

## Viseslojne mreze - propagacija unazad

Perceptroni, posto imaju samo jedan sloj, imaju mogucnost da rese samo probleme ciji su ulazni obrasci **linearno separabilni** ili **linearno nezavisni** (to znači da odbirci klase mogu da se podele jednom pravom).

Viseslojne feedforward mreze nemaju ta ogranicenja. Zbog toga je generalizovanjem Widrow-Hoff-ovog pravila obucavanja na viseslojne mreze i nelinearno diferencijabilne transfer funkcije napravljen algoritam **backpropagation** (propagacije unazad).

Standardni backpropagation je **algoritam opadajuceg gradijenta**, isto kao i Widrow-Hoff-ov, u kom se pojacanja kreću ka negativnom gradijentu.

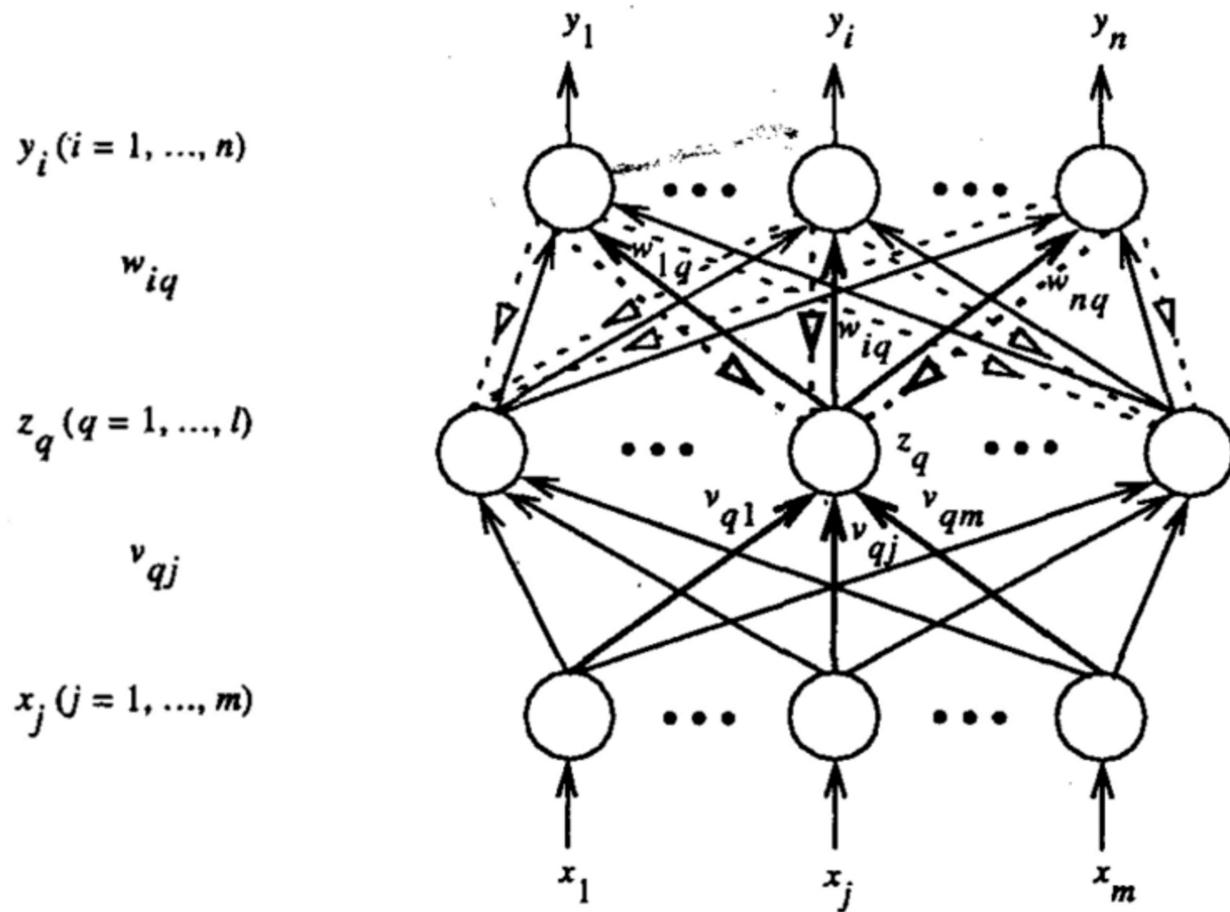
Funkcije aktivacije su **kontinualne diferencijabilne** funkcije (bez prekida), a ucenje je **supervizijsko**.

Motivacija za koriscenje ovog algoritma: tesko je pronaci izvode za sva pojacanja izmedju svih neurona u viseslojnoj mrezi, pa se ovim algoritmom to olaksava, jer se traže samo pojedini izvodi.

