Řešení domácích úloh

Napište skript (nebo funkci) v Matlabu, který nastaví architekturu, váhy a prahy a prahy perceptronové síťě tak, aby implementovala funkci XOR.

1. Pro binární model
2. Pro biparitní model

Program otestuje, že sítě dávají pro všechny vstupy správný výstup.

Algoritmus zpětného šíření (Back-propagation)

(Werbos, Rumelhart, 1974-1986)

Máme k dispozici

- Trénovací množina T s N trénovacími vzory (x^p, d^p).
  - $x^p = (x_1^p, ..., x_n^p)$ ... vstupní vzor
  - $d^p = (d_1^p, ..., d_m^p)$ ... požadovaný výstup
- Vrstevnatá neuronová síť s danou architekturou s n vstupními a m výstupními neurony. Neurony musí mít **spojitou, diferencovatelnou** přenosovou funkci.

Problém

- Nastavit váhy a práhy všech neuronů v síti tak, aby byl skutečný výstup sítě stejný jako požadovaný.

Neuron se sigmoidální přenosovou funkcí

- binární model: $f(\xi) = \frac{1}{1+e^{-\lambda\xi}}$ ... logsig
- biparitní model: $f(\xi) = \frac{1-e^{-\lambda\xi}}{1+e^{-\lambda\xi}}$ ... tansig
- parametr *lambda* ... strmost
  určuje míru nepřesnosti výsledku (klasifikace na hranicích)
  - *lambda* $\rightarrow \infty$ ... diskrétní model
  - čím je *lambda* menší ... tím je širší hranice mezi odlišnými případy
  - *lambda* $\rightarrow 0$ ... neuron nerozlišuje (výstup vždy 0.5 resp. 0)
  - Obvyklá volba $\lambda = 1$ nebo $\lambda = 1/4$ pro logsig, $\lambda = 2$ pro tansig

Algoritmus zpětného šíření (Back-propagation)

Cílová (chybová) funkce

- MSE ... vyjadřuje odchylku mezi skutečnou a požadovanou odezvou sítě:
  pro jeden trénovací vzor: $E^p = \frac{1}{2} \sum_i (d_i^p - y_i^p)^2$
  pro celou trénovací množinu:
  $E = \frac{1}{N} \sum_p E^p = \frac{1}{2N} \sum_p \sum_i (d_i^p - y_i^p)^2$
  i je index přes výstupní neurony
  p je index přes trénovací vzory

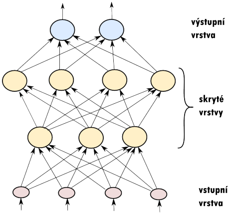
Cíl algoritmu zpětného šíření

- minimalizace chybové funkce E na dané trénovací množině T

Algoritmus zpětného šíření (Back-propagation)

Základní princip

- Spočteme skutečnou odezvu sítě pro daný trénovací vzor.
- Porovnáme skutečnou a požadovanoy odezvu sítě.
- Adaptujeme váhy a prahy:
  - proti směru gradientu chybové funkce
  - od výstupní vrstvy směrem ke vstupní



Back-propagation - Adaptační pravidla

Aktualizace synaptických vah a prahů proti směru gradientu chybové funkce

- Pro váhu z neuronu i do neuronu j v čase t platí:

$$w_{ij}^{t+1} = w_{ij}^{t} + \triangle w_{ij}^{t},$$

- Pro práh neuronu j v čase t platí:

$$h_{j}^{t+1} = h_{j}^{t} + \triangle h_{j}^{t},$$

- $\triangle w_{ij}^{t}$ je přírůstek váhy w_{ij} přispívající k minimalizaci E^{t}. Po vynechání indexu trénovacího vzoru t:

$$\triangle w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}} = -\alpha \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} \frac{\partial \xi_j}{\partial w_{ij}}$$

$$\triangle h_{j} = -\alpha \frac{\partial E}{\partial hj} = -\alpha \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} \frac{\partial \xi_j}{\partial h_j}$$

- α je parametr učení (reguluje délku adaptačního kroku)

