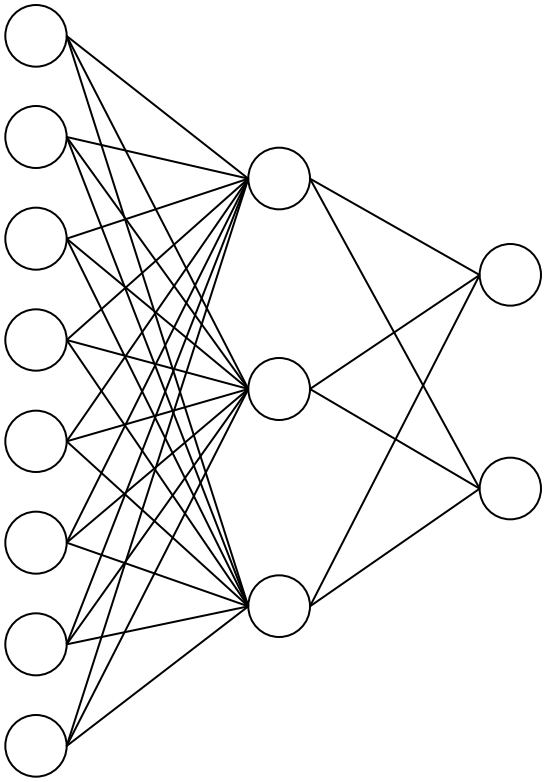
Osnove neuronskih mreža

(*Fundamentals of Neural Networks*)

Umjetna neuronska mreža (*artifical neural network*) je programski ili sklopovski sustav građen od jednakih, jednostavnih i međusobno povezanih dijelova (čvorova) koji može obavljati raznovrsne i složene zadaće ovisno o parametrima dijelova i veza među njima. Dijelovi (čvorovi) neuronske mreže nazivaju se neuroni (*neurons*).



Građa umjetnih neuronskih mreža je slojevita i nalikuje na (točnije, oponaša) biološka živčana tkiva pa im otuda i naziv.

U tehničkim disciplinama, gdje se "umjetnost" mreže podrazumijeva, najčešće se govori samo *neuronska mreža*.

Ako signali (podatci) u mreži putuju samo u jednom smjeru, od ulaza prema izlazu, radi se o *unaprijednoj* (*feedforward*) mreži. Postoje i složenije strukture u kojima signali putuju u oba smjera i u kojima ima i povratnih veza, ali to su specijalizirane mreže (automatika, obrada signala) kojima se u okviru ovog predmeta nećemo baviti ... ☺

Neuronske mreže mogu se primijeniti u raznorodnim problemima, na primjer:

1. aproksimiranje funkcija: one koje je teško izraziti formulama (to je zapravo preslikavanje domene u kodomenu; *mapping*); regresijska analiza
2. razvrstavanje (*classification*) podataka i prepoznavanje uzoraka, odnosno neobičnosti (izuzetaka)
3. obrada signala i podataka, npr. komplicirano filtriranje ili upravljanje procesima (automatika)

Razlikujemo tri osnovna sloja u mreži: ulazni, skriveni i izlazni sloj, pri čemu skriveni sloj može imati proizvoljno podslojeva (razina).

Broj neurona u pojedinom sloju ovisi o zadaći mreže:

- ulazni sloj ima onoliko neurona koliko je ulaznih varijabli (= dimenzija domene)
- izlazni sloj ima onoliko neurona koliko je željenih izlaznih rezultata (= dimenzija kodomene)
  - na primjer, ako samo treba ustanoviti je li na ulazu neki uzorak ili nije, dovoljan je jedan neuron na izlazu (rezultat DA-NE). Ako, pak, treba filtrirati signal, izlaz treba imati onoliko neurona koliko izlazni signal treba imati točaka.
- ukupni broj neurona u skrivenom sloju mreže s jednim izlazom uglavnom je unutar 2...30 % broja ulaznih varijabli, pri čemu razine u skrivenom sloju redovito imaju sve manje i manje neurona što je razina bliža izlazu

Manji potrebni broj neurona u skrivenom sloju intuitivno se može razumjeti ako se neuronska mreža shvati posve apstraktno kao struktura koja dodjeljuje vrijednosti svakoj točki u *N*-dimenzionalnom prostoru (*hiperprostor*) ulaznih varijabli (pojedinim područjima (djelićima) tog prostora).

Preslikavanje bi se, teorijski, moglo definirati pohranom željenih izlaznih vrijednosti za sve točke tog prostora (npr. u nekakvu *lookup* tablicu), ali čak i kad bismo znali vrijednosti u svim tim točkama, a redovito ih ne znamo, previše ih je za pohranu.

- Na primjer, za samo 10 ulaznih varijabli koje mogu poprimiti samo 10 različitih vrijednosti prostor već ima 10^{10} točaka!

Neuronska mreža "simulira" tabličnu definiciju preslikavanja i izravno izračunava izlaz za svaku moguću kombinaciju vrijednosti ulaznih parametara.

Budući da za “omeđivanje” jednog djelića N-dim. prostora trebamo $2N$ točaka, identifikacija jednog djelića zahtijeva $2N$ prilagodljivih parametara. Ako je ta identifikacija zadaća neuronske mreže, onda je $2N$ ujedno i najmanji potrebni broj njezinih prilagodljivih parametara. Ako je, nadalje, mreža troslojna, potpuno povezana i ima N ulaznih varijabli, slijedi da skriveni sloj, načelno, može imati samo dva neurona.

Identifikaciji doprinose i kombinacije neurona u skrivenom sloju te njihove veze s drugim razinama skrivenog sloja i izlaznim slojem pa ovisnost u konačnici nije tako jednostavna.

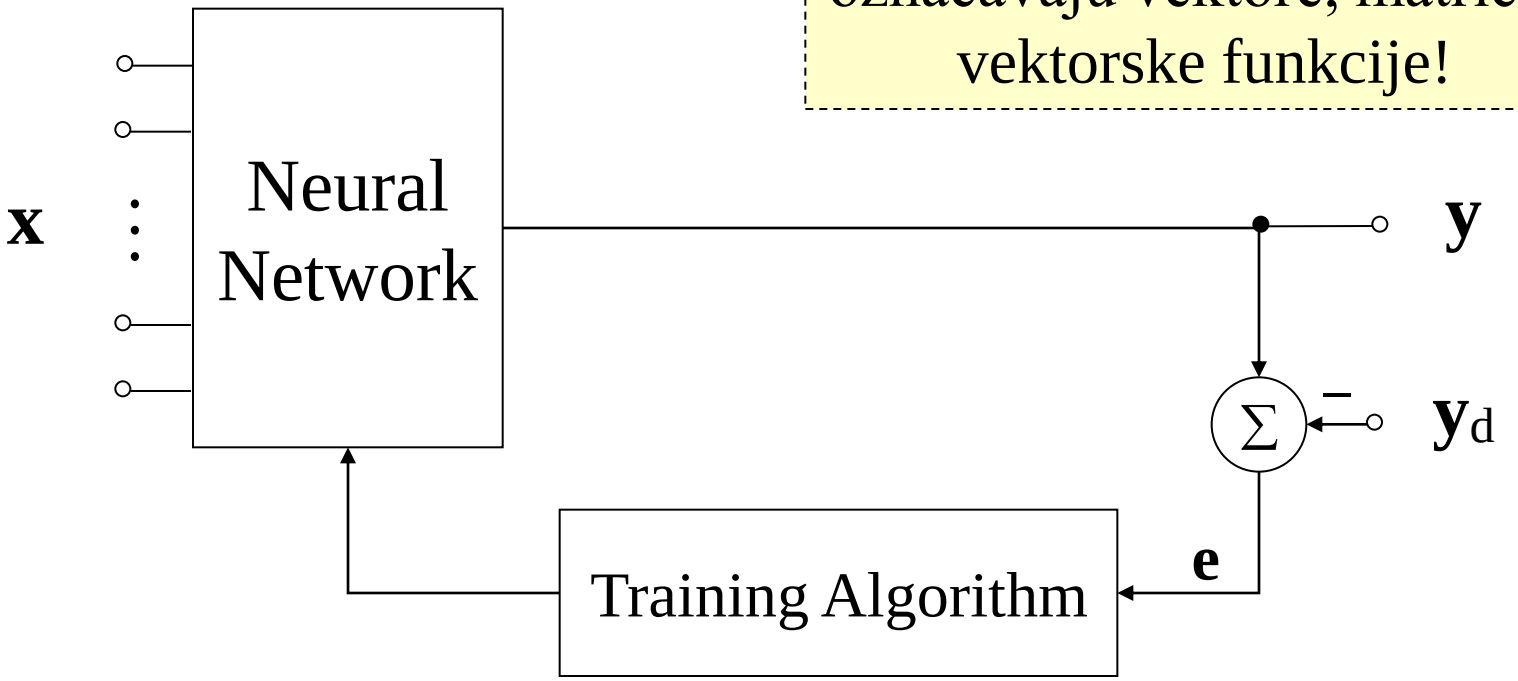
Broj potrebnih prepoznatljivih djelića prostora ovisi o zadaći mreže, ali prethodno razmatranje intuitivno objašnjava zašto je broj neurona u skrivenom sloju redovito manji od broja ulaznih varijabli ...

Izlaz mreže je odraz (“slika”) uzlaza izmijenjena u skladu s definicijom preslikavanja koja je pohranjena u parametrima mreže pa se neuronska mreža može shvatiti kao mapa (“šablona”) po kojoj se na temelju ulaza određuje izlaz. Neuroni u mreži djeluju jedni na druge šaljući svoje izlazne signale na ulaze susjednih neurona, ali svaki neuron u mreži “odlučuje” samostalno, tj. jednom kad primi signale (“mišljenja”) susjednih neurona, svoj izlaz namješta isključivo na temelju svojih parametara. Pri tome svi neuroni istog sloja odluku donose istodobno pa neuronska mreža djeluje kao “paralelni obrađivač” (procesor).

Da bi mreža mogla pravilno preslikavati ulaz u izlaz, moramo joj “objasniti” što želimo, tj. “naučiti ju” kako obaviti željeno preslikavanje. Postoji više načina “podučavanja”, odnosno “uvježbavanja” mreže, a najizravniji je dati joj primjere pravilnog rada.

Neka je $\boldsymbol{f}$: $R^n \rightarrow R^m$ preslikavanje (mapa) koju želimo postići. Uvježbavanje mreže je proces tijekom kojeg odgovarajućim algoritmom ugađamo parametre mreže na temelju razlike između njezinog stvarnog izlaznog signala (rezultata) **y** i željenog izlaznog signala $\mathbf{y}_d = \boldsymbol{f}(\mathbf{x})$.

Podebljana (*bold*) slova označavaju vektore, matrice i vektorske funkcije!



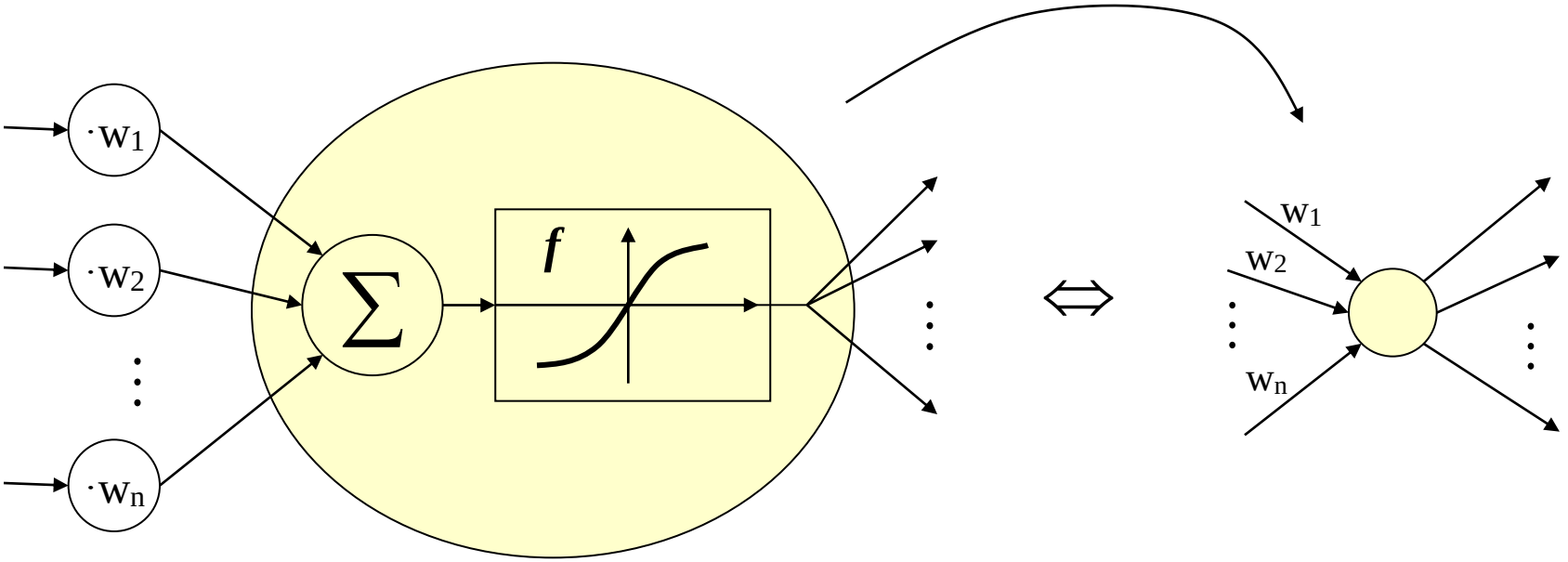
Dakle, mreži dajemo neke od točaka mape ***f*** kao primjere željenog preslikavanja, konkretno neki skup od *n* parova $(x_{d,1}, y_{d,1})$, ... , $(x_{d,n}, y_{d,n})$ koji nazivamo *skup za uvježbavanje* (*training set*), i na temelju tog ograničenog skupa primjera određujemo parametre mreže koji će ju osposobiti za predviđene zadaće.

Nakon uvježbavanja očekujemo (!) od mreže pravilno preslikavanje i onih kombinacija ulaznih varijabli koje nisu bile u skupu za uvježbavanje. Prema tome, svrha uvježbavanja mreže, tj. njezina "učenja", jest njezino "osamostaljivanje" u radu.

Budući da u opisanom načinu učenja mreža ima "učitelja", taj se pristup naziva *nadzirano učenje* (*supervised learning*). Postojanje tog naziva daje naslutiti da postoji i nenadzirano (*unsupervised*) učenje ... ☺

Uvježbavanje jednog neurona

Neuron je “mapa” preslikavanja strukture kao na slici, dakle više ulaza i samo jedan izlaz, ali koji se može granati. Izlaz neurona je funkcija *f* zbroja njegovih ulaza, a kako se izlaz često naziva i *aktivnost* (*activity*), izlazna funkcija se naziva *aktivacijska funkcija* (*activation function*).



S obzirom na građu, neuron obavlja preslikavanje $\mathrm{R}^n \to \mathrm{R}$. Označimo li njegov izlaz s y, a ulazne varijable s x_i, za $f(z) = z$ (tj. izlazne funkcije nema; *Adaline* = *adaptive linear element*) preslikavanje je

$$y = f\left(\sum_{i=1}^{n} w_i x_i\right) = \sum_{i=1}^{n} w_i x_i \quad .$$

Uobičajeno je podatke organizirati u stupčane vektore u skladu s prikazom mreže, što znači da će ulazi x_i i težine w_i činiti stupce $\mathbf{x} = [x_1 \;\; x_2 \;\; \ldots \;\; x_n]^{\mathrm{T}}$ i $\mathbf{w} = [w_1 \;\; w_2 \;\; \ldots \;\; w_n]^{\mathrm{T}}$. Izlaz se tada može izračunati kao skalarni umnožak vektora

$$y = \sum_{i=1}^{n} w_i x_i = \mathbf{w}^{\mathrm{T}}\mathbf{x} = \mathbf{x}^{\mathrm{T}}\mathbf{w} \quad .$$

Napomena: u statistici je najčešći upravo obrnut zapis, tj. varijable koje čine jedan uzorak (ulaz) se smještaju u isti redak, a uzorci se nižu jedan ispod drugog. No, takav zapis ne odražava intuitivnu sliku neuronske mreže, stoga ćemo koristiti zapis koji smo upravo uveli.

