Univerzita Palackého v Olomouci
Přírodovědecká fakulta

BAKALÁŘSKÁ PRÁCE

2012 **Michal Trněný**

Přírodovědecká fakulta Univerzity Palackého v Olomouci
Katedra matematické analýzy a aplikací matematiky

Analýza obrazu pomocí neuronových sítí
Image analysis by means of neural networks

Michal Trněný

Bakalářská práce

Vedoucí diplomové práce:
RNDr. Tomáš Fürst, Ph.D.
Rok odevzdání: 2012

Vypracoval:
Michal Trněný
MAP, 3. rocník

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a všechny použité zdroje jsem uvedl v seznamu literatury.

V Olomouci dne 16.4.2012

Obsah

1 Úvod

1.1 Proč vznikla tato práce

Z několika různých důvodů jsem se začal zajímat o neuronové sítě. Kolega Fürst se v té době zabýval mimo jiné analýzou obrazu. Protože neuronové sítě lze použít také na analýzu obrazu, shodli jsme se na tématu analýza obrazu pomocí neuronových sítí.
První přiblížení, co to jsou neuronové sítě: Neuronová síť je shluk různě pospojovaných neuronů (nervových buněk) a tím shlukem různě procházejí informace například ve formě elektrických impulsů. Obvykle vedou do sítě vstupy (např. hluk blížícího se nákladního vlaku kódovaný do elektrických impulsů) a ven ze sítě vedou výstupy (např. do svalů, abychom mohli uhnout). Dále budu značit neuronové sítě zkratkou NN (z angl. Neural Network(s)).
Tady je důvod, proč se zabývám neuronovými sítěmi. Zajímá mě, jak funguje lidská mysl a co to vůbec je. Mysl je v mozcích a mozky jsou složité NN. V mozcích se odehrává mimo jiné veškerá matematika. Pochopení, jak funguje mysl, nám umožní stavět inteligentní roboty. Takoví roboti nás pak mohou sledovat, učit se naši práci a když budeme potřebovat, tak nám pomohou. Abychom však co nejpřesněji pochopili funkce mysli, musíme k tomu použít různé matematické nástroje, jako třeba teorii pravděpodobnosti, statistiku, dynamické systémy, grafy a sítě, logiku, optimalizaci, diferencilní rovnice, teorii složitosti a vyčíslitelnosti a zároveň poznatky ze zdánlivě nesouvisejících oborů: informatiky, psychologie, biologie, chemie, fyziky.
Kromě toho umělé NN mají široké uplatnění. Dají se využívat jako asociativní paměti, jako zařízení k rozhodování (klasifikace obrazů), předvídání budoucnosti (aproximace funkcí), řízení (roboti) a dalším věcem, které nečiní lidem potíže, zatímco sériově pracujícím počítačům ano.

1.2 Cíle práce

V rámci Programu bezpečnostního výzkumu České republiky v letech 2010 – 2015 (BV II/2-VS) je řešen projekt, jehož cílem je vývoj metody vyšetření buněk vlasových a chlupových folikulů (lidských a zvířecích) ke stanovení míry expozice organismu ionizujícímu záření nebo genotoxickým látkám (viz [6]). Na základě naměřených dat pak tato neinvazivní metoda umožní predikci reakce postiženého organismu pro snadnější volbu vhodné léčebné strategie, tento přístup navíc umožní souběžné stanovení míry nebezpečnosti kontaminovaného území.
Vedoucí práce se podílí na vývoji specializované softwarové aplikace pro kvantifikaci míry poškození DNA s využitím standardního osobního počítače, fluorescenčních signálů jader buněk a automatické analýzy digitální fotografie mikroskopického obrazu buněk chlupových folikulů.
Máme dva typy fotografie získaných buněk. První typ fotografe obsahuje desítky

obarvených jader buněk. Nazývejme ho jádra. Na druhém typu fotografe jsou vidět fluorescenčně označené zlomy DNA jader buněk. Druhému typu fotografe říkejme signál. Celková intenzita signálu v dané buňce vypovídá o jejím poškození. Proto je třeba zjistit intenzity signálu v jednotlivých buňkách.
Na fotografii s jádry je kromě jader ještě šum pozadí. Některá jádra jsou nalepená na sobě ve dvojicích nebo v hloučku. My bychom chtěli získat obrazy jader na černém pozadí a pokud možno jader jednotlivých buněk. Proto provedeme následující úpravy fotografií:

- **Prahování** slouží k odstranění šumu - hodnoty pixelů fotografie menší než určitý práh se změní na nulu a větší se změní na jedničku (práh byl získán pomocí matlabovské funkce graythresh a prahování bylo provedeno pomocí funkce im2bw)

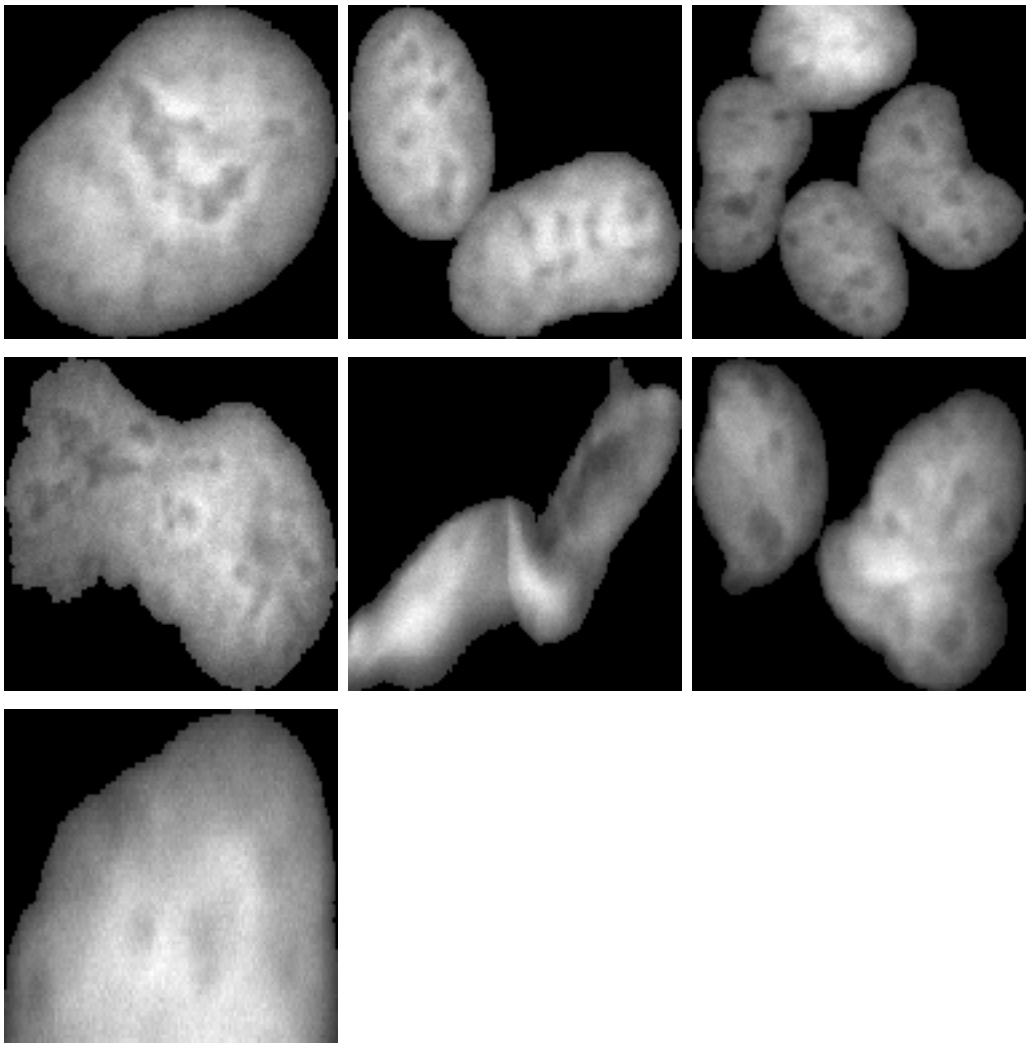
- **Segmentace** odliší od sebe komponenty souvislosti různými číselnými hodnotami, to lze provést pomocí matlabovské funkce bwlabel

- **Rozklipování** rozdělí fotografii s jádry na několik obrazů (klipů) a na každém z obrazů je jeden souvislý objekt, jedna komponenta souvislosti.

Na těch malých obrazech je většinou jedno jádro. Někdy je tam více jader, která dohromady tvoří jedinou souvislou oblast.

Cílem této práce je sestavit počítačový program, který automaticky roztřídí obrazy podle toho, je-li na nich jediné jádro nebo více jader.

Obrazy, na nichž je jediná buňka, resp. jádro jediné buňky, nazývejme od teď Jednotky. Obrazy, na nichž jsou aspoň dvě jádra, nazývejme Vícečetné. Na některých obrazech jsou dvě jádra. Tyto obrazy občas nazývám Dvojicemi. Dvojice jsou speciálním případem Vícečetných. Některé obrazy obsahují takové útvary, o nichž neumím říct, jsou-li to Jednotky nebo Vícečetné. Nepoznám to a takovým obrazům říkám Nepoznané.
Třídící program musí umět poznat, co je na vstupním obraze. Pro představu, co se zde může vyskytovat, uvádím Obrázek 1. V levém horním čtverci je jediné jádro (Jednotka). V prostřední horní části jsou dvojice jader (Dvojice, patří mezi Vícečetné). V pravé horní části je čtveřice jader (patří mezi Vícečetné) Na celém prostředním řádku jsou jádra o nichž se dá těžko říct, je-li na nich jedno deformované nebo více (řadím je do kategorie Nepoznané). Vlevo dole je Uřízlé jádro (taková jádra ležící na okraji velké fotografie nás nezajímají).



Obrázek 1: Jádra buněk, z leva do prava shora dolů označeno A, B, C, D, E, F, G

1.3 Možnosti řešení

Tady je několik možností, co udělat, aby počítač místo nás roztřídil obrazy na Jednotky a Vícečetné.

1.3.1 Nejpřesnější řešení

V paměti počítače je uložena množina M všech možných obrazů, takových, že každý obraz splňuje následující podmínky:

1. má rozměry 100 x 100 pixelů

2. každý pixel obrazu má některý odstín šedé barvy, jemuž odpovídá některá celočíselná hodnota od 0 do 255

3. ke každému obrazu je přiřazeno jedno z těchto tří slov: Jednotka, Vícečetná, Nepoznaná.

Třídící program pracuje následujícím způsobem: Na vstup přijde obraz O_1. O_1 je roztažen nebo smrštěn na obraz O_2 o velikosti 100 x 100 pixelů s hodnotami nabývajícími celých čísel od 0 do 255. Obraz O_2 je teď porovnáván se všemi obrazy z množiny M. Porovnávání probíhá tak dlouho, než je v paměti nalezen obraz O_3 spolu se slovem S. O_3 je stejný jako O_2. Výstupem tohoto programu je slovo S. Aby tento program fungoval vždy, tak by v paměti počítače muselo být uloženo $256^{10000} \doteq 2.5 \cdot 10^{24082}$ obrazů. To je číslo větší než počet atomů ve viditelném vesmíru. Podle zdroje [7] je ve viditelném vesmíru řádově 10^{78} atomů. Takže žádný počítač ve viditelném vesmíru není schopný pamatovat si všechny obrazy. A i kdyby byl, tak prohledávání by trvalo tak dlouho, že je nepraktické vůbec se pokoušet o výrobu takového programu. Praktičtější je následující způsob řešení.

