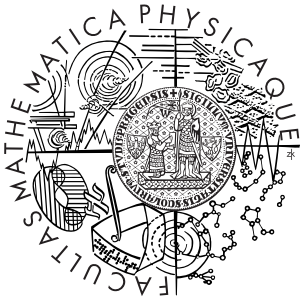
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Diana Fašungová

Neuronové sítě a statistika

Katedra pravděpodobnosti a matematické statistiky

Vedoucí bakalářské práce: Mgr. Peter Bubelíny
Studijní program: Matematika, Obecná matematika

2010

Na tomto mieste by som sa rada poďakovala vedúcemu mojej bakalárskej práce Mgr. Petrovi Bubelínymu za jeho rady, odborné vedenie a čas, ktorý mi venoval.

Prohlašuji, že jsem svou bakalářskou práci napsala samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 6. 8. 2010 Diana Fašungová

Obsah

Název práce: Neuronové sítě a statiska
Autor: Diana Fašungová
Katedra: Katedra pravděpodobnosti a matematické statistky
Vedoucí bakalářské práce: Mgr. Peter Bubelíny
e-mail vedoucího: bubeliny@karlin.mff.cuni.cz

Abstrakt: V predloženej práci študujeme dopredné neurónové siete a ich využitie v štatistike. Zavedieme základné pojmy z prostredia neurónových sietí. Uvedieme vybrané algoritmy trénovania sietí, predovšetkým metódu najmenších štvorcov pre sumu štvorcov chybovej funkcie pre jednovrstvové, algoritmus spätného šírenia chyby, gradientovú metódu s momentom a Levenberg–Marquardtov algoritmus pre viacvrstvové dopredné siete. Pomocou neurónových sietí vyriešime niekoľko úloh regresie a odhadu funkcie. Výsledky porovnáme s tradičným štatistickým prístupom.

Klíčová slova: dopredné neurónové siete, algoritmus spätného šírenia, regresia.

Title: Neural networks and statistics
Author: Diana Fašungová
Department: Department of Probability and Mathematical Statistics
Supervisor: Mgr. Peter Bubelíny
Supervisor's e-mail address: bubeliny@karlin.mff.cuni.cz

Abstract: In the present work we study neural networks and their application in statistics. We define elementary terms for neural networks. We present chosen algorithms for training of neural networks, especially the least-square technique for sum-of-squares error function for single-layer networks and the error back-propagation algorithm, gradient descent with momentum and the Levenberg-Marquardt algorithm for multi-layer feed-forward networks. We solve multiple regression and function fit problems using neural networks and compare the findings with the traditional statistical approach.

Keywords: feed-forward neural networks, error back-propagation algorithm, regression.

Kapitola 1

Úvod

V posledných desaťročiach sa neurónové siete vďaka svojej schopnosti paralelne spracovávať informácie, učiť sa a efektívne usporadúvať dáta začali používať vo fyzike, psychológií, informatike, biológií a ďalčích vedných odboroch. Väčšinou sa vyžívajú na problémy klasifikácie dát do tried a rozoznávania vzorov. Dodnes ale neexistuje jednotná odpoveď na otázku, či sú neurónové siete lepšie ako tradičný prístup.

Model zjednodušenej neurónovej siete bol prvý krát predstavený v roku 1943 McCullochom a Pittsom. Reakciou bola vlna záujmu zo strany vedeckej obce, ktorá ale upadla po publikácií knihy Perceptrons v roku 1969. V nej Minsky a Papert poukázali na nedostatky neurónových sietí akým je napríklad neschopnosť riešiť XOR problém. Pozornosť sa k nim vrátila až v osemdesiatych rokoch po objavení algoritmu spätného šírenia chyby a iných teoretických poznatkov.

Cieľom tejto práce je zaviesť základné pojmy z prostredia neurónových sietí, popísať štruktúru jedno- a viacvrstvových dopredných sietí, uviesť významné učebné algoritmy, aplikovať tieto poznaky na riešenie úloh regresie a odhadu funkcie a porovnať ich s tradičným štatistickým prístupom.

Táto práca čerpá z kníh Neural Networks for Pattern Recognition [1] a An Introduction to Neural Networks [2]. V kapitole 2 predstavíme model neurónovej siete a zadefinujeme základnú terminológiu. V kapitole 3 sa zaoberáme jednovrstvovými sieťami, ich schopnosťou pôsobiť ako lineárne diskriminačné funkcie a učebnými algoritmami: metódou najmenších štvorcov a gradientovou metódou. V kapitole 4 sa venujeme viacvrstvovým dopredným sieťam. Opíšeme ich štruktúru a ukážeme viaceré učebné algoritmy, menovite algoritmus spätného šírenia chyby, gradientovú

metódu s momentom a Levenberg–Marquardtov algoritmus. Ku koncu kapitoly zhodnotíme kvalitu dopredných sietí. V kapitole 5 použijeme dopredné viacvrstové siete na riešenie problému lineárnej regresie a odhadu funkcie. Výsledky vyhodnotíme a porovnáme s tradičným štatistickým prístupom.

Kapitola 2

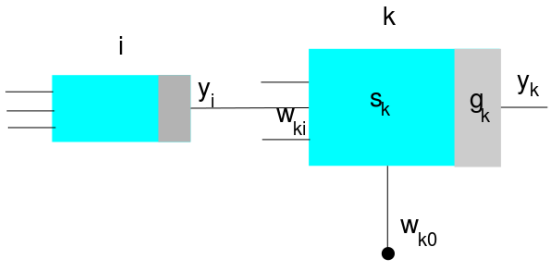
Úvod do neurónových sietí

2.1 Základné pojmy

Neurónové siete sa skladajú z jednoduchých spracovateľských uzlov (neurónov) komunikujúcich medzi sebou pomocou vážených spojení. Každý z týchto spracovateľských uzlov príjme vstupný signál od svojich susedov a vonkajších zdrojov, spracuje ho a vyšle signál ostatným uzlom. Rozlišujeme tri typy spracovateľských uzlov: vstupné, skryté a výstupné. Vstupné uzly, ktoré označíme ako x, získavajú dáta z externých zdrojov. Skryté uzly prijímajú a vysielajú signály len v rámci neurónovej siete. Výstupné uzly určujú výstup neurónovej siete.

Výstup každého uzla nazveme jeho aktiváciou. Hodnota aktivácie uzla k sa určí pomocou aktivačnej funkcie g_k na základe celkového vstupu do uzla s_k. Celkový vstup $s_k = \sum_{i=1}^{n} w_{ki} y_i + w_{k0}$ je vážená suma výstupov od n susedných uzlov a externého vstupu w_{k0}. Teda výstup k-teho uzla je $y_k = g_k(s_k)$. Externý vstup w_{k0} sa nazýva odchýlka. Odchýlku si môžeme predstaviť ako ďalší susedný uzol y_0 s konštantnou aktiváciou rovnou jednej a váhou spojenia w_{k0}. Teda po tejto úprave sa celkový vstup môže zapísať v podobe $s_k = \sum_{i=0}^{n} w_{ki} y_i$ (viď Obr. 1).

Štruktúru siete popisujeme vrstvami. Vrstvy delíme do troch kategórií: na vrstvu vstupných, skrytých a výstupných uzlov. Jednotlivé vrstvy sú medzi sebou prepojené váženými spojeniami.



Obr. 1: V uzle k sa z prijatých signálov spočíta celkový vstup s_k, ktorý sa potom spracuje pomocou aktivačnej funkcie g_k a následne sa pošle z uzla výstup y_k.

2.2 Spracovanie dát

Podľa prepojenia spracovateľských uzlov delíme neurónové siete na dve kategórie:*dopredné (feed-forward)* a *rekurentné (recurrent)*. V dopredných sieťach sa dáta pohybujú len v smere vstup-výstup bez toho, aby sa výstup uzla vrátil späť do vstupu uzla z tej istej alebo jednej z predchádzajúcich vrstiev. Rekurentné siete obsahujú spätné väzby. Táto práca sa sústreďuje na dopredné siete.

Aby neurónová sieť na základe vstupných dát dala žiadaný výstup, musí byť správne nastavená. Za predpokladu, že je k dispozícií počiatočná znalosť o riešenej úlohe, sa váhy spojení nastavia priamo. V prípade, že žiadnu takúto znalosť nemáme, tak sa musí sieť učiť z dát, ktoré sa nazývajú *trénovacie dáta* alebo *trénovacia množina*. Ak sa sieť učí pomocou vstupných a k nim prislúchajúcich výstupných vzorov, tak hovoríme o *učení s učiteľom*. *Učenie bez učiteľa* je proces, v ktorom sa výstupný uzol učí rozoznávať štatisticky význačné zoskupenia vstupných dát. Tento proces je úzko spätý s odhadom hustoty rozdelenia.

Oba učiace prístupy majú za úlohu upraviť váhy spojení medzi uzlami. Existuje veľa učebných algoritmov, ale v podstate všetky vychádzajú z myšlienky Hebbovho algoritmu. Ak uzol k dostáva vstupné dáta od uzla j, tak zmena váhy medzi týmito uzlami je $\Delta w_{kj} = \eta y_k y_j$, kde η je kladná konštanta

nazývaná *učebný parameter*. Zvoliť vhodný učebný parameter je dôležité ako ukážeme neskôr.