Uvježbavanje takvog neurona s p parova za vježbu $(\mathbf{x}_{\mathrm{d},1}, y_{\mathrm{d},1}), \ldots, (\mathbf{x}_{\mathrm{d},p}, y_{\mathrm{d},p})$ je problem oblika

$$\text{minimizirati} \quad \frac{1}{2}\sum_{i=1}^{p}\left(\mathbf{x}_{\mathrm{d},i}^{\mathrm{T}}\mathbf{w} - y_{\mathrm{d},i}\right)^2$$

pri čemu se minimizacija provodi s obzirom na vektor $\mathbf{w} = [w_1 \ldots w_n]^{\mathrm{T}} \in \mathrm{R}^n$.

Izraz koji treba minimizirati je zapravo zbroj kvadrata pogrešaka mreže u svim točkama (parovima) za uvježbavanje, dakle radi se o kriteriju najmanjih kvadrata (LMS = *least mean squares*).

$$\text{minimizirati} \quad \frac{1}{2}\sum_{i=1}^{p}(y_i - y_{\mathrm{d},i})^2 = \frac{1}{2}\sum_{i=1}^{p}e_i^2$$

Ako p uzoraka (parova za vježbu) smjestimo u matricu $\mathbf{X}_\mathrm{d} = [\mathbf{x}_1 \ \ldots \ \mathbf{x}_p] \in \mathrm{R}^{n \times p}$ i vektor $\mathbf{y}_\mathrm{d} \in \mathrm{R}^{p \times 1}$

$$\mathbf{X}_\mathrm{d} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \quad \text{i} \quad \mathbf{y}_\mathrm{d} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix},$$

problem se može zapisati kao

$$\text{minimizirati} \quad \frac{1}{2}\left\|\mathbf{X}_\mathrm{d}^\mathrm{T}\mathbf{w} - \mathbf{y}_\mathrm{d}\right\|^2 = \frac{1}{2}\|\mathbf{e}\|^2 \ .$$

Izraz koji treba minimizirati je zapravo kvadrat norme vektora pogrešaka $\mathbf{e} = [e_1 \ \ e_2 \ \ \ldots \ \ e_p]^\mathrm{T}$.

S obzirom na to da minimiziramo kvadrat nekog izraza, jasno je da će teorijski minimum biti nula, dakle cilj je postići da taj izraz, u ovom slučaju norma vektora pogrešaka, postane nula.

Prema tome, optimalne težine w_i jesu rješenje sustava $\mathbf{X}_d^T \cdot \mathbf{w} = \mathbf{y}_d$ od p jednadžbi s n nepoznanica ili raspisano

$$
\begin{aligned}
x_{11} \cdot w_1 + x_{12} \cdot w_2 + \ldots + x_{1n} \cdot w_n &= y_1 \\
x_{21} \cdot w_1 + x_{22} \cdot w_2 + \ldots + x_{2n} \cdot w_n &= y_2 \\
\ldots \qquad\qquad\qquad \ldots & \\
x_{p1} \cdot w_1 + x_{p2} \cdot w_2 + \ldots + x_{pn} \cdot w_n &= y_p \qquad ,
\end{aligned}
$$

tj. želimo savršeno uvježbati neuron tako da bude $\mathbf{y} = \mathbf{y}_d$ u svim točkama za uvježbavanje, ako je to uopće moguće.

Općenito, rješenje sustava $\mathbf{Ax} = \mathbf{b}$ je $\mathbf{x} = \mathbf{A}^{-} \cdot \mathbf{b}$, gdje je $\mathbf{A}^{-}$ opći inverz (*generalized inverse*) matrice $\mathbf{A} \in \mathrm{R}^{p \times n}$. Opći inverz je svaka matrica koja zadovoljava relaciju

a) $\mathbf{A}\mathbf{A}^{-}\mathbf{A} = \mathbf{A}$

i takva matrica uvijek postoji, ali nije uvijek jedinstvena. Zato se gotovo uvijek koristi takozvani Moore-Penrose pseudoinverz koji jest jedinstven, a to je matrica koja zadovoljava još sljedeća tri dodatna uvjeta:

b) $\mathbf{A}^{-}\mathbf{A}\mathbf{A}^{-} = \mathbf{A}^{-}$

c) $(\mathbf{A}\mathbf{A}^{-})^{\mathrm{T}} = \mathbf{A}\mathbf{A}^{-}$

d) $(\mathbf{A}^{-}\mathbf{A})^{\mathrm{T}} = \mathbf{A}^{-}\mathbf{A}$.

Moore-Penrose pseudoinverz se najčešće označava s $\mathbf{A}^{+}$, ali dosta je zastupljena i oznaka $\mathbf{A}^{\dagger}$.

Prilikom rješavanja sustava $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A} \in \mathrm{R}^{p \times n}$ i $\mathbf{A}$ je punog ranga, moguća su tri slučaja:

1. p = n : rješenje je jedinstveno i nema optimizacije

2. p < n: rješenja ima beskonačno i uobičajeno je odabrati ono najmanje norme $\|\mathbf{x}\|$, a to je upravo rješenje $\mathbf{x} = \mathbf{A}^{+} \cdot \mathbf{b}$, pri čemu se pseudoinverz $\mathbf{A}^{+}$ računa po formuli (*desni* pseudoinverz)

$$pseudoinverz = \mathbf{A}^{+} = \mathbf{A}^{\mathrm{T}} \left(\mathbf{A}\mathbf{A}^{\mathrm{T}} \right)^{-1}$$

što primijenjeno na naš problem ($\mathbf{A} = \mathbf{X}_{\mathrm{d}}^{\mathrm{T}} \in \mathrm{R}^{p \times n}$) daje

$$\mathbf{w}^{*} = \mathbf{X}_{\mathrm{d}} \left(\mathbf{X}_{\mathrm{d}}^{\mathrm{T}} \mathbf{X}_{\mathrm{d}} \right)^{-1} \mathbf{y}_{\mathrm{d}} \quad .$$

Do istog rješenja, ali bez potrebe za invertiranjem matrice $\mathbf{X}_\mathrm{d}^\mathrm{T} \cdot \mathbf{X}_\mathrm{d}$, vodi Kaczmarzov algoritam:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \mu \frac{e^{(k)} \mathbf{x}_{\mathrm{d},R(k)+1}}{\left\| \mathbf{x}_{\mathrm{d},R(k)+1} \right\|^2} \quad , \quad k = 0, 1, 2, \ldots$$

gdje su: $\mathbf{w}^{(0)} = \mathbf{0}$,

$$e^{(k)} = \mathbf{x}_{\mathrm{d},R(k)+1}^\mathrm{T} \mathbf{w}^{(k)} - y_{\mathrm{d},R(k)+1} \quad ,$$

$$0 < \mu < 2$$

i R(k) ostatak dijeljenja k s p, tj. R(k) = k mod p.

Nedostatak Kaczmarzovog algoritma je taj da je traženi $\mathbf{w}^*$ tek limes za $k \to \infty$.

3. $p > n$: točno rješenje može, ali i ne mora postojati.
   Izraz $\mathbf{x} = \mathbf{A}^{+} \cdot \mathbf{b}$ je jednak točnom rješenju ako ono postoji, a ako ne, to je optimalno rješenje po kriteriju najmanjih kvadrata odstupanja, odnosno po kriteriju najmanje norme $\|\mathbf{b} - \mathbf{A}\mathbf{x}\|$.

Pseudoinverz A^{+} se računa po formuli (*lijevi* pseudoinv.)

$$pseudoinverz = \mathbf{A}^{+} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathrm{T}}$$

pa slijedi (za $\mathbf{A} = \mathbf{X}_{\mathrm{d}}^{\mathrm{T}} \in \mathrm{R}^{p \times n}$)

$$\mathbf{w}^{*} = \left(\mathbf{X}_{\mathrm{d}}\mathbf{X}_{\mathrm{d}}^{\mathrm{T}}\right)^{-1}\mathbf{X}_{\mathrm{d}}\mathbf{y}_{\mathrm{d}} \quad .$$

Ako neuron ima nelinearnu aktivacijsku funkciju, nema formule za izravno izračunavanje rješenja, nego primjenjujemo razne optimizacijske (iteracijske) algoritme kojima se postupno približavamo optimalnom rješenju.

Na primjer, za $p < n$ i aktivacijsku funkciju f, primjenjujemo Kaczmarzov algoritam koji ostaje jednak, samo se mijenja izraz za pogrešku $e^{(k)}$.

$$e^{(k)} = f\left(\mathbf{x}_{\mathrm{d},R(k)+1}^{\mathrm{T}}\mathbf{w}^{(k)}\right) - y_{\mathrm{d},R(k)+1}$$

Za $p > n$ možemo primijeniti bilo koju optimizacijsku metodu, npr. gradijentnu, Newtonovu, metodu konjugiranih smjerova itd. Najjednostavnija je gradijentna pa ćemo ju i mi koristiti u svim primjerima.

Gradijentna metoda optimizacije

$$f(\mathbf{x}) \quad , \qquad f : \mathrm{R}^n \rightarrow \mathrm{R}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \cdot \nabla f\left(\mathbf{x}^{(k)}\right) \qquad ; \quad \alpha_k > 0$$

Ciljna funkcija za *Adaline*: $$f(\mathbf{w}) = \frac{1}{2}\left\|\mathbf{X}_\mathrm{d}^\mathrm{T}\mathbf{w} - \mathbf{y}_\mathrm{d}\right\|^2$$

Gradijent ciljne funkcije: $$\nabla f(\mathbf{w}) = \mathbf{X}_\mathrm{d}\left(\mathbf{X}_\mathrm{d}^\mathrm{T} \cdot \mathbf{w} - \mathbf{y}_\mathrm{d}\right)$$
$$= \mathbf{X}_\mathrm{d} \cdot \mathbf{e}$$

⇩

Iteracija: $$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \cdot \mathbf{X}_\mathrm{d} \cdot \mathrm{e}^{(k)} \quad ,$$

gdje je $$\mathrm{e}^{(k)} = \mathbf{X}_\mathrm{d}^\mathrm{T} \cdot \mathbf{w}^{(k)} - \mathbf{y}_\mathrm{d} \quad .$$ (vektor pogreška *k*-tog rješenja)

Gradijentna metoda zahtijeva izračunavanje gradijenta ciljne funkcije $E(\mathbf{w})$, u našem slučaju

$$\nabla E(\mathbf{w}) = \nabla \frac{1}{2} \sum_{i=1}^{p} e_i^2 = \frac{1}{2} \sum_{i=1}^{p} \nabla e_i^2 = \sum_{i=1}^{p} e_i \cdot \nabla e_i \quad .$$

Gornji izraz pokazuje da se točan gradijent može odrediti tek nakon izračunavanja gradijenta ∇e_i svake pojedine pogreške (u svakoj točki skupa za uvježbavanje).

Prema tome, ispravljanje parametara bi bilo moguće tek nakon što se neuronu "predoči" cijeli skup za uvježbavanje pa se takvo "podučavanje" naziva **skupno uvježbavanje** (*Batch Learning*).

Jedno takvo izračunavanje gradijenta (prolazak svim uzorcima) naziva se *iteracija* ili *epoha*.

No, postoji i druga mogućnost za koju ideja slijedi iz

$$\nabla E(\mathbf{w}) = \nabla \frac{1}{2}\sum_{i=1}^{p} e_i^2 = \frac{1}{2}\sum_{i=1}^{p} \nabla e_i^2 = \frac{1}{2}\cdot p \cdot \left(\nabla e_i^2\right)_{mean}$$

$$= p \cdot \left(e_i \cdot \nabla e_i\right)_{mean} \approx p \cdot e_i \cdot \nabla e_i \qquad .$$

Ukupni gradijent kvadrata norme vektora pogrešaka je $p\cdot$(prosječni gradijent) pa kad bi za svaki par za uvježbavanje umnožak $e_i \cdot \nabla e_i$ bio približno jednak, ispravak parametara nakon "učenja" u jednoj točki

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha'_k \cdot 1/2 \cdot \nabla e^2_i = \mathbf{w}^{(k)} - \alpha_k \cdot e_i \cdot \nabla e_i$$

bio bi $1/p$ skupnog (ukupnog) ispravka. To, naravno, nije posve točno, ali postupak je djelotvoran i naziva se **koračno uvježbavanje** (*on-line learning*).

Ako se radi o jednom neuronu bez aktivacijske funkcije (*Adaline*), pogreška je $e_i = \mathbf{x}^{\mathrm{T}}_{\mathrm{d},i}\mathbf{w} - y_{\mathrm{d},i}$, njezin gradijent $\nabla e_i = \mathbf{x}_{\mathrm{d},i}$ i formula za ispravak vektora parametara postaje

$$\mathbf{w}^{(\mathrm{k}+1)} = \mathbf{w}^{(\mathrm{k})} - \alpha_{\mathrm{k}} \cdot e_i^{(\mathrm{k})} \cdot \mathbf{x}_{\mathrm{d},i} \ ,$$

odnosno ako se koraci broje kao i točke za uvježbavanje,

$$\mathbf{w}^{(\mathrm{k}+1)} = \mathbf{w}^{(\mathrm{k})} - \alpha_{\mathrm{k}} \cdot \mathbf{x}_{\mathrm{d},k} \cdot e_k \quad .$$

Dobiveni izraz opisuje koračno uvježbavanje jednog neurona koje se temelji na minimizaciji kvadrata trenutačne pogreške (u jednoj točki), stoga se algoritam uvježbavanja koji proizlazi iz primjene dobivene formule naziva *algoritam najmanjih kvadrata* (LMS *Algorithm*).

Ovo je bio vrlo pojedonstavnjen izvod! Strogi dokaz ispravnosti (i konvergencije) LMS algoritma polazi od pretpostavke *ergodičnosti* okoline...

