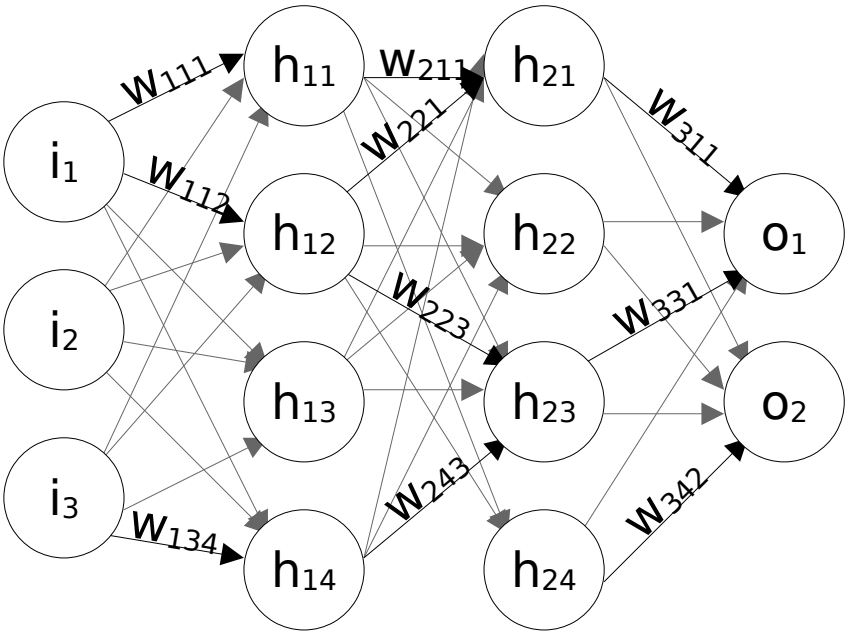
Vaja 11

Ime in priimek: ..... **Datum:**.....

Osnovni pojmi nevronskih mrež

Definicija in struktura nevronske mreže

*Nevronske mreže (NN ali ANN – (Artificial) Neural Networks) so modeli, ki so navdihnjeni z delovanjem nevronov v živalskih možganih. Nevronska mreža je sestavljena iz več nevronov, ki so med seboj povezani s povezavami oziroma utežmi. Zaradi računske učinkovitosti so nevroni zloženi v **plasti**. Prvi plasti pravimo **vhodna plast**, zadnji plasti pa **izhodna**. Vmesnim plastem pravimo **skrite plasti**. Vsak nevron ima neko realno številčno vrednost, ki je nek matematični izračun vrednosti nevronov iz prejšnje plasti. Izjema je vhodna plast (ki nima prejšnje plasti), ki vrednost dobi kot vhodne podatke. Vrednosti nevronov v izhodni plasti so tudi rezultati modela.*



Slika 1: Primer nevronske mreže s tremi vhodnimi in dvema izhodnima nevronoma ter dvema plastema skritih nevronov. Zaradi preglednosti vse povezave niso označene (te so narisane s sivo barvo).

*Na zgornji sliki vidimo primer nevronske mreže. Vsaka puščica kaže, od kod dobi dani nevron vrednost. Vsaki povezavi ustreza **utež**, ki nam pove, koliko moramo upoštevati dani nevron pri izračunu vrednosti naslednjega. Vsaka utež je na sliki označena s črko w in tremi indeksi. Prvi indeks pove, v kateri plasti se nahaja, drugi indeks pove, s katerega nevrona povezava jemlje vrednost, tretji pa, kateremu nevronu bomo vrednost pripisali.*

Bodite pozorni na povezave. Vhodni nevron i_1 je povezan samo z nevroni iz prve skrite plasti (h_{11}, h_{12}, h_{13} in h_{14}).

∴ **Primer** Poiščite povezavo w_{223} in dopolnite stavek. Nahaja se v plasti

_____, povezuje nevron _____ z nevronom _____.

Od katerih nevronov dobi vrednost nevron h_{22}?

Katerim nevronom posreduje svojo vrednost?