Za dati input-output par:  $\{(x^{(k)}, d^{(k)})\}$

- prvo se propagiraju ulazi na sve nivoe (iz svakog neurona u jednom sloju postoji veza do svakog neurona u sledecem sloju - videti sliku) dok se ne dodje do poslednjeg i **kreira izlaz  $y(k)$**
- onda se **signali gresaka** koji su dobijeni od  $d(k)$  i  $y(k)$  **propagiraju unazad** od poslednjeg sloja preko svih prethodnih slojeva da bi se pomocu toga vrsio **update tezina (pojacanja)** i **racunale greske** za svaki prethodni sloj, tj. da bi se videlo koliko je svaki sloj doprineo gresci

Primer sa troslojnom mrezom:



PE (procesirajući element - neuron) q:

- prima net input:

$$\text{net}_q = \sum_{j=1}^m v_{qj} x_j$$

- **net<sub>q</sub>** predstavlja vrednost cele mreze koja ide do neurona q
- na net uticu svi neuroni iz prethodnog sloja ( $x_j$ ) zajedno sa svojim pojacanjima  $v_{qj}$  (q znači da to pojaganje ide do neurona q, a sa j se označavaju svi neuroni koji učestvuju:  $x_1..x_m$ )

- proizvodi output (kada se ulaz proučice kroz aktivacionu funkciju) :

$$z_q = a(\text{net}_q) = a\left(\sum_{j=1}^m v_{qj} x_j\right)$$

PE i:

- prima net input:

$$\text{net}_i = \sum_{q=1}^l w_{iq} z_q = \sum_{q=1}^l w_{iq} a\left(\sum_{j=1}^m v_{qj} x_j\right)$$

- proizvodi output:

$$y_i = a(\text{net}_i) = a\left(\sum_{q=1}^l w_{iq} z_q\right) = a\left(\sum_{q=1}^l w_{iq} a\left(\sum_{j=1}^m v_{qj} x_j\right)\right)$$

Kriterijumska funkcija za signal greske:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n (d_i - y_i)^2 = \frac{1}{2} \sum_{i=1}^n [d_i - a(\text{net}_i)]^2 = \frac{1}{2} \sum_{i=1}^n \left[ d_i - a\left(\sum_{q=1}^l w_{iq} z_q\right) \right]^2$$

Slicno kao kod Adaline, da bi se pronašao **minimum greske** trazi se izvod ove funkcije E i na osnovu toga se dobija **update za tezine izmedju skrivenog i output sloja** (u opstem slučaju izmedju preposlednjeg i poslednjeg sloja):

$$\Delta w_{iq} = -\eta \frac{\partial E}{\partial w_{iq}}$$

Iz konacnog racuna sa dobija:

$$\Delta w_{iq} = -\eta \left[ \frac{\partial E}{\partial y_i} \right] \left[ \frac{\partial y_i}{\partial \text{net}_i} \right] \left[ \frac{\partial \text{net}_i}{\partial w_{iq}} \right] = \eta [d_i - y_i] [a'(\text{net}_i)] [z_q] \triangleq \eta \delta_{oi} z_q$$

- $\delta_{oi}$  (cita se delta) je **signal greske**

- o znači **output** jer je u poslednjem sloju, a i znači **i-ti neuron** jer do njega idu te **tezine w** cija se greska gleda (videti sa slike)

- iz gornjeg izraza se vidi da je:

$$\delta_{oi} \triangleq -\frac{\partial E}{\partial \text{net}_i} = -\left[ \frac{\partial E}{\partial y_i} \right] \left[ \frac{\partial y_i}{\partial \text{net}_i} \right] = [d_i - y_i] [a'(\text{net}_i)]$$

- na ovu gresku uticu **очекivani izlaz** (d) i **stvarni izlaz** (y)
- ovo je isto kao kod Widrow-Hoff-ovog algoritma samo sto je kod njega drugi deo izraza (izvod a funkcije) jednak 1

Kada se nadju greske u poslednjem sloju pocinje **propagacija unazad**, tj. traze se update-i pojacanja za **prethodne slojeve** (redom unazad):

$$\Delta v_{qj} = -\eta \left[ \frac{\partial E}{\partial v_{qj}} \right] = \eta \delta_{hq} x_j$$

- signal greske za ovaj sloj:

$$\delta_{hq} = - \left[ \frac{\partial E}{\partial \text{net}_q} \right] = - \left[ \frac{\partial E}{\partial z_q} \right] \left[ \frac{\partial z_q}{\partial \text{net}_q} \right] = a'(\text{net}_q) \sum_{i=1}^n \delta_{oi} w_{iq}$$

- h znaci *hidden* jer je u skrivenom sloju, a q znaci (isto kao gore i) *q-ti neuron* do kojeg idu **tezine v**
- na ovu gresku uticu **greske odozgo**  $\delta_{oi}$  (gde i ide od 1 do n - broj neurona u izlaznom sloju) pomnozene sa svim **pojacanjima koja izlaze iz neurona q** (I to vazi I za propagaciju na svaki sledeci nivo (kada bi ih bilo vise))