Dakle, dva osnovna načina uvježbavanja jesu:

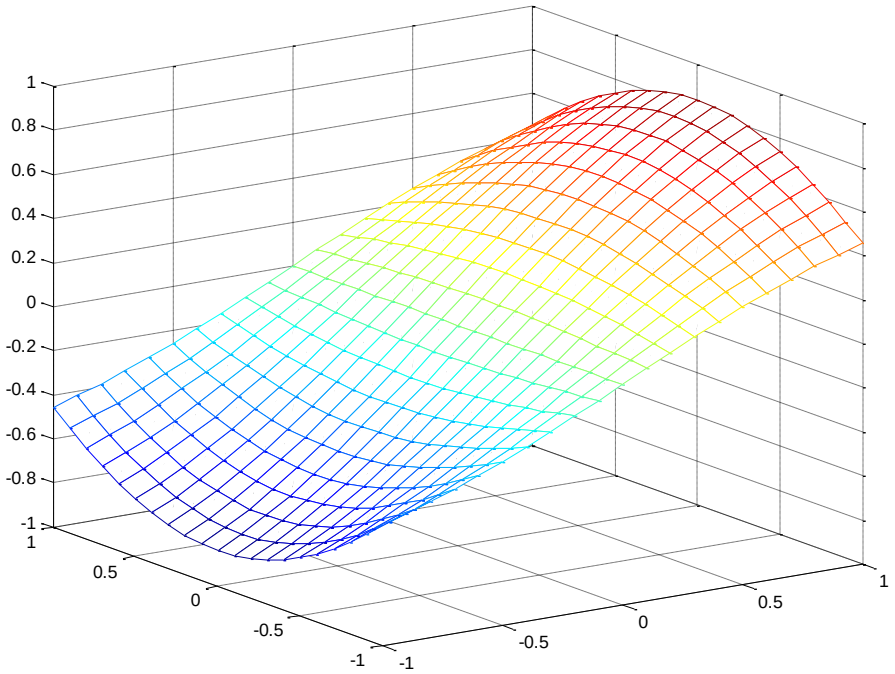
1) **skupno** uvježbavanje (*batch learning*) – gradijent se izračunava na temelju svih točaka za uvježbavanje i tek tada se osvježavaju parametri mreže (*epoha po epoha*)
   - memorijski i računski zahtjevnije
   - sigurna i mirnija, ali često sporija konvergencija
2) **koračno** uvježbavanje (*on-line learning*) – gradijent se izračunava i parametri mreže osvježavaju u svakoj pojedinoj točki skupa za uvježbavanje (LMS algoritam)
   - konvergencija s oscilacijama (Brownovo gibanje)
   - postupak uvježbavanja bolje prati male promjene podataka (lokalne promjene u prostoru) $\Rightarrow$ lakše izvlačenje iz okoline lokalnog minimuma
   - ponovljeni podatci se bolje iskorištavaju (utjecajniji)

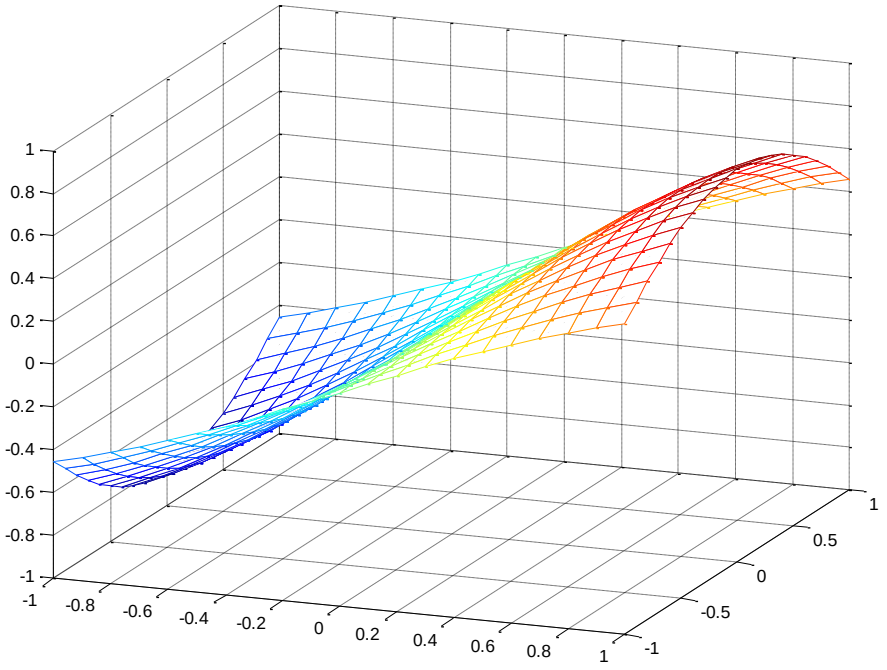
Primjer: uvježbavanje jednog neurona s dva ulaza i bez aktivacijske funkcije (*Adaline*)

$f: \mathrm{R}^2 \rightarrow \mathrm{R}$; $f(x_1, x_2) = \sin(x_1) \cdot \cos(x_2)$

Domena: $x_1, x_2 \in [-1,1]$.

SingleNeuron.m



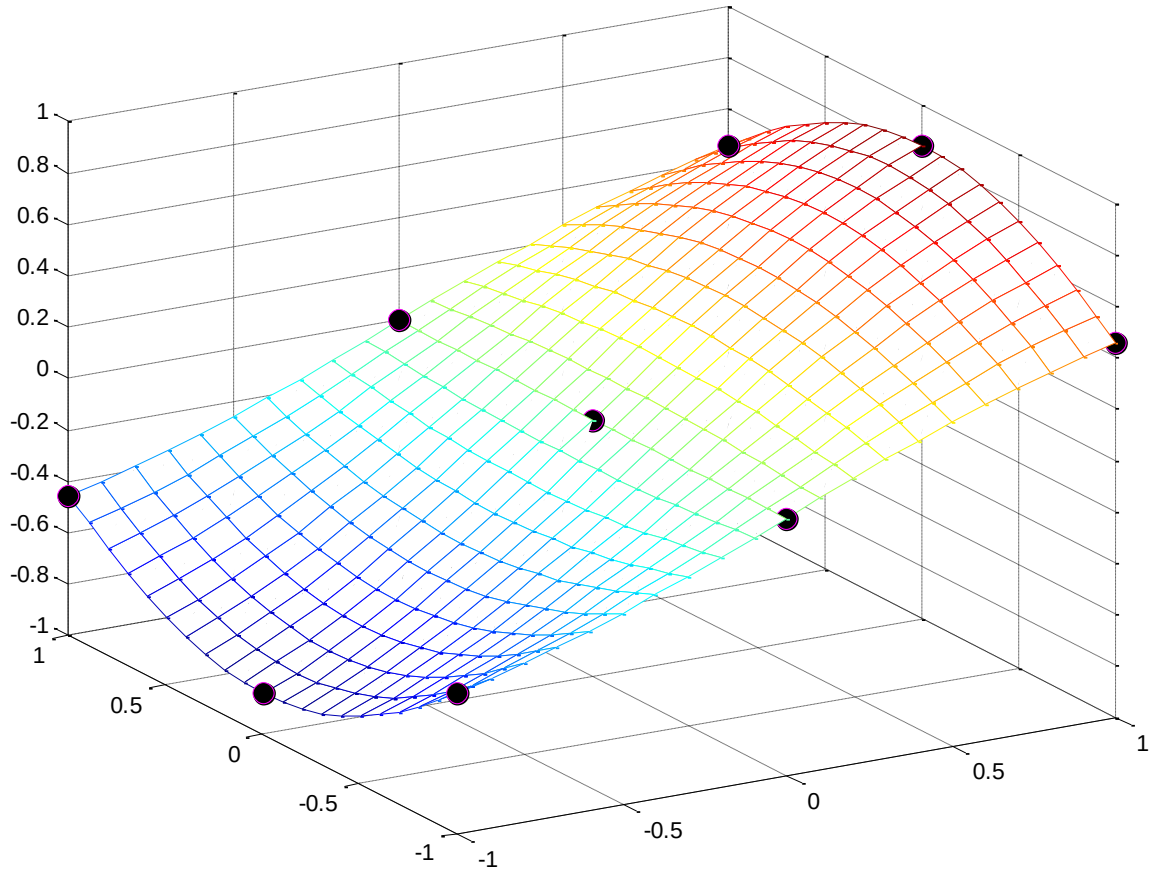


Skup za uvježbavanje: 9 točaka, argumenti funkcije neka su

$$\mathbf{x}_i = [x_{1i} \;\; x_{2i}]^{\mathrm{T}} \quad ; \quad x_1, x_2 \in \{-1, 0, 1\}.$$

Dakle, $\mathbf{X}_{\mathrm{d}} \in \mathrm{R}^{2\times 9}$ i $\mathbf{y}_{\mathrm{d}} \in \mathrm{R}^{9\times 1}$, a cilj je naći vektor $\mathbf{w}$ koji će minimizirati funkciju

$$\frac{1}{2}\left\|\mathbf{X}_d^{\mathrm{T}}\mathbf{w} - \mathbf{y}_d\right\|^2 .$$



Tražimo (najbolje moguće) rješenje sustava $\mathbf{X}_d^{\mathrm{T}} \cdot \mathbf{w} = \mathbf{y}_d$. Rješenje je uvijek $(\mathbf{X}_d^{\mathrm{T}})^{-} \cdot \mathbf{y}_d$, a kako je broj jednadžbi veći od broja nepoznanica, računamo lijevi pseudoinverz, odnosno rješenje je

$$\mathbf{w} = \left(\mathbf{X}_d \mathbf{X}_d^{\mathrm{T}}\right)^{-1} \mathbf{X}_d \mathbf{y}_d = \begin{bmatrix} 0{,}5836 \\ 0 \end{bmatrix} \quad .$$

Do istog rezultata (postupno) dolazimo i gradijentnom metodom. Gradijent ciljne funkcije za *Adaline* je $\mathbf{X}_d(\mathbf{X}_d^{\mathrm{T}} \cdot \mathbf{w} - \mathbf{y}_d)$ pa gradijentna metoda vodi u iteraciju

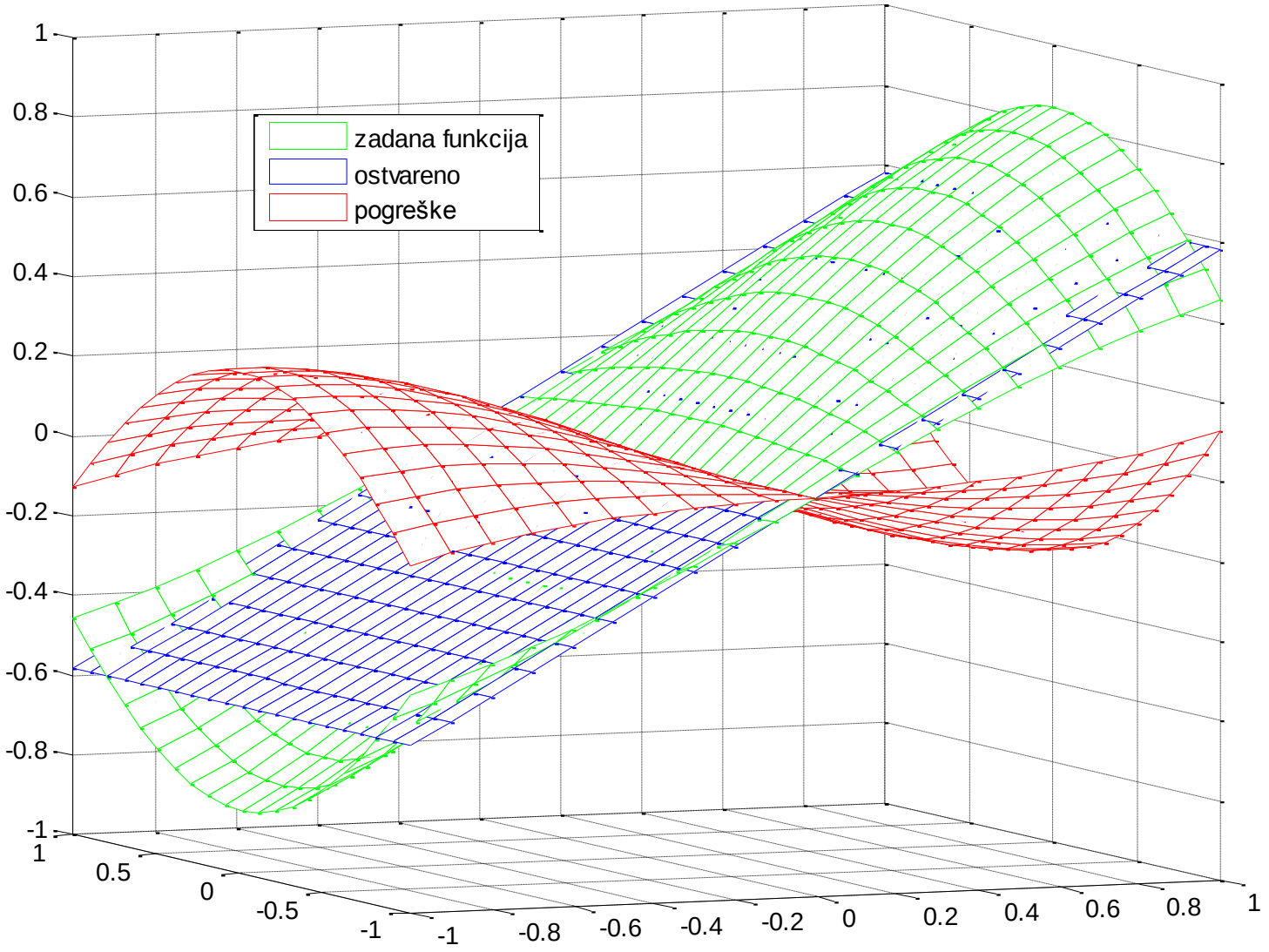
$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \cdot \mathbf{X}_d \cdot \mathrm{e}^{(k)} \quad , \quad \text{(skupno uvježbavanje!)}$$

gdje je $\mathrm{e}^{(k)} = \mathbf{X}_d^{\mathrm{T}} \cdot \mathbf{w}^{(k)} - \mathbf{y}_d$ (smisao = pogreška k-tog rješenja).

Konkretno, za $\mathbf{w}^{(0)} = [0\ 0]^{\mathrm{T}}$ i $\alpha_k = \text{konst.} = 10^{-3}$, nakon 200 iteracija (epoha) imamo $\mathbf{w}^{(200)} = [0{,}4095 \quad 0]^{\mathrm{T}}$.

(do rješenja s pogreškom $< 10^{-6}$ dolazimo nakon 1357 koraka)

Konačno rješenje bilo bi... Nije loše, a riječ je o samo jednom neuronu ...



Prijedlog za vježbu

Prethodna funkcija je bila relativno blago zaobljena ("slična" ravnini) pa je jedan neuron bio dovoljan za njenu aproksimaciju. Međutim, već samo malo složeniju, a vrlo srodnu funkciju

$$f: R^2 \to R \quad ; \quad f(x_1, x_2) = \sin^2(x_1)\cdot\cos^2(x_2),$$
$$x_1, x_2 \in [-1,1]$$

jedan neuron ne može pratiti.

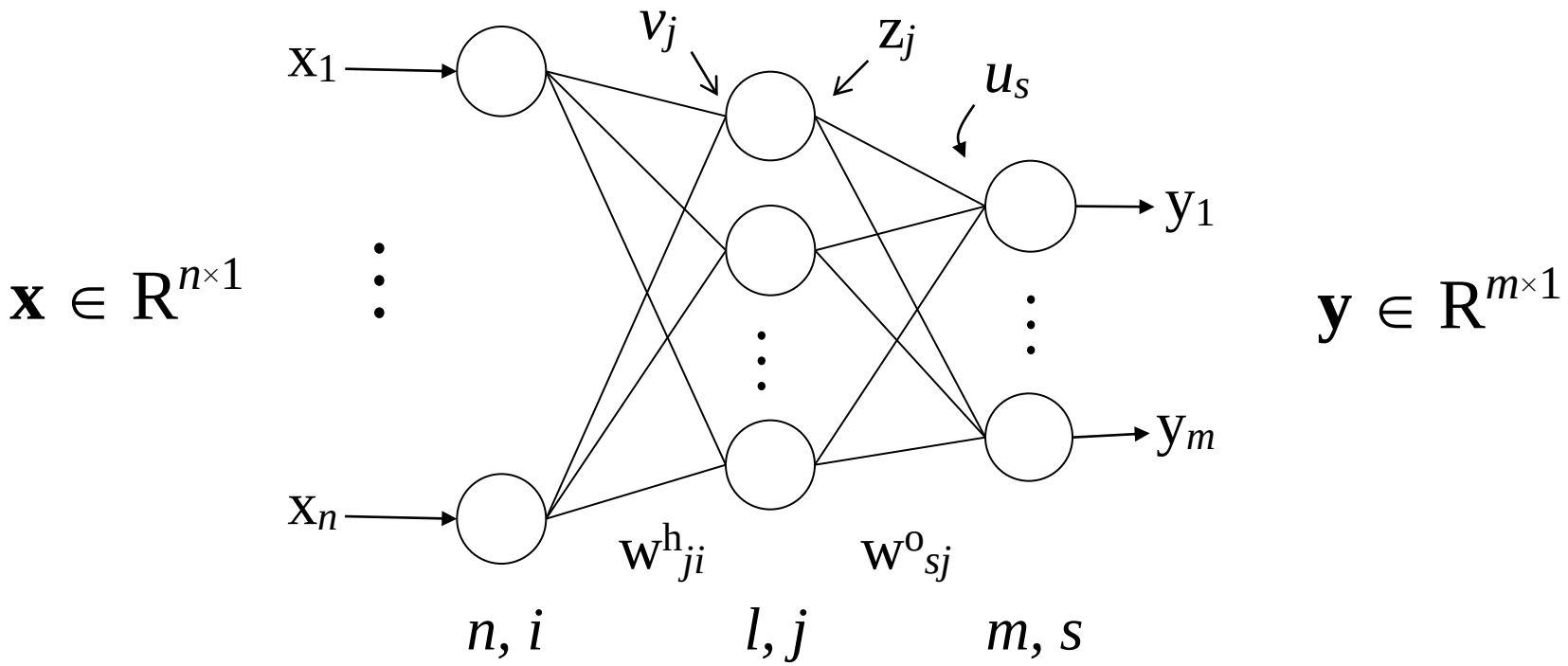
Uvjerite se u tu činjenicu koristeći program kojim je pripremljen prethodni primjer!