Back-propagation - Adaptační pravidla

Aktualizace synaptických vah pro výstupní vrstvu

$$
\begin{aligned}
\triangle w_{ij} &= -\alpha \frac{\partial E}{\partial w_{ij}} = -\alpha \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} \frac{\partial \xi_j}{\partial w_{ij}} \\
&= -\alpha (y_j - d_j) f'(\xi_j) y_i \\
&= -\alpha \delta_j y_i
\end{aligned}
$$

Aktualizace prahů pro výstupní vrstvu

$$
\begin{aligned}
\triangle h_j &= -\alpha \frac{\partial E}{\partial hj} = -\alpha \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} \frac{\partial \xi_j}{\partial h_j} \\
&= -\alpha (y_j - d_j) f'(\xi_j) 1 \\
&= -\alpha \delta_j
\end{aligned}
$$

Back-propagation - Adaptační pravidla

Aktualizace synaptických vah pro skryté vrstvy

$$
\begin{aligned}
\triangle w_{ij} &= -\alpha \frac{\partial E}{\partial w_{ij}} = -\alpha \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} \frac{\partial \xi_j}{\partial w_{ij}} \\
&= -\alpha \Big(\sum_k \frac{\partial E}{\partial \xi_k} \frac{\partial \xi_k}{\partial y_j}\Big) f'(\xi_j) y_i \\
&= -\alpha \Big(\sum_k \frac{\partial E}{\partial \xi_k} w_{jk}\Big) f'(\xi_j) y_i \\
&= -\alpha \Big(\sum_k \delta_k w_{jk}\Big) f'(\xi_j) y_i \\
&= -\alpha \delta_j y_i
\end{aligned}
$$

Back-propagation - Adaptační pravidla

Aktualizace prahů pro skryté vrstvy

$$\begin{aligned}
\triangle h_j &= -\alpha \frac{\partial E}{\partial hj} = -\alpha \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} \frac{\partial \xi_j}{\partial h_j} \\
&= -\alpha (\sum_k \frac{\partial E}{\partial \xi_k} \frac{\partial \xi_k}{\partial y_j}) f'(\xi_j) 1 \\
&= -\alpha (\sum_k \delta_k w_{jk}) f'(\xi_j) \\
&= -\alpha \delta_j
\end{aligned}$$

Back-propagation - Adaptační pravidla

Celkem:

$$w_{ij}^{t+1} = w_{ij}^{t} + \alpha \delta_j y_i,$$

$$h_j^{t+1} = h_j^{t} + \alpha \delta_j,$$

kde

- pro výstupní neuron j:

$$\delta_j = f'(y_j)(y_j - d_j).$$

- pro skrytý neuron j:

$$\delta_j = f'(y_j) \sum_k (\delta_k w_{jk}).$$

Back-propagation - Adaptační pravidla

Výpočet derivace přenosové funkce

- Sigmoidální funkce: logsig / dlogsig

$$f(\xi_j) = \frac{1}{1 + e^{-\lambda \xi_j}}$$

$$f'(\xi_j) = \lambda y_j (1 - y_j)$$

- Hyperbolický tangens: tansig / dtansig

$$f(\xi_j) = \frac{1 - e^{-\lambda \xi_j}}{1 + e^{-\lambda \xi_j}}$$

$$f'(\xi_j) = \lambda^2 (1 + y_j)(1 - y_j)$$

- Lineární funkce: purelin / dpurelin

$$f(\xi_j) = \xi_j$$

$$f'(\xi_j) = 1$$

Algoritmus zpětného šíření

1. **Inicializace sítě**
   Inicializujeme váhy a prahy malými náhodnými hodnotami
2. **Předložíme další trénovací vzor** ve tvaru $(\overrightarrow{x}, \overrightarrow{d})$
   $\overrightarrow{x} = (x_1, ..., x_n)$ ... vstupní vzor
   $\overrightarrow{d} = (d_1, ..., d_m)$... požadovaný výstup

Algoritmus zpětného šíření

3. **Dopředný výpočet**
   Vypočteme skutečný výstup (odezvu) sítě $\overrightarrow{y} = (y_1, ..., y_m)$:
   pro každou vrstvu L (ve směru od vstupní k výstupní):
   spočteme aktivitu (výstup) každého neuronu ve vrstvě L
   (využijeme k tomu aktivity neuronů v předchozí vrstvě)
   - Pro každý neuron j spočteme jeho aktivitu: $y_j = f(\xi_j)$,
     kde $\xi_j = \sum_i y_i w_{ij}$
     (i je index přes neurony ve vrstvě pod neuronem j)

