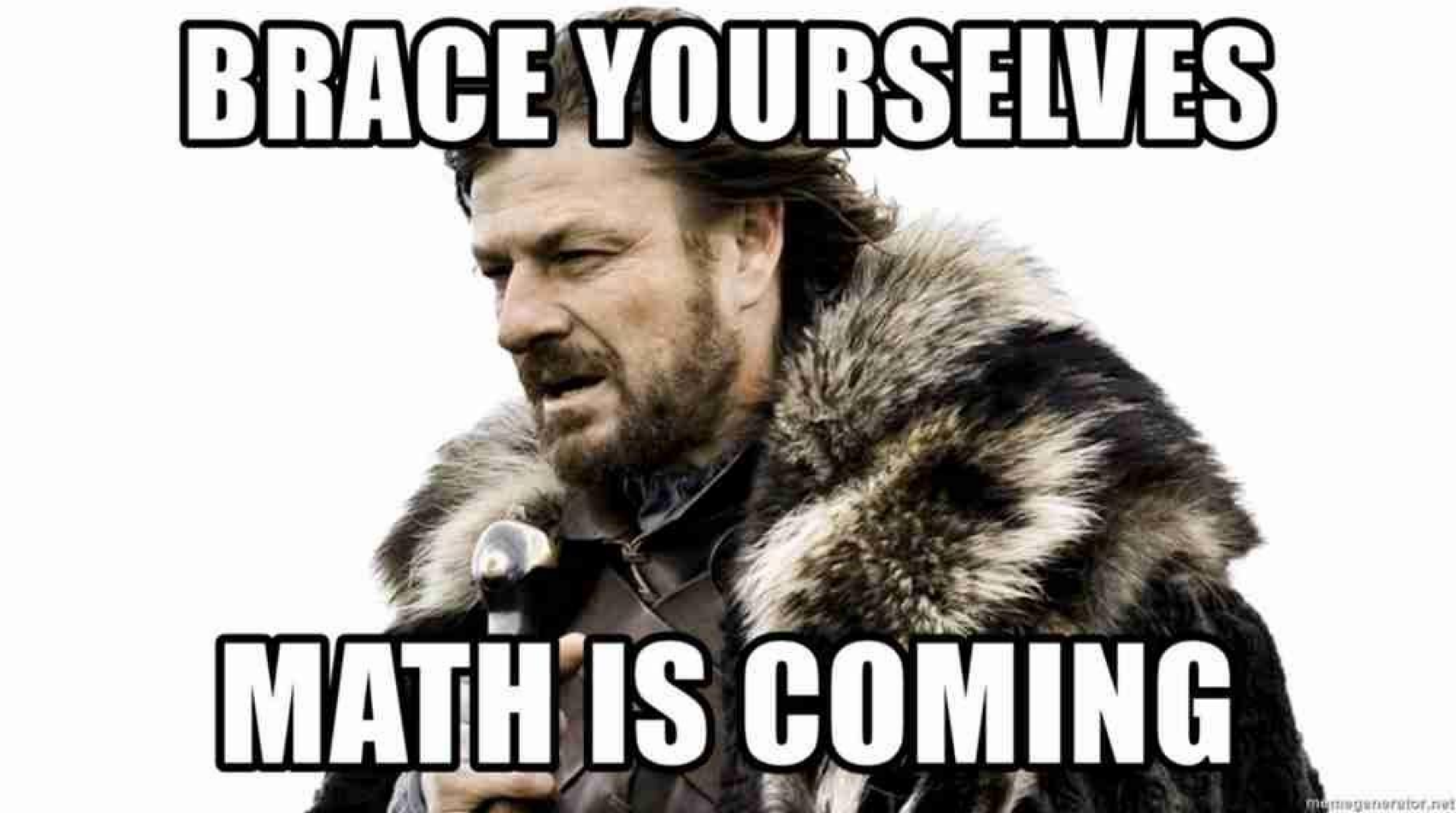
www.lanaco.com

Elementaran uvod u neuronske mreže

Jelena Gojić
Vladimir Mijić


BRACE YOURSELVES
MATH IS COMING

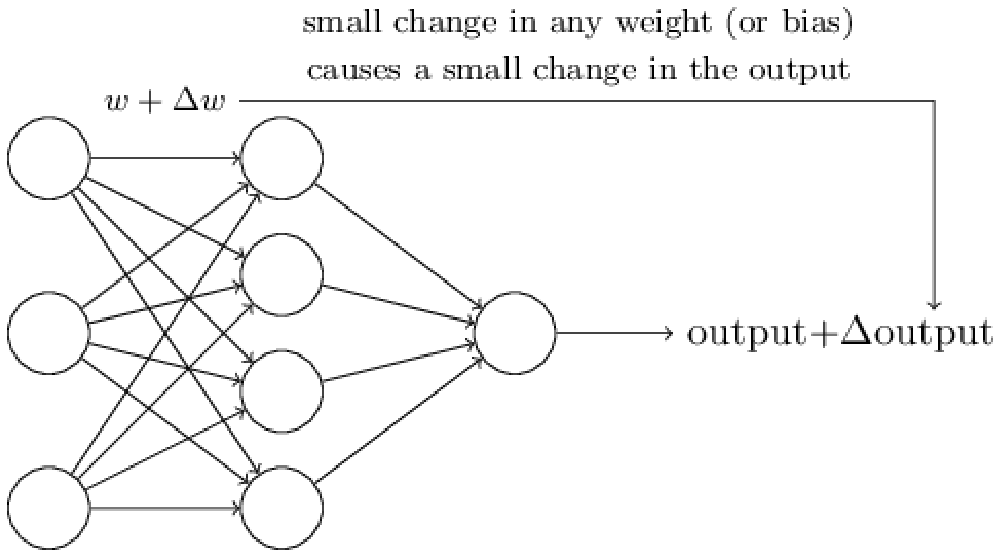
Funkcija aktivacije

- Svaki neuron (čvor) skrivenih slojeva i izlaznog sloja, kao ulaz uzima izlaz iz prethodnog sloja i primjenjuje funkciju na skalarni proizvod težina i ulaza.
- Funkcija aktivacije
- Određuje da li će neuron biti aktiviran ili ne – tj. da li će izlaz neurona doći do narednog sloja (threshold).

$$f(\sum_i (w_i x_i + b_i))$$

Funkcija aktivacije

- Kako bismo imali kontrolu nad procesom učenja, bilo bi dobro da izbjegnemo situacije u kojima male promjene u težinskim faktorima dovode do velikih promjena u dobijenom rezultatu.
- Dosadašnja postavka nam to ne garantuje.
- Jedna od uloga funkcije aktivacije je upravo osiguravanje ove osobine.
- Postiže se odabirom adekvatne funkcije.