Uvježbavanje cijele mreže

Radi preglednosti i razumljivosti izlaganja, ograničit ćemo se na mrežu sa samo jednim skrivenim slojem. Međutim, logika algoritma i građa formula su identični i za složenije mreže pa je analiza koja slijedi valjana općenito.

Imamo n ulaza x_i, $i = 1, \ldots, n$. Nadalje, imamo m izlaza y_s, $s = 1, \ldots, m$. Skriveni sloj ima l neurona čiji su izlazi z_j, $j = 1, \ldots, l$. Aktivacijsku funkciju neurona skrivenog sloja označimo s f^{h}, a onu izlaznog sloja s f^{o}. Težine veza iz ulaznog u skriveni sloj su w^{h}_{ji}, a onih iz skrivenog u izlazni sloj w^{o}_{sj}.

Takva mreža obavlja (definira) preslikavanje iz R^n u R^m.



Odredimo eksplicitnu formulu preslikavanja u jednoj točki.

Ukupni ulaz v_j u j-ti neuron skrivenog sloja (naziva se i *lokalno polje*; *local field*) i njegov izlaz z_j jesu

$$v_j = \sum_{i=1}^{n} w_{ji}^h x_i \quad , \quad z_j = f_j^h\left(\sum_{i=1}^{n} w_{ji}^h x_i\right) = f_j^h\left(v_j\right) \quad .$$

Izlaz iz s-tog neurona izlaznog sloja je

$$y_s = f_s^{\mathrm{o}}\left(\sum_{j=1}^{l} w_{sj}^{\mathrm{o}} z_j\right) = f_s^{\mathrm{o}}\left(\sum_{j=1}^{l} w_{sj}^{\mathrm{o}} f_j^{h}\left(v_j\right)\right)$$

$$= f_s^{\mathrm{o}}\left(\sum_{j=1}^{l} w_{sj}^{\mathrm{o}} f_j^{\mathrm{h}}\left(\sum_{i=1}^{n} w_{ji}^{\mathrm{h}} x_i\right)\right) = F_s^{\mathrm{o}}(x_1, x_2, \ldots, x_n) \quad ,$$

a iz cijele mreže

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} F_1^{\mathrm{o}}(x_1, \ldots, x_n) \\ \vdots \\ F_m^{\mathrm{o}}(x_1, \ldots, x_n) \end{bmatrix} = \begin{bmatrix} F_1^{\mathrm{o}}(\mathbf{x}) \\ \vdots \\ F_m^{\mathrm{o}}(\mathbf{x}) \end{bmatrix} = \mathbf{F}^{\mathrm{o}}(\mathbf{x}) \quad .$$

Radi sažetijeg i prikladnijeg zapisa uvodimo

$$\mathbf{W}^{\mathrm{h}} = \begin{bmatrix} w_{11}^{\mathrm{h}} & \cdots & w_{1n}^{\mathrm{h}} \\ \vdots & \ddots & \vdots \\ w_{l1}^{\mathrm{h}} & \cdots & w_{ln}^{\mathrm{h}} \end{bmatrix} \quad \text{i} \quad \mathbf{W}^{\mathrm{o}} = \begin{bmatrix} w_{11}^{\mathrm{o}} & \cdots & w_{1l}^{\mathrm{o}} \\ \vdots & \ddots & \vdots \\ w_{m1}^{\mathrm{o}} & \cdots & w_{ml}^{\mathrm{o}} \end{bmatrix},$$

gdje indeksi elemenata (težinskih faktora *w*) u matricama imaju smisao “u koji neuron promatranog sloja” (prvi indeks) “iz kojeg neurona prethodnog sloja” (drugi indeks), dakle u redcima su težine svih veza iz prethodnog sloja u jedan te isti neuron promatranog sloja.

Uz uvedene oznake vrijedi $\mathbf{v} = [v_1\ v_2\ \ldots\ v_l]^{\mathrm{T}} = (\mathbf{W}^{\mathrm{h}}\mathbf{x})$. Ako još uvedemo vektor $\mathbf{z} = [z_1\ z_2\ \ldots\ z_l]^{\mathrm{T}}$, onda je izlaz s-tog neurona izlaznog sloja jednak $y_s = f^{\mathrm{o}}{}_s(\mathbf{W}^{\mathrm{o}}{}_{s*}\mathbf{z})$. Svi izlazi čine vektor $\mathbf{y} = [y_1\, y_2\ \ldots\ y_m]^{\mathrm{T}}$, a ako su sve $f^{\mathrm{o}}{}_s = f^{\mathrm{o}}$, proizlazi da je $\mathbf{y} = \boldsymbol{f}^{\mathrm{o}}(\mathbf{W}^{\mathrm{o}}\mathbf{z})$. Zapis se jednako proširuje i za $\mathbf{X} \in \mathrm{R}^{n \times p}$.

Ukratko, izlazi mreže za p točaka na ulazu, smještenih u matricu $\mathbf{X} \in \mathrm{R}^{n \times p}$, čine matricu $\mathbf{Y} \in \mathrm{R}^{m \times p}$ koja se dobiva računanjem redom:

ulazi u skrivene neurone:	$\mathbf{V}^{\mathrm{h}} = \mathbf{W}^{\mathrm{h}} \cdot \mathbf{X}$	$(l \times p)$
izlazi skrivenih neurona:	$\mathbf{Z}^{\mathrm{h}} = \boldsymbol{f}^{\mathrm{h}}(\mathbf{V}^{\mathrm{h}})$	$(l \times p)$
ulazi u izlazne neurone:	$\mathbf{U}^{\mathrm{o}} = \mathbf{W}^{\mathrm{o}} \cdot \mathbf{Z}^{\mathrm{h}}$	$(m \times p)$
izlazi mreže:	$\mathbf{Y}^{\mathrm{o}} = \boldsymbol{f}^{\mathrm{o}}(\mathbf{V}^{\mathrm{o}})$	$(m \times p)$

Za p parova za uvježbavanje dobivamo p vektora $\boldsymbol{e}_i = [e_1, e_2, \ldots, e_m]^{\mathrm{T}}$ izlaznih pogrešaka koje smještamo u matricu pogrešaka $\mathbf{e} \in \mathrm{R}^{m \times p}$

$$\mathbf{e} = \begin{bmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \cdots & \boldsymbol{e}_p \end{bmatrix} = \begin{bmatrix} e_{11} & \cdots & e_{1p} \\ \vdots & \ddots & \vdots \\ e_{m1} & \cdots & e_{mp} \end{bmatrix}$$

Cilj uvježbavanja je minimizirati zbroj kvadrata normi svih vektora $\boldsymbol{e}_i$, odnosno ciljnu funkciju $E(\mathbf{w})$

$$\text{minimizirati} \quad E(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{p} \left\| \boldsymbol{e}_k \right\|^2 = \frac{1}{2} trace\left(\mathbf{e}^{\mathrm{T}} \cdot \mathbf{e}\right) \cdot$$

Minimizacija se opet provodi gradijentnom metodom, a uz poznati gradijent $\nabla E(\mathbf{w})$, parametri mreže se ispravljaju kao i u svakoj drugoj gradijentnoj optimizaciji.

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha_k \cdot \nabla E(\mathbf{w})$$

Gradijent nalazimo slično kao i u slučaju jednog neurona.

$$\nabla E(\mathbf{w}) = \nabla \frac{1}{2} \sum_{i=1}^{p} \left\| \boldsymbol{e}_i \right\|^2 = \frac{1}{2} \sum_{i=1}^{p} \nabla \left\| \boldsymbol{e}_i \right\|^2 = \sum_{i=1}^{p} \left\| \boldsymbol{e}_i \right\| \cdot \nabla \left\| \boldsymbol{e}_i \right\|$$

Točan gradijent ovisi o pogreškama u svim točkama za uvježbavanje pa bi se mreža trebala uvježbavati po načelu skupnog uvježbavanja.

Skupno uvježbavanje je najpouzdanije jer su svojstva tog postupka (uvjeti i brzina konvergencije) poznati iz opće analize gradijentne optimizacije.

Usprkos tome, u primjeni je češće koračno uvježbavanje (*on-line learning*) zbog već navedenih prednosti.

$$\nabla E(\mathbf{w}) = \nabla \frac{1}{2}\sum_{i=1}^{p}\|\boldsymbol{e}_i\|^2 = \frac{1}{2}\sum_{i=1}^{p}\nabla\|\boldsymbol{e}_i\|^2 = \frac{1}{2}\cdot p\cdot\left(\nabla\|\boldsymbol{e}_i\|^2\right)_{mean}$$

$$= p\cdot\left(\|\boldsymbol{e}_i\|\cdot\nabla\|\boldsymbol{e}_i\|\right)_{mean} \approx p\cdot\|\boldsymbol{e}_i\|\cdot\nabla\|\boldsymbol{e}_i\|$$

Ukupni gradijent je p puta veći od prosječnog gradijenta pa kad bi u svakom koraku (za svaki par za uvježbavanje) trenutačni umnožak $\|\boldsymbol{e}_i\|\cdot\nabla\|\boldsymbol{e}_i\|$ bio približno jednak svojoj prosječnoj vrijednosti, koračni ispravak parametara

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha'_k\cdot 1/2\cdot\nabla\|\boldsymbol{e}_i\|^2 = \mathbf{w}^{(k)} - \alpha_k\cdot\|\boldsymbol{e}_i\|\cdot\nabla\|\boldsymbol{e}_i\|$$

bio bi $1/p$ skupnog ispravka.

Koračno uvježbavanje ćete, uglavnom, primjenjivati i na ispitima. ☺

Algoritam rasprostiranja (prenošenja) unatrag (*backpropagation algorithm*)

Za oba postupka uvježbavanja trebamo gradijent kvadrata norme $\nabla\|\boldsymbol{e}_i\|^2$ izlazne pogreške u jednoj točki. U osnovi, problem je ekvivalentan minimizaciji trenutačne pogreške

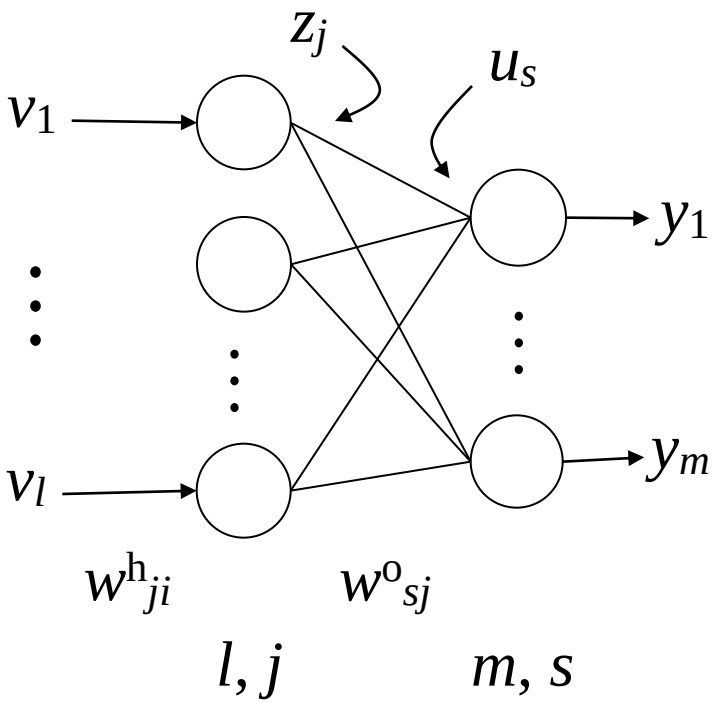
$$E_i(\mathbf{w}) = \frac{1}{2}\left\|\boldsymbol{e}_i\right\|^2 = \frac{1}{2}\left\|\mathbf{y}_i - \mathbf{y}_{d,i}\right\|^2 = \frac{1}{2}\sum_{s=1}^{m}(y_{s,i} - y_{ds,i})^2$$

s obzirom na parametre **w**. Izračunavanje gradijenta $\nabla E_i(\mathbf{w})$ iziskuje izračunavanje svih parcijalnih derivacija kvadrata norme $\|\boldsymbol{e}_i\|^2$ (po svim w_{ij}, za trenutačni **w**)...

$\Rightarrow$ izračunljivo za mrežu s jednim skrivenim slojem, ali što ako skrivenih slojeva ima više, s desetcima ili stotinama čvorova u svakoj razini?!?

Treba uočiti da je neuronska mreža “prirodno rekurzivna” struktura; iz sloja u sloj građena je i funkcionira jednako. Uspijemo li izračunati parcijalne derivacije po težinama između dva susjedna sloja samo na temelju stanja (aktivnosti) čvorova u ta dva sloja, na isti ćemo način moći izračunati cijeli gradijent!

Promotrimo zadnja dva sloja:



v_j = ukupni ulaz u j-ti neuron prethodnog sloja

z_j = izlaz (*aktivnost*) j-tog neurona prethodnog sloja

u_s = ukupni ulaz u s-ti neuron promatranog sloja

y_s = izlaz (*aktivnost*) s-tog neurona promatranog sloja

1) Derivacija s obzirom na izlaz (aktivnost) izlaznog sloja

$$E_i(\mathbf{w}) = \frac{1}{2}\sum_{s=1}^{m}(y_{s,i} - y_{ds,i})^2 \quad \Rightarrow \quad EA_s^{o} = \frac{\partial E_i}{\partial y_s} = y_{s,i} - y_{ds,i} = (\boldsymbol{e}_i)_s$$

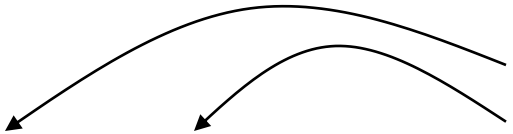
Sve EA^{o} u i-toj točki, s obzirom na sve izlaze, smještamo u stupčani vektor $\mathbf{EA}^{o} = \mathbf{y}_i - \mathbf{y}_{di}$ dimenzija ($m \times 1$).

2) Derivacija s obzirom na uzlaz u izlazni sloj

$$EI_s = \frac{\partial E_i}{\partial u_s} = \frac{\partial E_i}{\partial y_s}\frac{\partial y_s}{\partial u_s} = EA_s^{o} \cdot f'^{o}(u_s) = EA_s^{o} \cdot y_s(1 - y_s)$$

Vektorski:

$$\underset{(m\times 1)}{\mathbf{EI}} = \underset{(m\times 1)}{\mathbf{EA}^{o}} .* \underset{(m\times 1)}{\mathbf{y}} .* \underset{(m\times 1)}{(\mathbf{1} - \mathbf{y})} \quad .$$



Pozor!

