Masarykova univerzita v Brně

Fakulta informatiky

Diplomová práce

Rozpoznávání biomedicínských obrazů pomocí neuronových sítí

Marek Winkler

2004

Prohlášení

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny použité zdroje v práci řádně uvádím.

Poděkování

Na tomto místě bych rád poděkoval doc. RNDr. Michalu Kozubkovi, Ph.D. za vedení práce a především za trpělivost při spolupráci se mnou. Dále bych chtěl poděkovat všem, kteří mi jakýmkoli způsobem pomohli, zejména Jakubu Foukalovi za konzultaci odlišností tkáně od pozadí obrazu a Šimonu Macharáčkovi za poskytnutí dodatečné výpočetní kapacity pro učení sítě.

Shrnutí

Tato práce se zabývá možnostmi aplikace neuronových sítí při segmentaci a klasifikaci obrazu. Nejprve popíše přístupy k výběru charakteristik obrazu (tzv. rysů) použitelných jako vstupy neuronové sítě, dále seznámí čtenáře s některými modely neuronových sítí a popíše implementaci použitého řešení problému. Závěrem navrhne možná zlepšení a poukáže na objevené nedostatky popsaného způsobu řešení.

Klíčová slova: digitální zpracování obrazu, segmentace, klasifikace, neuronové sítě, backpropagation, Haralickovy rysy, matice sousednosti intenzit

Obsah

1. Úvod

Tato práce je zaměřena na použití tzv. umělých neuronových sítí k segmentaci a rozpoznávání obrazu, konkrétně tkáně tlustého střeva. Obraz znázorňuje příčný řez tzv. kryptami – výrůstky (směrem ven) ze střevní stěny sloužící k prokrvení střevní tkáně. Těchto krypt může být na jednom obraze několik a nemusí být zachyceny celé. Úkolem je rozpoznat tkáň od pozadí obrazu a na základě jejích vlastností rozhodnout, zda se jedná o tkáň zdravou či nádorovou (na jednom obraze je vždy tkáň jen jednoho druhu). Úspěšné řešení problému by mohlo pomoci při diagnóze rakoviny tlustého střeva, podobný postup se již uplatnil při diagnóze kostních nádorů [Pelikan et al. 99]. Tato práce má za úkol popsat možnosti použití neuronových sítí v této oblasti, navrhnout řešení problému a řešení implementovat.

Rozpoznávání obrazu je jednou z hlavních aplikačních oblastí umělých neuronových sítí. Tento výpočetní model se ukázal být poměrně úspěšným a elegantním řešením některých problémů, zvláště díky své schopnosti generalizace a tolerance vůči chybám. Neuronové sítě se od klasického imperativního programování liší především tím, že svou funkci nemají dánu napevno a předem, ale postupně se jí učí. (Ve většině praktických případů je sice fáze učení oddělena od fáze aplikační, ale obecně tomu tak být nemusí.) Výhoda tohoto přístupu se při zpracování obrazu projeví zejména tolerancí ke zkresleným nebo neúplným vstupním datům, dále také schopností vybrat si z nich to podstatné pro řešení daného problému a v neposlední řadě zobecněním zjištěných faktů a jejich využití při práci s novými daty podobnými těm známým. Neuronové sítě se kromě rozpoznávání obrazu využívají např. v medicíně při diagnostikování určitých chorob nebo v oblasti umělé inteligence. Podrobněji se neuronovým sítím věnuji ve zvláštní kapitole.

Po upřesnění několika základních pojmů vyložíme problematiku výběru vstupních dat pro neuronovou síť, tzv. *rysů* (viz dále, podrobně druhá kapitola). Ve třetí kapitole popíšeme vybrané modely neuronových sítí s ohledem na problematiku zpracování obrazu, čtvrtá kapitola osvětlí způsob implementace řešení problému. V páté kapitole vyložíme konkrétní modely sítí použitých v implementaci a jejich výsledky. Závěrem navrhneme možná zlepšení a odhalíme objevené slabiny použitého způsobu řešení.

1. 1 Základní pojmy

V digitálním zpracování obrazu se většinou setkáme s následujícím postupem práce (výčet není pro jednoduchost úplný): předzpracování obrazu, určení rysů (angl. *features*), segmentace, rozpoznání/klasifikace objektů. Tyto pojmy se nyní pokusím přiblížit.

Předzpracování obrazu má za úkol odstranit nepříznivé vlastnosti vzniklé při snímání obrazu, např. odstranit šum, upravit jas, případně vypustit nepotřebné informace (v některých aplikacích např. převést barevný obraz na šedotónní).

Určení rysů obrazu znamená výběr informací užitečných pro řešení daného problému. *Rysem* může být jakákoli charakteristická vlastnost obrazu, případně některé jeho části, např. intenzita (příp. barva) bodu v obraze, jeho histogram nebo charakteristika textury (viz 2.

kapitola). Výběr vhodných rysů je klíčový pro další zpracování, lze jím snadno dosáhnout invarianci vůči rotaci a změně měřítka.

Segmentace je proces nalezení objektů (nebo jejich částí), které nás zajímají, v obraze. Jinými slovy, obraz rozdělíme na segmenty, o kterých víme, co reprezentují (v našem případě šlo o odlišení tkáně od pozadí obrazu).

Rozpoznání/klasifikace objektů nalezených segmentací zcela závisí na konkrétní aplikaci, obecně jde o zjištění vlastností objektů, na jejichž základě přiřadíme daný objekt do nějaké třídy objektů. V této práci šlo o klasifikaci nalezené tkáně jako zdravé nebo nádorové.

1. 2 Obrazová data

Laboratoř optické mikroskopie mi poskytla pět obrazů tkání v rozlišení 1300 na 1030 obrazových bodů při barevné hloubce 24 bitů. Tři obrazy znázorňovaly zdravou tkáň, dva nádorovou. Z obrazů jsem pomocí programu Adobe Photoshop vyřezal různě velké části (ale podstatně menší než původní obrazy) zobrazující vždy jak tkáň, tak pozadí. Tato data byla později použita pro učení neuronových sítí. Při výběru výřezů jsem přihlížel pouze k tomu, aby na výřezu byla jak tkáň, tak i pozadí, a aby byla zastoupena tkáň zdravá i nádorová.