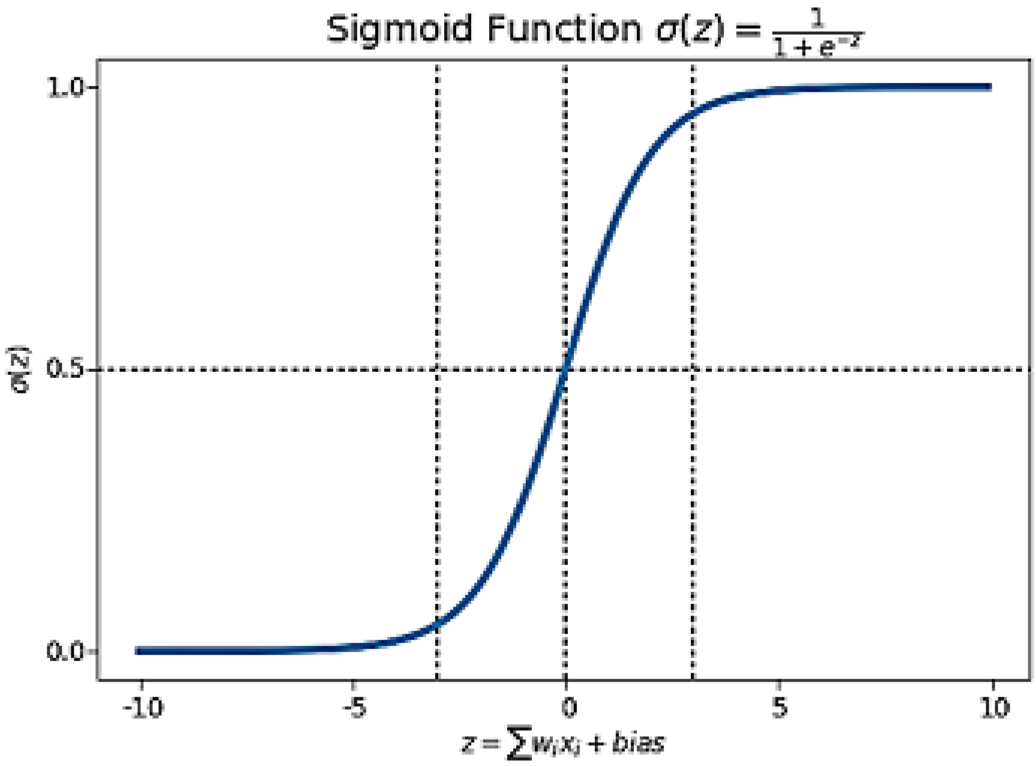
Funkcija aktivacije

- Osigurava nelinearnost i preciznost.
- Kakve osobine funkcija aktivacije treba imati?
  - Monotona
  - Diferencijabilna
  - Neprekidna
  - **Glatka**
- Zašto baš ove osobine?

Sigmoid neuron

Sigmoid neuron = neuron koji koristi sigmoidnu funkciju kao funkciju aktivacije

Iskoristićemo osobine funkcije $\sigma(z) = \frac{1}{1+e^{-z}}$ i dobiti „zaglađeno“ ponašanje



Sigmoid neuron

- Kako bismo postigli željenu osobinu, jednostavno ćemo $wx + b$ preslikati funkcijom $\sigma(z) = \frac{1}{1+e^{-z}}$ i tako ga „smjestiti“ u interval (0, 1)
- Takođe, ulazne informacije u sigmoidni neuron ne moraju da budu binarnog tipa nego proizvoljan realan broj!
- Dakle, za ulazne realne brojeve $(x_1, x_2, \dots)$, težine $(w_1, w_2, \dots)$ i bias b, izlaz iz sigmoidnog neurona je realan broj:

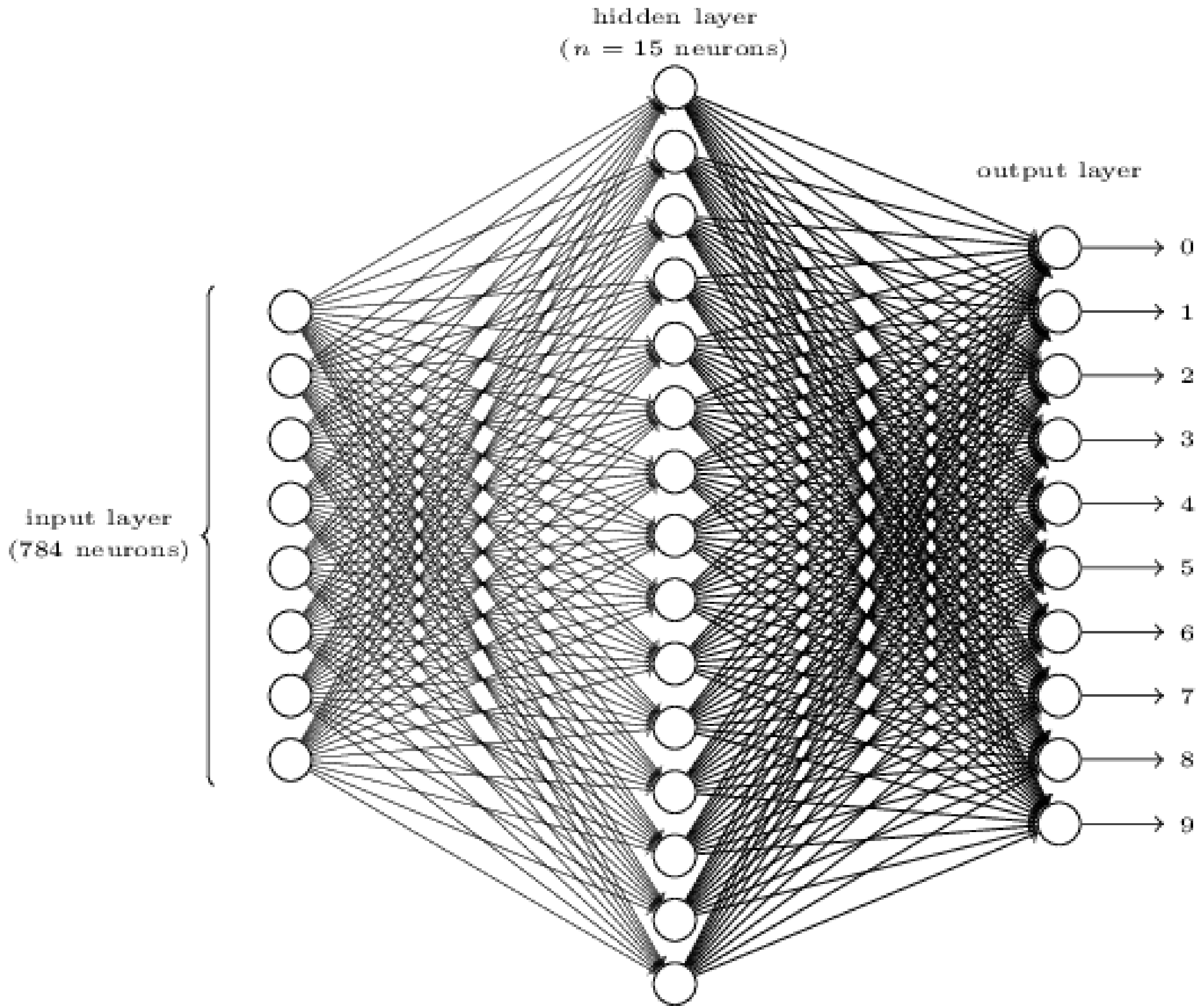
$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$

Sigmoid neuron

- Šta smo postigli primjenom ove funkcije aktivacije?
  - Tip ulaznih i izlaznih informacija (realni brojevi)
  - Male promjene težina i bias-a rezultuju malim promjenama na izlazu (neprekidnost)
  - Sigmoidna funkcija je glatka!
  - Osnovne osobine diferencijabilnosti garantuju nam željenu osobinu
- Zašto baš sigmoidna funkcija?
- Sigmoidna funkcija ima „lijepe“ osobine pri diferenciranju

Podsjetimo se problema

- Implementirati deep feedforward neuronsku mrežu koristeći Numpy
- Arhitektura mreže:
  - Ulazni sloj: 784 neurona (= 28 * 28)
  - Skriveni sloj: 15 neurona
  - Izlazni sloj: 10 neurona (10 cifara)



  - Pitanja:
    - Zašto ulazni sloj ima baš 784 neurona?
    - Zašto skriveni sloj ima baš 15 neurona?
    - Zašto izlazni sloj ima baš 10 neurona?