3) Derivacija s obzirom na težinu poveznica s prethodnim slojem

$$EW_{sj} = \frac{\partial E_i}{\partial w_{sj}} = \frac{\partial E_i}{\partial u_s}\frac{\partial u_s}{\partial w_{sj}} = EI_s \cdot z_j$$

Uz $\mathbf{z} = [z_1\ z_2\ \ldots\ z_l]^{\mathrm{T}}$, sve EW, s obzirom na sve w_{sj}, čine matricu $\mathbf{EW}^{\mathrm{o}} = \mathbf{EI} \cdot \mathbf{z}^{\mathrm{T}}$ dimenzija $(m \times l)$.

4) Derivacija s obzirom na izlaz prethodnog sloja

$$EA_j^{\mathrm{h}} = \frac{\partial E_i}{\partial z_j} = \sum_s \frac{\partial E_i}{\partial u_s}\frac{\partial u_s}{\partial z_j} = \sum_s EI_s w_{sj} = \mathbf{EI}^{\mathrm{T}} \cdot \mathbf{W}^{\mathrm{o}}_{*j}$$

Matrično:

$$\underset{(l\times 1)}{\mathbf{EA}^{\mathrm{h}}} = \left(\underset{(1\times m)}{\mathbf{EI}^{\mathrm{T}}} \cdot \underset{(m\times l)}{\mathbf{W}^{\mathrm{o}}} \right)^{\mathrm{T}} = \underset{(l\times m)}{(\mathbf{W}^{\mathrm{o}})^{\mathrm{T}}} \cdot \underset{(m\times 1)}{\mathbf{EI}} \quad .$$

Posljednja formula zatvara krug!

Krenuvši od derivacije $\mathbf{EA}^{o}$ s obzirom na izlaz mreže, izračunali smo derivaciju $\mathbf{EA}^{h}$ s obzirom na izlaz prethodnog sloja, a "putem" i derivaciju $\mathbf{EW}^{o}$ s obzirom na težine poveznica, koju smo zapravo i trebali.

Ponavljajući korake 2...4, derivacije s obzirom na izlaze promatranog sloja prevodimo u derivacije s obzirom na izlaze njemu prethodnog sloja i taj se postupak može provesti nad proizvoljnim brojem slojeva. Pritom, prelazeći iz sloja u sloj, u trećem koraku nalazimo potrebne derivacije s obzirom na težine poveznica.

Tijekom opisanog postupka, pogreška izlaza se prenosi u (rasprostire, pronosi kroz) sve prethodne slojeve i to je osnovno obilježje tog algoritma po kojem je i nazvan.

Za one koji ne vole matrice... ☺

Općenito, derivacija kvadrata norme izlazne pogreške po parametru w_{ji} je (j je ovdje indeks u promatranom sloju)

$$\mathbf{EW}_{ji} = \partial E(\mathbf{w})/\partial w_{ji} = \delta_j \cdot z_i \qquad (z_i \text{ je izlaz } i\text{-tog prethodnika}) \; .$$

Faktor δ (u izvodu EI) se naziva *lokalni gradijent* i računa:

$$\delta_j = f\,'(v_j)\cdot(y_j - y_{dj}) \qquad ;\; j = \text{izlazni sloj}$$

ili

$$\delta_j = f'(v_j)\sum_k \delta_k w_{kj} \qquad ;\; j \neq \text{izlazni sloj} \;\; ,$$

gdje je k indeks neurona u sljedećem sloju. Za uvijek istu *stopu učenja* α_k = konst.= η, ispravak parametra je $\Delta w_{ji} = -\eta\cdot\mathbf{EW}_{ji} = -\eta\cdot\delta_j\cdot z_i$, odnosno osvježeni parametri su

$$(w_{ji})_{\text{novi}} = (w_{ji})_{\text{trenutačni}} + \Delta w_{ji} = (w_{ji})_{\text{trenutačni}} - \eta\cdot\delta_j\cdot z_i \quad .$$

Ta se relacija naziva *delta pravilo*.

Struktura backpropagation algoritma može se dobro uočiti na primjeru troslojne mreže pa ćemo za nju izvesti eksplicitnu formulu gradijenta.

Radi jednostavnosti zapisa, neka je zadan samo jedan par $(\mathbf{x}_d,\mathbf{y}_d)$ za uvježbavanje, $\mathbf{x}_d \in \mathrm{R}^{n\times 1}$ i $\mathbf{y}_d \in \mathrm{R}^{m\times 1}$. Problem određivanja optimalnih parametara mreže je

$$\text{minimizirati} \quad E(\mathbf{w}) = \frac{1}{2}\left\|\mathbf{y}-\mathbf{y}_d\right\|^2 = \frac{1}{2}\sum_{s=1}^{m}(y_s - y_{ds})^2 \ ,$$

a optimizacija se provodi po svim elementima $\mathbf{w}^{\mathrm{h}}$ i $\mathbf{w}^{\mathrm{o}}$.

Za formulaciju algoritma trebamo gradijent $E(\mathbf{w})$, tj. parcijalne derivacije $E(\mathbf{w})$ po svim w^{h}_{ji} i w^{o}_{sj}.

U troslojnoj mreži derivacije po w^{o}_{sj} su

$$\frac{\partial E(\mathbf{w})}{\partial w^{o}_{sj}} = (y_s - y_{ds}) f_s'^{o}\left(\sum_{q=1}^{l} w^{o}_{sq} z_q\right) z_j = \mathbf{EA}^{o}_{s} \cdot f_s'^{o}\left(\sum_{q=1}^{l} w^{o}_{sq} z_q\right) z_j \ ,$$

$$z_q = f_q^{h}\left(\sum_{i=1}^{n} w^{h}_{qi} x_{di}\right) = f_q^{h}\left(v_q\right) \quad .$$

Prepoznajemo faktor δ:

$$\delta^{o}_{s} = (y_s - y_{ds}) f_s'^{o}\left(\sum_{q=1}^{l} w^{o}_{sq} z_q\right) = (y_s - y_{ds}) f_s'^{o}\left(u_s\right) \quad = \mathrm{EI}^{o}$$

Taj faktor predstavlja skaliranu (pomnoženu nekim faktorom) pogrešku *s*-tog neurona izlaznog sloja.

Sada se parcijalna derivacija $E(\mathbf{w})$ po w^{o}_{sj} može kratko pisati

$$\frac{\partial E(\mathbf{w})}{\partial w^{o}_{sj}} = \delta^{o}_{s} z_j \quad . \qquad = \mathrm{EW}^{o}sj$$

Nadalje, derivacija $E(\mathbf{w})$ po w^{h}_{ji} je

$$\frac{\partial E(\mathbf{w})}{\partial w^{h}_{ji}} = \sum_{p=1}^{m} (y_p - y_{dp}) f_p^{'o}\left(\sum_{q=1}^{l} w^{o}_{pq} z_q \right) w^{o}_{pj} f_j^{'h}\left(\sum_{r=1}^{n} w^{h}_{jr} x_{dr} \right) x_{di}$$

ili s prethodno uvedenim oznakama

$$\frac{\partial E(\mathbf{w})}{\partial w^{h}_{ji}} = \underbrace{\left(\sum_{p=1}^{m} \delta^{o}_{p} w^{o}_{pj} \right)}_{\mathrm{EA}^{h}_{j}} f_j^{'h}(v_j) x_{di} = \delta^{h}_{j} \cdot x_{di} \quad .$$

$f^{'o}(u_j)$ z_j

Algoritam rasprostiranja unatrag (*backpropagation*) za mrežu s jednim skrivenim slojem:

$$w_{sj}^{\mathrm{o}(k+1)} = w_{sj}^{\mathrm{o}(k)} - \eta \frac{\partial E(\mathbf{w}^{(k)})}{\partial w_{sj}^{\mathrm{o}}} = w_{sj}^{\mathrm{o}(k)} - \eta \delta_s^{\mathrm{o}^{(k)}} z_j^{(k)}$$

$$w_{ji}^{\mathrm{h}(k+1)} = w_{ji}^{\mathrm{h}(k)} - \eta \frac{\partial E(\mathbf{w}^{(k)})}{\partial w_{ji}^{\mathrm{h}}} = w_{ji}^{\mathrm{h}(k)} - \eta \delta_j^{\mathrm{h}^{(k)}} x_{di}$$

$$= w_{ji}^{\mathrm{h}(k)} - \eta \left(\sum_{p=1}^{m} \delta_p^{\mathrm{o}^{(k)}} w_{pj}^{\mathrm{o}(k)} \right) f_j^{'\mathrm{h}}(v_j^{(k)}) x_{di} \quad ,$$

pri čemu je η (konstantni) korak u smjeru gradijenta.

Sve druge oznake uvedene su prethodno, ali sada ih pregledno navodimo sve zajedno.

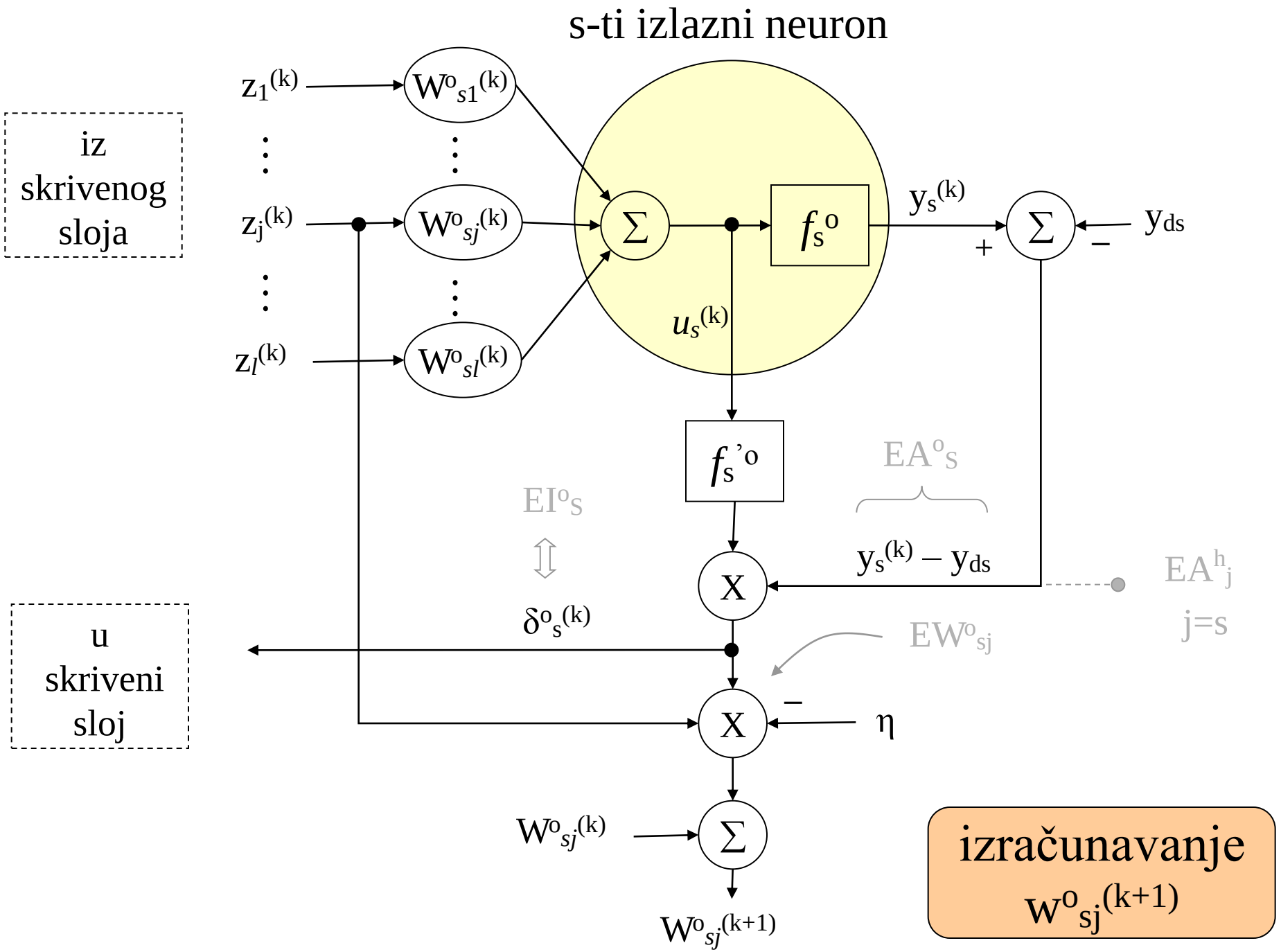
Oznake u algoritmu rasprostiranja unatrag (troslojna mreža):

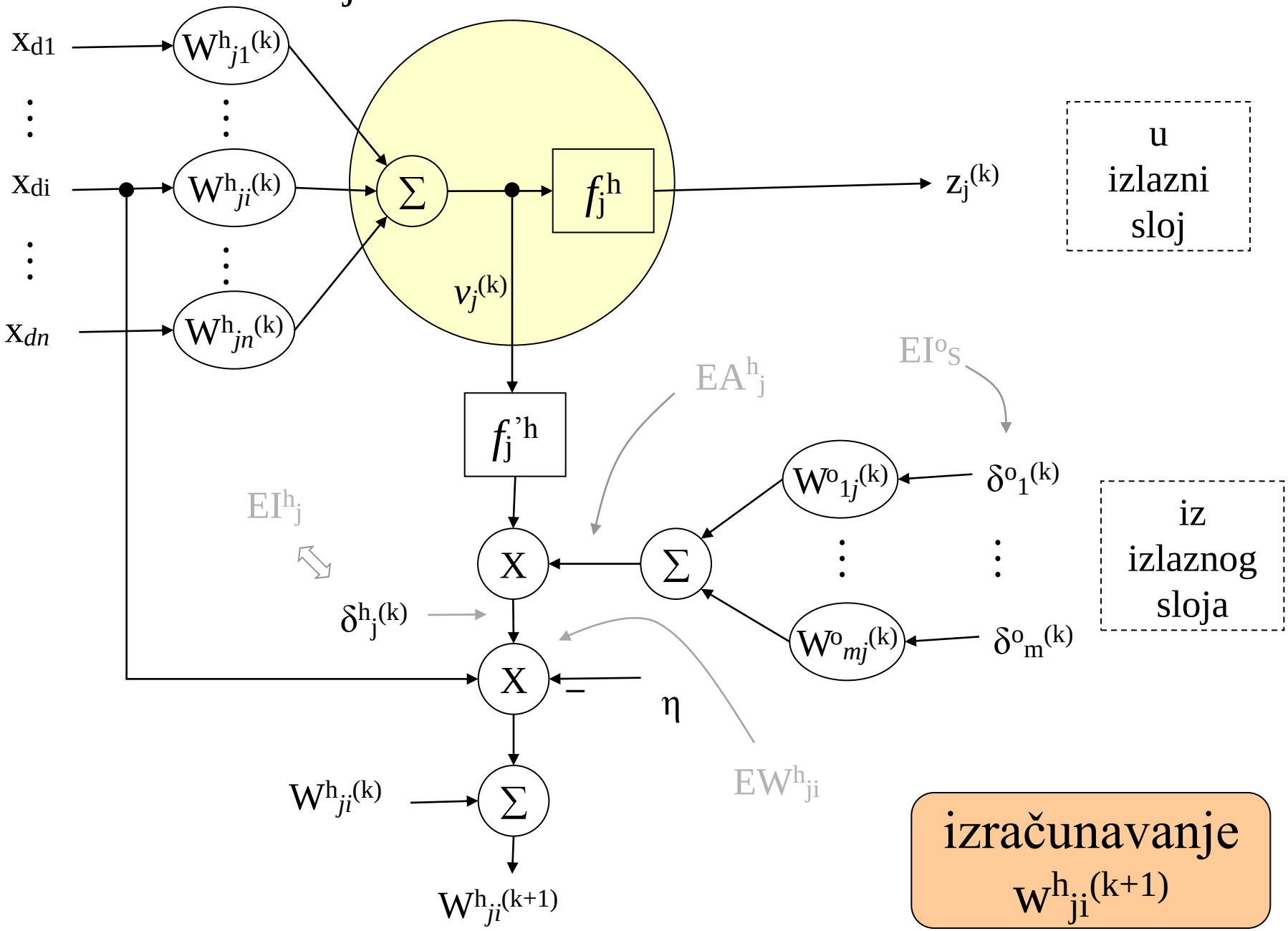
$$v_j^{(k)} = \sum_{i=1}^{n} w_{ji}^{h(k)} x_{di} \quad , \quad z_j^{(k)} = f_j^h\left(\sum_{i=1}^{n} w_{ji}^{h(k)} x_{di}\right) = f_j^h\left(v_j^{(k)}\right)$$