2. Předzpracování obrazu a rysy

2. 1 Úvod do problematiky

Předzpracování obrazu si klade za úkol převést vstupní obraz do podoby výhodné vzhledem k dalšímu zpracování. Může zdůraznit vlastnosti využitelné např. k následné segmentaci nebo naopak potlačit to, co nás nezajímá. V této části se úzce prolíná s další fází, tj. s určením rysů obrazu. Obě fáze jsou závislé na konkrétním problému, naprosto obecný přístup není dost dobře použitelný. V této práci byly obrazy ve fázi předzpracování pouze převedeny na šedotónní s intenzitou v rozsahu 0..255. Rozsah 256 úrovní intenzity je v případě biomedicínských obrazů buněk optimální pro výpočet Haralickových rysů založených na charakteristice textury (viz dále) podle [Murphy et al. 03].

Pro následnou segmentaci můžeme použít přímo intenzity jednotlivých obrazových bodů, nebo rysy charakterizující různé obrazové vlastnosti, např. texturu. Použil jsem druhý přístup, důvody se nyní pokusím nastínit.

Přímé použití intenzity obrazových bodů má jednoznačnou výhodu v tom, že se nemusíme obávat o ztrátu informace důležité k úspěšnému rozpoznání/segmentaci obrazu. Na druhou stranu, pro neuronové sítě to není příliš vhodný postup díky obrovskému počtu vstupních dat (neuronová síť by musela mít ve vstupní vrstvě tolik neuronů, kolik je bodů v obraze), a to i v případě rozdělení obrazu na několik menších. Nejmenší použitelný výřez by měl jistě rozměry alespoň několika desítek bodů v jednom směru, ve 2D tedy celkem několik stovek. To je sice technicky realizovatelné, ale vektor vstupních dat o dimenzi řádu několika set by významně snížil úspěšnost neuronové sítě, díky jevu zvaném *Kletba dimenzionality* (*The curse of dimensionality* – [Bishop 95], [Scott et al. 98]). Tento jev prakticky znamená, že přidávání dalších vstupních údajů od jistého počtu znamená snížení úspěšnosti neuronové sítě (obecně jakéhokoli klasifikátoru) namísto očekávaného zlepšení.

Pro názornost si můžeme představit jednoduchý příklad. Nechť je dán rys A, který může nabývat deseti různých hodnot, a množina dat D obsahující dvacet prvků. Pokud budeme učit neuronovou síť s jedním vstupem A na této množině dat, je pravděpodobné, že data budou rys A dobře popisovat (tj. každá z deseti hodnot A bude nějak reprezentována v D). Představme si, že přidáme další rys B, který může nabývat také deseti hodnot jako A, ale nepřidáme žádná nová data. Pak prostor vstupních dat A x B může nabývat 100 různých hodnot, které budou obsazeny dvaceti příklady z D poměrně řídce. Pokud přidáme další rys o deseti hodnotách, prostor vstupních dat se zvětší na 1000, a tak bychom mohli pokračovat. Problémem je, že počet potřebných tréninkových dat v D roste *exponenciálně* s počtem rysů. V případě takto řídce popsaného vstupního prostoru tréninkovými daty nebude mít neuronová síť dostatek dat pro správné zobecnění problému.

Další nevýhodou použití intenzity je citlivost na umístění, otočení a velikost objektů v obraze. Jedním z možných řešení je vždy doplnit tréninková data o všechny přípustné kombinace otočení, posunutí a zvětšení/zmenšení, což ale může být náročné jak časově, tak prostorově. Obrazy se také mohou lišit jasem, kontrastem apod., proto by měly být takové obrazy předzpracovány např. technikou vyrovnání histogramu (viz dále).

Druhý přístup založený na rysech spočtených na základě různých vlastností obrazu má naproti tomu jednu základní nevýhodu – jak určit, které rysy (vlastnosti obrazu) jsou relevantní pro řešení daného problému? Také není znám obecný postup, který by zaručil, že si množina rysů uchovala veškerou informaci potřebnou pro řešení problému. Tyto otázky je nutno vždy řešit s ohledem na konkrétní problém. Pokud ovšem zvolíme rysy vhodně, můžeme efektivně dosáhnout velmi dobrých výsledků. Do této kategorie rysů patří např. tzv. *Run Length Coding* [Sonka et al. 95] nebo Haralickovy rysy založené na matici sousednosti intenzit v obrazu.

2. 2 Matice sousednosti intenzit a Haralickovy rysy

Matice sousednosti intenzit (také gray level co-occurrence matrix, GLCM) pro vzdálenost *d* je definována následujícím způsobem: hodnota na pozici (*i, j*) odpovídá počtu výskytů intenzity *i* a intenzity *j* vzdálených od sebe *d* bodů v obraze. Obecně je GLCM definována i pro určitý směr, takže můžeme mít GLCM pro intenzity bodů vzdálených od sebe o *d* např. ve vodorovném a svislém směru, ale pokud chceme zachovat vlastnost invariance vůči rotaci, doporučuje se GLCM spočítat pro všechny směry, případně matice pro jednotlivé směry sečíst.

Haralick navrhl 14 statistických rysů založených právě na GLCM [Haralick et al. 73], [Sonka et al. 95]. Protože některé Haralickovy rysy spolu korelují, v praxi se většinou používá jen několik z nich, proto zde detailněji uvedu jen rysy použité v této práci. Ve vzorcích definujících jednotlivé rysy znamená $m_{i,j}$ prvek v GLCM na pozici (*i, j*).