Kako dobijo nevroni svojo vrednost?
Nevron dobi svojo vrednost na sledeči način. Vrednost nevronov, ki so povezani z danim nevronom na vhodni strani (puščica obrnjena v dotični nevron) pomnožimo z ustrezno utežjo (vrednost na puščici) in zmnožke seštejemo. Tej vrednosti lahko dodamo tudi **odmik**, ni pa nujno. Celotna vsota še ni vrednost nevrona. Da bi dobili slednjo, moramo uporabiti še **aktivacijsko funkcijo** na vsoti. Zapišimo to matematično za primer nevrona h_{23}

$$h_{23} = a(w_{213}h_{11} + w_{223}h_{12} + w_{233}h_{13} + w_{243}h_{14} + b_{23})$$

ali splošneje

$$h_{ij} = a\left(\sum_k w_{ikj}h_{11} + w_{223}h_{i-1,k} + b_{ij}\right)$$

Linearnost in aktivacijska funkcija
Zakaj aktivacijsko funkcijo sploh potrebujemo? Zato da sistemu dodamo nekaj nelinearnosti in s temu dodamo sistemu kompleksnost. Oglejmo si to na našem primeru s slike 1. Recimo, da aktivacijske funkcije ne bi uporabljali. Vhodne nevrone lahko zložimo v matriko I 3×1, prvo skrito plast v matriko H_1 4×1, isto drugo skrito plast (H_2), izhodno pa v matriko O 2×1.
Tudi uteži zložimo v matrike in sicer prvo plast v matriko W_1 4×3. drugo W_2 4×4 in tretjo W_3 2×4. Če odmislimo odmike, vidimo, da lahko nevrone dobimo z matričnim množenjem (tudi odmike lahko dodamo k temu, če dodamo navidezen nevron z vrednostjo 1). Torej velja

$$H_1 = W_1 I, \quad H_2 = W_2 H_1, \quad O = W_3 H_2$$

To lahko združimo v matrično enačbo:

$$O = W_3 W_2 W_1 I = T I \,,$$

kjer smo vse matrike W pomnožili v enotno matriko T dimenzije 2×3. Brez aktivacijske funkcije je sistem linearen. To prvič pomeni, da lahkov principu vso kompleksnost parametrov v W matrikah (teh je 4×3+4×4+2×4=36) zreduciramo na zgolj 6 parametrov matrike T. Kot drugo pa to pomeni, da se bo sistem odzival sorazmerno in predvidljivo.

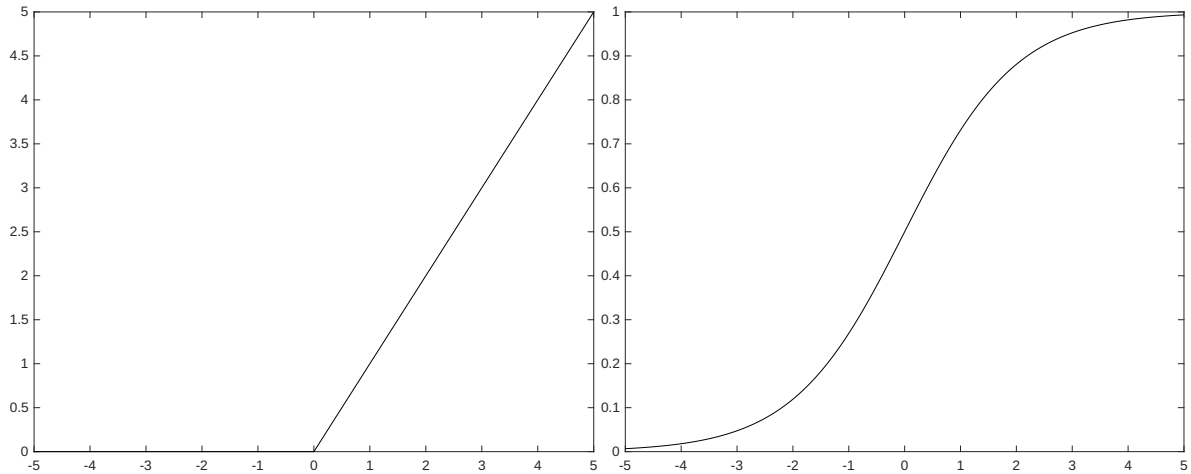
Torej nelinearno funkcijo nujno potrebujemo, da sistem ne postane trivialen. Po drugi strani pa bi premočna nelinearnost hitro privedla sistem v kaotičen (eksponentno občutljiv na začetne pogoje), česar pa spet ne želimo. Zato se v ta namen v večini uporablja ti dve aktivacijski funkciji: **ReLU** in **sigmoid**.

