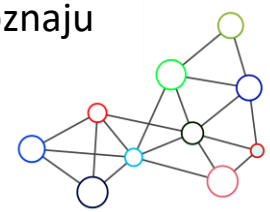
Neuronske mreže

- **Primene neuronskih mreža**
- **Perceptron, neuron i njihova ograničenja**
- **Neuronska mreža sa direktnom strukturom (eng. *feed-forward*)**
  - **Struktura**
  - **Primena u klasifikaciji i regresiji**
- Obuka neuronske mreže sa direktnom strukturom
  - Algoritam propagacije unazad (eng. *backpropagation*)
  - Uzoračka i *mini-batch* verzija algoritma propagacije unazad
- Izbor složenosti mreže
- Regularizacija
- Praktični saveti

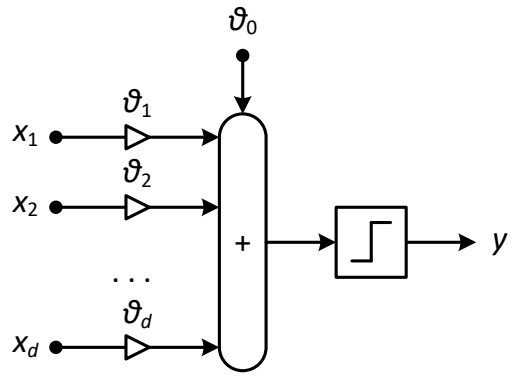
Primene neuronskih mreža

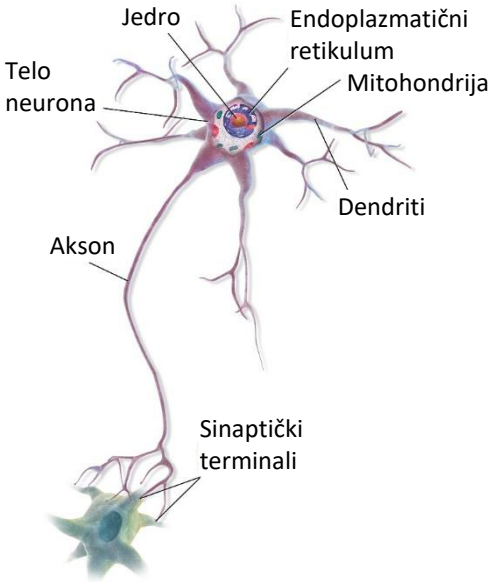


- Neuronske mreže su grafovski modeli za učenje iz podataka sposobni da prepoznaju obrasce, donose odluke i predviđaju rezultate u složenim zadacima
  - Rešavaju zadatke klasifikacije i regresije, klasterizacije i potpomognutog učenja
- Primene neuronskih mreža, između ostalog, obuhvataju sledeće:
  - Aproksimacija funkcija, modelovanje nizova podataka i predviđanje budućih vrednosti
  - Obrada podataka (vizuelizacija, filtriranje, kompresija, razdvajanje izvora samo na osnovu signala)
  - Prepoznavanje obrazaca (identifikacija i praćenje objekata sa slike ili videa, *data mining*)
  - Prepoznavanje sekvenci (govor, rukopis, štampani tekst, biometrijski podaci)
  - Upravljanje nelinearnim procesima i sistemima (robotika, navigacija, autonomna vozila)
  - Ekonomija i finansije (predviđanje cena akcija i kurseva, procena kreditnog rizika, detekcija prevara)
  - Generativna veštačka inteligencija (zvuk, slike, video, 3D modeli, hemijska jedinjenja)
  - Veliki jezički modeli (tekstualna ili govorna komunikacija čovek-mašina)
  - Kompjuterski potpomognuta dijagnostika (medicinska slika i signali, personalizovana medicina)
- Neuronske mreže daleko nadmašuju druge modele mašinskog učenja u slučajevima:
  - Kada je problem dinamičan, zavisan od konteksta i suviše složen da bi se mogle identifikovati eksplicitne zakonitosti
  - Kada je na raspolaganju **izuzetno velika količina podataka za obuku**

Perceptron (podsećanje)

- Jednostavan koncept mašine za učenje namenjene rešavanju problema linearne klasifikacije, koja daje dobre rezultate ako su klase linearno razdvojive
  - Perceptron izračunava skalarnu funkciju više ulaznih promenljivih x_i pomnoženih težinskim faktorima
    - U slučaju binarne klasifikacije, uzorak se klasifikuje u jednu od dve klase na osnovu znaka izlaza, što se može predstaviti step-funkcijom ($y \in \{0, 1\}$ ili $y \in \{-1, 1\}$)
  - Perceptron je u stanju da automatski odredi vrednosti težinskih koeficijenata na osnovu uzoraka, koristeći algoritam *perceptronskog učenja*
- Ovako definisan perceptron oponaša funkcionisanje biološkog neurona, koji:
  - prima ulazne signale preko većeg broja ulaza
  - sabira ih, pri čemu svaki ulaz utiče na zbir u meri koju određuje jačina veze sa neuronom koji šalje signal posmatranom neuronu (sinaptička težina)
  - ako zbir pređe određeni prag, šalje se određeni signal (tzv. akcioni potencijal) narednim neuronima

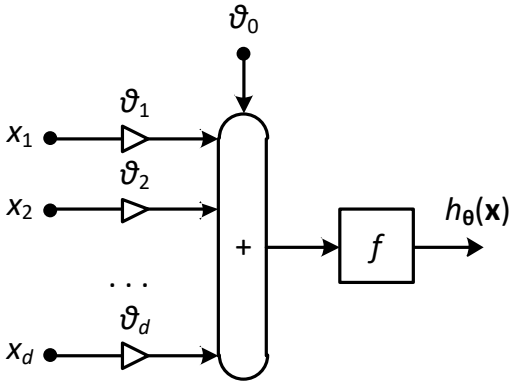




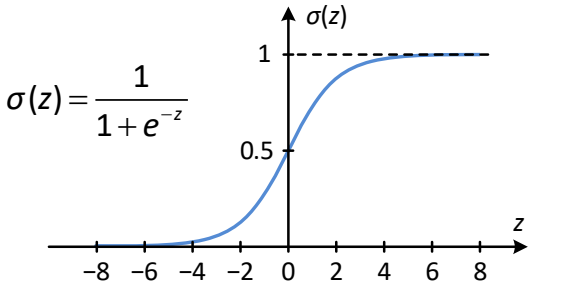
Neuron (podsećanje)

- Perceptron nadograđen uvođenjem neke druge, po pravilu neprekidne nelinearne funkcije na izlazu
- Neuron formira skalarnu funkciju ulaza x_i, od kojih se svaki množi odgovarajućim težinskim faktorom ϑ_i
  - Izlaz neurona (njegova *aktivacija*) predstavlja nelinearnu funkciju ovako dobijene linearne kombinacije $\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = f(\vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2 + \ldots + \vartheta_d x_d) = f(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x})$$



  - Nelinearnost f naziva se *aktivacionom funkcijom*
  - Neuron sa sigmiodnom aktivacionom funkcijom zapravo vrši *logističku regresiju*
  - Binarna klasifikacija se vrši na osnovu praga, dok za više klasa treba koristiti više neurona



- Kao i u slučaju (pojedinačnog) perceptrona:
  - Postoji pouzdan i efikasan algoritam za obuku
  - Nije moguće realizovati nelinearnu granicu odlučivanja, odnosno, opisati složenije interakcije između obeležja

---

U nastavku će se, iz praktičnih razloga, pod terminom neuron podrazumevati i perceptron (perceptron će se smatrati specijalnim slučajem neurona sa step aktivacionom funkcijom). Većina zaključaka je ista, a razlika dolazi do izražaja tek u kontekstu obuke neuronskih mreža.

Ograničenja neurona kao klasifikatora

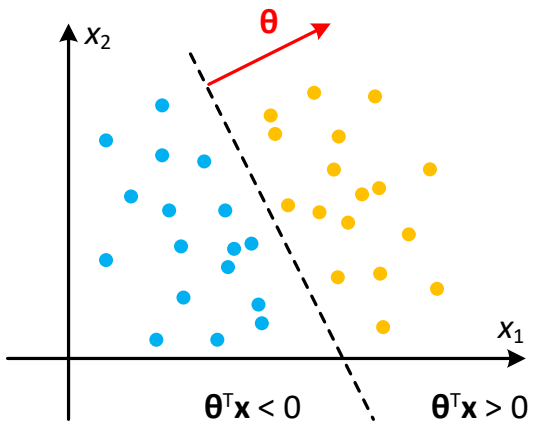
- Pojedinačni neuron uspešno vrši klasifikaciju samo u jednostavnom slučaju linearno razdvojivih klasa:

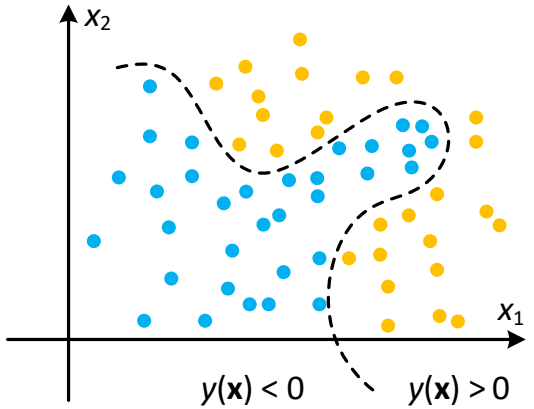
$$y(\mathbf{x}) = f(\vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2) = f(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x})$$

- Jedan način da se ostvari složenija, nelinearna klasifikacija bio bi da se i različiti kvadratni, kubni, kao i članovi viših stepena uključe u regresiju:

$$y(\mathbf{x}) = f(\vartheta_0 + \vartheta_1 x_1 + \vartheta_2 x_2 + \vartheta_3 x_1 x_2 + \vartheta_4 x_1^2 + \vartheta_5 x_2^2 + \ldots)$$

