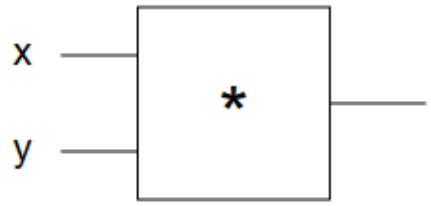
Priprema za vežbe iz predmeta Osnovi računarske inteligencije

Redni broj vežbe: 7

Oblast:

NEURONSKE MREŽE

Najbolji način za razumevanje Neuronskih mreža jeste posmatrati ih kao kola na čije ulaze se dovode realne vrednosti, koje interaguju na izlazu.



Slika 1: Primer prostog kola

Na ulaz kola, prikazanog na slici 1, dovode se dve realne vrednosti **x** i **y** čiji se proizvod **x*y** računa na izlazu ***, matematički: ***f(x,y) = xy.*** Izlaz uzima jednu (**exp**) ili dve (***, +, max**) vrednosti sa ulaza i daje realan broj.

Problem koji posmatramo izgleda ovako:

1. na ulaz kola smo doveli neke specifične ulazne vrednosti (npr. **x = -2, y = 3**)
2. kolo je izračunalo izlaznu vrednost (npr. **-6**)
3. ključno pitanje tada postaje: **kako blago promeniti ulaz da bi se povećao izlaz?**

U našem slučaju, u kom smeru treba da promenimo **x, y** da dobijemo broj veći od **-6**? Primećujemo da npr. **x = -1.99** i **y = 2.99** daje **x*y = -5.95**, što je veće od **-6.0**. Dobili smo poboljšanje od **0.05**, iako je vrednost **-5.95** (rastojanje od nule) manja.

Strategija #1: Nasumična lokalna pretraga
Nasumično blago menjati **x** i **y** i uzeti najbolje vrednosti:
x = -1.9928, y = 2.9901, x * y = -5.9588 > -6.0.
Da li je dobra strategija za kola sa mnogo ulaza?

Strategija #2: Numerički gradijent
Nezavisno menjamo ulaze **x** i **y** za malu vrednost **h** da bi smo povećali izlaz.

$$\frac{\partial f(x,y)}{\partial x} = \frac{f(x+h,y) - f(x,y)}{h}$$

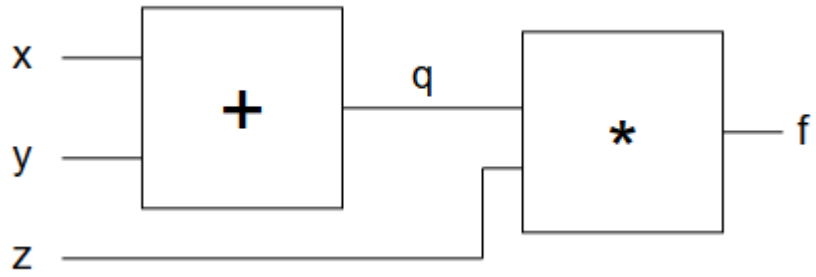
Osobine: spor, aproksimativan, lak za implementaciju.

Strategija #3: Analitički gradijent
Izvodimo direktan izraz za gradijent koji je jednostavan za procenu, kao što je i izlazna vrednost kola jednostavna za procenu. Nema potrebe za menjanjem vrednosti ulaza. Ne treba nam složena matematika kako bismo procenili gradijent, dovoljan nam je osnovni slučaj (*base case*). Izvodimo gradijent za veoma mali i jednostavan izraz, a potom slažemo uz pomoć pravila lanca (*Chain rule*) da bismo procenili ukupan gradijent.

$$f(x,y)=xy$$

$$\frac{\partial f(x,y)}{\partial x}=\frac{f(x+h,y)-f(x,y)}{h}=\frac{(x+h)y-xy}{h}=\frac{xy+hy-xy}{h}=\frac{hy}{h}=y$$

Osobine: brz, precizan, sklon greškama.



