Vícevrstvý perceptron a RBF sítě

Michal Krákora, Michal Sojka

5. června 2001

Obsah

1 Vícevrstvý perceptron

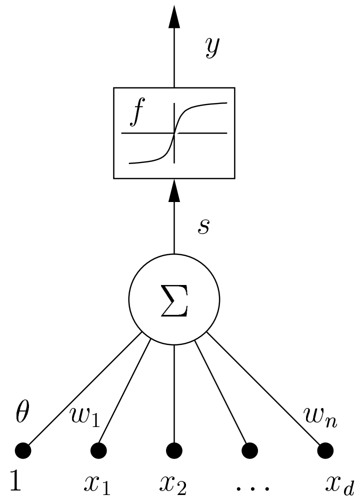
Jednovrstvé neuronové sítě (perceptrony) jsou schopny řešit pouze omezený okruh problémů. Vícevrtsvé sítě některá jejích omezení odstraňují. V této kapitole se zaměříme na popis základních pricipů vícevrstvých perceptronů a na způsob jejich učení.

V roce 1969 Minsky a Papert ukázali, že dvouvrstvá síť je schopná překonat omezení jednovrstvých perceptronů, ale nenašli způsob jak upravovat váhy skrytých neuronů na vstupech. Odpověď na tuto otázku byla publikována až roku 1986 vědci Rumelhartem, Hintonem a Wiliamsem. Hlavní myšlenkou jejich řešení je nahrazení nespojité prahovací funkce funkcí spojitou (diferencovatelnou), což umožňuje použití gradientní metody optimalizace při učení.

Pro výpočet hodnot vah se používá *metoda zpětného šíření*, která může být chápána jako zobecnění delta pravidla používaného u perceptronu sítích (viz [1], kapitola 3.4)

1.1 Struktura vícevrstvé perceptronové sítě

Základní stavební jednotkou vícevrstvých percepronových sítí (multi-layer perceptron, MLP) je perceptron (neuron) zobrazený na obrázku 1.



Obrázek 1: struktura perceptronu

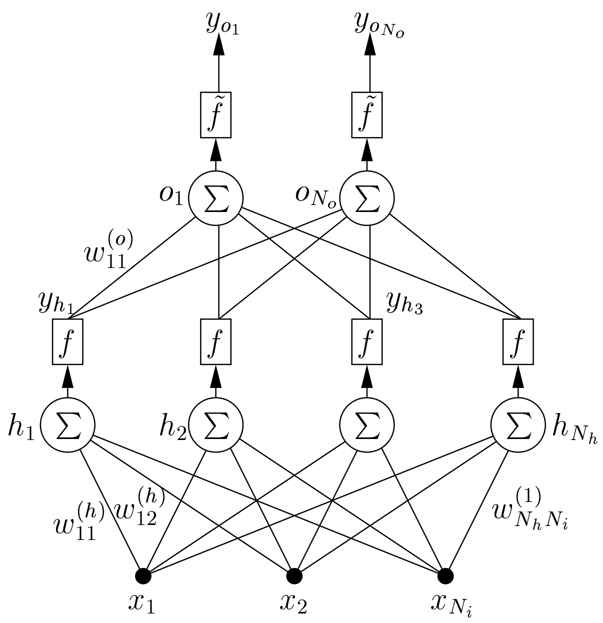
Tento perceptron transformuje svůj vstup $x_1, \ldots, x_d$ na výstup y dle vzorce

$$y = f\left(\sum_{i=1}^{d} w_i x_i + \theta\right) = f\left(\sum_{i=0}^{d} w_i x_i\right) = f(s) \tag{1}$$

kde se θ nahradí vahou w_0 na vstupu $x_0 = 1$. Funkce f se nazývá *aktivační funkce* a budeme se jí podrobněji zabývat dále.

Vícevrstvé sítě můžeme podle jejich struktury rozdělit do několika kategorií. My se zde budeme věnovat pouze tzv. *sítím s dopředným šířením*. Kromě nich existují také například *rekurentní* sítě, ve kterých jsou výstupy sítě přivedeny zpět na vstup.

Sítě s dopřeným šířením mají vrstvenou strukturu. Každá vrstva se skládá z jednotek (neuronů), jejichž vstupy x jsou spojeny buď se vstupy celé sítě nebo s výstupy předchozí vrstvy. Výstupy směřují pouze do následující vrstvy. Příklad často používané dvouvrstvé sítě je na obrázku 2.



Obrázek 2: Struktura vícevrstvé sítě

V každé vrstvě jsou propojeny všechny vstupy každého neuronu se všemi výstupy předchozí vrstvy. Poslední vrstva se nazývá *výstupní* (output). Ostatním vrstvám říkáme *skryté* (hidden). To proto, že výstup neovlivňují přímo, ale pouze přes další neurony.

