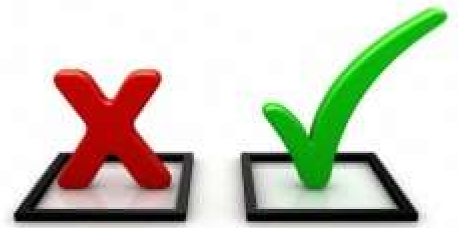
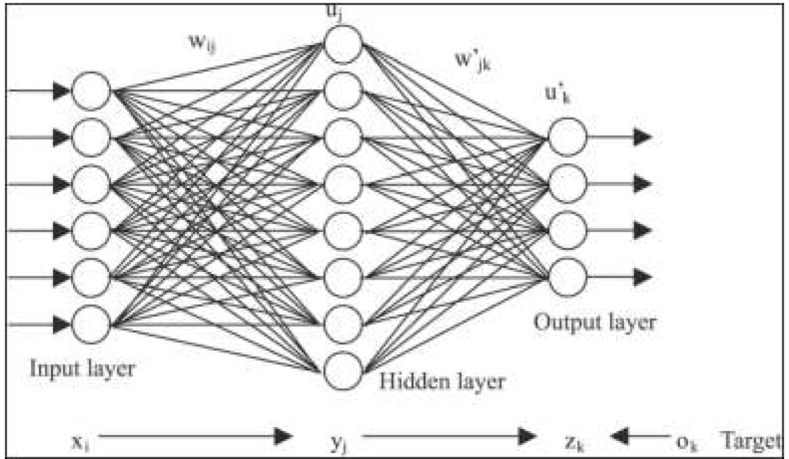
# Umetna inteligenca

Ocenjevanje učenja

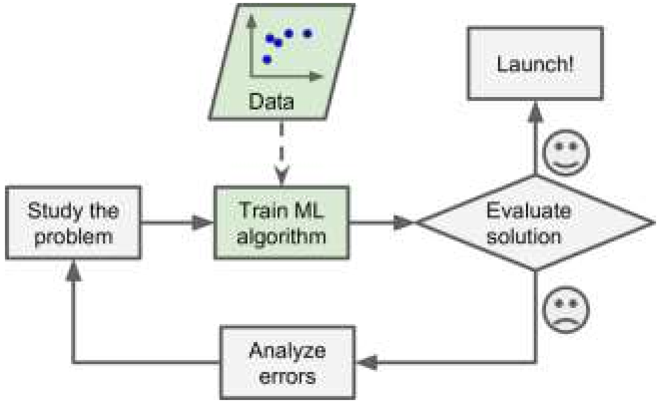


Učenje ansamblov

Umetne nevronske mreže



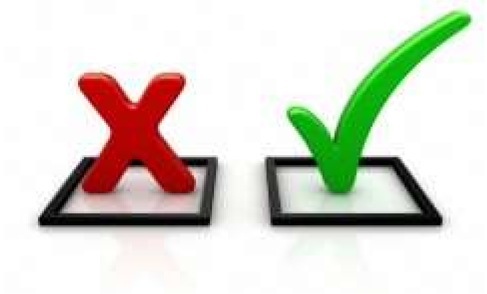
# Ocenjevanje učenja




- kriteriji za ocenjevanje hipotez:
  - točnost (angl. *accuracy*)
  - kompleksnost (angl. *complexity*)
  - razumljivost (angl. *comprehensibility*) – subjektivni kriterij

- ocenjevanje točnosti:
  - na **učnih** podatkih (angl. *training set, learning set)*
  - na **testnih** podatkih  (angl. *testing set, test set*)
    - izločimo del učnih podatkov, s katerimi simuliramo ne-videne podatke
    - želimo si, da je testna množica reprezentativna za nove podatke
    - uporabimo lahko **intervale zaupanja** v oceno uspešnosti na testni množici, ki upoštevajo število testnih primerov
  - na **novih** (ne-videnih) podatkih  (angl. *new data, unseen data*)
    - na njih bo naučeni sistem dejansko deloval

# Ocenjevanje učenja



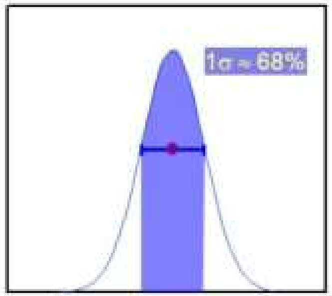
Klasifikacija:

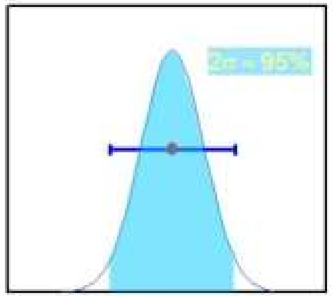
- klasifikacijska točnost,
- tabela napačnih klasifikacij,
- cena napačne klasifikacije,
- Brierjeva mera,
- informacijska vsebina,
- senzitivnost in specifičnost, krivulja ROC,

Regresija:

- Srednja kvadratna napaka
- Relativna srednja kvadratna napaka
- Srednja absolutna napaka
- Relativna srednja absolutna napaka

# Ocenjevanje učenja



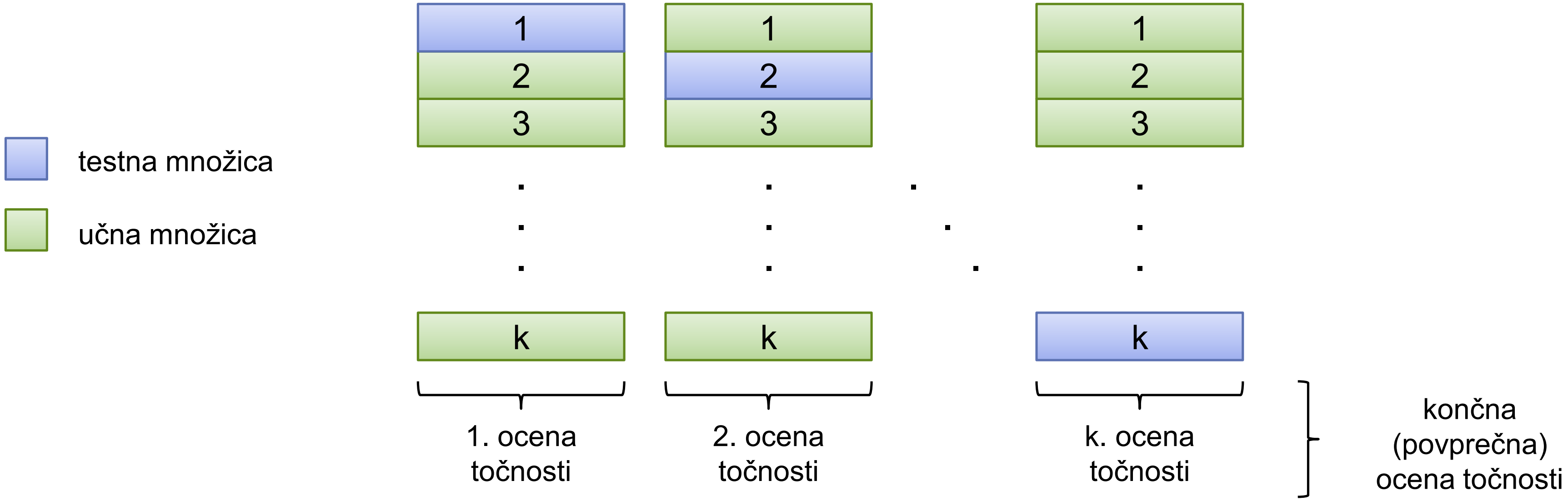
- nasprotujoča si cilja:
  - potrebujemo čim več podatkov za **uspešno učenje**
  - potrebujemo čim več podatkov za **zanesljivo ocenjevanje točnosti** (večje število testnih primerov nam daje ožji interval zaupanja v oceno točnosti)

- rešitev:
  - kadar je učnih podatkov dovolj, lahko izločimo **testno množico** (angl. *holdout test set*)
  - alternativa: **večkratne delitve** na učno in testno množico

- različni načini **vzorčenja testnih primerov**:
  - naključno, nenaključno (npr. prečno preverjanje)
  - poljubno ali stratificirano (zagotovimo enako porazdelitev razredov kot v učni množici)

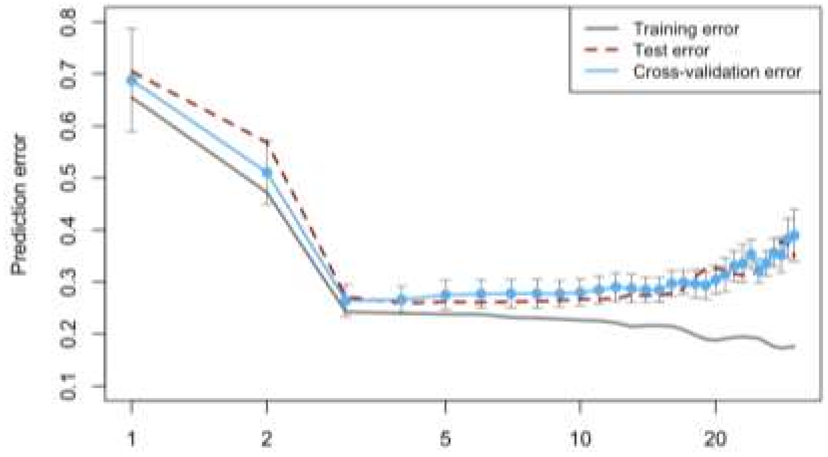
# Prečno preveranje

- poseben primer večkratnega učenja in testiranja
- k-kratno prečno preverjanje (angl. *k-fold cross-validation*):
  - celo učno množico razbij na *k* disjunktnih podmnožic
  - za vsako od *k* podmnožic:
    - uporabi množico kot testno množico
    - uporabi preostalih *k-1* množic kot učno množico
  - povpreči dobljenih *k* ocen točnosti v končno oceno



# Prečno preveranje

- v praksi najpogosteje: k=10 (10-kratno prečno preverjanje)
- vplive izbranega razbitja podatkov na podmnožice lahko zmanjšamo tako, da tudi prečno preverjanje večkrat (npr. 10x) ponovimo (torej 10×10=100 izvajanj učnega algoritma) in rezultate povprečimo
- poseben primer prečnega preverjanja je metoda **izloči enega** (angl. leave-one-out, LOO)
  - *k* je enak številu primerov (vsaka testna množica ima samo en primer)
  - najbolj stabilna ocena glede učinkov razbitja na podmnožice
  - časovno zelo zamudno, primerno za manjše množice
- iz meritev na vseh podmnožicah je možno izračunati tudi varianco/ intervale zaupanja



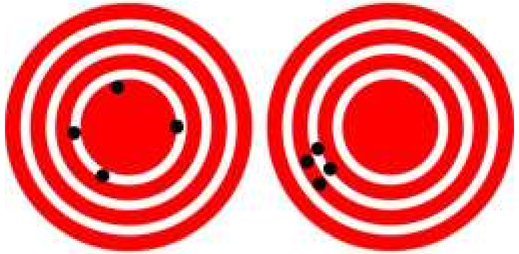
# Primerjanje uspešnosti različnih učnih algoritmov

- Učenje: *učna množica* (learning/training set)
- **Nastavljanje parametrov: *nastavitvena množica* (validation/tuning set)**
- Testiranje: *testna množica* (testing set)
- **Pogosta napaka: testna množica se uporabi kot nastavitvena...**
  - **Testni primeri ne smejo nikoli biti uporabljeni v učnem procesu!**
  - **Nastavljanje parametrov je del učnega procesa!!!**
  - **Izbira najboljšega modela je tudi del učnega procesa!**
- Parametrični testi značilnosti odstopanj:
  - ocenimo stopnjo zaupanja v ocenjene razlike uspešnosti

