UNIVERZITA PALACKÉHO V OLOMOUCI
PŘÍRODOVĚDECKÁ FAKULTA
KATEDRA MATEMATICKÉ ANALÝZY A APLIKACÍ MATEMATIKY

DIPLOMOVÁ PRÁCE

Neuronové sítě a jejich aplikace

Vedoucí diplomové práce:
RNDr. Pavel Ženčák, Ph.D.
Rok odevzdání: 2012

Vypracoval:
Bc. Michal Roubal
MPM, II. ročník

Prohlášení

Prohlašuji, že jsem vytvořil tuto diplomovou práci samostatně za vedení RNDr. Pavla Ženčáka, Ph.D. a že jsem v seznamu použité literatury uvedl všechny zdroje použité při zpracování práce.

V Olomouci dne 28. března 2012

Poděkování

Rád bych na tomto místě poděkoval vedoucímu diplomové práce RNDr. Pavlu Ženčákovi, Ph.D. a RNDr. Tomáši Fürstovi, Ph.D. za obětavou spolupráci i za čas, který mi věnovali při konzultacích. Dále si zasluží poděkování můj počítač, že vydržel moje pracovní tempo, a typografický systém TEX, kterým je práce vysázena. A v neposlední řadě mojí rodině, která mě podporovala při náročném psaní diplomové práce.

Obsah

Značení

j, k	neurony j, k
m	označuje číslo epochy m
l	index vrstvy sítě
p	index prvku trénovací množiny
$\mathbf{x}^p$	p-tý prvek trénovací množiny
x_j^p	j-tý index p-tého trénovacího prvku
$\boldsymbol{t^p}$	daný výstup p-tého prvku sítě, také označován jako *target*
t_j^p	j-tý index daného výstupu p-tého prvku sítě
$\mathbf{s}^l$	celkový vstup do neuronu v l-té vrstvě
$\mathbf{s}_j^l$	j-tý index celkového vstup do neuronu v l-té vrstvě
$\boldsymbol{y^l}$	výstupní hodnoty v l-té vrstvě sítě
y_j^l	j-tý index výstupní hodnoty v l-té vrstvě sítě
$\boldsymbol{W^l}$	matice vah v l-té vrstvě sítě
w_{jk}	váhy mezi neurony j, k
$\mathcal{F}$	aktivační funkce neuronu
w_0	bias
γ	parametr učení
μ	moment sítě

Úvod

Důvod proč jsem si vybral diplomovou práci na téma neuronové sítě byl jednoduchý. Toto téma mě přitahovalo vzhledem k jeho ohromným možnostem aplikací v různých odvětvích, ať už medicíny, financí nebo robotiky. První zmínky o neuronových sítích jsem slyšel v souvisloti s rozpoznáním obrazu, proto jsem si vybral neuronové sítě a jejich aplikaci s rozpoznáním obrazu. Doufám, že na konci práce budu mít stejné nadšení jako mám na začátku, akorát budu bohatší o několik programů na rozpoznání registračních značek pomocí neuronových sítí. Vytvoření těchto neuronových sítí je cílem této diplomové práce.

V této práci si rozebereme do detailu neuronové sítě s dopředným šířením. Nebudeme se zabývat dalšími typy sítí, jako jsou například samoorganizační nebo rekurentní sítě. Nicméně uvedeme jejich krátký popis a typy úloh na které se hodí. Sítě s dopředným šířením se hodí na klasifikační, aproximační a predikční úlohy. Na počátku je vždy dána dvojice $\{\mathbf{x}_i, \mathbf{t}(\mathbf{x}_i)\}$, kde $\mathbf{x}_i$ je i-tá vlastnost trénovacího prvku a $\mathbf{t}(\mathbf{x}_i)$ je požadovaný výstup, na který bychom chtěli síť natrénovat.

Neuronová síť může být nejadekvátněji popsána jako výpočetní model, který má schopnosti učit se, rozpoznávat, zobecňovat, shromažďovat nebo, organizovat data nebo, provádět operace, jež jsou založeny na paralelním zpracování. Neuronová síť se učí paralelně díky své architektuře, učí se pomocí nastavování jednotlivých vah a prahů citlivosti neuronů. Obyčejný program se řídí víceméně sekvenčně na základě předem zadaných instrukcí.

Použití neuronových sítí je velmi široké. Například v automatickém řízení, kybernetice, umělé inteligenci, v těchto oborech se dají neuronové sítě uplatnit pro rozpoznání obrazu, kontrolu trajektorie robotické ruky, rozpoznání kvality výrobku. Neuronová síť byla například použita pro distribuci pohonných hmot pro Los Angeles. Neuronová síť se dá použít také na predikci časových řad, řízení rizik, plánování splátkového kalendáře. Využití je nepřeberné.

V první kapitole si řekneme něco o základech neuronových sítí, co je jejich inspirací pro jejich vznik. Popíšeme si základní pojmy neuronových sítí a jak

vlastně neuronové sítě fungují.

Dále přijde na řadu pojem **perceptron**. Perceptron je základním kámenem všech neuronových sítí, jehož schématické principy se používají ve všech neuronových sítích. Ukážeme si jeho stavbu a spolu s ním si ukážeme základní učební pravidlo na jednovrstvé klasifikační síti.

V následující kapitole přijdou na řadu vícevrstvé neuronové sítě s dopředným šířením. Budeme se zde také zabývat algoritmem Back Propagation. Dále si zde ukážeme metody 1. a 2. řádu, které efektivně minimalizují chybovou funkci sítě. Na konci této kapitoly si ukážeme chybovou funkci s logistickou regresí, kterou budeme minimalizovat pomocí metody BFGS.

V kapitole Image Processing (zpracování obrazu) si řekneme, jak jsme postupovali při získávání vstupu pro neuronovou síť. Seznámíme se s prostředím pro zpracování obrazu v softwaru MATLAB^{TM}(Image Processing Toolbox - IPT) a popíšeme si jednotlivé funkce, které jsme použili.

Kapitola vyhodnocení se bude zabývat rozebráním výsledků, kterých jsme dosáhli při použití různých metod na minimalizaci obou chybových funkcí.

Nyní si povíme něco málo o historii neuronových sítí. Historie neuronových sítí se začala psát v 20. letech 20. století, kdy Američan Walter McCulloh poprvé napsal svoje pojednání o neuronech. O 20 let později se svým studentem Warrenem Pittsem přišli s jednoduchým modelem neuronu, který se v podstatě používá dodnes. V roce 1949 vydal svoji knihu **Organization of Behavior** Donald Hebb. Tato kniha poskytla první učící pravidlo pro synapse neuronů. V roce 1951 byl dokonce postaven první neuropočítač **Snark** Marvinem Minskym. Minsky byl také první, kdo upozornil na nedostatky jednovrstvé sítě spočívající v tom, že dokáže řešit pouze lineárně separabilní problémy.

V roce 1957 zobecnil Frank Rosenblatt původní model neuronu a nazval ho Perceptron. Sám také navrhl algoritmus na nalezení jednotlivých vah v reálném čase. V průběhu dalších let se objevovaly další typy jednovrstvých sítí jako třeba ADALINE a dokonce jiné typy architektur sítí, jako třeba asociativní sítě.

Trvalo nějaký čas a řešení nelineárně separabilních problémů bylo na světě.

Stalo se tak na počátku 80. let. Přispěly k tomu dvě věci. První z nich bylo přidání další vrstvy neuronů a jako druhé bylo objevení algoritmu Back Propagation.

1 Základy neuronových sítí

Neuronová síť je složena z neuronů (výpočetních jednotek). Tyto neurony tvoří architekturu (topologii) sítě. V této kapitole si ukážeme, co inspirovalo vznik neuronů, jejich stavbu, jak tvoří vlastní síť a co je dělá tak mocným nástrojem. Dále si popíšeme základní pojmy neuronové sítě, které budeme používat v celé práci.

1.1 Podobnost biologických a technických neuronů

Podobnost biologického a technického neuronu je čistě schématická. Biologický neuron je mnohem složitější než technický neuron. V dnešní době superpočítačů stále nejsme schopni pomocí technických neuronů vytvořit síť, která by byla alespoň trochu podobná nejvyspělejšímu neuropočítači vůbec, **mozku**.

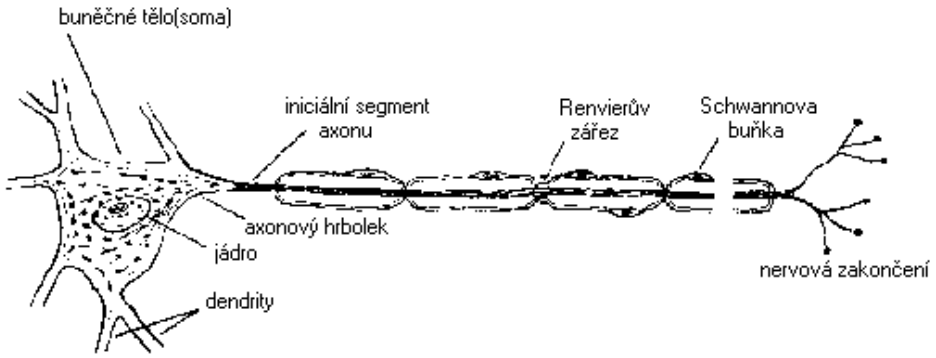
Mozek člověka má asi 10^{13-15} neuronů, nemluvě o jejich spojeních. Každý den odumře zhruba 10^4 neuronů, což za průměrný život člověka dělá $2.5.10^{-4}\%$ z celkového počtu. Každý neuron má v průměru 6000 somatických a dendrických spojení pokrývajících asi 40% jeho plochy.

Následující celá část o biologickém neuronu byl vypůjčena z [Zelinka].

Biologický neuron: Biologický neuron se v blokovém popisu skládá ze vstupů (dendritů), těla (somatu) a výstupu (axonu). Jeho funkce je však složitější. Každý neuron je pokryt dvojvrstvou membránou z lipidů o tloušťce asi $20nm$. V této membráně jsou zapuštěny proteiny, které mají funkci kanálu přenášejícího ionty. Přenos Na^+ a K^+ iontů a jejich nerovnoměrné rozdělení na membráně způsobuje výsledný potenciál asi $70mV$. Při otevření kanálu dochází ke změně membránového potenciálu a vytvořená potenciálová vlna - vzruch se šíří po dendritech přes tělo neuronu do axonu. Podle toho, jak reaguje membrána na změnu potenciálu, rozeznáváme membránu vodivou (konduktivní) a nevodivou (transmisní). Vodivá membrána pokrývá axon a je tvořena myelinovou pochvou s Ranvierovými zářezy, které dovolují šíření vzruchu vysokou přenosovou rychlostí až $120m.s^{-1}$. Její reakce na vzruch je elektrická. Nevodivá membrána pokrývá dendrity a tělo neuronu. Její reakce na vzruch je chemická a rychlost přenosu

mnohem nižší.

Každý neuron má svůj práh, což znamená, že odezva neuronu na vstupní podnět se objeví až po překročení této hodnoty. Odezva se šíří po axonu opět ve formě potenciálové vlny. Vlastní výstup (axon) je rozvětven a spojen se vstupy (dendrity) dalších neuronů pomocí spojek nazývaných synapse, viz. obrázek (1). Známe dva druhy přenosu na synapsích, a to elektrický (vývojově starší) a che-



Obrázek 1: Skladba neuronu zdroj:
http://neuron.felk.cvut.cz/courseware/html/chapters.html

mický (mladší, dominující u savců). Synapse funguje jako ventil, a to ve dvojím režimu - excitačním (budícím) a inhibičním (tlumícím). Jaká bude odezva (excitace, inhibice) záleží na transmiterech (nosiči informace - acetylcholin, glutamát (excitace), glycin a GABA (inhibice)), které se uvolní v presynaptické části axonu. Bylo zjištěno, že transmitery, které působí krátkodobě jsou zodpovědné za přenosovou funkci a transmitery, které působí dlouhodobě, jsou zodpovědné za paměť. Pokud si promítneme odezvu neuronu na časovou osu, zjistíme, že potenciálové vlny tvoří sled impulsů v rozmezí $250 - 1000$ impulsů za sekundu. Mezi impulsy je tzv. refrakterní perioda, což je doba, po kterou je neuron necitlivý na vstupní podněty. Zkoumáním bylo zjištěno, že v případě živých tvorů se výše uvedený přenos informace děje frekvenční modulací.

Technický neuron napodobuje pouze formu, nikoliv obsah. Je to jednoduchá jednotka neuronové sítě, která ohodnotí vstupy vahami a takto vzniklé hodnoty sečte. Tuto sečtenou hodnotu dosadí do příslušné aktivační (prahové,

transformační) funkce neuronu, výstup této funkce je i celkový výstup tohoto neuronu a může být použit jako vstup do dalších neuronů.

1.2 Základní pojmy užívané v neuronových sítích

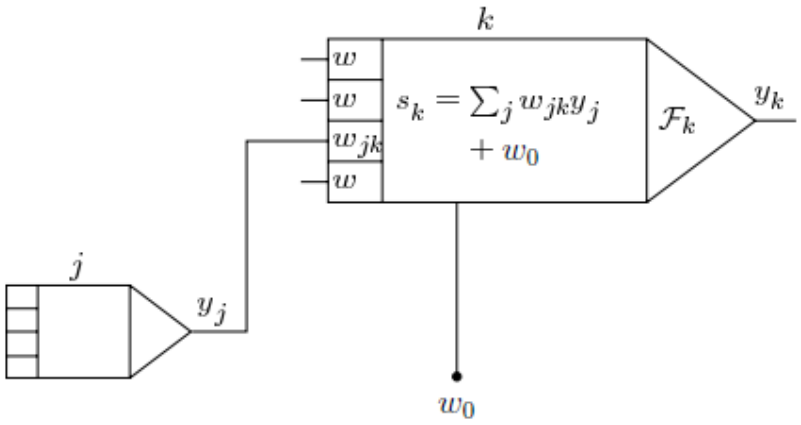
Neuronové sítě obsahují plno neuronů, které spolu komunikují posíláním signálů po vážených spojeních mezi nimi. U těchto sítí rozlišujeme několik základních pojmů:

- Množina neuronů (procesních jednotek, buňky).
- Stav aktivace pro každý neuron y_k, což je to samé jako výstup tohoto neuronu, index k označuje k-tý neuron.
- Spoje mezi jednotlivými procesními jednotkami. Těmto spojením přiřazujeme váhy w_{jk}^l, což označuje spojení mezi neuronem j a k v l-té vrstvě.
- Celkový vstup do neuronu v l-té vrstvě značíme s_k^l.
- Vstupem do aktivační funkce je celkový vstup (gradient response), značíme $\mathcal{F}\left(s_k^l\right)$, výstup aktivační funkce je výstup (aktivace) neuronu, značíme y_k^l v l-té vrstvě.
- Bias (externí vstup, odchylku) značíme jako w_0^l v l-té vrstvě.
- Metoda jakou získáváme informaci (učební pravidla).

1.2.1 Neurony

Technické neurony provádějí relativně jednoduchou práci. Obdrží vstupní infomaci skrze svá spojení, včetně biasu a z těchto informací spočítají výstupní signál pro další jednotky.

V rámci neuronových sítí bychom mohli rozlišovat tři typy neuronů. Vstupní neurony získávají data mimo neuronovou síť. Někteří autoři nepovažují vstup jako vstupní vrstvu, my se budeme spíše přiklánět k tomu, že první vrstva bude



Obrázek 2: Technický neuron [Kröse,Smagt]

trénovací množinou. Vnější neurony jsou výstupními neurony celé sítě, výstup těchto neuronů je výstup celé sítě. Tyto neurony mohou mít aproximační funkci, potom je na výstupu pouze jeden neuron. Nebo také klasifikační funkci, v tomto případě potřebujeme tolik neuronů jako je klasifikačních tříd. Posledním typem neuronů jsou neurony skryté. Jejich vstup a výstup zůstává uvnitř sítě, nicméně tyto neurony tvoří mozek celé sítě, kde se provádějí ty nejdůležitější procesy.

1.2.2 Spojení mezi neurony

Ve většině případů neurony sečtou všechny váhy, kterými jsou spojeny s neurony v předešlé vrstvě. Celkový vstup do jednotky je tedy vážená suma jednotlivých výstupů neuronů spojených s tímto neuronem plus jejích bias.

$$s_k^l = \sum_j w_{jk}^l y_j + w_0^l. \tag{1.1}$$

Kladný výsledek této formule se podle biologického neuronu nazývá **excitace** a záporný **inhibice**.

1.2.3 Aktivace a výstupní pravidla

Potřebujeme nějaké aktivační funkce, které budou počítat účinek celkového vstupu. Naše aktivační (transformační, prahová) funkce $\mathcal{F}$ vezme celkový vstup

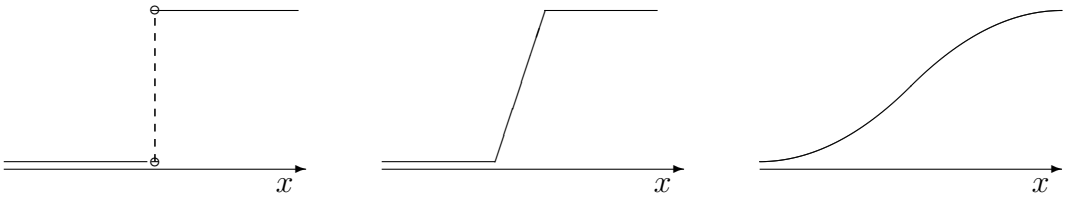
$s_k^l(m)$ a aktuální aktivační hodnotu $y_k^l(m)$ a vytvoří nový výstup neuronu k, což lze zapsat takto:

$$y_k^{l+1} = \mathcal{F}(y_k^l, s_k^l) \tag{1.2}$$

Aktivační funkce je často neklesající funkce celkového vstupu neuronu.

$$y_k^{l+1} = \mathcal{F}(s_k^l) = \mathcal{F}\left(\sum_j w_{jk}^l y_j^l + w_0^l\right). \tag{1.3}$$

V neuronových sítích máme na výběr z mnoha aktivačních funkcí, některé z nich jsou na obrázku (3):



signum semilineární logistická

Obrázek 3: Některé aktivační funkce neuronů

1.3 Topologie sítí

V minulé podkapitole jsme se bavili o neuronech. Nemluvili jsme však o architektuře těchto sítí a o rozložení těchto neuronů a rozložení spojení mezi nimi. Rozlišujeme dva druhy sítí.

1. **Sítě s dopředným šířením:**

   Data v nich plynou od vstupu směrem k výstupu, ze vstupní do skrytých vrstev a potom do vnější vrstvy sítě, stále jedním směrem. Neexistují zde žádná zpětná spojení vah.

2. **Rekurentní sítě**

   Topologie rekurentní sítě může obsahovat i spojení, kde se váží neurony

s vrstvou, kde se sami vyskytují nebo i s vrstvou předešlou. Topologie sítě tedy vytváří cykly. V rekurentních sítích je možné užívat něco obdobného jako Back Propagation až je dosaženo stabilního bodu (atraktoru). Existují sítě, které jsou založeny na principu atraktoru. Příkladem rekurentních sítí jsou například Hoppfieldovy sítě.

1.4 Trénování sítě

Budeme usilovat, abychom nastavili váhy sítě tak, aby množina vstupů produkovala správný výstup. K tomuto vedou dvě cesty, buď nastavit jednotlivé váhy a priori, na základě předešlé znalosti, nebo tuto síť trénovat pomocí předem daných trénovacích množin a měnit váhy podle nějakého učebního pravidla.

