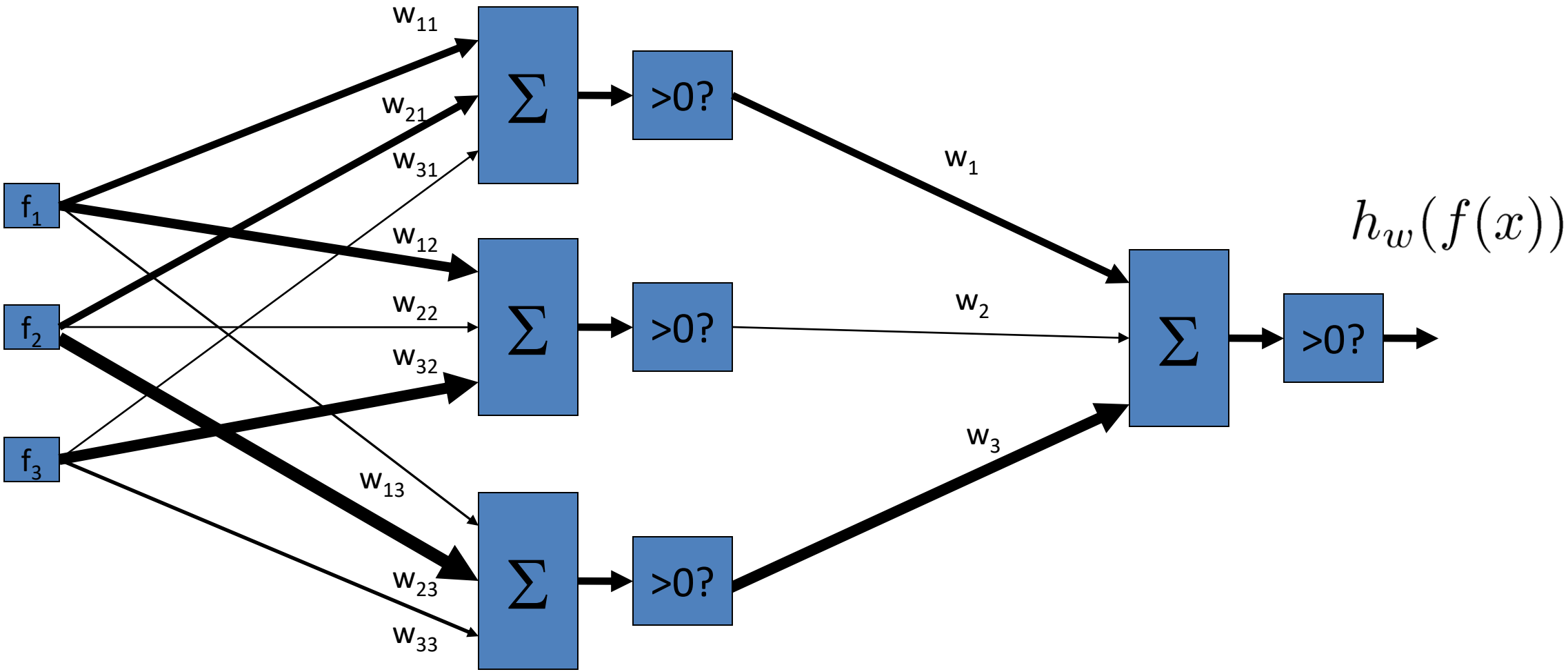


Neuronska Mreža

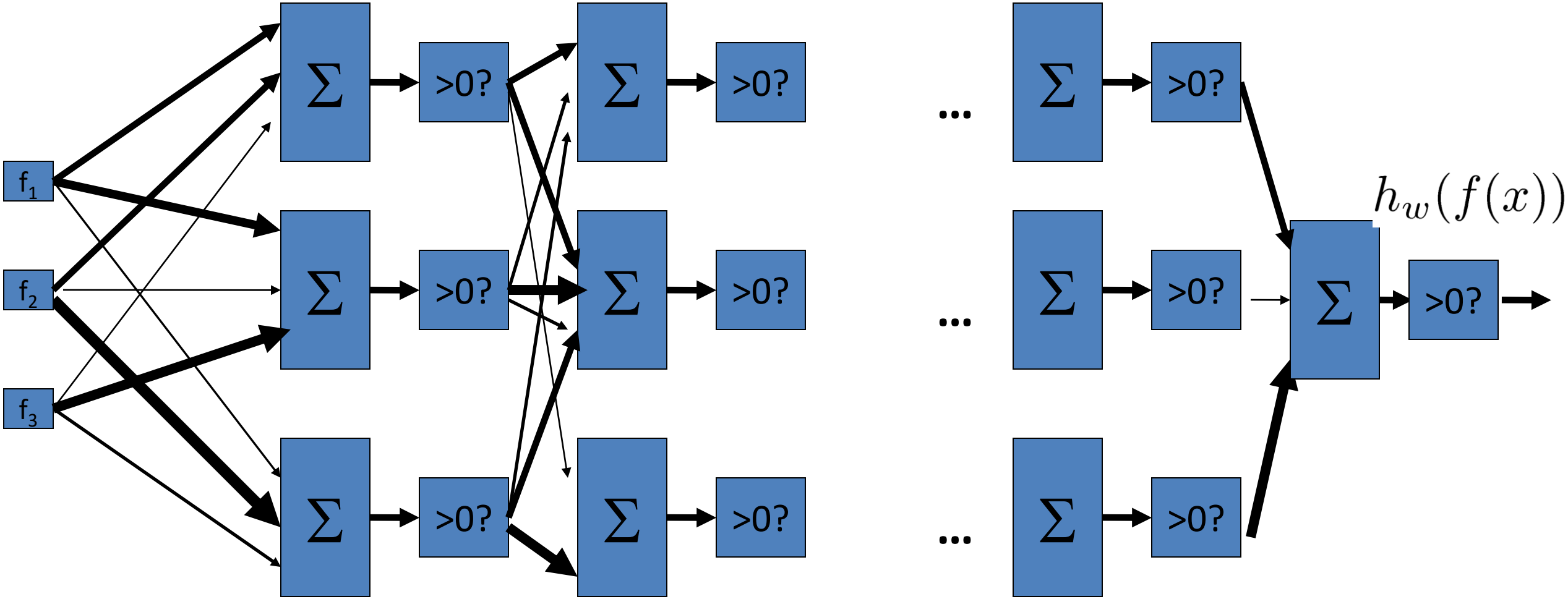
- Višeslojni perceptron je onda Neuronska Mreža!



Dvoslojni Perceptron



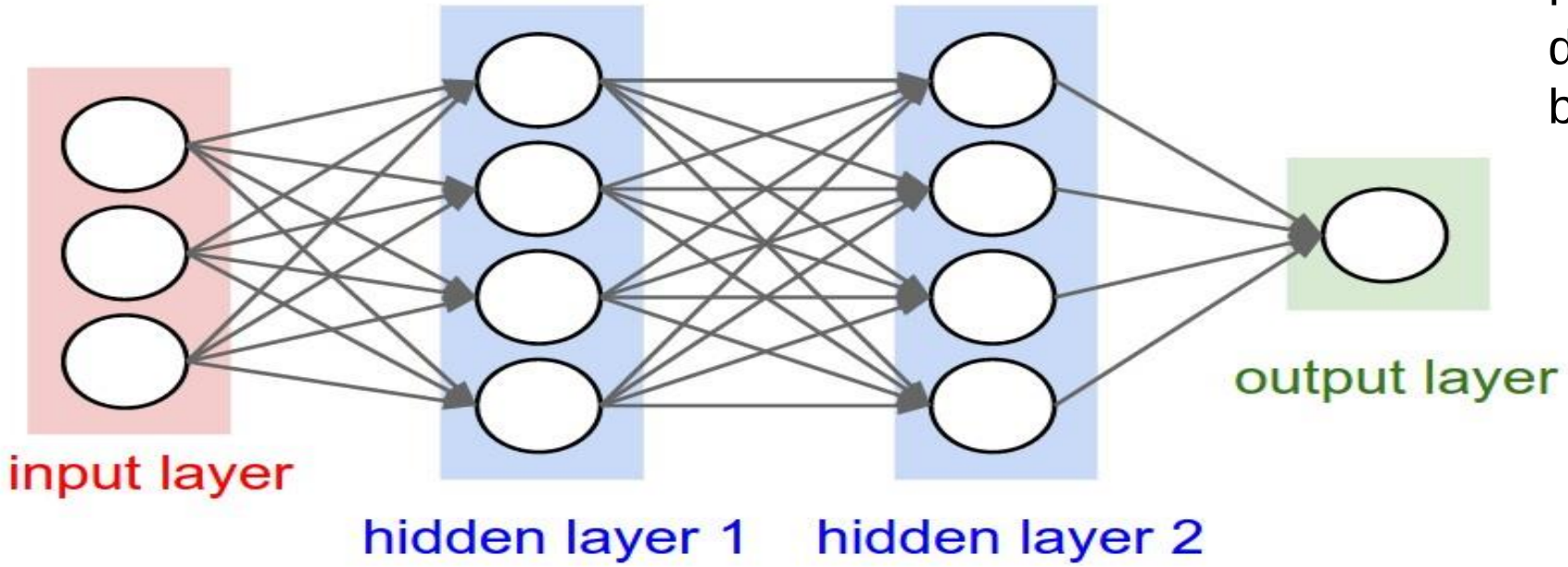
N-Slojni Perceptron



Neuronska Mreža - Organizacija

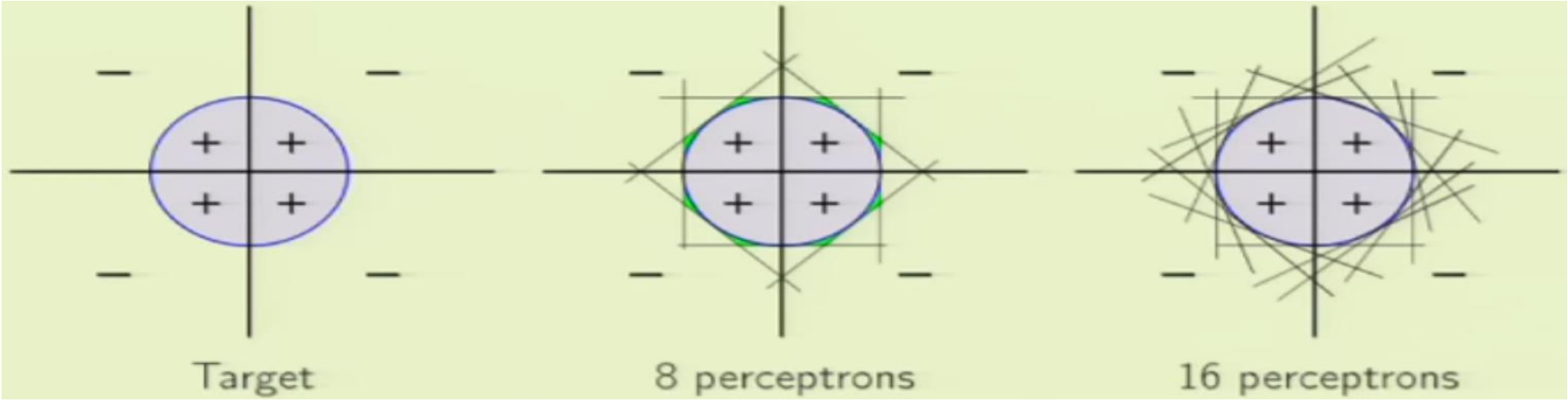
- Sloj obuhvata neurone i grane koje ulaze u njih
- Ovo je struktura sa propagacijom unapred (*feedforward*)
- Izlazi neurona su povezani isključivo na neurone sledećeg sloja
- Ovo nije jako restriktivno: ako možemo da implementiramo AND, OR i XOR možemo da implementiramo bilo šta!
- Dakle, možemo imati veoma sofisticirane granice odluke – samo nam treba dovoljno neurona koje ćemo kombinovati

Neuronska mreže sa tri sloja: dva skrivena i jedan izlazni (ne brojimo ulazni sloj)



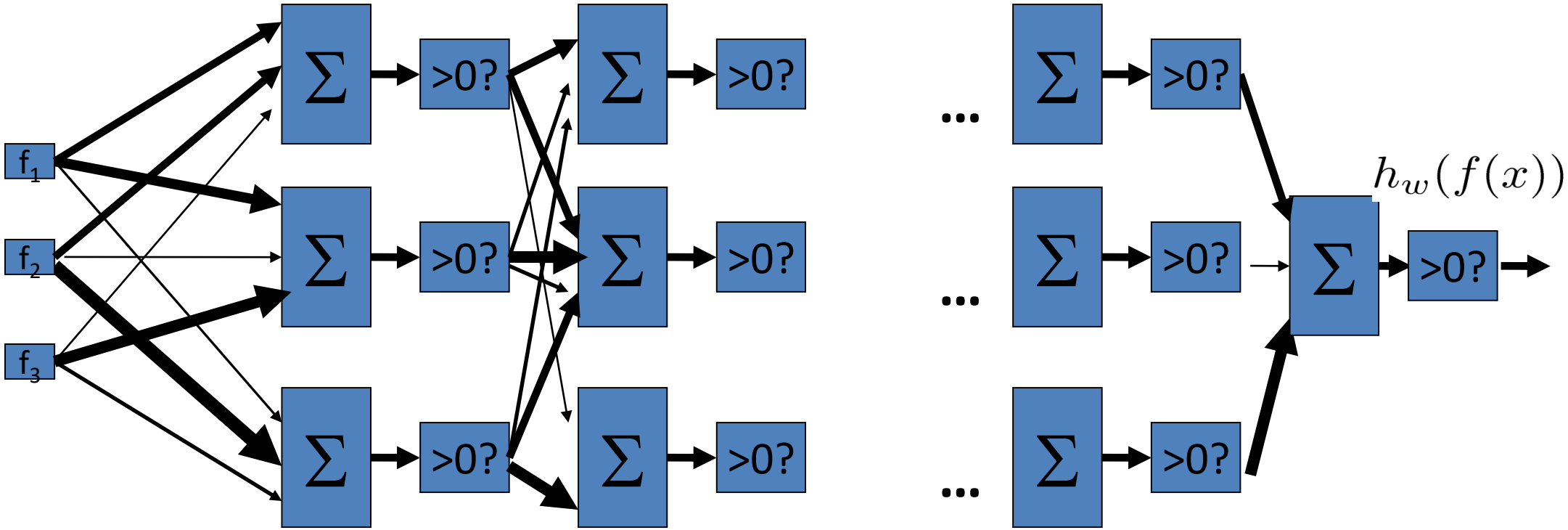
Neuronska Mreža – Moćan Model

- Možemo dokazati da MLP (Multy Layer Perceptron) sa dovoljno neurona može da aproksimira bilo koju funkciju
- Imamo izuzetno sofisticiran model, što je dobro, ali...
- Problem generalizacije – moramo pažljivo da prilagodimo sofisticiranost modela (arhitekturu – broj slojeva i neurona u svakom sloju) podacima kako ne bi došlo do overfittinga
- Problem optimizacije – velika količina parametara (težina) koja treba da se optimizuje, a treba nam optimalna kombinacija svih tih težina! Ovo je veoma težak optimizacioni problem.



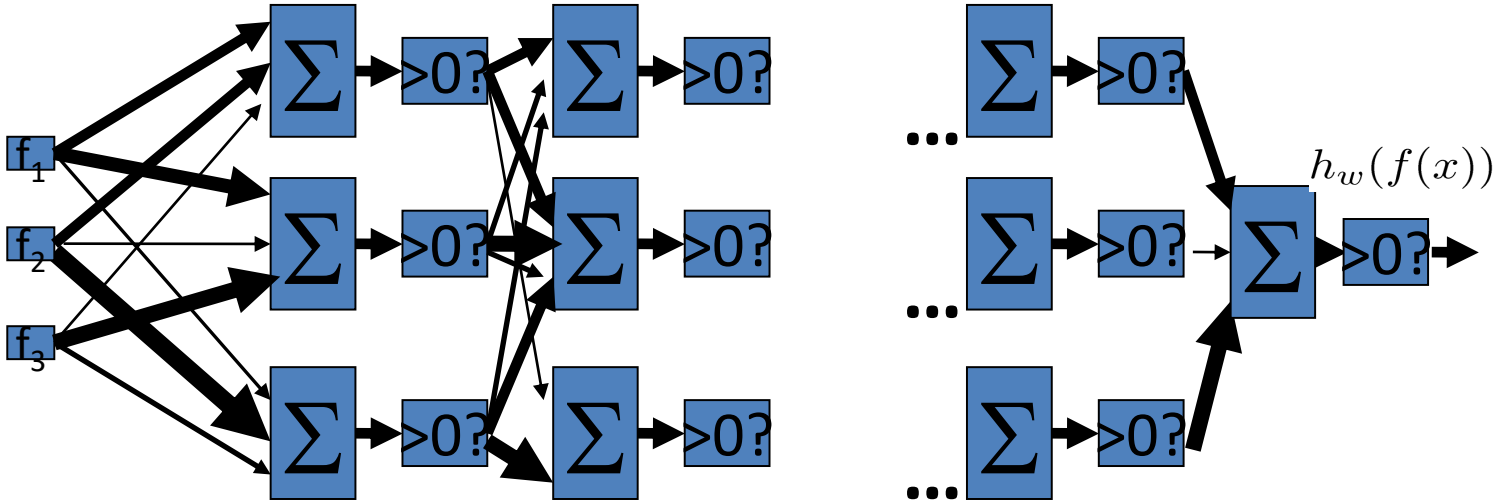
Kako obučiti Neuronsku Mrežu?

- Setimo se na koji smo način obučavali linearne klasfikatore.
- Koristili smo Gradijentni Spust. Da li ga možemo korisiti i sad?
- Možemo, ali postoji nekoliko ozbiljnih problema koje prvo moramo da rešimo.



Kako obučiti Neuronsku Mrežu?

- Problem 1: Funkcija greške. Višeslojni perceptron ima funkciju greške koja nije diferencijabilna i kod koje se izlaz naglo menja sa malom promenom ulaza.



$$l^{\text{acc}}(w) = \frac{1}{m}\sum_{i=1}^{m}\left(\text{sign}(w^\top f(x^{(i)})) == y^{(i)}\right)$$

Soft-Max Funkcija Greške

- Jedna alternativa za funkciju greške je softmax funkcija koju smo već pominjali
- Ako imamo dve klase:

- Skor za klasu y=1: $w^\top f(x)$ Skor za klasu y=-1: $-w^\top f(x)$
- Verovatnoće za klase:

$$p(y=1|f(x);w) = \frac{e^{w^\top f(x^{(i)})}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

$$p(y=-1|f(x);w) = \frac{e^{-w^\top f(x)}}{e^{w^\top f(x)} + e^{-w^\top f(x)}}$$

Soft-Max Funkcija Greške

- Ukupna verovatnoća za jedan vektor težina je:

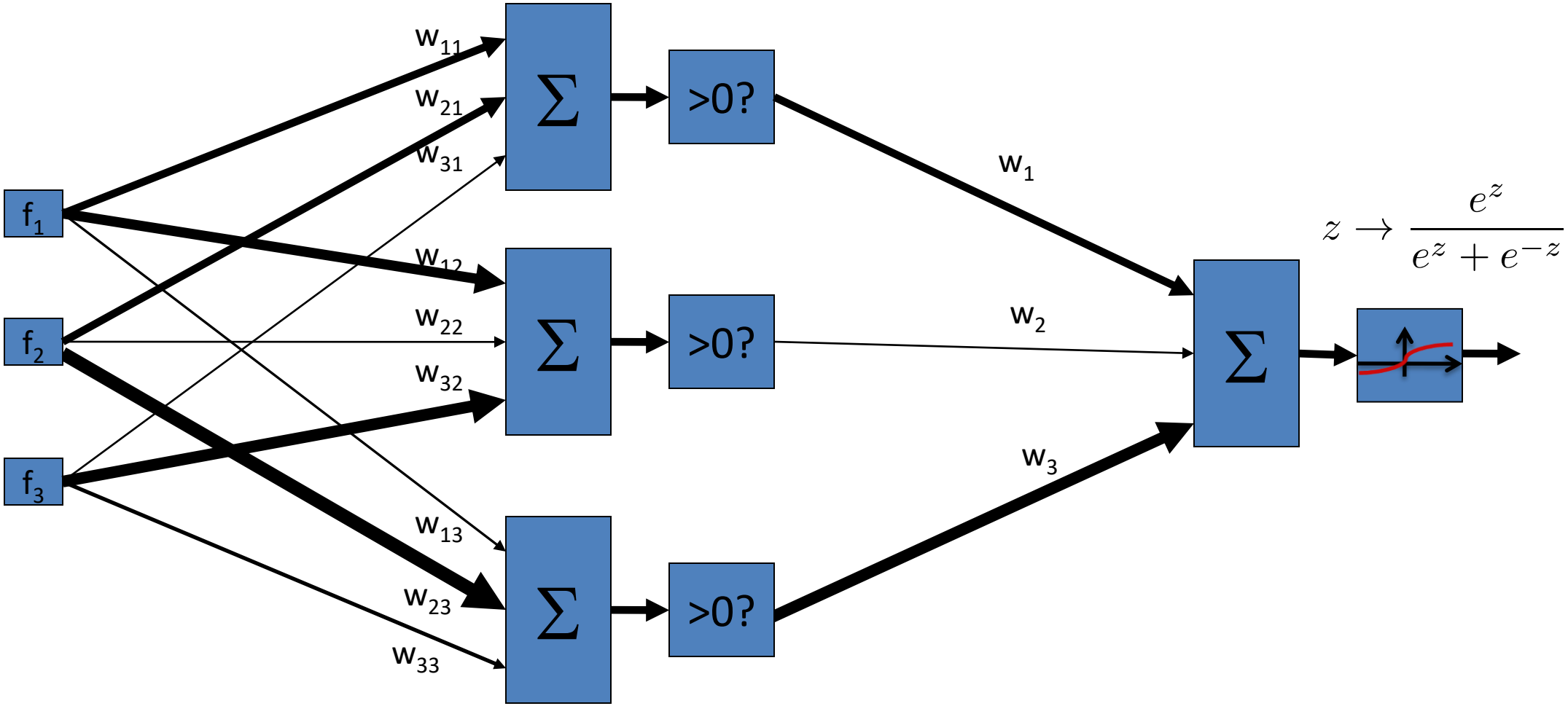
$$l(w) = \prod_{i=1}^{m} p(y = y^{(i)} | f(x^{(i)}); w)$$

- Dakle gradijentim spustom tražimo w takvo da je l(w) maksimalno
- U praksi se koristi logaritam od l(w) umesto l(w):

$$ll(w) = \sum_{i=1}^{m} \log p(y = y^{(i)} | f(x^{(i)}); w)$$

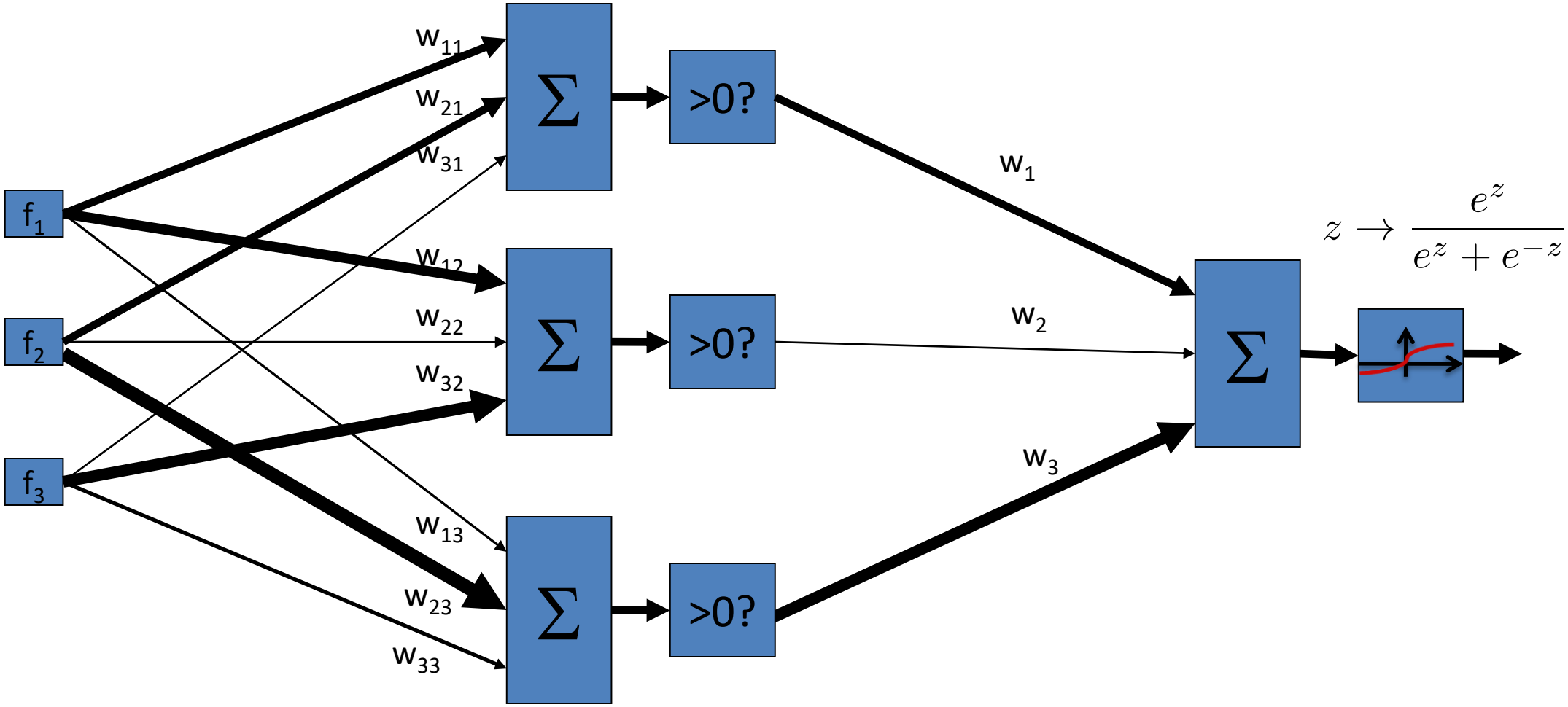
Dvoslojna Neuronska Mreža

- Sad imamo funkciju greške. Drugi problem je kako odrediti gradijent težina.



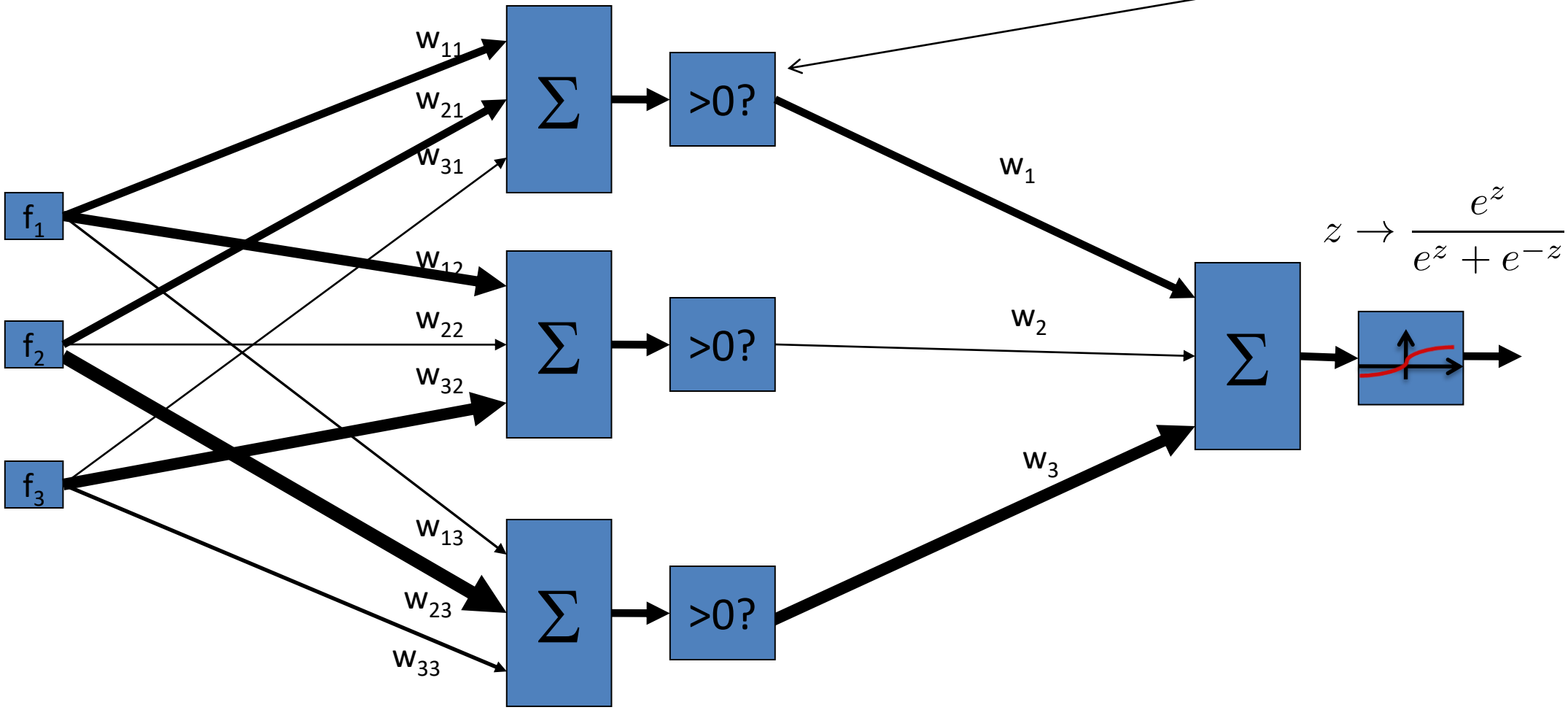
Dvoslojna Neuronska Mreža

- Lako je pomoću parcijalnih izvoda otkriti kako male promene w_1,w_2 i w_3 utiču na fukciju greške. Da li je to lako i za npr. w_{21}? Ne baš.



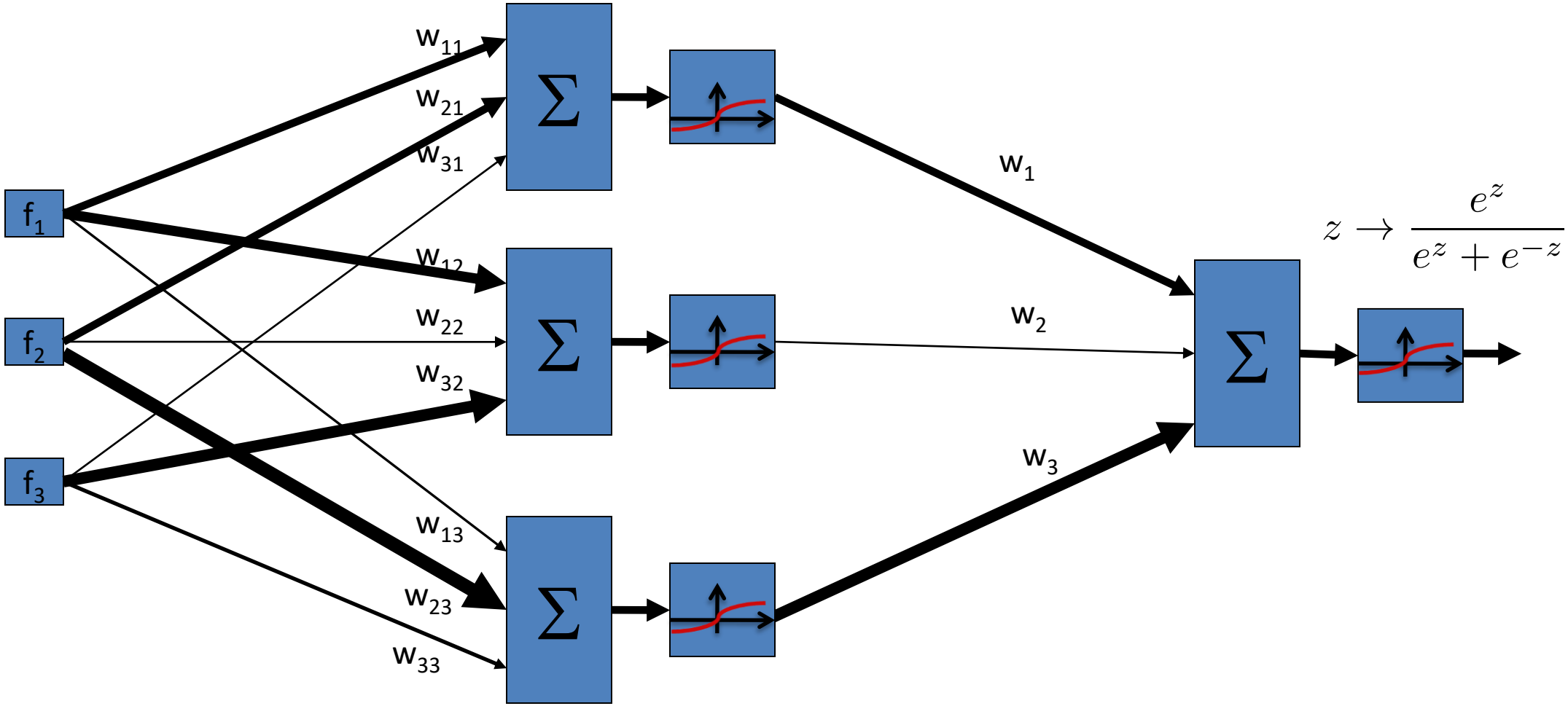
Dvoslojna Neuronska Mreža

- Da li je to lako i za npr. w_{21}? Ne baš.
- Prvi problem pri određivanju parcijalnih izvoda za w_{21} je što je ta težina uključena u funkciju aktivacije koja nije diferencijabilna (sign funkcija).



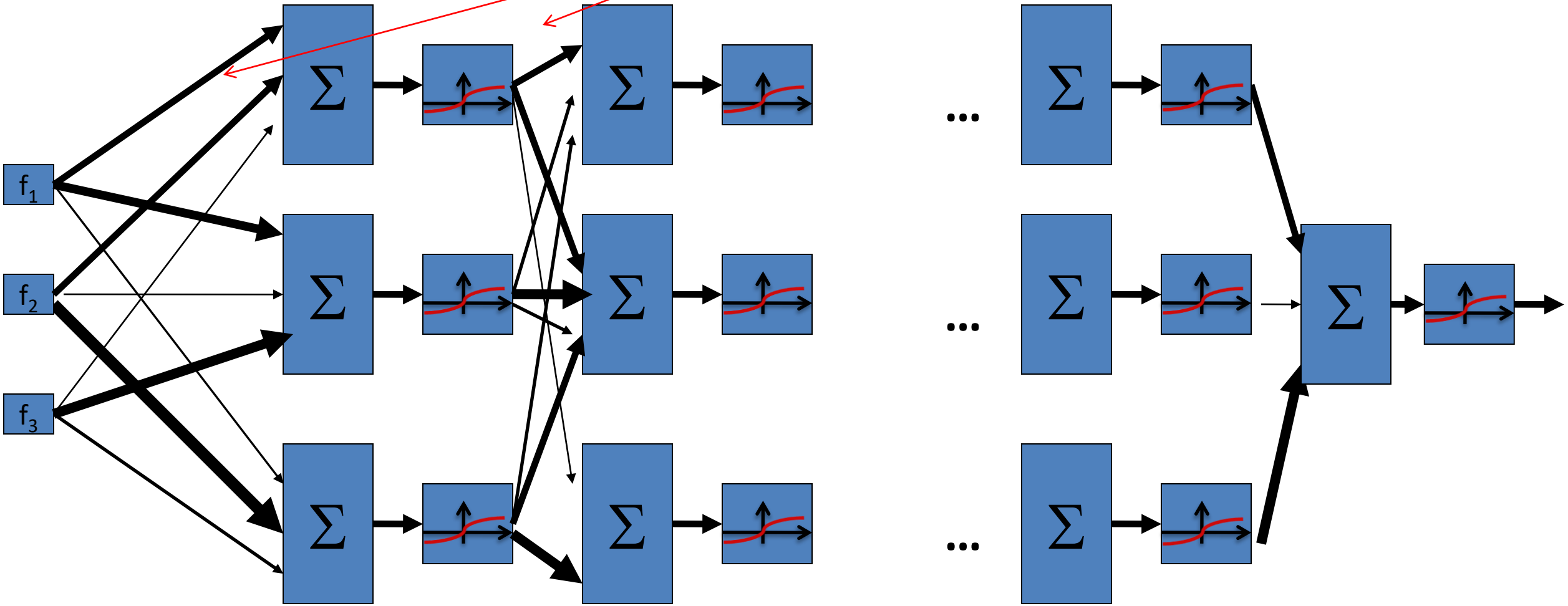
Dvoslojna Neuronska Mreža

- Prvi problem pri određivanju parcijalnih izvoda za w_{21} je što je ta težina uključena u funkciju aktivacije koja nije diferencijabilna (sign funkcija).
- Problem rešavamo uvođenjem diferencijabilnih funkcija aktivacije – više detalja na naredim predavanjima.



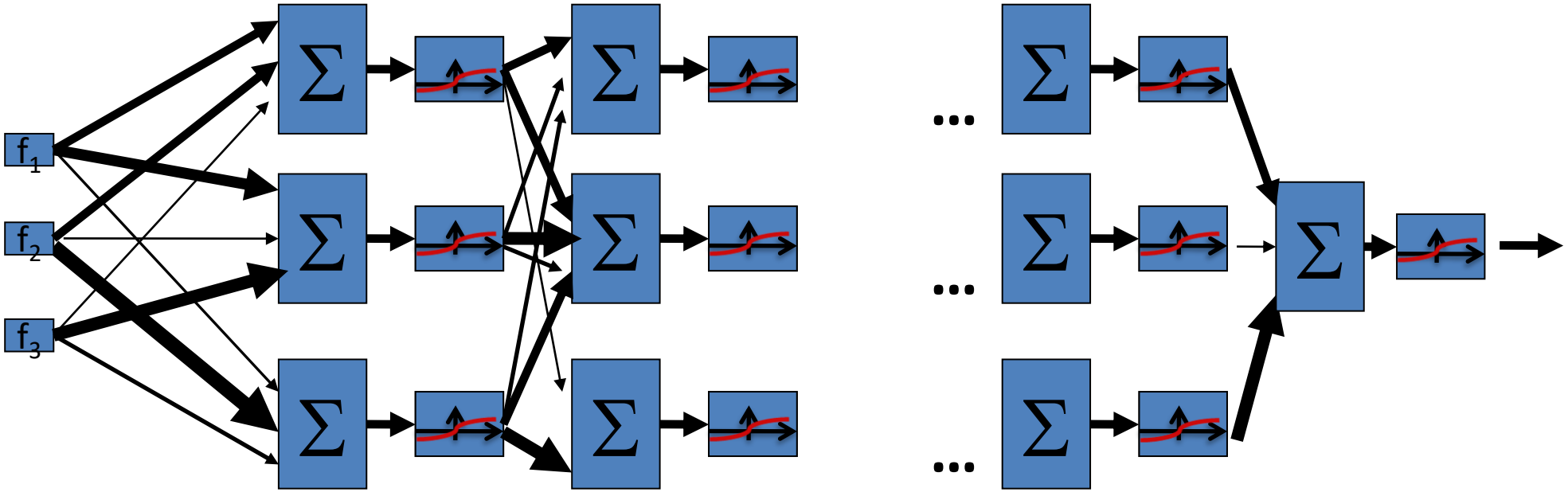
N-slojna Neuronska Mreža

- Da li je sad lako odrediti uticaj neke od ovih težina?



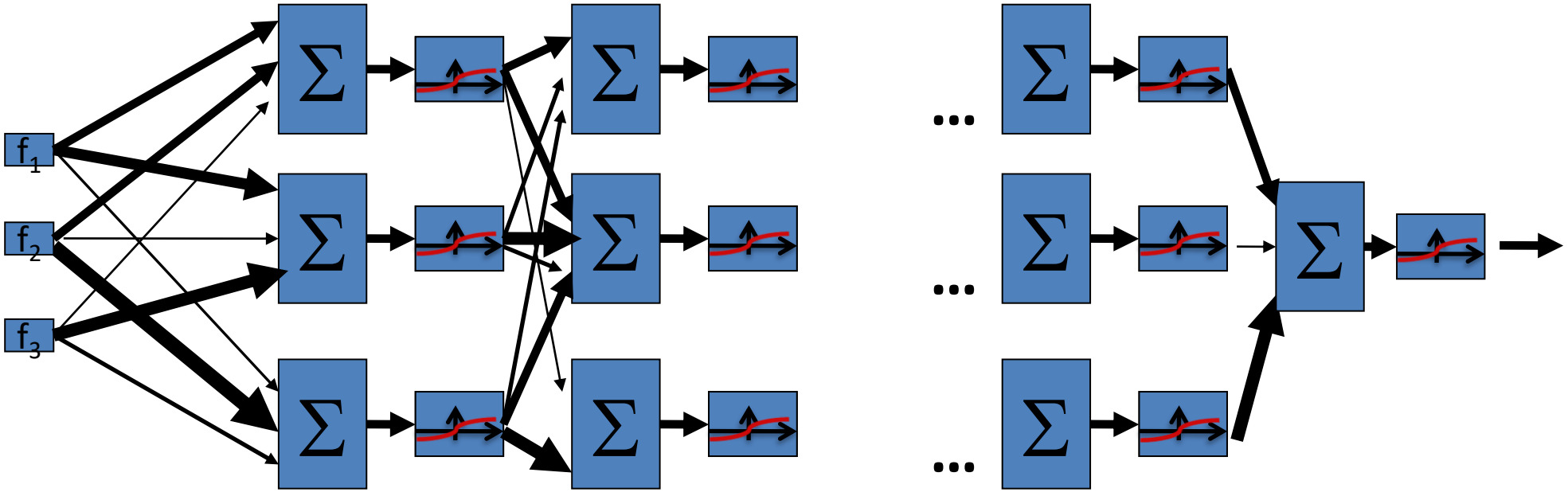
N-slojna Neuronska Mreža

- Da li je sad lako odrediti uticaj neke od ovih težina?
- Definitivno nije lako. Treba nam parcijalni izvod funkcije greške koja je na kraju, ali po težini koja je N slojeva ispod, gde N može biti 150 npr.
- Nećemo ga tražiti klasično, pomoću papira i olovke (iako je i to moguće).
- Posmatraćemo Neuronsku Mrežu kao Graf Izračunavanja i videti koliko je lako odrediti uticaj „dalekih težina“. O tome na sledećoj prezentaciji.



N-slojna Neuronska Mreža

- Pre nego što pređemo na sledeću prezentaciju, jedna napomena.
- Algoritam pomoću koga se izračunavaju uticaji (formalnije gradijenti) svih težina u mreži zove se:
  - Probagacija Unazad ili *Backpropagation*
- To je fundamentalni algoritam za obučavajnje neuronskih mreža i morate ga razumeti.
- Ako ga ne razumete, nećete moći da se izborite sa svim situacijama do koji može doći kad obučavate npr. konvolutivnu mrežu od 150 slojeva.



Osnovi Računarske Inteligencije

Neuronske Mreže i Algoritam Propagacije Unazad (*Backpropagation*)

Predavač: Aleksandar Kovačević

Slajdovi preuzeti sa CS 231n, Stanford

http://cs231n.stanford.edu/

Šta smo do sada naučili...

$$s = f(x; W) = Wx$$

skor funkcija

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

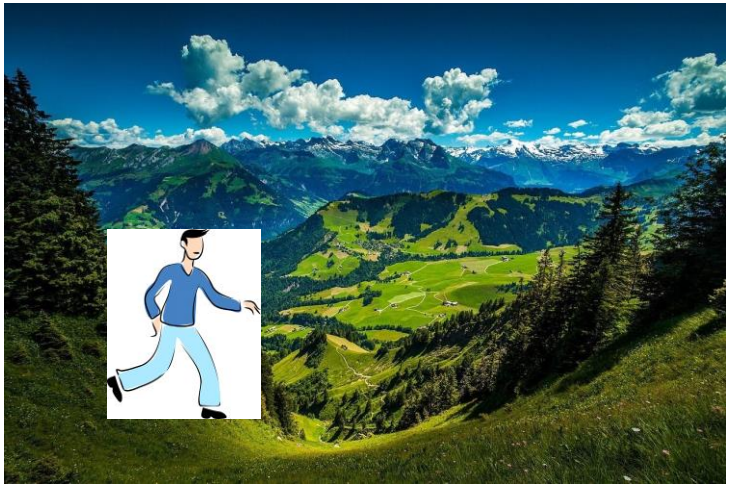
SVM funkcija greške

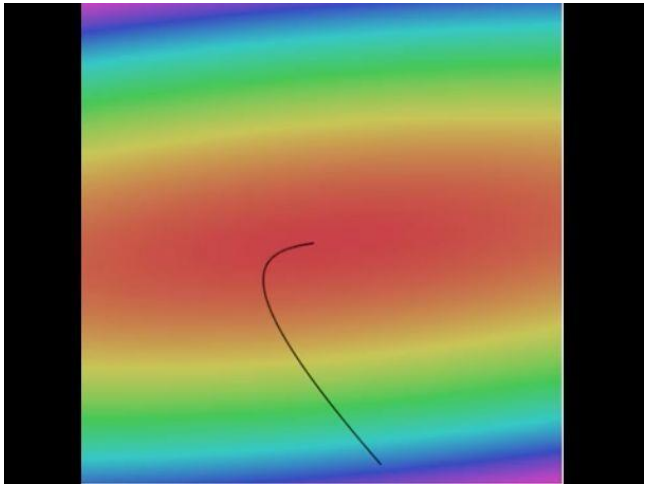
$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