V učení s trénovacou množinou je niekedy výhodné si dáta predspracovať. Uvažujme príklad, v ktorom chceme sadu ručne napísaných znakov priradiť písmenu ‘a‘ alebo písmenu ‘b‘. Každý z týchto znakov je reprezentovaný poľom hodnôt, ktoré nadobúdajú hodnoty 0 a 1 . Ak je znak načítaný z obrázku o rozlíšení 256×256 pixelov, tak predstavuje jeden bod v priestore o rozmere $d = 65536$. Teda ak by sme chceli uskladniť všetky možné znaky a k nim náležiacu kategóriu, tak by sme potrebovali uložiť $2^d = 2^{65536}$ znakov, čo je nepraktické. Ak teda chceme, aby sieť rozoznala nový vzor, ktorý nemá uložený v databáze, tak je treba problém zovšeobecniť. To sa docieli vyextrahovaním určitej črty vstupných dát, v tomto prípade napríklad pomerom dĺžky ku šírke napísaného znaku. Označme tento pomer ako x_1. Je vidno, že znak ‘b‘ má v priemere vyššie hodnoty x_1 ako znak ‘a‘. Zavedením premennej x_1 sa teda problém zjednodušil.

Kapitola 3

Jednovrstvové siete

Neurónové siete môžu obsahovať viac vrstiev. Vrstva vstupných uzlov sa do celkového počtu vrstiev nepočíta, a preto neurónovú sieť skladajúcu sa len zo vstupných a výstupných uzlov nazývame jednovrstvovou.

Jednovrstvová sieť sa skladá z d vstupných a c výstupných uzlov a z odchýlok w_{k0}, kde $k = 1, ..., c$. Na odchýlku sa pozeráme ako na váhu zo vstupného uzla x_0 s aktiváciou jedna (viď Obr. 2). Výstup siete je daný aktiváciami výstupných neurónov

$$y_k = g(\sum_{i=0}^{d} w_{ki} x_i),$$

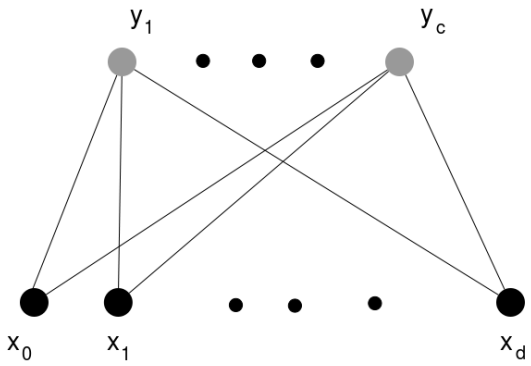
kde $k = 1, ..., c$.

3.1 Lineárne diskriminačné funkcie

Na jednovrstvové neurónové siete sa dá pozerať ako na lineárne diskriminačné funkcie. Tie sa používajú na priradenie pozorovania do vopred známej triedy. Majú tvar lineárnej kombinácie pozorovaného vektora, pričom úlohu koeficientov lineárnej kombinácie plnia váhy w_{kj}. Hoci je málo typov diskriminačných funkcií, ktoré je jednovrstvová sieť schopná namodelovať, majú jednovrstvové siete výhodu nad zložitejšími sieťami. Doba ich učenia je relatívne krátka a požaduje oveľa menšie výpočetné úsilie.

Majme c tried, ktorým máme priradiť vektory z trénovacej množiny $\{\mathbf{x}^1, ..., \mathbf{x}^N\}$. Pre každú triedu použijeme diskriminačnú funkciu v tvare

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0},$$



Obr. 2: Sieť skladajúca sa z d vstupných a c výstupných uzlov a z odchýlok w_{k0}, kde $k = 1, ..., c$.

kde $k = 1, ..., c$, $\mathbf{w}_k = (w_{k1}, ..., w_{kd})^T$ je vektor váh, parameter w_{k0} odchýlka a $\mathbf{x} = (x_1, ..., x_d)^T$ je vstupný vektor. Vektor $\mathbf{x}^n$, $n = 1, ..., N$, zaradíme do triedy k, ak pre všetky $j \neq k$ platí

$$y_k(\mathbf{x}^n) \geq y_j(\mathbf{x}^n).$$

V prípade rovnosti môžeme vektor $\mathbf{x}^n$ zaradiť do triedy k alebo do triedy j.

Takúto funkciu si môžeme predstaviť ako jednovrstvovú neurónovú sieť znázornenú na obrázku 2. Na základe tohoto modelu môžeme diskriminačnú funkciu prepísať do formy

$$y_k(\mathbf{x}) = \sum_{i=0}^{d} w_{ki} x_i.$$

Logistické diskriminačné funkcie Logistické diskriminačné funkcie

$$y_k = g(\mathbf{w}_k^T \mathbf{x} + w_{k0}),$$

kde g je nelineárna monotónna aktivačná funkcia, sa kvôli monotónii g radia k lineárnym diskriminačným funkciám.

Jednou z najpoužívanejších aktivačných funkcií je sigmoidová funkcia

$$g(a) = \frac{1}{1 + e^{-a}}, \tag{3.1}$$

ktorá zohráva dôležitú rolu vo viacvrstvových neurónových sieťach.

Obecné lineárne diskriminačné funkcie Ak povolíme transformáciu vektora $\mathbf{x} = (x_1, ..., x_d)^T$ pomocou množiny nelineárnych funkcií $\{\phi_j(\mathbf{x})\}_{j=1}^M$, získame oveľa väčšiu triedu diskriminačných funkcií v tvare

$$y_k(\mathbf{x}) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0}.$$

Funkcie $\phi_j(\mathbf{x})$ sa nazývajú *bázické funkcie* a predpokladá sa o nich, že sú nemenné, vopred určené a nezávislé na dátach.

Opäť si prestavme odchýlku w_{k0} ako váhu spojenia z $\phi_0(\mathbf{x})$ a diskriminačná funkcia nadobudne formu

$$y_k(\mathbf{x}) = \sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x}). \tag{3.2}$$

Lineárna separabilita Hranica medzi dvomi triedami C_k a C_l je daná rovnicou $y_k(\mathbf{x}) = y_l(\mathbf{x})$. V prípade lineárnej diskriminačnej funkcie sa jedná o nadrovinu v d-dimenzionálnom priestore

$$(\mathbf{w}_k - \mathbf{w}_l)^T\mathbf{x} + (w_{k0} - w_{l0}) = 0.$$

Ak sú všetky vstupné vektory $\mathbf{x}^1, ..., \mathbf{x}^N$, rozdelené hranicou klasifikované správne, tak hovoríme, že vektory sú lineárne separabilné.

V lineárne neseparabilných priestoroch jednovrstvové siete nedokážu klasifikovať všetky vstupné dáta správne. Jednoduchým príkladom je *XOR problém*, v ktorom uvažujeme dvojrozmerný priestor so štyrmi bodmi

$$x_1 \ = \ [0,0], x_2 = [1,0], x_3 = [0,1], x_4 = [1,1].$$

Nech body x_1 a x_3 náležia triede C_1 a zvyšné dva body triede C_2. Je zrejmé, že žiadna priamka neoddelí tieto body tak, aby sa v každom zo vzniknutých polopriestorov nachádzali len body z jednej triedy.

Zo štatistického hľadiska nie je schopnosť siete diagnostikovať všetky vektory z trénovacej množiny správne podstatná, pretože sa hľadá model, ktorý dokáže čo najpresnejšie klasifikovať nové dáta.

3.2 Učenie jednovrstvových sietí

Na určenie optimálnych hodnôt w_{kj} existujú viaceré algoritmy. V tejto podkapitole predstavíme metódu najmenších štvorcov pre sumu štvorcov chybovej funkcie.

Metóda najmenších štvorcov Nech $y_k(\phi(\mathbf{x}^n), \mathbf{w}_k)$ je výstup uzla $k = 1, ..., c$ závislý na vstupnom vektore $\phi(\mathbf{x}^n) = (\phi_1(\mathbf{x}^n), ..., \phi_M(\mathbf{x}^n))^T$ pre $n = 1, ..., N$ a váhovom vektore $\mathbf{w}_k$, kde c je počet výstupov, M je počet bázických funkcií, N je počet vstupov z trénovacej množiny a t_k^n je cieľová hodnota výstupu uzla k pri vstupnom vektore $\phi(\mathbf{x}^n)$. Označme $\mathbf{w} = (w_{10}, ..., w_{1M}, ..., w_{c0}, ..., w_{cM})^T$. Potom chybová funkcia je daná sumou cez všetky vektory v trénovacej množine a cez všetky výstupy rovnosťou

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} \{y_k(\phi(\mathbf{x}^n), \mathbf{w}_k) - t_k^n\}^2.$$

Pretože (3.2) je lineárna funkcia váh, je $E(\mathbf{w})$ kvadratická funkcia váh a z jej hladkosti plynie, že derivácie vzhľadom k váham sú lineárne funkcie. Po dosadení za $y_k(\phi(\mathbf{x}^n), \mathbf{w})$ z rovnice (3.2) dostaneme

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} \{\sum_{j=0}^{M} w_{kj} \phi_j(\mathbf{x}^n) - t_k^n\}^2.$$

Ďalej budeme pre jednoduchosť zápisu značiť $\phi_j(\mathbf{x}^n) = \phi_j^n$. Deriváciou podľa parametru w_{kj} a položením výsledku rovno nule, dostaneme

$$\sum_{n=1}^{N} \{\sum_{j'=0}^{M} w_{kj'} \phi_{j'}^n - t_k^n\} \phi_j^n = 0.$$

Prepísaním do maticovej podoby máme

$$(\mathbf{\Phi}^T \mathbf{\Phi}) \mathbf{W}^T = \mathbf{\Phi}^T \mathbf{T}, \tag{3.3}$$

kde $\mathbf{\Phi}$ je matica (ϕ_j^n) typu $(N \times M)$, $\mathbf{W}$ je matica (w_{kj}) typu $(c \times M)$ a $\mathbf{T}$ je matica (t_k^n) typu $(N \times c)$. Predpokladáme, že štvorcová matica $(\mathbf{\Phi}^T \mathbf{\Phi})$ je regulárna. Jej invertovaním dostaneme riešenie (3.3), ktoré zapíšeme ako

$$\mathbf{W}^T = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{T},$$

V praxi sa môže stať, že $(\mathbf{\Phi}^T \mathbf{\Phi})$ je singulárna. Vtedy sa na nájdenie matice $\mathbf{W}$ použije algoritmus *SVD (singular value decomposition)* známy z lineárnej algebry.