# Primerjanje uspešnosti različnih učnih algoritmov

- Dva algoritma na eni domeni
  - Prečno preverjanje: enosmerni t-test
  - Izloči enega ali neodvisna testna množica: enosmerni z-test
  - Bonferronijeva korekcija ($N$ primerjav): $\alpha_B \approx \alpha/N$
- Two algorithms on several domains: nonparametric Wilcoxon signed rank test
- Več algoritmov na več domenah: Friedmanov test
  - En algoritem proti ostalim: test Bonferroni-Dunn
  - Vsak z vsakim: Nemenyijev test

# Dva algoritma na eni domeni



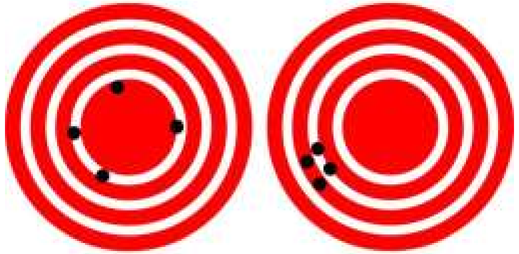
*K-kratno prečno preverjanje:*
Enosmerni (one-tailed) $t$-test.
($t$-test: število primerjanj je majhno, $K < 30$)
(enosmerni test: ali je eden od algoritmov boljši od drugega)

$$raz_i = \hat{U}_1 - \hat{U}_2 \qquad\qquad i = 1, \ldots, K$$

Za $raz_i$ predpostavimo, da je porazdeljena normalno.
Hipoteza: oba algoritma dosegata enako uspešnost ($\overline{raz} = 0$).

$$\overline{raz} = \frac{1}{K}\sum_{i=1}^{K} raz_i \ ; \quad s = \sqrt{\frac{1}{K-1}\sum_{i=1}^{K}(raz_i - \overline{raz})^2} \ ; \quad \overline{raz} \geq 0$$

# Dva algoritma na eni domeni



Statistika $t$ je porazdeljena po Studentovem zakonu:

$$t = \frac{\overline{raz}}{s}\sqrt{K}$$

Pri stopnji zaupanja $1 - \alpha$ hipotezo zavržemo, če $t > t(\alpha, K - 1)$

$t(\alpha, K - 1)$ določa mejo intervala vrednosti spremenljivke, porazdeljene po Studentovem zakonu s $K - 1$ prostostnih stopenj.

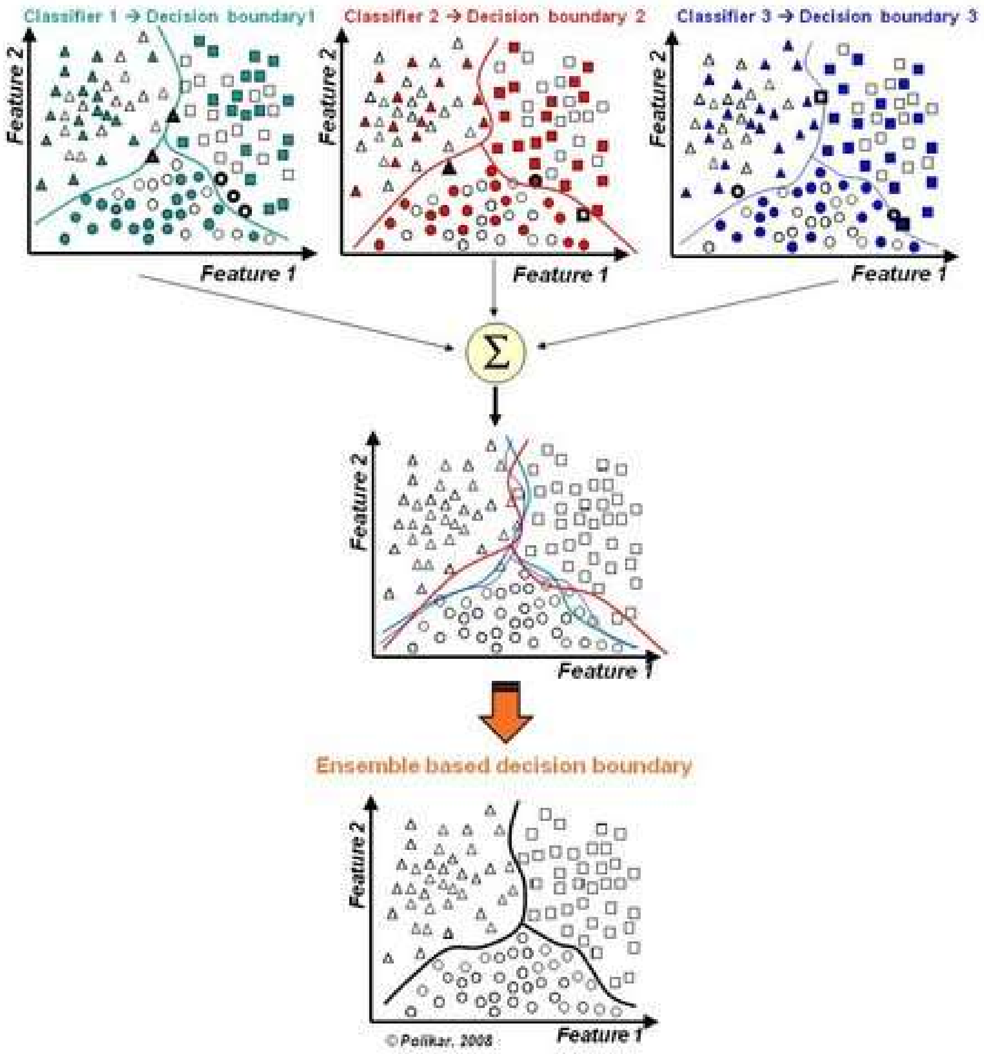
Tipične vrednosti za $\alpha$ so 0.05, 0.01 in 0.001.

# Učenje ansamblov

- Želimo izboljšati točnost, zmanjšati varianco napovedi
- Uporabimo princip večkratne razlage
- Lahko gradimo več modelov:
  - Z različnimi učnimi algoritmi (v praksi redko)
  - *En* učni algoritem poganjamo večkrat z
    - različnimi nastavitvami parametrov
    - nad različno pripravljenimi vhodnimi podatki
- Kombiniranje napovedi različnih modelov:
  - Glasovanje (angl. voting)
  - Uteženo glasovanje, npr. z zanesljivostjo napovedi
  - Naučeno kombiniranje z meta-učenjem (angl. stacking), npr. z NB ali LR
  - Lokalno uteženo glasovanje

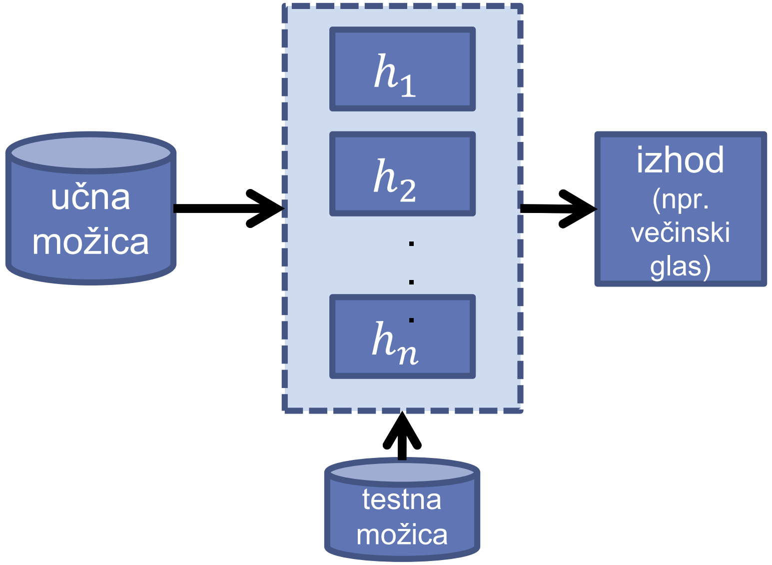
# Učenje z ansambli

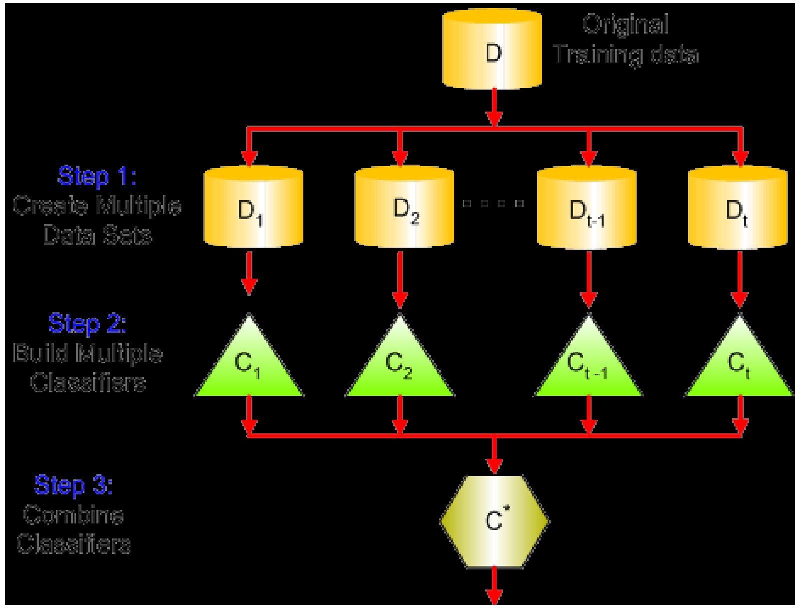
- **druga prednost: preprost način za povečanje izraznosti prostora hipotez**
- s kombiniranjem modelov dobimo kombinacijo posameznih hipotez, ki lahko presegajo omejitve glede oblik hipotez, ki so rezultat posamenih algoritmov za učenje



# Učenje ansamblov

- **ansambel** – zbirka hipotez, iz katerih oblikujemo končno hipotezo s kombiniranjem napovedi (npr. z večinskim glasovanjem)
- kombiniramo lahko hipoteze, pridobljene na **različne načine**:
  - z različnimi učnimi algoritmi (odločitvena drevesa, kNN, naivni Bayes)
  - hipoteze, pridobljene na različnih učnih množicah (ali vzorčenjih) iste množice



# Učenje ansamblov

- Bagging


- Boosting

- Naključni gozdovi

# Bagging

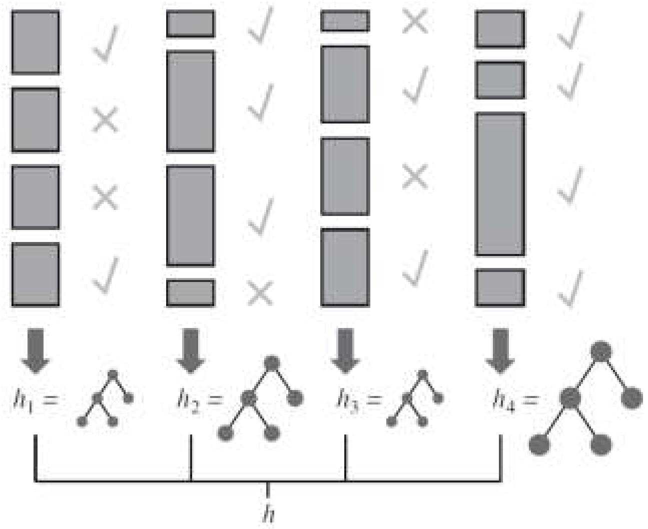

- "bootstrap agreggating"(Breiman, 1996): zaporedje modelov iz različnih učnih množic
- Za eno učno množico z *n* učnimi primeri:
  - *n* krat naključno izberemo primer z vračanjem
  - nekaterih primerov množica sploh ne vsebuje (cca 36.8%)
  - Zgradimo model na taki učni množici
- Vsi modeli glasujejo za napoved.
- Dober pri nestabilnih učnih algoritmih z visoko varianco (odločitvena in regresijska drevesa).
- Z večanjem števila modelov ne pride do prevelikega prileganja (v praksi 100 ali več modelov)