1.3.2 Srovnávání vstupů s typickými obrazy v paměti

Definice 1 *Konečná podmnožina S prostoru $\mathbb{R}^n$ se nazývá shluk bodů v n-dimenzionálním prostoru. Častěji jen shluk.*

Definice 2 *Řekneme, že shluky S_1 a S_2 v $\mathbb{R}^n$ jsou lineárně oddělitelné, jestliže $\exists \mathbf{w} \in \mathbb{R}^{n+1}$ takový, že:*

$$\forall \mathbf{x} \in S_1 : w_1 \cdot x_1 + \ldots + w_n \cdot x_n + w_{n+1} \leq 0$$

$$\forall \mathbf{x} \in S_2 : w_1 \cdot x_1 + \ldots + w_n \cdot x_n + w_{n+1} > 0$$

Uvažujme nyní 10000-dimenzionální eukleidovský vektorový prostor a mějme v něm metriku definovanou pomocí eukleidovské normy tímto způsobem: $\varrho(x, y) = \|x - y\|$, kde x, y, jsou prvky výše zmíněného prostoru a $\|.\|$ je eukleidovská norma. Každý obraz O_3 můžeme chápat jako vektor, který má 10000 složek. Některé obrazy jsou si podobné více než jiné. Některé dvojice obrazů jsou si podobné odstíny šedé barvy a jejich rozložením, jiné jsou si podobné spíše tvarem jádra (příp. jader). Nahlížíme-li na obraz jako na vektor, tak můžeme čekat, že vektory podobných obrazů leží v prostoru s eukleidovskou normou blízko sebe (v kapitole 3.1 vytvoříme metriku vhodnější než je metrika zde použitá). Z 1000 Jednotek, vyberu jednu, jejíž součet vzdáleností od všech 999 zbývajících jednotek je nejmenší možný. Takto vybraný obraz nazvu zástupcem Jednotek. Podobným postupem ještě získám zástupce Vícečetných. Program bude fungovat tímto způsobem: Na vstup přijde obraz O_1. O_1 je změněn na obraz O_2 o velikosti 100×100

pixelů stejným způsobem jako v předchozí sekci. Probíhá výpočet vzdálenosti O_2 od zástupce Jednotek a od zástupce Vícečetných. Vypočítaná vzdálenost O_2 od zástupce Jednotek je d_1, od zástupce Vícečetných je d_2. Která vzdálenost je větší? Pokud $d_1 > d_2$, znamená to, že O_2 je blíže k zástupci Vícečetných a na výstupu se objeví slovo: Vícečetná. Pokud $d_1 < d_2$, na výstupu se objeví slovo: Jednotka. Pokud $d_1 = d_2$, na výstupu se objeví slovo: Nepoznaná. Tento program bude fungovat dobře, bude-li splněna tato podmínka: shluk vektorů Jednotek je lineárně oddělitelný od shluku vektorů vícečetných. Čím více budou oba shluky promíchané, tím horší výsledky bude program dávat.

1.3.3 Rozhodovací stromy

Předložme obrazy, které chceme roztřídit, lidskému expertovi. On dokáže o většině z nich říct, je-li to Jednotka nebo Vícečetná. Už od experta víme, který obraz patří do které třídy. Pak můžeme dostat sadu nových obrazů. Abychom o nich uměli rozhodnout, do které třídy patří který obraz, zkoumejme společné vlastnosti obrazů expertem označené jako Jednotky. Potom prozkoumejme společné vlastnosti obrazů expertem označené jako Vícečetné. Vlastnosti, kterými jsou charakterizovány obrazy, mohou být například maximum z délek stran obrazu (D) - měřeno v pixelech, solidita (S) a maximální úhel (U).
Solidita je podíl plochy masky obrazu a plochy jejího konvexního obalu. Maska obrazu je černobílý obraz, který vznikl z obrazu tím, že pixely s nenulovou hodnotou byly nahrazeny pixely s hodnotou jedna a všude jinde zůstaly nuly. Charakteristiku U získáme následujícím způsobem: Rektifikujeme hranici masky obrazu, tozn. nahradíme ji vhodnými úsečkami $\overline{B_1B_2}$, $\overline{B_2B_3}$, ..., $\overline{B_{n-1}B_n}$, $\overline{B_nB_1}$. Pro všechny úsečky musí platit, že $\max_{C \in H_i} \varrho(C, \overline{B_iB_{i+1}}) \leq tol$, kde C je bod části hranice, která leží mezi body B_i, B_{i+1}. H_i je množina bodů hranice ležící mezi body B_i, B_{i+1}, ϱ je eukleidovská vzdálenost, tol je vhodně zvolené kladné číslo (tzv. tolerance). Teď vypočítejme vnitřní úhel, který svírají dvě sousední úsečky. Tento výpočet proveďme pro všechny zbývající body rektifikace hranice. Ze všech vypočítaných vnitřních úhlů vyberme ten největší. Tím získáme charakteristiku maximální úhel.
Pozorováním charakteristik jsme zjistili, že většina Vícečetných má $D \geq d$, $S \leq s$, $U \geq u$. Teď vytvořme rozhodovací strom. Pro více informací o rozhodovacích stromech viz decision trees na straně 395 v knize [3].

1. Na vstup programu přišel obraz O_1, získáme jeho masku a posléze charakteristiky D, S, U.

2. Když $D \geq d$, tak O_1 je Vícečetná.

3. Když $D < d$ a

   (a) když $S \leq s$, tak O_1 je Vícečetná.

   (b) když $S > s$ a

      i. když $U \geq u$, tak O_1 je Vícečetná.

      ii. když $U < u$, tak O_1 je Jednotka.

1.3.4 Neuronové sítě

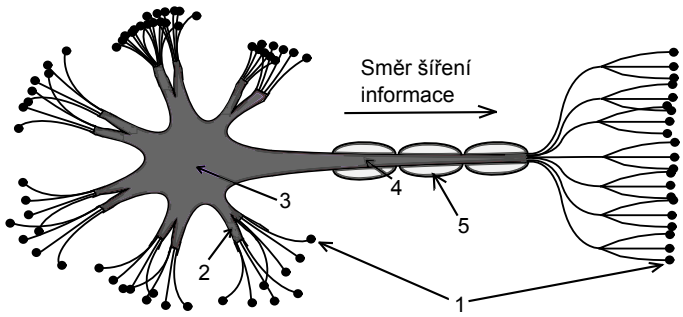
Protože chci vyzkoušet, jak si neuronové sítě poradí s tímto úkolem a protože jim chci lépe porozumět, budu se jim věnovat v následujících kapitolách tohoto textu.

2 Umělé neuronové sítě

Existuje několik výpočetních modelů, jejichž fyzické realizace nám umožňují řešit různé praktické problémy. Mezi výpočetní modely patří například Turingův stroj, von Neumannova architektura počítače, celulární automaty, neuronové sítě. Umělé neuronové sítě jsou modelem inspirovaným lidským mozkem, který zpracovává informace jiným způsobem než digitální sekvenční počítače [2]. Mozek je nelineární dynamický systém propojených neuronů, které zpracovávají informace paralelně. Na NN se lze dívat jako na černou skříňku, do níž vstupuje p čísel a z níž vystupuje q čísel. Je to nějaké zobrazení $NN : \mathbb{R}^p \to \mathbb{R}^q$ a my nemusíme na začátku vědět jaké přesně. Chtěli bychom, aby vektor konkrétních vstupních hodnot $(x_1, \ldots, x_p)$ nebo libovolný tomuto vektoru „blízký“ vektor se zobrazil na daný vektor výstupních hodnot $(y_1, \ldots, y_q)$. Toho lze aspoň přibližně docílit tím způsobem, že nastavíme jednu z několika vhodných kombinací parametrů sítě nebo necháme síť ať si parametry nastaví sama, pokud to dokáže. Procesu nastavování parametrů neuronové sítě se říká učení. Proces učení ukazuje schopnost sítě adaptovat své parametry a tak se přizpůsobovat vnějšímu prostředí - vstupům přizpůsobit výstupy, akcím reakce. To je asi nejnápadnější schopnost, která odlišuje NN od ostatních výpočetních modelů.

2.1 Skutečný neuron

Neuron je základní stavební kámen nervové soustavy živých organizmů. Je to buňka, která vypadá přibližně tak, jak ta na obrázku 2. Existují i jiné tvary neuronů, ale tento je typický. Pro nás podstatné části neuronu jsou synypse (1), dendrity (2), tělo neuronu (3) a neurit (4). Neuritu se také říká axon a je obalen tzv. myelinovu pochvou (5), která má izolační funkci. Z jiných neuronů, receptorů (např. sítnice očí) nebo vnějšího prostředí proudí do těla neuronu skrze dendrity



Obrázek 2: Typický neuron živých organizmů

elektrické impulsy. V těle se impulsy skládají. Pokud součet těchto impulsů překročí určitý práh, neuron vyšle svým neuritem elektrický impuls, který pokračuje přes synapse do dalších neuronů nebo do efektorů (např. svaly). Živý neuron, podobný tomu na obrázku 2, se stal inspirací pro matematický model neuronu.

2.2 Umělý neuron

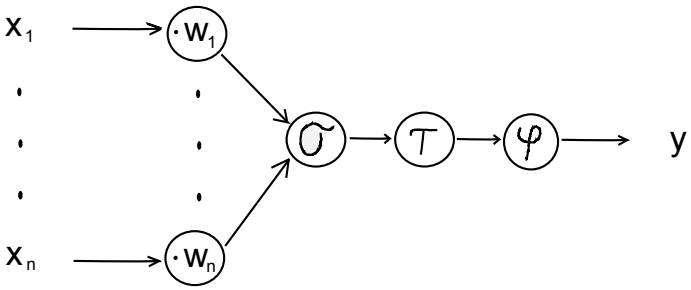
Umělý neuron je matematický model skutečného neuronu nebo počítačový program, který se chová jako neuron nebo nějaká fyzická součástka se stejnou funkcí jakou má neuron. Ve zbytku celé kapitoly 2 budu slovem neuron rozumět matematický model skutečného neuronu.

2.2.1 Obecný model neuronu

Obecný model neuronu je zobrazení $(\varphi \circ \tau \circ \sigma) : \mathbb{R}^n \to \mathbb{R}$ složené ze tří zobrazení σ, τ, φ, kde $\sigma : \mathbb{R}^n \to \mathbb{R}$, $\tau : \mathbb{R} \to \mathbb{R}$ a $\varphi : \mathbb{R} \to \mathbb{R}$. σ se nazývá agregační funkce, τ se nazývá prahovací funkce a φ se nazývá aktivační funkce. Neuron lze pak matematicky zapsat:

$$y = (\varphi \circ \tau \circ \sigma)(\mathbf{x}) = \varphi(\tau(\sigma(\mathbf{x}))) \tag{1}$$

Vektor $\mathbf{x} \in \mathbb{R}^n$ je n-tice číselných hodnot $(x_1, \cdots, x_n)$ vstupujících do neuronu. Skalár $s \in \mathbb{R}$ je výsledek agregační funkce σ, $s = \sigma(\mathbf{x})$, obvykle bývá touhle funkcí vážený součet vstupů daný rovnicí $s = x_1 w_1 + \ldots + x_n w_n$, kde w_i je i-tá složka váhového vektoru $\mathbf{w}$, $\mathbf{w} \in \mathbb{R}^n$. Dál s vstupuje do funkce τ, jejímž výsledkem je $p = \tau(s) = s - b$. $b \in \mathbb{R}$ je tzv. práh. Výsledek $p \in \mathbb{R}$ pokračuje tělem neuronu do aktivační funkce φ a ta mu přiřadí hodnotu $y = \varphi(p)$. $y \in \mathbb{R}$ je výstupní hodnotou neuronu. Aktivační funkcí φ může být v případě obecného neuronu jakákoliv reálná funkce jedné reálné proměnné, obvykle jí však bývá nějaká ohraničená, monotónní funkce, ale není to pravidlem (např. aktivační funkce perceptronu je ohraničená, ale není monotónní).
Každý model neuronu má stejné součásti, jaké jsou na obrázku 3. Těmi součástmi jsou: dendrity se synapsemi, tělo neuronu, neurit. Vstupní informace, t.j. číselné



Obrázek 3: Obecný model neuronu

hodnoty: $x_1, \ldots x_n$, vstupují skrze synapse do dendritů. Každá ze synapsí má nějakou váhu $w_1, \ldots w_n$. Váhu synapse dendritu lze chápat, jako její schopnost propouštět informace do (nebo z) neuronu. Váhami přenásobené vstupní hodnoty se šíří dál přes dendrity do těla neuronu. Tam jsou zpracovány agregační funkcí, jejíž výstup je zpracován prahovací funkcí. To, jakou neuron vyšle výstupní informaci y ven, závisí na aktivační funkci. y může pokračovat do dalších neuronů.