Primjer

- MNIST skup podataka
- Tensorflow, instalacija, učitavanje MNIST skupa podataka, elementarne transformacije (reshape, to_categorical, ...)

Notacija

- x – ulaz u neuronsku mrežu (784-dimenzionalan vektor)
- $y = y(x)$ – željeni izlaz iz neuronske mreže (tačan izlaz)
- $a = a(x)$ – trenutni izlaz iz neuronske mreže

Notacija

- Krajnji cilj:

  - Izračunati (podesiti) težine i bias-e na takav način da je metrika udaljenosti između y i a minimalna za sve vektore x koji su iskorišćeni za treniranje

  - Generalizacija – Dobijene težine i bias-i moraju „raditi“ na test skupu ali i za ostale moguće pojavne oblike ručno napisanih slika (train skup je dovoljno reprezentativan za sve buduće pojave)

Loss funkcija

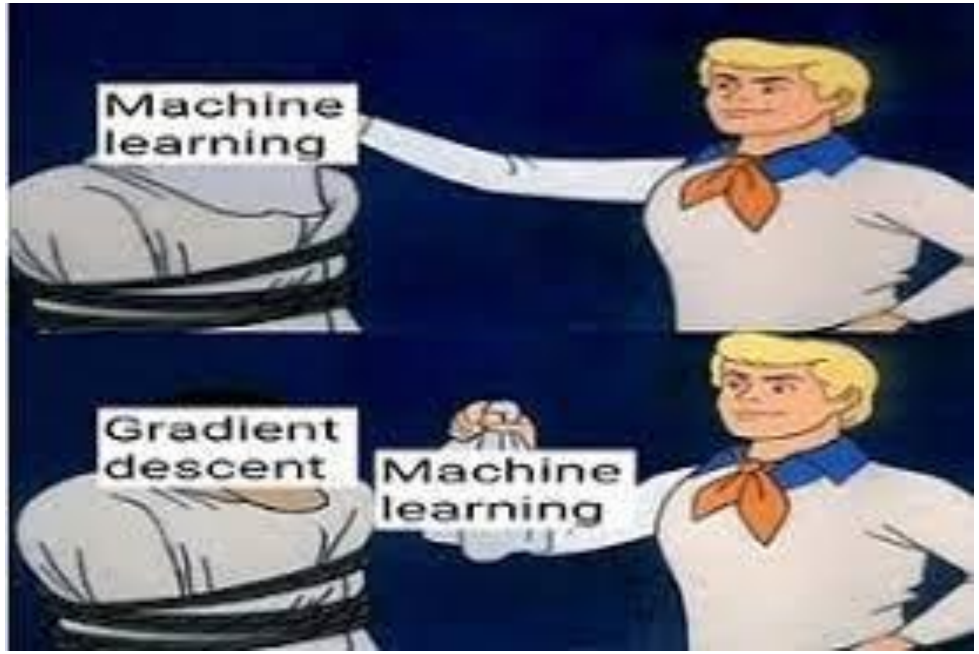
- Da bismo mogli pratiti koliko dobro postižemo prethodno opisani cilj, definišemo loss funkciju.
- Loss funkcija - funkcija po težinama i bias-u
- Cilj je pronaći minimum po težinama i bias-ima
- Kvadratna cost funkcija (Mean Squared Error)

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

- Pitanja:
  - Zašto baš kvadratna cost funkcija?
  - Zar ne možemo koristiti zdravorazumsku logiku – direktno maksimizovati broj tačno klasifikovanih ulaznih vektora?

Gradijentni spust (Gradient descent)

- Dakle, problem podešavanja težina i bias-a s ciljem da metrika udaljenosti između vektora y i a bude najmanja moguća, svodimo na problem minimizovanja cost funkcije C i riješićemo ga tehnikom poznatom pod nazivom gradijentni spust



Gradijentni spust (Gradient descent)

- Ima li smisla problemu pristupiti analitički?

- Posmatrajmo sada generalnu funkciju $C: A \to \boldsymbol{R}, A \subset \boldsymbol{R}^n$ u nekoj fiksnoj tački $v = (v_1, \dots, v_n)$
- Ako vektor v „dovoljno malo“ pomjerimo u nekom smjeru, na primjer $(v_1 + \Delta v_1, \dots, v_n + \Delta v_n)$, nas zanima koliko će se promijeniti vrijednost funkcije C

Stav 2.1.1. Neka je funkcija $C(A) \to \boldsymbol{R}$ diferencijabilna u tački $v = (v_1, \dots, v_n) \in A$. Tada je funkcija C neprekidna u tački v, postoje parcijalni izvodi $\frac{\partial C}{\partial v_i}(v), i = 1, \dots, n$, a diferencijal ima oblik

$$dC(v)(\Delta v) = \frac{\partial C}{\partial v_1}(v)\Delta v_1 + \cdots + \frac{\partial C}{\partial v_n}(v)\Delta v_n$$

pri čemu je $\Delta v = (\Delta v_1, \dots, \Delta v_n)$ takvo da $v + \Delta v \in A$

tj. $\Delta C(v, \Delta v) = C(v_1 + \Delta v_1, \dots, v_n + \Delta v_n) - C(v_1, \dots, v_n) \approx \frac{\partial C}{\partial v_1}(v)\Delta v_1 + \cdots + \frac{\partial C}{\partial v_n}(v)\Delta v_n,$

ili skraćeno

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 + \ldots + \frac{\partial C}{\partial v_n}\Delta v_n.$$

- Pri čemu su parcijalni izvodi definisani sa:

$$\frac{\partial C}{\partial v_i} = \lim_{\Delta v_i \to 0} \frac{C\left(v_1, v_2, \ldots, v_i + \Delta v_i, \ldots, v_n\right) - C\left(v_1, v_2, \ldots, v_n\right)}{\Delta v_i}$$

i daju nam stopu promjene funkcije *C* po jednoj od promjenljivih

Sigmoid neuron (flashback)

- Ako se sjetimo uvodne priče o sigmoid neuronima, imali smo isti problem:

$$\Delta \text{output} \approx \sum_j \frac{\partial\, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial\, \text{output}}{\partial b} \Delta b$$

  - Odnosno da se promjena vrijednosti na izlazu može aproksimirati linearnom kombinacijom promjena težina i bias-a!

Gradijentni spust (Gradient descent)

- Ako sa $\nabla C = (\frac{\partial C}{\partial v_1}, \dots, \frac{\partial C}{\partial v_n})$ označimo gradijent (smjer najbržeg rasta funkcije *C*), koristeći skalarni proizvod vektora, priraštaj funkcije *C* se može izraziti i kao

$$\Delta C = \nabla C \cdot \Delta v$$

- S obzirom da nam je cilj pronaći minimum, ideja je da se malim koracima „u pravom smjeru“, prije ili kasnije stigne do minimuma

- Šta je pravi smjer?

Gradijentni spust (Gradient descent)

- Prije svega, jasno je da ΔC mora biti negativno, ako želimo malim koracima stići do minimuma
- „Pravi smjer“:

$$\Delta v = -\eta \nabla C$$

gdje je η dovoljno mali (ili veliki) realan broj da se obezbijedi „pravi put“ do mimimuma koji nazivamo **learning rate** (stopa učenja).