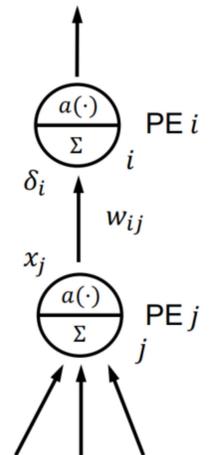
**Uopsteno delta pravilo obucavanja:**

$$\Delta w_{ij} = \eta \delta_i x_j$$

- $x_j$  je izlaz iz neurona j i ulaz u neuron i izmedju kojih se gleda konekcija  $w_{ij}$
- $\delta_i$  je signal obucavanja za prethodni sloj
- **$\Delta w_{ij} = \text{koefficijent obucavanja} * \text{greska} * \text{ulaz_u_neuron}$** 
  - iz ovoga se vidi da je promena pojacanjaj lokalna, jer koristi samo informacije sa obe strane veze

Opste predstavljanje mreze:  $y_j \rightarrow y_i$

- u gornjem levom uglu je **broj sloja** u kom smo
- u donjem desnom uglu je **broj neurona** u sloju
- kao I kod ostalih mreza, I ovde dodajemo da poslednji signal u ulazu predstavlja prag (**bias**) i on je jednak -1



**ALGORITAM:**

#### 0. *inicijalizacija*

- $\eta > 0$  jer nam treba **negativan gradijent**  $-\eta$
- bira se **Emax** - maksimalna dozvoljena greska, do koje kad dodjemo smatramo da se mreza dovoljno obucila
  - najbolje da se na pocetku stavi na 0, pa da se odredi kasnije kada mreza dodje do zasicenja (tj. kada se na grafiku priblici nuli ali osciluje iznad neke vrednosti I ne moze da se spusti dalje od nje - ta vrednost se bira za Emax)
  - to je **jedan od kriterijuma** koji sluze da se zaustavi obucavanje
- inicijalizujemo pojacanja na slucajne vrednosti, a pocetno E je 0

#### 1. *pocetak treniranja*

- racunamo izlaze za sve neurone prvog (ulaznog) sloja

#### 2. *propagacija unapred*

- **racuna se y** -> vrednosti output sloja

#### 3. *racunanje greske E i signala greske δ za poslednji sloj*

- na E se dodaje greska prethodnog pacijenta

$$E = \frac{1}{2} \sum_{i=1}^n (d_i^{(k)} - y_i)^2 + E$$

4. propagacija greske unazad
  - prvo se pronadju svi  $\delta$
  - onda se nadju svi update-i  $\Delta w$
5. provera kraja epohe
  - ako smo prosli kroz sve podatke (sve pacijente) prelazi se na sledeci korak -> **zavrsili smo jednu epohu**
  - **iteracija** - kada se prodje 1 pacijent, **epoha** - kada se prodju svi pacijenti
  - u suprotnom se vracamo na korak 1 gde nastavljamo treniranje dok ne prodjemo kroz sve (ako imamo 100 pacijenata vraticemo se 100 puta)
6. provera greske
  - ako je  $E < E_{\text{max}}$  to je kraj
  - u suprotnom se vracamo na Step 1 i zapocinje se nova epoha, i vracamo  $E$  na pocetnu vrednost 0
  - koristicemo ista pojacanja - ne inicializujemo nova

3 vrste algoritama opadajuceg gradijenta:

- **inkrementalni mod - SGD** - pojacanja se azuriraju **nakon svakog novog ulaza** u mrezu, tj. za svakog pacijenta (gore navedeni algoritam je zbog toga ovog tipa)
- **batch mod - GD** - prvo se primenjuju **svi ulazi** na mrezu, pa se onda ceo zbir dodaje za update (npr. pronadjemo  $\delta$  za prvog, pa za drugog itd.. i dobijemo  $\Delta w_9, 7 = \Delta w_9, 7(1) + \Delta w_9, 7(2) + \dots + \Delta w_9, 7(1000)$ ; i tek onda radimo **jedno azuriranje pojacanja**)

$$E = \frac{1}{2} (d - y)^2 \quad p \text{ je ceo skup podataka}$$

- **mini batch SGD** - isti princip kao obican batch mod samo sto se ovde ne primenjuje svih  $p$  ulaza odjednom, vec se ista stvar radi na manjem skupu ulaza (k iteracija)

$$E = \frac{1}{2} \sum_{i=1}^k (d^{(i)} - y^{(i)})^2, k < p$$

- $k=1$  za inkrementalni mod

Kod obucavanja moramo da pazimo na koji nacin pustamo ulaze u mrezu.

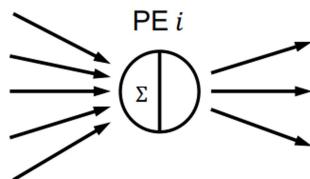
Na primer ako imamo 1000 pacijenata, i prvo unesemo 500 zdravih, mreza ce se navici na zdrave. Ako posle njih unesemo 500 bolesnih, mreza ce se navici na bolesne(imace i dalje neke informacije o zdravim, ali jako malo, jer su bolesni modifikovali pojacanja kako njima odgovara). I onda bi nastao problem kada bismo propustili zdravog pacijenta kroz mrezu, jer bi postojala veca verovatnoca da mreza pogresi. Zato je bitno da pri obucavanju propustamo cas zdravog, cas bolesnog, kako bi mreza modifikovala pojacanja u odnosu na obe grupe pacijenata.