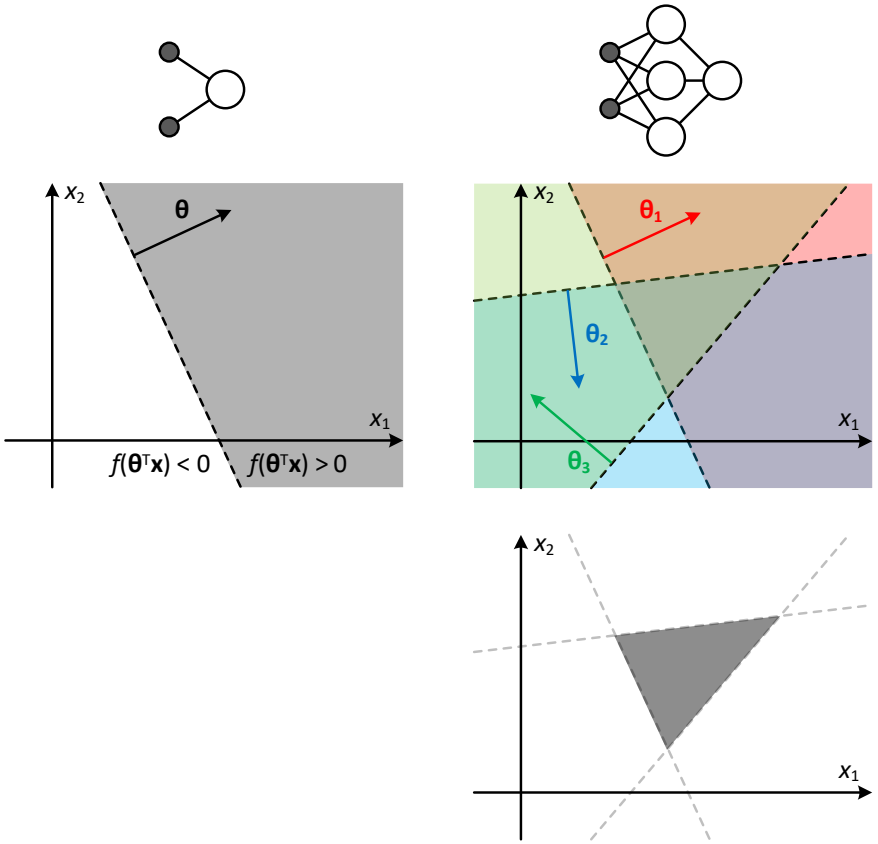
- Sa porastom broja dimenzija prethodni pristup postaje izuzetno nepraktičan
  - Kombinovanje više neurona i formiranje odgovarajućih mreža predstavlja efikasnu alternativu
  - Pristup zasnovan na formiranju mreže oponaša način povezivanja i funkcionisanja bioloških neurona
  - Taj pristup nije nov, ali široku primenu doživljava tek relativno nedavno, zahvaljujući tehnološkom razvoju računarske industrije, i predstavlja *state of the art* za mnoge praktične probleme





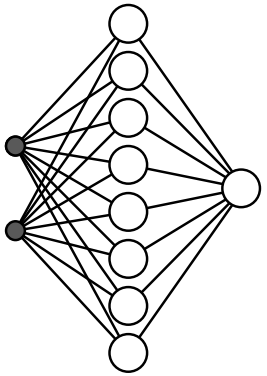
Ograničenja neurona kao klasifikatora

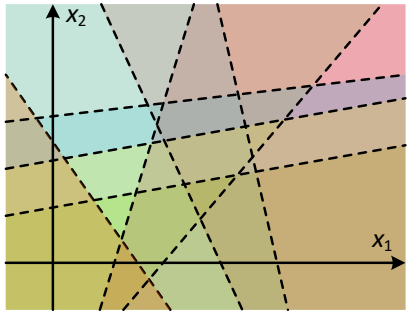
- Iako pojedinačni neuron ne može da realizuje nelinearnu granicu odlučivanja, ona se može realizovati kombinovanjem više neurona
- U prikazanom primeru svaki neuron prvog sloja postavlja po jednu hiperravan, a konačnu odluku o tome da li uzorak pripada nekoj od oblasti ograničenih tim hiperravnima donosi neuron u sledećem sloju

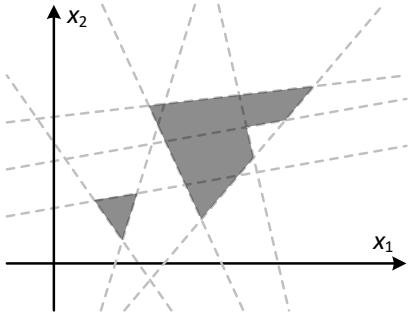


Ograničenja neurona kao klasifikatora

- Već sa jednim dodatnim slojem teorijski se može aproksimirati bilo koja granica odlučivanja
  - O ovome govori teorema univerzalne aproksimacije (1989), koja je dokazana za široku klasu aktivacionih funkcija
- U praksi bi aproksimacija proizvoljne granice odlučivanja samo jednim dodatnim slojem često zahtevala vrlo širok sloj (vrlo velik broj neurona u njemu) pa se umesto toga radije uvodi više dodatnih slojeva
  - Izbor broja slojeva i broja neurona u pojedinim slojevima predstavlja otvoren problem

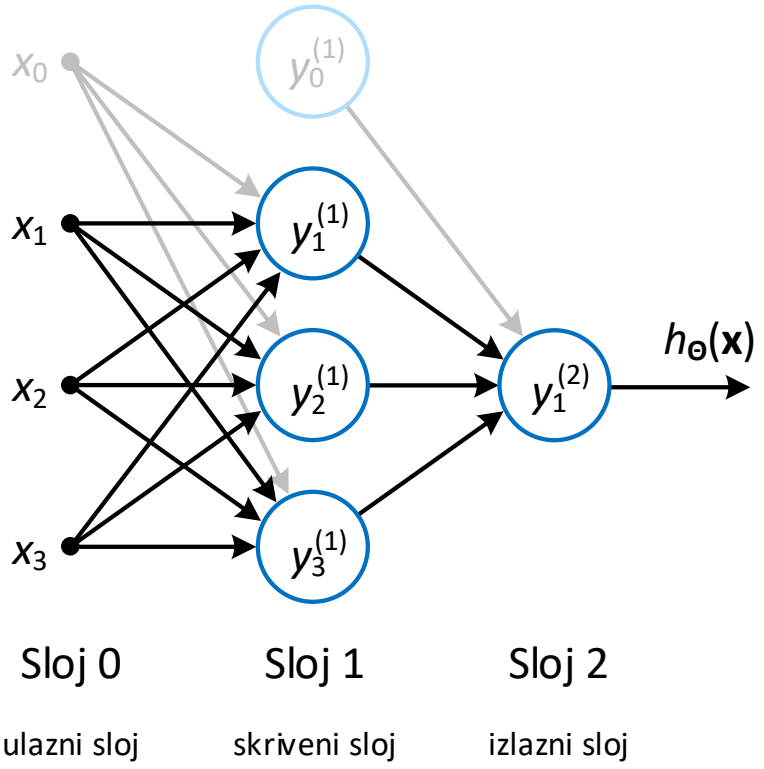






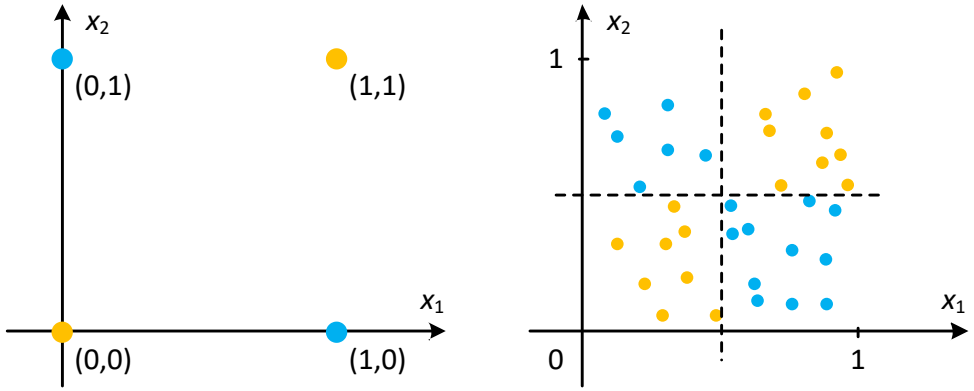
Primer neuronske mreže s jednim skrivenim slojem

- Naziv *skriveni sloj* odnosi se na to da, za razliku od ulaznog i izlaznog sloja, izlazi iz skrivenih slojeva nisu vidljivi u skupu za obuku
- $y_j^{(r)}$ označava aktivaciju (izlaz) j-tog neurona u r-tom sloju
  - Izlaz neuronske mreže $h_\Theta(\mathbf{x})$ predstavlja aktivaciju izlaznog neurona $y_1^{(2)}$, odnosno, $h_\Theta(\mathbf{x}) = y_1^{(2)}$
  - Ulaz neuronske mreže $\mathbf{x}$ je ujedno aktivacija ulaznog sloja, odnosno, $x_j = y_j^{(0)}$
  - Ulazno obeležje x_0 uvek je 1, što se podrazumeva (i ne prikazuje)
  - Nulti ulaz određenog neurona može se tretirati kao izlaz nultog neurona iz prethodnog sloja



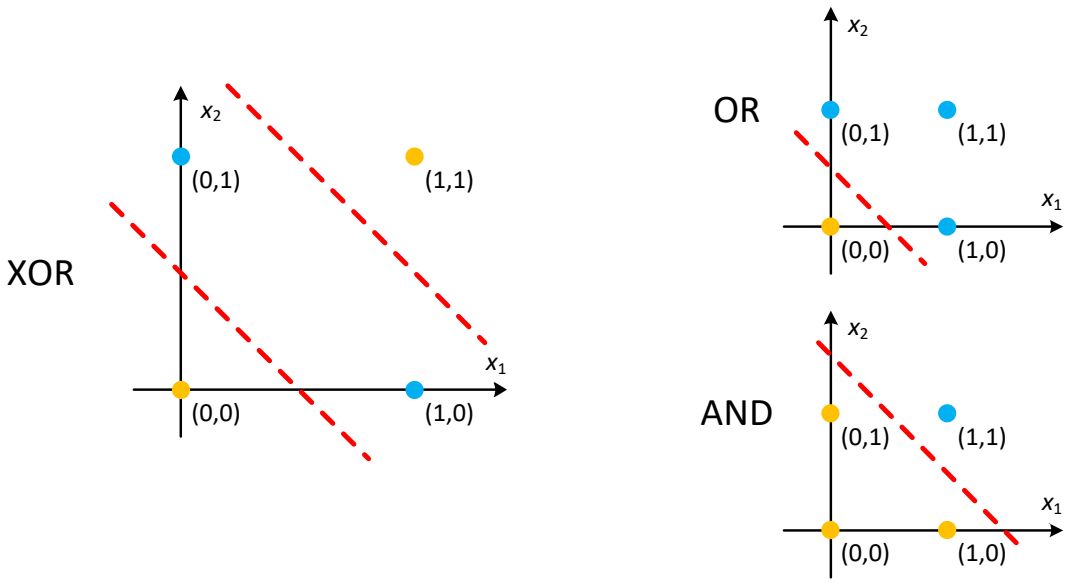
Primer nelinearne klasifikacije (XOR problem)