Gradijentni spust (Gradient descent)

- Tako da sada imamo:

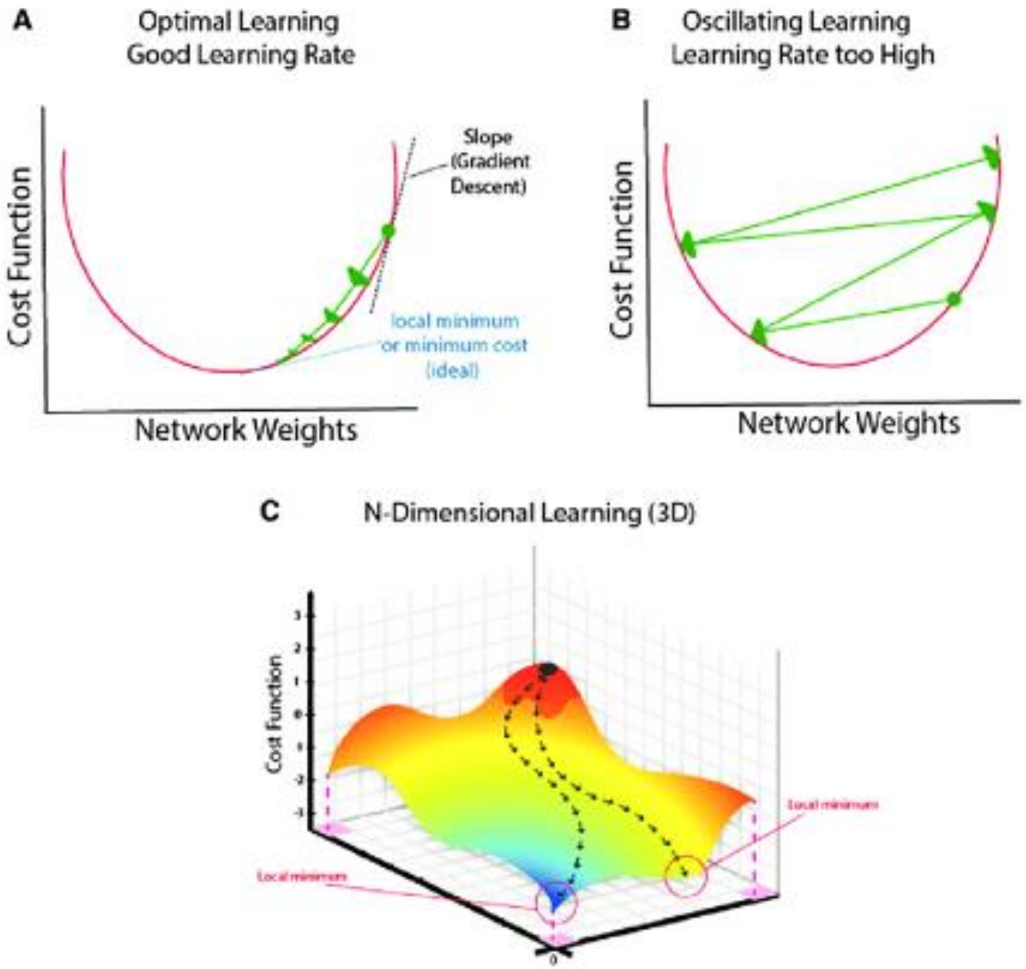
$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$$

i s obzirom da je $\|\nabla C\|^2 \geq 0$, to znači da je $\Delta C \leq 0$, odnosno C će se uvijek smanjivati s ovakvim izborom Δv

- Ovako opisani izbor smjera i dužine koraka se naziva **gradijentni spust**

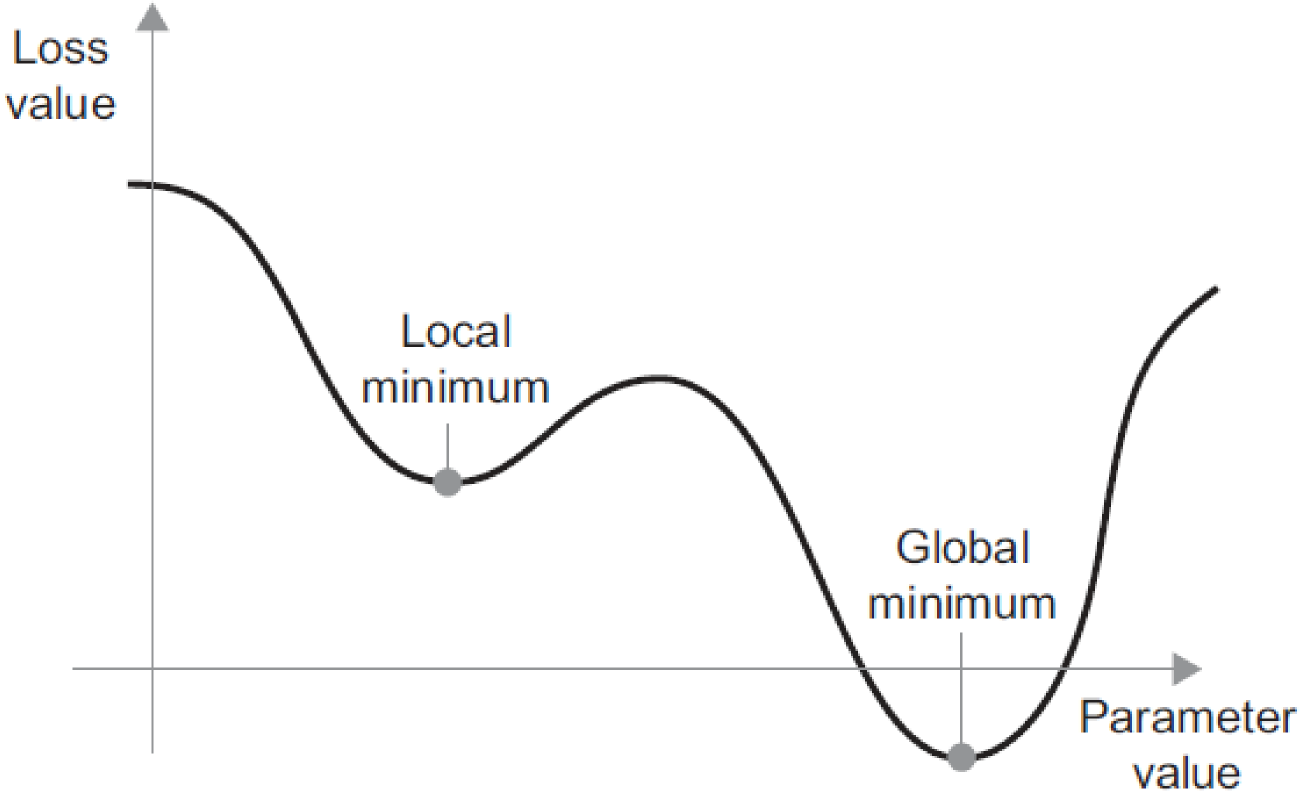
Gradijentni spust (Gradient descent)

- Međutim, neophodno je biti pažljiv:



Gradijentni spust (Gradient descent)

- Međutim, neophodno je biti pažljiv:



Gradijentni spust (Gradient descent)

Gradijentni spust (Gradient descent)

- Ako se vratimo na originalni problem, odnosno na podešavanje težina i bias-a, sljedeći korak prema pravim njihovim vrijednostima, pravimo na način:

$$w_k \rightarrow w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\partial C}{\partial b_l}.$$

- Postoji više načina kako podešavati težine i bias-e
- Imajući u vidu definiciju cost funkcije, gradijente je moguće izračunavati za sve x-ove za treniranje pojedinačno i na osnovu toga graditi strategiju

Gradijentni spust (Gradient descent)

- Načini podešavanja parametara – terminologija:
  - Batch – komad podataka za treniranje
  - Stochastic = random (batch je odabran nasumično)
  - Mini-batch stochastic gradient descent – nasumično se bira podskup skupa za treniranje
  - True-SDG – nasumično se bira jedan primjer iz skupa za treniranje
  - Batch-SDG – svaki korak se pokreće na cijelom skupu za treniranje

Gradijentni spust (Gradient descent)

- Najjednostavniji načini su sledeći:
- Ažiriranje poslije svakog izračunatog ∇C_x za x iz skupa za treniranje (bilo redom ili slučajnim izborom)
- Ažuriranje izračunati prosjekom gradijenata ∇C_x za sve x ili za neki dovoljno veliki podskup skupa za treniranje
- Postupak je moguće organizovati po mini-batch-evima, kojima će se „pokriti“ čitav skup za treniranje
- Adagrad, RMSprop, Adam, ...