Faktori obucavanja:

- pocetne tezine
- konstanta obucavanja
- kriterijumska funkcija (cost function)
- velicina i priroda skupa za obucavanje
- arhitektura mreze - broj slojeva i broj neurona u svakom sloju

## Inicijalizacija tezina:

- radi se da bi se izbegli sledeći problemi
  - **vanishing gradients** - kada se tezine ne menjaju tokom treniranja (slučaj kada su tezine inicijalizovane na broj mnogo manji od 1)
  - **exploding gradients** - kada se tezine mnogo brzo povecavaju (slučaj kada su tezine inicijalizovane na broj veci od 1)
- pravila:
  - ne treba da budu inicijalizovane na male slučajne vrednosti
  - ne treba da budu **jednake**
  - ne treba da budu **0**
  - treba da imaju dobro variranje
  - ne treba da budu prevelike, jer ce onda sigmoidi doci do **zasicanja** i zaglavice se na tome, pa necemo moci da vrsimo update tih pojacanja



$fan_{in}$        $fan_{out}$   
broj grana      broj grana  
na ulazu      na izlazu

- prva dva nacina nam ne trebaju na klk (Uniform i Xavier/Gorat inicijalizacije)
- **He Initialization** - dobre za mreze koje koriste **ReLU** aktivacionu funkciju

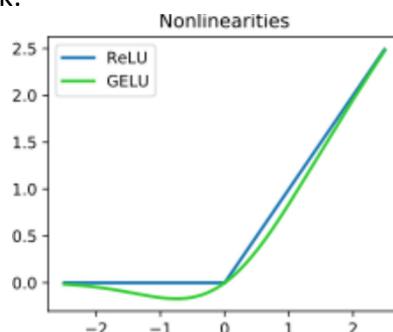
Normal

$$w_{ij} \sim Normal(0, \sigma), \quad \sigma = \sqrt{\frac{2}{fan_{in}}}$$

Uniform

$$w_{ij} \sim Uniform\left[-\frac{\sqrt{6}}{\sqrt{fan_{in}}}, +\frac{\sqrt{6}}{\sqrt{fan_{in}}}\right]$$

- ReLU - **rectified linear activation function** je linearna funkcija ciji ce izlaz biti jednak ulazu ukoliko je pozitivan, a ako je negativan bice 0
  - cesto se koristi u neuralnim mrezama zato sto je ovaj model lak za treniranje i ima dobre performanse
  - drugacije se naziva **funkcija ispravljalaca ili rampe**
  - grafik:



pojacanja koja se nalaze na levom delu grafika (tj. tamo gde su  $=0$ ) ne mozemo da update-ujemo, pa se cesto umesto strogog ravne linije na 0 koristi i **GeLu funkcija - glatka aproksimacija funkcije ReLu**

### Konstanta obucavanja $\eta$ :

- bitna za efektivnost i konvergenciju propagacije unazad
- bira se eksperimentalno za svako obucavanje, jer ne postoji jedna koja je pogodna za razlicite skupove podataka za obucavanje
- ako je jako veliko moze da dosta ubrza konvergenciju, ali takodje i da dovede do toga da se udje u zasicenje
- ista konstanta koja je izabrana na pocetku obucavanja ne mora da bude pogodna i kasnije - iako se zove konstanta, nije konstantna, jer se menja tokom citavog procesa (adaptira se)
- jedna metoda za odredjivanje  $\eta$  je da se odredi da li je update pojacanja **smanjio kriterijumsku funkciju E**:
  - ako nije, onda smo dosli do prekoracenja i  $\eta$  bi trebalo **da se smanji**
  - ako je u vise koraka doslo do smanjenja E, onda bi moglo da se proba povecanje  $\eta$
- trebalo bi da se  $\eta$  povecava prema sledecem pravilu:

$$\Delta\eta = \begin{cases} +a & \text{if } \Delta E < 0 \text{ consistently} \\ -b\eta & \text{if } \Delta E > 0 \\ 0 & \text{else} \end{cases}$$

- $\eta_{\text{new}} = \eta_{\text{old}} + \Delta\eta$
- $\Delta E$  je promena kriter. f-je
- $a>0, 1>b>0$
- **ADAGRAD optimizeri**
  - adaptive gradient algorithm
- $$w_t = w_{t-1} - \eta_{w_t} \frac{\partial E}{\partial w}$$

$$\eta_{w_t} = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

$$\alpha_t = \sum_{i=1}^t \left( \frac{\partial E}{\partial w_i} \right)^2$$
- $\varepsilon$  je mali pozitivan broj oko 0 koji sluzi da ako je  $\alpha=0$  ne bude deljenje sa 0
- $\eta$  je inicialna konstanta obucavanja
- $\eta$  se deli sa  $\alpha$  - **velicina promene pojacanja**, pa ako je promena veca  $\eta$  vise smanjujemo, a ako je manja  $\eta$  ostaje priblizno isto
- ovaj algoritam ima par nedostataka:
  - stopa efikasnosnog ucenja opada vrlo brzo zbog akumulacije kvadrata gradijentata
    - ovo je dobra stvar kada se primenjuje **na konveksnim funkcijama**, jer omogucava algoritmu da brzo konvergira
    - s druge strane, **nekonveksnim funkcijama** Adagrad moze smanjiti efektivnost tako da ona postane zaglavljena pre nego sto postigne lokalno konveksnu strukturu i los lokalni minimum
  - ako na pocetku stavimo velike tezine  $w$  onda mozemo **trajno da smanjimo  $\eta$**