- Tipičan primer nelinearnog problema koji linearni klasifikator ne može da reši je Booleova funkcija *„ekskluzivno ili“* (eng. XOR) nad dva binarna obeležja
  - Ovo se može posmatrati i kao jednostavnija verzija sličnog nelinearnog problema sa kontinualnim obeležjima

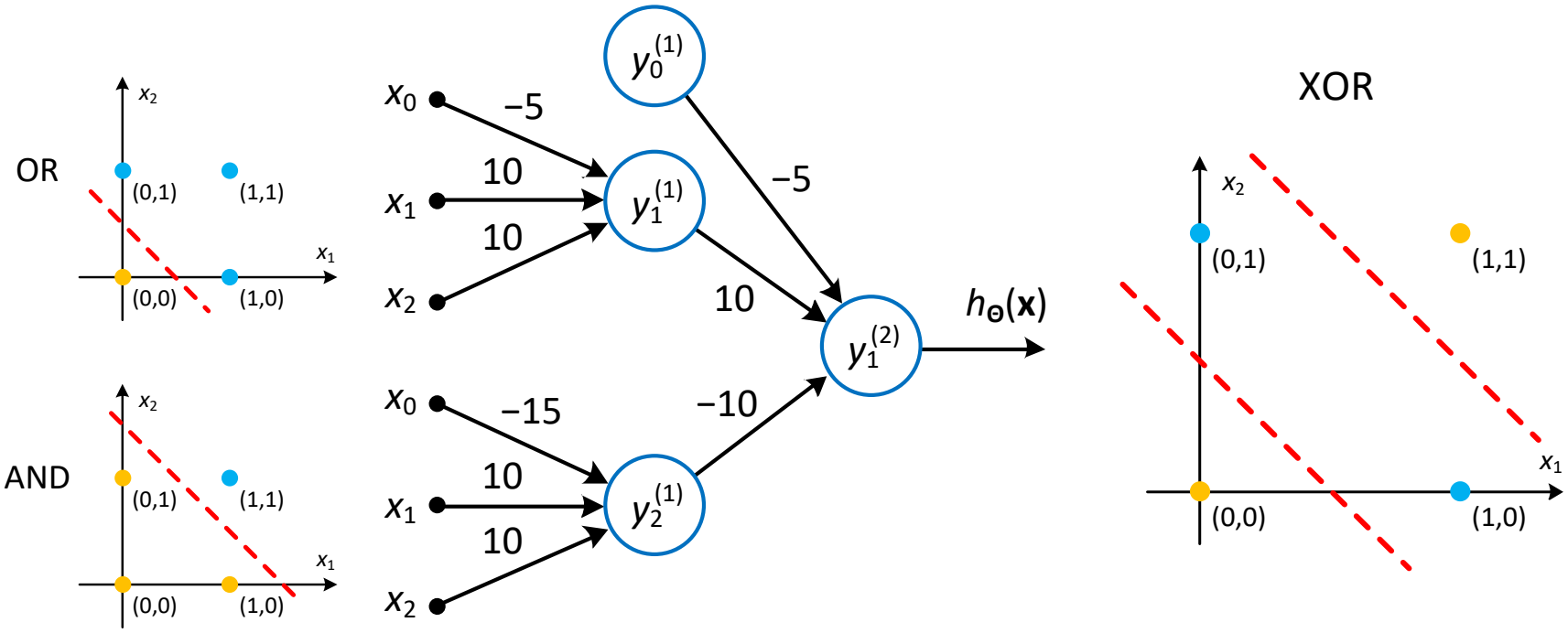


Primer nelinearne klasifikacije (XOR problem)

- Tipičan primer nelinearnog problema koji linearni klasifikator ne može da reši je Booleova funkcija „*ekskluzivno ili*“ (eng. XOR) nad dva binarna obeležja
  - Ovo se može posmatrati i kao jednostavnija verzija sličnog nelinearnog problema sa kontinualnim obeležjima
  - Problem se može rešiti u dva koraka, kombinovanjem rezultata dva linearna klasifikatora koji realizuju funkcije „*i*“ (eng. AND) i „*ili*“ (eng. OR) funkcije



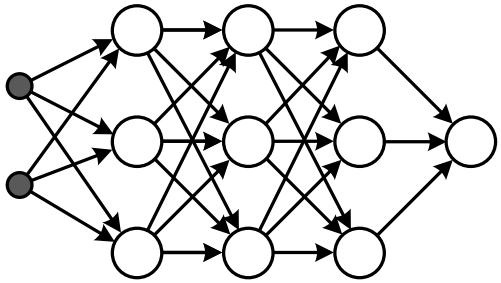
Primer nelinearne klasifikacije (XOR problem)



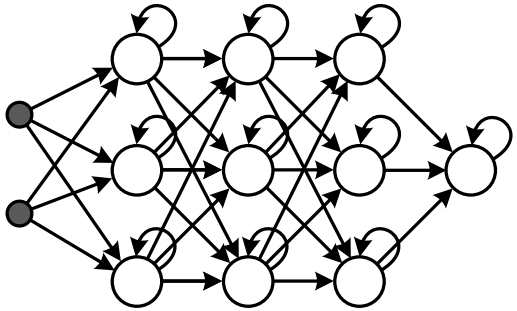
- Umesto navedenih težinskih faktora mogli su poslužiti i drugi pozitivni i negativni brojevi u sličnom relativnom odnosu
  - Ako se na izlazu koristi aktivaciona funkcija kao što je npr. sigmoid, težinski faktori morali bi biti dovoljno veliki da je odvedu u zasićenje

Različite arhitekture neuronskih mreža

- Mreža može imati i više skrivenih slojeva, kao i direktne veze između nesusednih slojeva, povratne veze...
  - Broj slojeva naziva se i *dubinom* mreže



neuronska mreža
sa direktnom strukturom



rekurentna neuronska mreža

- Postoje posebne arhitekture mreža, kao i tipovi neurona, namenjeni specifičnim zadacima

Primer neuronske mreže s jednim skrivenim slojem

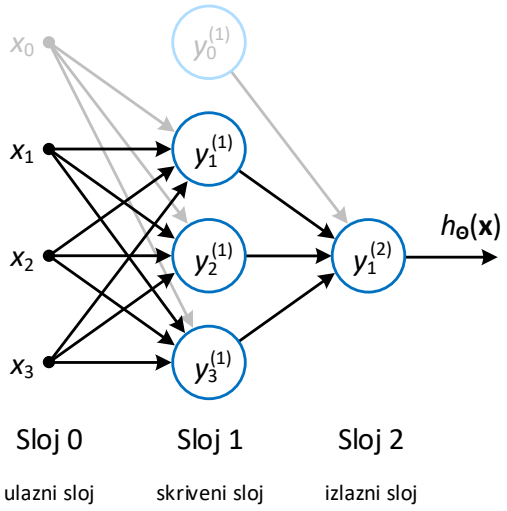
- $\boldsymbol{\Theta}^{(r)}$ je matrica težinskih (sinaptičkih) faktora između slojeva $r-1$ i r
  - Ako sloj r ima k_r neurona a sloj $r-1$ ima k_{r-1} neurona, dimenzije ove matrice su $k_r \times (k_{r-1}+1)$

$$y_1^{(1)} = f(\vartheta_{10}^{(1)} x_0 + \vartheta_{11}^{(1)} x_1 + \vartheta_{12}^{(1)} x_2 + \vartheta_{13}^{(1)} x_3) = f(z_1^{(1)})$$
$$y_2^{(1)} = f(\vartheta_{20}^{(1)} x_0 + \vartheta_{21}^{(1)} x_1 + \vartheta_{22}^{(1)} x_2 + \vartheta_{23}^{(1)} x_3) = f(z_2^{(1)})$$
$$y_3^{(1)} = f(\vartheta_{30}^{(1)} x_0 + \vartheta_{31}^{(1)} x_1 + \vartheta_{32}^{(1)} x_2 + \vartheta_{33}^{(1)} x_3) = f(z_3^{(1)})$$

  - ($\boldsymbol{\Theta}^{(1)}$ je dimenzija 3×4)

$$y_1^{(2)} = f(\vartheta_{10}^{(2)} y_0^{(1)} + \vartheta_{11}^{(2)} y_1^{(1)} + \vartheta_{12}^{(2)} y_2^{(1)} + \vartheta_{13}^{(2)} y_3^{(1)}) = f(z_1^{(2)})$$

($\boldsymbol{\Theta}^{(2)}$ je dimenzija 1×4)

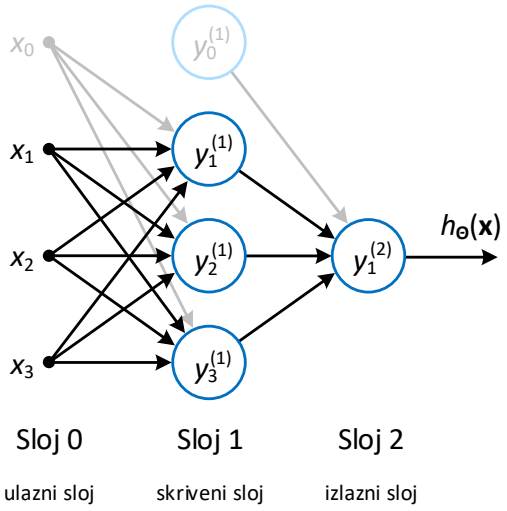


- Prikazani postupak naziva se *propagacija unapred*
- Matrice $\boldsymbol{\Theta}^{(r)}$ definišu ponašanje mreže i podešavaju se u procesu obuke
  - Radi se o ranom nadgledanom učenju, kao i u slučaju pojedinačnih perceptrona, odnosno, neurona