Primjer

Notebook: 03 – simple gradient descent

Notebook: 04 - mse gradient descent

Backpropagation algorithm

- Originalno osmišljen 60-ih godina prošlog vijeka, ali nije popularizovan sve do objave rada
  - Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986)

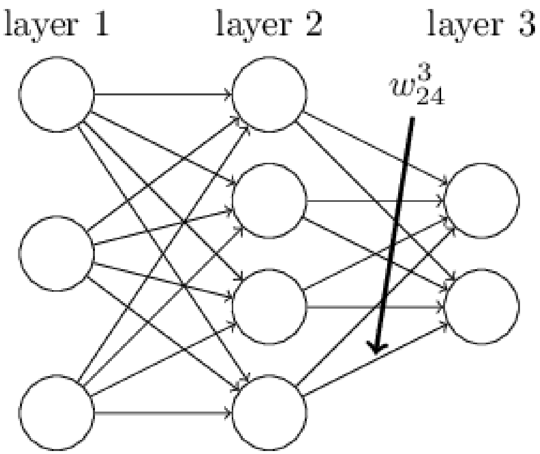
- Dvije faze:
  - Forward pass: za ulazni vektor x izračunavamo izlazni vektor $a = a(x)$
  - Backward pass: izračunavamo gradijent loss funkcije u finalnom sloju i koristimo ga za postupno ažuriranje inicijalnih težina i bias-a

Backpropagation algorithm - notacija

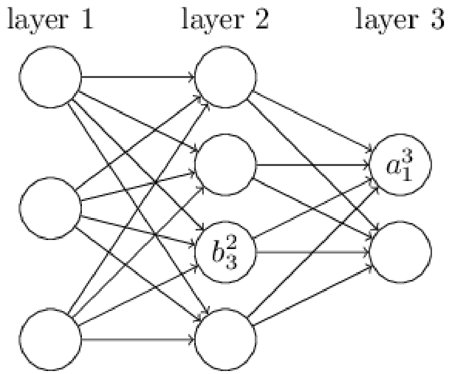
- Vještačka neuronska mreža sa L slojeva
- Vrijednosti aktivacija u l-tom sloju se čuvaju u vektor koloni $\boldsymbol{a}^l$ za sve slojeve, $l = 1, \dots, L$
- Veze između neurona sloja $(l-1)$ i l-tog sloja se čuvaju u matrici $\boldsymbol{W}^l$, dok se odgovarajući bias-i pomenute matrice veza čuvaju u vektor koloni $\boldsymbol{b}^l$

Backpropagation algorithm - notacija

- w_{jk}^{l} je težina veze između k-tog neurona sloja $(l-1)$ i j-tog neurona l-tog sloja



w_{jk}^{l} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer



- Način računanja aktivacije j-tog neurona, l-tog sloja:

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

pri čemu suma ide po neuronima k iz sloja $(l-1)$!!!

Backpropagation algorithm - notacija

- Na primjer, ako sloj $(l-1)$ ima 6 neurona, a l-ti ima 3 neurona

$$\boldsymbol{W}^l = \begin{bmatrix} w_{11}^l & w_{12}^l & w_{13}^l & w_{14}^l & w_{15}^l & w_{16}^l \\ w_{21}^l & w_{22}^l & w_{23}^l & w_{24}^l & w_{25}^l & w_{26}^l \\ w_{31}^l & w_{32}^l & w_{33}^l & w_{34}^l & w_{35}^l & w_{36}^l \end{bmatrix}, \boldsymbol{a}^{l-1} = \begin{bmatrix} a_1^{l-1} \\ a_2^{l-1} \\ a_3^{l-1} \\ a_4^{l-1} \\ a_5^{l-1} \\ a_6^{l-1} \end{bmatrix}, \boldsymbol{b}^l = \begin{bmatrix} b_1^l \\ b_2^l \\ b_3^l \end{bmatrix}$$

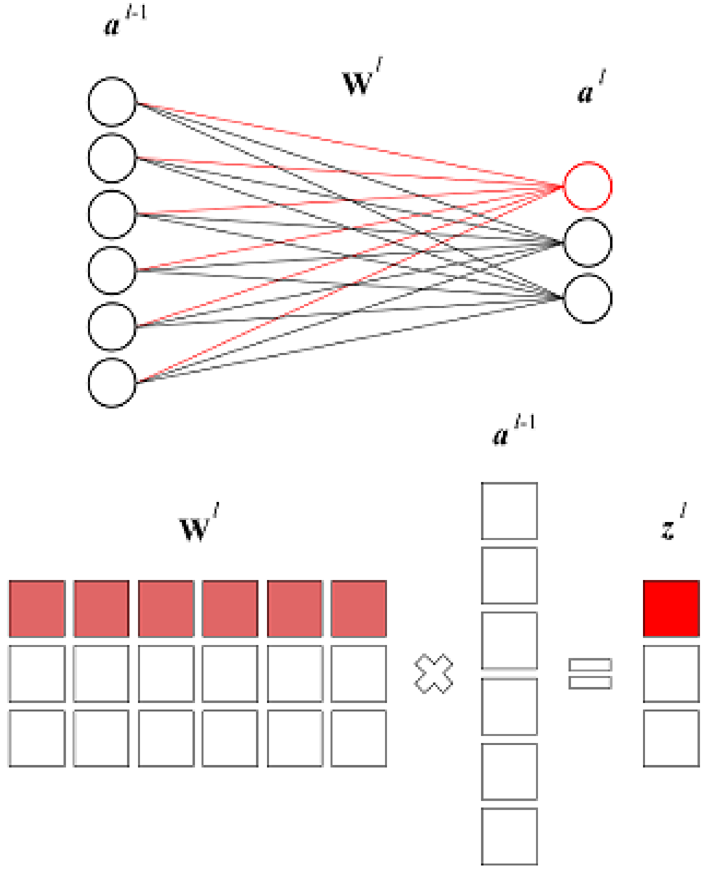
- To znači da se forward pass iz sloja $(l-1)$ prema l-tom sloju izračunava pomoću:

$$a^l = \sigma(\boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l)$$

- Funkcija aktivacije σ nad vektor kolonom se primjenjuje po elementima vektor kolone

Backpropagation algorithm - notacija

- Množenje matrice $\boldsymbol{W}^l$ i vektor kolone $\boldsymbol{a}^{l-1}$ se može prikazati na sljedeći način:



Backpropagation algorithm - notacija

- Pri čemu je $\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$ ulaz sa težinama neurona u l-tom sloju