1.4.1 Typy učení

Typy učení můžeme rozdělit do dvou skupin:

1. **Trénování s učitelem:**
   Můžeme to též označit jako asociativní učení. Při tomto typu učení je síti poskytnuta trénovací množina plná trénovacích vzorů a příslušná množina správných výsledků. Jak takovéto sítě fungují si popíšeme později

2. **Trénování bez učitele:**
   Tento typ učení se také nazývá samoorganizace. Při tomto typu učení je též poskytnuta trénovací množina, ale není zadána množina daných výstupů. Síť si musí sama vytvořit nějaké statistické, charakteristické vlastnosti jednotlivých vzorů. Trénování probíhá bez přítomnosti daných výstupů. Algoritmy jsou založeny na globální soutěži mezi jednotlivými neurony. Uvedeme si několik příkladů typického použití těchto sítí.

   - **Shluková analýza**: Vstupní data mohou tvořit různé skupinky a neuronová síť musí najít jednotlivé shluky a výstup sítě by měl jednotlivé shluky od sebe nějak odlišit.

- **Kvantifikace vektoru**: Tento problém se vyskytne, když spojitý prostor musí být diskretizován. Vstup je n dimenzonální vektor a síť musí najít optimální diskretizaci. vstupu
- **Redukce dimenze**: Data jsou shromážděna v podprostoru menší dimenze než samotná dimenze dat. Síť se musí naučit optimální zobrazení takové, že většina rozdílu ve vstupních datech je zachována i na výstupu.
- **Vytažení vlastností signálu**: Síť se musí naučit rozpoznat různé vlastnosti signálu, což je často spojeno s redukcí dimenze signálu.

Více o rekurentních a samoorganizačních sítích se můžete dočíst v [Kröse,Smagt].

V této práci se samoorganizačním sítím dále nebudeme věnovat, protože nejsou potřeba pro praktický příklad, kterým se budeme zabývat.

1.4.2 Nastavování vah

Pro nastavování vah se používají dva základní principy.

První myšlenka nastavování vah je založena na úvaze, že **jsou-li jsou dva neurony aktivní současně, tak by měla být váha mezi nimi zesílena**. Jestliže neuron k obdrží signál z neuronu j, tak nastavení vah w_{jk} je následující

$$\Delta w_{jk} = \gamma y_j y_k,$$

kde γ je parametr učení náležící do intervalu $\langle 0, 1\rangle$. Toto se často nazývá **Hebbovo učení**.

Další způsob nastavování vah nepoužívá aktivační hodnotu jednotky k, nýbrž její **odchylku od kýženého výstupu** t_k, dostáváme tedy

$$\Delta w_{jk} = \gamma y_j (t_k - y_k).$$

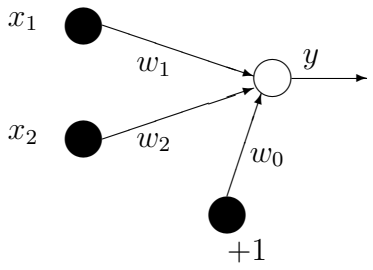
Toto pravidlo se často nazývá **Widrow-Hoffovo** nebo také **delta pravidlo**.

2 Jednovrstvé neuronové sítě

Perceptron je možné považovat za základní stavební kámen neuronových sítí. Byl navržen v roce 1958 F. Rosenblattem. V tomto roce vytvořil tzv. perceptronovou síť. V této kapitole si popíšeme detailněji, jak takový perceptron funguje a jaké problémy se pomocí perceptronu mohou řešit a posléze si ukážeme nějaká učební pravidla.

2.1 Sítě s prahovými aktivačními funkcemi

Jednovrstvé neuronové sítě s dopředným šířením obsahují jeden nebo více výstupních neuronů spojených se vstupní vrstvou váženými spojeními **W**. V nejjednodušším případě síť obsahuje pouze jeden vstup, bias a jeden výstup. Na obrázku si ukážeme 2 vstupy, bias a jeden výstup sítě. Pro zjednodušení vynecháme index výstupní vrstvy. Výstupem této sítě je výstup neuronu ve



Obrázek 4: Jednoduchá perceptronová síť

výstupní vrstvě. Tento výstup je vypočítán pomocí aktivační funkce tohoto neuronu:

$$y = \mathcal{F}\left(\sum_{i=1}^{2} w_i x_i + w_0\right). \tag{2.1}$$

Aktivační funkce $\mathcal{F}$ bývá lineární pro jednovrstvé neuronové sítě, ale může být i nelineární. My ale budeme nyní pouze uvažovat **Heavisidovu** nebo **Signum** fci:

$$\mathcal{F}(\mathbf{s}) = \begin{cases} 1 & \text{jestli } \mathbf{s} > 0 \\ -1 & \text{jinak} \end{cases} \tag{2.2}$$

Výstup závisí na celkovém vstupu **s** celé sítě. Jestliže celkový vstup bude kladný, tak výstupu přiřadíme hodnotu 1, v opačném případě -1. Nyní můžeme síť používat na klasifikační úlohy. K odlišení dvou různých tříd nám slouží oddělovací přímka. K nalezení takovéto oddělovací přímky mezi oběmi třídami nám pomůže rovnice celkového vstupu položená rovno nule, tedy:

$$w_1 x_1 + w_2 x_2 + w_0 = 0 \tag{2.3}$$

Tuto klasifikační přímku nazýváme **lineární diskriminační funkce**. Snadnou geometrickou reprezentaci této diskriminační funkce získáme tak, že osamostatníme x_2 na levé straně a dostaneme:

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}. \tag{2.4}$$

Nyní se budeme věnovat tématice nastavení vah v perceptronové síti. Popíšeme si dvě základní učební metody. První je tzv. **perceptronové učební pravidlo** a druhé se nazývá **delta pravidlo**, což je obdoba metody nejmenších čtverců. Obě metody jsou iterativní. U obou metod váhy nastavíme tak, že k nim v další epoše přičteme opravy jejich starých vah. Epocha je průchod všech prvků trénovací množiny neuronovou sítí. Stejně tak postupujeme s aktualizací biasu, máme tedy:

$$w_i(m+1) = w_i(m) + \Delta w_i(m) \tag{2.5}$$

$$w_0(m+1) = w_0(m) + \Delta w_0(m) \tag{2.6}$$

Otázkou je nyní, jak tedy správně nalezneme $\Delta w_i(m)$ a $\Delta w_0(m)$.

2.2 Perceptronovové pravidlo a konvergenční věta

Předpokládejme, že máme trénovací množinu vstupních vektorů a jejich daný výstup $t(\mathbf{x})$. Pro klasifikační úlohy nám postačí pouze hodnoty 1 a -1, proto použijeme jako aktivační funkci *sgn*. Chtěli bychom optimálně nastavit váhy $\mathbf{W}$, algoritmus vypadá následovně:

1. Nastav váhy pro spojení náhodně pomocí malých hodnot, někde v rozmezí $\langle -\epsilon, \epsilon \rangle$, kde ϵ je zhruba 0.5.

2. `while` $y \neq t(\mathbf{x})$ opakuj

   (a) Vyber vstupní vektor z trénovací množiny.

   (b) Změň všechny váhy w_i podle $\Delta w_i = t(\mathbf{x})x_i$ a bias podle

   $$\Delta w_0 = \begin{cases} 0 & \text{jestliže perceptron odpovídá správně} \\ t(\mathbf{x}) & \text{jinak} \end{cases} \tag{2.7}$$

3. Vrať optimální váhy w_i.

Obdobně se modifikuje i bias w_0.

Nyní si vyslovíme větu o konvergenci perceptronového učebního pravidla. Důkaz této věty je možný najít v [Kröse,Smagt] strana 25.

Věta 2.1. Nechť existují vážená spojení w^*, která jsou schopná provést transformaci $y = t(\mathbf{x})$, potom perceptronové učení konverguje k nějakému řešení v konečném počtu kroků pro jakékoliv počáteční nastavení vah.

Zpravidla existuje více řešení klasifikační úlohy, která správně klasifikují. Věta říká, že perceptronové učení najde jedno z nich.

2.3 Delta pravidlo

Delta pravidlo bylo navrženo v **ADALINE** (Adaptive Linear Element) síti. Pro jednovrstvé sítě s výstupním neuronem a lineární aktivační funkcí je výstup dán takto:

$$y = \sum_j w_j x_j + w_0. \tag{2.8}$$

Předpokládáme, že chceme síť natrénovat tak, aby co nejlépe odpovídala vztahu $y = t(\mathbf{x})$. Toho docílíme nastavením vah v této jednovrstvé síti. Každý trénovací prvek se liší od daného výstupu t^p takto $t^p - y^p$, kde y^p je výstup sítě. **Delta pravidlo** se snaží najít ideální váhy pomocí minimalizace celkové chybové funkce (penalizační funkce, cenového funkcionálu).

Chybová funkce E je v tomto případě součet čtverců rozdílu aktuálního a daného výstupu přes celou trénovací množinu, a je tedy dán vztahem

$$E = \frac{1}{P}\sum_p E^p, \text{kde } E^p = \frac{1}{2}(t^p - y^p)^2. \tag{2.9}$$

Snažíme se najít takové váhy, aby hodnota chybové funkce byla minimální. Nejprve zvolíme počáteční váhy, potom spočítáme derivaci chybové funkce E a vydáme se po směru záporného gradientu takto:

$$\Delta_p w_j = -\gamma \frac{\partial E^p}{\partial w_j}, \tag{2.10}$$

kde γ je parametr učení většinou v rozmezí $\langle 0, 1\rangle$, více o γ v kapitole (3.4.2).

Derivaci (2.9) podle vah si rozepíšeme takto:

$$\frac{\partial E^p}{\partial w_j} = \frac{\partial E^p}{\partial y^p}\frac{\partial y^p}{\partial w_j}. \tag{2.11}$$

Protože je výstup sítě (2.8) lineární platí

$$\frac{\partial y^p}{\partial w_j} = x_j, \tag{2.12}$$

a protože derivace (2.9) podle výstupu je

$$\frac{\partial E^p}{\partial y^p} = -(t^p - y^p), \tag{2.13}$$

dosazením (2.13) a (2.12) do (2.11) dostáváme

$$\frac{\partial E^p}{\partial w_j} = -x_j(t^p - y^p), \tag{2.14}$$

dosazením do (2.10) dostaneme finální podobu delta pravidla

$$\Delta_p w_j = \gamma x_j(t^p - y^p) := \gamma \delta^p x_j \tag{2.15}$$

3 Vícevrstvé sítě

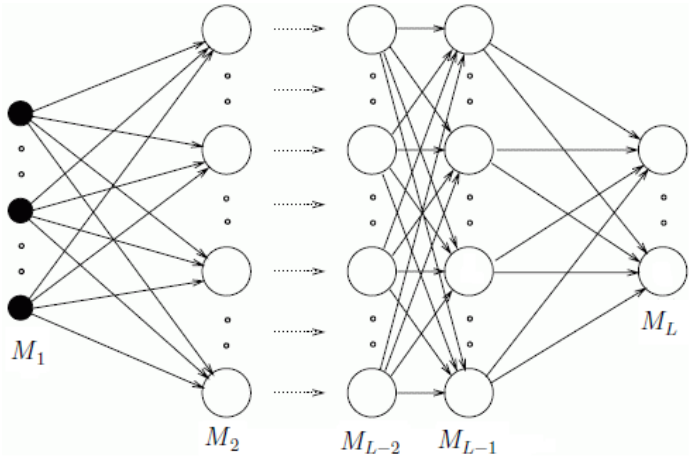
Za vícevrstvé sítě se označují sítě, které mají aspoň jednu vrstvu, kde neurony nemají externí vstup a výstup. To znamená, že tato vrstva není součást trénovací množiny a zároveň její výstup není výstup neuronové sítě. Vícevrstvé sítě vznikly pro potřeby řešit komplexnější lineárně neseparabilní problémy, tj. klasifikace, kde třídy nemůžeme oddělit pomocí přímek. U vícevrstvých sítí jsme nemohli použít klasické delta pravidlo, proto musel přijít na řadu nový algoritmus pro počítání derivací chybové funkce, který se nazývá Back Propagation.

3.1 Back Propagation

Back propagation, nebo také zobecněné delta pravidlo se používá pro výpočet derivací chybové funkce podle jednotlivých vah ve vícevrstvých neuronových sítích s dopředním šířením. Aby bylo možné algoritmus používat, musí aktivační funkce být diferencovatelná a pro klasifikační úlohy i nelineární. Můžeme použít například **logistickou funkci** (sigmoida) nebo **hyperbolický tangens**. V dalším textu budeme výhradně používat logistickou funkci vzhledem k jejím klasifikačním výhodám.

Než se vrhneme na samotný algoritmus Back Propagation, tak si ukážeme a popíšeme topologii vícevrstvé neuronové sítě na obrázku (5), která se skládá z několika vrstev. Vstupní vrstva, také označována jako trénovací množina, je označena plnými kroužky. Tato vrstva je spjata váženými spojeními (znázorněny čárami) do první skryté vrstvy, která je označena prázdnými kroužky. První skrytá vrstva posílá výstup do další skryté vrstvy. Takto to pokračuje, až výstup poslední skryté vrstvy je vstupem do výstupní vrstvy. Obecně bude naše síť mít L vrstvev. Počet neuronů v l-té vrstvě budeme značit $M_l, l = 1, \ldots, L$. Počet vstupů do sítě se rovná rozměru trénovacích prvků. Počet trénovacích prvků označme $P, p = 1, \ldots, P$.

K Back Propagation lze přistupovat dvěma způsoby, buď pomocí indexové notace nebo maticově. Nyní si odvodíme Back Propagation indexově, nicméně už zde zmíníme maticové označení pro dopředné šíření informace při aktivaci



Obrázek 5: Architektura vícevrtstvé neuronové sítě, $M_l, l = 1, \ldots, L$ značí počet neuronů ve vrstvách. Zdroj [Kröse,Smagt].

jednotlivých trénovacích prvků neuronovou sítí. Ukažme si nyní následující diagram neuronové sítě

$$\begin{pmatrix} x_0^1 \\ x_1^1 \\ \vdots \\ x_P^1 \end{pmatrix} \overset{\mathbf{W}^1}{\rightarrow} \begin{pmatrix} s_1^2 \\ \vdots \\ s_{M_1}^2 \end{pmatrix} \overset{\mathcal{F}}{\rightarrow} \begin{pmatrix} y_0^2 \\ y_1^2 \\ \vdots \\ y_{M_1}^2 \end{pmatrix} \overset{\mathbf{W}^2}{\rightarrow} \begin{pmatrix} s_1^3 \\ \vdots \\ s_{M_2}^3 \end{pmatrix} \overset{\mathcal{F}}{\rightarrow} \begin{pmatrix} y_0^3 \\ y_1^3 \\ \vdots \\ y_{M_2}^3 \end{pmatrix} \overset{\mathbf{W}^3}{\rightarrow} \ldots \overset{\mathcal{F}}{\rightarrow} \begin{pmatrix} y_1^L \\ \vdots \\ y_{M_L}^L \end{pmatrix}, \tag{3.1}$$

kde $\mathbf{x}^1 = (x_0^1, \ldots, x_P^1)^T$ je vstup z trénovací množiny do neuronové sítě. Aktivace první vrstvy je rovna vstupu sítě, tedy

$$\mathbf{y}^1 = \mathbf{x}^1. \tag{3.2}$$

Nechť $\mathbf{W}$ je matice vah mezi jednotlivými vrstvami sítě, l značí l-tou vrstvu sítě, $l = 1, \ldots, L$, rozměry matice $\mathbf{W}^l$ jsou $M_{l+1} \times (M_l + 1)$, kde M_l jsou počty neuronů v l-té vrstvě. Celkový vstup $\mathbf{s}_{l+1}$ do $l+1$ vrstvy se rovná součinu vah a aktivací z předešlé vrstvy,

$$\mathbf{s}^{l+1} = \mathbf{W}^l \mathbf{y}^l. \tag{3.3}$$

Je to také vstup aktivační funkce $\mathcal{F}$, která formuje výstup $\mathbf{y}_{l+1}$ následovně

$$\mathbf{y}^{l+1} = \mathcal{F}(\mathbf{s}^{l+1}). \tag{3.4}$$

Poté musíme přidat k $\mathbf{y}^{l+1}$ ještě bias (tj. $y_0^{l+1} = 1$) a dál pokračujeme obdobně.

Nyní přikročíme k algoritmu Back Propagation. Horní index p značí, že se vše dělá pro p-tý prvek trénovací množiny. Aktivační funkce je hladká funkce celkového vstupu, která je dána vztahem

$$y_k^p = \mathcal{F}(s_k^p), \tag{3.5}$$

kde

$$s_k^p = \sum_j w_{jk} y_j^p + w_0. \tag{3.6}$$

Změnu vah musíme provést obdobně jako v (2.10)

$$\Delta_p w_{jk} = -\gamma \frac{\partial E^p}{\partial w_{jk}}. \tag{3.7}$$

Chyba sítě E^p pro jeden vzor je definovaná jako u delta pravidla

$$E^p = \frac{1}{2M_L} \sum_{L=1}^{M_L} (t_L^p - y_L^p)^2, \tag{3.8}$$

kde t_L^p je daný výstup sítě, index L značí výstupní vrstvu, pro celkovou chybu všech vzorů platí

$$E = \frac{1}{P} \sum_p E^p. \tag{3.9}$$

Usilujeme o to, aby chyba (3.8) resp. (3.9) byla co nejmenší a hledáme tedy směr poklesu chyby E v závislosti na vahách sítě. Proto budeme počítat derivaci chyb podle vah sítě. Platí:

$$\frac{\partial E^p}{\partial w_{jk}} = \frac{\partial E^p}{\partial s_k^p} \frac{\partial s_k^p}{\partial w_{jk}}. \tag{3.10}$$

Podle rovnice (3.6) vídíme, že druhý člen je

$$\frac{\partial s_k^p}{\partial w_{jk}} = y_j^p. \tag{3.11}$$

Definujeme-li

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p} \tag{3.12}$$

dostaneme stejnou aktualizaci vah jako je u delta pravidla, tedy

$$\Delta_p w_{jk} = \gamma \delta_k^p y_j^p. \tag{3.13}$$

Nyní odvodíme rekurentní vzorec pro výpočet δ_k^p. Rozepíšeme si (3.12)

$$\delta_k^p = -\frac{\partial E^p}{\partial s_k^p} = -\frac{\partial E^p}{\partial y_k^p}\frac{\partial y_k^p}{\partial s_k^p}. \tag{3.14}$$

Druhý činitel spočítáme lehce podle (3.5)

$$\frac{\partial y_k^p}{\partial s_k^p} = \mathcal{F}'(s_k^p), \tag{3.15}$$

což je pouze derivace aktivační funkce. První činitel rovnice (3.14) spočítáme nejprve pro výstupní vrstvu L, budeme značit dolním indexem L. Po zderivování (3.8) dostaneme

$$\frac{\partial E^p}{\partial y_L^p} = (t_L^p - y_L^p). \tag{3.16}$$

Dosazením (3.16) a (3.15) do (3.14) dostaneme vzorec pro δ_L^p ve tvaru

$$\delta_L^p = \mathcal{F}'(s_L^p)(t_L^p - y_L^p). \tag{3.17}$$

Nyní se budeme zabývat výpočtem δ^l ze skryté vrstvy $l, l = 2, \ldots, L-1$. Chybová funkce v l-té vrstvě může být napsána jako funkce celkového vstupu do $l+1$ vrstvy, tedy $E^p = E^p(s_1^p, \ldots, s_{M_l})$. Označme $k = l+1$, při použití derivace složené funkce dostaneme

$$\begin{aligned}\frac{\partial E^p}{\partial y_l^p} &= \sum_{k=1}^{M_k} \frac{\partial E^p}{\partial s_k^p}\frac{\partial s_k^p}{\partial y_l^p} = \sum_{k=1}^{M_k} \frac{\partial E^p}{\partial s_k^p}\frac{\partial}{\partial y_l^p}\sum_{l=1}^{M_l} w_{lk} y_l^p = \sum_{k=1}^{M_k} \frac{\partial E^p}{\partial s_k^p} w_{lk} \\ &= -\sum_{k=1}^{M_k} \delta_k^p w_{lk},\end{aligned} \tag{3.18}$$

dosazením rovnice (3.18) do (3.14) dostaneme

$$\delta_l^p = \mathcal{F}'(s_l^p) \sum_{k=1}^{M_k} \delta_k^p w_{lk}. \tag{3.19}$$

Rovnice (3.13), (3.17) a (3.19) nám udávají rekurentní vzorce, jak vypočítat jednotlivá δ a také, jak nastavit jednotlivé váhy v rámci vícevrstvých neuronových sítí.