ReLU (Rectified Linear Unit) je preprosta funkcija, ki negativnim vrednostim pripiše 0, pozitivim pa samo sebe. To lahko zapišemo matematično kot

$$a(x) = \begin{cases} x & ; x > 0 \\ 0 & ; x \leq 0 \end{cases}.$$

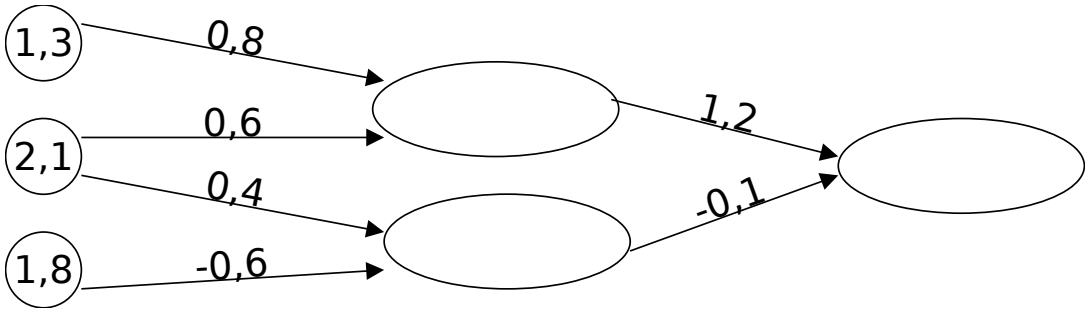
Sigmoid je nekoliko bolj zapletena funkcija. Njena zaloga vrednosti je med 0 in 1, je strogo naraščajoča, najbolj se spreminja v intervalu med -2 in 2. Njena definicija je

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



Slika 2: Graf funkcij ReLU (levo) in sigmoid (desno)

∴ **Primer** Dopolnite vrednosti v nevronski mreži, če vsi nevroni uporabljajo ReLU za aktivacijsko funkcijo. Odmikov ne računamo.



Strojno učenje

Doslej še nismo nič povedali, odkod dobijo uteži (in morebitni odmiki, v kolikor smo jih vključili) svoje vrednosti. V grobem imamo tri možnosti, ki jih bomo prikazali na primeru. Recimo, da nas zanima potovalni čas po neki ulici. Imamo podatek, koliko vozil v uri se pelje skozi, ki ga dobimo s pomočjo indukcijskih zank na začetku ceste. Drugi podatek, ki ga pa lahko nastavimo sami, pa je delovni cikel semaforja (delež časa, ko sveti zelena luč). Operater nato nastavi delovni cikel glede na potrebo ceste. Torej imamo dva vhodna parametra: število vozil na uro (ki ga je smiselno normalizirati, npr. deliti s 1000) in delovni cikel. Izhodni parameter je potovalni čas, saj je to podatek, ki ga želimo napovedati iz obeh vhodnih parametrov. Naj takoj povemo, da za take preproste primere, nevronske mreže niso najbolj primerno orodje, da bi jih obravnavali.

V prvem primeru nismo problema še nikoli obravnavali z nevronskimi mrežami. Zato najprej naredimo nekaj terenskih meritev, ko potovalni čas dejansko izmerimo. Torej bomo imeli za vsak par prometni tok - delovni cikel znan čas potovanja. Tem podatkom bomo rekli **učni podatki**. Naš cilj je nastaviti nevronsko mrežo tako, da bodo njene napovedi čim bolj usklajene z znanimi. Nastavitev pomeni določiti vrednosti uteži in odmikov. Temu postopku pravimo **strojno učenje**. Podrobnosti bomo opisali malo kasneje.

V drugem primeru smo podoben primer že uporabljali, konkretne ceste pa ne, ali pa so se zaradi spremenjenega prometnega režima (npr. otvoritev nove vpadnice) razmere toliko spremenile, da moramo mrežo ponovno nastaviti. V bistvu je ta primer matematično gledano identičen prejšnjemu, je pa zaradi boljšega izhodišča hitreje rešljiv

V tretjem primeru pa imamo nevronsko mrežo že naučeno (to je nastavljeno) in jo zgolj uporabimo.

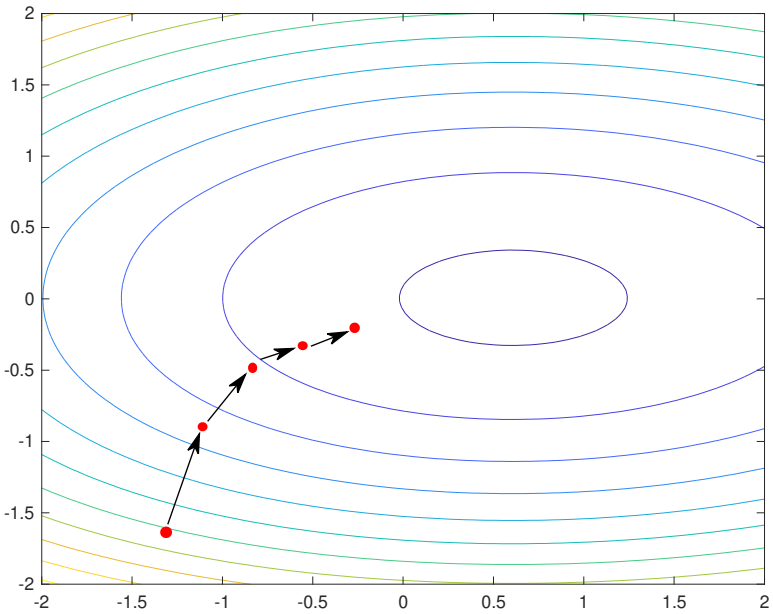
Kaj pomeni prilagoditev učnim podatkom z matematičnega zornega kota? Potrebujemo neko objektivno kvantitativno merilo, ki nam pove, koliko smo