- Ili napisano po komponentama:

$$z_j^l = \sum_k w_{jk}^l \, a_k^{l-11} + b_j^l$$

- **$\boldsymbol{w_{jk}^l}$ je težina veze između $\boldsymbol{k}$-tog neurona sloja $(\boldsymbol{l}-\mathbf{1})$ i $\boldsymbol{j}$-tog neurona $\boldsymbol{l}$-tog sloja**

Backpropagation algorithm - notacija

- Forward pass čitave mreže se može napisati kao:

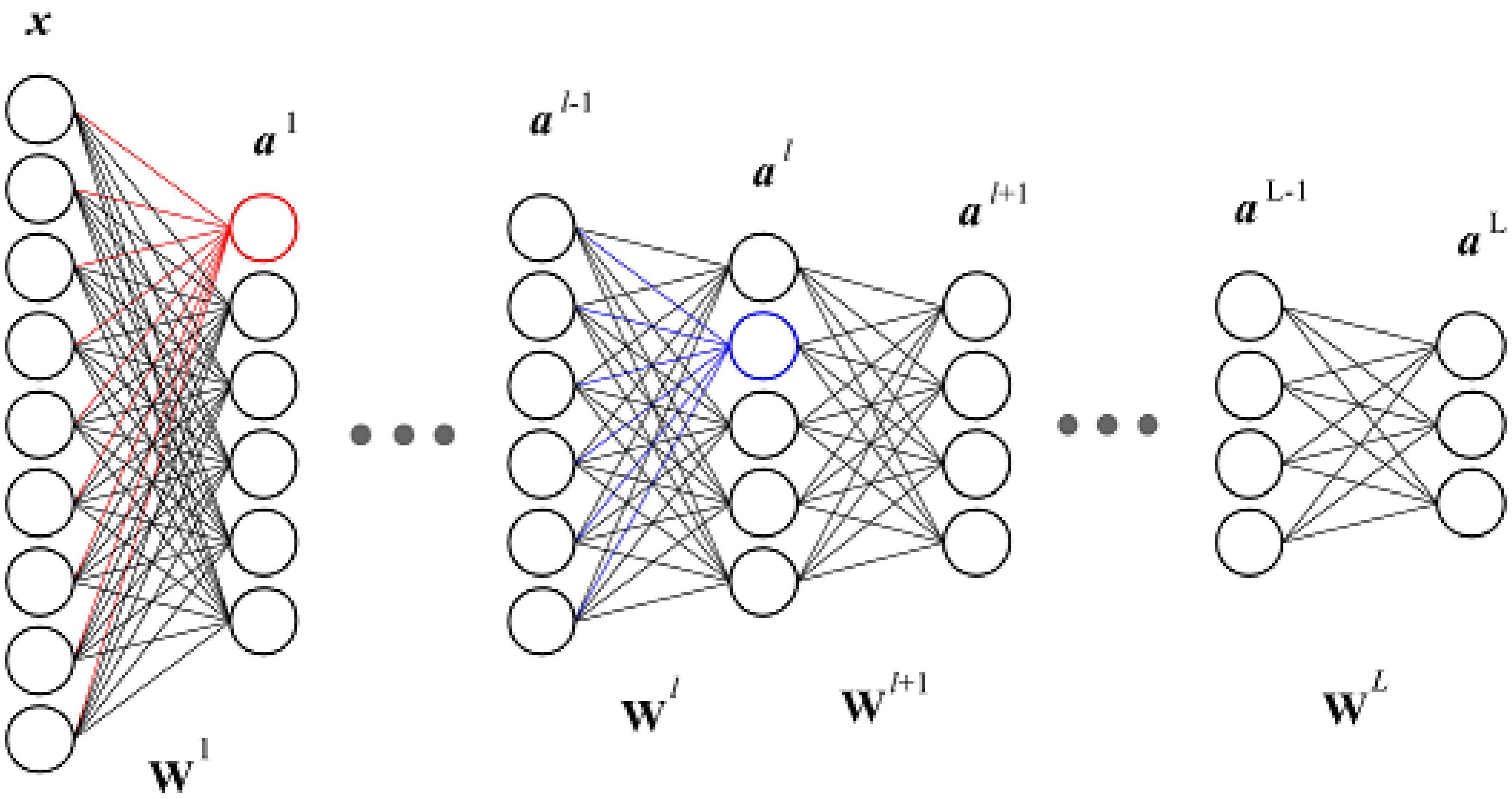
$$\boldsymbol{a}^L = \sigma(\boldsymbol{W}^L[\ldots[\sigma(\boldsymbol{W}^2[\sigma(\boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1)] + \boldsymbol{b}^2)]\ldots] + \boldsymbol{b}^L),$$

odnosno:

$$a_n^L = [\sigma(\sum_m w_{nm}^L[\ldots[\sigma(\sum_j w_{kj}^2[\sigma(\sum_i w_{ji}^1 x_i + b_j^1)] + b_k^2)]\ldots]_m] + b_n^L)]_n$$

- gdje w_{uv}^l označava vezu od v-tog neurona u sloju $l-1$ do u-tog neurona u sloju l
- dok je b_u^l bias u-tog neurona u sloju l

Backpropagation algorithm - notacija



Primjer

Notebook: 05 – matrices cheat sheat

Notebook: 05.1 – matrices task

Backpropagation algorithm

- Backpropagation algoritam se odnosi na razumijevanje kako promjene težina i bias-a utiču na promjenu vrijednosti cost funkcije.
- U praksi to znači izračunati parcijalne izvode $\frac{\partial C}{\partial w_{jk}^{l}}$ i $\frac{\partial C}{\partial b_{j}^{l}}$

1. Izračunati grešku u posljednjem (izlaznom) sloju
2. Izračunati grešku u l-tom sloju koristeći se greškom u sloju $(l+1)$
3. Izračunati stopu promjene cost funkcije u odnosu na bilo koji bias u mreži
4. Izračunati stopu promjene cost funkcije u odnosu na bilo koju težinu u mreži

Backpropagation algorithm – cost funkcja

- U svijetlu novouvedene notacije, cost funkcija ima oblik:

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

- n je broj elemenata u skupu za treniranje
- suma ide po svim elementima skupa za treniranje
- $y(x)$ je „tačna vrijednost“, odnosno vektor kolona koja je označava
- $a^L(x)$ je vektor kolona aktivacija u izlaznom sloju za neki konkretan ulaz x

- Uslov koji funkcija *C* treba da zadovolji jeste da se može izračunati za svako pojedino x, a zatim objediniti za sve x

Backpropagation algorithm – „chain rule“ podsjetnik

- Ako je $y = f(x), x = g(t)$, onda je $\frac{dy}{dt} = \frac{dy}{dx}\frac{dx}{dt}$
- Ako je $z = f(x, y)$, $x = g(t)$, $y = h(t)$, onda je $\frac{dz}{dt} = \frac{dz}{dx}\frac{dx}{dt} + \frac{dz}{dy}\frac{dy}{dt}$
- U multivarijantnom slučaju, ako je *z* funkcija od *n* varijabli $x_1, x_2, \dots, x_n$, a koje su istovremeno funkcije od *m* varijabli $t_1, t_2, \dots, t_m$, onda je:

$$\frac{\partial z}{\partial t_i} = \frac{\partial z}{\partial x_1}\frac{\partial x_1}{\partial t_i} + \frac{\partial z}{\partial x_2}\frac{\partial x_2}{\partial t_i} + \cdots + \frac{\partial z}{\partial x_n}\frac{\partial x_n}{\partial t_i}$$