Energie (Second Angular Moment)

Energie měří uniformitu textury v obraze. Jinými slovy, tento rys dosahuje nejvyšších hodnot v případě, kdy obraz obsahuje málo dominantních přechodů mezi různými úrovněmi intenzity (je homogenní).

$$f_1 = \sum_i \sum_j m_{i,j}^2$$

Entropie[1]

Entropie vyjadřuje míru neuspořádanosti textury. Entropie je inverzně proporční energii, tj. entropie dosahuje vysokých hodnot právě když je energie nízká a naopak.

$$f_2 = -\sum_i \sum_j m_{i,j} \ln(m_{i,j})$$

Diferenční moment

Diferenční moment je inverzně proporční inverznímu diferenčnímu momentu, dosahuje vysokých hodnot pro textury s velkými variacemi intenzit. Použil jsem jeho dvě varianty

[1] Poznámka k implementaci: v implementaci byla entropie počítána od začátku podle článku [Pelikan et al. 99] vzorcem $f_2 = \sum_i \sum_j m_{i,j} \ln(m_{i,j})$, což pro další výpočet neuronové sítě nenese žádné důsledky, pouze bych tuto veličinu měl asi spíše nazývat „opačnou entropií“. Za tuto nepřesnost v terminologii se omlouvám.

kontrast a zkreslení. Obecný diferenční moment je dán následujícím vzorcem, kde $\lambda \in Z$ (záporné pro $i \neq j$) a K je většinou 1.

$$f_D = \sum_i \sum_j (i-j)^\lambda m_{i,j}^K$$

Kontrast

Kontrast je variantou diferenčního momentu, dosahuje vysokých hodnot pro textury s velkými variacemi intenzit.

$$f_3 = \sum_i \sum_j (i-j)^2 m_{i,j}$$

Zkreslení (Distortion)

Zkreslení (také diferenční moment třetího řádu), podobně jako kontrast, je jistou variantou diferenčního momentu.

$$f_4 = \sum_i \sum_j (i-j)^3 m_{i,j}$$

Inverzní diferenční moment (Inverse Difference Moment)

Inverzní diferenční moment nabývá nejvyšších hodnot pro matice s většinou hodnot kolem hlavní diagonály, tj. pro textury s malým počtem sousednosti různých intenzit. Jak název napovídá, je inverzně proporční diferenčnímu momentu (řádu 2).

$$f_5 = \sum_i \sum_j \frac{1}{1+(i-j)^2} m_{i,j}$$

2. 3 Rysy použité pro segmentaci obrazu

Pro segmentaci jsem se rozhodl použít jeden globální a osm lokálních obrazových rysů (globální rysy vycházejí z celého obrazu, zatímco lokální pouze z určitého okolí nějakého bodu), všechny byly invariantní vůči rotaci. Konkrétně se jednalo o vyhlazený histogram (jediný globální rys), intenzitu daného bodu, medián, tzv. unsharp masking a pět Haralickových rysů: energii, inverzní diferenční moment, kontrast, entropii a zkreslení. Uvedené Haralickovy rysy jsem již popsal výše, zbývají tedy následující:

Vyhlazení histogramu (Histogram equalization)

Histogram intenzity je důležitým globálním rysem, protože pomocí něj můžeme určit celkovou úroveň jasu obrazu, která se ve vstupních datech může často velice lišit vlivem různých okolností při pořizování obrazu. Proto je důležité poskytnout neuronové síti informaci o intenzitě vzhledem k histogramu obrazu.

Histogram intenzity je funkce, která přiřadí každé hodnotě intenzity počet výskytů dané intenzity v obraze. Většinou se zobrazuje sloupcovým grafem pro rozsah intenzit odpovídající rozsahu intenzit daného obrazu.

Rys vyhlazení histogramu přiřadí každé intenzitě *i* v obrazu normalizovaný součet hodnot histogramu pro intenzity menší nebo rovny *i*. Konkrétně, nechť *i* je intenzita, *MaxI* je maximální možná hodnota intenzity v obrazu (v tomto případě 255), *f(x,y)* je hodnota intenzity bodu na pozici *(x,y)* v obraze. Pak rys vyhlazení histogramu f_6 je definován následovně:

$$f_6(x,y) = g(f(x,y)) \qquad g(i) = \frac{\sum_{(x',y')} f(x',y') \le i}{\sum_{(x',y')} f(x',y')} MaxI$$

Medián

Tento rys spočítá hodnotu mediánu v daném okolí bodu. Velikost okolí je nastavitelná, výchozí hodnotou je 9x9 bodů. Medián jakožto filtr se používá v digitálním zpracování obrazu pro vyhlazování, případně odstranění drobného šumu. Medián množiny čísel D lze spočítat seřazením čísel podle velikosti a jako medián vybrat prostřední prvek. Pokud D nemá prostřední prvek (v D je sudý počet čísel), jako medián vybereme aritmetický průměr dvou „prostředních“ hodnot. Oproti prostému aritmetickému průměru má medián výhodu v daleko menší ovlivnitelnosti malým počtem výrazně odlišných hodnot. Takový případ může nastat při výskytu šumu nebo nějaké chyby, pro lepší představu uvedeme příklad:

Mějme množinu hodnot {5, 10, 5, 20, 250}. Její aritmetický průměr bude 58, zatímco medián 10, což lépe reprezentuje „většinu“.

$$f_7(x,y) = Med(x,y)$$

Unsharp masking

Jedná se o aplikaci zostřovacího filtru, tento rys zvýrazňuje vysoké frekvence v obrazu. Jako filtr se používá někdy v tiskařském průmyslu. V definici znamená *f(x,y)* opět intenzitu bodu na pozici *(x,y)*, *Med(x,y)* značí medián s centrálním bodem *(x,y)*.

$$f_8(x,y) = 2f(x,y) - Med(x,y)$$

Vlastní intenzita bodu

Jde o intenzitu bodu na pozici *(x,y)*, pouze pro úplnost:

$$f_9(x,y) = f(x,y)$$

2. 4 Rysy použité pro klasifikaci

Pro klasifikaci segmentovaných obrazů jsem použil Haralickovy rysy stejné jako pro segmentaci, pouze matice sousednosti intenzit nebyla počítána z obrazu, ale ze segmentované masky (maska – viz 4. kapitola o implementaci). Matice sousednosti se navíc generovala z celé masky a nikoli lokálně jako v případě segmentace, tedy všechny odvozené Haralickovy rysy lze v tomto případě označit jako globální. Jediným novým (také globálním) rysem byl poměr počtu segmentů tkáně k počtu segmentů pozadí.