greška + regularizacija

hoćemo $\boxed{\nabla_W L}$

Optimizacija





```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Landscape image is CC0 1.0 public domain
Walking man image is CC0 1.0 public domain

Gradijentni spust

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

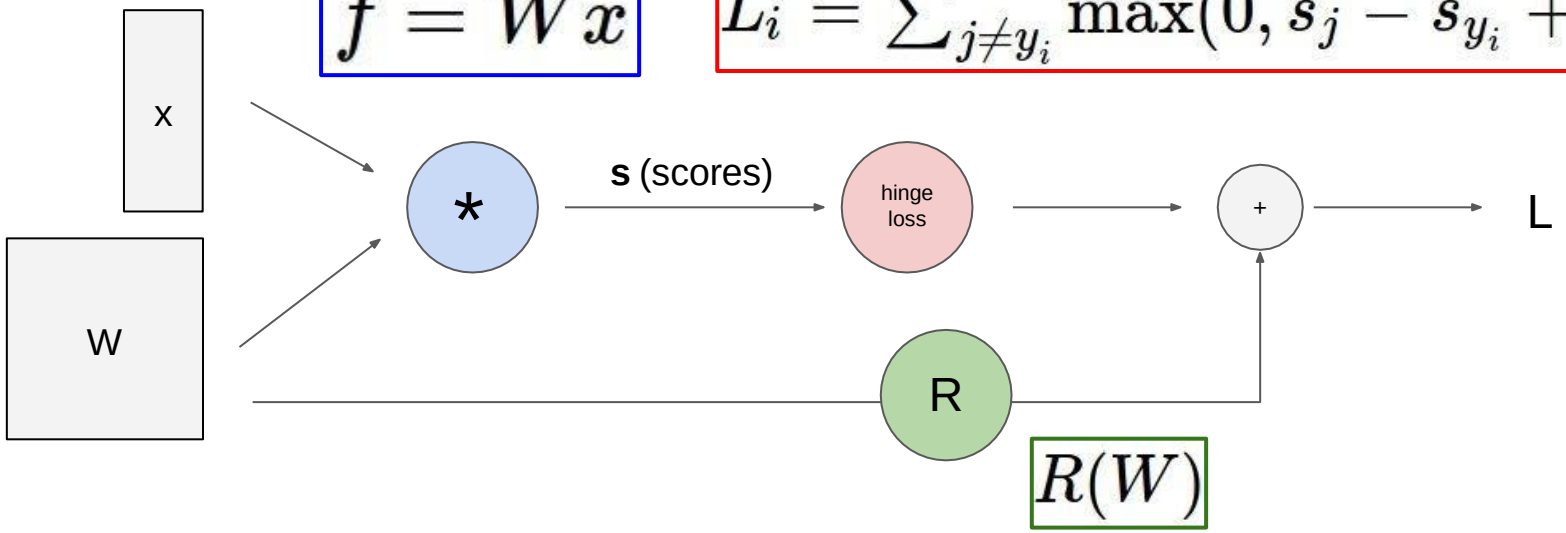
Numerički gradijent: spor :(, aproksimacija :(, lako se računa :)
Analitički gradijent: brz :), tačan :), teško se računa i moguće su greške :(

U praksi: Prvo simbolički izvedemo analitički gradijent, pa ga implemenitramo, a implementaciju proverimo pomoću numeričkog gradijenta

Grafovi Izračunavanja
Computational graphs

$$f = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



Konvolutivna Mreža (AlexNet)

ulazna slika

težine

greška

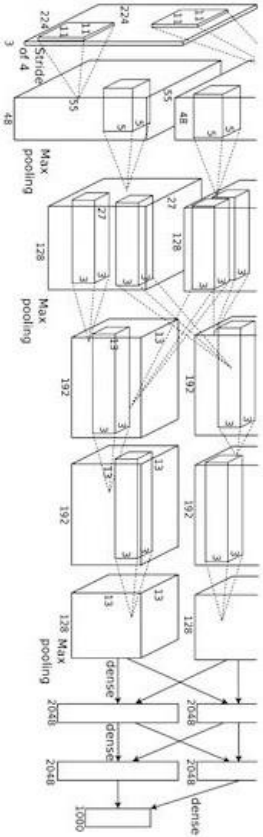
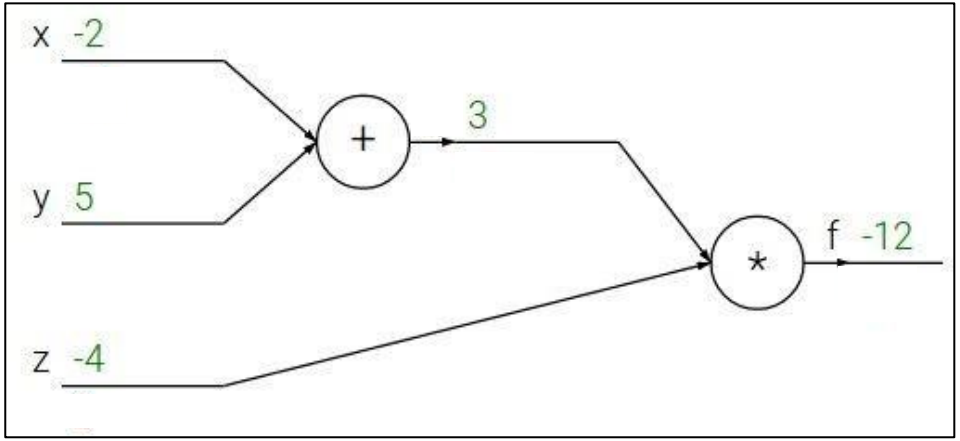


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Backpropagation: jednostavan primer

$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4



Backpropagation: jednostavan primer

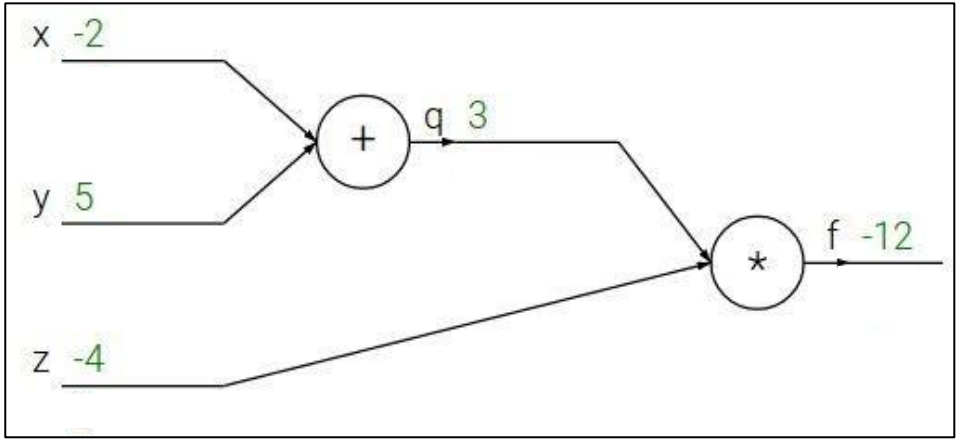
$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Tražimo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: jednostavan primer

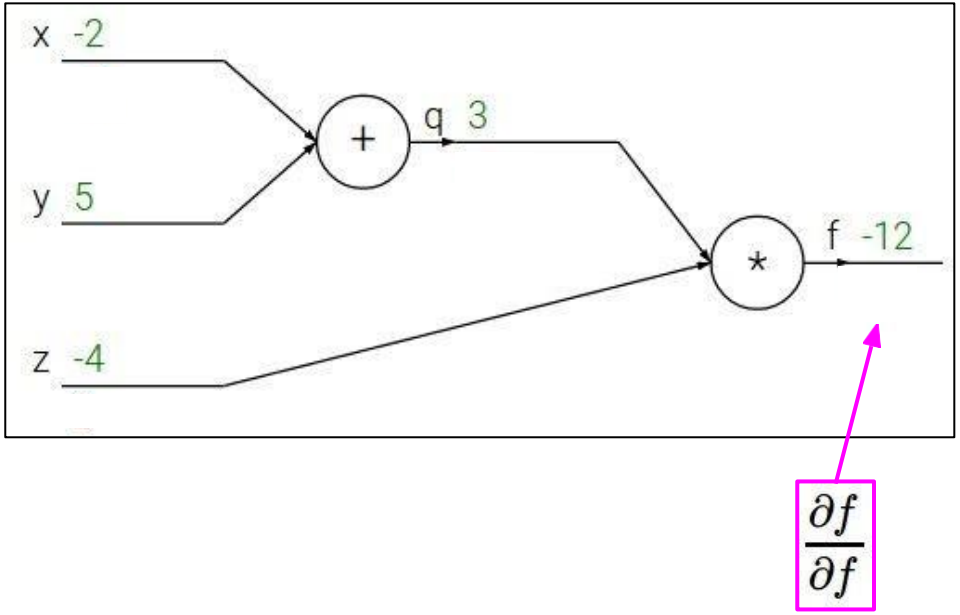
$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Tražimo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: jednostavan primer

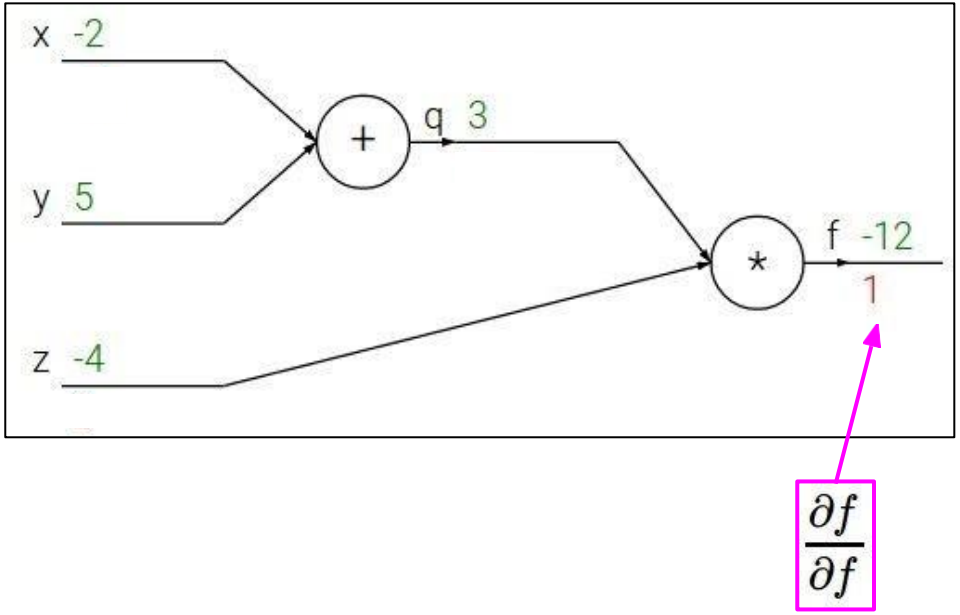
$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Tražimo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: jednostavan primer

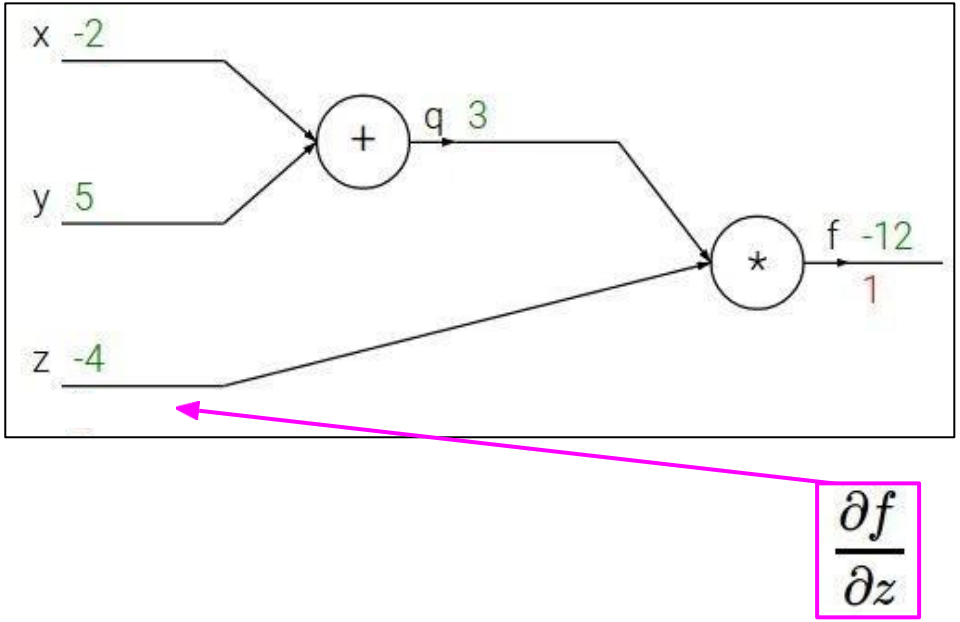
$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Tražimo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: jednostavan primer

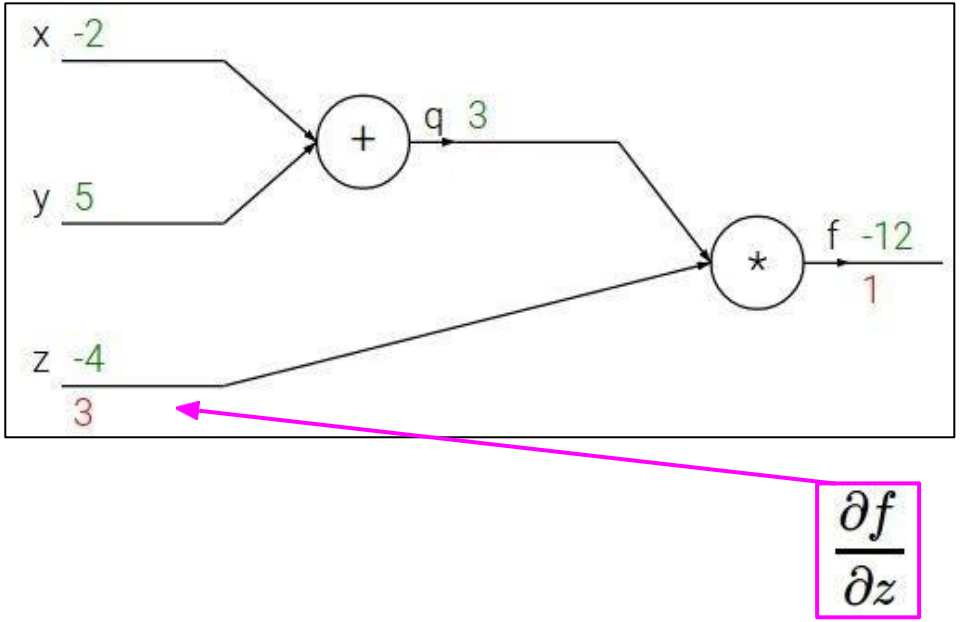
$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Tražimo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: jednostavan primer

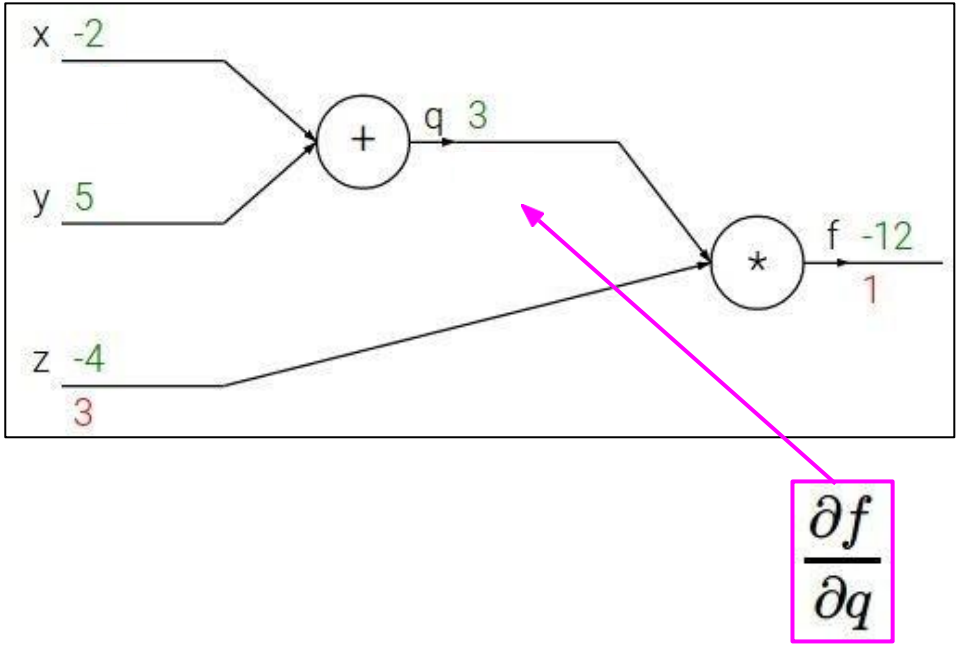
$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Tražimo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: jednostavan primer

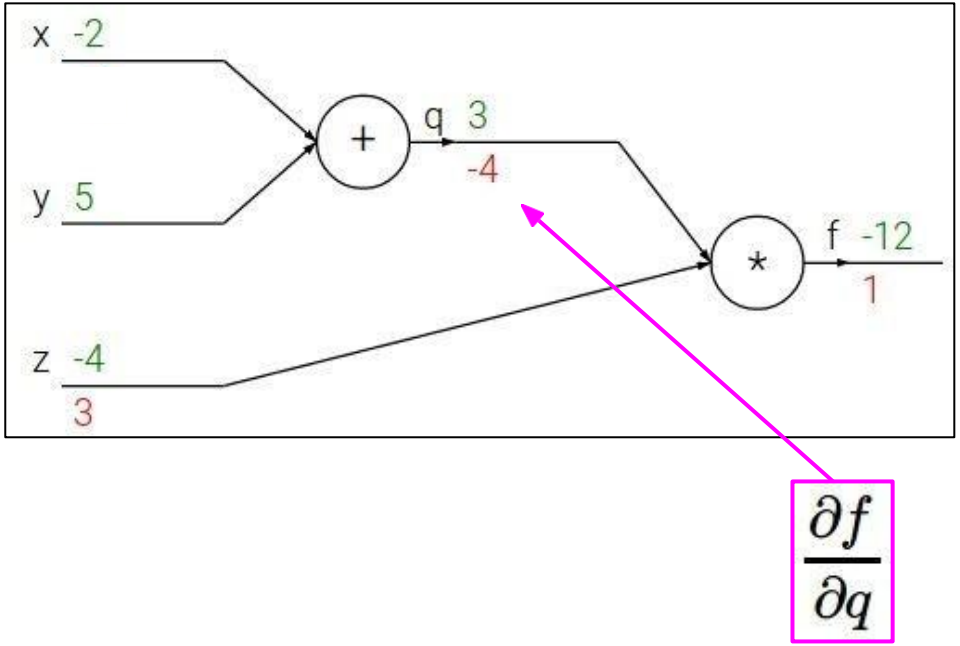
$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Tražimo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: jednostavan primer

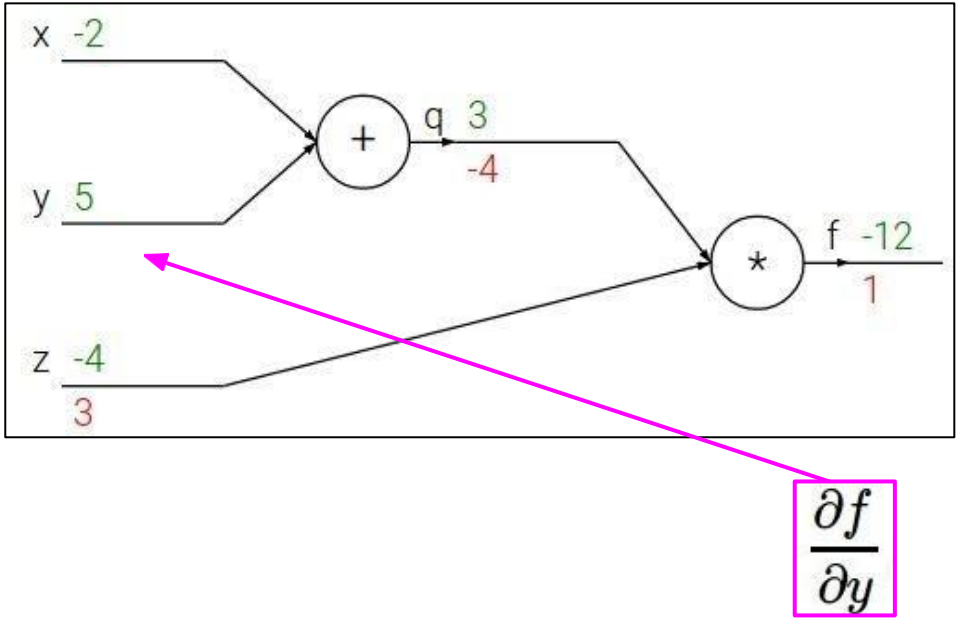
$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Tražimo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: jednostavan primer

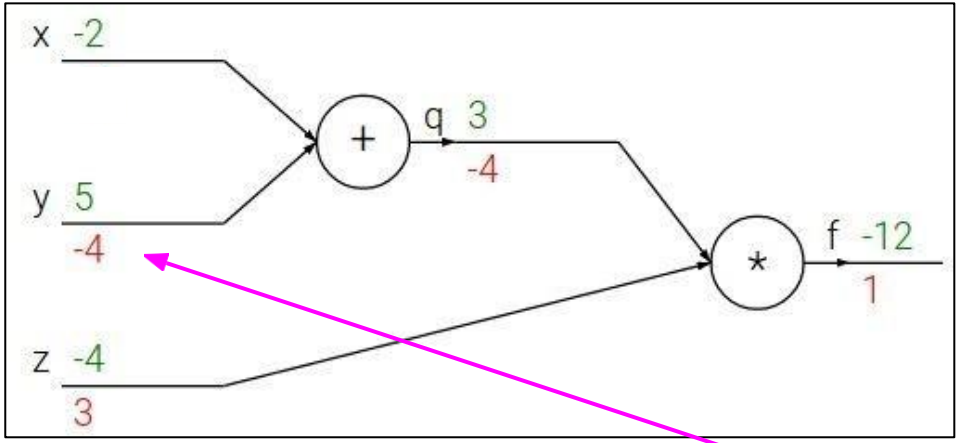
$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Tražimo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

Ulačavanje izvoda:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

izvod složene funkcije (*Chain rule*):

Backpropagation: jednostavan primer

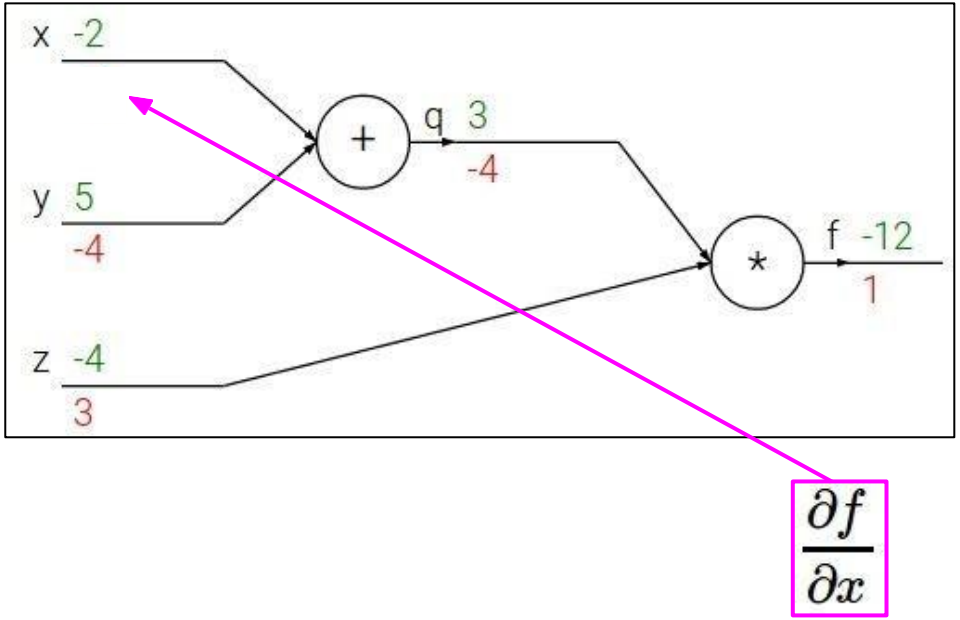
$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Tražimo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Backpropagation: jednostavan primer

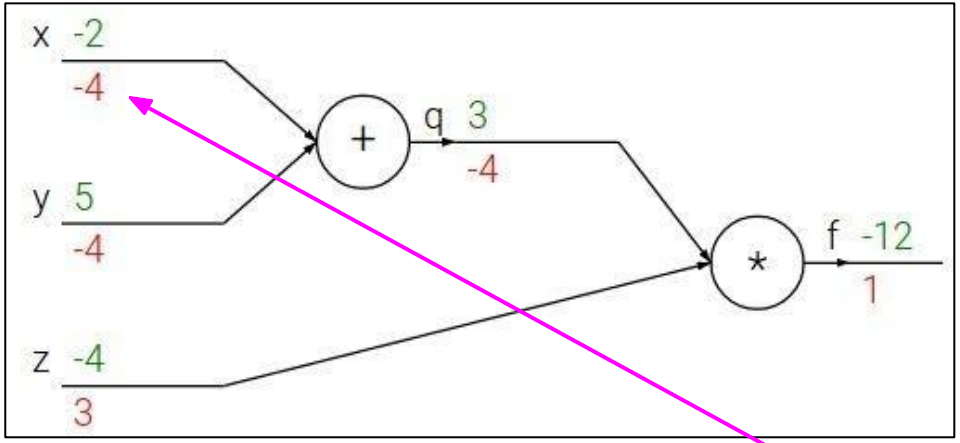
$$f(x, y, z) = (x + y)z$$

npr. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

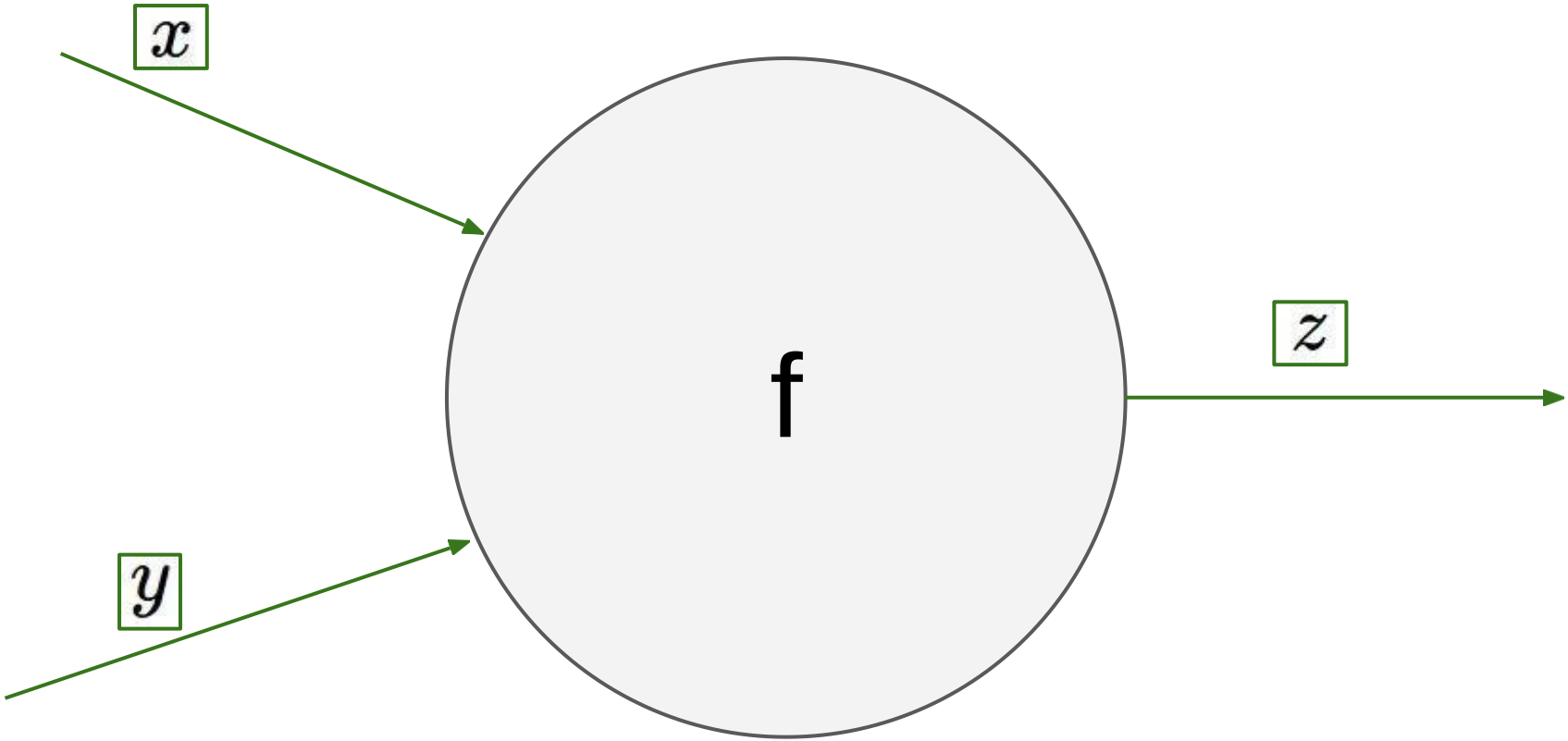
Tražimo: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

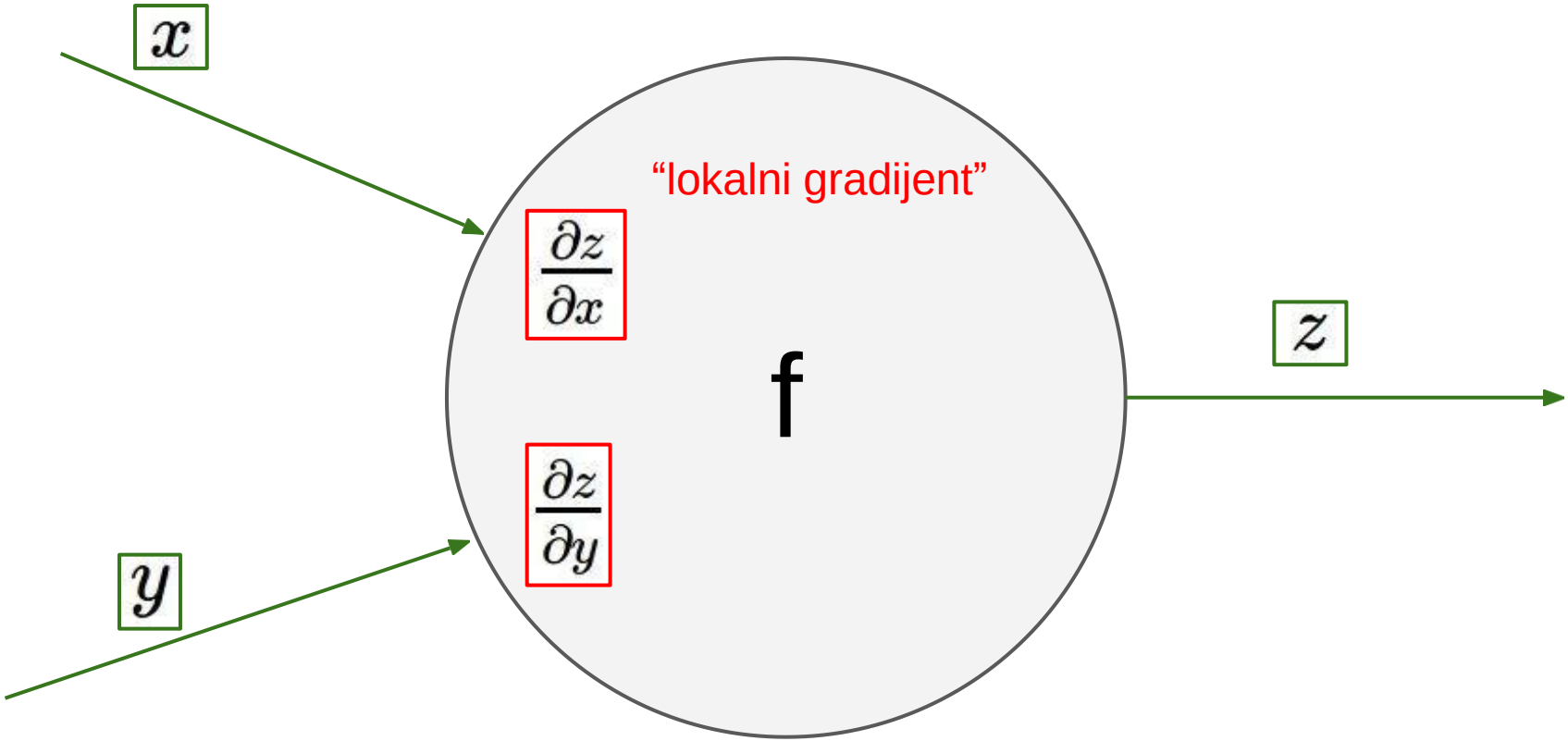


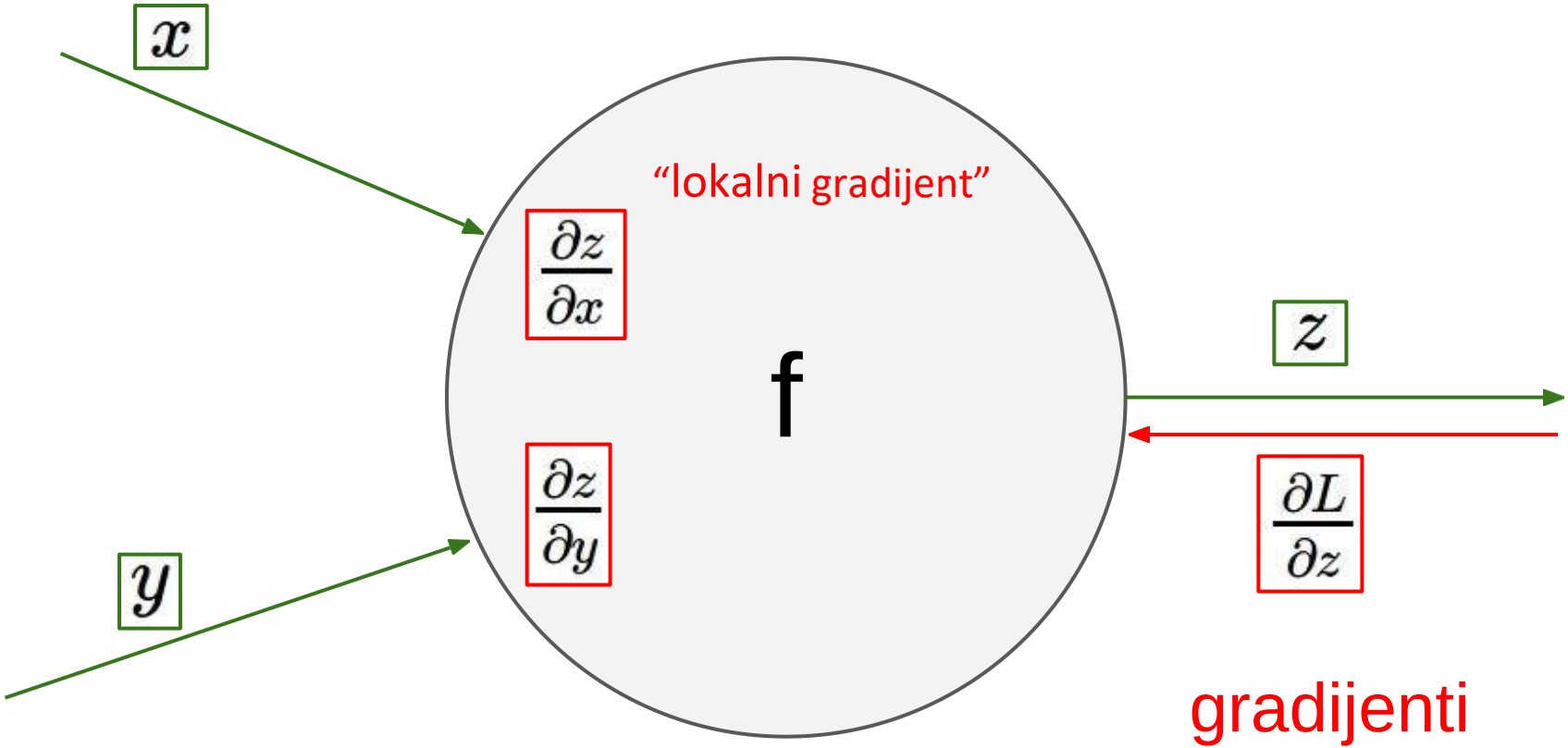
$$\frac{\partial f}{\partial x}$$

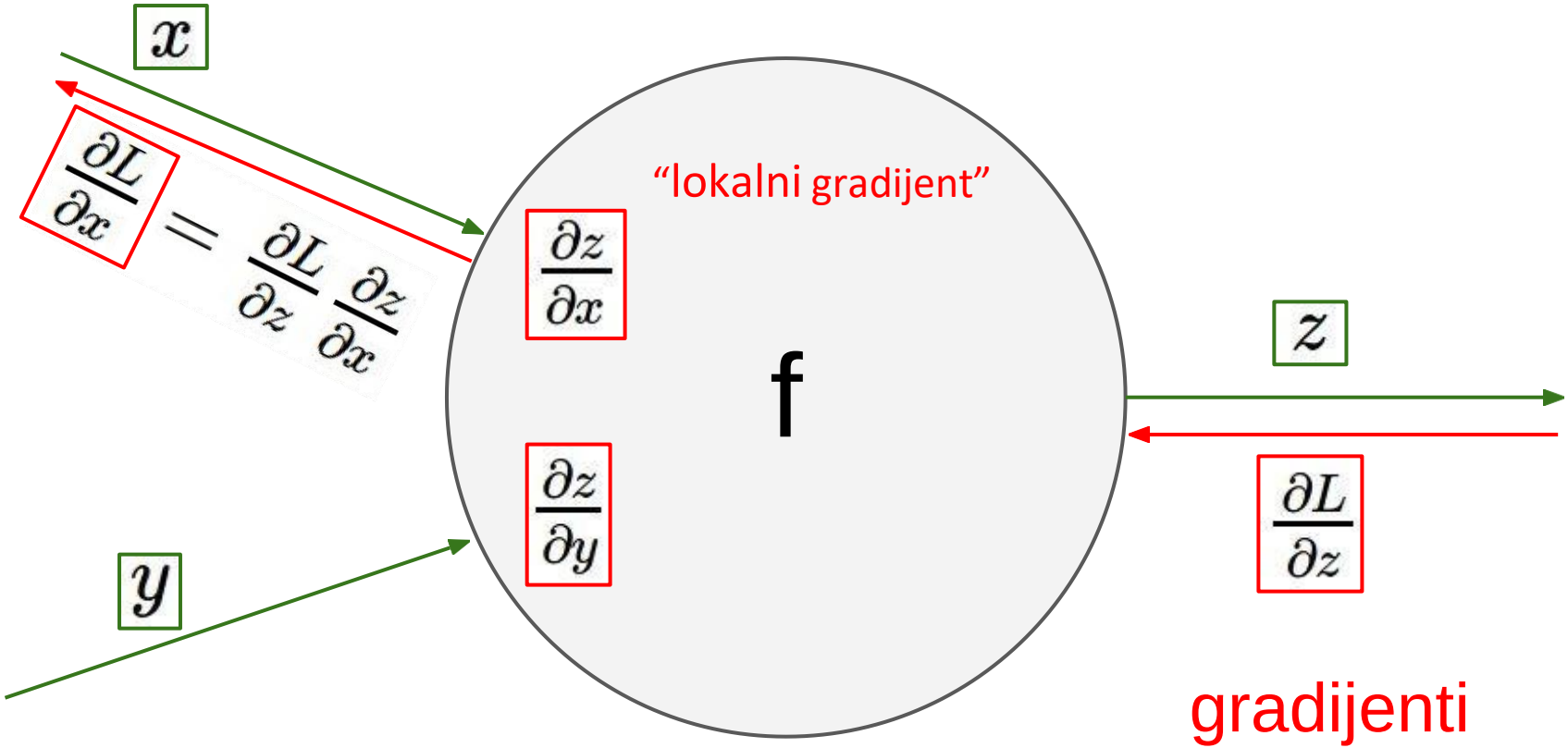
Ulačavanje izvoda:

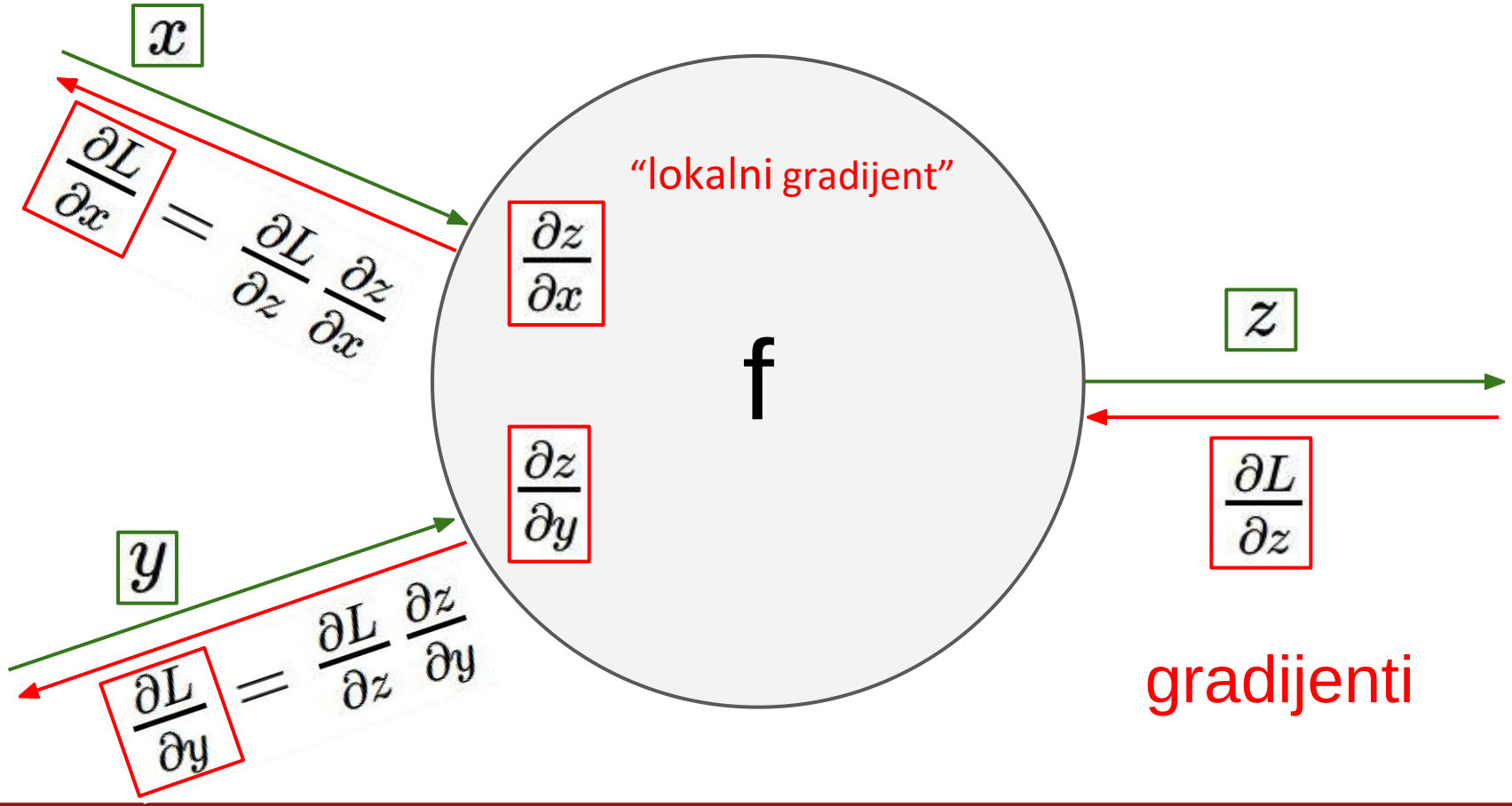
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

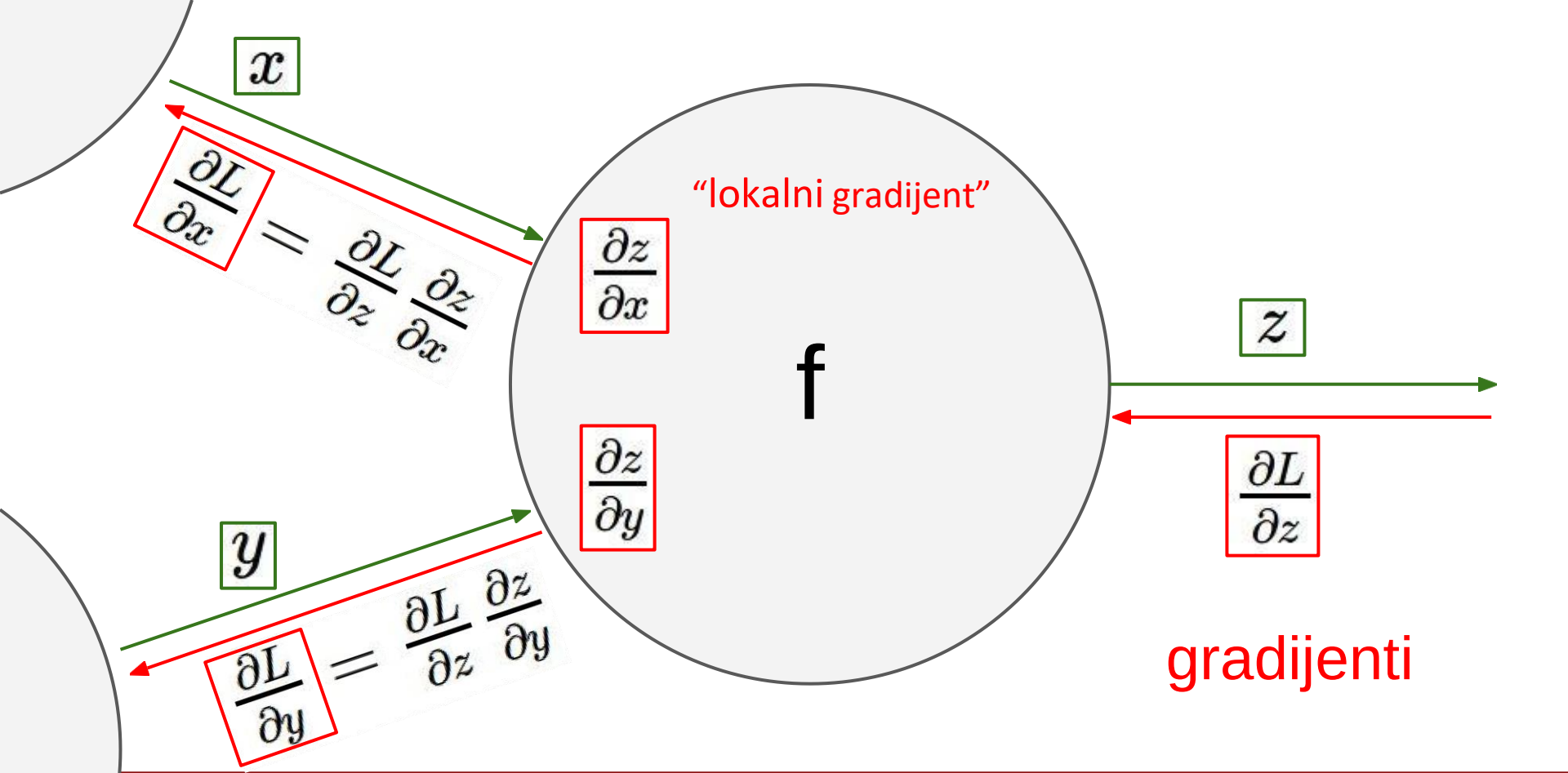

x
y
f
z


x
y
z
f
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
“lokalni gradijent”


x
y
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
"lokalni gradijent"
f
z
$\frac{\partial L}{\partial z}$
gradijenti

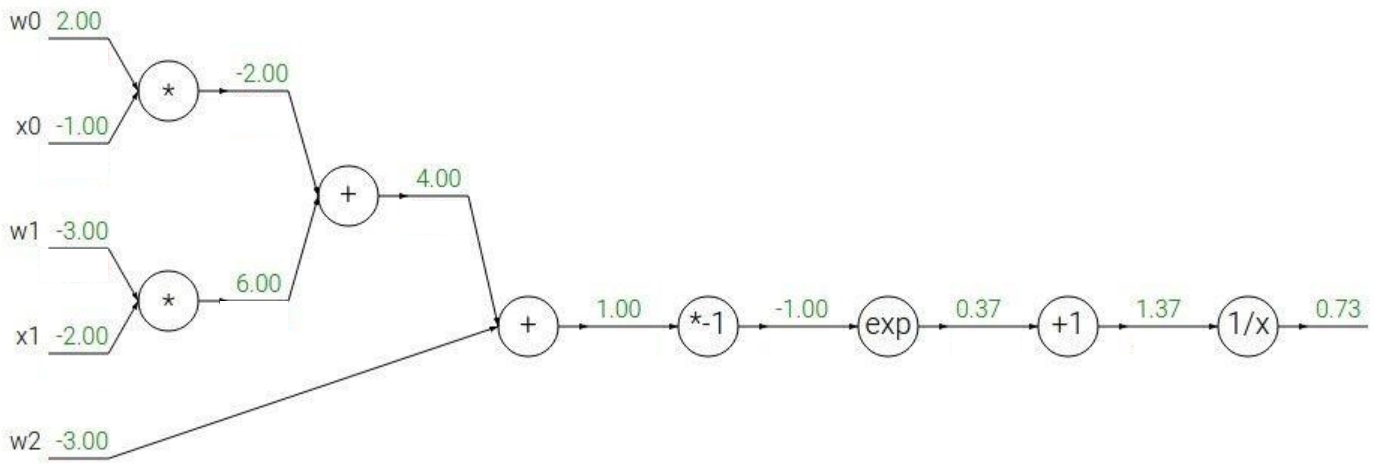

x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"lokalni gradijent"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
gradijenti


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"lokalni gradijent"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradijenti


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"lokalni gradijent"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradijenti

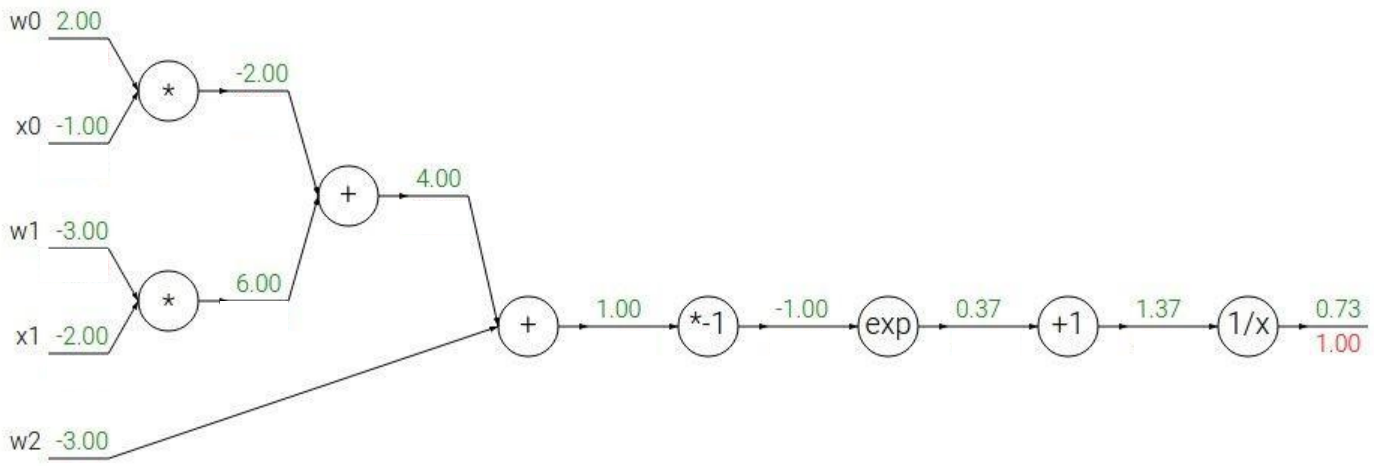
Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

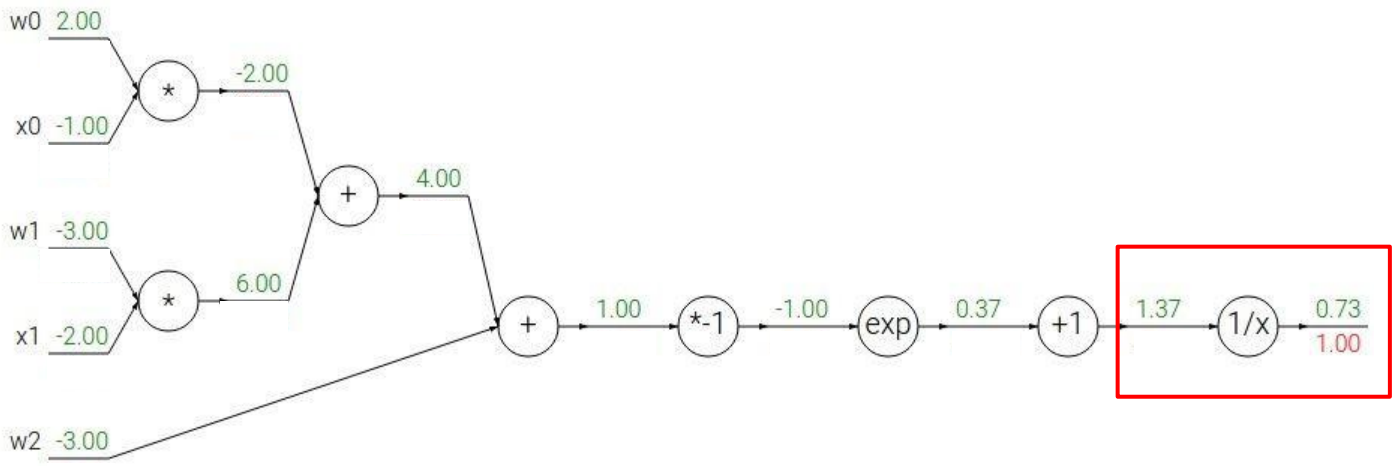
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

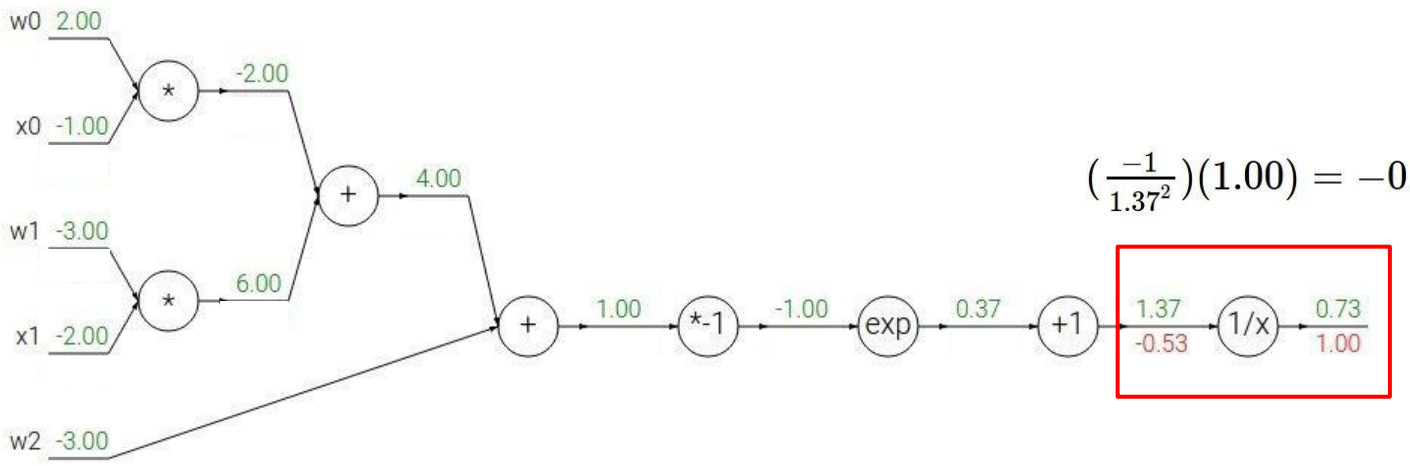
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$\left(\frac{-1}{1.37^2}\right)(1.00) = -0.53$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

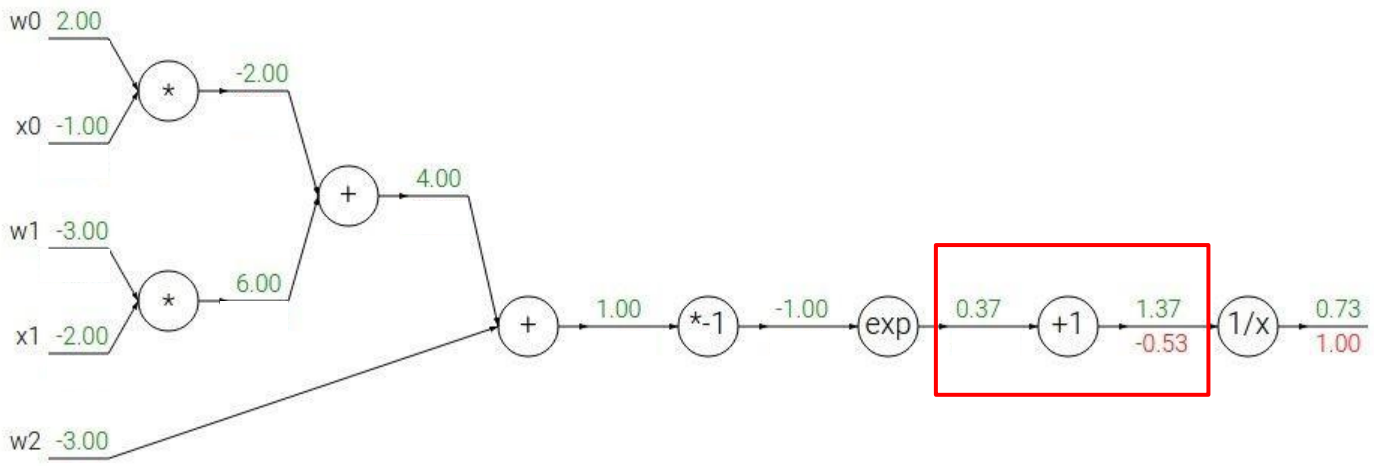
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

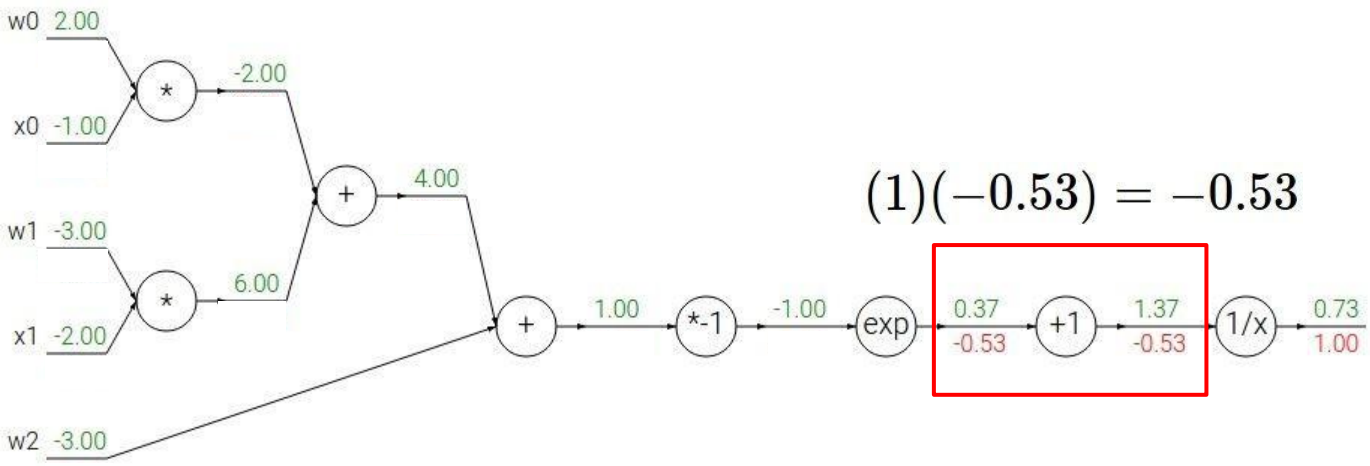
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$(1)(-0.53) = -0.53$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

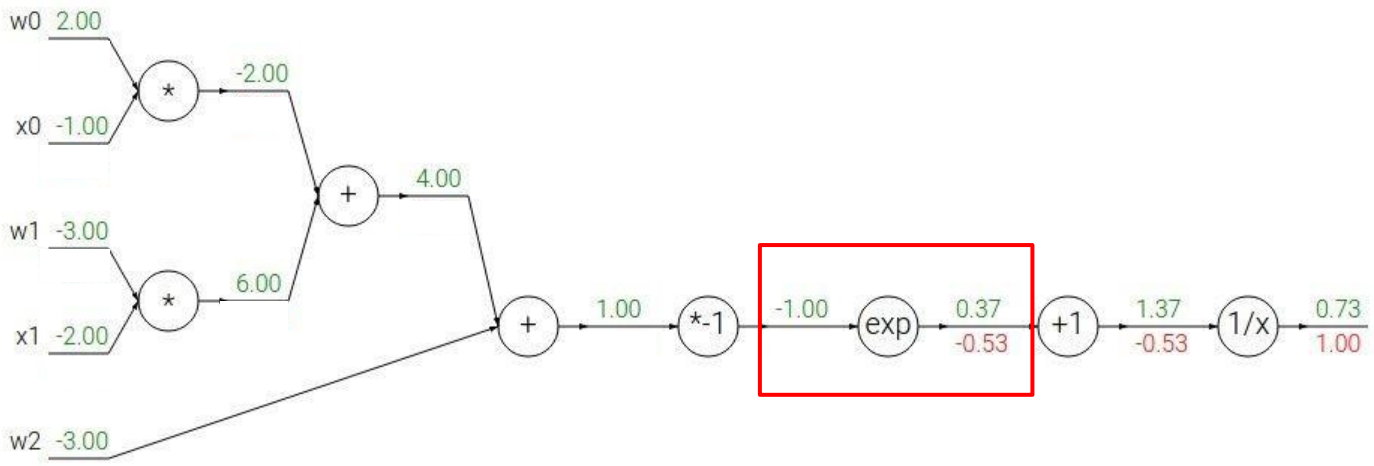
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

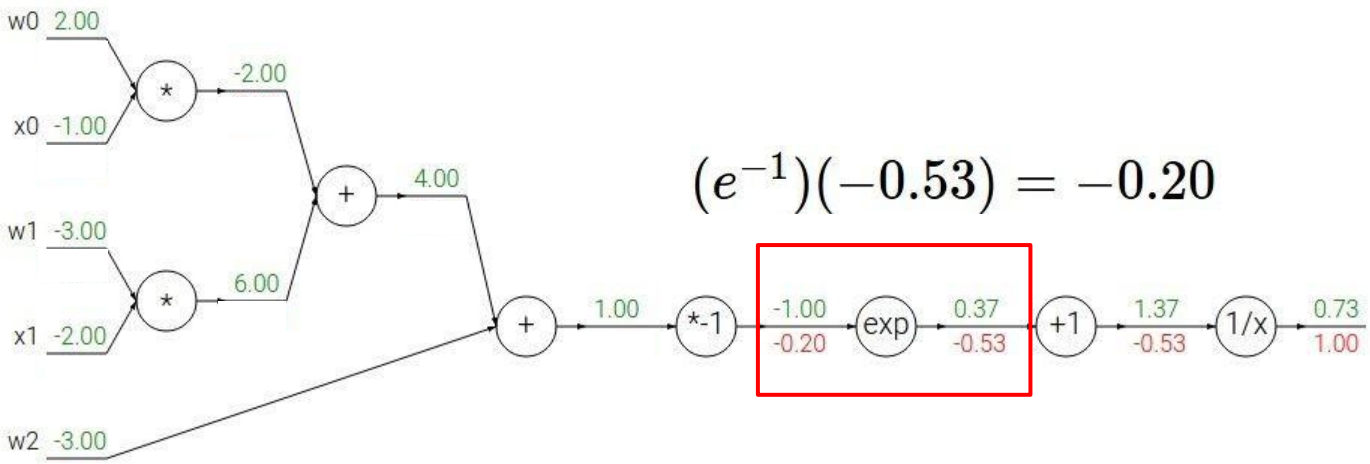
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$(e^{-1})(-0.53) = -0.20$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

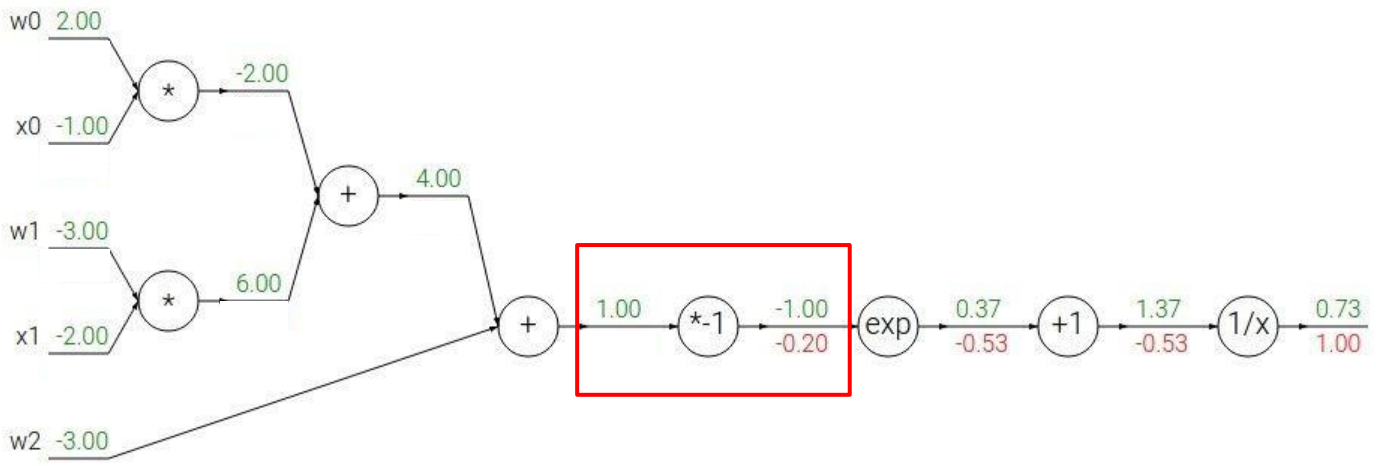
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

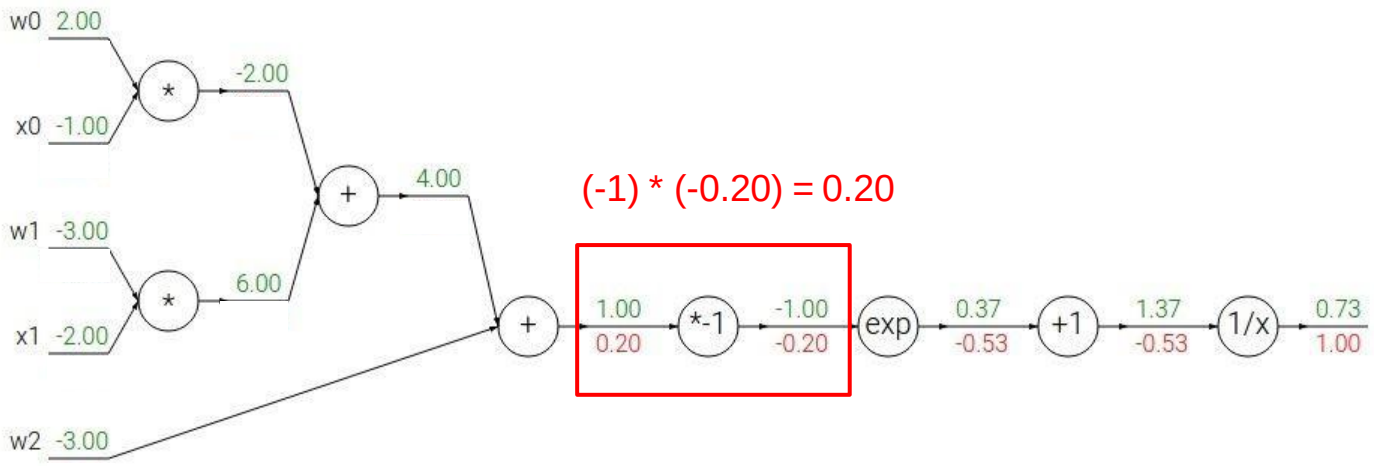
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



(-1) * (-0.20) = 0.20

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

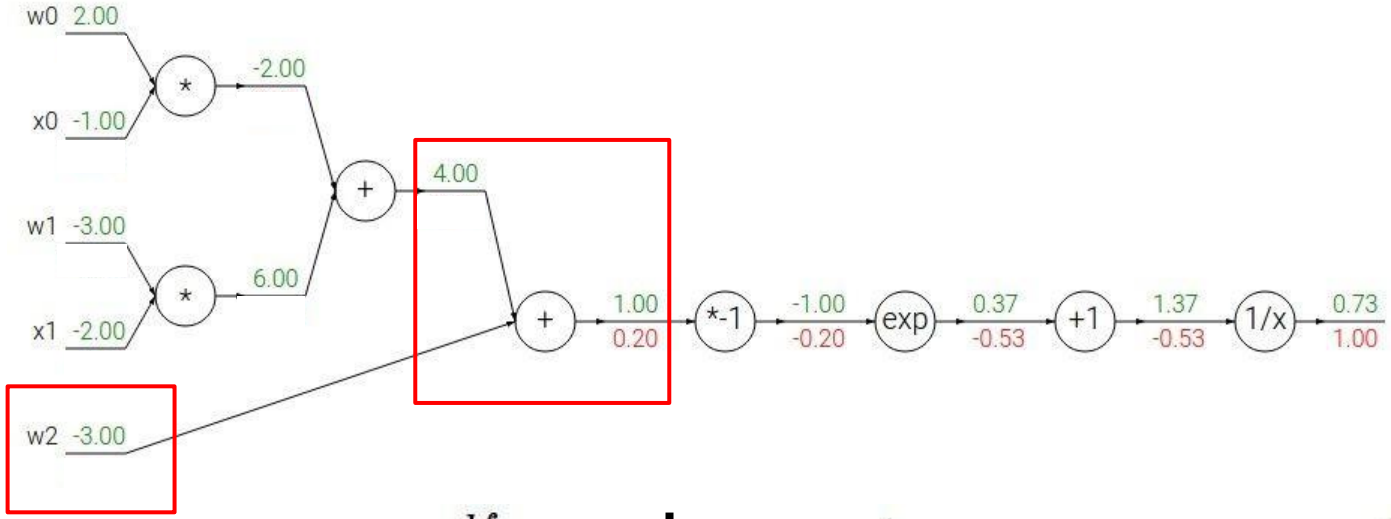
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

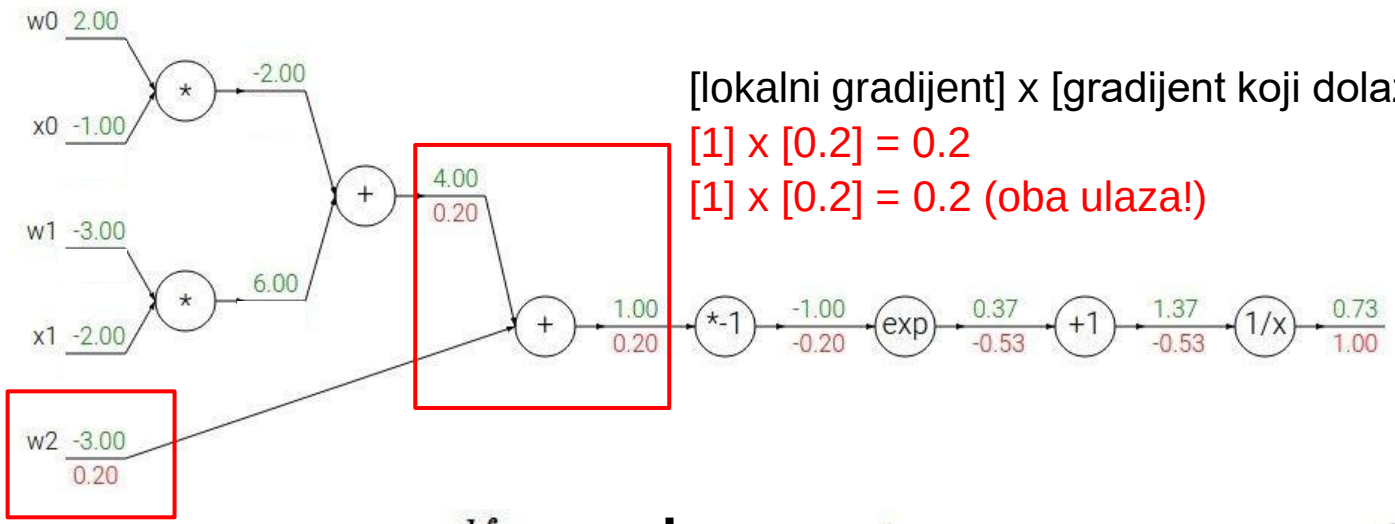
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[lokalni gradijent] x [gradijent koji dolazi od „gore“]

[1] x [0.2] = 0.2

[1] x [0.2] = 0.2 (oba ulaza!)

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

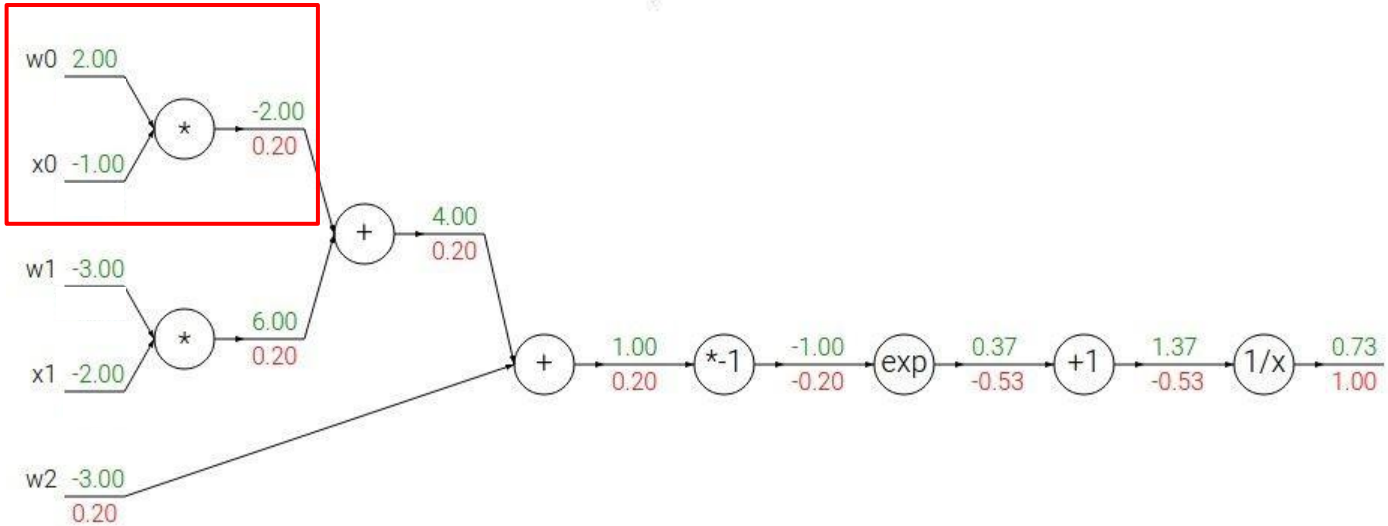
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

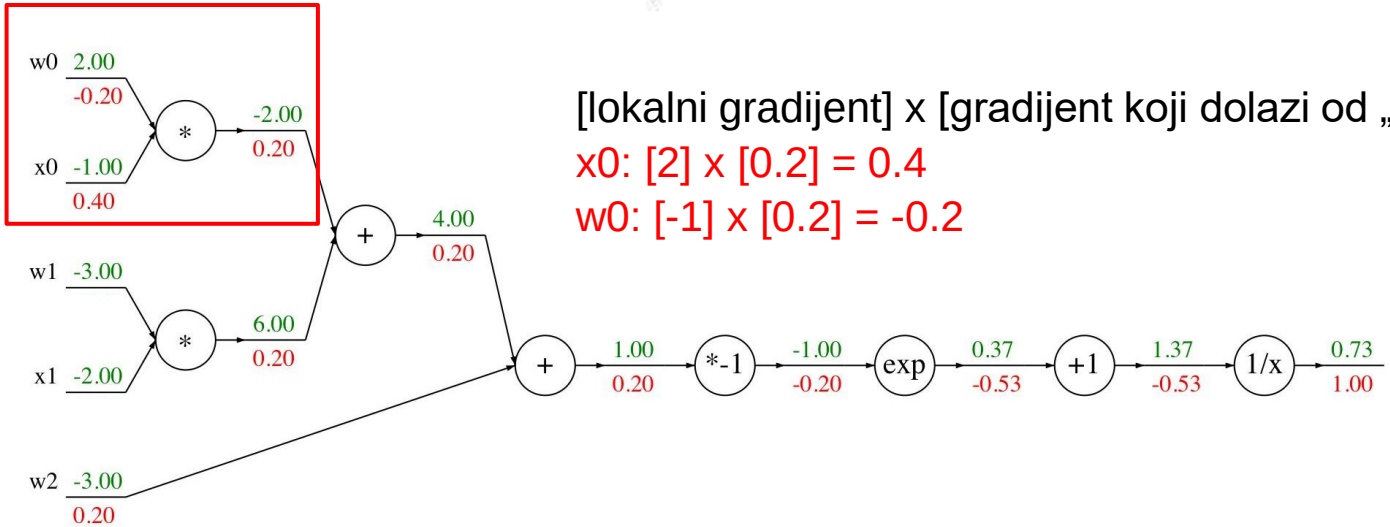
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

Još jedan primer:

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



[lokalni gradijent] x [gradijent koji dolazi od „gore“]

x0: [2] x [0.2] = 0.4

w0: [-1] x [0.2] = -0.2

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

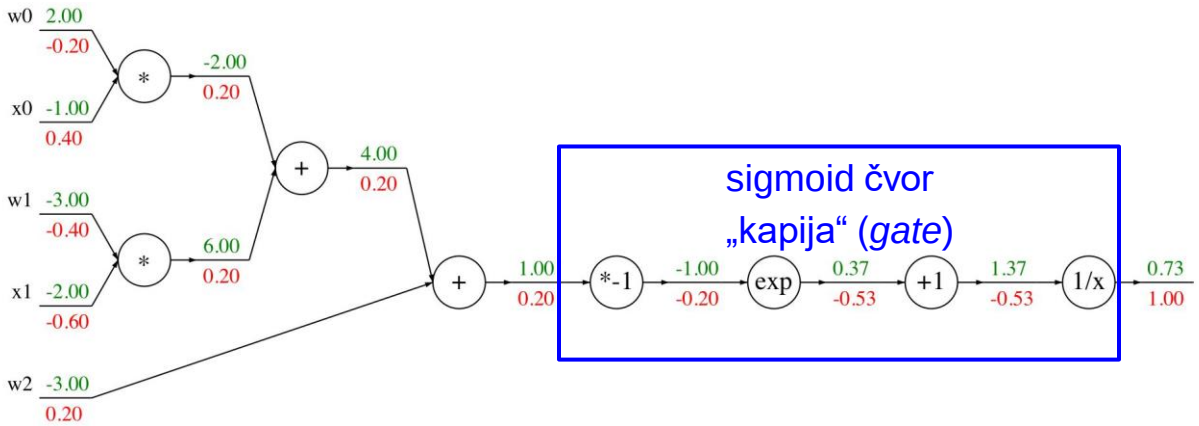
$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

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

sigmoid funkcija

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

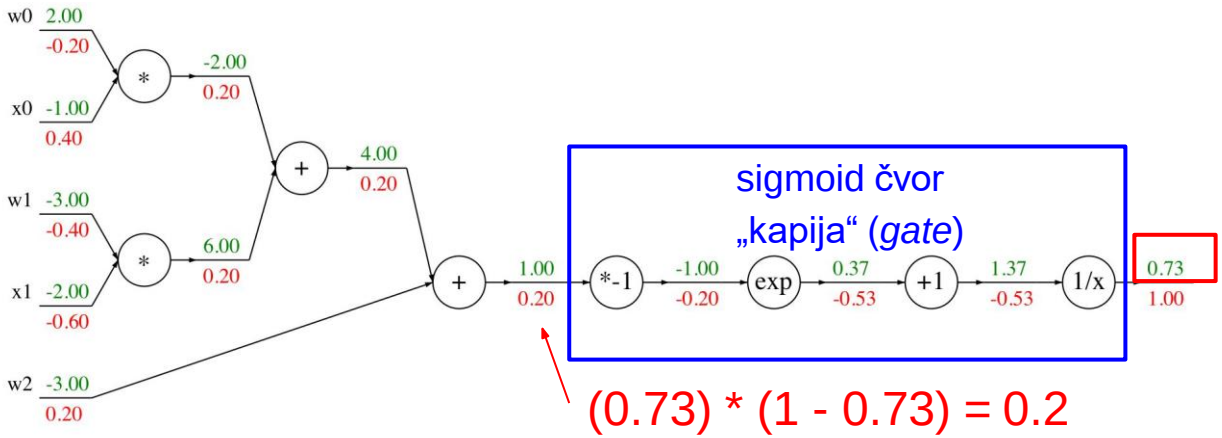


$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

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

sigmoid funkcija

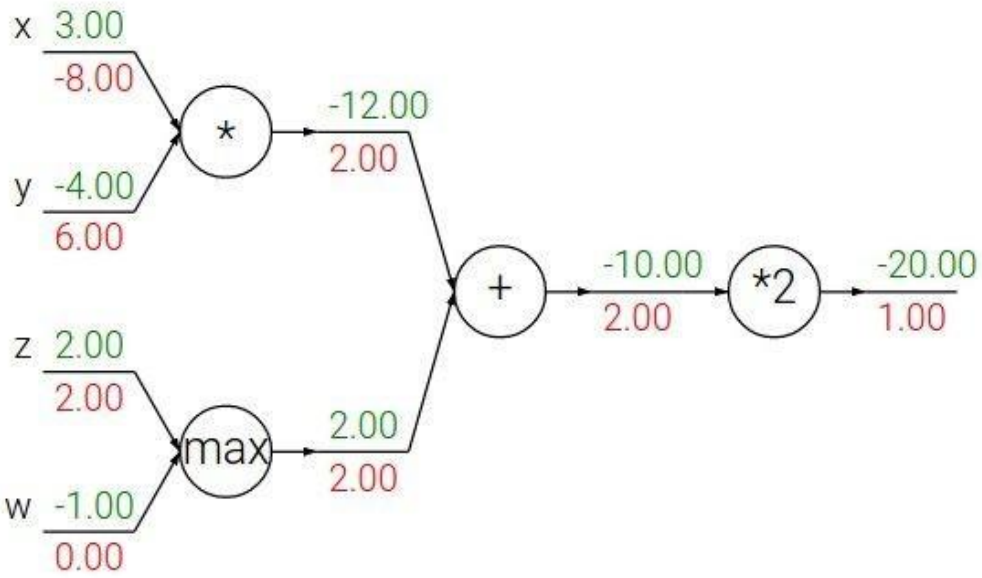
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



(0.73) * (1 - 0.73) = 0.2

Ponašanja čvorova pri prolasku gradijenta unazad

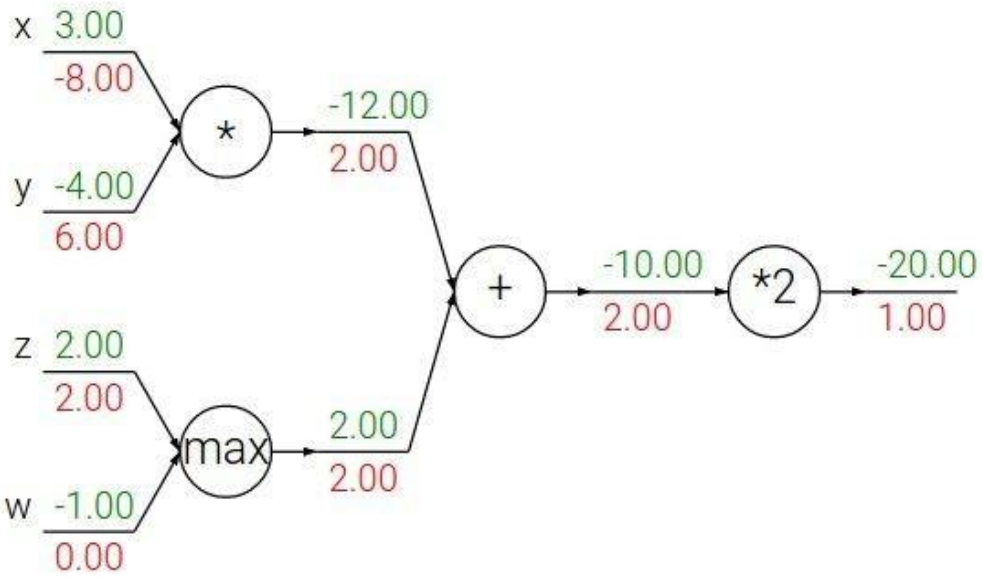
čvor **sabiranja**: distributor gradijenta



Ponašanja čvorova pri prolasku gradijenta unazad

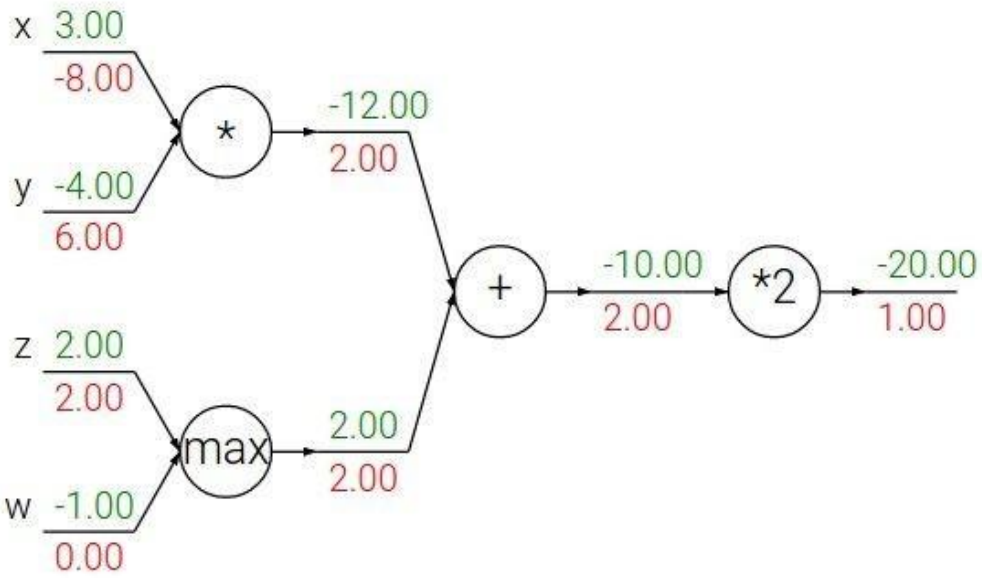
čvor **sabiranja**: distributor gradijenta

Šta radi **max** čvor?



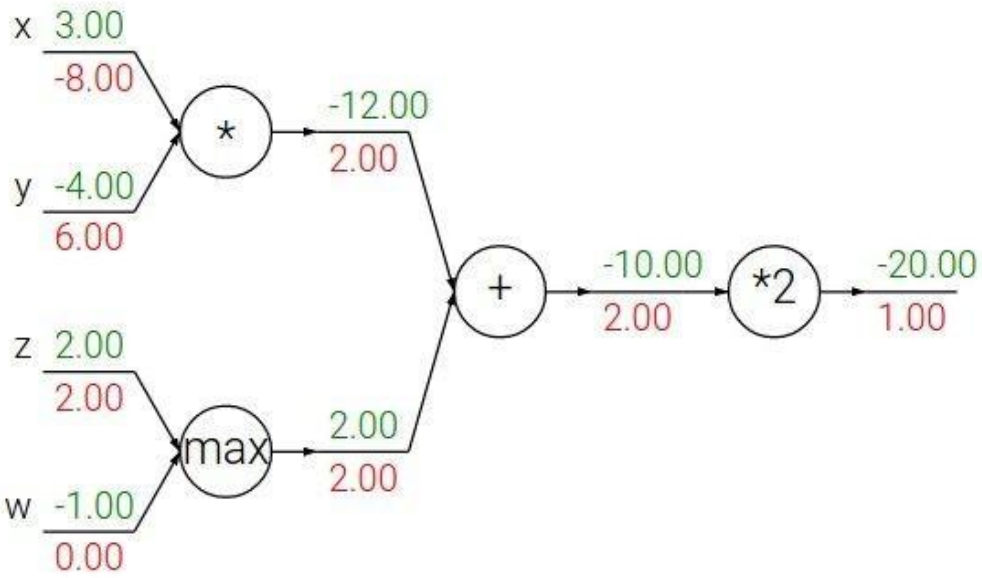
Ponašanja čvorova pri prolasku gradijenta unazad

čvor **sabiranja**: distributor gradijenta
max čvor: ruter gradijenta



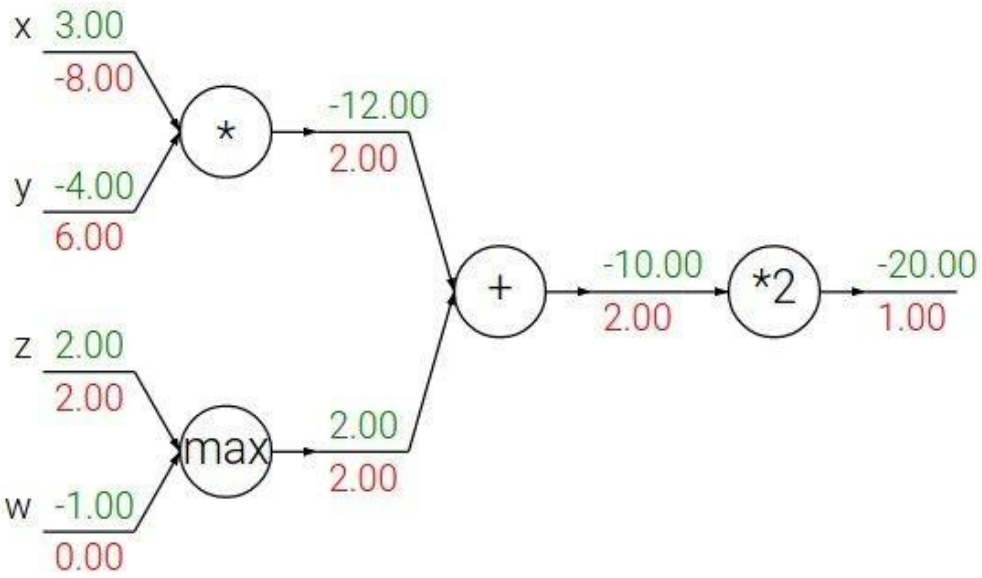
Ponašanja čvorova pri prolasku gradijenta unazad

čvor **sabiranja**: distributor gradijenta
max čvor: ruter gradijenta
Šta radi čvor **množenja**?

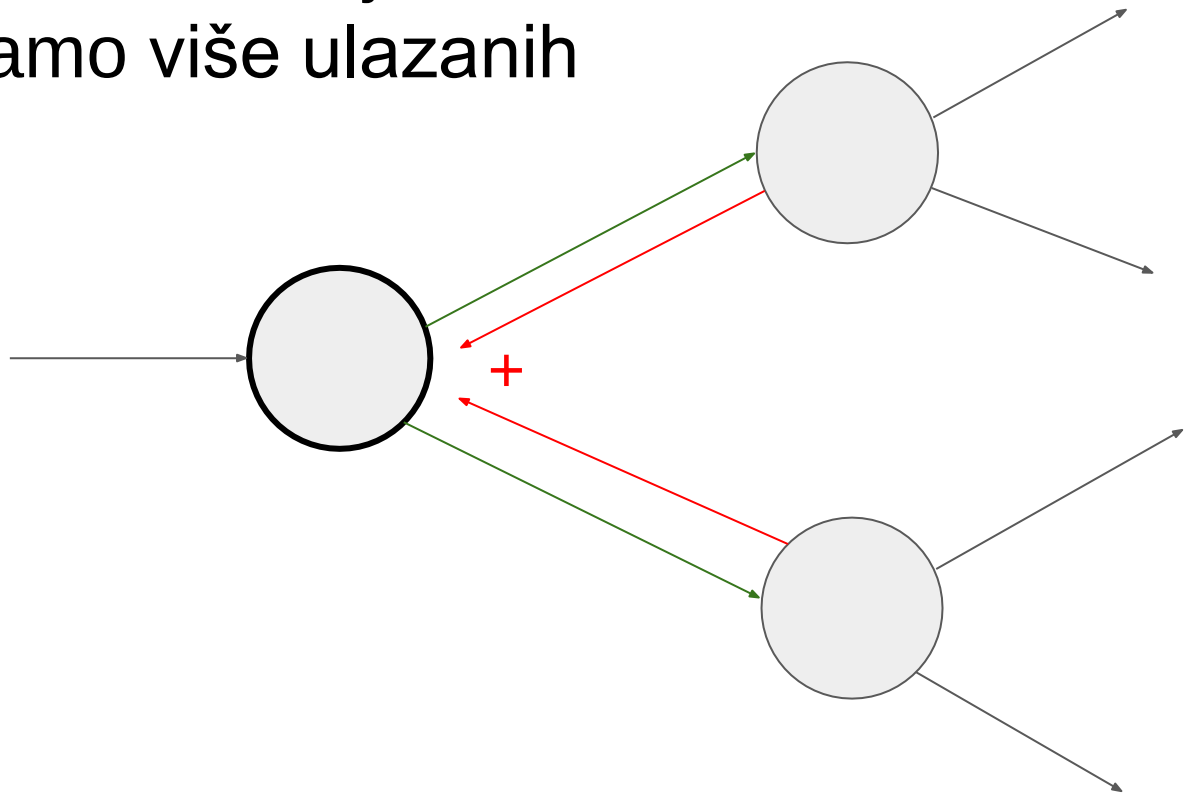


Ponašanja čvorova pri prolasku gradijenta unazad

čvor **sabiranja**: distributor gradijenta
max čvor: ruter gradijenta
čvor **množenja**: razmenjuje gradijente



Gradijenti se sabiraju kada imamo više ulazanih grana



Primer sa vektorima $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

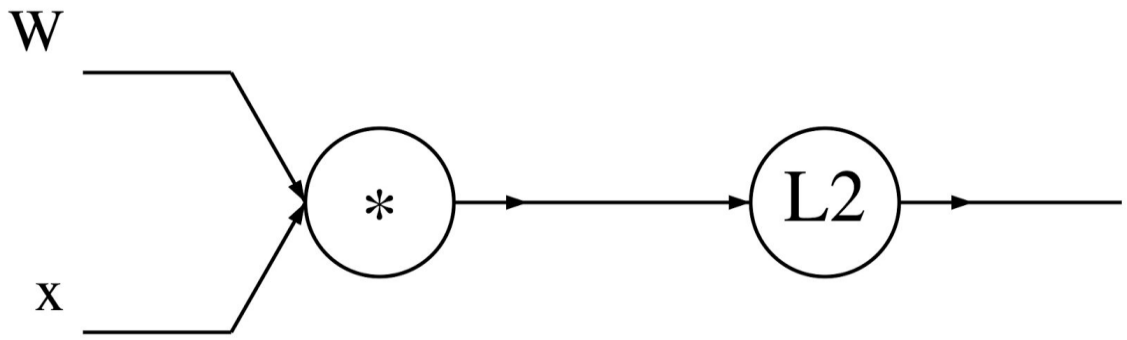
Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$

$$\in \mathbb{R}^n \quad \in \mathbb{R}^{n \times n}$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$



Primer sa vektorima

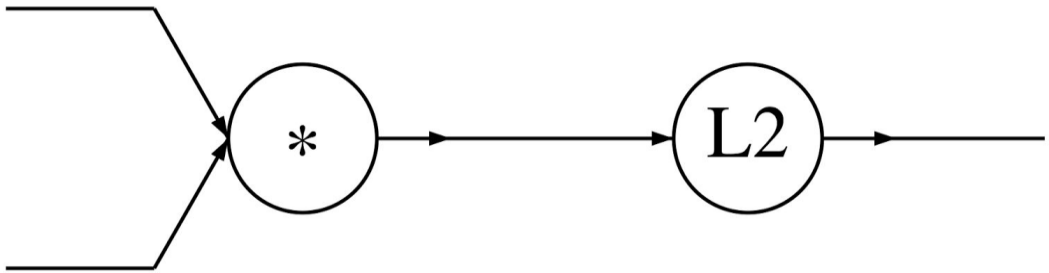
$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$

W

$$\begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$

x

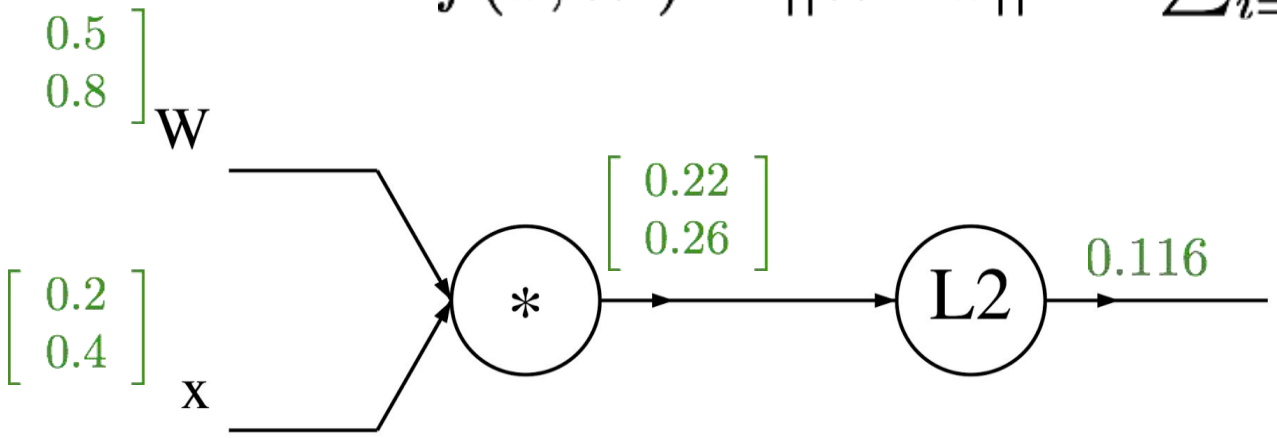


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$

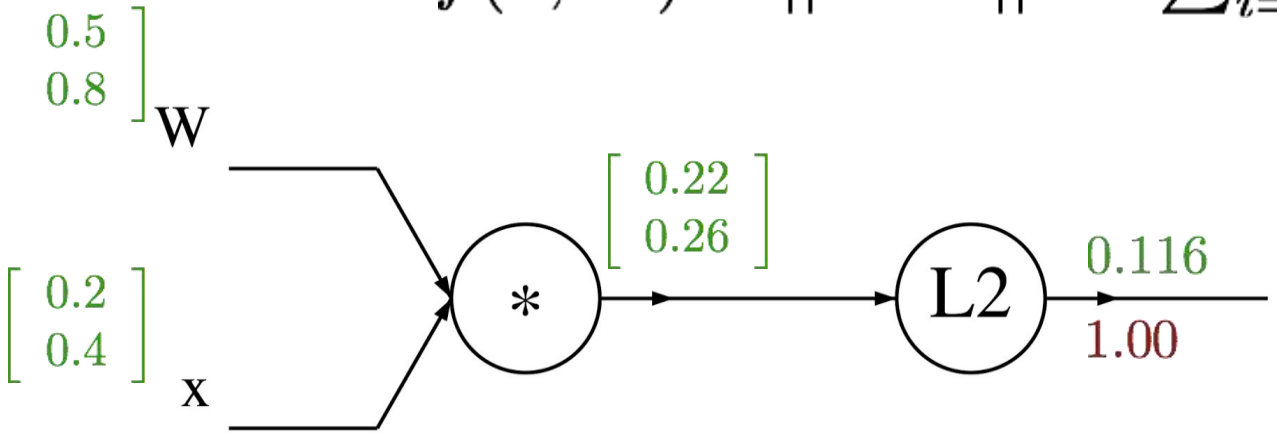


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$

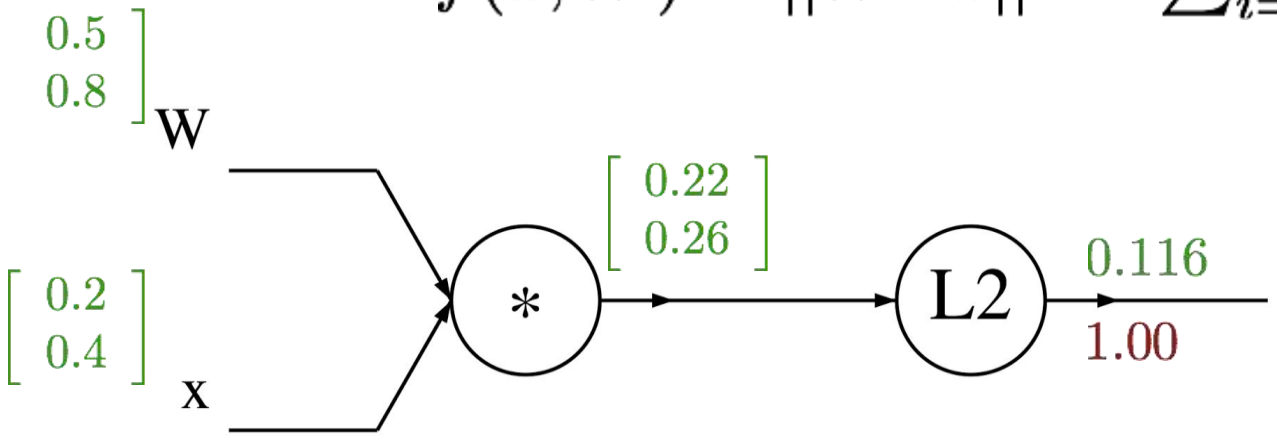


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

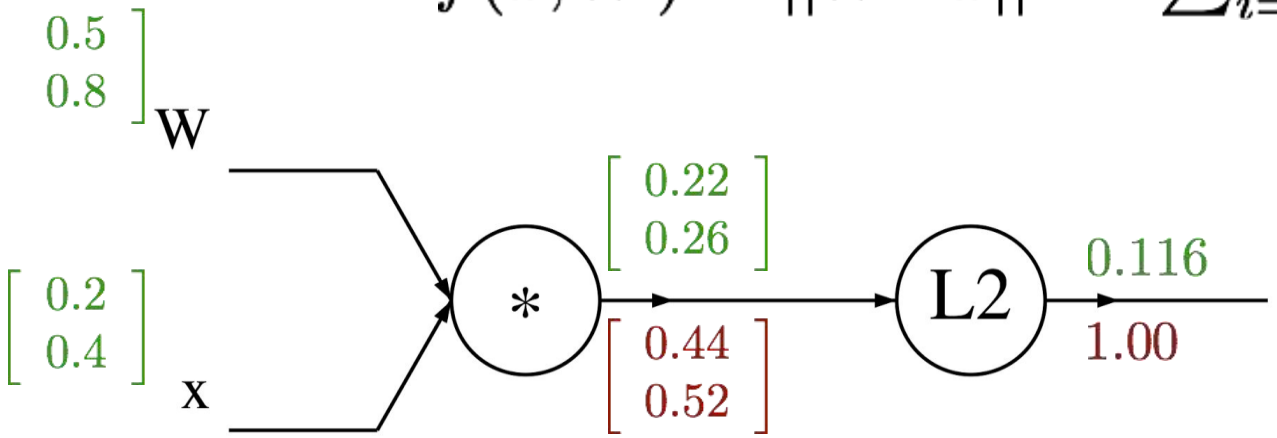
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\boxed{\nabla_q f = 2q}$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

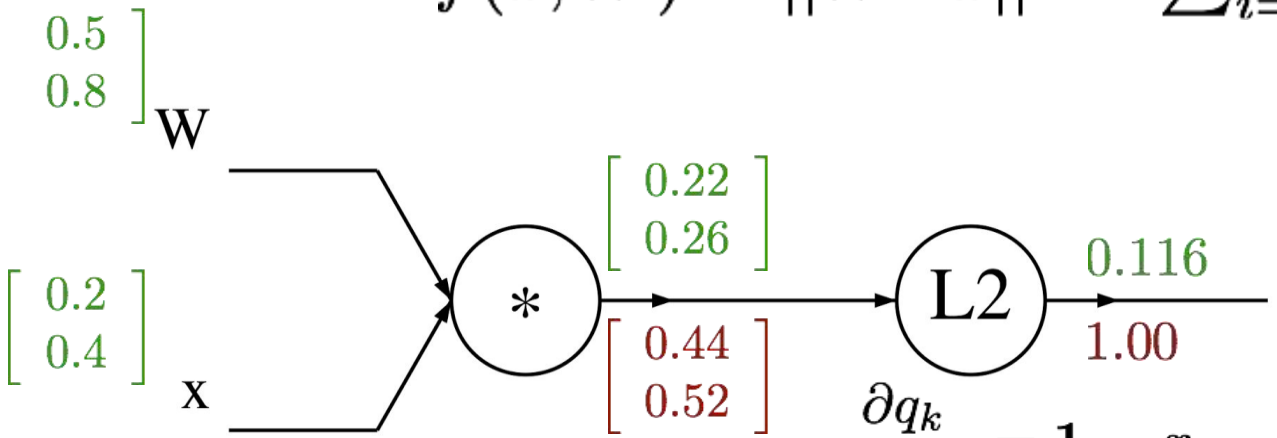
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\boxed{\nabla_q f = 2q}$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$



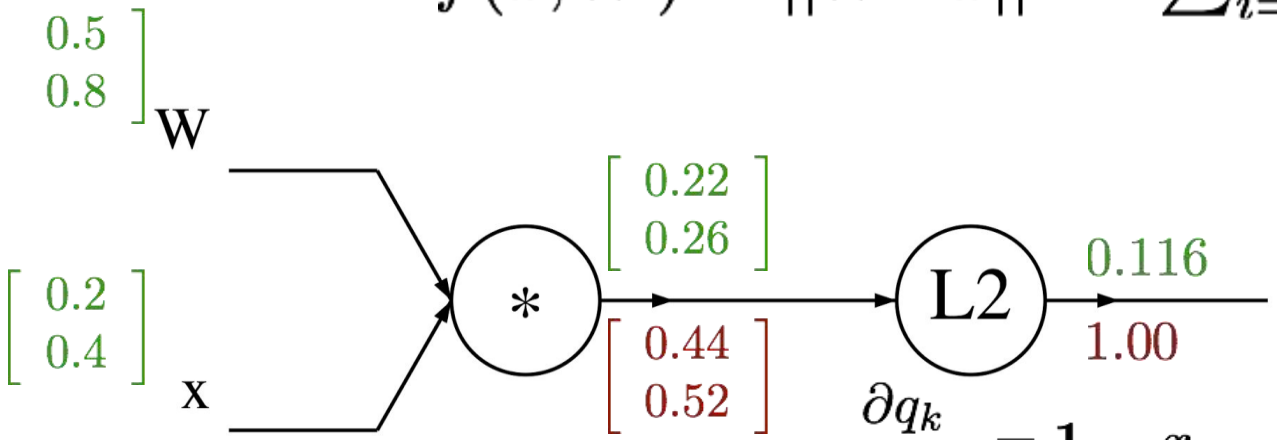
$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

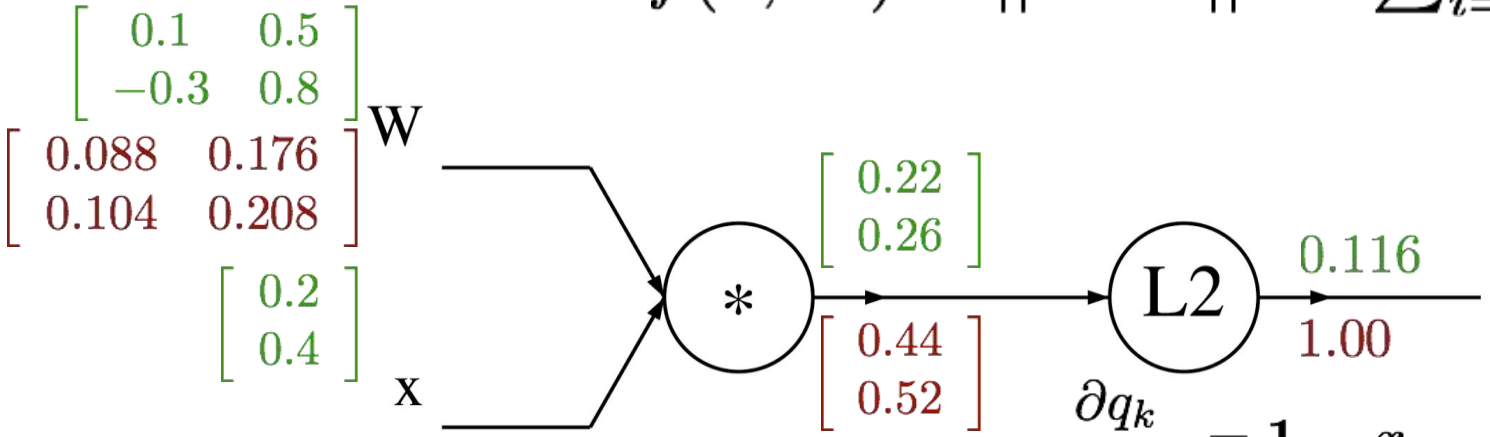
$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

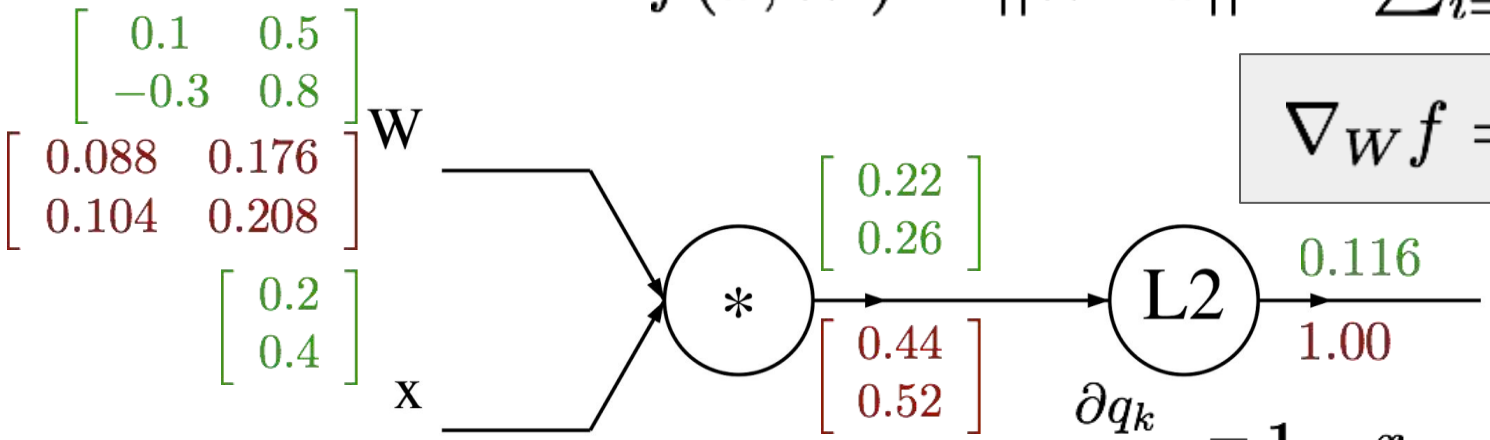
$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$

$$\nabla_W f = 2q \cdot x^T$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

Primer sa vektorima - dodatak

$$\boxed{\nabla_W f = 2q \cdot x^T}$$

$$\begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} \\ a_{21}b_{11} & a_{21}b_{12} \end{bmatrix}$$

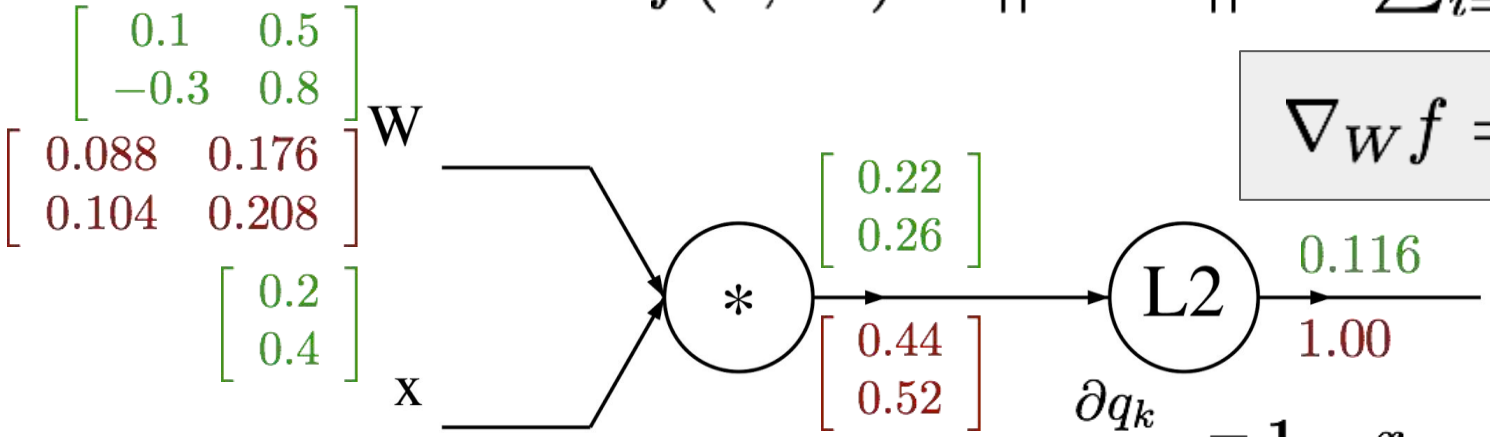
$$\begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix} \begin{bmatrix} 0.2 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.44 * 0.2 & 0.44 * 0.4 \\ 0.52 * 0.2 & 0.52 * 0.4 \end{bmatrix}$$

$$\begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix} \begin{bmatrix} 0.2 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.088 & 0.176 \\ 0.104 & 0.208 \end{bmatrix}$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$

$$\nabla_W f = 2q \cdot x^T$$



Uvek proverite: Gradijent po nekoj promenljivoj mora da ima isti oblik (dimenzije) kao ta promenljiva

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

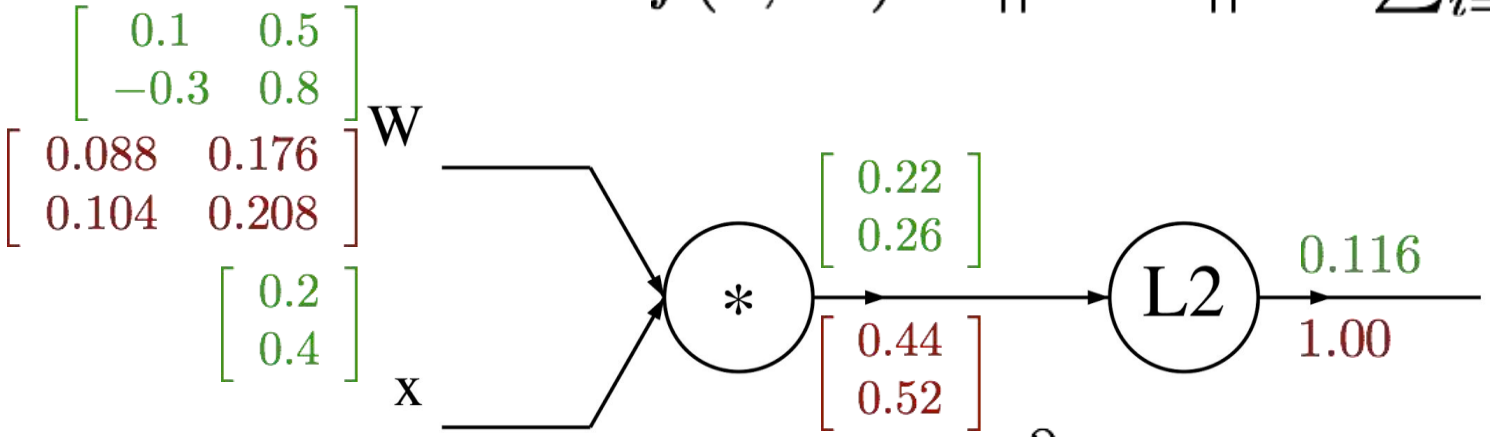
$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$



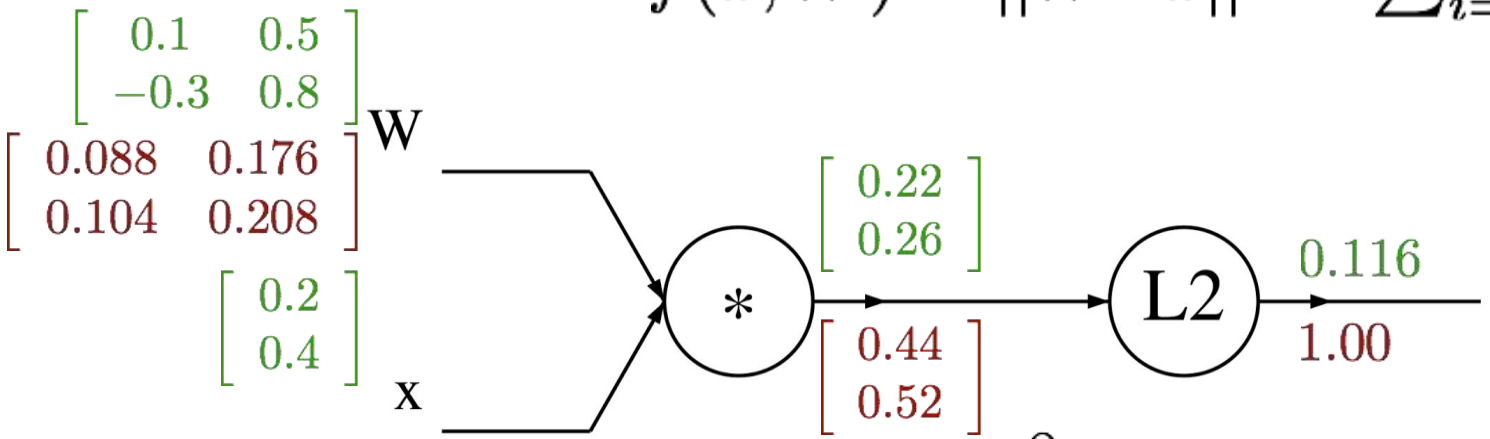
$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

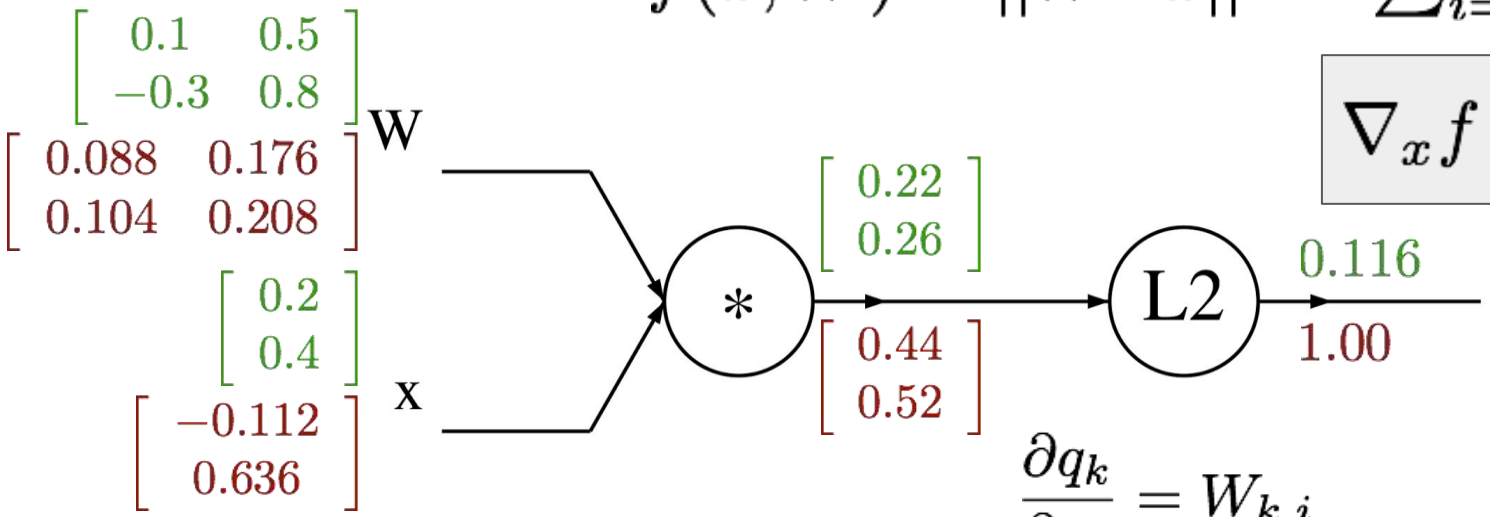
$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k 2 q_k W_{k,i}$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$

$$\nabla_x f = 2W^T \cdot q$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k 2 q_k W_{k,i}$$

Primer sa vektorima - dodatak

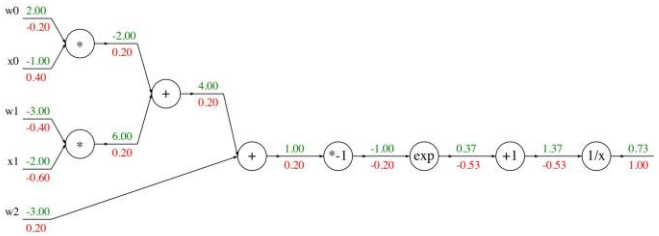
$$\boxed{\nabla_x f = 2W^T \cdot q}$$

$$2\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix}\begin{bmatrix} q_1 \\ q_2 \end{bmatrix} = \begin{bmatrix} w_{11}q_1 + w_{21}q_2 \\ w_{12}q_1 + w_{22}q_2 \end{bmatrix}$$

$$2\begin{bmatrix} 0.1 & -0.3 \\ 0.5 & 0.8 \end{bmatrix}\begin{bmatrix} 0.44 \\ 0.52 \end{bmatrix} = \begin{bmatrix} 0.1 * 0.44 + (-0.3 * 0.52) \\ 0.5 * 0.44 + (0.8 * 0.52) \end{bmatrix}$$

$$\begin{bmatrix} -0.112 \\ 0.636 \end{bmatrix}$$

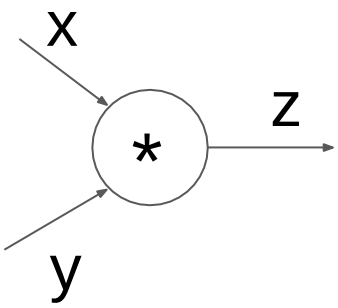
Modularizovana implementacija: *forward / backward API*



Graf (ili Mreža) objekat *(grubi pseudo-kod)*

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

Modularizovana implementacija: *forward / backward API*



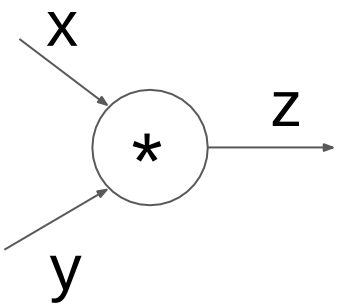
(x,y,z su skalari)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

$$\frac{\partial L}{\partial z}$$

$$\frac{\partial L}{\partial x}$$

Modularizovana implementacija: *forward / backward API*



(x,y,z su skalari)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Primer: Slojevi u *Caffe* okruženju za *Deep Learning*

Branch: master ▾ caffe / src / caffe / **layers** / | Create new file | Upload files | Find file | History

shelhamer committed on GitHub Merge pull request #4630 from BlGene/load_hdf5_fix ... Latest commit e687a71 21 days ago

..

absval_layer.cpp	dismantle layer headers	a year ago
absval_layer.cu	dismantle layer headers	a year ago
accuracy_layer.cpp	dismantle layer headers	a year ago
argmax_layer.cpp	dismantle layer headers	a year ago
base_conv_layer.cpp	enable dilated deconvolution	a year ago
base_data_layer.cpp	Using default from proto for prefetch	3 months ago
base_data_layer.cu	Switched multi-GPU to NCCL	3 months ago
batch_norm_layer.cpp	Add missing spaces besides equal signs in batch_norm_layer.cpp	4 months ago
batch_norm_layer.cu	dismantle layer headers	a year ago
batch_reindex_layer.cpp	dismantle layer headers	a year ago
batch_reindex_layer.cu	dismantle layer headers	a year ago
bias_layer.cpp	Remove incorrect cast of gemm int arg to Dtype in BiasLayer	a year ago
bias_layer.cu	Separation and generalization of ChannelwiseAffineLayer into BiasLayer	a year ago
bnll_layer.cpp	dismantle layer headers	a year ago
bnll_layer.cu	dismantle layer headers	a year ago
concat_layer.cpp	dismantle layer headers	a year ago
concat_layer.cu	dismantle layer headers	a year ago
contrastive_loss_layer.cpp	dismantle layer headers	a year ago
contrastive_loss_layer.cu	dismantle layer headers	a year ago
conv_layer.cpp	add support for 2D dilated convolution	a year ago
conv_layer.cu	dismantle layer headers	a year ago
crop_layer.cpp	remove redundant operations in Crop layer (#5138)	2 months ago
crop_layer.cu	remove redundant operations in Crop layer (#5138)	2 months ago
cudnn_conv_layer.cpp	dismantle layer headers	a year ago
cudnn_conv_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_lcn_layer.cpp	dismantle layer headers	a year ago
cudnn_lcn_layer.cu	dismantle layer headers	a year ago
cudnn_lrn_layer.cpp	dismantle layer headers	a year ago
cudnn_lrn_layer.cu	dismantle layer headers	a year ago
cudnn_pooling_layer.cpp	dismantle layer headers	a year ago
cudnn_pooling_layer.cu	dismantle layer headers	a year ago
cudnn_relu_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_relu_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_sigmoid_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_sigmoid_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_softmax_layer.cpp	dismantle layer headers	a year ago
cudnn_softmax_layer.cu	dismantle layer headers	a year ago
cudnn_tanh_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_tanh_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
data_layer.cpp	Switched multi-GPU to NCCL	3 months ago
deconv_layer.cpp	enable dilated deconvolution	a year ago
deconv_layer.cu	dismantle layer headers	a year ago
dropout_layer.cpp	supporting N-D Blobs in Dropout layer Reshape	a year ago
dropout_layer.cu	dismantle layer headers	a year ago
dummy_data_layer.cpp	dismantle layer headers	a year ago
eltwise_layer.cpp	dismantle layer headers	a year ago
eltwise_layer.cu	dismantle layer headers	a year ago
elu_layer.cpp	ELU layer with basic tests	a year ago
elu_layer.cu	ELU layer with basic tests	a year ago
embed_layer.cpp	dismantle layer headers	a year ago
embed_layer.cu	dismantle layer headers	a year ago
euclidean_loss_layer.cpp	dismantle layer headers	a year ago
euclidean_loss_layer.cu	dismantle layer headers	a year ago
exp_layer.cpp	Solving issue with exp layer with base e	a year ago
exp_layer.cu	dismantle layer headers	a year ago

Caffe is licensed under BSD 2-Clause

Caffe Sigmoid Sloj

```cpp
#include <cmath>
#include <vector>

#include "caffe/layers/sigmoid_layer.hpp"

namespace caffe {

template <typename Dtype>
inline Dtype sigmoid(Dtype x) {
  return 1. / (1. + exp(-x));
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
    const vector<Blob<Dtype>*>& top) {
  const Dtype* bottom_data = bottom[0]->cpu_data();
  Dtype* top_data = top[0]->mutable_cpu_data();
  const int count = bottom[0]->count();
  for (int i = 0; i < count; ++i) {
    top_data[i] = sigmoid(bottom_data[i]);
  }
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down,
    const vector<Blob<Dtype>*>& bottom) {
  if (propagate_down[0]) {
    const Dtype* top_data = top[0]->cpu_data();
    const Dtype* top_diff = top[0]->cpu_diff();
    Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
    const int count = bottom[0]->count();
    for (int i = 0; i < count; ++i) {
      const Dtype sigmoid_x = top_data[i];
      bottom_diff[i] = top_diff[i] * sigmoid_x * (1. - sigmoid_x);
    }
  }
}

#ifdef CPU_ONLY
STUB_GPU(SigmoidLayer);
#endif

INSTANTIATE_CLASS(SigmoidLayer);


}  // namespace caffe
```

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

$$(1-\sigma(x))\,\sigma(x)$$

* top_diff (pravilo ulančavanja izvoda)

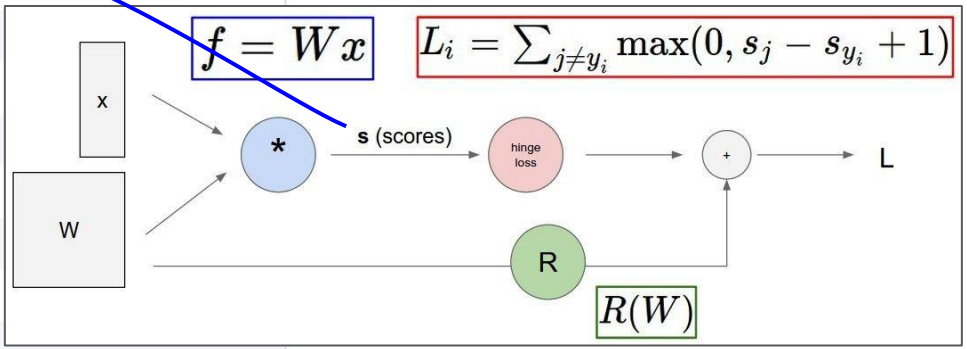
Caffe is licensed under BSD 2-Clause

Kako bi izgledao SVM / Softmax klasifikator kao Graf Izračunavanja? Da li možemo obučavanje da organizujemo na forward/backward način?

Npr. SVM:

margine

```
# receive W (weights), X (data)
# forward pass (we have 8 lines)
scores = #...
margins = #...
data_loss = #...
reg_loss = #...
loss = data_loss + reg_loss
# backward pass (we have 5 lines)
dmargins = # ... (optionally, we go direct to dscores)
dscores = #...
dW = #...
```



Rezime...

- neuronske mreže su u praksi veoma velike: nije praktično da se ručno određuju gradijenti za svaki parametar (gradijente određujemo onkako kako smo to radili danas)
- **backpropagation** = rekurzivna primena ulančavanja izvoda kroz graf izračunavanja da bi izračunali sve potrebne gradijente
- implementacije neuronskih mreža koriste strukturu grafa gde svaki čvor implementira **forward**() / **backward**() API
- **forward**: izračunavamo odgovaraću operaciju i čuvamo rezultate u memoriji da bi mogli da ih iskoristimo za računanje gradijenata kasnije u **backpropagation**
- **backward**: primenjujemo pravilo ulančavanja da bi izračunali gradijente za sve parametre u odnosu na funkciju greške

Neuronske Mreže
Neural Networks

Neuronske Mreže: prvo bez analogije sa ljudskim mozgom

(**Ranije**) Linearna skor funkcija: $f = Wx$

Neuronske Mreže: prvo bez analogije sa ljudskim mozgom

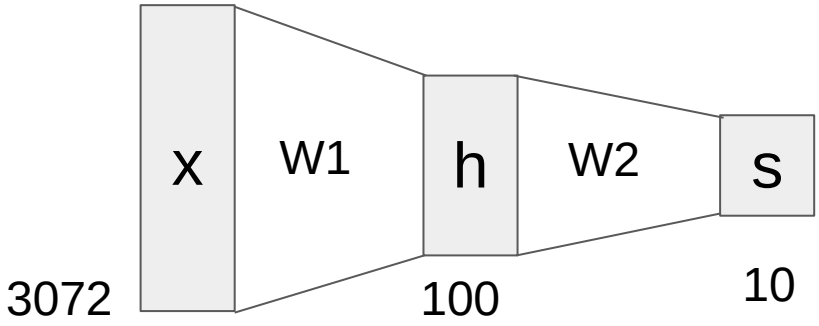
(**Ranije**) Linearna skor funkcija: $$f = Wx$$

(**Sada**) Neuroska mreža sa dva sloja: $$f = W_2 \max(0, W_1 x)$$

Neuronske Mreže: prvo bez analogije sa ljudskim mozgom

(**Ranije**) Linearna skor funkcija: $$f = Wx$$

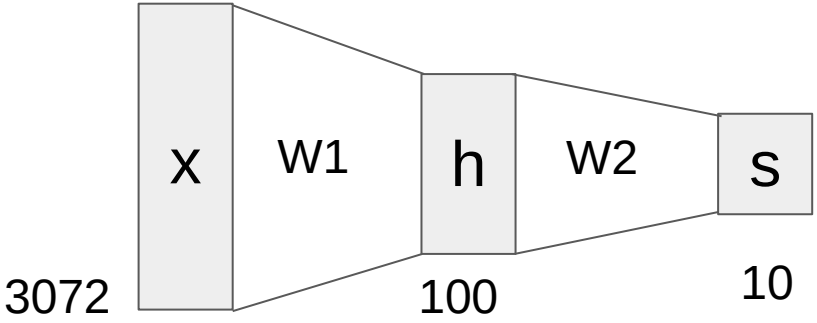
(**Sada**) Neuroska mreža sa dva sloja: $$f = W_2 \max(0, W_1 x)$$

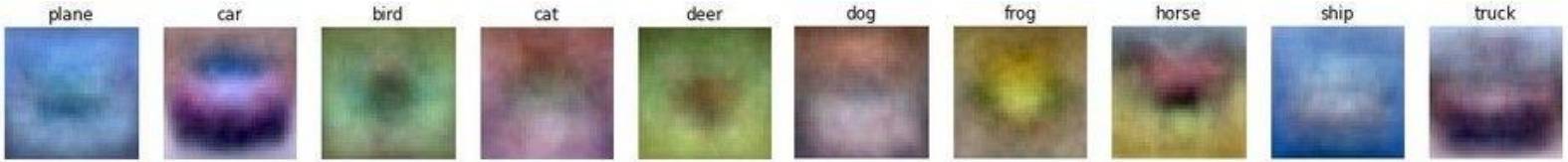


Neuronske Mreže: prvo bez analogije sa ljudskim mozgom

(**Ranije**) Linearna skor funkcija: $$f = Wx$$

(**Sada**) Neuroska mreža sa dva sloja: $$f = W_2 \max(0, W_1 x)$$





Neuronske Mreže: prvo bez analogije sa ljudskim mozgom

(**Ranije**) Linearna skor funkcija: $$f = Wx$$

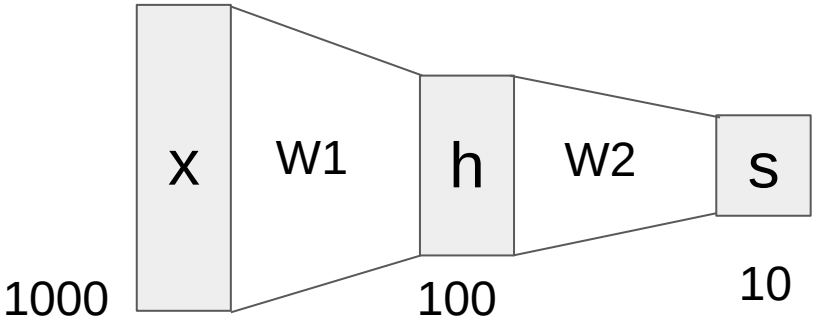
(**Sada**) Neuroska mreža sa dva sloja: $$f = W_2 \max(0, W_1 x)$$

Neuroska mreža sa tri sloja:

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

Kompletna implementacija dvo-slojne Neuronske Mreže ima ~20 linija koda:

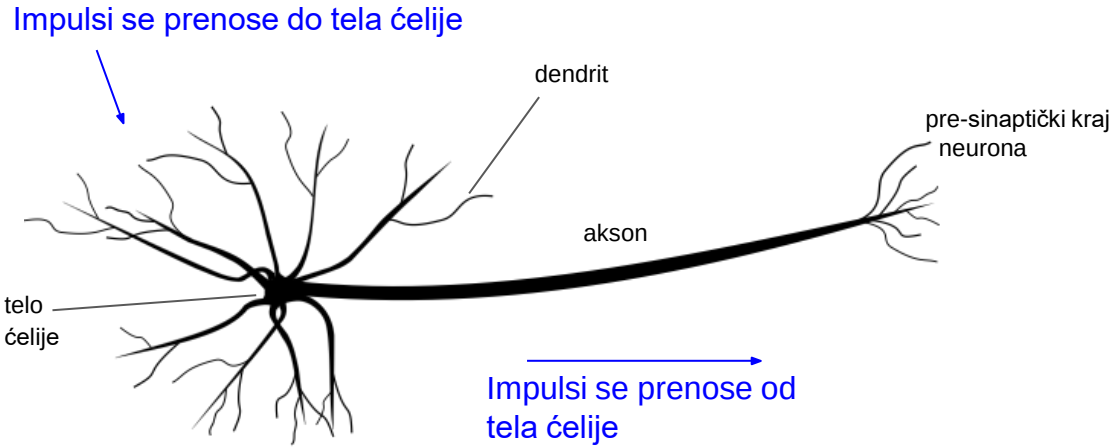
```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```



Kako bi ste to sami uradili?

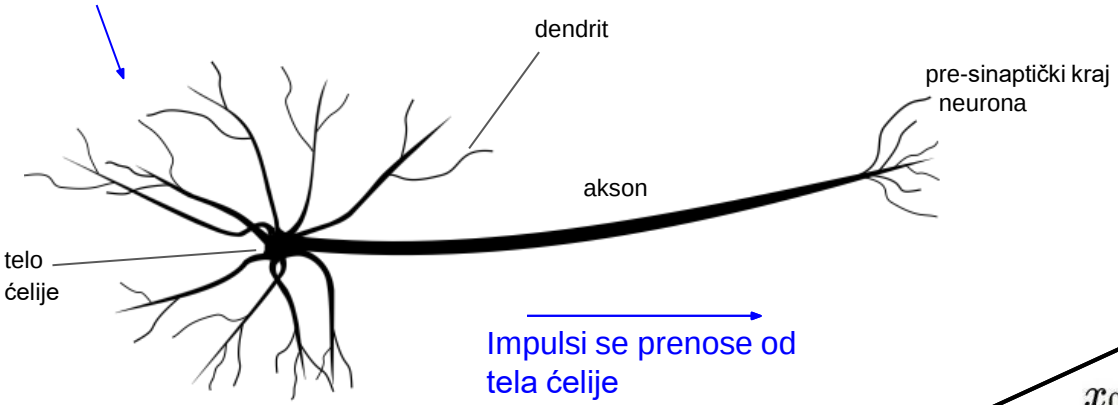
```
# receive W1,W2,b1,b2 (weights/biases), X (data)
# forward pass:
h1 = #... function of X,W1,b1
scores = #... function of h1,W2,b2
loss = #... (several lines of code to evaluate Softmax loss)
# backward pass:
dscores = #...
dh1,dW2,db2 = #...
dW1,db1 = #...
```

This image by Fotis Bobolas is
licensed under CC-BY 2.0

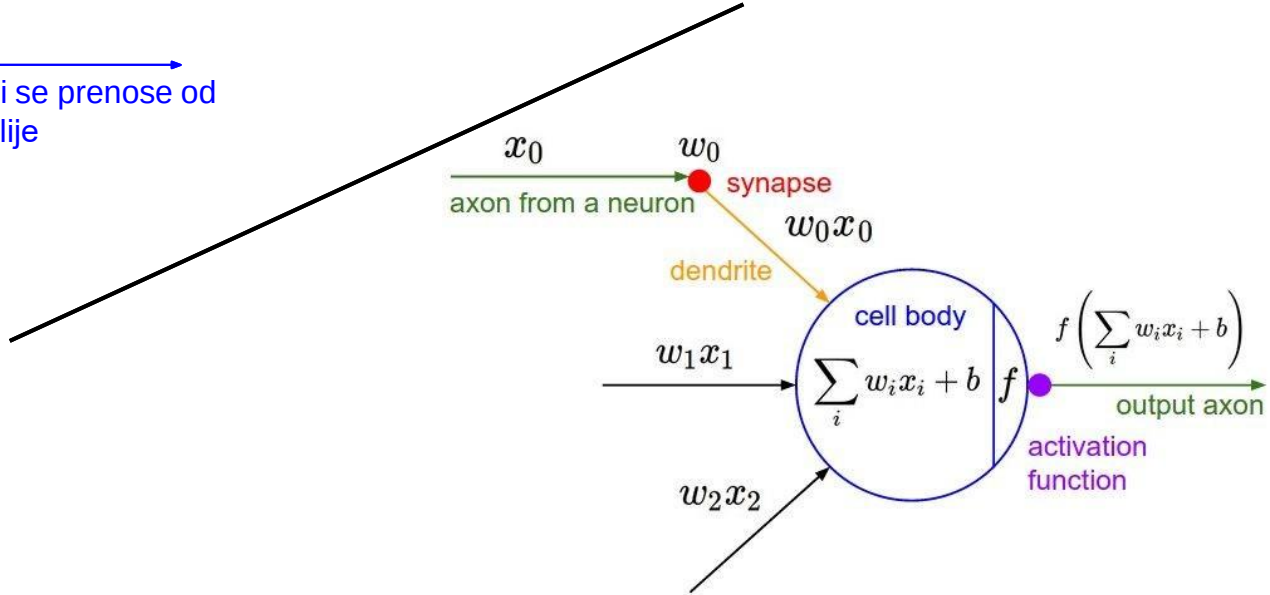


This image by Felipe Perucho is licensed under CC-BY 3.0

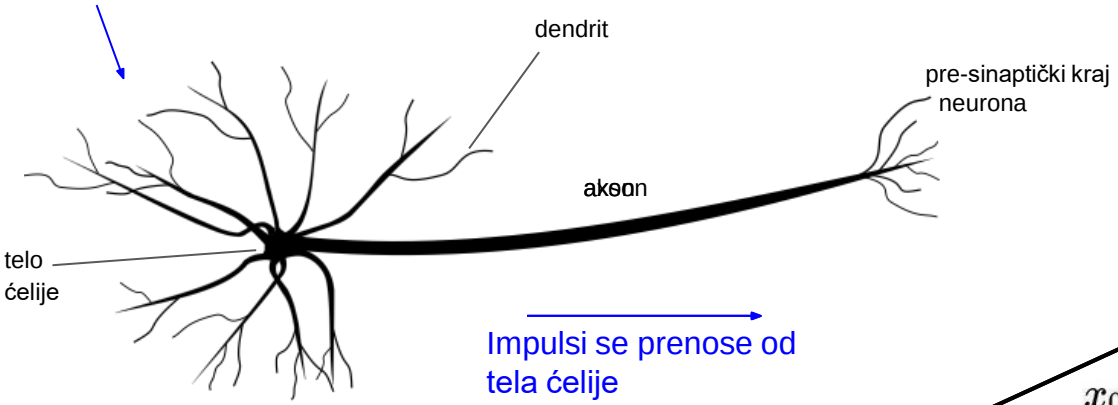
Impulsi se prenose do tela ćelije



This image by Felipe Perucho is licensed under CC-BY 3.0

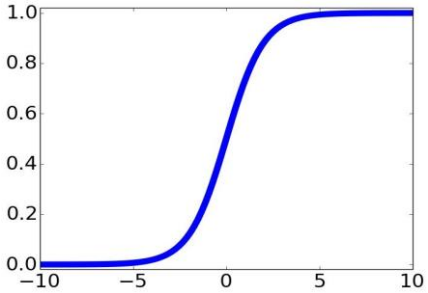


Impulsi se prenose do tela ćelije



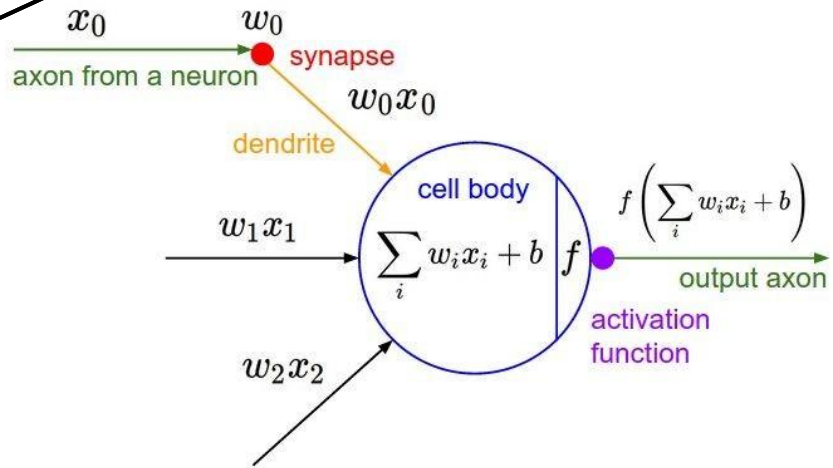
Impulsi se prenose od tela ćelije

This image by Felipe Perucho is licensed under CC-BY 3.0

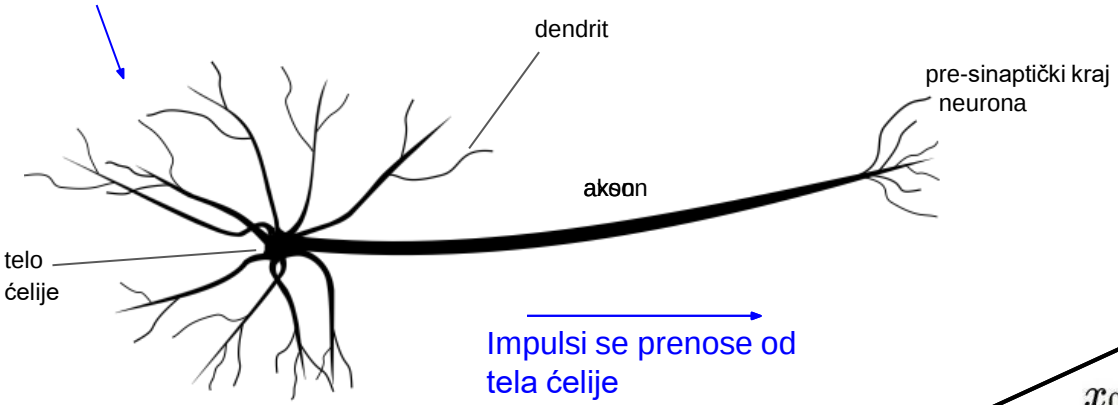


sigmoid aktivacija

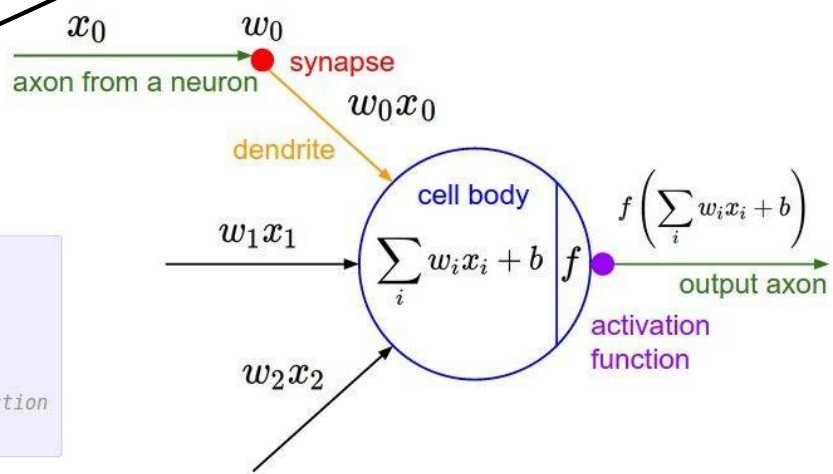
$$\frac{1}{1+e^{-x}}$$



Impulsi se prenose do tela ćelije



This image by Felipe Perucho is licensed under CC-BY 3.0




```python
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

Treba biti pažljiv u pravljenju analogija sa mozgom!

Biološki neuroni:

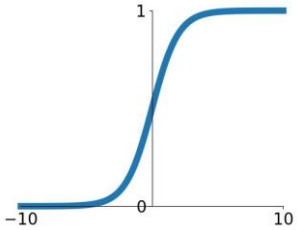
- Mnogo različitih tipova
- Dendriti vrše kompleksne ne-linearne operacije
- Sinapse nisu samo jedna težina već kompleksan ne-linearan dinamički sistem
- ....

[Dendritic Computation. London and Hausser]

Aktivacione funkcije

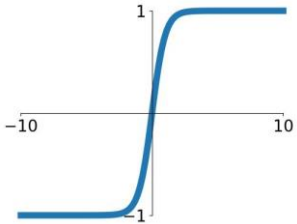
Sigmoid

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



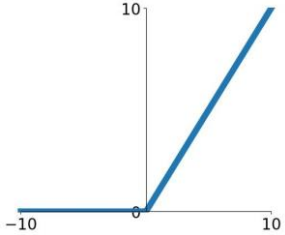
tanh

$$\tanh(x)$$



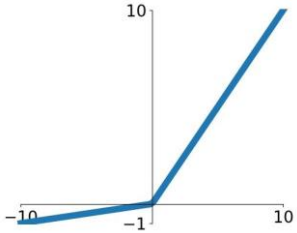
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

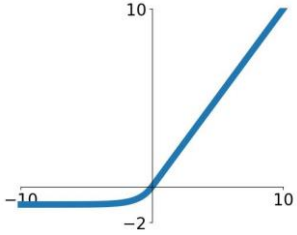


Maxout

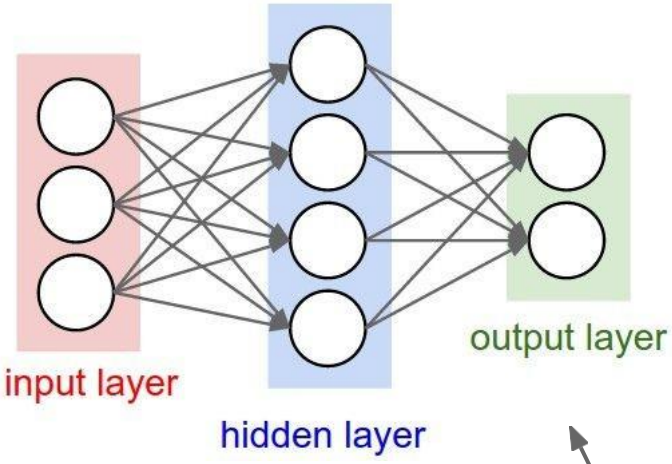
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

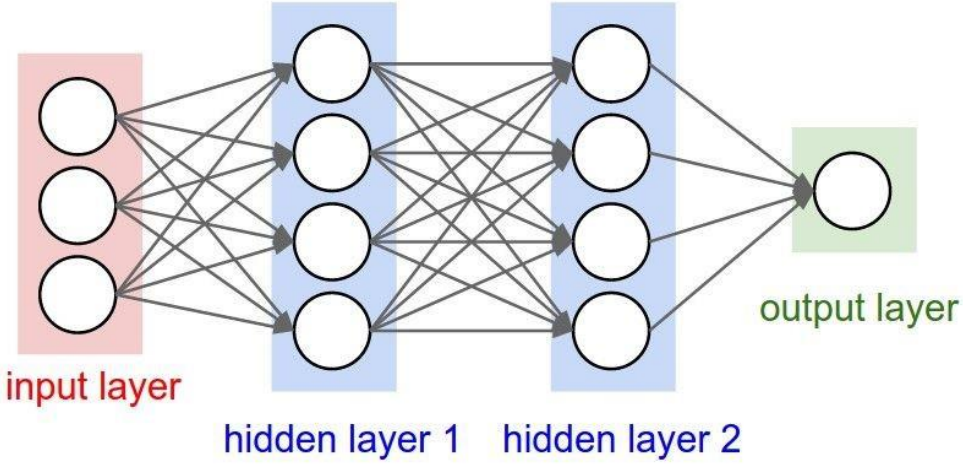
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neuronske Mreže: Arhitekture





“2-slojna NN”, ili
“Neuronska mreža koja ima jedan skriveni sloj”

“3-slojna NN”, ili
“Neuronska mreža koja ima dva skrivena sloja”

“Potpuno-povezani” slojevi

Primer izračunavanja prenosa unapred (*feed-forward*) u Neuronskoj Mreži

```
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

Možemo, vrlo efikasno, vektorskim operacijama da izračunamo aktivacije za jedan ceo sloj u mreži.

Rezime

- Uređujemo neurone u potputno-povezane slojeve
- Apstrakcija jednog sloja nam dozvoljava da koristimo vektorske operacije zarad efikasnosti (npr. množenje matrica)
- Neuronske mreže nisu baš stvarno *neuronske* u biološkom smislu

Osnovi Računarske Inteligencije

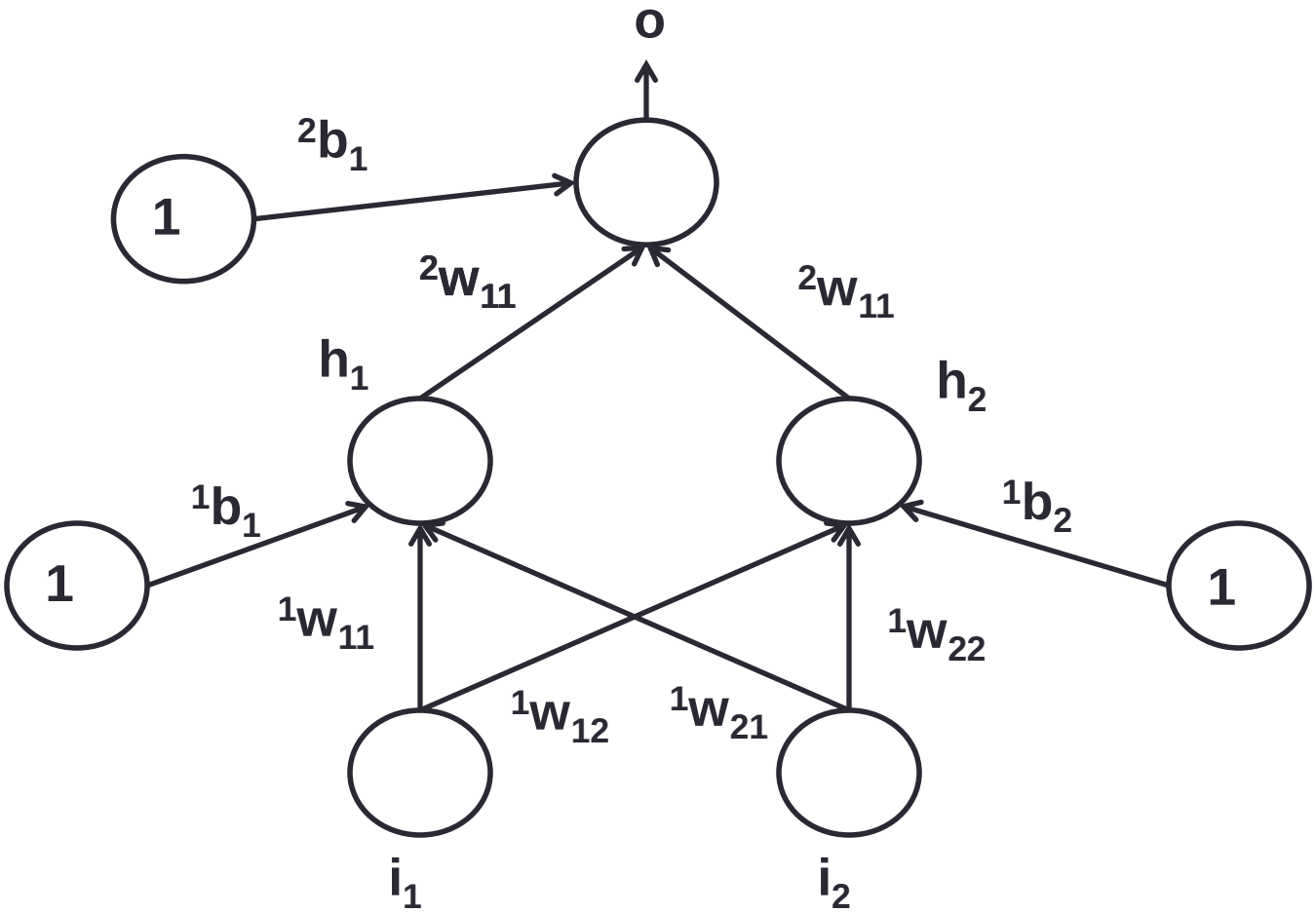
Neuronska Mreža
Primer

predavač: Aleksandar Kovačević

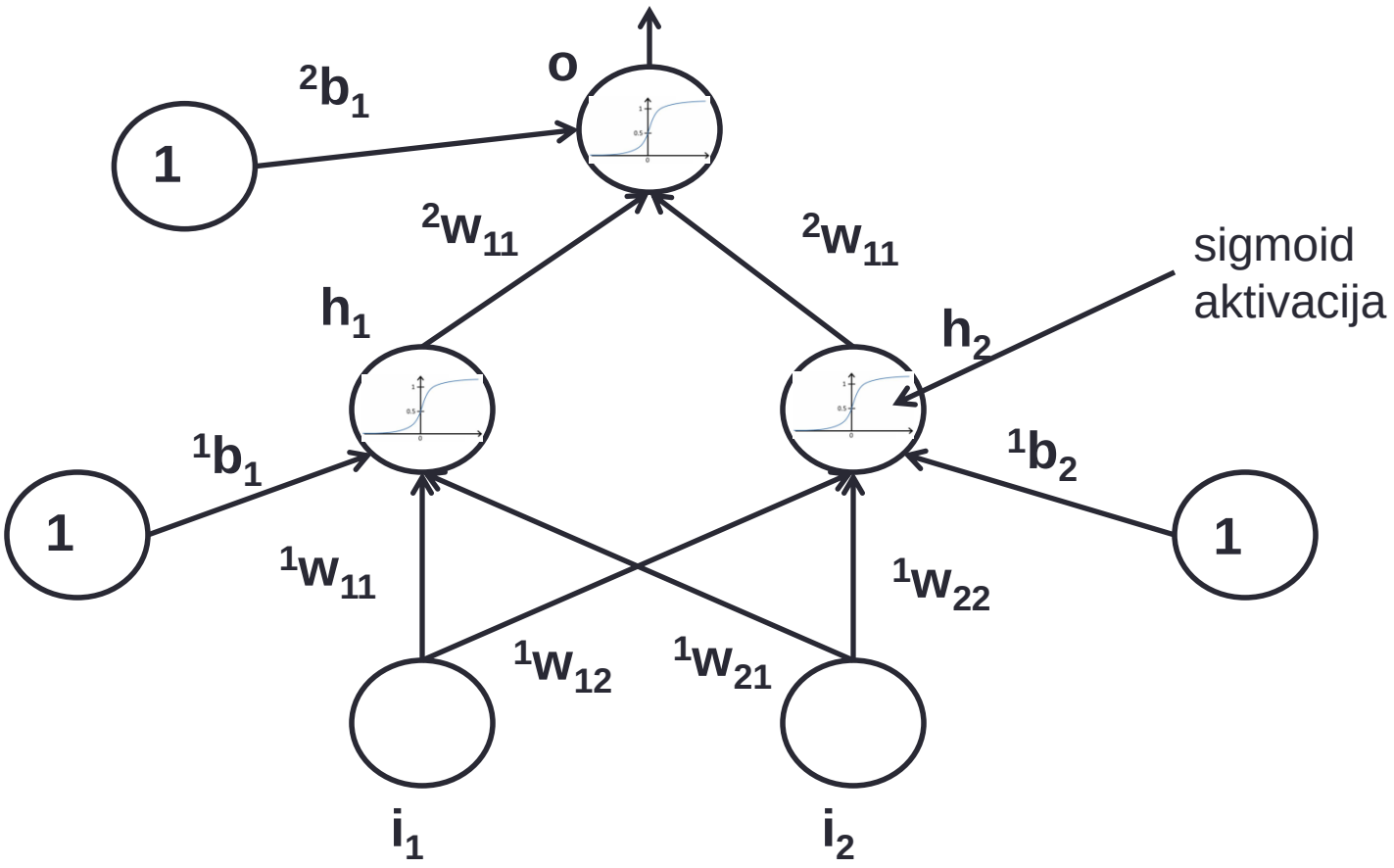
Uvodne napomene

- U nastavku je dat jedan primer obučavanja jednostavne neuronske mreže.
- Arhitektura mreže je odabrana proizvoljno, jedini cilj je da se demostriraju svi koraci obučavanja.
- Skup podataka ima samo jedan primer, vektor [1 1], koji ima zadati izlaz 1.
- Prateći Matlab kod takođe ima za cilj samo da demonstira korake obučavanja. Optimalnost i brzina nisu cilj.