2.2.2 Konkrétní modely neuronu

McCulloch-Pittsův neuron
Roku 1943, inspirováni neurobiologií, Warren McCulloch a Walter Pitts představili světu první, nejjednodušší model neuronu [1]. Ten se skládá z n vstupních hran (dendrity), m inhibičních hran (dendrity), těla, a výstupní hrany (neurit). Do těla neuronu vstupují skrze hrany hodnoty $x_1, \ldots x_{n+m}$. Pro všechna i platí $x_i \in \{0; 1\}$. Hodnoty $x_1, \ldots, x_n$ procházejí vstupními hranami a hodnoty $x_{n+1}, \ldots, x_{n+m}$ procházejí inhibičními hranami. Vstoupí-li do libovolné inhibiční hrany hodnota 1, způsobí, že výstupem neuronu bude hodnota $y = 0$. Inhibiční hrany plní deaktivační funkci. Vstoupí-li současně do všech inhibičních hran hodnota 0, pak výstupem neuronu bude číslo $y = \varphi(s - b)$, kde $s = x_1 + \ldots + x_n$ je výsledek agregační funkce, $b \in \mathbb{R}$ je práh a aktivační funkce φ je definovaná následujícím způsobem: $\varphi(p) = 0$ pro $p < 0$ a $\varphi(p) = 1$ pro $p \geq 0$.
Agregační a aktivační funkce, počet dendritů, propojení těchto neuronů v síti jsou dány na začátku. Jedinými parametry, které lze měnit v síti sestavené z těchto neuronů jsou prahy a topologie sítě. Vhodným propojením dostatečně mnoha McCulloch-Pittsových jednotek lze získat všechny booleovské funkce (viz 2.2.3). Tento model se podobá běžným logickým hradlům.

Perceptron
Roku 1958 Americký psycholog Frank Rosenblatt navrhnul perceptron, což je výpočetní model obecnější než McCulloch-Pittsův neuron. Viz [1]. Perceptron se nejvíce podobá obecnému modelu neuronu. Do neuronu vstupují reálná čísla

$x_1, \ldots, x_n$. Procházejí přes synapse ohodnocené váhami $w_1, \ldots, w_n$, což jsou také reálná čísla. Váhami přenásobené vstupní informace pokračují do agregační funkce σ, kde se sečtou.

$$\sigma(x_1, \ldots, x_n) = x_1 w_1 + \ldots + x_n w_n \tag{2}$$

Výsledkem agregační funkce je číslo s, které pokračuje tělem neuronu než narazí na prahovací funkci τ, v níž je od hodnoty s odečtena hodnota b. Výsledek p prahovací funkce τ pokračuje na vstup aktivační funkce φ, která se definuje následujícím způsobem: $\varphi(p) = 1$ pro $p \geq 0$ a $\varphi(p) = 0$ pro $p < 0$. Je-li překročen práh, tj. je-li $w_1 x_1 + \ldots + w_n x_n \geq b$, pak $\varphi(p) = 1$ a znamená to, že neuron vysílá hodnotu 1. Pokud je $\varphi(p) = 0$, neuron vysílá hodnotu 0.
Protože práh b je nastavitelný parametr, podobně jako váhy $w_1, \ldots, w_n$, lze ho považovat za (n+1). váhu (přesněji: hodnotu $-b$ lze považovat za (n+1). váhu). Pokud použijeme následující konvenci, zjednoduší se zápis funkce perceptronu a perceptron se stane zobrazením složeným z dvojice zobrazení (σ, φ) místo z trojice (σ, τ, φ). Nechť $\mathbf{x} \in \mathbb{R}^n$ je vstupní vektor. Dál nechť $\mathbf{w} \in \mathbb{R}^{n+1}$ je váhový vektor, jehož poslední složka $w_{n+1} = -b$ je opačná hodnota prahu. Od teď chápejme agregační funkci σ jako zobrazení $\sigma : \mathbb{R}^n \to \mathbb{R}$ dané předpisem:

$$\sigma(\mathbf{x}) = (\mathbf{x}, 1) \cdot \mathbf{w} \tag{3}$$

kde $\cdot$ je skalární součin a $(\mathbf{x}, 1) = (x_1, \cdots, x_n, 1) \in \mathbb{R}^{n+1}$.
Proto můžeme na perceptron nahlížet jako na složené zobrazení $(\varphi \circ \sigma) : \mathbb{R}^n \to \mathbb{R}$ dané předpisem:

$$(\varphi \circ \sigma)(\mathbf{x}) = \varphi(\sigma(\mathbf{x})) = \varphi((\mathbf{x}, 1) \cdot \mathbf{w}) \tag{4}$$

Tento předpis bude užitečný při programové implementaci perceptronu v Matlabu.

Perceptron se spojitou aktivační funkcí
Protože hledání optimální kombinace vah sítě sestavené z neuronů předchozího typu se stalo obtížným problémem [1], začal se v 80. letech 20. století používat backpropagation algoritmus. Je to algoritmus založený na gradientní optimalizační metodě (metoda největšího spádu), která je použita na hledání minima chybové funkce. Více o backpropagation a chybové funkci je v kapitole 2.4. Abychom mohli metodu největšího spádu použít, musíme požadovat, aby chybová funkce byla spojitá a měla derivace alespoň prvního řádu. Chybová funkce bude spojitá když budou spojité také všechny aktivační funkce v uvažované neuronové síti. Proto vznikla potřeba mít neuron se spojitou aktivační funkcí.
Perceptron se spojitou aktivační funkcí má stejnou agregační funkci jako perceptron, ale aktivační funkce φ je spojitá (narozdíl od předchozího modelu, kde byla binární). Příkladem často používaných aktivačních funkcí může být: logistická funkce $y = 1/(1+e^{-p})$ nebo $y = \tanh(p)$, kde $p = (\mathbf{x}, 1) \cdot \mathbf{w}$. V případě první

aktivační funkce y nabývá hodnot z intervalu $(0, 1)$. V případě druhé aktivační funkce y nabývá hodnot z intervalu $(-1, 1)$.

Neuron s nelineární rozhodovací hranicí
Nechť $\mathbf{x} \in \mathbb{R}^n$. Mějme množinu D bodů $\mathbf{x}$, takových, že $\sigma(\mathbf{x}) = 0$. Množina D se nazývá rozhodovací hranice. Ve všech předchozích případech byla rozhodovací hranicí lineární $(n-1)$-dimenzionální nadrovina v n-dimenzionálním prostoru. Rozhodovací hranici bylo možné napsat rovnicí:

$$w_1 x_1 + \ldots + w_n x_n + w_{n+1} = 0 \tag{5}$$

Rozhodovací hranicí tohoto neuronu je kvadratická nadplocha v prostoru $\mathbb{R}^n$. Ta má předpis:

$$(\mathbf{x}, 1) \cdot \mathbf{W} \cdot (\mathbf{x}, 1)^T = 0 \tag{6}$$

kde $(\mathbf{x}, 1) = (x_1, x_2, \ldots, x_n, 1) \in \mathbb{R}^{n+1}$, $\mathbf{x}$ jsou vstupy do neuronu a $\mathbf{W}$ je váhová matice typu $(n+1) \times (n+1)$. V tomto případě má agregační funkce předpis $\sigma(\mathbf{x}) = (\mathbf{x}, 1) \cdot \mathbf{W} \cdot (\mathbf{x}, 1)^T$ a proto je lineární rozhodovací hranice speciálním případem nelineární rozhodovací hranice. Jediný neuron, jehož rozhodovací hranicí je elipsa, parabola nebo hyperbola s vhodnými parametry, například dokáže realizovat logickou funkci XOR (viz část 2.2.3), pro jejíž realizaci byly potřeba alespoň 3 neurony s lineárními rozhodovacími hranicemi uspořádané do dvou vrstev (viz část 2.3.2). Více informací o neuronech tohoto typu a sítích z nich sestavených lze najít v [2] nebo [4].

Je možné vymýšlet další a další typy neuronů podle potřeby. I celý shluk složitě propojených neuronů lze chápat jako jeden zvláštní neuron se složitou agregační a aktivační funkcí.

2.2.3 Co umí perceptron

Definice 3 *Nechť $n \in \mathbb{N}$. Mějme zobrazení $B : \{0; 1\}^n \to \{0; 1\}$. Zobrazení B se nazývá booleovská funkce. Pro funkci B se také používá termín logická funkce.*

Tady jsou některé booleovské funkce: NOT (7), AND (8), OR (9), XOR (10).

$$f_1(0) = 1, f_1(1) = 0 \tag{7}$$

$$f_2(0,0) = 0, f_2(0,1) = 0, f_2(1,0) = 0, f_2(1,1) = 1 \tag{8}$$

$$f_3(0,0) = 0, f_3(0,1) = 1, f_3(1,0) = 1, f_3(1,1) = 1 \tag{9}$$

$$f_4(0,0) = 0, f_4(0,1) = 1, f_4(1,0) = 1, f_4(1,1) = 0 \tag{10}$$

Definice 4 *Nechť pro každý bod* $\mathbf{x}$ *ze shluku* S_1 *platí* $\varphi((\mathbf{x},1)\cdot\mathbf{w})=0$ *a pro každý bod* $\mathbf{y}$ *ze shluku* S_2 *platí* $\varphi((\mathbf{y},1)\cdot\mathbf{w})=1$. *Pak řekneme, že perceptron s váhami* $\mathbf{w}$ *umí poznat, zda dostal na vstup souřadnice bodu ze shluku* S_1 *nebo ze shluku* S_2.

Máme-li dva shluky bodů v n-dim prostoru, shluky, které jsou lineárně oddělitelné, pak lze nastavit váhy synapsí perceptronu tak, aby uměl poznat, zda jsme mu dali na vstup souřadnice bodu ze shluku S_1 nebo ze shluku S_2. Tomu nastavování vah perceptronu se říká učení perceptronu. Jediný perceptron se umí naučit chovat jako některé logické funkce, například jako konjunkce (AND), disjunkce (OR), implikace, negace (NOT) atd. Uvažujme funkční předpis pro perceptron (rovnice 4) uvedený na konci odstavce o perceptronu v sekci 2.2.2. Označme $f(\mathbf{x})=\varphi(\sigma(\mathbf{x}))$. Logické funkce lze psát následujícím způsobem: Chceme-li získat funkci NOT, stačí do rovnice 4 dosadit $\mathbf{w}=(-1,0.5)$. To je perceptron s pouze jedním dendritem. AND lze realizovat pomocí perceptronu se dvěma vstupy dosadíme-li $\mathbf{w}=(1,1,-1.5)$. Aby se perceptron choval jako funkce OR, dosadíme do rovnice (4) $\mathbf{w}=(1,1,-0.5)$.
Nechť dva body se souřadnicemi $(0,0)$ a $(1,1)$ z prostoru $\mathbb{R}^2$ tvoří množinu S_1 a dva body se souřadnicemi $(0,1)$ a $(1,0)$ tvoří množinu S_2. Bodům množiny S_1 přiřazuje funkce f_4 hodnotu 0 a bodům množiny S_2 hodnotu 1. Množiny S_1 a S_2 nejsou lineárně oddělitelné. Proto jediný perceptron nezvládne poznat, který bod patří do které množiny a proto se neumí chovat jako logická funkce „buď a nebo“ (XOR).
Viděli jsme, že pouhou změnou vah synapsí perceptronu lze získat různé logické funkce. Jak nastavit váhy perceptronu, aby dělal to, co chceme, najdete v následujícím odstavci. Procesu nastavování (upravování, adaptování) vah se říká učení.

2.2.4 Jak se učí perceptron

Nechť A a B jsou lineárně oddělitelné shluky v prostoru $\mathbb{R}^n$. Množina A obsahuje vektory $\mathbf{a}^1,\mathbf{a}^2,\dots\mathbf{a}^p$ a množina B obsahuje vektory $\mathbf{b}^1,\mathbf{b}^2,\dots\mathbf{b}^q$. $C=A\cup B$.
Označme $T(\mathbf{x})$ odpověď experta na dotaz, je-li $\mathbf{x}$ ze shluku A nebo B. Matematicky lze znalost experta o příslušnosti $\mathbf{x}$ zapsat jako funkci $T:C\to\{0;1\}$, $T(\mathbf{x})=0$ pro všechna $\mathbf{x}\in A$ a $T(\mathbf{x})=1$ pro všechna $\mathbf{x}\in B$.
Označme y odpověď perceptronu na vstup $\mathbf{x}$.
Označme $\mathbf{w}(t)$ hodnotu váhového vektoru perceptronu v t-tém kroku algoritmu.

Teď bychom chtěli, aby se perceptron naučil od experta jeho znalost. Na to máme algoritmus, jehož t-tá iterace se skládá z následujících kroků:

1. Náhodně vybereme $\mathbf{x}$ z množiny C.

2. (a) Když $y < 0$ a zároveň $T(\mathbf{x}) = 0$, tak $\mathbf{w}(t+1) = \mathbf{w}(t)$.
   (b) Když $y \geq 0$ a zároveň $T(\mathbf{x}) = 0$, tak $\mathbf{w}(t+1) = \mathbf{w}(t) - (\mathbf{x}, 1)$.
   (c) Když $y \leq 0$ a zároveň $T(\mathbf{x}) = 1$, tak $\mathbf{w}(t+1) = \mathbf{w}(t) + (\mathbf{x}, 1)$.
   (d) Když $y > 0$ a zároveň $T(\mathbf{x}) = 1$, tak $\mathbf{w}(t+1) = \mathbf{w}(t)$.