$$y_s^{(k)} = f_s^o\left(\sum_{q=1}^{l} w_{sq}^{o(k)} z_q^{(k)}\right) = f_s^o\left(u_s^{(k)}\right) \quad ,$$

$$\delta_s^{o(k)} = f_s'^o\left(u_s^{(k)}\right) \cdot (y_s^{(k)} - y_{ds}) \quad ,$$

$$\delta_j^{h(k)} = f_j'^h(v_j^{(k)}) \cdot \sum_{p=1}^{m} \delta_p^{o(k)} w_{pj}^{o(k)} \quad .$$


s-ti izlazni neuron
iz skrivenog sloja
u skriveni sloj
$z_1^{(k)}$
$z_j^{(k)}$
$z_l^{(k)}$
$w^o_{s1}(k)$
$w^o_{sj}(k)$
$w^o_{sl}(k)$
Σ
$u_s^{(k)}$
f_s^o
$y_s^{(k)}$
Σ
+
−
y_{ds}
$f_s'^o$
EA^o_S
EI^o_S
$y_s^{(k)} - y_{ds}$
EA^h_j
j=s
X
$\delta^o_s(k)$
EW^o_{sj}
X
−
η
$w^o_{sj}(k)$
Σ
$w^o_{sj}(k+1)$
izračunavanje
$w^o_{sj}(k+1)$


j-ti skriveni neuron
x_{d1}
x_{di}
x_{dn}
$W^h_{j1}(k)$
$W^h_{ji}(k)$
$W^h_{jn}(k)$
Σ
f_j^h
$v_j(k)$
$z_j(k)$
u izlazni sloj
$f_j'^h$
EA^h_j
EI^o_S
EI^h_j
$W^o_{1j}(k)$
$W^o_{mj}(k)$
$\delta^o_1(k)$
$\delta^o_m(k)$
iz izlaznog sloja
X
$\delta^h_j(k)$
η
EW^h_{ji}
$W^h_{ji}(k)$
$W^h_{ji}(k+1)$
izračunavanje $w^h_{ji}(k+1)$

Kao što se vidi iz formula i dijagrama, algoritam rasprostiranja unatrag ima dvije faze:

1. na temelju ulaza x_{di} i trenutačnih parametara mreže (težina $\mathbf{w}^{h}$ i $\mathbf{w}^{o}$) izračunamo redom $v_j^{(k)}$, $z_j^{(k)}$, $y_s^{(k)}$ i $\delta_s^{(k)}$. Ova se faza naziva *prolazak prema naprijed* (*forward pass*) jer se utjecaj (stanje) ulaza prenosi sve do izlaza.
2. iz veličina izračunanih tijekom prve faze računamo nove parametre mreže ($w^{o}_{sj}{}^{(k+1)}$ i $w^{h}_{ji}{}^{(k+1)}$). Ova faza se naziva *prolazak unazad* ili *obrnuti prolazak* (*backward or reverse pass*) jer se izlazne pogreške $\delta_s^{(k)}$ prenose (rasprostiru) unatrag, od izlaza prema ulazu.

Algoritam ima ista ova svojstva i u složenim mrežama koje imaju višerazinski skriveni sloj.

Samostalni neuron može i ne mora imati aktivacijsku funkciju. A neuroni u mreži?

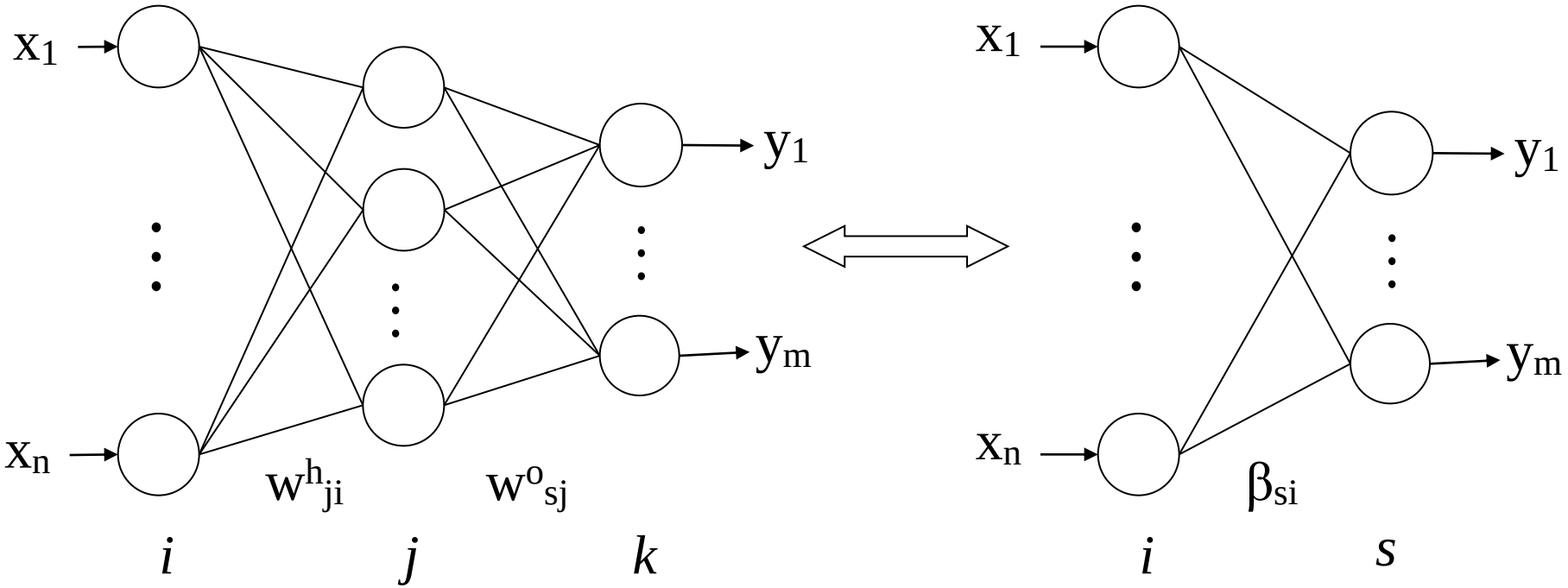
Moraju, i to nelinearnu, inače mreža ne bi bila ono što se čini da je, tj. izgubila bi cijeli skriveni sloj!

$$y_s = \sum_{j=1}^{l} w_{sj}^{\mathrm{o}} \sum_{i=1}^{n} w_{ji}^{\mathrm{h}} x_i = \sum_{j=1}^{l} 1 \cdot \sum_{i=1}^{n} w_{sj}^{\mathrm{o}} w_{ji}^{\mathrm{h}} x_i = \sum_{j=1}^{l} 1 \cdot \sum_{i=1}^{n} \alpha_{sji} x_i$$

To bi bila mreža u kojoj sve poveznice skrivenog i izlaznog sloja imaju težinu =1, a poveznice ulaznog i skrivenog sloja težine α_{sji}. Izlazne težine =1 znače da veze skrivenog i izlaznog sloja nisu “aktivne”, tj. mreža ima samo jedan sloj prilagodljivih poveznica (iz ulaza u prvi skriveni sloj) i ponaša se kao da je dvoslojna: ulaz-izlaz.

Gubitak cijelog skrivenog sloja (svih razina) je još uočljiviji ako prethodni izraz napišemo malo drugačije.

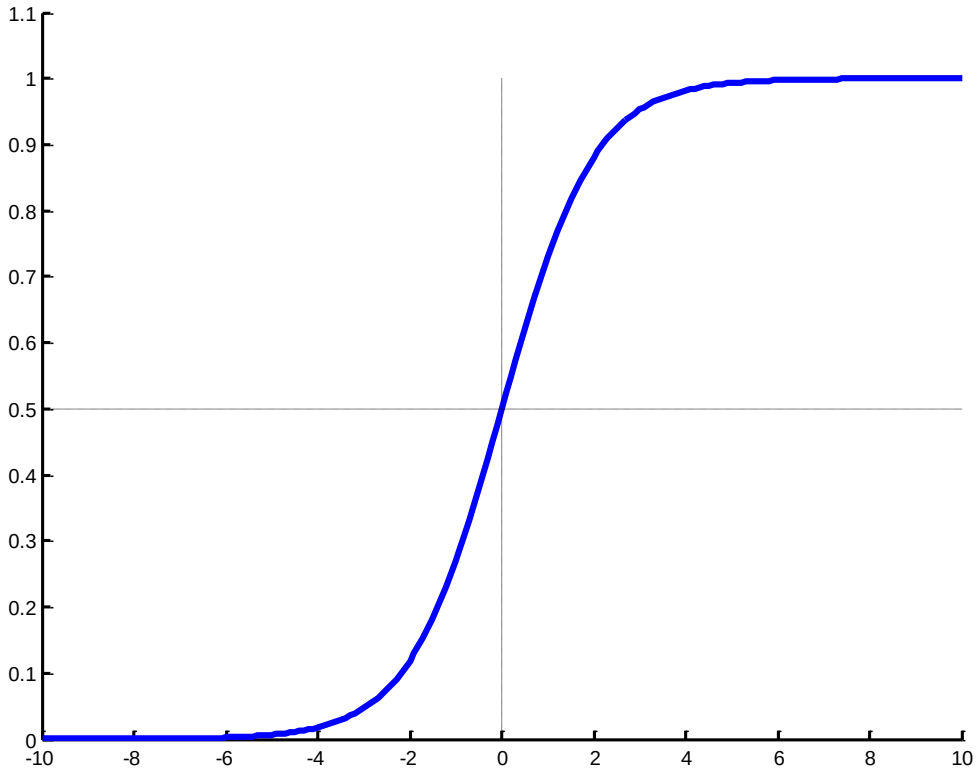
$$y_s = \sum_{i=1}^{n}\sum_{j=1}^{l} w_{sj}^{o} w_{ji}^{h} x_i = \sum_{i=1}^{n} x_i \sum_{j=1}^{l} w_{sj}^{o} w_{ji}^{h} = \sum_{i=1}^{n} x_i \beta_{si}$$



Očito je da aktivacijska funkcija mora biti nelinearna ako se želi izbjeći gubitak skrivenog sloja, a da bi se mogla primijeniti gradijentna optimizacijska metoda, aktivacijska funkcija mora biti i derivabilna.
Osobito prikladna iz više razloga i najčešće primjenjivana funkcija je *sigmoid* čija definicijska formula glasi

$$f(x) = \frac{1}{1+e^{-x}} \quad ,$$

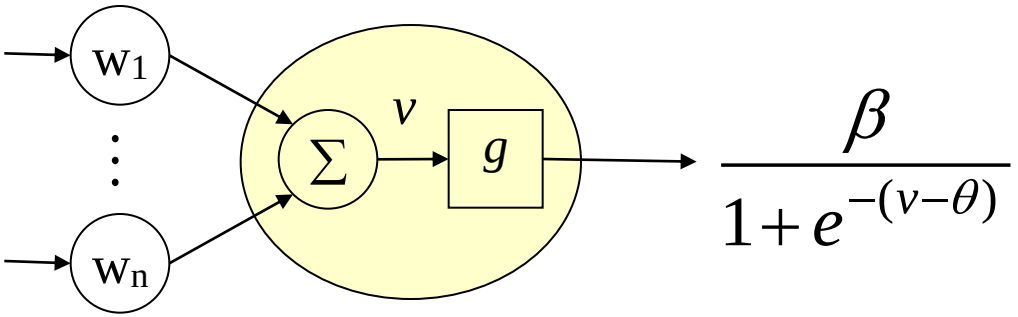
$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = f(x)\left[1 - f(x)\right] \quad .$$



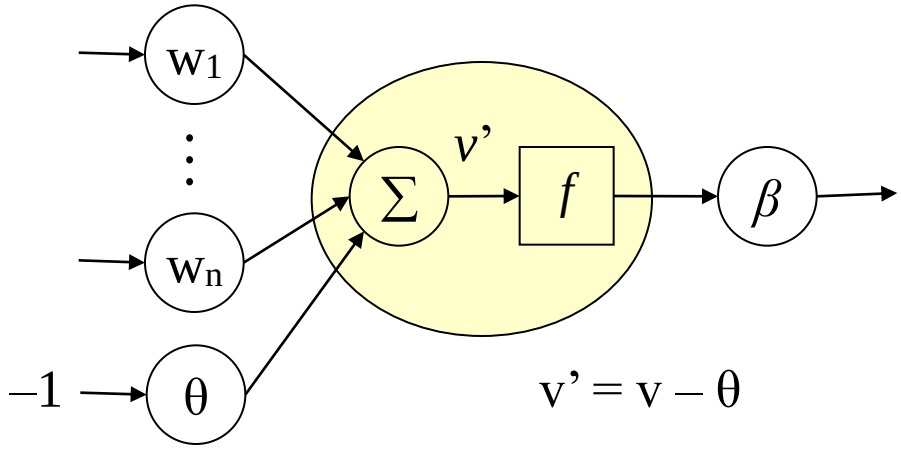
Opći oblik te funkcije je

$$g(x) = \frac{\beta}{1+e^{-(x-\theta)}} \quad , \qquad g'(x) = g(x)\left[1-\frac{g(x)}{\beta}\right]$$

i ona za svaki neuron može imati drugačije parametre, ali svi se ti parametri lako ugrađuju u algoritam rasprostiranja unatrag.



$$\frac{\beta}{1+e^{-(v-\theta)}}$$



$$\frac{\beta}{1+e^{-v'}}$$

v’ = v – θ

$$= \frac{\beta}{1+e^{-(v-\theta)}}$$

Parametri β zapravo ne mogu postojati samostalno jer:

1. β neurona i težinu w poveznice sa sljedećim slojem nije moguće razlučiti tijekom optimizacije jer izlaz mreže ovisi o njihovom umnošku, dakle o umnošku dva neovisna faktora, pa rješenja za kombinaciju ta dva faktora ima beskonačno, a samim time beskonačno je i rješenja za cijeli skup parametara
2. izlaz ovisi o umnošku faktora β izlaznog neurona i funkcije svih drugih parametara, dakle opet o dva neovisna faktora pa ih ni u tom umnošku nije moguće razlučiti.

To znači da se mrežu može uvježbati samo za rezultate u opsegu (0,1) pa svaki problem treba svesti na taj opseg. Ili, ukloniti nelinearnost (sigmoide) izlaznim neuronima...