Algoritmus zpětného šíření

4. **Zpětný výpočet**
   Spočteme chybový člen δ_j pro každý neuron j a aktualizujeme všechny váhy a prahy v síti (ve směru od výstupní vrstvy k vstupní):
   - Pro výstupní neurony spočteme: $\delta_j = f'(y_j)(y_j - d_j)$.
   - Pro skryté neurony spočteme: $\delta_j = f'(y_j)\sum_k(\delta_k w_{jk})$.
     (k je index přes neurony ve vrstvě nad neuronem j)
   - Pro váhu každé hrany (z neuronu i do neuronu j):
     $w_{ij}^{t+1} = w_{ij}^{t} + \triangle w_{ij}$,
     kde $\triangle w_{ij} = -\alpha\delta_j y_i$.
   - Pro práh každého neunonu j:
     $h_j^{t+1} = h_j^{t} + \triangle h_j$,
     kde $\triangle h_j = -\alpha\delta_j$.

Algoritmus zpětného šíření

5. **Ukončující podmínka**
   Pokud není splněna ukončující podmínka, vrátíme se zpět ke kroku 2.
   - Maximální počet cyklů (= epoch = předložení celé trénovací množiny).
   - Časový limit.
   - Dolní mez pro hodnotu chybové funkce na trénovací množině.
   - Dolní mez pro střední hodnotu přírůstku vah $\triangle w_{ij}$ v jednom cyklu.
   - Využití tzv. validační množiny – tzv. early stopping.

Algoritmus zpětného šíření

Učící (trénovací strategie)
způsob předkládání trénovacích vzorů

- iterace = předložení jednoho trénovacího vzoru ... i vícekrát
- epocha (cyklus) = iterace přes celou trénovací množinu

Typický průběh chybové funkce během učení

- klesá v čase, mohou ale existovat úseky s roztoucí E
- algoritmus snižuje chybu pro aktuální vzor $\rightarrow$ chyba se může z zvýšit pro ostatní vzory

BP sítě - analýza modelu

- Jeden z nejpoužívanějších modelů
- Jednoduchý algoritmus učení
- Poměrně dobré výsledky - aproximační a generalizační schopnosti

Nevýhody

- Interní reprezentace znalostí (černá skříňka).
- Učení s učitelem. Trénovací množina by měla být dostatečně velká a vyvážená.
- Aproximační a generalizační schopnosti: volba vhodné architektury a dalších parametrů (přenosové funkce, parametr učení, strmost sigmoidy) pro danou úlohu.
- Pomalá konvergence, lokální minima, nebezpečí přeučení

Aproximační schopnost BP-sítí

Věta (Kolmolgorov)

- Libovolnou spojitou funkci $f : [0,1]^n \rightarrow (0,1)^m$ lze vyjádřit (aproximovat libovolně přesně) pomoci BP-sítě s odpovídajícím počtem neuronů a s vhodnou přenosovou funkcí.
    - Pro neurony se sigmoidální přenosovou funkcí stačí dvě skryté vrstvy [1987].
    - $\times$ RBF sítě - stačí jedna skrytá vrstva [1990].

V praxi je nutné určit (pro danou úlohu)

- volba vhodné architektury (počet vrstev a počet neuronů v těchto vrstvách, propojení neuronů)
- volba vhodných přenosových funkcí
- volba algoritmu učení (adaptace vah a prahů) a učící strategie

NP-úplnost problému učení

Věta

- Obecný problém učení umělých neuronových sítí je NP-úplný. Výpočetní náročnost roste exponenciálně s počtem proměnných.

Poznámka

- Pro některé speciální typy jednoduchých neuronových sítí je problém učení řešitelný v polynomiálním čase (pomocí metod lineárního programování).