# Boosting

- metoda boosting vsakemu učnemu primeru pripiše nenegativno utež $w_j \geq 0$, ki vpliva na pomen primera pri učenju hipoteze
  - utež si lahko predstavljamo tudi kot faktor, ki poveča število kopij istega učnega primera v učni množici

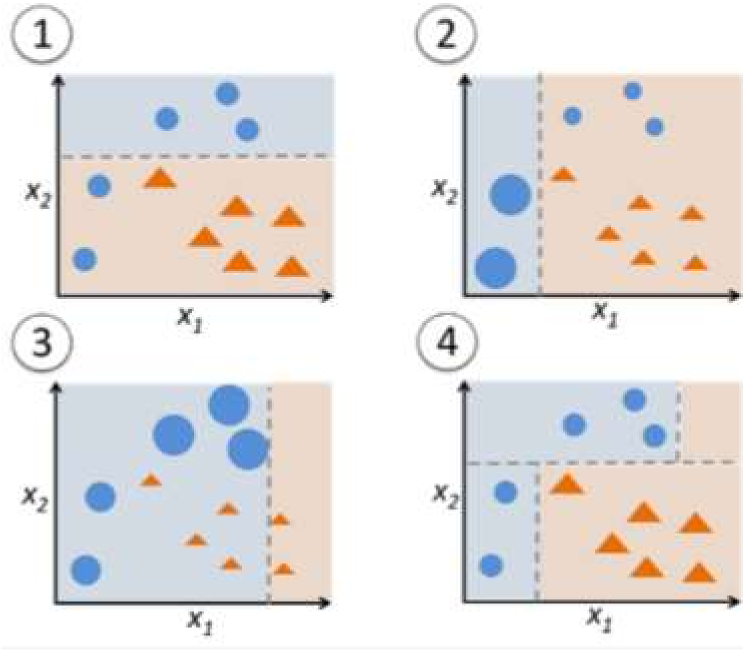
- algoritem:
  - na začetku za vse primere določi $w_j = 1$ in generiraj $h_1$
  - za vsak nadaljnji $h_i$ **povečaj utež nepravilno klasificiranim** primerom in **zmanjšaj utež pravilno klasificiranim** primerom s strani hipoteze $h_{i-1}$
  - nadaljuj, dokler ni generirano ciljno število hipotez ($k$) ALI je zadnja hipoteza „preslaba“ ALI „predobra“
  - končna napoved je utežena vsota vseh posameznih napovedi hipotez, ki jih utežimo z uspešnostmi posameznih hipotez

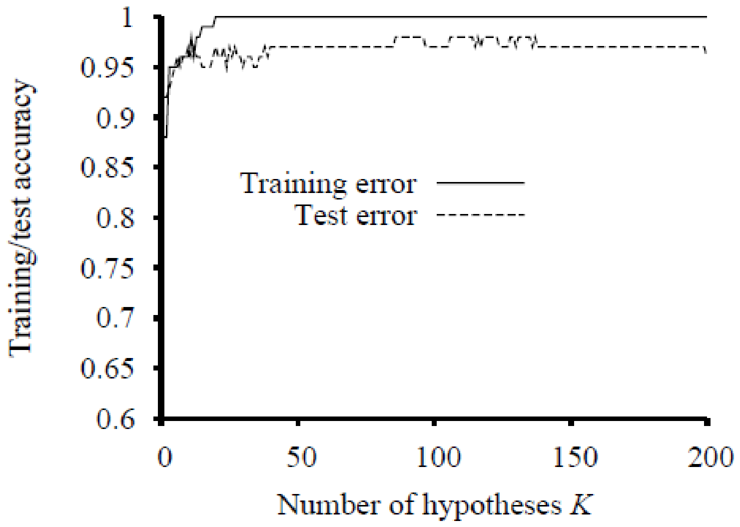
- primer implementacije: algoritem ADABOOST (*adaptive boosting,* Freund in Schapire, 1997)



# Boosting

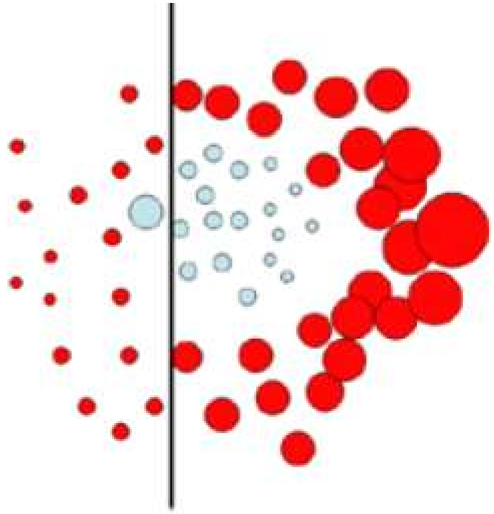
- levo: postopek uteževanja nepravilno klasificiranih primerov:
- desno: uspešno naraščanje klasifikacijske točnosti na testni množici navkljub popolni (100%) klasifikacijski točnosti na učni množici



# Boosting

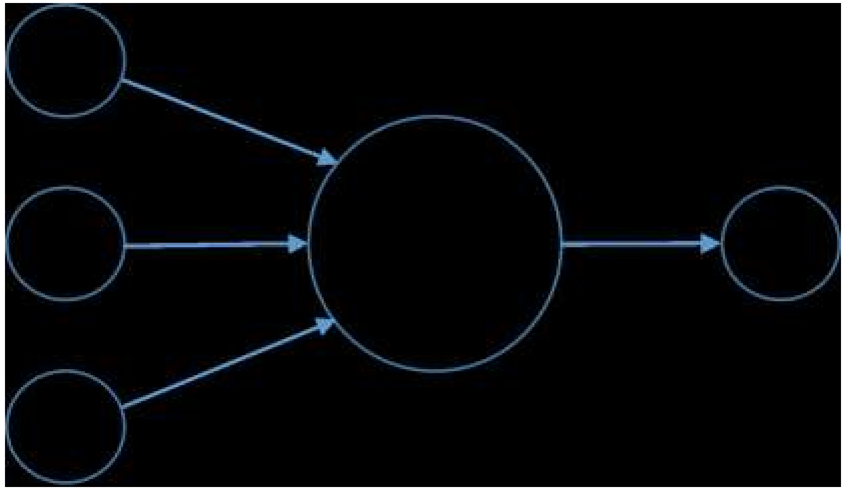
- *Spreminjanje uteži učnim primerom:*
  *e* = napaka hipoteze na učnem primeru,
  utež primera se pomnoži z *e*/(1 - *e*). Na koncu normalizacija.
- *Utež napovedi za hipotezo z napako f na uteženi učni množici:*
  -*log*(*f*/(1 - *f*))
- Boosting pogosto dosega boljšo točnost od bagginga in ga lahko uporabimo tudi na stabilnih učnih algoritmih z majhno varianco.
- Lahko pride do prevelikega prileganja učni množici (overfitting).

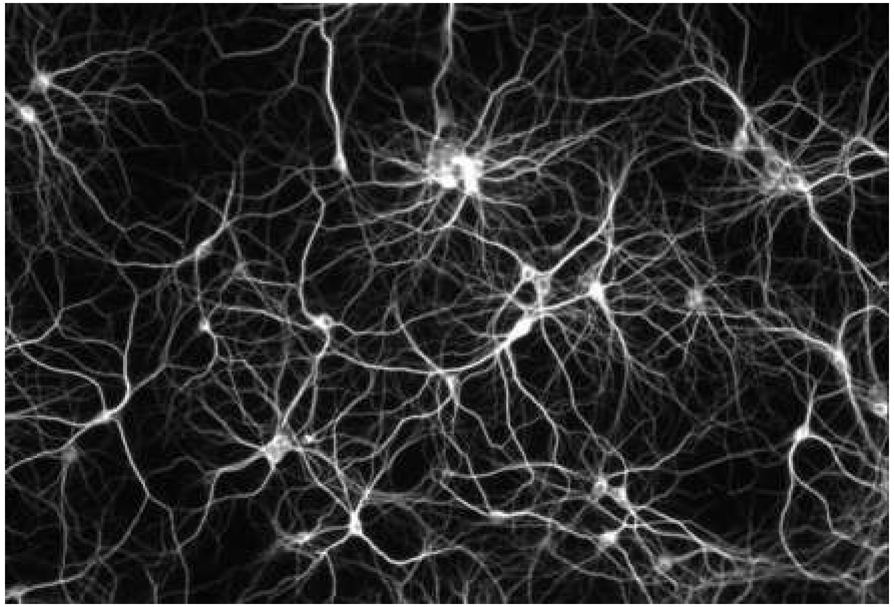


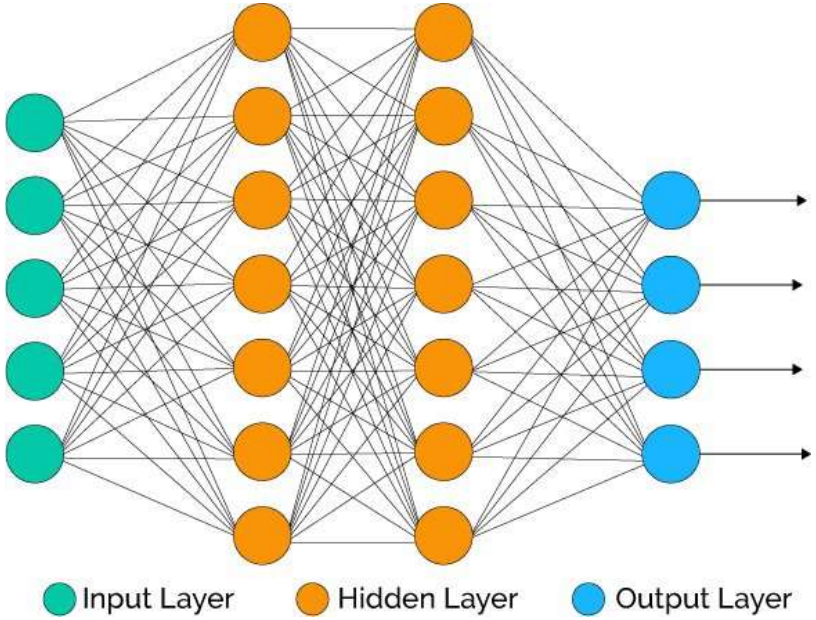
# Naključni gozdovi

- Gradimo veliko število dreves (100 ali več), podobno kot pri baggingu – iz razlčnih učnih množic.
- Pri izbiri najboljšega atributa v vsakem vozlišču naključno izbere relativno majhno število atributov, ki vstopajo v izbor za najboljši atribut.
- Klasifikacija: glasovanje vseh dreves.
- Regresija: povprečje napovedi vseh dreves.
- Robustna metoda, saj zmanjša varianco drevesnih algoritmov.
- Dosega vrhunske rezultate na mnogih problemih.
- Slabost: razlaga odločitev je otežena…