Eksplicitne formule za troslojnu mrežu kada su aktivacijske funkcije sigmoidi s pomakom (tj. sigmoidi s θ, a svi $\beta = 1$):

$$y_s = g_s^o\left(\sum_{j=1}^{l} w_{sj}^o g_j^h\left(\sum_{i=1}^{n} w_{ji}^h x_i\right)\right)$$

$$= f_s^o\left(\sum_{j=1}^{l} w_{sj}^o f_j^h\left(\sum_{i=1}^{n} w_{ji}^h x_i - \theta_j^h\right) - \theta_s^o\right)$$

$$= f_s^o\left(\sum_{j=1}^{l} w_{sj}^o f_j^h\left(v_j\right) - \theta_s^o\right) = f_s^o\left(\sum_{j=1}^{l} w_{sj}^o z_j - \theta_s^o\right) \quad ,$$

$$v_j = \sum_{i=1}^{n} w_{ji}^h x_i - \theta_j^h \quad ,$$

$$z_j = f_j^h\left(\sum_{i=1}^{n} w_{ji}^h x_i - \theta_j^h\right) = f_j^h\left(v_j\right) \quad .$$

Parcijalne derivacije su:

$$\frac{\partial E(\mathbf{w})}{\partial w_{sj}^{o}} = (y_s - y_{ds}) f_s^{'o}\left(\sum_{q=1}^{l} w_{sq}^{o} z_q - \theta_s^{o}\right) z_j = \delta_s^{o} z_j \quad ,$$

$$\delta_s^{o} = (y_s - y_{ds}) f_s^{'o}\left(\sum_{q=1}^{l} w_{sq}^{o} z_q - \theta_s^{o}\right) \quad ,$$

$$\frac{\partial E(\mathbf{w})}{\partial w_{ji}^{h}} = \sum_{p=1}^{m} (y_p - y_{dp}) f_p^{'o}\left(\sum_{q=1}^{l} w_{pq}^{o} z_q - \theta_p^{o}\right) w_{pj}^{o} f_j^{'h}\left(\sum_{r=1}^{n} w_{jr}^{h} x_{dr} - \theta_j^{h}\right) x_{di}$$

$$= x_{di} f_j^{'h}(v_j) \sum_{p=1}^{m} \delta_p^{o} w_{pj}^{o} = x_{di} \delta_j^{h} \quad ,$$

$$\frac{\partial E(\mathbf{w})}{\partial \theta_s^o} = -(y_s - y_{ds}) f_s^{'o}\left(\sum_{q=1}^{l} w_{sq}^o z_q - \theta_s^o\right) = -\delta_s^o \quad ,$$

$$\frac{\partial E(\mathbf{w})}{\partial \theta_j^h} = -\sum_{p=1}^{m}(y_p - y_{dp}) f_p^{'o}\left(\sum_{q=1}^{l} w_{pq}^o z_q - \theta_p^o\right) w_{pj}^o f_j^{'h}\left(\sum_{r=1}^{n} w_{jr}^h x_{dr} - \theta_j^h\right)$$

$$= -f_j^{'h}(v_j)\sum_{p=1}^{m}\delta_p^o w_{pj}^o = -\delta_j^h \quad .$$

Prolazak prema naprijed matrično

$$\underset{(n_h \times 1)}{\mathbf{v}} = \underset{(n_h \times n_i)}{\mathbf{W}^h} \cdot \underset{(n_i \times 1)}{\mathbf{x}} - \underset{(n_h \times 1)}{\boldsymbol{\theta}^h} \qquad \underset{(n_h \times 1)}{\mathbf{z}} = sigmoid\left(\underset{(n_h \times 1)}{\mathbf{v}}\right)$$

$$\underset{(n_o \times 1)}{\mathbf{u}} = \underset{(n_o \times n_h)}{\mathbf{W}^o} \cdot \underset{(n_h \times 1)}{\mathbf{z}} - \underset{(n_o \times 1)}{\boldsymbol{\theta}^o} \qquad \underset{(n_o \times 1)}{\mathbf{y}} = sigmoid\left(\underset{(n_o \times 1)}{\mathbf{u}}\right)$$

Rasprostiranje unatrag matrično

$$\underset{(n_o \times 1)}{\mathbf{EA}^o} = \underset{(n_o \times 1)}{\mathbf{y}} - \underset{(n_o \times 1)}{\mathbf{y}_d}$$

$$\underset{(n_o \times 1)}{\mathbf{EI}^o} = \underset{(n_o \times 1)}{\mathbf{EA}^o} .* \underset{(n_o \times 1)}{\mathbf{y}} .* \left(\underset{(n_o \times 1)}{\mathbf{1}} - \underset{(n_o \times 1)}{\mathbf{y}}\right)$$

$$\underset{(n_o \times 1)}{\boldsymbol{\delta}^o} = \underset{(n_o \times 1)}{\mathbf{EI}^o}$$

.* = množenje po parovima, a ne matrično!

$$\underset{(n_o \times n_h)}{\mathbf{EW}^o} = \underset{(n_o \times 1)}{\boldsymbol{\delta}^o} \cdot \underset{(1 \times n_h)}{\mathbf{z}^T} = \underset{(n_o \times 1)}{\mathbf{EI}^o} \cdot \underset{(1 \times n_h)}{\mathbf{z}^T}$$

$$\underset{(n_h \times 1)}{\mathbf{EA}^h} = \underset{(n_h \times n_o)}{(\mathbf{w}^o)^T} \cdot \underset{(n_o \times 1)}{\mathbf{EI}^o}$$

.* = množenje po parovima, a ne matrično!

$$\underset{(n_h \times 1)}{\mathbf{EI}^h} = \underset{(n_h \times 1)}{\mathbf{EA}^h} .* \underset{(n_h \times 1)}{\mathbf{z}} .* \left(\underset{(n_h \times 1)}{\mathbf{1}} - \underset{(n_h \times 1)}{\mathbf{z}} \right)$$

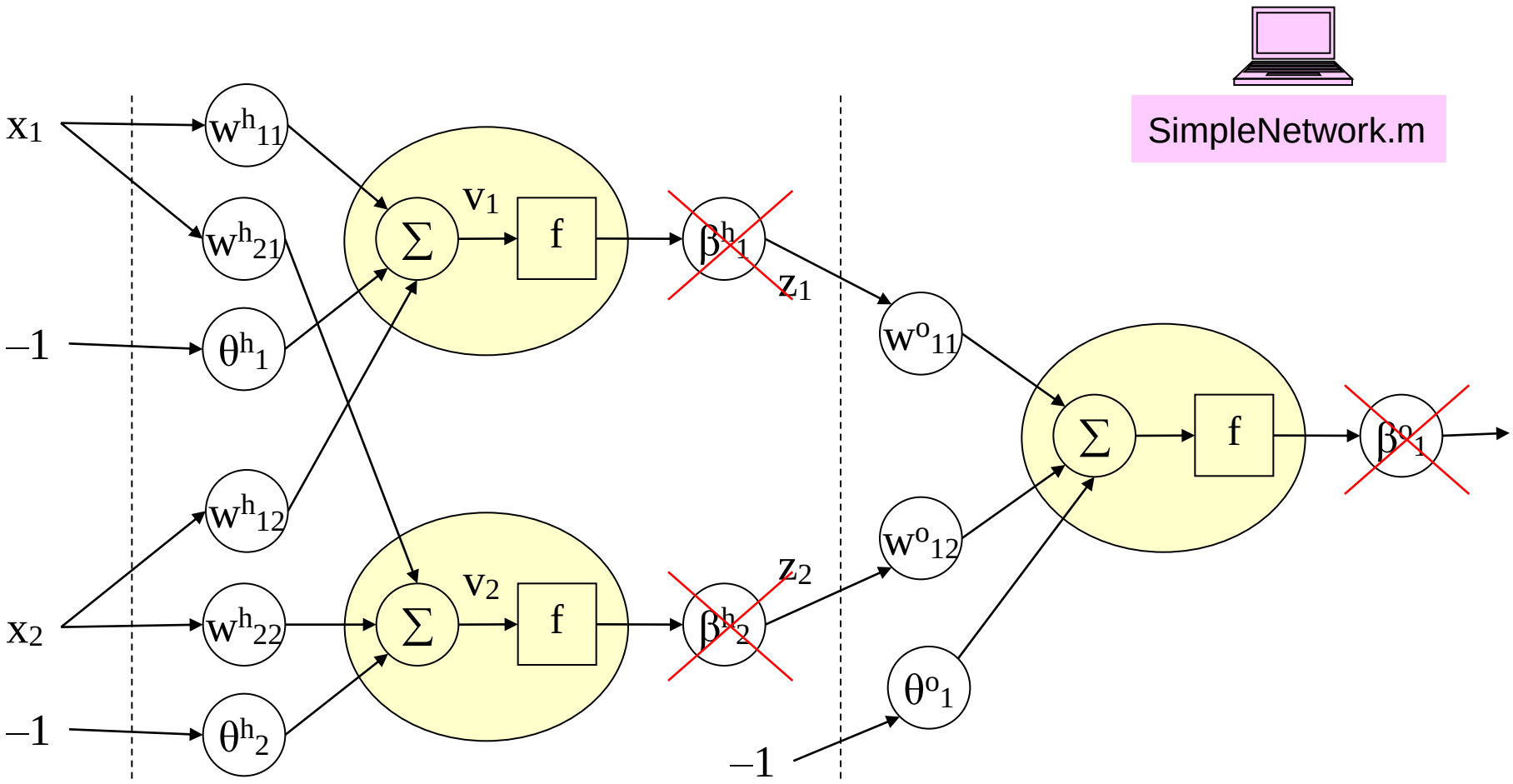
$$\underset{(n_h \times 1)}{\boldsymbol{\delta}^h} = \underset{(n_h \times 1)}{\mathbf{EI}^h}$$

$$\underset{(n_h \times n_i)}{\mathbf{EW}^h} = \underset{(n_h \times 1)}{\boldsymbol{\delta}^h} \cdot \underset{(1 \times n_i)}{\mathbf{x}^T} = \underset{(n_h \times 1)}{\mathbf{EI}^h} \cdot \underset{(1 \times n_i)}{\mathbf{x}^T}$$

$$\underset{(n_o \times 1)}{\mathbf{E\Theta}^o} = -\underset{(n_o \times 1)}{\mathbf{EI}^o} \qquad \underset{(n_h \times 1)}{\mathbf{E\Theta}^h} = -\underset{(n_h \times 1)}{\mathbf{EI}^h}$$

Primjer: uvježbavanje mreže 2×2×1

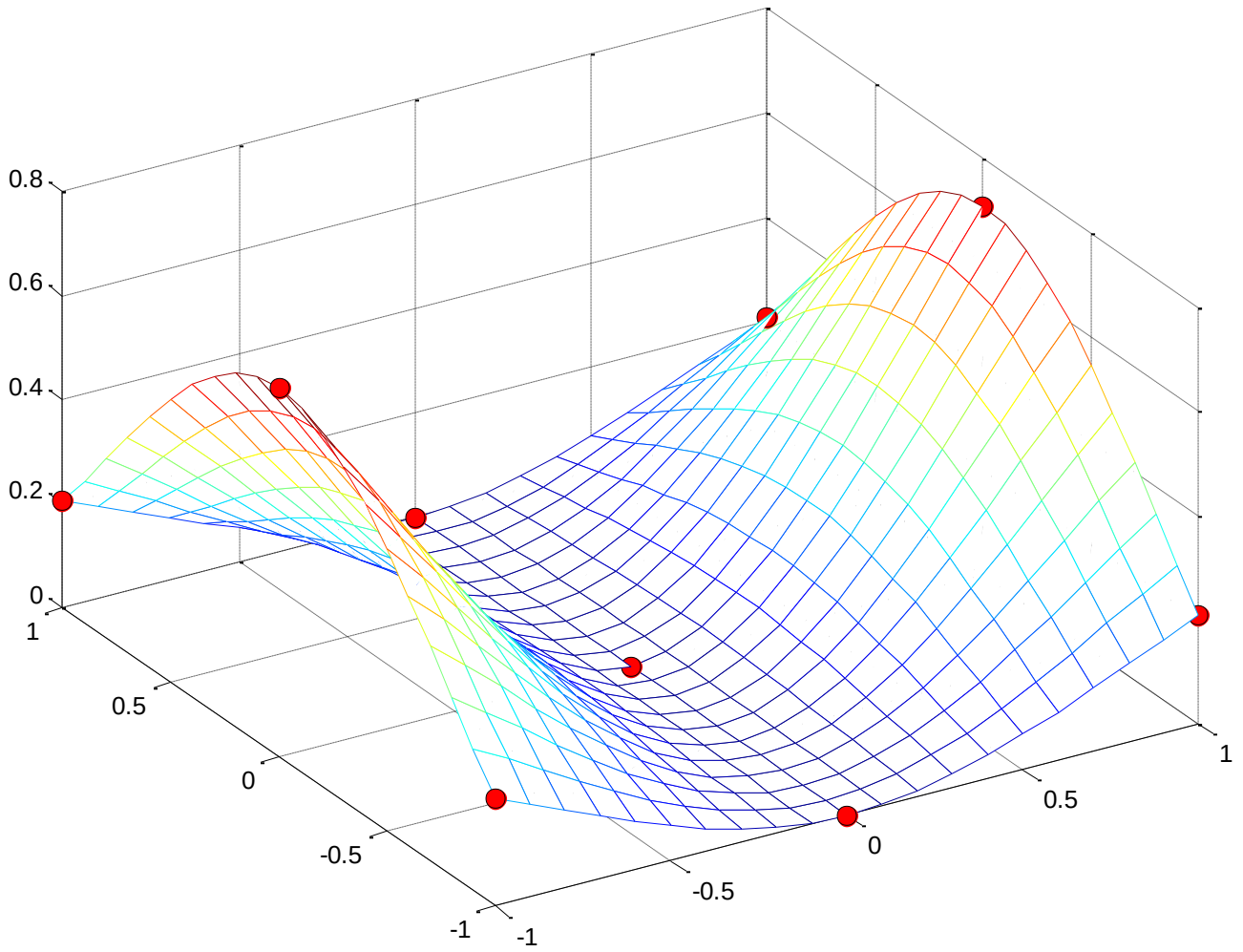
Mreža ima dva ulaza, dva neurona u skrivenom sloju i jedan izlaz, a aktivacijska funkcija je (opći) sigmoid.



Zadaća mreže je aproksimirati funkciju

$$f\colon \mathrm{R}^2 \to \mathrm{R} \quad ; \quad f(x_1, x_2) = \sin^2(x_1)\cdot\cos^2(x_2),$$
$$x_1, x_2 \in [-1,1]$$

koju jedan neuron ne može pratiti (prisjetite se prijedloga za vježbu!).