Primer neuronske mreže s jednim skrivenim slojem

- $\boldsymbol{\Theta}^{(r)}$ je matrica težinskih (sinaptičkih) faktora između slojeva $r-1$ i r
  - Ako sloj r ima k_r neurona a sloj $r-1$ ima k_{r-1} neurona, dimenzije ove matrice su $k_r \times (k_{r-1}+1)$

$$y_1^{(1)} = f(\vartheta_{10}^{(1)} x_0 + \vartheta_{11}^{(1)} x_1 + \vartheta_{12}^{(1)} x_2 + \vartheta_{13}^{(1)} x_3) = f(z_1^{(1)})$$
$$y_2^{(1)} = f(\vartheta_{20}^{(1)} x_0 + \vartheta_{21}^{(1)} x_1 + \vartheta_{22}^{(1)} x_2 + \vartheta_{23}^{(1)} x_3) = f(z_2^{(1)})$$
$$y_3^{(1)} = f(\vartheta_{30}^{(1)} x_0 + \vartheta_{31}^{(1)} x_1 + \vartheta_{32}^{(1)} x_2 + \vartheta_{33}^{(1)} x_3) = f(z_3^{(1)})$$



  - ($\boldsymbol{\Theta}^{(1)}$ je dimenzija 3×4) $\quad \mathbf{y}^{(1)} = f(\mathbf{z}^{(1)}) = f(\boldsymbol{\Theta}^{(1)}\mathbf{x})$

$$y_1^{(2)} = f(\vartheta_{10}^{(2)} y_0^{(1)} + \vartheta_{11}^{(2)} y_1^{(1)} + \vartheta_{12}^{(2)} y_2^{(1)} + \vartheta_{13}^{(2)} y_3^{(1)}) = f(z_1^{(2)})$$

($\boldsymbol{\Theta}^{(2)}$ je dimenzija 1×4) $\quad h_{\boldsymbol{\Theta}}(\mathbf{x}) = \mathbf{y}^{(2)} = f(\mathbf{z}^{(2)}) = f(\boldsymbol{\Theta}^{(2)}\mathbf{y}^{(1)})$

- Prikazani postupak naziva se *propagacija unapred*
- Matrice $\boldsymbol{\Theta}^{(r)}$ definišu ponašanje mreže i podešavaju se u procesu obuke
  - Radi se o ranom nadgledanom učenju, kao i u slučaju pojedinačnih perceptrona, odnosno, neurona

Klasifikacija i regresija pomoću neuronske mreže

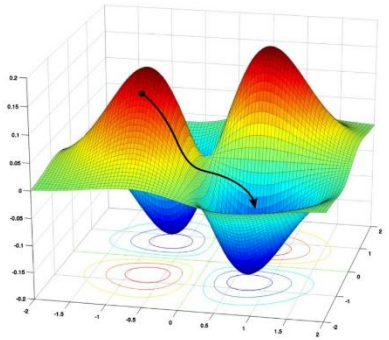
- U slučaju klasifikacije u dve klase, izlazni sloj sadrži jedan neuron i uzorak se klasifikuje prema tome da li je izlaz iznad ili ispod određenog praga
  - Uzorcima iz skupa za obuku pridružuju se izlazne vrednosti 0 i 1 (ili −1 i 1), u zavisnosti od toga kojoj klasi uzorak pripada
- U slučaju klasifikacije u K klasa ($K > 2$) izlazni sloj sadrži K neurona i uzorak se klasifikuje u onu klasu čiji je neuron dao najveći odziv
  - U idealnom slučaju trebalo bi da izlazni neuron koji se odnosi na određenu klasu dâ odziv 1, a da svi ostali daju odziv 0
  - Shodno tome, za svaki uzorak $\mathbf{x}$ iz skupa za obuku odgovarajuća oznaka klase $\mathbf{y}$ biće formirana u obliku vektora dužine K:

$$\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \ldots \quad \text{ili} \quad \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

- Neuronske mreže mogu se koristiti i za regresiju, odnosno, predviđanje vrednosti nepoznate kontinualne veličine (ili više njih)
  - Uzorak za obuku predstavlja uređeni par ($\mathbf{x}$, $\hat{y}$), gde je $\hat{y}$ vrednost izlazne veličine

Od perceptrona do obuke neuronskih mreža

- Perceptron (1958)
  - Povezivanjem više perceptrona u mreže (primer XOR funkcije) teoretski se mogu ostvariti složene nelinearne granice odlučivanja
  - Ne postoji efikasan algoritam za obuku kao u slučaju pojedinačnog perceptrona
    - Funkcija greške je konstantna osim na mestima gde ima skokove
    - Iterativne procedure kao što je gradijentni silazak ne funkcionišu
    - Objavljivanje knjige *Perceptrons* (Minsky & Papert, 1969), koja ukazuje na ovaj problem, dovelo je do toga da dve decenije istraživački fokus u ovoj oblasti ode u drugim smerovima, koji na kraju takođe nisu doveli do željenih rezultata
- Neuron sa sigmoidnom aktivacijom (kraj 1960-ih)
  - Korišćen za istraživanja nervnog sistema i modelovanje sistema sa kontinualnim izlazom
    - Razvoj u oblasti u to vreme bio je otežan nedovoljnim nivoom tehnološkog razvoja
- Efikasan algoritam obuke neuronskih mreža (1986)
  - Pošto je aktivaciona funkcija diferencijabilna, to važi i za funkciju greške na nivou mreže, pa se može primeniti iterativni algoritam za njenu minimizaciju
    - Do ovog rezultata došlo se već početkom 1970-ih, ali nije stekao široku popularnost jer se u to vreme istraživačka zajednica fokusirala u drugim pravcima



Obuka neuronske mreže

- Neka mreža ima ukupno L slojeva i neka u r-tom sloju ima k_r neurona
- Neka je dostupno N parova za obuku $(\mathbf{x}(i), \hat{\mathbf{y}}(i))$:

$$\mathbf{x}(i) = \begin{bmatrix} x_1(i) \\ x_2(i) \\ \vdots \\ x_{k_0}(i) \end{bmatrix} \qquad \hat{\mathbf{y}}(i) = \begin{bmatrix} \hat{y}_1(i) \\ \hat{y}_2(i) \\ \vdots \\ \hat{y}_{k_L}(i) \end{bmatrix}$$

- Ako bi se, za vreme obuke, vektor $\mathbf{x}(i)$ postavio na ulaz, bio bi dobijen izlaz $\mathbf{y}(i)$, koji bi se u opštem slučaju razlikovao od $\hat{\mathbf{y}}(i)$
- Sinaptičke težine biraju se **tako da se minimizuje odgovarajuća funkcija cene *J*** (funkcija greške) koja zavisi od vrednosti $\mathbf{y}(i)$ i $\hat{\mathbf{y}}(i)$
  - Funkcija cene zavisi od težina preko $\mathbf{y}(i)$ i odgovarajuća zavisnost je nelinearna i obično vrlo složena, pa se za minimizaciju funkcije cene najčešće koriste iterativne tehnike
- Neka je $\boldsymbol{\theta}_j^{(r)} = \begin{bmatrix} \vartheta_{j0}^{(r)} & \vartheta_{j1}^{(r)} & \dots & \vartheta_{jk_{r-1}}^{(r)} \end{bmatrix}^{\mathrm{T}}$ težinski vektor j-tog neurona u r-tom sloju
- Osnovni korak iteracije biće:

$$\boldsymbol{\theta}_j^{(r)}(\text{novo}) = \boldsymbol{\theta}_j^{(r)}(\text{staro}) + \Delta\boldsymbol{\theta}_j^{(r)} = \boldsymbol{\theta}_j^{(r)}(\text{staro}) - \alpha \frac{\partial J}{\partial \boldsymbol{\theta}_j^{(r)}}$$

Izbor funkcije cene

- Logično je usredsrediti se na funkcije cene koje imaju oblik:

$$J = \sum_{i=1}^{N} \varepsilon(i),$$

gde je $\varepsilon(i)$ mera odstupanja $\mathbf{y}(i)$ od $\hat{\mathbf{y}}(i)$, odnosno, mera greške na i-tom uzorku

- Čest izbor funkcije cene je suma (ili prosek) kvadratnih odstupanja $\mathbf{y}(i)$ od $\hat{\mathbf{y}}(i)$:

$$J = \frac{1}{2}\sum_{i=1}^{N}\sum_{m=1}^{k_L} e_m^2(i) = \frac{1}{2}\sum_{i=1}^{N}\sum_{m=1}^{k_L} (y_m(i) - \hat{y}_m(i))^2$$

kao i međuentropijska funkcija cene:

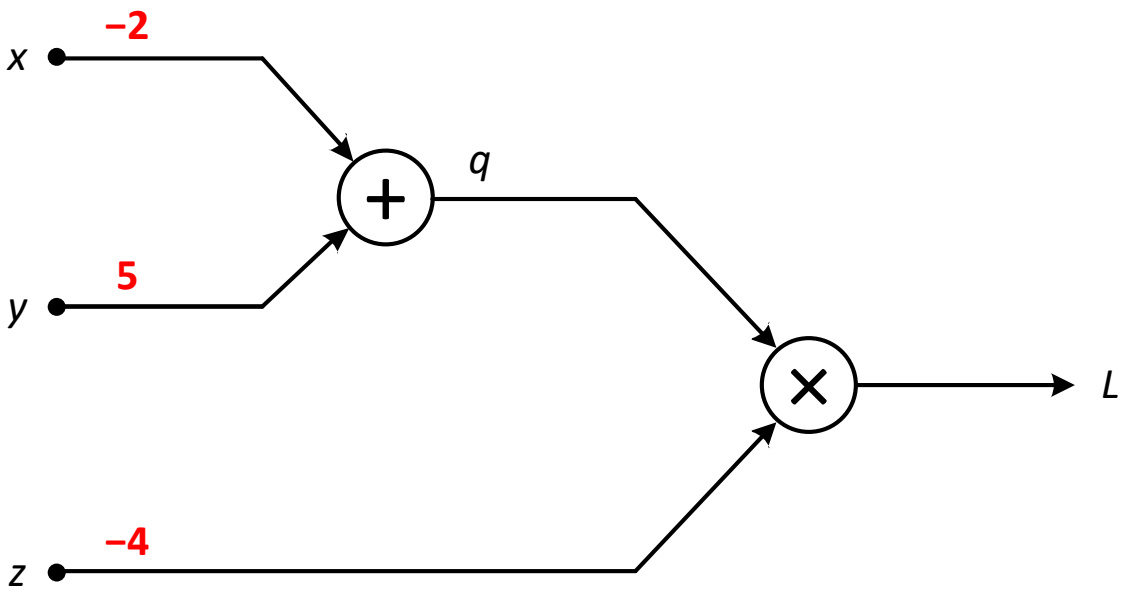
$$J = -\frac{1}{N}\left[\sum_{i=1}^{N}\sum_{m=1}^{k_L} y_m(i)\ln(h_\Theta(\mathbf{x}(i)))_m + (1-y_m(i))\ln(1-(h_\Theta(\mathbf{x}(i)))_m)\right] + \frac{\lambda}{2N}\sum_{r=1}^{L-1}\sum_{i=1}^{k_r}\sum_{j=1}^{k_{r+1}}(\vartheta_{ji}^{(r)})^2,$$

koja ravnomernije uzima u obzir uticaje izlaza čiji su dinamički opsezi različiti