# Umetne nevronske mreže

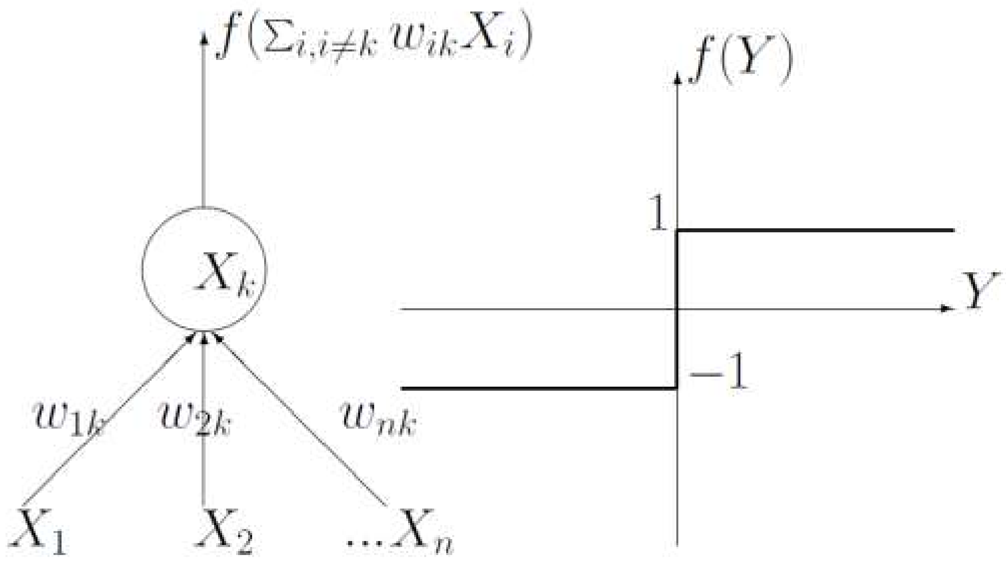






# Umetne nevronske mreže

- En umetni nevron je (zelo huda) abstrakcija naravnega nevrona: preprost element, ki zna izračunati uteženo vsoto in jo poslati skozi pragovno/normalizacijsko funkcijo
- Vrednost izhoda je določena z vhodom in z vrednostmi uteži
- Spomin (znanje) nevronske mreže predstavljajo uteži na povezavah (sinapsah) med nevroni
- Funkcija, ki jo nevron izračunava, je določena z utežmi
- učna naloga je torej
  - Izbrati topologijo mreže
  - nastaviti vrednosti uteži



# Preprost primer

Naloga: razpoznavati naslednja dva vzorca:

$$X_1 = (1,1,1)^T$$

in

$$X_2 = (1,-1,-1)^T$$

Sestavimo matriko:

$$M = X_1X_1^T + X_2X_2^T = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 2 \\ 0 & 2 & 2 \end{bmatrix}$$

Definirajmo še odločitveno funkcijo:

$$f(X) = \begin{cases} 1, & X > 0 \\ 0, & X = 0 \\ -1, & X < 0 \end{cases}$$

Sedaj velja:

$$f(MX_1) = f((2,4,4)^T) = (1,1,1)^T = X_1$$

$$f(MX_2) = f((2,-4,-4)^T) = (1,-1,-1)^T = X_2$$

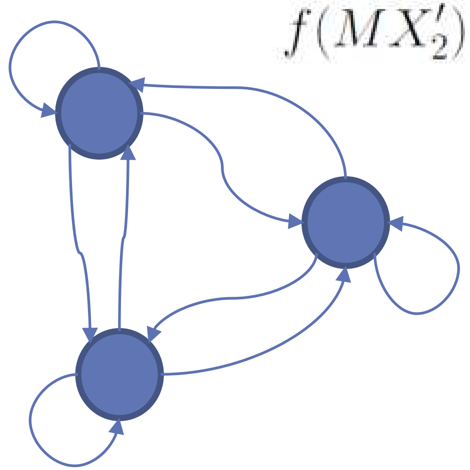
Poskusimo z delno poznanimi vektorji:

$$X_1' = (1,1,0)^T$$

$$X_2' = (1,0,-1)^T$$

$$f(MX_1') = f((2,2,2)^T) = (1,1,1)^T = X_1$$

$$f(MX_2') = f((2,-2,-2)^T) = (1,-1,-1)^T = X_2$$



Pred učenjem so vse uteži enake 0.
Za vsak učni primer:
Če imata nevrona enake vrednosti,
se utež vezi poveča za 1,
sicer se zmanjša za 1

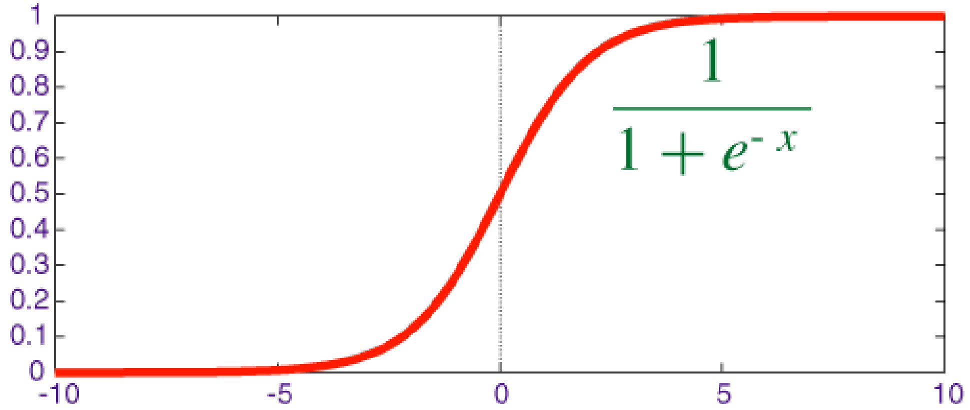
Izvajanje: nevroni izračunavajo izhod po pravilu:

$$Y_i = f(\sum_j W_{ji}X_j)$$

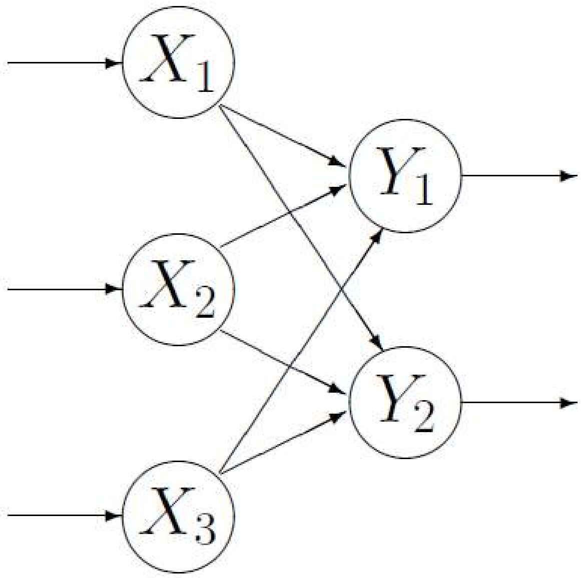
# Umetne nevronske mreže

Nevronske mreže delimo po naslednjih kriterijih:

- **topologija nevronske mreže**: brez nivojev, dvonivojske, večnivojske (globoke)
- **namen nevronske mreže:** avto/hetero-asociativni/časovni pomnilnik, klasifikacija, regresija, razvrščanje, samoorganizacija, razpoznavanje signalov/slik/videoposnetkov/besedil
- **pravilo učenja:** (posplošeno) Hebbovo pravilo, (posplošeno) delta pravilo - gradientno, tekmovalno, pozabljanje.
- **funkcija kombiniranja vhodov nevrona v izhod**:
  - Funkcija aktivacije: utežena vsota, Sigma-pi, Naivni Bayes
  - Izhodna funkcija/normalizacija: pragovna, (ne)deterministična, sigmoidna (odvedljiva)

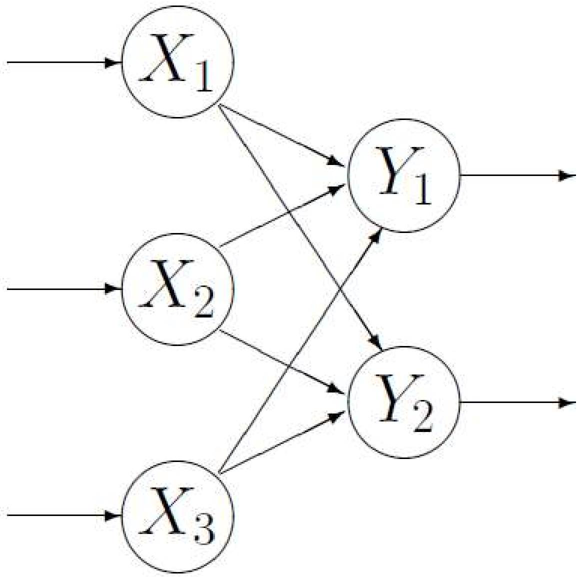


# Dvonivojske (enonivojske) usmerjene nevronske mreže




- Lahko rešijo samo linearne probleme
- Število vhodov = število atributov
- Klasifikacija: Število izhodov = število razredov
- Vsak nevron lahko obravnavamo posebej, saj je izračun neodvisen
- Regresija: En sam izhod (en sam nevron)
- Učna naloga: Nastavi uteži na povezavah tako, da bo mreža uspešno rešila (skoraj vse) učne primere

# Dvonivojske (enonivojske) usmerjene nevronske mreže




**Pravilo delta**

Upošteva razliko med izhodom $Y$ in želenim izhodom $d$:

$$W(n+1) = W(n) + \eta\Big(d(n) - Y(n)\Big)X(n)$$

$$W(n+1) = W(n) + \eta\Big(d(n) - W^T(n)X(n)\Big)X(n)$$

Temu pravilu pravimo tudi *gradientno pravilo.*

Kvadrat napake $E(n)$ je namreč podan z:

$$E(n) = \Big(d(n) - W^T(n)X(n)\Big)^2$$

in je odvod napake enak

$$\frac{dE(n)}{dW(n)} = -2\Big(d(n) - W^T(n)X(n)\Big)X(n)$$

# Dvonivojske (enonivojske) usmerjene nevronske mreže

*Paketna* **varianta pravila delta:**

Izračuna razliko med želenim in dejanskim izhodom za vse učne primere, $X(1)$, ..., $X(m)$.

Napaka je podana z: $E(n) = \Sigma_{i=1}^{m}\left(d(i) - W^T(n)X(i)\right)^2$

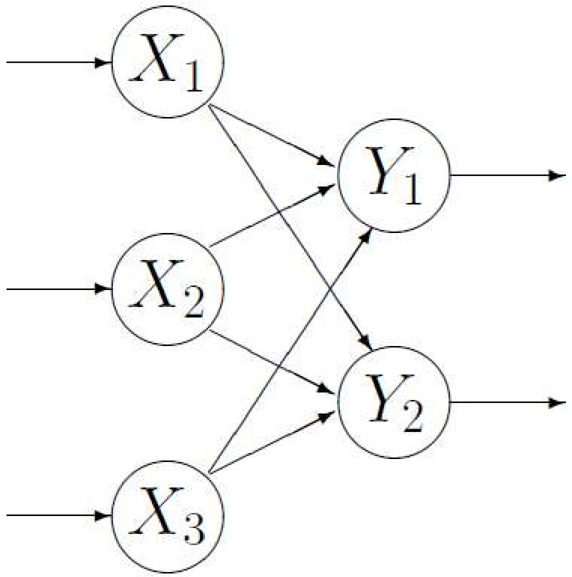
odvod napake pa z:

$$\frac{dE(n)}{dW(n)} = -2\sum_{i=1}^{m}\left(d(i) - W^T(n)X(i)\right)X(i)$$

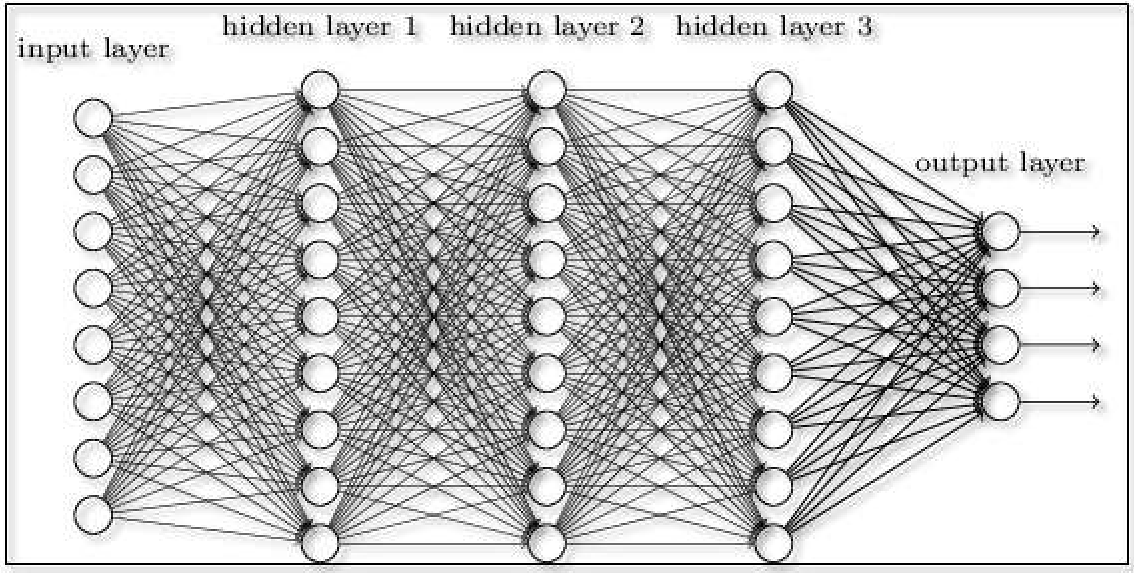
Paketno pravilo delta je:

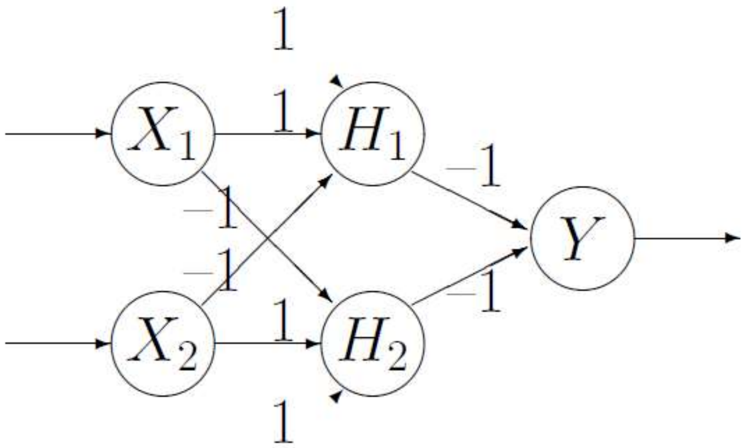
$$W(n+1) = W(n) + \eta\frac{dE(n)}{dW(n)} = W(n) + \eta\sum_{i=1}^{m}\left(d(i) - W^T(n)X(i)\right)X(i)$$

Paketno pravilo upošteva več informacije naenkrat.



# Večnivojske usmerjene nevronske mreže




| $X_1$ | $X_2$ | $Y$ |
|---|---|---|
| 1 | 1 | −1 |
| 1 | −1 | 1 |
| −1 | 1 | 1 |
| −1 | −1 | −1 |

- En ali več skritih nivojev: Lahko rešijo poljuben nelinearni problem
- Število vhodov = število atributov
- Klasifikacija: Število izhodov = število razredov
- Regresija: En sam izhod (en sam nevron)
- Učna naloga:
  - Izberi ustrezno število skritih nivojev in število skritih nevronov na vsakem nivoju
  - Nastavi uteži na povezavah tako, da bo mreža uspešno rešila (skoraj vse) učne primere

# Večnivojske usmerjene nevronske mreže

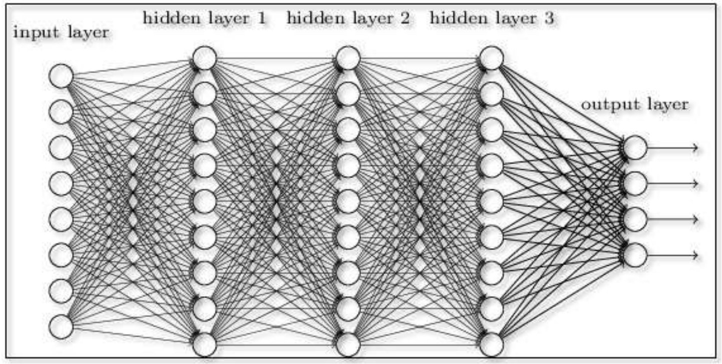
*Posplošeno pravilo delta* **ali** *pravilo vzvratnega razširjanja napake* **(backpropagation of error):**

1. Na začetku so uteži naključne.
2. Na vhodu mreža dobi vhodni vzorec in izračuna izhod.
3. Zatem se izračuna razlika med dejanskim in želenim izhodom.
4. Najprej se spremenijo uteži med zadnjim in predzadnjim nivojem kot pri osnovnem pravilu delta.

5.1 Zatem se izračunajo želene vrednosti nevronov na predzadnjem nivoju.

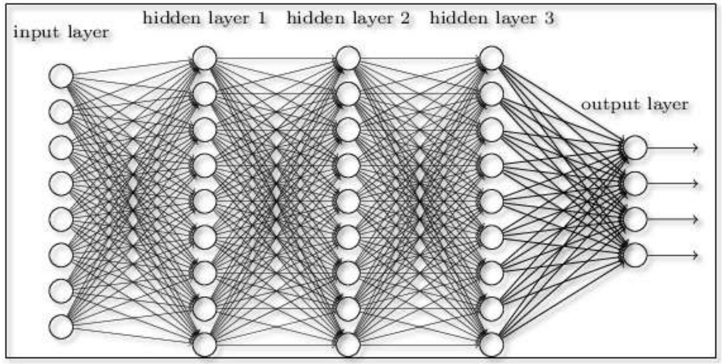
5.2 Izračuna se razlika med želenimi in dejanskimi vrednostmi nevronov na predzadnjem nivoju.

5.3 Rekurzivno se nadaljuje spreminjanje uteži vse do vhodnega nivoja nevronov.



# Večnivojske usmerjene nevronske mreže

**Slabosti posplošenega pravila delta**



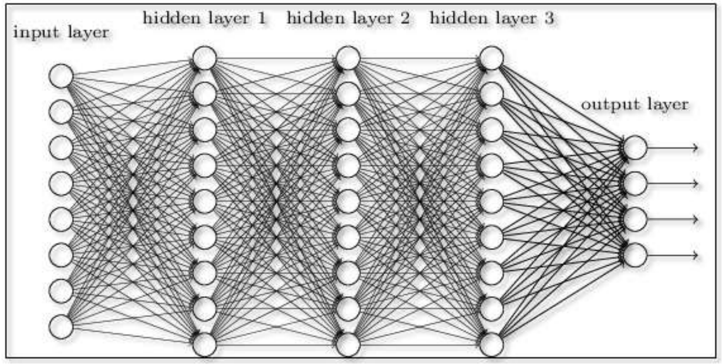

1. Ne konvergira vedno k optimalni mreži (lahko obtiči v lokalnem minimumu).
2. Problematična je izbira topologije mreže.
3. Preveliko prileganja učni množici.
4. Nastavitev parametra $\eta$ vpliva na stabilnost.
5. Zahteva zelo veliko število prehodov preko učnih primerov.
6. Pravilo delta nima biološke analogije z možgani.

Prve tri probleme se da omiliti...

# Večnivojske usmerjene nevronske mreže

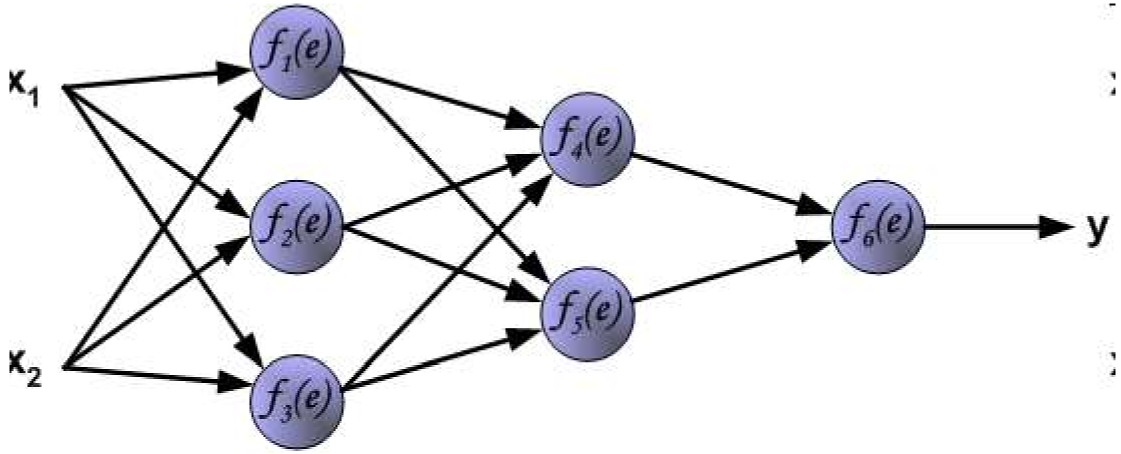
1. Vpeljemo momentni člen:

$$W_{ji}(n+1) = W_{ji}(n) - \Delta W_{ji}(n+1) - \alpha \Delta W_{ji}(n)$$



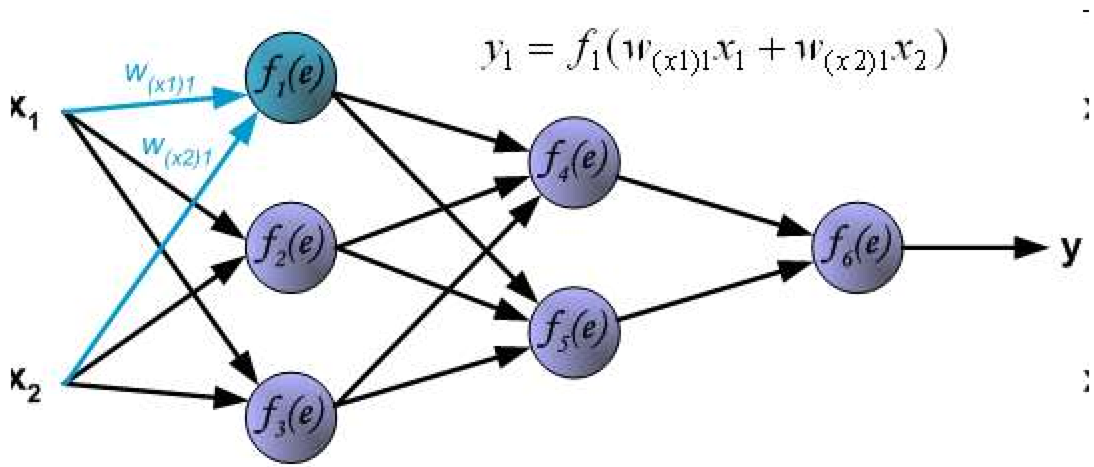

2. Metoda eliminacije uteži funkciji napake doda člen, ki "kaznuje" velike uteži.
   - Učenje začnemo s preveliko mrežo in med učenjem se odstranijo odvečni nevroni.
   - Hkrati se zaradi avtomatske nastavitve optimalne velikosti mreže izognemo tudi prevelikemu prileganju učni množici.
   - Zato pa metoda vpelje še dodatne parametre za kontrolo učenja, ki jih ni trivialno nastaviti.