3. Umělé neuronové sítě

3. 1 Motivace

Umělé neuronové sítě (dále jen neuronové sítě) získaly svůj název díky podobnosti se sítěmi biologických neuronů. Původně šlo především o výzkum funkce lidského mozku, kdy zjednodušené matematické modely měly pomoci pochopit některé vlastnosti tohoto prozatím nepříliš probádaného fenoménu. Později se ale matematické modely začaly svému vzoru vzdalovat a řešit spíše praktické úlohy z nejrůznějších oblastí, pro některé se ukázaly být dokonce výhodnějším řešením než klasické postupy. Vyvinulo se mnoho různých modelů, lišících se topologií, aktivačními funkcemi, procesem učení a mnoha dalšími parametry. V této kapitole se po krátkém úvodu pokusím zaměřit pouze na modely, které by mohly být užitečné z hlediska zpracování obrazu, zejména na model vícevrstvé sítě s učícím pravidlem *backpropagation*, zmíním sítě typu *RBF*. V části o *samoorganizaci* uvedu *Kohonenovy samoorganizační mapy* a jejich použití pro klasifikaci dat (algoritmus *LVQ*). V samém závěru kapitoly okrajově zmíním možnosti použití dalších modelů neuronových sítí, např. jako asociativních pamětí. Zájemce o detailnější popis, příp. o další modely, odkazuji na [Šíma a Neruda 96], [Petersen et al. 02] a [Bishop 95], odkud jsem čerpal většinu zde uváděných informací.

3. 2 Formální neuron

K získání alespoň základního povědomí o neuronových sítích musíme začít od jejich prvků – neuronů. Tzv. formální neuron je ve své podstatě zformalizováním (doposud poznané) funkce svého biologického protějšku. Biologický neuron (velmi zjednodušeně) odpovídá na vzruchy přijaté od jiných neuronů změnou svého stavu, kterou mohou zachytit další neurony. Neuron v aktivním stavu posílá vzruch dál (excitační funkce neuronu), pokud neuron aktivní není, vzruch se dál nedostává (inhibiční funkce). Podobně formální neuron (dále jen neuron) získává informaci ze svého okolí pomocí několika vstupů, na jejich základě spočte svůj *vnitřní potenciál*, který po dosažení jisté prahové hodnoty indukuje aktivaci neuronu (obr. 4.1).

Formálně má tedy neuron obecně n reálných vstupů $x_1,..x_n$ ohodnocených n reálnými vahami $w_1,..w_n$ určujícími propustnost vstupů. Vnitřní potenciál ξ se spočte jako vážená suma:

$$\xi = \sum_{i=1}^{n} w_i x_i$$

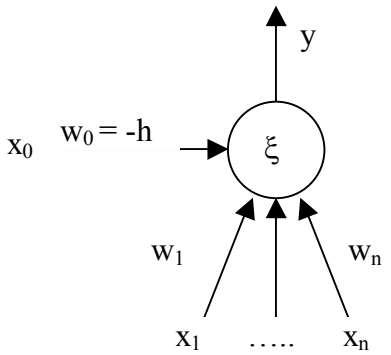
Stav (výstup) neuronu je dán *aktivační funkcí* vnitřního potenciálu $y = \sigma(\xi)$, přičemž nejčastějším příkladem aktivační funkce je tzv. *ostrá nelinearita*:

$$\sigma(\xi) = \begin{cases} 1 & pro\ \xi \geq h \\ 0 & jinak \end{cases},$$

kde *h* značí *práh* aktivace neuronu. Úpravou můžeme prahovou hodnotu přesunout do nulté váhy, tzv. *biasu*, díky tomu bude mít aktivační funkce nulový práh:

$$w_0 = -h$$
$$x_0 = 1$$
$$y = \sigma(\xi) = \begin{cases} 1 & pro\ \xi \geq 0 \\ 0 & jinak \end{cases}, \quad kde \quad \xi = \sum_{i=0}^{n} w_i x_i$$

Formální neuron je však schopen vyřešit pouze tzv. lineárně separabilní problémy. Představme si, že vstupy x_i určují souřadnice bodu v euklidovském prostoru E_n. Nadrovina daná rovnicí $w_0 + \sum_{i=1}^{n} w_i x_i = 0$ dělí tento prostor na dva podprostory. Zařazení bodu se souřadnicemi $[x_1,..,x_n]$ v E_n do příslušného podprostoru odpovídá hodnotě aktivace neuronu při předložení vzoru s hodnotami $[x_1,..,x_n]$. Např. při segmentaci obrazu může odpovídat aktivace neuronu (tj. y = 1) označení daného segmentu jako tkáň, pokud je naopak neuron pasivní (tj. y = 0), označíme segment jako pozadí obrazu. Při správné segmentaci pak všechny body E_n ležící v jednom poloprostoru a na nadrovině odpovídají vzorům reprezentujícím tkáň, body v opačném poloprostoru vzorům pozadí. Proces adaptace vah si lze představit jako úpravu koeficientů nadroviny, tedy jako její „natočení“ takovým směrem, aby se bod reprezentující chybně klasifikovaný vzor dostal do opačného poloprostoru. Je zřejmé, že pro jistá rozložení bodů v prostoru E_n není možné body správně rozdělit do podprostorů daných jednou nadrovinou. O problémech reprezentovaných takovým rozložením bodů říkáme, že nejsou lineárně separabilní. Často uváděným příkladem problému, který není lineárně separabilní, je výpočet logické funkce XOR (exklusivní disjunkce). V E_2 totiž není možné od sebe oddělit přímkou množiny bodů {(0,0), (1,1)} a {(0,1), (1,0)}. Neschopnost neuronu řešit jiné než lineárně separabilní problémy lze překonat spojováním několika neuronů do sítě.



obr. 4.1 – formální neuron

3. 3 Síť neuronů

Neuronová síť se skládá z neuronů spojených obecně tak, že výstup jednoho neuronu může být vstupem několika dalším neuronům. Počet neuronů a způsob jejich vzájemného propojení určuje tzv. *topologii* sítě. Neurony rozdělujeme podle funkce na *vstupní*, *skryté* (*pracovní*) a *výstupní*. Mohou být zapojeny do cyklu, pak hovoříme o tzv. *rekurentní* topologii, v opačném

případě o *dopředné* (*feed forward*). Obecně platí, že čím více neuronů je zapojeno do sítě, tím složitější problém je síť schopna řešit. Na druhou stranu u sítě s velkým počtem neuronů hrozí snížení schopnosti generalizace díky přesnému naučení tréninkových vzorů bez dostatečného zobecnění zákonitostí mezi nimi (tzv. *přeučení*, *overfitting*). Navíc s počtem neuronů v síti roste náročnost výpočtu sítě.