Takovouto síť můžeme chápat jako strukturu realizující funkci

$$y_k(\mathbf{x}) = \tilde{f}\left(\sum_{j=1}^{N_h} w_{kj}^{(o)} f\left(\sum_{i=0}^{N_i} w_{ji}^{(h)} x_i\right)\right) \tag{2}$$

kde:

N_i	je dimenze příznakového prostoru
N_h	je počet neuronů ve skryté vrstvě
N_o	je počet výstupů
$f, \tilde{f}$	jsou nelineární aktivační funkce (mohou být stejné)

Kromě nejrozšířenějších dvouvrstvých sítí se používají i vícevrstvé sítě (například čtyřvrstvé) a to i přesto, že bylo dokázáno, že i sítí s jednou skrytou vrstvou a nelineární aktivační funkcí lze s libovolnou přesností aproximovat jakoukoliv funkci se spočetně mnoha nespojitostmi.

1.2 Aktivační funkce

V jednovrstvém perceptronu je jako aktivační funkce použita funkce $f = \mathrm{sign}(x)$. Funkce signum není diferencovatelná a proto se u vícevrstvývh sítí používají jiné funkce, které se jí blíží svým průběhem, ale diferencovatelné jsou. Nejčastšji používanými funkcemi jsou sigmoida

$$f(s) = \frac{1}{1 + e^{-s}} \tag{3}$$

hyperbolický tangens

$$f(s) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

nebo lineární omezená fce.

1.3 Učení vícevrstvých sítí

Učením vícevrstvé neuronové sítě rozumíme proces, při němž se snažíme nalézt váhy w tak, abychom minimalizovali chybu na trénovací množině.

Vzhledem k diferenciovatelnosti aktivačních funkcí můžeme pro učení použít *gradientní metodu optimalizace*. Ta je založena na výpočtu gradientu ztrátové funkce a následném upravení všech vah ve směru gradientu. Úpravy se provádí pro každý vzorek z trénovací množiny tak dlouho dokud nenalezneme přijatelné řešení.

1.4 Metoda zpětného šíření (back-propagation)

Mějme na počátku vícevrstvou neuronovou sít' s náhodně inicializovanými vahami w_{ki}. Na vstup sítě budeme postupně přikládat jednotlivé vzorky z trénovací množiny a budeme měnit váhy. Po přiložení p-tého vzorku změníme váhu $w'_{ki} = w_{ki} + \Delta_p w_{ki}$. V následujícím textu ukážeme jak spočítat $\Delta_p w_{ki}$

Připomeňme, že je-li na vstupu sítě p-tý vektor z trénovací množiny, pak je aktivita (výstup) k-tého neuronu dána

$$y_k^p = f(s_k^p), \tag{4}$$

kde

$$s_k^p = \sum_j w_{kj} y_j^p + \theta_k. \tag{5}$$

Nejprve je nutné definovat ztrátovou funkci

$$E = \sum_{p=1}^{l} E^p, \tag{6}$$

kde l je délka trénovací množiny a $E^p = E^p(y_1, \ldots, y_c)$ je funkce výstupů sítě. Typicky se zavádí

$$E^p = \frac{1}{2} \sum_{o=1}^{N_o} (K_o^p - y_o^p)^2. \tag{7}$$

V tomto vztahu je K_o^p žádaný výstup o-tého neuronu ve výstupní vrstvě[1] při p-tém vzorku na vstupu.

Hodnotu o kterou budme při učení měnit váhy budeme hledat jako

$$\Delta_p w_{kj} = -\gamma \frac{\partial E^p}{\partial w_{kj}}, \tag{8}$$

což je vztah velmi podobý delta pravidlu používanému v jednovrstvých sítích (viz [1], kapitola 3.4).

Parciální derivace na pravé straně je složka gradientu ztrátové funkce. Proto se tato metoda řadí mezi *gradientní metody*. Jelikož jde o derivaci složené funkce, bude vztah pro její výpočet vypadat takto:

$$\frac{\partial E^p}{\partial w_{kj}} = \frac{\partial E^p}{\partial s_k^p} \frac{\partial s_k^p}{\partial w_{kj}} \tag{9}$$

Ze vztahu (5) vyplývá, že druhý člen tohoto součinu je

$$\frac{\partial s_k^p}{\partial w_{kj}} = y_j^p. \tag{10}$$

Definujeme-li

[1] Často je počet výstupních neuronů roven počtu tříd do kterých klasifikujeme a chceme aby pro vzorek patřící do třídy k bylo $y_k = 1$ a $y_i = 0$ pro $i \neq k$. Pak můžeme prohlásit, že $y_j = P(j \mid \mathbf{x})$

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p}, \tag{11}$$

můžeme vztah (8) přepsat na

$$\boxed{\Delta_p w_{kj} = \gamma \delta_k^p y_j^p.} \tag{12}$$

Nyní stačí spočítat δ_k^p pro každý neuron k. Zajímavý výsledek, který dále odvodíme, je, že existuje jednoduchá rekurzivní metoda výpočtu δ, která může být chápána jako zpětné šíření chybového signálu sítí.