3. Opakuj 1. a 2. krok dokud nebudou všechny vektory $\mathbf{x}$ správně klasifikovány.

Tomu se říká perceptronové učení. Jsou-li splněny shluky A a B lineárně oddělitelné, pak algoritmus konverguje vždy po konečném počtu u kroků k váhovému vektoru $\mathbf{w}(u)$ takovému, že nadrovina v prostoru $\mathbb{R}^n$ určená rovnicí $(\mathbf{x}, 1) \cdot \mathbf{w}(u) = 0$, odděluje od sebe shluky A, B. Důkaz tohoto tvrzení lze nalézt například v [1], s. 87, nebo v [2], s.139.

2.3 Architektury sítí a typy učení

Pokud NN chceme učit nebo chceme, aby se učila sama, už nestačí dívat se na ni jako na černou skříňku, do níž vstupuje vektor čísel $(x_1, \ldots, x_p)$ a z níž vystupuje vektor čísel $(y_1, \ldots, y_q)$. Musíme vědět, jakou má ta černá skříňka vnitřní strukturu. Struktura NN je určena tím, jaké neurony jsou v síti obsaženy, které neurony jsou spolu spojeny, do kterých neuronů vstupují čísla $x_1, \ldots, x_p$ a ze kterých neuronů vystupují čísla $y_1, \ldots, y_q$. Struktura sítě se běžně nazývá architektura sítě. Příklady různých architektur sítě můžete vidět na obrázku 4.

2.3.1 Třídění architektur

Architektury sítí lze třídit podle různých kritérií, mezi která patří mimo jiné účel využití architektury NN nebo jméno jejího objevitele. Uvádím zde třídění podle tří kritérií: podle vlastností neuronů v síti obsažených, podle spojení neuronů a šíření informací a podle typu učení. O těchto kritériích se domnívám, že jsou základní, protože vycházejí přímo ze struktury a funkcí NN.

Třídění podle vlastností neuronů:

1. **Agregační funkce** jejíž rozhodovací hranice je
   (a) Nadrovina: viz rovnice (5)
   (b) Obecná nadplocha: např. kvadratické nadroviny dané rovnicí (6)

2. **Aktivační funkce** s počtem výstupních hodnot
   (a) Konečným: např. binární funkce $\varphi : \mathbb{R} \to \{0; 1\}$, kde $\varphi(p) = 0$ pro $p < 0$ a $\varphi(p) = 1$ pro $p \geq 0$.

(b) Nekonečným: např. logistická funkce (viz sekce 2.2.2 odstavec Perceptron se spojitou aktivační funkcí), nebo $\varphi(p) = ap + b$, $a, b \in \mathbb{R}$

3. **Načasování výstupů z neuronů**

   (a) Synchronní

   Všechny neurony v síti vysílají své výstupy současně.

   (b) Asynchronní

   V síti existuje alespoň jeden neuron, který nevysílá své výstupní hodnoty současně s ostatními neurony.

4. **Rozložení druhů neuronů v síti**

   (a) Homogenní

   Všechny neurony mají stejné agregační a aktivační funkce a jsou všechny synchronizované.

   (b) Nehomogenní

   Alespoň jeden neuron v síti se svými výše zmíněnými vlastnostmi liší od ostatních neuronů.

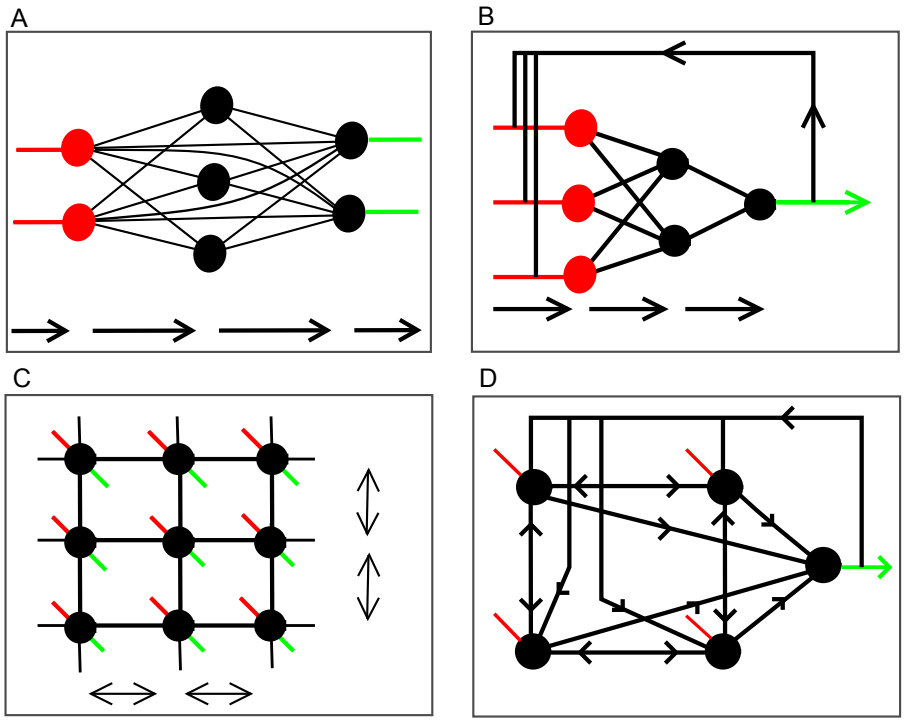
Třídění podle spojení neuronů a šíření informací:

1. **Sítě se šířením informace vpřed** (Feed-Forward networks) Neurony jsou v nich obvykle uspořádány do vrstev (viz sekce 2.3.2). V žádné vrstvě se nevyskytují dva nebo více neuronů, které by byly spolu propojeny. Jsou spolu spojeny jen neurony z i-té vrstvy s neurony z j-té vrstvy. $i \neq j$. Informace se šíří vždy z i-té vrstvy do j-té vrstvy, kde $i < j$. Jednu Feed-Forward síť můžete vidět na obrázku (4)A. Pro tyto sítě dál budu používat zkratku FF.

2. **Rekurentní sítě** jsou vrstevnaté sítě, které navíc obsahují cykly. To znamená, že některé (ne nutně všechny) výstupy neuronů j-té vrstvy jsou napojeny na vstupy neuronů té samé vrstvy nebo dokonce na vstupy některých (ne nutně všech) neuronů i-té vrstvy, kde $i < j$. Jedna taková síť je na obrázku (4)B.

3. **Mřížka** je speciálním případem rekurentních sítí. Je to jedna vrstva neuronů rekurentní sítě, v níž jsou spolu propojeny jen nejbližší sousední neurony a informace se po hranách šíří v obou směrech. Příkladem může být obrázek (4)C.

4. **Hybridní sítě** jsou libovolnou kombinací předchozích typů, např. obrázek (4)D.



Obrázek 4: Architektury NN podle spojení neuronů a šíření informací. Červenou barvou jsou označeny vstupní hrany a zelenou barvou výstupní hrany. Šipky udávají směr šíření informací.

Třídění podle typu učení (neboli míry autonomie adaptace vah):

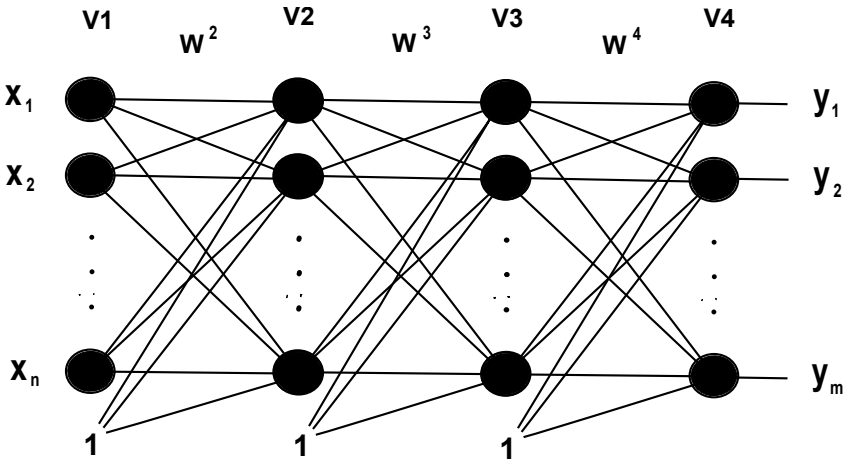
1. **S učitelem**: ke každému vstupu do sítě, který pochází z tréningové množiny, učitel (expert) přiřadí požadovaný výstup sítě. Ze skutečného a požadovaného výstupu spočítá chybovou funkci a podle toho různými metodami (např. backpropagation, viz 2.4.1) aktualizuje váhy spojů mezi neurony.

2. **S kritikem**: ke všem vstupům síť počítá výstup a k některým výstupům kritik přiřadí hodnocení výkonu sítě. To je kladné nebo záporné. V případě záporného hodnocení síť příště vyhodnotí daný vstup jinak. Tomuto typu učení se také říká reinforcement learning (posilované učení). [2]

3. **Samoorganizace**: každý neuron v mřížce sítě se pomocí procesu samoorganizace stává citlivým na nějakou oblast vstupního prostoru V. Citlivost k-tého neuronu na k-tou oblast O_k vstupního prostoru znamená, že přijde-li na vstup sítě $\mathbf{x} \in O_k$, tak největší výstupní hodnotu bude mít k-tý neuron v síti. Samoorganizace vah synapsí sítě probíhá tak, že po předložení vektoru $\mathbf{x}$ síti je nalezen neuron, jehož váhy jsou nejblíže tomuto vektoru. Následně váhy tohoto neuronu a jeho okolí jsou upraveny takovým způsobem, aby se přiblížily vektoru $\mathbf{x}$. Po tom je síti předložen další vektor $\mathbf{x}$. NN, které se tímto způsobem učí se nazývají samo-organizující se mapy. [1]

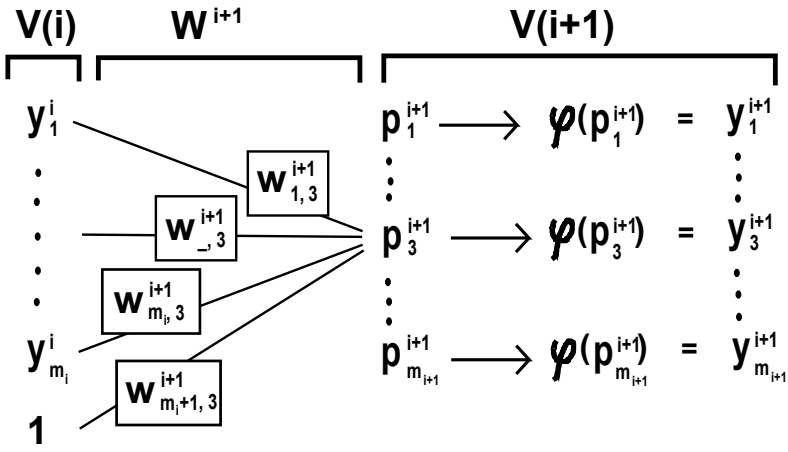
2.3.2 Vrstevnaté sítě

Vrstevnaté sítě jsou speciálním případem FF sítí [1] a je to asi nejpopulárnější architektura NN. V této práci je použita pro analýzu obrazu.



Obrázek 5: Vrstevnatá feed-forward síť

Na obrázku 5 můžete vidět síť, která obsahuje n vstupů $x_1, \ldots, x_n$, čtyři vrstvy V1, V2, V3, V4 a m výstupů $y_1, \ldots, y_m$. První vrstva V1 je vstupní a neobsahuje neurony, ale jsou to pouze hodnoty vstupující do sítě. Všechny ostatní vrstvy už obsahují neurony a v i-té vrstvě jich je m_i, $i > 1$. Pro vrstevnaté FF sítě platí, že výstupy i-té vrstvy procházejí přes synapse dendritů neuronů $(i+1)$. vrstvy. Váhy synapsí mezi výstupy i-té vrstvy a neurony $(i+1)$. vrstvy jsou vyjádřeny maticí $\mathbf{W}^{i+1}$. V každé vrstvě, kromě té poslední, se nachází tzv. fixovaný vstup, který vždy nabývá hodnoty 1 a má význam ve výpočtech agregačních funkcí neuronů následující vrstvy.