- **ADADELTA + RMSprop optimizeri**

$$w_t = w_{t-1} - \eta_{w_t} \frac{\partial E}{\partial w_t}$$

$$\eta_{w_t} = \frac{\eta}{\sqrt{w_{avg}(t) + \varepsilon}}$$

$$w_{avg}(t) = \gamma w_{avg}(t-1) + (1 - \gamma) \left( \frac{\partial E}{\partial w(t)} \right)^2$$

$$\gamma = 0.95$$

### Kriterijumska funkcija:

- ne mora da bude samo kvadratna, moze da ima bilo koji stepen, samo je bitno da bude diferencijabilna

$$E = \frac{1}{p} \sum_i (d_i - y_i)^p \quad \text{where } 1 \leq p < \infty$$

- kriterijum najmanjeg kvadrata (L2 norma) koristi kvadratnu kriter. f-ju zbog svoje jednostavnosti
- **L<sup>∞</sup> norma:**

$$E^\infty = \sup_i |d_i - y_i|$$

- $\sup |x|$  je oznaka za **najvecu komponentu vektora**, jer kada se gleda beskonacni stepen ona je ta koja preuzima ceo zbir

$$\delta_{oi} = -\frac{\partial E}{\partial \text{net}_i} = \begin{cases} 0 & \text{if } i \neq i^* \\ a'(\text{net}_{i^*}) \text{sgn}(d_{i^*} - y_{i^*}) & \text{if } i = i^* \end{cases}$$

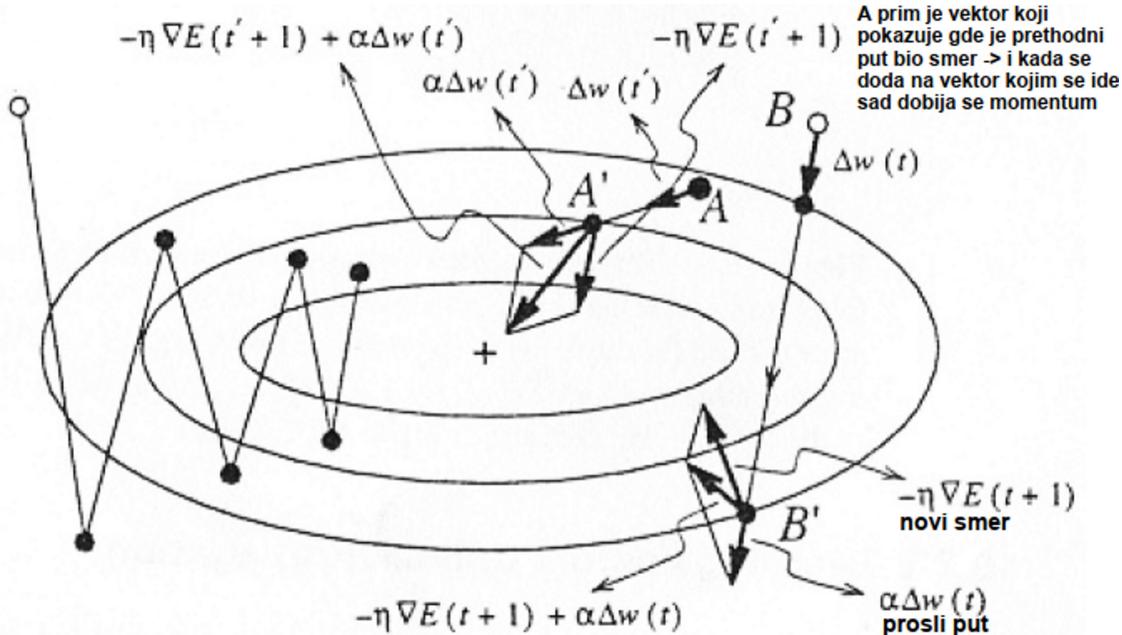
- $i^*$  je indeks najvece komponente output error vektora
- signal greske  $\delta_{oi}$  ukazuje na to da je jedina greska koja utice na propagaciju unazad je ona koja je najveca medju svim izlaznim cvorovima i koja se propagira od  $i^*$  tog izlaznog cvora

### Momentum:

- sluzi da se smanje oscilacije konstante obucavanja (da se one usrednje) i da se ubrza konvergencija
- svakoj tezini se dodaje inercija ili momentum, tako da **tezi da se promeni u smeru u kom se vec kretala** (sile koju oseca)

$$\Delta w(t) = -\eta \nabla E(t) + \alpha \Delta w(t-1)$$

- $\alpha$  je **parametar momentuma** i to je broj izmedju 0 i 1 (cesto se koristi 0.9)