- Pošto je potrebno minimizovati funkciju cene u odnosu na sve težinske faktore u mreži, treba analizirati uticaj svakog od njih na nju

* U ovom slučaju, polovina sume kvadratnih odstupanja, ali pozitivan konstantan faktor svakako ne utiče na rezultat minimizacije

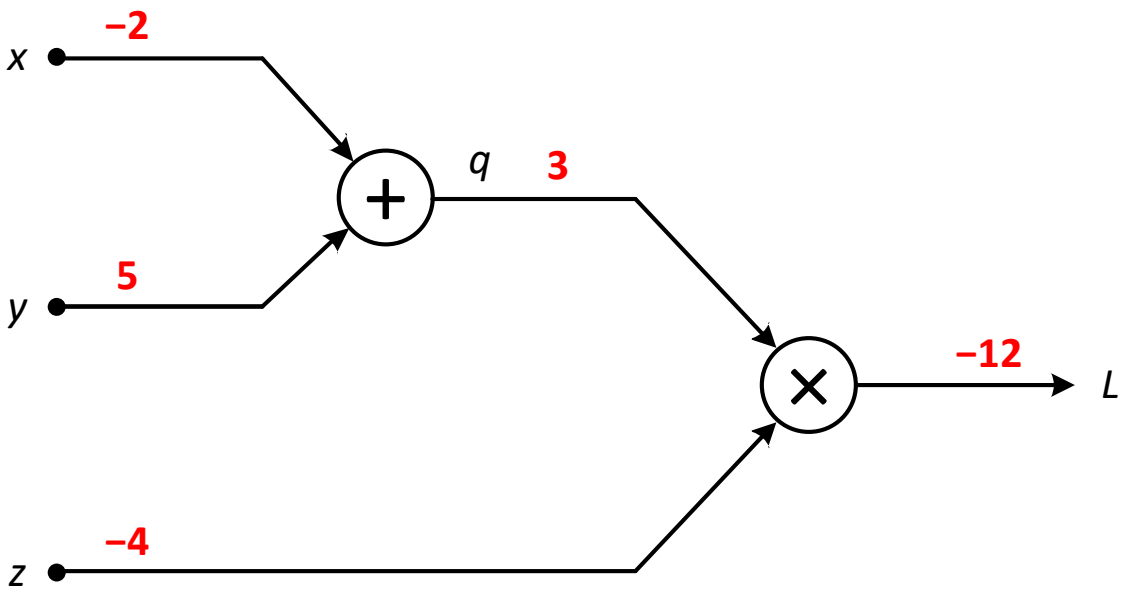
Uticaj promene ulaza na izlaz (primer 1)



$$L = qz$$

$$q = x + y$$

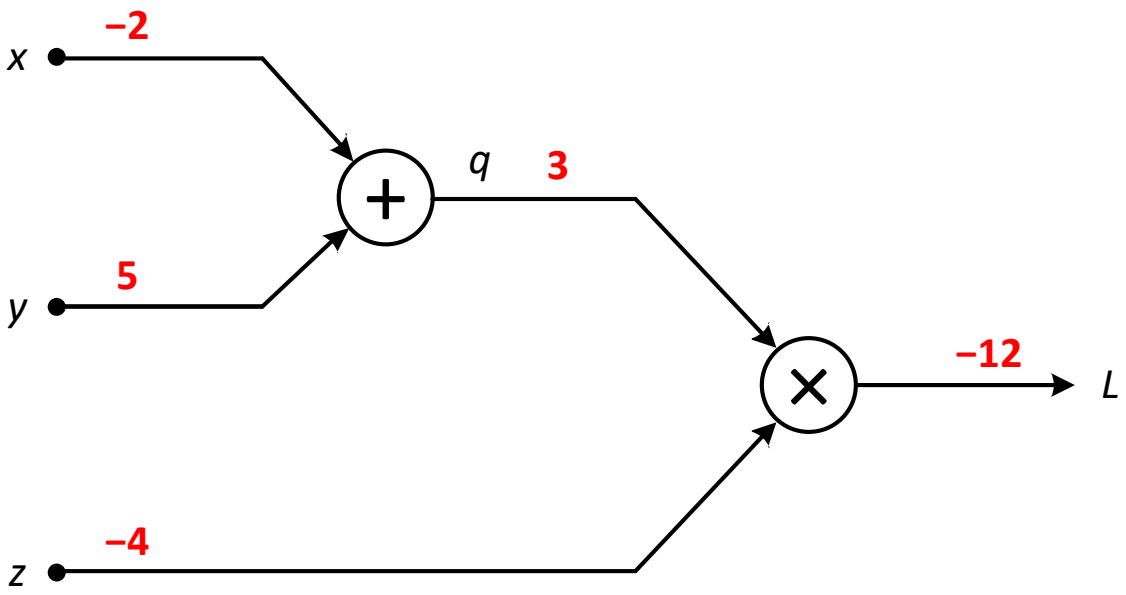
Uticaj promene ulaza na izlaz (primer 1)



$$L = qz$$

$$q = x + y$$

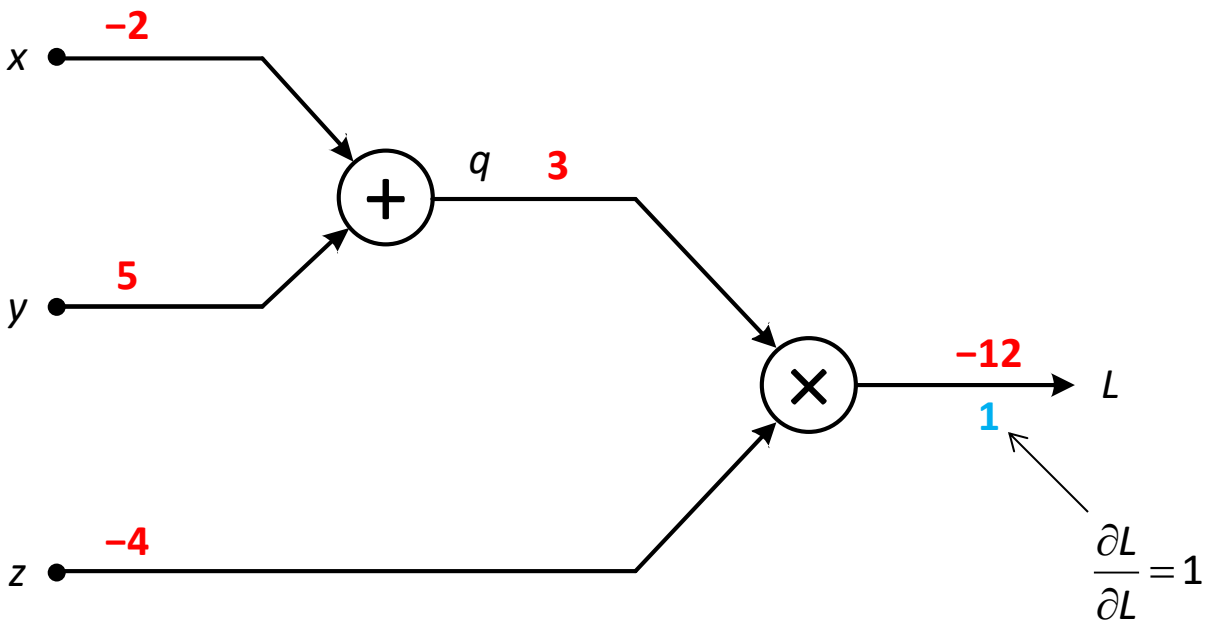
Uticaj promene ulaza na izlaz (primer 1)



$$L = qz \qquad \frac{\partial L}{\partial q} = z \qquad \frac{\partial L}{\partial z} = q$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

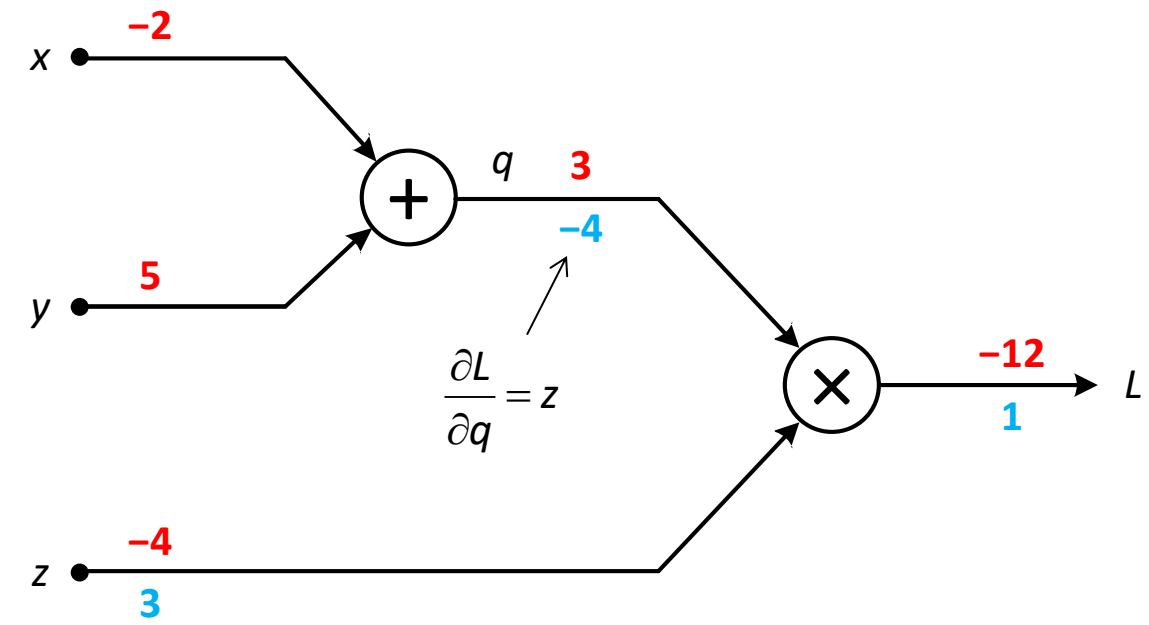
Uticaj promene ulaza na izlaz (primer 1)



$$L = qz \qquad \frac{\partial L}{\partial q} = z \qquad \frac{\partial L}{\partial z} = q$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