Porovnávání algoritmů učení (modelů)

Problém

- Chci srovnat algoritmy nebo modely L_A a L_B .. který z nich lépe aproximuje hledanou funkci f?
- Mám k dispozici jen omezenou trénovací množinu dat T

Obvyklé metody

- Párový test
- K-násobná křížová validace

Testování algoritmů

Skutečná chyba algoritmu

- f je aproximovaná funkce
- L_T je funkce naučená algoritmem L na množine T
- $ERR_D(L_T) = Pr_D(L_T(x) \neq f(x))$
  D . . . pravděpodobnost výskytu případu x v trénovací množině

Jak odhadnout $ERR_D(L_T)$? Jak dobrý bude tento odhad?

Vyhodnocení kvality modelu I. V praxi

Vhodná chybová funkce

1. MSE (střední kvadratická chyba)... vyjadřuje odchylku mezi skutečnou a požadovanou odezvou sítě:
   pro jeden trénovací vzor: $E^p = \frac{1}{2}\sum_i (d_i^p - y_i^p)^2$
   pro celou trénovací množinu:
   $E = \frac{1}{N}\sum_p E^p = \frac{1}{2N}\sum_p \sum_i (d_i^p - y_i^p)^2$
   i je index přes výstupní neurony
   p je index přes trénovací vzory
2. MAE (střední absolutní chyba)
3. Klasikační chyba

Vyhodnocení kvality modelu I. V praxi

Vhodná chybová funkce

1. MSE (střední kvadratická chyba)
2. MAE (střední absolutní chyba)
   pro jeden trénovací vzor: $E^p = \frac{1}{2}\sum_i |d_i^p - y_i^p|$
   pro celou trénovací množinu:
   $E = \frac{1}{N}\sum_p E^p = \frac{1}{2N}\sum_p \sum_i |d_i^p - y_i^p|$
3. Klasikační chyba - procento chybně klasifikovaných vzorů:
   pro celou trénovací množinu: $E = \frac{1}{N}\sum_p AND_i|d_i^p = y_i^p|$
   i je index přes výstupní neurony
   p je index přes trénovací vzory

Vyhodnocení kvality modelu I. V praxi

- Chyba na trénovacích datech (na kterých se model učil) není vypovídající ... *Proč?*
- Množinu trénovacích dat T rozdělím na disjunktní množiny Tr (trénovací) a S (testovací). S musí být dostatečně velká (alespoň 30 vzorků).
- Model naučím na Tr.
- Chybu modelu spočítám na množině Tr a na množině S.
- Obvyklé dělení: 70% vzorků trénovací množina Tr, 30% vzorků testovací množina S.

Vyhodnocení kvality modelu I. V praxi

Statisticky významné výsledky

- Model naučím k-krát pro různé náhodné inicializace vah (typicky minimálně $k = 100$) za použití trénovací množiny Tr a testovací množiny S.
- Pokaždé spočtu chybu modelu na testovací množině (E_S^k).
- Spočtu odhad střední hodnoty a směrodatné odchylky chyby $mean_k E_S^k$ a $std_k E_S^k$
- Obdobně mohu odhadnout chybu na trénovací množině (E_{Tr}^k) a časovou náročnost učení (čas učení, počet cyklů)

Vyhodnocení kvality modelu II. Precizně

Testování algoritmů

Jak odhadnout $ERR_D(L_T) = Pr_D(L_T(x) \neq f(x))$? Jak dobrý bude tento odhad?

- Jako odhad použiji $ERR_S(L_T) = \frac{1}{N_S} \sum_{x \in S} I_{f(x) \neq L_T(x)}$
  S je testovací množina dat ... binomické rozdělení
- **Pro oboustranný interval spolehlivosti:**
  S pravděpodobností 95% leží $ERR_D(L))$ v intervalu
  $ERR_S(L_T) \pm 1.96 \sqrt{\frac{ERR_S(L_T) * (1 - ERR_S(L_T))}{N_S}})$

Testování algoritmů II. Precizně

Párový test – obecně:

- Odhaduji $\delta = ERR_D(L_A) - ERR_D(L_B)$
- Odhad pomocí $E_{S\subset D}(ERR_S(L_A) - ERR_S(L_B))$ střední hodnota přes všechny N-prvkové podmnožiny S vybrané podle rozdělení D.