# ANN Backpropagation

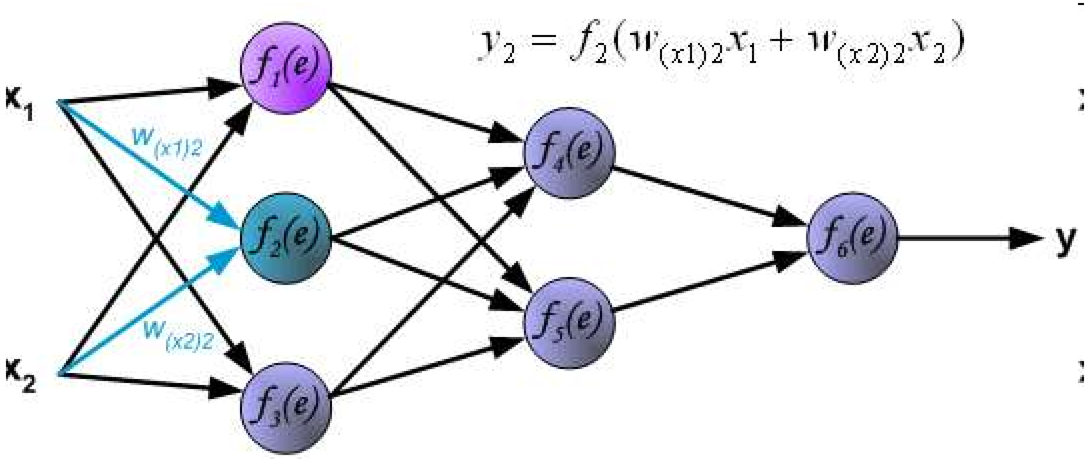



http://home.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

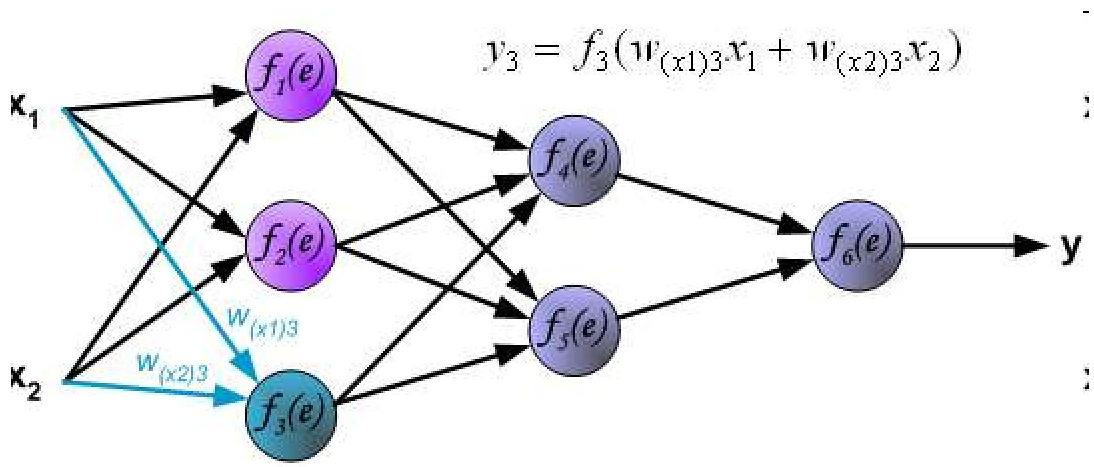
# ANN Backpropagation



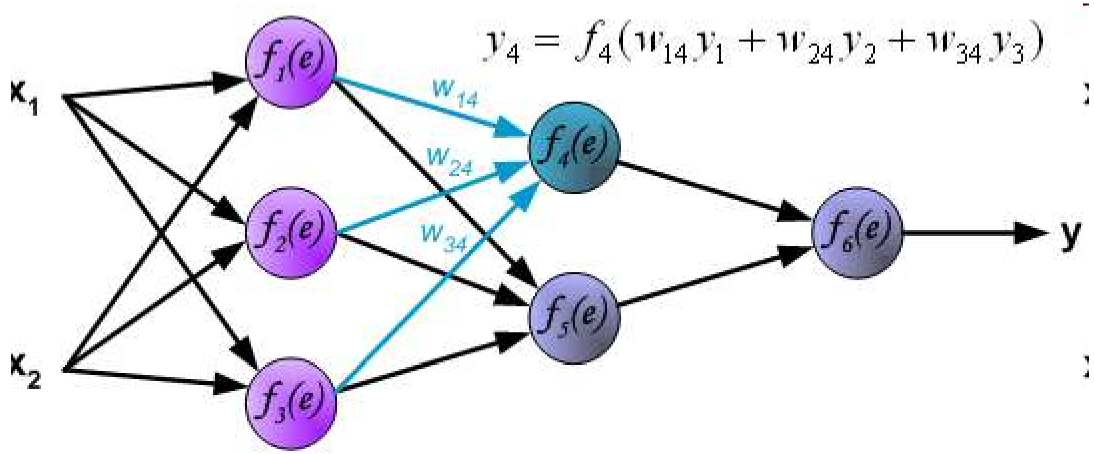
# ANN Backpropagation



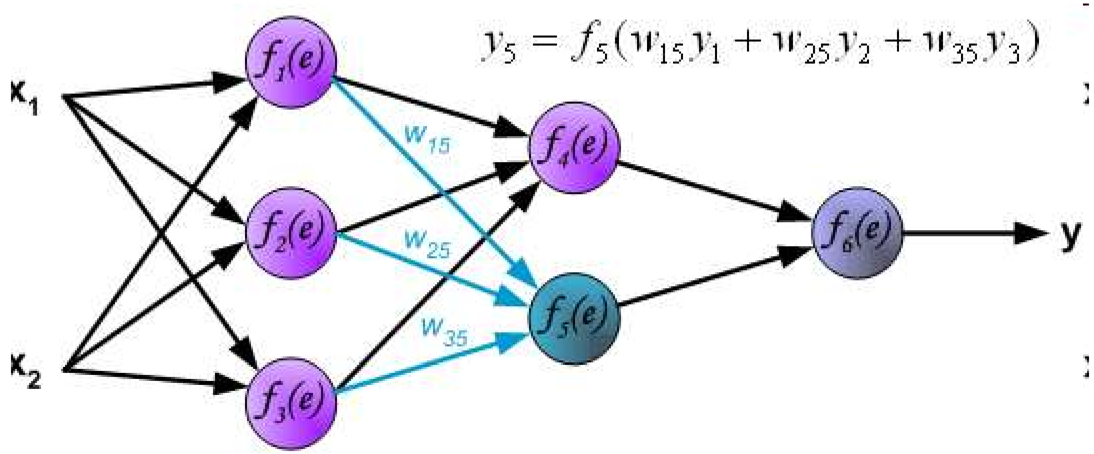
# ANN Backpropagation



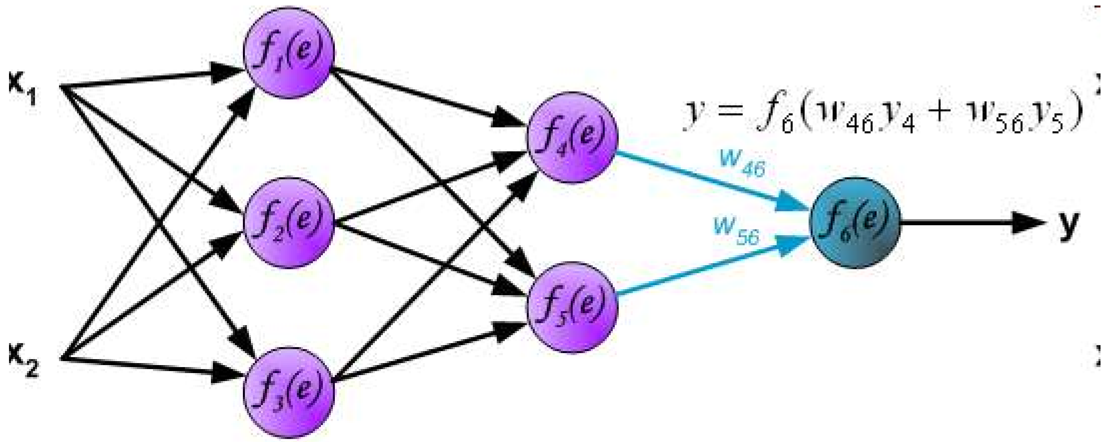
# ANN Backpropagation



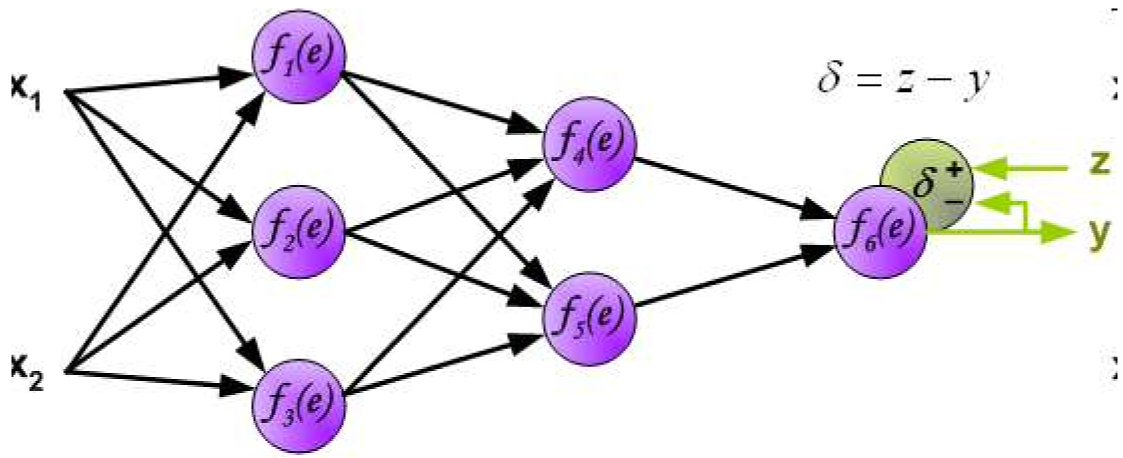
# ANN Backpropagation



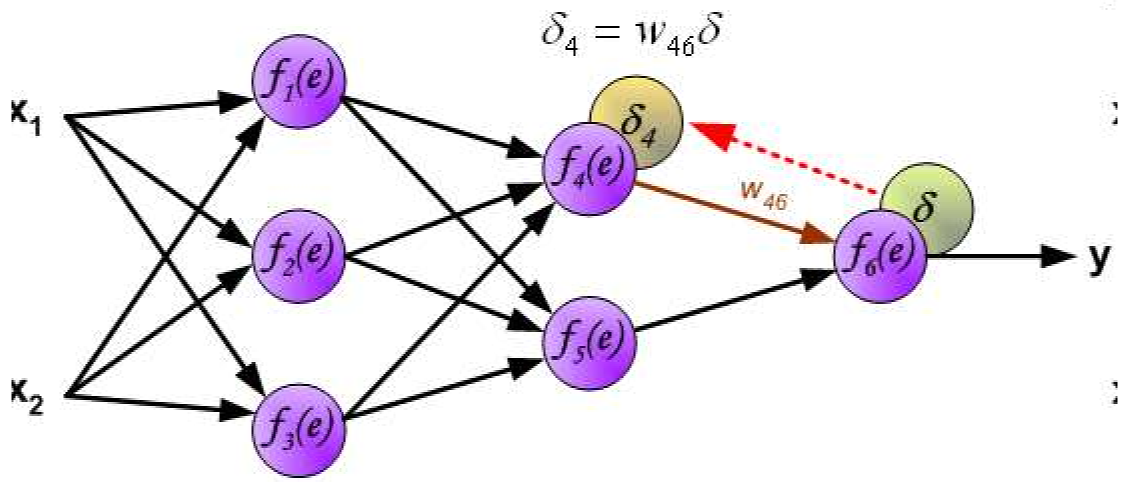
# ANN Backpropagation



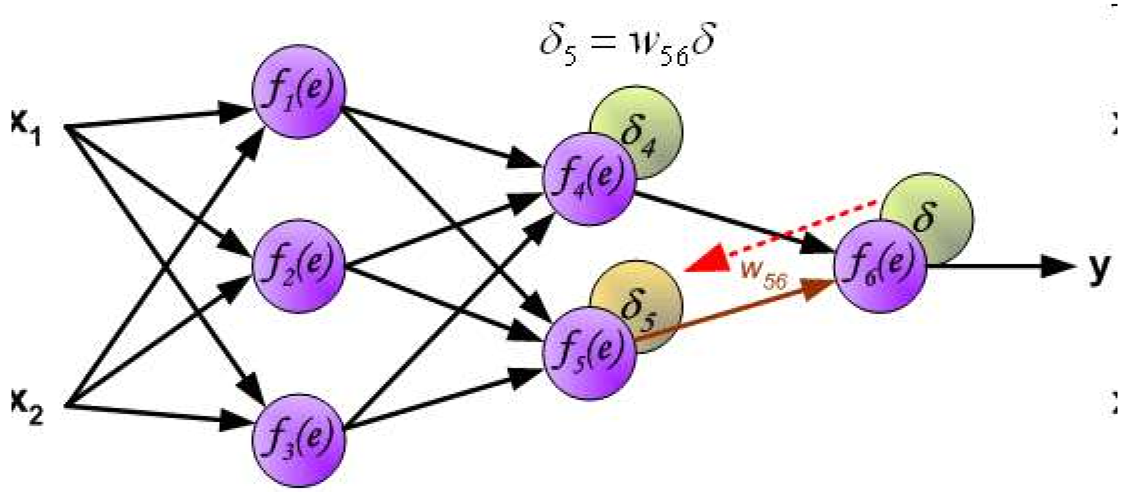
# ANN Backpropagation



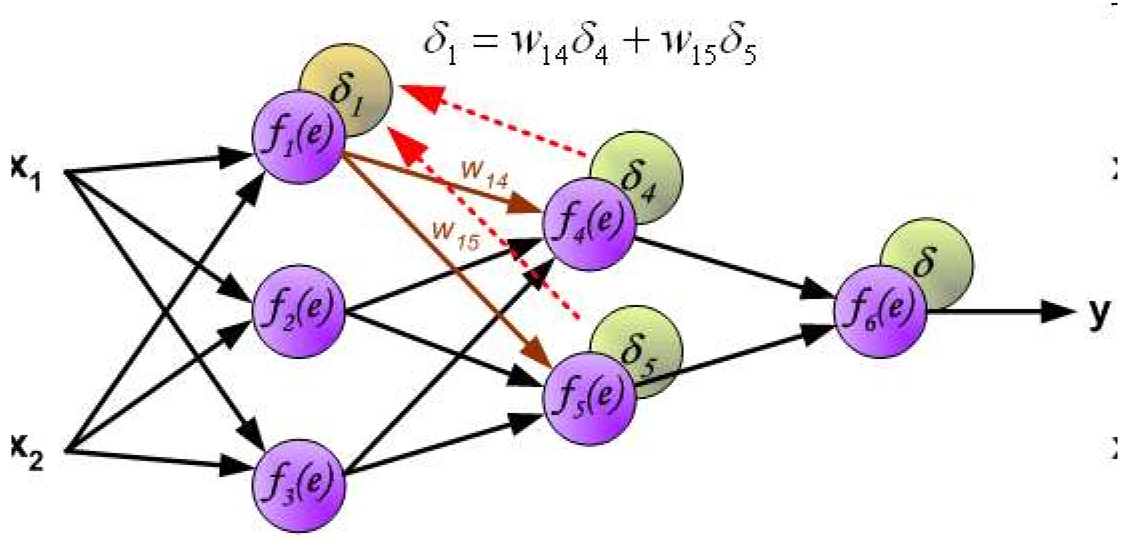