Obrázek 6: Dvě sousední vrstvy feed-forward sítě podrobněji

Výpočet probíhá následujícím způsobem:

1. Pokud V(i) není výstupní vrstvou, tak řádkovému vektoru $\mathbf{y}^i$ výstupů neuronů i-té vrstvy je přidána jednička navíc, takže vznikne vstupní vektor $(\mathbf{y}^i, 1)$ pro následující vrstvu neuronů.

2. Složky vektoru $(\mathbf{y}^i, 1)$ procházejí přes synapse dendritů neuronů následující vrstvy, jejichž váhy jsou určené maticí $\mathbf{W}^{i+1}$ sestavenou ze sloupcových vektorů $\mathbf{w}_j^{i+1} = (w_{1,j}^{i+1}, \ldots, w_{m_i,j}^{i+1}, w_{m_i+1,j}^{i+1})$ a j-tý sloupec této matice obsahuje váhy synapsí j-tého neuronu v $(i+1)$. vrstvě. Viz obrázek 6. Takže výsledek agregačních funkcí $(i+1)$. vrstvy je řádkový vektor $\mathbf{p}^{i+1} = (p_1^{i+1}, \ldots, p_{m_{i+1}}^{i+1})$ a vypočítá se následujícím způsobem: $\mathbf{p}^{i+1} = (\mathbf{y}^i, 1)\mathbf{W}^{i+1}$.

3. Výstupy neuronů $\mathbf{y}^{i+1}$ vrstvy V(i+1) jsou vypočítány aktivačními funkcemi φ neuronů, takže $\mathbf{y}^{i+1} = \varphi(\mathbf{p}^{i+1}) = (\varphi(p_1^{i+1}), \ldots, \varphi(p_{m_{i+1}}^{i+1}))$.

V tělech neuronů FF sítí se často užívá aktivační funkce logistická. Já budu kvůli symetrii používat funkci $y = \tanh(p)$. Všechny váhy v síti obsažené, lze zapsat do jediného vektoru $\mathbf{W} \in \mathbb{R}^Z$, kde

$$Z = \sum_{i=1}^{b-1} (m_i m_{i+1} + m_{i+1}) \tag{11}$$

Vektoru $\mathbf{W}$ bude využito v následující části textu.

2.4 Back propagation

Vrstevnatou FF síť můžeme chápat jako zobrazení, které prvkům $\mathbf{x}$ z prostoru $\mathbb{R}^n$ přiřazuje prvky $\mathbf{y}$ prostoru $\mathbb{R}^m$, kde $m = m_b$ udává počet neuronů v poslední vrstvě sítě a b počet vrstev. Na vstup sítě dáme vektor $\mathbf{x}$, ten je zpracován sítí a na výstupu se objeví vektor $\mathbf{y}$. Mějme v prostoru $\mathbb{R}^n$ q shluků $S_1, \ldots, S_q$. Předpokládejme, že FF síť má daný počet b vrstev, v každé vrstvě daný počet m_i neuronů, je to synchronní a homogenní síť sestavená z neuronů s lineárními rozhodovacími hranicemi a aktivačními funkcemi tangens hyperbolický (tanh). Chceme, aby pro všechna i od 1 do q síť zobrazovala všechny prvky shluku S_i na vektor $\mathbf{y}_i$. Jinými slovy: chceme, aby síť poznala, ze kterého shluku pochází vektor, který jsme jí dali na vstup. Jediné, co můžeme měnit, jsou váhy synapsí a prahy. Jak nastavit tyto volné parametry, aby FF síť dělala to, co po ní chceme? Jak měnit hodnoty vah synapsí? Dělá se to pomocí algoritmů, kterým se říká učení neuronové sítě. Je spousta různých a různě efektivních algoritmů. Ukážu jeden základní. Jmenuje se algoritmus zpětného šíření chyby neboli Back Propagation (BP). Jedná se o učení s učitelem.

My chceme, aby síť uměla co nejlépe klasifikovat předložené vzorky $\mathbf{x}$. Výstup $\mathbf{y}$ sítě závisí na vstupu $\mathbf{x}$ do sítě, na architektuře sítě a na aktuálním vektoru $\mathbf{W}$

všech vah, které jsou v síti obsaženy. To, jak moc se liší skutečný výstup $\mathbf{y}$ sítě od očekávaného výstupu $\mathbf{d}$, udává dílčí chybová funkce $E_{\mathbf{x}}$ pro vzorek $\mathbf{x}$:

$$E_{\mathbf{x}}(\mathbf{W}) = \frac{1}{2}(\mathbf{y} - \mathbf{d})^2 = \frac{1}{2}(\mathbf{y}(\mathbf{x}, \mathbf{W}) - \mathbf{d}(\mathbf{x}))^2 \tag{12}$$

Rádi bychom, aby chyba $E_{\mathbf{x}}$ byla nejmenší možná pro každý vzorek, který kdy na vstup sítě přijde. To se nám nepodaří, protože máme k dispozici jen omezený počet vzorků, na nichž lze síť trénovat. Proto se musíme spokojit se skromnějším cílem. Tím je minimalizace celkové chybové funkce E sítě pro všechny vzorky tréninkové množiny. Celková chybová funkce sítě je součtem dílčích chybových funkcí:

$$E = \sum_{i=1}^{r} \frac{1}{2}(\mathbf{y}_i - \mathbf{d}_i)^2 \tag{13}$$

kde r je počet vzorků v tréninkové množině, $\mathbf{y}_i$ je odpověď sítě na i-tý vzorek $\mathbf{x}_i$ a $\mathbf{d}_i$ je požadovaný výstup sítě, čili to, jak by měla síť odpovědět, kdyby správně fungovala. Chyba E je závislá na proměnných $w^i_{j,k}$, což jsou složky vektoru $\mathbf{W}$. Pojďme hledat bod lokálního minima $\mathbf{W}^*$ funkce E.
Použijme k tomu metodou největšího spádu (m.n.s.) [8] s pevným krokem γ. Podle m.n.s. bude posloupnost bodů $\mathbf{W}(t)$ v prostoru vah dána rekurentně vzorcem:

$$w^i_{j,k}(t+1) = w^i_{j,k}(t) - \gamma \cdot \frac{\partial E}{\partial w^i_{j,k}} \tag{14}$$

kde $w^i_{j,k}(t)$ je složka vektoru $\mathbf{W}$ v t-té iteraci. Krok γ, tzv. učební konstanta [1] je kladné reálné číslo, a že je to krok pevný znamená, že po celou dobu hledání minima, bude mít tutéž velikost, jakou měl na začátku.

Algoritmus BP:
Vhodné $w_{i,j,k}$ algoritmus hledá tak dlouho, než je E menší než předem zadaná hodnota E_z nebo než uplyne uživatelem zadaný počet T epoch. $T \in \mathbb{N}$. Epocha je jedna iterace v minimalizaci chybové funkce. Během jedné epochy jsou síti předloženy všechny učební vzorky. Abychom mohli užít vzorec (14) co nejefektivněji, vyjádřeme si, čemu se rovná derivace chybové funkce E podle $w^i_{j,k}$.
Nejdříve však pomocí řetězového pravidla pro derivaci složené funkce derivujme funkci E podle váhy $w^b_{i,j}$ j-tého neuronu v poslední vrstvě (v b-té).

$$\frac{\partial E}{\partial w^b_{i,j}} = (y^b_j(p^b_j) - d_j)\frac{\mathrm{d}y^b_j}{\mathrm{d}p^b_j}\frac{\mathrm{d}p^b_j}{\mathrm{d}w^b_{i,j}} \tag{15}$$

Rovnici (15) lze psát také takto:

$$\frac{\partial E}{\partial w_{i,j}^b} = (y_j^b - d_j)(1 - \tanh^2(p_j^b))y_i^{b-1} \tag{16}$$

kde jsme využili vztahu $(\tanh(p))' = (1 - \tanh^2(p))$. Pojďme se nyní podívat na derivaci E podle váhy $w_{i,j}^{b-1}$ j-tého neuronu v $(b-1)$. vrstvě.

$$\frac{\partial E}{\partial w_{i,j}^{b-1}} = \sum_{k=1}^{m_b}(y_k^b(p_k^b) - d_k)\frac{\mathrm{d}y_k^b}{\mathrm{d}p_k^b}\frac{\mathrm{d}p_k^b}{\mathrm{d}y_j^{b-1}}\frac{\mathrm{d}y_j^{b-1}(p_j^{b-1})}{\mathrm{d}p_j^{b-1}}\frac{\mathrm{d}p_j^{b-1}}{\mathrm{d}w_{i,j}^{b-1}} \tag{17}$$

Rovnici (17) lze psát také takto:

$$\frac{\partial E}{\partial w_{i,j}^{b-1}} = \sum_{k=1}^{m_b}(y_k^b - d_k)(1 - \tanh^2(p_k^b))w_{j,k}^b(1 - \tanh^2(p_j^{b-1}))y_i^{b-2} \tag{18}$$

Podobně lze pokračovat tak dlouho, než dojdeme k derivacím funkce E podle vah mezi první a druhou vrstvou. Všimněme si, jak lze zjednodušit zápis rovnice (15) a rovnice (17). Podle [1] zaveďme označení

$$\delta_k^b = (y_k^b(p_k^b) - d_k)\frac{\mathrm{d}y_k^b}{\mathrm{d}p_k^b} \tag{19}$$

potom rovnici (15) lze psát:

$$\frac{\partial E}{\partial w_{i,j}^b} = \delta_j^b y_i^{b-1} \tag{20}$$

a rovnici (17) lze psát:

$$\frac{\partial E}{\partial w_{i,j}^{b-1}} = \sum_{k=1}^{m_b}\delta_k^b w_{j,k}^b\frac{\mathrm{d}y_j^{b-1}(p_j^{b-1})}{\mathrm{d}p_j^{b-1}}\frac{\mathrm{d}p_j^{b-1}}{\mathrm{d}w_{i,j}^{b-1}} \tag{21}$$

Když rovnici (21) napíšeme následujícím způsobem,

$$\frac{\partial E}{\partial w_{i,j}^{b-1}} = \delta_j^{b-1} y_i^{b-2} \tag{22}$$

dostaneme vztah pro δ_j^{b-1}:

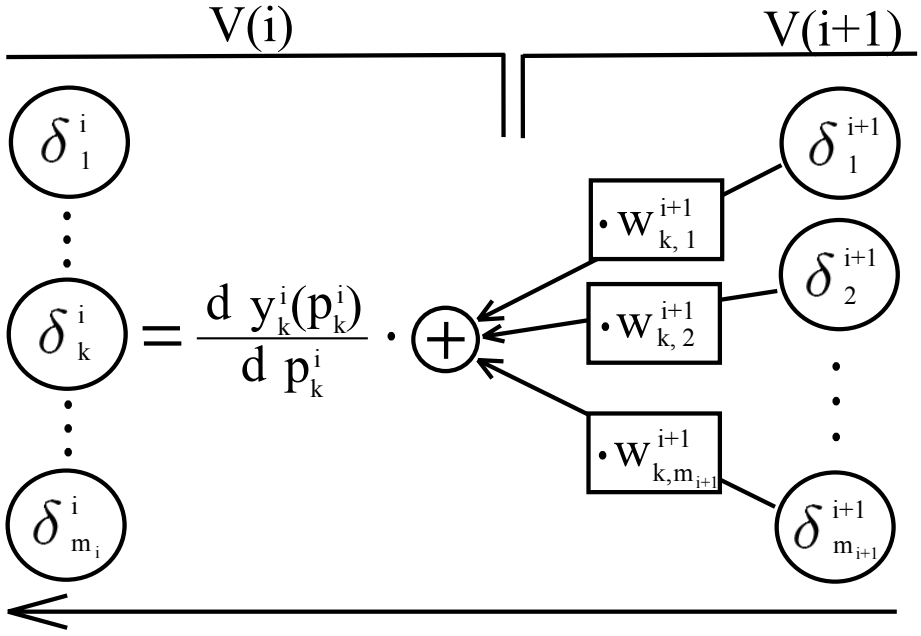
$$\delta_j^{b-1} = \sum_{k=1}^{m_b} \delta_k^b w_{j,k}^b \frac{\mathrm{d}y_j^{b-1}(p_j^{b-1})}{\mathrm{d}p_j^{b-1}} \tag{23}$$

A když ho zobecníme pro i-tou vrstvu (derivováním E můžete ověřit, že to platí), dostaneme rekurentní vztah pro δ_k^i:

$$\delta_k^i = \frac{\mathrm{d}y_k^i(p_k^i)}{\mathrm{d}p_k^i} \sum_{n=1}^{m_{i+1}} w_{k,n}^{i+1} \delta_n^{i+1} \tag{24}$$

Tímto způsobem se vypočítají delty ve všech vrstvách neuronů od poslední až do druhé. Známe-li delty, můžeme s jejich pomocí vypočítat, jak budou vypadat parciální derivace E podle $w_{j,k}^i$.

$$\frac{\partial E}{\partial w_{j,k}^i} = \delta_k^i y_j^{i-1} \tag{25}$$



Obrázek 7: Rekurentní výpočet delt v i-té vrstvě pomocí delt z (i+1). vrstvy

S využitím vzorců (19), (24), (25) a (14) lze pak zjistit všechny váhy $w_{j,k}^i(t+1)$ synapsí dendritů FF sítě v $(t+1)$. iteraci.
Algoritmus BP lze provést dvěma způsoby: dávkově (off-line) nebo on-line. Dávkový trénink probíhá tak, že v jediné epoše jsou síti předloženy všechny tréninkové

vzorky, vypočítá se celková chyba a podle toho si síť upraví váhy. On-line trénink sítě znamená, že síť dostane pouze jeden učební vzorek v jedné epoše, je vypočtena dílčí chyba a podle toho jsou upraveny váhy v síti.
V obou způsobech provedení BP se postupuje následujícím způsobem:

1. Na vstup FF sítě přijde vektor $\mathbf{x}$ a spolu s ním očekávaný výstup $\mathbf{d}$. Síť zpracuje vektor $\mathbf{x}$. Výstupem sítě je vektor $\mathbf{y}$.