Keď sa počet vstupných dát rovná počtu bázických funkcií $N = M$ a $\mathbf{\Phi}^T$ je štvorcová matica tvorená lineárne nezávislými vektormi, tak z (3.3) dostaneme

$$\mathbf{\Phi} \mathbf{W}^T = \mathbf{T},$$

čo sa ekvivalentne prepíše ako

$$\sum_{j=0}^{M} w_{kj}\phi_j^n = t_k^n.$$

Z toho plynie, že pre každý vstupný vektor $\mathbf{x}^n$ je výstup rovný cieľovej hodnote a chybová funkcia sa bude rovnať nule.
Ak bázické vektory nie sú lineárne nezávislé, teda platí, že počet lineárne nezávislých vstupných vektorov je menší ako M, tak je úloha najmenších štvorcov neriešiteľná. V situácií, keď máme k dispozícií menej vzorov v trénovacej množine ako je bázických funkcií, je úloha opäť neriešiteľná.

Postup popísaný v predchádzajúcom odseku je možno aplikovať len na lineárne diskriminačné funkcie. Ak sa v sieti používa nelineárna diferencovateľná diskriminačná funkcia akou je napríklad sigmoid, tak sa využíva *gradientová metóda*.

Označme $E = E(\mathbf{w})$. Myšlienka gradientovej metódy je veľmi jednoduchá. Najprv si náhodne zvolíme hodnotu vektora váh $\mathbf{w}$ a potom budeme vektor $\mathbf{w}$ posúvať v smere, v ktorom hodnota E najviac klesá, teda v smere $-\nabla_{\mathbf{w}} E$. Iteráciou pomocou vzorca

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \frac{\partial E}{\partial w_{kj}} \mid_{\mathbf{w}^{(\tau)}}$$

dostaneme postupnosť váhových vektorov $\mathbf{w}^{(\tau)}$. Za parameter η sa volí malé kladné číslo a slúži ako učebný parameter. Ako bolo spomenuté už v predošlej kapitole, voľba η je dôležitá. Pre η malé bude algoritmus bežať pomaly a pre η veľké nastanú divergentné oscilácie.

Obecne sa chybová funkcia zadáva ako suma členov $E^n(\mathbf{w})$, ktoré sú rátané pre konkrétny vzor z trénovacej množiny $\mathbf{x}^n$

$$E(\mathbf{w}) = \sum_{n=1}^{N} E^n(\mathbf{w}).$$

V tomto prípade sa postupuje po jednotlivých vzoroch

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \frac{\partial E^n}{\partial w_{kj}}, \tag{3.4}$$

čo umožňuje použitie tohto algoritmu v aplikáciách, kde dáta prichádzajú priebežne. Výhodou je, že po spracovaní sa môžu vymazať.

Deriváciou chybovej funkcie n-tého vzoru podľa príslušnej váhy w_{kj} dostávame

$$\frac{\partial E^n}{\partial w_{kj}} = \{y_k(\mathbf{x}^n) - t_k^n\}\phi_j(\mathbf{x}^n) = \delta_k^n \phi_j^n, \tag{3.5}$$

kde δ_k^n sa definuje ako

$$\delta_k^n = y_k(\mathbf{x}^n) - t_k^n.$$

Využitím rovníc (3.4) a (3.5) dostaneme vyjadrenie zmeny vo váhe w_{kj}

$$\triangle w_{kj} = -\eta \delta_k^n \phi_j^n.$$

Tento vzťah sa názyva *metódou najmenších štvorcov*, adalinové pravidlo alebo aj pravidlo delta.

Pre diferencovateľné nelineárne aktivačné funkcie je výstup daný rovnosťou $y_k = g(s_k)$, kde g je aktivačná funkcia a $s_k = \sum_{j=0}^{M} w_{kj}\phi_j$. Pre tento typ funkcií platí

$$\frac{\partial E^n}{\partial w_{kj}} = g^{'}(s_k)\delta_k^n \phi_j^n = g^{'}(\sum_{j=0}^{M} w_{kj}\phi_j^n)(y_k(\mathbf{x}^n) - t_k^n)\phi_j^n.$$

Kapitola 4

Viacvrstvové dopredné siete

Viacvrstvové dopredné siete sú tvorené vrstvami vstupných, skrytých a výstupných uzlov, v ktorých sa výstupné dáta z vrstvy posielajú len uzlom o vrstvu vyššie. Označme k-ty skrytý uzol v i-tej vrstve ako $z_{k,i}$ a k-ty výstupný uzol ako y_k.

Majme L-vrstvovú sieť s počtom skrytých uzlov v každej vrstve M_i, kde $i = 1, ..., L-1$. Celkový vstup do k-teho skrytého uzla v prvej vrstve dostaneme ako váženú sumu vstupných dát x_i, $i = 1, ..., d$, a odchýlky $w_{k0}^{(1)}$

$$s_{k,1} = \sum_{i=0}^{d} w_{ki}^{(1)} x_i,$$

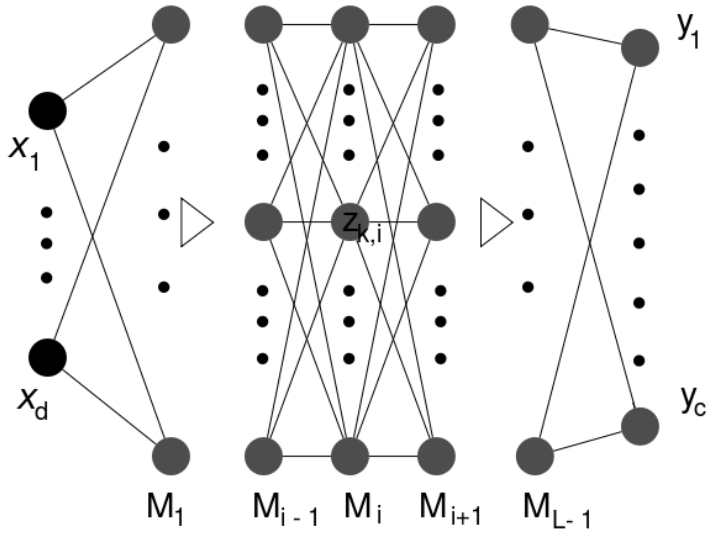
kde sme pre zjednodušenie zápisu uvažovali uzol x_0 s konštantnou aktiváciou rovnou jednej a $w_{ki}^{(1)}$ označujú váhy z i-tého vstupu do k-teho skrytého uzla v prvej vrstve.
Aktiváciu skrytého uzla k v prvej vrstve získame transformáciou celkového vstupu pomocou aktivačnej funkcie g

$$z_{k,1} = g(s_{k,1}).$$

Pre k-ty uzol z i-tej vrstvy, $i = 2, ..., L$, je celkový vstup rovný lineárnej kombinácií výstupov uzlov z predchádzajúcej vrstvy a odchýlky $w_{k0}^{(i)}$

$$s_{k,i} = \sum_{j=0}^{M_{i-1}} w_{kj}^{(i)} z_{j,i-1},$$

kde sme uvažovali uzol $z_{0,i-1}$ s aktiváciou jedna a kde váhy $w_{kj}^{(i)}$ popisujú spojenie z j-teho uzla z $(i-1)$-vej vrstvy do k-teho uzla z i-tej vrstvy.



Obr. 3: Štruktúra L-vrstvovej doprednej siete.

Aktiváciu k-teho výstupného uzla dostaneme užitím aktivačnej funkcie $\overline{g}$ na celkový vstup $s_{k,L}$,

$$y_k = \overline{g}(\sum_{j=0}^{M_{L-1}} w_{kj}^{(L)} z_{j,L-1}).$$

Poznamenajme, že nemusí platiť rovnosť $g = \overline{g}$.

4.1 Algoritmus spätného šírenia chyby

Pri sieťach s nelineárnymi diferencovateľnými aktivačnými funkciami a diferencovateľnou chybovou funkciou sa na nastavenie váh používa *algoritmus spätného šírenia chyby (error back-propagation)*. Derivácie chybovej funkcie zohrávajú významnú úlohu vo väčšine učebných algoritmov pre viacvrstvové siete, a preto nie je požiadavka na diferencovateľnosť nijak obmedzujúca.

Algoritmus sa delí na dve etapy. V prvej sa vypočítajú derivácie chybovej funkcie podľa váh. Práve v tejto etape dochádza k spätnému šíreniu. V druhej fáze sa spočítané derivácie používajú na úpravu váh napríklad pomocou gradientovej metódy, gradientovej metódy s momentom

alebo Levenberg-Marquardtovým algoritmom. Gradientová metóda bola predstavená v kapitole 3, gradientová metóda s momentom a Levenberg-Marquardtov algoritmus budú popísané ďalej v tejto kapitole.

Uvažujme obecnú chybovú funkciu $E = \sum_n E^n$, kde E^n označujú chybové funkcie jednotlivých vstupných vzorov. Pri E^n predpokladáme, že sú diferencovateľnými funkciami výstupov, tj. $E^n = E^n(y_1, ..., y_c)$. Vďaka sumovému tvaru chybovej funkcie po jej zderivovaní podľa váh dostaneme sumu derivácií chybových funkcií cez jednotlivé vzory. Preto sa pri odvodení obmedzíme len na n-tý vzor.