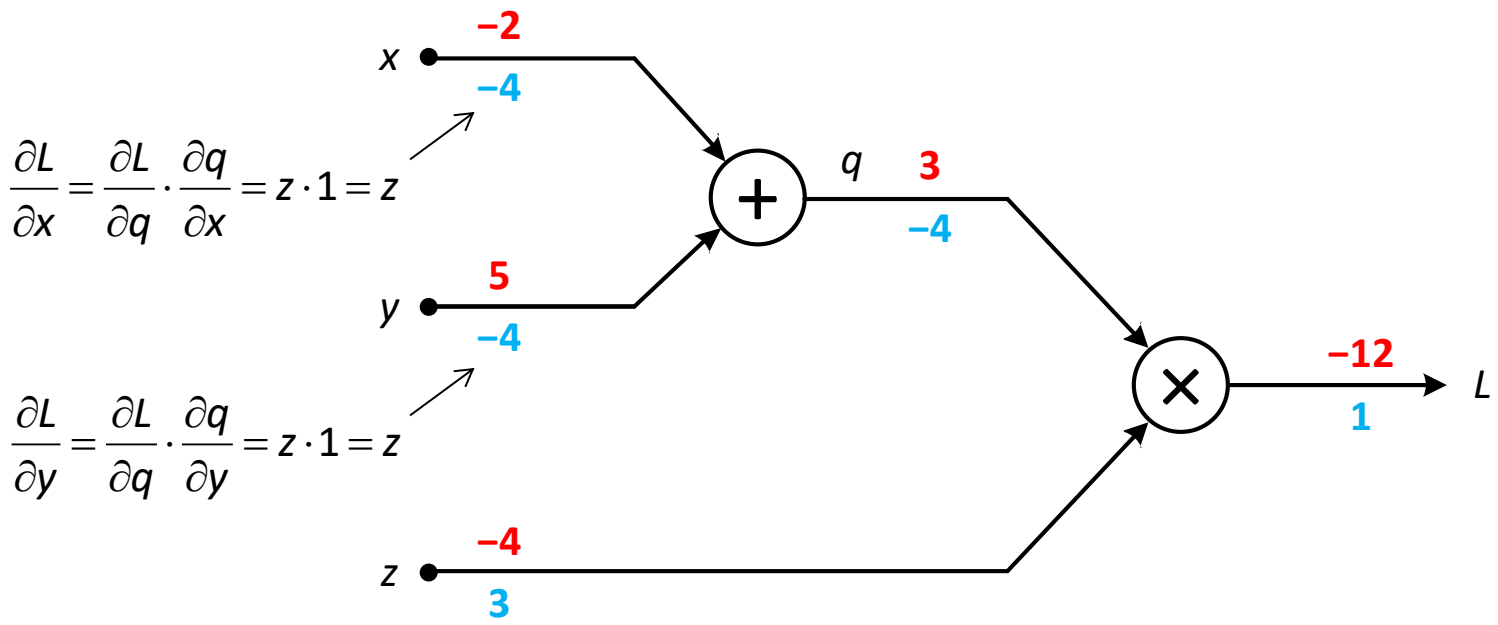
Uticaj promene ulaza na izlaz (primer 1)



$$L = qz \qquad \frac{\partial L}{\partial q} = z \qquad \frac{\partial L}{\partial z} = q$$

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

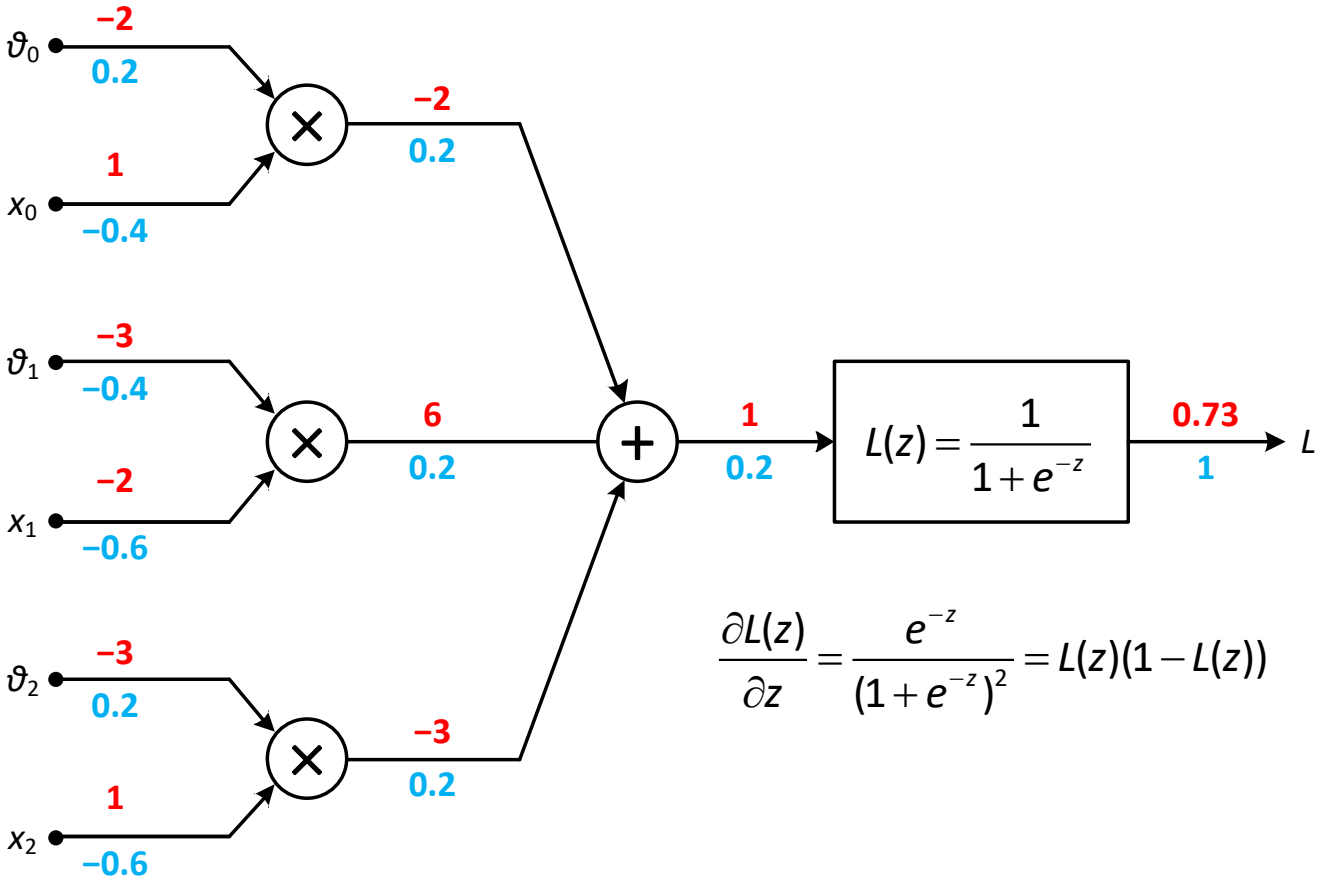
Uticaj promene ulaza na izlaz (primer 1)



$$L = qz \qquad \frac{\partial L}{\partial q} = z \qquad \frac{\partial L}{\partial z} = q$$

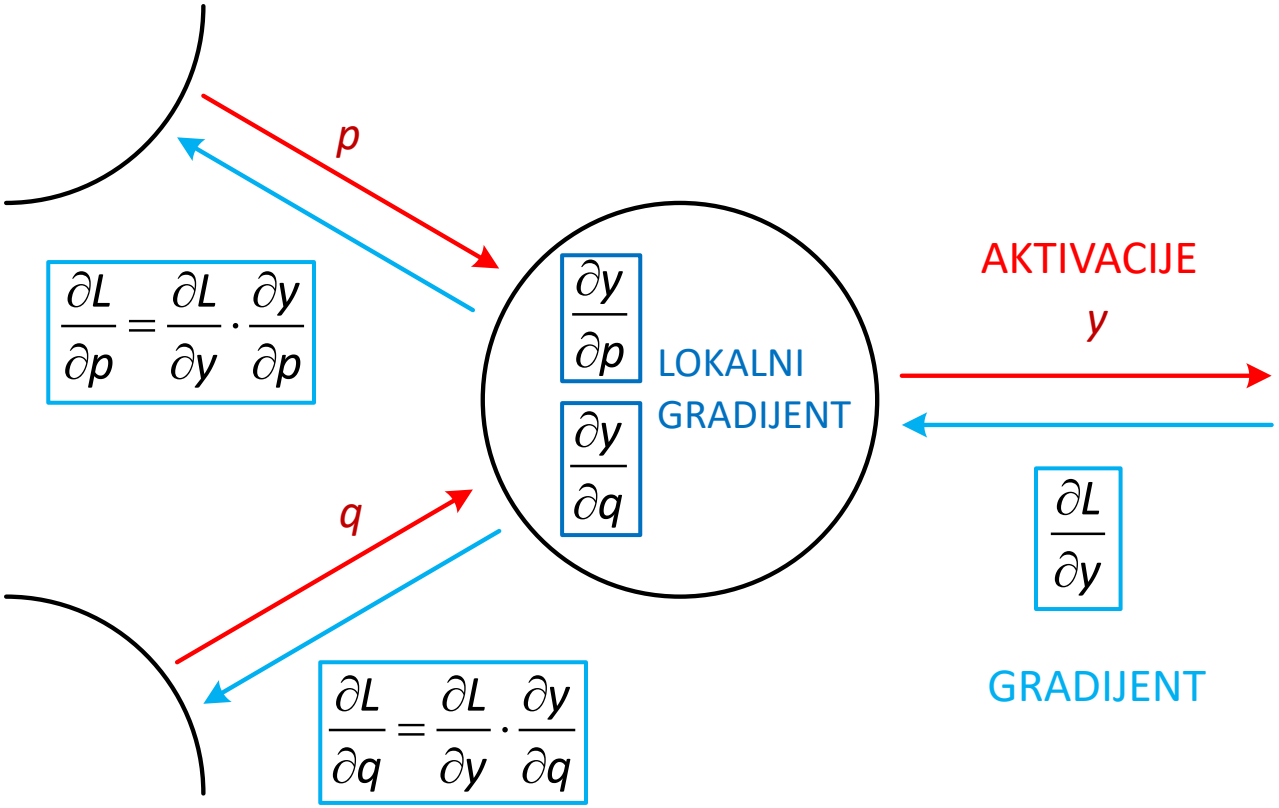
$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