2. Vektor $\mathbf{y}$ je porovnán s $\mathbf{d}$ a vypočítá se chybová funkce E.

3. Pomocí vzorce (19) se vypočítají delty v poslední vrstvě neuronů.

4. Pomocí vzorce (24) se vypočítají delty postupně ve všech předchozích vrstvách neuronů.

5. S využitím delt vypočítaných v každé vrstvě neuronů a s využitím vzorců (25) a (14) se upraví (aktualizují) váhy.

6. Krokem 5. končí jedna epocha. Kroky 1. až 5. jsou prováděny tak dlouho než je chyba sítě E menší než uživatelem zadaná chyba E_z nebo než je dosažen uživatelem zadaný počet T epoch.

Delty se počítají od konce (od výstupů) a šíří se na začátek (ke vstupům), to zn. proti směru šíření vstupních informací sítí. A o tom je celý algoritmus BP. Usnadní nám to výpočty v metodě největšího spádu. Algoritmu BP se také říká delta pravidlo kvůli šíření chyb delta sítí od konce zpět. Viz orázek 7.
Není zaručeno, že posloupnost vektorů W určená pomocí BP vždy dokonverguje do nějakého budu minima funkce E, a když někam dokonverguje, tak to pravděpodobně bude do oblasti s velmi malým gradientem nebo k nějakému lokálnímu minimu.
To, po kolika epochách dokonverguje BP do bodu $\mathbf{W}^*$ s přijatelnou chybou, závisí na vhodné volbě počátečních vah $\mathbf{W}$, učební konstantě γ, typu učení (on/off), rozložení shluků a v případě on-line tréninku i na pořadí učebních vzorků.
Podle některých zdrojů, např. [5], je on-line tréning obecně rychlejší než off-line tréning a také je díky němu snadnější vyhnout se plýtkým lokálním minimům chybové funkce a dokonvergovat do některého hlubšího minima. Při on-line tréninku totiž nemusí být směr určený vektorem $\overrightarrow{W(t)W(t+1)}$ stejný jako směr největšího spádu (jako u m.n.s.) a směr $\overrightarrow{W(t)W(t+1)}$ ve váhovém prostoru s měnícím se t „náhodně osciluje“ kolem záporné hodnoty gradientu celkové chybové funkce.

3 Analýza obrazu

Máme soubor obrazů, které chceme analyzovat (t.j. zjistit z nich informaci o tom, co se na nich nachází), udělat to co nejjednodušeji, s využitím poznatků o NN napsaných v předchozí kapitole a pokud možno co nejvíce automaticky. Každý z obrazů, který budeme analyzovat, se skládá z řádově 10^4 čísel (pixelů). Mohli bychom mít FF síť, do které by vstupovalo 10^4 hodnot (přímo celý obraz), síť by pro jednoduchost měla tři vrstvy, dva neurony ve druhé vrstvě, jeden ve třetí vrstvě a vystupovala by z ní jen jedna hodnota. Tím velkým počtem vstupních hodnot ale riskujeme, že síť se bude příliš dlouho učit a nakonec ještě bude klasifikovat obrazy hůře než jsme si přáli. Podle [1], s. 171, může i jednoduché síti trvat 600 iterací, než se naučí simulovat funkci XOR.
Proto se v první části této kapitoly zaměřím na získání veličiny nebo několika veličin, které by co nejvíce vypovídaly o tom, je-li na daném obraze Jednotka, Vícečetná nebo Nepoznaná a počet těchto veličin charakterizujících obrazy by byl maximálně deset. Tyto veličiny budu nazývat charakteristikami.
V dalších částech budu rozebírat, jak probíhalo učení různých sítí a jak moc správně uměla naučená síť klasifikovat vektory charakteristik obrazů. Začnu s testováním jednoho neuronu a podle potřeby budu testovat stále složitější sítě, než najdu tu, která bude dávat uspokojivé výsledky.
V poslední části vytvořím a popíšu výsledný program, jehož součástí bude nejlepší NN. Výsledný program pak bude umět automaticky roztřídit soubor obrazů do čtyř skupin: Uřízlé, Jednotky, Vícečetné a Nepoznané.

3.1 Charakteristiky obrazů

Z výše zmíněného důvodu hledejme charakteristiky obrazů, které by byly schopny od sebe číselně odlišit Jednotky a Vícečetné. To znamená: hledejme veličiny X takové, že $X(O) \in (a, b)$, je-li O Jednotka a $X(O) \in (c, d)$, je-li O Vícečetná, kde (a, b), (c, d) jsou disjunktní intervaly nebo nejsou-li disjunktní, tak míra $\mu((a, b) \cap (c, d))$ jejich překrytí je co nejmenší. μ je Lebesgueova míra - v tomto případě délka.

Budeme-li pozorovat obrazy a představovat si ke každému z nich konvexní obal, tak si lze všimnout, že plocha masky (viz 1.3.3) Jednotky je stejná jako plocha jejího konvexního obalu nebo blízká ploše konvexního obalu. U Dvojic a Vícečetných jsou mezery mezi jádry buněk (viz např. obrázek 1), takže poměr mezi plochou masky Vícečetné a plochou jejího konvexního obalu je menší než u Jednotek. Poměru mezi plochou masky Obrazu a plochou jejího konvexního obalu se nazývá **solidita**. Dá se očekávat, že většina obrazů, na nichž je pouze jedno jádro, bude mít soliditu blízkou číslu 1, zatímco obrazy, na nichž je více jader budou mít soliditu kolem 0.5 a určitě menší než 1. Tato samotná charakteristika rozliší většinu Jednotek od Dvojic a Vícečtných buněk, ale není ideální. Existují

totiž obrazy, na nichž je spousta na sebe nalepených jader a jejich solidita je např. 0.8 a existuje obraz, na němž je jediné jádro, které je ale pokroucené jako banán, takže jeho solidita je třeba 0.6. Takové případy jsou výjimečné, ale také se vyskytují.

Chceme-li, aby NN uměla od sebe rozlišit více obrazů, potřebujeme další charakteristiku. Tou může být například cirkularita. Nechť plocha útvaru na masce O_2 obrazu O_1 má velikost A_u a jeho obvod má délku p_u. Dál mějme kruh, jehož plocha má velikost A_k, kde $A_k = A_u$ a jeho obvod má délku p_k. Potom číslu $C = p_k/p_u$ říkejme **cirkularita**. $C \in \langle 0, 1\rangle$ a přesněji ho lze vyjádřit pomocí vzorce: $C = 2\sqrt{A_u\pi}/p_u$. Cirkularita kulatějších jader je blízká číslu 1. Cirkularita deformovaných jader je menší než 1. Jestliže každé jádro na daném obraze se dotýká alespoň jednoho a nejvýše dvou sousedních jader, tak s rostoucím počtem jader, která jsou na sebe nalepená, klesá cirkularita. Tento předpoklad splňuje asi 97% všech obrazů, na nichž jsou Vícečetné (včetně Dvojic).

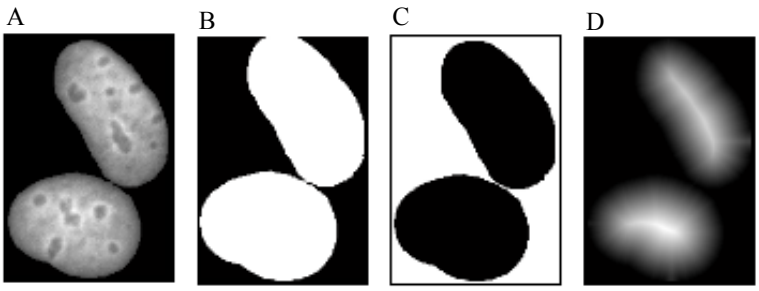
Podívejme se na jednu charakteristiku, která by mohla být schopná rozlišit Vícečetné od Jednotek a ještě navíc rozlišit Dvojice od ostatních Vícečetných lépe než solidita a cirkularita. Tou charakteristikou je **průměrný počet komponent souvislosti** (ppks) a funkce, s jejíž pomocí ji získáme, funguje následujícím způsobem: Na vstup funkce ppks přijde obrázek ve formátu bitmapy (viz obrázek 8 A). Každý pixel tohoto obrázku nabývá celočíselných hodnot od 0 do 255. Algoritmus funkce ppks pracuje tak:

1. vytvoří masku vstupního obrázku (viz obrázek 8 B)
2. získá negativ masky (viz obrázek 8 C)
3. spočítá distance transform (d.t.). Vznikne tak matice D. Distance transform je matlabovská funkce, která na pozici v matici obrazu dá číslo, které znamená eukleidovskou vzdálenost (v pixelech) této pozice od nejbližšího pixelu, který má hodnotu jedna.
4. normuje a upraví matici D tak, aby nabývala celočíselných hodnot od 0 do 255. Vznikne D_2 (viz obrázek 8 D). Normování je provedeno vydělením všech prvků matice jejím maximem, vynásobení matice číslem 255 a následuje zaokrouhlení, aby v matici byla jen přirozená čísla.
5. Teď si můžeme matici D_2 představit jako prázdnou vodní nádrž. Nechť matice D_2 znamená hloubky na každém metru čtverečním (v našem případě pixelu čtverečním). Nejvíce bílá barva v D_2 má hodnotu 255 metrů a odpovídá největší hloubce. Hladina podzemní vody se zvedá stejně rychle pod celou nádrží. Dnem nádrže začne prosakovat voda a my sledujme, kolik malých louží se v nádrži objeví, když vodní hladina sahá do hloubky $255m, 254m, \ldots 0m$, což označím h. Počet louží v hlouce h je n_h.

6. ppks sečte počty n_h louží na všech hladinách, kterých je 256 a vypočítá průměrný počet T louží (komponent souvislosti) na jedné hladině (v jedné vrstvě). $T = \frac{1}{256} \sum_{h=0}^{255} n_h$.

7. Výpočtem ppks pro asi 6000 obrazů bylo zjištěno, že je vhodné vydělit číslo T dvěma, aby většina čísel vycházela menší nebo rovna číslu 1. Tato drobná úprava je zde provedena pro to, že mírně urychlí učení NN. Když jsou vstupní hodnoty do sítě příliš velké nebo naopak příliš malé, zpomaluje to BP učení FF sítě [1]. Zkoušemím jsem zjistil, že optimální hodnota se pohybuje kolem jedničky.

   Obraz O_1, na němž je ideální jednotka, bude mít ppks$(O_1) = 0.5$. Obraz O_1, na němž je ideální dvojice, bude mít ppks$(O_1) = 1$. S rostoucím počtem buněk na obraze roste i ppks obrazu.



Obrázek 8: Ukázka co se děje s obrazem (A) jader v prvních třech krocích algoritmu funkce ppks

Nápad na charakteristiku ppks jsem dostal, když jsem uslyšel o algoritmu watershed [9]. Protože se mi nechtělo porozumět mu, vymyslel jsem si vlastní, který se watershedem něco společné.
Mohli bychom vymýšlet další charakteristiky, ale pojďme se podívat, jak vektory obrazů určené trojicí těchto charakteristik budou rozloženy v prostoru a jak budou později využity neuronovými sítěmi.