Pro výpočet δ_k^p rozepíšeme parciální derivaci (11) na součin dvou členů. Jeden člen představuje změnu chyby jako funkce výstupu neuronu a druhý představuje změnu výstupu neuronu jako funkci jeho vstupu. Tím dostaneme

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p} = -\frac{\partial E^p}{\partial y_k^p}\frac{\partial y_k^p}{\partial s_k^p}. \tag{13}$$

Zde si všimneme, že podle rovnice (4)

$$\frac{\partial y_k^p}{\partial s_k^p} = f'(s_k^p), \tag{14}$$

což není nic jiného, než derivace aktivační funkce f k-tého neuronu v bodě s_k^p. Pro výpočet prvního členu rovnice (13) budeme uvažovat dva případy.

1. Pro neurony ve výstupní vrstvě, kde $k = o$, z definice (7) plyne

   $$\frac{\partial E^p}{\partial y_o^p} = -(K_o^p - y_o^p). \tag{15}$$

   Dosazením tohoto a vztahu (14) do rovnice (13) dostaneme

   $$\boxed{\delta_o^p = (K_o^p - y_o^p)\tilde{f}'(s_o^p)} \tag{16}$$

   pro každý výstupní neuron o.

2. Pro neurony ve skrytých vrstvách ($k = h$) nejsme schopni jednoduše určit vliv neuronu na velikost chyby na výstupu sítě. Nicméně jsme schopni určit měřítko chyby, které může být zapsáno jako funkce vstupů ze skryté do výstupní vrstvy $E^p = E^p(s_1^p, s_2^p, \ldots, s_j^p, \ldots)$. Můžeme tedy psát

$$\begin{aligned}\frac{\partial E^p}{\partial y_h^p} &= \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p}\frac{\partial s_o^p}{\partial y_h^p} = \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p}\frac{\partial}{\partial y_h^p}\sum_{j=1}^{N_h} w_{oj} y_j^p = \\ &= \sum_{o=1}^{N_o} \frac{\partial E^p}{\partial s_o^p} w_{oh} = -\sum_{o=1}^{N_o} \delta_o^p w_{oh}. \end{aligned} \tag{17}$$

Dosazením tohoto výsledku do rovnice (13) dostaneme

$$\boxed{\delta_h^p = f'(s_h^p) \sum_{o=1}^{N_o} \delta_o^p w_{oh}.} \tag{18}$$

Hodnotu δ_h pro h-tý neuron ve skryté vrstvě tedy počítáme z hodnot δ_o všech neuronů do kterých vede vede výstup h-tého neuronu. Tyto hodnoty ještě navíc váženy (násobeny) váhou spoje mezi těmito dvěma neurony.

Na celý proces se můžeme podívat take z druhé strany – od výstupu sítě. Nebudeme-li pro jednoduchost brát v úvahu nelineární aktivační funkce, dospějeme k tomu, že změny vah ve výstupní vrstvě jsou úměrné chybě daného výstupu (viz (12) a (16)).

$$\Delta_p w_{oh} = \delta_o^p y_h^p = (K_o^p - y_o^p) y_h \tag{19}$$

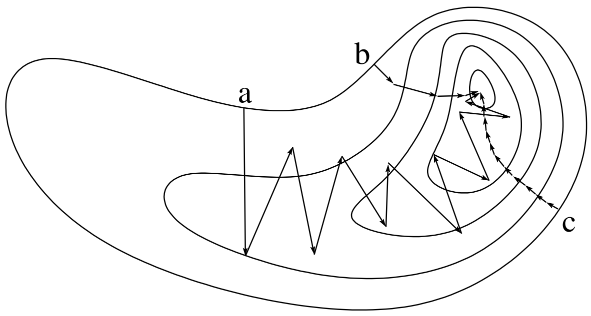
Tato chyba se díky (18) promítne také do neuronů skryté vrstvy. Na tyto neurony se chyba rozpočítá v závislosti na velikosti váhy spoje mezi neurony.

Je tedy vidět, že chyba na výstupu se šíří zpět celou sítí a podle ní jsou upravovány všechny váhy.