3.2 Pochopení algoritmu učení

Nyní si vysvětlíme, co vlastně znamenají jednotlivé vzorce v předešlé podkapitole.

Celý proces učení bychom mohli rozdělit na dvě fáze. První fází je **fáze aktivační**. Vezměme nějaký trénovací prvek, který prochází sítí, je ohodnocen pomocí vah, a celkový vstup je dosazen do aktivační funkce neuronu a ta spočítá aktivaci. Takto tento prvek prostupuje sítí až do výstupní vrstvy. Aktivační fáze však samostatně nestačí. Aby se síť mohla učit, je třeba vhodně aktivovat váhy.

Aktualizace vah se nazývá **adaptační fáze**. V adpaptační fázi nejprve porovnáne výstup sítě s daným výstupem a bude spočítána chyba podle (3.8), označíme ji E_o. Budeme usilovat o to, aby tato chyba byla co nejmenší. To se budeme snažit udělat tak, že změníme aktuální nastavení vah, aby byla chyba co nejmenší, a to pomocí aktualizačního vzorce vah

$$\Delta_p w_{jk} = \gamma \delta_k^p y_j^p. \tag{3.20}$$

Nyní musíme spočítat δ pro skryté vrstvy sítě. Za pomoci složené derivace podle celkového vstupu skryté vrstvy jsme schopni spočítat algoritmem Back Propagation δ_l z předešlé vrstvy. Toto δ_l se spočítá jako vážená suma δ_{l+1} ohodnocená váhami mezi skrytou a výstupní vrstvou a vynásobena aktivační funkcí celkového vstupu s_{l+1}^p, což nás dovedlo ke vzorečku (3.19). Aktualizace vah se nazývá adaptační fáze.

3.2.1 Učení s logistickou aktivační funkcí

Ve vícevrstvých neurononových sítích používáme nejčastěji jako aktivační funkci funkci logistickou. Nyni si popíšeme, jak se změní vzorce (3.17) a (3.19), pokud používáme logistickou funkci.

Jestliže se neuron nachází ve výstupní vrstvě, potom je δ_L definovaná takto

$$\delta_L^p = \mathcal{F}'(s_L^p)(t_L^p - y_L^p). \tag{3.21}$$

Aktivační funkce $\mathcal{F}$ je definovaná takto:

$$y^p = \mathcal{F}(s^p) = \frac{1}{1 + e^{-s^p}}. \tag{3.22}$$

Spočítejme nyní derivaci logistické funkce

$$\begin{aligned} \mathcal{F}'(s^p) &= \frac{\partial}{\partial s^p} \frac{1}{1 + e^{-s^p}} \\ &= \frac{1}{\left(1 + e^{-s^p}\right)^2} \left(e^{-s^p}\right) \\ &= \frac{1}{1 + e^{-s^p}} \frac{e^{-s^p}}{1 + e^{-s^p}} \\ &= y^p \left(1 - y^p\right). \end{aligned} \tag{3.23}$$

Jakmile máme derivaci sigmoidy, tak ji můžeme dosadit do vzorce (3.21) a dostaneme

$$\delta_L^p = (t_L^p - y_L^p) y_L^p \left(1 - y_L^p\right). \tag{3.24}$$

Chyba $\delta_l^p, l = 2, \ldots, L-1$ je určena rekurzivně a jako taková dáva předpis, jak spočítat jednotlivé chyby v různých skrytých vrstvách sítě. Derivace sigmoidy je nezávislá na vrstvě, kde ji provádíme, tudíž chyba skryté vrstvy může být napsána obdobně jako ve vzorci (3.19) takto:

$$\delta_l^p = \mathcal{F}'(s_l^p) \sum_{k=1}^{M_k} \delta_k^p w_{lk} = y_l^p \left(1 - y_l^p\right) \sum_{k=1}^{M_k} \delta_k^p w_{lk}, \tag{3.25}$$

kde $k = l + 1$.

3.3 Online a offline učení

Historicky se vyvinuly dva způsoby učení neuronové sítě, a to online a offline učení.

Online učení reprezentuje změnu vah po každém ukázaném prvku z trénovací množiny. Tento proces se může zdát nevýhodný vzhledem k častému nastavování vah, avšak pro nějaké speciální případy může tato metoda vést k uspokojivým výsledkům. U online učení se síť může naučit na počáteční prvky trénovací množiny a bude potom špatně určovat váhy pro prvky z „konce“ trénovací množiny. Tento problém se řeší volbou různých permutací prvků trénovací množiny pro každou epochu. Podle těchto permutací se potom ukazují jednotlivé prvky síti. Nedostatky online učení řeší offline učení.

Offline učení nenastavuje jednotlivé váhy po každém průchodu prvku trénovací množiny, ale až po projítí celé trénovací množiny neuronovou sítí. Nejprve pro každý prvek trénovací množiny spočítáme gradient chybové funkce podle jednotlivých vah. Výsledný gradient chybové funkce jsme získali jako součet gradientů dílčích chybových funkcí E^p, matematicky to lze zapsat takto:

$$d_m = \sum_{p=1}^{P} \left(\frac{\partial E^p}{\partial w} \right)_m , \tag{3.26}$$

kde m je epocha. Epocha je jeden průchod celé trénovací množiny neuronovou sítí.

3.4 Metody prvního řádu

V této části si povíme, jak minimalizovat chybovou funkci pomocí metod prvního řádu. Metody prvního řádu používají při aktualizování vah pouze první derivace. Mezi metody prvního řádu patří tzv. adaptivní a neadaptivní metody. **Adaptivní metody** jsou takové, kde se během aktualizace mění kromě vah i parametr učení podle jistých pravidel. **Neadaptivní metody** mají konstantní parametr učení. Mezi neadaptivní metody patří například Gradient Descent.

V následujících kapitolách si popíšeme dvě adaptivní metody, to jsou Delta-Bar-Delta metoda a metoda největšího spádu.

Dříve než začneme s popisem různých metod, ukážeme si několik ukončovacích kritérií, která mohou být společná pro tyto metody.

3.4.1 Ukončovací kritéria

Pokud minimalizujeme chybovou funkci, tak se můžeme ptát kdy a jak ukončit minimalizační algoritmus. Ukončení může být různé na základě toho, jak velkou chybu nebo velikost kroku chceme tolerovat. Typů kritérií může být hned několik. Některá si zde ukážeme.

$$\left|\frac{E(w_{m-1})-E(w_m)}{E(w_{m-1})}\right| \leq Q \tag{3.27}$$

$$\gamma \leq \gamma_{min}, \tag{3.28}$$

$$E \leq E_{min}. \tag{3.29}$$

První kritérium ukončí algoritmus, pokud je odhad relativní chyby menší než zadaná relativní přesnost Q. Druhé kritérium ukončí výpočet, pokud je parametr učení menší, než námi zvolená hodnota γ_{min}. Toto kritérium je vhodné pouze pro adaptivní metody. Poslední kritérium ukončí algoritmus, pokud je celková chyba sítě menší než E_{min}, což je volitelná tolerance velikosti chyby sítě.

Tyao kritéria by měla být vesměs společná pro následující metody.

3.4.2 Parametr učení a moment sítě

Parametr učení (learning rate) vyžaduje, aby změna vah byla úměrná ku $\partial E/\partial w$. Parametr učení v podstatě ukazuje, jak daleko se má postoupit ve směru záporného gradientu. V optimalizaci se parametr učení nazývá **délka kroku**.

Snažíme se volit parametr učení co největší, ale takový aby nedocházelo k oscilaci řešení. Často se volí γ v intervalu $\langle 0, 1\rangle$, nicméně to není pravidlem. Nastavení vah tedy probíhá podle vzorce:

$$\begin{aligned} w_{m+1} &= w_m + \Delta w_m \\ \Delta w_m &= -\gamma d_m, \end{aligned} \tag{3.30}$$

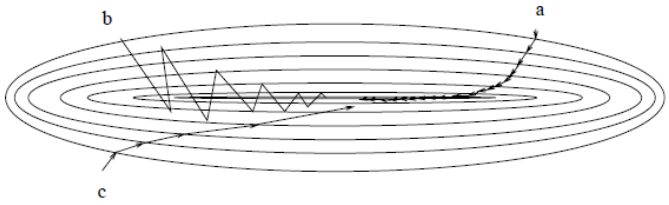
kde znaménko mínus reprezentuje sestup a γ reprezentuje délku kroku ve směru d_m.

Pro různou volbu parametru učení existuje několik nežádoucích scénářů, jak se bude chovat výsledná chyba sítě. Zvolíme-li příliš malé γ nemusíme k minimu dospět v rozumném počtu epoch. Dalším problémem může být příliš velký krok, kde můžeme minimum přeskočit a může tak docházet k oscilaci. Přidáním dalšího parametru do rovnice (3.30), můžeme dopad oscilace zmírnit jejím vyhlazením. Nový parametr nazýváme moment sítě a značíme μ.

Moment sítě je v podstatě vyhlazovací parametr předešlých vah. Je to obdoba relaxačního parametru u algoritmů pro numerické řešení soustav rovnic. Matematicky moment zapisujeme takto:

$$\Delta w_m = \mu \Delta w_{m-1} - (1-\mu)\gamma d_m. \tag{3.31}$$

Jak je z rovnice (3.31) vidět, tak by měl být moment μ z intervalu $\langle 0, 1\rangle$. Odtud se odvíjí, jakému členu přiřadíme jakou důležitost. Pokud zvolíme μ rovno 0, tak dostáváme vzorec (3.30). Pokud za μ dosadíme 1, tak změna vah závisí pouze na předešlých vahách. Neexistuje přesné pravidlo, které by nám uvedlo, jak velké bychom μ měli volit. Pro ilustraci uvádíme různé případy volby μ, γ na následujícícm obrázku:



Obrázek 6: Spád ve váhovém prostoru *a*) malý učební poměr *b*) velký učební poměr se sklonem k oscilaci *c*) učební poměr s přidaným momentem. Zdroj: [Kröse,Smagt]

3.4.3 Metoda Gradient Descent

Nejprimitivnější metodou na minimalizaci je přímo vzorec (3.30), který představuje metodu v teorii neuronových sítí nazývanou **Gradient Descent**. Para-

metr γ je pevně dán a během výpočtu se nemění. Tato metoda se může lišit přidáním momentu sítě (3.31).

3.4.4 Delta-Bar-Delta metoda

U Gradient Descent jsme měli pevně stanovený parametr učení pro všechny váhy, čímž jsme nacházeli minimum buď příliš pomalu nebo, při volbě velkého parametru učení, jsme minimum přeskočili. Tento problém řeší lépe metoda zvaná **Delta-Bar-Delta**.

Metoda spočívá v indiviuálním nastavení vah díky adaptaci γ na základě předešlých derivací jednotlivých vah.

Tato metoda v podstatě vyžaduje porovnání znaménka aktuální chyby gradientu d_m a jejich předešlých chyb. Nedávná historie směru, ve kterém se chyby snižovaly až do epochy číslo m je vyjádřena pomocí funkce f_m, ta je definovaná takto:

$$f_m = \theta f_{m-1} + (1 - \theta) d_{m-1}, \tag{3.32}$$

kde θ je váhový parametr minulých derivací, $1 - \theta$ je váha poslední derivace, tyto váhy spolu určují, jaký vliv budou mít nejaktuálnější gradienty na funkci f_m, tj. na směr ve kterém se chyba v nedávné historii snižovala.

Jestliže θ je rovna 0, potom je f_m závislé pouze na posledním gradientu, předešlé gradienty nemají žádný efekt. Jestliže θ je 1 je f_m vypočteno pouze z předešlého gradientu a jeho prostřednictvím závisí i na předešlých gradientech. Abychom poznali, zda-li f_m ubíhá ve stejném směru jako d_m, musíme spolu tyto členy vynásobit.

Jestliže $f_m d_m$ je kladné, což znamená stejné směry gradientů, potom zvětšujeme učební poměr.

Jestliže $f_m d_m$ je záporné, což znamená opačné směry gradientů, někde po cestě jsme přeskočili minimum, potom zmenšujeme učební poměr.

Tyto dvě poslední skutečnosti mohou být vyjádřeny takto:

$$\gamma_m = \begin{cases} \gamma_{m-1} + \kappa & \text{pro} f_m d_m > 0 \\ \gamma_{m-1} * \varphi & \text{pro} f_m d_m \leq 0, \end{cases} \tag{3.33}$$

kde parametry κ a φ jsou mezi 0 a 1. Když už máme vyřešenou otázku adaptace parametru učení, tak můžeme adaptovat váhy tímto způsobem:

$$w_m = w_{m-1} - \gamma_m d_m. \tag{3.34}$$

nebo s přidaným momentem sítě

$$w_m = w_{m-1} + \mu \Delta w_{m-1} - (1-\mu)\gamma_m d_m. \tag{3.35}$$

Jako ukončovací kritéria můžeme použít (3.27), (3.29), (3.28).

3.4.5 Metoda nejstrmějšího spádu (Steepest Descent Method)

V metodě **nejstrmějšího spádu** se chyba zmenšuje podél záporného gradientu jako u Gradient Descent a Delta-Bar-Delta metody, akorát s tím rozdílem, že γ, které je stejné pro všechny váhy, se mění během učení sítě v takzvané zkušební epoše.

Nastavení γ ve zkušební epoše funguje následovně. Nejprve si zvolíme počáteční hodnotu γ_0 a potom je γ_0 zdvojnásobeno. Je spočítána celková chyba sítě a pokud se chyba zmenšila, tak aktualizujeme váhy podle (3.30) nebo (3.31) s novým parametrem učení. Pokud se ovšem chyba nezmenšila, tak se vrátíme k původnímu γ_0, a to zmenšíme o polovinu, pokud se ani tak chyba nezmenší, pokračujeme se zmenšováním, dokud se chyba nezmenší. Poté skončí tzv. zkušební epocha a můžeme aktualizovat váhy jako obvykle podle (3.30) nebo (3.31) s naším novým γ.

Při následující epoše se proces opakuje, tedy napřed zdvojnásobit γ, prozkoumat chybu, jestli se zmenšila, pokud se nezmenšila, tak se vrátíme se k původnímu γ a zmenšujeme γ o polovinu, dokud se celková chyba nezmenší.

Zbývá otázka, jako ukončit tuto metodu. Dají se použít tyto kritéria (3.27), (3.29), (3.28).

3.5 Metody druhého řádu

V metodách druhého řádu nevyužíváme pouze sklonu, tj. derivace funkce, ale i druhou derivaci v aktuálním bodě

Všechny metody prvního a druhého řádu jsou iterativní a pokouší se minimalizovat chybu, která vypadala následovně

$$E = \frac{1}{P}\sum_{p} E^p = \frac{1}{2P}\sum_{p=1}^{P}(t^p - y^p)^2. \tag{3.36}$$

Jakmile jsme měli spočítané derivace chybové funkce pomocí Back Propagation, tak jsme začali aktualizovat váhy ve směru záporného gradientu pomocí metody Gradient Descent pomocí vzorce (3.30). U metod druhého řádu budeme používat podobnou formuli

$$w_m = w_{m-1} - \gamma \mathbf{R} d_m, \tag{3.37}$$

kde m je epocha, γ parametr učení a d_m je součet gradientů přes trénovací množinu. Jediný nový parametr oproti (3.30) je $\mathbf{R}$. Tento parametr má několik variant pro různé metody druhého řádu a zahrnuje i metody 1. řádu. Pokud například dosadíme jedničku za $\mathbf{R}$ dostaneme Gradient Descent metodu. Nyní se pokusíme odvodit podobu tohoto parametru $\mathbf{R}$.

3.5.1 Levenbergova - Marquardtova metoda

Levenbergova - Marquardtova metoda, dále jen LM metoda, je komplexní metoda druhého řádu, pro řešení úloh vznikající při nelineární metodě nejmenších čtverců, tj. pro úlohy ve tvaru:

$$\min_{\mathbf{x}} \left(f(\mathbf{x}) - \mathbf{y}\right)^2 \tag{3.38}$$

Tato metoda využívá jisté aproximace Hessiánu a přidává k němu stabilizační parametr λ z důvodu možné singularity aproximace Hessiánu, λ je volitelný parametr.

Nyní si podrobně ukážeme, jak najdeme $\mathbf{R}$. Mějme chybovou funkci

$$E = \frac{1}{P}\sum_{p} E^p = \frac{1}{2P}\sum_{p}(t^p - y^p(\mathbf{x}, \mathbf{w}))^2, \tag{3.39}$$

kde t^p je požadovaný výstup, y^p výstup sítě pro p-tý trénovací prvek.

Jeden způsob, jak minimalizovat funkci (3.39) je aproximovat $y^p(\mathbf{x}, \mathbf{w}) \approx \hat{y}^p(\mathbf{x}, \mathbf{w})$ jako lineární funkci $\mathbf{w}$. Definujme aproximaci takto:

$$\hat{y}^p(\mathbf{x}, \mathbf{w}) = y^p(\mathbf{x}, \mathbf{w}_0) + (\mathbf{w} - \mathbf{w}_0)^T \frac{\partial y^p}{\partial w_i^p}. \tag{3.40}$$

Zderivujme E^p a dostaneme

$$\frac{\partial E^p(\mathbf{w})}{\partial w_i^p} = (t^p - \hat{y}^p(\mathbf{x}, \mathbf{w})) \left(-\frac{\partial y^p}{\partial w_j^p} \right). \tag{3.41}$$

Nyní dosadíme (3.40) do (3.41) a dostaneme

$$\frac{\partial E^p(\mathbf{w})}{\partial w_i^p} = \left(t^p - y^p(\mathbf{x}, \mathbf{w}_0) - (\mathbf{w} - \mathbf{w}_0)^T \frac{\partial y^p}{\partial w_i^p} \right) \left(-\frac{\partial y^p}{\partial w_j^p} \right) \tag{3.42}$$

Definujme derivaci

$$\mathbf{d} = (t^p - y^p(\mathbf{x}, \mathbf{w}_0)) \frac{\partial y^p}{\partial w_j^p} \tag{3.43}$$

a $\mathbf{H}$ aproximaci Hessiánu následovně

$$\mathbf{H} = \frac{\partial y^p}{\partial w_i^p} \frac{\partial y^p}{\partial w_j^p}. \tag{3.44}$$

Nyní přepíšeme (3.42) pomcí $\mathbf{H}$ a $\mathbf{d}$ a položíme rovno 0

$$\frac{\partial E^p(\mathbf{w})}{\partial w_i^p} = \mathbf{H}(\mathbf{w} - \mathbf{w}_0) + 2\mathbf{d} = 0 \tag{3.45}$$

Z tohoto lehce odvodíme vztah pro aktualizaci vah metodou LM

$$w_m = w_{m-1} - \mathbf{H}^{-1}\mathbf{d}, \tag{3.46}$$

kde m je číslo epochy.