3.2 Shluky vektorů charakteristik

Jestliže každý obraz nahradíme trojicí čísel - vektor: (solidita, cirkularita, ppks) - tak se dá čekat, že 2 obrazy, které jsou si podobné, budou reprezentovány dvojicí blízkých vektorů. Proto čekám, že vektory Jednotek budou shromážděny v jednom shluku, vektory Vícečetných v druhém a vektory Nepoznaných budou rozptýleny někde mezi prvním a druhým shlukem.
A ukazuje se, že tomu tak skutečně je. Na průměty 3-d shluků do rovin se můžete podívat do přílohy na obrázky *shlukysm.bmp, shlukysp.bmp, shlukymp.bmp* nebo

na obrázky srovnání krabicových diagramů tříd pro každou ze tří charakteristik - viz *boxplotys.bmp, boxplotym.bmp, boxplotyp.bmp.*
Navíc, a to je nepříjemné, se ukazuje, že shluky nejsou lineárně oddělitelné. Domnívám se, že čím více neuronů a vrstev bude obsahovat NN, tím lepších výsledků s rozpoznáváním obrazů by měla dosahovat. Ta domněnka vychází z poznatku, že tři neurony uspořádané do dvou vrstev si poradí s řešením XOR problému, zatímco jeden neuron ne (viz 2.2.3 nebo [1]). Na druhou stranu čím více neuronů bude síť obsahovat, tím více časově náročnější bude učit ji a ještě hrozí, že bude přeučena - to znamená, že se perfektně naučí poznávat obrazy z tréninkového souboru a pak jí předložíme nový obraz a ona ho nepozná. Takže vzniká další problém. Z kolika a z jakých neuronů síť sestavit a jak ty neurony propojit, aby byla co nejjednodušší a zároveň uměla správně klasifikovat co nejvíce obrazů?
Začněme s tou nejjednodušší sítí - s jedním neuronem, postupně přidávejme další neurony a vrstvy a dívejme se na úspěšnosti jednotlivých sítí.

3.3 Jak si poradí perceptron s rozpoznáváním obrazů

Tento neuron má 3 vstupy, nastavitelné váhy, nastavitelný práh, jeho agregační fukcí je součet a aktivační funkcí je signum: $\text{sign}(x) = -1$ pro $x < 0$, $\text{sign}(x) = 1$ pro $x \geq 0$. Lze ho chápat jako funkci $P : \mathbb{R} \to \{-1; 1\}$ danou předpisem:
$y = \text{sign}((\mathbf{x}, 1)\mathbf{w})$.

Soubor s obrazy jader, který mám k dispozici obsahuje přibližně desetkrát více Jednotek než Dvojic. S využitím funkce *charakteristiky2.m* (kterou uvádím v příloze) jsem získal jejich charakteristiky: soliditu, cirkularitu a ppks a napsal je do řádkových vektorů $\mathbf{x}$. Ke každému vektoru $\mathbf{x}$ jsem přiřadil číslo $t = -1$, je li obraz reprezentovaný vektorem $\mathbf{x}$ Jednotka a je-li vektorem $\mathbf{x}$ reprezentovaná Dvojice, pak $t = 1$. Vybral jsem 1000 vektorů $(\mathbf{x}, t)$ Jednotek a 100 vektorů Dvojic a vytvořil z nich matici tréninkových vektorů *train3ch2tr*, která má rozměry 1100×4. Potom jsem vybral 1000 jiných vektorů Jednotek a 100 jiných vektorů Dvojic a vytvořil z nich matici testovacích vektorů *test3ch2tr*. Do tréninkového souboru pro trénování sítí jsem nezařadil vektory charakteristik takových Vícečetných, které nejsou Dvojicemi. Měl jsem k tomu tyto důvody:
- na obrázcích shluků vektorů charakteristik je jasně vidět, že těžiště Vícečetných, které nejsou Dvojicemi je od těžiště Jednotek dál než těžiště Dvojic a přibližně stejným směrem.
- experimentováním jsem zjistil, že výsledky učení sítě jsou lepší, když je síť učena rozlišovat Jednotky od Dvojic než Jednotky od obecně Vícečetných.

Jelikož shluky vektorů výše uvedených charakteristik se překrývají, nejsou lineárně oddělitelné a není zaručena konvergence perceptronového učení neuronu po konečném počtu iterací. Proto je potřeba zvolit nějaké kritérium, které zastaví učení neuronu po konečném počtu kroků. Tím může být právě počet I iterací al-

goritmu učení nebo velikost R relativní chyby sítě, což je relativní četnost špatně klasifikovaných vzorků. Protože $y_i, t_i \in \{-1, 1\}$, tak $|y_i - t_i|$ může být 0 nebo 2. Proto $R = \frac{1}{2200} \sum_{i=1}^{1100} |y_i - t_i|$, kde y_i je výstup neuronu pro vzorek $\mathbf{x}_i$ a t_i je požadovaný výstup neuronu pro vzorek $\mathbf{x}_i$, což je vzorek z testovacího souboru.

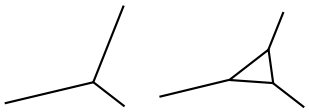
Nejprve jsem provedl algoritmické učení perceptronu tak, jak je uvedeno v části 2.2.4 s tím rozdílem, že učební vzorky z tréninkového souboru jsem nedával neuronu náhodně, ale postupně nejdříve všechny vektory zastupující Jednotky a potom všechny Dvojice a když došly vzorky, učení skončilo. Dopadlo tak, že všechny Dvojice byly neuronem klasifikovány jako Jednotky.
Potom jsem k učení použil přesně ten algoritmus z části 2.2.4 a přidal k němu zastavovací kritérium. Kód najdete v příloze v matlabovské funkci
perc_uc_01042012.m . Učební vzorky byly perceptronu předkládány náhodně střídavě Jednotky a Dvojice. Toto učení bylo provedeno několikrát a pokaždé s jinými výsledky. Nejlepším výsledkem je, že pro jistou kombinaci vah je relativní chyba neuronu $R = 0.031818$ a tento neuron s 98.8% pravděpodobností řekne o obraze, že to je Jednotka, za předpokladu, že totéž řekl i expert a se 77.0% pravděpodobností řekne o obraze, že to je Dvojice, za předpokladu, že totéž řekl i expert. Takže pořadí vzorků tady mělo vliv na učení perceptronu. A perceptron mě překvapil svým výkonem, čekal jsem něco horší. Teď už bych ani nemusel vymýšlet lepší síť, protože mým vedlejším cílem bylo, aby síť uměla poznat aspoň 95% Jednotek. A to se stalo. Přesto se ještě podívejme na učení vrstevnaté sítě a její výsledky.

3.4 Použití vrstevnaté neuronové sítě

3.4.1 Struktura sítě

Protože máme tři charakteristiky, bude mít síť tři vstupy. Ve druhé vrstvě nechť jsou čtyři neurony a ve třetí jen jeden neuron. Dále nechť všechny neurony mají spojité a stejné aktivační funkce, tangens hyperbolický $y = \tanh(x)$, spojité abychom pak mohli použít BP pro nalezení co nejlepších vah synapsí této sítě. Totéž nechť platí pro agregační funkce - sumy. V příloze uvádím funkci *sit_341.m.* V této a následující části najdete popisy fungování použitých matlabovských kódů, které jsou v příloze a které byly použity také k učení sítě *sit_341.*
Tady je důvod proč jsem se rozhodl dát čtyři neurony do druhé vrstvy. Rozhodovací hranicí jednoho neuronu, který má tři vstupy, je rovina. Čtyři neurony odpovídají čtyřem rovinám ve 3d prostoru. Pokud roviny v prostoru správně umístíme, získáme dohromady pěknou hraniční plochu, která od sebe může oddělovat vektory charakteristik Jednotek a vektory charakteristik Vícečetných lépe než jediná rovina. Výsledná rozhodovací hranice, kterou můžete vidět na obrázku 9 vpravo, připomíná useknutý roh. Jeden neuron ve třetí vrstvě je zvolen proto, že nás zajímá jen jedno číslo. To číslo bude nabývat reálných hodnot větších než

-1 a menších než 1. Čím blíže bude číslu -1, tím více pravděpodobné je, že vektor vstupních charakteristik odpovídá obrázku, na němž je právě jedno jádro. Čím blíže bude číslu 1, tím více pravděpodobné je, že vektor vstupních charakteristik odpovídá obrázku, na němž jsou aspoň dvě buňky.



Obrázek 9: Dva rohy, první roh je částí sjednocení tří rovin a druhý je částí sjednocení čtyř rovin

3.4.2 Učení sítě

Pro trénink sítě byla opět použita data: *train3ch2tr.mat*. Na začátku nastavme váhy všech synapsí v síti náhodně a to tak, že hodnota každé váhy pochází z intervalu $\langle -1, 1 \rangle$. Zde nemůžeme použít perceptronové učení, protože je definované pouze pro jediný perceptron. Proto budeme síť *sit_341.m.* učit jiným způsobem. V kapitole 2.4 jsme si nachystali BP algoritmus a teď ho aplikujeme na tuto síť. BP algoritmus je zde realizován funkcí *epoch_chyb_sit_341*, která je uvedena v příloze. Vstupem do této funkce jsou parametry: Vzorky (soubor vstupních vektorů $\mathbf{x}$ a jim příslušných očekávaných výstupů t), Test (soubor jiných vstupních vektorů $\mathbf{x}$ a jim příslušných očekávaných výstupů t), poc_w12 (počáteční váhy synapsí mezi první a druhou vrstvou), poc_w23 (počáteční váhy synapsí mezi druhou a třetí vrstvou), Line (druh BP algoritmu - buď off-line nebo on-line), gama (krok algoritmu, učební konstanta, $\gamma \in \mathbb{R}_0^+$), Epoch (počet epoch, které uplynou při učení sítě). Výstupem z této funkce je vektor WWW všech vah sítě na konci učení a graf závislosti veličin R, J, D počtu uběhlých Epoch. R je relativní chyba sítě, což je chyba daná vzorcem (13) a ještě dělená dvojnásobkem počtu testovacích vzorků. J je podíl počtu Jednotek, o nichž síť řekla, že to jsou Jednotky a počtu skutečných Jednotek v testovacím souboru. D je podíl počtu Dvojic, o nichž síť řekla, že to jsou Dvojice a počtu skutečných Dvojic v testovacím souboru. Funkce *epoch_chyb_sit_341* pracuje podle toho, zadá-li uživatel on-line nebo off-line trénink. Je-li zadán on-line trénink, tak funkce *epoch_chyb_sit_341* opakovaně používá funkci *uc_sit_341_bp_on* pro aktualizaci vah sítě pomocí BP algoritmu tolikrát jako je uživatelem zadaný počet epoch. Pokud uživatel zadá na začátku off-line trénink, tak probíhá totéž, jak už jsem popsal, ale s tím rozdílem, že pro aktualizaci vah je využívána funkce *uc_sit_341_bp_off*.

U perceptronového učení jsme museli mít jen tréninkový a testovací soubor vektorů, nějak zvolit počáteční váhy a zastavovací kritérium. Zde, když budeme učit síť *sit_341*, už musíme volit více parametrů: počáteční váhy, tréninkové vzorky a jejich pořadí, testovací vzorky, druh učení (dávkové nebo on-line), velikost učební

kostanty (krok γ m.n.s. v BP algoritmu) a počet Epoch (iterací, v nichž proběhne úprava vah). Je zde spousta kombinací parametrů, které můžeme zkoušet a následně pozorovat, jak probíhá učení sítě.
Nejlepších experimentálně získaných výsledků, t.j. $R = 0.02647$, $J = 0.989$, $D = 0.76$, síť dosahuje pro váhy uvedené ve funkci *sit_341_h*, což už je naučená síť. Váhy jsou získány pomocí off-line BP s parametry: $\gamma = 0.00045$, Epoch = 2000,

$$poc\_w12 = \begin{pmatrix} -0.9003 & -0.7826 & -0.6428 & -0.8436 \\ 0.5377 & -0.5242 & 0.1106 & -0.4764 \\ -0.2137 & 0.0871 & -0.2309 & 0.6475 \\ 0.0280 & 0.9630 & -0.5839 & 0.1886 \end{pmatrix},$$

$$poc\_w23 = \begin{pmatrix} -0.8709 \\ -0.8338 \\ 0.1795 \\ -0.7873 \\ 0.8842 \end{pmatrix},$$

Průběh tohoto učení můžete vidět v příloze na obrázku *BpOf45.bmp*. Pro srovnání a tytéž hodnoty ještě uvádím průběh on-line tréninku na obrázku *BpOn45.bmp* a on-line tréninku pro jiné počáteční váhy a $\gamma = 0.0001$ na obrázku *BpOn10.bmp*. Tyto obrázky byly získány pomocí funkce *epoch_chyb_sit_341*.
Zdá se, že ačkoliv on-line trénink je asi 14× rychlejší než dávkový trénink (soudě podle doby potřebné na průběh stejného počtu epoch), tak při dávkovém tréninku síť dosahuje lepších výsledků po uplynutí menšího (cca 50× menšího) počtu epoch než při on-line tréninku. Z toho usuzuji, že off-line trénink sítě *sit_341* pro dané parametry (počáteční váhy a γ) je efektivnější.
Srovnáme-li náročnost učení a výkon sítě *sit_341* s náročností učení a výkonu perceptronu, dojdeme k závěru, že bylo jednodušší a rychlejší učit perceptron a jeho výkony jsou srovnatelné s výkony sítě *sit_341*.