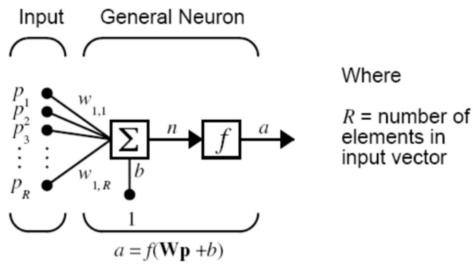


4 koraka u procesu obucavanja:

1. sastavi se skup podataka za obucavanje
2. napravi se mreza
3. vrsti se obucavanje
4. simulira se odgovor mreze na novim ulazima

## Arhitektura mreze :

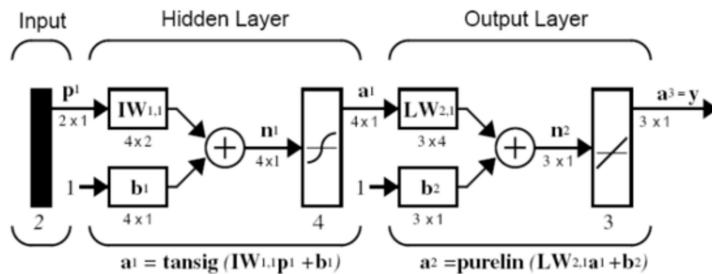
- model neurona:



Where

$R$  = number of elements in input vector

- na slajd su stavljene aktivacione funkcije koje se NE koriste (log-sigmoid, tan-sigmoid I linear function), jer cemo **uvek koristiti ReLu funkciju**
- feedforward mreza sa 2 sloja (hidden I output)
  - 2 ispod input dela znaci da je 2 broj ulaza



Po pravilu, podaci za obuku treba da pokriju **ceo ocekivani ulazni skup**, a zatim da se tokom procesa obuke nasumicno biraju training-vector parovi iz tog skupa

**Generalizacija - uopstavanje** je osobina da mreza izvaci znanje iz podataka, tj. sposobnost modela da dobro radi ne samo na podacima koji su korisceni za treniranje, vec i na novim, do sada nevidjenim podacima

- cilj je postici model koji može efikasno obraditi nove ulazne podatke umesto da jednostavno "memorise" oblike iz trening skupa
- back-propagation mreza je dobra za generalizaciju

**Preobucavanje (overfitting)** je pojava kada se mreza sa previse parametara za obuku za datu kolicinu podataka obuci dobro, ali ne radi generalizaciju. Tada je model previse prilagodjen trenirajucom skup I na njemu postize visoku tacnost, ali **za malu promenu ulaznih podataka veliki je stepen greske**. Medjutim, ukoliko se mrezi da premalo parametara za obuku, ona ne moze da uspe da se obuci.

Function of the Jacobian:

- tehnika koja sluzi da se poboljsa generalizacija, tako sto se izlaz cini **neosteljivim** na inkrementalne promene u ulazu ( $\partial x_j$ )

$$E_b = \frac{1}{2} \left( \frac{\partial E_f}{\partial x_1} \right)^2 + \frac{1}{2} \left( \frac{\partial E_f}{\partial x_2} \right)^2 + \dots + \frac{1}{2} \left( \frac{\partial E_f}{\partial x_m} \right)^2$$

- izvod kriterijumske f-je po nekom ulazu -> **funkcija osetljivosti** - kolika je promena Ef ukoliko se promeni odredjeni ulaz
- poenta je da ukoliko se malo promeni ulaz, kriterijumska funkcija Ef ne bi trebalo da se menja (npr. ako je mreza naucila da je neki objekat jedne nijanse crvene boje, ukoliko vidi drugu, malo drugaciju nijansu crvene boje ona ne zna da ga prepozna)
- ne treba da neki od ulaza bude dominantan

### Potkresivanje mreze - *pruning neural networks*:

- jos jedan nacin na koji moze da se spreci preobucavanje jeste da se zapocne obucavanje na vecoj mrezi, pa onda da se pokusa **sto vise da se smanji dimenzija** neuralne mreze
- **weight decay**
  - uklanjanje grana koje nisu od koristi tokom obucavanja, tj. one koje nemaju uticaja na smanjivanje greske
  - ako vidimo da se nista ne menja dok smanjujemo neko pojacanje mozda mozemo da ga ponistimo
  - to se radi na svakoj konakciji  $w_{ij}$  da bi imala tendenciju da opadne do 0 osim ukoliko se ne pojaca

$$w_{ij}(k+1) = -\eta \frac{\partial E}{\partial w_{ij}}(k) + \beta w_{ij}(k) \quad w_{ij}(k) \approx \beta^n w_{ij}(0)$$