Backpropagation algorithm

- Da bismo izračunali parcijalne izvode $\frac{\partial C}{\partial w_{jk}^{l}}$ i $\frac{\partial C}{\partial b_{j}^{l}}$, neophodno je definisati međuvrijednost koja će nam biti od koristi:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

- i koju ćemo zvati greškom j-tog neurona u l-tom sloju

- Prvo ćemo pronaći način da izračunamo δ_j^l i onda tu vrijednost povežemo sa $\frac{\partial C}{\partial w_{jk}^{l}}$ i $\frac{\partial C}{\partial b_{j}^{l}}$

Backpropagation algorithm

Backpropagation algorithm (1)

- **1. Izračunati komponente vektora grešaka izlaznog sloja:**

$$
\begin{aligned}
\delta_j^L &= \frac{\partial C}{\partial z_j^L} \\
&= \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L} && \text{(by chain rule, sum is over all neurons in the output layer)} \\
&= \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} && \left(\frac{\partial a_k^L}{\partial z_j^L} = 0 \text{ for all } k \neq j\right) \\
&= \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) && (\text{because } a_j^L = \sigma(z_j^L))
\end{aligned}
$$

Backpropagation algorithm (2)

- **2. Izračunati komponente vektora grešaka $\boldsymbol{\delta^l}$ koristeći komponente vektora grešaka sljedećeg sloja $\boldsymbol{\delta^{l+1}}$:**
- Drugačije rečeno želimo $\delta_j^l = \frac{\partial C}{\partial z_j^l}$ napisati koristeći $\delta_k^{l+1} = \frac{\partial C}{\partial z_k^{l+1}}$

$$
\begin{aligned}
\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\
&= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} && \text{(by chain rule)} \\
&= \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} && \text{(by definition)}
\end{aligned}
$$

Backpropagation algorithm (2)

- **2. Izračunati komponente vektora grešaka δ^l koristeći komponente vektora grešaka sljedećeg sloja δ^{l+1} :**

- Po definiciji je:

$$z_k^{l+1} = \sum_i w_{ki}^{l+1} a_i^l + b_k^{l+1} = \sum_i w_{ki}^{l+1} \sigma(z_i^l) + b_k^{l+1},$$

- Diferencirajući po z_j^l, dobijamo:

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

- I vraćajući u izraz na prethodnoj strani, dobijamo:

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$$

Backpropagation algorithm (3)

- **3. Izračunati stopu promjene cost funkcije u odnosu na promjenu bilo kojeg bias-a**

$$\frac{\partial C}{\partial b_j^l} = \sum_k \frac{\partial C}{\partial z_k^l} \frac{\partial z_k^l}{\partial b_j^l} \qquad \text{(by chain rule)}$$

$$= \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} \qquad \left(\frac{\partial z_k^l}{\partial b_j^l} = 0 \text{ for all } k \neq j\right)$$

$$= \frac{\partial C}{\partial z_j^l} \qquad \left(\text{because } \frac{\partial z_j^l}{\partial b_j^l} = 1 \text{ after differentiation}\right)$$

$$= \delta_j^l$$

Backpropagation algorithm (4)

- **4. Izračunati stopu promjene cost funkcije u odnosu na promjenu bilo koje težine**

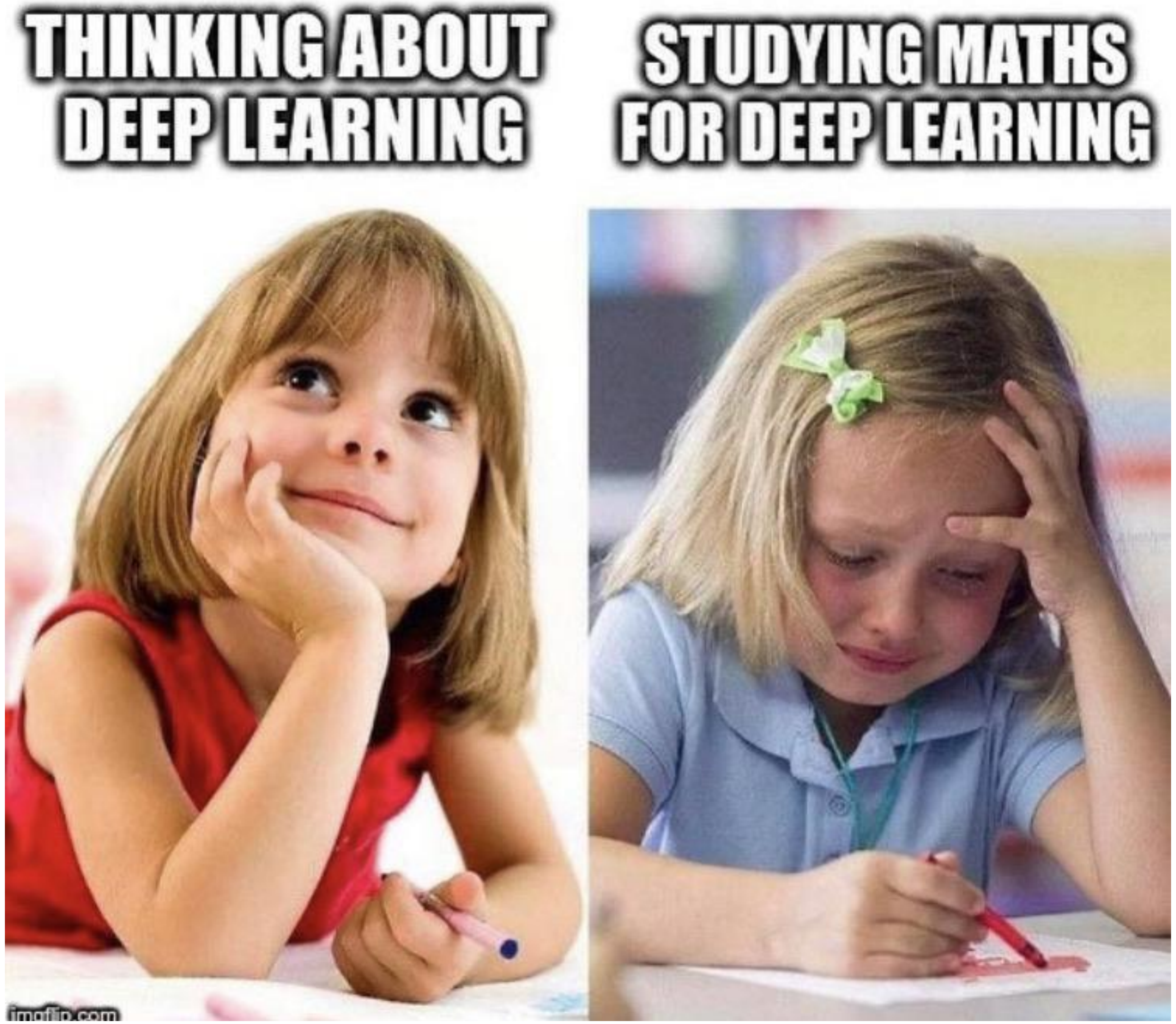
$$\frac{\partial C}{\partial w^l_{jk}} = \sum_k \frac{\partial C}{\partial z^l_k} \frac{\partial z^l_k}{\partial w^l_{jk}} \qquad \text{(by chain rule)}$$

$$= \frac{\partial C}{\partial z^l_j} \frac{\partial z^l_j}{\partial w^l_{jk}} \qquad \left(\frac{\partial z^l_k}{\partial w^l_{jk}} = 0 \text{ for all } k \neq j\right)$$

$$= \frac{\partial C}{\partial z^l_j} a^{l-1}_k \qquad \left(\text{because } \frac{\partial z^l_j}{\partial w^l_{jk}} = a^{l-1}_k \text{ after differentiation}\right)$$

$$= \delta^l_j a^{l-1}_k$$

Backpropagation algorithm



Backpropagation algorithm - koraci

- 1. Ulazni vektor $\boldsymbol{x}$ postaviti kao $\boldsymbol{a}^1$ za ulazni sloj
- 2. Feedforward, za svako $l = 2, 3, \ldots, L$ izračunati $\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$ i $\boldsymbol{a}^l = \sigma(\boldsymbol{z}^l)$
- 3. Izračunati vektor grešaka sloja L: $\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)$
- 4. Za svako L-1, L-1, ..., 2 izračunati ostale vektore grešaka: $\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$
- 5. Izračunati gradijent cost funkcije:

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

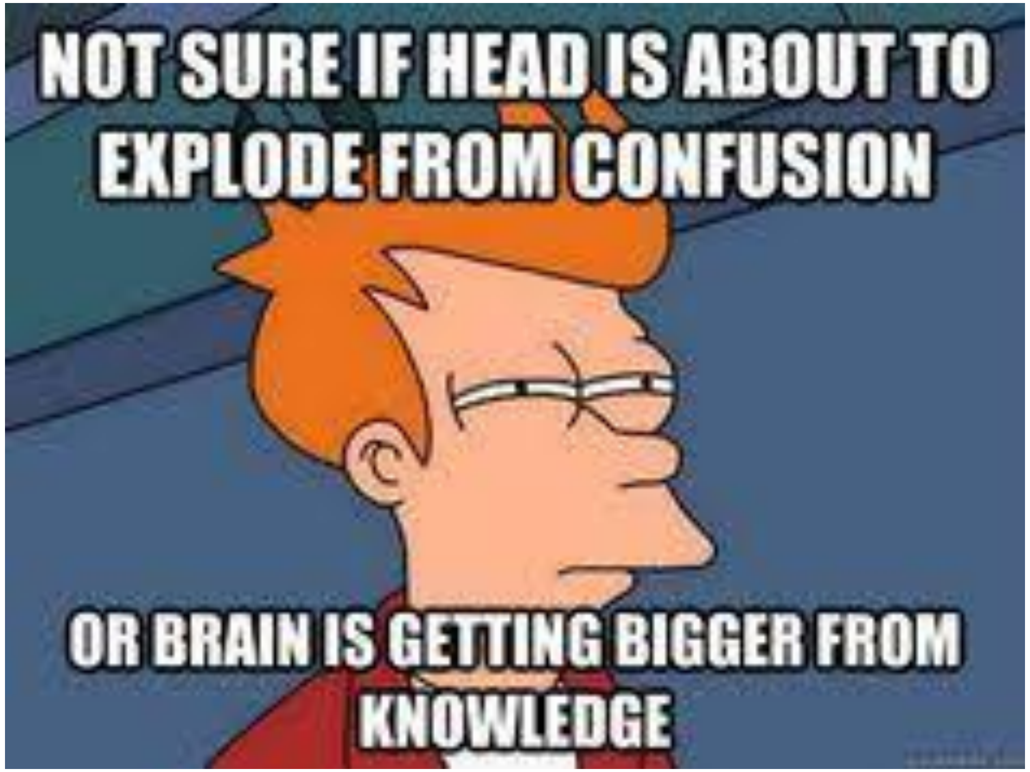
Primjer

Notebook: 06 – zip function

Notebook: 07 – argmax, to categorical

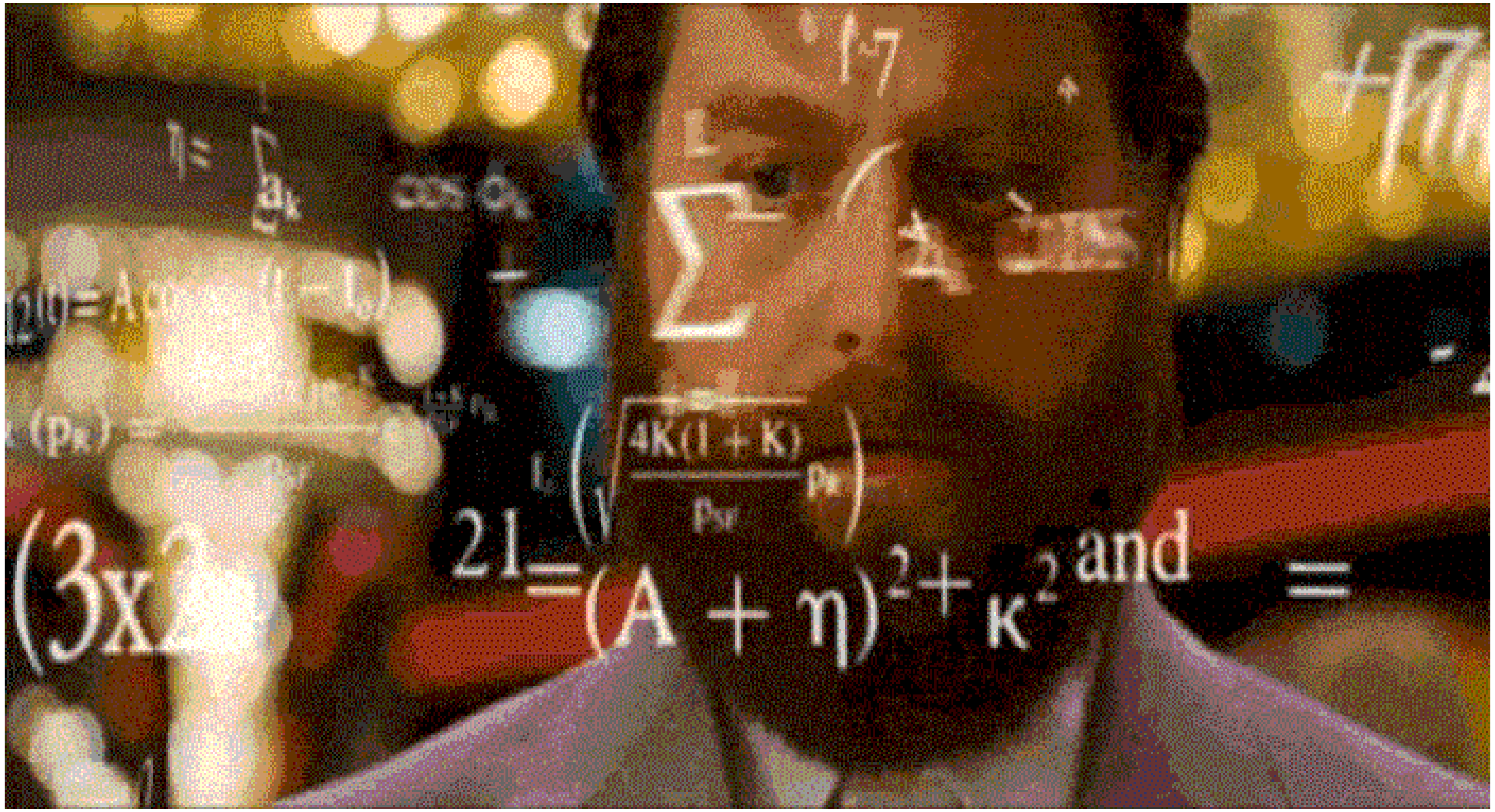
Notebook: 08 – second matrices task

Notebook: 09 – mse gradient descent



Primjer

- Implementacija feed forward neuronske mreže




CHEERS!!
YOU DID IT
imgflip.com