Uticaj promene ulaza na izlaz (primer 2)



Uticaj promene ulaza na izlaz

- Svaki neuron može odrediti uticaj svojih ulaza na određenu funkciju *L* znajući:
  - uticaj svog izlaza na tu funkciju (što može dobiti od neurona iz narednog sloja)
  - vezu između svojih ulaza i izlaza (što svakako zna)



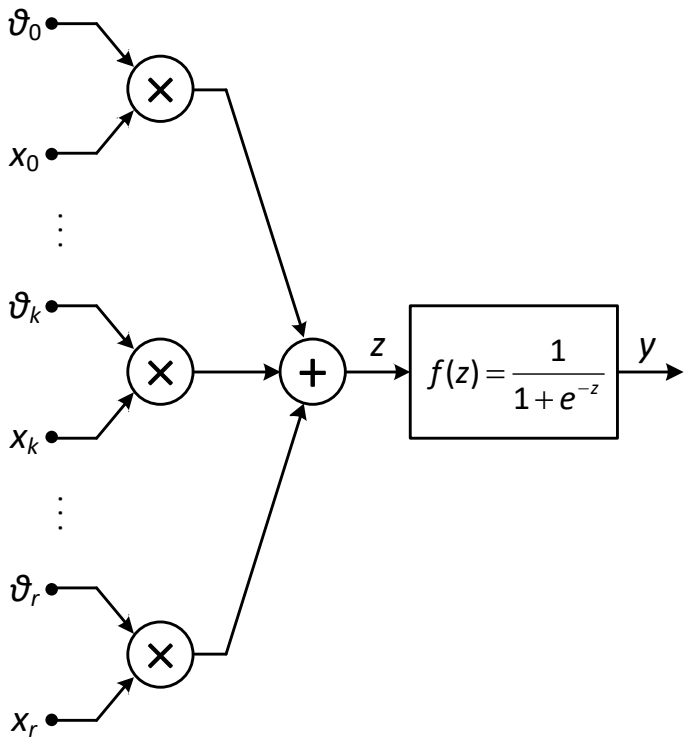
Uticaj težinskih faktora na funkciju cene

- Za slučaj jednog neurona u izlaznom sloju i jedan uzorak:
  - Neka je dobijeni izlaz y, a željeni izlaz $\hat{y}$
  - Funkcija cene je jednaka $J = \frac{1}{2}(y - \hat{y})^2$
  - Zavisnost funkcije cene od određenog težinskog faktora izlaznog neurona može se naći na sledeći način:

$$\frac{\partial J}{\partial \vartheta_k} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial \vartheta_k}$$

- $\frac{\partial J}{\partial y} = y - \hat{y}$
- $\frac{\partial y}{\partial z} = \frac{\partial f(z)}{\partial z} = f(z)(1 - f(z)) = y(1 - y)$
- $\frac{\partial z}{\partial \vartheta_k} = \frac{\partial}{\partial \vartheta_k} \sum_{i=0}^{r} \vartheta_i x_i = x_k$

$$\frac{\partial J}{\partial \vartheta_k} = (y - \hat{y}) y (1 - y) x_k = \delta \cdot x_k$$



Koeficijent δ može se interpretirati kao osetljivost funkcije cene J na aktivaciju određenog neurona

Uticaj težinskih faktora na funkciju cene

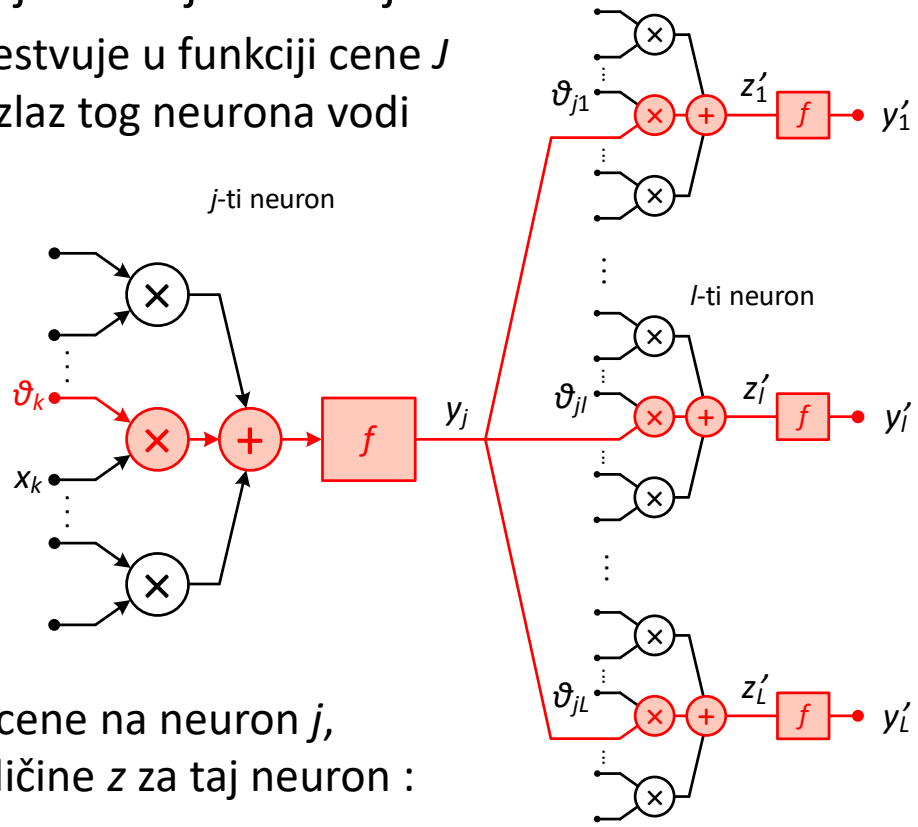
- Za slučaj neurona u skrivenom sloju stvar je složenija
  - Težinski faktor nekog neurona učestvuje u funkciji cene J preko svakog neurona na koji se izlaz tog neurona vodi

$$\frac{\partial J}{\partial y_j} = \sum_{l=1}^{L} \frac{\partial J}{\partial z_l'} \cdot \frac{\partial z_l'}{\partial y_j}$$

$$\frac{\partial J}{\partial \vartheta_k} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial \vartheta_k} = \frac{\partial J}{\partial y_j} \cdot y_j(1-y_j)x_k$$

što se takođe može zapisati kao:

$$\frac{\partial J}{\partial \vartheta_k} = \delta_j \cdot x_k$$



pri čemu je δ_j osetljivost funkcije cene na neuron j,
tj. njena promena pri promeni veličine z za taj neuron :

$$\delta_j = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial z_j} = \begin{cases} (y_j - \hat{y}_j) \cdot y_j(1-y_j), & \text{ako je neuron } j \text{ u izlaznom sloju} \\ \left[\sum_{l=1}^{L} \delta_l \vartheta_{jl}\right] \cdot y_j(1-y_j), & \text{ako je neuron } j \text{ u skrivenom sloju} \end{cases}$$

Algoritam propagacije unazad

- **Inicijalizacija:** Inicijalizovati sve težine malim slučajnim vrednostima
- **Propagacija unapred:** Za svaki uzorak $\mathbf{x}(i)$ naći sve:

$$\left.\begin{aligned} z_j^{(r)}(i) &= \boldsymbol{\theta}_j^{(r)\mathrm{T}}\mathbf{y}^{(r-1)} \\ y_j^{(r)}(i) &= f(z_j^{(r)}(i)) \end{aligned}\right\} r=1,2,\ldots,L;\ j=1,2,\ldots,k_r$$

  i izračunati funkciju cene za trenutnu estimaciju težina*:

$$J=\frac{1}{2}\sum_{i=1}^{N}\sum_{m=1}^{k_L}e_m^2(i)=\frac{1}{2}\sum_{i=1}^{N}\sum_{m=1}^{k_L}(f(z_m^{(L)}(i))-\hat{y}_m(i))^2$$

- **Propagacija unazad:** Za svaki uzorak $\mathbf{x}(i)$ i svaki neuron u poslednjem sloju izračunati:

$$\delta_j^{(L)}(i)=e_j(i)\cdot f'(z_j^{(L)}(i)),\quad j=1,2,\ldots,k_L,$$

  a zatim, po sličnom principu, za svaki neuron u preostalim slojevima izračunati:

$$\delta_j^{(r-1)}(i)=e_j^{(r-1)}(i)\cdot f'(z_j^{(r-1)}(i)),\quad r=L,L-1,\ldots,2;\ j=1,2,\ldots,k_r,$$

  pri čemu je $e_j^{(r-1)}(i)=\sum_{l=1}^{k_r}\delta_l^{(r)}(i)\cdot\vartheta_{jl}^{(r)}$

- **Korekcija težina:** Korigovati sinaptičke težine na osnovu izraza:

$$\boldsymbol{\theta}_j^{(r)}(\text{novo})=\boldsymbol{\theta}_j^{(r)}(\text{staro})-\alpha\sum_{i=1}^{N}\delta_j^{r}(i)\mathbf{y}^{(r-1)}(i),\quad r=1,2,\ldots,L;\ j=1,2,\ldots,k_L$$

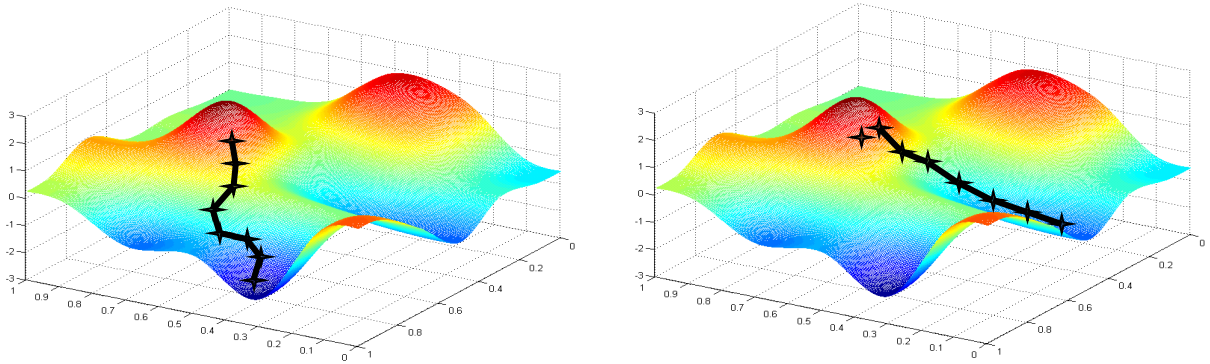
  i ponavljati poslednja tri koraka do ispunjenja izlaznog kriterijuma