3.5 Výsledný program

Dosud jsme mohli vidět, že perceptron nebo síť *sit_341* zpracuje vektor charakteristik libovolného obrazu s jádry, pokud má charakteristiky k dispozici. V praxi se ale stane to, že dostaneme soubor s obrazy jader a máme je roztřídit. To síť neumí. Ona umí pouze přiřadit číslo vektoru charakteristik jednoho obrazu. Abychom tedy nemuseli třídit obrazy ručně, musíme mít program, který automaticky roztřídí obrazy do adresářů a může využít již natrénovanou NN k rozhodování, který obraz kam patří.
V této kapitole popíšu program, který bude umístěn do adresáře, v němž se nachází obrazy jader ve formátu bitmapy. V tom adresáři program vytvoří čtyři nové adresáře (Uřízlé, Jednotky, Nepoznané, Vícečetné) a do nich roztřídí obrazy. Kód tohoto programu se jmenuje: *roztrid.m*.

3.5.1 Popis programu

Program roztrid se používá tak, že ho umístíme do adresáře naplněného obrazy, které chceme roztřídit. Obrazy musí být ve formátu bitmapy ve stupních šedé. Protože roztrid potřebuje ke své činosti ještě další funkce, musíme do adresáře spolu s ním umístit také: *jakmocrizla.m*, *chrakteristiky2.m*, *solidita.m*, *ppks.m*, *mpk.m*, *sit_341_h.m*. Funkce *chrakteristiky2.m* slouží k získání vektoru charakteristik ze vstupního obrazu. Využívá k tomu funkce: *solidita.m, mpk.m, ppks.m.* Funkce *jakmocrizla.m* spočítá $\max(n_i/v_i)$. $i \in \{1, 2, 3, 4\}$, i je pořadové číslo okraje obrazu. n_i je počet nenulových pixelů na i-tém okraji. v_i je délka i-tého okraje (počet všech pixelů na i-tém okraji). Číslujeme-li okraje po nebo proti směru hodinových ručiček, platí $v_1 = v_3, v_2 = v_4$. Je-li buňka na obraze uřízlá moc (nevešla se na obraz celá), tak nás nezajímá a program roztrid ji hned zařadí do adresáře Urizle. Buňky jsou uřízlé moc, když $jakmocrizla('obraz') > 0.6$.
Dalšími vstupy do *roztrid.m* (kromě obrazů) jsou ještě parametry dm, hm, kde dm je dolní mez pro Nepoznané, hm je horní mez pro Nepoznané.
NN *sit_341_h* po zpracování vstupních charakteristik obrazu dá výstup, což je reálné číslo z otevřeného intervalu od -1 do 1. Naučená NN dává výstupy blízké číslu -1, pokud dostala na vstup charakteristiky Jednotky a výstupy blízké číslu 1, pokud dostala na vstup charakteristiky Vícečetné. Pokud NN dostala na vstup charkteristiky Nepoznané buňky, mělo by se na jejím výstupu objevit číslo blízké nule. U těchto dvou vstupů (dm, hm) jde o to, že pokud bude výstup v sítě z intervalu $(-1, dm)$, pak *roztrid* zařadí vstupní obraz do adresáře Jednotky. Pokud bude výstup sítě z intervalu ⟨dm,hm⟩, pak *roztrid* zařadí vstupní obraz do adresáře Nepoznané. Pokud bude výstup v sítě z intervalu $(hm, 1)$, pak *roztrid* zařadí vstupní obraz do adresáře Vícečetné. Pro správnou funkčnost, by měl uživatel zadat dm a hm tak, aby platilo: $-1 < dm < hm < 1$.
Funkce *roztrid.m* má kromě roztříděných obrazů ještě tyto výstupy: ur, jed, vic, ne, kde ur je počet obrazů zařazených do adresáře Urizle, jed je počet obrazů zařazených do adresáře Jednotky, vic je počet obrazů zařazených do adresáře Vicecetne a ne je počet obrazů zařazených do adresáře Nepoznane.

3.5.2 Fungování programu

Do Příkazového okna v Matlabu zadáme:

```
[ur,jed,vic,ne] = roztrid(dm,hm);
```

kde dm a hm jsou už konkrétní čísla a program začne pracovat:

1. Zkontroluje, zda-li jsou vstupní parametry zadány správně.
2. Vytvoří 4 adresáře, do nichž budou řazeny obrazy.

3. Do seznamu načte názvy obrazů v aktuálním adresáři.

4. Projíždí seznamem obrazů a pomocí *jakmocrizla.m* zkontroluje, jak moc je jádro (jádra) na každém obraze uřízlé.

   (a) Pokud $jakmocrizla('obraz') > 0.6$, je obraz zařazen do složky Urizle.

   (b) Pokud $jakmocrizla('obraz') \leq 0.6$, pak na obraz jsou aplikovány další funkce:

      i. Pomocí *chrakteristiky2.m* je získán vektor charakteristik obrazu.

      ii. Ten je předložen síti sit_341_h.

      iii. Pokud je výstup sítě číslo z intervalu:

         A. (-1,dm)
         pak *roztrid.m* zařadí vstupní obraz do adresáře Jednotky

         B. ⟨dm,hm⟩
         pak *roztrid.m* zařadí vstupní obraz do adresáře Nepoznané

         C. (hm,1)
         pak *roztrid.m* zařadí vstupní obraz do adresáře Vícečetné.

A když budeme chtít obrazy znovu smíchat, nemusíme to dělat ručně, ale stačí spustit funkci *smichej.m*.

3.5.3 Výsledky

Ještě před provedením celé této práce jsem obdržel adresář, který obsahoval 6887 různých obrazů jader buněk. Ručně jsem obrazy roztřídil do šesti složek: 1) uřízlé jednotky, 2) jednotky, 3) uřízlé dvojice, 4) dvojice, 5) vícečetné, které nejsou dvojicemi, 6) nepoznané. Adresář, s jehož pomocí budu testovat tento program obsahuje 20 uřízlých jednotek, 40 jednotek, 20 uřízlých dvojic, 20 dvojic, 20 vícečetných, 40 dalších (mnou nepoznaných) braných z původních šesti adresářů od konce. Takže výsledek by měl být v ideálním případě: $ur = 40, jed = 40$, $vic = 40, ne = 40$.
Experimentováním jsem zjistil, že zvolím-li hodnoty $dm = -0.93$ a $hm = 0.7$, pak výsledky pro počty ur, jed, vic, ne jsou nejblíže ideálnímu případu.
Na počítači s procesorem AMD Athlon(tm) 64 X2 Dual-Core Processor TJ-55 1.79 GHz, 1.87 GB RAM výsledný program roztřídil 160 obrazů za 50 sekund. Rychlost je tedy asi 3.2 obrázků za sekundu.
Některá jádra z těch, co jsem já nepoznal, jsou řazena do Uřízlých a Nepoznané jsou promíchány více s Vícečetnými než s Jednotkami, což je dobře, protože díky tomu umí program rozeznat přes 95% Jednotek.

4 Závěr

Cílem bylo sestavit počítačový program, který s využitím neuronové sítě automaticky roztřídí velké množství obrazů podle toho, je-li na nich jediné jádro buňky nebo více jader. Při tom nejdůležitější bylo pro zadavatele tohoto problému, aby byly rozeznány Jednotky od všech ostatních obrazů. To se podařilo výslednému programu v 98% případů. Všechny zdrojové kódy všech programů uvedených v této práci jsou napsané a spustitelné v Matlabu. V této práci jsem otestoval, jak si poradí jeden perceptron s řešením tohoto problému. Po tom jsem vytvořil jednoduchou vrstevnatou neuronovou síť se šířením informace vpřed. Já v roli učitele ji naučil pomocí algoritmu zpětného šíření chyby rozeznávat co nejvíce obrazů s jediným jádrem od ostatních obrazů s jádry. Napsal jsem kód, který v sobě obsahuje neuronovou síť a po spuštění dokáže roztřídit obrazy do čtyř adresářů podle toho, co na nich je. Uživatel tohoto programu sám volí šířku „šedého pásma“, kam patří jádra, které je i pro lidi těžké rozeznat od Jednotek nebo Vícečetných. Za svůj praktický přínos považuji vytvoření jednoduchého programu speciálně pro řešení problému se tříděním jader buněk, který funguje překvapivě dobře. Dovoluji si tvrdit, že cíl práce byl splněn.

Reference

[1] ROJAS, Raúl. *Neural networks : A Systematic Introduction.* Springer-Verlag Berlin Heidelberg, 1996. 502 s. ISBN 3-540-60505-3.

[2] HAYKIN, Simon. *Neural networks : A comprehensive foundation.* second. ed. Upper Saddle River, New Jersy 07458 : Prentice-Hall Inc., 1999. 842 s. ISBN 0-13-908385-5.

[3] DUDA, Richard O.; HART, Peter E.; STORK, David G. *Pattern Classification.* Canada : John Wiley & Sons, Inc., 2001. 654 s. ISBN 0-471-05669-3.

[4] NOVÁK, Mirko. *Umělé neuronové sítě: teorie a aplikace.* Praha: C.H. Beck, 1998. ISBN 8071791326.

[5] WILSON, D. Randall; MARTINEZ, Tony R. The general inefficiency of batch training for gradient descent learning. Neural Networks [online]. 8 April 2003, 2003, 16, [cit. 2011-12-04]. s. 1429{1451. Dostupný z WWW: <www.ElsevierComputerScience.com>.

[6] VG20102014001 - Nové postupy biodozimetrické kontroly účinku radiačního záření a genotoxických látek založené na indukci dvouřetězcových zlomů DNA v buňkách vlasových a chlupových folikulů (2010-2014, MV0/VG). In: *Informační systém výzkumu, experimentálního vývoje a inovací* [online]. 28.3.2011 [cit. 2012-03-10]. Dostupné z: http://www.isvav.cz/projectDetail.do?rowId=VG20102014001

[7] *FyzWeb* odpovědna. FyzWeb [online]. 10.04.2008 [cit. 2012-03-16]. Dostupné z: http://fyzweb.cz/odpovedna/index.php?limit_od=10&hledat=n%C4%9Bjak%C3%BD

[8] Gradient descent. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-03-28]. Dostupné z: http://en.wikipedia.org/wiki/Gradient_descent

[9] VINCENT, Luc a Pierre SOILLE. Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations. IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE. JUNE 1991, VOL. 13, NO. 6,. Dostupné z: http://csce.uark.edu/∼jgauch/library/Segmentation/Vincent.1991.pdf

Seznam souborů přiložených na CD:

```
README.txt

uceni_siti
        epoch_chyb_sit_341.m
        charakter21022012.mat
        reler.m
        senzspec.m
        sit_341.m
        sit_341_h.m
        train3ch2tr.mat
        uc_sit_341_bp_off.m
        uc_sit_341_bp_on.m

vysledny_program
        charakteristiky2.m
        jakmocrizla.m
        mpk.m
        ppks.m
        roztrid.m
        sit_341_h.m
        smichej.m
        solidita.m

ilustrace_textu
        boxplotym.bmp
        boxplotyp.bmp
        boxplotys.bmp
        BpOf45.bmp
        BpOn10.bmp
        BpOn45.bmp
        shlukymp.bmp
        shlukysm.bmp
        shlukysp.bmp

jadra_bunek
        160 obrázků ve formátu bitmapy
        pozorovani.txt
```