Slika 2: Primer složenijeg kola

Na slici 2 prikazano je kolo koje računa izraz:

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

Pretvaramo se da **+** kolo ne postoji, i posmatramo samo promenljive **q** i **z**. Pošto ne posmatramo ulaze **x** i **y**, imamo slučaj prostog kola za koji već znamo da odredimo izvod:

$$\frac{\partial f(q,z)}{\partial q}=z,\frac{\partial f(q,z)}{\partial z}=q$$

Pošto nam ne treba gradijent po **q**, već po **x** i **y**, **q** računamo kao funkciju od **x** i **y:**

$$\frac{\partial q(x,y)}{\partial x}=1,\frac{\partial q(x,y)}{\partial y}=1$$

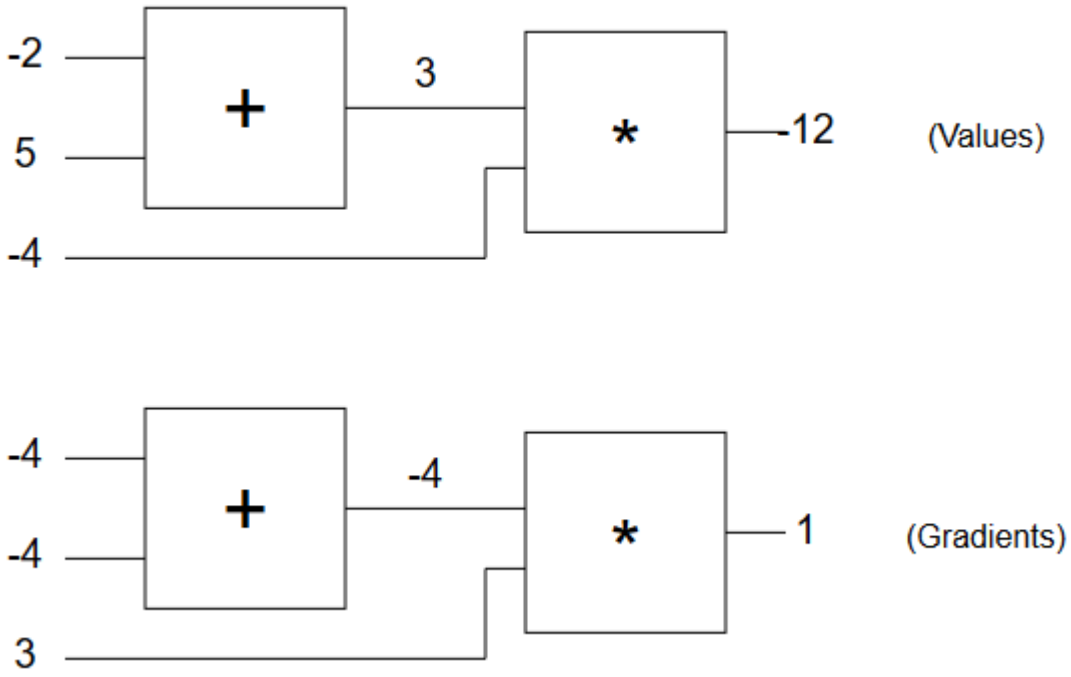
Oba izvoda su 1, bez obzira na prave vrednosti promenljivih, što i ima smisla, da se izlaz **+** kola povećava ako povećavamo ulazne promenljive u pozitivnom smeru.

Pravilo lanca(*Chain rule*) nam govori kako da kombinujemo gradijente kako bismo dobili konačan gradijent po **x** i **y**. Sve što treba da uradimo jeste da pomnožimo gradijente kako bismo ih pravilno ulančali. Tako će konačan gradijent po **x** biti:

$$\frac{\partial f(q,z)}{\partial x} = \frac{\partial q(x,y)}{\partial x}\frac{\partial f(q,z)}{\partial q}$$

Dakle, jedina razlika između slučaja sa prostim kolom i slučaja sa složenim kolom jeste dodatna operacija množenja.

Obrasci u backward flow-u
Posmatrajmo sledeće primere na slici 3.



Slika 3: Primer koji ilustruje *backward flow*

Prvi dijagram pokazuje vrednosti ulaza i izlaza, a drugi pokazuje gradijente koji se vraćaju u ulaze kao što je diskutovano. Na drugom dijagramu sa slike 3 mogu se uočiti obrasci protoka gradijenta. Npr. **+** kolo uvek uzima gradijent sa izlaza i prosledi ga na oba ulaza kola. Pošto je gradijent samog kola 1, bez obzira na vrednosti ulaznih promenljivih, gradijent se prosto prenosi na ulaze (tačnije, množi se sa 1 po pravilu lanca).