Pro ještě lepší výsledky této metody přidáme další člen k $\mathbf{H}$, dostaneme symbolicky

$$\mathbf{R} = (\mathbf{H} + \lambda \mathbf{I})^{-1}. \tag{3.47}$$

Vzorec pro změnu gradientu je ve tvaru

$$\Delta w_m = -d_m \left(\mathbf{H} + \lambda \mathbf{I}\right)^{-1}, \tag{3.48}$$

kde d_m je suma gradientů chyby. Celkový vzorec pro nastavení vah pomocí LM metody bude tedy vypadat následovně

$$w_m = w_{m-1} + \Delta w_m = w_{m-1} - d_m \left(\mathbf{H} + \lambda \mathbf{I}\right)^{-1}, \tag{3.49}$$

kde $\mathbf{H}$ je aproximace Hessiánu. U LM je λ vybráno nejprve ručně. Pokud se celková chyba zmenšuje, položíme nové $\lambda/10$. Pokud se nám chyba zvětšuje, tak nové lambda bude 10λ. LM algoritmus bychom mohli popsat následujícím způsobem

Algoritmus LM metody

`while` není splněno ukončovací kritérium

(a) Aktualizovat váhy podle

$$w_m = w_{m-1} + \Delta w_m = w_{m-1} - d_m \left(\mathbf{H} + e^{\lambda} \mathbf{I}\right)^{-1}.$$

(b) Vyčíslit celkovou chybu s novými váhami.

(c) Jestliže se chyba zvedla, vrátíme váhy do předešlého stavu a λ vynásobíme 10.

(d) Jestliže se chyba snížila, snižíme λ 10krát..

Ukončovací kritéria by mohla být tyto (3.27), (3.29) nebo

$$\lambda > 10\Delta\lambda + \mu_{max}\mathbf{H}, \tag{3.50}$$

kde $\mu_{max}(\mathbf{H})$ je největší vlastní číslo matice $\mathbf{H}$. Kritéria by mohla být použita samostatně, ale můžeme je použít i společně. LM metoda je spojník mezi dvěma metodami v závislosti na parametru λ. Pokud je λ malé podobá se LM metodě zvané Gauss-Newtonova(GN), což je vlastně LM, akorát bez stabilizačního parametru. V GN se navíc adaptivně počítá parametr učení. Pokud je λ velké, potom se LM podobá metodě Steepest Descent. Další metodu 2.řádu si uvedeme ve spojení s novým přístupem k minimalizaci chyby za pomoci nové chybové funkce, který získáme pomocí metody maximální věrohodnosti.

3.6 Minimalizace nové chybové funkce

V této kapitole si ukážeme minimalizaci nové chybové funkce neuronové sítě založenou na logistické regresi a metodě maximální věrohodnosti. Důvodem proč se minimalizuje nová chybová funkce je ten, že chyba (3.36) není konvexní a byla v podstatě vybrána z důvodu, že se hodila pro lineární aktivační funkci. Nový funkcionál bude mít pro minimalizaci lepší vlastnosti.

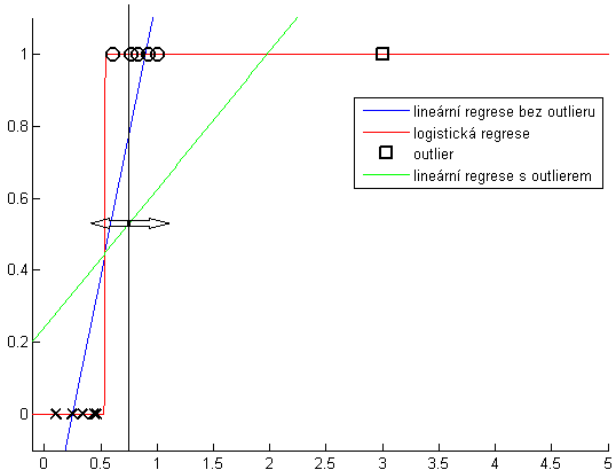
V této kapitole si ukážeme motivaci, proč byla nová funkce vybrána. V krátkosti si představíme lineární regresi, konkrétně metodu nejmenších čtverců, a logistickou regresi na krátkém příkladu.

Lineární regrese je v podstatě aproximační metoda, při které se snažíme danými daty proložit přímku tak, aby součet druhých mocnin odchylek různých bodů byl minimální.

Logistická regrese je zobecněný model. Logistická regrese se nám bude hodit, protože nabývá hodnot mezi 0, což znamená, že jev nenastal, a 1, jev nastal. V našem případě bude náhodný jev znamenat náležení do jisté třídy(kategorie).

3.6.1 Motivace

Mějme množinu dat, kde křížky označují nezhoubné nádory, přiřaďme jim hodnotu 0, a kolečka označují zhoubné nádory s hodnotou 1. Na ose x jsou vyznačeny různé velikosti nádorů. Nejprve jsme chtěli proložit daty přímku pomocí lineární regrese, a to bez jednoho prvku, který je označen čtverečkem, dále o něm budeme mluvit jako o **odlehlém pozorování** (Outlierovi). V obrázku (7) je lineární regrese znázorněna plnou modrou čárou. Poté jsme proložili přímku daty i s odlehlým pozorováním, zelená barva. Podle obrázku (7) vidíme, že lineární regrese není vhodná pro tento typ klasifikace. Oddělila totiž špatně jednotlivé třídy nádorů. Šipky na tomto obrázku ukazují od jaké x-ové souřadnice lineární regrese klasifikuje. Z tohoto důvodu si pomůžeme logistickou regresí, která není citlivá na odlehlá pozorování a dobře se hodí na klasifikační úlohy. Nabývá totiž hodnot mezi 0 a 1. Na obrázku (7) můžeme vidět logistickou regresi vyobrazenou červeně.



Obrázek 7: Klasifikace nádorů pomocí lineární a logistické regrese

3.6.2 Interpretace logistické funkce

Nechť x_j je prvek trénovací množiny, výstup $y_j = 0$ z logistické regrese znamená, že prvek nenaleží do třídy a podobně $y_j = 1$ znamená, že prvek náleží do třídy. Mějme dány vektory $\mathbf{x}_1, \ldots, \mathbf{x}_N$, kde každý může být $\mathbf{x}_i = (x_1, \ldots, x_M)^T$, $\mathbf{x} = (\mathbf{x}_1^T, \ldots, \mathbf{x}_N^T)$ a $\boldsymbol{\Theta}$ je matice parametrů logistické regrese. Logistická funkce má následující předpis:

$$h_\Theta(\mathbf{x}) = \frac{1}{1 + exp(-\boldsymbol{\Theta}^T\mathbf{x})}, \tag{3.51}$$

kde $h_{\boldsymbol{\Theta}}(\mathbf{x})$ je pravděpodobnost, že bude prvek x klasifikován jako pozitivní pro dané hodnoty $\boldsymbol{\Theta}$. Hranice rozhodování budiž:

$$h_\Theta(x) > \frac{1}{2} \Rightarrow \quad \text{prvek označen jako pozitivní.} \tag{3.52}$$

Otázka zní, jak najít optimální hodnoty pro $\boldsymbol{\Theta}$.

3.6.3 Odvození chybové funkce

Pro nalezení použijeme tzv. metodu maximální věrohodnosti[1], která je známá při odhadech parametrů ze statistiky.

Mějme dány vektory $\mathbf{x}_i$, chceme nastavit $\boldsymbol{\Theta}$ tak, abychom maximalizovali pravděpodobnost, že $h_{\boldsymbol{\Theta}}(\mathbf{x}_i)$ bude rovno y_i pro všechna i. Pro $y_i = 1$ chceme maximalizovat hodnoty $h_{\boldsymbol{\Theta}}(\mathbf{x}_i)$ a pro hodnoty $y_i = 0$ chceme minimalizovat hodnoty $h_{\boldsymbol{\Theta}}(\mathbf{x}_i)$. Jednotlivé náhodné veličiny jsou navzájem nezávislé, proto můžeme psát

$$L\left(\boldsymbol{\Theta}\right) = h_{\boldsymbol{\Theta}}(\mathbf{x}_1)h_{\boldsymbol{\Theta}}(\mathbf{x}_2)\ldots h_{\boldsymbol{\Theta}}(\mathbf{x}_k)(1-h_{\boldsymbol{\Theta}}(\mathbf{x}_{k+1}))(1-h_{\boldsymbol{\Theta}}(\mathbf{x}_{k+2}))\ldots(1-h_{\boldsymbol{\Theta}}(\mathbf{x}_N)), \tag{3.53}$$

kde pro prvních k členů je klasifikace rovna jedné a pro zbytek je roven nule. Tento součin bychom rádi minimalizovali, ale jistě se nám bude lépe minimalizovat součet, proto (3.53) zlogaritmujeme, dostaneme

$$\ln L(\boldsymbol{\Theta}) = \sum_{i=1}^{k} \ln h_{\boldsymbol{\Theta}}(\mathbf{x}_i) + \sum_{i=k+1}^{N} \ln(1-h_{\boldsymbol{\Theta}}(\mathbf{x}_i)). \tag{3.54}$$

Vzhledem k tomu, že chceme funkcionál (3.54) minimalizovat, tak změníme znaménko. Dále provedeme normalizaci tak, že funkcionál (3.54) podělíme N a převedeme na jednu sumu takto:

$$J(\boldsymbol{\Theta}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \ln h_{\boldsymbol{\Theta}}(\mathbf{x}_i) + (1-y_i)\ln(1-h_{\boldsymbol{\Theta}}(\mathbf{x}_i))\right]. \tag{3.55}$$

Vzorec (3.55) zderivujeme podle $\boldsymbol{\Theta}$ a dostaneme

$$\frac{\partial J(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}_j} = \frac{1}{N}\sum_{i=1}^{N}\left(h_{\boldsymbol{\Theta}}(\mathbf{x}_i) - y_i\right)x_i). \tag{3.56}$$

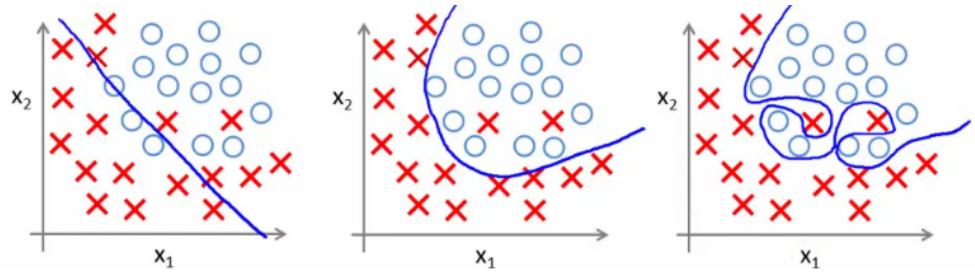
Při minimalizaci může docházet k různým problémům, například k overfittingu. Při overfittingu jsou výborně modelována námi zadaná data, ale získaný model nedokáže dobře zobecňovat. Tento problém řešíme tak, že budeme penalizovat

[1] `http://cs.wikipedia.org/wiki/Metoda_maxim%C3%A1ln%C3%AD_v%C4%9Brohodnosti`

jednotlivé $\boldsymbol{\Theta}_1, \ldots, \boldsymbol{\Theta}_N$. Přidáním regulárních členů tak dostaneme nový cenový funkcionál:

$$J(\boldsymbol{\Theta}) = -\frac{1}{N}\left\{\sum_{i=1}^{N}\left[y_i \ln h_{\boldsymbol{\Theta}}(\mathbf{x}_i) + (1-y_i)\ln(1-h_{\boldsymbol{\Theta}}(\mathbf{x}_i))\right] + \frac{\lambda}{2}\sum_{i=1}^{N}(\boldsymbol{\Theta}_i)^2\right\}. \tag{3.57}$$

Výsledná klasifikace bude záviset na správné volbě parametru λ. Pro malá λ dochází k overfittingu a pro velká λ dochází k underfittingu. Naproti overfitingu může nastat opačný problém underfitting, který dobře neaproximuje data, a proto také nepopisuje dobře námi modelovanou situaci. Jak by to mohlo vypadat pro různá nastavení λ se můžeme podívat v následujícím obrázku. V následující ka-



Obrázek 8: Na obrázku vlevo je zobrazen underfitting, uprostřed je ideální nastavení hodnoty *lambda* a vpravo je zobrazen overfitting. Zdroj: http://holehouse.org/mlclass/07_Regularization.html

pitole si ukážeme funkcionál, který se minimalizuje pro neuronové sítě, jelikož každý neuron v síti je vlastně samostatná logistická regrese, tak bude tento nový funkcionál značně podobný (3.57).

3.7 Neuronové sítě s logistickou regresí

V této podkapitole si ukážeme nový přístup k neuronovým sítím spolu s logistickou regresí a sofistikovaným algoritmem pro nastavení vah při minimalizaci naší nové chybové funkce.

3.7.1 Značení

Mějme nejprve logistickou funkci

$$\mathcal{F}(s) = \frac{1}{1+e^{-s}}, \tag{3.58}$$

kde s je celkový vstup nebo také skalární součin $w_i x_i$, kde $\mathbf{x} = (x_0, \ldots, x_P)$ je vstupní vektor a $x_0 = 1$ a $\mathbf{w} = (w_0, \ldots, w_P)$ jsou váhy, $w_0 = \theta$ je bias.

Značení, které si nyní ukážeme, jsme již představili v kapitole (3.1), pro pořádek ho zde ukážeme ještě jednou. Ukažme si nyní následující diagram neuronové sítě

$$\begin{pmatrix} x_0^1 \\ x_1^1 \\ \vdots \\ x_P^1 \end{pmatrix} \overset{\mathbf{w}^1}{\to} \begin{pmatrix} s_1^2 \\ \vdots \\ s_{M_1}^2 \end{pmatrix} \overset{\mathcal{F}}{\to} \begin{pmatrix} y_0^2 \\ y_1^2 \\ \vdots \\ y_{M_1}^2 \end{pmatrix} \overset{\mathbf{w}^2}{\to} \begin{pmatrix} s_1^3 \\ \vdots \\ s_{M_2}^3 \end{pmatrix} \overset{\mathcal{F}}{\to} \begin{pmatrix} y_0^3 \\ y_1^3 \\ \vdots \\ y_{M_2}^3 \end{pmatrix} \overset{\mathbf{w}^3}{\to} \ldots \overset{\mathcal{F}}{\to} \begin{pmatrix} y_1^L \\ \vdots \\ y_{M_L}^L \end{pmatrix}, \tag{3.59}$$

kde $\mathbf{x}^1 = (x_0^1, \ldots, x_P^1)^T$ je vstup z trénovací množiny do neuronové sítě. Aktivace první vrstvy je rovna vstupu sítě, tedy

$$\mathbf{y}^1 = \mathbf{x}^1. \tag{3.60}$$

Nechť $\mathbf{W}$ je matice vah mezi jednotlivými vrstvami sítě, l značí l-tou vrstvu sítě, $l = 1, \ldots, L$, rozměry matice $\mathbf{W}^l$ jsou $M_{l+1} \times (M_l + 1)$, kde M_l jsou počty neuronů v l-té vrstvě. Celkový vstup do $l+1$ vrstvy $\mathbf{s}_{l+1}$ se rovná součinu vah a aktivací z předešlé vrstvy, takto

$$\mathbf{s}^{l+1} = \mathbf{W}^l \mathbf{y}^l. \tag{3.61}$$

Je to také vstup aktivační funkce $\mathcal{F}$, která formuje výstup $\mathbf{y}_{l+1}$ následovně

$$\mathbf{y}^{l+1} = \mathcal{F}(\mathbf{s}^{l+1}). \tag{3.62}$$

Poté musíme přidat k $\mathbf{y}^{l+1}$ ještě bias (tj. $y_0^{l+1} = 1$) a dál pokračujeme obdobně.

3.7.2 Back Propagation s logistickou regresí

Nyní přikročíme k minimalizaci nové celkové chyby podobné cenovému funkcionálu (3.57) u logistické regrese

$$J(\mathbf{W}) = -\frac{1}{P}\sum_{p=1}^{P}\sum_{k=1}^{M_l}\left\{t_k^p \log\left[h_{\mathbf{W}}\left(\mathbf{x}^p\right)\right]_k + \left(1-t_k^p\right)\log\left[1-h_{\mathbf{W}}\left(\mathbf{x}^p\right)\right]_k\right\}$$
$$+\frac{\lambda}{2P}\sum_{l=1}^{L-1}\sum_{i=1}^{M_l}\sum_{j=1}^{M_{l+1}}\left(\mathbf{W}_{ij}^l\right)^2, \tag{3.63}$$

kde index k značí k tý výstupní neuron, index p předtavuje p-tý prvek trénovací množiny. Zbylé značení bylo popsáno již dříve.

Nyní potřebujeme efektivně spočítat $\partial J(\mathbf{W})/\partial \mathbf{w}_{ij}$. Tohle je ovšem přesně to, co dělá algoritmus Back Propagation pomocí δ, což jsou vlastně chyby na jednotlivých neuronech j ve vrstvách l. Mějme j-tý neuron ve výstupní vrstvě. Výstupní chyba sítě v l-té vrstvě pro dvojici $(\mathbf{x}, \mathbf{t}(\mathbf{x}))$, kde $\mathbf{t}(\mathbf{x})$ je daný výstup pro vstup $\mathbf{x}$. Pro j-tý neuron můžeme psát výstupní chybu takto:

$$\delta_j^L = y_j^L - t_j, \tag{3.64}$$

což lehce můžeme zapsat vektorově

$$\delta^L = \mathbf{y}^L - \mathbf{t} \tag{3.65}$$

Definujme

$$\delta^l = \left(\mathbf{W}^l\right)^T \delta^{l+1} .* \mathcal{F}'\left(\mathbf{s}^l\right), \tag{3.66}$$

kde $.*$ značí násobení odpovídajících si prvků, δ^{l+1} je delta z následující vrstvy. Derivaci $\mathcal{F}'\left(\mathbf{s}^l\right)$ lehce spočítáme a dostaneme

$$\mathcal{F}'\left(\mathbf{s}^l\right) = \mathbf{y}^l .* \left(\mathbf{1} - \mathbf{y}^l\right), \tag{3.67}$$

a dosadíme ji do (3.66)

$$\delta^l = \left(\mathbf{W}^l\right)^T \delta^{l+1} .* \mathbf{y}^l .* \left(\mathbf{1} - \mathbf{y}^l\right). \tag{3.68}$$

Nyní zderivujeme $J(\mathbf{W})$ obdobným způsobem jako v (3.10) a dostaneme

$$\frac{\partial E(w_{ij})}{\partial w_{ij}} = y_j^l \delta_i^{l+1} \tag{3.69}$$

pro $\lambda = 0$. Pro $\lambda > 0$ si uvedeme následující algoritmus.

3.7.3 Algoritmus Back Propagation

Postup algoritmu Back Propagation si shrneme následovně.