blizu izmerjeni vrednosti. Če smo na primer izmerili potovalni čas 20s, naš model pa nam napove 25s, potem je lahko napaka modela absolutna vrednost 5s. Vendar nimamo samo enega učnega podatka. Po navadi jih imamo mnogo več. Zato je smiselno kot napako modela vpeljati povprečje absolutnih vrednosti razlik za vse učne podatke. Ni pa to edina možnost. Namesto absolutne vrednosti bi lahko vpeljali kvadrat razlike. Neglede na izbiro funkcije, pravimo tej funkciji **funkcija izgub**. Navedimo tri najpogosteje uporabljene primere funkcije izgub:

- povprečje absolutnih vrednosti razlik (L1 metrika),
- povprečje kvadratov razlik (L2 metrika),
- izguba križne entropije.

Razumevanje zadnje funkcije zahteva nekaj predznanja iz teorije informacij ali/in statistične mehanike, zato je ne bomo podrobneje opisali. Naj povemo le, da se jo pogosto uporablja pri klasifikaciji (delitev objektov v razrede).

Ko imamo enkrat izbrano funkcijo izgub, nam preostane „le“ še, da izračunamo njen minimum. To počnemo iterativno (postopoma, korak za korakom). Iskanje minimuma je podobno spuščanju v dolino (slika 3).



Slika 3: Grafični prikaz iskanja minimuma po korakih.

Vprašanje ostaja, v katero smer se moramo spuščati in za koliko naj se spustimo. Pri čemer imejmo v mislih, da je vsaka utež nastavljiv parameter v večdimenzionalnem prostoru za razliko od zemljevida, ki je dvodimenzionalni. Poleg tega nimamo vpogleda v celoten „zemljevid“ (to bi zahtevalo eksponentno preveč izračunov), ampak zgolj v bližnjo okolico. To slednje pomeni, da lahko analitično izračunamo le naklon in smer najstrmejšega spuščanja. Oboje dobimo iz **gradienta** funkcije izgube (na funkcijo izgube gledamo kot na funkcijo več spremenljivk – uteži modela). Ponovimo znanje iz matematike funkcije več spremenljivk. Gradient je vektor, katerega komponente so parcialni odvodi vsake od spremenljivk in kaže v smer največjega naraščanja (torej se moramo spustiti v nasprotni

smeri gradienta). Vendar se ne spustimo za celotno velikost gradienta, ampak le za delček slednje. Deležu koraka, ki ga pri tem opravimo, pravimo **krivulja učenja**. Če je ta prevelika, lahko pristanemo na drugi strani doline, če pa je premajhna, moramo narediti več korakov za isti učinek in s tem zmanjšujemo učinkovitost algoritma.

Matematično gledano, naredimo ob vsaki iteraciji sledečo spremembo:

$$w_n^{\text{nov}} = w_n^{\text{star}} - r\frac{\partial l}{\partial w_n},$$

kjer je w_n n-ta utež, r krivulja učenja in l funkcija izgub. Ravno pri izračunu gradienta se pozna prednost strukturiranja nevronskih mrež v plasti. Oglejmo si to na primeru mreže na sliki 1. Uporabili bomo funkcijo izgub L2.

$$l = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{2}\left(o_j^{(i)} - t_j^{(i)}\right)^2,$$

kjer so $o_j^{(i)}$ napovedi modela, $t_j^{(i)}$ podatki iz učnega seta in N število podatkov v učnem setu.

∴ **Primer:** Izračunajte parcialne odvode l po utežeh $w331$, $w223$ in $w112$. Izračunajte jih v tem vrstnem redu! Namenoma je izbrana po vsaka utež iz ene plasti.

Na zgornjemu primeru smo videli, da se vse parcialne odvode da dobiti s kombinacijo napake in vrednosti nevronov v plasteh za utežjo (proti izhodni smeri). Zato takemu načinu izračuna gradienta pravimo **povratno širjenje** napake (backpropagation).

Beležke

Literatura

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville, **Deep learning**, MIT Press, ZDA, 2022.