1.5 Použití metody zpětného šíření

V předchozím textu jsme odvodili vztahy potřebné pro učení vícevrstvých neuronových sítí metodou zpětného šíření. Nezmínili jsme se ale o tom, jak určit velikost měřítka γ ve vztahu (8). Tato hodnota určuje velikost kroku s jakým se budeme pohybovat prostorem všech vah při hledání minima ztrátové funkce. Pro správné fungování gradientní metody bychom se potřebovali pohybovat po krocích infiniezimální velikosti, ale to bychom se výsledku moc rychle nedobrali. Nejlepší by bylo, aby velikost kroku byla co největší a tím by byl počet kroků potřebný k nalezení minima malý.

Volíme-li γ příliš velké, může docházet k "oscilacím" jako na obrázku 3a. Pro malé γ trvá učení přiliš dlouho (viz 3c).



Obrázek 3: Vliv velikosti kroku na proces učení. a) přiliš velký krok; b) optimální krok; c) příliš malý krok

Nalézt optimální velikost γ jakou znázorňuje obrázek 3b není lehké a proto se používají různé metody jak upravit velikost kroku v závislosti na tom, na jakém místě v prostoru vah se zrovna nacházíme.

Vlastní učení probíhá tak, že přivedeme na vstup příznakový vektor z trénovací množiny a "zapropagujememe" (vypočteme aktivace všech neuronů). Výstupní hodnoty se budou určitým způsobem lišit od žádaných hodnot. Změnou hodnot jednotlivých vah docílíme toho, aby se chyba zmenšila. Gradientní metoda postupuje tak, že zjišťuje jak která váha ovlivňuje výstup.

Jeden krok algoritmu zpětného šíření

1. přivedeme na vstup vektor $\mathbf{x}^p$
2. spočítáme odezvy ve všech bodech sítě
3. vyčíslíme δ_o^p pro výstupní vrstvu podle vzorce (16)
4. vypočítáme δ_h^p pro skryté vrstvy sítě – zpětné šíření, vzorec (18)
5. spočítáme všechny $\Delta_p w_{kj}^p$ podle (12) a upravíme váhy
6. pozorujeme co se děje se ztrátovou funkcí a případně postup opakujeme

Tento krok by se měl provést minimálně jednou, ale spíš několikrát pro všechny vektory z trénovací množiny.

Použití sigmoidy jako aktivační funkce Použití sigmoidy jako aktivační funkce je výhodné díky jednoduchému tvaru její derivace. Derivaci aktivační funkce potřebujeme znát pro výpočet změn vah jak ve výstupní vrstvě (16), tak ve vrstvách skrytých (18).

Sigmoida je funkce

$$y^p = f(s^p) = \frac{1}{1+e^{-s^p}} \tag{20}$$

a její derivace je rovna

$$\begin{aligned} f'(s^p) &= \frac{\partial}{\partial s^P}\frac{1}{1+e^{-s^p}} = -\frac{1}{(1+e^{-s^p})^2}(-e^{-s^p}) \\ &= \frac{1}{1+e^{-s^p}}\frac{e^{-s^p}}{1+e^{-s^p}} = \\ &= \boxed{y^p(1-y^p)} \end{aligned} \tag{21}$$

1.6 Problémy metody zpětného šíření

Minimalizace střední kvadratické chyby na trénovacích datech Při učení minimalizujeme střední kvadratickou chybu na trénovacích datech a ne na skutečných testovacích datech (nemáme je k dispozici). Můžeme ale zajistit aby horní odhad chyby na testovacích datech byl malý.

Můžeme to přirovnat k tomu jako bychom si při řešení otázky jakou tramvají se dostanu nejrchleji do cíle vybírali tramvaje podle čalounění.

Lokální minima Gradientní metody konvergují do lokálního minima. Fungují tak, že jdou ve směru největšího gradientu. Chybová funkce je plná lokálních minim a maxim. Síť pak může "uvíznout" v lokálním minimu i když je v blízkosti minimum mnohem "hlubší".

Pro zabránění uvíznutí v lokálním minimu se používají různé metody. Gradientní sestup z jedné náhodné polohy se nazývá *epocha*. Použijeme-li při učení více epoch, vybereme na konci tu nejlepší. Musíme si ale uvědomit, že má-li neuronová síť deset neuronů ve skryté vrstvě a deset neuronů ve výstupní vrstvě, znamená to, že gradientní metoda pracuje v prostoru dimenze 100. Počáteční body epoch tedy můžeme volit mnoha způsoby a do globálního minima se stejně nemusíme dostat.

Simplexová metoda pracuje tak, že si pamatuje hodnoty vah ve vrcholech nejjednodušího mnohostěnu v prostoru dané dimenze a posouvá ten vrchol, u kterého je největší ztrátová funkce.