1. mějme trénovací množinu $\left\{(x^1, t^1), \ldots, \left(x^P, t^P\right)\right\}$
2. nastav $\Lambda_{ij}^l = 0 \quad \forall \quad i, j, l$
3. `for` $p = 1$ do P

   nastav výstup vstupní vrstvy takto $y^1 = x^p$

   spočítej podle (3.59) všechny y^l pro $l = 2, 3, \ldots, L$

   spočítej $\delta^L = \mathbf{y}^L - \mathbf{t}^p$

   spočítej $\delta^{L-1}, \delta^{L-1}, \ldots, \delta^2$, pro $l = L - 1 : (-1) : 2$

   polož $\Lambda_{ij}^l := \Lambda_{ij}^l + y_j^l \delta_i^{l+1}$
4. `end`
5. $D_{ij}^l := \frac{1}{P}\Lambda_{ij}^l + \frac{1}{P}\lambda W_{ij}^l$ jestliže $j \neq 0$
6. $D_{ij}^l := \frac{1}{P}\Lambda_{ij}^l$ jestliže $j = 0$

3.7.4 Gradent Descent

Otázkou zůstavá, jakou minimalizační metodu na rovnici (3.63) můžeme použít. Lze zvolit metody prvního řádu, které jsme si popsali již dříve, například Gradient Descent odpovídá aktualizaci vah podle vzoru

$$\mathbf{W}_m = \mathbf{W}_{m-1} - \gamma \mathbf{D}_m, \tag{3.70}$$

kde index m značí epochu, γ parametr učení. V předešlé kapitole jsme definovali D_{ij}^{l}, kde i, j byly řádky a sloupce a l značilo vrstvu, ve které se gradienty chybové funkce (3.63) počítaly. V této metodě budeme pro jednoduchost tyto matice gradientu značit pouze $\mathbf{D}_m$. Metody prvního řádu nejsou příliš efetivní, proto si popíšeme metodu BFGS.

3.7.5 BFGS metoda

BFGS (Broyden, Fletcher,Goldfarb, Shanno), která patří mezi quasi Newtonovské metody a jako taková je to metoda 2. řádu, která používá jistou aproximaci Hessiánu. Tato metoda se používá v optimalizaci při hledání minima vícerozměrných nelineárních funkcí. Hodí se tudíž na minimalizaci (3.63).

Metodu si popíšeme v označení odpovídající řešení úlohy

$$\min_{\mathbf{B}} f(\mathbf{x}). \tag{3.71}$$

$f_k, \nabla f_k, \nabla^2 f_k$ je chybová funkce resp. první derivace resp. druhá derivace v k-té iteraci, $\mathbf{B}_k$ je aproximace Hessiánu.

V quasi Newtonovských metodách hledáme nový směr jako $p_k = \mathbf{B}_k^{-1} \nabla f_k$, kde $\mathbf{B}_k$ je aproximací $\nabla^2 f_k$. Matice $\mathbf{B}_k$ musí splňovat určitá pravidla, aby byla metoda efektivní:

1. $\mathbf{B}_k$ se počítá rekurzivně podle vzorce $\mathbf{B}_{k+1} = \mathbf{B}_k + \mathbf{M}_k$, kde $\mathbf{M}_k$ je aktualizační matice hodnosti maximálně 2.

2. $\mathbf{B}_k$ je symetrická

3. $\mathbf{B}_k$ splňuje tzv. quasi Newtonovskou rovnici $\mathbf{B}_{k+1}\mathbf{s}_k = \mathbf{y}_k$, kde $\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ a $\mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)$.

4. $\mathbf{B_{k+1}}$ je nejlepší matice k $\mathbf{B}_k$ splňující 1-3 v tom smyslu, že řeší úlohu

$$\min_{\mathbf{B}} = \|\mathbf{B} - \mathbf{B}_k\|, \quad \mathbf{B} = \mathbf{B}^T, \mathbf{B}\mathbf{s}_k = \mathbf{y}_k. \tag{3.72}$$

Různé quasi Newtonovské metody dostaneme různou volbou normy (3.72). Jeden takový způsob nám dává BFGS formuli

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} \tag{3.73}$$

Algoritmus BFGS

Zadej počáteční hodnoty $\mathbf{x}_0$ a $\mathbf{B}_0$

1. vypočítej směr $\mathbf{p}_k = \mathbf{B}_k^{-1} \nabla f_k$
2. minimalizací na polopřímce určené bodem x_k a směrem p_k se vypočítá krok α_k a ve směru $\mathbf{p}_k$, aktualizuj $\mathbf{x}_k$ takto $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
3. nastav $\mathbf{s}_k = \alpha_k \mathbf{p}_k$ a $\mathbf{y}_k = \nabla f\left(\mathbf{x}_{k+1}\right) - \nabla f\left(\mathbf{x}_k\right)$
4. $\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k}$
5. ukončíme, pokud je splněno předepsané kritérium

Při trénování neuronové sítě minimalizajeme chybovou funkci $J(\mathbf{W})$ danou (3.63) a proměnnými váhami sítě. Tyto váhy v tomto algoritmu odpovídají $\mathbf{x}$.. Za aproximaci Hessiánu $\mathbf{B}_0$ dosazujeme jednotkovou matici, $\mathbf{x}_0$ jsou počáteční zvolené váhy volené většinou v rozmezí $\langle -0,5; 0,5 \rangle$.

4 Image processing

V této kapitole se budeme zabývat Image Processingem (zpracováním obrazu). Popíšeme si jak se dělí a jaké operace se provádí v rámci Image Processingu. Dále si popíšeme zadaný úkol pro tuto diplomovou práci, co obnáší a jednotlivé kroky, jak budeme tento úkol řešit. Tento úkol budeme řešit v matematickém prostředí zvaném MATLAB^{TM}. Popíšeme si jednotlivé funkce, které z MATLABu použijeme, nejvíce funkcí budeme používat z Image Processing Tooolboxu, dále jen IPT.

Image processing se začal nejvíce rozvíjet v posledních desítkách let vzhledem k jeho využití v aplikacích v reálném čase. Dříve totiž nebyly počítače natolik výkonné, aby mohly zpracovat takové množství dat. Obrázek se skládá z jednotek zvaných pixely, které jsou definovány jako nejmenší jednotky digitální rastrové (bitmapové) grafiky. Uvažujme obrázek o rozměrech 768×1024, má 786432 pixelů. Pokud bychom chtěli provést nějakou operaci na tomto snímku, bylo by to pro dřívější počítače příliš časově náročné. Image processing, což je zpracování obrazu, má široké použití, jako například medicinské skenery, tomografie, mýtné brány.

Obrázek je obecně definován jako $2D$ funkce $f(x, y)$, kde x, y jsou indexy pixelu a f je hodnota pixelu v bodě o souřadnicích (x, y). Tuto hodnotu budeme nazývat intezita. K obrázkům se v MATLABu přistupuje jako k diskrétní mřížce. Obrázek nemusí být vždy reprezentován $2D$ maticí, například u barevných obrázků (RGB)[1] každou dvojici prostorových souřadnic (x, y) reprezentuje trojice bodů, takže snímek je 3-rozměrné pole. Souřadnice obrázku nabývají pouze nezáporných hodnot. V prostředí MATLAB se indexuje od 1 dále + celočíselně.

Existují tři úrovně při zpracování obrazu: nízká, střední a vysoká. Do nízké úrovně patří úkony jako vylepšení kontrastu, odstranění šumu, zaostření obrazu. Dá se říct, že před a po provedení procesů z nízké úrovně dostaneme obrázek stejných rozměrů jako původní. Mezi procesy střední úrovně patří lokalizace objektů, segmentace, normalizace, nalezení hran, hladin intezit obrazu, rozpoznání objektů atp. Výstup po provedení procesů ze střední úrovně je opět obrázek, ale

[1] http://cs.wikipedia.org/wiki/RGB

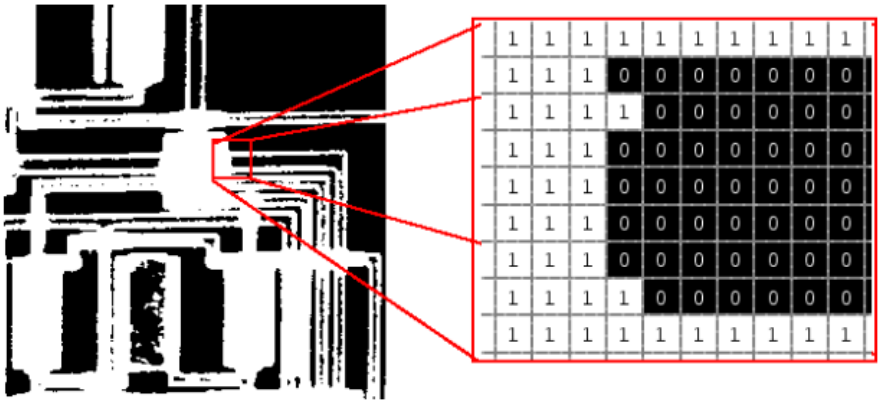
už jenom jeho atribut, který jsme hledali. Konečně vysoká úroveň Image Processingu je porozumění těmto objektům a jejich analýza, tak jak by tyto objekty dokázal popsat lidský mozek.

Než si řekneme něco o tom, jak jsme využívali Image Processing k nalezení vhodného vstupu do neuronové sítě, tak si povíme něco o různých typech obrázků.

4.1 Typy obrázků

Nyní si povíme něco o různých typech obrázků, které budeme používat a které jsou široce rozšířené. Matlab podporuje 4 základní typy obrázků.

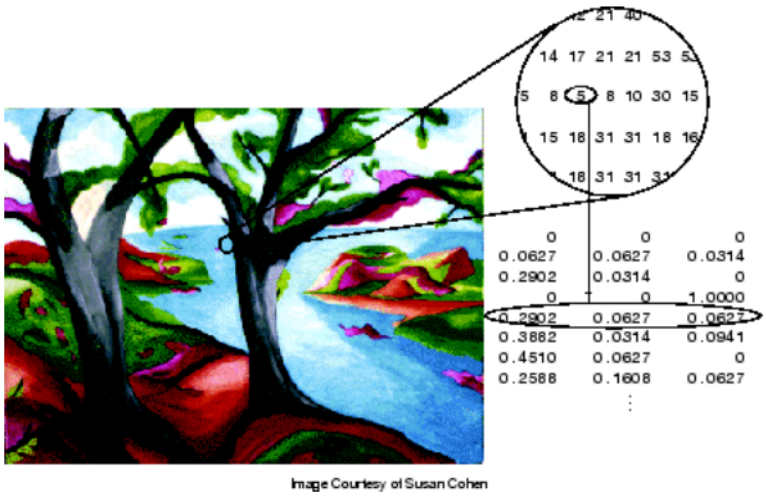
Binární obrázky, kde je obraz reprezentován pouze logickými hodnotami `0,1`. Obvykle v MATLAB kódech značíme tyto obrázky `BW`.



Obrázek 9: Ukázka reprezentace binárního obrázku maticí. Zdroj [MATLAB]

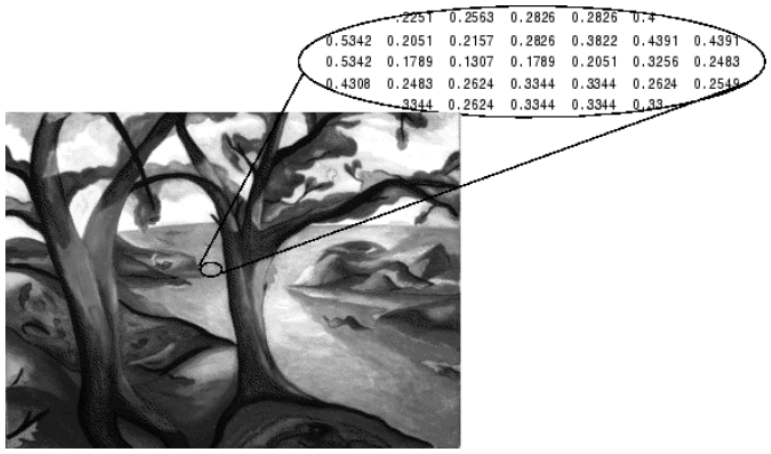
Indexované obrázky jsou popsány maticí a odkazem na mapu barev (paleta). Hodnoty pixelu v matici jsou přímo indexy v mapě barev. Mapa barev je matice o rozměrech $m \times 3$ typu `double` obsahující hodnoty v rozmezí $\langle 0, 1 \rangle$, kde m je počet definovaných barev. Obvykle tyto obrázky značíme `X`. Obrázek může patřit do různých tříd jako například `logical,double,uint8,uint16`[2], a to podle datového typu použitého pro reprezentaci obrázku.

[2]Typy třídy viz MATLAB Product Help



Obrázek 10: Ukázka reprezentace indexového obrázku třídy `double`. Zdroj [MATLAB]

Obrázky ve stupních šedé jsou také reprezentovány maticí, jenom hodnota každého pixelu obrázku reprezentuje pouze určitý stupeň šedé. Intezita tohoto pixelu se liší v závilosti z jaké třídy je tento obrázek. Například u typu `double` 0 reprezentuje černou a 1 bílou barvu. Tyto třídy jsou stejné jako u indexovaného obrázku `logical,double,uint8,uint16`. Podle konvence značíme tyto obrázky `I`.



Obrázek 11: Ukázka reprezentace obrázku ve stupních šedé třídy `double`. Zdroj [MATLAB]

Truecolor obrázky jsou barevné obrázky, ve kterých je každý pixel reprezentován třemi hodnotami, a to pro červenou, zelenou a modrou barvu. MATLAB ukládá Truecolor obrázky jako matice o rozměrech $m \times n \times 3$. Truecolor mohou být opět různých typů tříd `logical,double,uint8,uint16`. Pro zajímavost uvedeme, jak se mícháním `RGB` získají základní barvy. Předpokládejme obrázek typu `uint8` v rozmezí$\langle 0, 255 \rangle$

R	G	B	barva
0	0	0	černá
255	0	0	červená
0	255	0	zelená
0	0	255	modrá
255	255	0	žlutá
255	0	255	purpurová
0	255	255	azurová
255	255	255	bílá

Podle konvencí označujeme tyto obrázky `RGB`.

Obrázek 12: Ukázka Truecolor obrázku třídy `double`. Zdroj [MATLAB]

4.2 Platné registrační značky pro ČR

V současné době jsou v ČR platné tři formáty **registračních značek**(RZ). Nynější formát značek platí od roku 2001. Namísto prvních dvou písmen na označení okresů se označuje písmenem kraj. Formát značek vypadá takto `1A1 0000`. Na druhém místě je písmeno, které označuje kraj. Vzhledem k tomu, že už se překročilo značení `9A9 9999`, tak se v Pražském a Středočeském kraji používají další formáty `1AA 0000`.

Obrázek 13: Registrační značka platná od roku 2001

Po vstupu České republiky 1.května 2004 do Evropské unie přibylo na levé straně RZ její logo.

Obrázek 14: Registrační značka platná od roku 2004

Nejstarší formát používaný od 1994 do roku 2001 měl formát `ABC 00-00`, kde první dvě písmena značila okres a třetí písmeno označovalo sérii. Za písmeny byly dvě dvojice čísel oddělené pomlčkou. Mezeru mezi písmeny a číslicemi později vyplnilo osvědčení o emisích.

Obrázek 15: Registrační značka platná od roku 1994

Registrační značky pro osobní automobily mají rozměrny $520 \times 110mm$ a používají černá písmena na bílém podkladu. Důležitým poznatkem je také, že na

RZ nenajdeme písmena `G,O,Q,W`, z důvodu, že by se mohli plést s `C,V,0`. Platí to ovšem pouze u značek platných od roku 2001. Ale například u RZ pro olomoucký okres platných od roku 1994 bylo `O` velmi obvyklé, nicmémě mělo pevně danou pozici v prvních třech znacích, takže k záměně s nulou nemohlo dojít. Je plno dalších typů platných[3] RZ v České republice, ale těmi se v naší práci nebudeme zabývat. Některé z těchto RZ se například liší pouze barvou znaků a formátem.

4.3 Popis programu rozpoznání SPZ

V této práci je neuronová síť použita na rozpoznání obrazu, konkrétněji na rozpoznání **státních poznávacích značek** (SPZ). Abychom měli nějaký vstup do této sítě, tak si ho musíme připravit. Toto je náš úkol v Image Processingu. Máme několik desítek fotek SPZ různých aut a potřebujeme je zpracovat tak, aby s nimi uměla neuronová síť pracovat.

V našem příkladu budeme mít několik menších zjednodušení. Bohužel nemáme dostatek snímků pro rozpoznání písmen celé abecedy, zaměříme se tedy pouze na číslice, principiálně by byl postup úplně stejný. Neměli jsme přístup k fotkám z mýtných bran, nebo radarů, proto jsme fotili stojící auta zepředu a zezadu. Registrační značky jsou zhruba uprostřed obrázku. Nezaměřovali jsme se na rozpoznání jednotlivých typů RZ, jako například starší značky, značky bez loga Evropské Unie, užitková auta atd. Dále jsme hodně využívali vestavěných funkcí prostředí MATLAB, což také zjednodušilo samotný Image Processing.

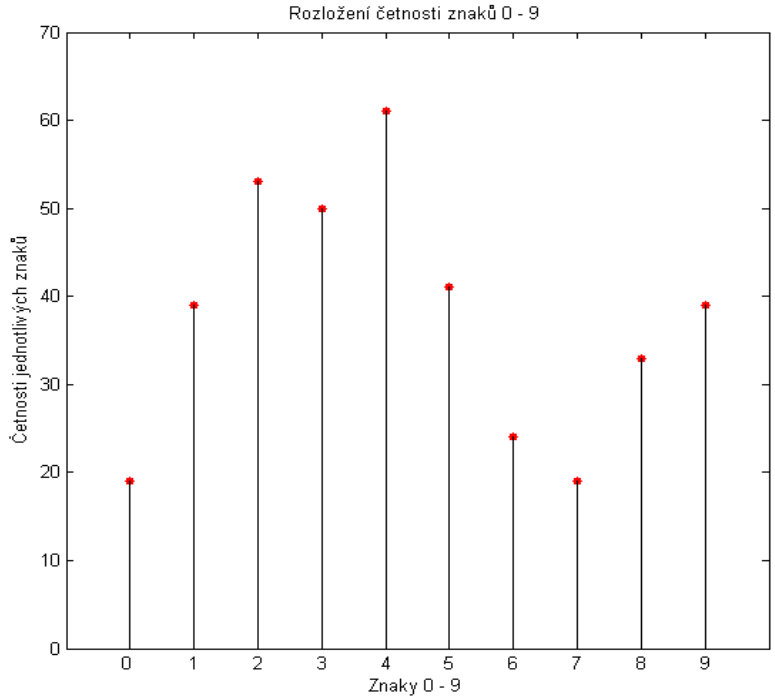
Nyní si stručně shrneme, jak výsledný program bude fungovat. Jako první **pořídíme snímek** podle určitých zásad, viz. podkapitola 4.3.1, což zjednoduší další práci. Nyní **lokalizujeme značku**, kterou vyřízneme z původního obrázku. Z vyříznuté SPZ **segmentujeme jednotlivé znaky**, **zarovnáme na základnu**, **normalizujme** na velikost 13×7. Nyní musíme **zakódovat jednotlivé snímky** a předat do neuronové sítě. To uděláme tak, že veškeré znaky[4] SPZ roztáhneme do jednoho sloupcového vektoru 91×1 a poskládáme je do matice. Nyní jsme

[3] http://www.mdcr.cz/NR/rdonlyres/D59EB03B-3985-4C7F-BB09-1D36FBBEBA7C/0/-regtab.pdf

[4] Kromě písmen, ty nerozpoznáváme vzhledem k neodostatku pořízených snímků

skončili s Image Processingem a přichází na řadu neuronové sítě.

Nyní když máme vstupy do sítě, tak musíme neuronovou síť naučit rozpoznávat na trénovacích prvcích. Námi zvolená trénovací množina byla matice rozměrů 91×378, což je 378 číslic roztáhlých do sloupcového vektoru 91×1. Ideální by bylo, kdyby byly všechny číslice v trénovací množině zastoupeny rovnoměrně. Zde je možné se podívat, na četnosti jednotlivých znaků trénovací množiny. Je



Obrázek 16: Četnosti jednotlivých číslic 0 až 9 v trénovací množině.

lehce vidět, že největší zastoupení má číslice 4, což koresponduje s tím, že se na tento znak neuronové sítě naučily nejlépe, jak se později přesvědčíme.