V případě dopředné topologie bývají neurony organizovány do tzv. *vrstev*. Do nulté (vstupní) vrstvy patří vstupní neurony, další vrstvu tvoří neurony, jejichž vstupy jsou stavy vstupních neuronů, třetí vrstvu tvoří neurony pracující se stavy neuronů druhé vrstvy a tímto způsobem postupujeme až k výstupním neuronům, jejichž vrstva se nazývá výstupní. Vrstvy číslujeme od nuly a vstupní vrstvu do celkového počtu vrstev sítě nepočítáme. Pro úplnost dodejme, že obecně je možné, aby spoje některou vrstvu „přeskočily“, a spojovaly tak např. neuron druhé vrstvy s neuronem vrstvy páté, ale pro jednoduchost u dopředných sítí tuto situaci neuvažujeme. Protože v dopředné síti je spojen každý neuron se všemi neurony bezprostředně následující vrstvy (chybějící spoje lze chápat jako spoje s nultými váhami), lze topologii dopředné vícevrstvé sítě zadat pouze počtem neuronů v jednotlivých vrstvách. Počty neuronů zadáváme od vstupní k výstupní vrstvě a oddělujeme je pomlčkou, např. topologie 3-4-3-2 znamená síť se třemi vstupními neurony, dvěma výstupními a se dvěma skrytými vrstvami.

Stav neuronové sítě odpovídá stavům všech neuronů v síti, *konfigurace* sítě hodnotám vah všech neuronů v síti. Síť můžeme zadat pomocí tří charakteristik – organizační, aktivní a adaptivní dynamiky. Tyto dynamiky bývají určeny počátečním stavem a pravidly, jak se bude daná dynamika vyvíjet v čase. *Modelem* neuronové sítě rozumíme konkretizaci těchto tří dynamik.

Organizační dynamika stanoví počet neuronů v síti a jejich vzájemné propojení (tedy *topologii* sítě) a její případnou změnu. Většina modelů má statickou organizační dynamiku, ale je možné např. v procesu učení k síti přidávat další neurony nebo naopak neurony ubírat (některé varianty backpropagation).

Aktivní dynamika reprezentuje počáteční stav sítě a způsob, jakým se mění v čase, při zachování pevné topologie a konfigurace. Nejprve se nastaví stavy vstupních neuronů na vstup sítě. Změna stavu sítě v čase může být *spojitá*, pak je aktivní dynamika popsána diferenciální rovnicí, nebo *diskrétní*, tj. stav se aktualizuje pouze v čase 0, 1, 2, … . Diskrétní dynamika je častější pro svou nižší výpočetní náročnost. Výsledek výpočtu sítě je dán stavy výstupních neuronů, které se sice obecně s časem mění, ale většina uvažovaných dynamik po určité době běhu sítě ponechá stavy výstupních neuronů konstantní. Díky tomu můžeme aktivní dynamiku považovat za realizaci nějaké funkce – vstupu přiřadí výstupní hodnotu.

Funkce jednotlivých neuronů je také určena aktivní dynamikou, v případě *homogenní* sítě je pro všechny neurony kromě vstupních stejná. Funkci neuronů ovlivňuje jak výběr aktivační funkce, tak funkce počítající vnitřní potenciál neuronu. Aktivační funkcí může být např. ostrá nelinearita (tzv. *hard limiter*), standardní sigmoida (spojitě aproximuje ostrou nelinearitu), hyperbolický tangens nebo saturovaná lineární funkce. Vnitřní potenciál může kromě klasicky použité vážené sumy odpovídat např. vzdálenosti vstupu od váhového vektoru (viz samoorganizace), případně polynomu (neuronové sítě vyšších řádů).

Příklady *aktivačních funkcí* neuronu:

ostrá nelinearita

$$\sigma(\xi) = \begin{cases} 1 & pro\ \xi \geq 0 \\ 0 & jinak \end{cases}$$

saturovaná lineární funkce $$\sigma(\xi) = \begin{cases} 1 & pro\ \xi > 1 \\ \xi & 0 \le \xi \le 1 \\ 0 & \xi < 0 \end{cases}$$

standardní sigmoida $$\sigma(\xi) = \frac{1}{1+e^{-\xi}}$$

hyperbolický tangens $$\sigma(\xi) = \frac{1-e^{-\xi}}{1+e^{-\xi}}$$

Adaptivní dynamika určuje počáteční konfiguraci sítě a způsob změn vah v čase. Adaptivní režim sítě začíná nastavením vah všech neuronů na počáteční hodnotu, většinou náhodně. Poté se podle učícího pravidla (algoritmu) adaptují jednotlivé váhy tak, aby bylo dosaženo cíle adaptace, tj. realizace požadované funkce (viz aktivní režim). Funkce sítě bývá zadána tréninkovou množinou – dvojicemi $[x_i, d_i]$, kde x_i představuje i-tý vstup sítě (stavy vstupních neuronů) a d_i k němu odpovídající požadovaný výstup sítě (stavy výstupních neuronů). Těmto dvojicím říkáme *vzory*. Uvedený přístup odpovídá tzv. učení s učitelem. Učení bez učitele znamená absenci požadovaných výstupů, síť se učí pouze na základě samoorganizace vstupních dat podle jejich společných vlastností, které v průběhu učení odhaluje. Takovému přístupu říkáme *samoorganizace*.

3. 4 Vícevrstvá síť a backpropagation

Model vícevrstvé sítě používající učící pravidlo backpropagation (také nazývané strategií zpětného šíření chyby) se používá přibližně v 80% aplikací neuronových sítí. Nejprve charakterizujeme všechny tři dynamiky, na závěr doplním některé možnosti jeho rozšíření.