Metoda *konjgovaného gradientu* ([4] strana 91) je založena na tom, že v určitém místě sestupu udělá krok ve směru konjugovaného gradientu, což je směr velmi blízký kolmici na gradient. Hodnota ztrátové funkce se tím téměř nezmění a můžeme se dostat do oblasti kde nedojde k uvíznutí v lokálním minimu.

Ochrnutí sítě (network paralysis). V průběhu učení mohou jednotlivé váhy nabývat velikých hodnot. V důsledku toho může i lineární kombinace (5) dosahovat velkých hodnot (kladných a i záporných) a při použití sigmoidy jako aktivační funkce bude aktivace neuronu y_k^p blízká nule nebo jedničce. Jak vyplývá z rovnice (21), je úprava vah úměrná $y_k^p(1 - y_k^p)$ což bude číslo velmi velmi blízké nule, a tak dojde téměř k zastavení učení.

Nemusí skončit Při použití gradientní metody nemusí dojít ke konvergenci algoritmu. Záleží to na volbě kroku gradientní metody (γ). Může se stát, že krok bude příliš velký a minimum vždy překročíme. Dojde k oscilacím okolo minima a hodnota ztrátové funkce bude téměř konstantní.

Příklad: Při kroku 1 km snadno překročíme i vrchol Mt. Everestu a je velmi malá pravděpodobnost, že se trefíme přímo na vrchol.

Proto se v praxi používatjí metody, které si například pamatují k posledních kroků a hodnoty ztrátové funkce v těchto bodech prokládají vhodnou funkci u které lze snadno analyticky nalézt minimum.

1.7 Přesnost vícevrstvých perceptronů

Chceme-li použít vícevrstvý perceptron ke klasifikaci, bude nás určitě zajímat, jak velká bude chyba na testovacích datech. Na velikost této chyby bude mít vliv několik faktorů.

1. Použitý algoritmus učení a počet iterací. Toto nám určuje, jak dobře je síť naučená – jaká je chyba na trénovacích datech.

2. Délka trénovací množiny. Ta určuje, jak přesně reprezentují vzorky z ténovací množiny hledanou funkci.

3. Počet neuronů ve skryté vrstvě. Ten nám udává “vyjadřovací schopnost” sítě. Zatímco u hladkých funkcí si vystačíme jen s několika málo neurony, pro složité funkce s mnoha zvlněními jich budeme potřebovat daleko víc.

Abychom se dále mohli bavit o velikosti chyb, musíme si definovat měřítka chyb. Všechny algoritmy pro učení neuronových sítí se snaží minimalizovat chybu na trénovací množině. Průměrná chyba na jeden vzorek je definována jako

$$E_{learn} = \frac{1}{l} \sum_{p=1}^{l} E^p, \qquad (22)$$

kde l je délka trénovací množiny a E^p je rozdíl mezi žádaným a skutečným výstupem (viz (7)). E_{learn} je měřitelná během učení sítě.

Je zřejmé, že chyba na testovacích datech může být v místech mimo body z trénovací množiny jiná. Chceme-li získat přesnější měřítko chyby, museli bychom integrovat rozdíl mezi žádaným a skutečným výstupem přes celý definiční obor sítě. Tento integrál můžeme odhadnout, máme-li velkou *testovací množinu*.

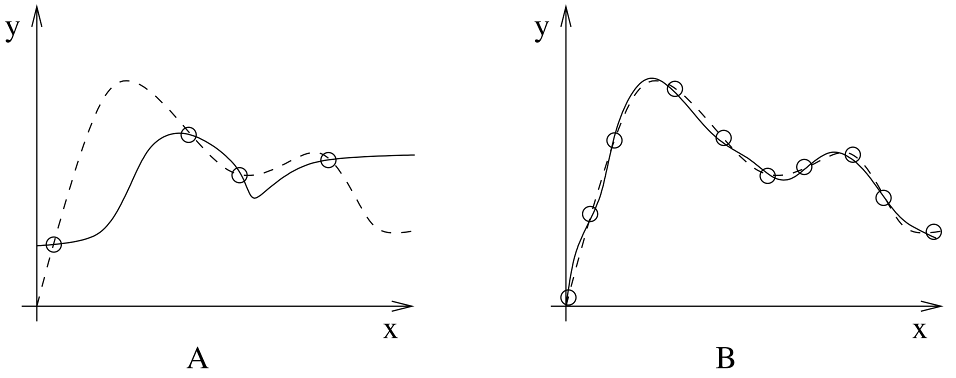
$$E_{test} = \frac{1}{l_{test}} \sum_{p=1}^{l_{test}} E^p \tag{23}$$

V dalších odstavcích se budeme zabývat tím, jak tato měřítka chyb souvisí s délkou trénovací množiny a počtem neuronů ve skryté vrstvě.