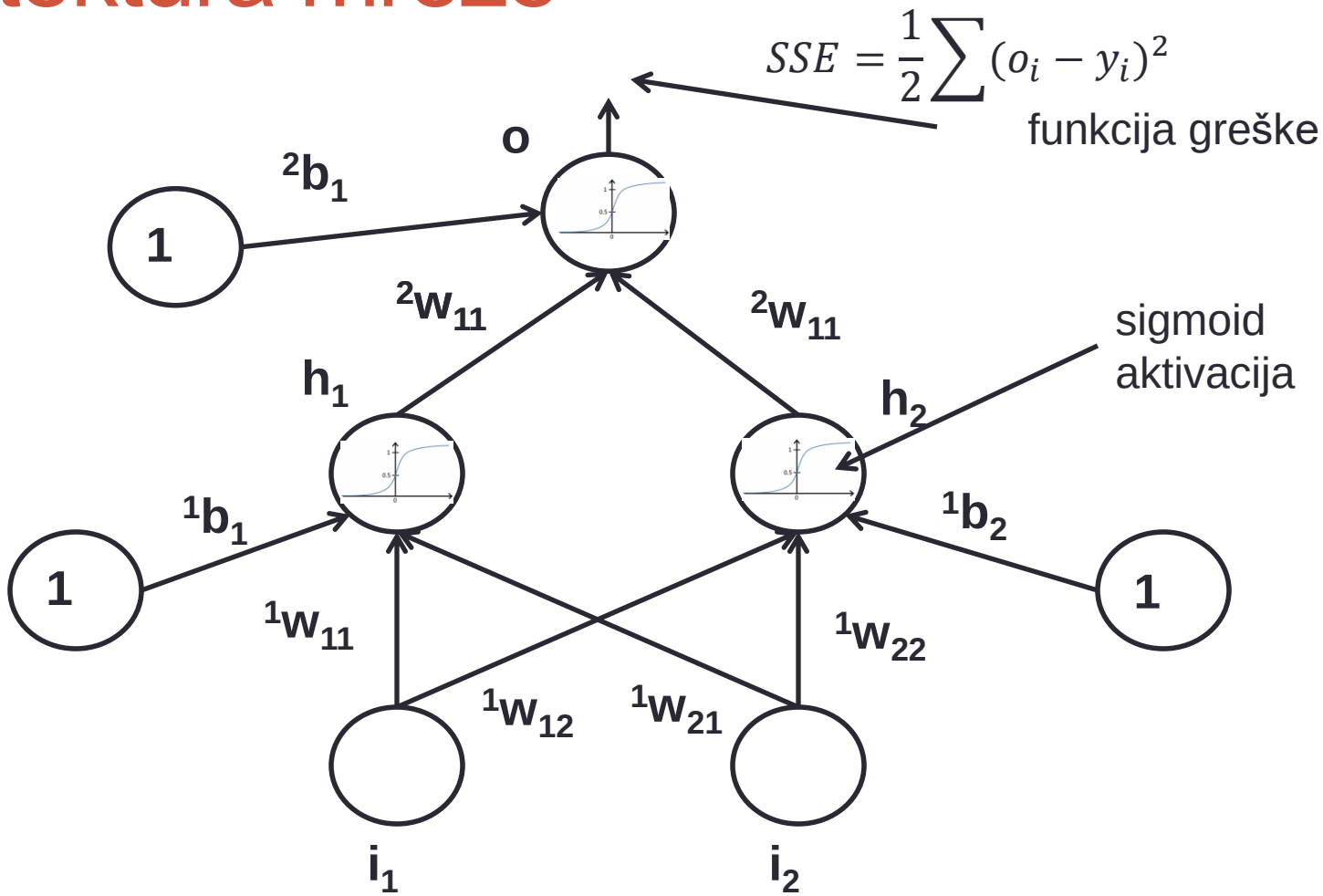
Arhitektura mreže



Arhitektura mreže



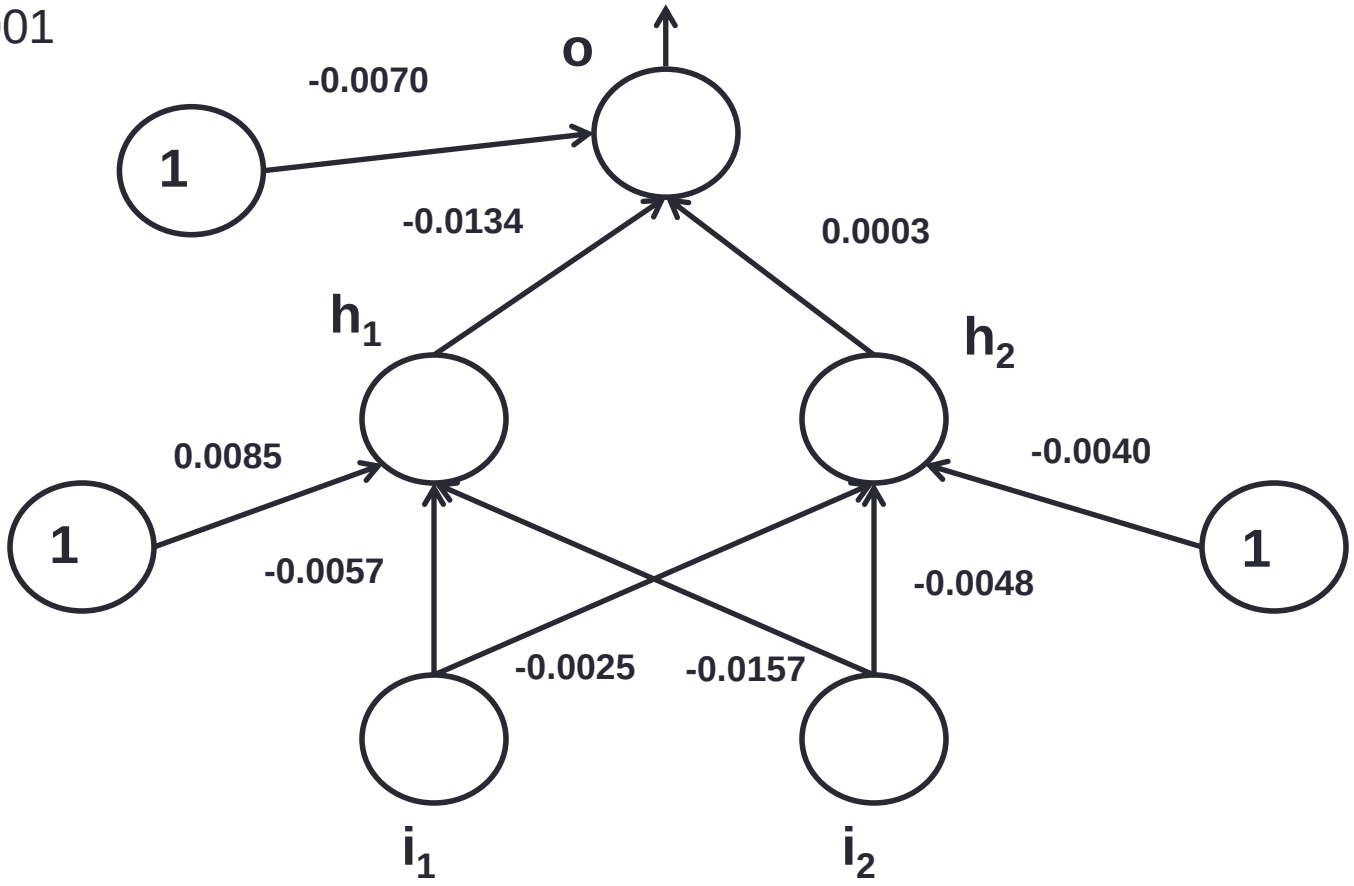
Arhitektura mreže



Inicijalizacija - Matlab

```
W1=randn(2,2)*0.001
W2=randn(1,2)*0.001
%bias, ulaz,... bice vektori kolone
b1=randn(2,1)*0.001
b2=randn(1,1)*0.001
```

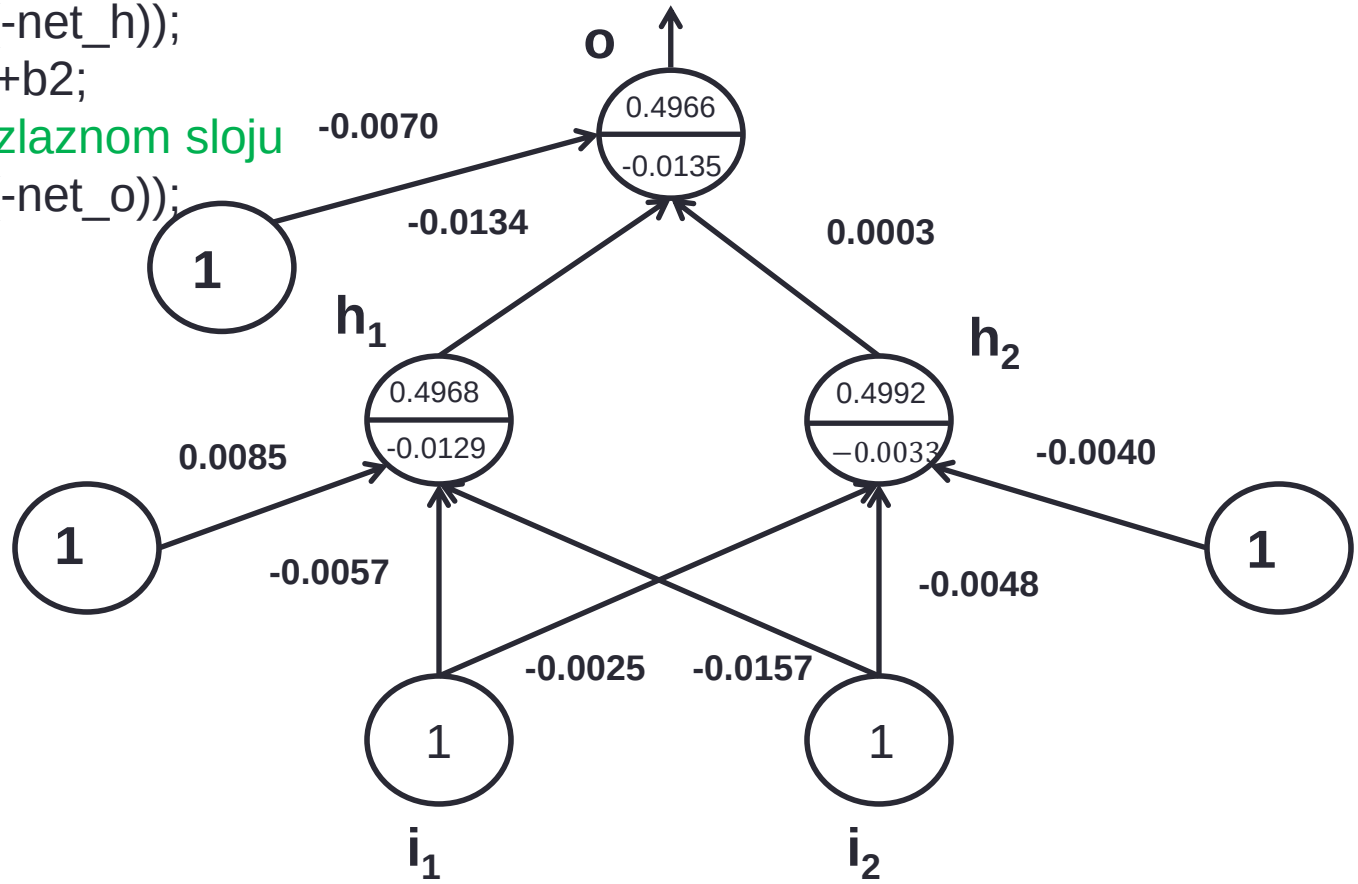
```
W1 =

   -0.0057   -0.0157
   -0.0025   -0.0048


W2 =

   -0.0134    0.0003


b1 =

    0.0085
    0.0040


b2 =

   -0.0070
```



Feed Forward – Izračunavanje unapred

```
%ulaz
input=[1;1]';
%ukupan ulaz u neurone u skrivenom sloju
net_h=W1*input+b;
%aktivacija neurona u skrivenom sloju
activation_h=1./(1+exp(-net_h));
net_o=W2*activation_h+b2;
%aktivacija neurona u izlaznom sloju
activation_o=1./(1+exp(-net_o));
```

```
net_h =
   -0.0129
   -0.0033

activation_h =
    0.4968
    0.4992

net_o =
   -0.0135

activation_o =
    0.4966
```



Feed Forward – Izračunavanje unapred

$$net_{h1} = i_1{}^1w_{11} + i_2{}^1w_{12} + {}^1b_1 = 1 * -0.0057 + 1 * -0.0157 + 0.0085 = -0.0129$$

$$a_{h1} = \frac{1}{1 + e^{-net_{h1}}} = 0.4968$$

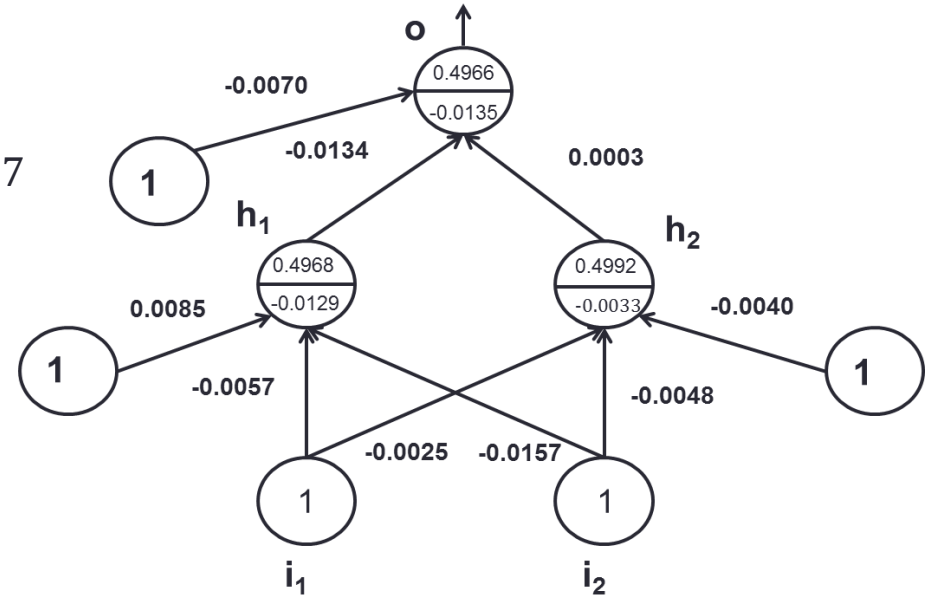
$$net_{h2} = i_1{}^1w_{21} + i_2{}^1w_{22} + {}^1b_1 = 1 * -0.0025 + 1 * -0.0048 - 0.0040 = -0.0033$$

$$a_{h2} = \frac{1}{1 + e^{-net_{h2}}} = 0.4992$$

$$net_0 = h_1{}^2w_{11} + h_2{}^2w_{12} + {}^2b_1 = -0.00225 * -0.0134 + -0.0048 * -0.0003 * -0.0040 = -0.0135$$

$$\mathrm{o} = \frac{1}{1 + e^{-net_o}} = 0.4966$$

$$SSE = \frac{1}{2}\sum(o_i - y_i)^2 = 0.5(1 - 0.4966)^2 = 0.1267$$



Backpropagation – Propagacija unazad

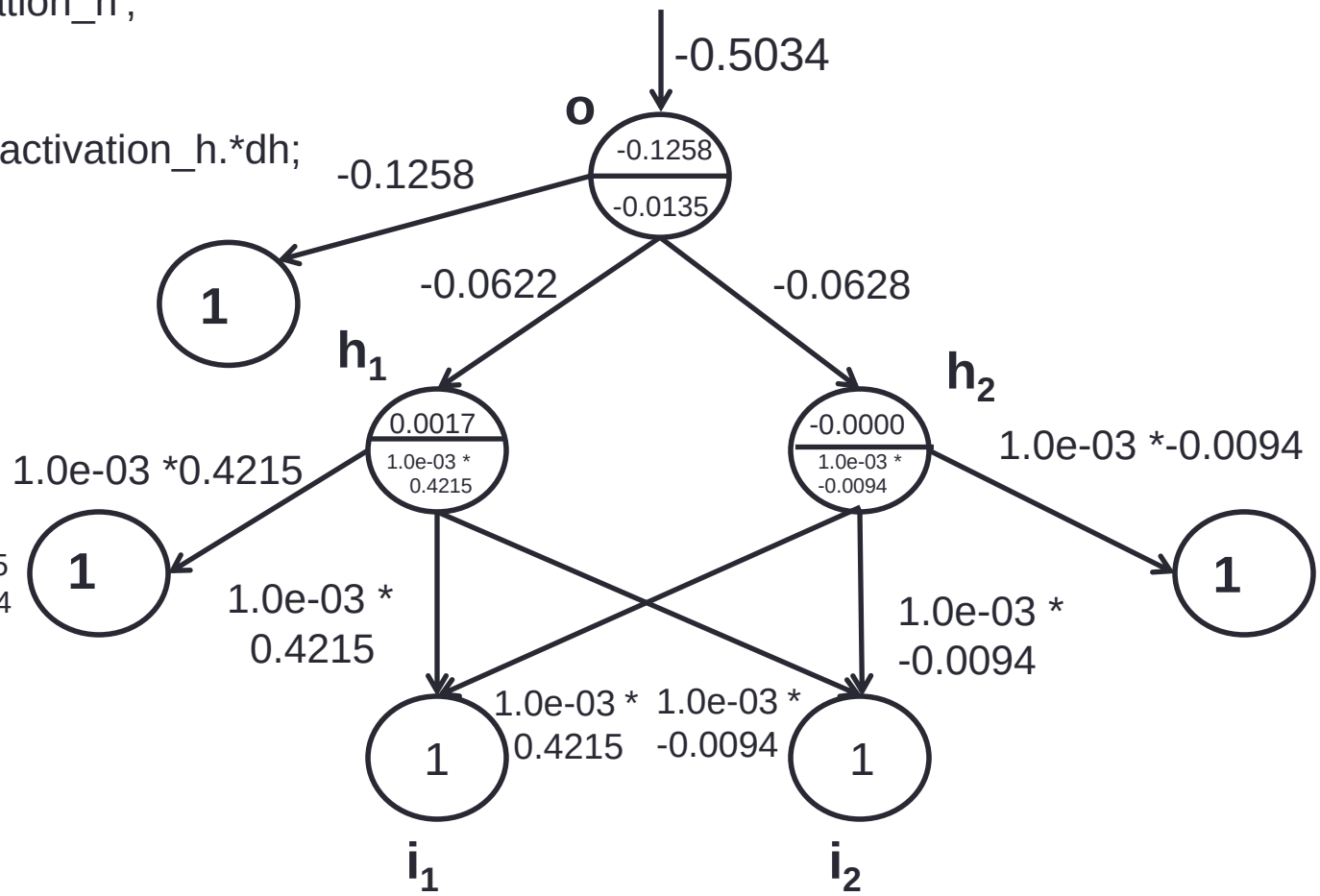
```
%tacna klasa je 1
error = 0.5*(activation_o-1)^2
%backpropagation
do = (activation_o-1);
do_activation = (1-activation_o)*activation_o*do;
dw2 = do_activation*activation_h';
db2 = do_activation;
dh = W2'*do_activation;
dh_akt = (1-activation_h).*activation_h.*dh;
dw1 = dh_akt*input'
db1 = dh_akt;
```

$$SSE = \frac{1}{2}\sum(o_i - y_i)^2$$

error = 0.1267

```
dw2 =

-0.0625   -0.0628

db2 = -0.1258

dh =
   0.0017
  -0.0000

dh_akt =
  1.0e-03 *

   0.4215
  -0.0094
```

```
dw1 =
   1.0e-03 *

    0.4215    0.4215
   -0.0094   -0.0094
db1 =
   1.0e-03 *

    0.4215
   -0.0094
```



Backpropagation – Propagacija unazad

$$derror = \underbrace{(o_i - yi)}_{\text{izvod SSE}} = 0.4966 - 1 = -0.5034$$

$$da_o = \underbrace{(1\text{-}a_o)*a_o}_{\text{izvod sigmoid funkcije}}*derror = (1\text{-}0.4966)*0.4966*(-0.5034) = -0.1258$$

Pogledati naredna dva slajda kao podsetnik.

$$dw^2 = da_o * ah^T = (-0.1258) * [0.4944 \quad 0.4988] = [-0.0622 \quad -0.0627] \qquad db^2 = da_o$$

$$dh = (w^2)^T * da_o = \begin{bmatrix} -0.0134 \\ 0.0003 \end{bmatrix} * (-0.1258) = \begin{bmatrix} 0.0017 \\ -0.0000 \end{bmatrix}$$

$$da_h = (1 - a_h) * a_h * dh = (1 - 0.4944) * 0.4944 * \begin{bmatrix} 0.0017 \\ -0.0000 \end{bmatrix} = \begin{bmatrix} 0.4215 \\ -0.0094 \end{bmatrix} * 10^{-3}$$

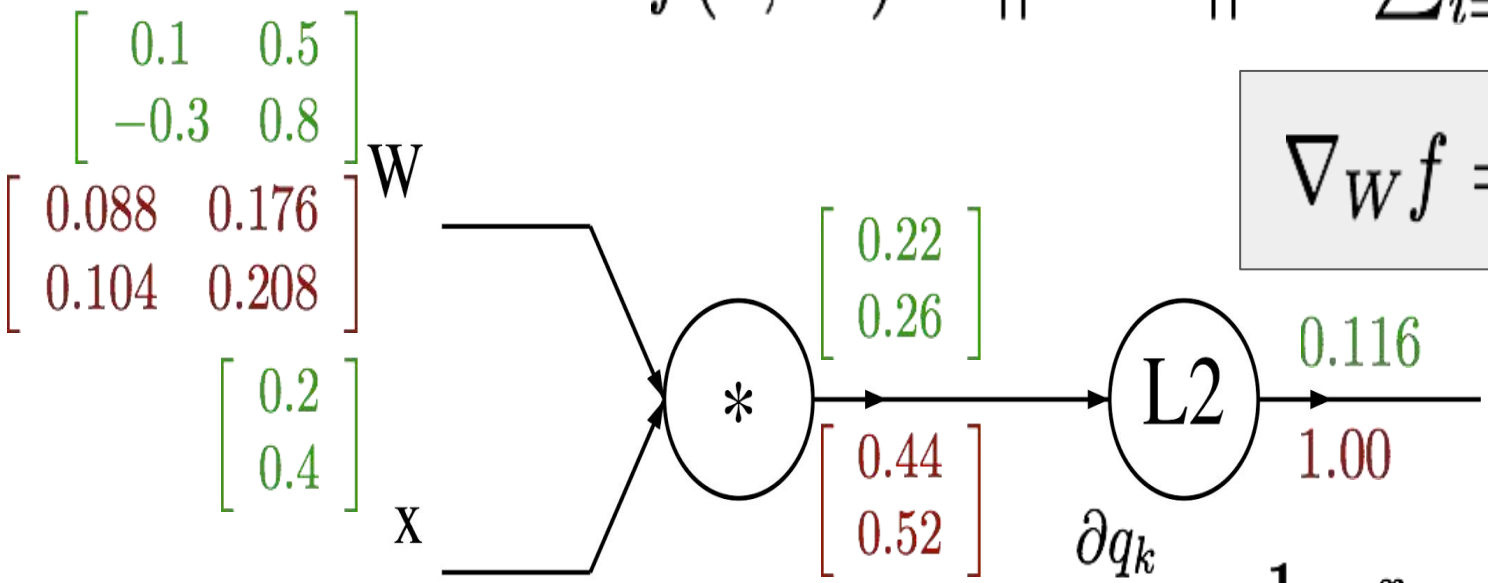
$$dw^1 = dah \begin{bmatrix} i_1 \\ i_2 \end{bmatrix}^T = \begin{bmatrix} 0.4215 \\ -0.0094 \end{bmatrix} * 10^{-3} * [1 \quad 1] = \begin{bmatrix} 0.4215 & 0.4215 \\ -0.0094 & -0.0094 \end{bmatrix} * 10^{-3}$$

$$db^1 = da_h = \begin{bmatrix} 0.4215 \\ -0.0094 \end{bmatrix} * 10^{-3}$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$

$$\nabla_W f = 2q \cdot x^T$$



$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

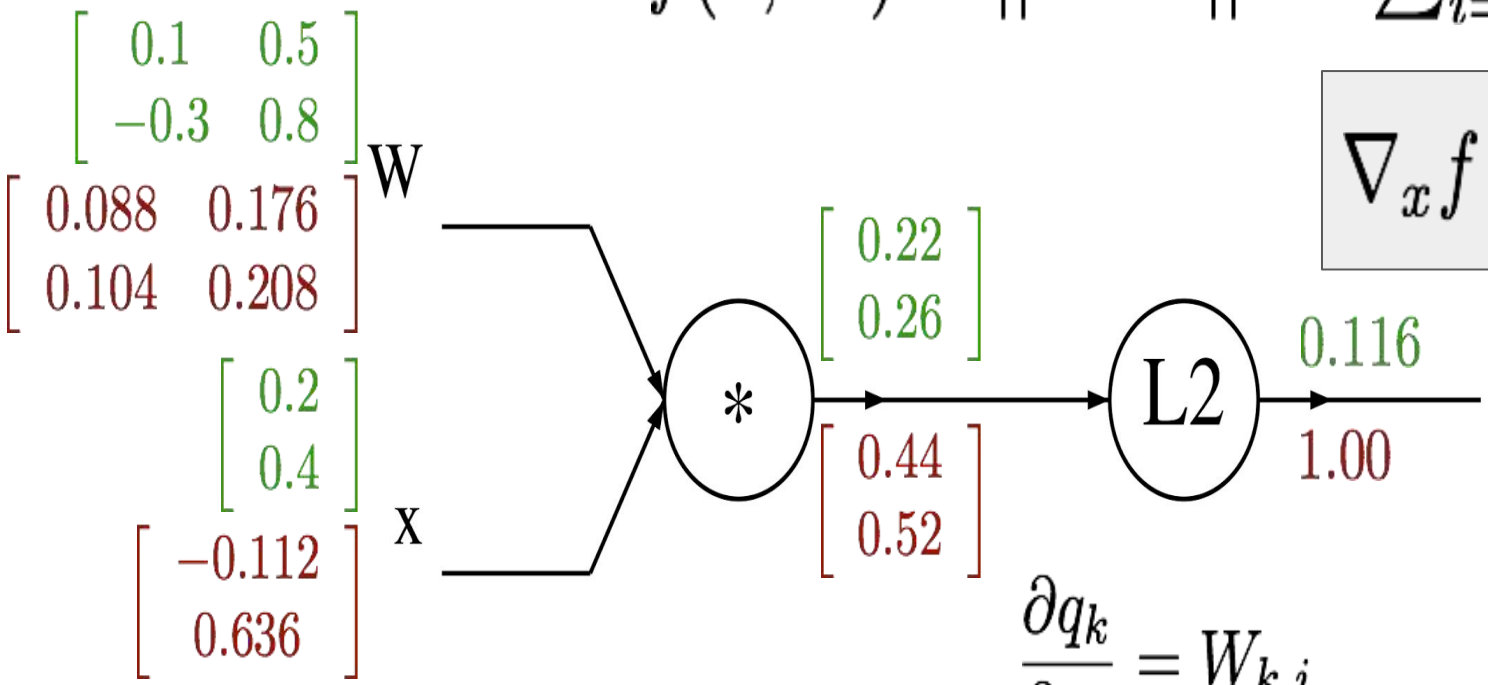
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$= 2q_i x_j$$

Primer sa vektorima

$$f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$$

$$\nabla_x f = 2W^T \cdot q$$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i}$$

$$\frac{\partial f}{\partial x_i} = \sum_k 2 q_k W_{k,i}$$

Promena vrednosti težina i biasa – Gradijentni Spust

*nova_vrednost_parametra = stara_vrednost – learning_rate*gradijent_parametra*

- *Learning rate* je 0.1

W2 = W2 - 0.1*dw2;

b2 = b2 - 0.1*db2;

W1 = W1 - 0.1*dw1;

b1 = b1 - 0.1*db1;

Promena vrednosti težina i biasa – Gradijentni Spust

- Nakon jedne korekcije parametara dobijamo novi izlaz za ulaz [1 1], kao i novu grešku:

activation_o = 0.5013
error = 0.1243

- Vidimo da je vrednost izlaza veća (cilj nam je da bude 1), a vrednost greške manja (cilj nam je da bude 0).

- Ako prethodno opisani postupak obučavanja mreže ponovimo određeni broj itreacija greška će vremenom pasti na 0 ili veoma malu vrednost.

Obučavanje Neuronskih Mreža
Prvi Deo

Predavač: Aleksandar Kovačević

Slajdovi preuzeti sa CS 231n, Stanford

http://cs231n.stanford.edu/

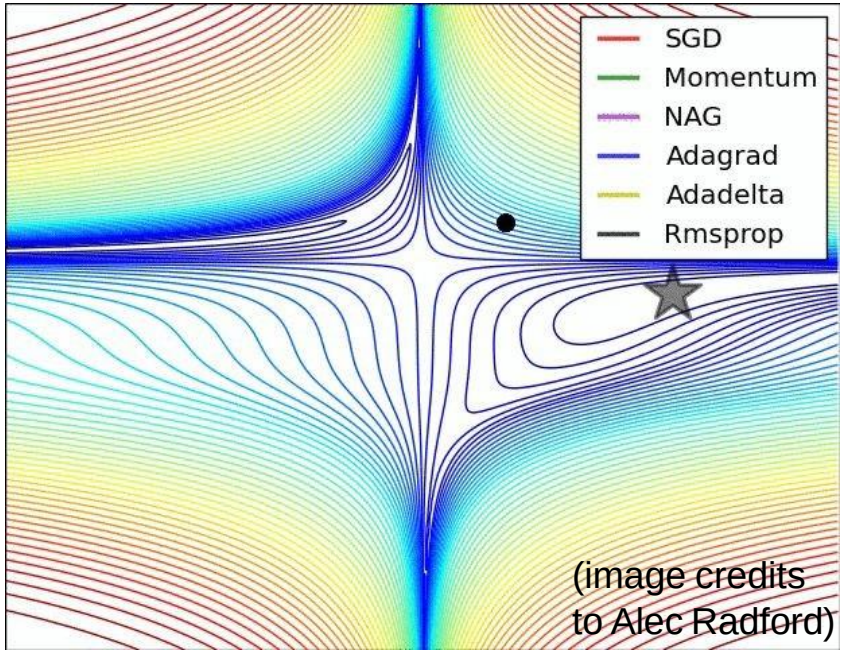
Šta smo do sada naučili...

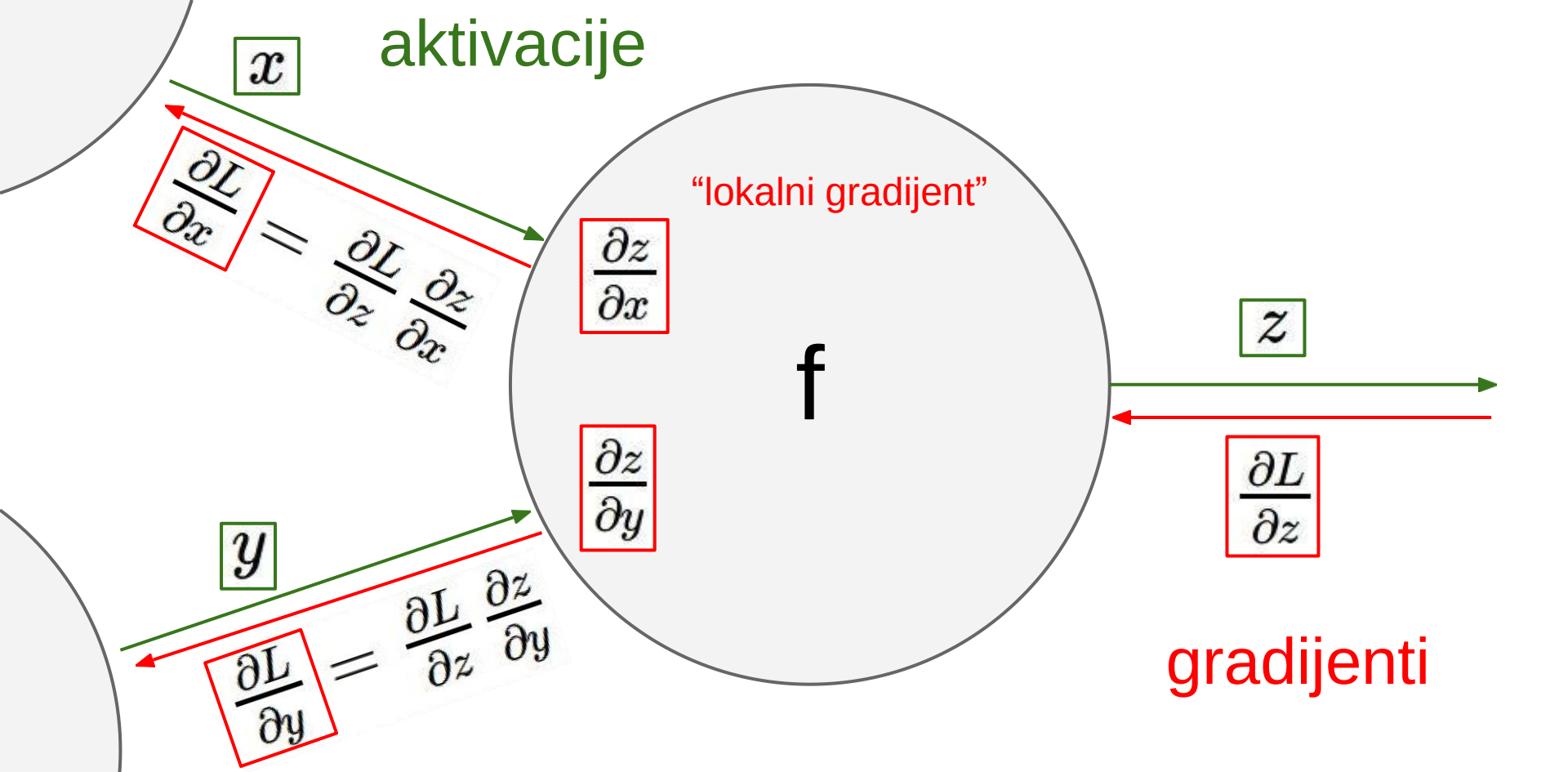
Stohastički Gradijentni Spust sa Mini-Podskupovima (*Mini-batch SGD*)

Loop:

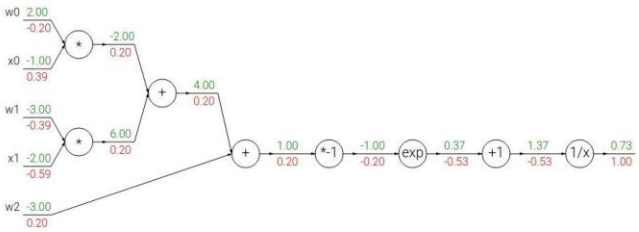
1. Semplujemo podskup obučavajućeg skupa
2. Za svaki primer iz podskupa
    1. izračunavamo izlaz iz mreže („guramo“ ga unapred kroz mrežu – feed forward)
    2. kada dobijemo izlaz za primer, izračunamo vrednost funkcije greške i tu vrednost dodajemo na ukupnu grešku za taj podskup
    3. izračunavamo gradijente za svaki parametar pomoću Backpropagation. Za svaki parametar dodajemo gradijent na zbir gradijenata koje on čuva.
3. Izračunavamo prosek gradijenta za svaki parametar. Prosek je zbir gradijenata (dobijen iz koraka 2.3) za ceo podskup podeljen sa veličinom podskupa.
4. Menjamo vrednosti parametara pomoću proseka gradijenta koji mu odgovara.

Šta smo do sada naučili...




aktivacije
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"lokalni gradijent"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradijenti

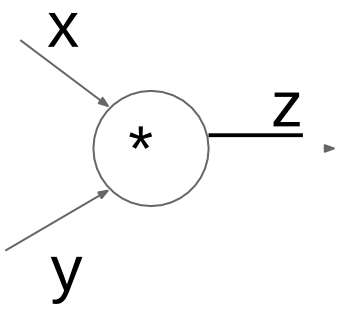
Implementacija: forward/backward API



Graf (ili Net) objekat. *(Pseudo kod)*

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

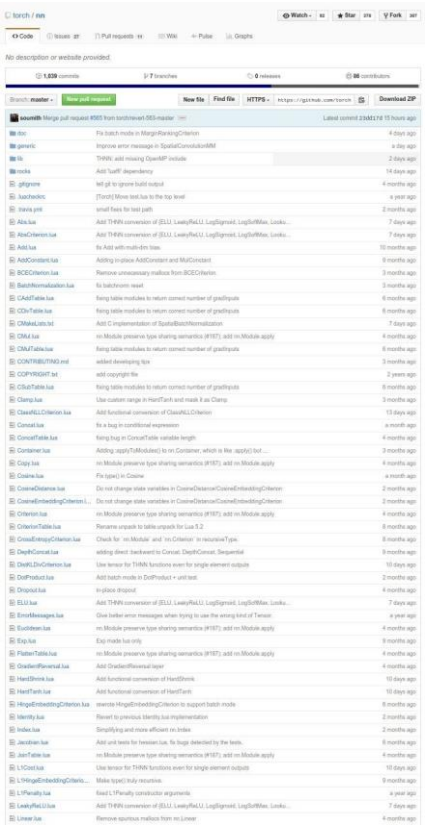
Implementacija: forward/backward API

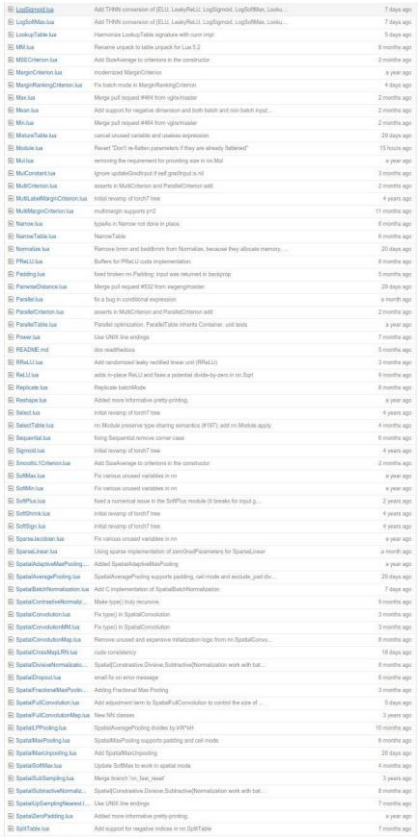



```python
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

(x,y,z su skalari)

Primer: Torch Slojevi





=

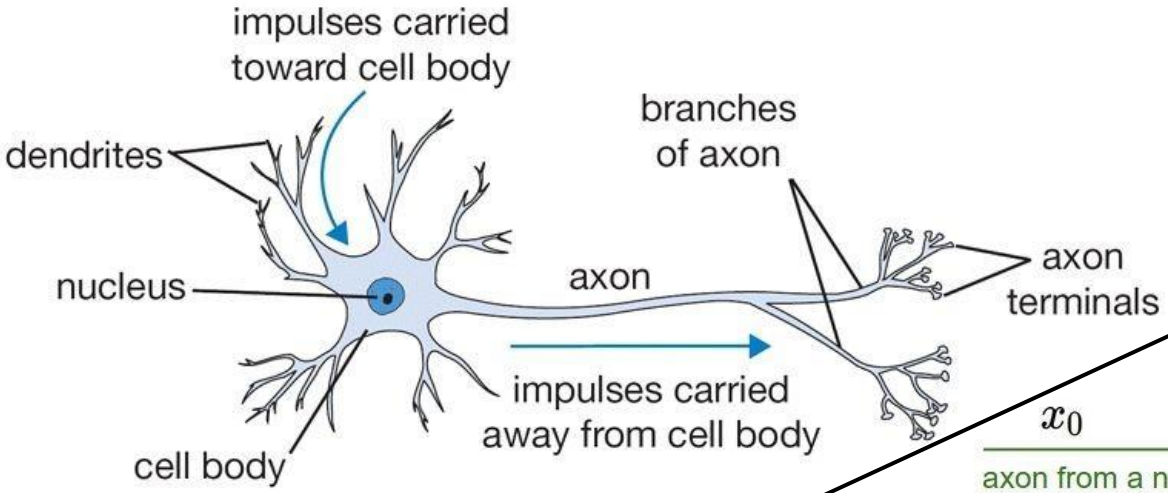
Neuronske Mreže: prvo bez analogije sa ljudskim mozgom

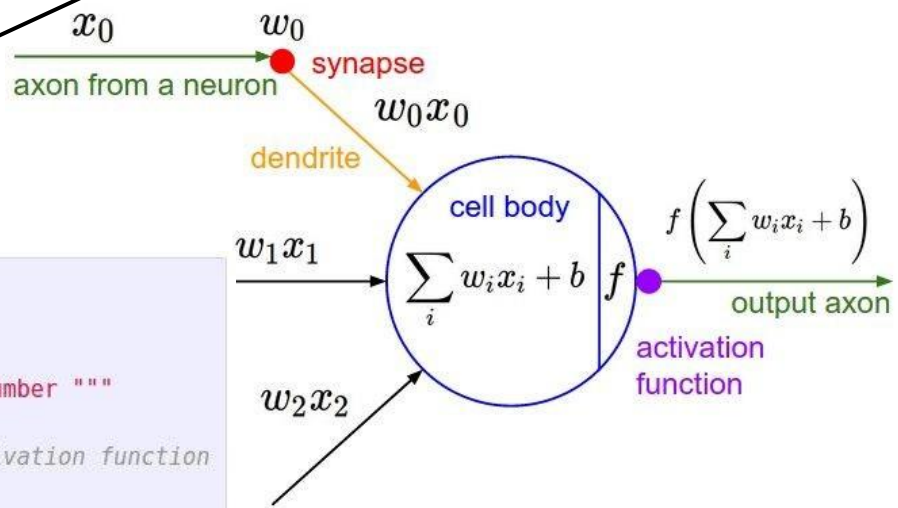
(**Ranije**) Linearna skor funkcija: $$f = Wx$$

(**Sada**) Neuroska mreža sa dva sloja: $$f = W_2 \max(0, W_1 x)$$

Neuroska mreža sa tri sloja:

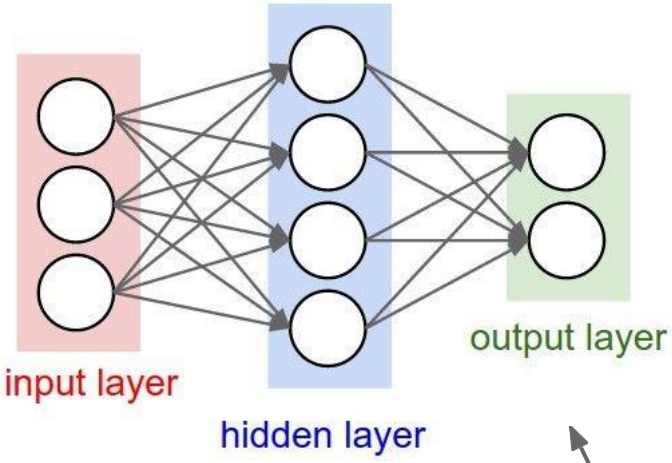
$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

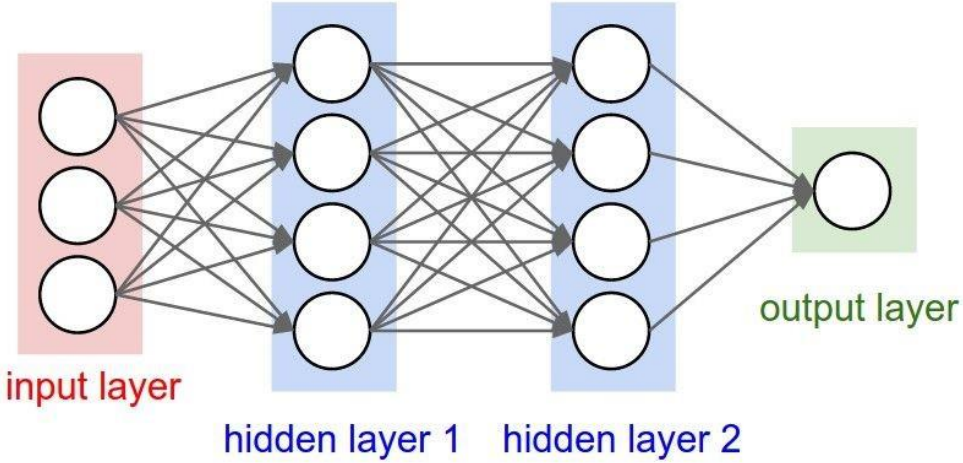





```python
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

Neuronske Mreže: Arhitekture





“2-slojna NN”, ili
“Neuronska mreža koja ima jedan skriveni sloj”

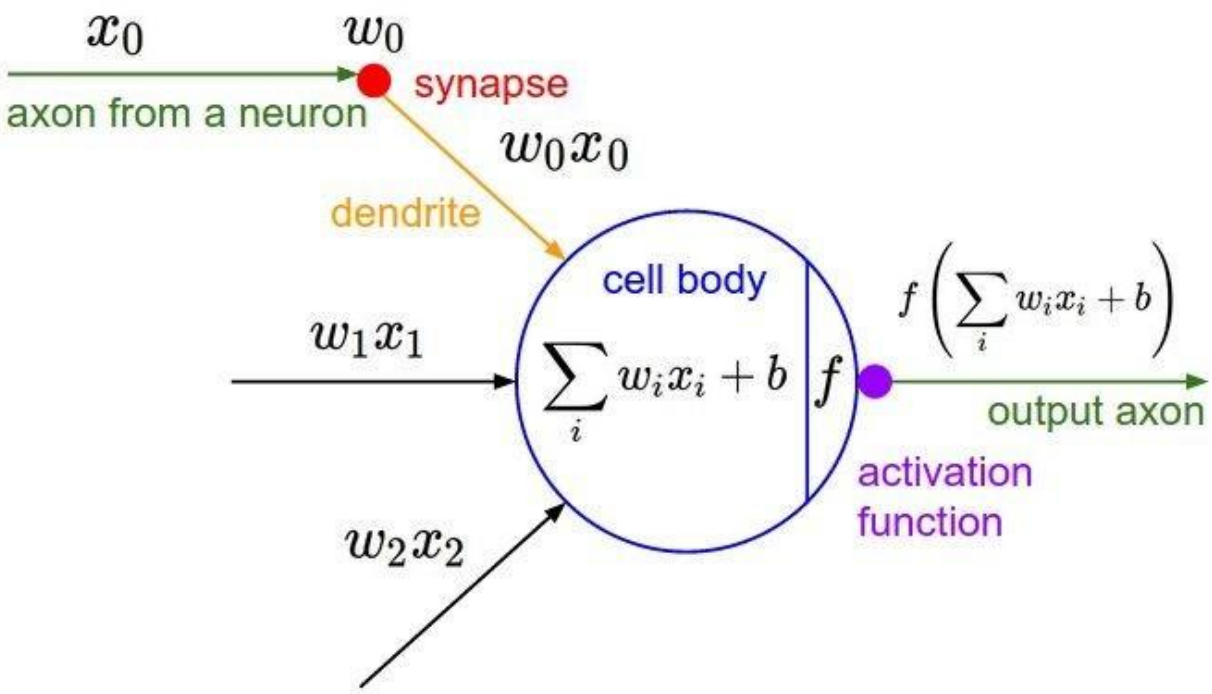
“3-slojna NN”, ili
“Neuronska mreža koja ima dva skrivena sloja”

“Potpuno-povezani” slojevi

Obučavanje Neuronskih Mreža

Funkcije Aktivacije

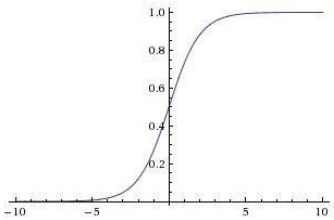
Funkcije Aktivacije



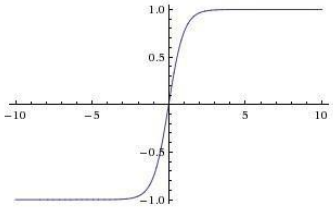
Funkcije Aktivacije

Sigmoid

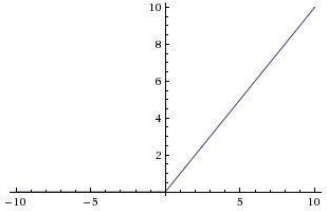
$$\sigma(x) = 1/(1+e^{-x})$$



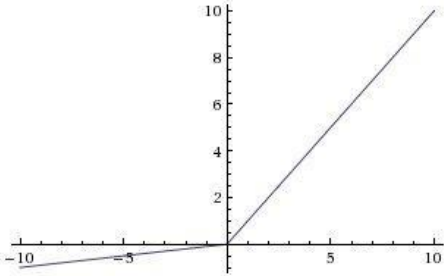
tanh tanh(x)



ReLU max(0,x)

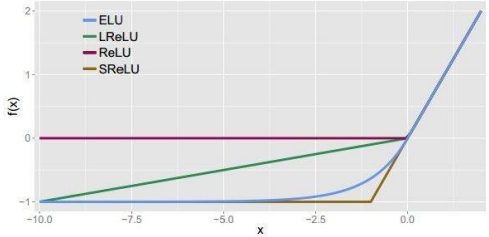


Leaky ReLU
max(0.1x, x)



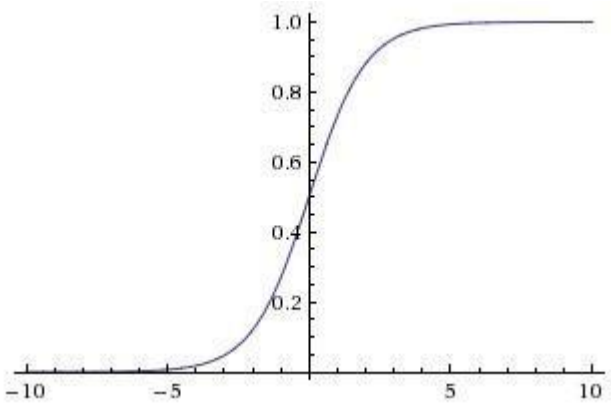
Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$



Funkcije Aktivacije

$$\sigma(x) = 1/(1 + e^{-x})$$

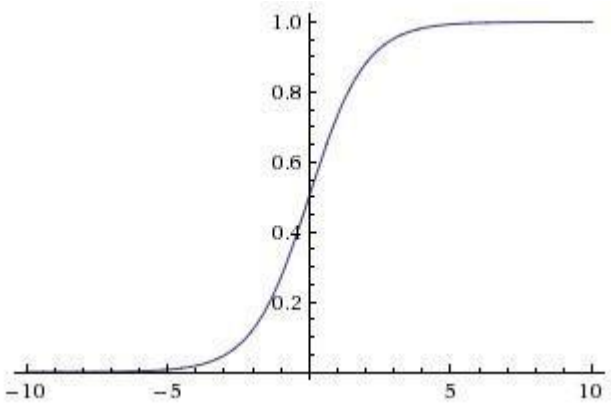


Sigmoid

- „Skuplja“ ulaz na raspon [0,1]
- Istorijski najpopularniji izbor za aktivaciju

Funkcije Aktivacije

$$\sigma(x) = 1/(1 + e^{-x})$$

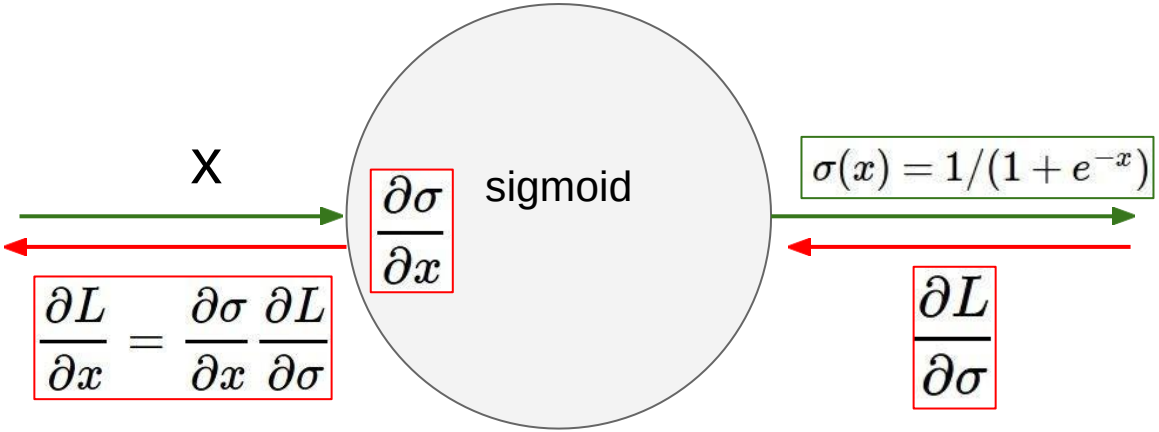


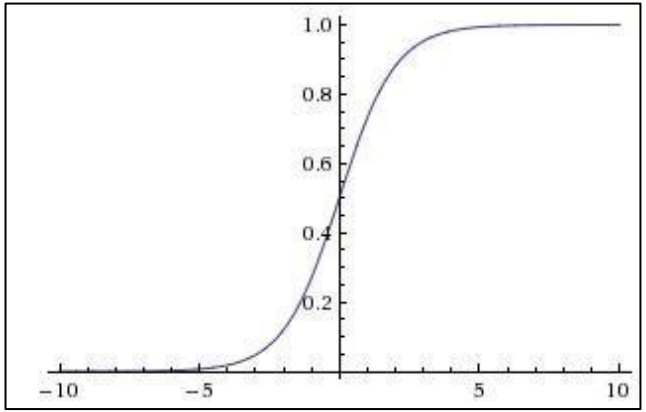
Sigmoid

- „Skuplja“ ulaz na raspon [0,1]
- Istorijski najpopularniji izbor za aktivaciju

Tri Problema:

1. Neuroni u saturaciji (jako male ili velike vrednosti) „ubijaju“ gradijent

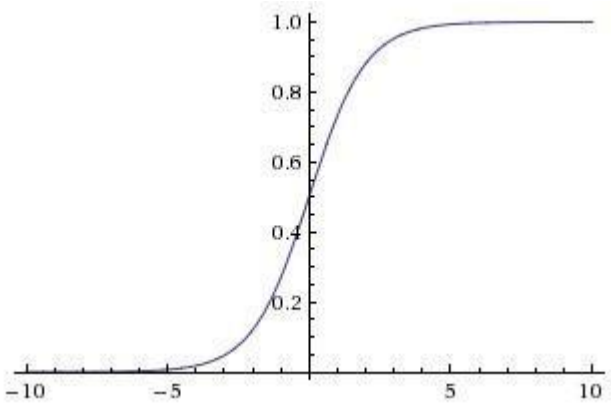




Šta se dešava kad je x = -10?
Šta se dešava kad je x = 0?
Šta se dešava kad je x = 10?

Funkcije Aktivacije

$$\sigma(x) = 1/(1 + e^{-x})$$



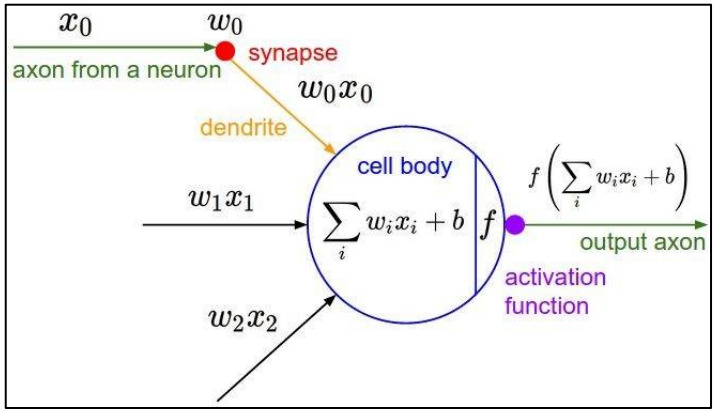
Sigmoid

- „Skuplja“ ulaz na raspon [0,1]
- Istorijski najpopularniji izbor za aktivaciju

Tri Problema:

1. Neuroni u saturaciji (jako male ili velike vrednosti) „ubijaju“ gradijent
2. Izlaz nema srednju vrednost 0 tj. nije *zero-centered*

Šta se dešava kad su svi ulazi u neuron (x) pozitivni:



$$f\left(\sum_i w_i x_i + b\right)$$

Kako tada izgledaju gradijenti **w**?

Zašto su gradijenti *w* svi uvek pozivini ili uvek negativni:

Recimo imamo neuron sa sigmoid aktivacionom funkcijom i da “u njega” dolazi gradijent $\frac{\delta L}{\delta \sigma}$, gde je σ izlaz iz sigmoid funkcije. Neka je mrežni ulaz u neuron: $f = Wx + b$. Onda važi:

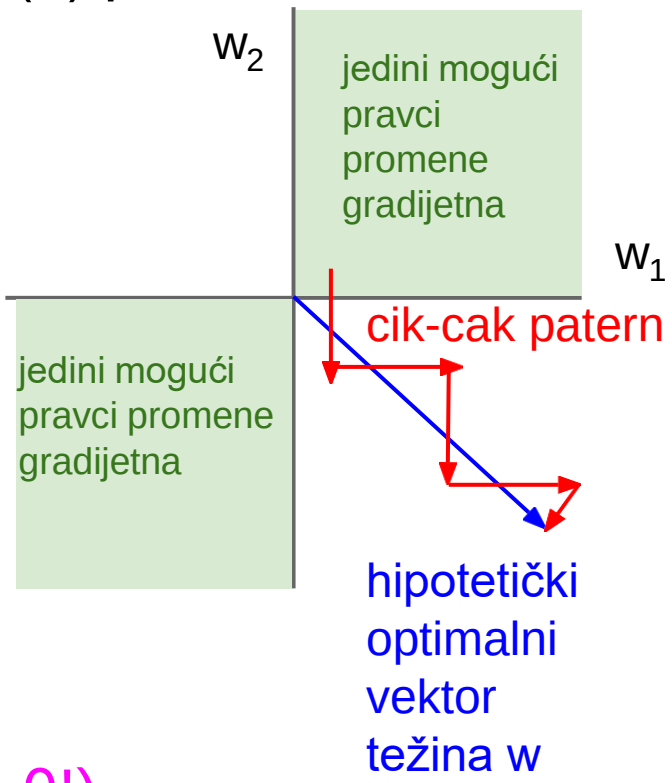
$$\frac{\delta L}{\delta w} = \frac{\delta L}{\delta \sigma}\frac{\delta \sigma}{\delta f}\frac{\delta f}{\delta w} = \frac{\delta L}{\delta \sigma}(1-\sigma)\sigma\frac{\delta f}{\delta w} = \frac{\delta L}{\delta \sigma}(1-\sigma)\sigma x$$

Pošto je izlaz sigmoida između 0 i 1 onda je $(1-\sigma)\sigma$ uvek pozitivno (ili nula).

Ako su svi ulazi x>0 onda znak gradijenta $\frac{\delta L}{\delta w}$ zavisi samo od znaka $\frac{\delta L}{\delta \sigma}$ i svi gradijenti za matricu težina imaju isti znak kao $\frac{\delta L}{\delta \sigma}$.

Šta se dešava kad su svi ulazi u neuron (x) pozitivni:

$$f\left(\sum_i w_i x_i + b\right)$$



Kako tada izgledaju gradijenti **w**?

Svi su uvek pozitivni ili su svi negativni :(

(zato želimo da su vrednosti centrirane oko 0!)

Šta se dešava kad su svi ulazi u neuron (x) pozitivni:

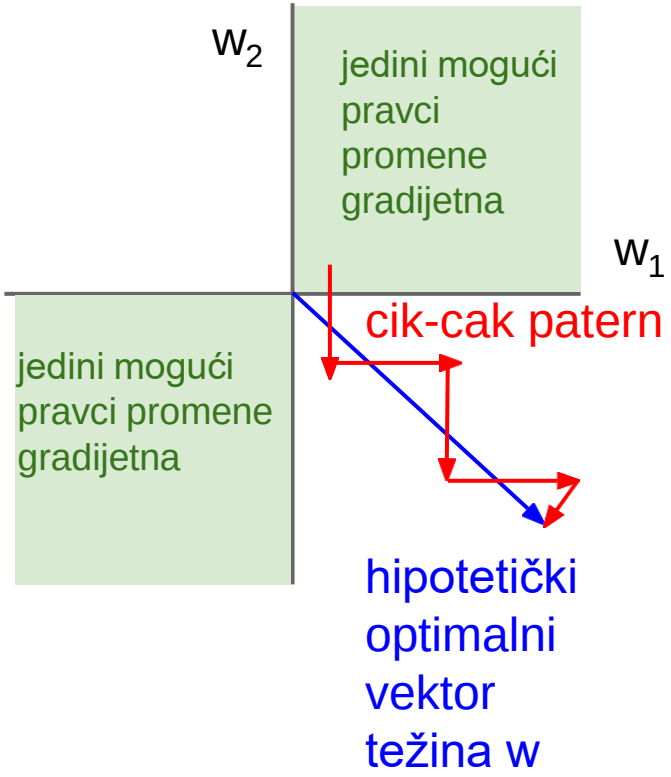
Da bi pronašli hipotetički optimalni w sa slike, gradijenti spust mora u isto vreme da povećava w_1 i smanjuje w_2.

To neće biti moguće jer mu gradijenti to ne dozvoljavaju pošto su istog znaka.

To će onda rezultovati time da će gradijenti spust menjati w_1 i w_2 jedno po jedno.

Prvo će malo da smanji w_2, pa posle poveća w_1.

Tako dobijamo cik-cak patern.



Šta se dešava kad su svi ulazi u neuron (x) pozitivni:

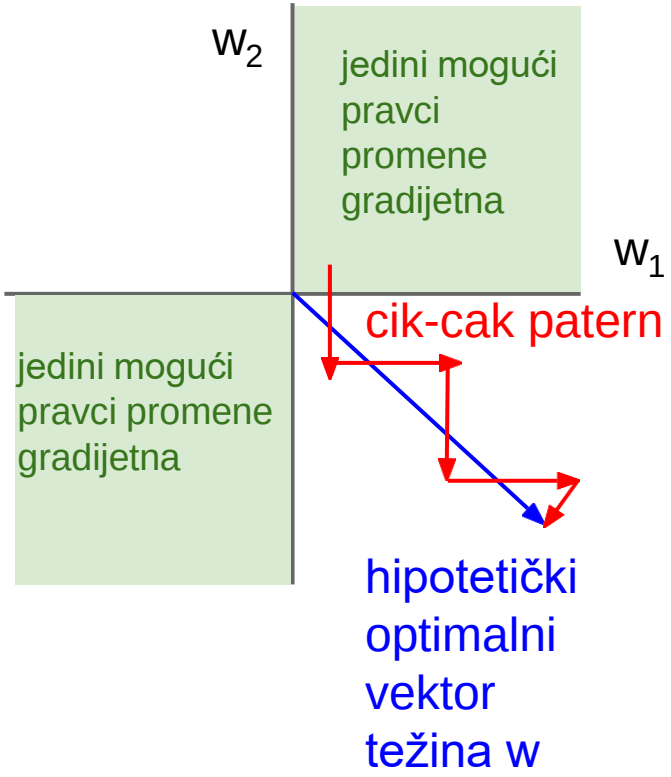
Šta dobijamo ako bi izlaz funkcije aktivacije bio centriran oko 0?

Znači da bi onda neki izlazi bili negativni, a neki pozitivni.

To znači da bi onda ulazi u neke od narednih neurona bili malo negativni malo pozitvni.

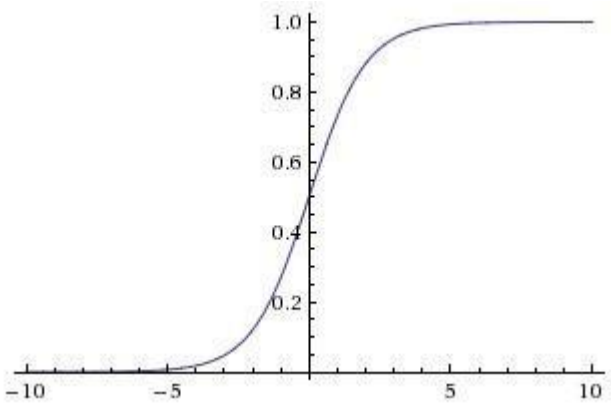
To bi nam onda omogućilo da gradijenti težina budu malo negativni malo pozitivni.

Tako bi izbegli cik-cak patern jer bi mogli da u isto vreme jednu težinu povećavamo, a drugu smanjujemo i da se tako krećemo do idealnog w.



Funkcije Aktivacije

$$\sigma(x) = 1/(1 + e^{-x})$$



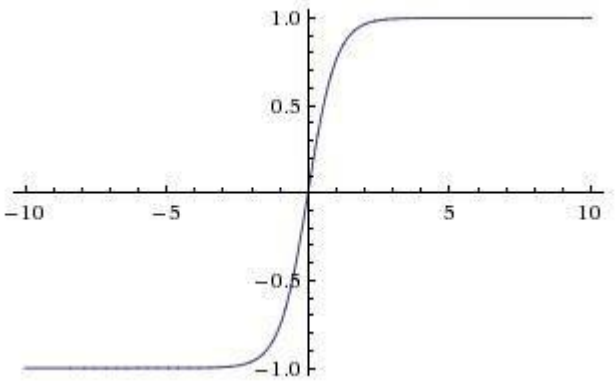
Sigmoid

- „Sabija“ ulaz na raspon [0,1]
- Istorijski najpopularniji izbor za aktivaciju

Tri Problema:

1. Neuroni u saturaciji (jako male ili velike vrednosti) „ubijaju“ gradijent
2. Izlaz nema srednju vrednost 0 tj. nije *zero-centered*
3. Računanje *exp* može biti skupo

Funkcije Aktivacije

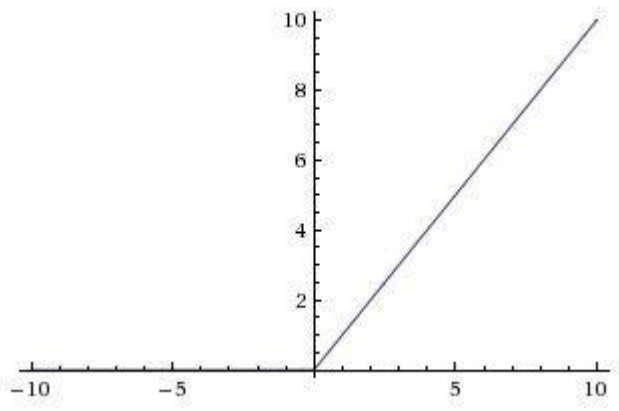


tanh(x)

- „Sabija“ ulaz na raspon [-1,1]
- izlaz je centiran oko 0
- i dalje „ubija“ gradjent prilikom staturacije :(

[LeCun et al., 1991]

Funkcije Aktivacije



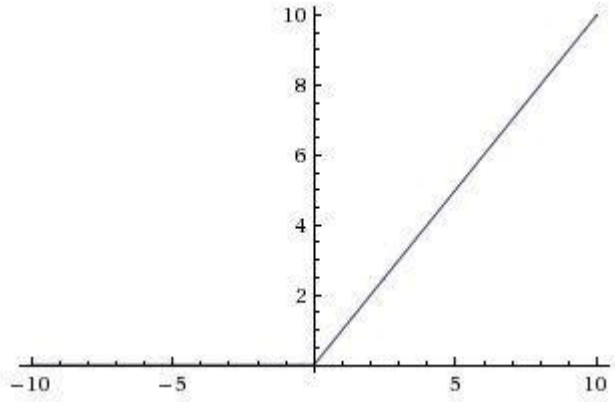
ReLU
(Rectified Linear Unit)

- Izračunava **f(x) = max(0,x)**

- Nema stauracije (u + regionu)
- Vrlo brzo izračunavanje
- Praksa je pokazala da GD mnogo brže konvergira nego kad koristimo sigmoid ili tanh (npr. nekad i do 6x brže)

[Krizhevsky et al., 2012]

Funkcije Aktivacije



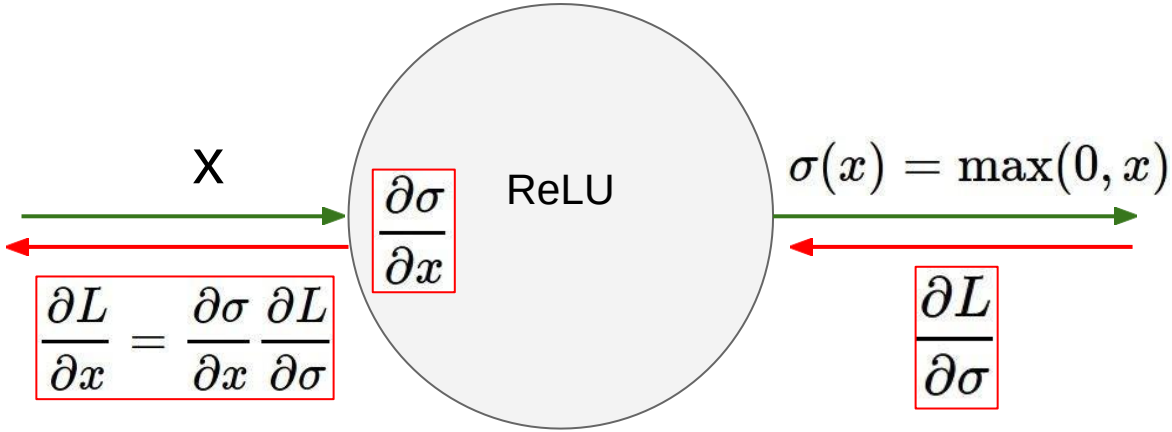
ReLU
(Rectified Linear Unit)

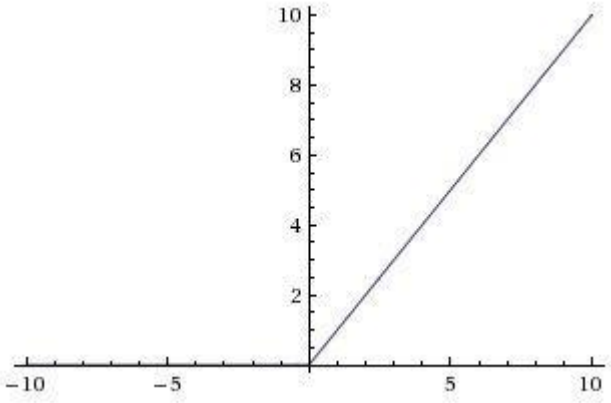
- Izračunava **f(x) = max(0,x)**

- Nema saturacije (u + regionu)
- Vrlo brzo izračunavanje
- Praksa je pokazala da GD mnogo brže konvergira nego kad koristimo sigmoid ili tanh (npr. nekad i do 6x brže)

- Izlaz nije centriran oko 0
- Problem:

koji je gradijent kad je x < 0?

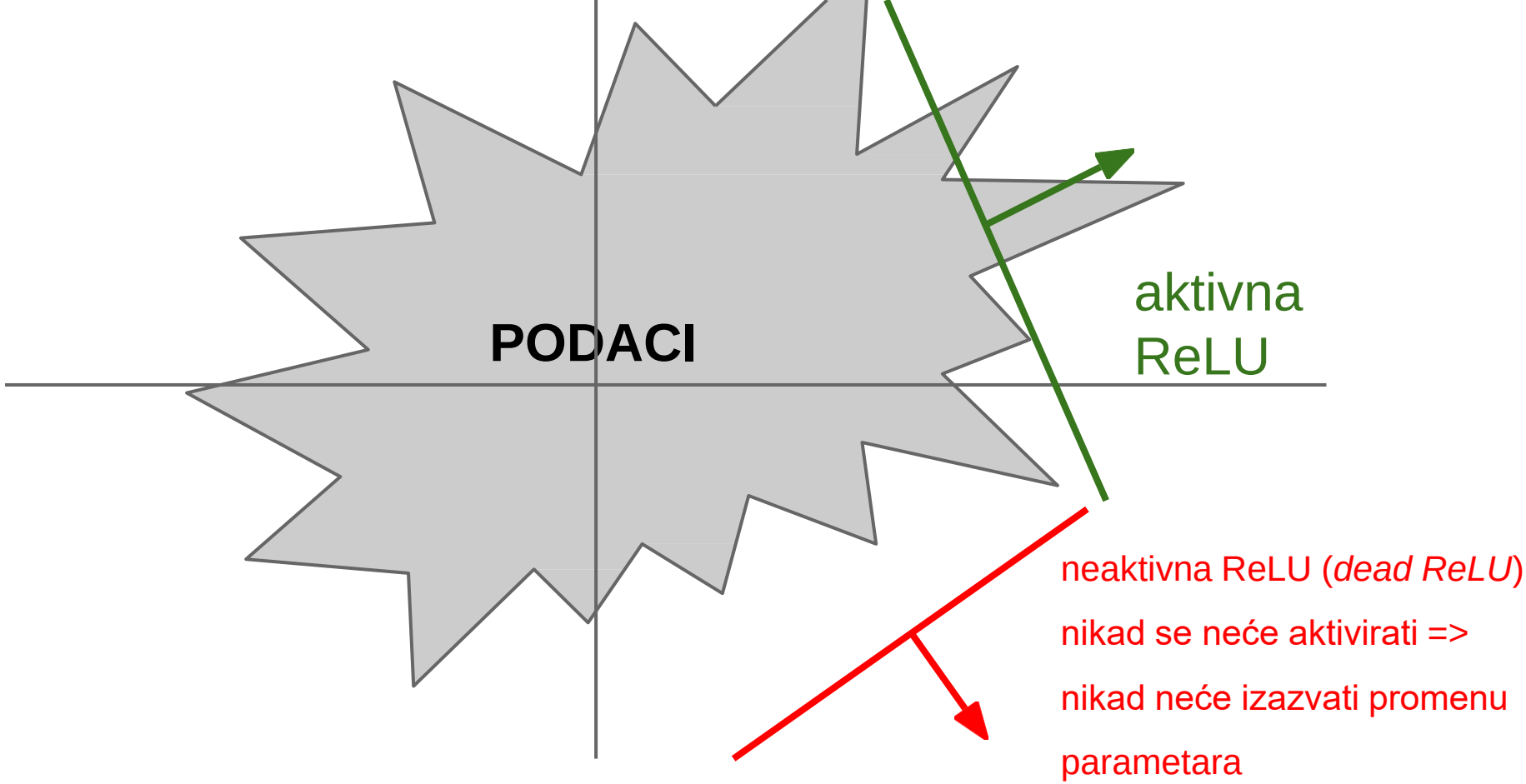


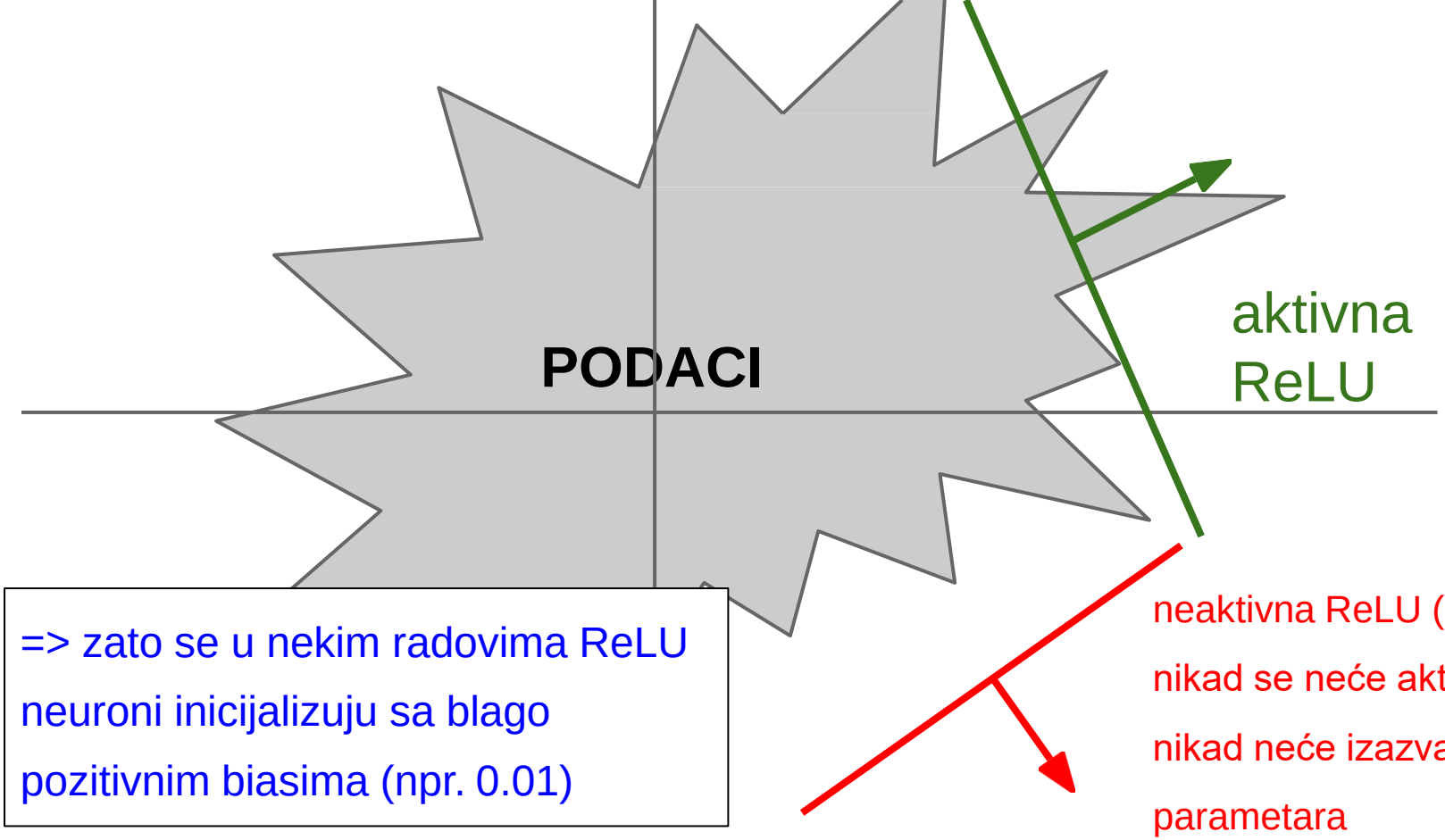


Šta se dešava kad je x = -10?

Šta se dešava kad je x = 0?

Šta se dešava kad je x = 10?

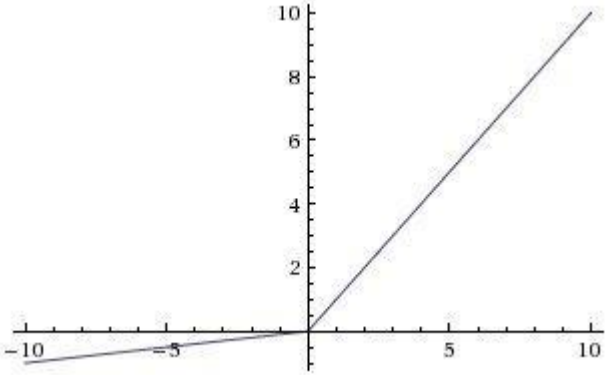

PODACI
aktivna
ReLU
neaktivna ReLU (*dead ReLU*)
nikad se neće aktivirati =>
nikad neće izazvati promenu
parametara



=> zato se u nekim radovima ReLU neuroni inicijalizuju sa blago pozitivnim biasima (npr. 0.01)

Funkcije Aktivacije

[Mass et al., 2013]
[He et al., 2015]



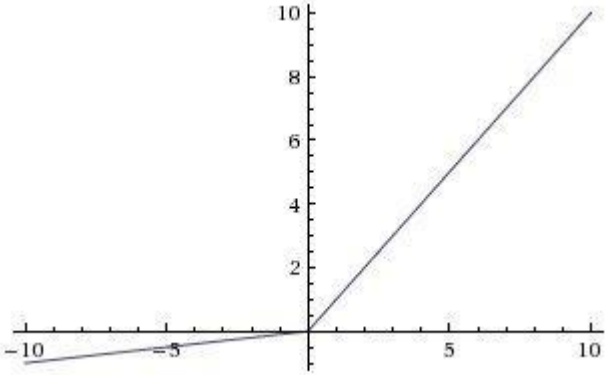
Leaky ReLU

$$f(x) = \max(0.01x, x)$$

- Nema stauracije (u + regionu)
- Vrlo brzo izračunavanje
- Praksa je pokazala da GD mnogo brže konvergira nego kad koristimo sigmoid ili tanh (npr. nekad i do 6x brže)
- **neće “umreti”.**

Funkcije Aktivacije

[Mass et al., 2013]
[He et al., 2015]



- Nema stauracije (u + regionu)
- Vrlo brzo izračunavanje
- Praksa je pokazala da GD mnogo brže konvergira nego kad koristimo sigmoid ili tanh (npr. nekad i do 6x brže)
- **neće "umreti".**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

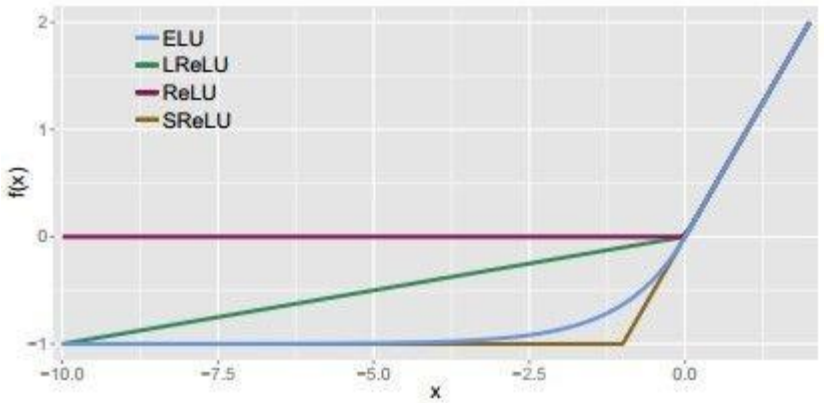
$$f(x) = \max(\alpha x, x)$$

koristimo backprop da odredimo parametar alfa kao i sve ostale

Funkcije Aktivacije

[Clevert et al., 2015]

Exponential Linear Units (ELU)



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

- Sve dobre strane ReLU
- Ne „umire“
- Izlaz bliži centriranosti oko 0

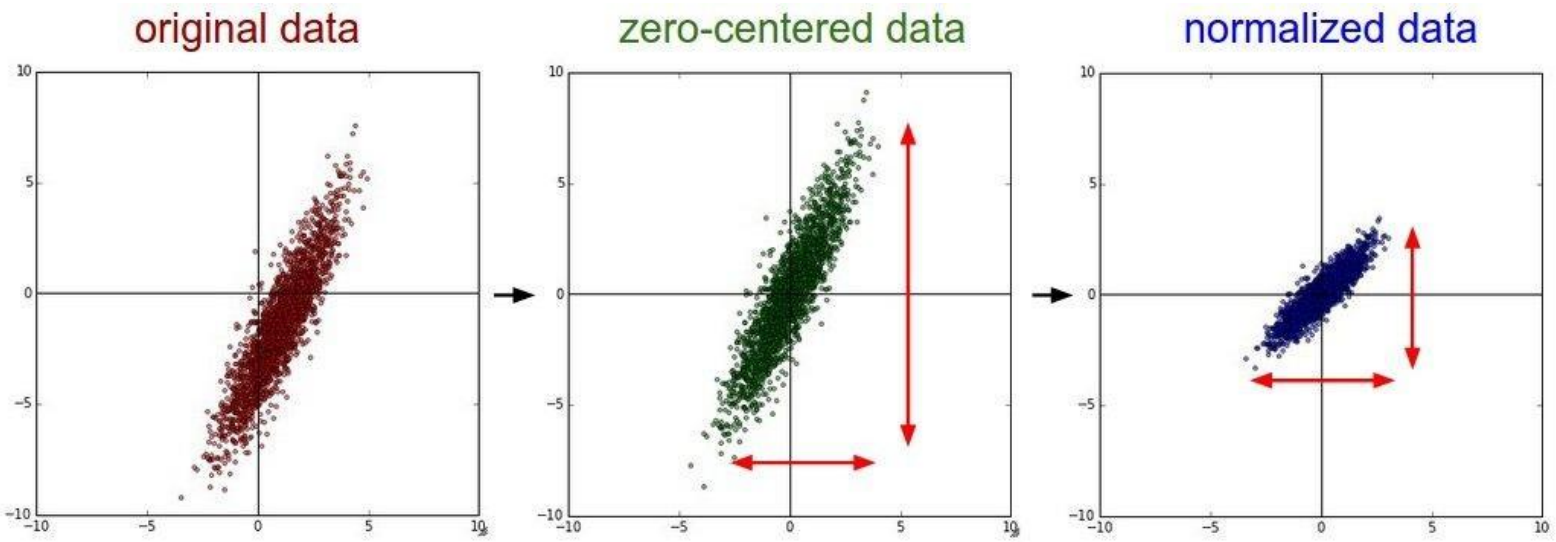
- Moramo da izračunavamo exp()

TLDR: U praksi:

- Koristite ReLU. Korak učenja (*learning rate*) je nešto na šta treba obratiti pažnju
- Probajte Leaky ReLU / Maxout / ELU
- Probajte tanh, ali ne očekujte puno
- Ne koristite sigmoid

Predprocesiranje podataka
Data Preprocessing

Korak 1: Predprocesirati podatke




```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

(X [NxD] je matrica podataka, svaki primer je jedan red)

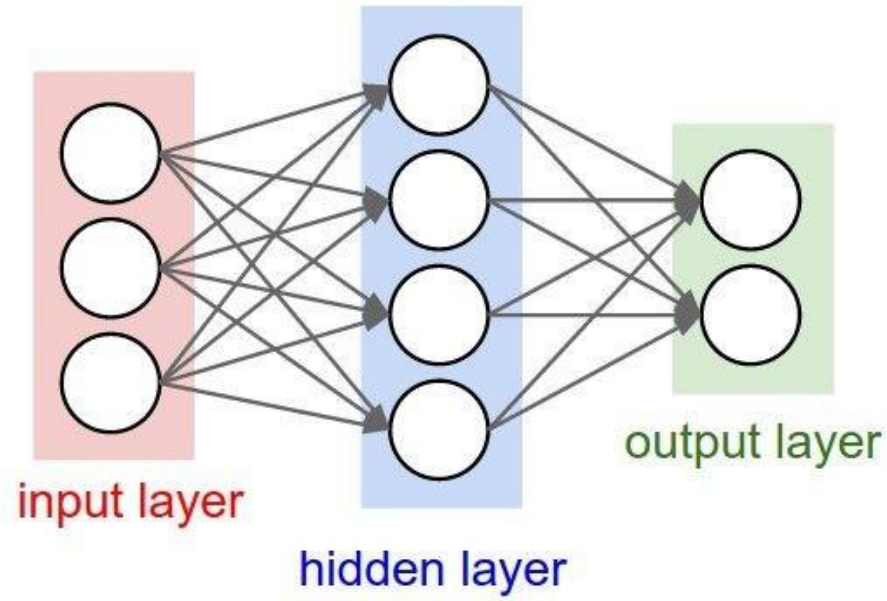
TLDR: U praksi, za slike: samo centriranje

Npr. CIFAR-10 skup podataka sa [32,32,3] slikama

- Oduzimamo srednju sliku (npr. AlexNet) (za svaki piksel srednja vrednost po celom skupu podataka = srednja slika [32,32,3])
- Oduzimamo srednju sliku po svakoj boji (npr. VGGNet) (srednja vrednost svih piksela u celom skupu podataka za svaku boju R, G, B posebno = 3 broja)

Inicijalizacija težina

- Šta se dešava kada je W=0 na početku?



- Prva ideja: **Mali slučajno odabrani brojevi**

(Gausijana sa srednjom vrednosti 0 i standardnom devijacijom od 1e-2)

```
W = 0.01* np.random.randn(D,H)
```

- Prva ideja: **Mali slučajno odabrani brojevi**

(Gausijana sa srednjom vrednosti 0 i standardnom devijacijom od 1e-2)

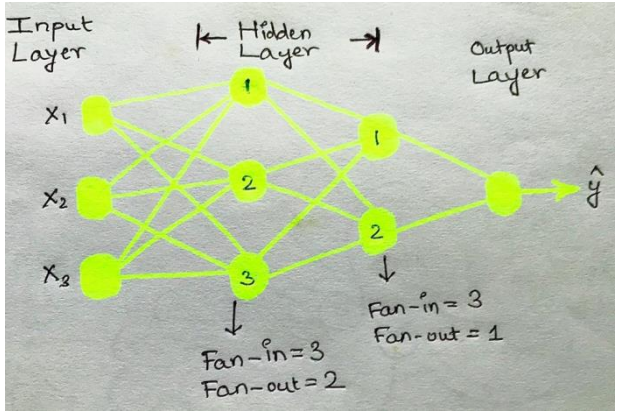
```
W = 0.01* np.random.randn(D,H)
```

Radi dobro za manje mreže. Kod većih mreža aktivacije brzo postaju previše male da bi backprop mogao da bilo šta uradi.

Pogledajmo statistike aktivacija za jednu mrežu

Npr. Mreža od 10 slojeva sa 500 neurona u svakom sloju, koristi se *tanh* i inicijializacija kao na prethodnom slajdu

Značenja *fan_in* i *fan_out*:

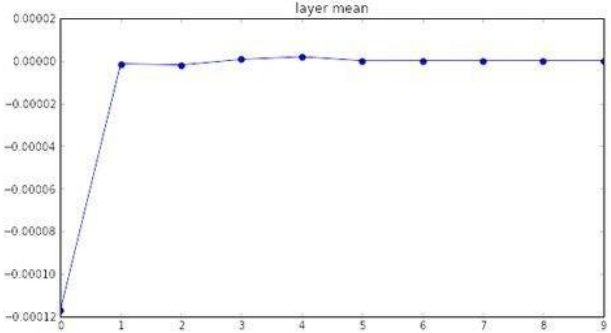


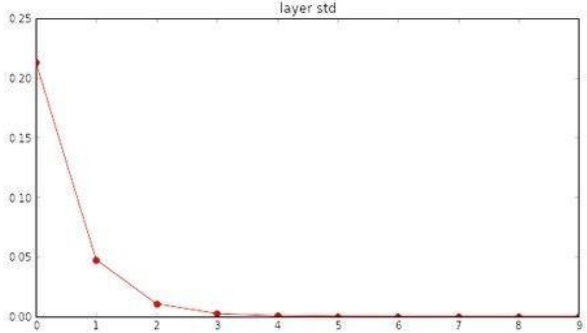

```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)
```

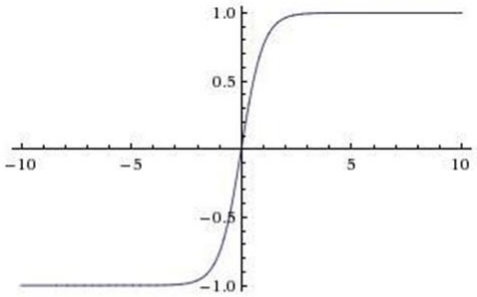
```
act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

    H = np.dot(X, W) # matrix multiply
    H = act[nonlinearities[i]](H) # nonlinearity
    Hs[i] = H # cache result on this layer
```

```
# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' % (i+1, layer_means[i], layer_stds[i])

# plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')

# plot the raw distributions
plt.figure()
for i,H in Hs.iteritems():
    plt.subplot(1,len(Hs),i+1)
    plt.hist(H.ravel(), 30, range=(-1,1))
```

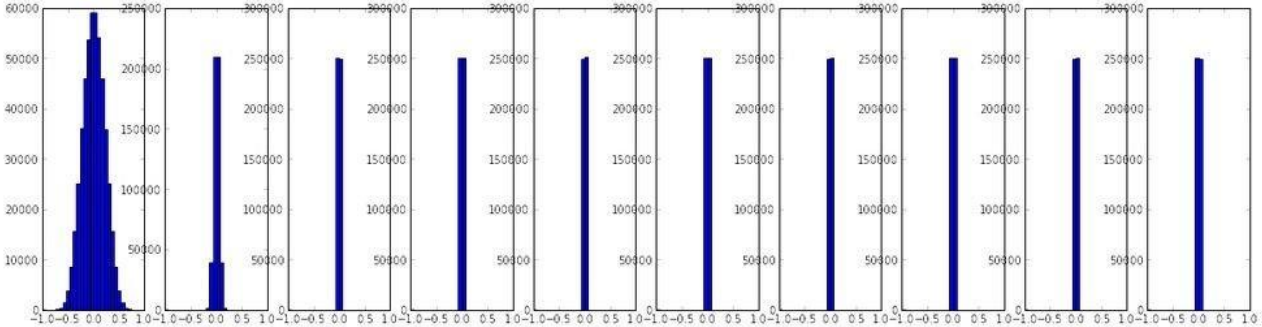
```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```



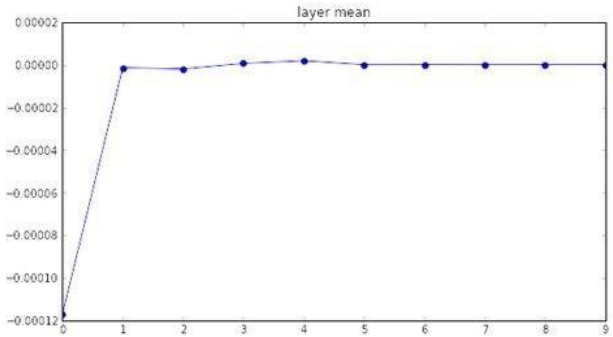


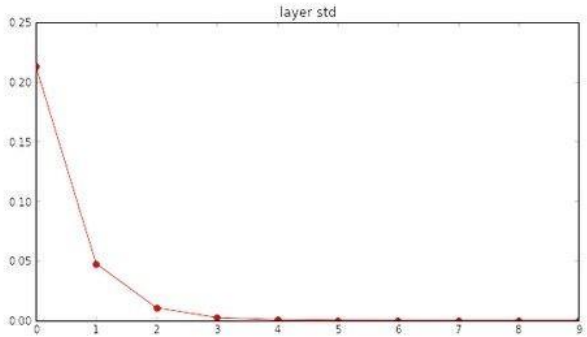


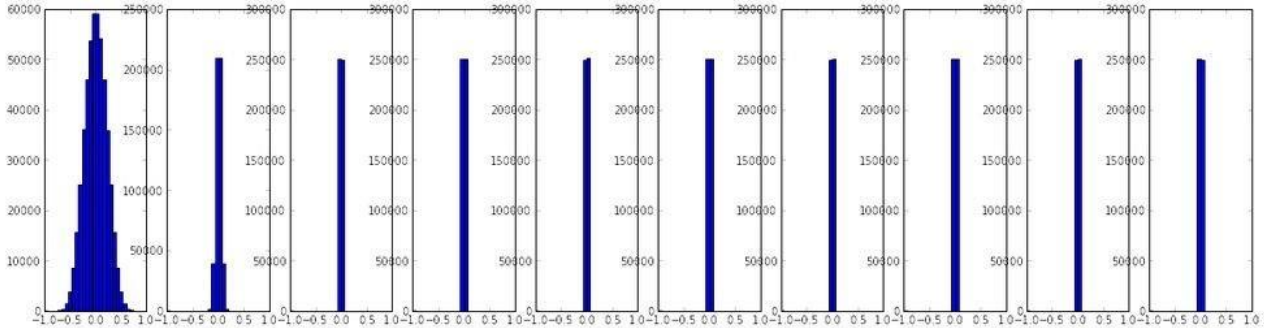
tanh(x)



```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```







Sve aktivacije postaju 0!

Šta se onda dešava sa gradijentima?

Pomoć: razmislite o propagaciji unazad kroz W*X deo.

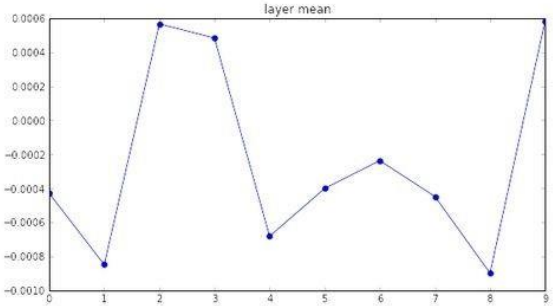
```
W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization
```

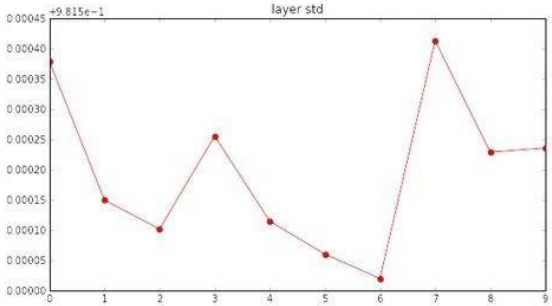
```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```

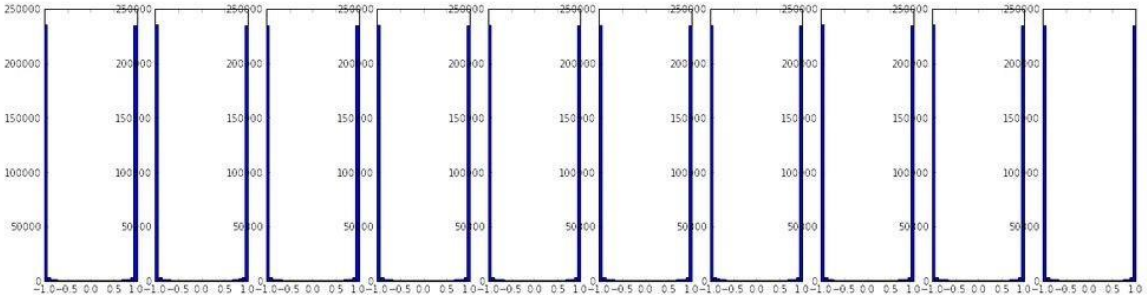
*1.0 umesto *0.01

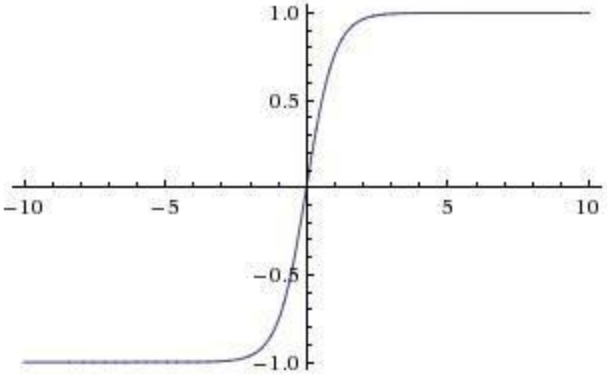
(Gausijana sa srednjom vrednosti 0 i standardnom devijacijom 1)

Skoro svi neuroni su saturirani, vrednosti su -1 ili 1. Gradijenti će svi biti 0.









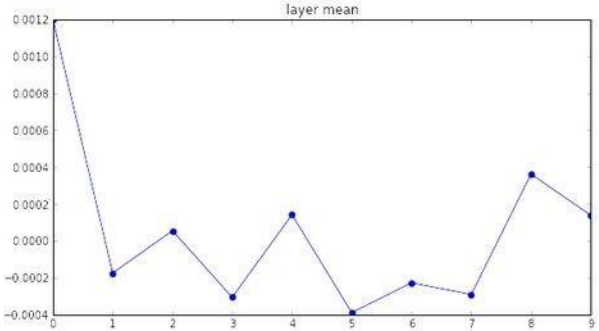
tanh(x)

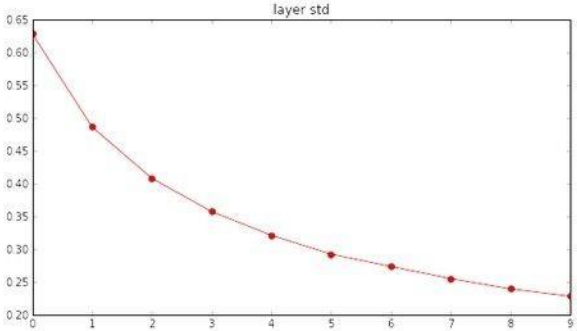
```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

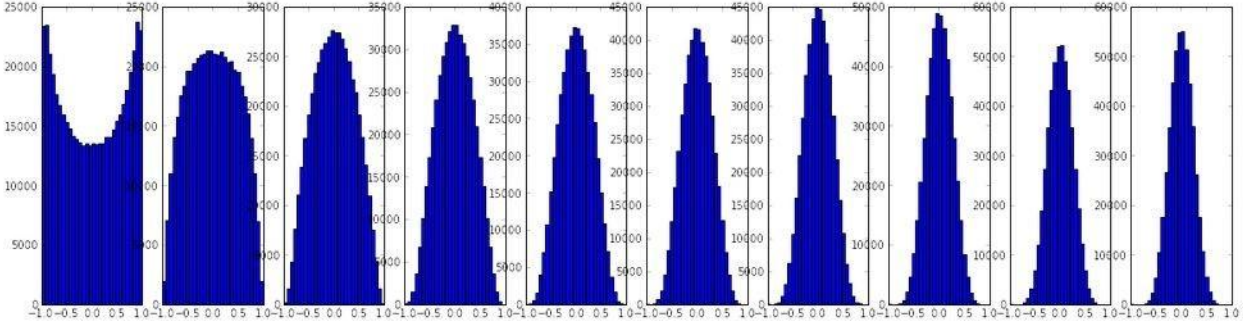
```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

(Gausijana sa srednjom vrednosti 0 i standardnom devijacijom 1) podeljena sa korenom broja ulaznih neurona.