P %-ní interval spolehlivosti odhadu $\hat{\delta}$:

- $\hat{\delta} \pm z_P s_{\hat{\delta}}$, $s_{\hat{\delta}}$ je odhad směrodatné odchylky:
  $s_{\hat{\delta}} = \sqrt{\frac{ERR_S(L_A)*(1-ERR_S(L_A))+ERR_S(L_B)*(1-ERR_S(L_B))}{N_S}}$
- z_P je $\frac{P+100}{2}$ %-ní kvantil normálního rozdělení
- v Matlabu z_P spočteme pomocí norminv((P+100)/200,0,1)

K-násobná křížová validace

1. Množinu dat T rozdělíme do k disjunktních stejně velkých podmnožin $T_1, \cdots T_k$, kde $|T_i| \geq 30$, $i = 1, \cdots, k$. Obvykle $k = 10$
2. Pro $i = 1, \cdots, k$:
   použij $T \setminus T_i$ jako trénovací množinu Tr
   použij T_i jako testovací množinu S
   nauč algoritmy L_a a L_b na Tr
   spočti chyby $E(A)_i = ERR_S(L_A)$ a $E(B)_i = ERR_S(L_B)$ na množině S.
   spočti rozdíl $\delta_i = E(A)_i - E(B)_i$
3. Vrať $\frac{1}{k}\sum_{i=1}^{k} E(A)_i$, $\frac{1}{k}\sum_{i=1}^{k} E(B)_i$, resp. $\bar{\delta} = \frac{1}{k}\sum_{i=1}^{k} \delta_i$ a odhady směrodatných odchylek

K-násobná křížová validace

P %-ní interval spolehlivosti:

- $\overline{\delta} \pm t_{P,k-1} s_{\overline{\delta}}$, $s_{\overline{\delta}}$ je odhad směrodatné odchylky: $s_{\overline{\delta}} = \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (\delta_i - \overline{\delta})^2}$
- $t_{P,k-1}$ je $\frac{P+100}{2}$ %-ní kvantil t-rozdělení s $k-1$ stupni volnosti.
- v Matlabu $t_{P,k-1}$ spočteme pomocí tinv((N+100)/200,k-1)

Cvičení - BP-sítě v Matlabu

User Guide $\rightarrow$ Backpropagation

- Nejprve sestavíme trénovací množinu dat:
  p ... vstupní vzory $n \times N$, t ... výstupní vzory $m \times N$
- Vytvoříme neuronovou síť:
  net = newff(p,t,[5,3],{'tansig','tansig','purelin'},'traingd');
  parametry:
  - Matice vstupních trénovacích vzorů
  - Matice požadovaných výstupů
  - Řádkový vektor = počty neuronů ve skrytých vrstvách. Podle délky vektoru se určí počet skrytých vrstev.
  - Jména přenosových funkcí použitých ve skrytých vrstvách a ve výstupní vrstvě.
  - Jméno učícího algoritmu
  - Další parametry a defaultní hodnoty ... podívejte se do Nápovědy Matlabu

Cvičení - BP-sítě v Matlabu

Příklad 1 - Začátek

- p = [-100:100; -1.*ones(1,100) 0 ones(1,100)];
  t = [100:-1:0 1:100];
- Vstupní i výstupní vzory normalizujte pomocí mapminmax a náhodně je seřaďte (pomocí randperm, p i t stejně!!).
- Vytvořte pro tuto trénovací množinu neuronovou síť s jednou skrytou vrstvou, 5 neurony v této skryté vrtsvě, přenosovou funkcí tansig ve skryté vrstvě a purelin ve výstupní vrstvě. Učící algoritmus nastavte na traingd.
- Prohlédněte si vytvořenou síť. Jak zjistit hodnoty vah a prahů?