1.7.1 Vliv délky trénovací množiny

Vliv délky trénovací množiny budeme demonstovat na jednoduchém příkldu. Pomocí neuronové sítě s pěti neurony ve skryté vrstvě budeme aproximovat funkci $y = f(x)$. Učení se zastavíme v okamžiku kdy se chyba na trénovací množině přestane zmenšovat. Máme-li pro učení k dispozici jen málo vzorků (řekněme 4) je situace podobná jako na obrázku 4a. Vidíme, že E_{learn} je téměř nulová a E_{test} je obrovská.

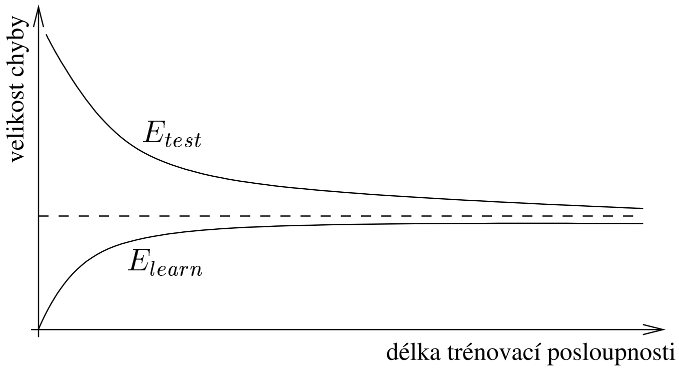
Zvětšíme-li počet vzorků na 10 (obrázek 4b), E_{learn} je větší než v předchozím příkladu, ale E_{learn} je mnohem menší.



Obrázek 4: Vliv délky trénovací množiny. Čárkovaná čára znázorňuje žádanou funkci, prvky trénovací množiny jsou znázorněny kolečky. Jak je funkce aproximována sítí ukazuje plná čára. a) 4 trénovací vzorky. b) 10 trénovacích vzorků.

Kdybychom předchozí pokus opakovali několikrát pro různé délky trénovací množiny a vynášeli bychom průměrnou chybu v závislosti na délce trénovací množiny, dospěli bychom k podobné závislosti jako na obrázku 5.

Vidíme zde, že chyba na trénovací množině se s rostoucím počtem vzorků zvětšuje a chyba na testovacích datech se naopak zmenšuje. Obě dvě chyby konvergují ke stejné hodnotě, která závisí na "vyjadřovací schopnosti" sítě.

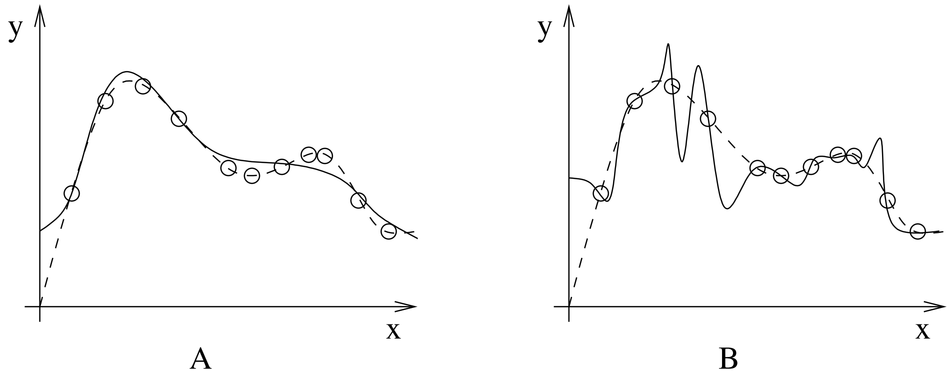


Obrázek 5: Závislost velikosti chyb na délce trénovací množiny.

1.7.2 Vliv počtu neuronů ve skryté vrstvě

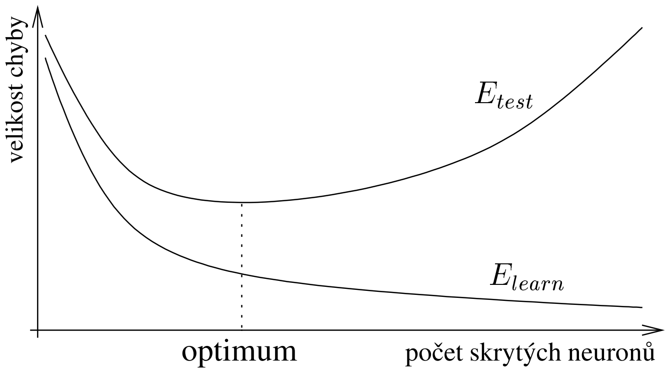
Podobný pokus jako v předchozím odstavci můžeme udělat i nyní, kdy zkoumáme vliv počtu neuronů ve skryté vrstvě na vlastnosti sítě. Při stálé délce trénovací množiny budeme měnit počet skrytých neuronů a budeme pozorovat jak se síť naučila. Výsledky jsou ukázány na obrázku 6. Dáme-li do skryté vrstvy velý počet neuronů (obr. 6b), dochází k jevu zvanému *přeučení*.