Trénování množiny je vlastně hledání ideálních vah, a to tak, že se snažíme minimalizovat chybu (3.8) nebo (3.63). Tyto chyby jsme minimalizovali pomocí metod Gradient Descent s různým nastavením parametru učení a momentu sítě a pomocí funkce `fminunc`[5] implementované v MATLAbu, což je vlastně me-

[5]viz. MATLAB Product Help

toda BFGS. Porovnání výsledků dosažených různými sítěmi se budeme věnovat v závěrečné kapitole.

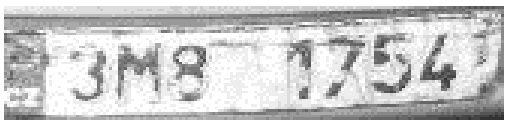
Teď si ukážeme, jak by měl vypadat výstup neuronové sítě při rozpoznání nějaké číslice. Vezměme například číslici 1, ideálně by síť na výstupu měla vrátit jednotkový vektor rozměrů 10×1, kde je 1 na 1. místě. Obdobně by to bylo pro další číslice. Nule jsme přiřadili jednotkový vektor s jedničkou na 10. místě.

Jakmile jsme naučili síť, tak si uložíme jednotlivé váhy. Poté na testovacích prvcích provedeme dopřednou propagaci sítí s těmito váhami podle diagramu (3.59). Výsledný výstup sítě by měl, za předpokladu, že jsme síť dobře natrénovali, co nejvíce odpovídat příslušným daným výstupům (targetům) těchto znaků.

Jednotlivým částem programu se nyní budeme věnovat podrobněji a popíšeme si, jaké jsme používali vlastní heuristiky a vestavěné funkce z MATLABu.

4.3.1 Lokalizace SPZ

Lokalizace SPZ je ze získání vstupu pro neuronovou síť vůbec to nejtěžší. Jednak samotná fotografie může být nekvalitní, například z důvodů špatných světelných podmínek, nebo může být nečitelná, špinavá nebo může být značka dokonce poškozená (což je ovšem proti předpisům).



Obrázek 17: Příklad poškozené SPZ

Na vzorku přibližně 90ti obrázků se nám podařilo lokalizovat všechny SPZ. Nicméně se může stát, že se značku nepodaří lokalizovat. V tom případě musí nastoupit lidský faktor pro rozpoznání. Matoucí prvky při lokalizaci mohou být různé, například podobně tvarovaná světla jako SPZ, nálepka podobných rozměrů, nápis nebo odraz světla na kapotě auta.

Nyní si ukážeme postup při lokalizaci SPZ. Používali jsme různé vestavěné

funkce Image Processing Toolboxu z MATLABu. Postup při lokalizaci by se dal shrhnout následovně.

Obrázek jsme nejprve načetli pomocí `imread(RGB)`[6], poté převedli do stupňů šedé pomocí funkce `rgb2gray` a ořízli o 1/5 ze všech stran z důvodů šetření paměti. SPZ není nikdy v této části fotografie. Poté jsme použili funkci `imadjust` na vylepšení kontrastu obrazu. To znamená, pokud je obrázek třídy `uint8`[7], tak jsme rozsah kontrastu roztáhli na celý interval $\langle 0, 255 \rangle$. Použití této funkce je takovéto: `imadjust(I)`, kde `I` je obrázek ve stupních šedé (může být i `RGB`). Dále jsme použili funkci, která vyznačuje místa, kde je změna intezity největší. Můžeme ji zadat práhový parametr. Pokud je tento prah překročen, je hodnota 1, jinak dostáváme 0. Funkce se použijí následovně: `imextendedmax(I, p)`, kde `I` je obraz ve stupních šedé a parametr `p` je práh. V našem případě se nejlépe osvědčilo $p = 70$. Výstupem této funkce je černobílý obrázek `BW`, viz. obrázek (18). Tato funkce se hodila, protože značka je ideálně černobílá, tudíž rozdíl intenzit na SPZ je velký. V praxi tento problém není až tak jednoduchý (viz důvody, proč ne vždy nalezeneme SPZ.)

Jakmile máme rozdíl intezit, tak následuje postupné odstranění objektů, které nejsou SPZ. Při redukci nevhodných objektů budeme postupovat následovně. Pokud se objekt dotýká jakéhokoliv kraje, tak ho odstraníme. Odstraníme všechny objekty, které nesplňují vhodný poměr šířky ku výšce SPZ, který je v rozmezí 3 až 6. Odstraníme všechny velké, či malé objekty. Při troše štěstí nám na snímku zbyde pouze jedna oblast s SPZ. Nicméně jsou další heuristiky, které jsme použili. Mohlo se stát, že SPZ byla z vrchu překrytá stínem, proto jsme do každého snímku vložili předělové linky, viz obrázek (19). Ty jsme umístily 4 pixely pod horní okraj a 4 pixely nad dolní okraj. Získali jsme tím uzavřený objekt a mohli jsme poté použít funkci `imfill`, která nám vyplnila celou SPZ bílou barvou. Dostali jsme takto další dva výsledky, podle kterých můžeme určit, zda je to SPZ. První výsledek: plocha objektu, tj. plocha jednotlivých bílých části na obrázku (18), ku

[6] viz. MATLAB Product Help, stejně tak ostaní funkce z MATLABu zde zmíněné

[7] unsigned integer 8bit, což znamená, že jas obrázku může nabývat intenzit až do hodnoty $2^8 = 256$. Počítáme i nulu, maximálni hodnota je tedy 255.

Obrázek 18: Výstup z funkce `imextendedmax(I,70)`

ploše opsaného obdelníka. Tento poměr by měl být co nejblíže 1. Druhý výsledek: Obvod objektu, by měl být co nejbližší k obvodu opsaného obdelníku. Poslední heuristika, kterou jsme použili byla založena na výpočtu vzdálenosti objektu od středu snímku. Vybírali jsme ze všech zbylých objektů ten, který byl nejblíže středu.

Obrázek 19: Předělová linka na SPZ se zastíněným horním okrajem.

Pokud všechny tyto techniky selhaly, tak jsme použili funkci pro detekci hran `edge(I,'canny')`[8], která našla obrys značky a postupovali jsme obdobně jako v předešlém případě. Jakmile jsme dostali jediný objekt a jeho nejmenší opsaný obdelník (Bounding Box), tak jsme tento obdelník vyřízli z původní fotografie. Proč bychom měli vybírat SPZ z původního obrázku? Důvod je ten, že po oříznutí dostaneme malý obrázek, na kterém znovu provedeme roztáhnutí

[8]canny je metoda pro detekci hran

kontrastu snímku pomocí funkce `imadjust`, což dává lepší výsledky, než když vyřízneme obrázek s upraveným kontrastem. Ztratíme tak vlastně upravené spektrum kontrastu. Nyní máme připravený snímek pro segmentaci jednotlivých znaků SPZ.

Obrázek 20: Nalezený Bounding Box a vzdálenost SPZ od středu obrázku.

Při lokalizaci jsme nepočítali s tím, že bychom fotili SPZ z ostrého úhlu, předpokládali jsme, že SPZ je zhruba uprostřed snímku a nebyla vážně poškozená, nejhorší vzorek SPZ je obrázek (17). Některé fotky nejsou vodorovné, nicméně jsme nepoužívali žádnou prostorovou transformaci. Způsob, jakým jsme tento problém vyešili bude popsán v kapitole Segmentace SPZ.

4.3.2 Segmentace SPZ

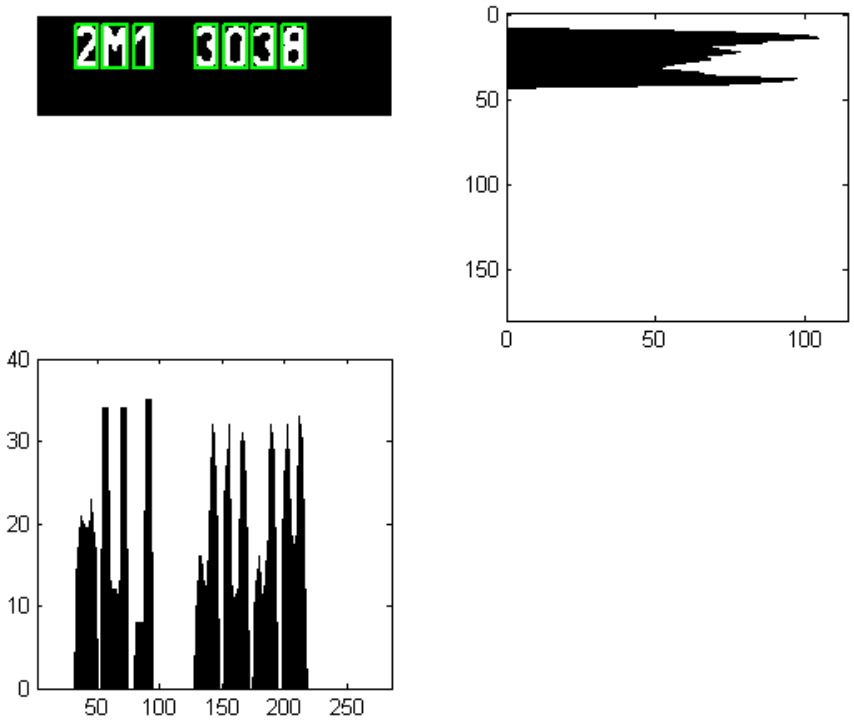
Poté, co jsme dostali SPZ z původního RGB obrázku, tak musíme rozčlenit její jednotlivé znaky. Znovu jsme použili podobné heuristiky jako při lokalizaci. Roztáhli jsme intenzitu snímku a SPZ jsme převedli na binární obraz `BW` metodou prahování. Výstup příkazu `prah=graythresh(I)` je hodnota mezi $\langle 0, 1 \rangle$, která rozdělí intenzity obrazu ve stupních šedé na dvě skupiny. S tímto prahem a touto funkcí `im2bw(I,prah)` vytvoříme černobílý obrázek. Hodnotám intezity nad prahem přiřadíme hodnotu jedna (bílá), zbytku hodnotu nula (černá).

Některé znaky se dotýkaly okraje obrázku získaného lokalizací, proto jsme museli nejprve znaky oddělit od okraje předělovou linkou, podobně jako v předešlé kapitole, viz obrázek (19). Poté jsme invertovali barvy, tedy, z černé se stala bílá a z bílé se stala černá. Důvodem této inverze barev bylo to, že jsme potřebovali, aby naše znaky byly bílé.

Máme připravený snímek k vlastní segmentaci a začneme postupně odstraňovat nevhodné objekty. Nejprve odstraníme vše, co je spojené s okrajem. Odstraníme malé objekty, které jsou v poměru ku celkové ploše SPZ příliš malé ve srovnání se znaky SPZ. Dále odstraníme objekty, jejichž poměr výšky ku šířce je menší než 1,1. Tento poměr vznikl z předpokladu, že znak je vyšší než širší. Navíc jsme tímto poměrem odfiltrovali nežádoucí prvky jako jsou nálepky o kontrole emisí. V drtivé většině jsou tyto heuristiky dostačující, pro to, aby nám zbylo pouze 7 znaků. Pokud jsme nedostali 7 znaků, tak jsme použili funkci `imextendedmax`, a použili jsme obdobné heuristiky a snažili jsme se dostat opsané obdelníky jednotlivých znaků. Tento způsob jsme použili například na obrázku (17).

Nyní máme 7 znaků. Posunuli jsme jednotlivé znaky na „základnu“. To znamená, že jsme našli nejmenší y-vou souřadnici jednotlivých opsaných obdelníku a posunuli všechny ostatní na tuto y-vou souřadnici. Tímto způsobem jsme vyrovnali všechny znaky do jedné linky. Dále jsme provedli projekci na osu y a osu x našeho obrázku. Dostali jsme takto indexy, kde máme jednotlivé znaky segmentovat. Výsledek můžeme vidět na obrázku (21).

Nyní máme segmentováno, každý znak má ale různou velikost. Proto všechny tyto znaky normalizujeme na rozměry 13×7. Tento rozměr není nijak náhodný, spočitali jsme si podíly všech opsaných obdelníků výška ku šířce a vyšel nám poměr zhruba $1,8$. Proto, aby nedošlo ke ztrátě informace, jsme volili poměr 13/7, což je přibližně 1.85. Tyto matice poté vektorizujeme, jak už jsme si řekli v části 4.3.



Obrázek 21: Projekce získaných znaků na osu x, vlevo dole, a osu y, vpravo nahoře.

4.3.3 Zobecnění programu

Předpokládali jsme jistá omezení našeho programu na rozpoznávání SPZ. Neměli jsme dostatek dat pro to, abychom mohli rozpoznávat písmena. jak by ale tento program mohl vypadat, kdybychom rozpoznávali i písmena?

Jsou dvě možnosti, buď bychom rozpoznávali písmena a číslice zvlášť, pro písmena i číslice bychom měli zvláštní síť. Druhá možnost je, že bychom rozpoznávali číslice a písmena najednou. Prvním případ by byl totožný, jako při rozpoznávání číslic. Jediný rozdíl by byl v počtu výstupních neuronů, na místo

10 bychom měli 22 výstupních neruonů[9]. V druhém případě bychom přidali ke stávajícm 10 výstupním neuronům dalších 22 a proces by byl obdobný.

Další zobecnění by mohlo být, že by program rozpoznával jednotlivé typy značek, tedy se znakem EU, bez něho a nebo starý formát SPZ. Dále různé značky různých států Evropské Unie, rozpoznávání značek v noci, či nějak poškozených značek.

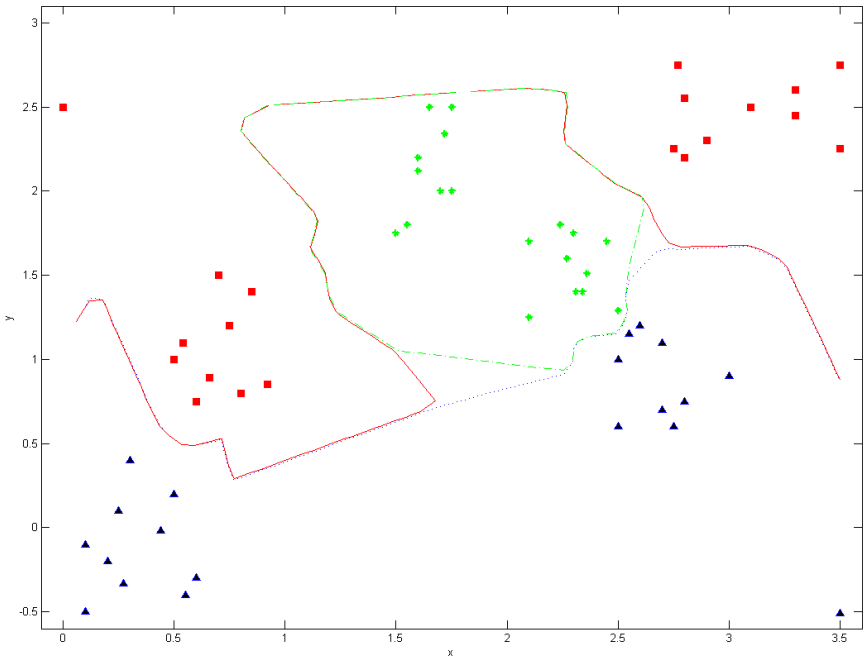
[9]22 neuronů z důvodu, že abeceda má 26 znaků, ale na SPZ v České republice jsou vyřazeny 4 znaky `G,W,Q,O`, aby nedošlo k záměně s `C,V,0`.

5 Porovnání neuronových sítí

V této kapitole se pokusíme shrnout veškeré výsledky, které jsme dostali z různých metod na minimalizaci chybových funkcí (3.8) nebo (3.63). Vzhledem k odlišnostem našich chybových funkcí nemůžeme klasicky porovnávat tyto hodnoty mezi sebou. Jediné, co můžeme poměřovat je fakt, jak dobře jednotlivé metody počítají jednotlivé úlohy. Jinými slovy všechny metody na minimalizaci vytvářejí výstup, který můžeme poměřovat, jak dobrý je náš výstup a jak dobře odpovídá realitě. Nejprve si porovnáme výsledky na jednodušších příkladech.

5.1 Klasifikace

Budeme řešit klasifikační úlohu znárněnou následujícím obrázkem Mějme nyní

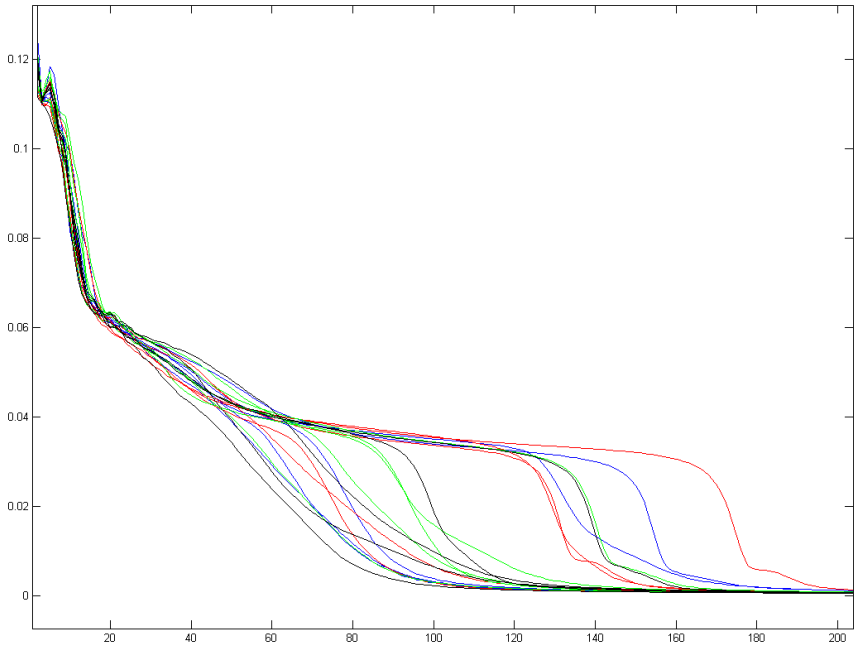


Obrázek 22: Příklad na klasifikaci jednotlivých tříd.

pro testování neuronových sítí toto nastavení. Struktura sítě budiž [3 8 3], první

číslo je dimenze trénovacího prvku +1 pro bias, tedy 3. Další číslo je počet neuronů ve skryté vrstvě včetně biasu. Poslední číslo je počet výstupních neuronů, což je přesně počet tříd, které chceme použít při klasfikaci. Dále jsme nastavili parametr učení $\gamma = 1$ a moment sítě na $\mu = 0.9$ a počáteční interval pro nastavení vah byl $\langle -0.1, 0.1 \rangle$. Abychom nějakou síť nezastavili předčasně, tak jsme trénování nechali běžet přes 1000 iterací.