Pre vstupný vektor $\mathbf{x}^n$ vyrátame aktivácie všetkých skrytých a výstupných uzlov. Použijeme na to nasledujúce vzťahy

$$z_{k,i} = g(s_{k,i}), \tag{4.1}$$

kde $z_{k,i}$ značí aktiváciu skrytého uzla k v i-tej vrstve pre $i = 1, ..., L-1$, a

$$y_k = \overline{g}(s_{k,L}), \tag{4.2}$$

kde y_k je aktivácia k-teho výstupného uzla.
Celkový vstup $s_{k,i}$, $i = 1, ..., L$, dostaneme ako

$$s_{k,i} = \sum_{j=0}^{M_{i-1}} w_{kj}^{(i)} z_{j,i-1}, \tag{4.3}$$

kde sme pre zjednodušenie zápisu zadefinovali $M_0 = d$ a $z_{j,0} = x_j$.
Užitím retiazkového pravidla pri derivovaní podľa $w_{kj}^{(i)}$ dostávame

$$\frac{\partial E^n}{\partial w_{kj}^{(i)}} = \frac{\partial E^n}{\partial s_{k,i}} \frac{\partial s_{k,i}}{\partial w_{kj}^{(i)}}. \tag{4.4}$$

Zo vzťahu pre celkový vstup (4.3) máme

$$\frac{\partial s_{k,i}}{\partial w_{kj}^{(i)}} = z_{j,i-1}. \tag{4.5}$$

Zavedieme značenie

$$\delta_{k,i} = \frac{\partial E^n}{\partial s_{k,i}}. \tag{4.6}$$

Dosadením (4.5) a (4.6) do (4.4) získame

$$\frac{\partial E^n}{\partial w_{kj}^{(i)}} = \delta_{k,i} z_{j,i-1}. \tag{4.7}$$

Ďalej vypočítame $\delta_{k,i}$ pre výstupné a skryté uzly. Pre výstupné uzly máme z (4.2) a (4.6)

$$\delta_{k,L} = \frac{\partial E^n}{\partial s_{k,L}} = \overline{g}'(s_{k,L}) \frac{\partial E^n}{\partial y_k}. \tag{4.8}$$

Pri skrytých uzloch nie je postup tak priamočiary.

Vieme, že zmena v celkovom vstupe $s_{k,i}$ skrytého uzla k vo vrstve i, $i = 1, ..., L-1$, ovplyvní zmenu hodnoty chybovej funkcie len prostredníctvom neurónov vo vyššej vrstve, s ktorými má uzol k spojenie. Označme indexom l uzol v $(i+1)$-vej vrstve, ktorý má spojenie s uzlom k. Všimnime si, že l môže byť buď skrytý alebo výstupný. Vyjdeme zo vzťahu (4.6) a použijeme retiazkové pravidlo

$$\delta_{k,i} = \frac{\partial E^n}{\partial s_{k,i}} = \sum_{l=0}^{M_{i+1}} \frac{\partial E^n}{\partial s_{l,i+1}} \frac{\partial s_{l,i+1}}{\partial s_{k,i}}. \tag{4.9}$$

Využitím vzťahov (4.1) a (4.3) a dosadením (4.6) do (4.9) získame *pravidlo spätného šírenia*

$$\delta_{k,i} = g'(s_{k,i}) \sum_{l=0}^{M_{i+1}} w_{lk}^{(i+1)} \delta_{l,i+1}. \tag{4.10}$$

Tento postup zopakujeme pre každý vstupný vzor z trénovacej množiny a vyrátame deriváciu chybovej funkcie

$$\frac{\partial E}{\partial w_{kj}^{(i)}} = \sum_{n=1}^{N} \frac{\partial E^n}{\partial w_{kj}^{(i)}}.$$

Príklad Algoritmus spätného šírenia ilustrujeme na príklade dvojvrstvovej doprednej siete so sigmoidovými aktivačnými funkciami pre skryté a lineárnymi aktivačnými funkciami pre výstupné uzly. Výhodou sigmoidovej funkcie (3.1) je jednoduchosť výpočtu jej derivácie

$$g'(a) = g(a)(1 - g(a)). \tag{4.11}$$

Chybovú funkciu uvažujme v tvare

$$E^n = \frac{1}{2} \sum_{k=1}^{c} (y_k^{(n)} - t_k^{(n)})^2, \tag{4.12}$$

kde $y_k^{(n)}$ je výstup uzla k a $t_k^{(n)}$ jeho cieľová hodnota pre vstupný vektor $\mathbf{x^n}$.

Pomocou (4.8) dostávame pre výstupné uzly z (4.12)

$$\delta_{k,2} = y_k^{(n)} - t_k^{(n)},$$

kde $\overline{g}'(s_{k,2}) = 1$ z linearity. Pre skryté uzly z (4.10) a (4.11)

$$\delta_{k,1} = z_{k,1}(1 - z_{k,1}) \sum_{l=1}^{c} w_{lk}^{(2)} \delta_{l,2}.$$

Dosadením hodnôt $\delta_{k,1}$ a $\delta_{k,2}$ do (4.7) získame

$$\frac{\partial E^n}{\partial w_{kj}^{(1)}} = \delta_{k,1} x_j,$$

$$\frac{\partial E^n}{\partial w_{kj}^{(2)}} = \delta_{k,2} z_{j,1}.$$

V druhej etape algoritmu použijeme napríklad gradientovú metódu, ktorá bola uvedená v kapitole 3. Po zadaní vstupu x_j, $j = 0, ..., d$, do siete zmeníme váhy o $\Delta w_{kj}^{(1)} = -\eta \delta_{k,1} x_j$ v prvej a o $\Delta w_{kj}^{(2)} = -\eta \delta_{k,2} z_{j,1}$ v druhej vrstve.

4.2 Gradientová metóda s momentom

Ako bolo spomenuté v kapitole 3, idea tohto algoritmu je jednoduchá. Inicializujeme váhy náhodnými číslami a následne iterujeme podľa vzťahu

$$\Delta \mathbf{w}^{(\tau)} = -\eta \nabla E^n|_{\mathbf{w}^{(\tau)}}, \tag{4.13}$$

kde $\Delta \mathbf{w}^{(\tau)}$ je zmena váhového vektora v kroku τ. Aby sme sa vyhli divergentným osciláciám pri vysokých hodnotách učebného parametru η a zmenšili dobu trénovania, zavedieme do (4.13) *moment* μ

$$\Delta \mathbf{w}^{(\tau)} = -\eta \nabla E^n|_{\mathbf{w}^{(\tau)}} + \mu \Delta \mathbf{w}^{(\tau-1)}.$$

4.3 Levenberg-Marquardtov algoritmus

Levenberg-Marquardtov algoritmus sa používa špeciálne na chybové funkcie v súčtovom tvare podľa jednotlivých vzorov $E = K \sum_{n=1}^{N} (\epsilon^n)^2 = K \|\epsilon\|^2$, kde

K je vopred zvolená konštanta, chybu n-tého vzoru sme označili ϵ^n a ϵ je vektor tvorený týmito chybami. Pre jednoduchosť zápisu uvažujme chybovú funkciu

$$E = \frac{1}{2}\sum_{n=1}^{N}(\epsilon^n)^2 = \frac{1}{2}\|\epsilon\|^2.$$

Majme vektor váh $\mathbf{w}_{old}$, ktorý zmeníme na $\mathbf{w}_{new}$. Ak je zmena $\mathbf{w}_{new} - \mathbf{w}_{old}$ malá, tak môžeme vektor chýb ϵ aproximovať Taylorovým rozvojom prvého rádu

$$\epsilon(\mathbf{w}_{new}) = \epsilon(\mathbf{w}_{old}) + J(\mathbf{w}_{new} - \mathbf{w}_{old}),$$

kde matica J je jakobián s prvkami $(J)_{ni} = \frac{\partial \epsilon^n}{\partial w_i}$. Pomocou tohto vzťahu prepíšeme chybovú funkciu

$$E = \frac{1}{2}\|\epsilon(\mathbf{w}_{old}) + J(\mathbf{w}_{new} - \mathbf{w}_{old})\|^2.$$

Minimalizáciou E vzhľadom k $\mathbf{w}_{new}$ dostaneme

$$\mathbf{w}_{new} = \mathbf{w}_{old} - (J^T J)^{-1} J^T \epsilon(\mathbf{w}_{old}).$$

Ak by sme na zmenu váh používali tento vzťah, tak hrozí, že zmena bude príliš veľká a už nebude platiť lineárna aproximácia Taylorovým rozvojom. Preto upravíme chybovú funkciu

$$\overline{E} = \frac{1}{2}\|\epsilon(\mathbf{w}_{old}) - J(\mathbf{w}_{new} - \mathbf{w}_{old})\|^2 + \lambda\|\mathbf{w}_{new} - \mathbf{w}_{old}\|^2,$$

kde parameter λ reguluje veľkosť kroku $\mathbf{w}_{new} - \mathbf{w}_{old}$. V praxi sa za počiatočnú hodnotu λ volí zvyčajne 0,1. Ak je λ veľké, tak je $\|\mathbf{w}_{new} - \mathbf{w}_{old}\|^2$ malé. Minimalizujme $\overline{E}$ vzhľadom k $\mathbf{w}_{new}$. Získame

$$\mathbf{w}_{new} = \mathbf{w}_{old} - (J^T J + \lambda I)^{-1} J^T \epsilon(\mathbf{w}_{old}). \tag{4.14}$$

V každom kroku algoritmu pozorujeme zmenu $\overline{E}$ po užití (4.14). Ak $\overline{E}$ klesne, ponecháme si vektor $\mathbf{w}_{new}$, desaťnásobne zmenšíme parameter λ a opakujeme. V prípade, že $\overline{E}$ sa zvýši, tak sa vrátime k $\mathbf{w}_{old}$, desaťnásobne zväčšíme λ a vypočítame nový vektor váh $\mathbf{w}_{new}$. Toto opakujeme, kým sa hodnota $\overline{E}$ nezmenší. Keď sa chybová funkcia $\overline{E}$ prestane zmenšovať, nastane zväčšovanie parametru λ. Keď λ presiahne vopred určenú hodnotu λ_{max}, tak algoritmus skončí.