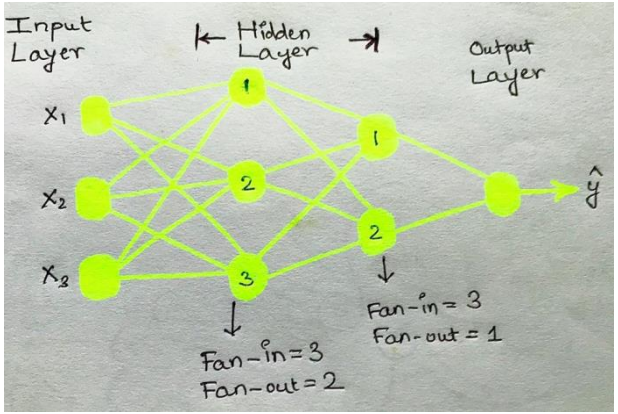
"Xavier inicijalizacija" [Glorot et al., 2010]







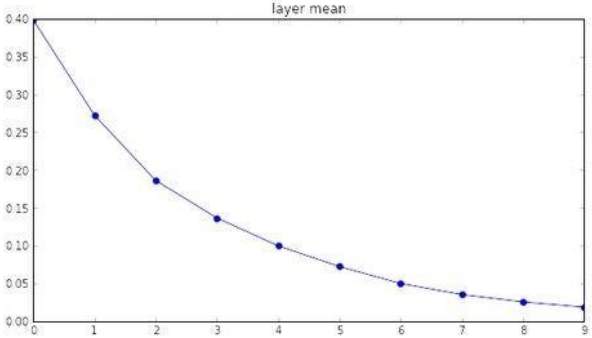
Razumna inicijalizacija.
Rešava problem koji smo imali u prethodnim slajdovima.
Ne radi dobro sa ReLU

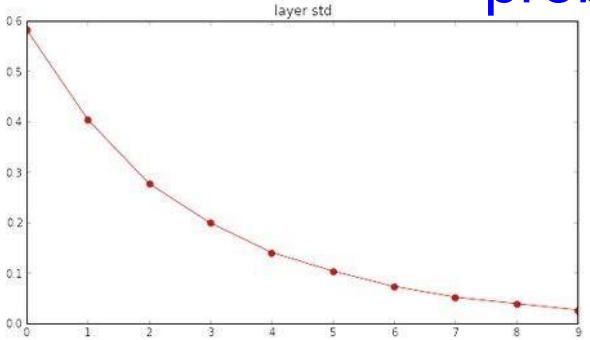


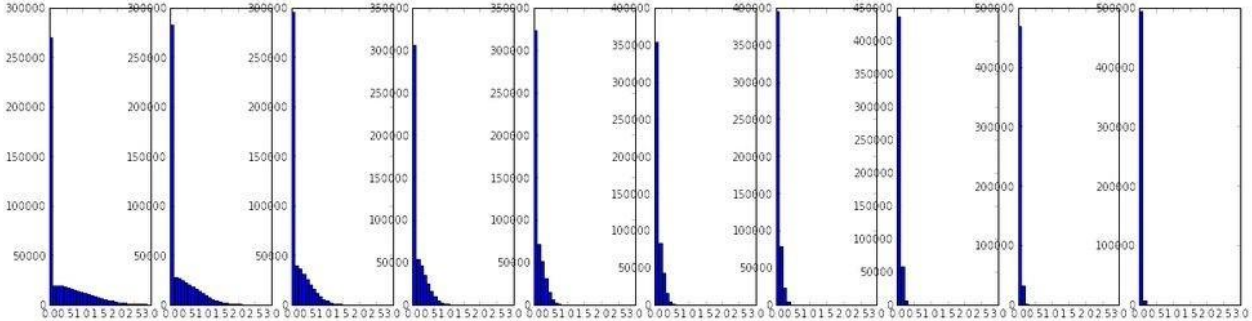
```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.272352 and std 0.403795
hidden layer 3 had mean 0.186076 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.140299
hidden layer 6 had mean 0.072234 and std 0.103280
hidden layer 7 had mean 0.049775 and std 0.072748
hidden layer 8 had mean 0.035138 and std 0.051572
hidden layer 9 had mean 0.025404 and std 0.038583
hidden layer 10 had mean 0.018408 and std 0.026076
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

Kada koristimo ReLU opet imamo stari problem



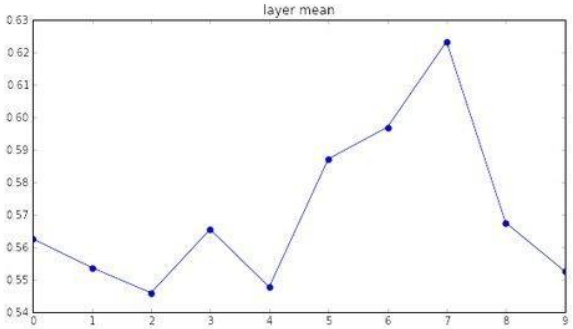


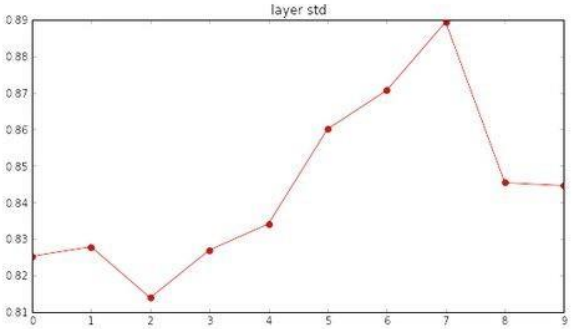


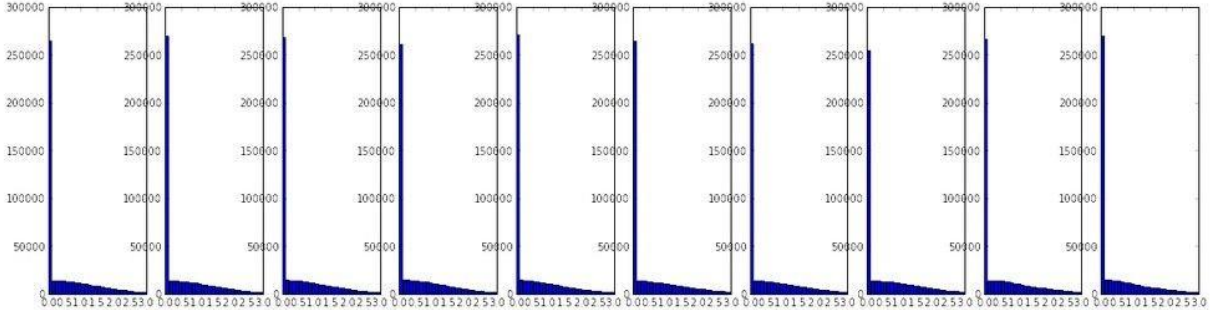
```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587103 and std 0.860035
hidden layer 7 had mean 0.596867 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

He et al., 2015 (autori su samo dodali /2 gore)



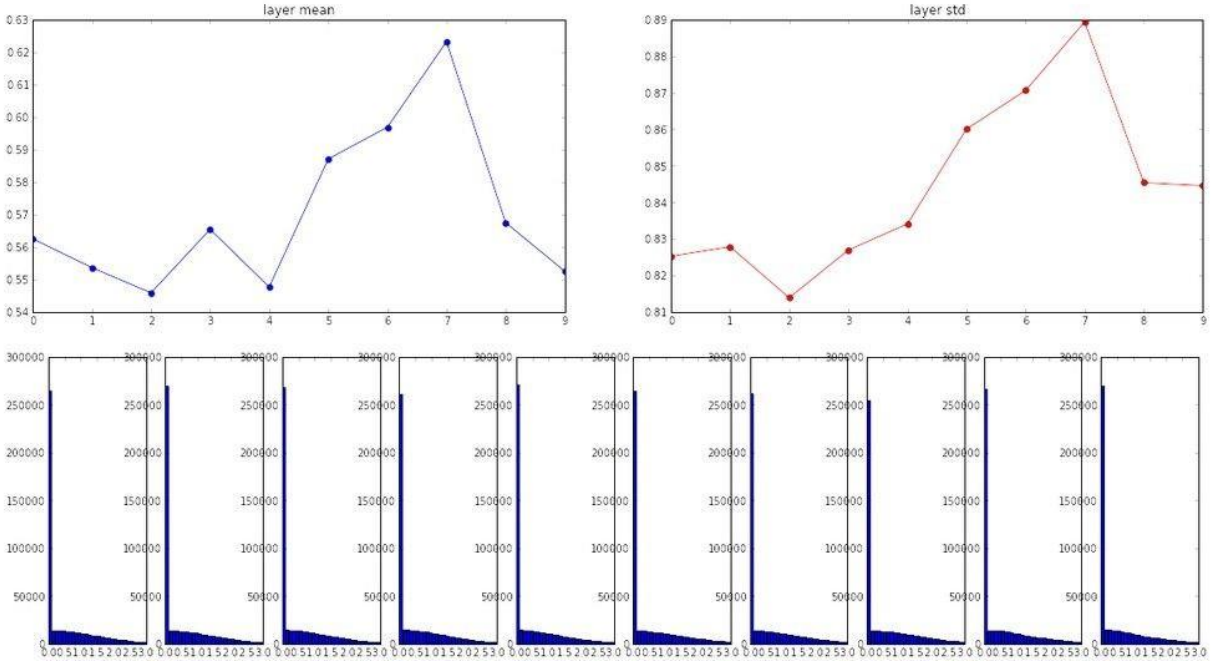




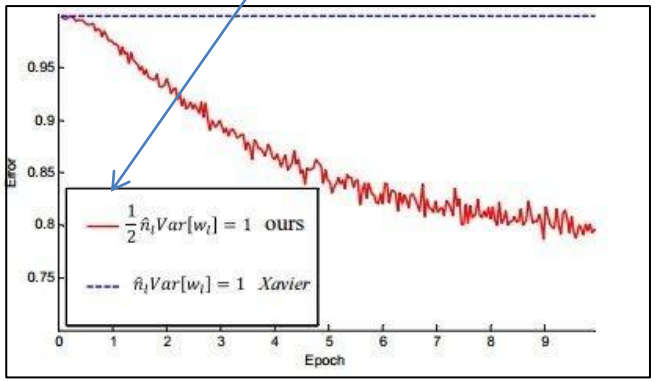
```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587103 and std 0.860035
hidden layer 7 had mean 0.596867 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in/2) # layer initialization
```

He et al., 2015 (note additional /2)



ReLU sa Xavier ima grešku 100%, a ako dodamo /2 sve radi



Inicijalizacija težina (kao i sve kod Deep Learning) je aktivna oblast istraživanja...

Understanding the difficulty of training deep feedforward neural networks by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by Saxe et al, 2013

Random walk initialization for training very deep feedforward networks by Sussillo and Abbott, 2014

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015

All you need is a good init, Mishkin and Matas, 2015

…

Batch Normalizacija

[Ioffe and Szegedy, 2015]

"Želite da aktivacije budu Gausijane? Transformišite ih u Gausijane ."

Recimo da sačuvamo aktivacije jednog podskupa (*batch*) za jedan sloj. Da bi od njih dobili Gausijanu sa sredjom vrednosti 0 i st.dev. 1 treba da primenimo:

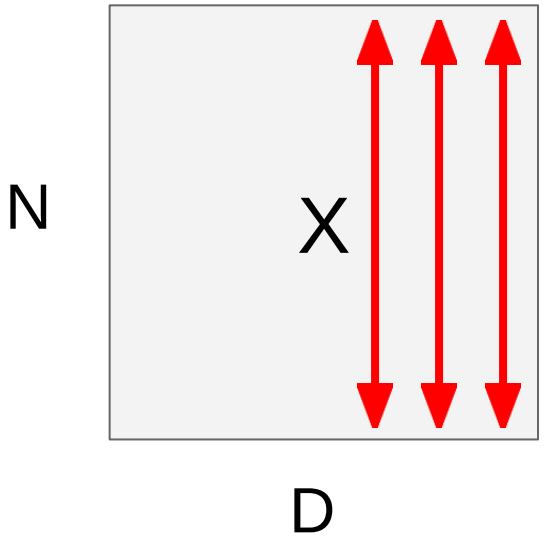
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

ovo je diferencijabilna fukcija, backprop neće imati problema...

Batch Normalizacija

[Ioffe and Szegedy, 2015]

"Želite da aktivacije budu Gausijane? Transformišite ih u Gausijane ."



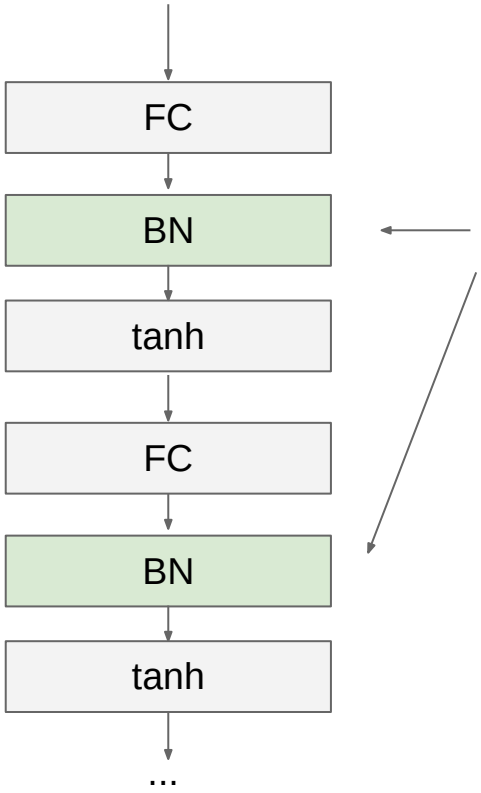
1. izračunavamo srednju vrednost i st. dev. po svakoj dimenziji (atributu) za N primera u podskupu.

2. Normalizujemo

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalizacija

[Ioffe and Szegedy, 2015]



Obično se ubacuje posle svakog potpuno povezanog (ili konvolutivnog) sloja pre funkcije aktivacije

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalizacija

[Ioffe and Szegedy, 2015]

Normalizacija:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Dozvoljavamo mreži da „zgnječi“ i pomeri aktivacije ako želi:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Mreža čak može i da nauči:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

i time poništi Batch Norm.

Batch Normalizacija

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Poboljšava tok gradijenata (*gradient flow*) kroz mrežu
- Dozvoljava veće korake učenja
- Mreža više nije toliko osetljiva na inicijalizacije težina

Batch Normalizacija

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

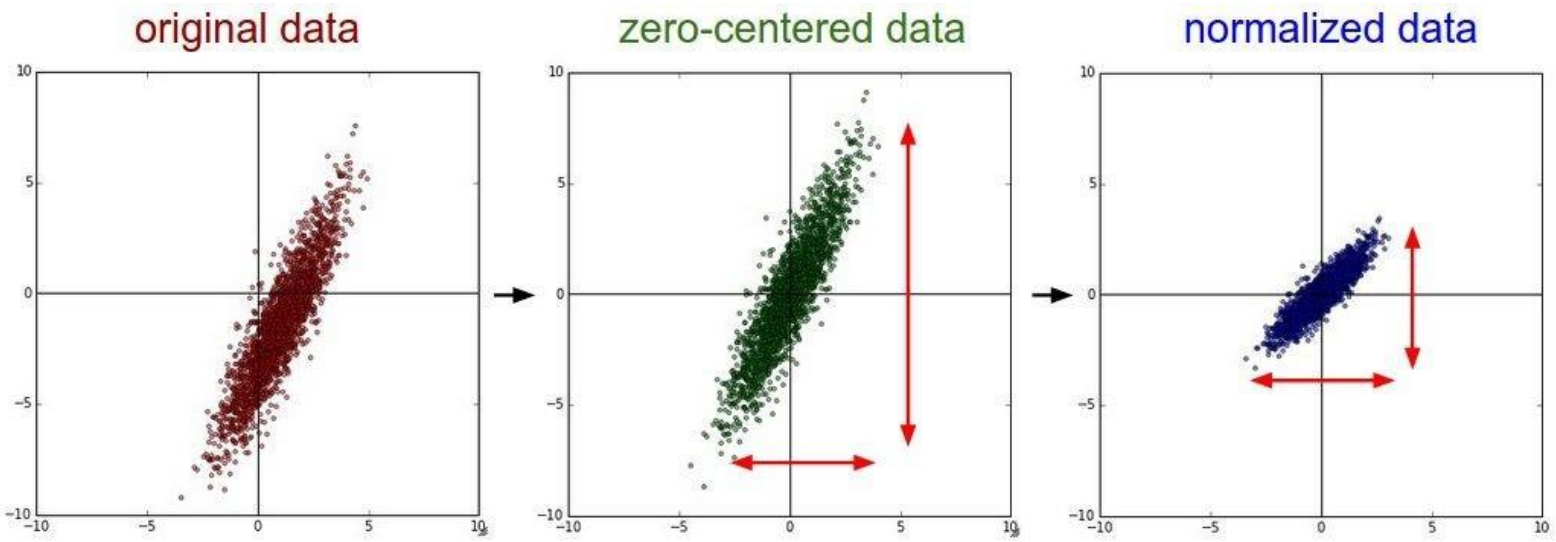
Napomena: kada primenjujemo mrežu na test podatke BatchNorm slojevi funkcionišu drugačije:

Srednja vrednost i st. dev se sad ne izračunavaju. Koriste se one koje su dobijene tokom obučavanja. – mreža je „naučila“ na te vrednosti, dok bi vrednosti iz test podataka pokvarile predikcije

(npr. možemo uzeti prosek srednjih vrednosti i st.dev. za jedan BN sloj tokom obučavanja)

Kako Pratiti Proces Obučavanja
Babysitting the Learning Process

Korak 1: Predprocesirati podatke




```
X -= np.mean(X, axis = 0)
```

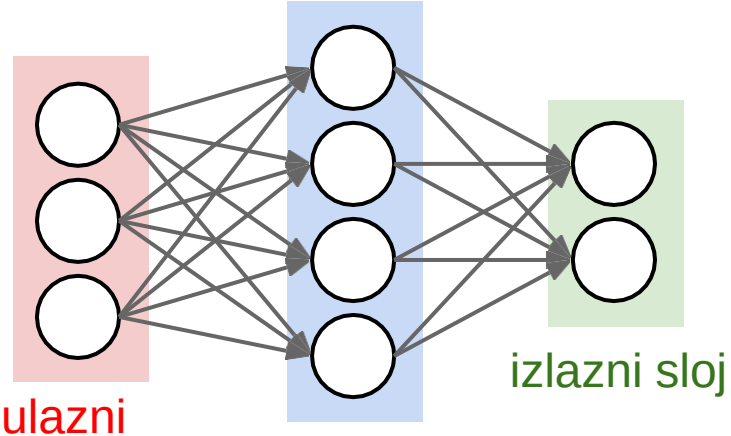
```
X /= np.std(X, axis = 0)
```

(X [NxD] je matrica podataka, svaki primer je jedan red)

Korak 2: Odabrati Arhitekturu Mreže:

recimo da krenemo sa mrežom sa jednim skrivenim slojem od 50 neurona:

50 skrivenih neurona



CIFAR-10 skup podataka, **3072** slika brojeva

10 izlaznih neurona, po jedan za svaku klasu

Proveravamo da li smo dobro implementirali funkciju greške: puštamo na ulaz ceo skup podataka, ali na ne-obučenu mrežu

```
def init_two_layer_model(input_size, hidden_size, output_size):
  # initialize a model
  model = {}
  model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
  model['b1'] = np.zeros(hidden_size)
  model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
  model['b2'] = np.zeros(output_size)
  return model
```

Treba da dobijemo skoro maksimalnu vrednost koja je moguća za tu funkciju greške.

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 0.0)
print loss
```

isključimo regularizaciju

```
2.30261216167
```

greška ~2.3. "tačna" za 10 klasa - softmax

metod vraća ukupnu grešku i gradijente za sve parametre

Proveravamo da li smo dobro implementirali funkciju greške: puštamo na ulaz ceo skup podataka, ali na ne-obučenu mrežu

```
def init_two_layer_model(input_size, hidden_size, output_size):
  # initialize a model
  model = {}
  model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
  model['b1'] = np.zeros(hidden_size)
  model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
  model['b2'] = np.zeros(output_size)
  return model
```

Treba da dobijemo skoro maksimalnu vrednost koja je moguća za tu funkciju greške.

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 1e3)
print loss
```

povećamo regularizaciju

```
3.06859716482
```

greška je porasla, što je dobro

Krećemo sa obučavanjem…

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)
```

Savet: Za početak probajte da overfitujete (dobijete training grešku od skoro 0%) mali podskup podataka.

To je još jedan znak da je implementacija dobra.

Kod gore:

- uzimamo prvih 20 primera iz CIFAR-10
- isključimo regularizaciju (reg = 0.0)
- koristimo običan SGD

Krećemo sa obučavanjem…

Savet: Za početak probajte da overfitujete (dobijete training grešku od 0%) mali podskup podataka.

Vrlo mala greška, train tačnost je 1.00 (100%), što je dobro!

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)
```

```
Finished epoch 1 / 200: cost 2.302603, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 2 / 200: cost 2.302258, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 3 / 200: cost 2.301849, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 4 / 200: cost 2.301196, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 5 / 200: cost 2.300044, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 6 / 200: cost 2.297864, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 7 / 200: cost 2.293595, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 8 / 200: cost 2.285096, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 9 / 200: cost 2.268094, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 10 / 200: cost 2.234787, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 11 / 200: cost 2.173187, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 12 / 200: cost 2.076862, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 13 / 200: cost 1.974090, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 14 / 200: cost 1.895885, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 15 / 200: cost 1.820876, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 16 / 200: cost 1.737430, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 17 / 200: cost 1.642356, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 18 / 200: cost 1.535239, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 19 / 200: cost 1.421527, train: 0.600000, val 0.600000, lr 1.000000e-03
```

```
Finished epoch 195 / 200: cost 0.002694, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 196 / 200: cost 0.002674, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 197 / 200: cost 0.002655, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 198 / 200: cost 0.002635, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 199 / 200: cost 0.002617, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 200 / 200: cost 0.002597, train: 1.000000, val 1.000000, lr 1.000000e-03
finished optimization. best validation accuracy: 1.000000
```

Krećemo sa obučavanjem…

Počinjemo sa malom regularizacijom i tražimo korak učenja na osnovu koga greška kreće da pada

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

Krećemo sa obučavanjem…

Počinjemo sa malom regularizacijom i tražimo korak učenja na osnovu koga greška kreće da pada

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Greška jedva pada

Krećemo sa obučavanjem…

Počinjemo sa malom regularizacijom i tražimo korak učenja na osnovu koga greška kreće da pada

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Greška jedva pada: korak učenja je previše mali

greška ne pada: korak učenja je previše mali

Krećemo sa obučavanjem…

Počinjemo sa malom regularizacijom i tražimo korak učenja na osnovu koga greška kreće da pada

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

greška ne pada:
korak učenja je previše mali

Greška jedva pada:
korak učenja je previše mali

Primetite da train/val tačnost stiže do 20%, otkud sad to? (pomoć: ovo je softmax greška)

Krećemo sa obučavanjem…

Počinjemo sa malom regularizacijom i tražimo korak učenja na osnovu koga greška kreće da pada

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e6, verbose=True)
```

Šta ako povećamo korak učenja na 10^6? Šta bi moglo loše da se dogodi ☺?

Krećemo sa obučavanjem…

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e6, verbose=True)
```

Počinjemo sa malom regularizacijom i tražimo korak učenja na osnovu koga greška kreće da pada

```
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:50: RuntimeWarning: divide by zero en
countered in log
  data_loss = -np.sum(np.log(probs[range(N), y])) / N
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:48: RuntimeWarning: invalid value enc
ountered in subtract
  probs = np.exp(scores - np.max(scores, axis=1, keepdims=True))
Finished epoch 1 / 10: cost nan, train: 0.091000, val 0.087000, lr 1.000000e+06
Finished epoch 2 / 10: cost nan, train: 0.095000, val 0.087000, lr 1.000000e+06
Finished epoch 3 / 10: cost nan, train: 0.100000, val 0.087000, lr 1.000000e+06
```

greška ne pada: korak učenja je previše mali
greška eksplodira: korak učenja je previše veliki

NaN vrednosti za grešku vrlo verovatno znače da je korak učenja preveliki...

Krećemo sa obučavanjem…

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=3e-3, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.186654, train: 0.308000, val 0.306000, lr 3.000000e-03
Finished epoch 2 / 10: cost 2.176230, train: 0.330000, val 0.350000, lr 3.000000e-03
Finished epoch 3 / 10: cost 1.942257, train: 0.376000, val 0.352000, lr 3.000000e-03
Finished epoch 4 / 10: cost 1.827868, train: 0.329000, val 0.310000, lr 3.000000e-03
Finished epoch 5 / 10: cost inf, train: 0.128000, val 0.128000, lr 3.000000e-03
Finished epoch 6 / 10: cost inf, train: 0.144000, val 0.147000, lr 3.000000e-03
```

Počinjemo sa malom regularizacijom i tražimo korak učenja na osnovu koga greška kreće da pada

3e-3 je i dalje previše. Greška eksplodira….

greška ne pada: korak učenja je previše mali
greška eksplodira: korak učenja je previše veliki

=> Na ovaj način došli bi do nekog raspona koji nam se čini dobar i u kome bi trebali da štelujemo korak učenja fino. U ovom slučaju je to [1e-3 … 1e-5]

Krećemo sa obučavanjem…

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=3e-3, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.186654, train: 0.308000, val 0.306000, lr 3.000000e-03
Finished epoch 2 / 10: cost 2.176230, train: 0.330000, val 0.350000, lr 3.000000e-03
Finished epoch 3 / 10: cost 1.942257, train: 0.376000, val 0.352000, lr 3.000000e-03
Finished epoch 4 / 10: cost 1.827868, train: 0.329000, val 0.310000, lr 3.000000e-03
Finished epoch 5 / 10: cost inf, train: 0.128000, val 0.128000, lr 3.000000e-03
Finished epoch 6 / 10: cost inf, train: 0.144000, val 0.147000, lr 3.000000e-03
```

Počinjemo sa malom regularizacijom i tražimo korak učenja na osnovu koga greška kreće da pada

greška ne pada: korak učenja je previše mali

greška eksplodira: korak učenja je previše veliki

Napomena: val vrednost možemo dobiti pomoću unakrsne validacije ili validacionog skupa.
Unakrsna validacija je realnija, ali spora
Sa validacionim skupom izračunavanje greške je brže, ali greška može biti nerealna.

Optimizacija Hiperparametara
Hyperparameter Optimization

Strategija

grubo -> fino u fazama

Prva faza: samo par epoha za svaku vrednost parametara, dok ne dobijemo grubu ideju koji rasponi su dobri
Druga faza: više epoha, fino štelovanje u okviru raspona iz prve faze
… (faze možemo i ponoviti po potrebi)

Savet:
Ako je greška > 3 * greške od koje smo krenuli,
prekinuti obučavanje i promeniti nešto

Na primer: radimo grubu pretragu 5 epoha

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)

      trainer = ClassifierTrainer()
      model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
      trainer = ClassifierTrainer()
      best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                              model, two_layer_net,
                                              num_epochs=5, reg=reg,
                                              update='momentum', learning_rate_decay=0.9,
                                              sample_batches = True, batch_size = 100,
                                              learning_rate=lr, verbose=False)
```

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

dobre vrednosti

Sada radimo finiju pretragu...

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

podešavamo raspon →

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relativno dobra tačnost za 2-slojnu mrežu sa 50 neurona u skrivenom sloju.

Sada radimo finiju pretragu...

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

podešavamo raspon →

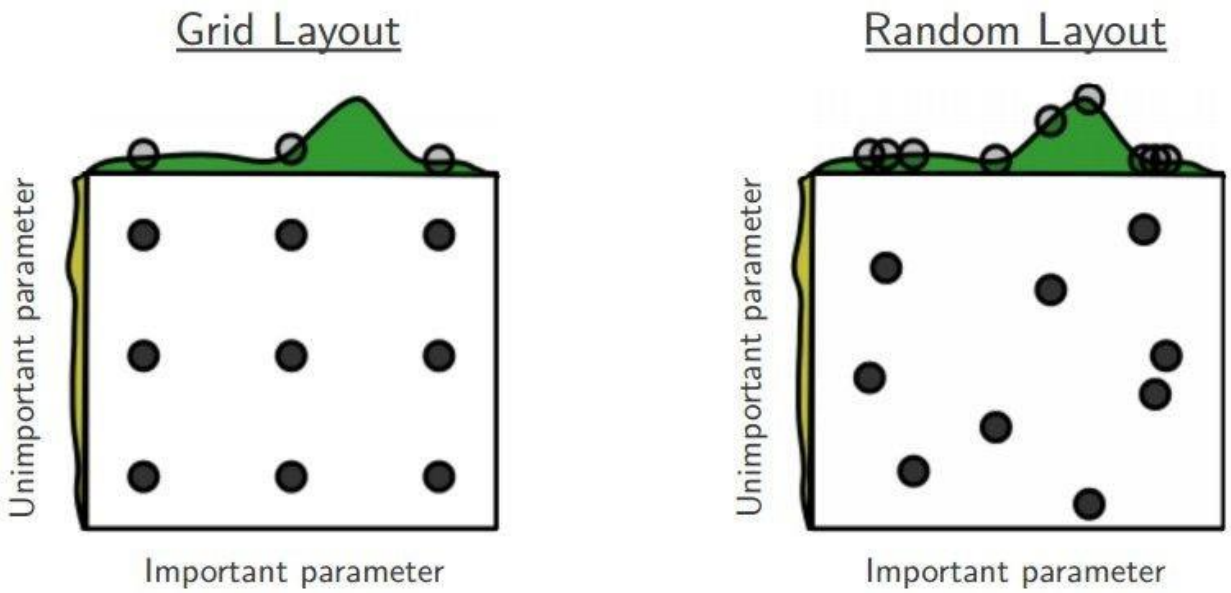
```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relativno dobra tačnost za 2-slojnu mrežu sa 50 neurona u skrivenom sloju.

Ovaj rezultat za val od 0.531.. bi trebalo da nas brine. Zašto?

Random Pretraga vs. Pretraga po Mreži (*Grid Search*)



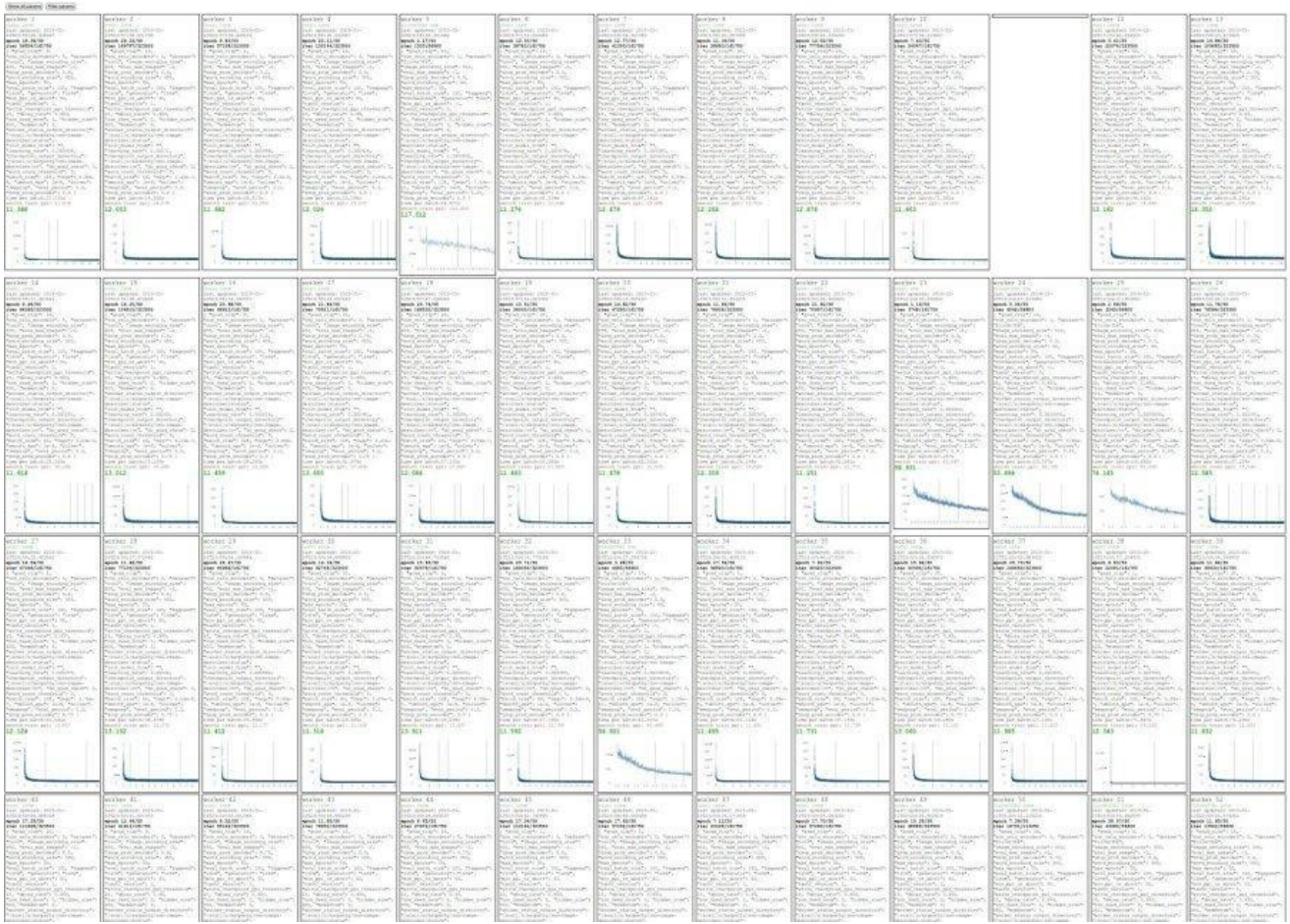
Random Search for Hyper-Parameter Optimization
Bergstra and Bengio, 2012

Hiperparametri sa kojima se igramo:

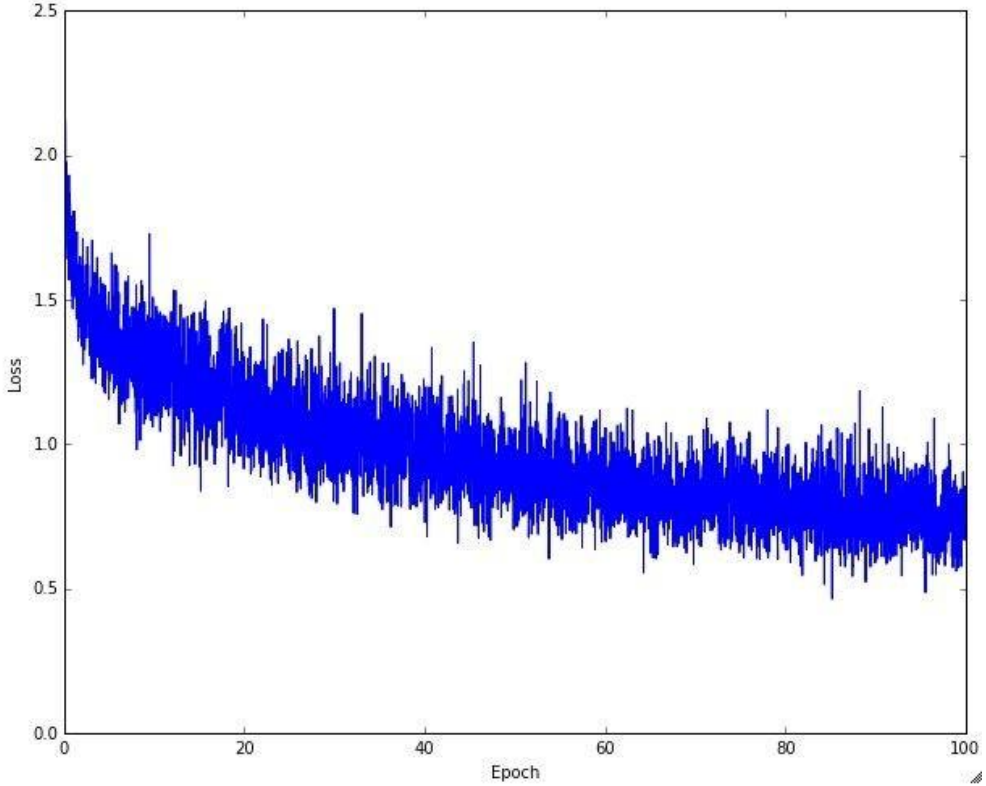
- arhitektrura mreže
- korak učenja, kako ga menjamo kroz vreme, način promene parametara
- regularizacija (L2/Dropout)
  - Neke od ovih pojmova tek treba da učimo

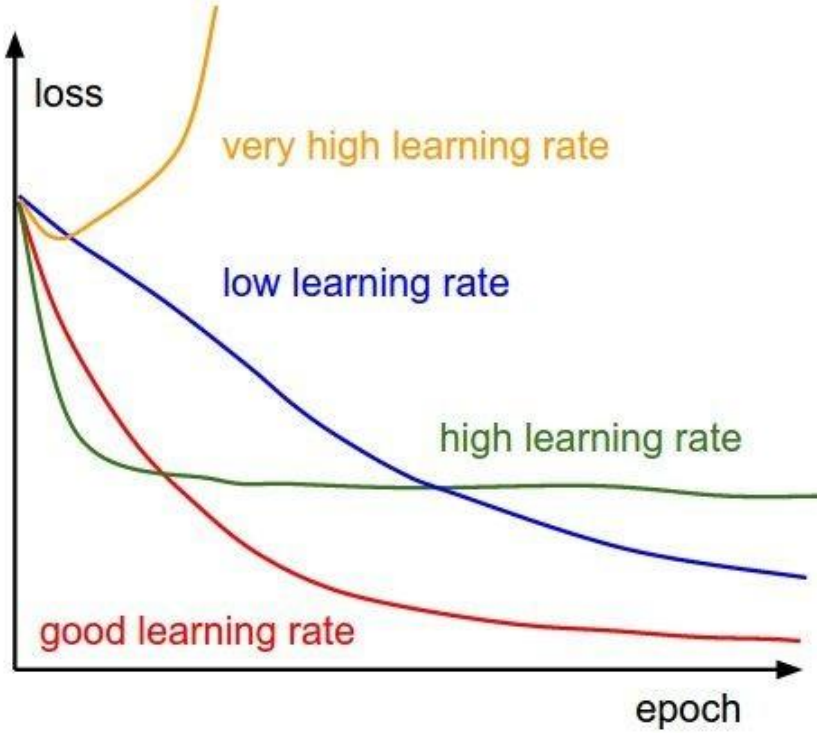
ovako izgleda štelovanje parametra sa ciljem da smanjimo grešku

Šta bi bilo da imamo klaster od 70 mašina za obučavanje ☺

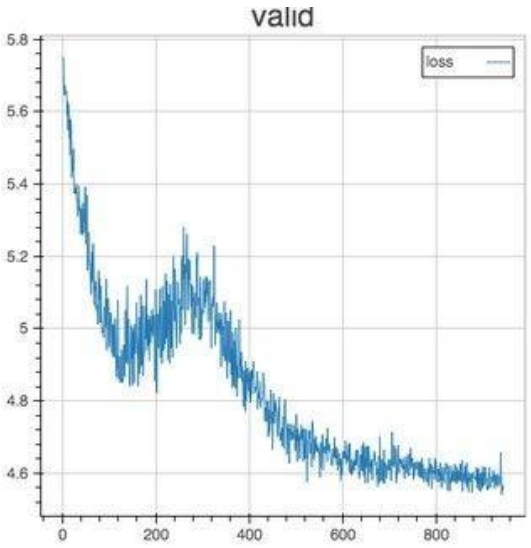


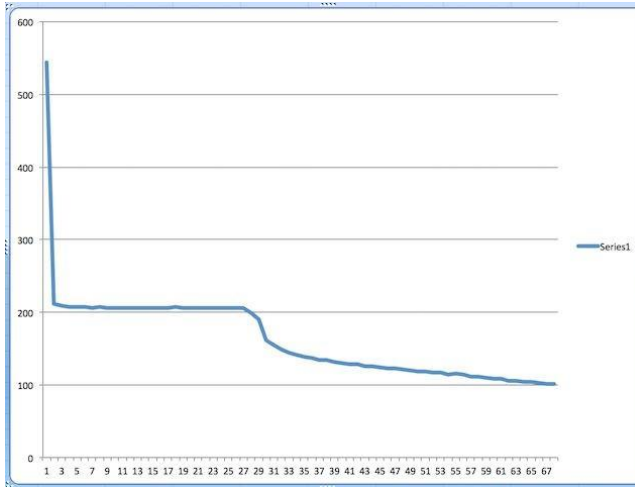
Obavezno vizualizujemo i pratimo krivu funkcije greške

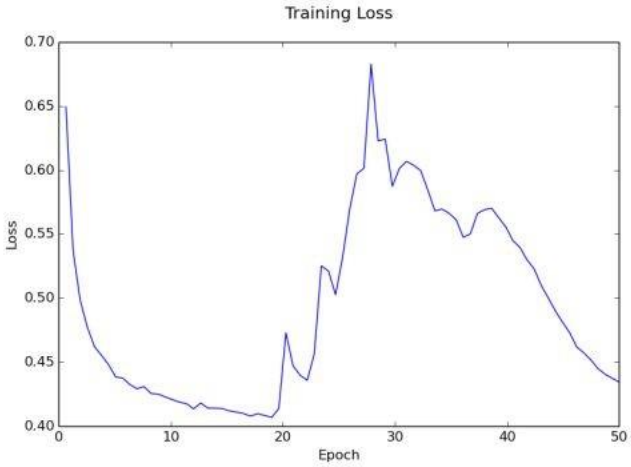




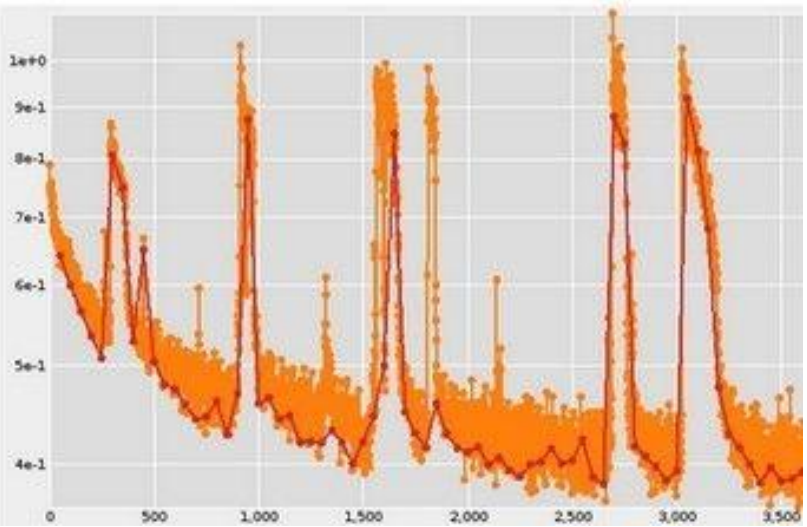
lossfunctions.tumblr.com Primeri funkcija greške

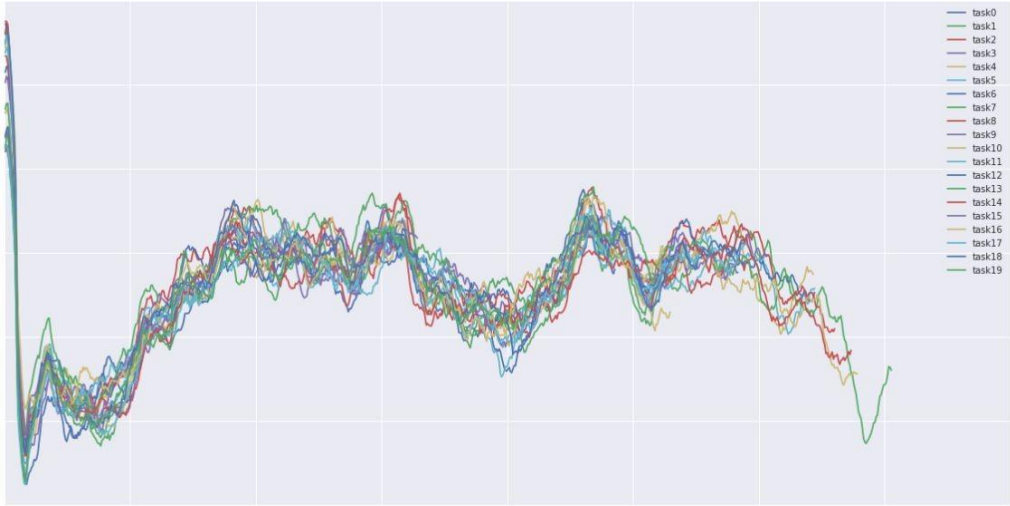


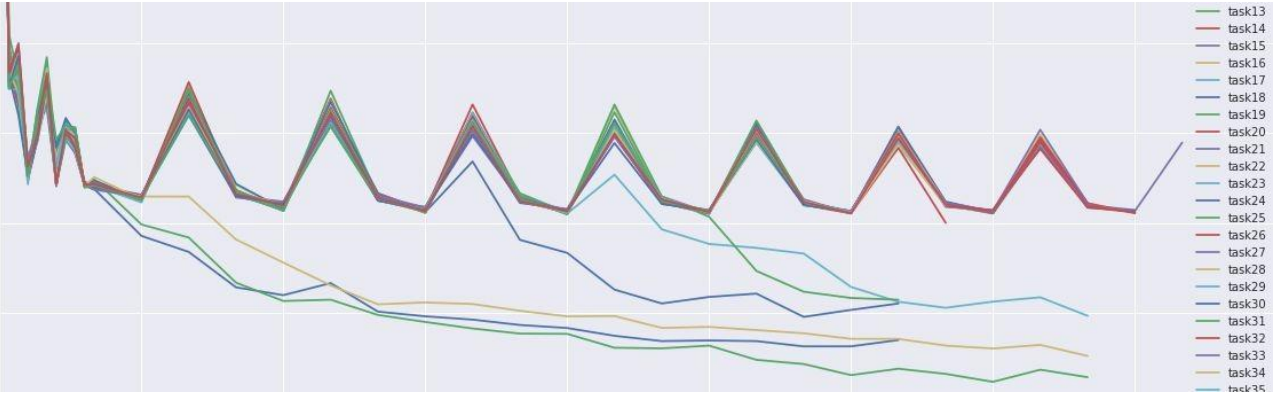


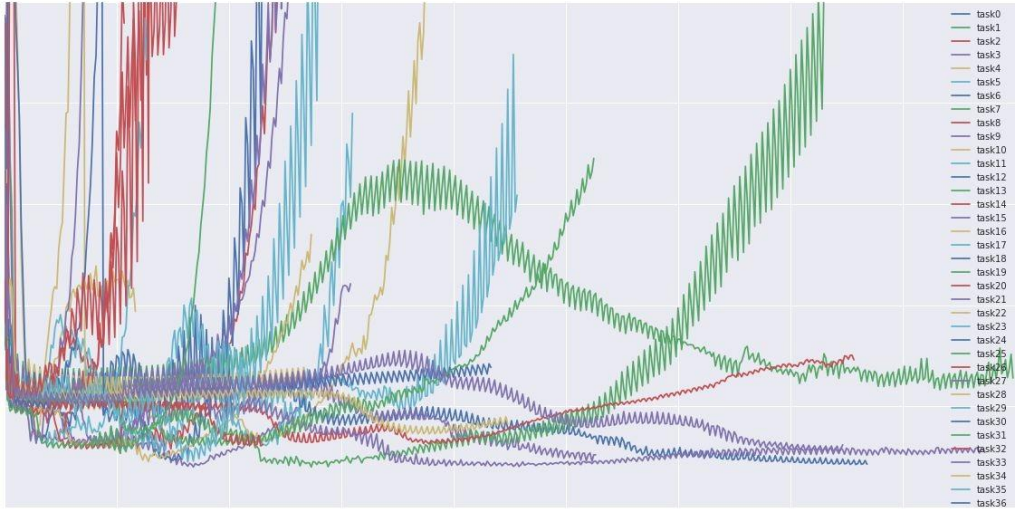


lossfunctions.tumblr.com



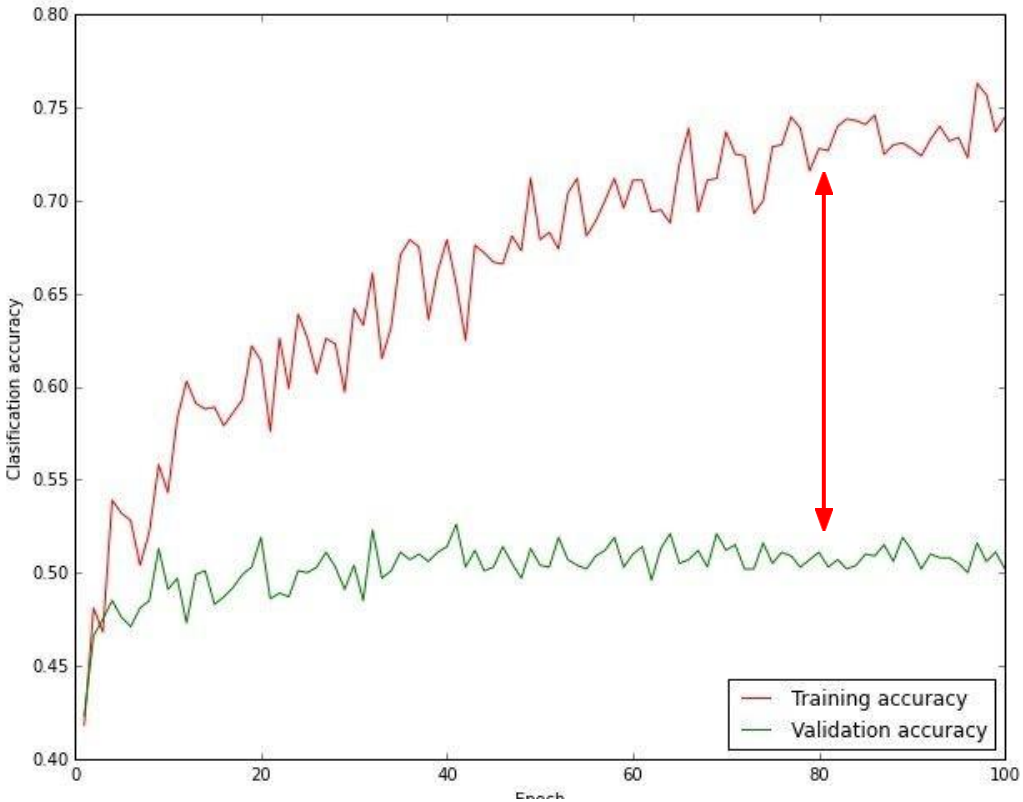






lossfunctions.tumblr.com

Obevezno vizualizujemo i pratimo tačnost:



velika razlika = overfitting

=> povećati uticaj regularizacije (lambda)?

nema razlike tj. i training accuracy ima malu vrednost

=> povećati kapacitet modela (npr. dodavanjem još neurona ili slojeva)?

Rezime

TLDRs

Danas smo se bavili sa:

- Aktivacionim Funkcijama (koristite ReLU)
- Predprocesiranjem Podataka (slike: oduzeti srednju sliku)
- Inicijalizacija Težina (koristite Xavier inicijalizaciju sa /2)
- Batch Normalizacije (koristite)
- Praćenje Procesa Učenja
- Optimizacija Hiper-parametara
  (vrednosti treba slučajno birati, iz log prostora, mada možete uvek probati i bez log)

Šta treba da radimo

- Razne načine promene parametara
- Da se više bavimo korakom učenja
- Provera Gradijenata
- Regularizacija (Dropout)
- Evaluacija (Ansambli)

Obučavanje Neuronskih Mreža
Drugi Deo

Predavač: Aleksandar Kovačević

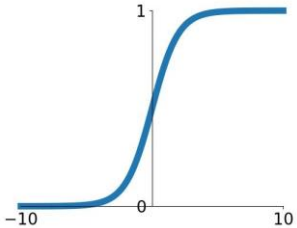
Slajdovi preuzeti sa CS 231n, Stanford

http://cs231n.stanford.edu/

Prošli put: Funkcije Aktivacije

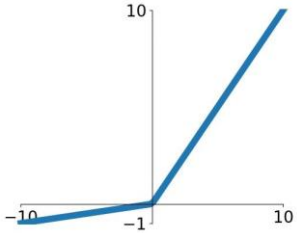
Sigmoid

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



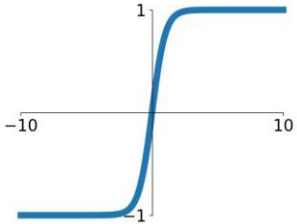
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

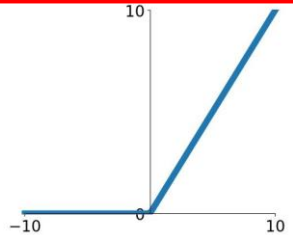


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ReLU

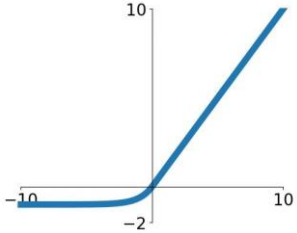
$$\max(0, x)$$



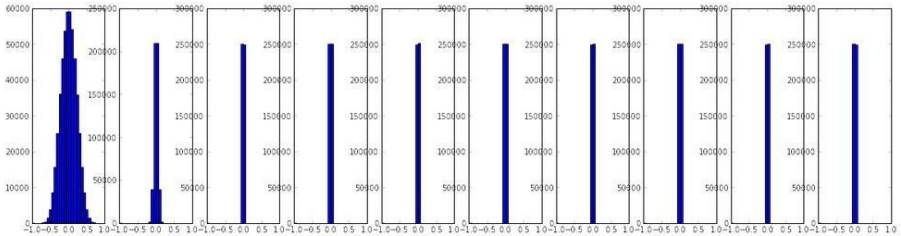
Dobar *default* izbor

ELU

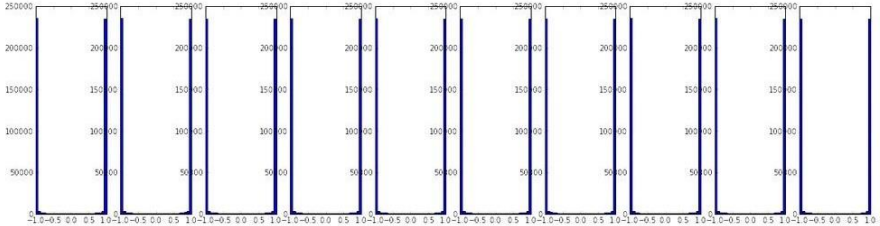
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



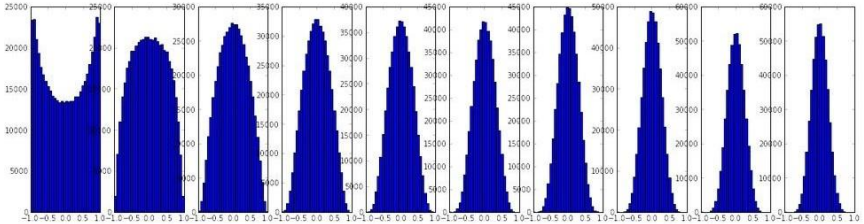
Prošli put: Inicijalizacija težina



inicijalne težine previše male:
Aktivacije i gradijenti postaju 0, nema obučavanja

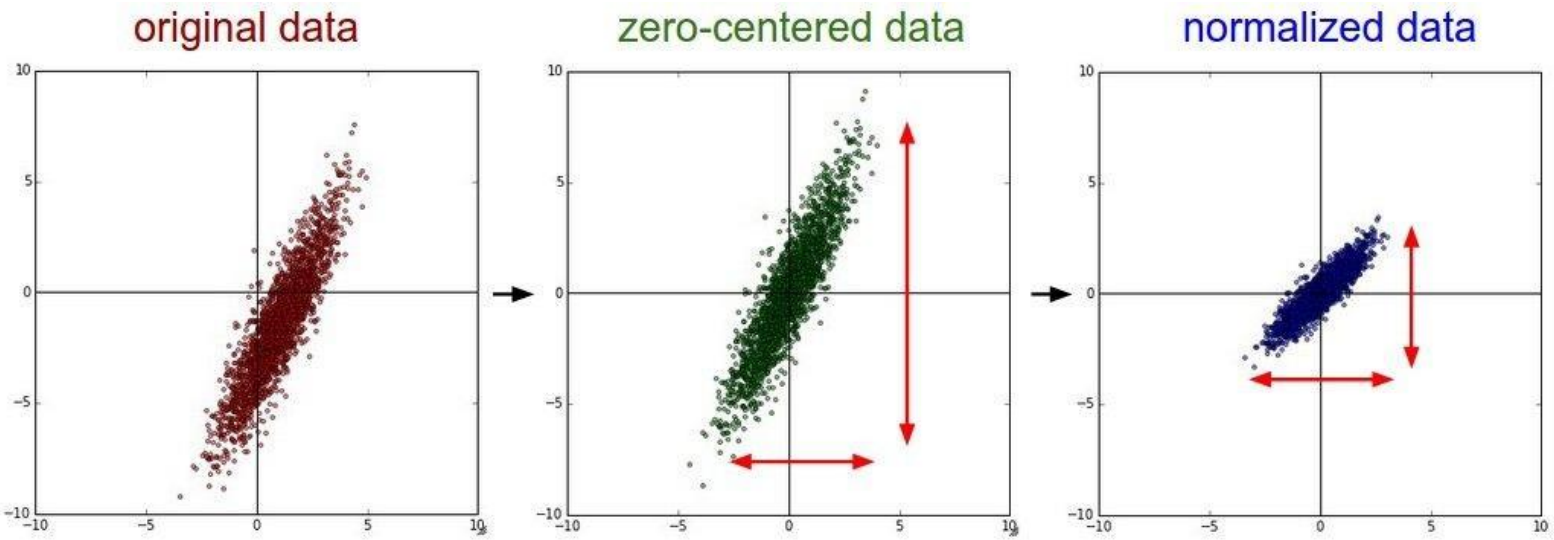


inicijlane težine previše velike:
Aktivacije su u saturaciji (za tanh ili sigmoid), gradijenti su 0, nema obučavanja



inicijalne težine dobro izabrane:
Dobre distribucije aktivacija po slojevima, obučavanje dobro napreduje

Prošli put: Predprocesiranje Podataka



Prošli put: Batch Normalizacija

Ulaz: $x : N \times D$

Parametri koji se uče:

$\gamma, \beta : D$

Vrednosti koje se računaju tokom obučavanja:

$\mu, \sigma : D$

$\hat{x} : N \times D$

Izlaz: $y : N \times D$

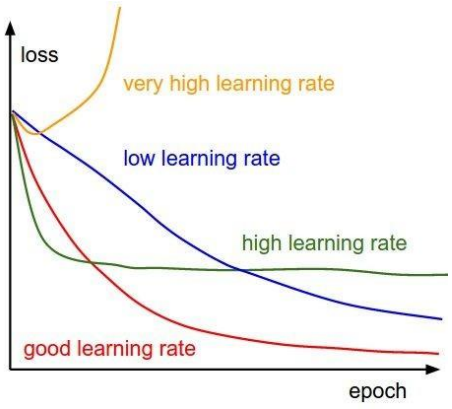
$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

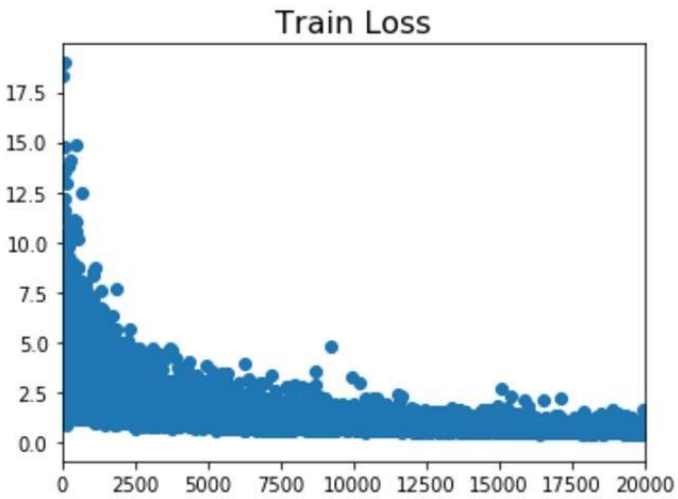
$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

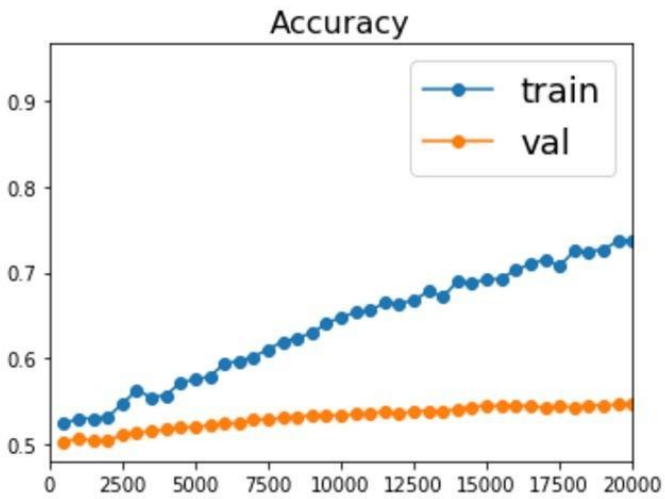
$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

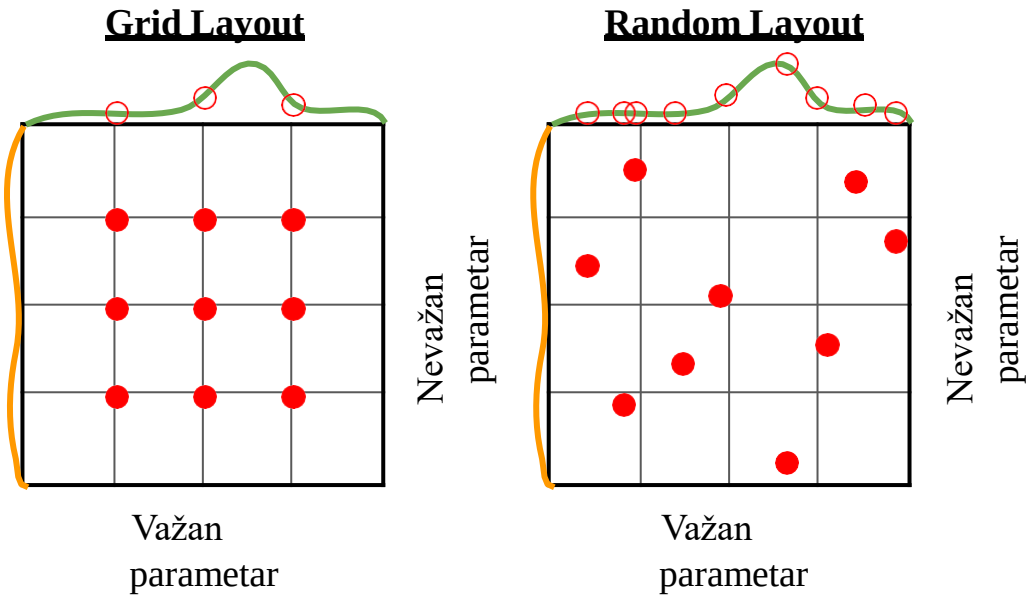
Prošli put: Praćenje Procesa Učenja







Prošli put: Optimizacija Hiperparametara



Od grublje ka finijoj pretrazi

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

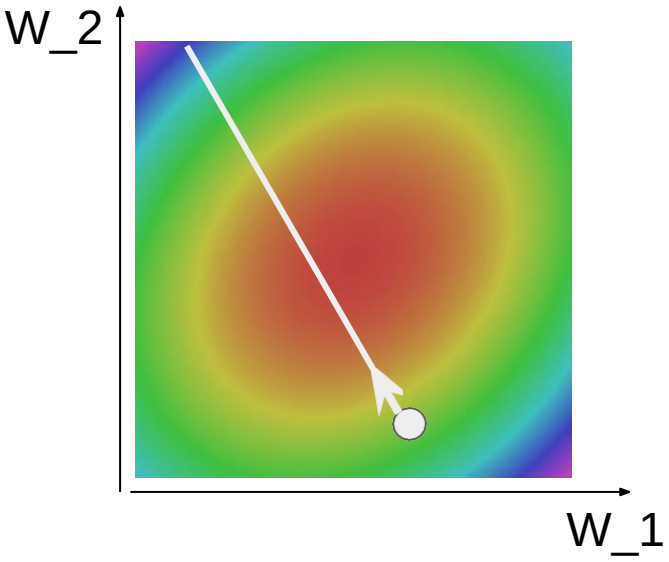
```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

Danas

- Poboljšanja optimizacije
- Regularziacija
- Učenje Transferom (*Transfer Learning*)

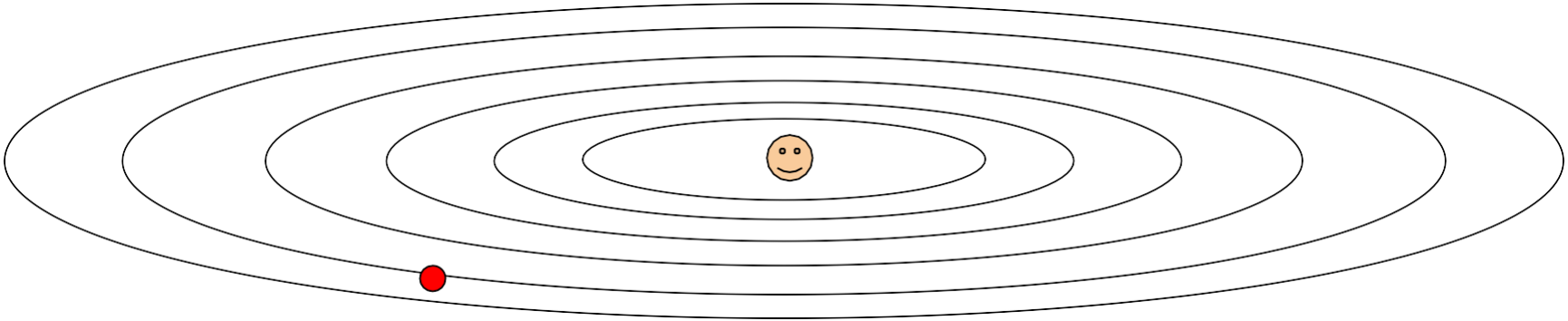
Optimizacija

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```



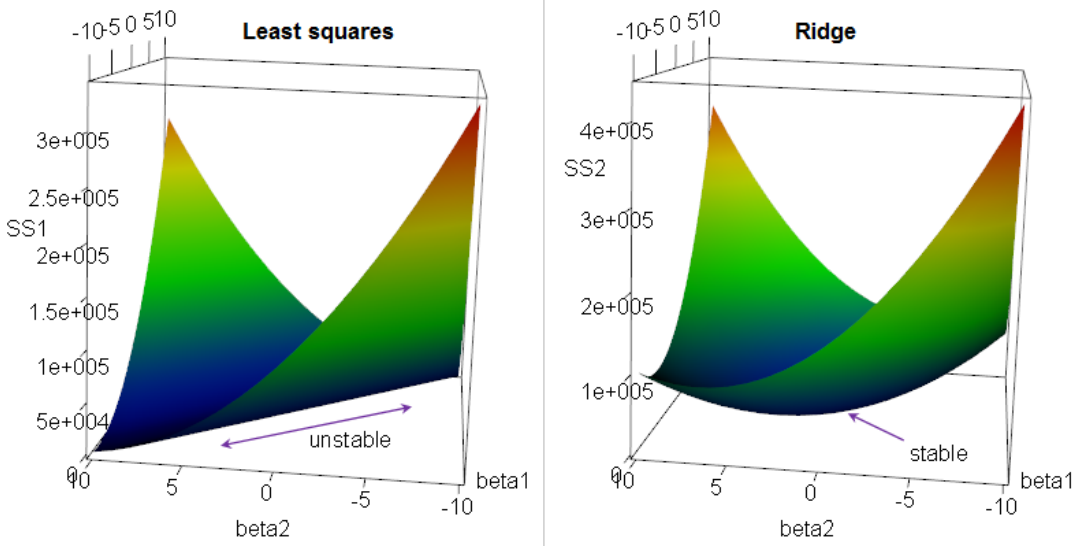
Optimizacija: Problemi sa SGD

Šta ako se greška brzo menja po jednoj koordinatnoj osi, a sporo po drugoj?
Kako onda izgleda gradijentni spust?



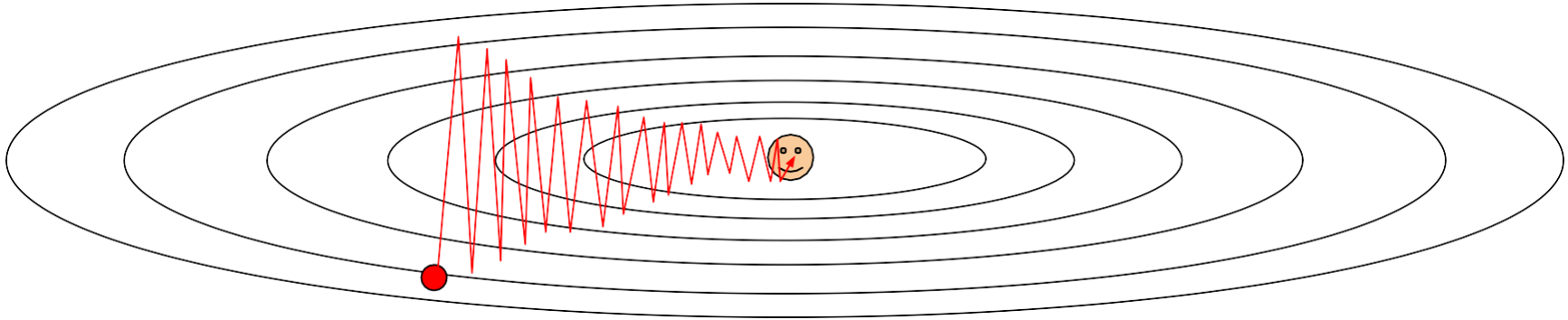
Optimizacija: Problemi sa SGD

Šta ako se greška brzo menja po jednoj koordinatnoj osi, a sporo po drugoj?
Kako onda izgleda gradijentni spust?



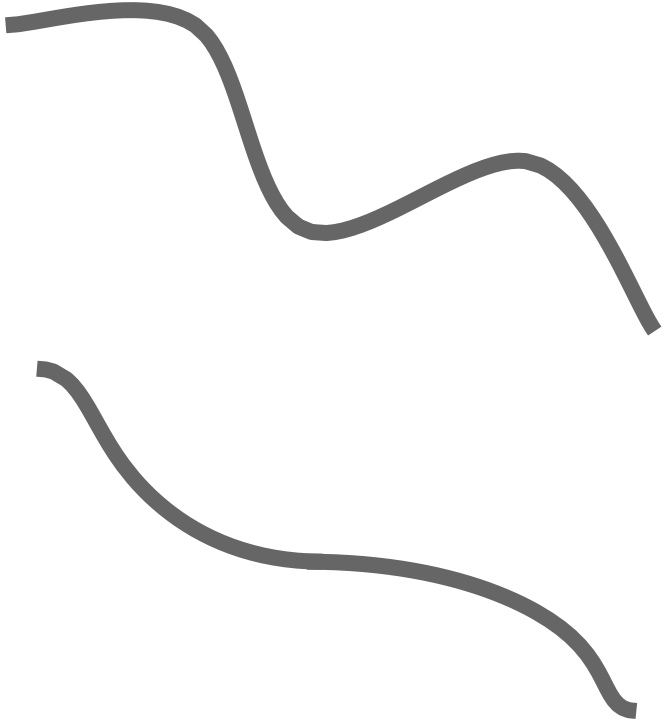
Optimizacija: Problemi sa SGD

Šta ako se greška brzo menja po jednoj koordinatnoj osi, a sporo po drugoj?
Kako onda izgleda gradijentni spust?
Jako spor napredak u plitkom smeru, „cimanje“ u strmom smeru



Optimizacija: Problemi sa SGD

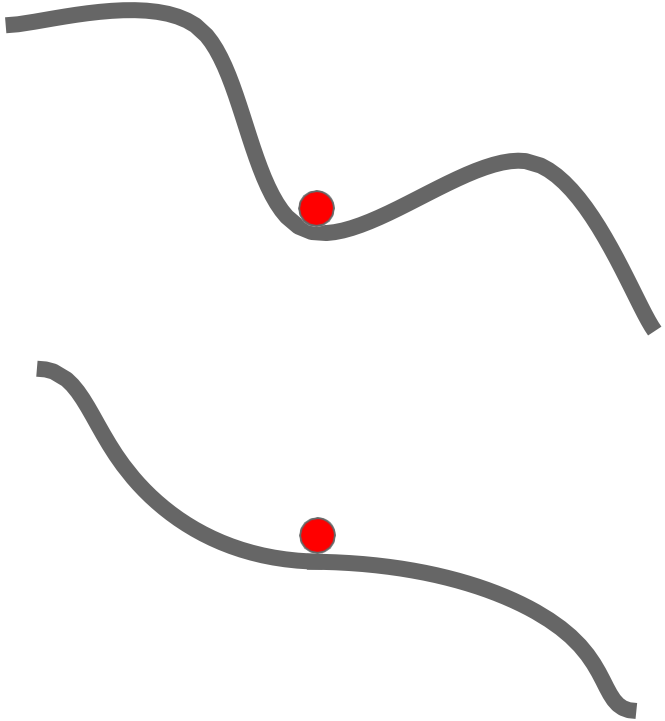
Šta ako funkcija greške ima **lokalne minimume** ili **tačke prevoja**?



Optimizacija: Problemi sa SGD

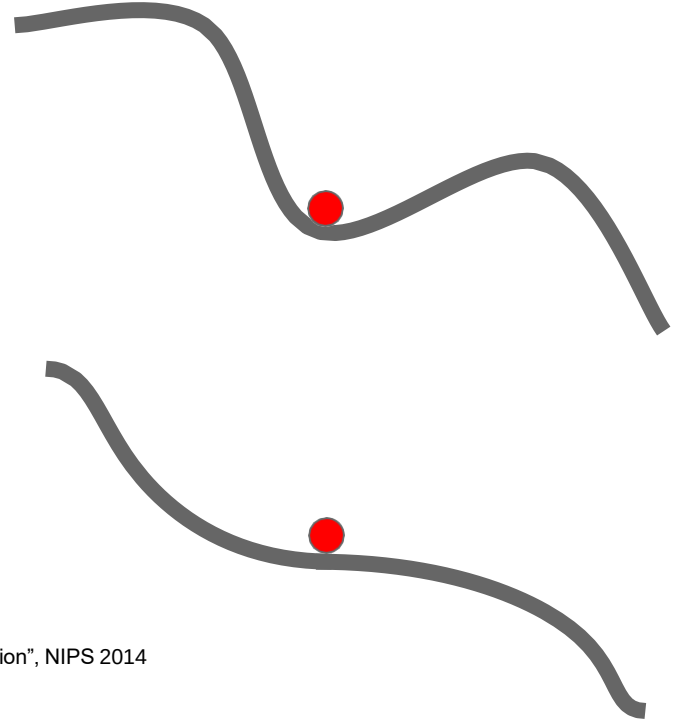
Šta ako funkcija greške ima **lokalne minimume** ili **tačke prevoja**?

Gradijent je 0 ili jako mali i SGD se zaglavljuje



Optimizacija: Problemi sa SGD

Šta ako funkcija greške ima **lokalne minimume** ili **tačke prevoja**?



Tačke prevoja su česte u visko-dimenzionim prostorima

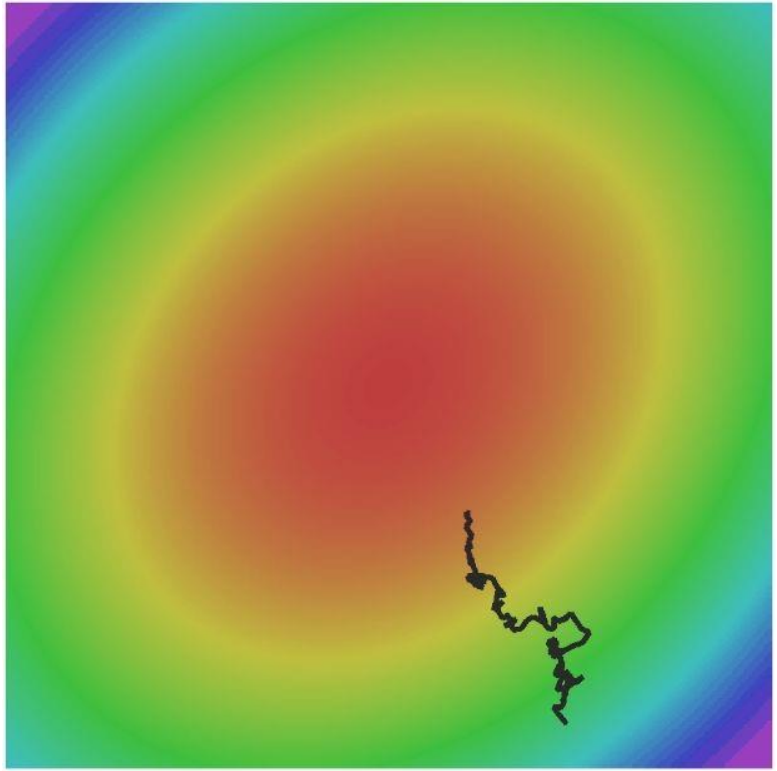
Dauphin et al, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”, NIPS 2014

Optimizacija: Problemi sa SGD

Gradijente računamo pomoću mini podskupova, pa nisu uvek dosledni više su kao na slici! Termin je *noisy gradients*.

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

```
while True:
  dx = compute_gradient(x)
  x += learning_rate * dx
```

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

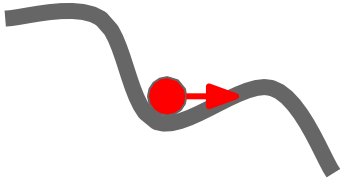
$$x_{t+1} = x_t - \alpha v_{t+1}$$

```
vx = 0
while True:
  dx = compute_gradient(x)
  vx = rho * vx + dx
  x += learning_rate * vx
```

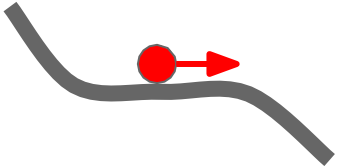
- Nakupljamo “ubrzanje” kao sumu prethodnih gradijenata
- Ro se tumači kao “trenje”; tipično je rho=0.9 ili 0.99

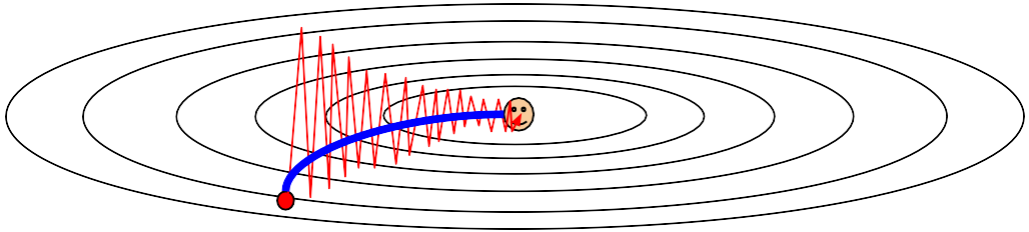
SGD + Momentum

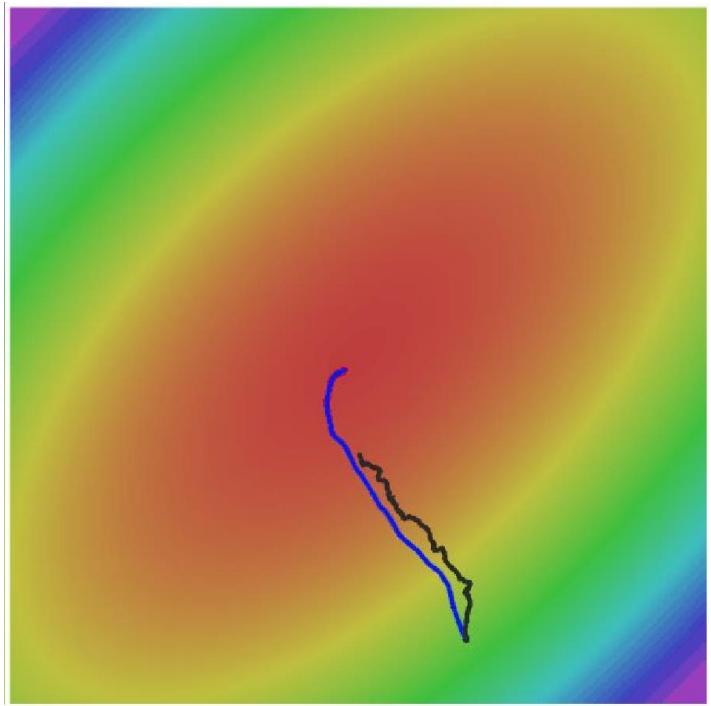
Lolalni min.



Tačka prevoja







AdaGrad

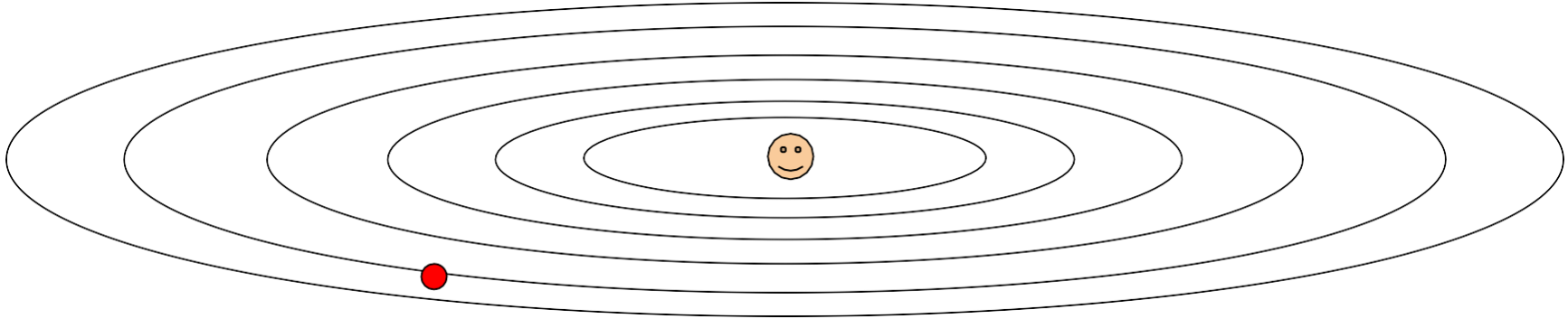
```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Skaliramo gradijent pomoću sume kvadrata dosadašnjih gradijenata. Skaliranje je po svakoj dimenziji posebono. Kod je vektorski.

Duchi et al, “Adaptive subgradient methods for online learning and stochastic optimization”, JMLR 2011

AdaGrad

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



Kako izgleda AdaGrad?

RMSProp

AdaGrad

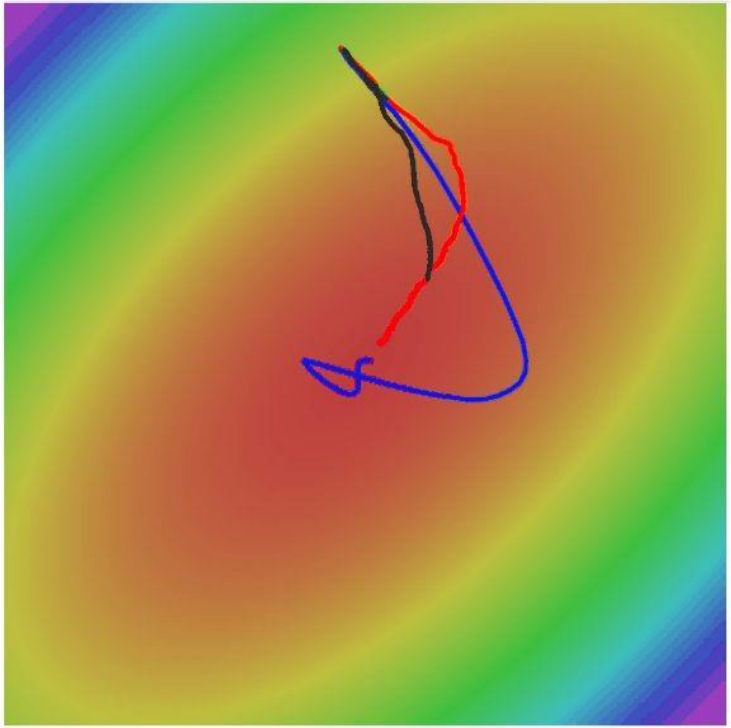
```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared += dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

RMSProp

```
grad_squared = 0
while True:
  dx = compute_gradient(x)
  grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
  x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Tieleman and Hinton, 2012

RMSProp



SGD

SGD+Momentum

RMSProp

Adam (skoro)

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (skoro)

```
first_moment = 0
second_moment = 0
while True:
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  x -= learning_rate * first_moment / (np.sqrt(second_moment) + 1e-7))
```

Momentum

AdaGrad / RMSProp

Kao RMSProp sa momentumom

Adam sa beta1 = 0.9, beta2 = 0.999, i learning_rate = 1e-3 ili 5e-4 je odličan izbor na početku obučavanja.

Šta se dešava u prvom koraku?

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam (kompletan)

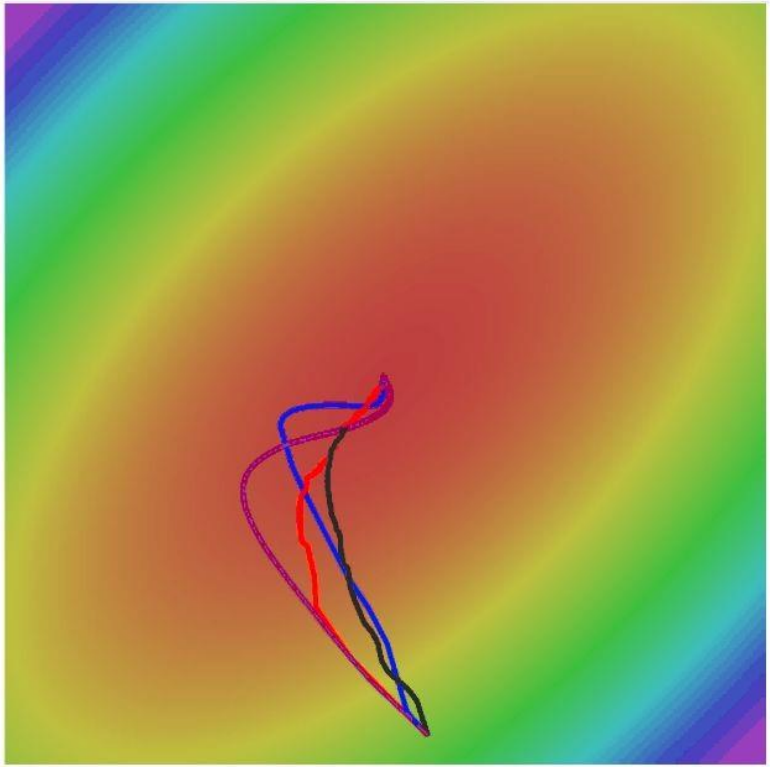
```
first_moment = 0
second_moment = 0
for t in range(num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

AdaGrad / RMSProp

Kingma and Ba, “Adam: A method for stochastic optimization”, ICLR 2015

Adam



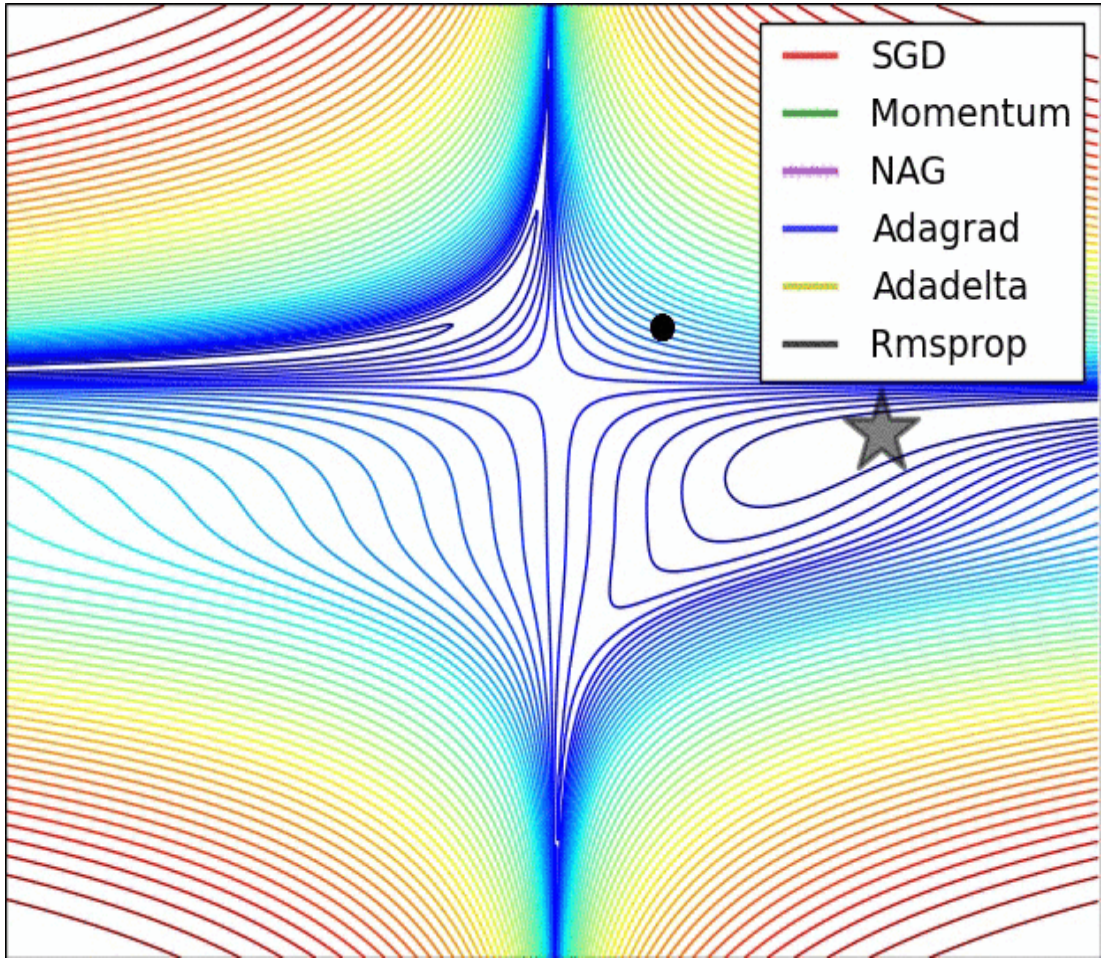
SGD

SGD+Momentum

RMSProp

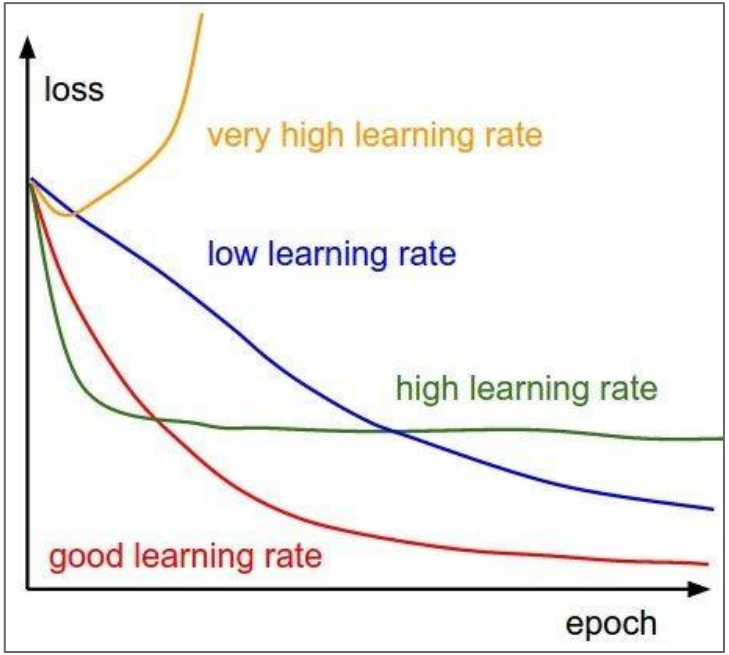
Adam

Ponašanje različitih formula za promene težina



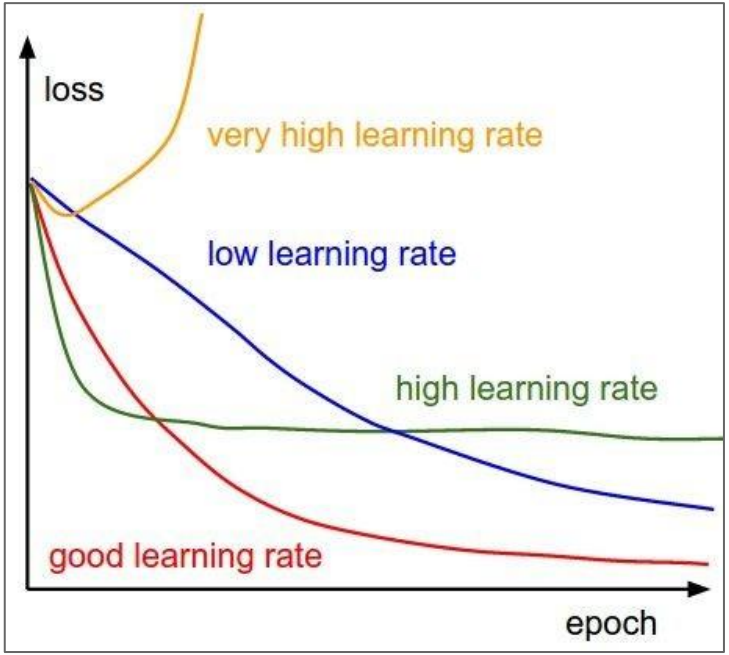
(image credits to Alec Radford)

SGD, SGD+Momentum, Adagrad, RMSProp, Adam sve imaju **korak učenja** kao hiperparametar.



Koji je od koraka učenja sa slike najbolji?

SGD, SGD+Momentum, Adagrad, RMSProp, Adam sve imaju **korak učenja** kao hiperparametar.



Očigledno je koja je kriva na slici najbolja, ali se ona neće dobiti sa fiksnom vrednošću koraka učenja

=> Korak učenja treba da opada kroz iteracije!

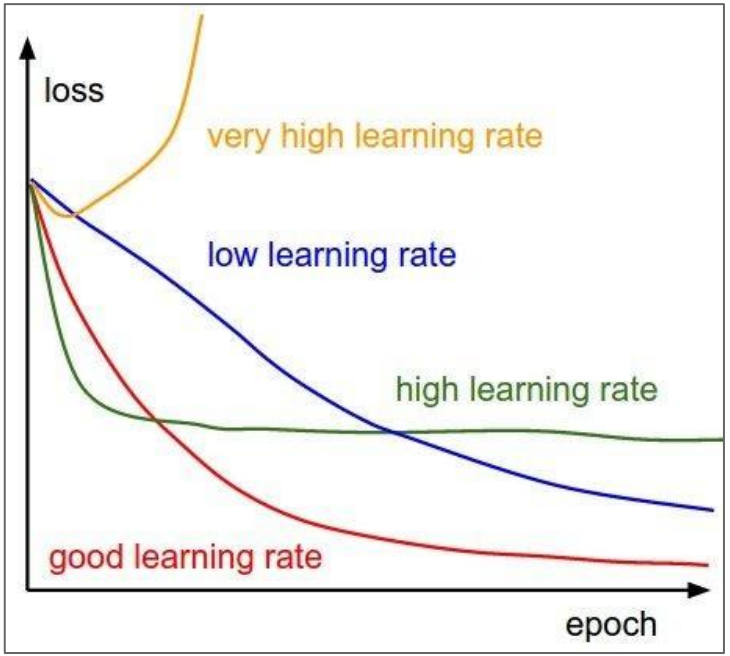
***opadanje po iteracijama (step decay)*:**
npr. delimo korak učenja sa 2 uvek nakon određenog broja epoha.

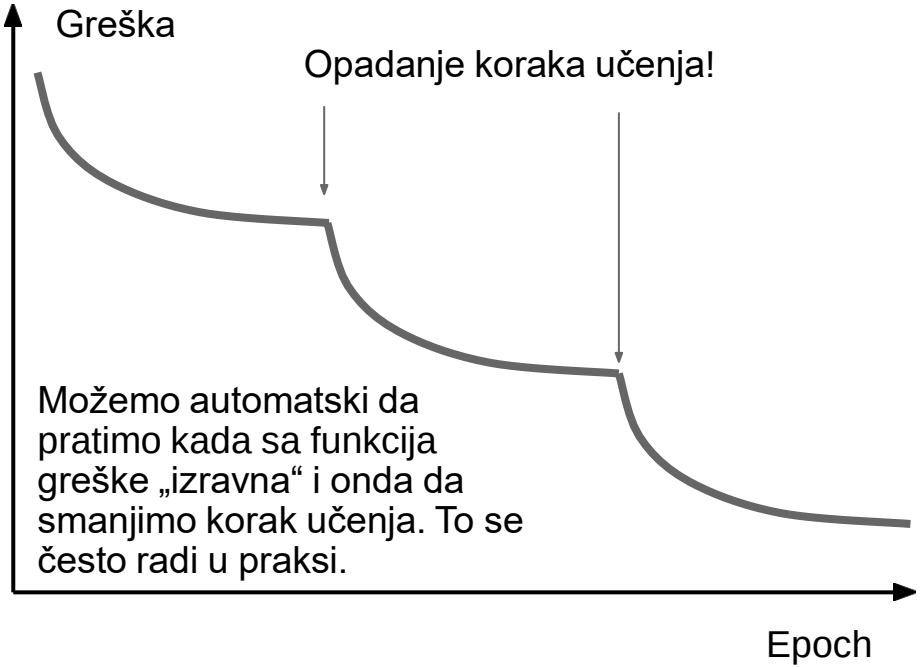
eksponencijalno opdanje: $\alpha = \alpha_0 e^{-kt}$

1/t opadanje:

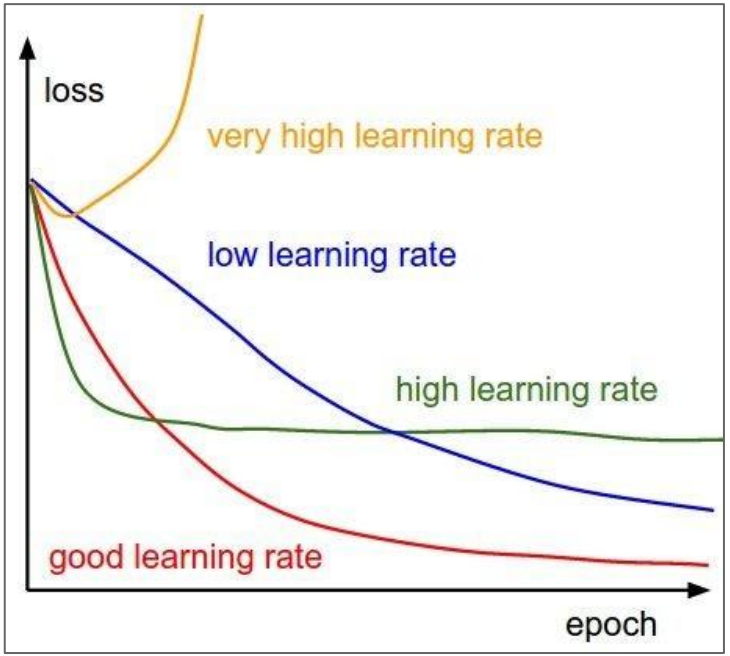
$$\alpha = \alpha_0 / (1 + kt)$$

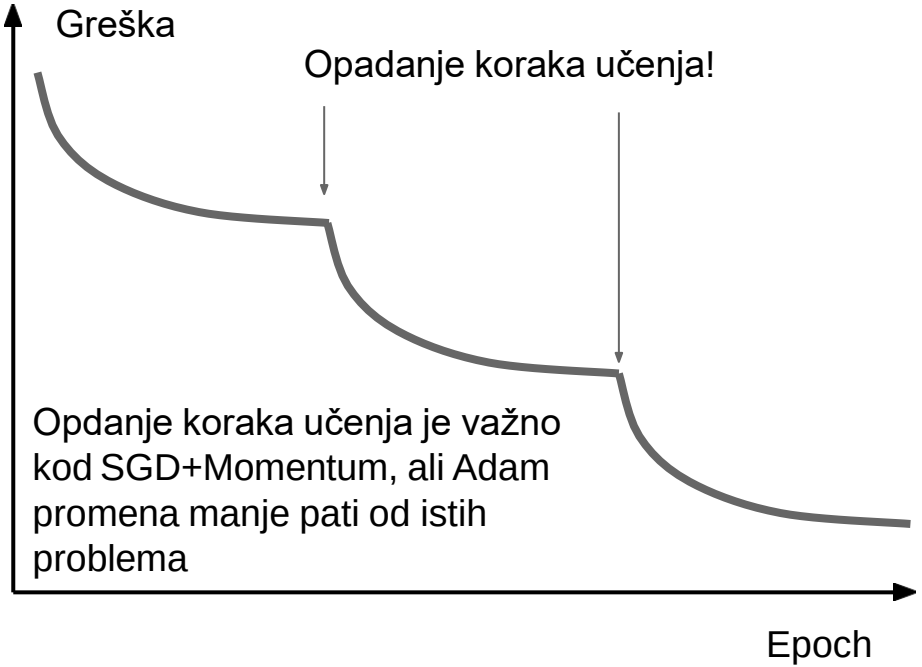
SGD, SGD+Momentum, Adagrad, RMSProp, Adam sve imaju **korak učenja** kao hiperparametar.





SGD, SGD+Momentum, Adagrad, RMSProp, Adam sve imaju **korak učenja** kao hiperparametar.



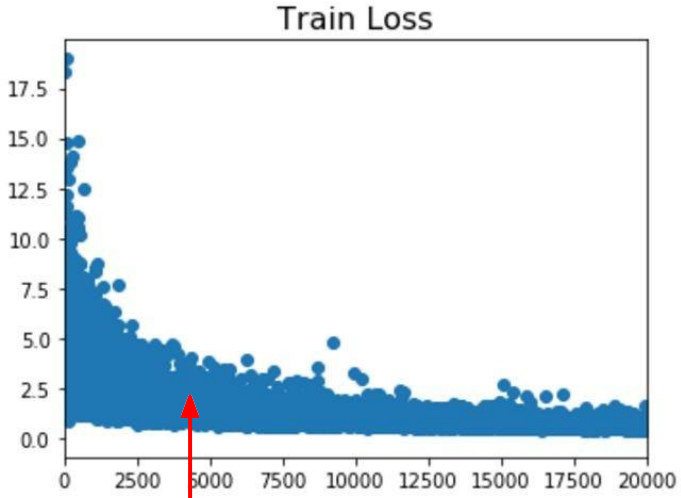


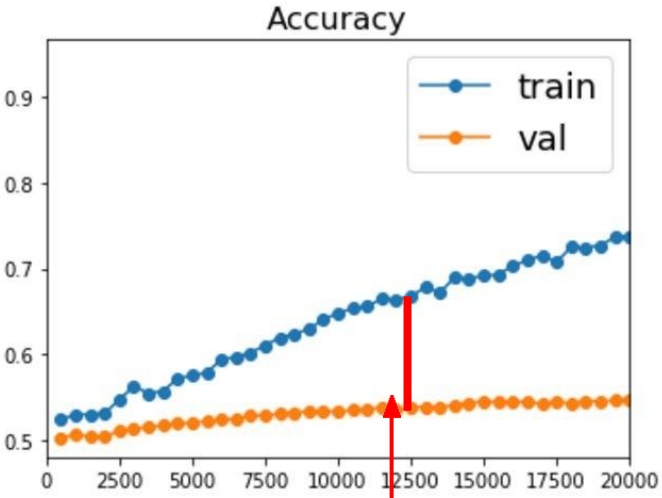
U praski:

- **Adam** je dobar izbor u većini slučajeva.

- Ako vam hardver dozvoli da promenu parametara radite posle celog ob. skupa, a ne mini podskupova onda probajte **L-BFGS** (ali imajte u vidu da je osetljiv na šum u podacima)
- **L-BFGS** je optmizacioni metod koji koristi drugi izvod, pomenut je samo da znate da postoji, nećemo se baviti njime na ovom kursu. Realno se retko koristi u Deep Learning.

Rešili smo training grešku, ali šta ako greška na validacionom skupu ipak ne opada?





Dobri optimizacioni algoritmi će smanjiti training rešku

Ali, mi želimo da model dobro radi na nepoznatim podacima. Kako da smanjimo raziku sa slike?

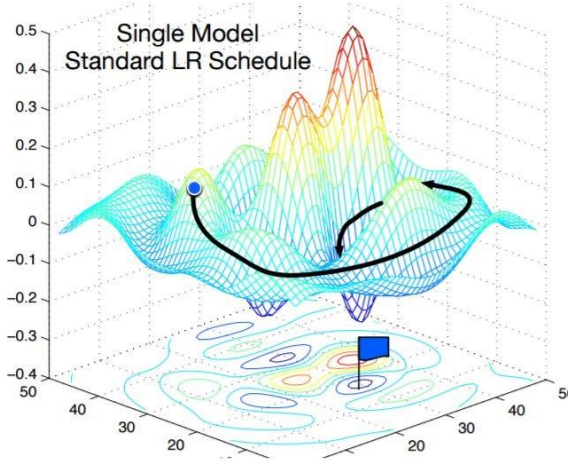
Ansambli Modela

1. Obučite više različitih nezavisnih modela na istim podacima
2. Kada se primenjuju uzmite prosek njihovih predikcija (ili neki bolji metod glasanja)

Uživaje u dodatnih 2% tačnosti ☺

Ansambli Modela: Saveti

Umesto da obučavate više nezavisnih modela, čuvamo neke od modela kroz iteracije (*snapshots*) i koristimo njih.



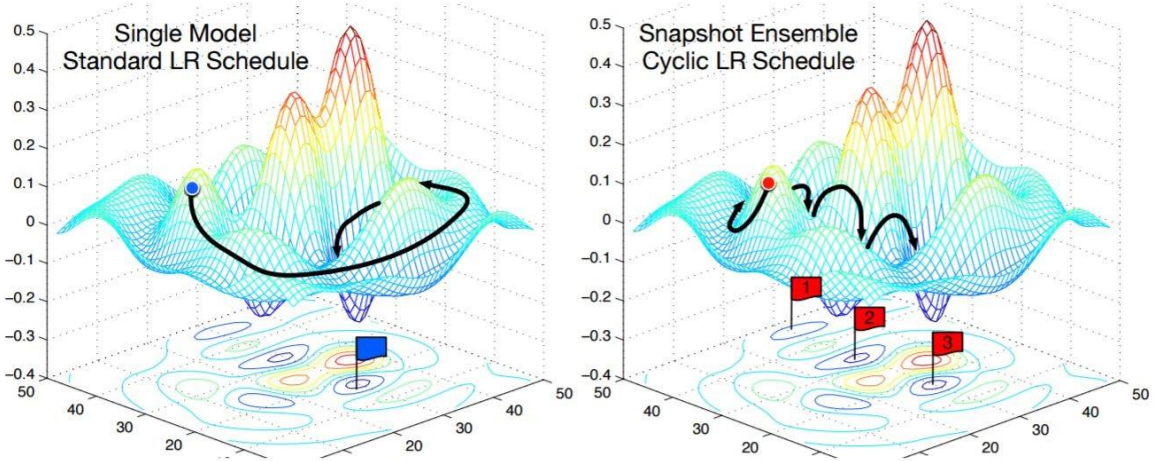
Na primer uzmemo model iz iteracije 1000, 2000, 3000,...., 10000

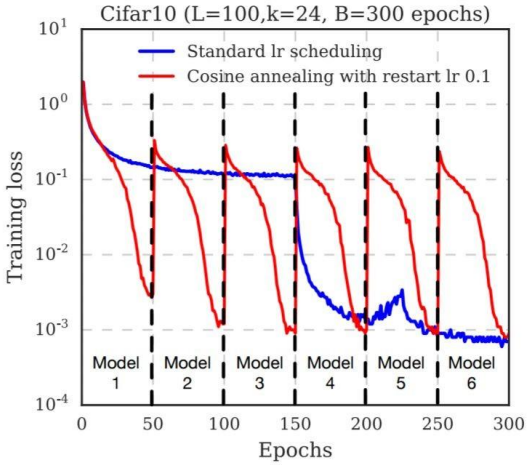
i onda imamo ansambl od 10 modela

Loshchilov and Hutter, “SGDR: Stochastic gradient descent with restarts”, arXiv 2016
Huang et al, “Snapshot ensembles: train 1, get M for free”, ICLR 2017
Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.

Ansambli Modela: Saveti

Umesto da obučavate više nezavisnih modela, čuvamo neke od modela kroz iteracije (*snapshots*) i koristimo njih.

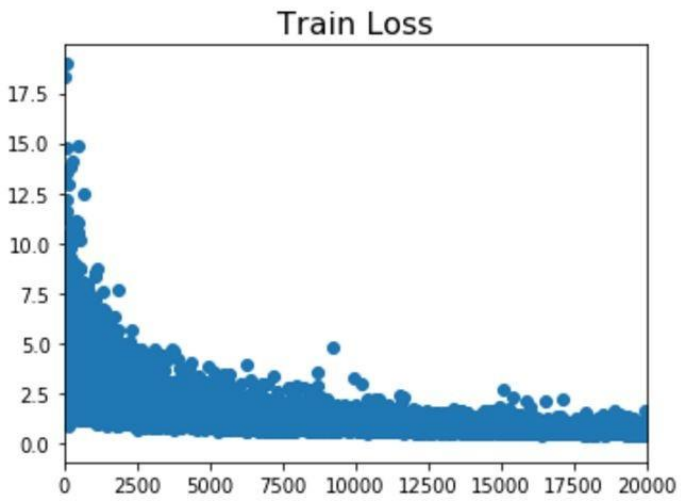


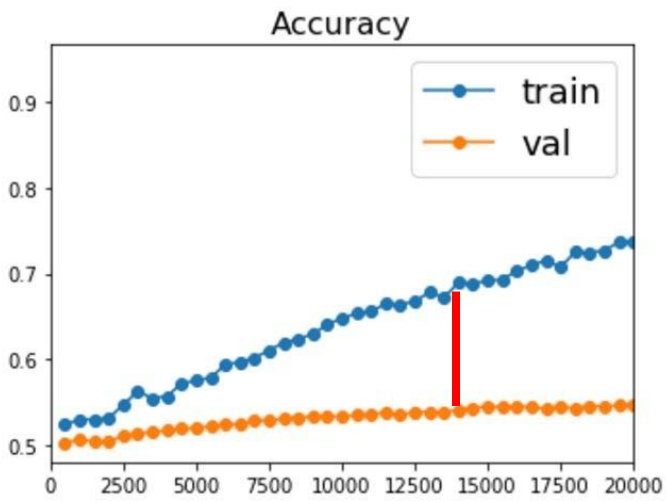


Loshchilov and Hutter, "SGDR: Stochastic gradient descent with restarts", arXiv 2016
Huang et al, "Snapshot ensembles: train 1, get M for free", ICLR 2017
Figures copyright Yixuan Li and Geoff Pleiss, 2017. Reproduced with permission.

ciklične promene koraka učenja mogu da budu dobar izvor ansambla!

Kako dodatno poboljšati performanse?





Regularizacija

Regularizacija: Dodajemo još jedan izraz u funckiju greške

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i;W)_j - f(x_i;W)_{y_i} + 1) + \boxed{\lambda R(W)}$$

Obično se koristi:

L2 regularizacija $R(W) = \sum_k \sum_l W_{k,l}^2$ (Termin je *Weight decay*)

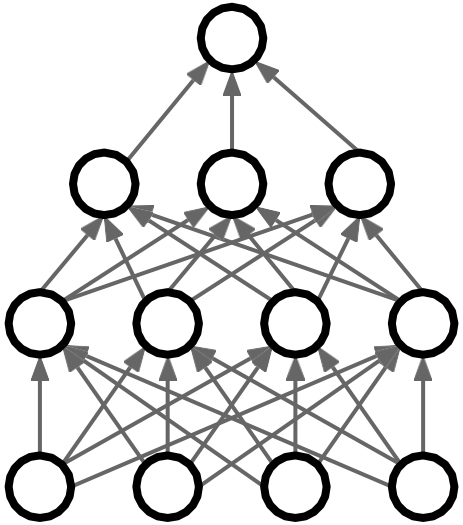
L1 regularizacija $R(W) = \sum_k \sum_l |W_{k,l}|$

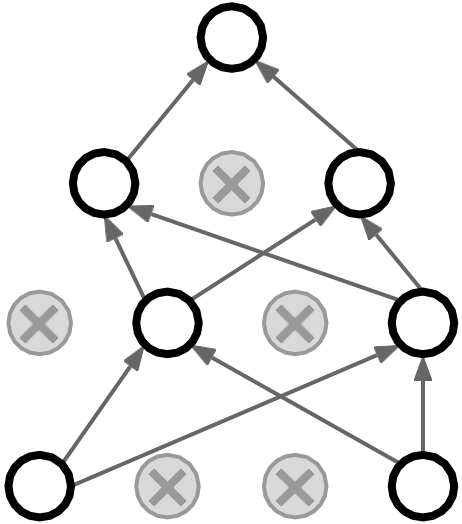
Elastic net (L1 + L2) $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

Regularizacija: Dropout

Pri svakoj propagaciji unapred, na slučajan način isključiti neke od neurona;

Verovatnoća da će neuron biti isključen je hiper parametar; 0.5 je uobičajena vrednost



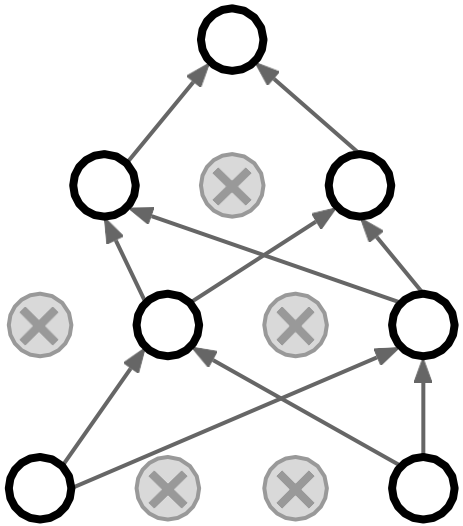


Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

Regularizacija: Dropout

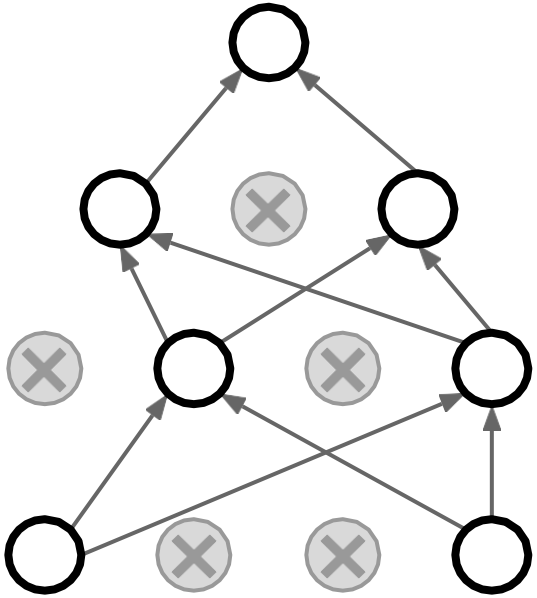
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)
```