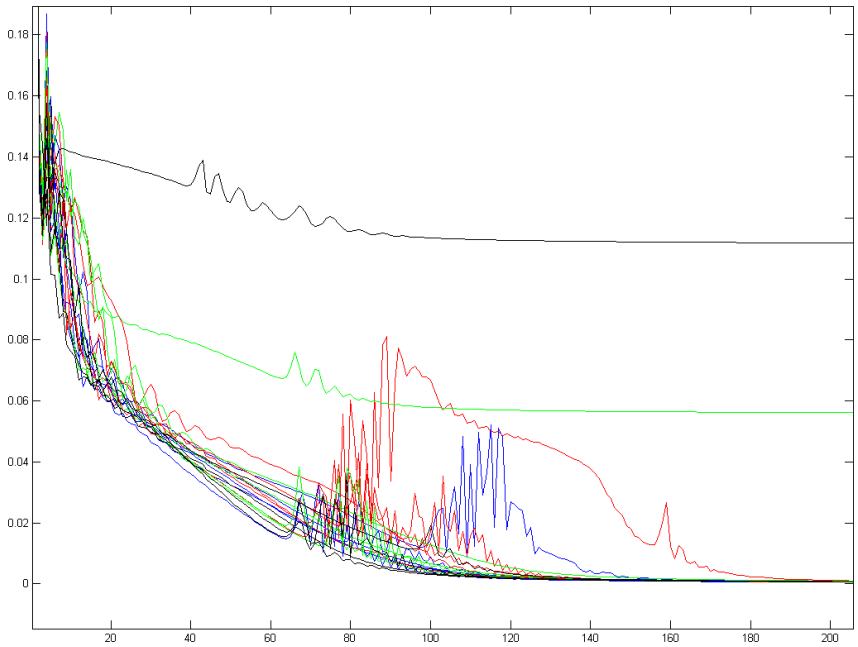
Nejprve jsme vyzkoušeli minimalizovat metodou Gradient Descent chybovou funkci (3.8) s výše zmíněným nastavením. Nechali jsme síť trénovat 20krát a pokaždé jsme dostali spravné klasifikační křivky v průměrném čase kolem $250ms$. Chyba klesla pokaždé pod hodnotu 0.001 do 200 iterace. Obrázek pro 20 různých trénování sítě je možné vidět na obrázku číslo (23)



Obrázek 23: Hodnota chybové funkce v závislosti na počtu iterací. Trénování 20ti různých neuronových sítí s chybovou funkcí (3.8) se strukturou [3 8 3]

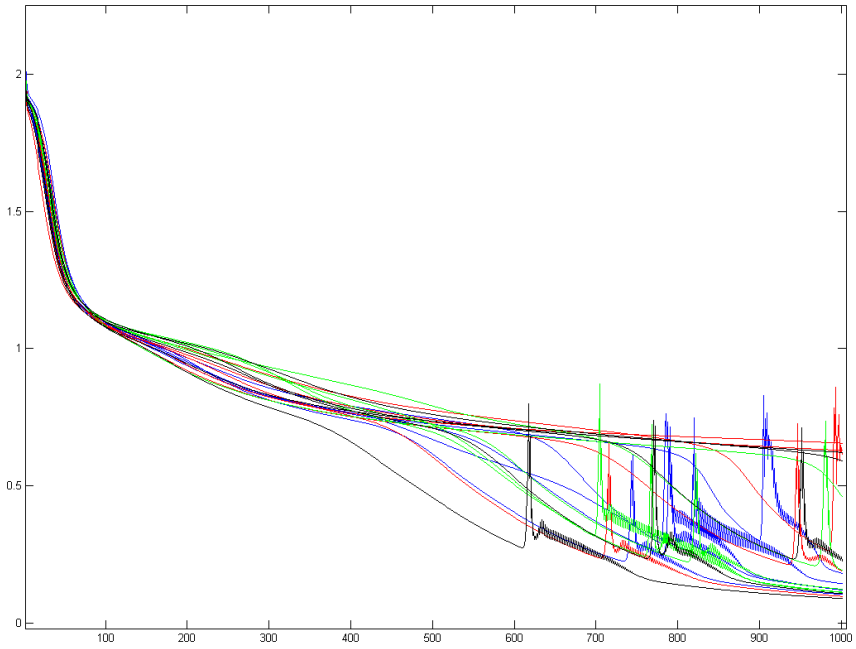
Nyní vyzkoušíme trochu pozměnit strukturu sítě takto [3 58 3] a uvidíme, jaký to bude mít vliv na trénování sítě. Na obrázku (24) můžeme vidět, že jsme

si s rychlostí konvergence nijak nepolepšili a co víc, pro některé počáteční hodnoty se ukazuje, že jsme zřejmě uvízli v bodu lokálního minima. Průměrná doba za kterou jsme trénovali síť stoupla na $550ms$



Obrázek 24: Hodnota chybové funkce v závislosti na počtu iterací. Trénování 20ti různých neuronových sítí s chybovou funkcí (3.8) se strukturou [3 58 3]

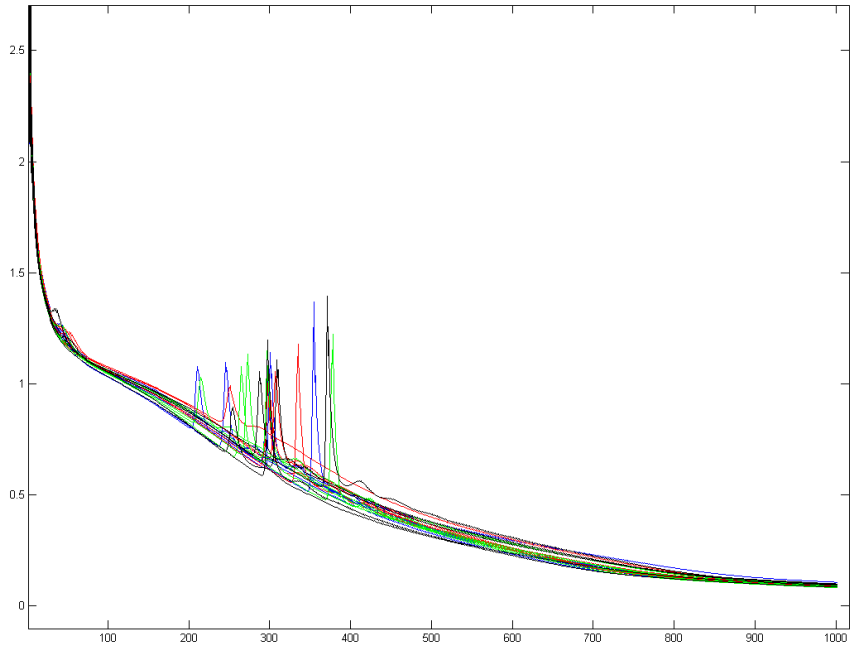
Nyní vyzkoušejme něco obdobného jako v předešlých dvou případech. Budeme minimalizovat chybovou funkci (3.63) se strukturou [3 8 3], máme zde ještě navíc regularizační parametr a ten jsme položili $\lambda = 0.01$. Za 1000 iterací jsme se nedočkali uspokojivého výsledku, potřebovali bychom více iterací. Zde nám toho hodnota chybové funkce neřekne tolik, jak tomu bylo v předešlém případě. Hodnota se nemusí blížit nule a přesto může být síť velmi dobře natrénovaná. Nicméně na základě pozorování můžeme říct, že hodnota 0.2 chybové funkce pro tuto úlohu už dobře klasifikuje. Síti trvalo učení zhruba $400ms$, avšak ne s příliš dobrým výsledkem. Na obrázku (25) můžeme vidět vývoj 20 odlišných trénování sítě.



Obrázek 25: Hodnota chybové funkce v závislosti na počtu iterací. Trénování 20ti různých neuronových sítí s chybovou funkcí (3.63) se strukturou [3 8 3]

Nakonec nám zbývá vyzkoušet minimalizovat chybovou funkci (3.63) spolu se strukturou sítě [3 58 3]. Zde tomu bylo naopak jako u minimalizace (3.8), protože se nám výsledek zlepšil. Všech 20 sítí s počátečními nastaveními nám správně klasifikovalo všechny prvky trénovací množiny za dobu $900ms$. Hodnoty jednotlivých chybových funkcí můžeme najít na obrázku (26).

Na tomto místě bychom mohli vyzkoušet ještě minimalizovat chybovou funkci (3.63) pomocí metody BFGS. Tato metoda dává občas hodně špatné výsledky a někdy se dokonce funkce `fminunc` ukončí s chybovou hláškou, že výsledné klasifikační hranice nelze najít. Nicméně, pokud vše proběhlo tak, jak mělo, tak klasifikace končila povětšinou kolem 80 iterace s funkční hodnotou chybové funkce kolem 0.2 zhruba za $50ms$.

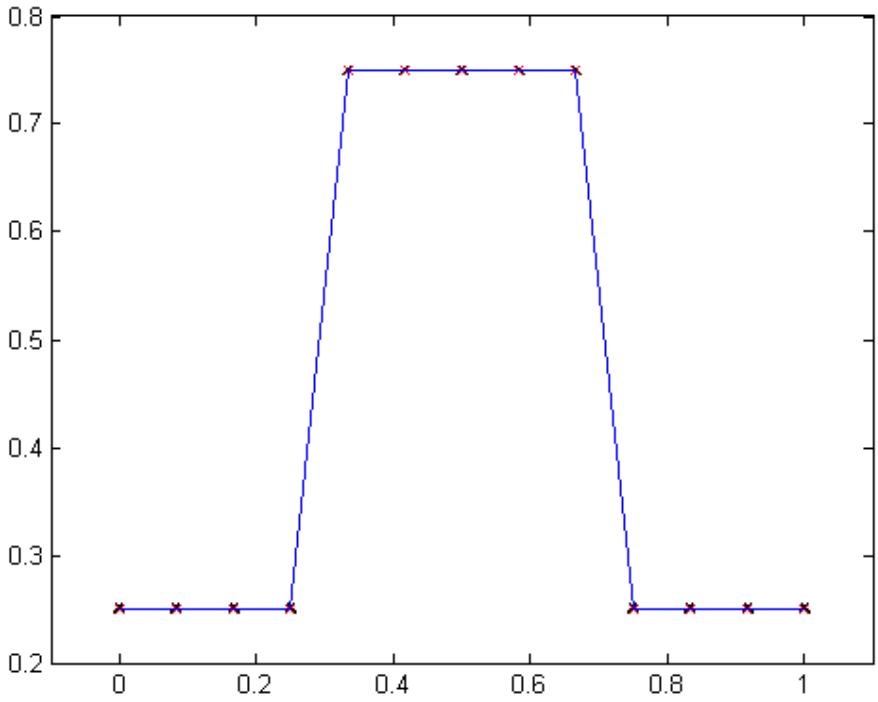


Obrázek 26: Hodnota chybové funkce v závislosti na počtu iterací. Trénování 20ti různých neuronových sítí s chybovou funkcí (3.63) se strukturou [3 58 3]

5.2 Aproximace

Nyní si jenom v rychlosti ukažme aproximační příklad, kde bude jako jediná metoda dávat rozumné výsledky metoda BFGS. V knize [[Samarasinghe]] je tento příklad na stráně 86. Jedná se o aproximaci jednotkového impulsu. Tato funkce je konstatní v hodnotě 0.25 pro x menší než 0.3 a větší než 0.7, mezi těmito dvěmi hodnotami je funkce konstatní s hodnotou 0.75. Metody Gradient Descent nedávaly pro obě chybové funkce (3.8) a (3.63) rozumné výsledky ani po 10000 iteracích učení, při jakémkoliv nastavení. Proto jsme vyzkoušeli metodu BFGS spolu s chybovou funkcí (3.63). Nastavení neuronové sítě vypadalo takto. Zvolili jsme strukturu [2 11 1], počáteční interval pro váhy $\langle -0,5; 0.5\rangle$, regularizační parametr jsme položili $\lambda = 0$. Neuronovou síť jsme opět nechali proběhnout 20krát s různými počátečními váhami, výsledek byl pokaždé téměř totožný. Síť

se průměrně naučila na jednotkový puls za 360 iterací za $390ms$. Celkový výsledek je vidět na obrázku (27).



Obrázek 27: Výsledný obrázek aproximace jednotkového pulsu.

5.3 Rozpoznání SPZ a neuronová síť

Nyní se budeme zabývat trénováním sítě na rozpoznávání SPZ z pořízených obrázků. Při rozpoznávání jsme neměřili rychlost, za jakou se síť dokázala naučit, protože jakmile síť natrénujeme, tak ji už nemusíme učit a používáme pouze naučené váhy. Nejdůležitějším měřítkem je, jestli jsme byli úpsěšní při rozpoznávání SPZ. **Všechny 3 metody byly úspěšné na 100%!**

Dále si podrobněji popíšeme, jak jsme sítě testovali a jaké jsme dostali průměrné výstupy. Pro všechny sítě byla společná struktura sítě [92 51 10].

Minimalizace funkce (3.8) pomocí Gradient Descent: Parametr učení jsme nastavili $\gamma = 0,1$ a moment sítě $\mu = 0,9$. Počáteční interval pro váhy

znaky	dolní průměr	znaky	horní průměr
1	0.000798645052873	1	0.997630838824985
2	0.000871897768751	2	0.997312573148042
3	0.001030077876407	3	0.997740157354993
4	0.000516438579215	4	0.998734919394994
5	0.005090759153502	5	0.998708178595212
6	0.000992223073024	6	0.981529262970775
7	0.000858534603610	7	0.995769866601838
8	0.001191978497643	8	0.992002574435131
9	0.000807916234494	9	0.997705097183570
0	0.000896630671830	0	0.986019683575523

Tabulka 1: Sloupec dolní průměr značí, že jsou zde průměrné hodnoty výstupu sítě, pokud nebyl na dané pozici SPZ znak $i, i = 1, \ldots, 10$. Sloupec horní průměr značí, že jsou zde průměrné hodnoty výstupu sítě, pokud byl na dané pozici SPZ znak $i, i = 1, \ldots, 10$.

jsme měli následující $\langle -0, 1; 0, 1 \rangle$. Síť jsme trénovali přes 10 000 iterací. Dosáhli jsme chyby zhruba 10^{-6} zhruba za $30s$, což je výborný výsledek. Uspěšnost rozpoznávání jsme testovali na 126 znacích, tedy na 21 SPZ. Vzpomeňme si, jak jsme jednotlivým znakům přiřazovali jednotlivé jednotkové vektory, viz kapitola 4.3. Číslu 1 jsme přiřadili jednotkový vektor s jedničkou na prvním místě, číslu 2 jednotkový vektor na druhém místě, atd. až 0 jsme přiřadili jednotkový vektor na desátém místě. Nyní si ukážeme, jak tato metoda průměrně klasifikovala jednotlivé znaky. To znamená, že jsme udělali průměr všech hodnot, kdy síť měla klasifikovat jako 1 a kdy měla klasifikovat jako 0. Tabulka (1) ukazuje, jakých výsledků tato metoda dosáhla.

Ve všech případech je rozdíl v klasifikaci minimálně 3 řády, což nám dává jistotu, že metoda znaky rozpoznává s dosti velkou spolehlivostí.

Minimalizace funkce (3.63) pomocí Gradient Descent: Parametr učení a moment sítě a inicializační váhy jsme nastavili stejně jako v minulém případě, regularizační parametr jsme položili $\lambda = 0.01$. Po 10 000 iteracích jsme dosáhli zhruba hodnoty chybové funkce 0.035 za dobu $55s$. Opět jsme testovali tyto váhy na stejném vzorku znaků SPZ a opět byla tato síť 100% úspěšná. Na její výsledky se můžeme podívat v tabulce (2).

znaky	dolní průměr	znaky	horní průměr
1	0.000094717754444	1	0.999098042074810
2	0.000127913974847	2	0.999137642632312
3	0.000379142121019	3	0.999265554605577
4	0.000106416046695	4	0.999718968941014
5	0.002279463599648	5	0.999352601388596
6	0.000198175814644	6	0.960206505788540
7	0.000103983709201	7	0.998818239710281
8	0.000257384706263	8	0.995476526271937
9	0.000150848378469	9	0.999412734265586
0	0.000113251201669	0	0.996173369468473

Tabulka 2: Sloupec dolní průměr značí, že jsou zde průměrné hodnoty výstupu sítě, pokud nebyl na dané pozici SPZ znak $i, i = 1, \ldots, 10$. Sloupec horní průměr značí, že jsou zde průměrné hodnoty výstupu sítě, pokud byl na dané pozici SPZ znak $i, i = 1, \ldots, 10$.

Minimalizace funkce (3.63) pomocí metody BFGS: Strukturu jsme volili stejnou jako v předešlých případech [92 51 10], stejně tak počáteční váhy v intervalu $\langle -0,1; 0.1 \rangle$. Tento algoritmus by měl být nejvhodnější metodou pro minimalizaci funkce (3.63). K této funkci jsme dále volili regularizační parametr $\lambda = 0.01$. Funkce `fminunc` si sama řídí ukončovací kritérium, tudíž i počet iterací. Tato funkce se ukončovala zhruba po 110ti iteracích s hodnotou chybové funkce srovnatelnou s předešlou metodou kolem 0.03 za zhruba 150 sekund, což napovídá, že jsme se dostali do bodu lokálního minima. Zajímavé bylo, jak se projevila klasifikace jednotlivých znaků u této metody. Na první pohled v tabulce (3) je vidět, že zdaleka nejlépe se klasifikoval znak 4. Na obrázku (16) můžeme vidět, že četnost znaku je doopravdy největší, konkrétně byl tento znak zastoupen v této trénovací množině 61krát. U předešlých metod tento rozdíl nebyl tak markantní.

5.3.1 Urychlení rozpoznávání SPZ

V předešlé kapitole jsme trénovali neuronovou síť na rozpoznávání SPZ a ukázali jsme si, jakých jsme dosahovali průměrných klasifikačních výsledků se strukturou [92 51 10]. Tato struktura rozpoznala jeden snímek za zhruba $300ms$.

znaky	dolní průměr	znaky	horní průměr
1	0.000033545548629	1	0.992918924707671
2	0.000195524648639	2	0.997529813076750
3	0.002683993768284	3	0.995263612261197
4	0.000000643365185	4	0.999999990506329
5	0.004289930831629	5	0.978585517678558
6	0.000502156369517	6	0.945778045845703
7	0.001370045411801	7	0.992492232778929
8	0.002527188394426	8	0.988907033574147
9	0.000609378849219	9	0.992191898518292
0	0.000597480641120	0	0.983455655127692

Tabulka 3: Sloupec dolní průměr značí, že jsou zde průměrné hodnoty výstupu sítě, pokud nebyl na dané pozici SPZ znak $i, i = 1, \ldots, 10$. Sloupec horní průměr značí, že jsou zde průměrné hodnoty výstupu sítě, pokud byl na dané pozici SPZ znak $i, i = 1, \ldots, 10$.

Jak bychom ale rozpoznávat SPZ rychleji? Než si řekneme, jak bychom mohli dosáhnout určitého urychlení rozpoznání SPZ, vzpomeňme si na diagram (3.59). Pro naši momentální danou strukturu sítě dostaneme váhy, které mají rozměry

$$dim\left(\mathbf{W}^1\right) = 50 \times 92,\ dim\left(\mathbf{W}^2\right) = 10 \times 51. \tag{5.1}$$

Vidíme, že rozměry těchto matic jsou relativně velké. Rozměry těchto matic vah bychom mohli zmenšit dvěma způsoby. Mohli bychom snížit počet neuronů ve skryté vrstvě. V předchozí kapitole jsme zkoušeli testovat neuronové sítě s 10ti skrytými neurony. A také bychom nemuseli neuronové síti předávat bitmapu znaku, ale nějaké napočítané charakteristiky. Těchto charakteristik je celá řada: Konektivita znaku, tj. počet sousedů jednotlivých vrcholů znaku, centrální moment, 2. centrální moment znaku, další vyšší centrální momenty, vertikální a horizontální projekce znaku, atd. Více těchto charakteristik je možné nalézt v [LPR].