Cvičení - BP-sítě v Matlabu

User Guide → Backpropagation

- Váhy a prahy
  - net.IW{1,1} ... váhy hran mezi vstupní a první skrytou vrstvou... L1 X n
    net.IW{1,1}(j,i) ... váha ze vstupního neuronu i do skrytého neuronu j
  - net.LW{L,K} ... váhy hran ze skryté vrstvy K do vrstvy L
    net.LW{L,K}(j,i) ... váha z neuronu i ve vrstvě K do neuronu j ve vrstvě L
  - net.b ... prahy neuronů
    net.b{L,1} ... prahy neuronů ve vrstvě L (počínaje první skrytou)
    net.b{L,1}(j,1) ... práh j-tého neuronu ve vrstvě L

Cvičení - BP-sítě v Matlabu

User Guide → Backpropagation

- Nastavení parametrů funkce traingd ... net.trainParam
  - net.trainParam.lr ... parametr učení
  - Další parametry a jejich defaultní hodnoty ... podívejte se do Nápovědy Matlabu
- Inicializace sítě
  - Síť je po vytvoření automaticky inicializovaná
  - Mohu ji inicializovat znova použitím:
    net = init(net);
  - Použitá inicializační funkce (mohu nastavit jinou):
    net.initFcn
    net.layers{i}.initFcn

Cvičení - BP-sítě v Matlabu

User Guide → Backpropagation

- Učení sítě
  $[net1, tr]$ = train(net,p,t);
  % tr ... training report
  - Množina vzorů se automaticky rozdělí na trénovací, validační a testovací množinu (implicitně v poměru $60\%, 20\%, 20\%$)
  - Použitá rozdělovací funkce (mohu nastavit jinou):
    net.divideFcn
    net.divideParam
  - Použitá chybová funkce (mohu nastavit jinou):
    net.performFcn
- Výstup (odezva) sítě
  y = sim(net1,p);

Cvičení - BP-sítě v Matlabu

Učení se ukončí v jednom z následujících případů

- Byl dosažen maximální počet cyklů (epoch). net.trainParam.epochs (implicitně 1000)
- Byla dosažena chyba na trénovací množině menší než net.trainParam.goal (implicitně 0)
- Gradient klesl pod net.trainParam.min_grad
- Čas učení překročil net.trainParam.time (implicitně Inf)
- Chyba na validační množině vzrostla v net.trainParam.max_fail po sobě jdoucích cyklech (implicitně 6)

Cvičení - BP-sítě v Matlabu

Automaticky přednastavená transformace dat

- Pro vstupní vzory:
  net.inputs{1}.processFcns =
  {'fixunknowns','removeconstantrows','mapminmax'}
  nahradí se chybějící hodnoty vstupů (NaN), pak se vynechají konstantní řádky, pak se provede minmax normalizace
- Pro výstupní vzory:
  net.outputs{2}.processFcns =
  {'removeconstantrows','mapminmax'}
  vynechají se konstantní řádky, provede se minmax normalizace
- Mohu nastavit, jak chci (Nápověda ... Processing Functions)

Cvičení - BP-sítě v Matlabu

Příklad 1 - pokračování

- Nastavte u vytvořené sítě maximální počet cyklů na 1500. Nastavte maximální počet zhoršení na validační množině na 10.
- Zrušte automatickou transformaci vstupních a výstupních dat.
- Síť naučte se zapnutým grafickým výstupem a prohlédněte si všechny zobrazené grafy. Rozumíte jim?
- Z jakého důvodu se učení zastavilo? Jaká je výsledná chyba na trénovací, validační a testovací množině?
- Zobrazte výstupy sítě pro vstupní vzory do grafu. Naučila se síť dobře?
- Zkuste skript pustit několikrát. Liší se výsledky?
- Výsledný skript mi pošlete na email.

Cvičení – K-násobná křížová validace

Příklad 1 - dokončení

- Napište funkci v = CrossVal(), která porovná pro tuto úlohu sítě se 2, 3 a 5 neurony metodou k-násobné křížové validace (pro k=10).
- Funkce vrátí matici, kde každý řádek bude odpovídat jednomu z modelů a ve sloupcích budou hodnoty:
  v(1,:) = průměr (mean) a směrodatná odchylka (std) výsledné chyby na trénovací a testovací množině (mse)
  v(2,:) = totéž pro druhý model
  v(3,:) = totéž pro třetí model
- Doporučuji pro urychlení vypnout při učení grafický výstup (trainParam.showWindow)
- Jak dopadlo srovnání?
- Výsledný skript mi pošlete na email.