Primer: propagacija unapred za 3-slojnu mrežu uz dropout

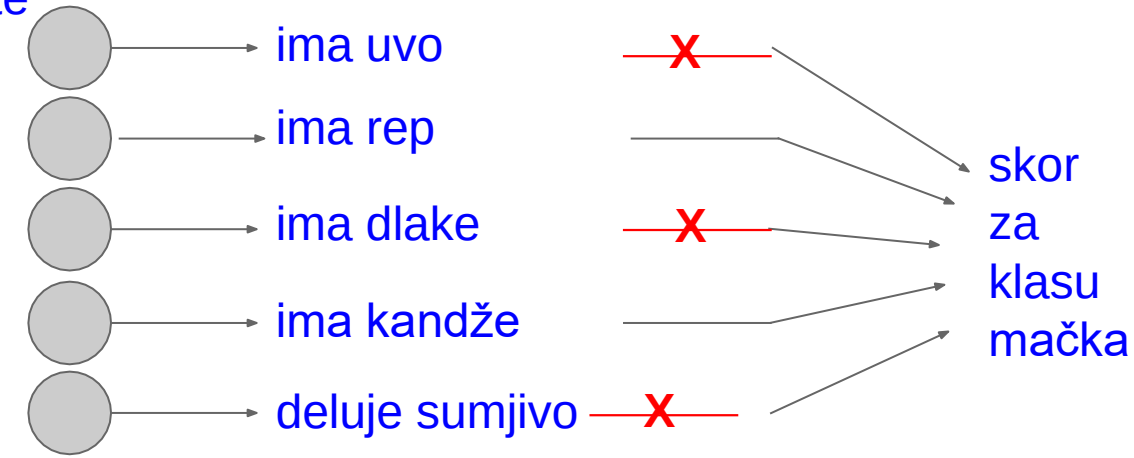


Regularizacija: Dropout

Kako ovo može biti dobra ideja?

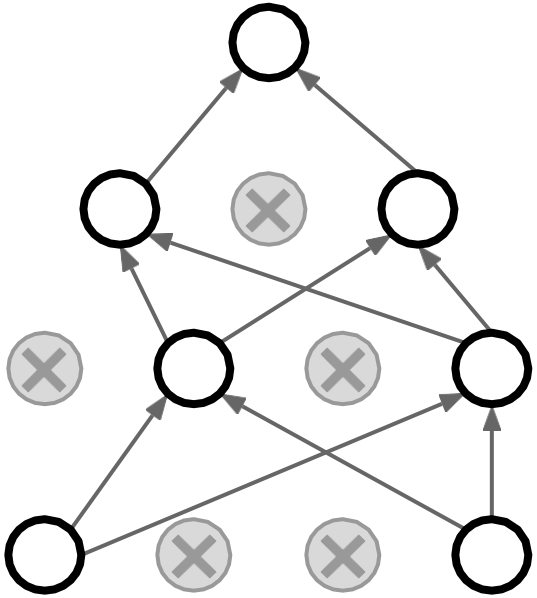


Nateraće mrežu da ima redudantne načine za predikciju iste stvari; Sprečava previše veliko oslanjanje na određene atribute



Regularizacija: Dropout

Kako ovo može biti dobra ideja?



Još jedna interpretacija:

Dropout je obučavanje **ansambla modela** (koji dele parametre).

Svaka binarna maska je jedan model

Potpuno povezan sloj sa 4096 neurona ima $2^{4096} \sim 10^{1233}$ mogućih maski! ~ 10^{82} atoma u svemiru…

Dropout: Primena na nepoznate primere

Dropout će učiniti izlaz nepredvidivm jer kad primenjujemo mrežu na nepoznate primere koristimo sve neurone.

Izlaz (klasa) — ulaz (slika) — Random maska

$$\boxed{y} = f_W(\boxed{x}, \boxed{z})$$

To ćemo rešiti tako što ćemo da „uprosečimo“ slučajnost

prosečna aktivacija koju je neuron imao tokom obučavanja

$$y = f(x) = E_z\left[f(x,z)\right] = \int p(z) f(x,z) dz$$

aktivacija

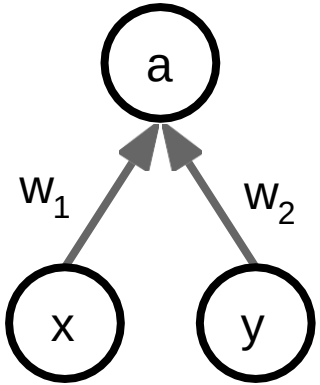
verotnoća da je neuron aktivan

Naravno da nećemo da rešavamo integral …

Dropout: Primena na nepoznate primere

Integral ćemo aproskimirati emprijskim prosekom

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$



Posmatrajmo jedan neuron.

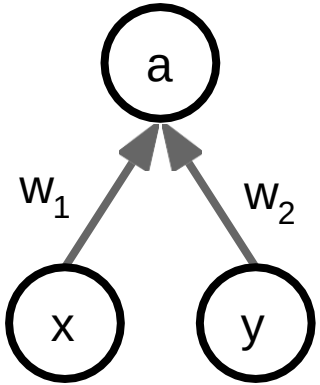
Dropout: Primena na nepoznate primere

Integral ćemo aproskimirati emprijskim prosekom

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$

Posmatrajmo jedan neuron.

Prilikom primene na nepoznati primer imamo: $E\big[a\big] = w_1 x + w_2 y$



Dropout: Primena na nepoznate primere

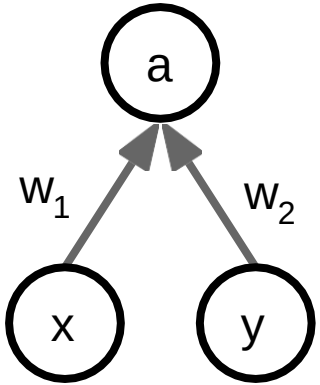
Integral ćemo aproskimirati emprijskim prosekom

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$

Posmatrajmo jedan neuron.

Prilikom primene na nepoznati primer imamo:

$$E\big[a\big] = w_1 x + w_2 y$$



Prilikom obučavanja imamo:

$$\begin{aligned} E\big[a\big] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

Dropout: Primena na nepoznate primere

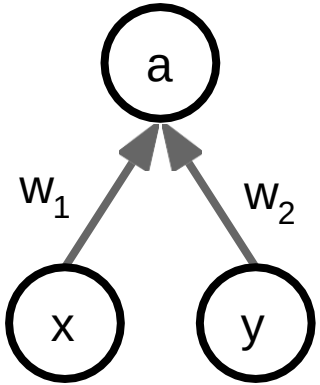
Integral ćemo aproskimirati emprijskim prosekom

$$y = f(x) = E_z\big[f(x,z)\big] = \int p(z) f(x,z) dz$$

Posmatrajmo jedan neuron.

Prilikom primene na nepoznati primer imamo:

$$E\big[a\big] = w_1 x + w_2 y$$



Prilikom obučavanja imamo:

$$\begin{aligned} E\big[a\big] =& \frac{1}{4}(w_1 x + w_2 y) + \frac{1}{4}(w_1 x + 0y) \\ &+ \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2 y) \\ =& \frac{1}{2}(w_1 x + w_2 y) \end{aligned}$$

To znači da prilikom primene na nepoz. množimo aktivaciju sa verovatnoćom za dropout. Ovde je to 0.5

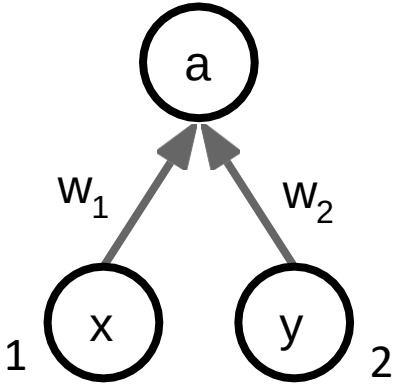
Dropout: Primena na nepoznate primere

<u>Još jedno objašnjenje:</u>

Posmatrajmo recimo neuron sa slike dole.

Očekivana aktivacija tog neurona prilikom obučavanja je $p*x+(1-p)*0=px$.

Dakle kada vršimo primenu na nepoznate primere onda moramo da aktivaciju neurona pomnožimo sa p.



Dropout: Primena na nepoznate primere

```
def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

Prilikom primene na nepoz. prim. svi neuroni su uključeni
=> Moramo da skaliramo aktivacije svih neurona:
<u>izlaz prilikom primene na nepoz. prim.</u> = <u>očekivani izlaz prilikom obučavanja</u>

Dropout Rezime

```
""" Vanilla Dropout: Not recommended implementation (see notes below) """

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  """ X contains the data """

  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = np.random.rand(*H1.shape) < p # first dropout mask
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = np.random.rand(*H2.shape) < p # second dropout mask
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations
  H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations
  out = np.dot(W3, H2) + b3
```

dropout u propagaciji unapred

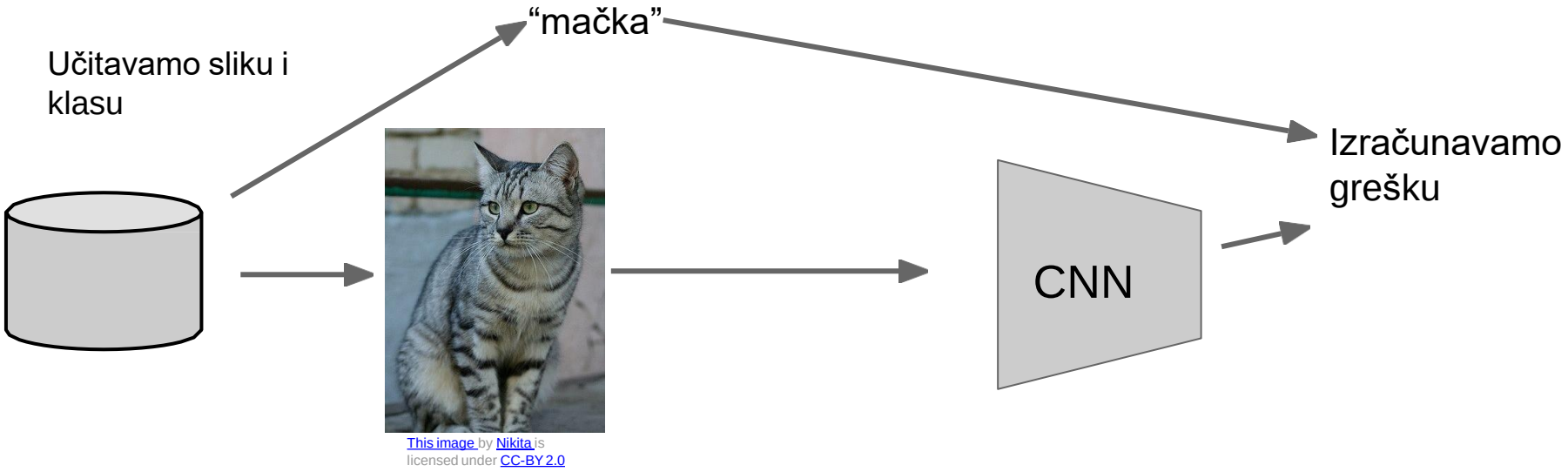
skaliramo kad radimo predikciju

Često se koristi: “Invertovani dropout”

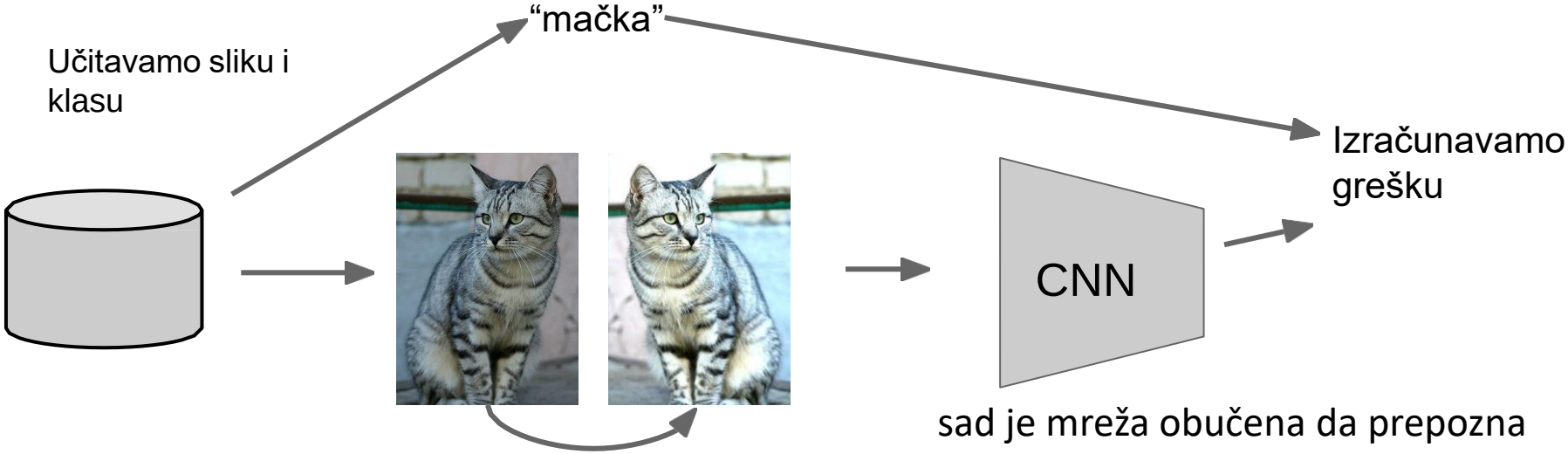
```
p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

ovde je aktivacija nepromenjena!

Regularizacija: Obogaćivanje Skupa Podataka (*Data Augmentation*)



Regularizacija: Obogaćivanje Skupa Podataka (*Data Augmentation*)



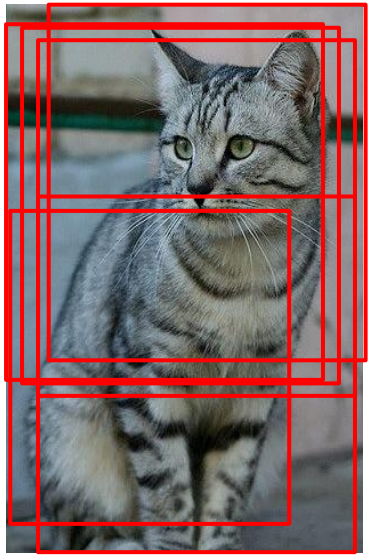
sad je mreža obučena da prepozna istu mačku, čak iako je slika obrnuta. Time imamo bolju generalizaciju.

Obogaćivanje Skupa Podataka

Horizontalno Obrtanje

Obogaćivanje Skupa Podataka

Slučajno kropovanje i skaliranje



Obučavanje: Slučajno kropovanje i skaliranje

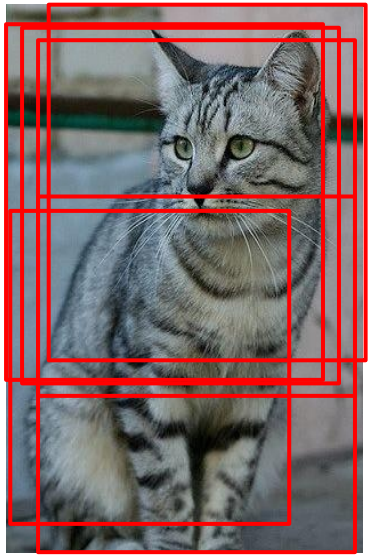
ResNet:

1. Odaberi slučajno L u rasponu [256, 480]
2. Skaliraj sliku tako da manja strana bude L
3. Sempluj slučajno deo dimenzija 224 x 224 iz originalne slike

2. i 3. su dve odvojene stvari, tako u obučavajući skup dodajemo malo reskaliranih primera jedne slike, a takođe i malo nekih njenih delova.

Obogaćivanje Skupa Podataka

Slučajno kropovanje i skaliranje



Obučavanje: Slučajno kropovanje i skaliranje

ResNet:

1. Odaberi slučajno L u rasponu [256, 480]
2. Skaliraj sliku tako da manja strana bude L
3. Sempluj slučajno deo dimenzija 224 x 224 iz slike

Primena: klasa je prosek predikcija koje mreža da za 1. i 2. dole

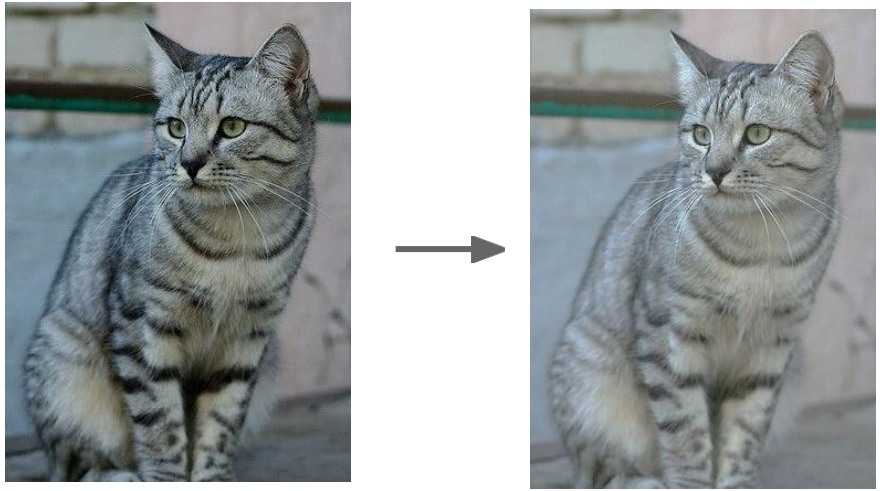
ResNet:

1. Skaliramo sliku u ovih 5 veličina: {224, 256, 384, 480, 640}
2. Za svaku veličinu uzimamo 10 kropova dim. 224 x 224: 4 ugla + center, + obrtanja

Obogaćivanje Skupa Podataka

Male promene boje

Jednostavno: Slučajno promenimo osvetljenje i kontrast



Postoje i kompleksnije metode, ali ih nećemo obrađivati na ovom kursu.

Obogaćivanje Skupa Podataka

Nema pravila, budite kreativni sa svojim podacima i problemom na kome radite!

Neki slučajan miks:

- translaacije
- rotacije
- istezanja,
- drugih distorzija, … (zavisi šta očekujete od primera na koje ćete da primenjujete mrežu)

Učenje Transferom
Transfer Learning

"Treba nam puno podataka da obučimo
Konvolutivnu Neuronsku Mrežu"

Učenje Transferom

Transfer Learning

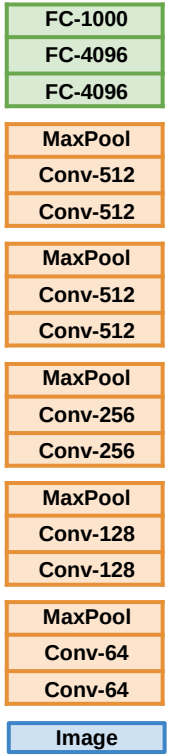
“Treba nam puno podataka da obučimo Konvolutivnu Neuronsku Mrežu”

NETAČNO

Učenje Transferom - CNN

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

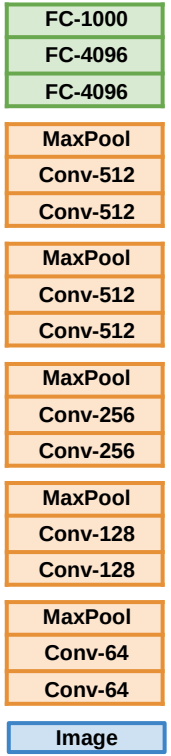
1. Obučiti na Imagenet



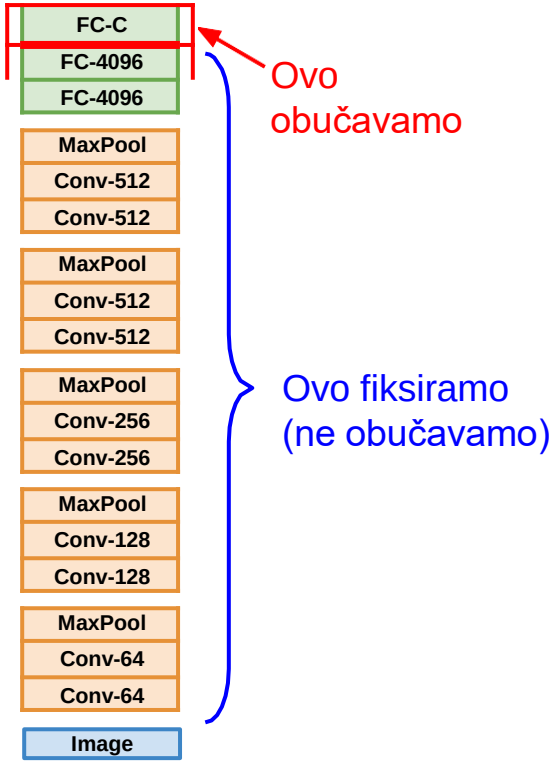
Učenje Transferom - CNN

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014

1. Obučiti na Imagenet



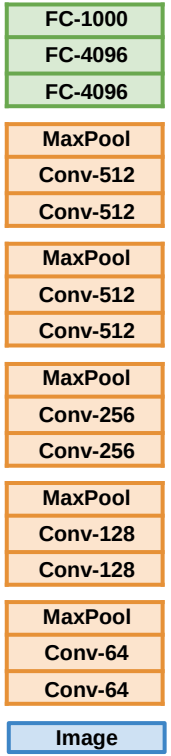
2. Mali skup podataka (C klasa)



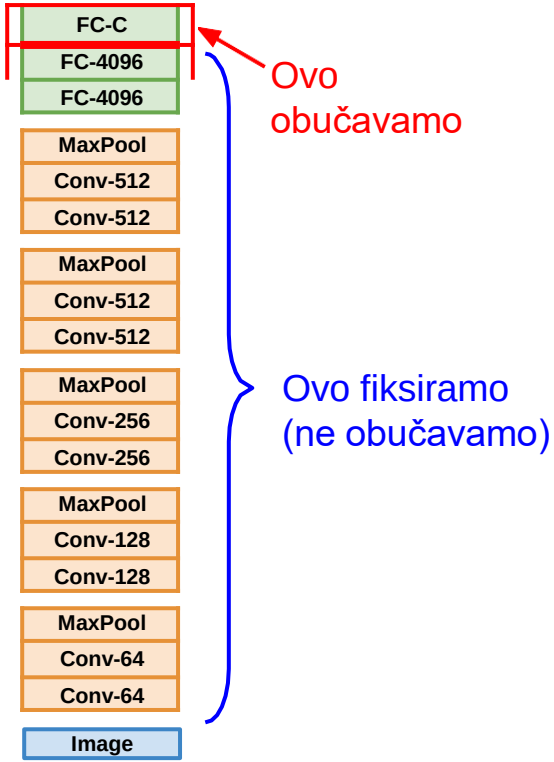
Učenje Transferom - CNN

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

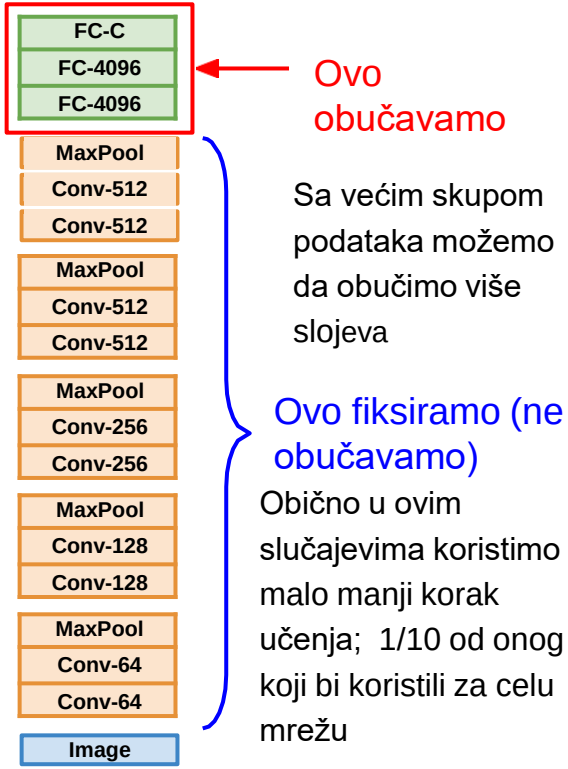
1. Obučiti na Imagenet



2. Mali skup podataka (C klasa)

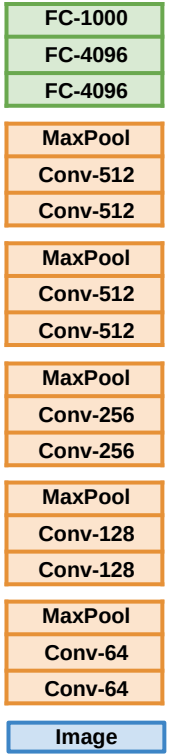


3. Malo veći skup podataka



Sa većim skupom podataka možemo da obučimo više slojeva

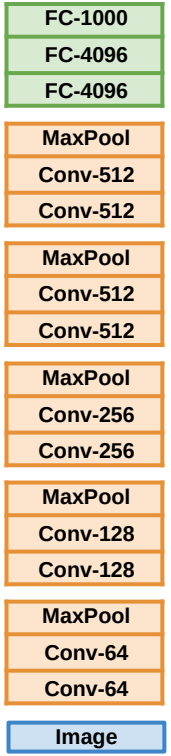
Obično u ovim slučajevima koristimo malo manji korak učenja; 1/10 od onog koji bi koristili za celu mrežu



Slojevi za specifične karakteristike slika

Slojevi za opštije karakteristike

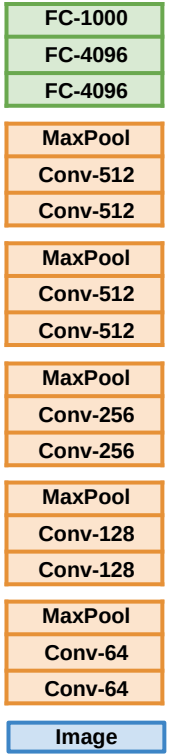
	vrlo sličan obučavajući skup	**vrlo drugačiji obučavajući skup**
mali obučavajući skup	?	?
veliki obučavajući skup	?	?



Slojevi za specifične karakteristike slika

Slojevi za opštije karakteristike

	vrlo sličan obučavajući skup	**vrlo drugačiji obučavajući skup**
mali obučavajući skup	upotrebiti linearni klasifikator umesto gornjeg sloja	?
veliki obučavajući skup	obučavati samo par slojeva na vrhu	?



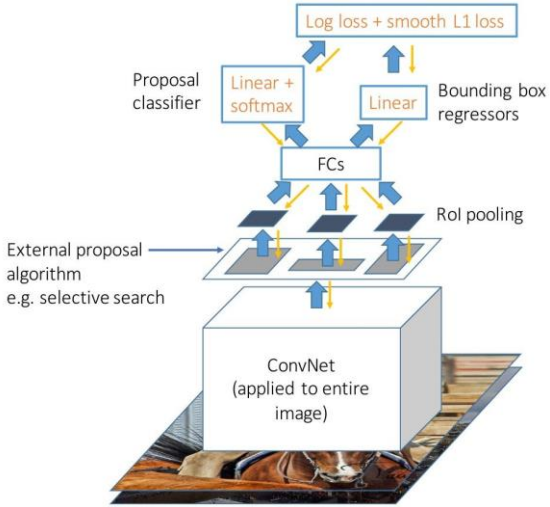
Slojevi za specifične karakteristike slika

Slojevi za opštije karakteristike

	vrlo sličan obučavajući skup	**vrlo drugačiji obučavajući skup**
mali obučavajući skup	upotrebiti linearni klasifikator umesto gornjeg sloja	Ovde nema mnogo pomoći... možda ansambli lin. klas. obučenih u različitim delovima mreže
veliki obučavajući skup	obučavati samo par slojeva na vrhu	obučavati samo više slojeva na vrhu

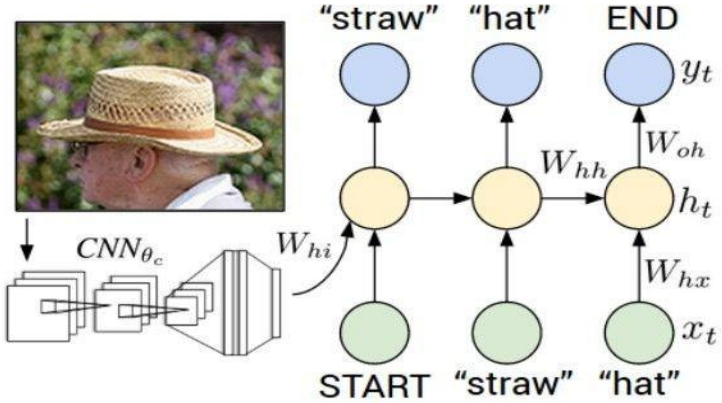
Učenje transferom kod CNN je vrlo često…
(to je više standardno, nego izuzetak)

Detekcija objekata (Fast R-CNN)



Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

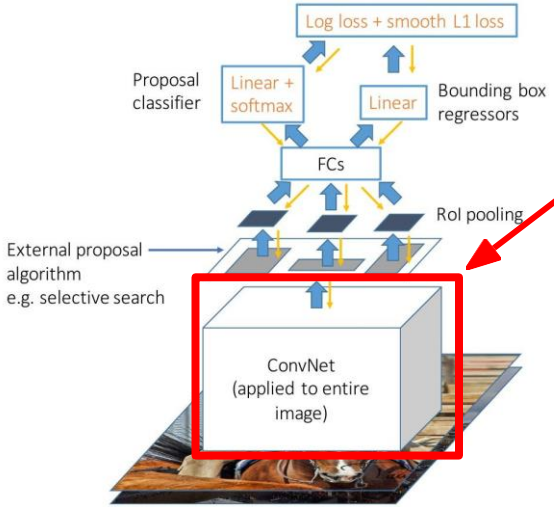
Dodeljivanje objašenja slikama:
CNN + RNN



Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

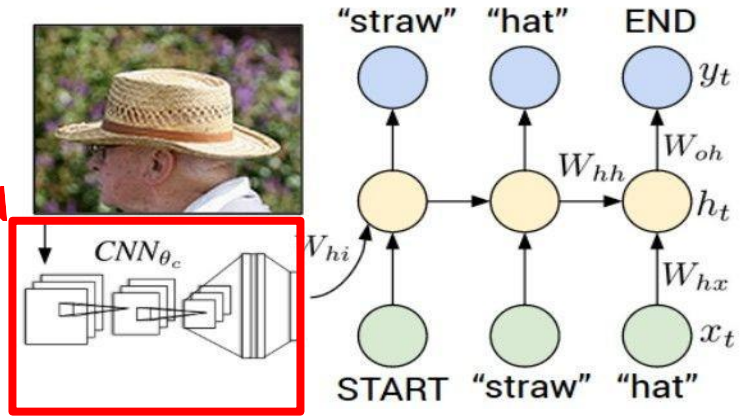
Učenje transferom kod CNN je vrlo često…
(to je više standardno, nego izuzetak)

Detekcija objekata (Fast R-CNN)



CNN ranije obučena na Imagenet

Dodeljivanje objašenja slikama: CNN + RNN

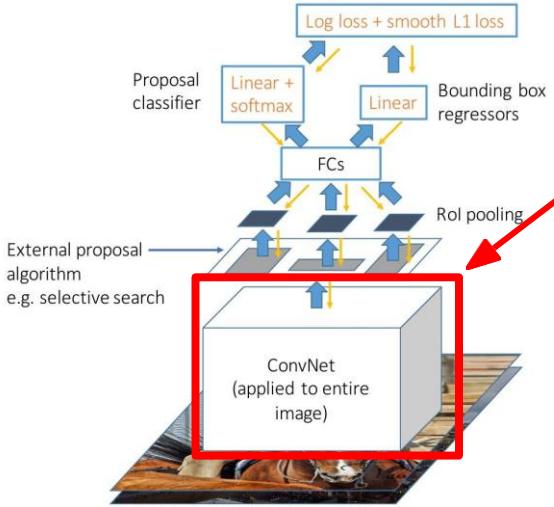


Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

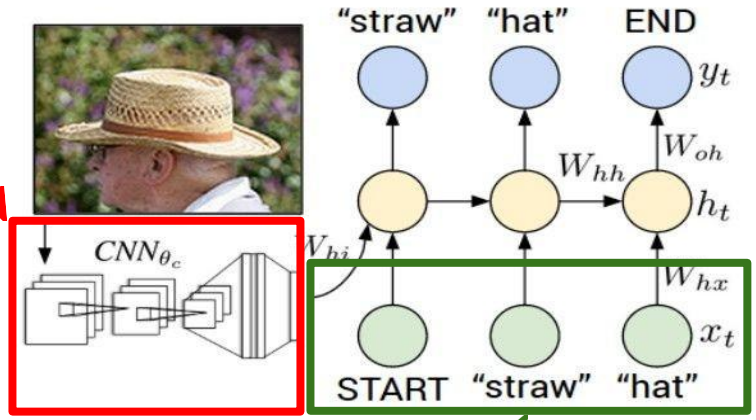
Učenje transferom kod CNN je vrlo često…
(to je više standardno, nego izuzetak)

Detekcija objekata (Fast R-CNN)



CNN ranije obučena na Imagenet

Dodeljivanje objašenja slikama: CNN + RNN



Vektor za svaku reč obučeni pomoću word2vec

Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Savet:

Imate interesantan ob. skup slika, ali ima < ~1M primera?

1. Naći veliki skup podataka koji ima slične slike, obučiti veliku CNN na tom skupu
2. Uraditi učenje transferom na vašem manjem skupu

Deep learning okruženja pružaju tzv. “Model Zoo” raznih modela obučenih na velikom broju ob. skupova (*pretrained models*)

Caffe: https://github.com/BVLC/caffe/wiki/Model-Zoo

TensorFlow: https://github.com/tensorflow/models

PyTorch: https://github.com/pytorch/vision

Rezime

- Optimizacija
  - Momentum, RMSProp, Adam, ...
- Regularizacija
  - Dropout, ...
- Učenje tranferom
  - Treba koristiti ako je moguće!

Rekurentne Neuronske Mreže
Recurrent Neural Networks, RNN

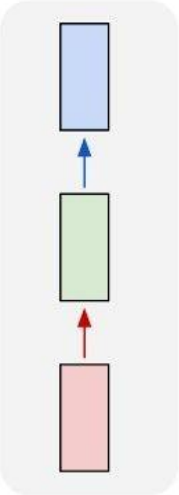
Predavač: Aleksandar Kovačević

Slajdovi preuzeti sa CS 231n, Stanford

http://cs231n.stanford.edu/

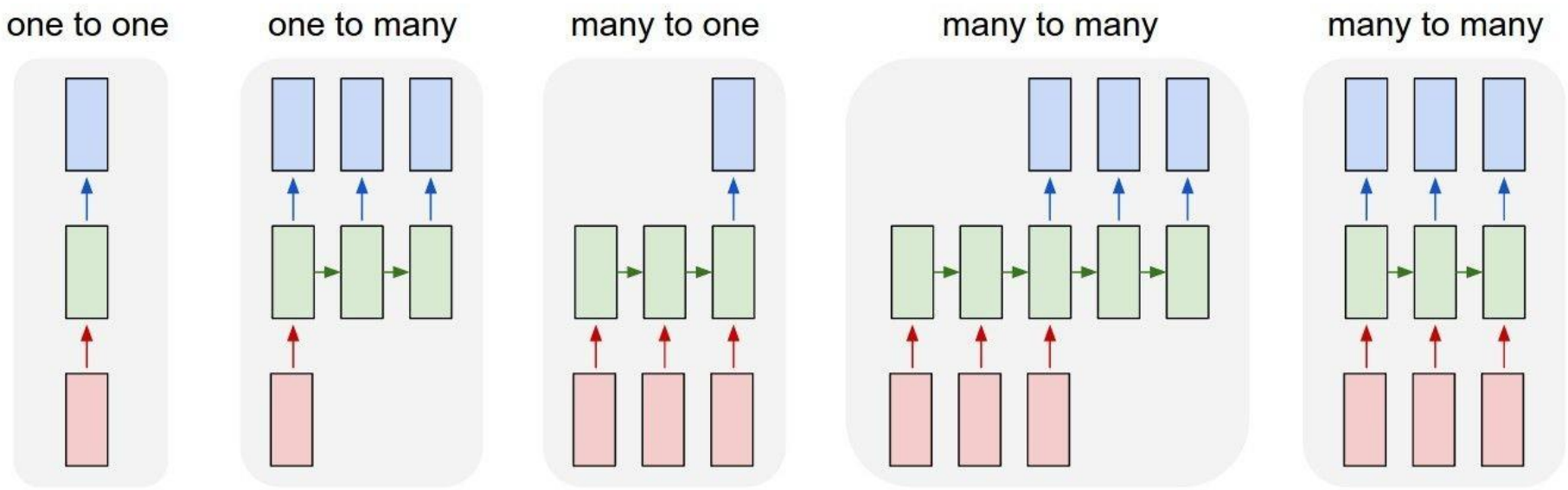
“Obična” Neuronska Mreža

one to one



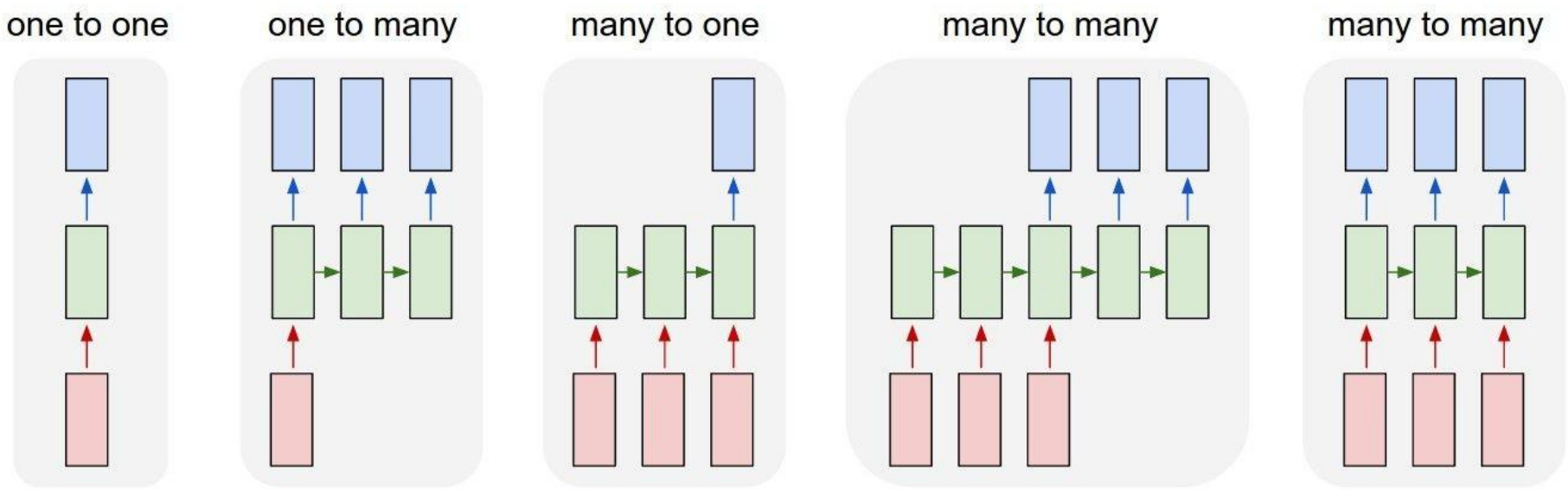
Obična Neuronska Mreža

Rekurentne Neuronske Mreže: Procesiranju sekvence ulaza



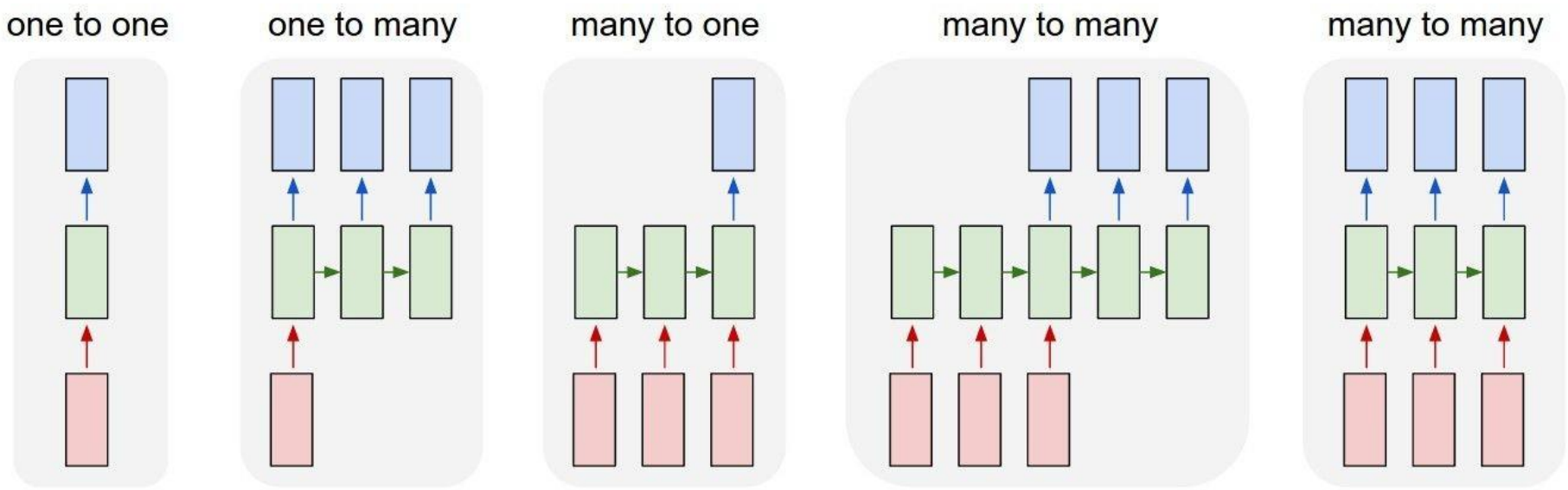
npr. **Dodela opisa slici**
slika -> sekvenca reči

Rekurentne Neuronske Mrežae: Procesiranju sekvence ulaza



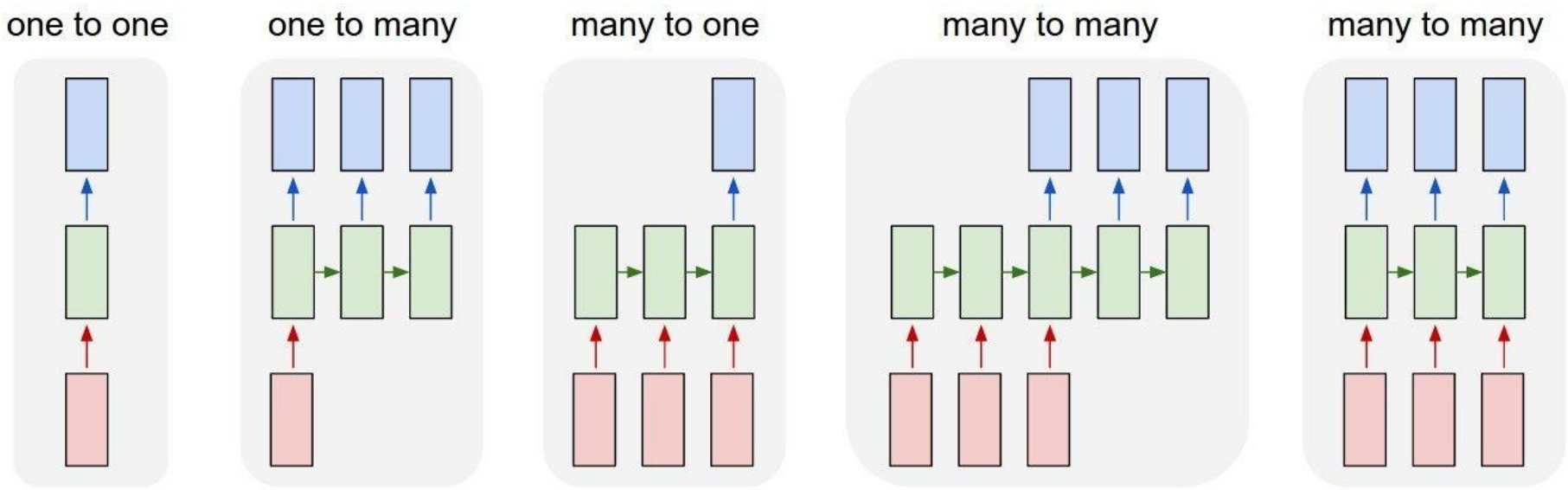
npr. **Detekcija Sentimenta**
sekvenca reči -> sentiment

Rekurentne Neuronske Mrežae: Procesiranju sekvence ulaza



npr. **Mašinski Prevod**
sek. reči -> sek. reči

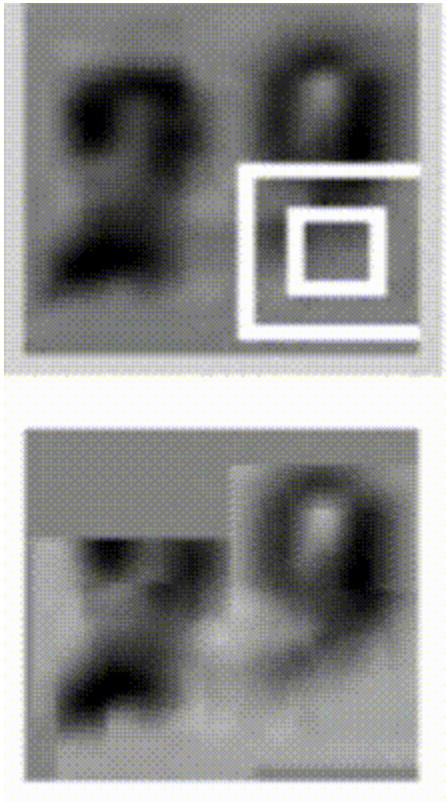
Rekurentne Neuronske Mrežae: Procesiranju sekvence ulaza



npr. **Klasifikacija videa na nivou frejma**

Sekvencijalno Procesiranje Ne-Sekvencijalnih Podataka

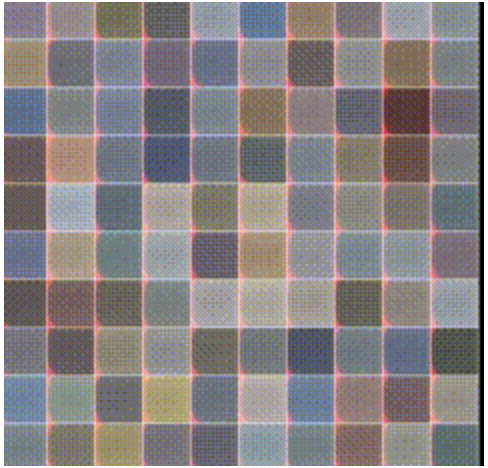
Klasifikacija slika tako što se uradi sekvenca “malih pogleda - *glimpses*” na sliku



Slika preuzeta sa: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

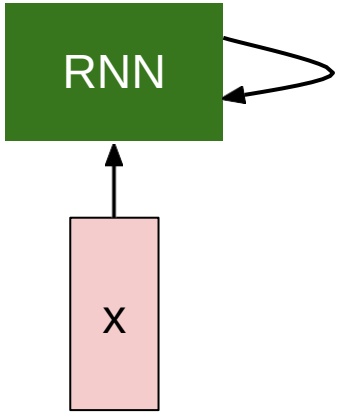
Sekvencijalno Procesiranje Ne-Sekvencijalnih Podataka

Generisanje slika!

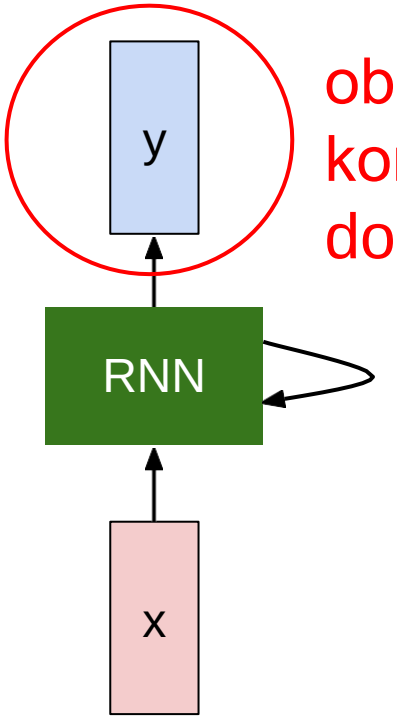


Slika preuzeta sa: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Rekurentna Neuronska Mreža



Rekurentna Neuronska Mreža



obično u nekom od koraka želimo i da dobijemo izlaz

Rekurentna Neuronska Mreža

Sekvencu vektora **x** procesiramo tako što primenjujemo **rekurentnu formulu** u svakom vremenskom koraku:

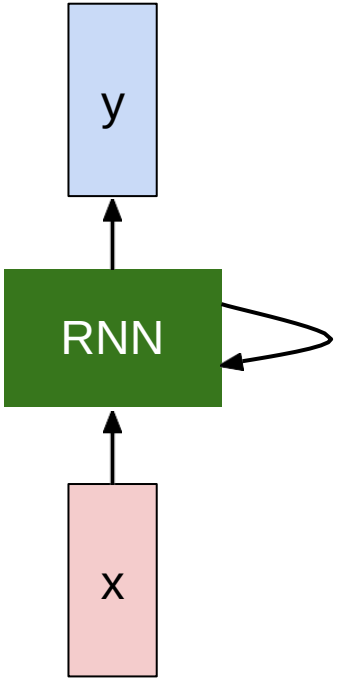
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

novo stanje

staro stanje

ulazni vektor u nekom vremeskom koraku

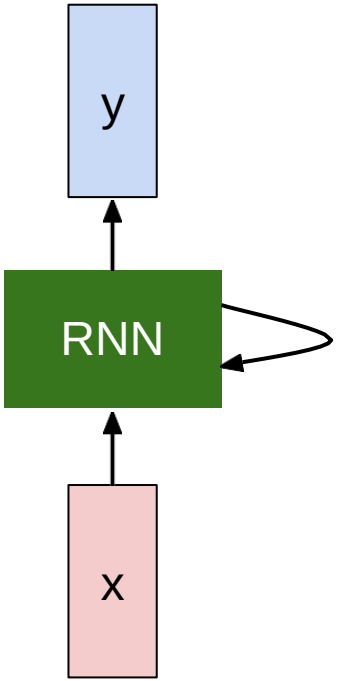
Neka funkcija sa parametrima W



Rekurentna Neuronska Mreža

Sekvencu vektora **x** procesiramo tako što primenjujemo **rekurentnu formulu** u svakom vremenskom koraku:

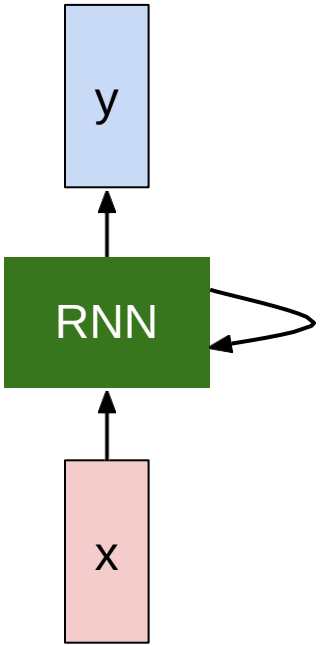
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$



Važno je primetiti: ista funkcija f, sa istim parametrima W primenjuje se u svakom vremeskom koraku. Dakle ne menja se kroz vreme.

(Obična) Rekurentna Neuronska Mreža

Stanje mreže sastoji se od samo jednog *"skrivenog"* vektora **h**:



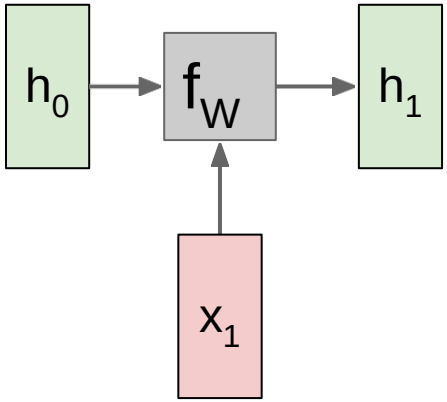
$$h_t = f_W(h_{t-1}, x_t)$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

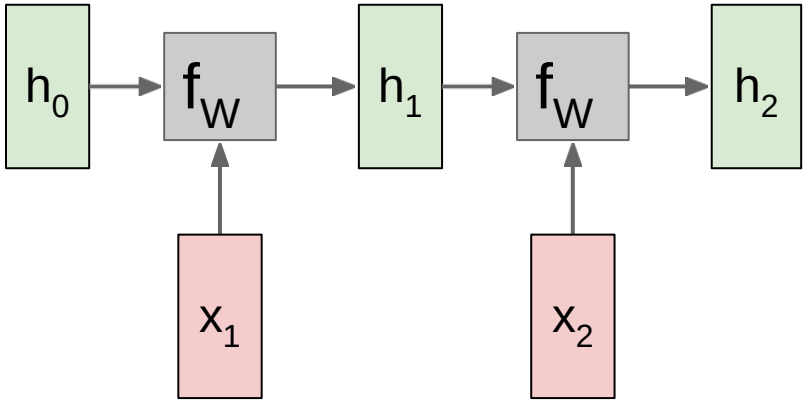
$$y_t = W_{hy}h_t$$

<u>Napomena</u>: često se koristi i bias za h, kao i za y tj. b_h i b_y, ali su radi jednostavnijeg objašnjavanja izostavljeni.

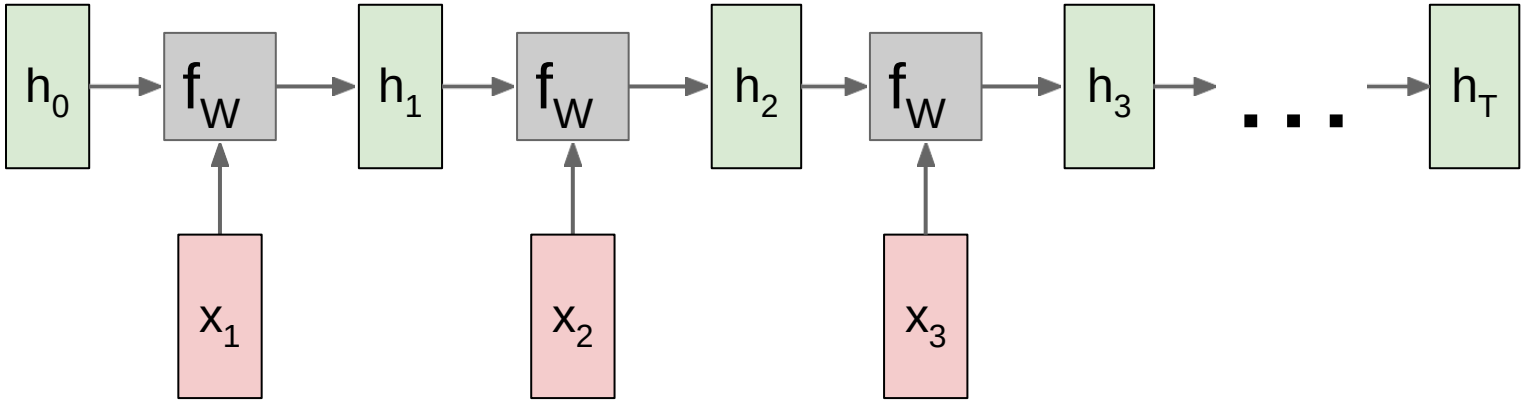
RNN: Graf Izračuavanja



RNN: Graf Izračuavanja

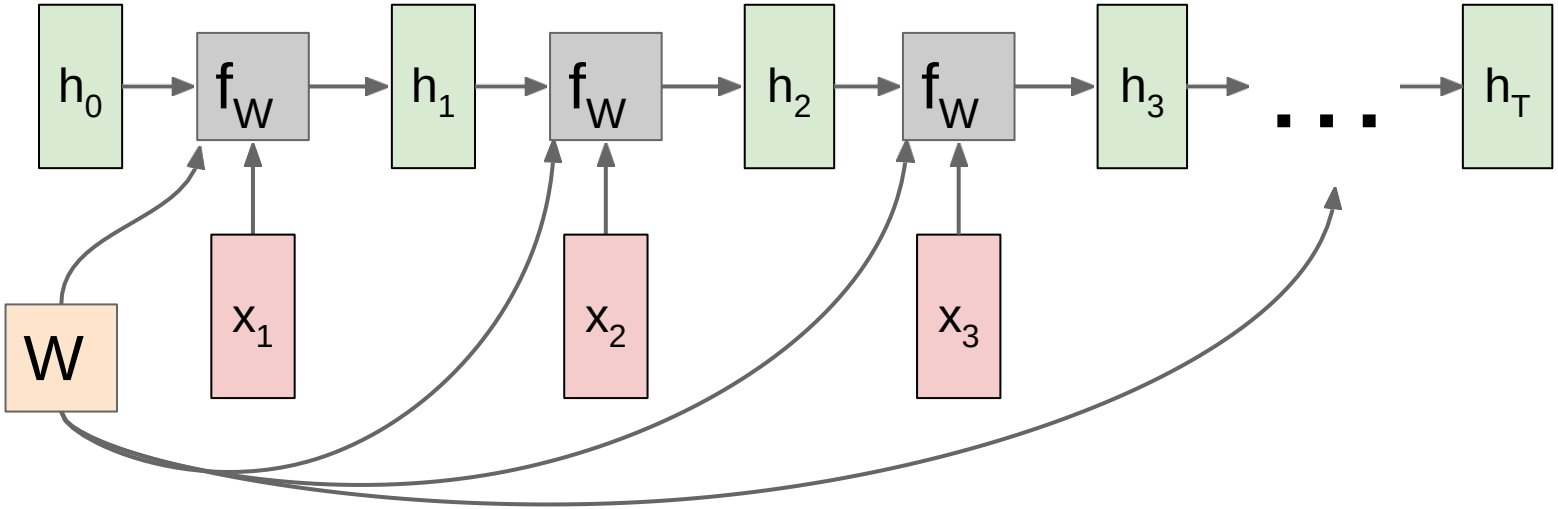


RNN: Graf Izračuavanja

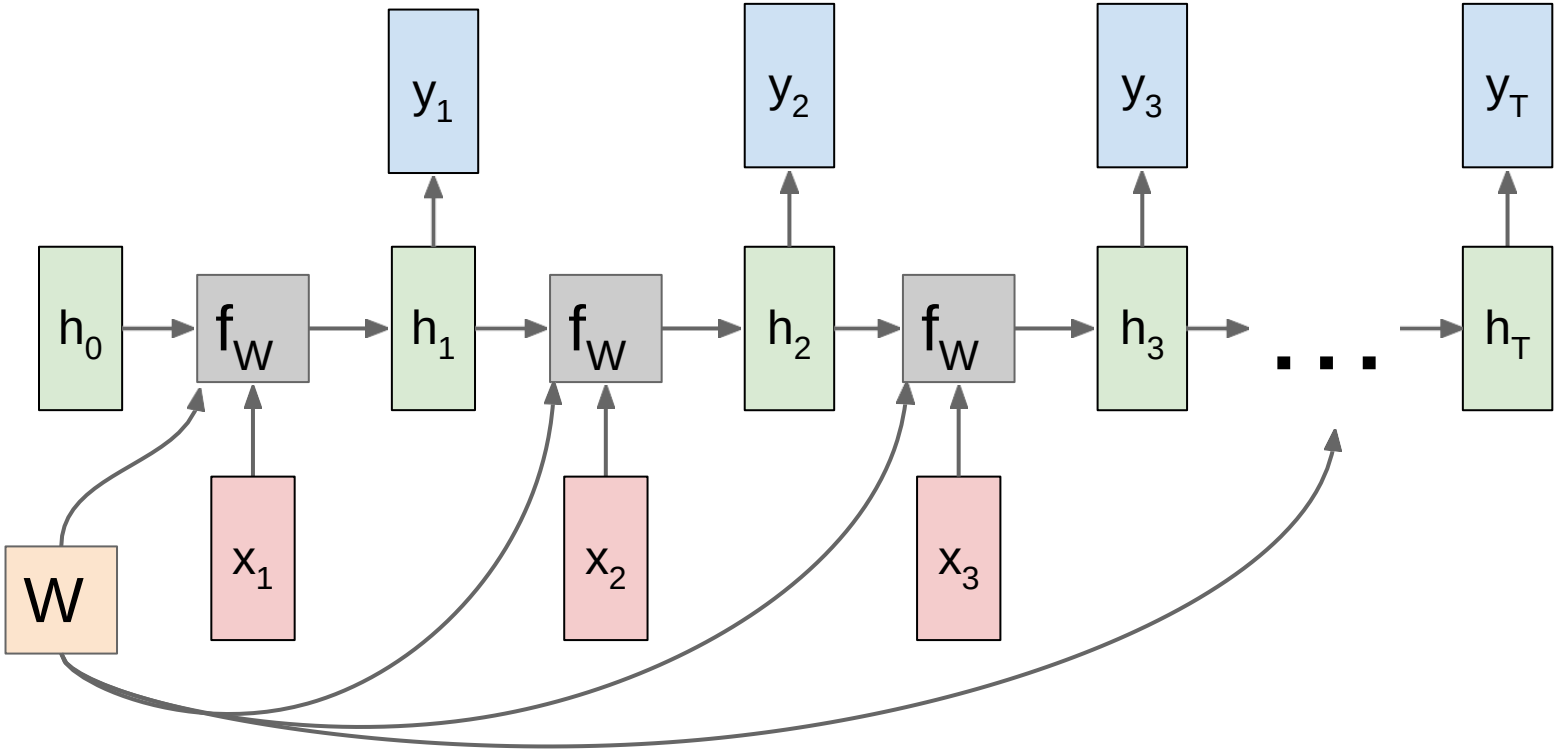


RNN: Graf Izračuavanja

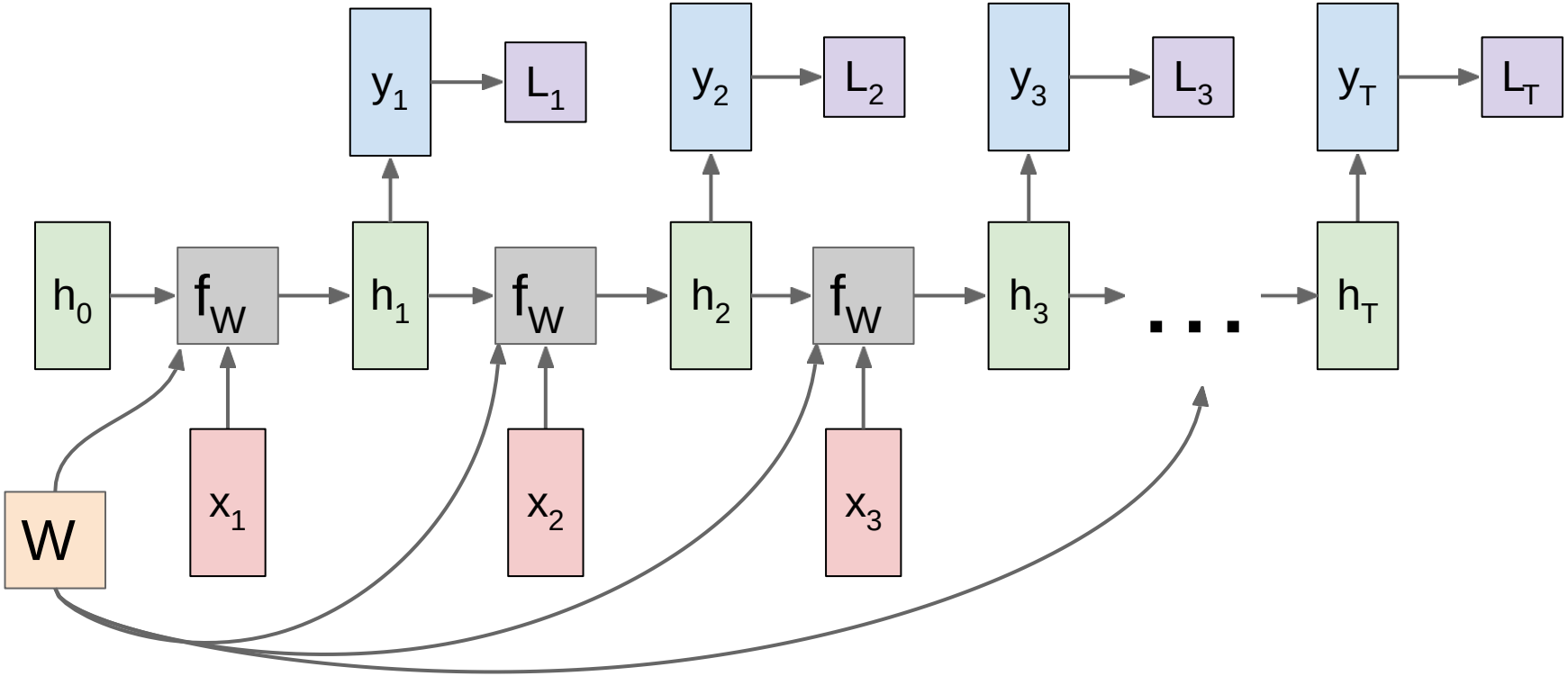
Koristimo istu matricu težina u svakom vremenskom koraku



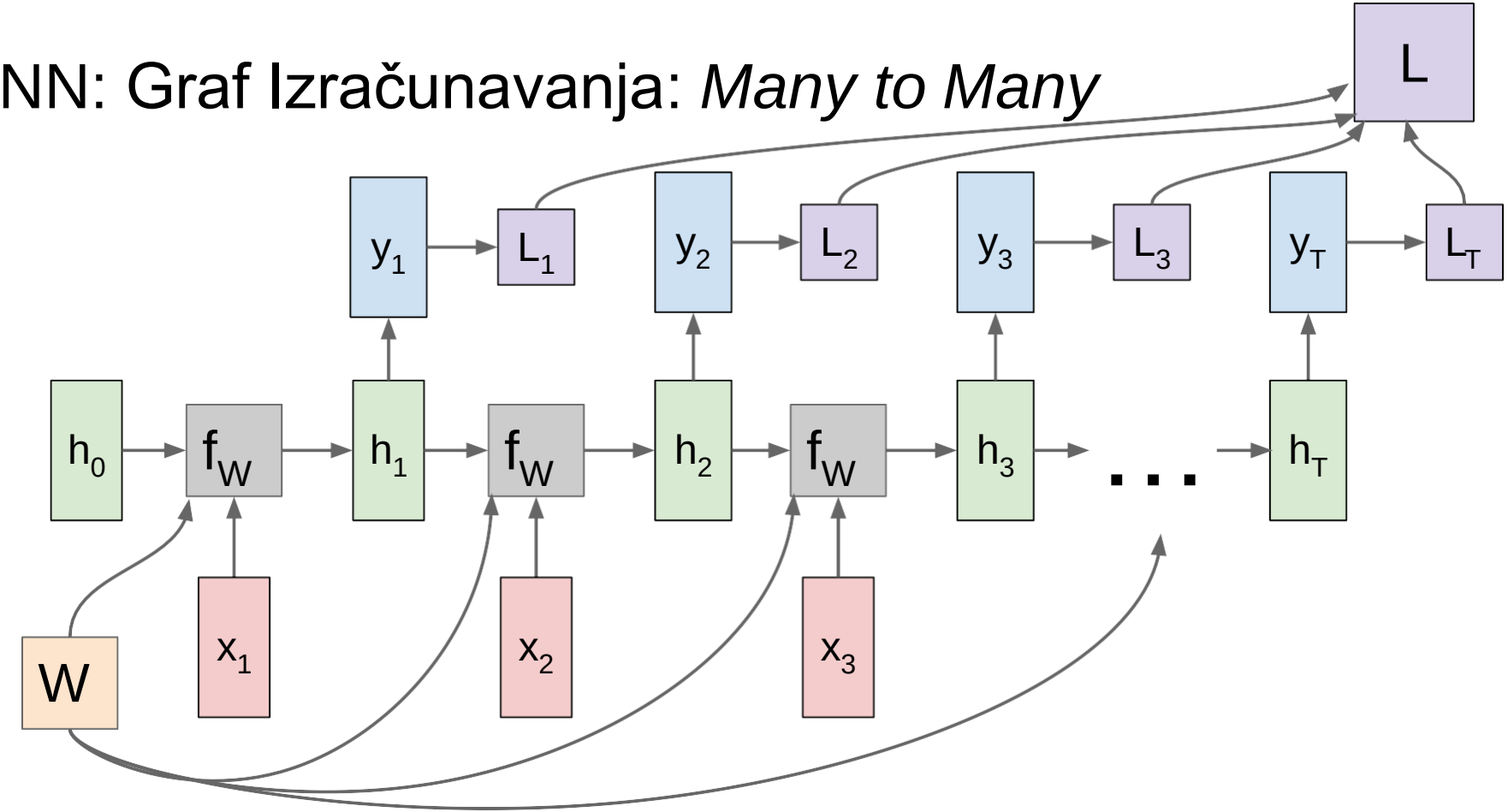
RNN: Graf Izračunavanja: *Many to Many*



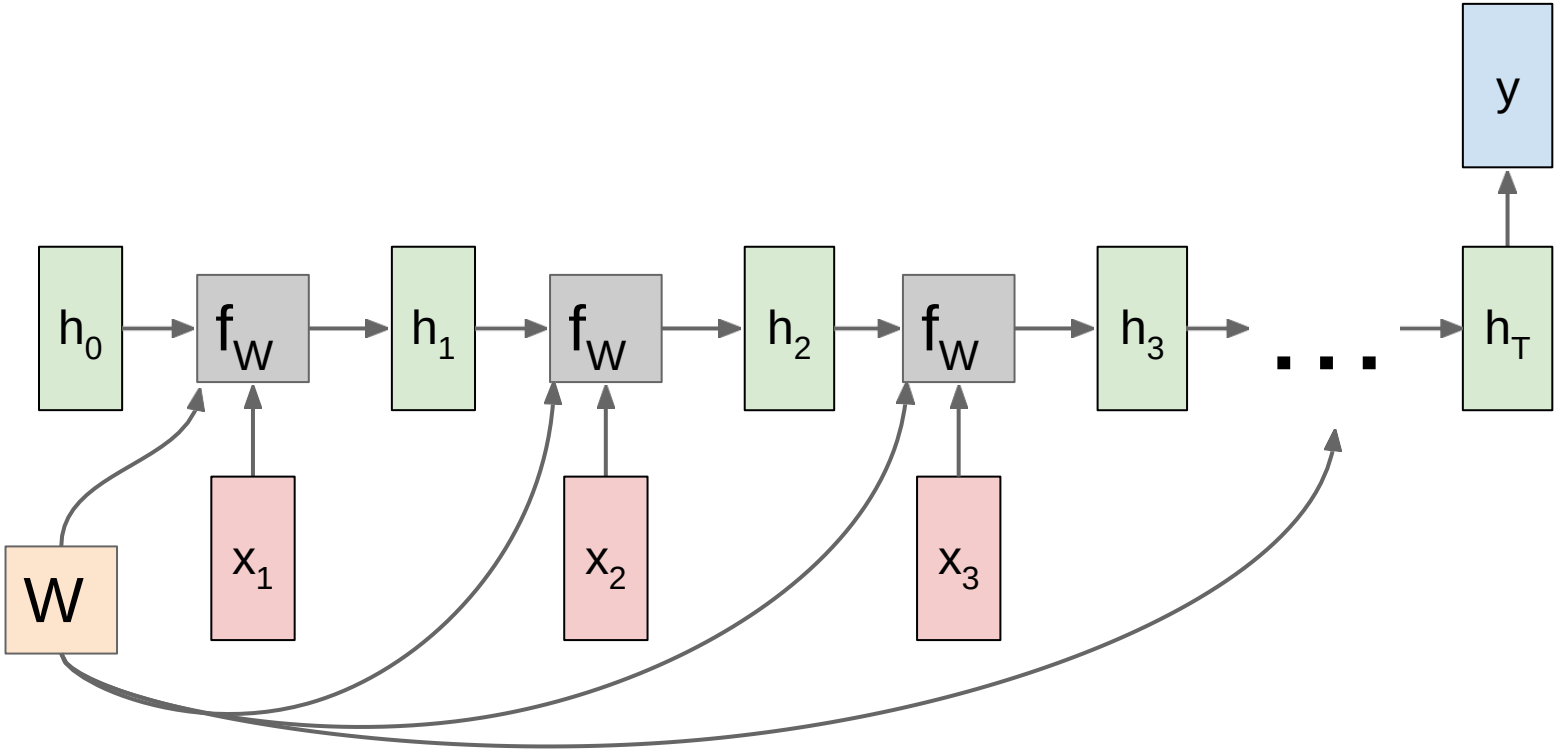
RNN: Graf Izračunavanja: *Many to Many*



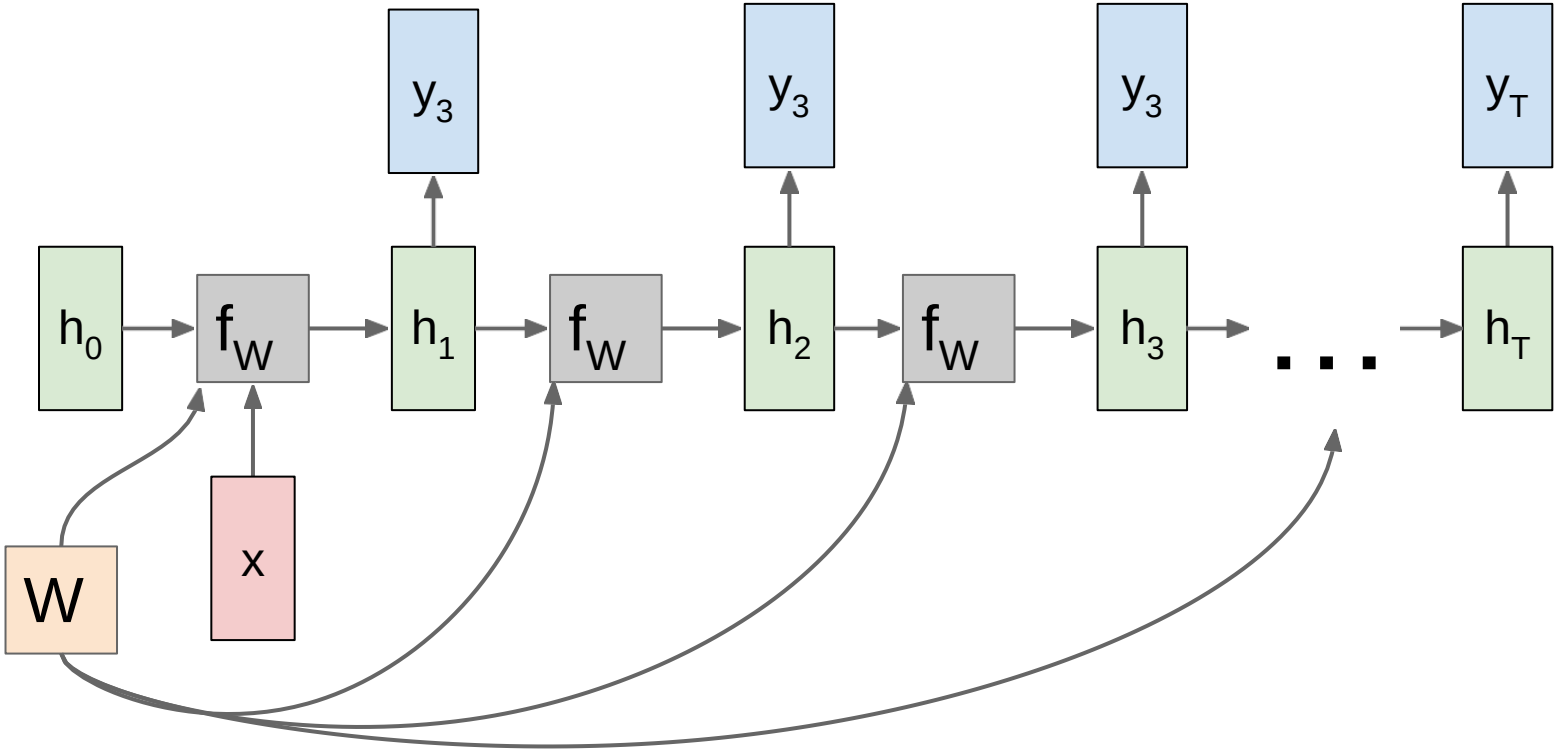
RNN: Graf Izračunavanja: *Many to Many*



RNN: Graf Izračunavanja: *Many to One*

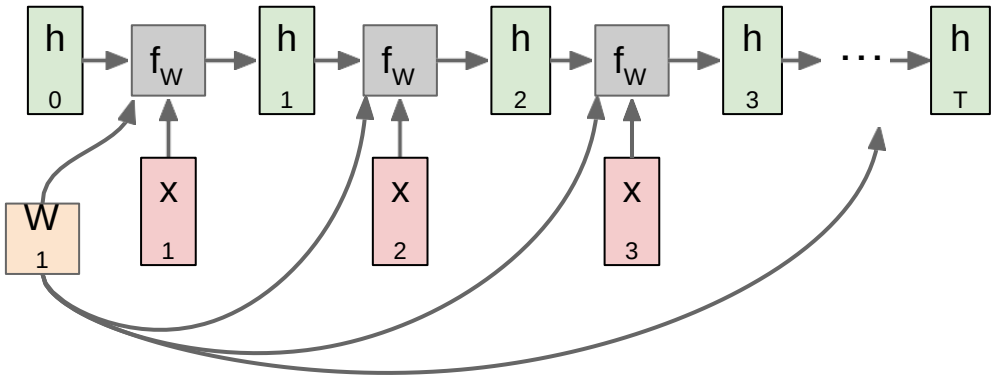


RNN: Graf Izračunavanja: *One to Many*



Sequence to Sequence: Many-to-one + one-to-many

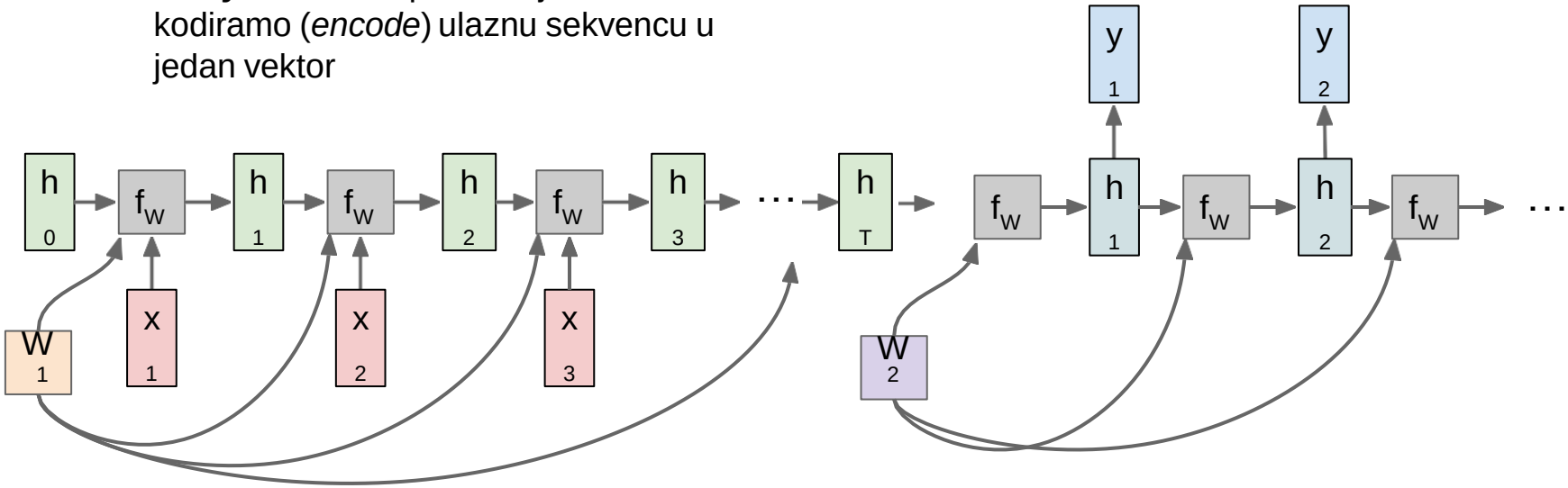
Many to one: Reprezentujemo, kodiramo (*encode*) ulaznu sekvencu u jedan vektor



Sequence to Sequence: Many-to-one + One-to-many

Many to one: Reprezentujemo, kodiramo (*encode*) ulaznu sekvencu u jedan vektor

One to many: Proizvodimo izlaznu sekvencu iz tog jednog vektora



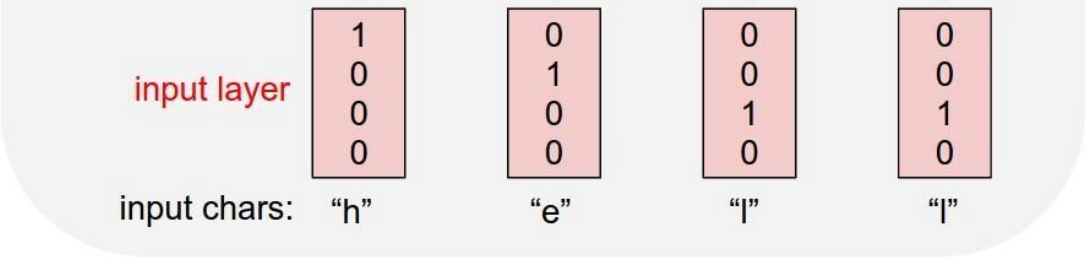
Primer:

Jezički model na nivou karaktera

Character-level Language Model

Rečnik: [h,e,l,o]

Obučavajuća sekvenca: **“hello”**

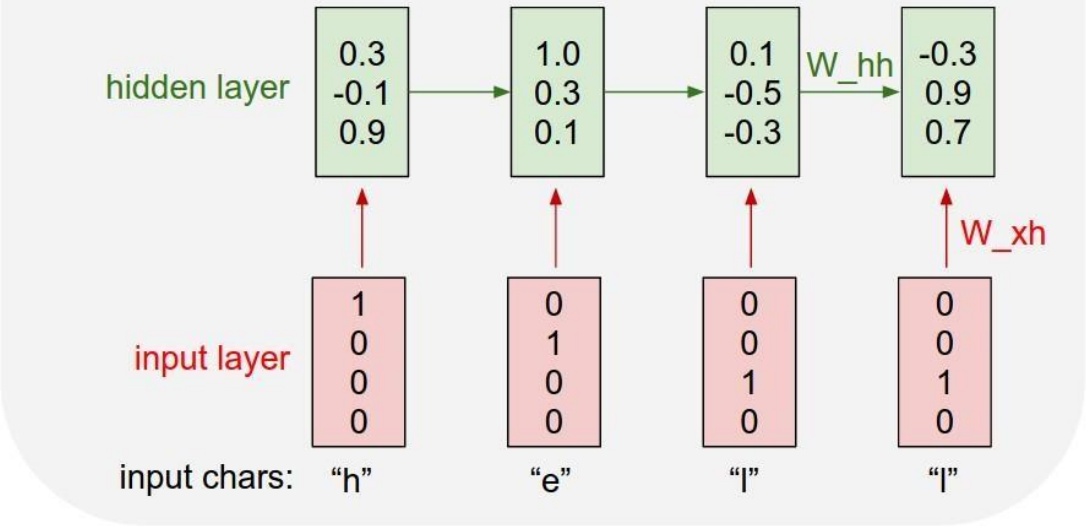


Primer:
Jezički model na nivou karaktera
Character-level Language Model

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

Rečnik: [h,e,l,o]

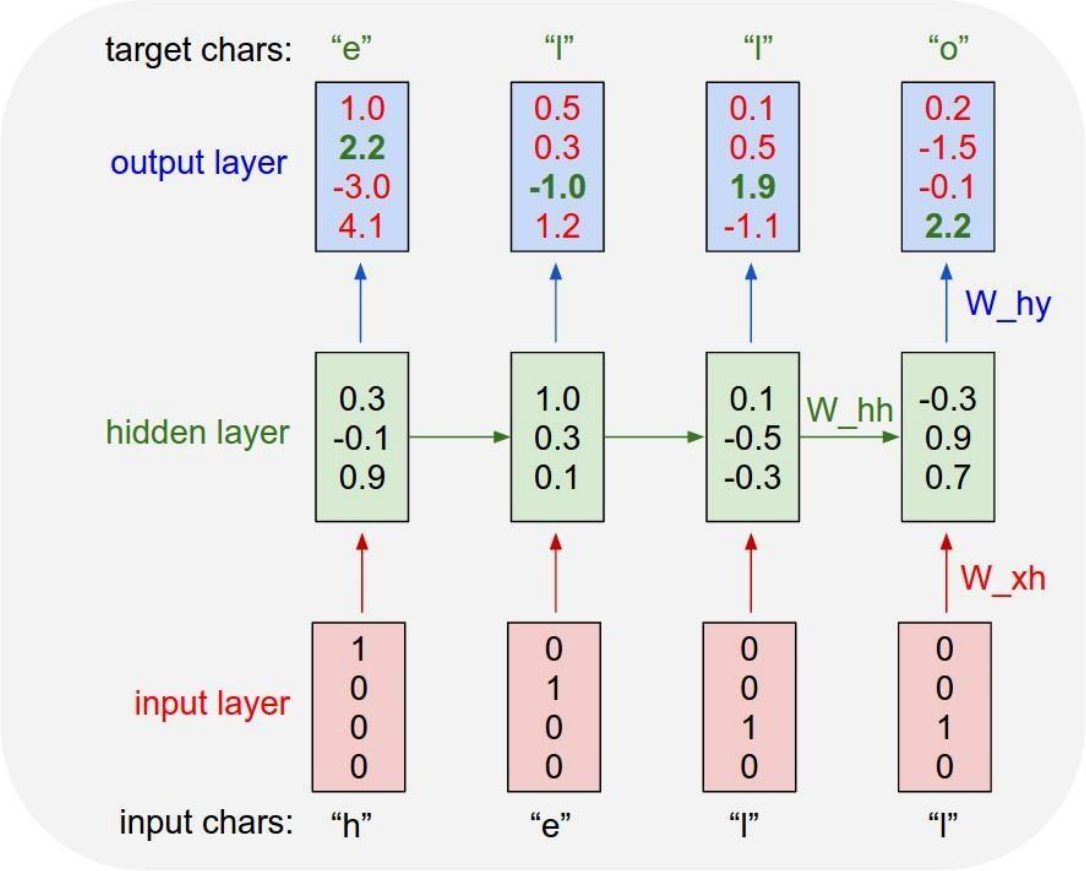
Obučavajuća sekvenca: **“hello”**



Primer:
Jezički model na nivou karaktera
Character-level Language Model

Rečnik: [h,e,l,o]

Obučavajuća sekvenca: **“hello”**



Primer:

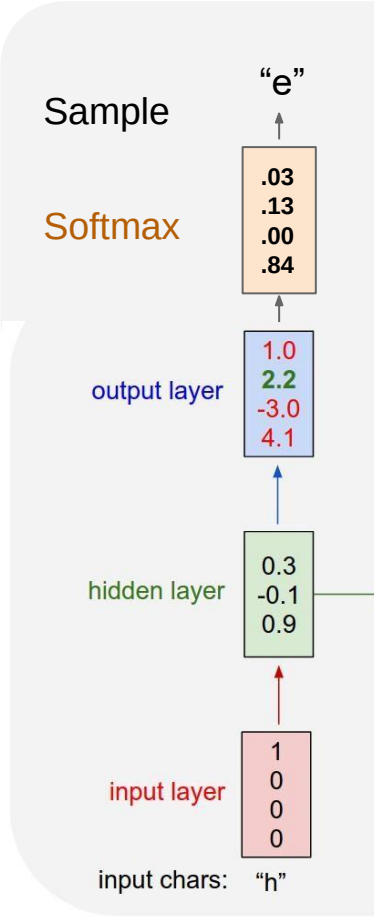
Jezički model na nivou karaktera

Character-level Language Model

Rečnik:

[h,e,l,o]

Kad koristimo mrežu nakon obučavanja semplujemo karakter po karakter sa izlaza i ubacujemo na ulaz



Primer:

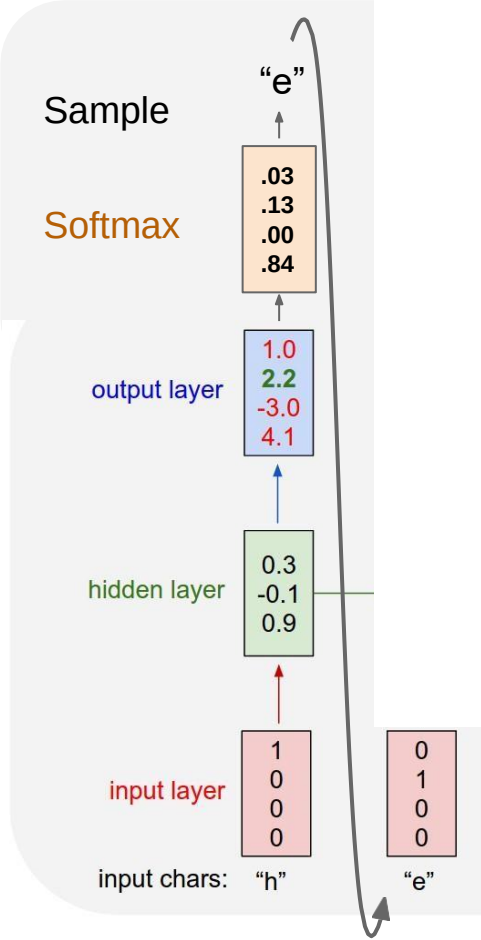
Jezički model na nivou karaktera

Character-level Language Model

Rečnik:

[h,e,l,o]

Kad koristimo mrežu nakon obučavanja semplujemo karakter po karakter sa izlaza i ubacujemo na ulaz



Primer:

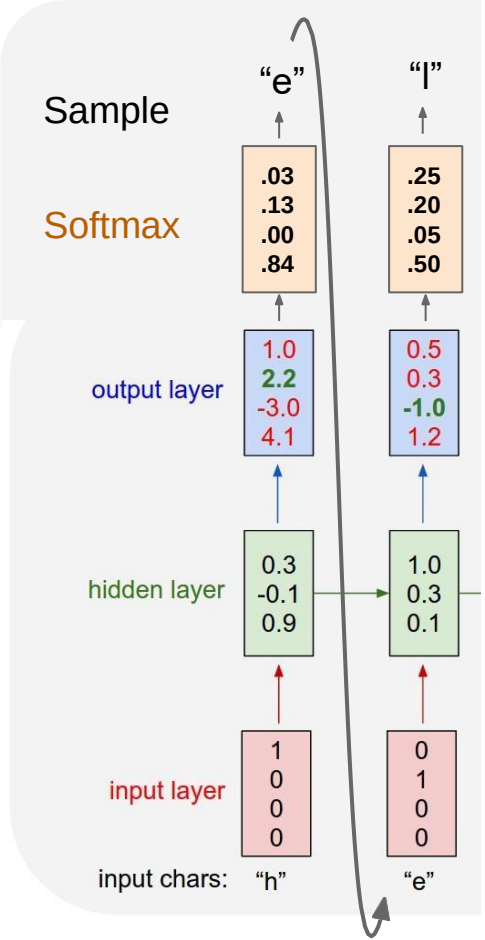
Jezički model na nivou karaktera

Character-level Language Model

Rečnik:

[h,e,l,o]

Kad koristimo mrežu nakon obučavanja semplujemo karakter po karakter sa izlaza i ubacujemo na ulaz



Primer:

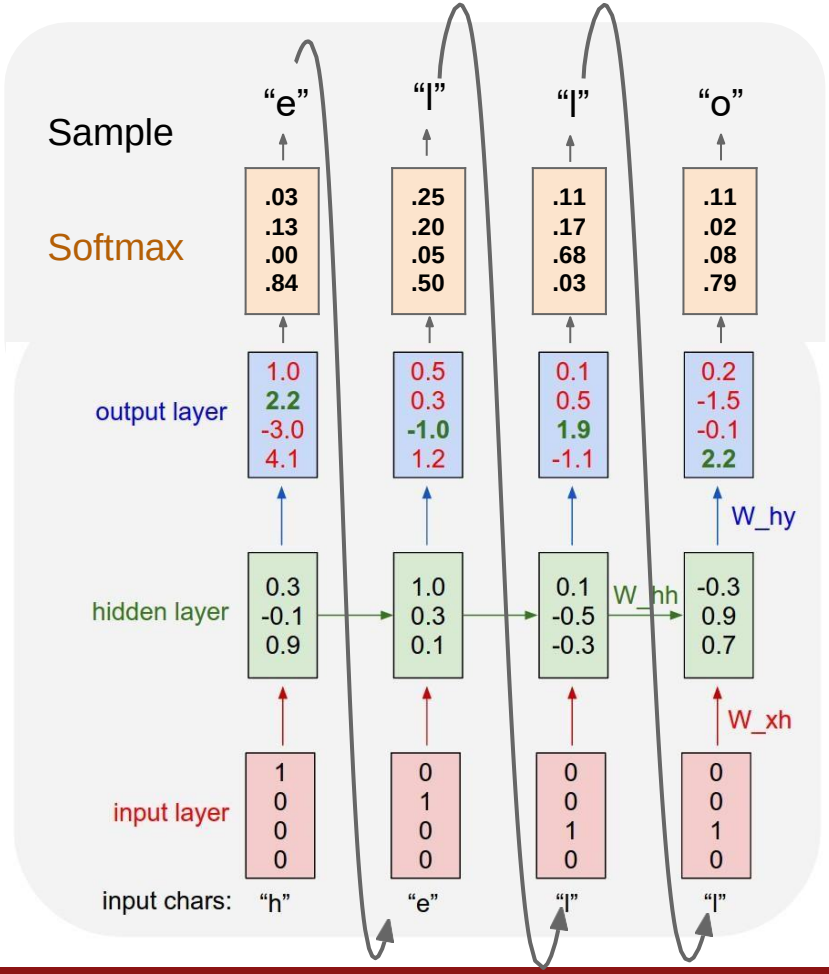
Jezički model na nivou karaktera

Character-level Language Model

Rečnik:

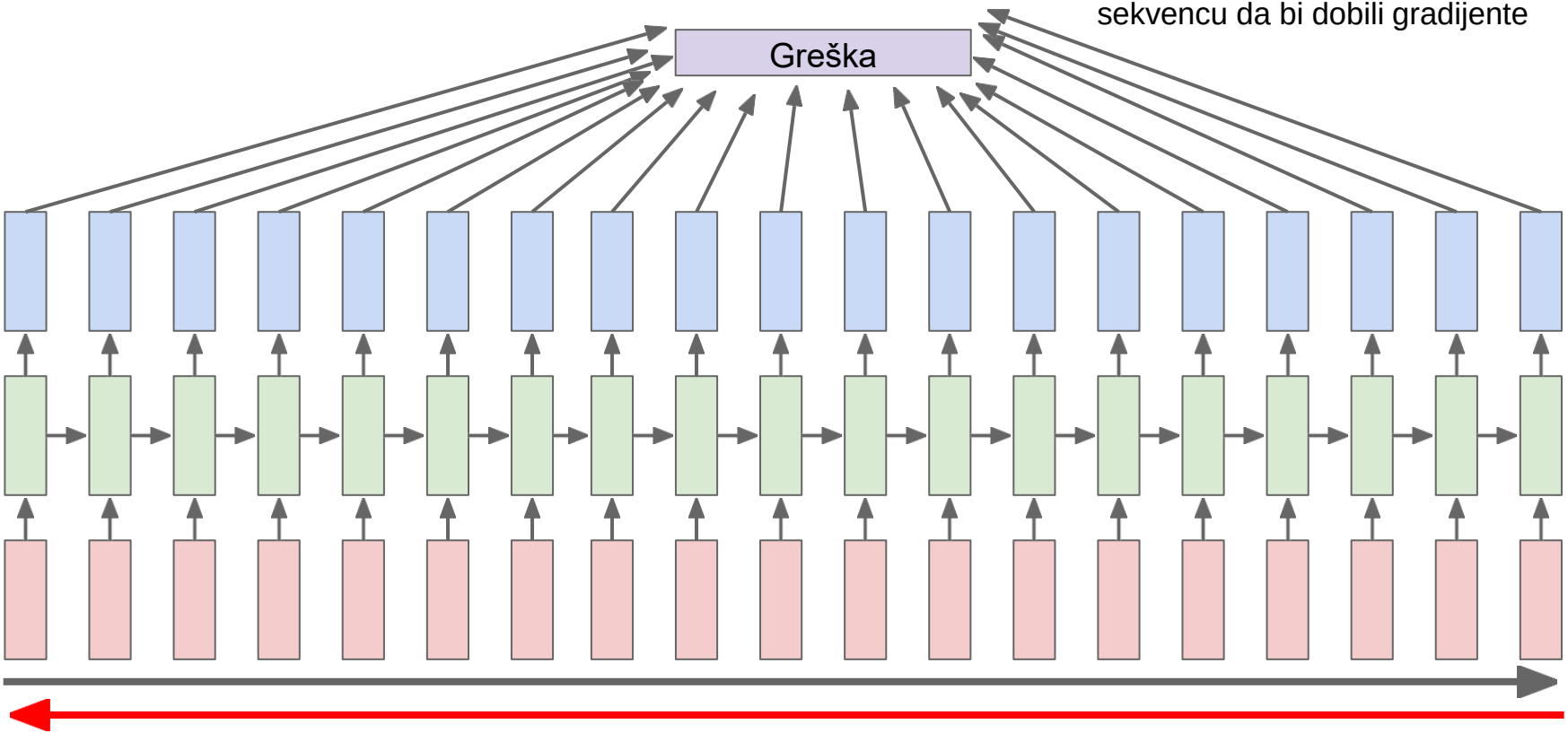
[h,e,l,o]

Kad koristimo mrežu nakon obučavanja semplujemo karakter po karakter sa izlaza i ubacujemo na ulaz

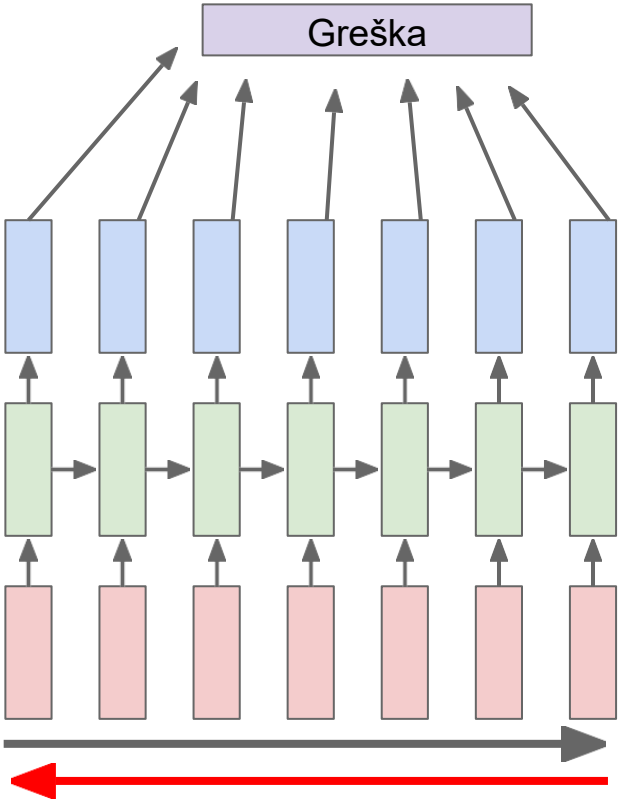


Backpropagation kroz vreme

Celu sekvencu izračunavamo unapred da bi dobili grešku, onda radimo backpropagation kroz celu sekvencu da bi dobili gradijente

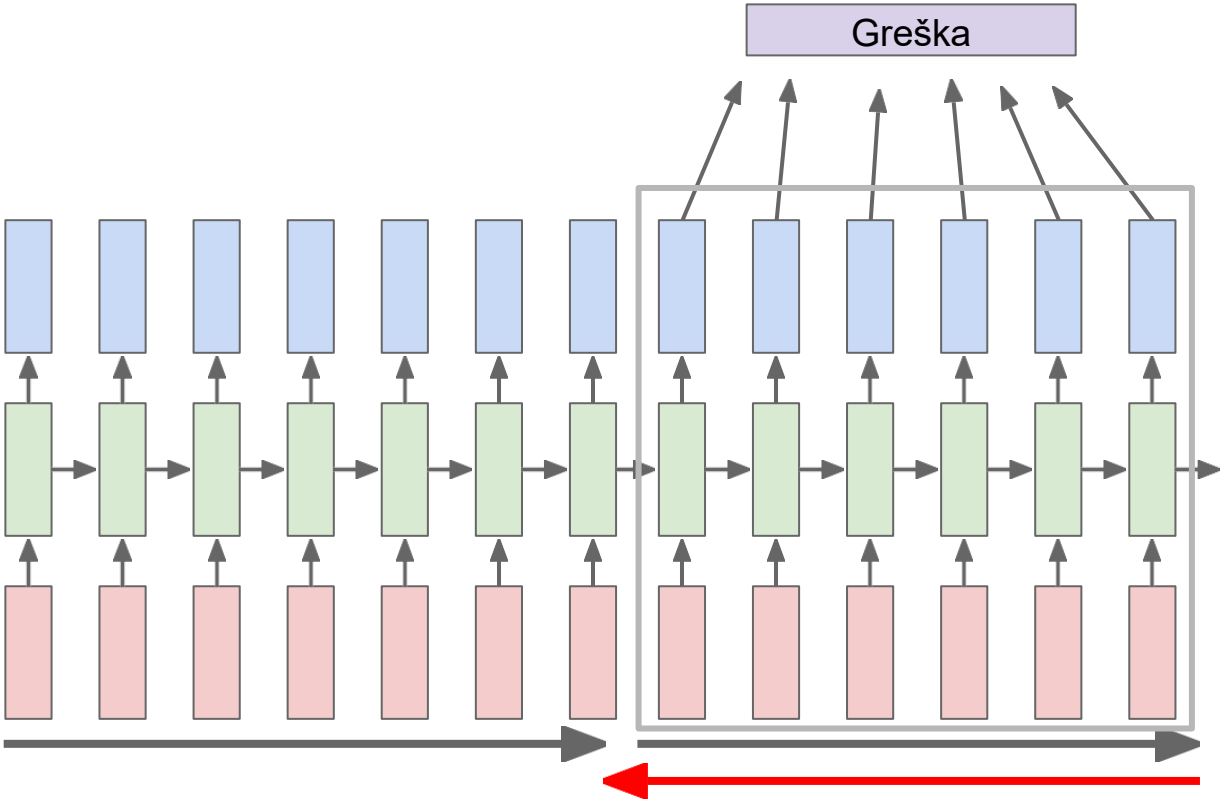


Backpropagation kroz vreme **Deo-po-deo**



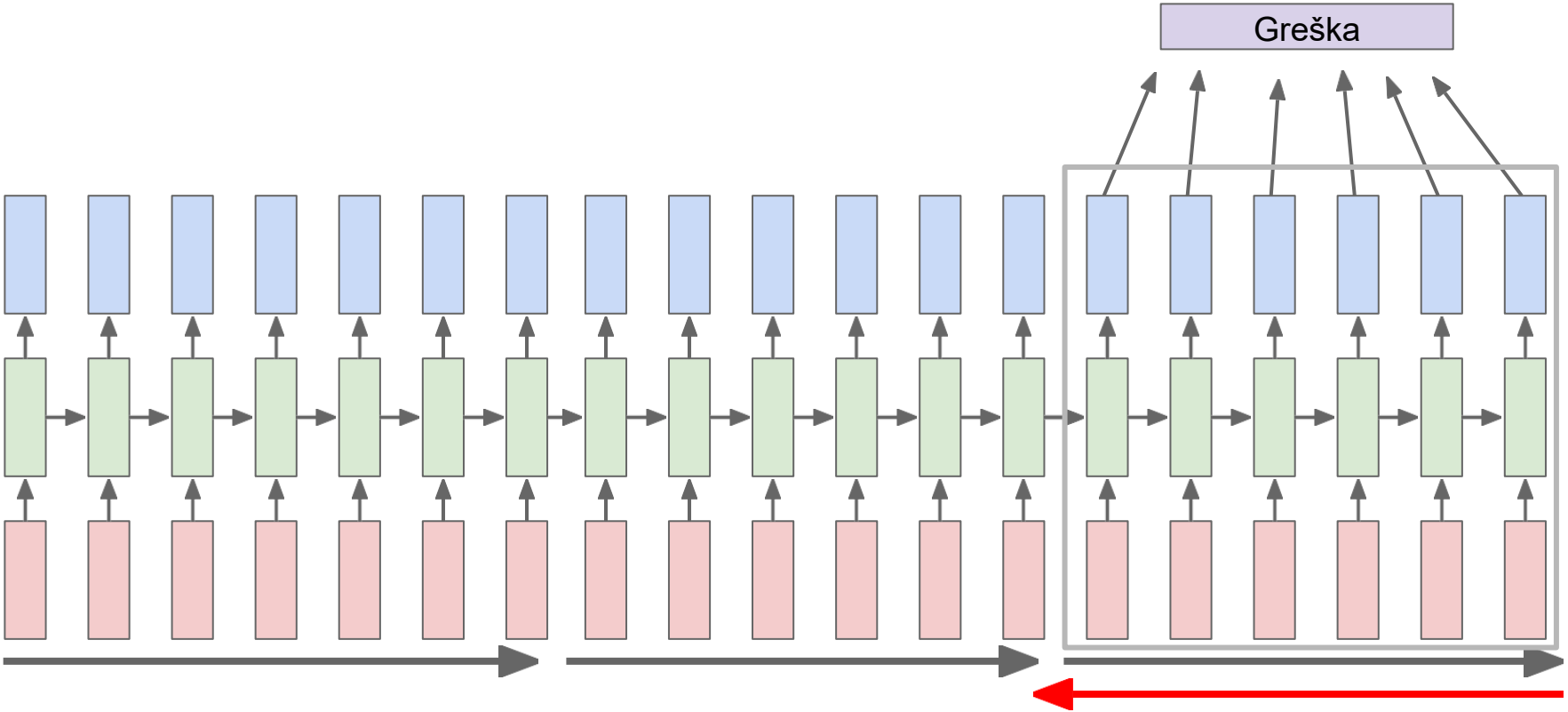
Radimo izračunavanje unapred i backpropagation unazad, ali za mini-podskupove ulaza.

Backpropagation kroz vreme **Deo-po-deo**



Prenosimo poslednje skriveno stanje u novi mini-podskup.

Backpropagation kroz vreme **Deo-po-deo**



min-char-rnn.py gist: 112 lines of Python

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

(https://gist.github.com/karpathy/d4dee566867f8291f086)

min-char-rnn.py gist

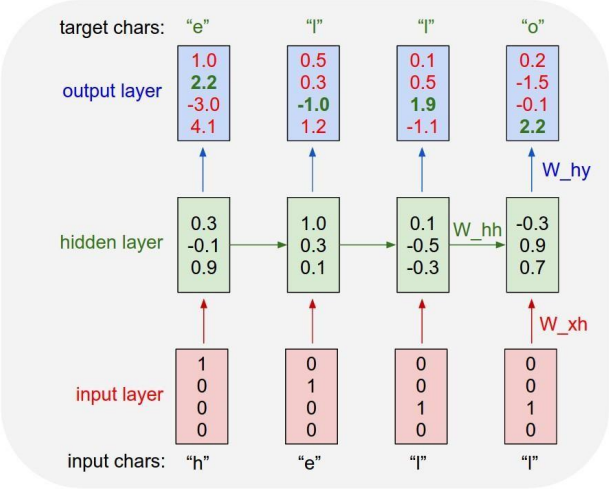
Data I/O

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }
```

min-char-rnn.py gist

Inicijalizacije

```python
# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias
```



min-char-rnn.py gist

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

Glavna petlja

```python
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

min-char-rnn.py gist

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

Glavna petlja

```python
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

min-char-rnn.py gist

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

Glavna petlja

```python
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

min-char-rnn.py gist

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

Glavna petlja

```python
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

min-char-rnn.py gist

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

Glavna petlja

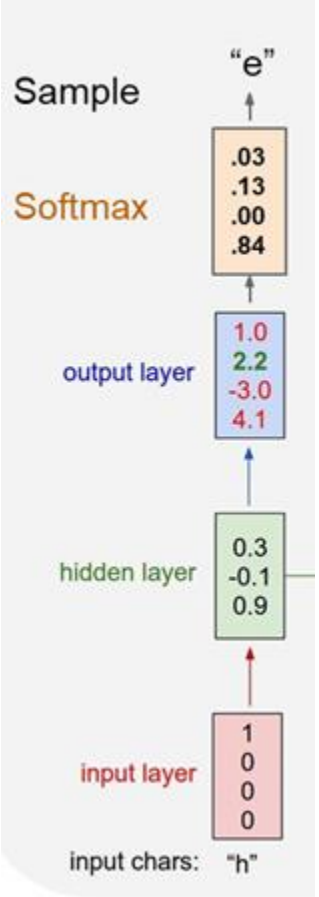
```python
n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

min-char-rnn.py gist

LossFun

- propagacija unapred (izračunavanje *loss*)
- propagacija unazad (izračunavanje gradijenata)

```python
def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```




```python
def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
```

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + \boldsymbol{b_h})$$

$$y_t = W_{hy} h_t + \boldsymbol{b_y}$$

Softmax

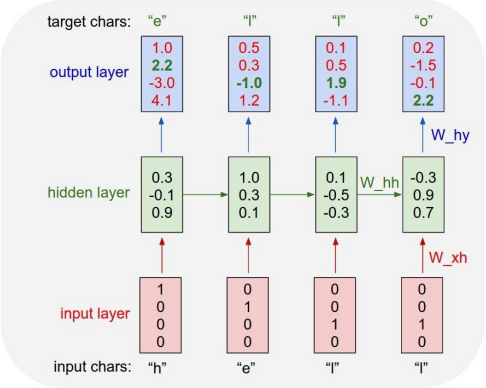
min-char-rnn.py gist

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 == len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

```python
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + \boldsymbol{b_h})$$

$$y_t = W_{hy}h_t + \boldsymbol{b_y}$$



min-char-rnn.py gist

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]

def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

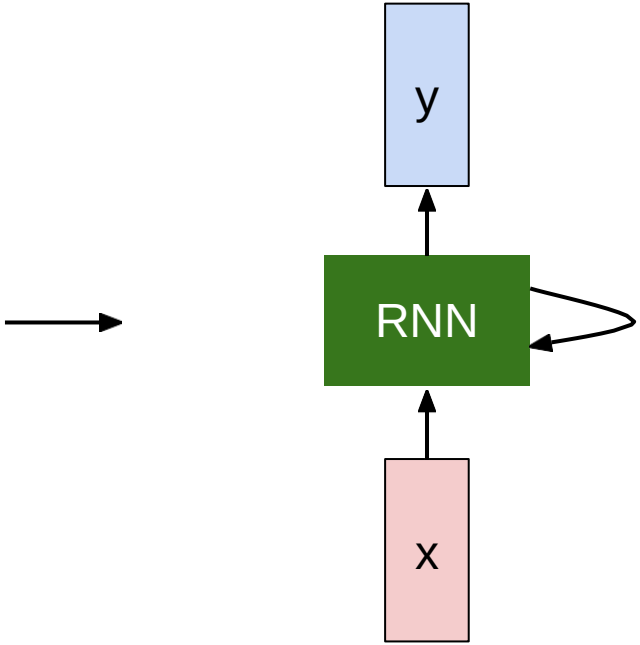
```python
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes
```

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
    Pity the world, or else this glutton be,
    To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
    This were to be new made when thou art old,
    And see thy blood warm when thou feel'st it cold.



at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.

VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

AI može da napiše scenario za film koji je nenamreno smešan

Movie written by algorithm turns out to be hilarious and intense

For *Sunspring*'s exclusive debut on Ars, we talked to the filmmakers about collaborating with an AI.

ANNALEE NEWITZ - 6/9/2016, 12:30 PM

Sunspring, a short science fiction movie written entirely by AI, debuts exclusively on Ars today.

Ars is excited to be hosting this online debut of *Sunspring*, a short science fiction film that's not entirely what it seems. It's about three people living in a weird future, possibly on a space station,

https://arstechnica.com/gaming/2016/06/an-ai-wrote-this-movie-and-its-strangely-moving/

AI može da napiše scenario za film koji je nenamreno smešan

„He is standing in the stars and sitting on the floor. He takes a seat on the counter and pulls the camera over to his back. He stares at it. He is on the phone. He cuts the shotgun from the edge of the room and puts it in his mouth. He sees a black hole in the floor leading to the man on the roof.“

He picks up a light screen and fights the security force of the particles of a transmission on his face.
H
(continuing)
What do you mean?
C
(smiles)
I don't know anything about any of this.
H
(to Hauk, taking his eyes from his mouth)
Then what?

H2
There's no answer.
C
(frowning)
We're going to see the money.
H
(reading)
"All right, you can't tell me that."

The Stacks Project: open source udžbenik algebarske geometrije

The Stacks Project

home | about | tags explained | tag lookup | browse | search | bibliography | recent comments | blog | add slogans

Browse chapters

Part	Chapter	online	TeX source	view pdf
Preliminaries				
	1. Introduction	online	tex	pdf
	2. Conventions	online	tex	pdf
	3. Set Theory	online	tex	pdf
	4. Categories	online	tex	pdf
	5. Topology	online	tex	pdf
	6. Sheaves on Spaces	online	tex	pdf
	7. Sites and Sheaves	online	tex	pdf
	8. Stacks	online	tex	pdf
	9. Fields	online	tex	pdf
	10. Commutative Algebra	online	tex	pdf

Parts

1. Preliminaries
2. Schemes
3. Topics in Scheme Theory
4. Algebraic Spaces
5. Topics in Geometry
6. Deformation Theory
7. Algebraic Stacks
8. Miscellany

Statistics

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex tekstovi

http://stacks.math.columbia.edu/
The stacks project is licensed under the GNU Free Documentation License

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariccoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ over U compatible with the complex

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S. Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1) *f is locally of finite type. Since $S = \mathrm{Spec}(R)$ and $Y = \mathrm{Spec}(R)$.*

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let X be a locally Noetherian scheme over S, $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.*

Lemma 0.3. *In Situation ??. Hence we may assume $\mathfrak{q}' = 0$.*

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{tikzcd}
\mathcal{S} \arrow[r] \arrow[d] & & \\
\xi \arrow[r] & \mathcal{O}_{X'} \arrow[dr] & \\
\mathrm{gor}_s \arrow[u] & & \\
 & = \alpha' \arrow[r] \arrow[d, leftrightarrow] & \\
 & = \alpha' \arrow[r] & \alpha \\
 & \mathrm{Spec}(K_\psi) & \mathrm{Mor}_{Sets} \quad \mathrm{d}(\mathcal{O}_{X_{X/k}}, \mathcal{G}) \arrow[u, "X"']
\end{tikzcd}$$

is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \quad \text{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{\eta}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

This repository Search Explore Gist Blog Help karpathy

torvalds / linux

Watch 3,711 Star 23,054 Fork 9,141

Linux kernel source tree

520,037 commits 1 branch 420 releases 5,039 contributors

branch: master linux / +

Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux

torvalds authored 9 hours ago latest commit 4b1706927d

Documentation	Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending	6 days ago
arch	Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l...	a day ago
block	block: discard bdi_unregister() in favour of bdi_destroy()	9 days ago
crypto	Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6	10 days ago
drivers	Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux	9 hours ago
firmware	firmware/ihex2fw.c: restore missing default in switch statement	2 months ago
fs	vfs: read file_handle only once in handle_to_path	4 days ago
include	Merge branch 'perf-urgent-for-linus' of git://git.kernel.org/pub/scm/...	a day ago
init	init: fix regression by supporting devices with major:minor:offset fo...	a month ago

Code

Pull requests 74

Pulse

Graphs

HTTPS clone URL

https://github.c

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

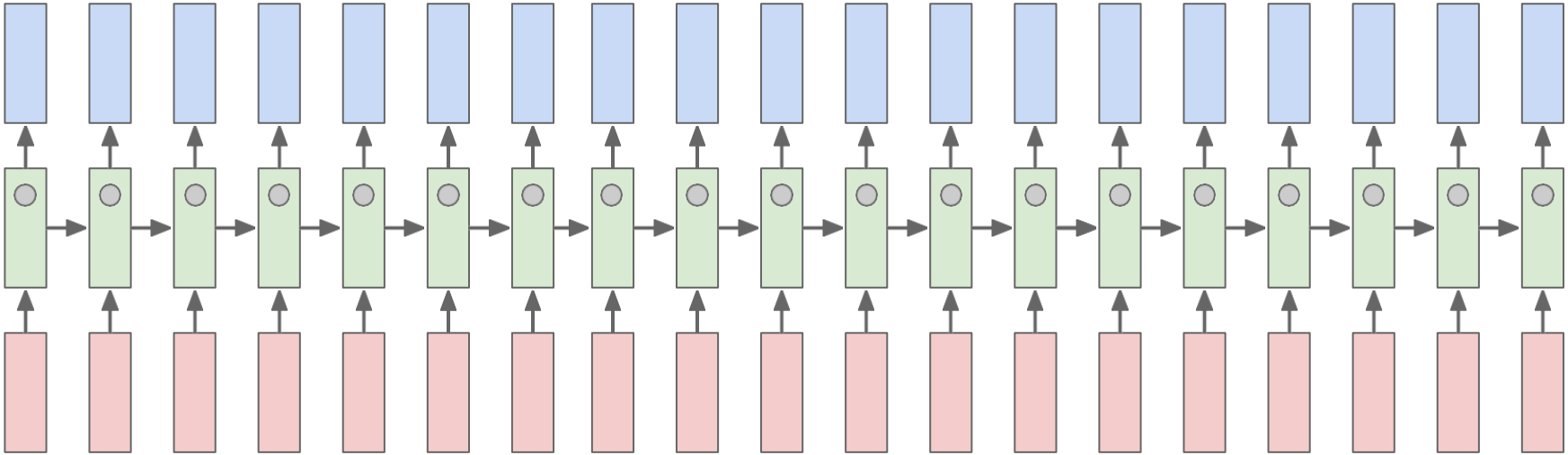
```c
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

Generisan C kod

```
/*
 *  Copyright (c) 2006-2010, Intel Mobile Communications.  All rights reserved.
 *
 *   This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 *        This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *
 *  GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software Foundation,
 *  Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>
```

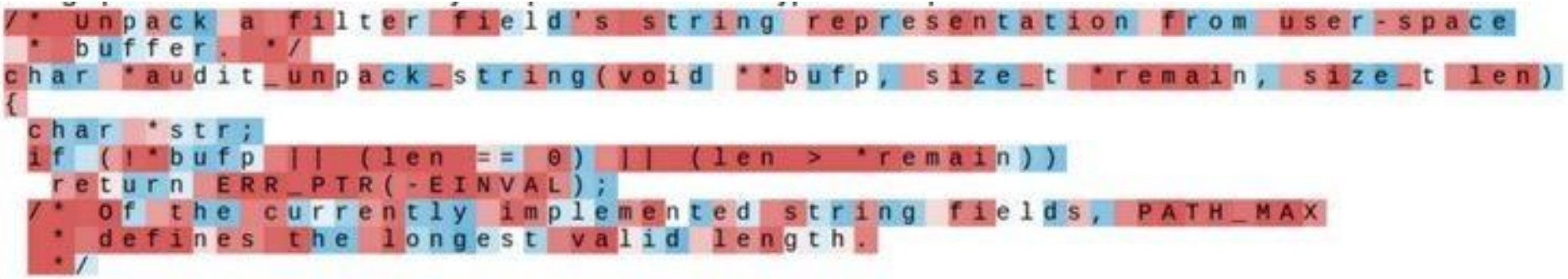
```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)     (func)

#define SWAP_ALLOCATE(nr)     (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0));   \
  if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
          pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
  PUT_PARAM_RAID(2, sel) = get_state_state();
  set_pid_sum((unsigned long)state, current_state_str(),
           (unsigned long)-1->lr_full; low;
}
```

Tumačenja aktivacija neurona u RNN



Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

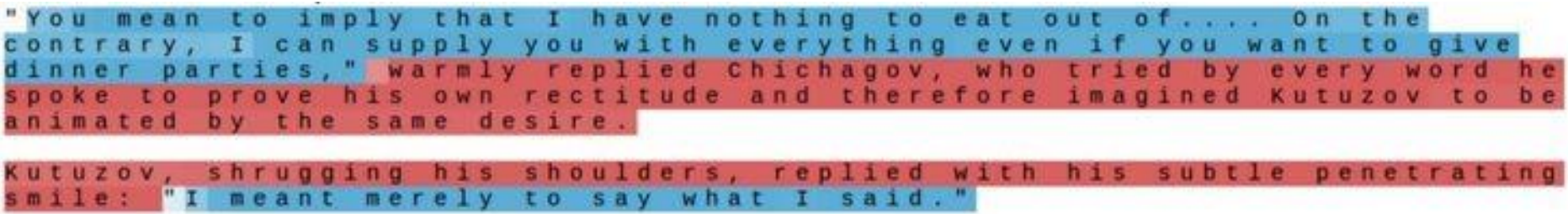
Tumačenja aktivacija neurona u RNN



Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Tumačenja aktivacija neurona u RNN



neuron koji detektuje
znak navoda

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Tumačenja aktivacija neurona u RNN

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.

neuron koji prati dužinu reda

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Tumačenja aktivacija neurona u RNN

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
    siginfo_t *info)
{
  int sig = next_signal(pending, mask);
  if (sig) {
    if (current->notifier) {
      if (sigismember(current->notifier_mask, sig)) {
        if (!(current->notifier)(current->notifier_data)) {
          clear_thread_flag(TIF_SIGPENDING);
          return 0;
        }
      }
    }
    collect_signal(sig, pending, info);
  }
  return sig;
}
```

neuron koji se aktivira kad naiđemo na if naredbu

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Tumačenja aktivacija neurona u RNN

Cell that turns on inside comments and quotes:

```
/* Duplicate LSM field information.  The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
           struct audit_field *sf)
{
  int ret = 0;
  char *lsm_str;
  /* our own copy of lsm_str */
  lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
  if (unlikely(!lsm_str))
    return -ENOMEM;
  df->lsm_str = lsm_str;
  /* our own (refreshed) copy of lsm_rule */
  ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                 (void **)&df->lsm_rule);
  /* Keep currently invalid fields around in case they
   * become valid after a policy reload. */
  if (ret == -EINVAL) {
    pr_warn("audit rule for LSM \'%s\' is invalid\n",
      df->lsm_str);
    ret = 0;
  }
  return ret;
}
```

neuron vezan za komentare

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Tumačenja aktivacija neurona u RNN

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
  int i;
  if (classes[class]) {
    for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
      if (mask[i] & classes[class][i])
        return 0;
  }
  return 1;
}
```

neuron koji prati „dubinu“ koda

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Dodeljivanje opisa slikama

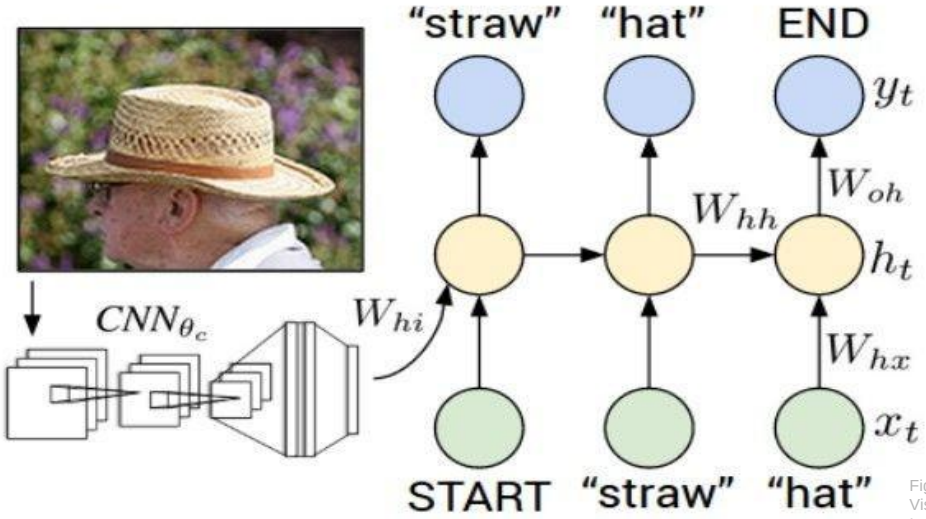
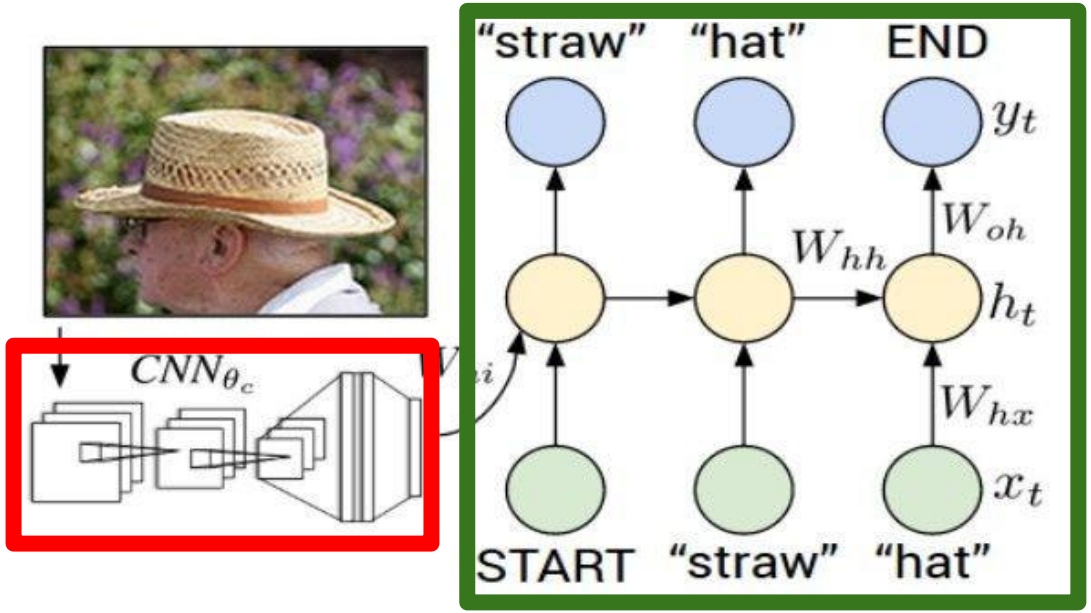


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015. Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Rekurentna Neuronska Mreža



Konvolutivna Neuronska Mreža

test slika

This image is CC0 public domain

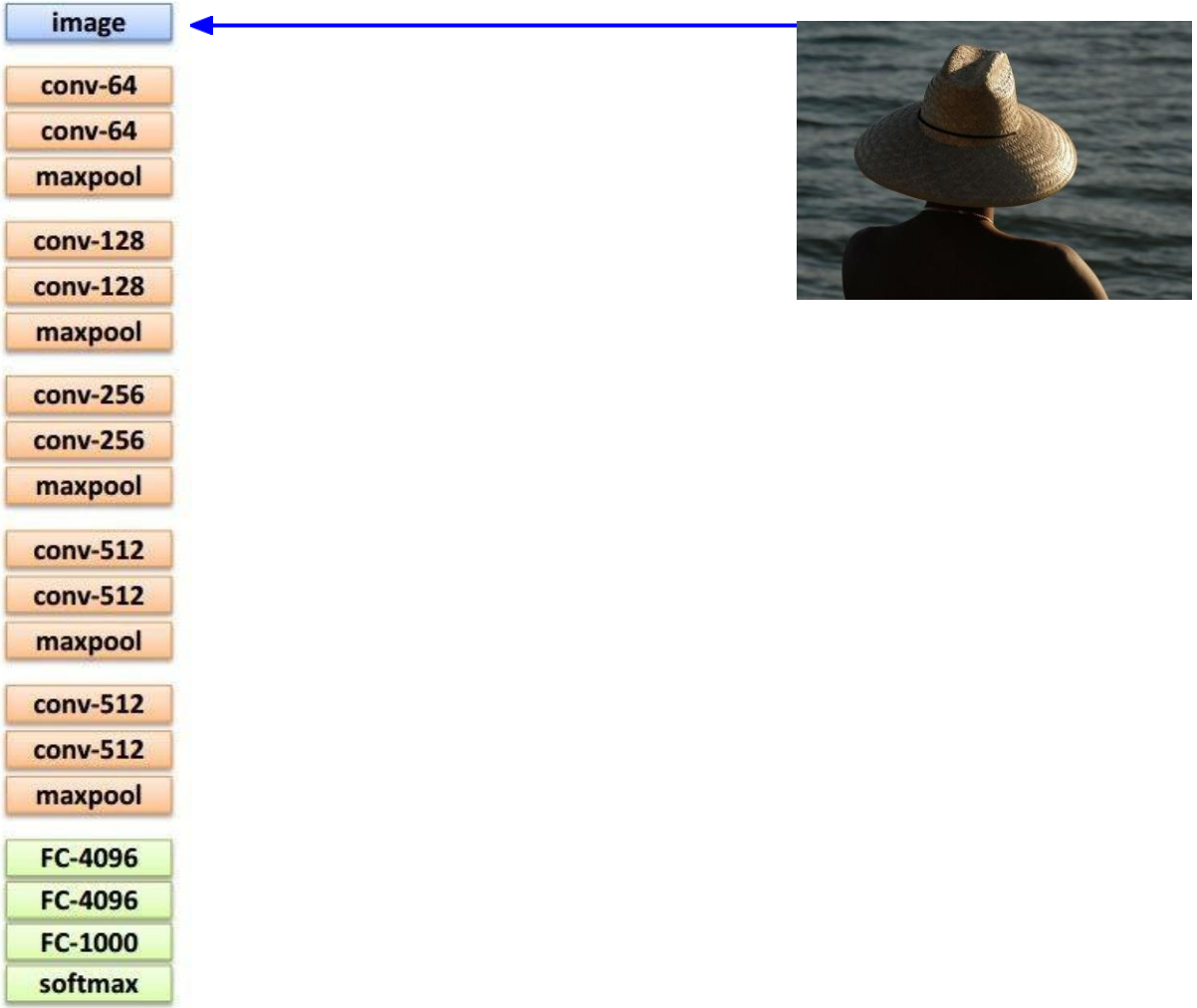


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

test slika

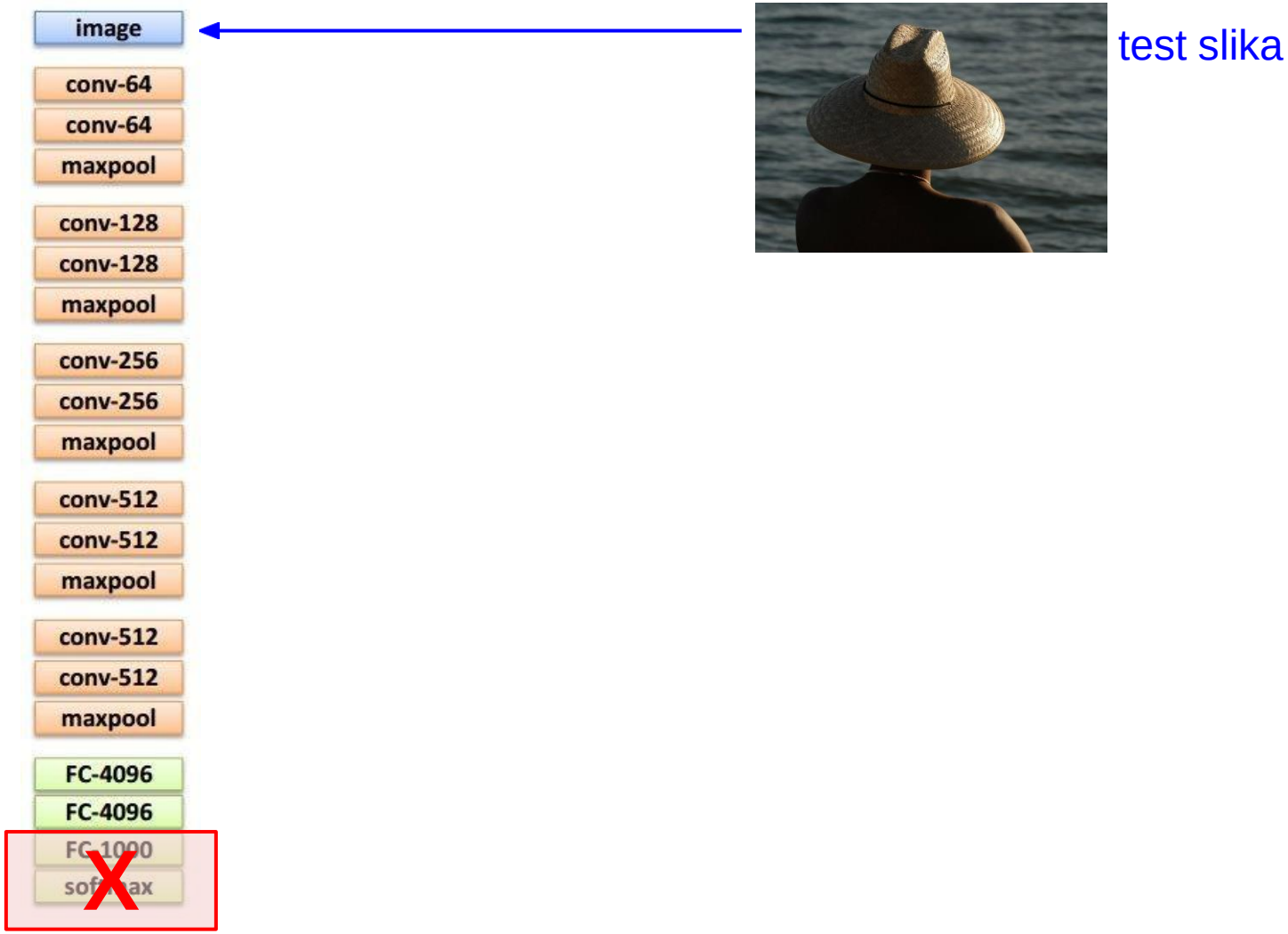


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax
test slika

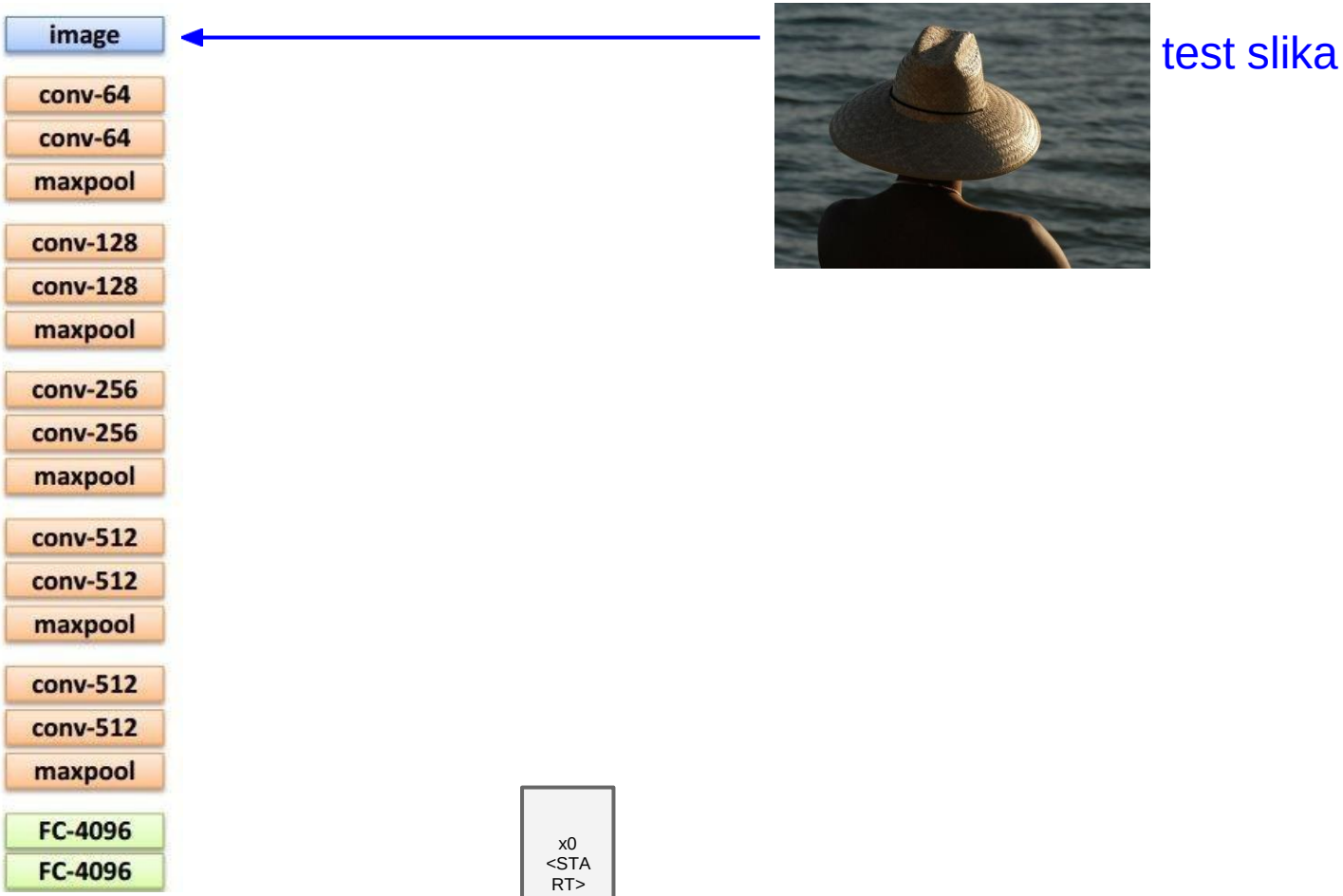
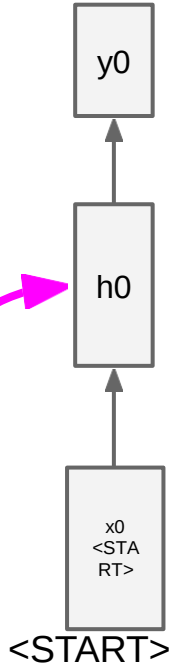


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
x0
<STA
RT>
<START>
test slika

test slika



Wih

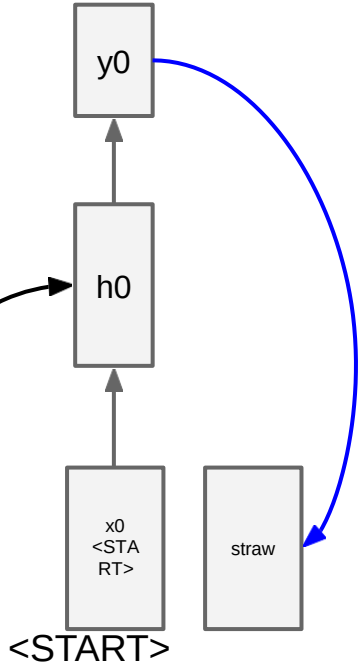
pre:

h = tanh(Wxh * x + Whh * h)

sad:

h = tanh(Wxh * x + Whh * h **+ Wih * v**)

test slika



semplujemo!

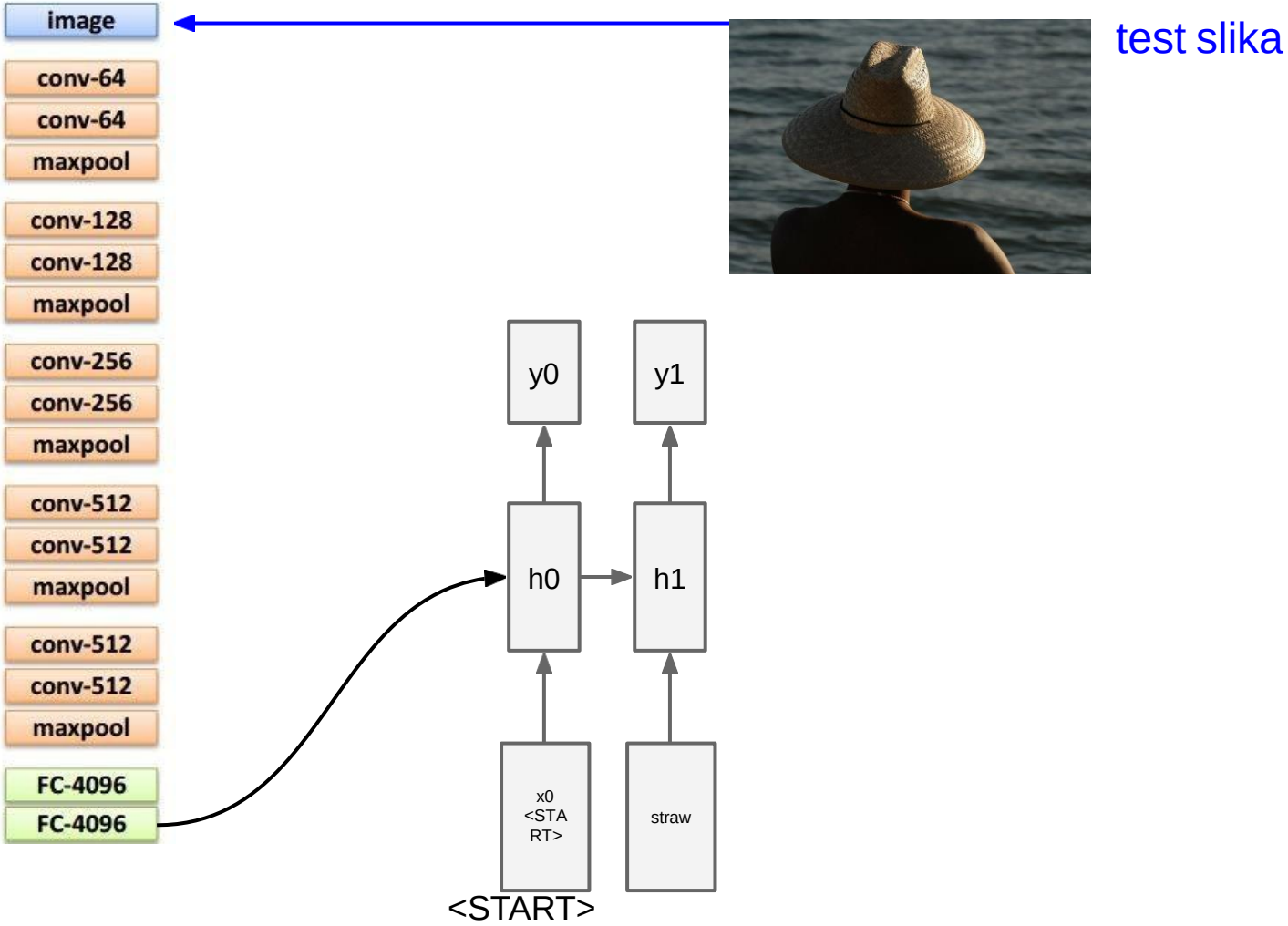


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test slika
y0
y1
h0
h1
x0
<STA
RT>
straw
<START>

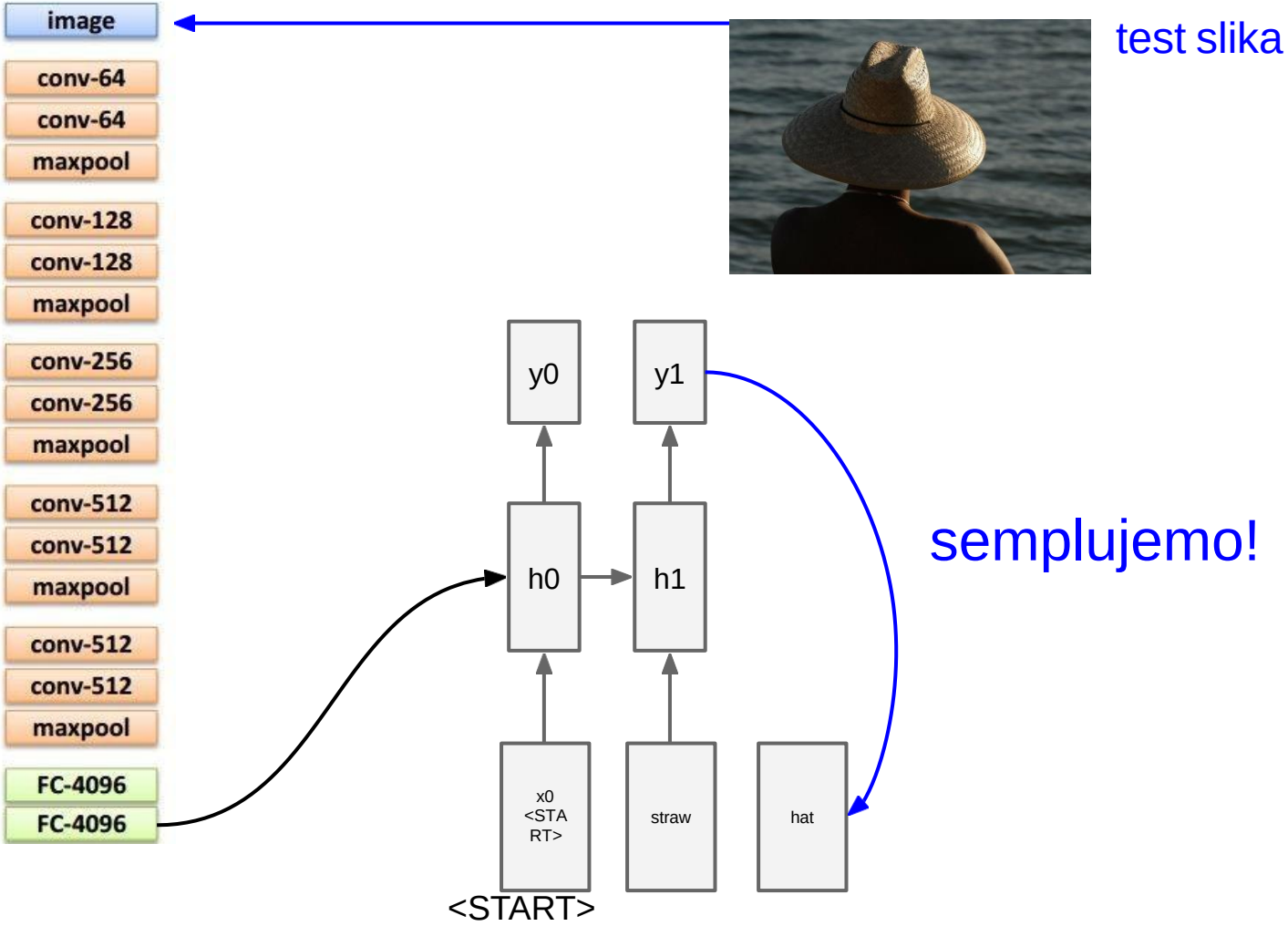


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test slika
y0
y1
h0
h1
x0
<STA
RT>
straw
hat
<START>
semplujemo!

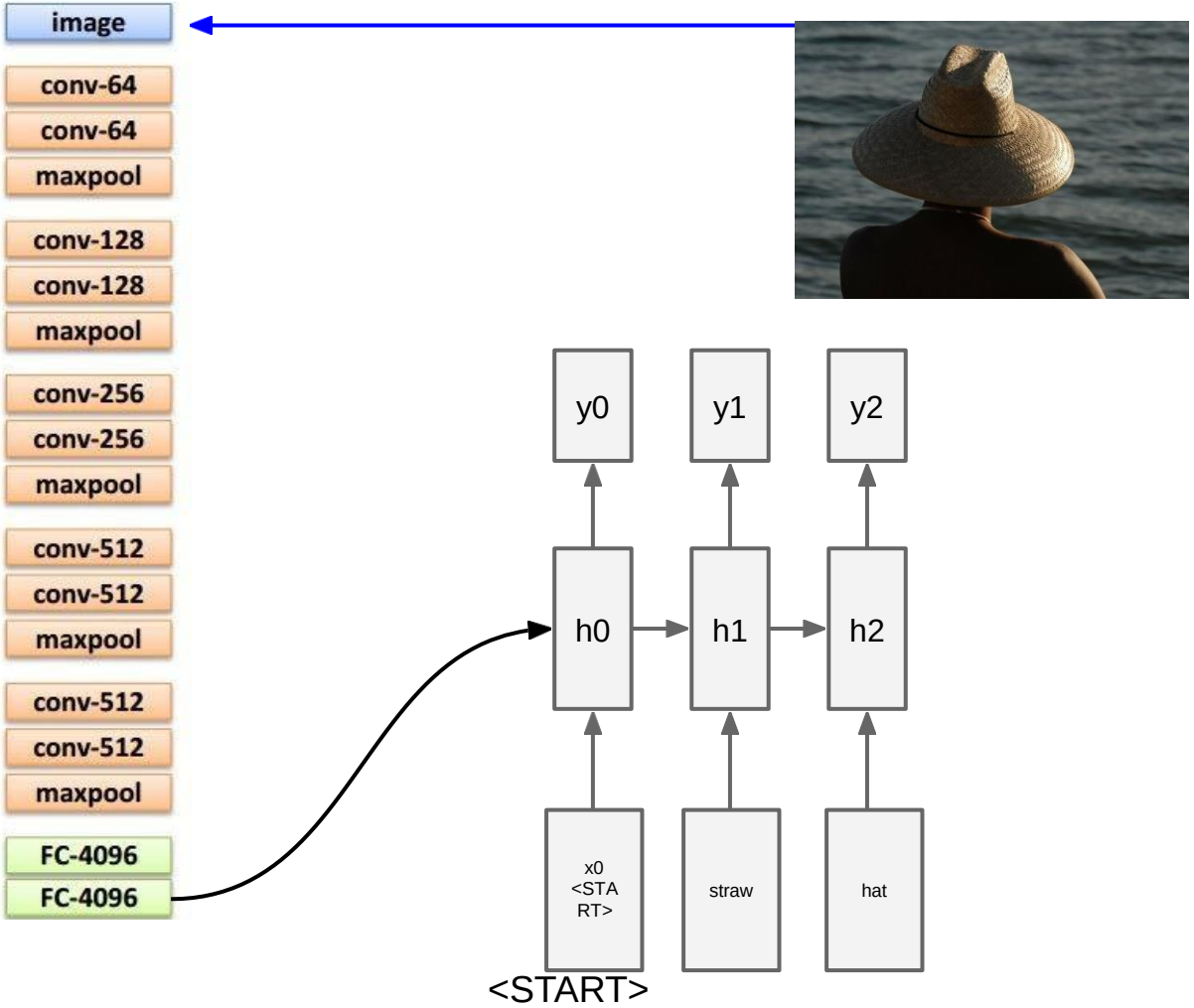
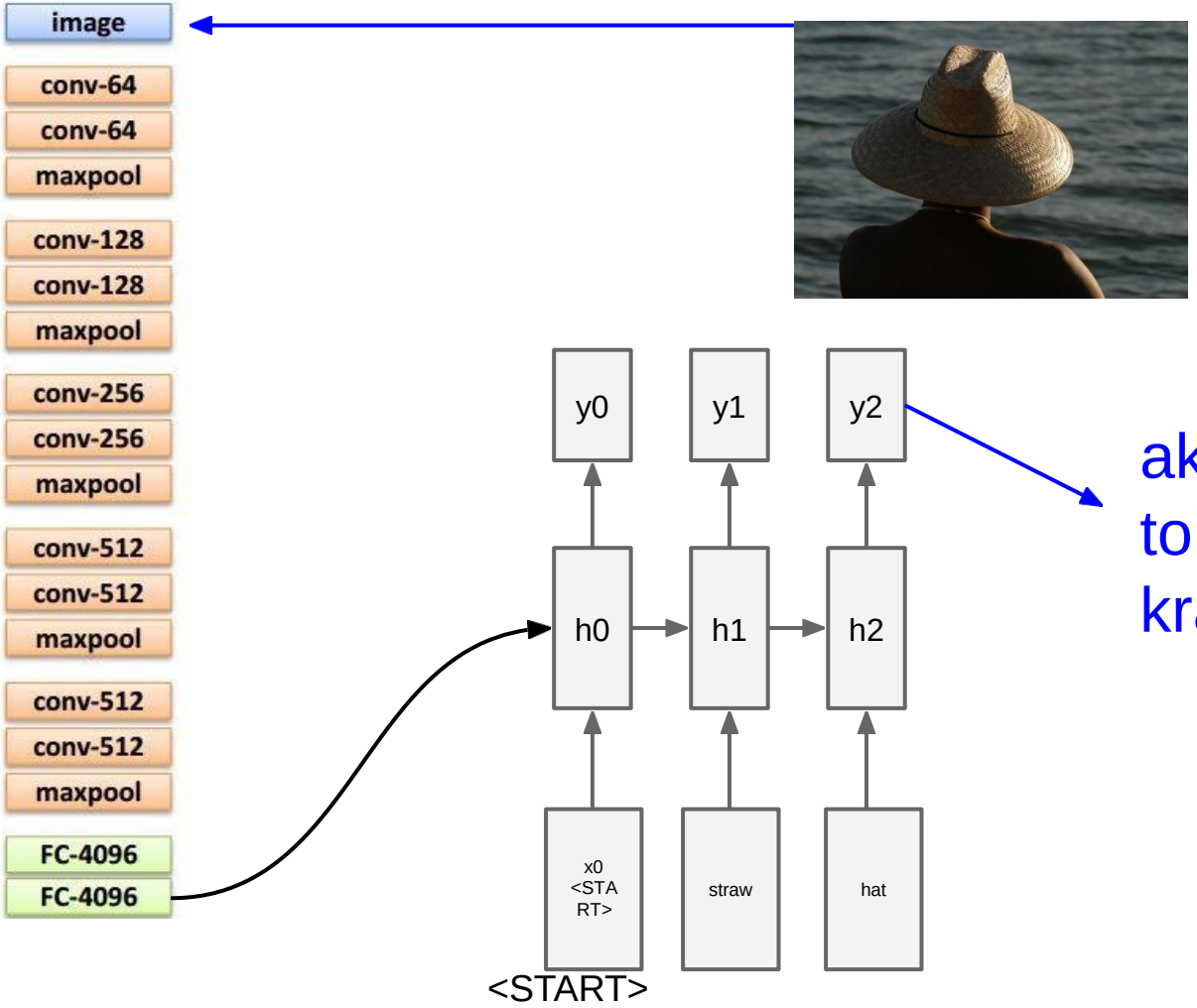


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
y1
y2
h0
h1
h2
x0
<STA
RT>
straw
hat
<START>

test slika



test slika

ako sempujemo token <END> => kraj.

Dodeljivanje opisa slikama: rezultati

Captions generated using neuraltalk2
All images are CC0 Public domain:
cat suitcase, cat tree, dog, bear,
surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

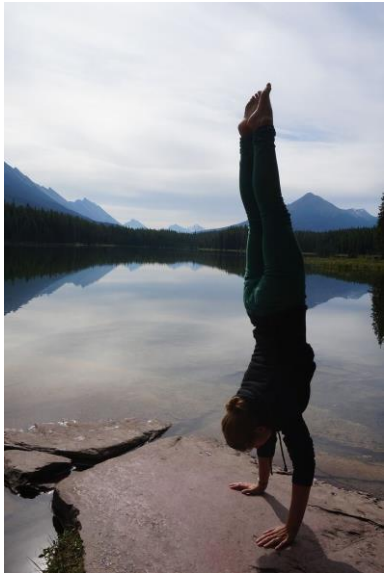
A man riding a dirt bike on a dirt track

Dodeljivanje opisa slikama: loši rezultati

Captions generated using neuraltalk2
All images are CC0 Public domain: fur coat, handstand, spider web, baseball

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



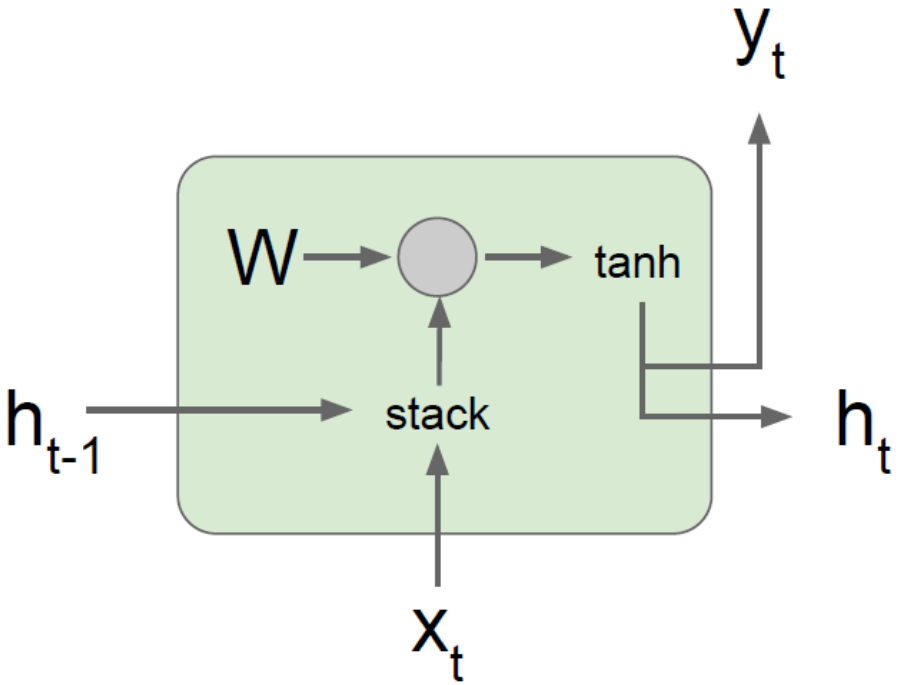
A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

Protok Gradijenta (*Gradient Flow*) u običnoj RNN

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

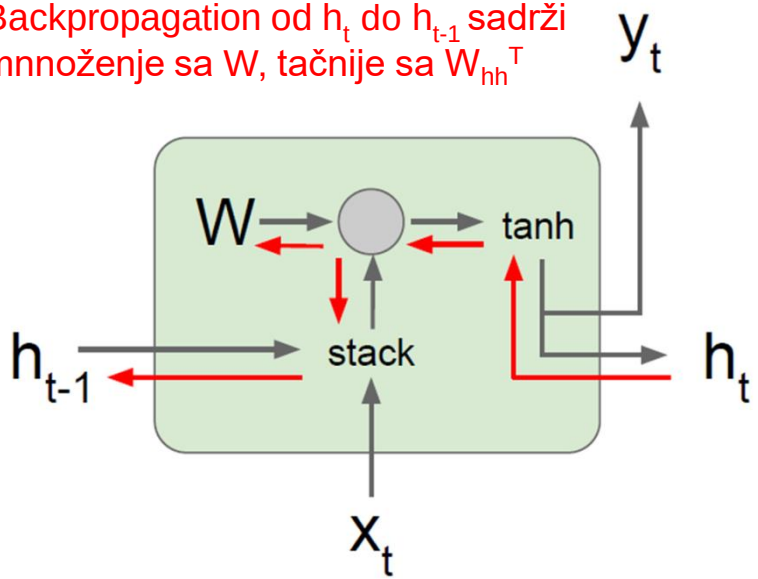


$$
\begin{aligned}
h_t &= \tanh(W_{hh} h_{t-1} + W_{xh} x_t) \\
&= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\
&= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)
\end{aligned}
$$

Protok Gradijenta u običnoj RNN

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Backpropagation od h_t do h_{t-1} sadrži mnnoženje sa W, tačnije sa W_{hh}^T



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix}\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

Protok Gradijenta u običnoj RNN – ponovljeno zbog lakšeg razumevanja prethodnog slajda

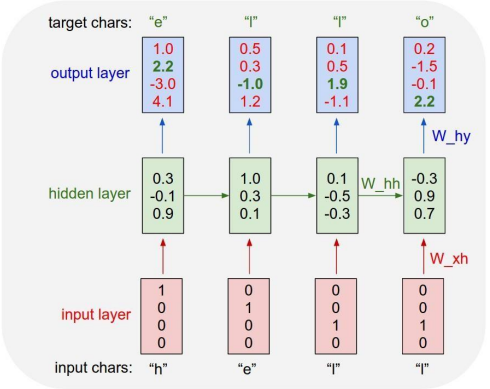
```python
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

```python
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                 [dWxh, dWhh, dWhy, dbh, dby],
                                 [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

```python
for t in reversed(xrange(len(inputs))):
  dy = np.copy(ps[t])
  dy[targets[t]] -= 1 # backprop into y
  dWhy += np.dot(dy, hs[t].T)
  dby += dy
  dh = np.dot(Why.T, dy) + dhnext # backprop into h
  dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
  dbh += dhraw
  dWxh += np.dot(dhraw, xs[t].T)
  dWhh += np.dot(dhraw, hs[t-1].T)
  dhnext = np.dot(Whh.T, dhraw)
for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
  np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + \boldsymbol{b_h})$$

$$y_t = W_{hy}h_t + \boldsymbol{b_y}$$



$$dh_3 = W_{hx}\, dy_3 + dhnext_4$$

$$dhraw_3 = (1 - hs_3^2)\, dh_3$$

$$dhnext_3 = W_{hh}\, dhraw_3$$

$$dh_2 = W_{hx}\, dy_2 + dhnext_3$$

$$dhraw_2 = (1 - hs_2^2)\, dh_2$$

$$dhnext_2 = W_{hh}\, dhraw_2$$

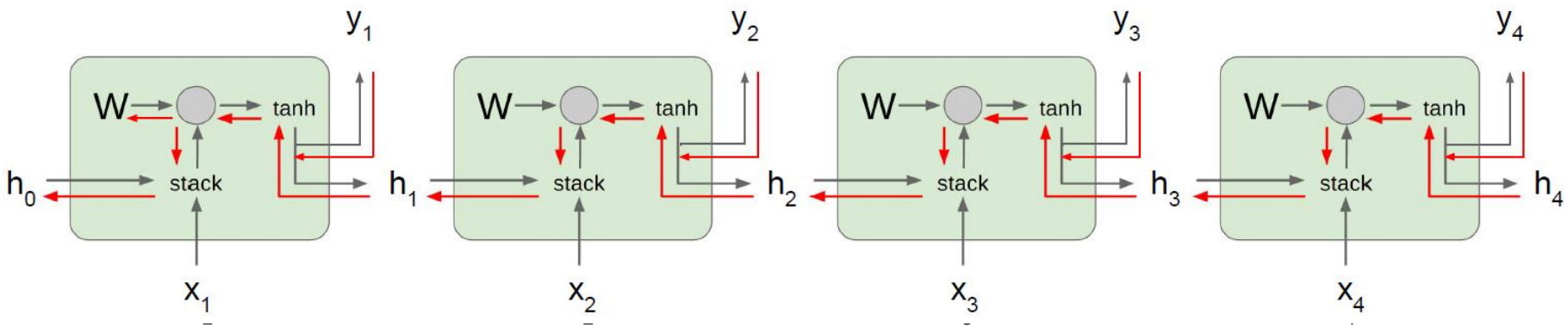
$$= W_{hh}(1 - hs_2^2)(W_{hx}\, dy_2 + dhnext_3)$$

$$= W_{hh}(1 - hs^2)(W_{hx}\, dy_2 + W_{hh}\, dhraw_3)$$

$$= W_{hh}^2 (1 - hs^2)\, dhraw_3 \ldots\ldots$$

Protok Gradijenta u običnoj RNN

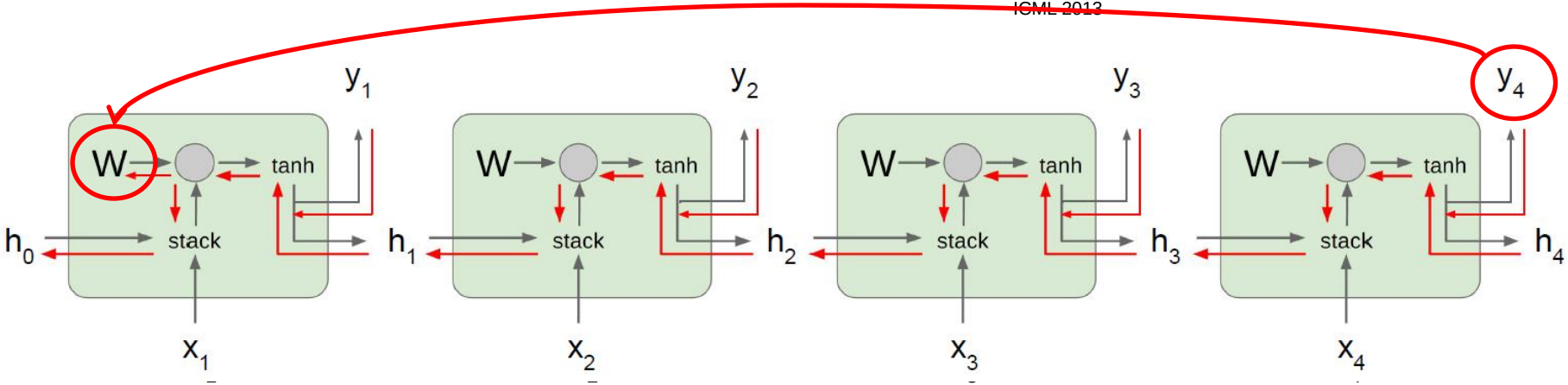
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Protok Gradijenta u običnoj RNN

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

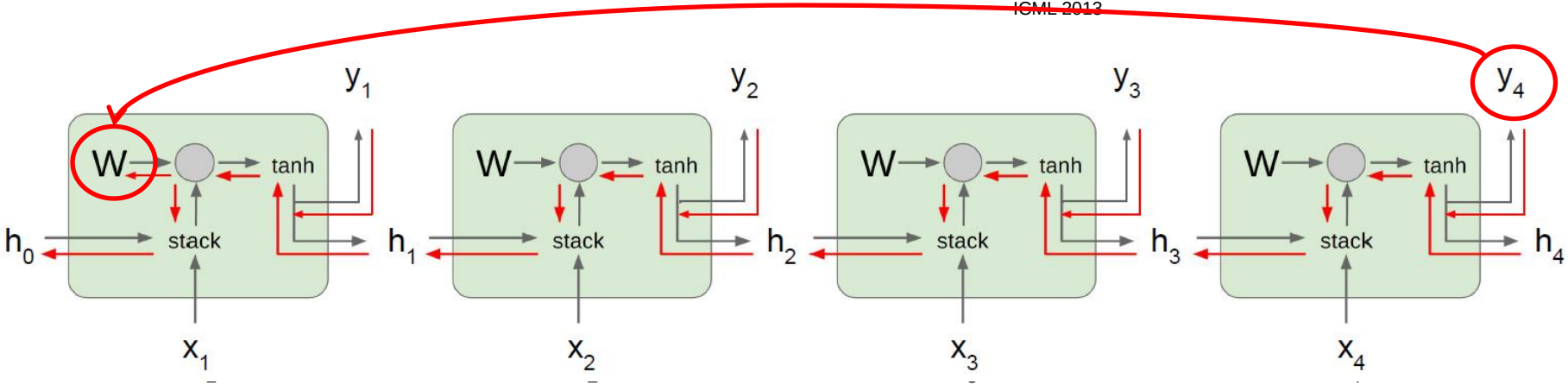


$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W}$$

Protok Gradijenta u običnoj RNN

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

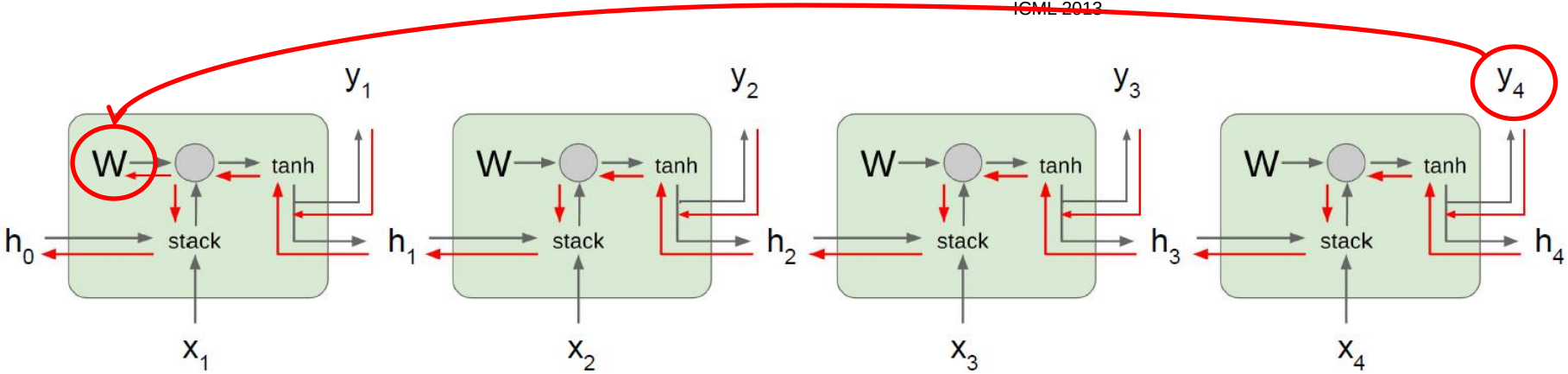


$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}$$

Protok Gradijenta u običnoj RNN

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



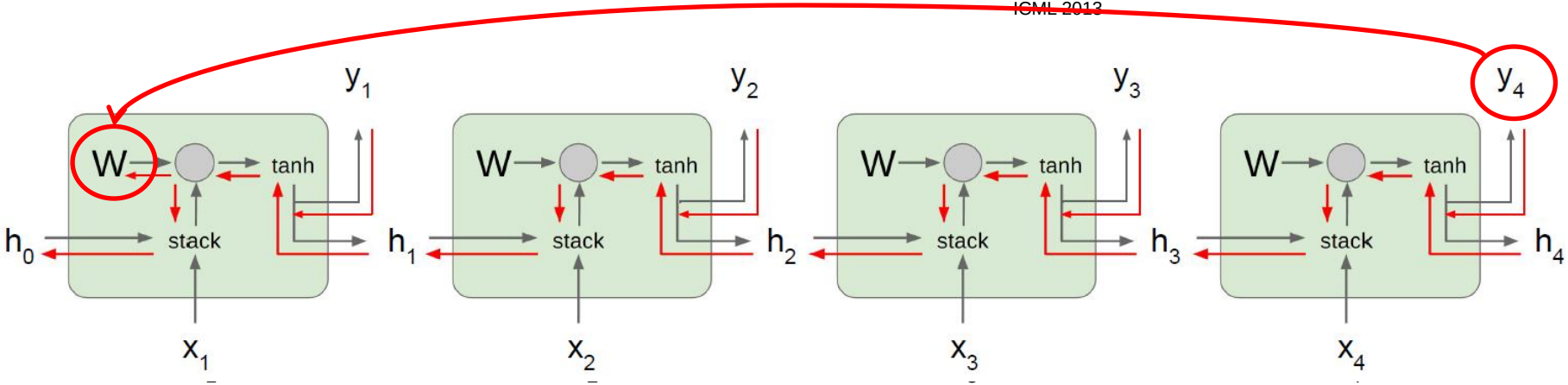
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\boxed{\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh} h_{t-1} + W_{xh} x_t) W_{hh}}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \boxed{\frac{\partial h_t}{\partial h_{t-1}}} \right) \frac{\partial h_1}{\partial W}$$

Protok Gradijenta u običnoj RNN

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



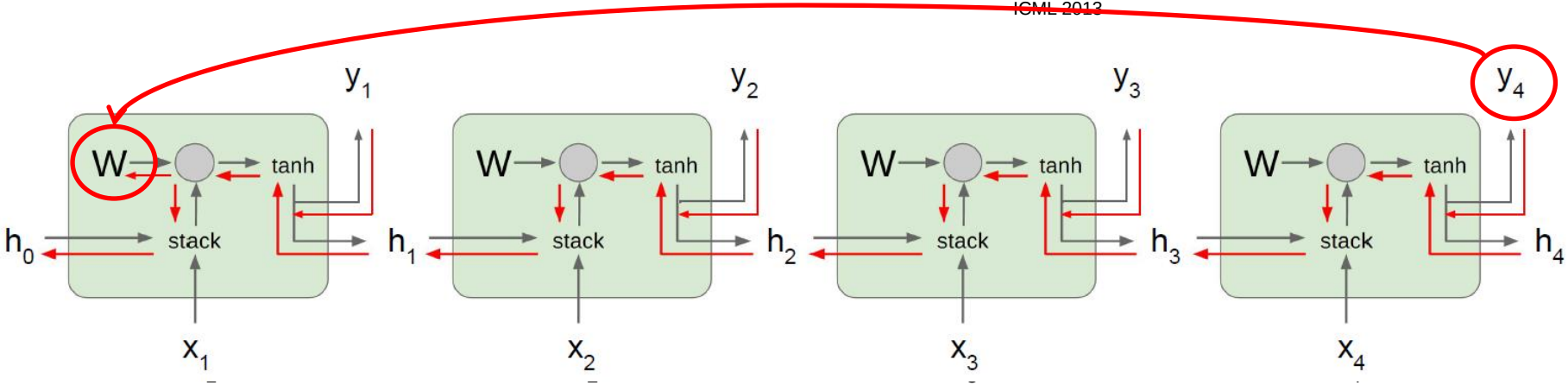
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Vrlo često << 1
Gradijenti postaju nula. (*vanishing gradient*)

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\left(\prod_{t=2}^{T} \boxed{tanh'\left(W_{hh}h_{t-1} + W_{xh}x_t\right)}\right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Protok Gradijenta u običnoj RNN

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



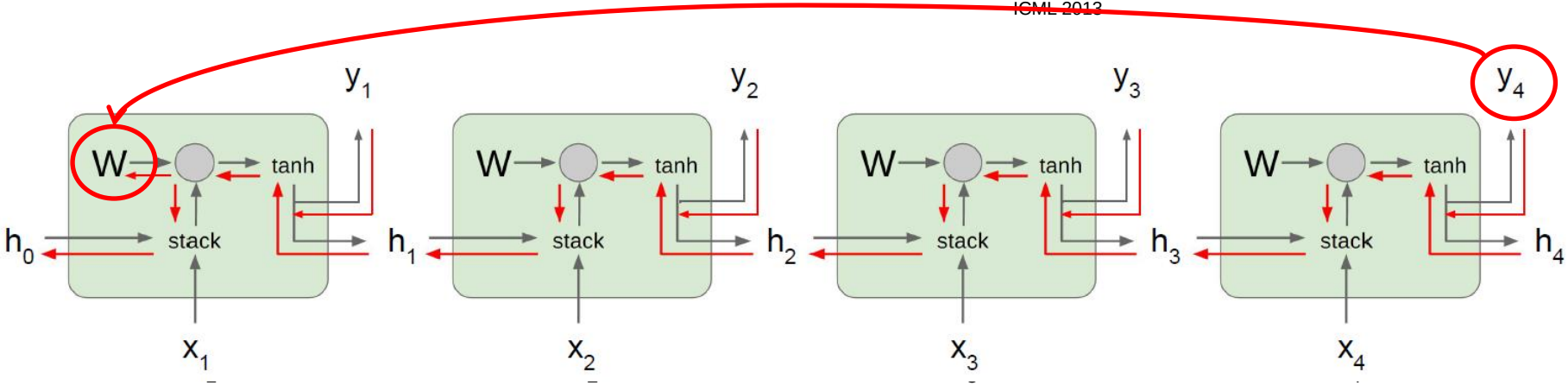
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Šta bi bilo ako ne koristimo *tanh*?

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

Protok Gradijenta u običnoj RNN

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Šta bi bilo ako ne koristimo *tanh*?

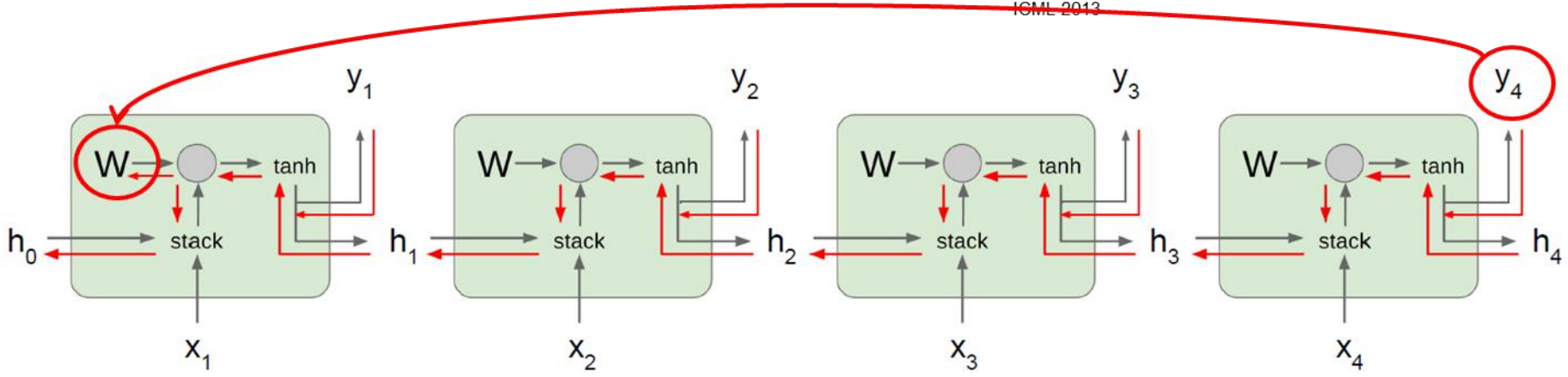
Ako su vrednosti W >> 1:
Vrednosti gradijenta eksplodiraju

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

Ako su vrednosti W << 1:
Vrednosti gradijenta idu u nula (*vanishing gradient*)

Protok Gradijenta u običnoj RNN

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Izračunavanje gradijenta za h_0 u odnosu na grešku na izlazu ključuje mnogo množenja sa W (i ponavljanja tanh)

Ako su vrednosti W >> 1:
Vrednosti gradijenta eksplodiraju

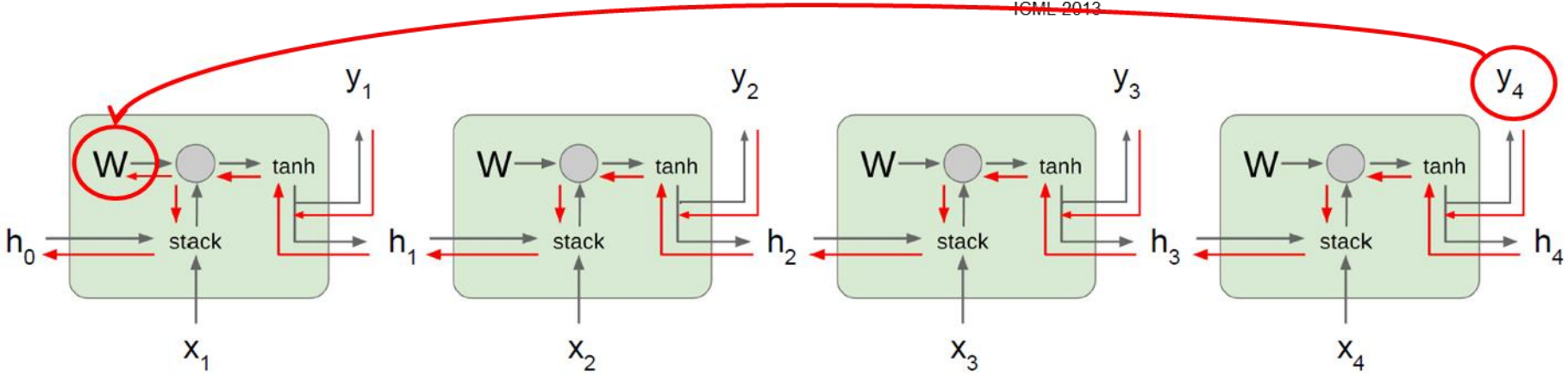
Ako su vrednosti W << 1:
Vrednosti gradijenta idu u nula (*vanishing gradient*)

Ograničavanje gradijenta: Skalirati gradijent ako mu je vrednost norme prevelika

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Protok Gradijenta u običnoj RNN

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



Izračunavanje gradijenta za h_0 u odnosu na grešku na izlazu ključuje mnogo množenja sa W (i ponavljanja tanh)

Ako je najveći spostvena vrednost W > 1:
Vrednosti gradijenta eksplodiraju

Ako su vrednosti W << 1:
Vrednosti gradijenta idu u nula (*vanishing gradient*)

→ Promenti arhitekturu RNN

Long Short Term Memory (LSTM)

Obična RNN

$$h_t = \tanh\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

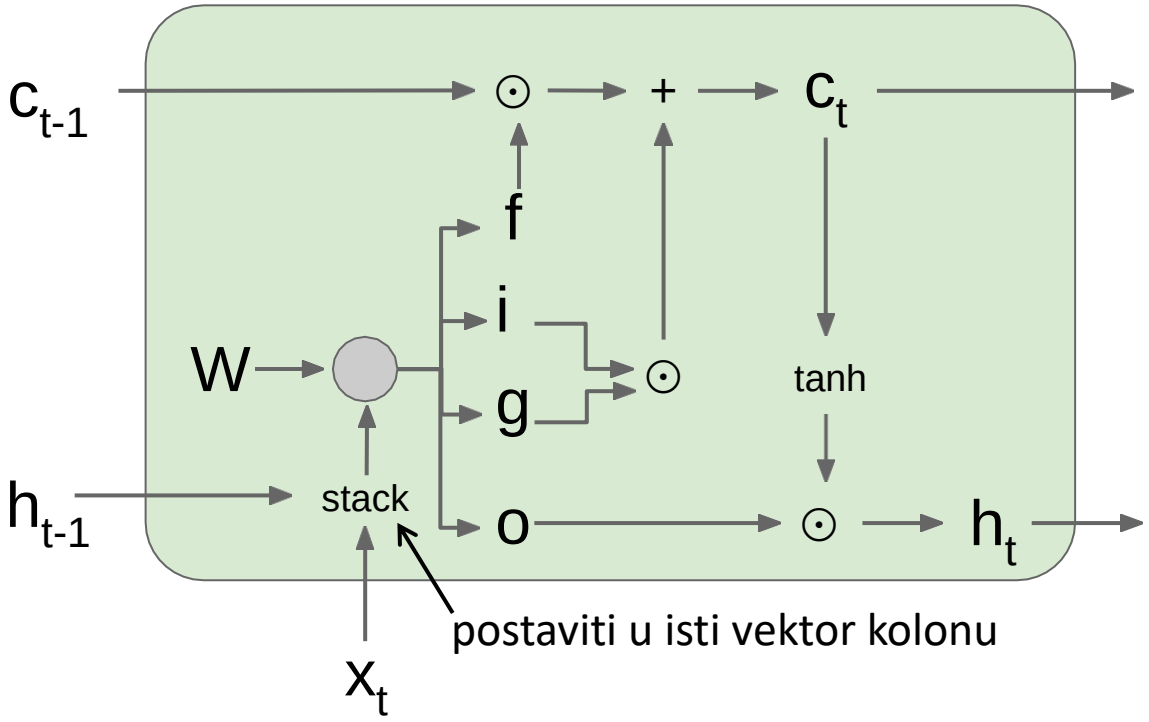
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

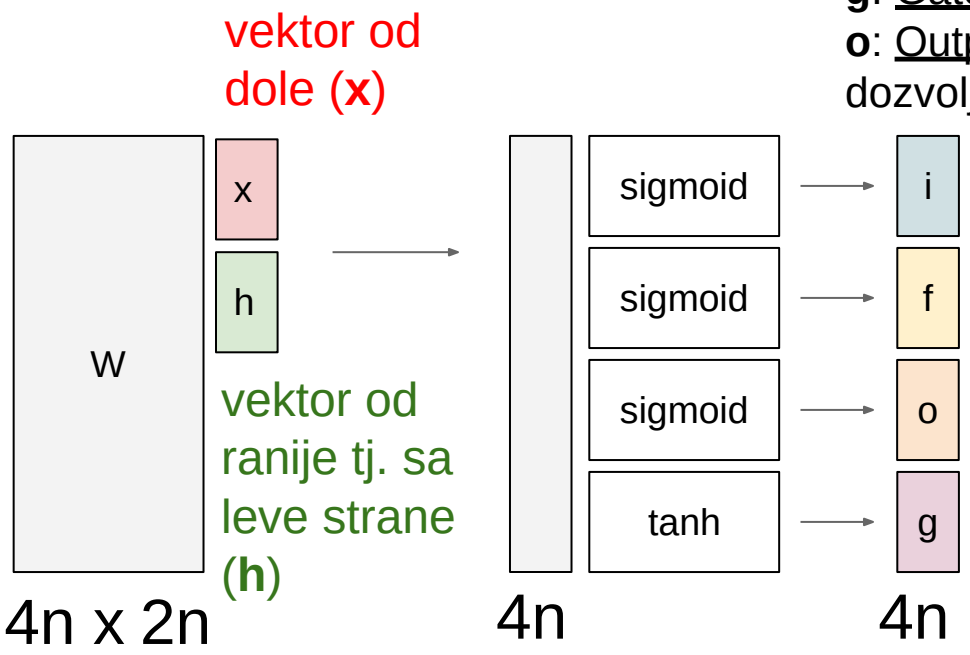
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

f: Forget gate, da li da resetujemo ćeliju c
i: Input gate, da li da upišemo nešto u c
g: Gate gate (?), šta pišemo tačno u c
o: Output gate, koliko vrednosti c dozvoljavamo da ode u h



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

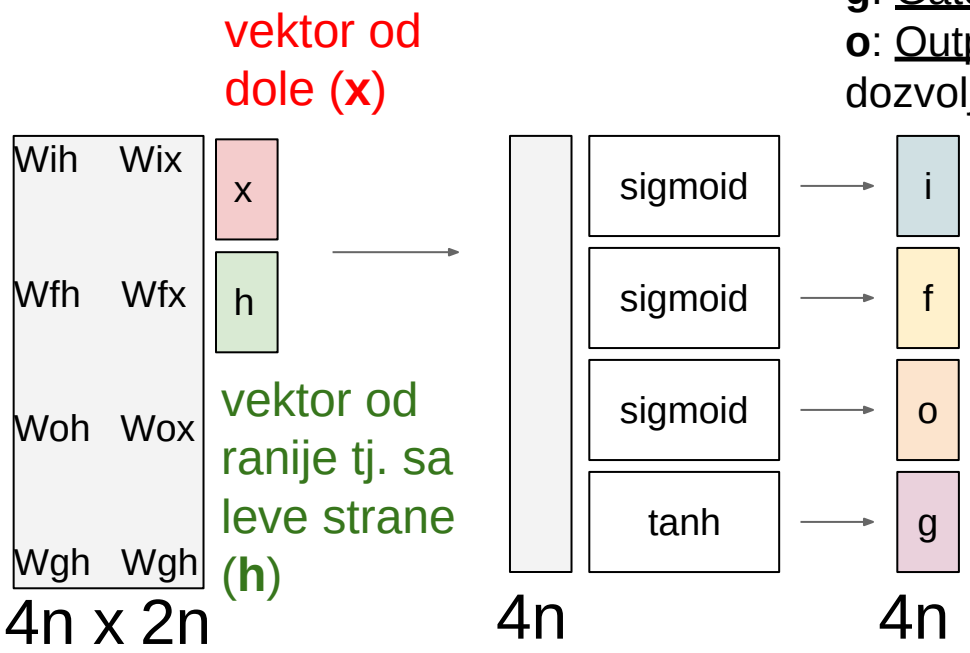
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

f: Forget gate, da li da resetujemo ćeliju c
i: Input gate, da li da upišemo nešto u c
g: Gate gate (?), šta pišemo tačno u c
o: Output gate, koliko vrednosti c dozvoljavamo da ode u h



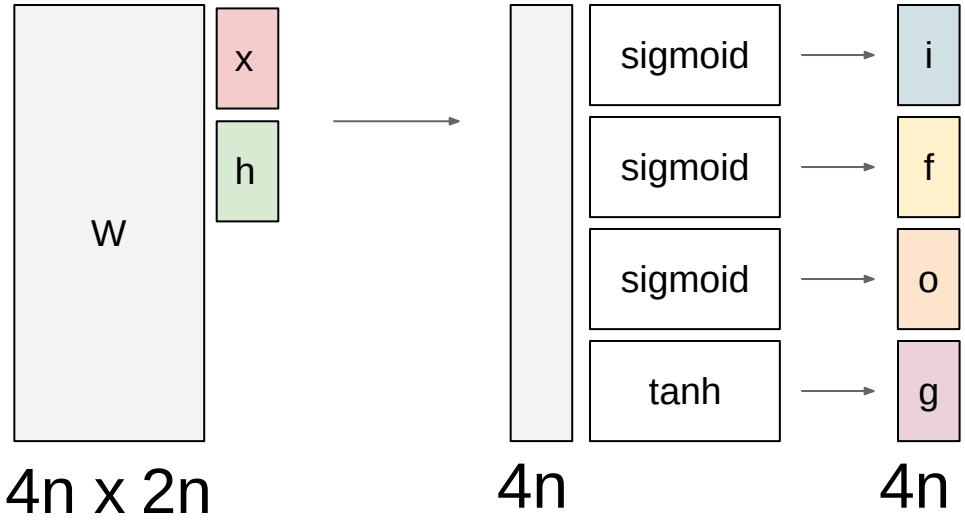
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

Ako je x dimenzija n=5 i h je dim. n=5
Onda spojimo x i h u jedan vektor dužine 2n=10 koji ima jednu kolonu.
W sadrži posebnu matricu za svaku kapiju.
Prva matrica je W_i za input kapiju i ona je dimenzija n=5 vrsta i 2n=10 kolona.
Ulaz u prvi sigmoid koji daje izlaz *i* je:
W_i *(x;h) – gde ';' znači da su x i h u istoj koloni.
Analogno imamo matrice W_f, W_o, W_g
Svaka od njih se obučava sa backpopagation.