4.4 Kvalita doprednýchsietí

Výhodou dopredných sietí je schopnosť dvojvrstvovej siete s konečným počtom skrytých uzlov s ľubovoľnou aktivačnou funkciou odhadnúť akúkoľvek spojitú funkciu na kompatnej podmnožine $\mathbf{R^d}$. O aktivačnej funkcii výstupných uzlov predpokladáme, že je lineárna.
Takýto odhad ale nie je absolútne presný. Na aproximačnú chybu vplýva hneď niekoľko faktorov: učebný algoritmus a počet iterácií, veľkosť trénovacích dát a počet skrytých uzlov.

Aby sme mohli posúdiť dopad týchto faktorov, tak musíme zaviesť spôsob merania minimalizácie chyby odhadu. Zaveďme *chybu trénovania* E_{train} ako

$$E_{train} = \frac{1}{N_{train}} \sum_{n=1}^{N_{train}} E^n,$$

kde N_{train} je počet vektorov v trénovacej množine a E^n popisuje rozdiel medzi cieľovou a výstupnou hodnotou pre n-tý vzor

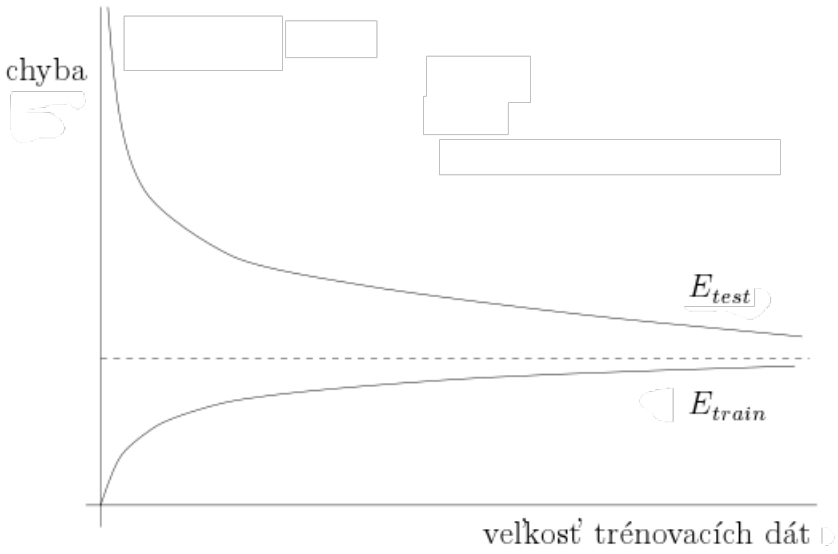
$$E^n = \frac{1}{2} \sum_{k=1}^{c} (t_k^n - y_k^n)^2.$$

Po procese trénovania je treba vyskúšať kvalitu nastavenia siete. Za týmto účelom definujeme *chybu testovania* E_{test} ako

$$E_{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} E^n,$$

kde N_{test} označuje veľkosť testovacích dát. Aby sme dostali realistický odhad chyby, potrebujeme vyrátať rozdiely medzi výstupmi siete a cieľovými hodnotami na celom vstupnom obore. Ak predpokladáme N_{test} dosť veľké, tak môžeme tento odhad dobre aproximovať.

Veľkosť trénovacích dát Pri nízkom počte trénovacích dát je síce chyba trénovania veľmi malá, ale chyba testu veľká. Zvyšovaním množstva trénovacích dát narastá chyba trénovania a zároveň sa znižuje chyba testovania (viď Obr. 4). Pri zväčšovaní trénovacej množiny konvergujú obe chyby k jednej hodnote. Táto hodnota závisí od reprezentačnej schopnosti siete, ktorá záleží na počte skrytých uzlov a použitých aktivačných funkciách.



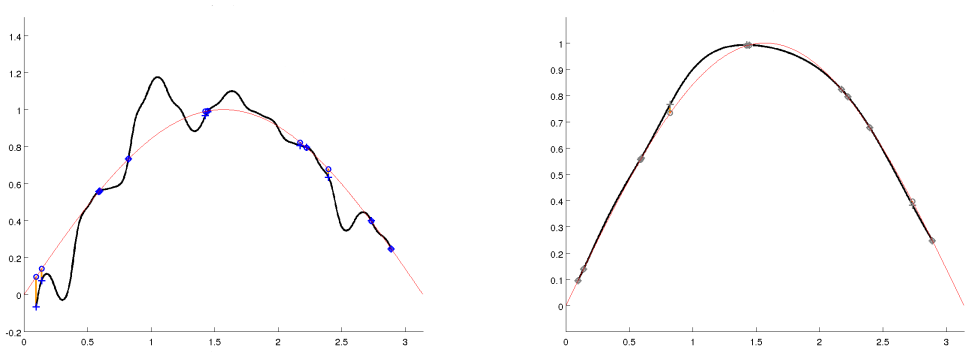
Obr. 4: Závislosť chyby trénovania a chyby testovania na veľkosti trénovacích dát.

Počet skrytých uzlov Závislosť aproximačnej chyby na počte skrytých uzlov ukážeme na príklade. Uvažujme úlohu odhadu funkcie $f(x) = \sin x$ pomocou dvojvrstvovej doprednej siete so sigmoidovými aktivačnými funkciami pre skryté a s lineárnymi aktivačnými funkciami pre výstupné uzly. Zvyšovaním počtu skrytých uzlov klesá chyba trénovania, ale od určitého množstva sa sieť stane *pretrénovanou (overtrained)* (viď Obr. 5(a)). Odhad funkcie síce bude mať malé hodnoty reziduí, ale výrazne sa odlišuje od funkcie $f(x)$.

Na pretrénovanie si treba dať pozor najmä u dát so šumom. Sieť vtedy aproximuje šum na úkor odhadu funkcie. So vzrastajúcim počtom skrytých uzlov chyba testovania najprv klesá, ale od určitého momentu začína stúpať ako vidno na obrázku 6.

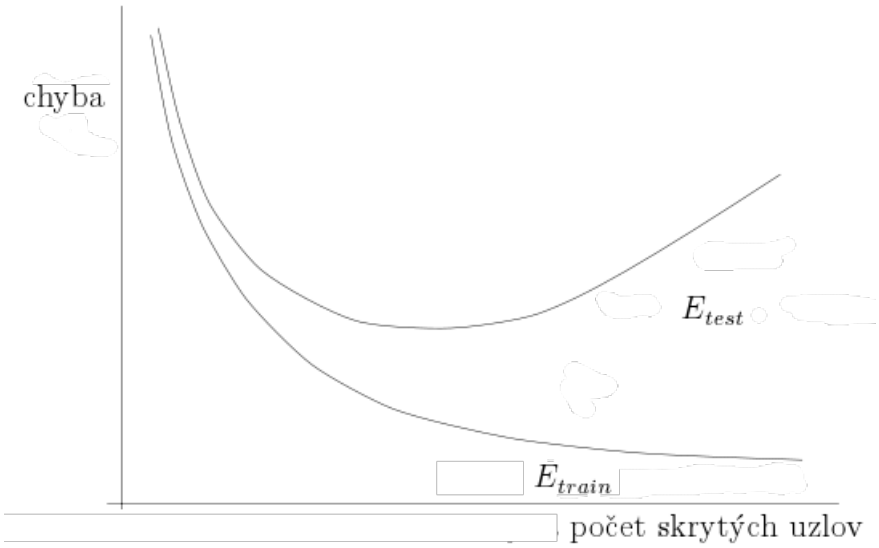
Opačným extrémom je malý počet skrytých neurónov vzhľadom k riešenému problému. Vtedy sieť nie je schopná vyťažiť všetku informáciu zo vstupných dát.

Neexistuje presný návod ako určiť počet skrytých uzlov, ale uvedieme dva empirické postupy. Prvý odporúča vytvoriť sieť s malým množstvom skrytých neurónov, čo zvyčajne znamená dva. Sieť sa potom trénuje a otestuje. Následne sa pridajú skryté uzly a proces sa opakuje dovtedy, kým sa



(a) (b)

Obr. 5: Odhad funkcie $f(x) = \sin x$ znázornenou červenou čiarou dvojvrstvovou doprednou sieťou s dvadsiatimi (a) a s tromi skrytými uzlami (b). Oranžové čiary značia reziduá odhadu.



Obr. 6: Závislosť chyby trénovania a chyby testovania na počte skrytých uzlov.

zlepšujú výsledky testu. Druhý prístup na začiatku vytvorí sieť s veľkým počtom skrytých uzlov, ktorá sa trénuje a testuje. Skryté uzly sa uberajú, kým sa schopnosť siete vyriešiť danú úlohu významne zlepšuje.

Kapitola 5

Aplikácia neurónových sietí

V tejto kapitole ukážeme riešenie piatich úloh týkajúcich sa lineárnej a nelineárnej regresie a odhadu funkcie pomocou neurónových sietí na náhodne vygenerovaných dátach. Na vytvorenie dát sme použili program R. Na odhad funkcií lineárnou a nelineárnou regresiou a na simuláciu siete sme využili program Matlab s prostredím Neural Network Toolbox. Zdrojový kód sa nachádza v dodatku A.