Organizační dynamika specifikuje ve většině případů pevnou topologii s jednou až dvěma skrytými vrstvami. Význam skrytých neuronů není znám, proto neexistují žádné metodiky návrhu speciální topologie a neurony bývají propojeny každý s každým. Počet skrytých neuronů by měl odpovídat složitosti řešení problému, ta se bohužel v praxi odhaduje obtížně.

Pro popis *aktivní* a *adaptivní dynamiky* zavedeme několik termínů. Množinu n vstupních neuronů budeme značit X, množinu m výstupních neuronů Y. Neuronům tvořícím tuto síť někdy říkáme *perceptrony* a značíme je indexy, ξ_j představuje *vnitřní potenciál* a y_j *stav* j-tého neuronu v síti. Spoj vedoucí od i-tého k j-tému neuronu v síti je ohodnocen reálnou vahou w_{ji}. Nultá váha odpovídá *biasu*, tedy hodnotě prahu aktivační funkce j-tého neuronu s opačným znaménkem, tj. $w_{j0} = -h_j$. Konečně $j_{\leftarrow}$ označuje množinu indexů všech neuronů, které jsou j-tému neuronu vstupem (tj. vedou z nich spoje do j-tého neuronu), a $j^{\rightarrow}$ značí množinu indexů všech neuronů, jejichž vstupem je j-tý neuron (tj. vedou do nich spoje z j-tého neuronu).

Aktivní dynamika vícevrstvé sítě počítá pro daný vstup funkci $y(w) : R \rightarrow (0,1)^m$, kde w je konfigurace sítě. Na začátku výpočtu (v čase nula) se nastaví hodnoty vstupních neuronů (tj. $y_j \quad pro\ j \in X$) na vstup sítě, ostatní neurony nemají stav určen. Dynamika je diskrétní, pro

každý následující krok výpočtu se stanoví reálné vnitřní potenciály neuronů, jejichž vstupy již mají určen svůj stav, podle předpisu

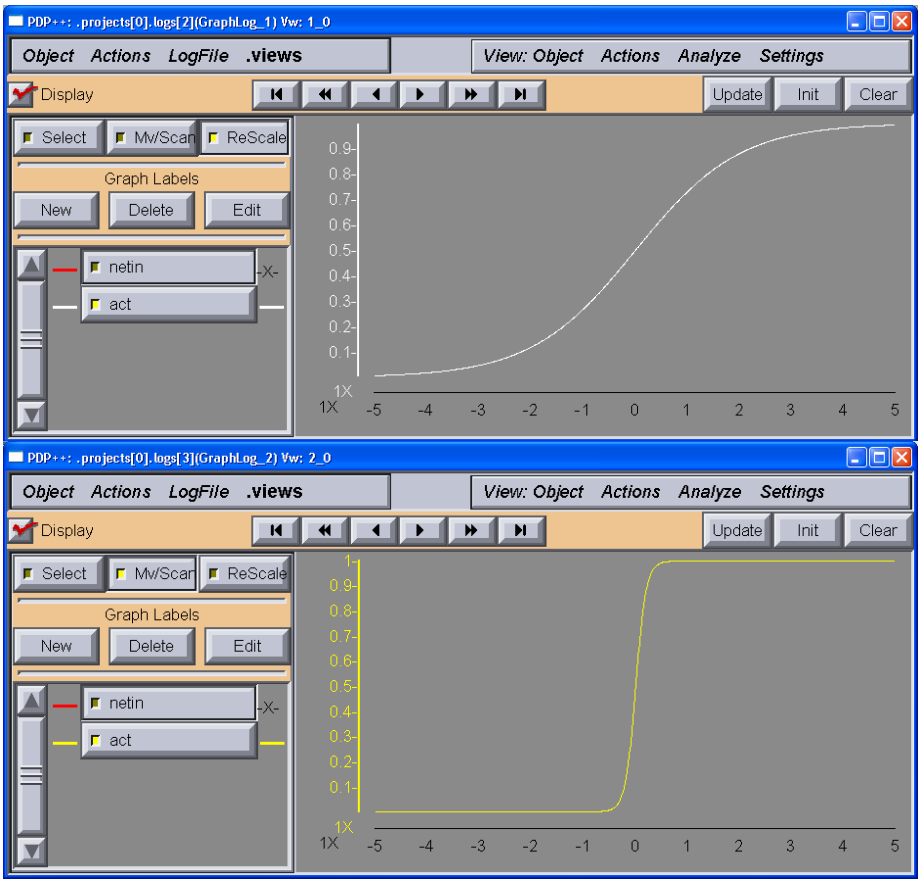
$$\xi_j = \sum_{i \in j_{\leftarrow}} w_{ji} y_i \ .$$

Výpočet tedy probíhá po jednotlivých vrstvách. Z vnitřních potenciálů se následně stanoví stavy neuronů pomocí aktivační funkce standardní sigmoidy, která je diferencovatelná (což je podstatný předpoklad pro učící algoritmus backpropagation), jako

$$y_j = \sigma_j(\xi_j) = \frac{1}{1 + e^{-\lambda \xi_j}} \ .$$

Reálný parametr λ určuje *strmost* funkce (tzv. *gain*), standardně se volí $\lambda = 1$ pro všechny neurony v síti, ale obecně tomu tak být nemusí. Pro $\lambda = \infty$ přechází standardní sigmoida v ostrou nelinearitu, strmost funkce tedy ovlivňuje míru rychlosti „rozhodování“ neuronu. Příklady grafů funkce pro různé hodnoty λ viz obr. 4.2.