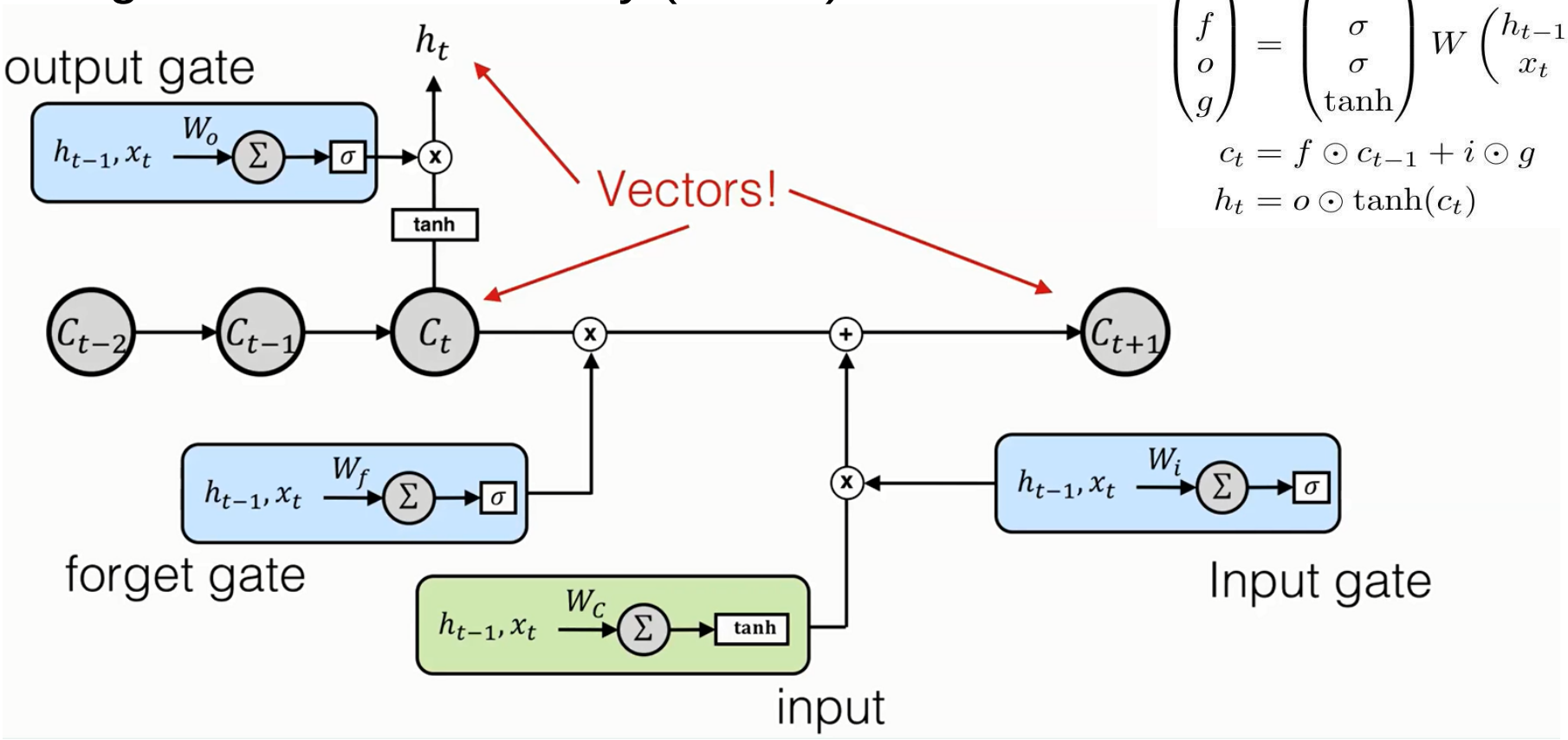


Long Short Term Memory (LSTM)

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

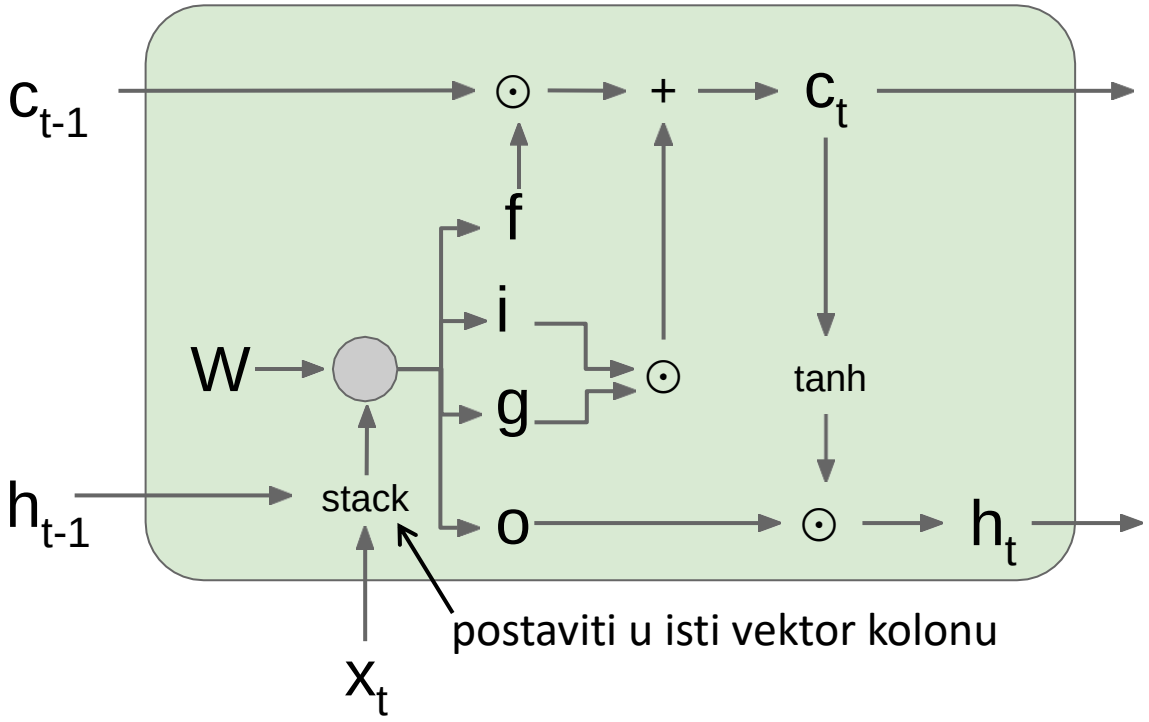
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

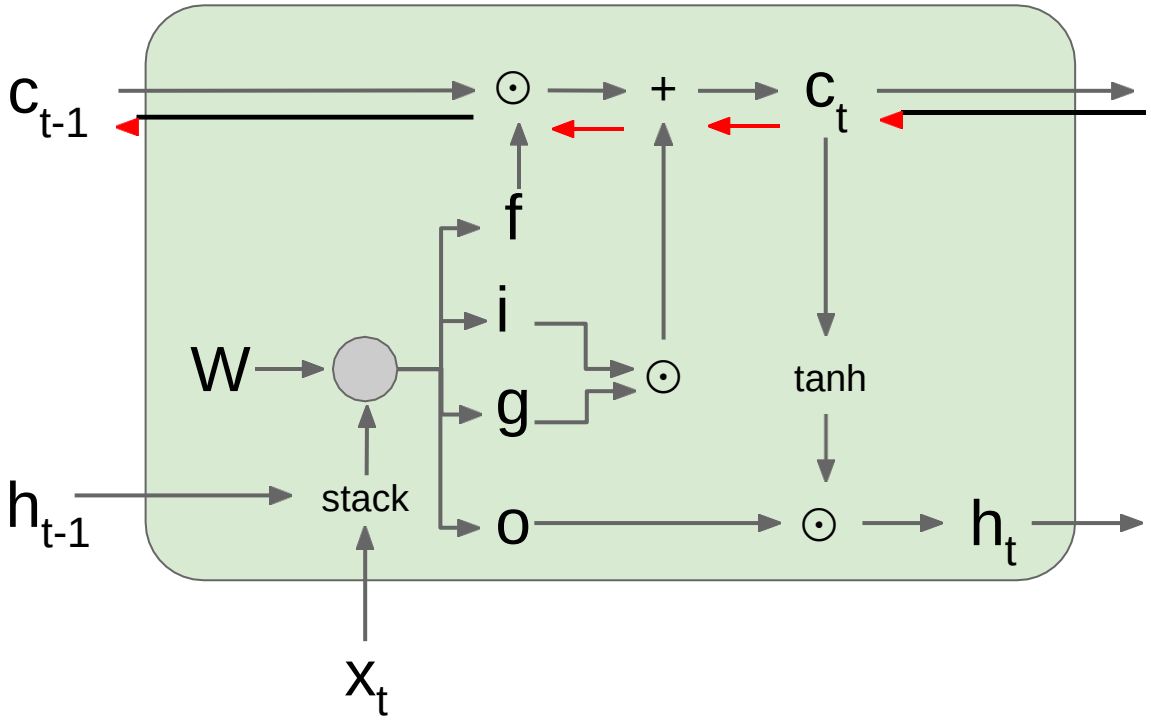


$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Protok Gradijenta



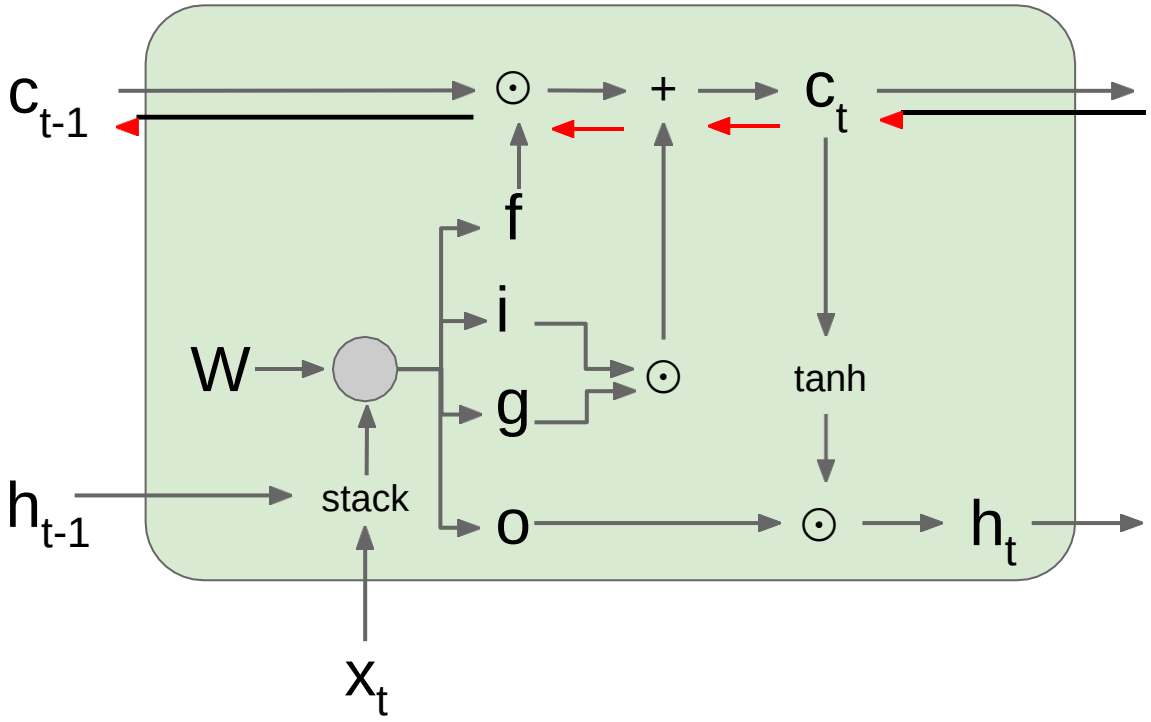
Backpropagation od c_t do c_{t-1} sadrži skalarani proizvod sa vektorom f bez množenja sa gradijentom tanh ili sigmoida koji mogu da utiču na odlazak gradijenta u 0.

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Protok Gradijenta



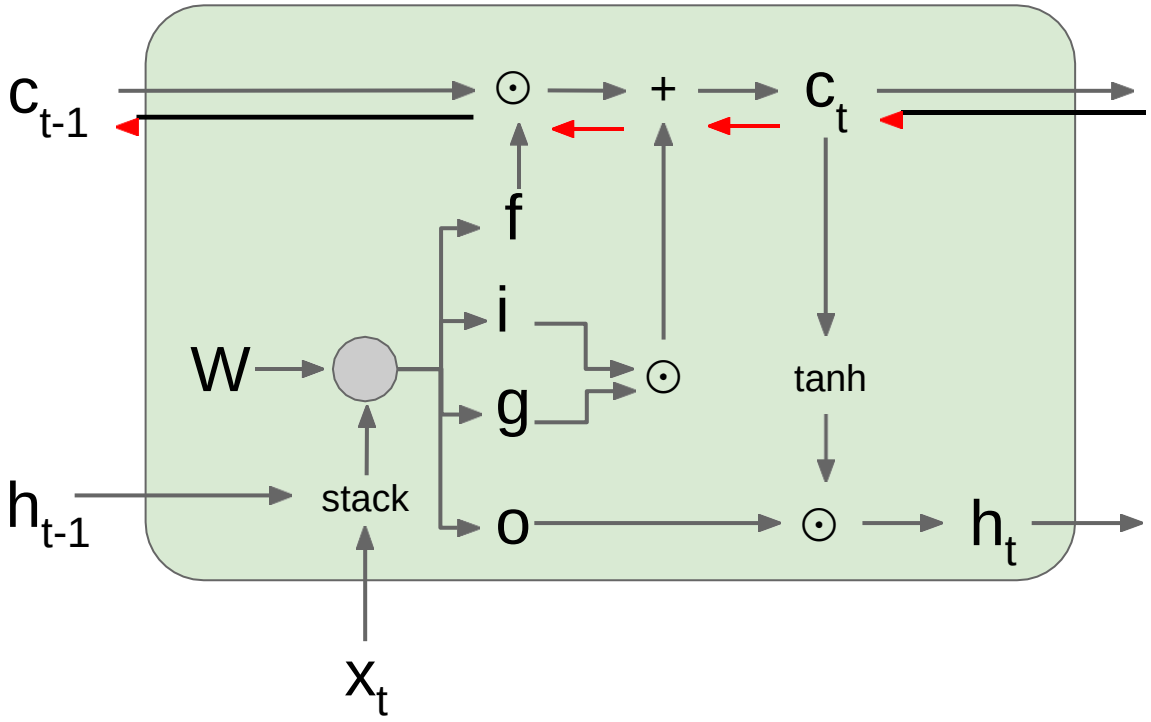
Backpropagation od c_t do c_{t-1} sadrži sadrži i druge komponente (gradijente kapija i i kapija g) koje mogu da dobiju 0 vrednost kroz iteracije.

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Protok Gradijenta



Suština je u tome da bar jedna kopomenta gradijenta od ct do ct-1 neće otići u 0, a to je f. Ako f ode u 0, onda je to slučaj kada mreža „odlučila“ da zaboravi prethodni ct. Koliko daleko će mreža „gledati unazad“ zavisi od tipa problema koji rešava.

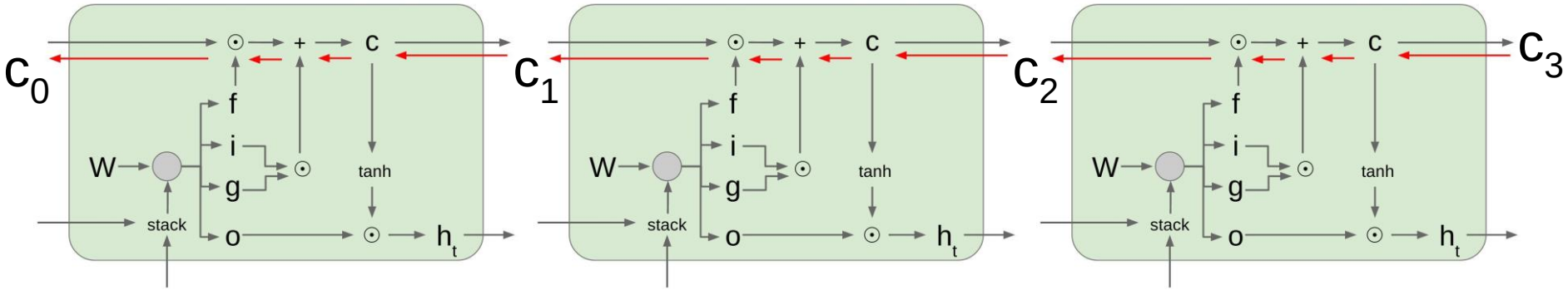
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

Gradijent teče bez prekida!



Druge RNN Varijante

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

[*An Empirical Exploration of Recurrent Network Architectures*, Jozefowicz et al., 2015]

MUT1:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + \tanh(x_t) + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}h_t + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh}x_t + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT3:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}\tanh(h_t) + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh}x_t + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

Rezime

- RNN pružaju mnogo fleksibilnosti u dizajnu arhitektura neuronskih mreža
- Obične RNN su jednostavne, ali ne rade baš dobro
- Obično se koriste LSTM ili GRU: kod kojih je protok gradijenta bolji
- Trenutno je jako puno istraživanja posvećenu pronalaženju boljih i jednostavnijih arhitektura RNN.

Konvolutivne Neuronske Mreže
Convolutional Neural Networks, CNN

Predavač: Aleksandar Kovačević

Slajdovi preuzeti sa CS 231n, Stanford

http://cs231n.stanford.edu/

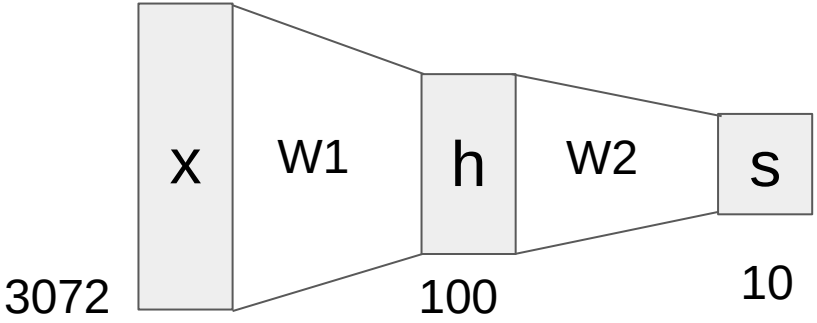
Do sada: Neuronske mreže

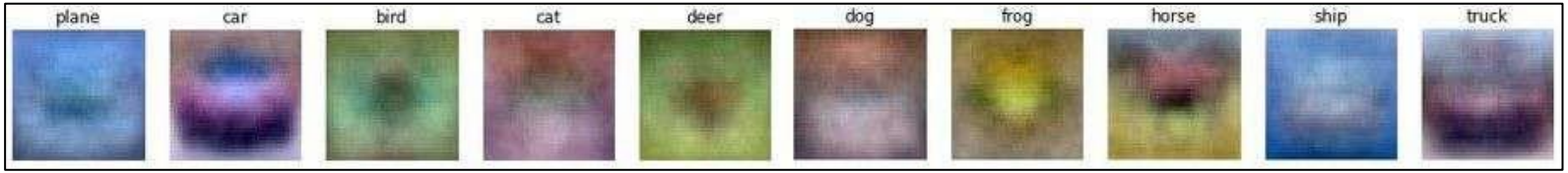
Linearna skor funkcija:

$$f = Wx$$

2-slojna Neuronska Mreža

$$f = W_2 \max(0, W_1 x)$$





Danas: Konvolutivne Neuronske Mreže

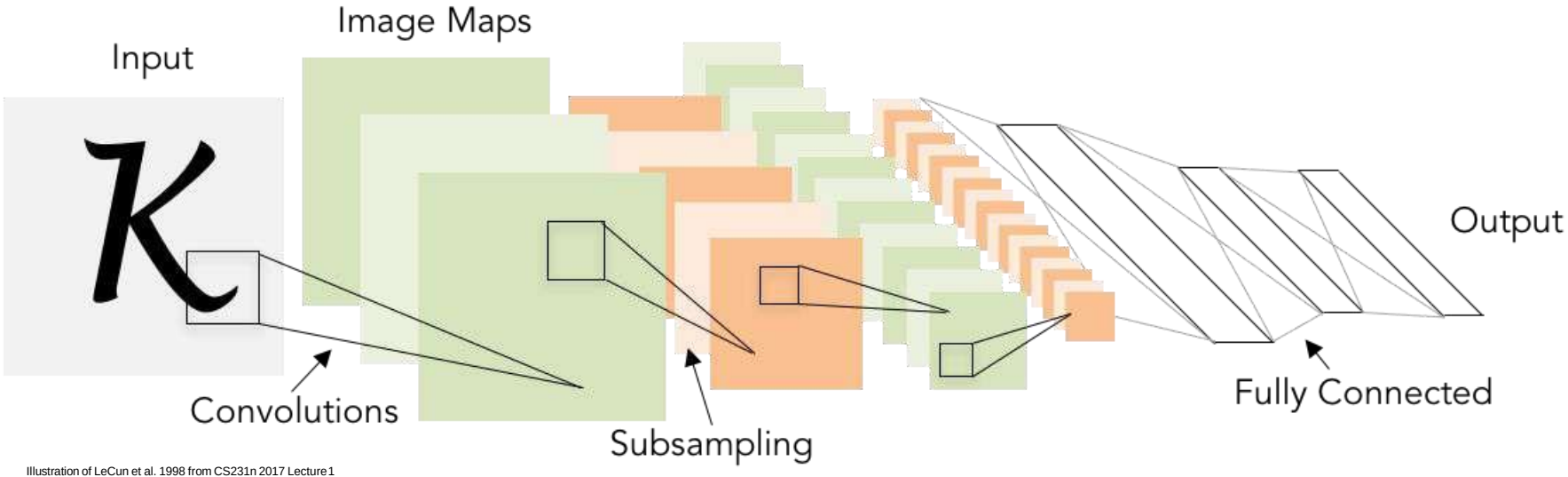


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

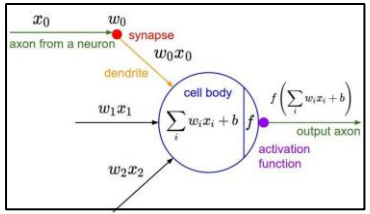
Malo istorije...

Mašina **Mark I Perceptron** je prva implementacija perceptron algoritma.

Mašina je bila povezana na kameru koja je koristila 20×20 kadmijum sulfid fotoćelije da proizvede sliku od 400 piksela.

mašina je mogla da prepoznaje slova abecede

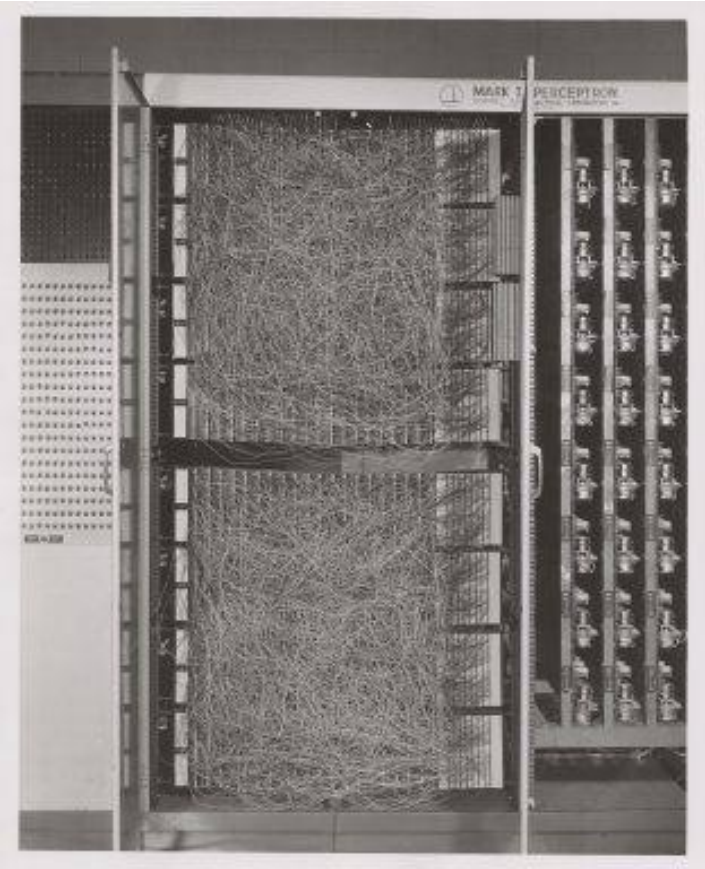
$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$



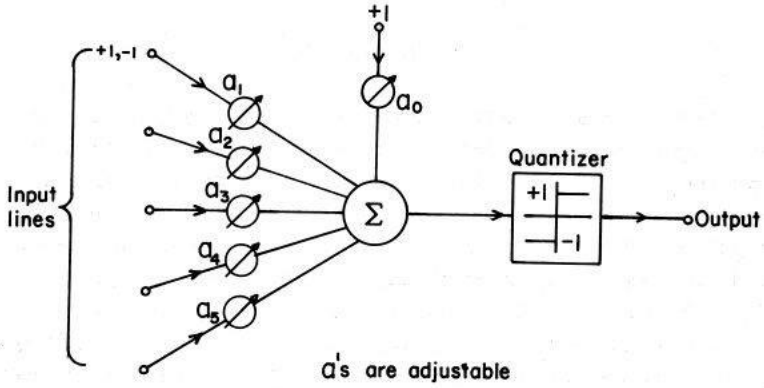
pravilo za promenu težina:

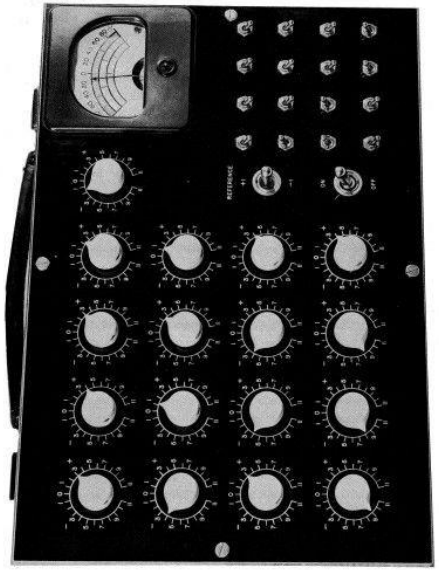
$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i},$$

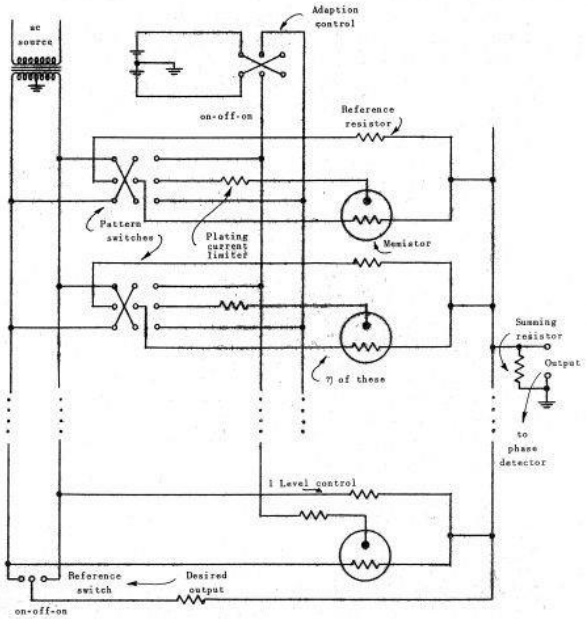
Frank Rosenblatt, ~1957: Perceptron



Malo istorije...



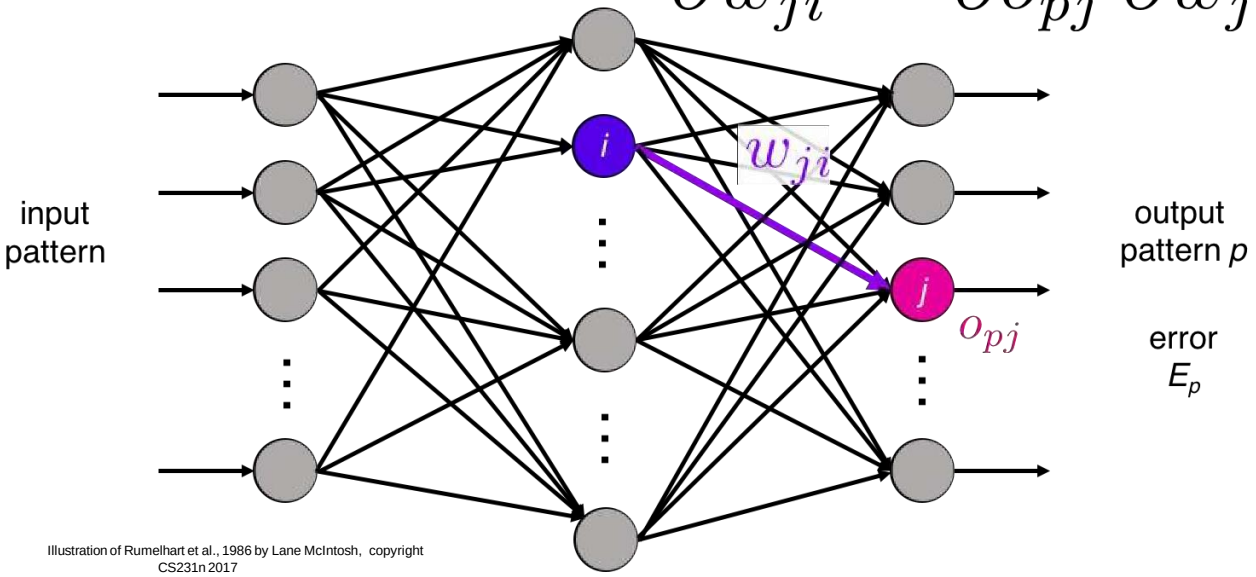




Widrow and Hoff, ~1960: Adaline/Madaline

Malo istorije...

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$



backprop
matematika

Rumelhart et al., 1986: Prvi put da je back-propagation postao popularan

Malo istorije...

[Hinton and Salakhutdinov 2006]

Počinje interesovanje za Deep Learning

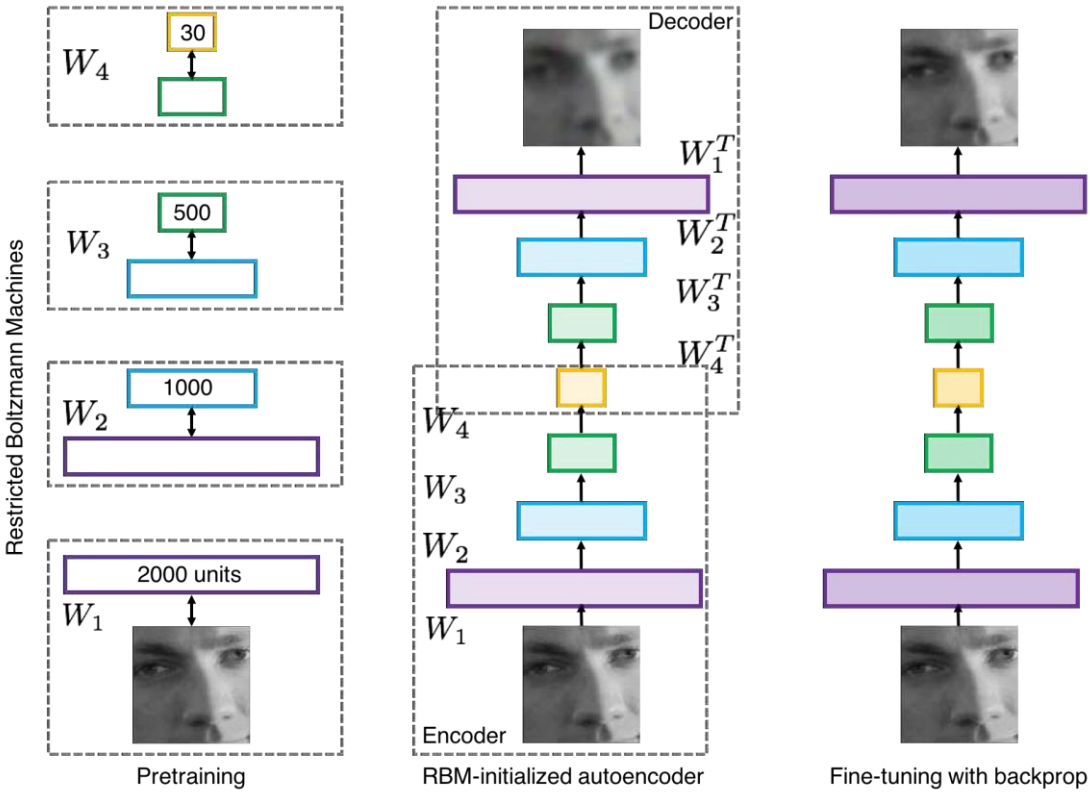


Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

Prvi dobri rezultati

Acoustic Modeling using Deep Belief Networks
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010
Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

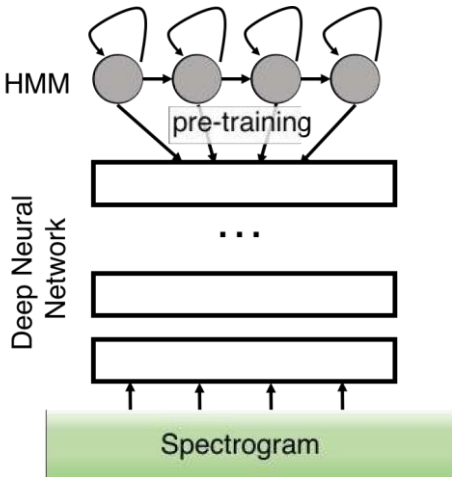
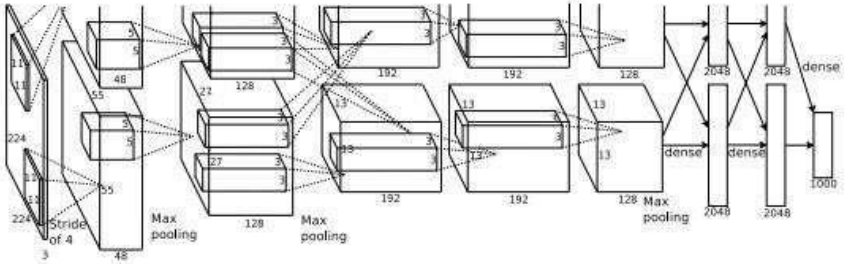
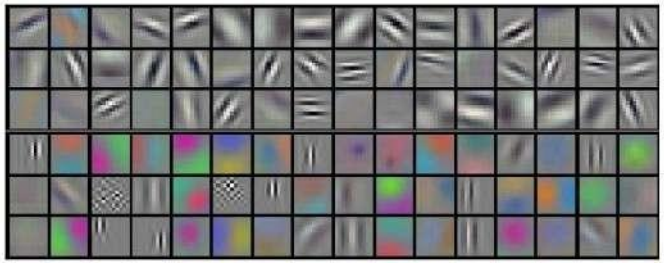


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012





Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Malo Istorije:

Hubel & Wiesel,

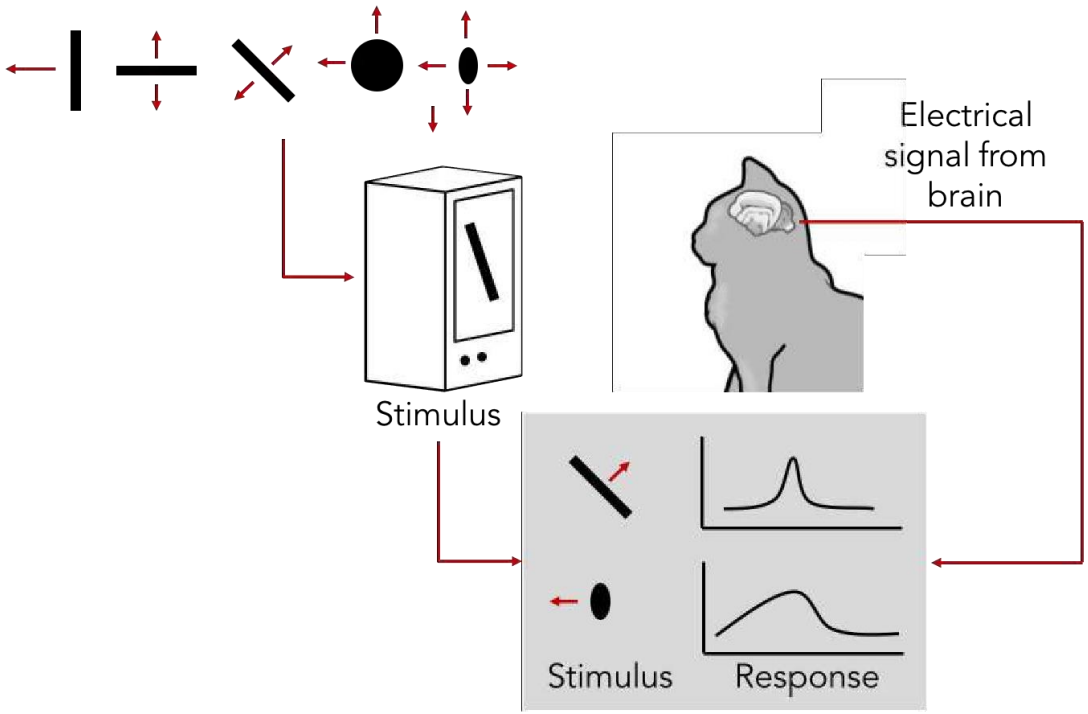
1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

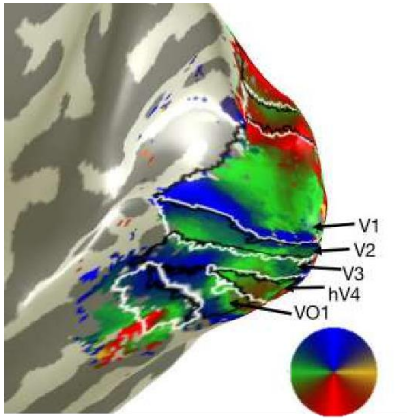
1968...



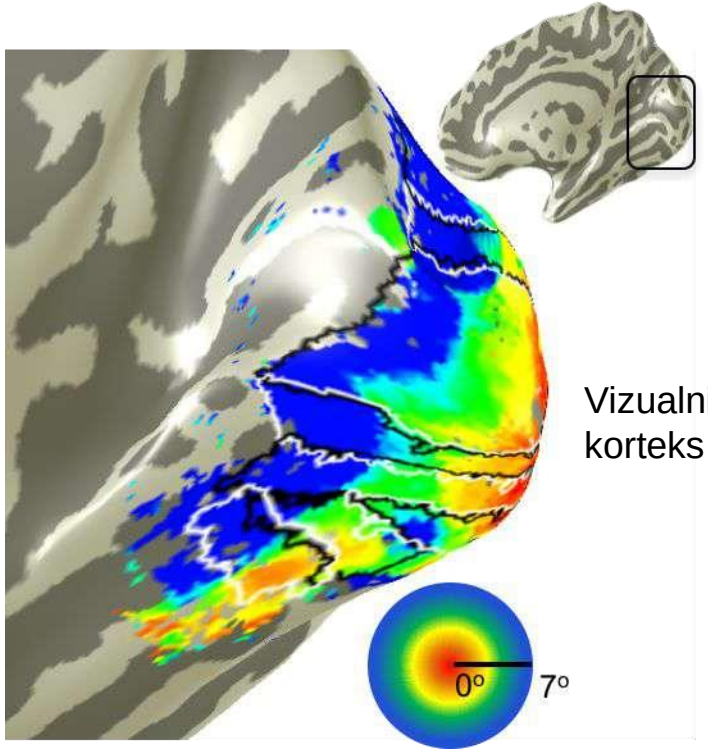
Cat image by CNX OpenStax is licensed under CC BY 4.0; changes made

Malo istorije...

Topografska mapa korteksa:
ćelije koje su blizu u korteksu procesiraju regione koji su blizu u vizualnom polju



Ljudski mozak



Vizualni korteks

Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

Hijerahjijska Organizacija

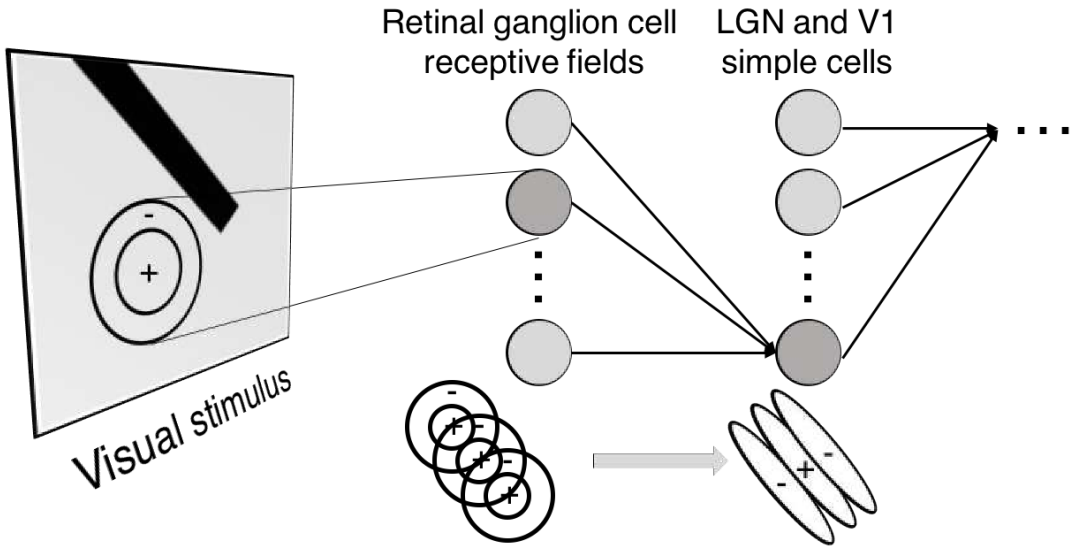
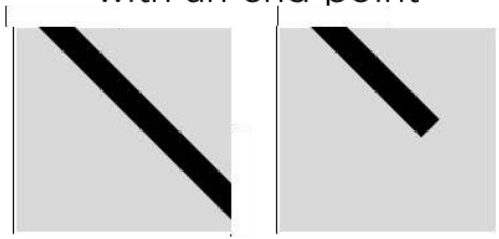


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point

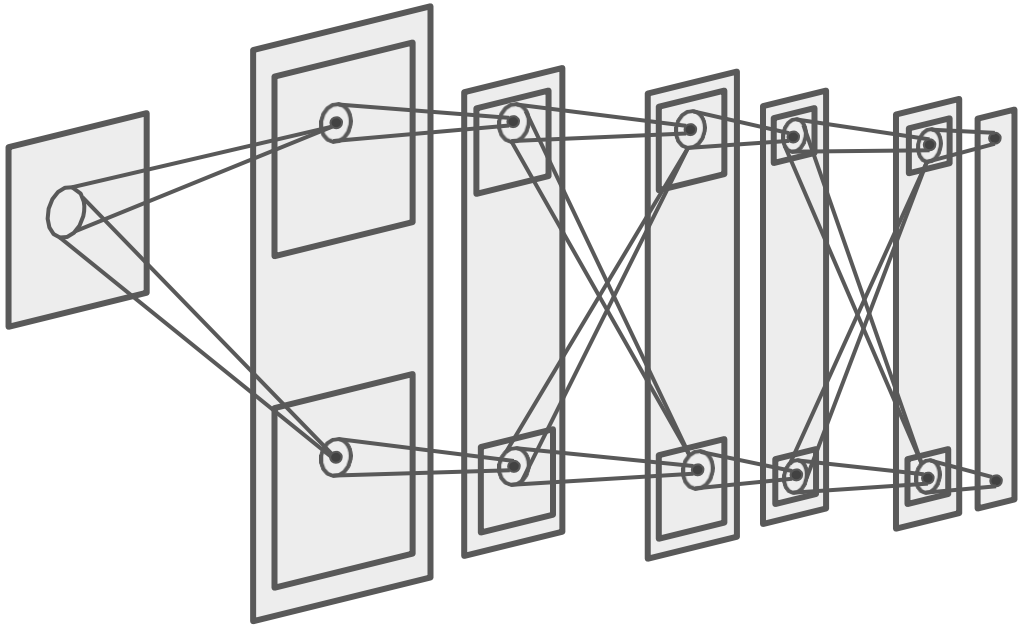


No response

Response (end point)

Malo istorije:

Neocognitron
[Fukushima 1980]

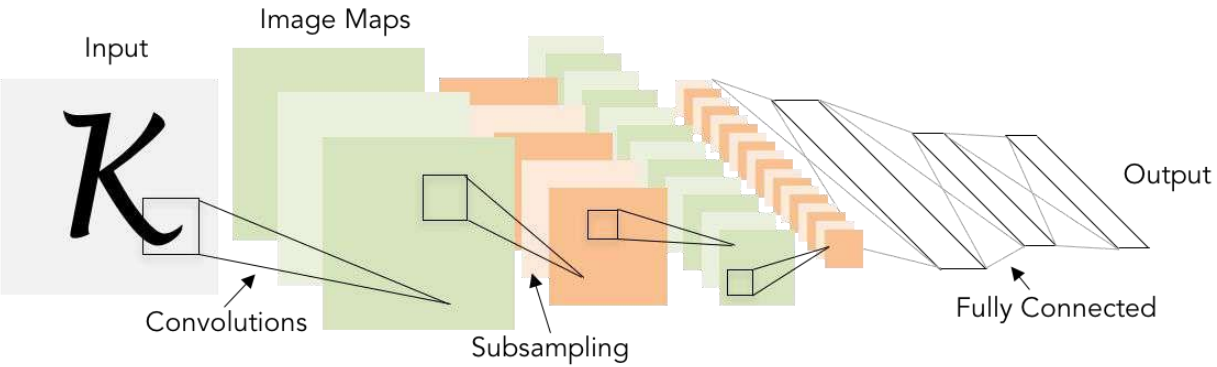


“sendvič” arhitektura (SCSCSC…)
jednostavne ćelije: parametri koji se uče
kompleksne ćelije: agregacija vrednosti jednostavnih ćelija

Malo istorije:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

Malo istorije:

ImageNet Classification with Deep Convolutional Neural Networks *[Krizhevsky, Sutskever, Hinton, 2012]*

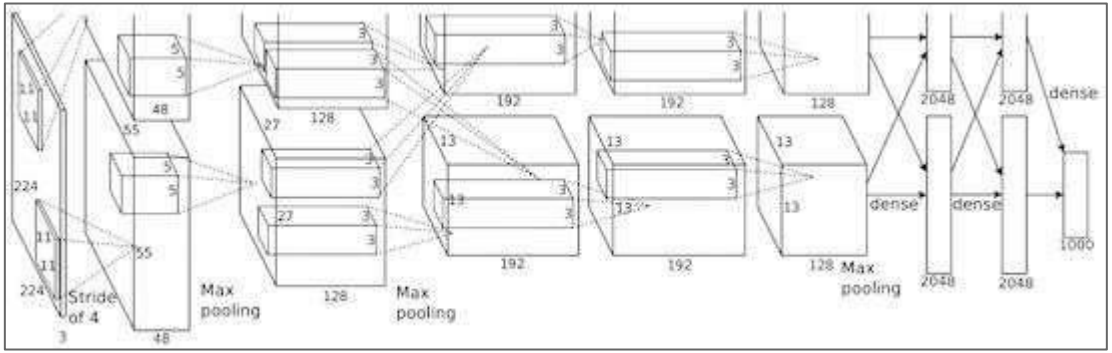


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

Danas: Konvolutivne Mreže su svuda

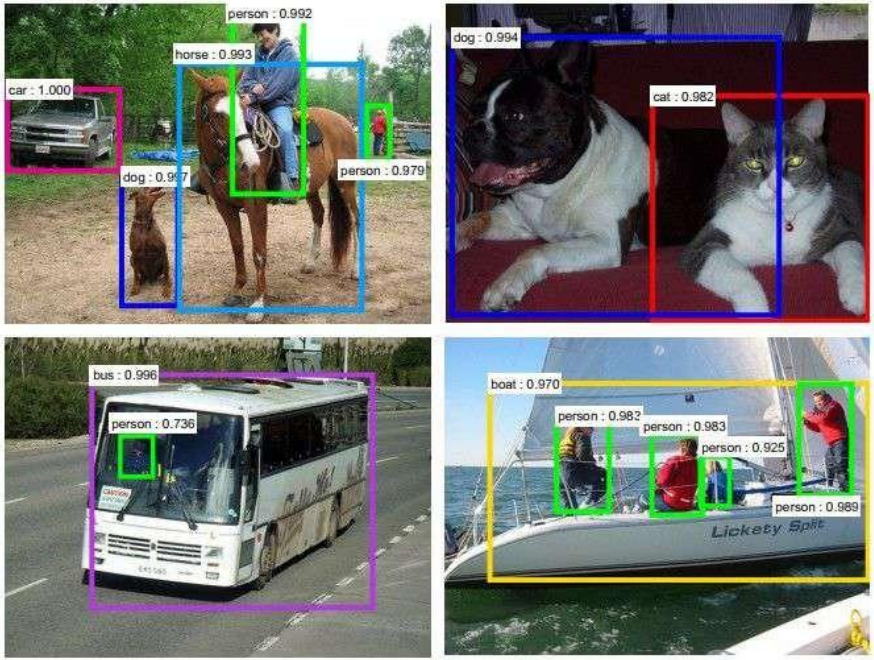
Klasifikacija

Pronalaženje

Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Danas: Konvolutivne Mreže su svuda

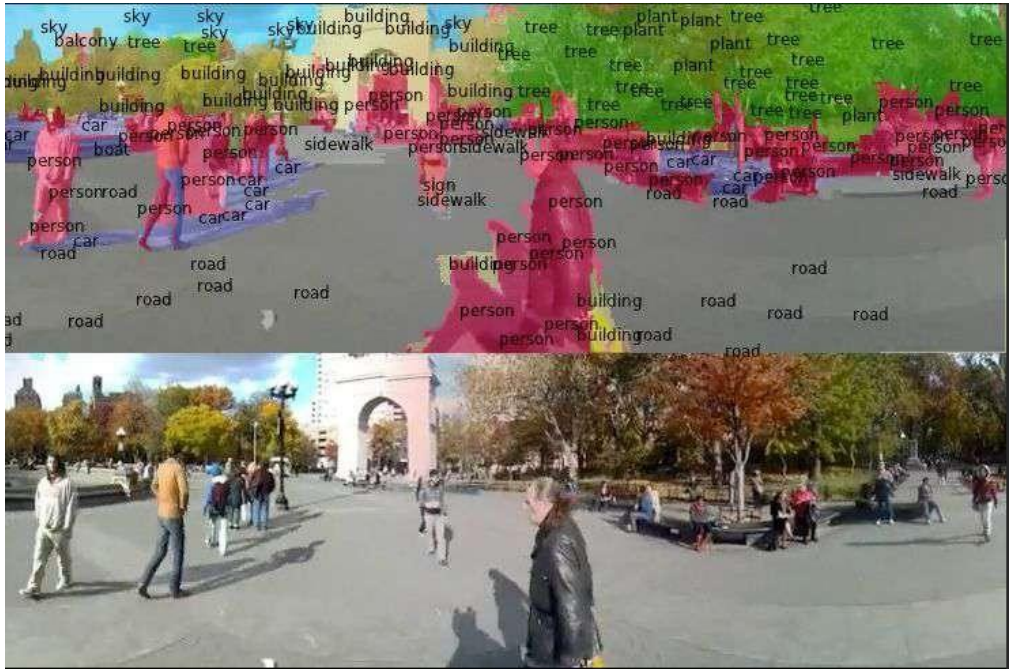
Detekcija



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentacija



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Danas: Konvolutivne Mreže su svuda

self-driving cars

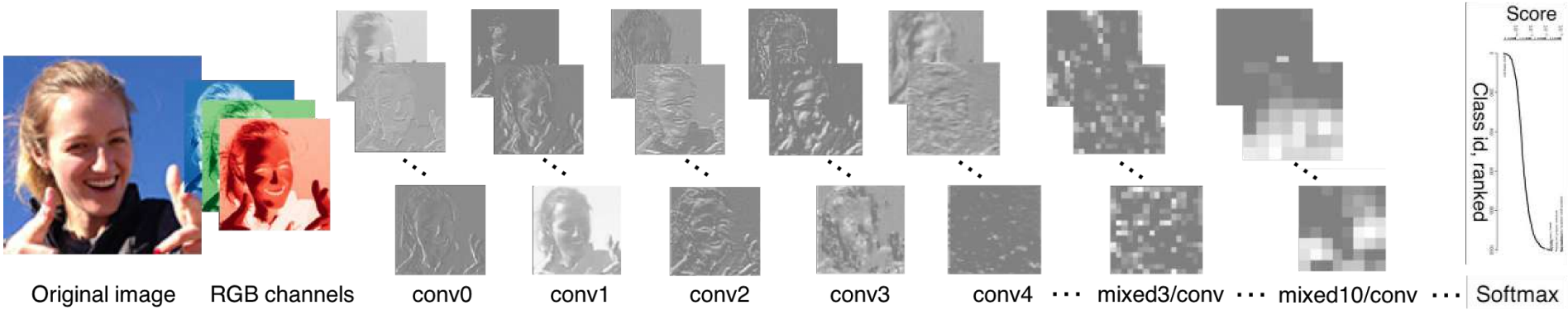
Photo by Lane McIntosh. Copyright CS231n 2017.

This image by GBPublic_PR is licensed under CC-BY 2.0

NVIDIA Tesla line

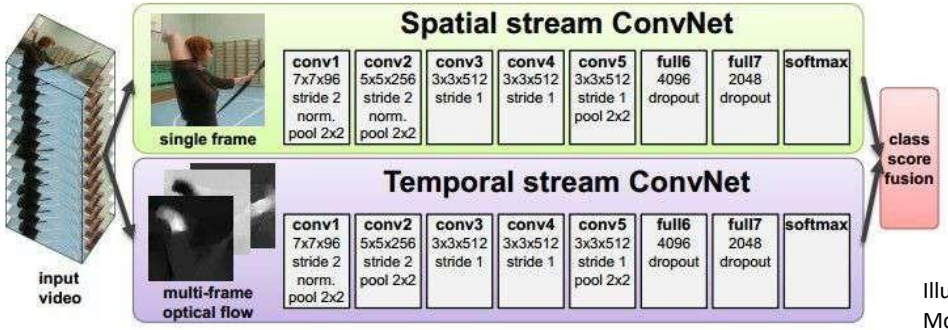
Za embedded sisteme tipično se koriste NVIDIA Tegra kartice, sa integrisanim GPU i procesoru zasnovanom na ARM arhitekturi.

Danas: Konvolutivne Mreže su svuda



[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

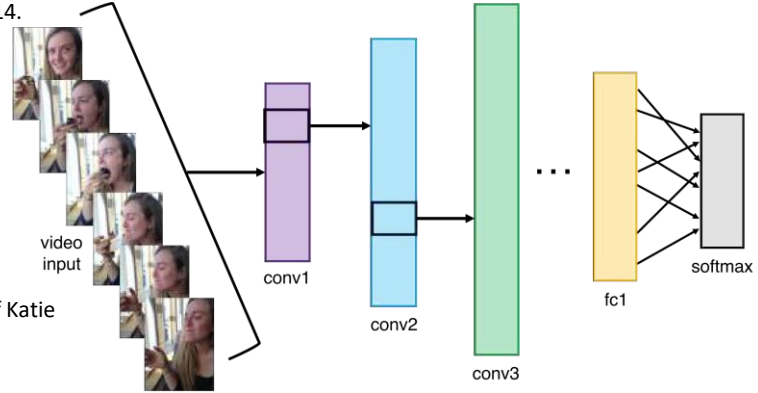
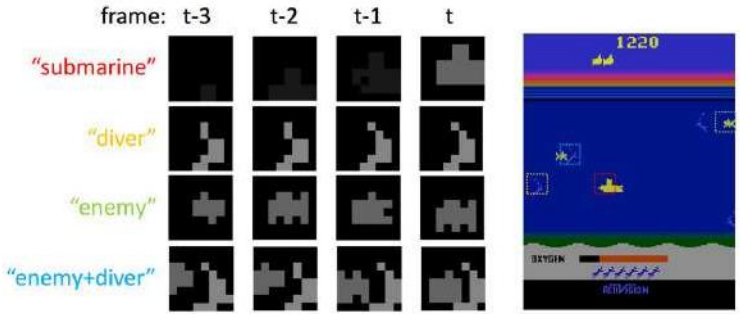


Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

Danas: Konvolutivne Mreže su svuda

[Toshev, Szegedy 2014]

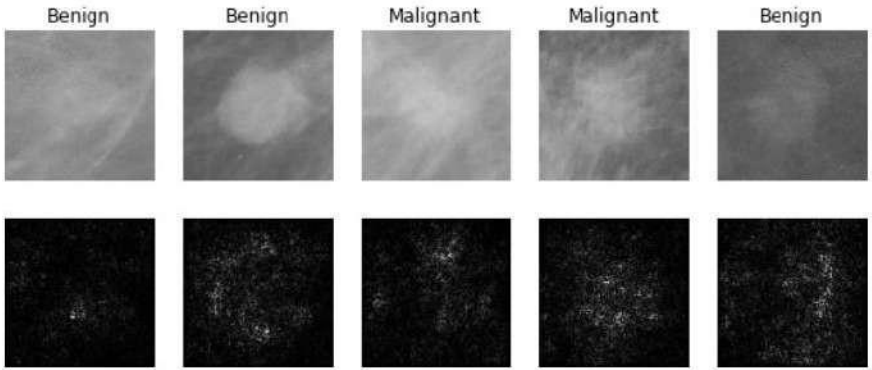
Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.



[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

Danas: Konvolutivne Mreže su svuda



[Levy et al. 2016]

Figure copyright Levy et al. 2016.
Reproduced with permission.

[Sermanet et al. 2011]
[Ciresan et al.]

Photos by Lane McIntosh.
Copyright CS231n 2017.

[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.

Mnih and Hinton, 2010

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

Bez grešaka

A white teddy bear sitting in the grass

Manje greške

A man in a baseball uniform throwing a ball

Ima neke veze sa slikom

A woman is holding a cat in her hand

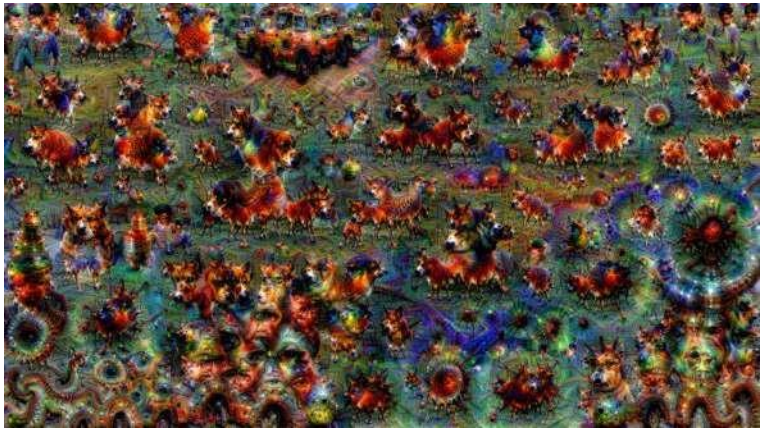
A man riding a wave on top of a surfboard

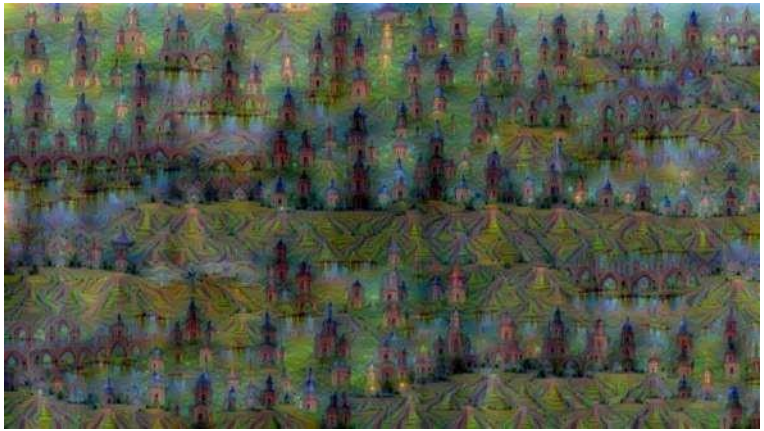
A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-player-shortstop-infield-1045263/

Captions generated by Justin Johnson using Neuraltalk2





Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.

Original image is CC0 publicdomain
Starry Night and Tree Roots by Van Gogh are in the public domain Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017; reproduced with permission

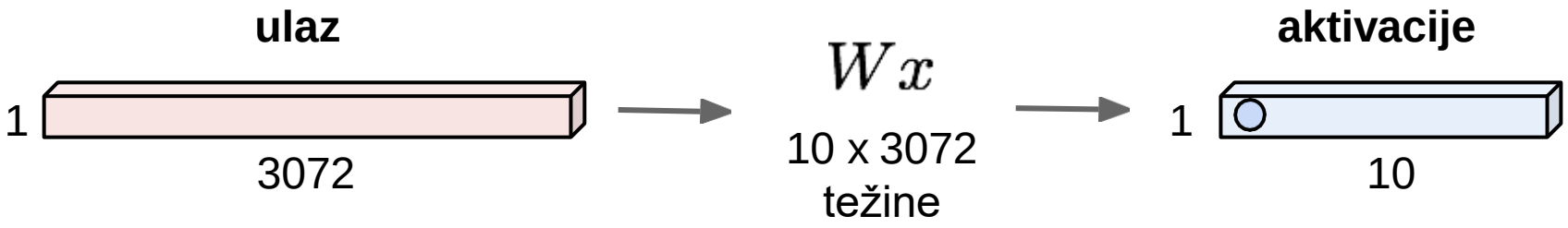
Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016 Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Konvolutivne Neuronske Mreže

(Prvo bez analogije sa ljudskim mozgom)

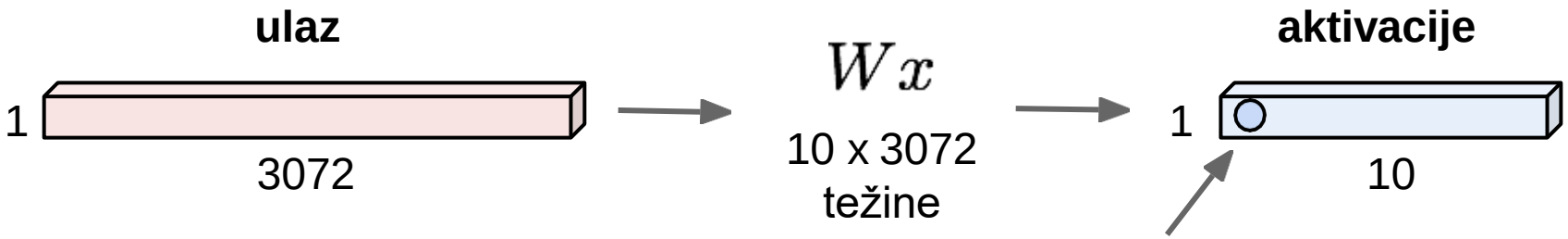
Potpuno Povezan Sloj

32x32x3 slika -> razvučemo u niz 3072 x 1



Potpuno Povezan Sloj

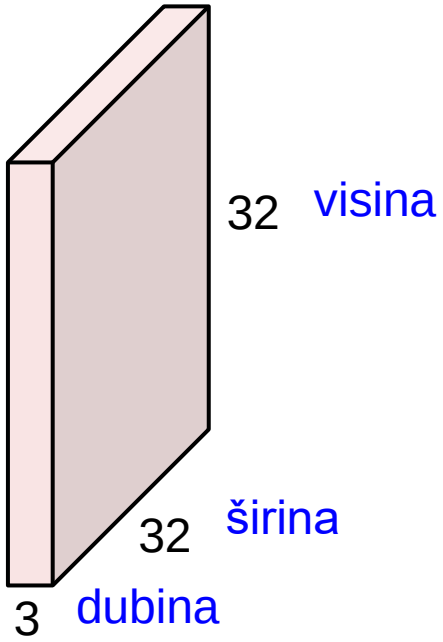
32x32x3 slika -> razvučemo u niz 3072 x 1



1 broj (vrednost):
rezultat skalarnog proizvoda jednog reda u W i ulaza (3072-dimenzioni skalarni proizvod)

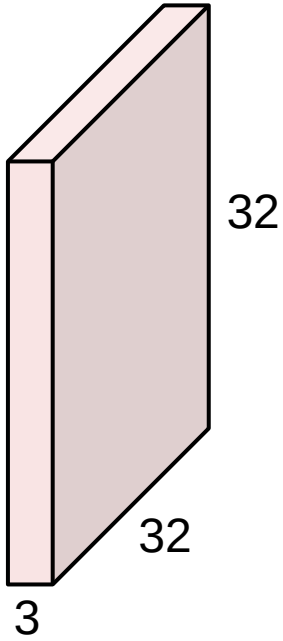
Konvolutivn Sloj

32x32x3 slika -> očuvavamo prostornu strukturu (slika ostaje 2d, a ne 1d niz)

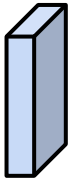


Konvolutivni Sloj

32x32x3 slika



5x5x3
filter

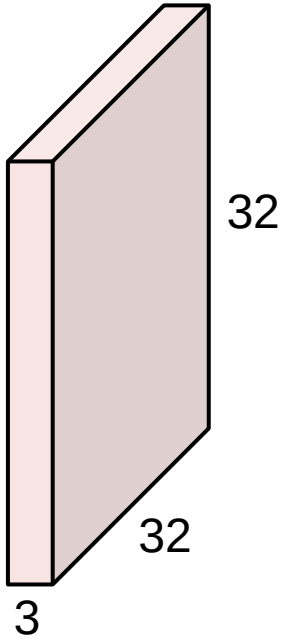


Vršimo Konvoluciju filtera sa slikom tj. “vučemo” filter prostorno po slici i računamo skalarni proizvod

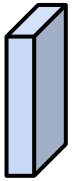
Konvolutivni Sloj

Filteri su uvek iste dubine kao i slika

32x32x3 slika

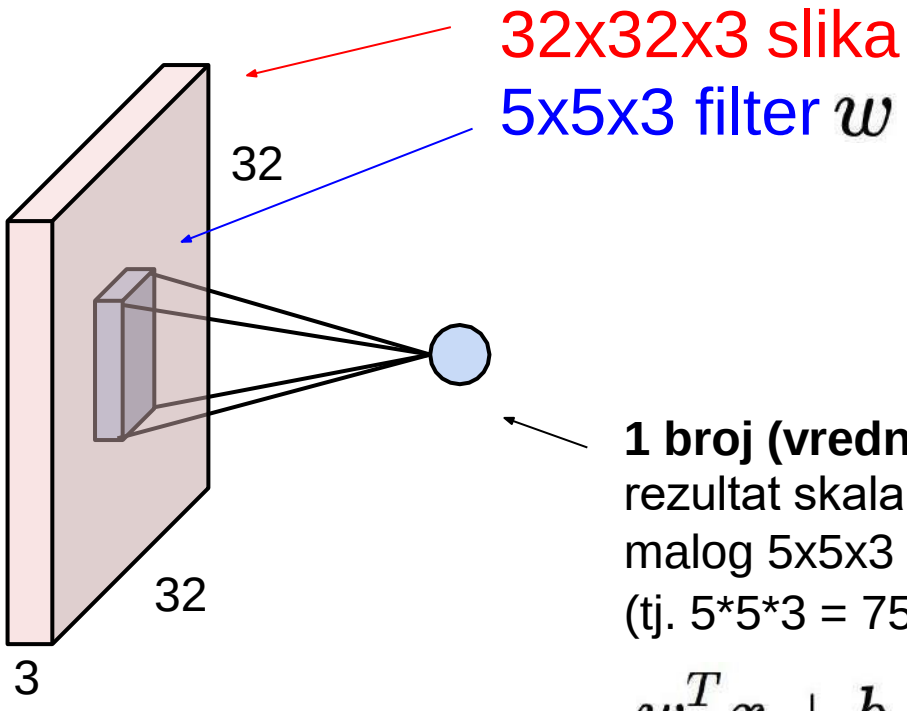


5x5x3 filter



Vršimo Konvoluciju filtera sa slikom tj. “vučemo” filter prostorno po slici i računamo skalarni proizvod

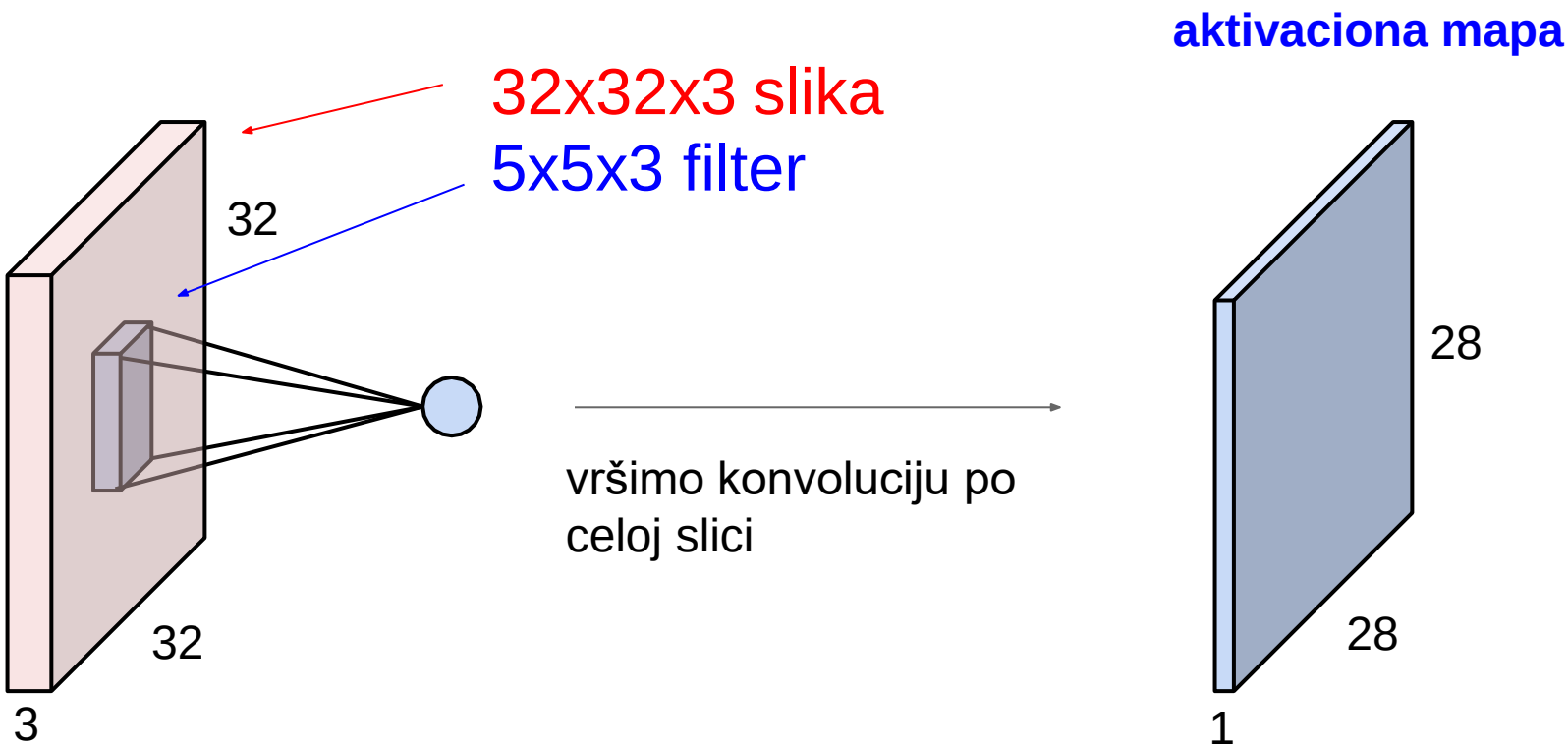
Konvolutivni Sloj



1 broj (vrednost):
rezultat skalarnog proizvoda između filtera i malog 5x5x3 dela slike
(tj. 5*5*3 = 75-dim. skalarni proizvod + bias)

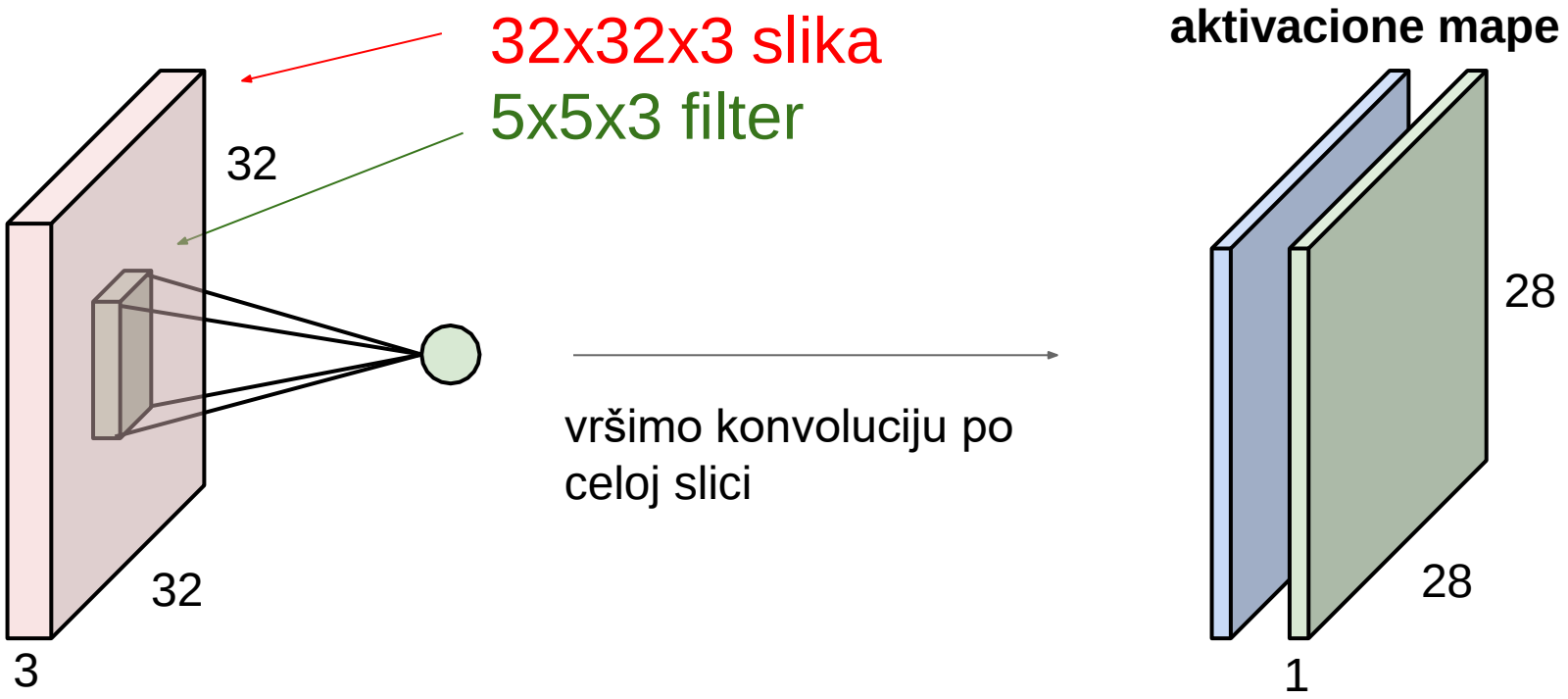
$$w^T x + b$$

Konvolutivni Sloj

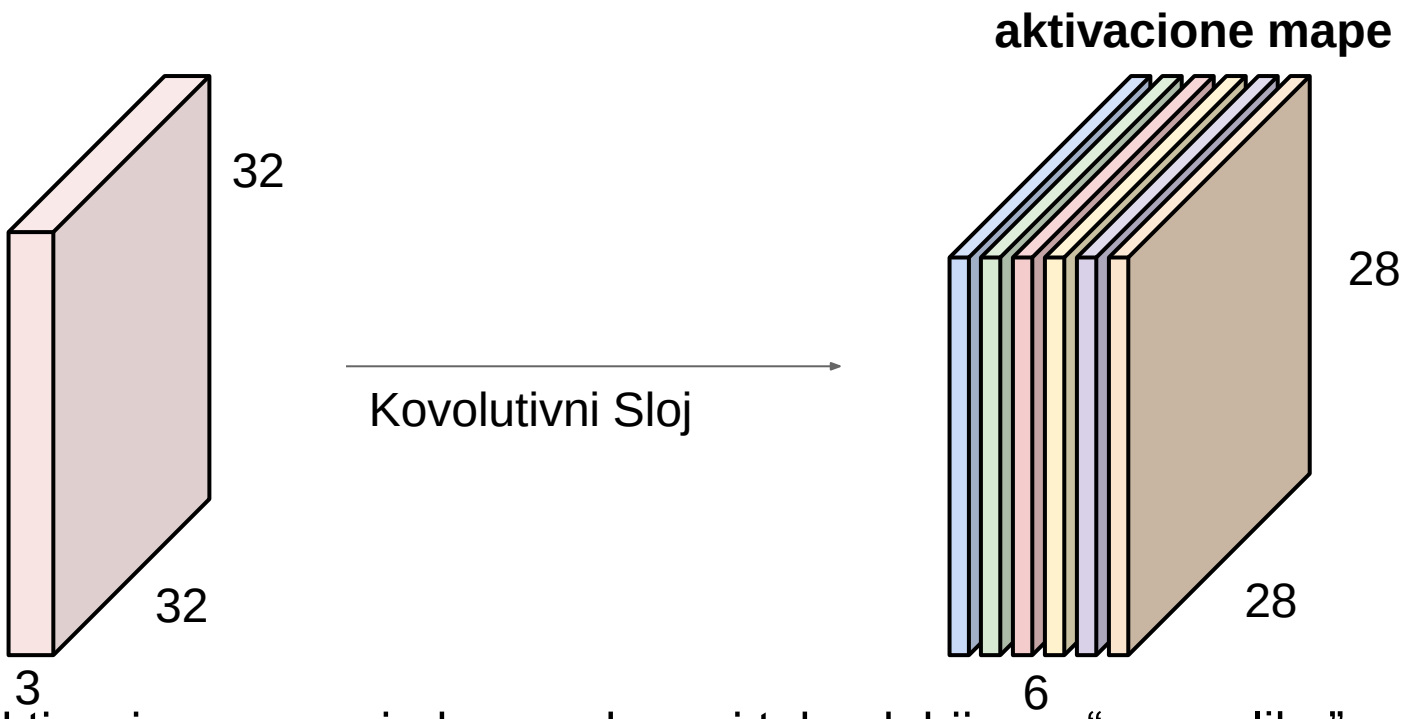


Konvolutivni Sloj

recimo da imamo, zeleni filter



Na primer, ako imamo 6 5x5 filtera, dobićemo 6 odvojenih aktivacionih mapa:

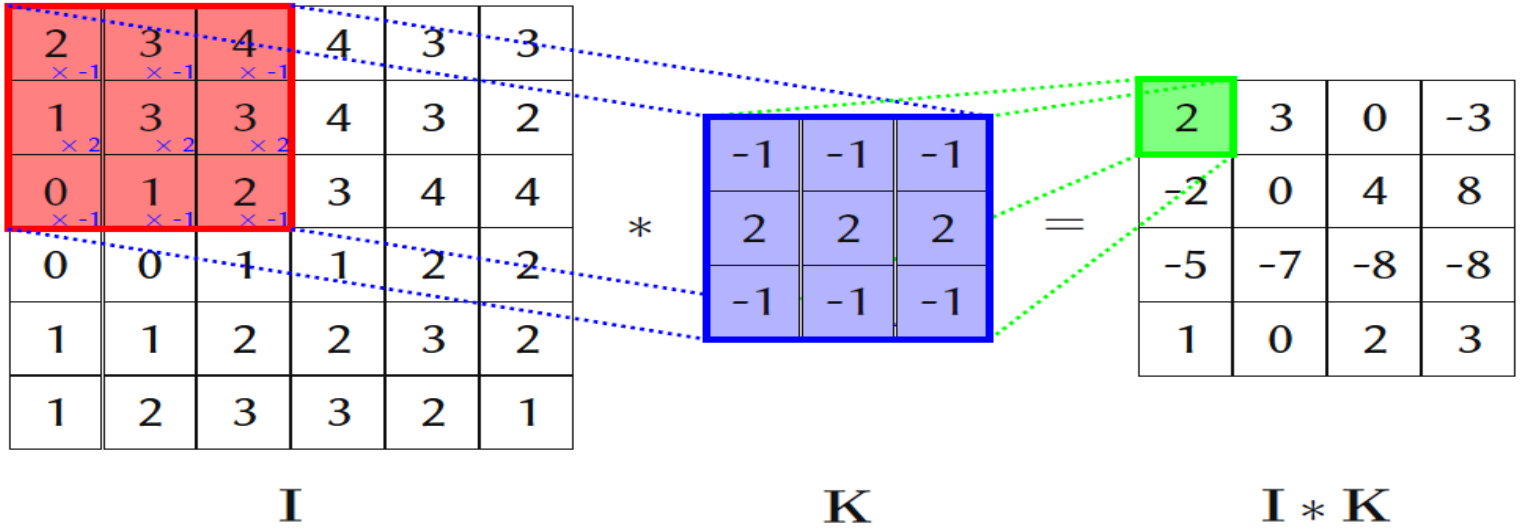


Slažemo aktivacione mape jednu na drugu i tako dobijamo “novu sliku” veličine 28x28x6!

Primer konvolucije

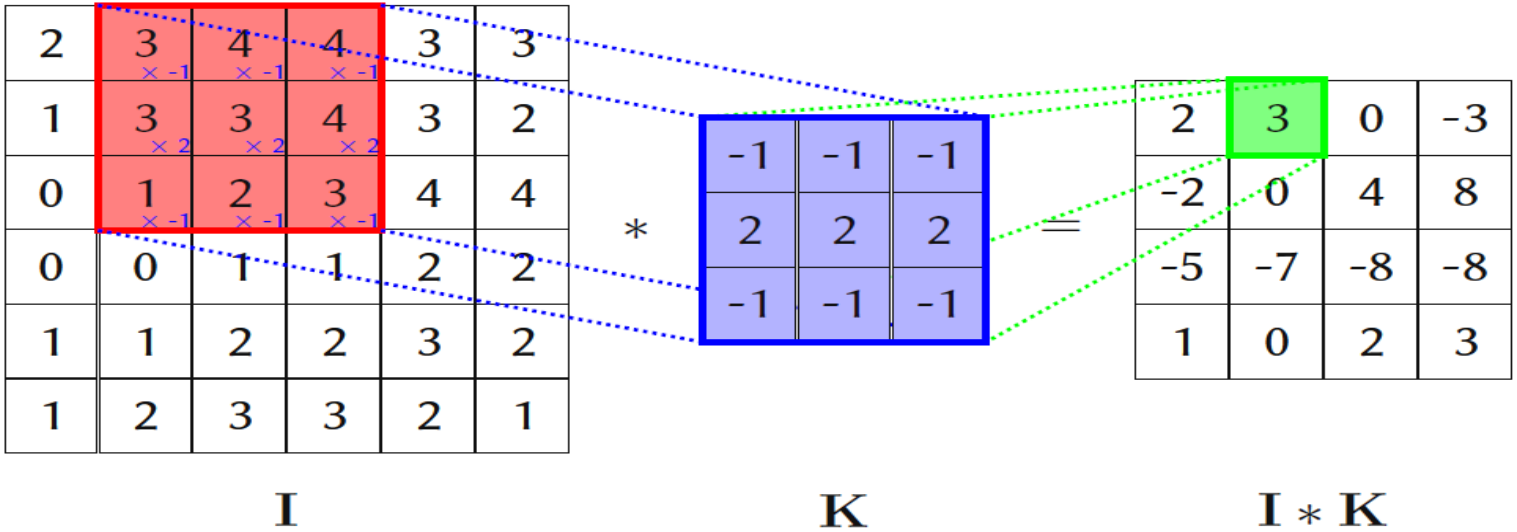
Narednih nekoliko slajdova zasnovano je na
http://www.csnedelja.mg.edu.rs/static/resources/v3.0/sre1_neuralne_pv.pdf

Primer konvolucije

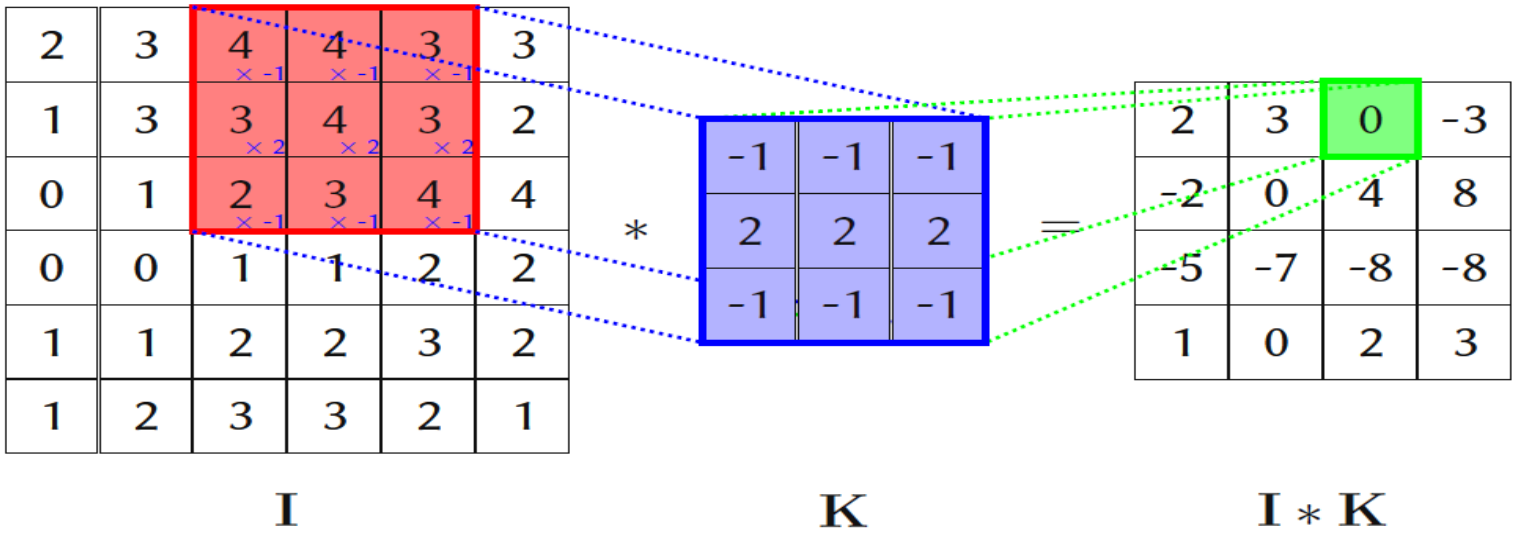


<u>Napomena</u>: ovaj konretan primer ne sadrži bias, ali je dovoljan za razumevanje načina na koji funkcionišu filteri kod konvolucija.

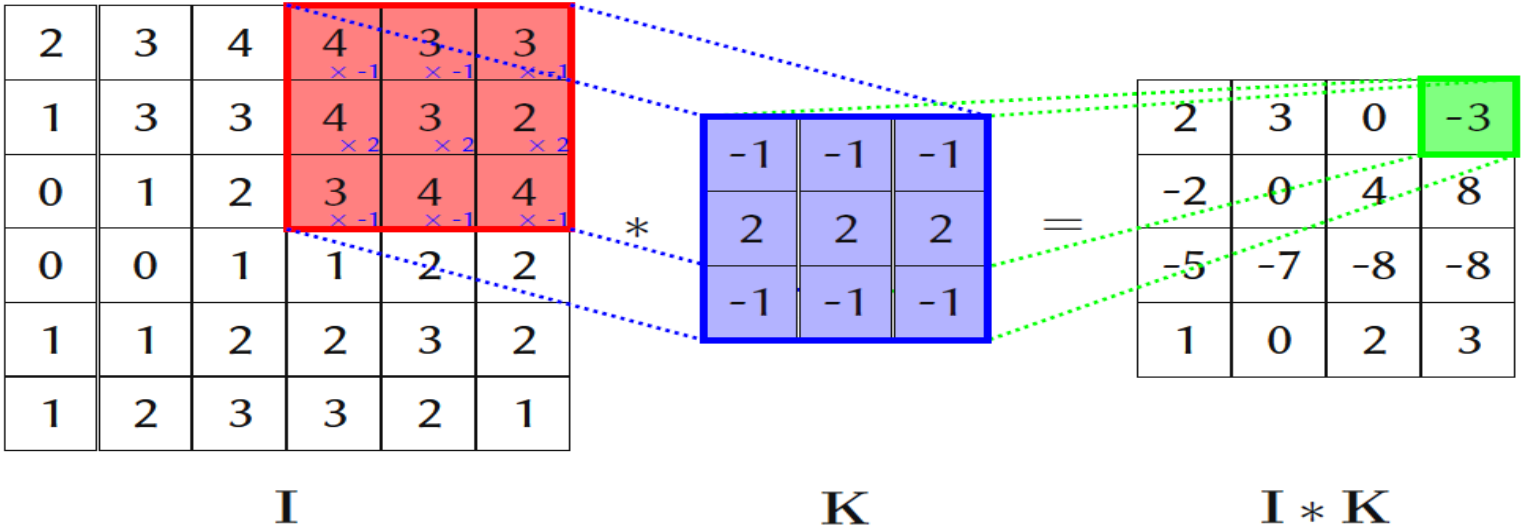
Primer konvolucije



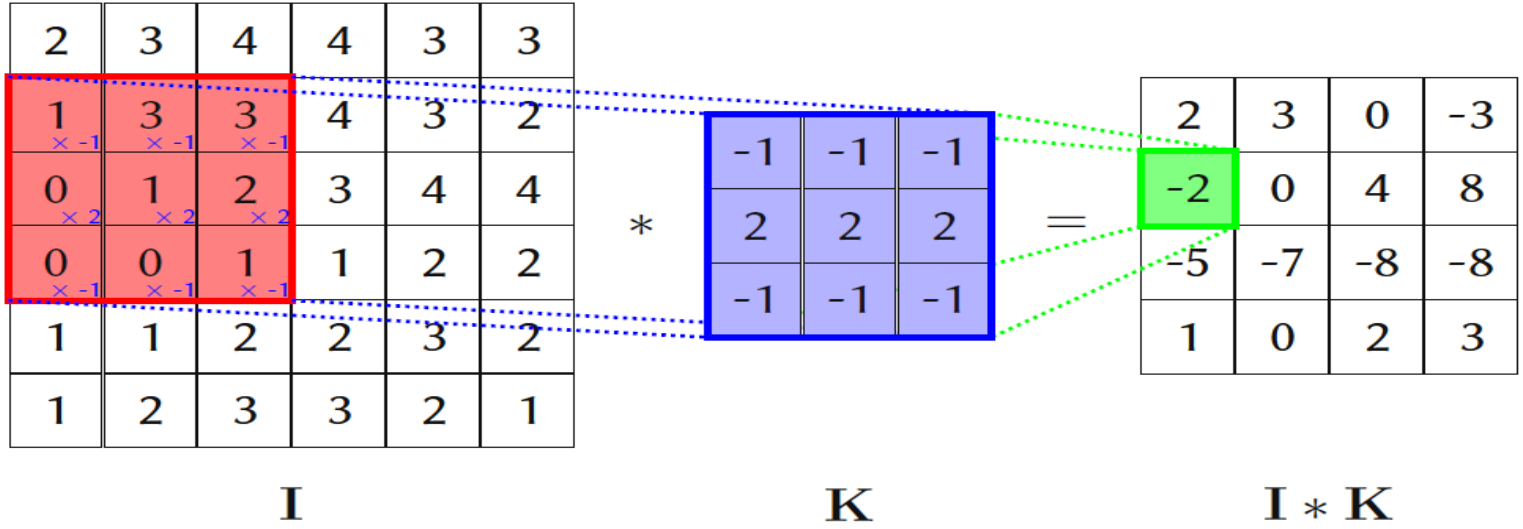
Primer konvolucije



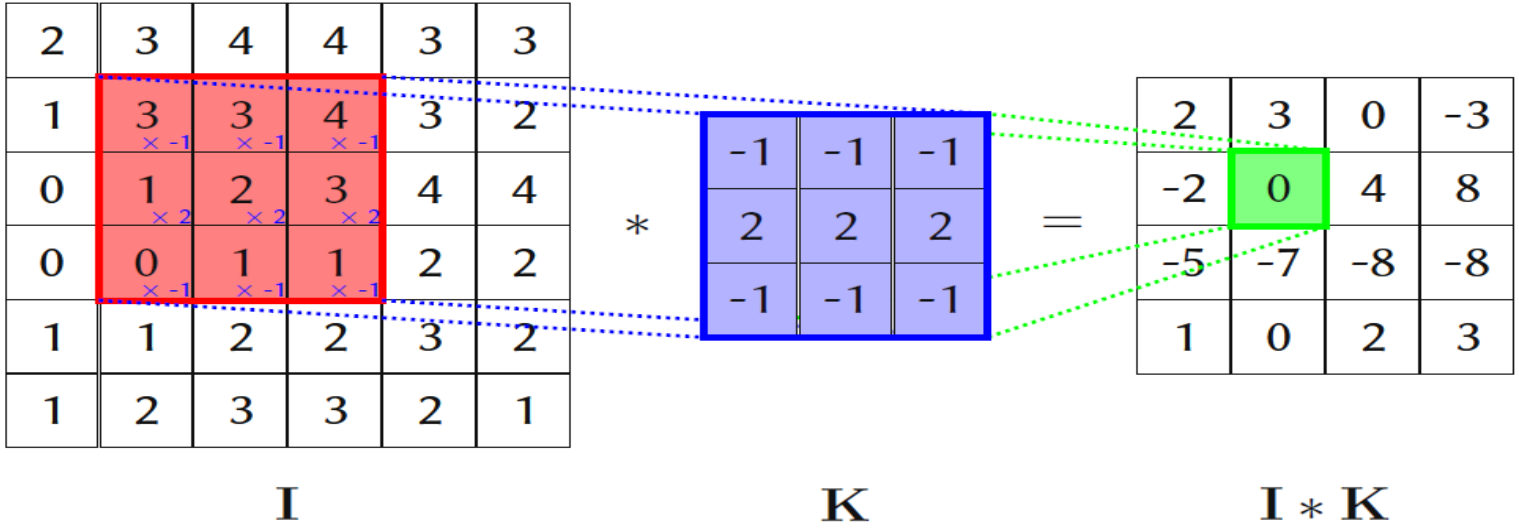
Primer konvolucije



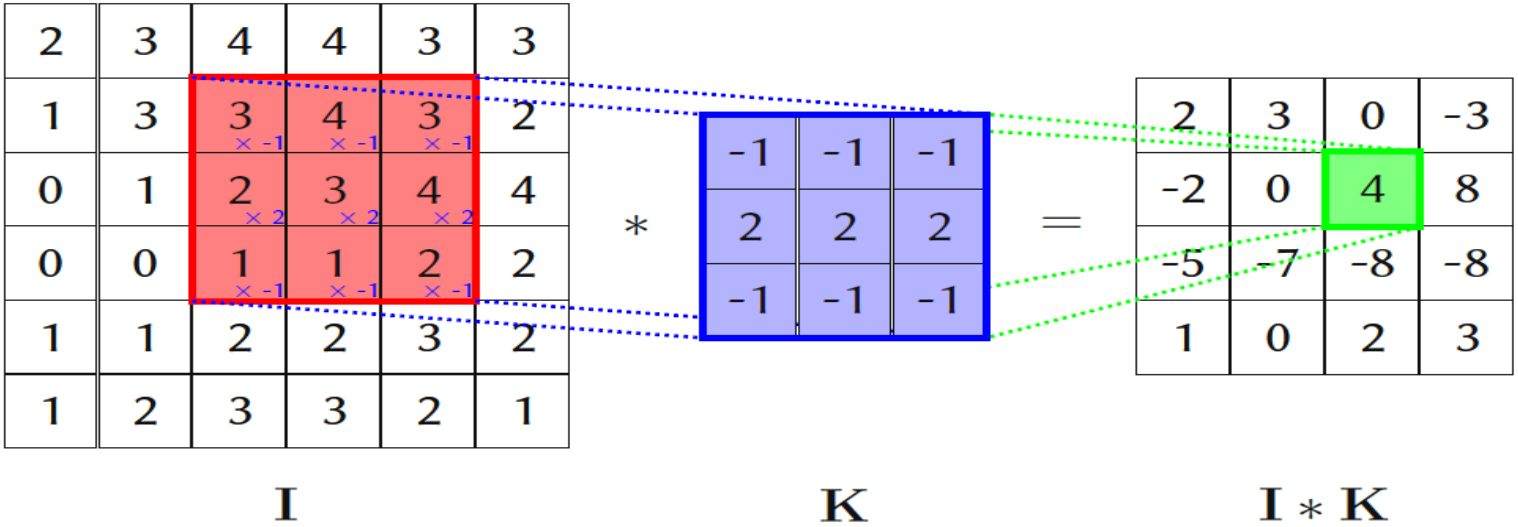
Primer konvolucije



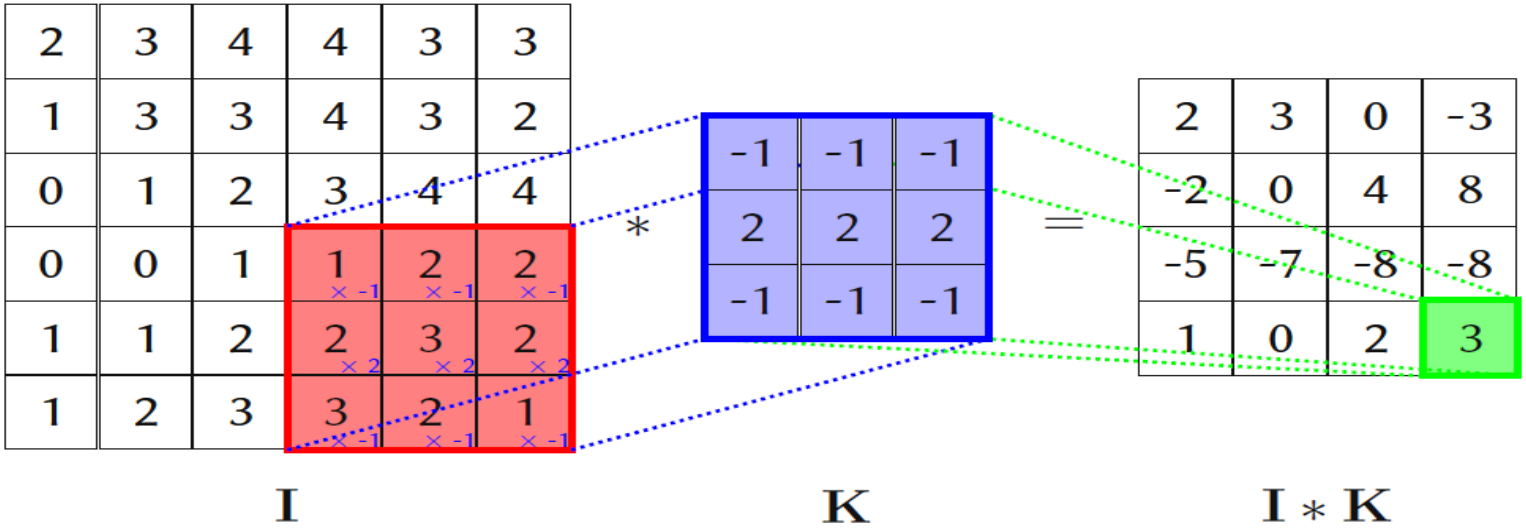
Primer konvolucije



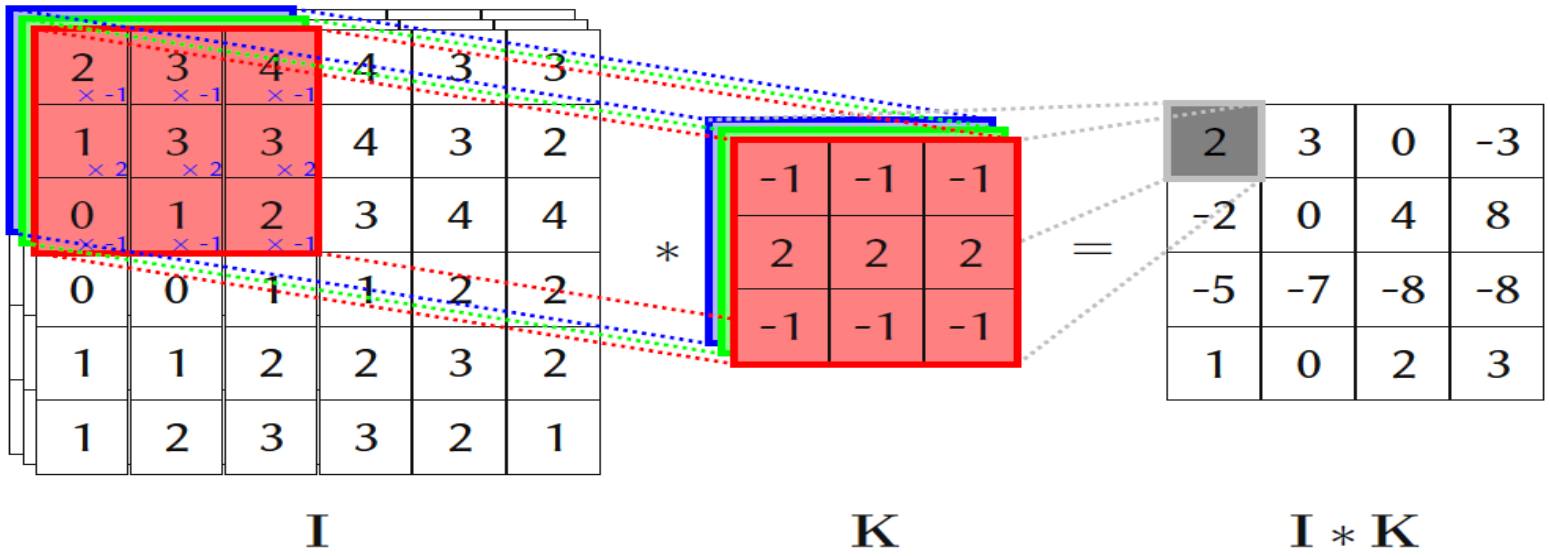
Primer konvolucije



Primer konvolucije



Konvolucija sa *bojom/kanalima*



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$$308 + -498 + 164 + 1 = -25$$

Bias = 1

Output

-25				...
				...
				...
				...
...	...	...	...	...

Konvolucija

Šta je ustvari konvolucija? To je filter koji nam služi za ekstrakciju karakteristika slike.

Konvolucija

Na primer na ovoj slici primenom kovolucije dobijamo ivice.

Konvolucija

Zajednica ljudi koji se bave Kompjuterskom Vizijom je ranije morala ručno da razvija konvolucije

Konvolucija

Konvolutivne mreže same uče konvolucije! To je poenta Deep Learning!