Vstupné dáta sme generovali z rovnomerného rozdelenia na intervale $(0, 10)$ pre úlohy 1 a 4, na $(0, 2\pi)$ pre úlohu 2, na $(-5, 5)$ pre úlohu 3 a na $(0, 6)$ pre úlohu 5. Aby sme zabezpečili dostatočné množstvo trénovacích dát, tak sme zvolili 50 vstupov $x^1, ..., x^{50}$. Zároveň sme vytvorili 50 hodnôt $x^1_{test}, ..., x^{50}_{test}$, ktoré budú slúžiť na testovanie siete.
Cieľové hodnoty t^n pre n-tý vstup, $n = 1, .., 50$, sa vypočítajú pre jednotlivé zadania pomocou odhadovanej funkcie $f(x)$ a šumu ε^n

$$t^n = f(x^n) + \varepsilon^n,$$

kde $\varepsilon^n \sim N(0, \sigma^2)$. Pre úlohy 1, 3 a 4 sme zvolili $\sigma = 1$. Pre úlohu 2 je $\sigma = 0, 5$. Pre úlohu 5 je $\sigma = 2$.
Analogicky sme vyrátali cieľové hodnoty t^n_{test}, $n = 1, ..., 50$.
Na aproximáciu sme vytvorili dvojvrstvovú doprednú sieť s jedným vstupným uzlom, M skrytými uzlami so sigmoidovou aktivačnou funkciou a jedným výstupným uzlom s lineárnou aktivačnou funkciou. Počet skrytých uzlov M je špecifický pre každú úlohu a určili sme ho pomocou prvého empirického postupu na zistenie optimálneho množstva skrytých uzlov opísaného v kapitole 4.
Na trénovanie siete sme použili algoritmus spätného šírenia chyby s využitím

Levenberg-Marquardtovho algoritmu pre druhú etapu algoritmu. Za chybovú funkciu sme zvolili strednú štvorcovú chybu (mean squared error). Pretože počiatočné hodnoty váh v sieti ovplyvňujú trénovanie, viackrát sme zopakovali inicializáciu váh náhodnými číslami a samotný trénovací proces. Z výsledných sietí sme vybrali najvhodnejšiu. Sieť sme trénovali vopred pripravenými dátami x^n a t^n, $n = 1, ..., 50$. Natrénovanú sieť sme otestovali dvomi spôsobmi: zadaním vstupných a testovacích dát. Po vložení vstupných dát $x^1, ..., x^{50}$ sme dostali výstup $y^1, ..., y^{50}$. Ten sme využili na vyrátanie strednej štvorcovej chyby odhadu sieťou MSE_{net}. Ďalej sme na dátach $(x^1, t^1), ..., (x^{50}, t^{50})$ aproximovali funkciu $f(x)$ v úlohe 1 pomocou lineárnej a v úlohach 2 až 5 pomocou nelineárnej regresie. Označme takto získanú aproximáciu ako $\widehat{f_{fit}(x)}$. Spočítali sme strednú štvorcovú chybu tohto odhadu MSE_{fit}. Z testovacích dát $x^1_{test}, ..., x^{50}_{test}$ sme dostali výstup $y^1_{test}, ..., y^{50}_{test}$, z ktorého sme vypočítali strednú štvorcovú chybu pre testovacie dáta $MSE_{testnet}$. Odčítaním hodnôt $\widehat{f_{fit}(x^n_{test})}$ od t^n_{test}, $n = 1, ..., 50$ sme získali reziduá pre odhad tradičným prístupom pre testovacie dáta, z ktorých sme vypočítali $MSE_{testfit}$. Nelineárnu regresiu sme používali so znalosťou tvaru funkcie a odhadovali sme len jej parametre. Napríklad v úlohe 2 odhadujeme parametre a, b, c funkcie $a \sin bx + c$.

Výsledky sú uvedené v podkapitolách 5.1 až 5.5. Interpretované a porovnávané s tradičným štatistickým prístupom sú v podkapitole 5.6. Koeficienty vo funkciách sme zvolili náhodne. Číselné výsledky sú zaokrúhlené na päť desatinných miest.

5.1 Úloha 1 - Lineárna funkcia

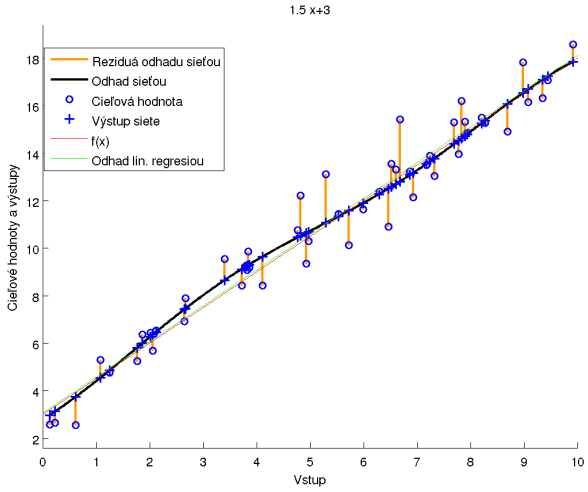
V tejto úlohe odhadujeme funkciu

$$f(x) = 1,5x + 3.$$

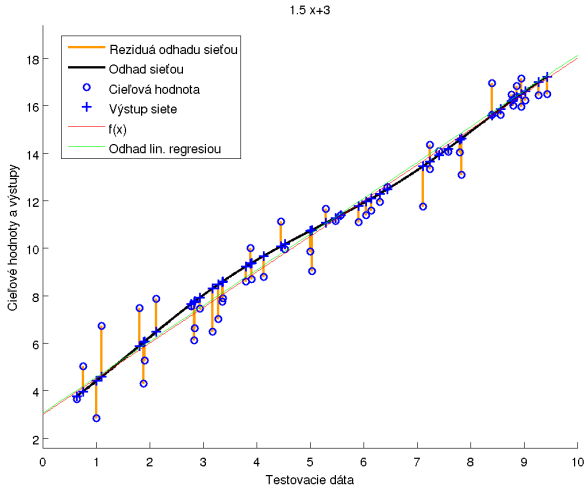
Odhad $f(x)$ sieťou a lineárnou regresiou na dátach $(x^1, t^1), ..., (x^{50}, t^{50})$ je znázornený na obrázku 7. Testovacie dáta spolu s ich reziduami sú zobrazené na obrázku 8. Hodnoty stredných štvorcových chýb sú zapísané v tabuľke 1.

Tab. 1: Hodnoty strednej štvorcovej chyby MSE pre úlohu 1.

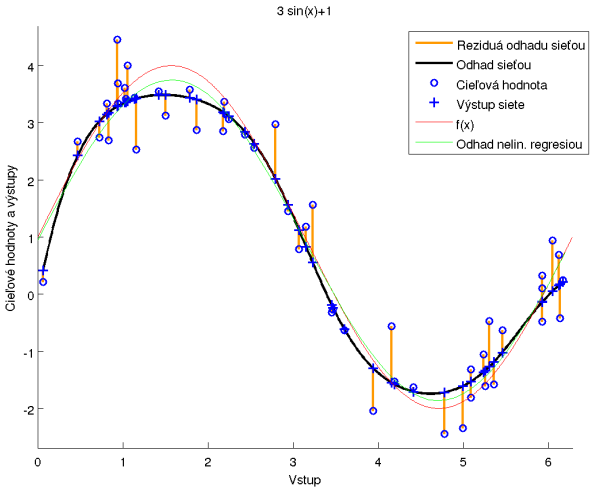
MSE_{net}	MSE_{fit}	$\text{MSE}_{testnet}$	$\text{MSE}_{testfit}$
0,78286	0,82984	0,91228	0,86023



Obr. 7: Odhad funkcie $f(x)$ z úlohy 1 na trénovacích dátach $(x^1, t^1), ..., (x^{50}, t^{50})$ neurónovou sieťou a lineárnou regresiou.



Obr. 8: Testovacie dáta a ich reziduá pre úlohu 1.



Obr. 9: Odhad funkcie $f(x)$ z úlohy 2 na trénovacích dátach $(x^1, t^1), ..., (x^{50}, t^{50})$ neurónovou sieťou a nelineárnou regresiou.

5.2 Úloha 2 - Sínusoida

Aproximujeme funkciu

$$f(x) = 3\sin(x) + 1.$$

Na obrázku 9 je znázornený odhad $f(x)$ sieťou a nelineárnou regresiou na dátach $(x^1, t^1), ..., (x^{50}, t^{50})$. Testovacie dáta a ich reziduá sú vykreslené na obrázku 10. Hodnoty stredných štvorcových chýb sú zapísané v tabuľke 2.

Tab. 2: Hodnoty strednej štvorcovej chyby MSE pre úlohu 2.

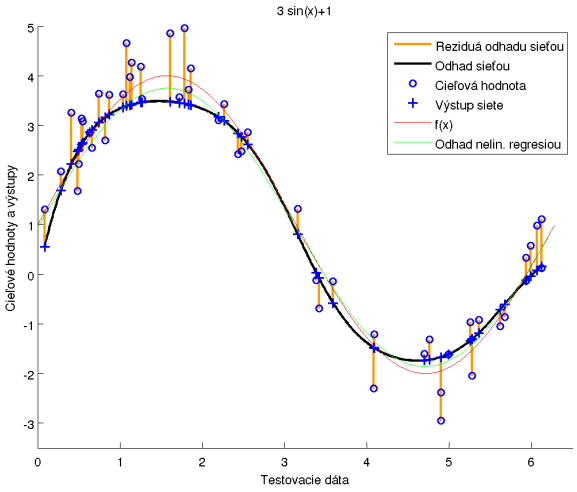
MSE_{net}	MSE_{fit}	$\text{MSE}_{testnet}$	$\text{MSE}_{testfit}$
0,23385	0,27894	0,3902	0,32761

5.3 Úloha 3 - Exponenciála

V tejto úlohe odhadujeme funkciu

$$2e^{0,45x} - 6.$$

Odhad $f(x)$ sieťou a nelineárnou regresiou je zobrazený na obrázku 11. Testovacie dáta a ich reziduá sú na obrázku 12. Stredné štvorcové chyby odhadov sú uvedené v tabuľke 3.



Obr. 10: Testovacie dáta a ich reziduá pre úlohu 2.