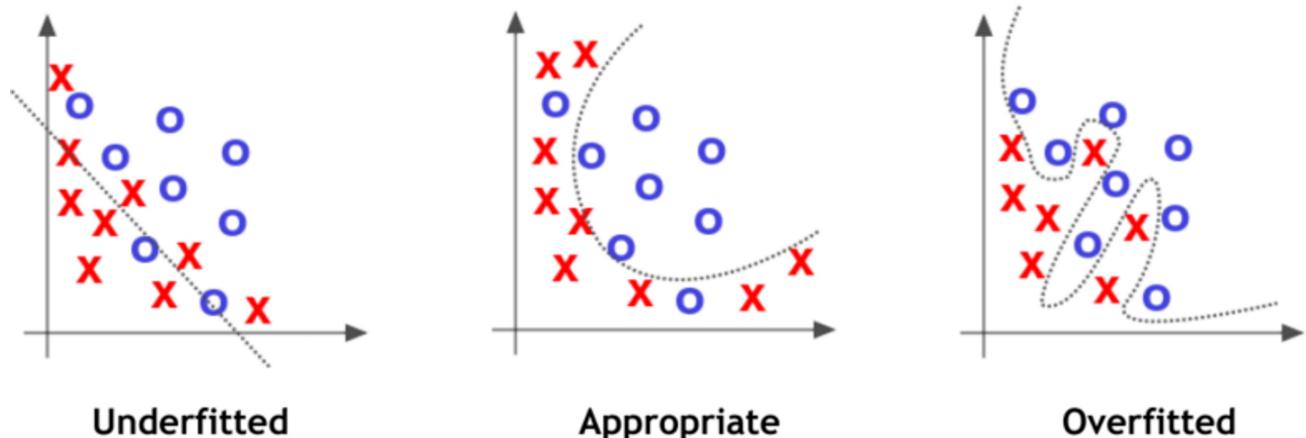
- $0 < \beta < 1$
- ako je npr.  $\beta = 0.9$  onda se  $w$  smanjuje u svakoj iteraciji na sledeci nacin:  
 $w(2)=w(1)*0.9=w(0)*0.9*0.9$
- ovo je ekvivalentno kao da se dodaje kvadrat pojacanja na originalnu kriterijumsku funkciju:

$$E' = E + \gamma \sum_{ij} w_{ij}^2 \quad \Delta w_{ij} = -\eta \partial E' / \partial w_{ij} \quad \beta = 1 - 2\gamma\eta$$

- bolje imati 100 manjih pojacanja, nego imati 99 malih i 1 veliko koje ce zauzeti najveci deo zbiru

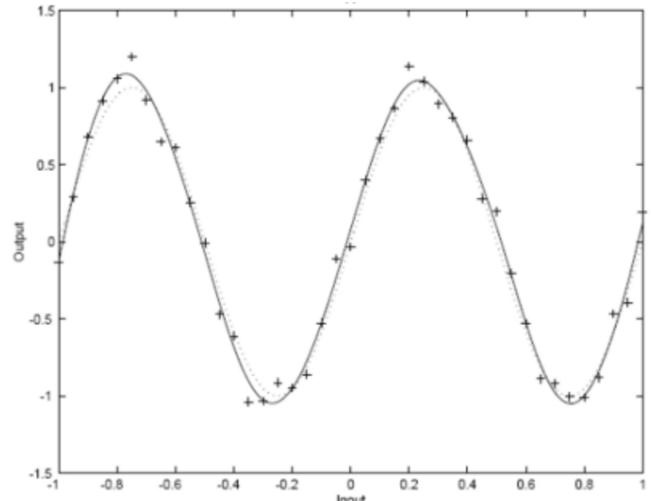
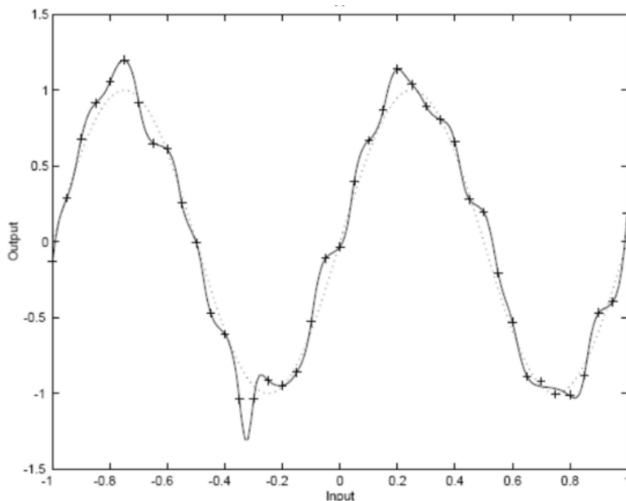
- **connection and node pruning**

## REGULARIZACIJA



- underfitted - kada se koristi perceptron - linearno
- overfitted - preobucena mreza

**Regularizacija** je pojam koji se odnosi na tehnike koje se koriste kako bi se sprecilo preobucavanje mreze. Pomaze u odrzavanju opste sposobnosti modela da se nosi s razlicitim podacima.