V praxi nebudeme znát hledanou funkci nikdy přesně – prvky trénovací množiny budou zatíženy šumem. Bude-li ve skryté vrstvě příliš mnoho neuronů, síť se bude snažit "naučit se i šum", což asi není to co bychom potřebovali.



Obrázek 6: Vliv počtu neuronů ve skryté vrstvě na použitelnost sítě. Čárkovaně je nakreslena žádaná funkce, plná čára znázorňuje aproximaci funkce sítí. Kroužky označují prvky trénovací množiny. a) 5 skrytých neuronů. b) 20 skrytých neuronů.

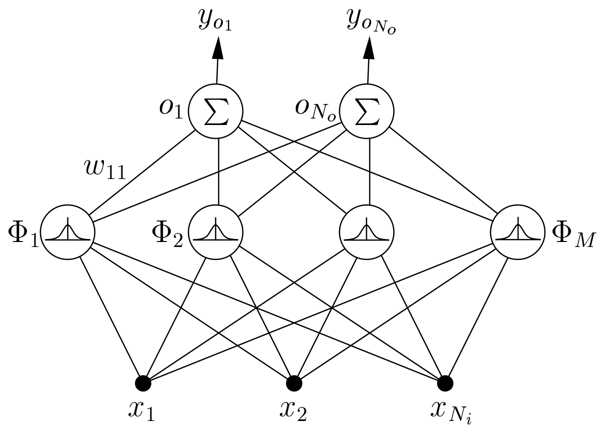
Jak se mění měřítka chyb v závislosti na počtu neuronů ve skryté vrstvě ukazuje obrázek 7. Je vidět, že optimální volba počtu skrytých neuronů je ta, kde je nejmenší chyba na testovacích datech.



Obrázek 7: Průměrná chyba v závislosti na počtu neuronů ve skryté vrstvě.

2 RBF sítě

RBF sítě (Radial Basis Function Networks) jsou sítě, které mají podobnou strukturu jako vícevrstvý perceptron. Narozdíl od něho mají ve skryté vrstvě tzv. kruhové bázové funkce a výstup sítě je většinou pouze lineární kombinací bázových funkcí. Počet bázových funkcí $M \ll l$ (délka trénovací množiny). Příklad RBF sítě je na obrázku 8.



Obrázek 8: RBF síť

2.1 Kruhové funkce

Kruhové funkce jsou funkce, jejichž hodota monoliticky klesá nebo stoupá v závislosti na vzdálenosti od středu bázové funkce.

Typickou a v RBF sítích často požívanou kruhovou funkcí je *gaussián*, který má tvar

$$\Phi(\mathbf{x}) = \exp\left(-\frac{\| \mathbf{x} - \mu \|^2}{2\sigma^2}\right). \tag{24}$$

Jiná kruhová funkce je například lomená funkce

$$\Phi(\mathbf{x}) = \frac{1}{\| \mathbf{x} - \mu \|}. \tag{25}$$

2.2 Vztah RBF sítí k interpolaci

Motivace vzniku RBF je dobře vidět na problému interpolace. Máme-li funkci $h(\mathbf{x})$ u které známe její hodnoty pouze v bodech $\mathbf{x}_i, \quad i = 1, 2, \ldots, l$

$$h(\mathbf{x}_i) = k_i, \tag{26}$$

můžeme hodnotu funkce $h(\mathbf{x})$ v libovolném bodě $\mathbf{x}$ hledat ve tvaru

$$h(\mathbf{x}) = \sum_{i=1}^{l} w_i \Phi(\| \mathbf{x} - \mathbf{x}_i \|), \tag{27}$$

kde $\Phi(x)$ je kruhová funkce se středem v počátku.