Za polazne parametre $w^h_{11}= -13$, $w^h_{21}=15$, $w^h_{12}= -5$, $w^h_{22}= -6$, $w^o_{11}= 6$, $w^o_{12}= 6$, $\theta^h_1= 8$, $\theta^h_2= 9$ i $\theta^o_1= 4$ te $\eta = 1$, nakon 30000 iteracija (epoha) dobivamo

$w^h_{11} = -16.1107$

$w^h_{21} = 17.1725$

$w^h_{12} = -6.5361$

$w^h_{22} = -6.8322$

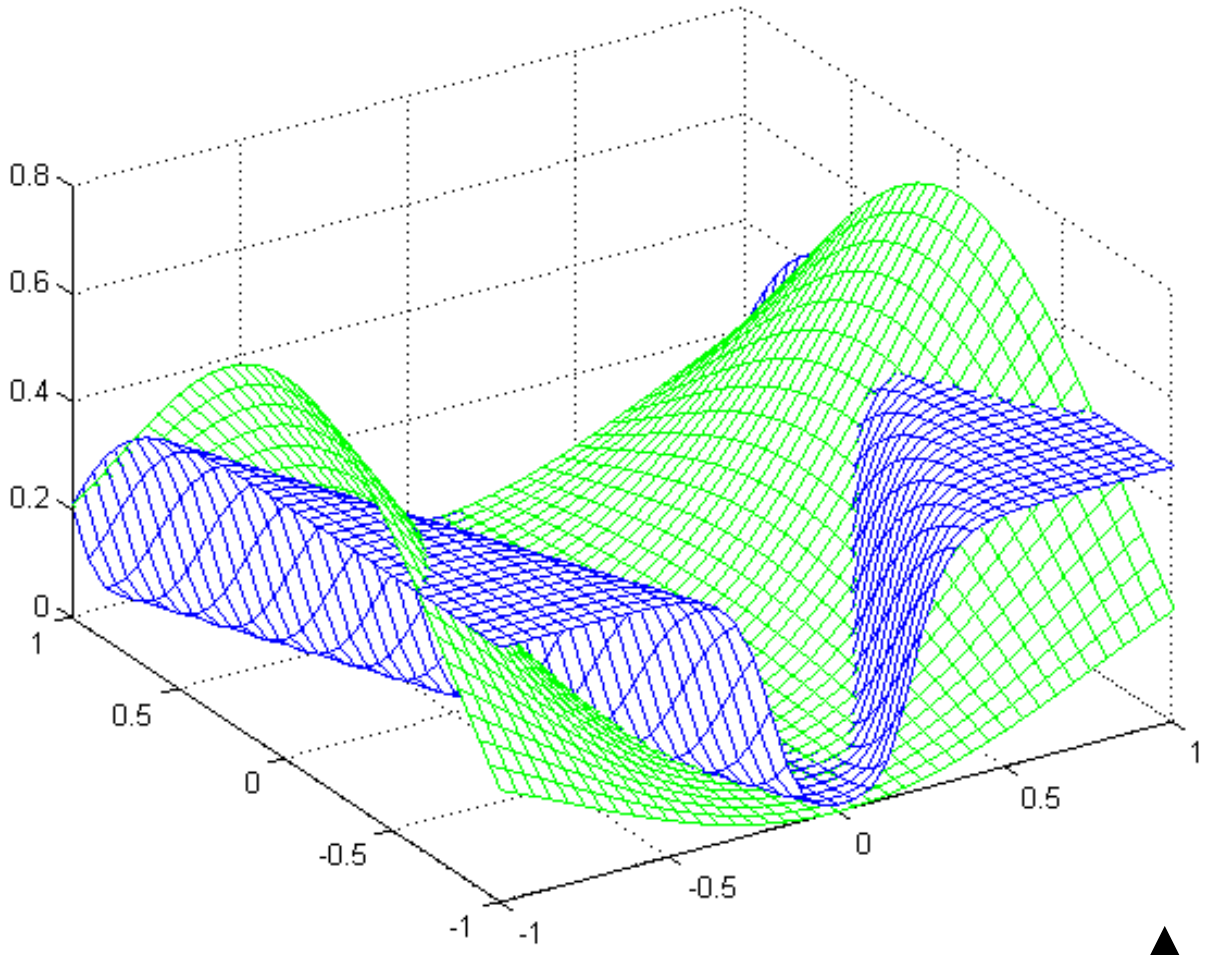
$w^o_{11} = 6.2848$

$w^o_{12} = 6.2848$

$\theta^h_1 = 8.1618$

$\theta^h_2 = 8.9283$

$\theta^o_1 = 6.3961$.



Primjer: uvježbavanje mreže 2×2×1

Mreža ima dva ulaza, dva neurona u skrivenom sloju i jedan izlaz, a aktivacijska funkcija je (opći) sigmoid.

Zadaća mreže je aproksimirati XOR (ekskluzivni ILI) logičku funkciju zadanu tablicom istine.

X1	X2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

SimpleNetwork.m

Tablica neka ujedno bude i skup za uvježbavanje, dakle imamo četiri zadane točke.

Za polazne parametre $w^h_{11}=0.1$, $w^h_{21}=0.3$, $w^h_{12}=0.3$, $w^h_{22}= 0.4$, $w^o_{11}= 0.4$, $w^o_{12}= 0.6$, $\theta^h_1= 0.1$, $\theta^h_2=1$ i $\theta^o_1= -0.3$ te $\eta=10$, nakon 4000 iteracija (epoha) dobivamo

$w^h_{11} = -7.8541$

$w^h_{21} = -5.7366$

$w^h_{12} = -8.443$

$w^h_{22} = -5.7857$

$w^o_{11} = -11.5318$

$w^o_{12} = 11.4628$

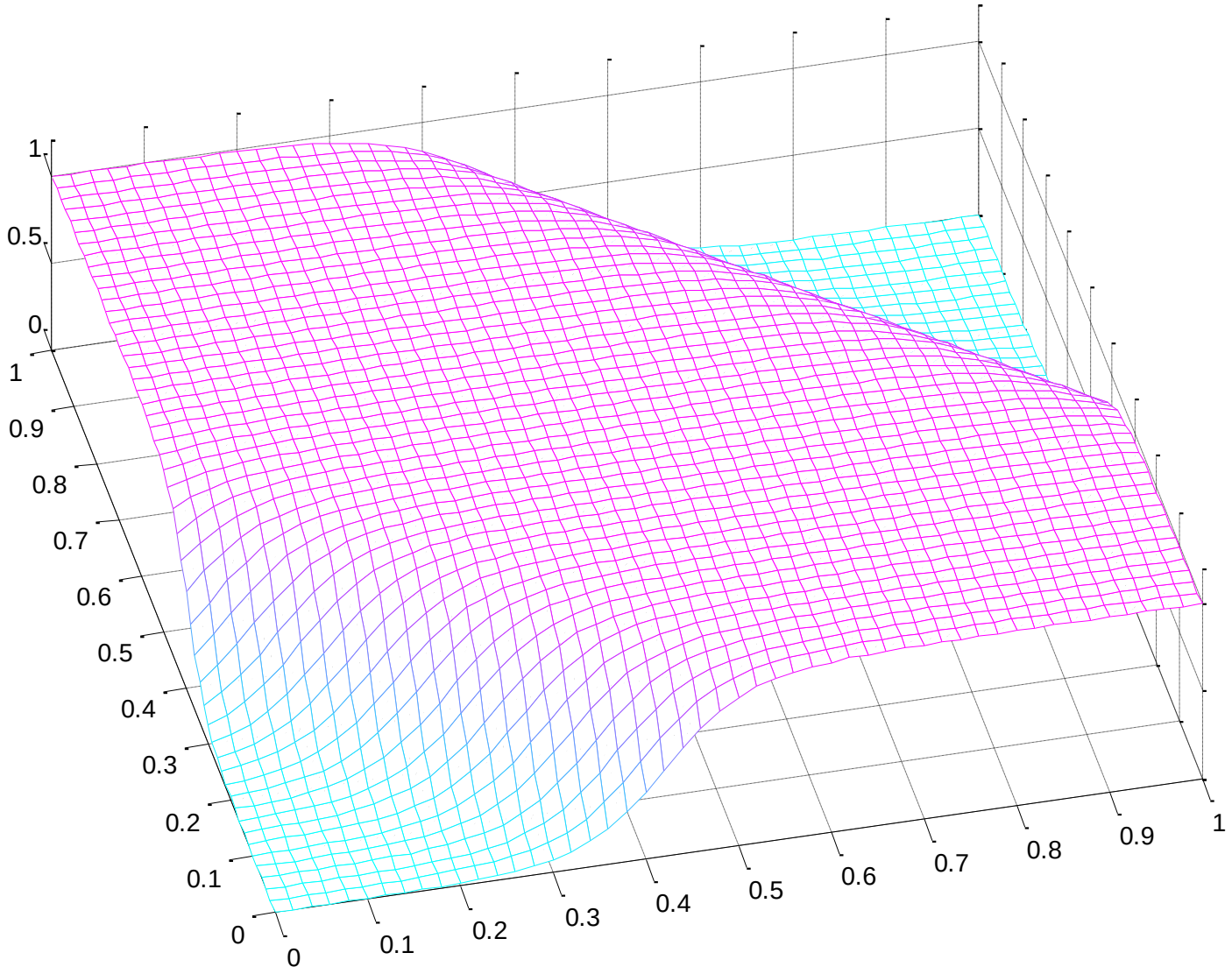
$\theta^h_1 = -3.3684$

$\theta^h_2 = -8.5833$

$\theta^o_1 = 5.5385$.

X1	**X2**	**izlaz mreže**
0	0	0.0053
0	1	0.9945
1	0	0.9943
1	1	0.0069

U cijelom prostoru aproksimacija je posve dobra...



Završne napomene i praktični savjeti

1. Početne vrijednosti parametara mreže
   - parametri θ svi =0, a težine w = nasumični brojevi u opsegu u kojem se očekuju njihove konačne vrijednosti

Proračun w se temelji na:

- opsegu vrijednosti (i varijabilnosti) ulaznih varijabli
- broju ulaza u neurone pojedinog sloja
  (= broj neurona u prethodnom sloju, ali samo u potpuno povezanoj mreži)
- aktivnom dijelu domene aktivacijske funkcije;
  za sigmoid je to $\approx -5...5$
- izlaznom opsegu (kodomeni) aktivacijske funkcije; za sigmoid je to interval (0,1) (asimptotske granice)

Primjer: troslojna mreža s n ulaza i l neurona u skrivenom sloju, očekivana vrijednost ulaza μ_x i očekivana vrijednost izlaza skrivenih neurona μ_h.

Očekivani zbroj ulaza (*lokalno polje*) u skriveni neuron je $n \cdot \mu_x \cdot w^h_{mean}$. Da bi neuron bio aktivan, taj zbroj treba biti <5. Zbog zalihosti odabiremo <2 pa slijedi da srednja težina w^h treba biti $w^h_{mean} < 2/(n \cdot \mu_x)$, tj. težine w^h inicijaliziramo slučajnim brojevima $w^h \in [-w^h_{mean}, w^h_{mean}]$.

Očekivani zbroj ulaza (*lokalno polje*) u izlazni neuron jednak je $l \cdot w^o_{mean} \cdot \mu_h$. Mreža mora ostati aktivna i u ekstremnom slučaju (najveći mogući ulaz u izlazni sloj) pa mora biti $l \cdot w^o_{mean} \cdot \mu_h = <5$ i kada je $\mu_h = \max = 1$. Slijedi $w^o_{mean} < 5/l$ i inicijalno postavljamo $w^o_{sj} \in [-w^o_{mean}, w^o_{mean}]$.

2. Stopa učenja

Teorijski uvjet konvergencije LMS algoritma prilikom uvježbavanja jednog neurona jest da je stopa učenja η

$$0 < \eta < 2/\lambda_{\max} \quad ,$$

gdje je $\lambda_{\max}$ najveća svojstvena vrijednost korelacijske matrice ulaznih vektora $\mathbf{R}xx = E(\mathbf{x}\cdot\mathbf{x}^T)$. U primjeni, $\mathbf{R}xx$ aproksimiramo izrazom $Rxx = 1/n\cdot\mathbf{X}^T\cdot\mathbf{X}$, gdje je $\mathbf{X} \in R^{n\times p}$ matrica ulaznih vektora (vrijedi da je $\mathbf{R}xx$ = limes od Rxx kada $n\rightarrow\infty$). Ako su svi ulazi međusobno statistički nezavisni i imaju očekivanje $E(x_j) = 0$, proizlazi da je $\lambda_{\max} = \max\{\sigma^2_j\}$, odnosno

$$\lambda_{\max} = \max_j \left\{ \frac{1}{n} \sum_{i=1}^{n} (x_j)_i^2 \right\} \quad ,$$

gdje je j indeks ulaza (ulaznog vektora), a i indeks podataka.

Dvije metode prilagodbe stope učenja najčešće u praksi jesu:

- motriti trenutačnu pogrešku ($y_s - y_{ds}$) i ovisno o njoj mijenjati stopu η tijekom uvježbavanja mreže
  - ako pogreška ($y_s - y_{ds}$) mijenja predznak (oscilira), stopa učenja je prevelika i treba ju smanjiti da bi se postigla brža konvergencija; vrijedi i obrat
  - bilo bi dobro kad bismo u svakom koraku mogli izračunati optimalnu stopu η, tj. primijeniti *steepest descent* algoritam, ali...
- "automatizirati" prilagodbu stope učenja uvođenjem tromosti u izračunavanje ispravaka parametara
  – poopćeno *delta pravilo*

$$\Delta w_{ji}(k) = \gamma \cdot \Delta w_{ji}(k-1) - \eta \cdot \delta_j(k) \cdot z_i(k) \quad ,$$

  gdje je γ *tromost* (konstanta inercije) postupka
  – unosi povratnu vezu, tj. memoriju u postupak

Poopćeno *delta pravilo* je zapravo jednadžba diferencija (*difference equation*) prvog reda po funkciji $\Delta w_{ji}(k)$. Rješenje te jednadžbe je

$$\Delta w_{ji}(k) = \eta \sum_{t=0}^{k} \gamma^{k-t} \delta_j(t) z_i(t) \quad ,$$

što je niz (signal; *time series*) od k+1 pribrojnika s eksponencijalno promjenjivim težinama γ^{k-t}. Ako se prisjetimo da je $\delta_j(t) \cdot z_i(t) = \mathbf{E}\mathbf{W}_{ji}(t) = \partial \boldsymbol{e}(t,\mathbf{w}) / \partial w_{ji}$, slijedi

$$\Delta w_{ji}(k) = \eta \sum_{t=0}^{k} \gamma^{k-t} \frac{\partial \boldsymbol{e}(t, \mathbf{w})}{\partial w_{ji}} \quad ,$$

a iz te formule možemo iščitati učinak uvođenja tromosti, kao i uvjete na parametar γ.

a) Kada su parcijalne derivacije $\partial \boldsymbol{e}(t,\mathbf{w})/\partial w_{ji}$ izlazne pogreške po parametru w_{ji} "mahom" istog predznaka, dakle kada je promjena u dimenziji nekog parametra "dugo" približno istog smjera, ispravak Δw_{ji} će rasti, odnosno parametar w_{ji} će biti značajnije promijenjen. Vrijedi i obratno pa je jasno da uvođenje tromosti ima stabilizacijski učinak na proces uvježbavanja mreže jer će možebitne oscilacije predznaka ispravaka u dimenziji pojedinog parametra dovesti do smanjenja ispravka.

b) Ispravci $\Delta w_{ji}(k)$, matematički gledano, jesu red, a iz uvjeta konvergencije redova slijedi da tromost γ mora biti u granicama $0 \leq |\gamma| < 1$. Tromost može biti i pozitivna i negativna, iako negativna baš i nema pretjeranog smisla...

3. Kriterij zaustavljanja

Gradijentna metoda sigurno konvergira ka točki u kojoj gradijent isčezava (jednak je nuli), ali pronalaženje globalnog optimuma nije zajamčeno pa tako nema ni jedinstvenog i matematički neprikosnovenog kriterija za zaustavljanje.

Neki češće primjenjivani kriteriji jesu:

- norma gradijenta manja od zadanog minimuma (upitno prilikom koračnog uvježbavanja)
- ukupna pogreška mreže (za cijeli skup) manja od zadanog minimuma
- promjena ukupne pogreške mreže, u uzastopnim koracima, manja od zadanog minimuma.

4. Ulazni podatci nejednake važnosti (pouzdanosti)
   - mrežu uvježbavamo pridjeljujući parovima za uvježbavanje različite težine t_i

$$\text{minimizirati} \quad \frac{1}{2}\sum_{s=1}^{m} t_i (y_s - y_{ds})^2 \quad ; \; i=1, 2, \ldots, p$$

   - za koračno uvježbavanje to znači da cijeli gradijent množimo s t_i u svakoj točki, tj. za svaki par vektora $(\boldsymbol{x}_d, \boldsymbol{y}_d)$

5. Promjena redoslijeda ulaznih podataka
   - prilikom koračnog uvježbavanja dobro je računati sa slučajno odabranim parovima, tj. skup za uvježbavanje svaki puta (u svakoj iteraciji; epohi) treba "predočiti" mreži drugačijim redoslijedom

6. Normalizacija ulaza

Ako to namjena mreže dozvoljava, poželjno je:

a) pomaknuti (translatirati) ulazne varijable tako da im srednja vrijednost bude približno nula
b) ukloniti svaku statističku vezu među ulazima
    - taj se postupak naziva *dekorelacija*, a može se postići primjenom statističke tehnike koja se naziva *analiza glavnih* (*najvažnijih*) *sastavnica* (*principal component analysis*, *PCA*)
c) skalirati ulazne vrijednosti tako da im kovarijance budu približno jednake

Normalizacija ulaza ujednačava brzinu prilagodbe (učenja) veza (parametra) mreže.