Pokud bychom měli k jednomu znaku například 7 charakteristik a použili bychom 10 skrytých neuronů, naše váhy by měly následující rozměry.

$$dim\left(\mathbf{W}^1\right) = 10 \times 8,\ dim\left(\mathbf{W}^2\right) = 10 \times 11. \tag{5.2}$$

Rozměry vah jsou nyní podstatně menší, nicméně při rozpoznávání SPZ dochází

jenom jednou, k dopředné propagaci aktivace trénovacího prvkuy tak ušetříme pouze relativně málo času.

Závěr

Cílem této práce bylo podat ucelený popis neuronových sítí a naprogramovat neuronovou síť pro rozpoznání SPZ. Nejprve jsme se věnovali základům neuronových sítí, řekli jsme si něco o jejich struktuře, jejich jednotlivých pojmech a popsali jsme si nějaké typy neurononových sítí. Podrobněji jsme se věnovali neuronovým sítím s dopředným šířením.

V praktické části jsme se věnovali vytvoření a trénování neuronových sítí na rozpoznávání jednotlivých znaků SPZ. Vzhledem k tomu, že jsme neměli dostatek snímků různých SPZ aut, tak jsme nemohli rozpoznávat písmena. Kdybychom měli rozpoznávat písmena a číslice zároveň, tak bychom, dle mého názoru potřebovali mnohem větší trénovací množinu než 378 znaků. Abychom získali vstupní data pro neuronové sítě, museli jsme se zabývat také Image Processingem, který jsme používali na lokalizaci, segmentaci SPZ a normalizaci jednotlivých znaků. Jakmile jsme měli připravenou trénovací množinu, tak jsme mohli začít trénovat neuronovou síť. Zde přišly na řadu různé metody pro nastavování vah. Všechny metody, které jsme použili, měly výborné výsledky na rozpoznání SPZ, viz kapitola Porovnání neuronových sítí.

Další možností aplikace neuronových sítí spojených s touto diplomovou prací by mohla být lokalizace SPZ. V kapitole o lokalizaci SPZ nám po použití funkce `imextendedmax` zůstalo na snímku několik objektů. Nakonec jsme vyloučili všechny objekty kromě jednoho a doufali jsme (v našem případě vždy správně), že poslední objekt je naše SPZ. Nemusíme být ovšem tak přísní. Na těchto objektech bychom napočítali nejrůznější heuristiky jako v kapitole Lokalizace SPZ. Tyto heuristiky bychom zadávali jako vstupní data do klasifikační neuronové sítě, podobné té v kapitole (5.1). Ovšem jenom s jedním výstupním neuronem, protože by stačilo určovat, zda je to SPZ, či není. Tento způsob lokalizace by byl velmi výhodný, pokud by na pořízených snímcích byla více než jedna SPZ.

Závěrem bych chtěl říct, že jsem věděl, že práce s neuronovými sítěmi není nejlehčí, ale za to, byla o to víc zábavná a velmi aplikovatelná. Díky neuronovým sítím jsem poznal nová odvětví matematiky jako třeba Image Processing, zpra-

cování signálu, Computer Science. Všechny tyto aplikace mne zaujaly a rád bych se jim věnoval do budoucna, i kdyby jenom pro zábavu při práci.

Přiložené CD

V této kapitole si popíšeme architekturu a programy CD přiloženého k této diplomové práci. Popíšeme si obsahy jednotlivých složek a přiložená data.

Alpha Version

Obsah složky Alpha Version		
Pomocné funkce	Spouštěcí funkce	Data
`SPZlocate`	`SPZrecog`	`W150,W250`
`SPZsegment`		`WGR150,WGR250`
`SPZcoding`		`WBF150,WBF250`
		`NEWSAM images`

- **Pomocné funkce:**
  - `SPZlocate` lokalizuje SPZ ze snímku.
  - `SPZsegment` segmentuje jednotlivé znaky lokalizované SPZ.
  - `SPZcoding` vektorizuje jednotlive znaky lokalizované SPZ a vytváří tak SPZ hotovou pro vlastní rozpoznání.
- **Spouštěcí funkce:**
  - `SPZrecog` funkce rozpoznává SPZ z vyfotografovaného snímku. Lokalizuje SPZ, segmentuje a vektorizuje jednotlive znaky SPZ vstupní parametry jsou:

    `IO` - uživateli zadávaný řetězec pro zobrazení vyříznuté SPZ, připustné hodnoty jsou: `'on'` - zobrazí SPZ, `'off'` - nezobrazí SPZ, vstupy nejsou citlivé na velká písmena.

    `Method` - metoda jakou byly získány jednotlivé váhy, je to řetězec, přípustné hodnoty jsou `'GRD'` - minimalizace chybové funkce (3.8) metodou Gradient Descent, `'LGRD'` - minimalizace chybové funkce (3.63) pomocí metody Gradient Descent, `'LBFGS'` - minimalizace chybové funkce (3.63) metodou BFGS, vstup `method` není citlivý na velka

písmena.

Vhodné zadání této funkce je následující:

```
SPZrecog('on','LBFGS')
```

- Data
  - `W150,W250,WGR150,WGR250,WBF150,WBF250` jsou váhy naučené různými metodami.
  - `NEWSAM images` 21 obrázků ze kterých se dané SPZ rozpoznávají.

Examples

Obsah složky Examples	
Pomocné funkce	Spouštěcí funkce
`BackProp`	`Clusters`
`GenWeights`	`SinWave`
`PatternPass`	
`SinPatterns`	
`StructNet`	

- **Pomocné funkce:**
  - `BackProp` počítá derivaci chybové funkce (3.8) neuronové sítě a následně ji minimalizuje pomocí metody Gradient Descent.
  - `GenWeights` automaticky vytváří počáteční váhy neuronové sítě.
  - `PaternPass` provádí dopřednou propagaci aktivace trénovací množiny a počítá celkovou chybu sítě.
  - `SinPatterns` vytváří trénovací množinu a daný výstup ze čtvrtiny periody funkce sin pro neuronovou síť.
  - `StructNet` vytváří celou strukturu sítě, kde je uložen počet vrstev, architektura sítě, váhy sítě, aktualizace vah sítě, aktivační funkce, derivace aktivační funkce.
- **Spouštěcí funkce:**
  - `Clusters` je příklad z kapitoly 5.1, který klasifikuje tři třídy trénovacích prvků pomocí metody Gradient Descent, která minimalizuje chybovou funkci (3.8. Vstupy této funkce jsou:
    `NumHid` - počet skrytých neuronů,
    `LR` - parametr učení,
    `Mu` - moment sítě,
    `Int` - rozmezí pro počáteční nastavení vah,

`MaxIter` - maximální počet iterací při trénování sítě,

`tol` tolerance chyby.

Výstupem je:

`cas` - měří za jakou dobu se daná síť natrénovala.

Nastavení hodnoty `MaxIter` nelze nijak blíže specifikovat, záleží na nastavení parametrů sítě. Při použití ukončovacího kritéria pomocí `tol`, stačí na pozici `MaxIter` psát `[]`. Vhodné zadání této funkce je následující:

```
cas=clusters(7,1,0.9,[-0.5 0.5],[],0.001)
```

– `SinWave` je příklad z kapitoly 5.2, který aproximuje čtvrtinu periody funkce sin pomocí metody Gradient Descent, která minimalizuje chybovou funkci (3.8. Vstupy této funkce jsou:

`NumHid` - počet skrytých neuronů,

`LR` - parametr učení,

`Mu` - moment sítě,

`Int` - rozmezí pro počáteční nastavení vah,

`MaxIter` - maximální počet iterací při trénování sítě,

`tol` - tolerance chyby.

Výstupem je: `cas` - měří za jakou dobu se daná síť natrénovala,

`rozdil` - udává rozdíl mezi výstupem sítě a daným výstupem.

Nastavení hodnoty `MaxIter` nelze nijak blíže specifikovat, záleží na nastavení parametrů sítě. Při použití ukončovacího kritéria pomocí `tol`, stačí na pozici `MaxIter` psát `[]`. Vhodné zadání této funkce je následující:

```
[rozdil,cas]=SinWave(1,0.1,0.9,[-0.5 0.5],[],0.0001)
```

Examples Log

Obsah složky Examples Log	
Pomocné funkce	Spouštěcí funkce
`BwProp`	`LogClustersBFGSNet`
`costfunction`	`LogClustersGrDesNet`
`FwProp`	`LSinWaveBFGS`
`GenWeights`	`LSinWaveGrDes`
`GradientDescent`	`LSqrWaveBFGS`
`grcostfunction`	
`SinPatterns`	
`SqrPatterns`	
`StructNet`	

- **Pomocné funkce:**
  - `GenWeights`, `SinPatterns`, `StructNet` byly popsány již dříve.
  - `BwProp` počítá derivaci chybové funkce (3.63) neuronové sítě.
  - `costfunction` chybová funkce (3.63) pro metodu BFGS.
  - `FwProp` provádí dopřednou propagaci aktivace trénovací množiny.
  - `GradientDescent` minimalizuje (3.63) pomocí Gradient Descent metody.
  - `grcostfunction` chybová funkce (3.63) pro metodu Gradient Descent.
  - `SqrPatterns` vytváří trénovací množinu a daný výstup z jednotkového impulsu pro neuronovou síť.
- **Spouštěcí funkce:**
  - `LogClustersBFGSNet` je příklad z kapitoly 5.1, který klasifikuje tři třídy trénovacích prvků pomocí metody BFGS, která minimalizuje chybovou funkci (3.63. Vstupy této funkce jsou:
    `NumHid` - počet skrytých neuronů,
    `Int` - rozmezí pro počáteční nastavení vah,

`lambda` - regularizační parametr.

Výstupem je:

`jval` - hodnota chybové funkce `cas` - měří za jakou dobu se daná síť natrénovala.

Vhodné zadání této funkce je následující:

```
[jval,cas]=LogClustersBFGSNet(50,[-0.5 0.5],0.01)
```

– `LogClustersGrDesNet` je příklad z kapitoly 5.1, který klasifikuje tři třídy trénovacích prvků pomocí metody Gradient Descent, která minimalizuje chybovou funkci (3.63. Vstupy této funkce jsou:

`NumHid` - počet skrytých neuronů,

`LR` - parametr učení,

`Mu` - moment sítě,

`Int` - rozmezí pro počáteční nastavení vah,

`MaxIter` - maximální počet iterací při trénování sítě,

`lambda`, - regularizační parametr.

Výstupem je:

`jval` - hodnota chybové funkce, `cas` - měří za jakou dobu se daná síť natrénovala.

Vhodné zadání této funkce je následující:

```
[jval,cas]=LogClustersGrDesNet(7,1,0.9,[-0.5 0.5],400,0)
```

– `LSinWaveBFGS` je příklad z kapitoly 5.2, který aproximuje čtvrtinu periody funkce sin metodou BFGS, která minimalizuje chybovou funkci (3.8. Vstupy této funkce jsou:

`NumHid` - počet skrytých neuronů,

`Int` - rozmezí pro počáteční nastavení vah,

`lambda` - regularizační parametr.

Výstupem je: `jval` - hodnota chybové funkce, `cas` - měří za jakou dobu se daná síť natrénovala.

Vhodné zadání této funkce je následující:

```
[jval,cas]=LSinWaveBFGS(1,[-0.1 0.1],0)
```

– `LSinWaveGrDes` je příklad z kapitoly 5.2, který aproximuje čtvrtinu periody funkce sin pomocí metody Gradient Dscent, která minimalizuje chybovou funkci (3.8. Vstupy této funkce jsou:
`NumHid` - počet skrytých neuronů,
`LR` - parametr učení,
`Mu` - moment sítě,
`Int` - rozmezí pro počáteční nastavení vah,
`MaxIter` - maximální počet iterací při trénování sítě,
`lambda` - regularizační parametr.
Výstupem je:
`jval` - hodnota chybové funkce,
`cas` - měří za jakou dobu se daná síť natrénovala.
Vhodné zadání této funkce je následující:

```
[jval,cas]=LSinWaveGrDes(1,1,0.9,[-0.5 0.5],100,0)
```

– `LSqrWaveBFGS` je příklad z kapitoly 5.2, který aproximuje jednotkový impuls pomocí metody BFGS, která minimalizuje chybovou funkci (3.8. Vstupy této funkce jsou:
`NumHid` - počet skrytých neuronů,
`Int` - rozmezí pro počáteční nastavení vah,
`lambda` - regularizační parametr.
Výstupem je:
`jval` - hodnota chybové funkce,
`cas` - měří za jakou dobu se daná síť natrénovala.
Vhodné zadání této funkce je následující:

```
[jval,cas]=LSqrWaveBFGS(10,[-0.1 0.1],0)
```

SPZ

Složka SPZ Obsahuje další tři složky, a to EU, NEW, OLD, každá z těchto složek má podobný obsah. V těchto složkách jsou roztříděné fotografie aut podle typu SPZ a také jsou zde soubory na lokalizaci SPZ a segmentaci znaků SPZ. Ve složkce EU je navíc funkce `trainsamples.m` a dva datové soubory `samp2` a `targ2`. Tyto soubory si popíšeme později popíšeme.

Obsah složky SPZ	
M-soubory	Data
`ImMaxThresh`	`samp2`
`SPZSegment`	`targ2`
`trainsamples`	`EUSAM images`
	`NEWSAM images`
	`SAM images`

- **M-soubory**:
  - `ImMaxThresh` je m-soubor, který lokalizuje SPZ. Nejprve načte všechny SPZ najednou a postupně lokalizuje SPZ.
  - `SPZSegment` je m-soubor, který segmentuje jednotlivé znaky SPZ. Obdobně jako předešlý m-soubor načte všechny snímky a postupně segmentuje znaky jedné SPZ za druhou.
  - `trainsamples` je funkce, která vytváří trénovací množinu a množinu daných výstupů pro rozpoznání SPZ.
- **Data**
  - `samp2` je vytvořená trénovací množina, vstup při trénování neuronové sítě různými metodami. Je zde vektorizováno a poskládáno za sebe 91 SPZ, tedy 378 znaků.
  - `targ2` soubor daných výstupů jednotlivých SPZ.
  - `EUSAM, NEWSAM, SAM images` jsou fotografie aut v EU, resp., NEW, resp., OLD složkách.

Training Net V této složce máme několik funkcí, které můžeme použít pro natrénování vlastních neuronových sítí pro rozpoznávání SPZ. Jsou to funkce `TrainGrDes, TrainLBFGS, TreinLGrDes`. Výstupem těchto trénovacích funkcí jsou natrénované váhy, které se uloží v této složce. Tyto váhy potom můžeme zkopírovat do složky **Alpha Version**, kde je můžeme použít pro vlastní rozpoznání SPZ. Více o tom, jaké parametry jsme zadávali a jakých výsledků jsme dosahovali při trénování, je možné najít v kapitole 5.3.

Obsah složky Training Net		
Pomocné funkce	Spouštěcí funkce	`Data`
`BackProp`	`TrainGrDes`	`samp2`
`BwProp`	`TrainLBFGS`	`targ2`
`costfunction`	`TrainLGrDes`	
`FwProp`		
`GenWeights`		
`grcostfunction`		
`PatternPass`		
`StructNet`		

- **Pomocné funkce**:
  - Všechny pomocné funkce, které zde v této složce máme jsme si popsali již dříve, proto si je nebudeme dále rozebírat.
- **Spouštěcí funkce**:
  - `TrainGrDes` je funkce, která minimalizuje chybovou funkci (3.8) pomocí metody Gradient Descent. Vstupy této funkce jsou:
    `NumHid` - počet skrytých neuronů,
    `LR` - parametr učení,
    `Mu` - moment sítě,
    `Int` - rozmezí pro počáteční nastavení vah,
    `MaxIter` - maximální počet iterací při trénování sítě,
    `tol` - tolerance chyby.

Výstupem je:

`MSE` - hodnota chybové funkce (3.8),

`cas` - měří za jakou dobu se daná síť natrénovala,

`W150,W250` - natrénované váhy.

Vhodné zadání této funkce je následující:

```
[MSE,cas,W150,W250]=TrainGrDes(50,0.05,0.9,[-0.1 0.1],[],1e-5)
```

– `TrainLBFGS` je funkce, která minimalizuje chybovou funkci (3.63) pomocí metody BFGS. Vstupy této funkce jsou:

`NumHid` - počet skrytých neuronů,

`LR` - parametr učení,

`Mu` - moment sítě,

`Int` - rozmezí pro počáteční nastavení vah,

`MaxIter` - maximální počet iterací při trénování sítě,

`lambda` - regularizační parametr.

Výstupem je:

`jval` - hodnota chybové funkce,

`cas` - měří za jakou dobu se daná síť natrénovala,

`WBF150,WBF250` - natrénované váhy.

Vhodné zadání této funkce je následující:

```
[jval,cas,WBF150,WBF250]=TrainLBFGS(50,[-0.1 0.1],0.01)
```

– `TrainLGrDes` je funkce, která minimalizuje chybovou funkci (3.63) pomocí metody Gradient Descent. Vstupy této funkce jsou:

`NumHid` - počet skrytých neuronů,

`LR` - parametr učení,

`Mu` - moment sítě,

`Int` - rozmezí pro počáteční nastavení vah,

`MaxIter` - maximální počet iterací při trénování sítě,

`lambda` - regularizační parametr.

Výstupem je:

`jval` - hodnota chybove funkce,

`cas` - měří za jakou dobu se daná síť natrénovala,

`WGR150,WGR250` - natrénované váhy.

Vhodné zadání této funkce je následující:

```
[jval,cas,WGR150,WGR250]=TrainLGrDes(50,0.5,0.9,[-0.1 0.1],1000,0.001)
```

Literatura

[Zelinka] Zelinka, I.: *Umělá inteligence* I, Zlín 1997

[Kröse,Smagt] Kröse, B., Van der Smagt, P.: *An Introduction to Neural Networks*. University of Amsterdam 1996

[González,Woods,Eddins] González, R.C., Woods, R.E., Eddins, S.L., *Digital Image Processing Using MATLAB*, Pearson Prentice Hall 2003

[MATLAB] *Image Procesing Toolbox: User's Guide* R2011b, The MathWorks, September 2011

[Ranganathan] Ranganathan, A., *The Levenberg - Marquardt Algorithm*

[Biskup] Biskup, R., *Možnosti neuronových sítí*, Česká zemědělská univerzita v Praze, Provozně ekonomická fakulta, Praha 2009

[LPR] Osorio, C.G., Francsico, J., Pastor, D., Rodríguez, J. J., Maudes, J., *License Plate Recognition, New Heuristics and a Comparative Study of Classifiers*

[Samarasinghe] Samarasinghe, S., *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition*, 2007 Taylor & Francis Group, LLC

[Kočvara] Kočvara, M., *Algoritmy numerické optimalizace*, 2004

[SPZ normy] *Typy tabulek s registrační značkou po* 1.6.2004

[Roweis] Roweis, S., *Levenberg-Marquardt Optimization*

Internetové odkazy

- `http://www.mdcr.cz/NR/rdonlyres/D59EB03B-3985-4C7F-BB09-1D36FBBEBA7C/0/regtab.pdf`
- `http://www.fi.muni.cz/usr/jkucera/pv109/2000/xneudert.html`
- `http://www.ml-class.org`
- `http://neuron.felk.cvut.cz/courseware/html/chapters.html`
- `http://cs.wikipedia.org/wiki/St%C3%A1tn%C3%AD_pozn%C3%A1vac%C3%AD_zna%C4%8Dky_v_%C4%8Cesku`
- `http://holehouse.org/mlclass/07_Regularization.html`
- `http://cs.wikipedia.org/wiki/RGB`
- `http://en.wikipedia.org/wiki/Logistic_regression`