Tab. 3: Hodnoty strednej štvorcovej chyby MSE pre úlohu 3.

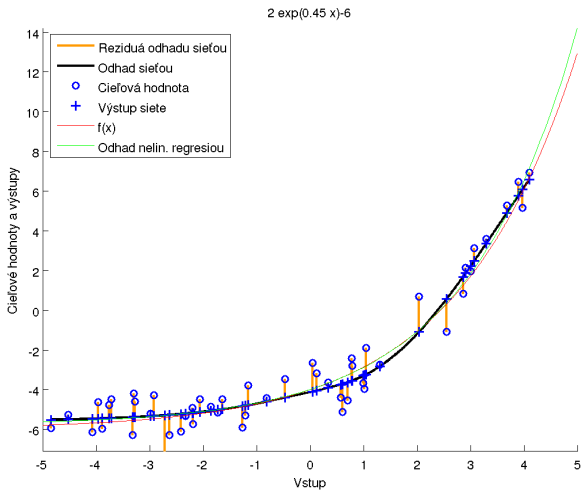
MSE_{net}	MSE_{fit}	$\text{MSE}_{testnet}$	$\text{MSE}_{testfit}$
0,76327	0,7864	1,1458	1,08358

5.4 Úloha 4 - Odmocnina

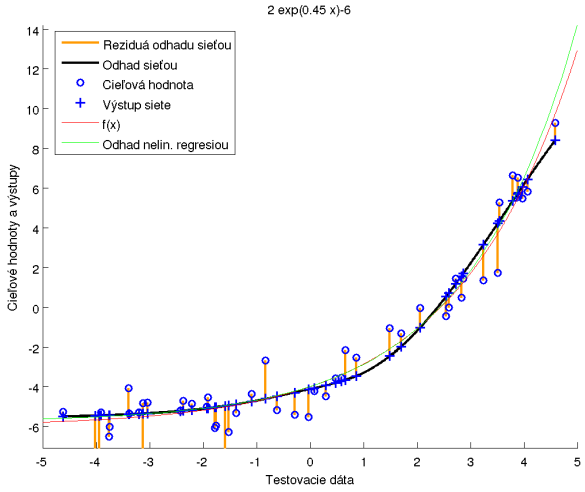
Odhadujeme funkciu

$$2\sqrt{5x} + 3.$$

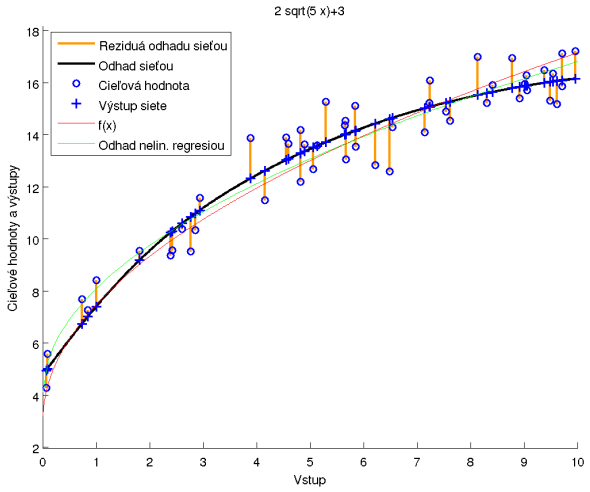
Odhad $f(x)$ sieťou a nelineárnou regresiou je zobrazený na obrázku 13. Testovacie dáta a ich reziduá sú vyobrazené na obrázku 14. Stredné štvorcové chyby odhadov sú v tabuľke 4.



Obr. 11: Odhad funkcie $f(x)$ z úlohy 3 na trénovacích dátach $(x^1, t^1), ..., (x^{50}, t^{50})$ neurónovou sieťou a nelineárnou regresiou.



Obr. 12: Testovacie dáta a ich reziduá pre úlohu 3.



Obr. 13: Odhad funkcie $f(x)$ z úlohy 4 na trénovacích dátach $(x^1, t^1), ..., (x^{50}, t^{50})$ neurónovou sieťou a nelineárnou regresiou.

Tab. 4: Hodnoty strednej štvorcovej chyby MSE pre úlohu 4.

MSE_{net}	MSE_{fit}	$\text{MSE}_{testnet}$	$\text{MSE}_{testfit}$
0,69636	0,73729	0,89335	0,86364

5.5 Úloha 5 - Kombinácia predošlých funkcií

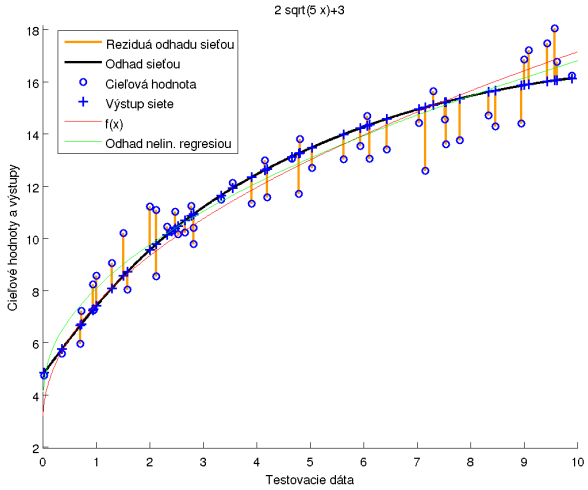
Súčtom odhadovaných funkcií z úloh 1 až 4 dostávame

$$f(x) = 1,5x + 2\sqrt{5x} + 3\sin x + 2e^{0,45x} + 1.$$

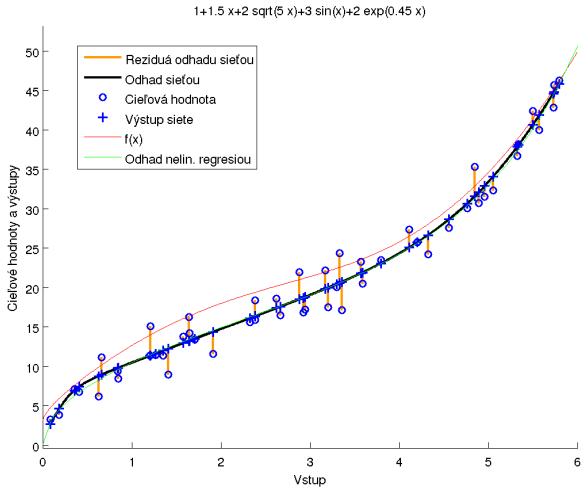
Na obrázku 15 je znázornený odhad $f(x)$ sieťou a nelineárnou regresiou na dátach $(x^1, t^1), ..., (x^{50}, t^{50})$. Testovacie dáta a ich reziduá sú vykreslené na obrázku 16. Hodnoty stredných štvorcových chýb sú zapísané v tabuľke 5.

Tab. 5: Hodnoty strednej štvorcovej chyby MSE pre úlohu 5.

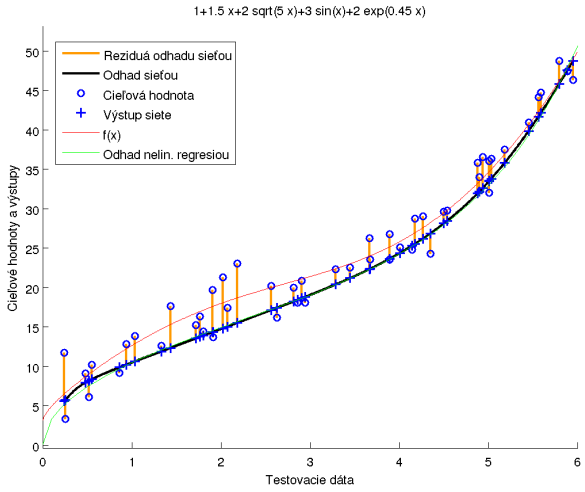
MSE_{net}	MSE_{fit}	$\text{MSE}_{testnet}$	$\text{MSE}_{testfit}$
3,42789	3,37141	8,06157	8,40422



Obr. 14: Testovacie dáta a ich reziduá pre úlohu 4.



Obr. 15: Odhad funkcie $f(x)$ z úlohy 5 na trénovacích dátach $(x^1, t^1), ..., (x^{50}, t^{50})$ neurónovou sieťou a nelineárnou regresiou.



Obr. 16: Testovacie dáta a ich reziduá pre úlohu 5.

5.6 Vyhodnotenie

Z hodnôt podielov $\frac{MSE_{net}}{MSE_{fit}}$ pre jednotlivé úlohy (viď Tabuľka 6) vidíme, že neurónové siete majú až na úlohu 5 lepšie výsledky pre trénovacie dáta ako tradičný prístup. To sa dá vysvetliť väčšou mierou prispôsobivosti neurónových sietí dátam ako má lineárna, poprípade nelineárna regresia. Hodnoty MSE_{net} a MSE_{fit} sa ale veľmi nelíšia. Najnižší podiel sme zaznamenali v úlohe 2, kde $\frac{MSE_{net}}{MSE_{fit}} = 0,83835$. Najvyšší podiel mala úloha 5, kde sme dostali $\frac{MSE_{net}}{MSE_{fit}} = 1,01675$.

Na testovacích dátach neurónová sieť vykazuje vyššiu strednú chybu ako tradičný prístup (viď Tabuľka 6). Výnimku tvorí úloha 6, kde sieť dosiahla lepší výsledok ako štatistický odhad. Najnižšia hodnota podielu sa dosiahla v úlohe 5, kde $\frac{MSE_{testnet}}{MSE_{testfit}} = 0,95923$. Najväčší podiel $\frac{MSE_{testnet}}{MSE_{testfit}}$ sme dostali v úlohe 2, kde $\frac{MSE_{testnet}}{MSE_{testfit}} = 1,19105$. Všimnime si, že najlepší pomer stredných štvorcových chýb pre testovacie dáta sme pozorovali v úlohe 5, pre ktorú sme dostali najhorší podiel $\frac{MSE_{net}}{MSE_{fit}}$ pre trénovacie dáta. Naopak, najhorší pomer $\frac{MSE_{testnet}}{MSE_{testfit}}$ sme dostali pre úlohu 2, pre ktorú máme najlepší podiel $\frac{MSE_{net}}{MSE_{fit}}$ pre trénovacie dáta.
Horší výkon neurónovej siete na testovacích dátach je zapríčinený vyššou adaptáciou na šum v trénovacích dátach ako má tradičný štatistický odhad.