# ANN Backpropagation



# ANN Backpropagation

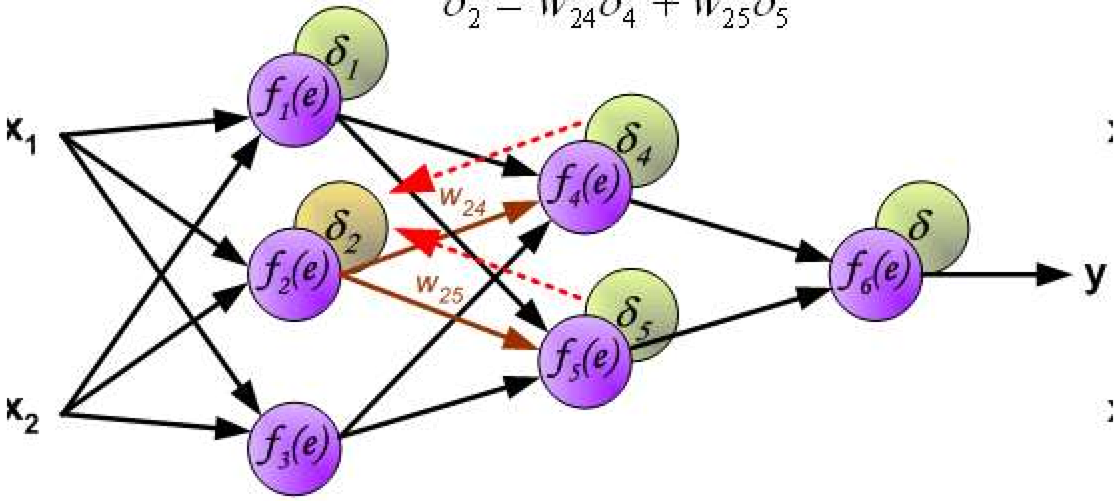


# ANN Backpropagation



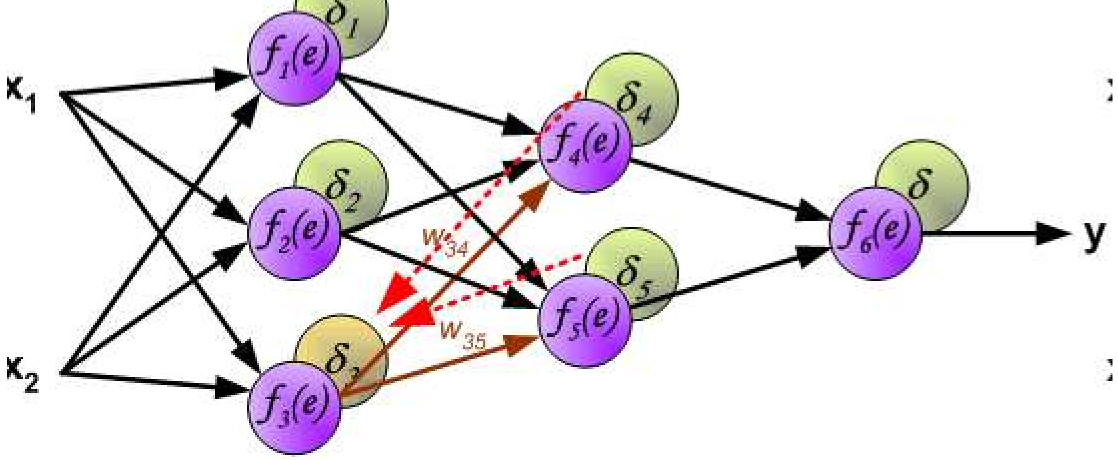
# ANN Backpropagation

$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$$



# ANN Backpropagation

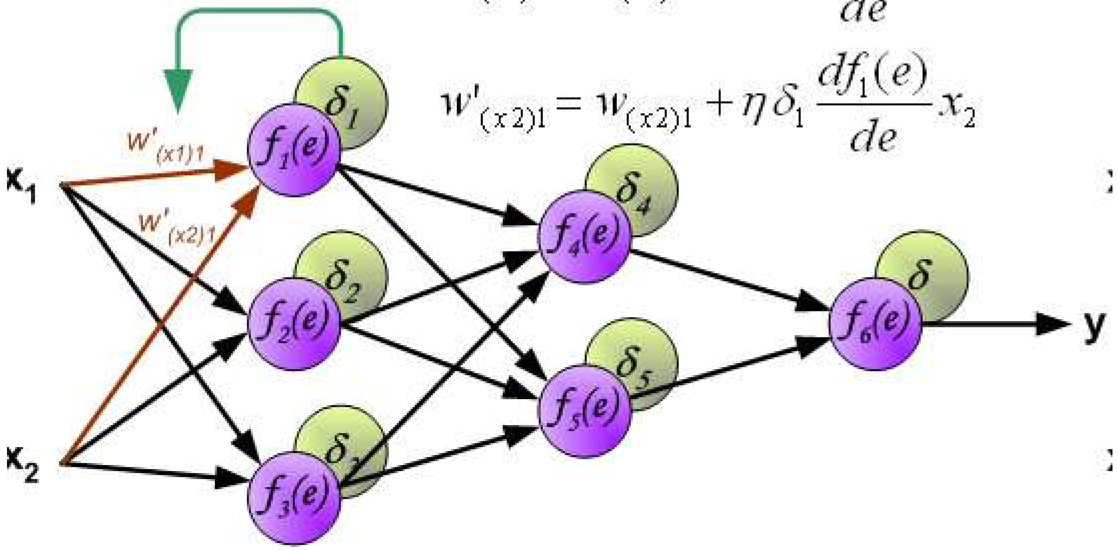
$$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5$$



# ANN Backpropagation

$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

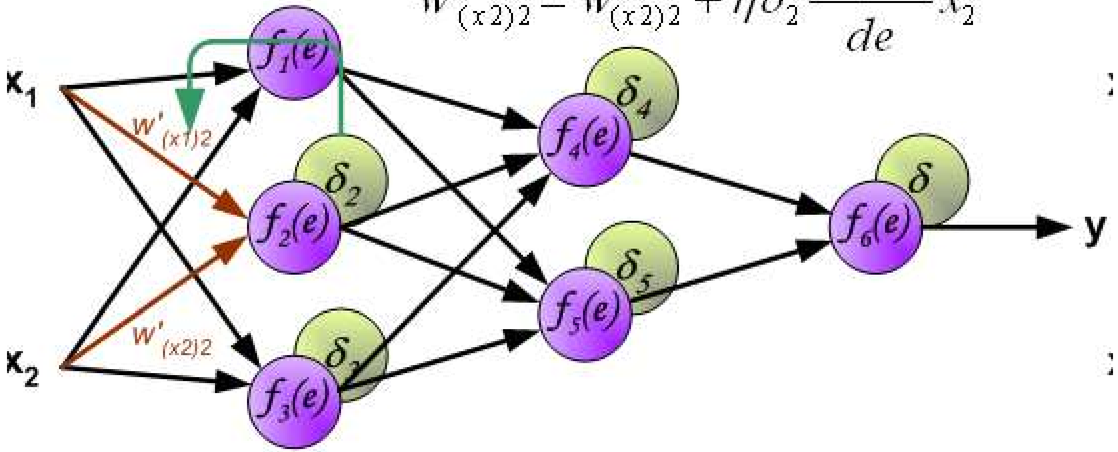
$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



# ANN Backpropagation

$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

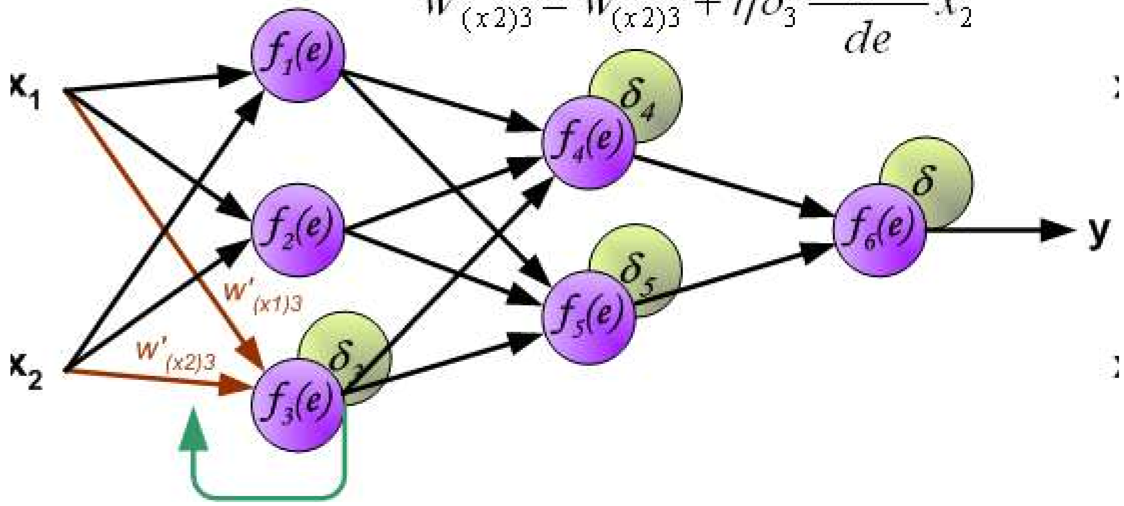
$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