Primer konvolucije sa za detekciju izvica

Input image

Convolution Kernel

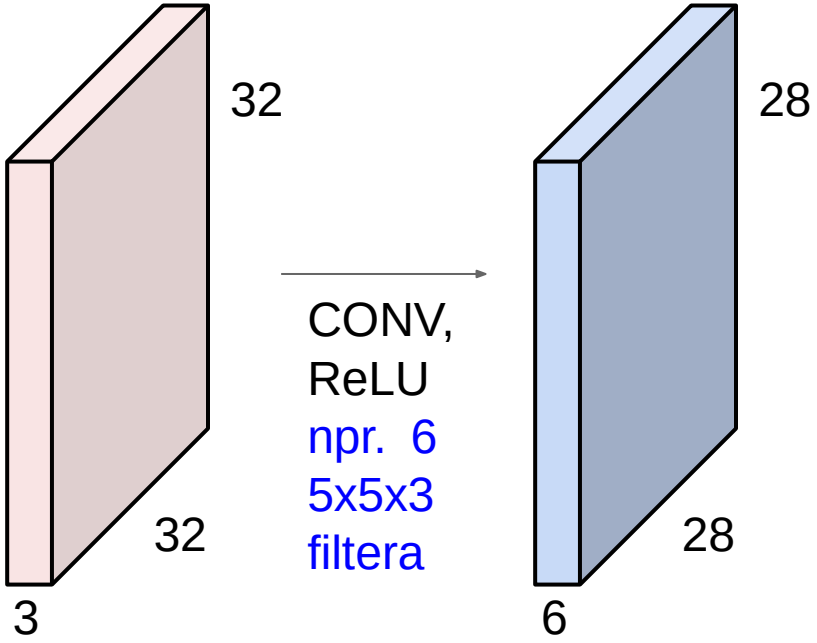
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map

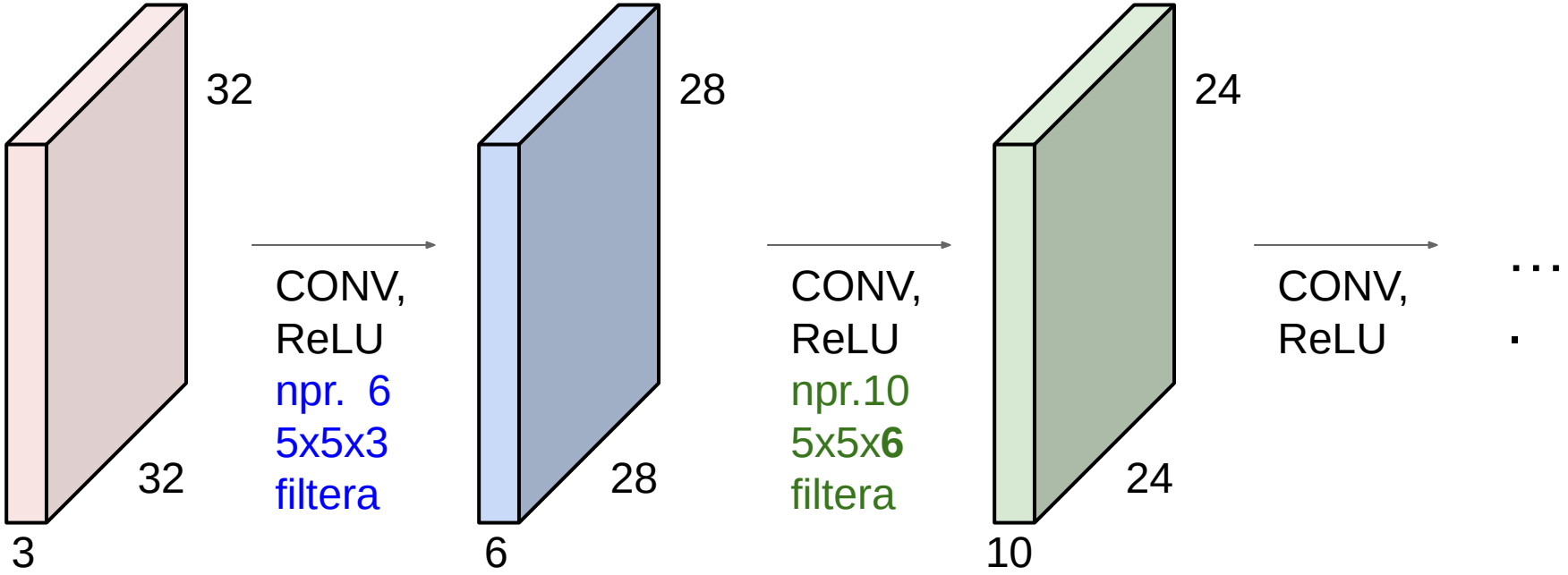
Vrednosti matrice je ručno kreirao ekspert.
Dok ih kovolutivna mreža sama uči!

slika preuzeta sa https://pgaleone.eu/neural-networks/2016/11/24/convolutional-autoencoders/

Konvolutivna mreža je niz Konvolutivnih Slojeva, između kojih su aktivacione fukcije

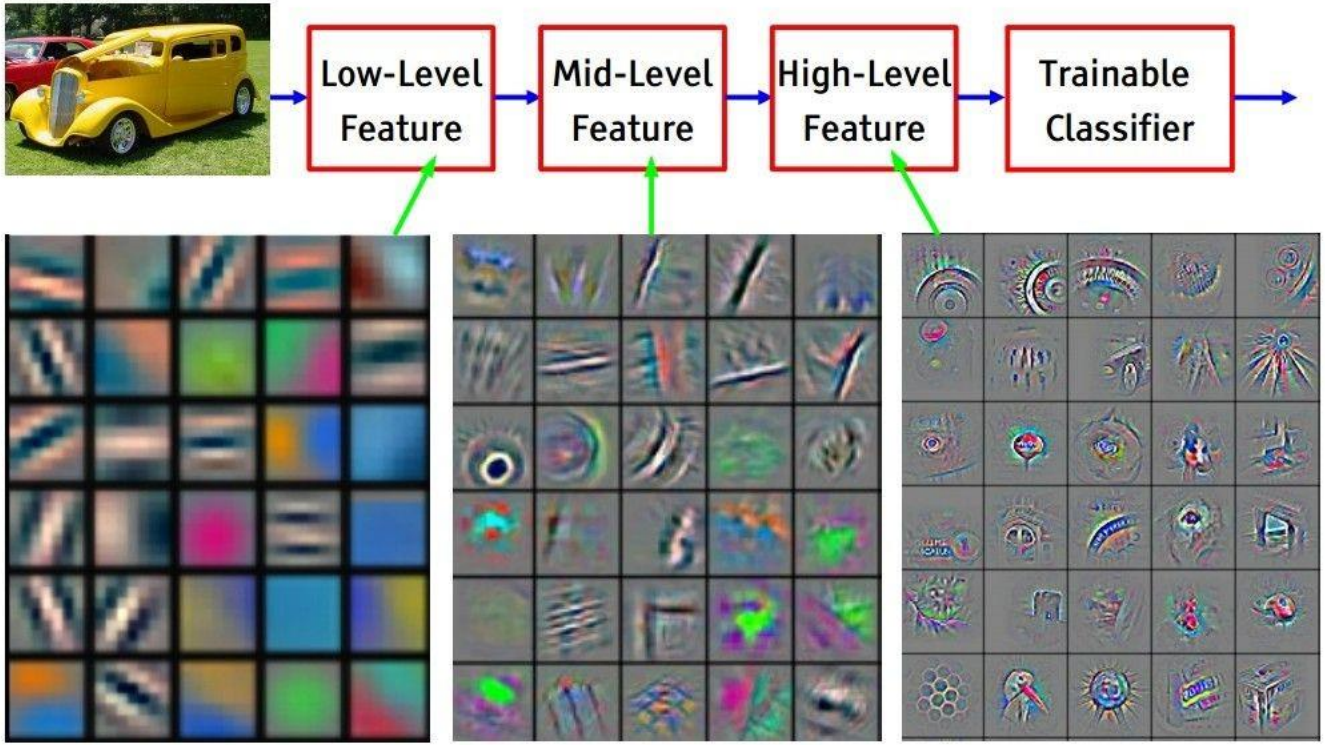


Konvolutivna mreža je niz Konvolutivnih Slojeva, između kojih su aktivacione fukcije



Kako izgledaju slojevi

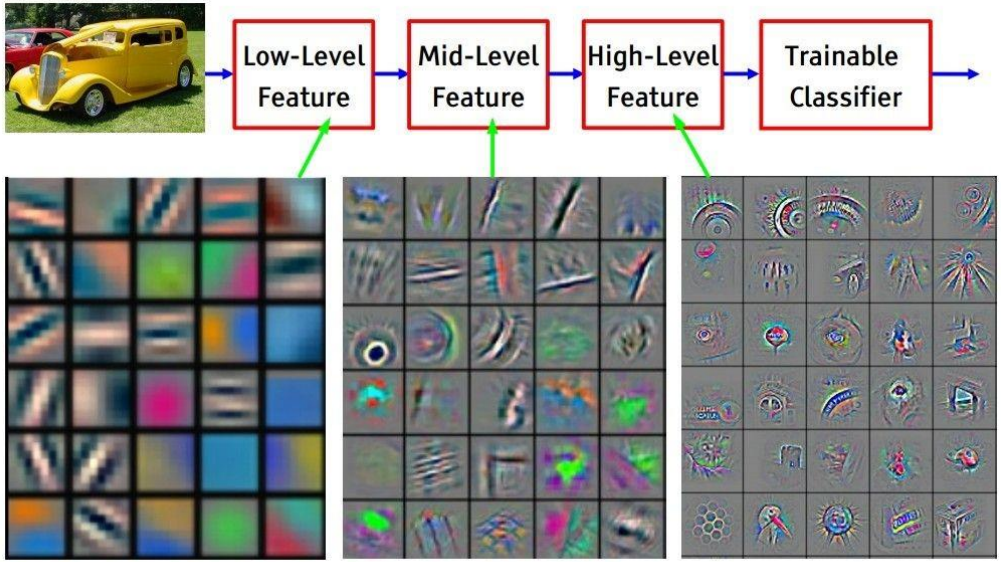
[From recent Yann LeCun slides]



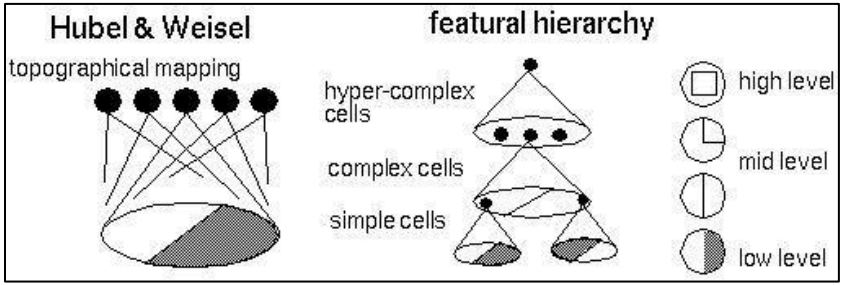
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

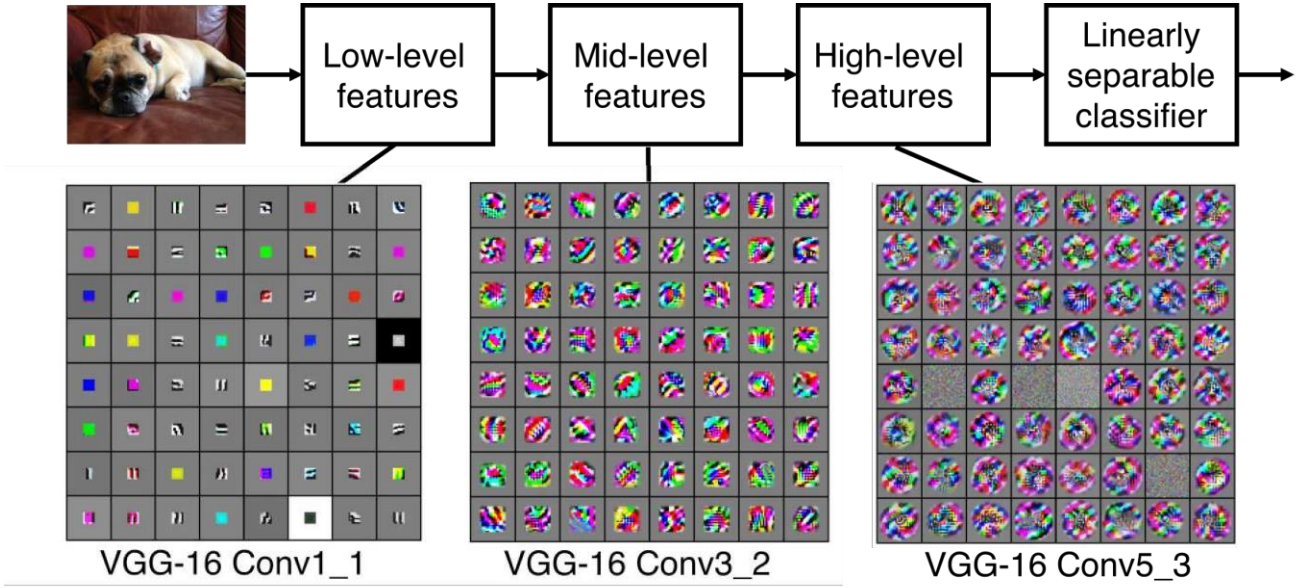
Kako izgledaju slojevi

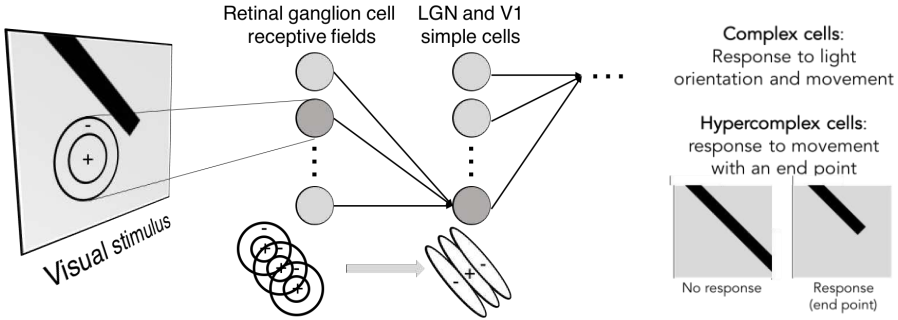
[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

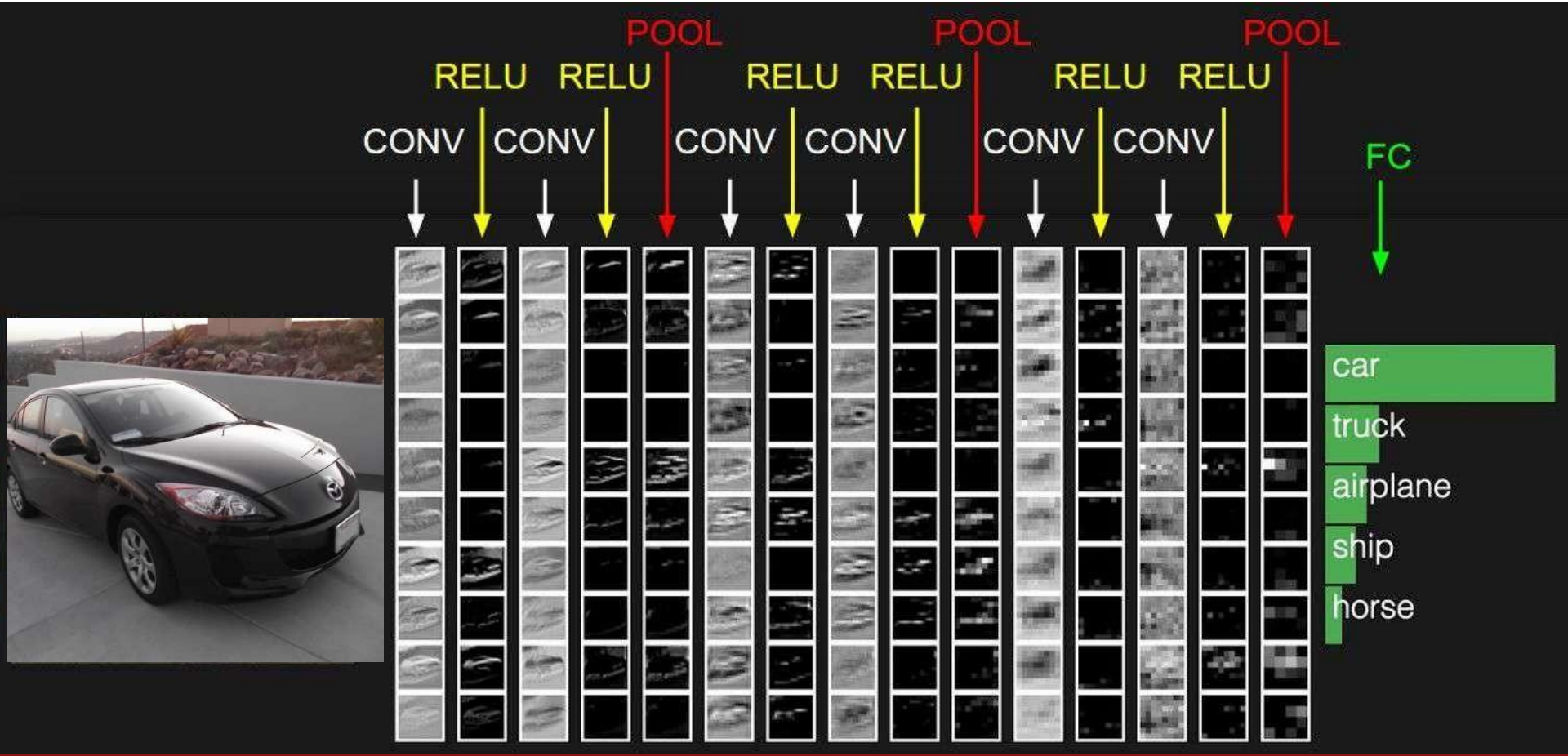



Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3

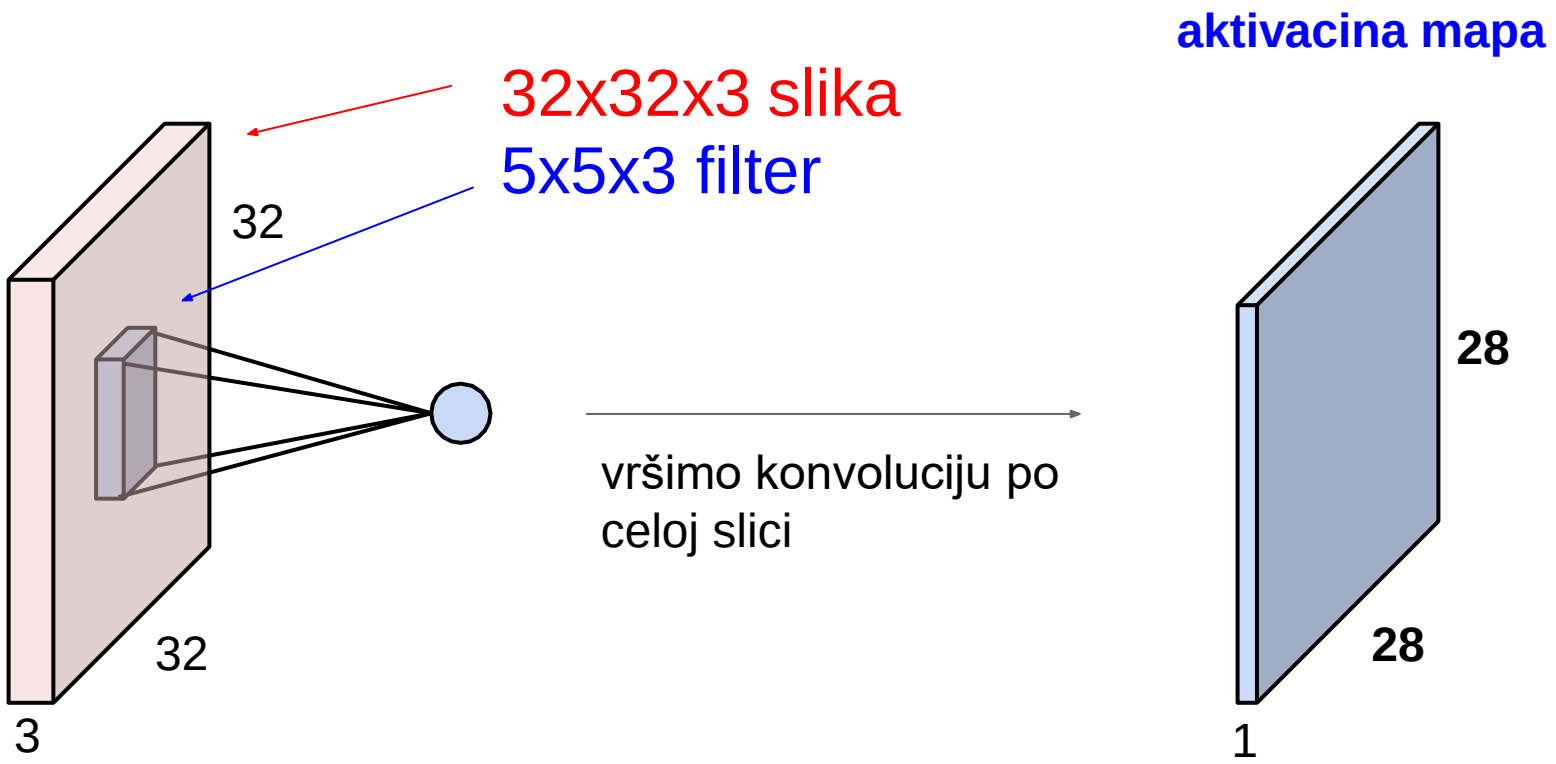

Retinal ganglion cell receptive fields
LGN and V1 simple cells
Visual stimulus
Complex cells:
Response to light orientation and movement
Hypercomplex cells:
response to movement with an end point
No response
Response (end point)

primer 5x5 filteri
(32 ukupno)

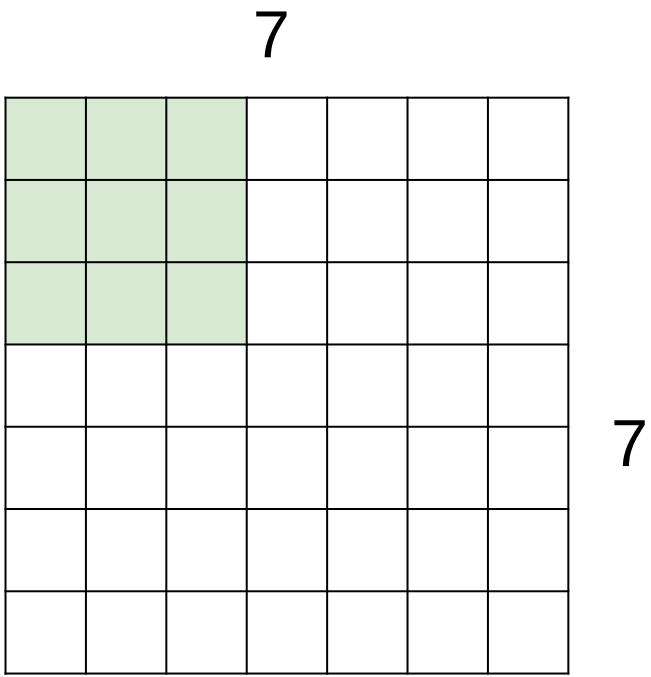
Kako tipično izgleda Konvolutivna Mreža



Detaljnije o dimenzijama:

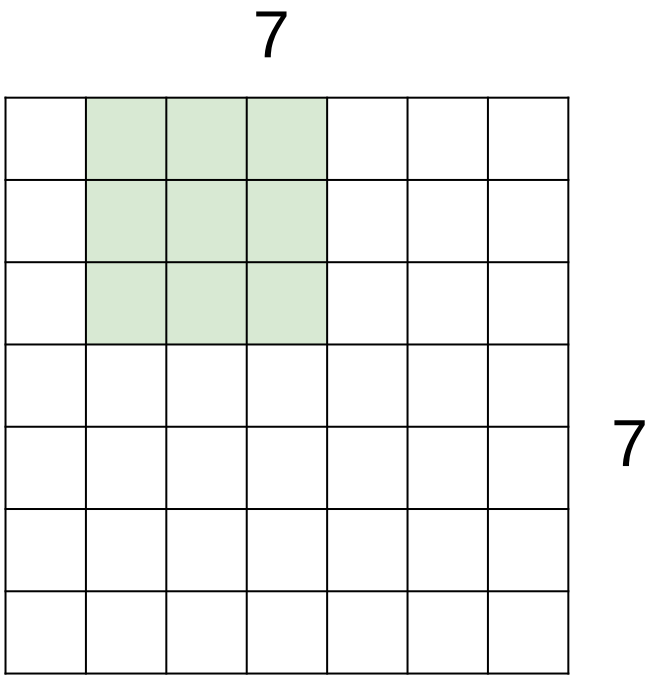


Detaljnije o dimenzijama:



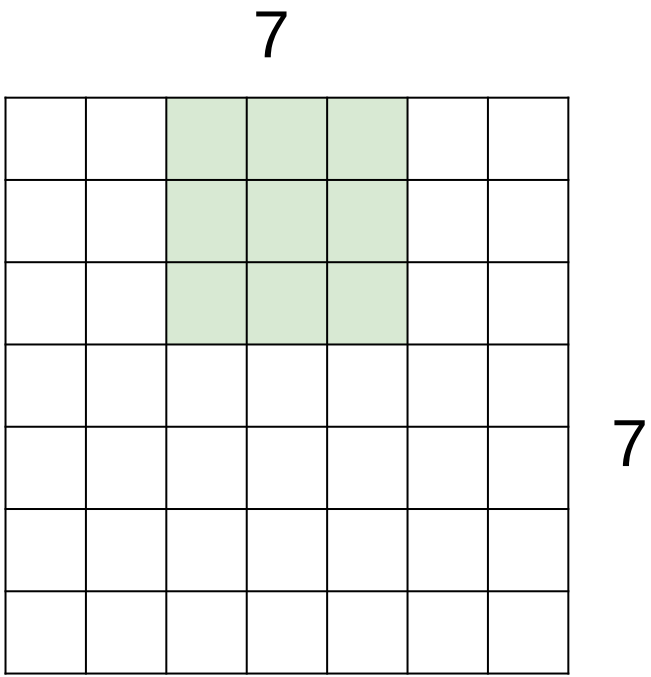
7x7 ulaz
3x3 filter

Detaljnije o dimenzijama:



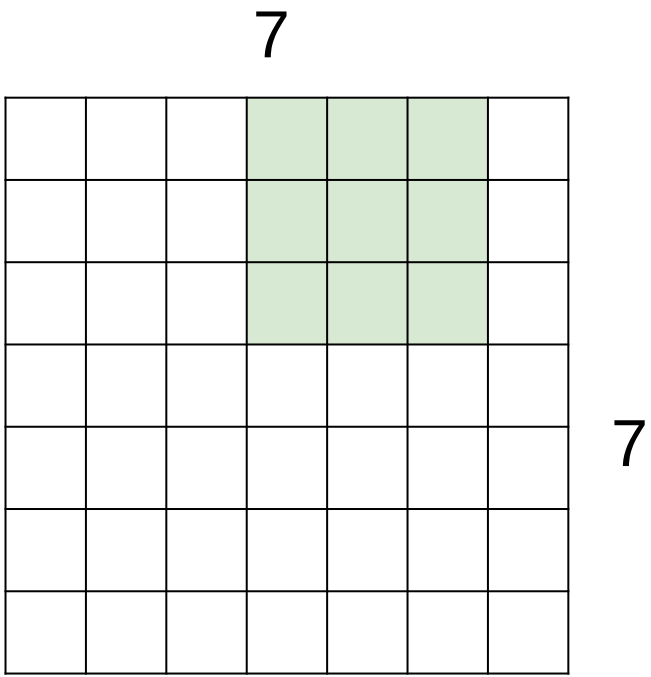
7x7 ulaz
3x3 filter

Detaljnije o dimenzijama:



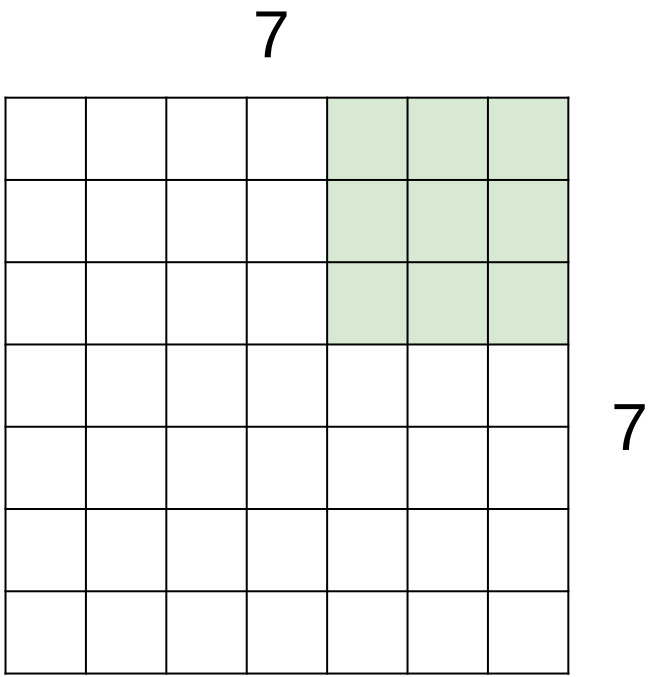
7x7 ulaz
3x3 filter

Detaljnije o dimenzijama:



7x7 ulaz
3x3 filter

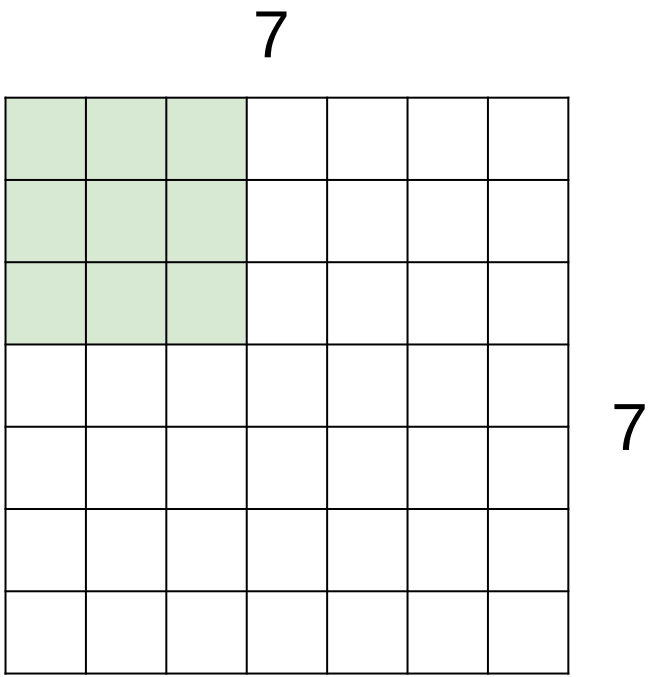
Detaljnije o dimenzijama:



7x7 ulaz
3x3 filter

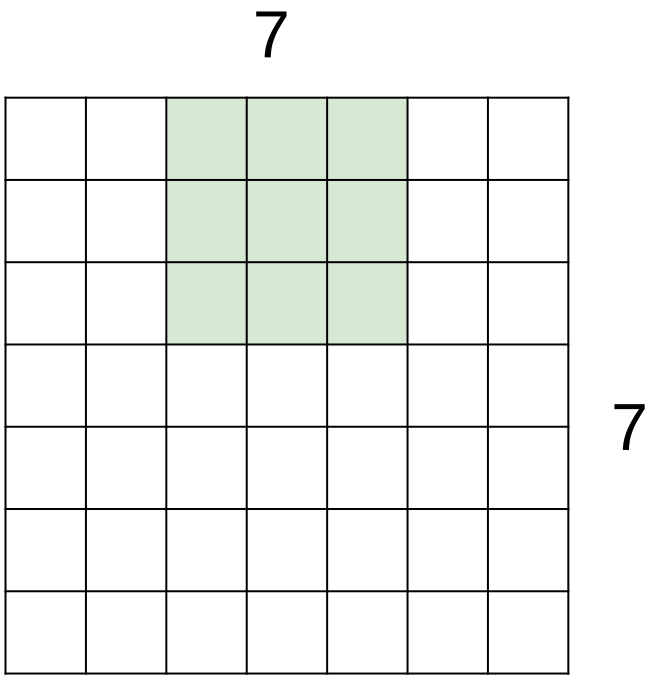
=> 5x5 izlaz

Detaljnije o dimenzijama:



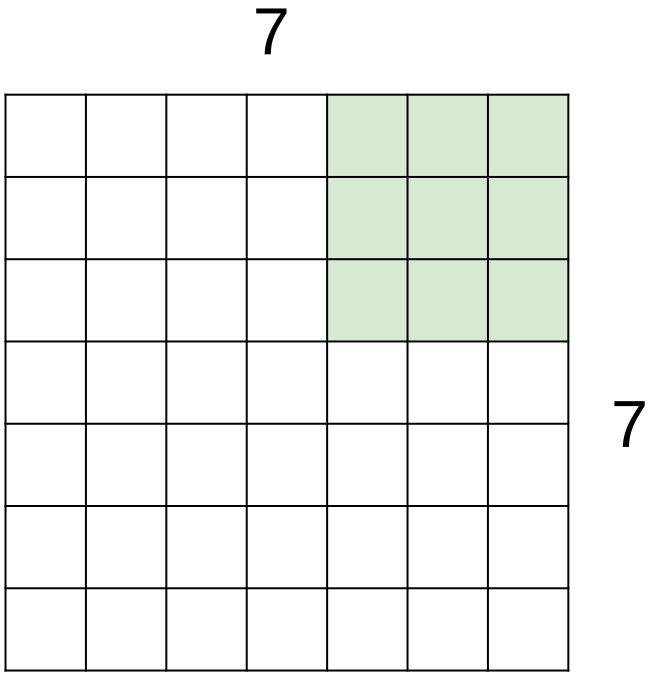
7x7 ulza
3x3 filter
primenjen sa **korakom (*stride*) 2**

Detaljnije o dimenzijama:



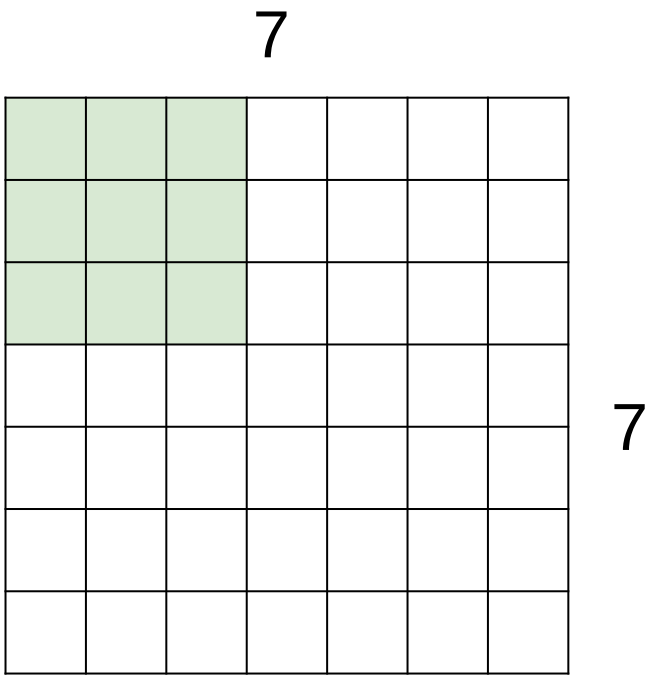
7x7 ulza
3x3 filter
primenjen sa **korakom (*stride*) 2**

Detaljnije o dimenzijama:



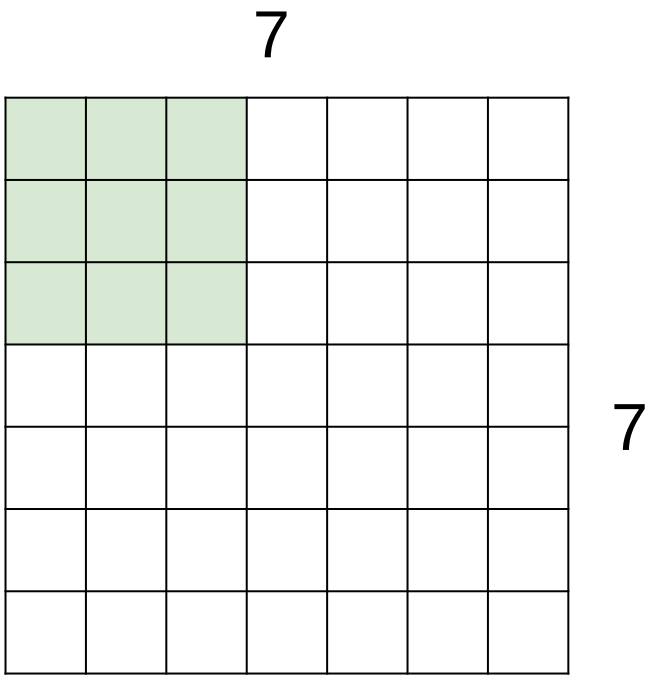
7x7 ulza
3x3 filter
primenjen sa **korakom (*stride*) 2**
=> 3x3 izlaz!

Detaljnije o dimenzijama:



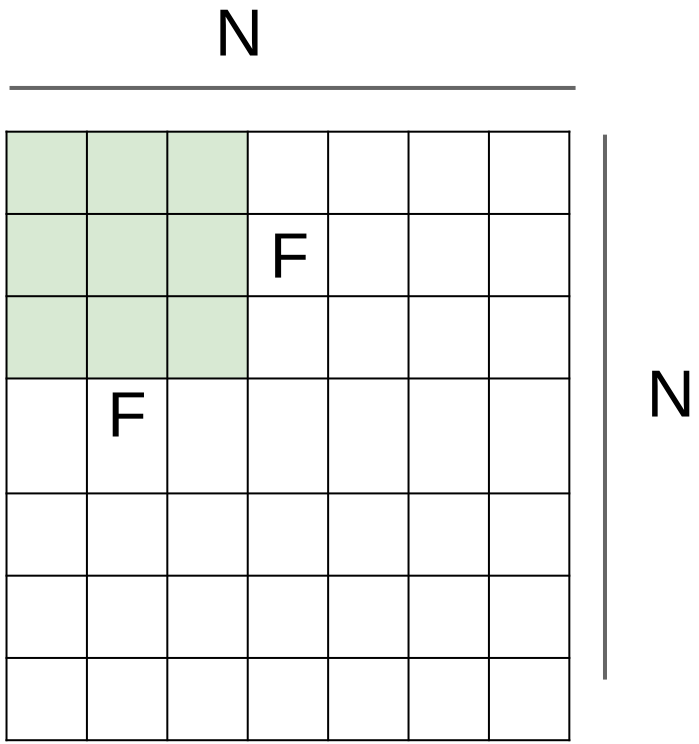
7x7 ulaz
3x3 filter
primenjen sa **korakom (*stride*) 3**
Da li je primena moguća?

Detaljnije o dimenzijama:



7x7 ulaz
3x3 filter
primenjen sa **korakom (*stride*) 3**
Da li je primena moguća?

Ne, filter se ne uklapa!
ne možemo da primenimo 3x3 filter na 7x7 ulaz sa korakom 3.



Dimenzionalnost izlaza:
((N – F) / korak) + 1

npr. N = 7, F = 3:
korak 1 => (7 - 3)/1 + 1 = 5
korak 2 => (7 - 3)/2 + 1 = 3
korak 3 => (7 - 3)/3 + 1 = 2.33
– ne dobijamo ceo broj

U praksi: Dodajemo nule u okvir slike

0	0	0	0	0	0			
0								
0								
0								
0								

npr. ulaz 7x7
3x3 filter, primenjen sa **korakom 1**
dodajemo okvir od 1 piksela => koje je su dimenzije izlaza?

(podsetite se:)
(N - F) / korak + 1

U praksi: Dodajemo nule u okvir slike

0	0	0	0	0	0			
0								
0								
0								
0								

npr. ulaz 7x7
3x3 filter, primenjen sa **korakom 1**
dodajemo okvir od 1 piksela => koje je su dimenzije izlaza?

Izlaz je 7X7! (7+2-3)/1 + 1=7

podsetite se:
(N - F) / korak + 1

U praksi: Dodajemo nule u okvir slike

0	0	0	0	0	0			
0								
0								
0								
0								

npr. ulaz 7x7

3x3 filter, primenjen sa **korakom 1**

dodajemo okvir od 1 piksela => koje je su dimenzije izlaza?

Izlaz je 7X7! (7+2-3)/1 + 1=7

tipično se koriste CONV slojevi sa korakom 1, filteri veličine FxF, dodavanje 0 u okvir dimenzije (F-1)/2.

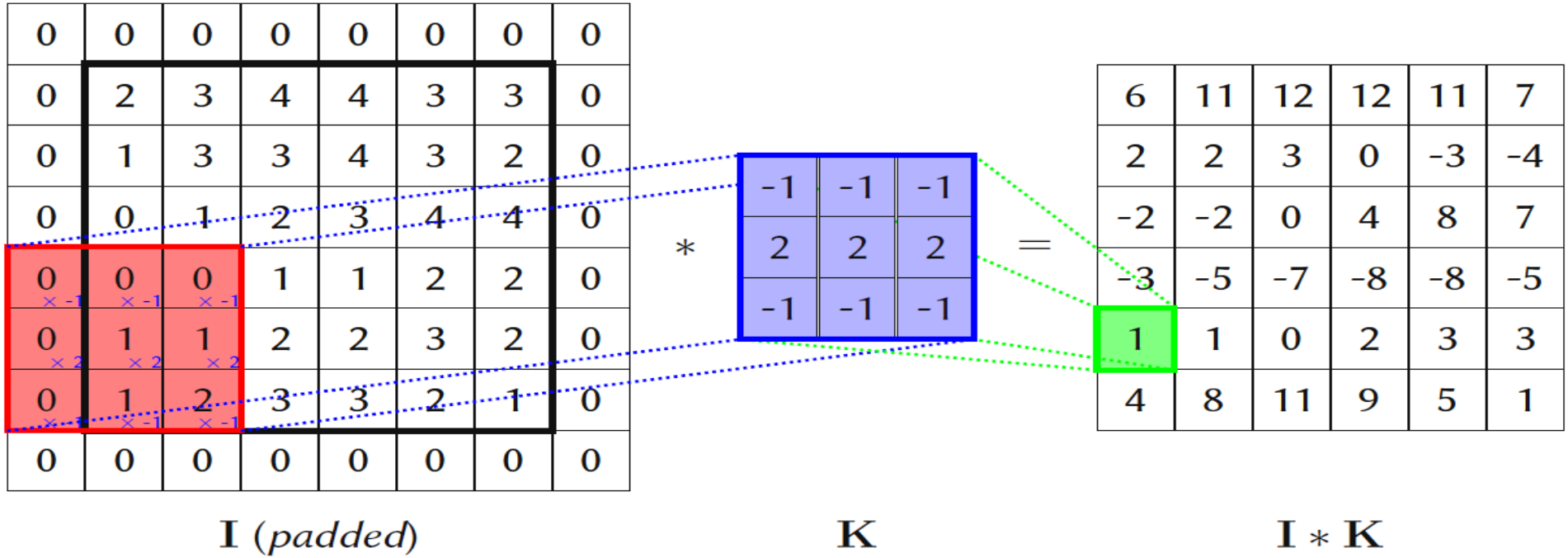
npr.

F = 3 => dodajemo okvir dim. 1

F = 5 => dodajemo okvir dim. 2

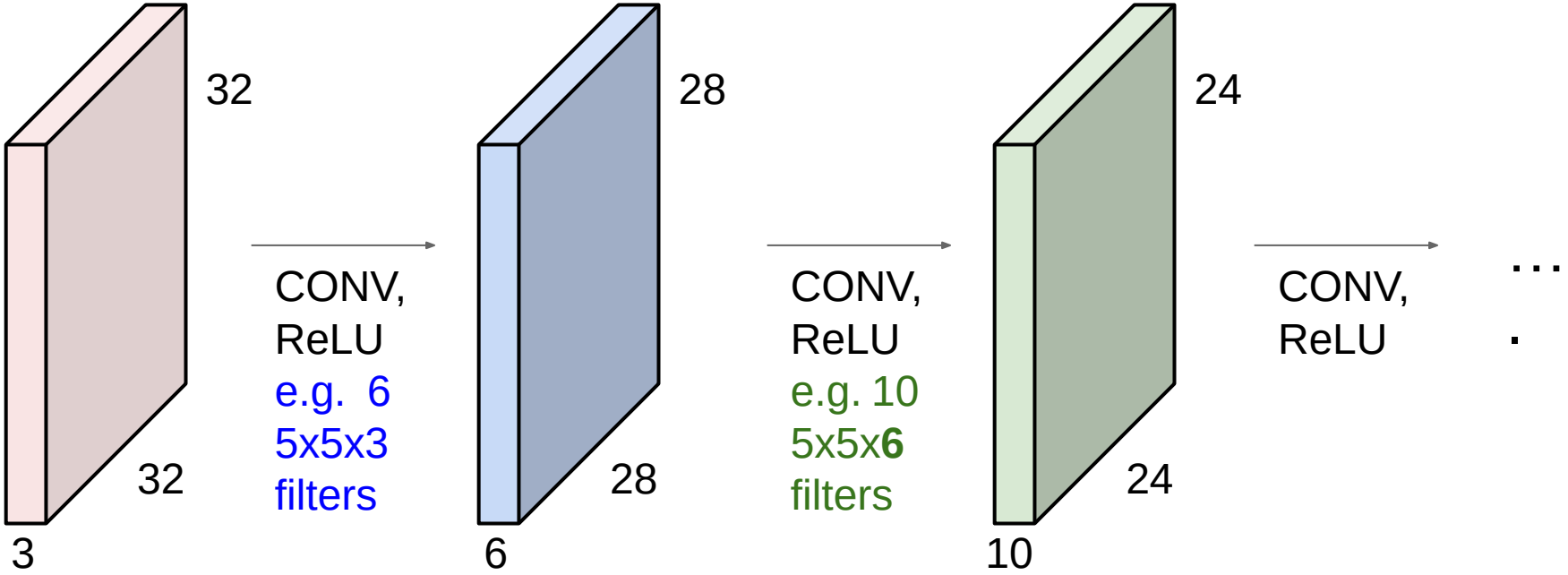
F = 7 => dodajemo okvir dim. 3

Primer konvolucije (sa *padding*-om)

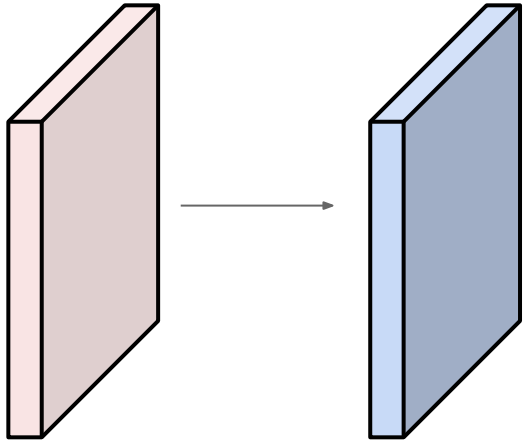


Prisetimo se…

Npr. ulaz 32x32 na kome se u više slojeva primenjuje filter 5x5 se smanjuje po veličini! (32 -> 28 -> 24 ...). Prebrzo smanjivanje nije dobro, gubimo na informacijama, dobijamo loše rezultate. Zato koristimo padding.



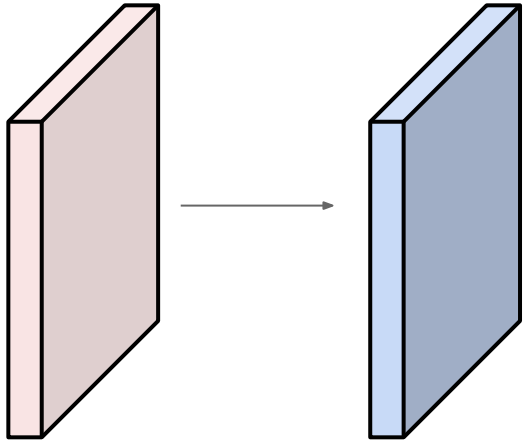
Primer:



Ulaz: **32x32x3**
10 5x5x3 filtera sa korakom 1,
i dodavanjem okvira 0 dim. 2

Koja je veličina izlaza: ?

Primer:



Ulaz: **32x32x3**

10 5x5x3 filtera sa korakom 1,

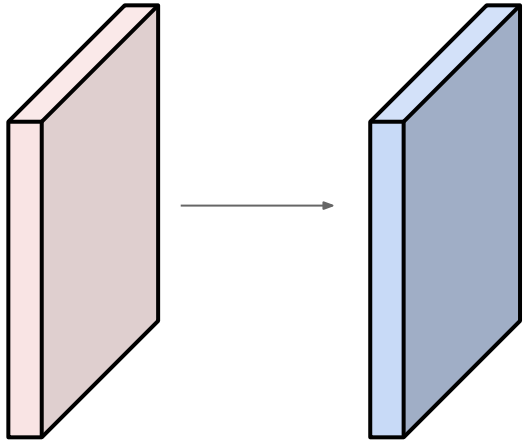
i dodavanjem okvira 0 dim. 2

Veličina izlaza:

(32+2*2-5)/1+1 = 32 što daje,

32x32x10

Primer:



Ulaz: **32x32x3**
10 5x5x3 filter sa korakom 1,
i dodavanjem okvira 0 dim. 2

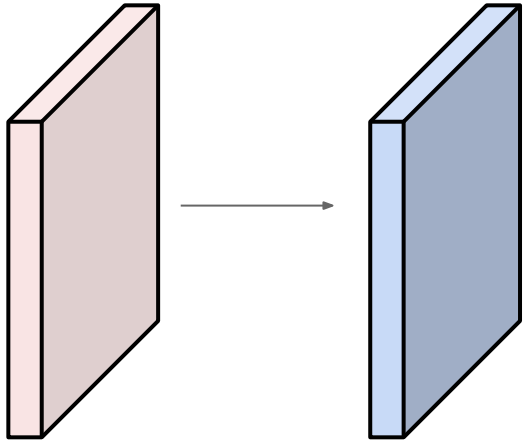
Koji je broj parametara (težina i biasa)
za obučavanje u ovom sloju?

Primer:

Ulaz: **32x32x3**

10 5x5x3 filter sa korakom 1,

i dodavanjem okvira 0 dim. 2



Koji je broj parametara (težina i biasa) za obučavanje u ovom sloju?

svaki filter ima 5*5*3 + 1 = 76 parametara (+1 za bias)

=> 76*10 = **760**

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Tipična podešavanja:

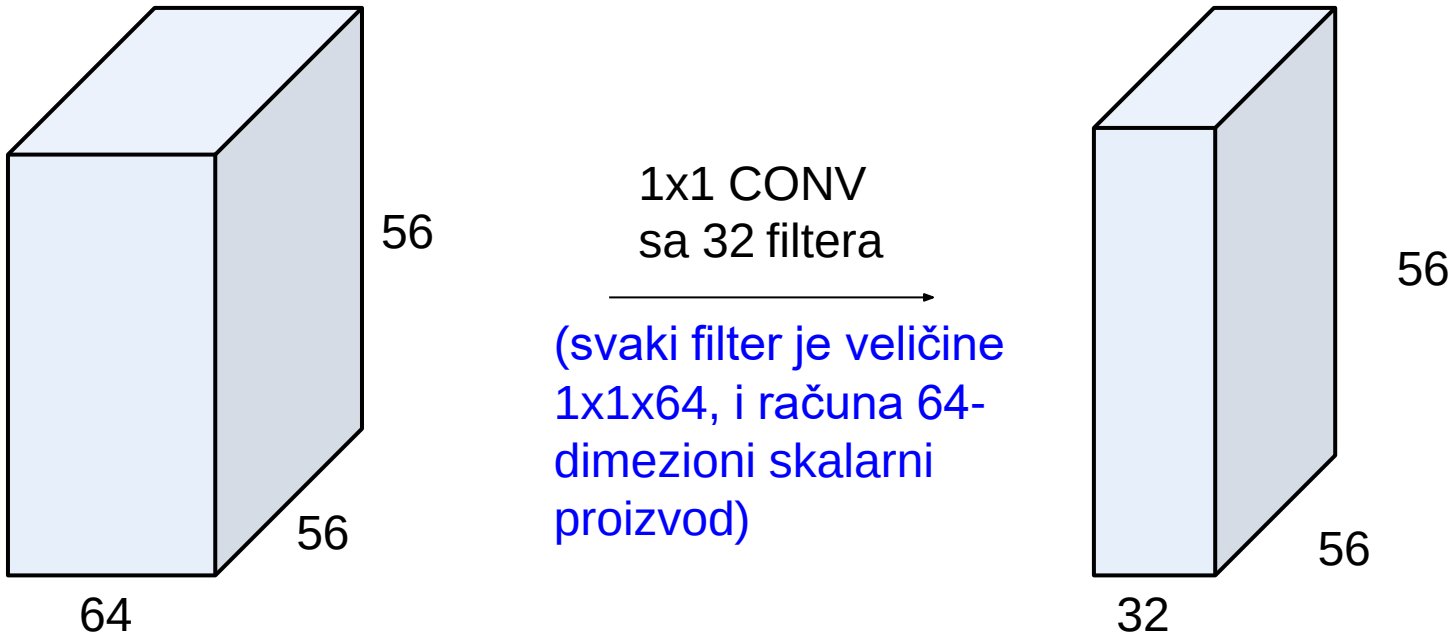
K = (stepeni 2, npr. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (šta god odgovara)
- F = 1, S = 1, P = 0

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

(napomena, 1x1 konvolutivni slojevi imaju smisla)



Primer: CONV sloj u okruženju *Torch*

SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

- `nInputPlane` : The number of expected input planes in the image given into `forward()` .
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `kH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1` .
- `dH` : The step of the convolution in the height dimension. Default is `1` .
- `padW` : The additional zeros added per width to the input planes. Default is `0` , a good number is `(kW-1)/2` .
- `padH` : The additional zeros added per height to the input planes. Default is `padW` , a good number is `(kH-1)/2` .

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane x height x width` , the output image size will be `nOutputPlane x oheight x owidth` where

```
owidth  = floor((width  + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

Torch is licensed under BSD 3-clause.

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

Primer: CONV sloj u okruženju *Caffe*

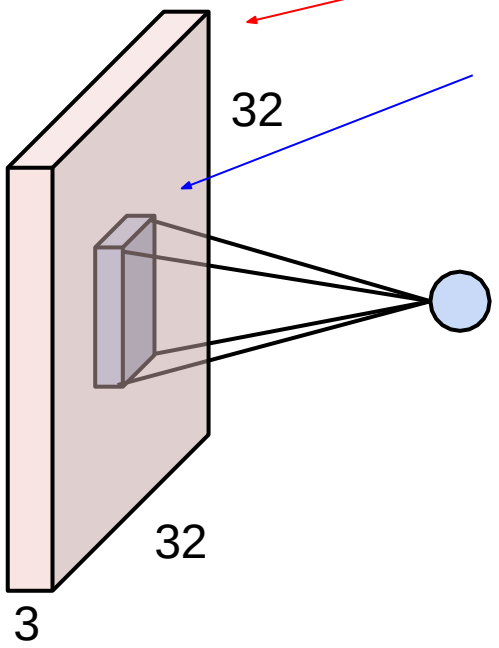
```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

Caffe is licensed under BSD 2-Clause.

Mozak/neuron pogled na CONV Sloj

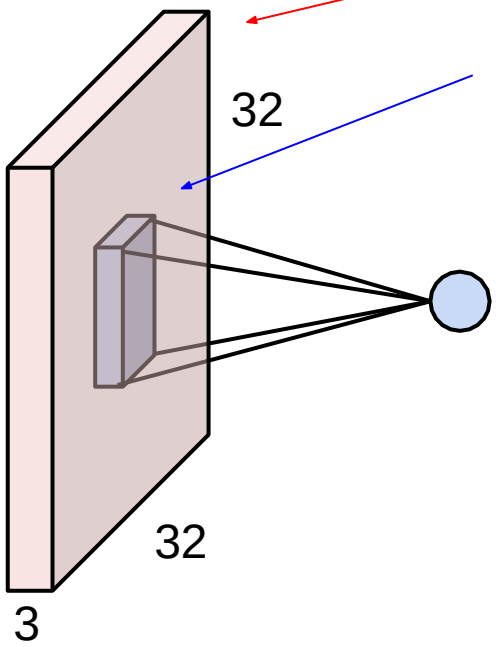


32x32x3 slika

5x5x3 filter

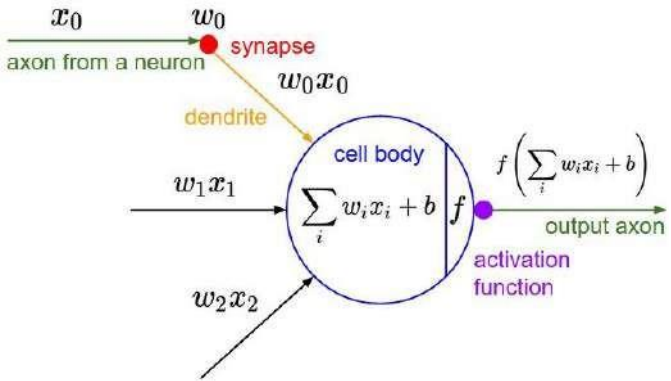
1 broj (vrednost):
rezultat skalarnog proizvoda filtera i tog dela slike
(tj. 5*5*3 = 75-dimenzioni skalarni proizvod)

Mozak/neuron pogled na CONV Sloj



32x32x3 slika

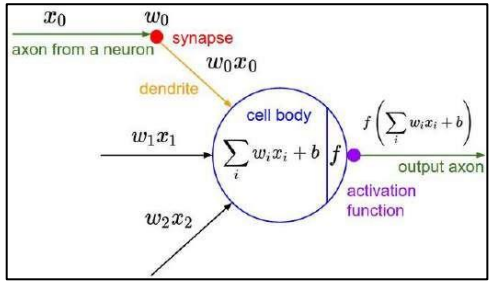
5x5x3 filter

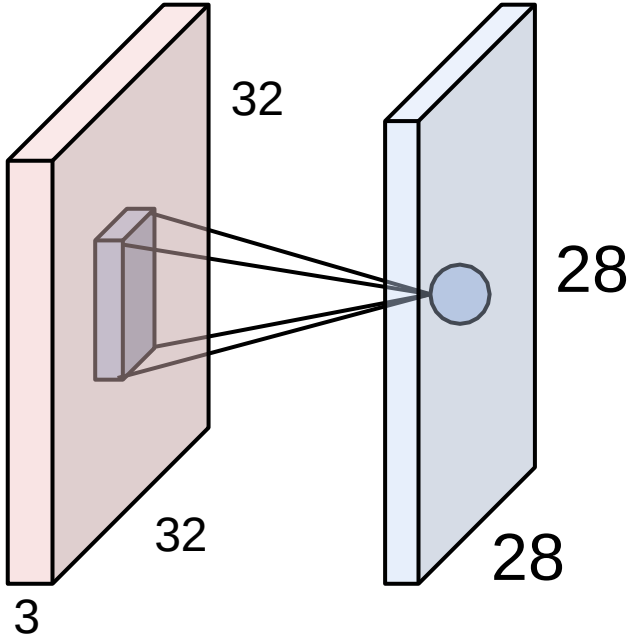


to je samo neuron koji je povezan sa samo jednim delom slike...

1 broj (vrednost):
rezultat skalarnog proizvoda filtera i tog dela slike
(tj. 5*5*3 = 75-dimenzioni skalarni proizvod)

Mozak/neuron pogled na CONV Sloj



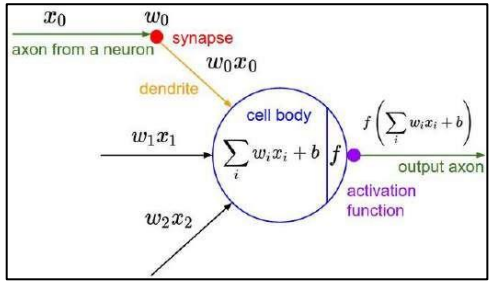


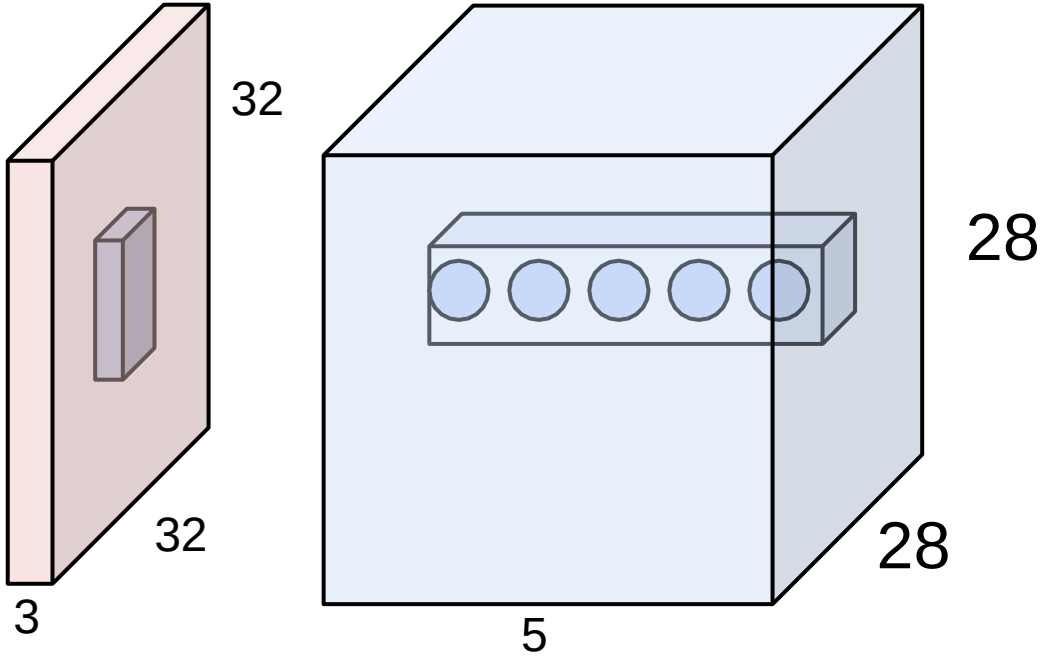
Aktivaciona mapa dimezionalnosti 28x28 je sloj izlaza neurona:

1. Svaki je povezan sa malim delom ulaza
2. Svi dele iste parametre

“5x5 filter” -> “vidno polje dimenzionalnosti 5x5”

Mozak/neuron pogled na CONV Sloj





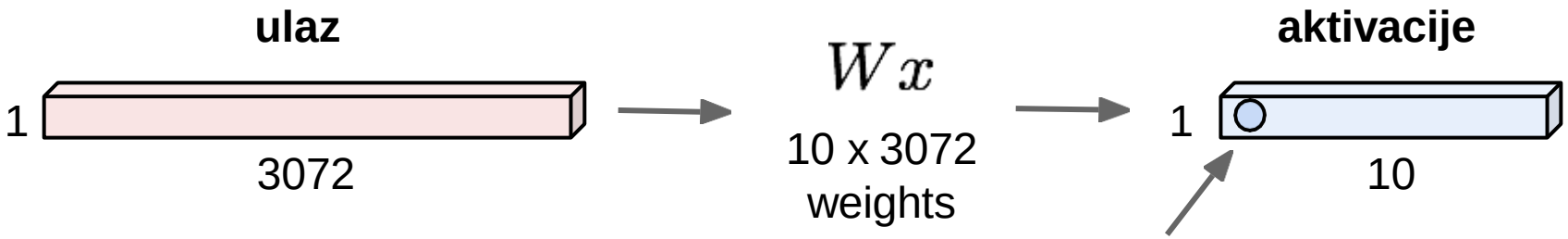
Npr. 5 filtera,
CONV sloj se sastoji od neurona složenih u mrežu (28x28x5)

Imamo 5 različitih neurona koji su povezani sa istim delom ulaza

Podsetnik: Potpuno Povezani Sloj

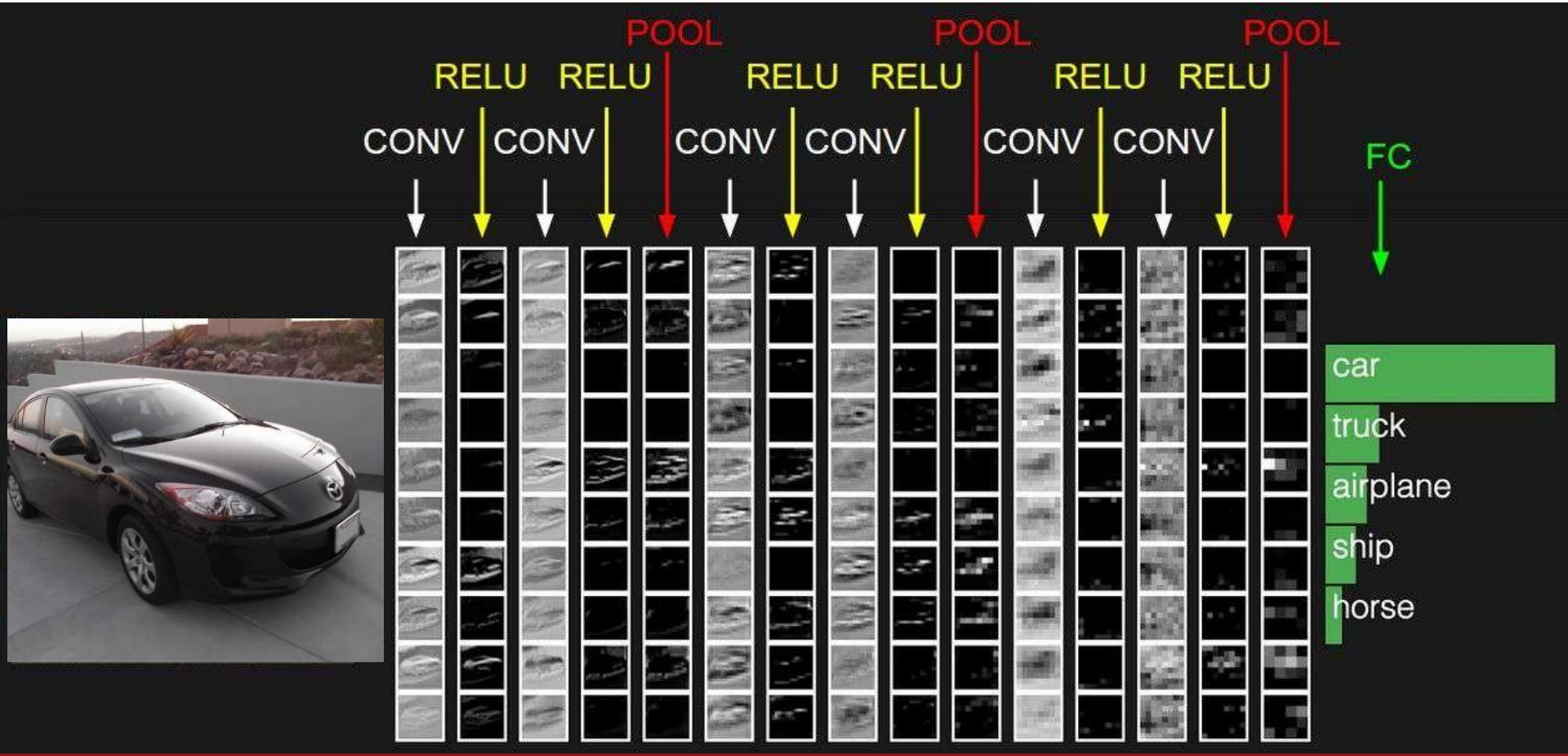
32x32x3 slika -> razvijamo u 3072 x 1

Svaki neuron je povezan sa celom slikom.



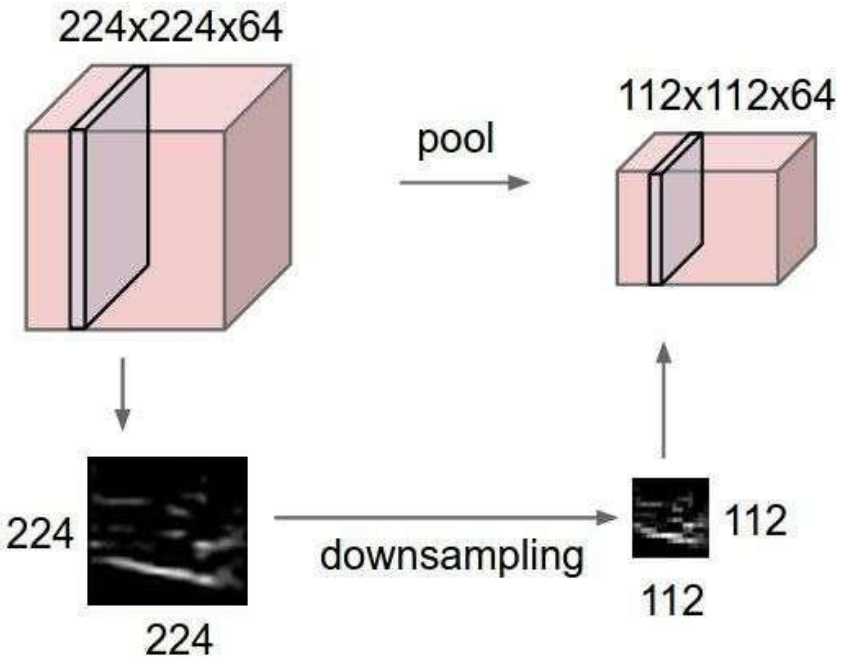
1 broj (vrednost):
rezultat skalarnog proizvoda jednog reda u W i celog ulaza (3072-dimenzioni skalarni proizvod)

Nismo još objasnili sledeće slojeve: POOL/FC



Pooling Sloj

- smanjujemo dimenzionalnost slojeva, sa ciljem da možemo lakše da obučimo mrežu (manje parametra), ali da pri tom sačuvamo što više informacija
- primenjuje se nezavisno na svaku aktivacionu mapu:



MAX POOLING

Jedna aktivaciona mapa dubine 1

x

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

y

max pool sa 2x2
fliterima i korakom 2

6	8
3	4

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

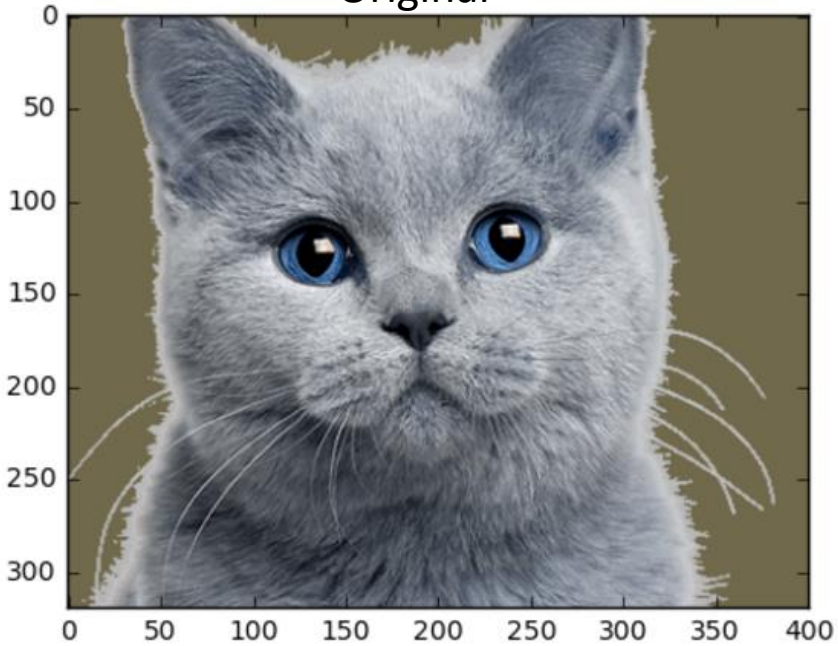
Tipična podešavanja:

F = 2, S = 2
F = 3, S = 2

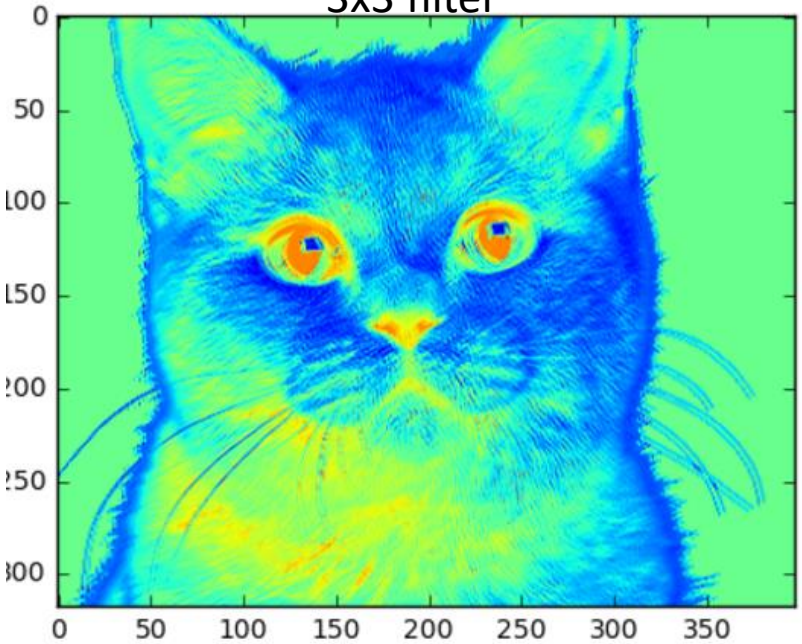
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Primer jednog konvolucionog filtera 1

Original



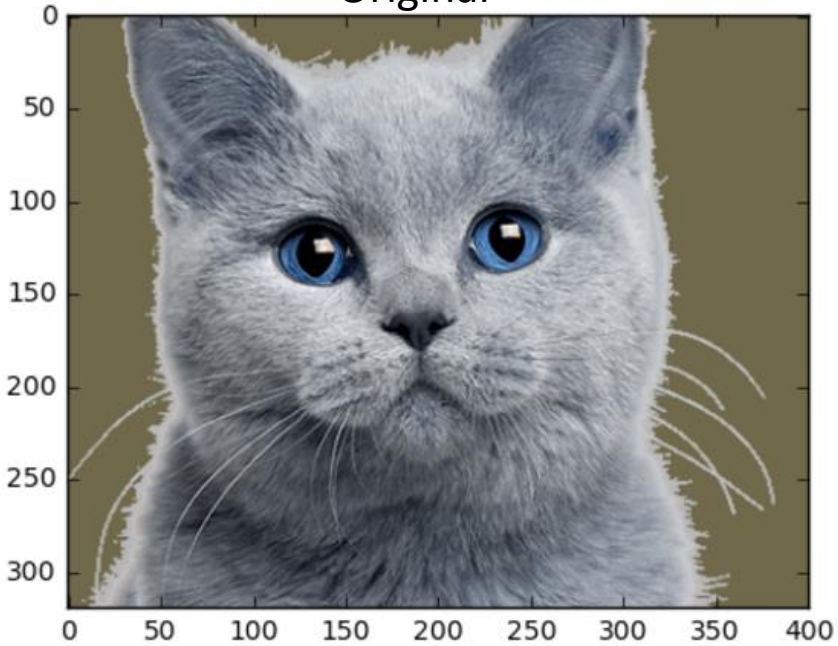
3x3 filter



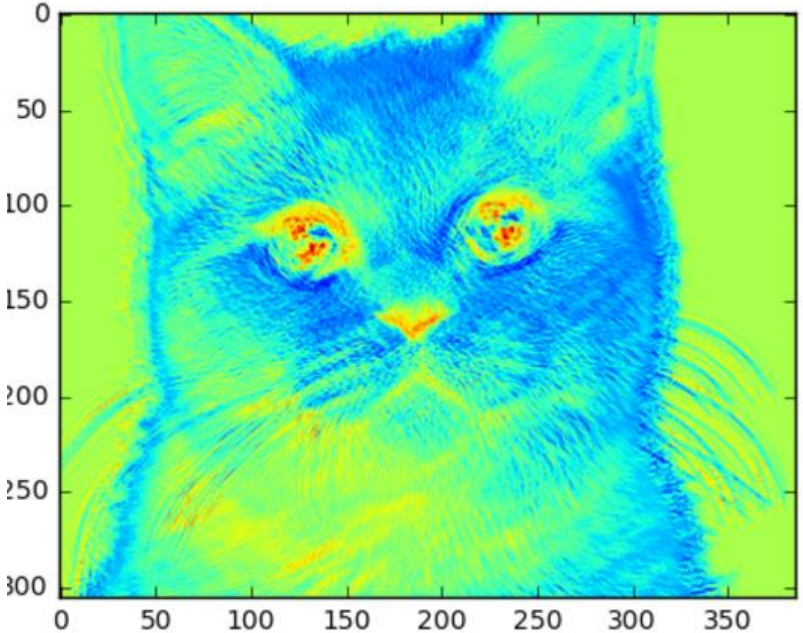
Izvor: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

Primer jednog konvolucionog filtera 2

Original



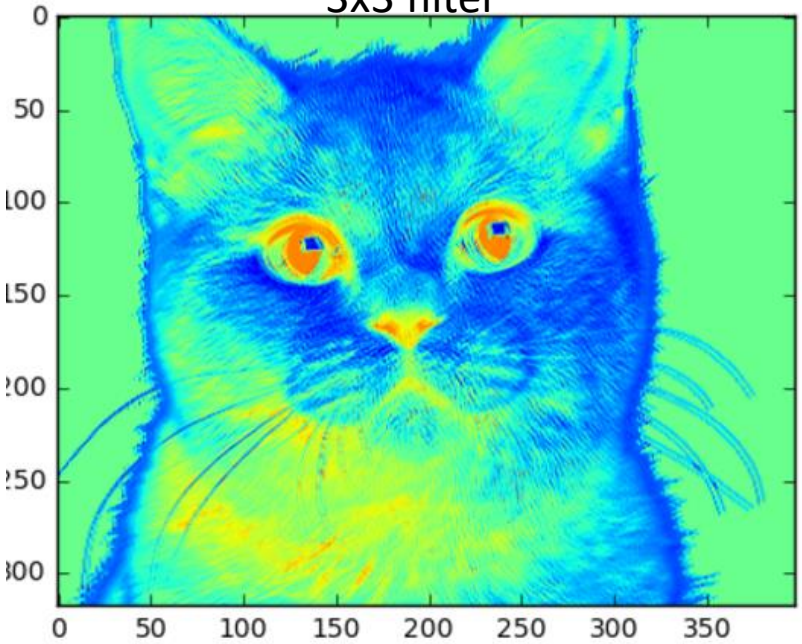
15x15 filter



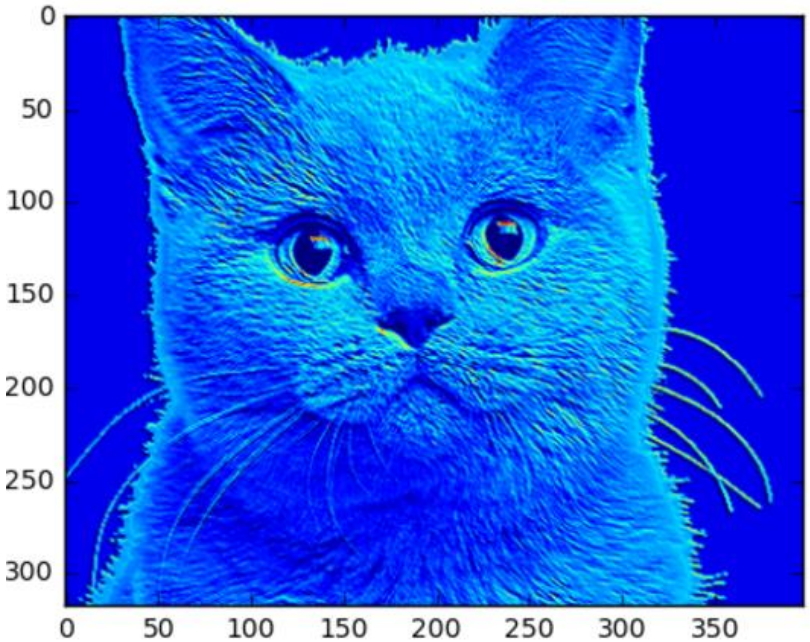
Izvor: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

Primer jednog konvolucionog filtera 3, ReLU posle 3x3 filtera

3x3 filter



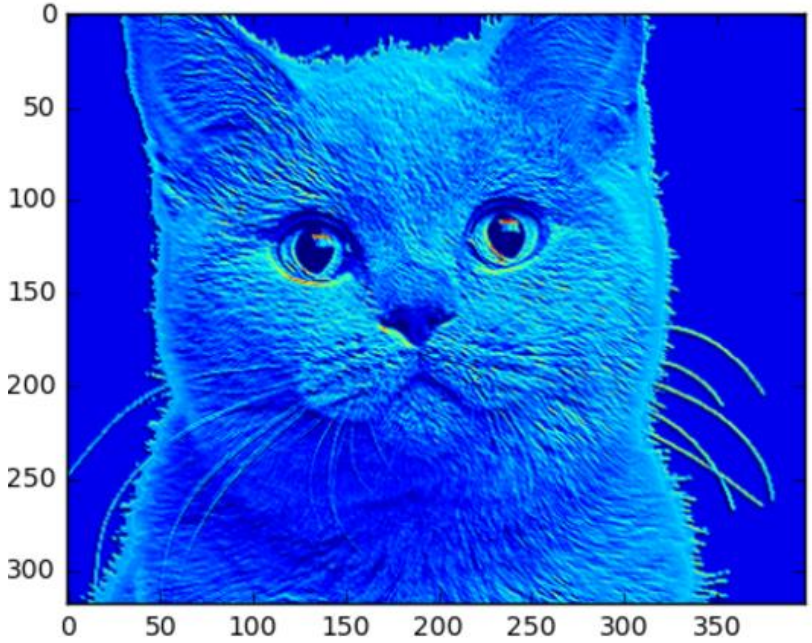
ReLU



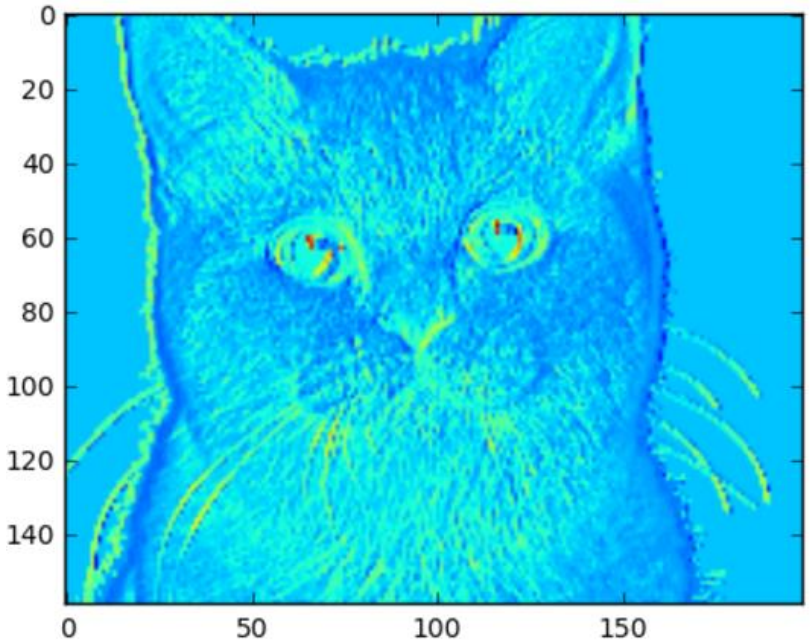
Izvor: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

Primer jednog konvolucionog filtera 4, Max Pooling 2x2 posle ReLU

ReLU

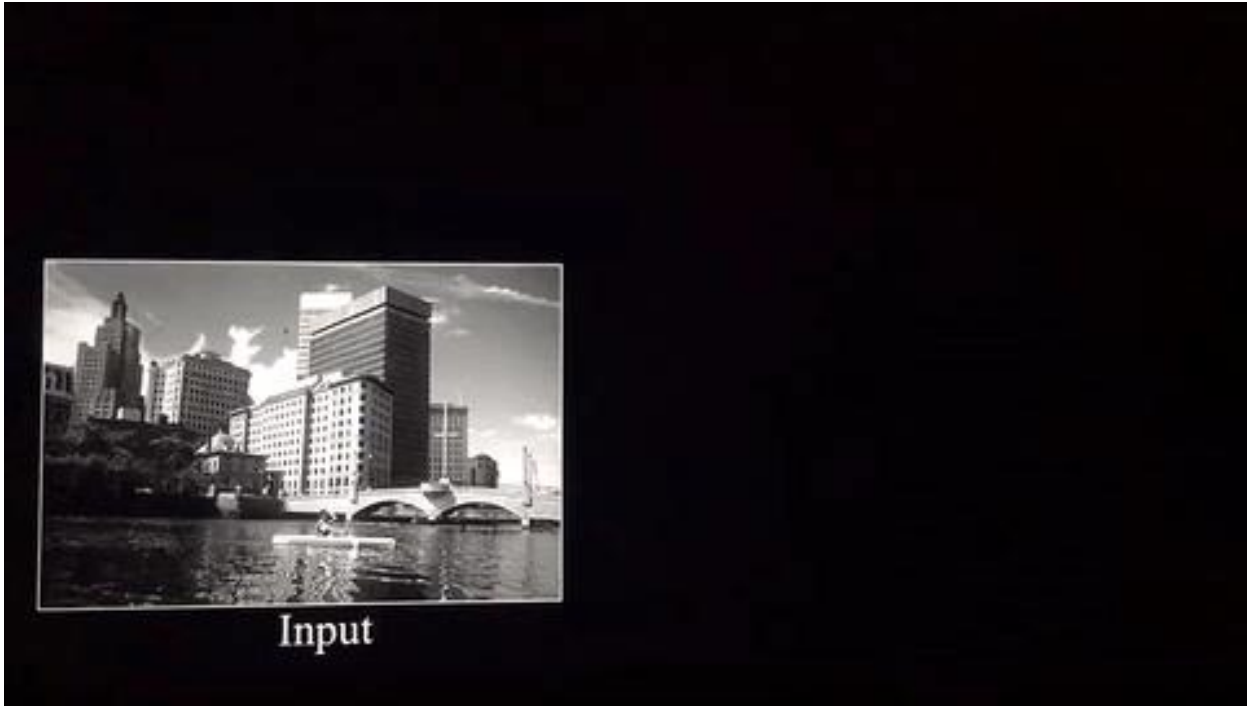


Max Pooling 2x2



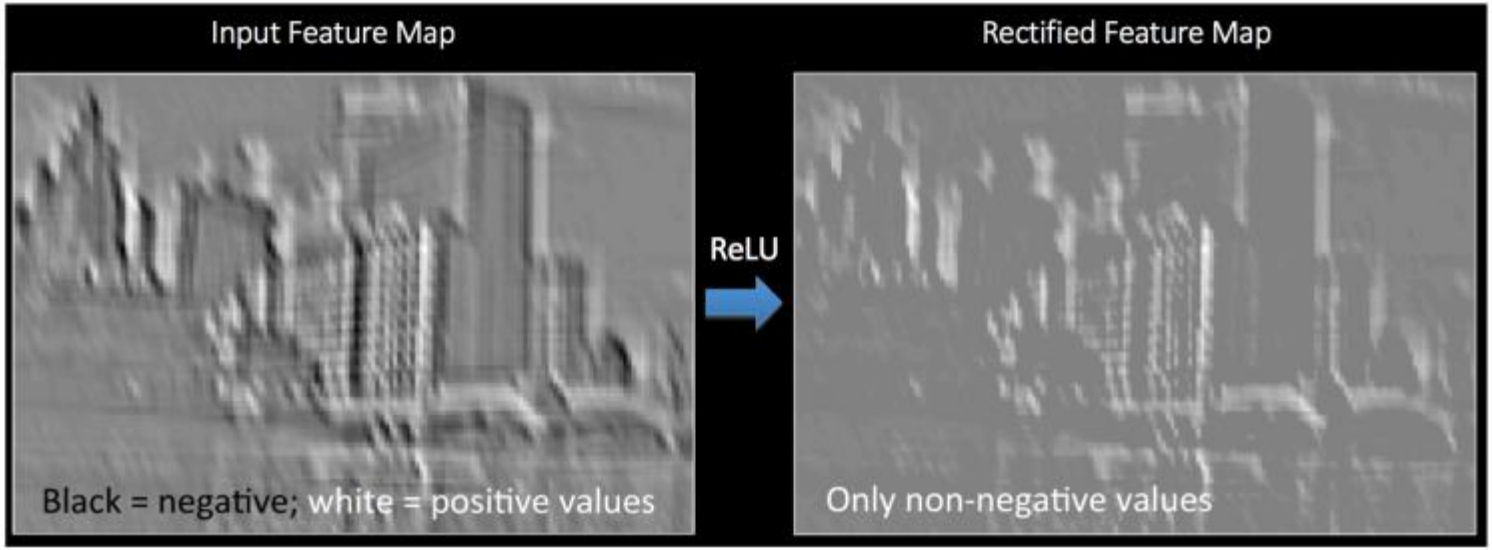
Izvor: https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59

Primer još jednog konvolucionog filtera 1



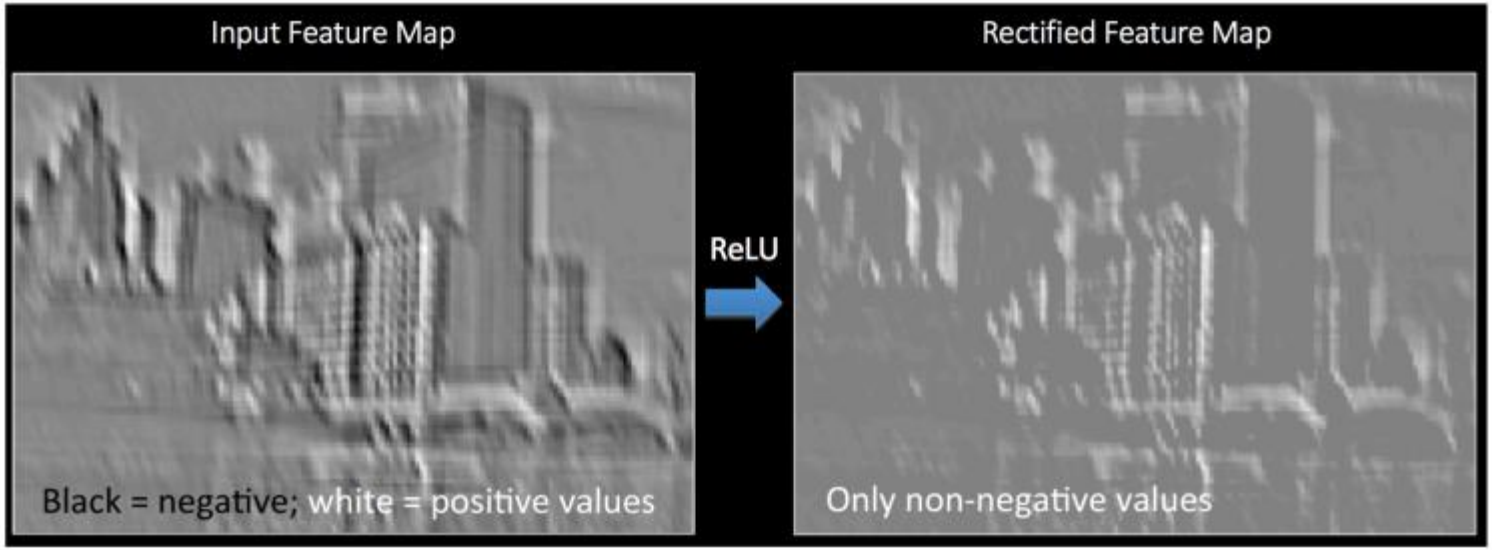
Izvor: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Primer još jednog konvolucionog filtera 2, ReLU



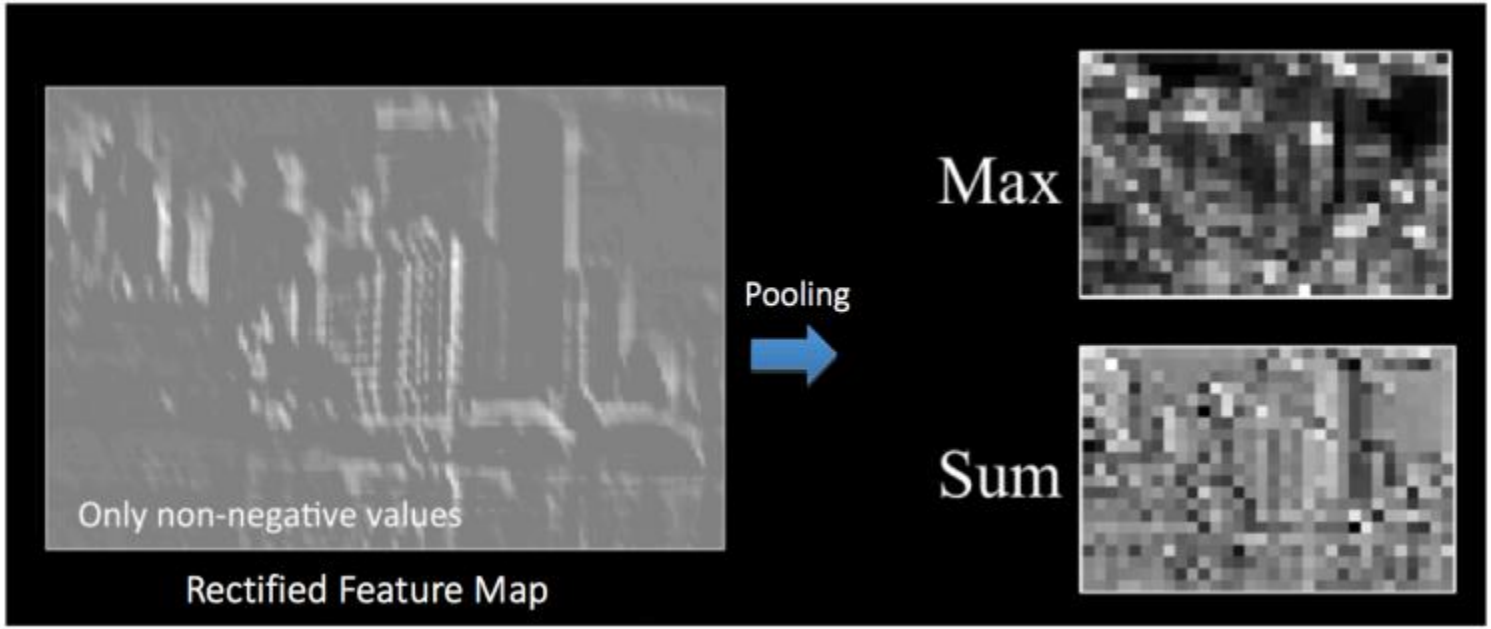
Izvor: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Primer još jednog konvolucionog filtera 2, ReLU



Izvor: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Primer još jednog konvolucionog filtera 3, Pooling

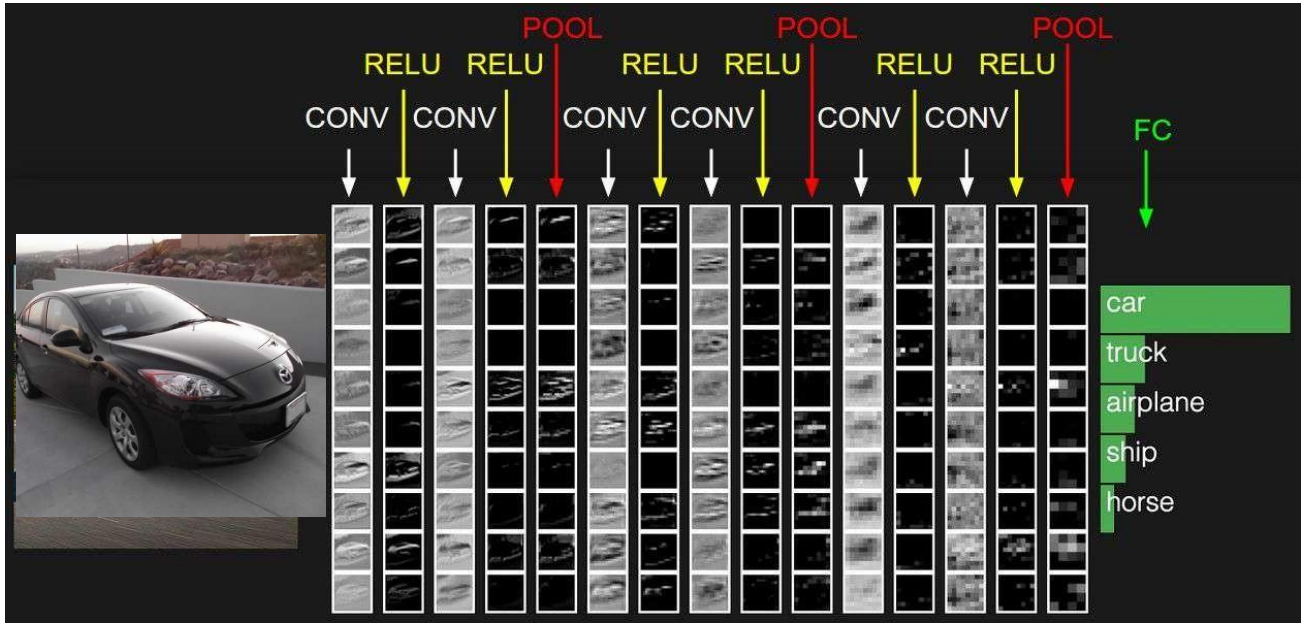


Izvor: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Potpuno Povezani Sloj

Fully Connected Layer (*FC layer*)

- Sadrži neurone koji su potpuno povezani sa svojim ulazom, kao u tipičnoj neuronskoj mreži

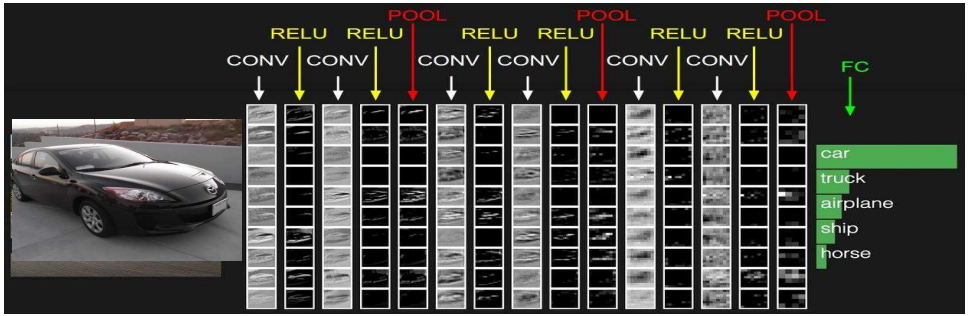


Potpuno Povezani Sloj

Fully Connected Layer (*FC layer*)

- Sadrži neurone koji su potpuno povezani sa svojim ulazom, kao u tipičnoj neuronskoj mreži
  Konvolutivne slojeve možemo da shvatimo kao ekstrakciju osobina iz slike koje pomoću kojih onda obučavamo potpuno povezanu neuronsku mrežu na kraju.

  Zašto nisu svi slojevi potpuno povezani?

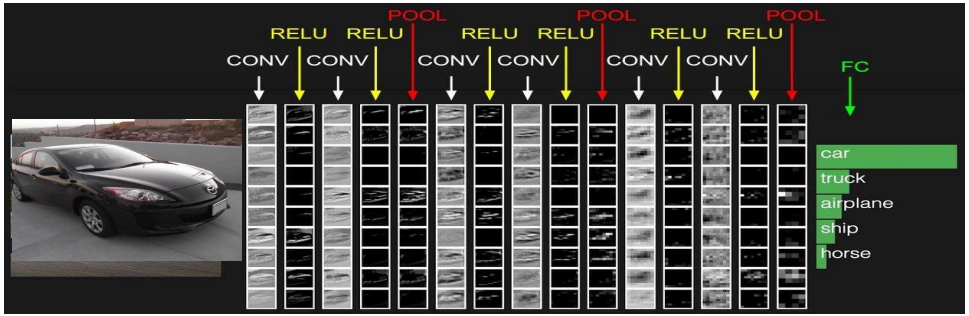


Potpuno Povezani Sloj

Fully Connected Layer (*FC layer*)

- Sadrži neurone koji su potpuno povezani sa svojim ulazom, kao u tipičnoj neuronskoj mreži
  Konvolutivne slojeve možemo da shvatimo kao ekstrakciju osobina iz slike koje pomoću kojih onda obučavamo potpuno povezanu neuronsku mrežu na kraju.

  Zašto nisu svi slojevi potpuno povezani?
  Takava mreža bi imala previše parametra da bi realno mogla da se obuči!



[ConvNetJS demo: obučavanje na CIFAR-10]

ConvNetJS CIFAR-10 demo

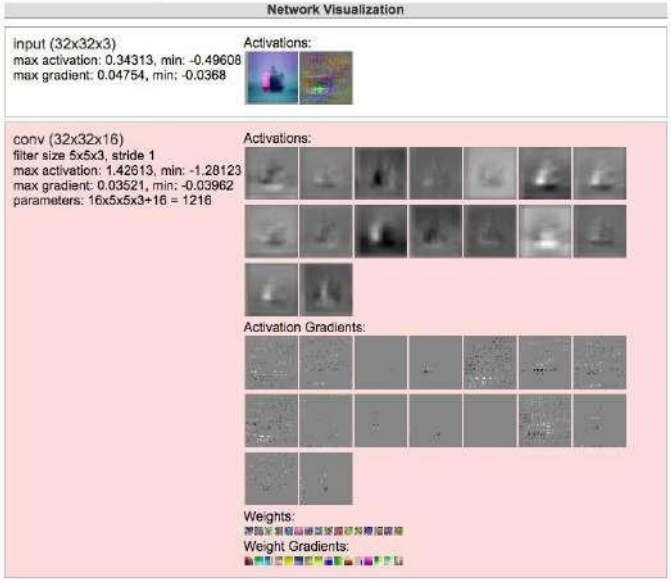
Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.



http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

Rezime

- Kovolutivne mreže slažu CONV,POOL i FC slojeve
- Trenutno je trend ka manjim filterima, a dubljim mrežama
- Eksperimentiše se i sa uklanjanjem POOL/FC slojeva (ostaju samo CONV)
- Tipična arhitektura izgleda ovako:

  [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX

  gde je N~5, M je veliko, 0 <= K <= 2.

  - ali neke od trenutno najboljih mreža ResNet/GoogLeNet menjaju ovu arhitekturu