Tab. 6: Podiely $\frac{MSE_{net}}{MSE_{fit}}$ a $\frac{MSE_{testnet}}{MSE_{testfit}}$.

Číslo úlohy	$\frac{MSE_{net}}{MSE_{fit}}$	$\frac{MSE_{testnet}}{MSE_{testfit}}$
1	0,94339	1,06051
2	0,83835	1,19105
3	0,97059	1,05742
4	0,94449	1,0344
5	1,01675	0,95923

Kapitola 6

Záver

V práci sme zadefinovali základné pojmy týkajúce sa neurónových sietí a popísali sme jedno- a viacvrstvové dopredné siete. Pre jednovrstvové siete sme predstavili učebný algoritmus metódu najmenších štvorcov pre sumu štvorcov chybovej funkcie a gradientovú metódu. Diskutovali sme podobnosť jednovrstvových sietí s lineárnymi diskriminačnými funkciami. Pre viacvrstvové siete sme uviedli učebné algoritmy algoritmus spätného šírenia chyby, gradientovú metódu s momentom a Levenberg–Marquardtov algoritmus. Ku koncu práce sme odhadovali päť náhodne zvolených funkcií lineárnou a nelineárnou regresiou.

Kvalitu odhadov sme merali strednou štvorcovou chybou. Na trénovacích dátach neurónová sieť obecne vykazovala lepšie výsledky ako tradičný prístup. Odhady sa ale sa veľmi nelíšili. Pri testovacích dátach sa tradičný prístup ukázal byť vhodnejším v štyroch z piatich úloh, čo pripisujeme schopnosti neurónovej siete sa veľmi dobre adaptovať na trénovacie dáta na úkor zanedbania šumu v dátach.

Neurónové siete poskytujú porovnateľne dobrý odhad ako tradičný prístup. Ich výhodou oproti lineárnej a nelineárnej regresii je, že nepožadujú znalosť tvaru funkcie, z ktorej sú dáta nagenerované. Zlepšenie presnosti odhadu by sa dalo docieliť napríklad zvýšením počtu vzorov v trénovacej množine. Použitím iných algoritmov by sme prišli k odlišným výsledkom.

Literatura

[1] Bishop, Ch. M.: *Neural Networks for Pattern Recognition*, Oxford University Press, 1995

[2] Kröse B., Smagt P. v. d.: *An Introduction to Neural Networks*, University of Amsterdam, 1996

Dodatek A

Zdrojový kód

V tejto prílohe uvedieme zdrojové kódy pre programy R a Matlab.

Na generáciu dát sme použili verziu 2.9.2 programu R. Postup pri tvorbe dát pre jednotlivé úlohy sa líši len v predpise funkcie na generáciu cieľových hodnôt a v nastavení hodnoty `set.seed()`, ľavej a pravej medze intervalu, na ktorom dáta generujeme, a smerodajnej odchýlky pri generácií šumu z normálneho rozdelenia. Z tohto dôvodu uvedieme zdrojový kód len pre úlohu 1. Zvyšné zdrojové kódy sú dostupné na priloženom CD.

Na simuláciu neurónovej siete a odhady funkcií pomocou lineárnej a nelineárnej regresie sme použili program Matlab s prostredím Neural Network Toolbox. Pretože zdrojový kód pre úlohy 2 až 5 sa odlišuje len v názvoch premenných a konečnom počte skrytých uzlov v prvej vrstve, uvedieme zdrojový kód len pre úlohu 2. Ostatné zdrojové kódy sú prístupné na priloženom CD.

Generovanie dát:

```
set.seed(1)
x1<-matrix(runif(50,min=0,max=10),1,50)
write(x1, file='x1',ncolumns=50)
eps1<-matrix(rnorm(50,sd=1),1,50)
t1<-1.5*x1+3+eps1
write(t1, file='t1',ncolumns=50)
testdata1<-matrix(runif(50,min=0,max=10),1,50)
write(testdata1,file='testdata1',ncolumns=50)
testeps1<-matrix(rnorm(50, sd=1),1,50)
testt1<-1.5*testdata1+3+testeps1
```

```
write(testt1,file='ttest1',ncolumns=50)
```

Zdrojový kód k úlohe 1:
(Načítanie dát vygenerovaných programom R zo súboru. Cestu k súborom sme označili ako `path.`)

```
x1=load('path/x1');
t1=load('path/t1');
testdata1=load('path/testdata1');
testt1=load('path/testt1');
```

(Vytvorenie neurónovej siete, inicializácia váh a trénovanie.)

```
net1=newfit(x1,t1,2,{'logsig','purelin'},'trainlm','learngdm','mse');
net1=initnw(net1,1);
net1=initnw(net1,2);
[net1,tr1]=train(net1,x1,t1);
```

(Výpočet výstupov siete a stredných štvorcových chýb MSE_{net} a $MSE_{testnet}$.)

```
y1=sim(net1,x1);
resnet1=t1-y1;
msenet1=mse(resnet1);
testy1=sim(net1,testdata1);
restestnet1=testt1-testy1;
msetestnet1=mse(restestnet1);
```

(Odhad lineárnou regresiou a výpočet stredných štvorcových chýb MSE_{fit} a $MSE_{testfit}$.)

```
fit1=polyfit(x1,t1,1);
resfit1=t1-polyval(fit1,x1);
msefit1=mse(resfit1);
restestfit1=testt1-polyval(fit1,testdata1);
msetestfit1=mse(restestfit1);
```

(Vykreslenie obrázkov.)

```
x=[0:0.1:10];
plotfit(net1,x1,t1)
hold on
a=ezplot('1.5*x+3',[0,10]);
```

```
set(a,'Color','red')
b=plot(x,polyval(fit1,x),'-');
set(b,'Color','green')
xlabel('Vstup')
ylabel('Cieľové hodnoty a výstupy')
legend('Reziduá odhadu sieťou','Odhad sieťou','Cieľová hodnota','Výstup siete',
'f(x)','Odhad lin. regresiou')
hold off

plotfit(net1,testdata1,testt1)
hold on
a=ezplot('1.5*x+3',[0,10]);
set(a,'Color','red')
c=plot(x,polyval(fit1,x),'-');
set(c,'Color','green')
xlabel('Testovacie dáta')
ylabel('Cieľové hodnoty a výstupy')
legend('Reziduá odhadu sieťou','Odhad sieťou','Cieľová hodnota','Výstup siete',
'f(x)','Odhad lin. regresiou')
hold off
```

Zdrojový kód k úlohe 2:
(Načítanie dát vygenerovaných programom R zo súboru. Cestu k súborom sme označili ako `path`.)

```
x2=load('path/x2');
t2=load('path/t2');
testdata2=load('path/testdata2');
testt2=load('path/testt2');
```

(Vytvorenie neurónovej siete, inicializácia váh a trénovanie.)

```
net2=newfit(x2,t2,3,{'logsig','purelin'},'trainlm','learngdm','mse');
net2=initnw(net2,1);
net2=initnw(net2,2);
[net2,tr2]=train(net2,x2,t2);
```

(Výpočet výstupov siete a stredných štvorcových chýb MSE_{net} a $MSE_{testnet}$.)

```
y2=sim(net2,x2);
plotfit(net2,x2,t2)
```

```
resnet2=t2-y2;
msenet2=mse(resnet2);
testy2=sim(net2,testdata2);
restestnet2=testt2-testy2;
msetestnet2=mse(restestnet2);
```

(Odhad nelineárnou regresiou a výpočet stredných štvorcových chýb MSE_{fit} a $MSE_{testfit}$.)

```
koef2=[3;1;1];
fit2=nlinfit(x2,t2,@fciasinusoida,koef2);
resfit2=t2-fciasinusoida(fit2,x2);
msefit2=mse(resfit2);
restestfit2=testt2-fciasinusoida(fit2,testdata2);
msetestfit2=mse(restestfit2);
```

(Vykreslenie obrázkov.)

```
x=[0:0.1:2*pi];
plotfit(net2,x2,t2)
hold on
a=ezplot('3*sin(x)+1',[0,2*pi])
set(a,'Color','red')
b=plot(x,fciasinusoida(fit2,x),'-')
set(b,'Color','green')
xlabel('Vstup')
ylabel('Cieľové hodnoty a výstupy')
legend('Reziduá odhadu sieťou','Odhad sieťou','Cieľová hodnota','Výstup siete',
'f(x)','Odhad nelin. regresiou')
hold off

plotfit(net2,testdata2,testt2)
hold on
a=ezplot('3*sin(x)+1',[0,2*pi])
set(a,'Color','red')
c=plot(x,fciasinusoida(fit2,x),'-')
set(c,'Color','green')
xlabel('Testovacie dáta')
ylabel('Cieľové hodnoty a výstupy')
legend('Reziduá odhadu sieťou','Odhad sieťou','Cieľová hodnota','Výstup siete',
```

```
'f(x)','Odhad nelin. regresiou')
axis([0 6.5 -3.5 5])
hold off
```

(Definovanie funkcie `fciasinusoida` v súbore fciasinusoida.m)

```
function z=fciasinusoida(koef,x)
b1=koef(1);
b2=koef(2);
b3=koef(3);
z=b1*sin(b2*x)+b3;
```