Vícevrstvá síť je acyklická, takže při konečném počtu neuronů vypočteme v konečném čase stavy výstupních neuronů reprezentující výsledek funkce sítě pro daný vstup.



obr. 4.2 – standardní sigmoida pro λ = 1 (nahoře, bíle) a λ = 10 (dole, žlutě)

V režimu *adaptace* síť přizpůsobuje hodnoty vah jednotlivých spojení vzhledem k dané tréninkové množině T definované jako

$$T = \left\{ (x_k, d_k) \;\middle|\; \begin{array}{l} x_k = (x_{k1},..,x_{kn}) \in R^n \\ d_k = (d_{k1},...,d_{km}) \in \langle 0,1 \rangle^m \end{array} \quad pro\ k = 1,...,p \right\},$$

kde p je počet tréninkových vzorů. Cílem adaptace je, aby hodnota sítě pro každý tréninkový vzor x_k z T byla rovna požadovanému výstupu sítě d_k (*d - desired*). Formálně, aby

$$y(w, x_k) = d_k \quad pro\ k = 1,..,p\,.$$

Definujeme *chybu sítě E(w)* vzhledem k tréninkové množině jako součet *parciálních chyb sítě* $E_k(w)$ vzhledem k jednotlivým vzorům jako

$$E(w) = \sum_{k=1}^{p} E_k(w)\,.$$

Parciální chyba sítě $E_k(w)$ je přímo úměrná součtu mocnin rozdílu hodnot výstupních neuronů pro vstupy *k*-tého tréninkového vzoru a odpovídajících hodnot požadovaného vzoru d_k, tedy

$$E_k(w) = \frac{1}{2} \sum_{j \in Y} (y_j(w, x_k) - d_{kj})^2\,.$$

Cílem adaptace je minimalizace chyby sítě *E(w)* pomocí gradientní metody, která vyžaduje diferencovatelnost chybové funkce (odtud i požadavek na diferencovatelnost aktivační funkce neuronu). Samotná adaptace probíhá v diskrétních krocích a začíná v čase nula inicializací vah hodnotami blízkými nule. Pro čas adaptace $t > 0$ určíme hodnotu váhy spoje z *i*-tého do *j*-tého neuronu pomocí následujícího učícího pravidla:

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} + \Delta w_{ji}^{(t)} \quad ,$$

kde změna vah $\Delta w^{(t)}$ v čase *t* odpovídá přímo úměrně zápornému gradientu chybové funkce *E(w)* v bodě $w^{(t-1)}$:

$$\Delta w_{ji}^{(t)} = -\varepsilon \frac{\delta E}{\delta w_{ji}}(w^{(t-1)})\,,$$

parametr $0 < \varepsilon < 1$ se nazývá *rychlost učení* (*learning rate*).

Gradientní metoda se (velmi zjednodušeně) při výpočtu nových vah posune v grafu chybové funkce o -ε-násobek tečného vektoru (gradientu) chybové funkce sítě v bodě $w^{(t-1)}$ dolů, takže z jakékoli počáteční konfigurace se nakonec dostane do lokálního minima chybové funkce, kde se proces zastaví (díky nulovému gradientu). Nemáme ovšem zaručeno, že nalezené minimum je globální.

Výpočet gradientu realizujeme pomocí tzv. *strategie zpětného šíření chyby* (odtud název *backpropagation*). Nejprve převedeme gradient na součet gradientů parciálních chybových funkcí (podle pravidla o derivaci součtu):

$$\frac{\delta E}{\delta w_{ji}} = \sum_{k=1}^{p} \frac{\delta E_k}{\delta w_{ji}}.$$

Dále použijeme pravidlo pro derivaci složené funkce, abychom se dostali k hodnotám jednotlivých neuronů:

$$\frac{\delta E_k}{\delta w_{ji}} = \frac{\delta E_k}{\delta y_j} \frac{\delta y_j}{\delta \xi_j} \frac{\delta \xi_j}{\delta w_{ji}}.$$

Poslední derivaci na pravé straně rovnice získáme derivováním vnitřního potenciálu

$$\frac{\delta \xi_j}{\delta w_{ji}} = y_i \quad neboť\ \xi_j = \sum_{r \in j_{\leftarrow}} w_{jr} y_r ,$$

prostřední člen obdržíme derivací aktivační funkce

$$\frac{\delta y_j}{\delta \xi_j} = \frac{\lambda_j e^{-\lambda_j \xi_j}}{(1+e^{-\lambda_j \xi_j})^2} = \lambda_j \frac{1}{1+e^{-\lambda_j \xi_j}} \left(1 - \frac{1}{1+e^{-\lambda_j \xi_j}}\right) = \lambda_j y_j (1-y_j).$$

Gradient parciální chybové funkce můžeme tedy počítat podle vzorce

$$\frac{\delta E_k}{\delta w_{ji}} = \frac{\delta E_k}{\delta y_j} \lambda_j y_j (1-y_j).y_i .$$

Zbylou derivaci $\frac{\delta E_k}{\delta y_j}$ vyřešíme podle strategie zpětného šíření následovně:

Neuron s indexem *j* patří mezi *výstupní* neurony (tj. $j \in Y$), pak $\frac{\delta E_k}{\delta y_j} = y_j - d_{kj}$.

Neuron s indexem *j* je *skrytý* neuron (tj. $j \notin (X \cup Y)$), pak určení derivace $\frac{\delta E_k}{\delta y_j}$ převedeme na výpočet $\frac{\delta E_k}{\delta y_r}$, kde *r* značí indexy neuronů, do kterých vedou spoje z neuronu s indexem *j*:

$$\frac{\delta E_k}{\delta y_j} = \sum_{r \in j^{\rightarrow}} \frac{\delta E_k}{\delta y_r} \frac{\delta y_r}{\delta \xi_r} \frac{\delta \xi_r}{\delta y_j} = \sum_{r \in j^{\rightarrow}} \frac{\delta E_k}{\delta y_r} \lambda_r y_r (1-y_r).w_{rj} .$$

Na základě uvedených vzorců probíhá adaptace za účasti aktivního režimu, kdy síť nejprve spočte vnitřní potenciály a stavy jednotlivých neuronů pro daný tréninkový vzor, poté postupně od výstupních neuronů až k první skryté vrstvě propaguje hodnoty chyby, na jejichž základě upravuje hodnoty vah. U tzv. *on-line* adaptace probíhá aktualizace vah po každém tréninkovém vzoru, při adaptaci *off-line* až po určitém počtu vzorů, případně po průchodu celou tréninkovou množinou (ovšem změny vah – člen $\Delta w_{ji}^{(t)}$ - se příslušně akumulují).