---

* za slučaj funkcije cene definisane na osnovu ukupnog kvadratnog odstupanja

Algoritam propagacije unazad

- Za razliku od linearnih modela, kod neuronskih mreža funkcija cene *J* zbog nelinearnosti može imati vrlo složen nekonveksan oblik sa velikim brojem lokalnih minimuma
  - Algoritam propagacije unazad radi s rizikom upadanja u lokalni minimum, odnosno, nije garantovana konvergencija ka globalnom minimumu
  - Ako je lokalni minimum dubok, to može biti prihvatljivo, ali u praksi nije loše reinicijalizovati algoritam iz različitog skupa početnih uslova
  - Inicijalizacija se vrši slučajnim vrednostima a ne nulama jer bi se u slučaju nulte inicijalizacije svi neuroni istog sloja nadalje vreme ponašali isto
  - Nepovoljan izbor inicijalne tačke može ugroziti brzinu konvergencije ili čak izazvati numeričke probleme koje algoritam neće uspeti da prevaziđe
- Izlazni kriterijum može biti da funkcija cene *J* padne ispod određenog praga ili da se njena vrednost ne menja značajno u određenom broju uzastopnih iteracija



Još neke varijante algoritma propagacije unazad

- Jedan ciklus obuke u okviru kog mreža „vidi“ sve uzorke iz skupa za obuku naziva se *epoha*
- Prethodno opisani algoritam rafinira težine na osnovu analize svih uzoraka *zajedno*, odnosno, radi u *grupnom* režinu rada (eng. *batch gradient descent*)
  - Moguć je i scenario u kom se težine ažuriraju nakon analize svakog pojedinačnog uzorka, što se naziva *uzoračkim* režimom rada (eng. *stochastic gradient descent*):

    $$\boldsymbol{\theta}_j^{(r)}(\text{novo}) = \boldsymbol{\theta}_j^{(r)}(\text{staro}) - \alpha\delta_j^r(i)\mathbf{y}^{(r-1)}(i), \quad i = 1,2,...,N; r = 1,2,...,L; j = 1,2,...,k_L$$

    čime se povećava nivo slučajnosti i može se izbeći zaglavljivanje u lokalnom minimumu (što je prednost), ali je u grupnom režimu estimacija gradijenta tačnija
    - U uzoračkom režimu dobra praksa je varirati redosled predstavljanja uzoraka mreži u okviru svakog ciklusa (epohe) obuke, a uzorcima se često dodaje i (vrlo mali) beli šum
  - Moguć je i kompromisni scenario, u kom se težine ažuriraju nakon analize slučajno odabrane podgrupe uzoraka (eng. *mini-batch gradient descent*)

Brzina konvergencije

- Brzina učenja α predstavlja kompromis između brzine konvergencije i rizika promašivanja minimuma funkcije cene J
- Uvođenje *momenta* u korekciji sinaptičkih težina ubrzava konvergenciju ublažavanjem oscilatornog ponašanja gradijenta funkcije cene između sukcesivnih koraka:

$$\boldsymbol{\theta}_j^{(r)}(\text{novo}) = \boldsymbol{\theta}_j^{(r)}(\text{staro}) + \Delta\boldsymbol{\theta}_j^{(r)}(\text{novo})$$

$$\Delta\boldsymbol{\theta}_j^{(r)}(\text{novo}) = -\alpha \sum_{i=1}^{N} \delta_j^r(i)\mathbf{y}^{(r-1)}(i) - \beta\Delta\boldsymbol{\theta}_j^{(r)}(\text{staro})$$

gde je β faktor momenta, koji se u praksi kreće između 0.1 i 0.8

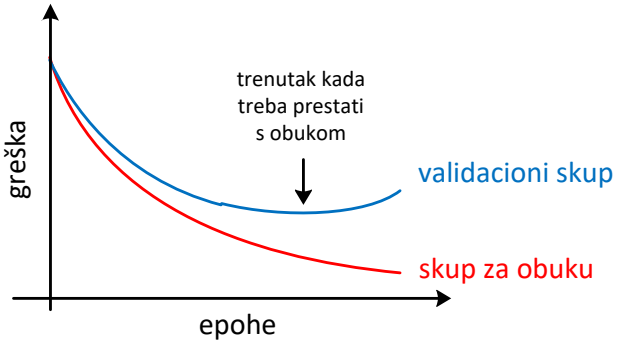
- Uvođenjem zavisnosti od vrednosti korekcije u prethodnoj iteraciji efektivno se uvećava brzina učenja α kada je gradijent približno konstantan, a smanjuje se kada on osciluje
- U momentu približavanja stvarnom minimumu momenat može izazvati oscilacije oko minimuma, čime malo usporava konvergenciju
- Bolja kontrola konvergencije može se postići sa eksplicitno promenljivom (adaptivnom) vrednošću brzine učenja α

Izbor složenosti mreže

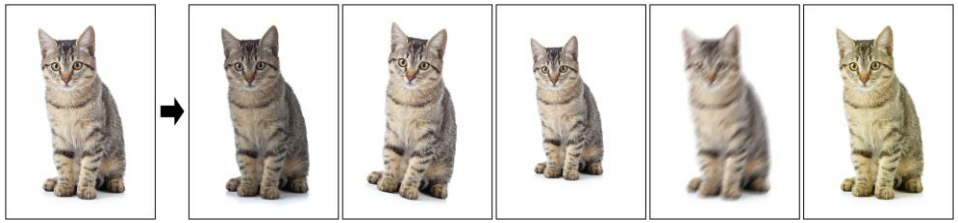
- Ako je mreža suviše složena:
  - količina izračunavanja može biti značajno veća
  - mreža može biti sklona tome da se suviše prilagodi uzorcima za obuku, odnosno, da nema dovoljnu sposobnost generalizacije (*overfitting*)
- Za fiksnu veličinu skupa za obuku *N*, broj slobodnih parametara (sinaptičkih težina):
  - mora biti dovoljno velik:
    - da nauči šta čini sličnim uzorke jedne klase
    - da nauči šta razlikuje uzorke iz različitih klasa
  - ali i dovoljno mali u odnosu na *N*:
    - da ne može da nauči razlike između podataka iste klase
- Mreža obično ima jedan ili više skrivenih slojeva (koji su najčešće iste veličine)
  - u zavisnosti od složenosti mreže mogu se aproksimirati različito složene funkcije
  - broj skrivenih slojeva određuje izražajne mogućnosti modela
- Izbor potrebnog broja skrivenih slojeva i neurona u njima u opštem slučaju je nerešen problem, a u praksi se za rešavanje može koristiti unakrsna validacija

Regularizacija

- Primena eksplicitnih metoda za sprečavanje natprilagođenja pri obuci
  - Rano zaustavljanje obuke (eng. *early stopping*)
    - Natprilagođenje može biti rezultat preterane obuke (eng. *overtraining*), pa treba izdvojiti poseban validacioni skup podataka i prekinuti obuku kad greška na validacionom skupu krene da raste

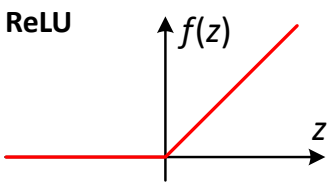


  - Ograničavanje L1 ili L2 norme parametara
    - Norma parametara uključuje se u funkciju cene (dodaje joj se sabirak $\lambda\sum_i|\vartheta_i|$ ili $\lambda\sum_i\vartheta_i^2$) sa ciljem da se u toku obuke istovremeno minimizuje i greška i veličina parametara
  - *Dropout*
    - U svakoj iteraciji obuke svaki neuron u mreži se s određenom verovatnoćom isključuje i ne učestvuje u toj iteraciji, čime se mreža stimuliše da distribuira naučene zavisnosti na više neurona
    - Ovo se može posmatrati kao ansambalsko učenje jer se u svakoj iteraciji obučava različita mreža
  - Augmentacija podataka
    - Uvećavanje raspoloživog skupa podataka dodavanjem veštački modifikovanih verzija postojećih podataka



Još neki praktični saveti

- Mogu se koristiti i druge aktivacione funkcije
  - Čest izbor aktivacione funkcije unutar mreže je tzv. *ispravljačka linearna jedinica* (eng. *rectified linear unit* – ReLU): $f(z) = \max(0, z)$
    - Postoje razne modifikacije u cilju prevazilaženja problema $f'(z) = 0$ za $z < 0$



  - Upotreba sigmoida uglavnom je ograničena na izlaz mreže pri binarnoj klasifikaciji, a za klasifikaciju u više klasa na izlazu mreže se često koristi *softmax* aktivacija: $f(z_i) = e^{z_i} / \sum_j e^{z_j}$
    - Izlazi su u opsegu (0, 1) a zbir im je 1 pa se mogu interpretirati kao verovatnoće pripadnosti klasi ω_i
- Pri obuci ciljne vrednosti moraju biti u kodomenu aktivacione funkcije
  - U suprotnom, neuron neće moći da proizvede te vrednosti
  - Za ciljne vrednosti nije dobro odabrati ni asimptotske vrednosti aktivacione funkcije
    - Algoritam bi težio da dovede neurone u zasićenje, čime bi se usporilo učenje
- Treba izvršiti normalizaciju i dekorelaciju ulaznih promenljivih
  - Bez normalizacije ulazna obeležja bi u opštem slučaju imala različite redove veličina
    - Mreža bi tokom obuke gubila vreme u pokušajima da prevaziđe te razlike
    - Obuka bi se mogla zaglaviti zbog zasićenja pojedinih neurona, a porasla bi i verovatnoća da se obuka završi u nepovoljnom lokalnom minimumu
    - Neke od tehnika regularizacije bile bi mnogo manje efikasne
  - Često se koristi standardizacija (normalizacija na srednju vrednost 0 i varijansu 1) u kombinaciji sa razlaganjem na osnovne komponente (eng. *principal component analysis – PCA*)