Primer: Jedan neuron
Sada posmatramo konkretan primer dvodimenzionalnog neurona koji računa sledeću funkciju:

$$f(x,y,a,b,c) = \sigma(ax+by+c)$$

U ovom izrazu, σ je *sigmoidna funkcija*. Ona uzima ulaznu promenljivu i sabija je na vrednost između 1 i 0, i to na sledeći način: veoma negativne vrednosti se sabijaju ka 0, a pozitivne ka 1. Sigmoidna funkcija se definiše na sledeći način:

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

gde znamo da je gradijent definisan sledećom formulom:

$$\frac{\partial \sigma(x)}{\partial x}=\sigma(x)(1-\sigma(x))$$

Npr. ako je ulaz u sigmoidnu funkciju x = 3, kolo će računati izlaz:

$$f=\frac{1}{1+e^{-x}}=0.95$$

a lokalni gradijent će biti:

$$dx=0.95(1-0.95)=0.0475$$

To je sve što nam treba da bismo koristili ovo kolo: znamo kako da propustimo ulaznu promenljivu kroz sigmoidno kolo, i takođe imamo izraz za gradijent po njegovim ulazima. Još treba napomenuti da je tehnički sigmoidna funkcija sastavljena od serije kola koja su redno vezana kako bi računala atomične funkcije: eksponencijalno kolo, **+** kolo i kolo deljenja.
Za rad jednog neurona, potrebno je definisati prosta kola koja računaju jednostavne, lokalne, parcijalne izvode po ulaznim promenljivima, povezati ih u graf, zatim uraditi *forward pass* za računanje izlazne vrednosti, a zatim *backward pass* koji će ulančati gradijente sve do ulaza.

Zadaci:

1. Otvoriti projekat **ComputationalGraph.sln**. Pre odrađivanja vežbe pogledati video tutorial-e na sledećem linku: https://www.youtube.com/playlist?list=PLBI8Ys-vopqbbP-XIj_L2BCdsWGxHxSSh
2. *TODO 1*: Implementirati metodu **forward**(...) u klasi **SumNode**. Metoda predstavlja forward korak sabirača i treba da vrati zbir svih vrednosti koje ulaze u sabirač.
3. *TODO 2*: Implementirati metodu **backward(...)** u klasi **SumNode**. Metoda predstavlja backward korak sabirača i treba da vrati vrednost izvoda funkcije po svakom elementu.
4. *TODO 3*: Implementirati metodu **forward(...)** u klasi **MultiplyNode**. Metoda predstavlja forward korak množača i treba da vrati proizvod svih vrednosti koje ulaze u množač, što je u našem slučaju proizvod dva elementa (ulaz i težina). Može se pretpostaviti da je ulazni parametar List<double> x uvek lista od dva elementa.
5. *TODO 4*: Implementirati metodu **backward(...)** u klasi **MultiplyNode**. Metoda predstavlja backward korak množača i treba da vrati vrednost izvoda funkcije po svakom elementu. Povratna vrednost metode će biti lista od dva elementa, gde će prvi element biti izvod funkcije po prvom elementu, a drugi će biti izvod funkcije po drugom elementu.
6. *TODO 5*: Implementirati metodu **sigmoid(...)** u klasi **SigmoidNode**. Metoda predstavlja forward korak sigmoidnog čvora i treba da vrati vrednost sigmoidne funkcije za prosleđeni x. Takođe, implementirati metodu **backward(...)** u klasi **SigmoidNode**. Metoda predstavlja backward korak i treba da vrati vrednost izvoda sigmoidne funkcije.

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

7. TODO 6: Izračunati vrednost koja treba da se nađe na izlazu neurona. Po McCulloch-Pitts modelu veštačkog neurona, ta vrednost je data izrazom:

   $$y=\sum_{i=1}^{n} x_i w_i + b\, w_b$$

   koju je na kraju potrebno propustiti kroz aktivacionu funkciju, a u ovom slučaju je to recimo sigmoidalna funkcija. Za potrebe množenja i saboranja **obavezno** koristiti već implementirani sabirač i množač i njihove **forward(...)** metode. Promenljiva **forSum** predstavlja listu svih proizvoda odgovarajućih ulaza i njima odgovarajućih težina. Promenljiva **summed** treba da predstavlja sumu svih proizvoda težina, odnosno y iz prethodne formule.
8. *TODO 7*: Izvršiti propagaciju signala u nazad, prvo kroz aktivacionu funkciju, onda kroz sabirač pa kroz svaki pojedinačan množač.
9. *TODO 8*: Implementirati metodu **updateWeights(…)** koja treba da izvrši korekciju težina u svim neuronima.