Zároveň chceme, aby platil vztah (26) pro všechny body $\mathbf{x}_i$. Tuto podmínku můžeme zapsat jako

$$k_i = \sum_{i=1}^{l} w_i \Phi(\| \mathbf{x} - \mathbf{x}_i \|)$$

nebo maticově

$$\begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_l \end{bmatrix} = \begin{bmatrix} \Phi(\| 0 \|) & \Phi(\| x_1 - x_2 \|) & \ldots \\ \vdots & & \\ \vdots & & \\ \Phi(\| x_1 - x_n \|) & \ldots & \Phi(\| 0 \|) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_l \end{bmatrix}, \tag{28}$$

Hodnoty koeficientů w_i pak nalezneme jako řešení této rovnice

$$\mathbf{w} = \mathbf{\Phi}^{-1}\mathbf{k}, \tag{29}$$

kde $\mathbf{\Phi}^{-1}$ je inverze matice z (28).

Při učení RBF nám nejde o nic jiného, než o interpolaci neznáme funkce u které známe její hodnoty jen pro body z trénovací množiny.

2.3 Učení RBF

Probíhá ve dvou fázích.

1. Nejdříve musíme nalézt parametry bázovývh funkcí v první vrstvě. K tomu použijeme některý z algoritmů shlukování jako například k-means, EM algoritmus nebo editaci k-NN.

2. Poté budeme hledat hodnoty vah neuronů ve výstupní vrstvě. Přímé analytické řešení se dá nalézt *metodou nejmenších čtverců*. Další použitelné metody jsou perceptronový algoritmus nebo gradientní metody.

   Metoda nejmenších čtverců hledá minimum kritéria J_q a bod ve kterém se toto minimum nachází jsou naše hledané váhy.

   $$\mathbf{W} = \arg(\min J_q(\mathbf{W})) \tag{30}$$

   Matice hledaných vah $\mathbf{W}$ je rozměru $N_o \times M$. Trénovací množina je množina dvojic $(\mathbf{x}_i, \mathbf{k}_i)$ délky l, kde $\mathbf{x}_i$ je N_i-rozměrný vektor a $\mathbf{k}_i$ N_o-rozměrný binární vektor tvaru $\mathbf{k}_i = (0, \ldots, 1, 0, \ldots)$ – žádana hodota výstupu. Označíme si hodnoty j-té bázové funkce při i-tém vzorku z trénovací množiny jako

   $$y_{ij}^{(H)} = \Phi_j(\mathbf{x}_i). \tag{31}$$

   Vektor $\mathbf{y}_i^{(H)} = (y_{i1}^{(H)}, y_{i2}^{(H)}, \ldots, y_{iM}^{(H)})$ bude označovat výstup celé skryté vrstvy při i-tém vzorku.

   Naše kritérium, které chceme minimalizovat je podobné jako (6) u vícevrstvého perceptronu.

   $$J_q(\mathbf{W}) = \sum_{i=1}^{l} (\mathbf{y}(\mathbf{x}_i) - \mathbf{k}_i)^2 = \sum_{i=1}^{l} \left(\sum_{j=1}^{M} w_{kj} y_{ij}^{(H)} - \mathbf{k}_i \right)^2 \tag{32}$$

   Toto kritérium bude nabývat svého minima pokud bude závorka na pravé straně minimální. Řešení minimalizace najdeme z následující rovnice

   $$\left[\begin{array}{ccc} \mathbf{k}_1 & \ldots & \mathbf{k}_l \end{array}\right] = \mathbf{W} \left[\begin{array}{cccc} \mathbf{y}_1^{(H)} & \mathbf{y}_2^{(H)} & \ldots & \mathbf{y}_l^{(H)} \end{array}\right], \tag{33}$$

   nebo zkráceně

   $$\mathbf{K} = \mathbf{W}\mathbf{Y}^{(H)}.$$

Tato rovnice je velmi podobná (28). Nemužeme ji ale vyřešit přesně, neboť matice $\mathbf{Y}^{(H)}$ není čtvercová (délka trénovací množiny je větší než počet výstupů). Řešení, které odpovídá minimálnímu kritériu J_q nalezneme pomocí *pseudoinverzní matice* $Y^{(H)^+}$.

$$\boxed{\mathbf{W} = \mathbf{K}\mathbf{Y}^{(H)^+}} \tag{34}$$

Matici $Y^{(H)^+}$ spočteme následovně

$$\mathbf{Y}^+ = (\mathbf{Y}^T\mathbf{Y})^{-1}\mathbf{Y}^T \tag{35}$$

Reference

[1] Ben Kröse, Patrick van der Smagt. *An itroduction to neural networks*, eight edition, November 1996

[2] R. M. Hristev. *The ANN Book*, edition 1

[3] Mark J. L. Orr. *Introduction to Radial Basis Function Networks*

[4] D. Michie, D. J. Spiegelhalter, C. C. Taylor. *Machine Learning, Neurual and Statistical Classification*, February 17, 1994