# ANN Backpropagation

$$w'_{(x1)3} = w_{(x1)3} + \eta \delta_3 \frac{df_3(e)}{de} x_1$$

$$w'_{(x2)3} = w_{(x2)3} + \eta \delta_3 \frac{df_3(e)}{de} x_2$$

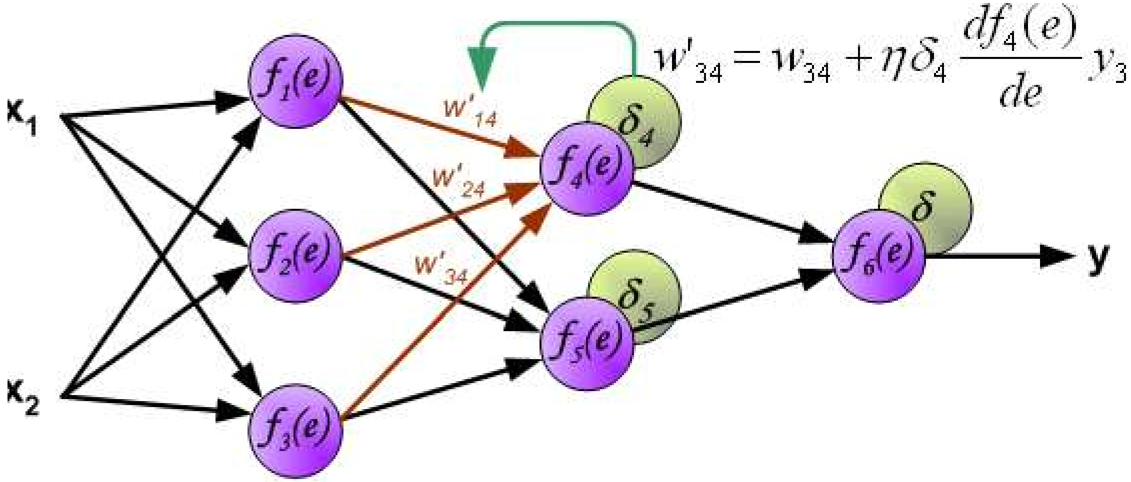


# ANN Backpropagation

$$w'_{14} = w_{14} + \eta \delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta \delta_4 \frac{df_4(e)}{de} y_2$$

$$w'_{34} = w_{34} + \eta \delta_4 \frac{df_4(e)}{de} y_3$$

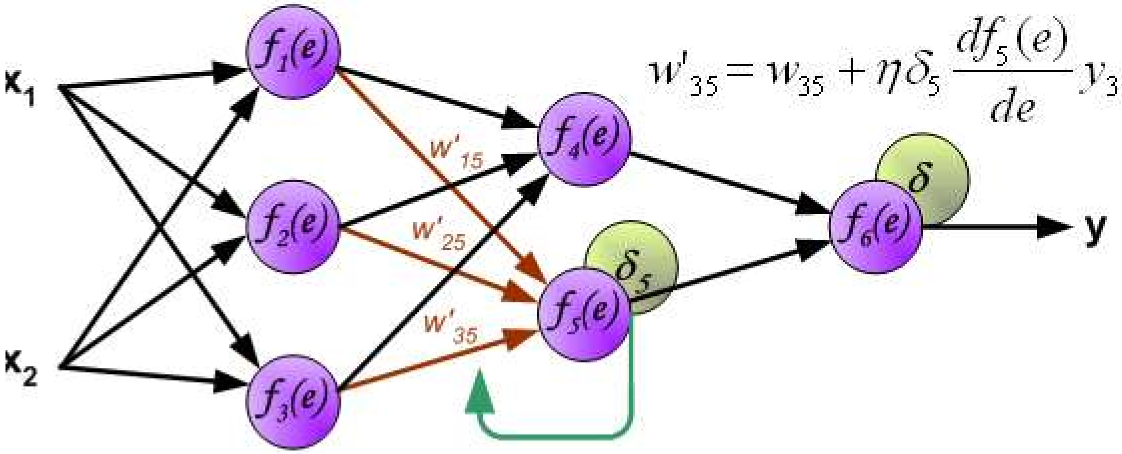


# ANN Backpropagation

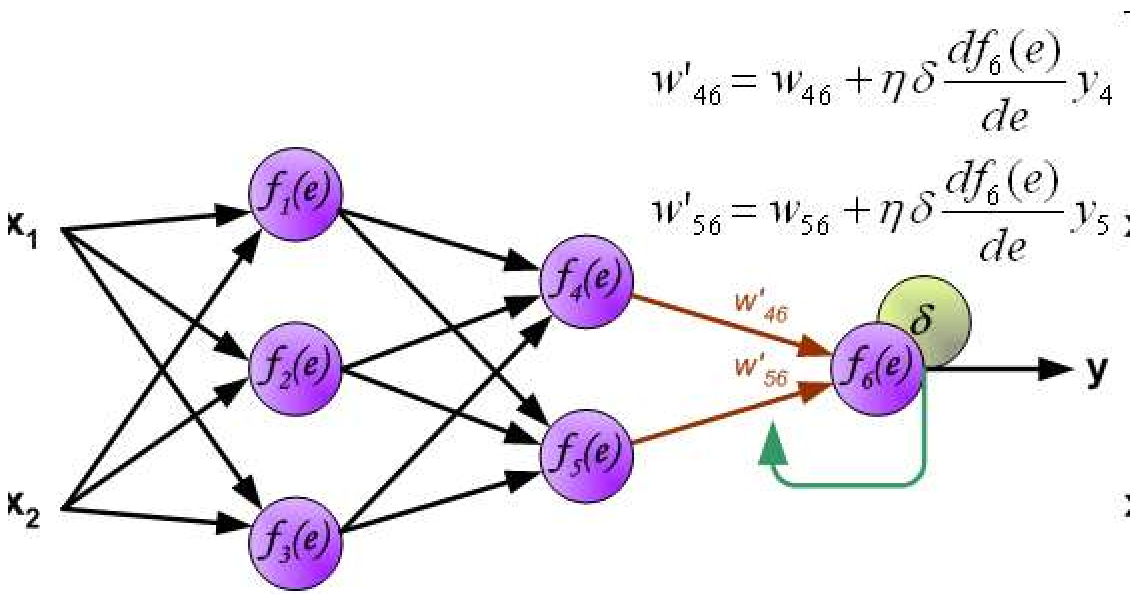
$$w'_{15} = w_{15} + \eta\delta_5 \frac{df_5(e)}{de} y_1$$

$$w'_{25} = w_{25} + \eta\delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta\delta_5 \frac{df_5(e)}{de} y_3$$

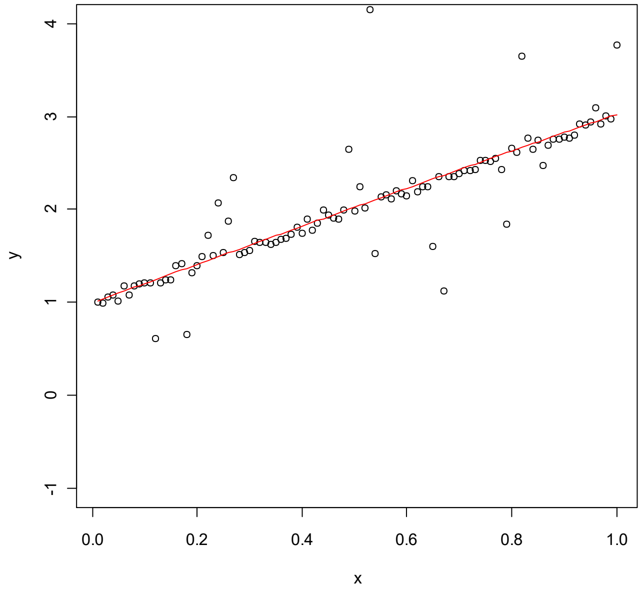


# ANN Backpropagation

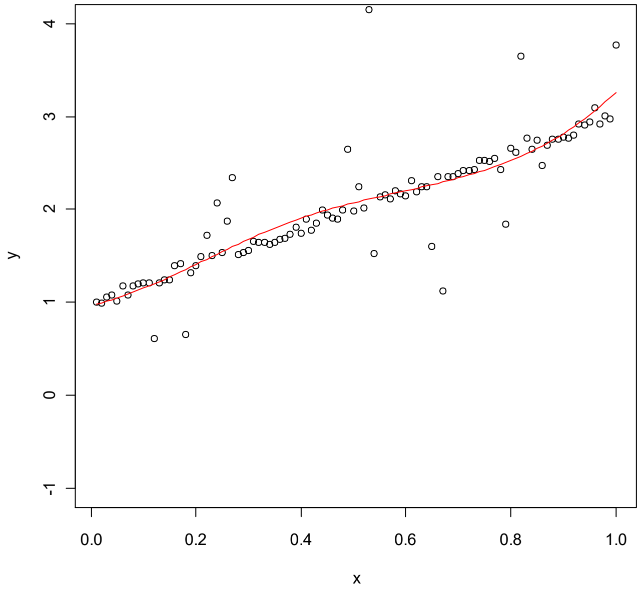


# ANN Overfitting

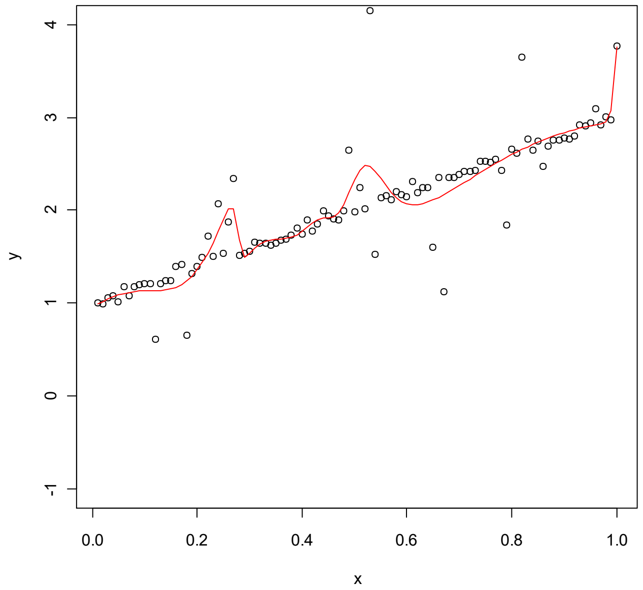
(1)



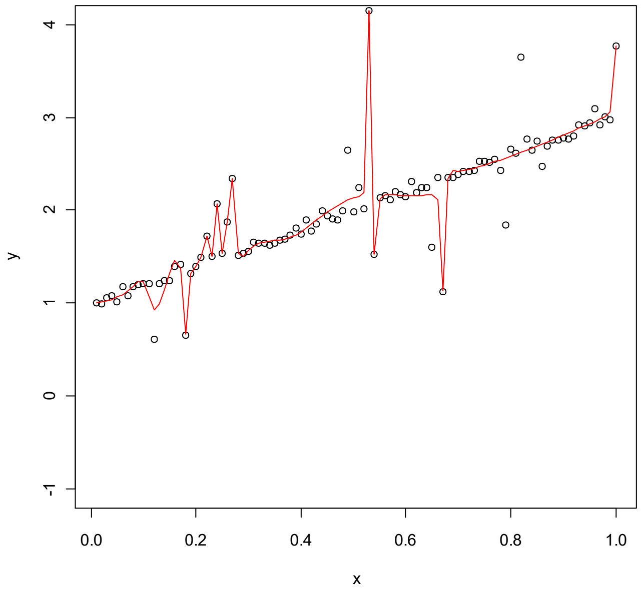

(3)




(15)




(50)




(# of neurons in hidden layer)