Tehnike regularizacije:

- **L2 regularizacija**

$$R(\boldsymbol{\theta}) = \sum_{i=1}^n \theta_i^2$$

- trudi se da sva pojacanja budu slična
- ovde je sa  $\theta$  označeno pojacanje
- **ravnomernost pojacanja**
- svi feature-i se uzimaju u obzir

$$\text{Loss } L(\mathbf{y}, \hat{\mathbf{y}}, \boldsymbol{\theta}) = \sum_{i=1}^n (x_i \theta_{ji} - y_i)^2 + \lambda R(\boldsymbol{\theta})$$

- **L1 regularizacija**

$$R(\boldsymbol{\theta}) = \sum_{i=1}^n |\theta_i|$$

$\theta_1 = [0, 0.75, 0] \rightarrow$  Ignores 2 features

$\theta_2 = [0.25, 0.5, 0.25] \rightarrow$  Takes information from all features

- **dropout**

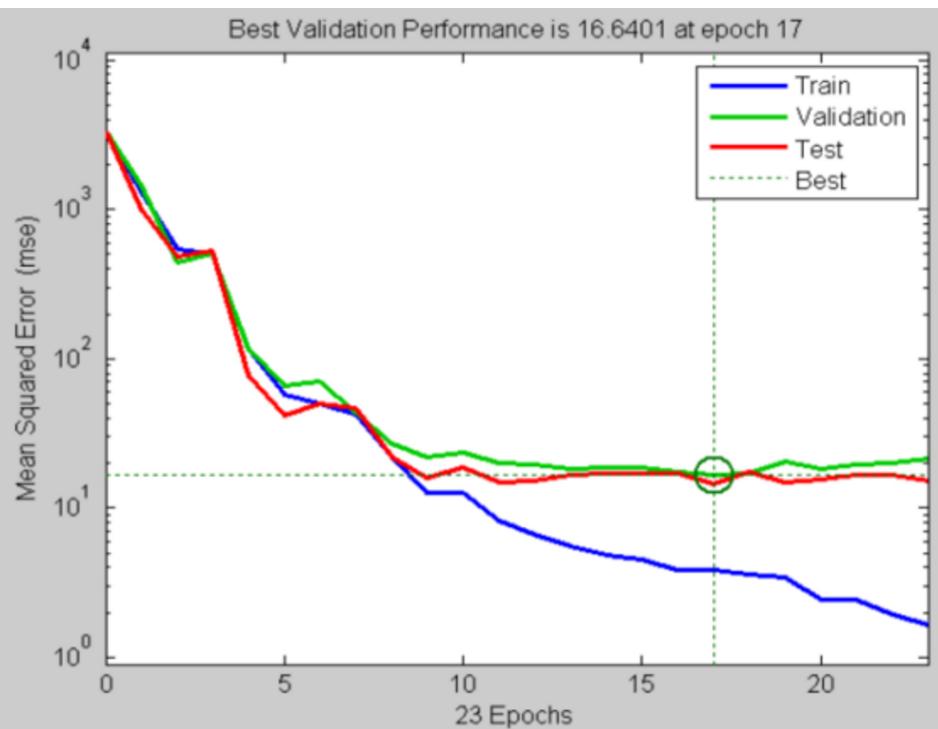
- tokom treninga, **nasumicno se isključuju** (postavljaju na nulu) određeni neuroni i veze između njih, i na taj način se sprecava da se mreža previse osloni na određene puteve ili karakteristike
- tokom svake iteracije, svaki neuron ima određenu **verovatnocu  $p$**  da bude isključen
  - verovatnoca je hiperparametar koji se postavlja prema potrebama problema, cesto se koristi vrednost između 0.2 i 0.5
- neuroni koji su isključeni ne doprinose treniranju i ne prenose informacije dalje u mreži
- dropout se obično primenjuje u fazama treniranja, dok se u fazi evaluacije (testiranja) svi neuroni uključuju, ali se izlazi umnožavaju faktorom  $1-p$  kako bi se kompenzovalo isključivanje neurona tokom treninga
- obucavanje će biti **duplo sporije**, ali se ovako omogućava da mreža **ima velike dimenzije** (puno neurona na ulazu) a ipak se znatno **smanjuje mogućnost preobucavanja**
- vrlo efikasno u praksi

- **early stopping**

- podaci se dele na 3 grupe



- podaci za validaciju **NE ucestvuju u update-u** vec se na njima samo gleda sta se desava sa greskom
  - **BITNO:** skup za validaciju **ucestvuje** u celokupnom obucavanju, ali posredno (moze da dodje kao pitanje na klk)
- podaci za testiranje su **potpuno nezavisni**
- osnovna ideja je **pracenje performansi modela na validacionom skupu** tokom treninga I zaustavljanje treniranja kada se performanse prestanu poboljsavati, tj. kada se na grafiku vidi da **kriva za validaciju raste X broj epoha** (koji smo odredili pre obucavanja)
- grafik:



- kako prepoznati na grafiku koja kriva je za koje podatke (moze da dodje kao pitanje na klk):
  - kriva za treniranje **uvek OPADA** I takodje je **najniza od svih**
  - izmedju krive za validaciju I testiranje: gledamo gde je krenulo da **RASTE** I ako vidimo da je zaustavljeno obucavanje tamo gde je preslo odredjeni X broj epoha rasta znamo da je to kriva za validaciju (npr. ovde na slici crvena kriva raste izmedju 11. i 16. epohe, ali posto se **tu ne zaustavlja** vidimo da je to skup za testiranje)