Při adaptaci pomocí backpropagation někdy zavádíme další veličiny do pravidla pro adaptaci vah. Existují heuristiky, jak automaticky volit hodnoty ε (rychlosti učení). Do výpočtu chyby můžeme také zahrnout i parametr λ (*strmosti* u aktivační funkce), což umožní procesu učení nastavovat i tento parametr.

Mezi často využívané modifikace pravidla pro adaptaci vah patří varianta s tzv. *momentem*. Díky němu gradientní metoda lépe opisuje tvar chybové funkce *E(w)*, protože zohledňuje předchozí změnu vah. Změna vah se v modifikovaném učícím pravidle určí jako

$$\Delta w_{ji}^{(t)} = -\varepsilon \frac{\delta E}{\delta w_{ji}}(w^{(t-1)}) + \alpha \Delta w_{ji}^{(t-1)},$$

kde $0 < \alpha < 1$ se nazývá parametr momentu a určuje míru vlivu předchozí změny vah, obvykle se volí kolem 0,9.

Další možností modifikace procesu učení je eliminace příliš malých vah, tímto způsobem lze dosáhnout podobného efektu jako při tzv. *prořezávání* (viz dále). V definici chybové funkce přidáme člen penalizující váhy úměrně jejich velikosti, např.

$$E'(w) = E(w) + \frac{1}{2}\gamma \sum_{j,i} \frac{w_{ji}^2}{1 + w_{ji}^2},$$

kde *E(w)* je původní chybová funkce a $\gamma > 0$ tzv. *decay* ovládající míru vlivu penalizace vah na chybovou funkci.

Poslední zde uváděnou modifikací adaptivní dynamiky vícevrstvé sítě je její spojení s organizační dynamikou, jinými slovy během učení se přidávají nebo ubírají neurony, příp. jejich spojení. Rozlišujeme dva základní přístupy, *konstrukční* algoritmy začínají s malou topologií a neurony postupně přidávají. Naopak tzv. *prořezávání* (*pruning*) odstraňuje spoje (příp. i odpovídající neurony), které mají v průběhu adaptace dostatečně malou váhu. Detailnější popis a odkazy na další zdroje je možné najít v [Šíma a Neruda 96], konkrétní algoritmy v [Bishop 95].

Zájemce o popis funkcí vhodných pro výpočet *parciální chyby sítě* odkazuji na [Bishop 95]. Tamtéž lze nalézt např. techniku *přidávání šumu* ke vstupům sítě v průběhu učení pro zamezení přeučení sítě a tedy zlepšení celkové generalizace.

Síť typu backpropagation lze při zpracování obrazu použít obecně jako klasifikátor, např. jednotlivých částí obrazu, na základě této klasifikace může být založena segmentace.

3. 5 Síť typu RBF

Sítě typu RBF připomínají vícevrstvé sítě uvedené výše, odlišují se však funkcí vnitřního potenciálu neuronu a aktivační funkcí. Neuronům v RBF sítích se říká RBF jednotky.

Aktivní dynamiku určíme zadáním funkce vnitřní potenciálu RBF jednotky

$$\xi = \frac{\|x - c\|}{b},$$

tedy jako *vzdálenost* vstupu od *středu c*, případně dělenou *šířkou b*.

Tento typ funkce patří do kategorie tzv. *radiálních bazických funkcí* (odtud název RBF). Často se říká, že takovéto funkce mají *lokální charakter*, neboť relevantní hodnoty jsou soustředěny do jistého okolí středu c. Velikost okolí ovlivňuje parametr *b,* nazývaný *šířka.*

Definici aktivní dynamiky RBF jednotek doplníme ukázkou několika používaných aktivačních funkcí, od identity po Gaussovu:

$$y(\xi) = \xi,$$

$$y(\xi) = \xi^2 \log \xi,$$

$$y(\xi) = e^{(-\frac{\xi}{\beta})^2}, \quad \beta \geq 0.$$

RBF sítě úspěšně řeší problémy, se kterými mohou mít sítě s perceptrony potíže, bohužel to platí i naopak. Například realizace funkce XOR pomocí jedné RBF jednotky je snadná, zatímco pro klasický model jednoho neuronu (perceptronu) se jedná o neřešitelný problém (viz kapitola 3.2). RBF sítím také stačí k řešení obecně jakéhokoli problému dostatečný počet neuronů v pouze jedné skryté vrstvě, typická *organizační dynamika* definuje jednu vstupní vrstvu, jednu skrytou vrstvu s RBF jednotkami, výstupní vrstva je realizována neurony počítajícími váženou sumu svých vstupů (tedy stavů RBF jednotek).

Adaptivní dynamika probíhá podobně jako u vícevrstvé sítě s perceptrony, lze použít mírně pozměněný postup backpropagation. Navíc je nutné při učení definovat parametry funkcí vnitřního potenciálu – *středy* a *šířky*, proto se učení RBF jednotek někdy říká *trojfázové učení*. Pro určení středů existuje několik postupů od náhodného rozložení středů po jejich umístění pomocí samoorganizace. Šířky RBF jednotek určíme formulací chybové funkce závislé na *šířce*, a její minimalizací. Formulaci chybové funkce lze nalézt v [Šíma a Neruda 96], podrobnější popis RBF sítí např. v [Bishop 95].

Z hlediska zpracování obrazu je možné RBF sítě použít jako sítě typu backpropagation, např. tedy pro klasifikaci segmentů obrazu, ovšem vhodnost použití jednoho typu sítě oproti druhému záleží na konkrétním problému.

3. 6 Samoorganizace

Termínu *samoorganizace* často používáme v souvislosti s tzv. *učením bez učitele*, kdy síť v adaptivním režimu nedostává zpětnou vazbu o chybě a namísto toho se sama snaží odhalit *podobnosti* mezi jednotlivými vstupy. *Podobným* vstupům pak přísluší aktivace stejného výstupního neuronu – *reprezentanta* těchto vstupů. Společným principem samoorganizačních modelů je *soutěžní strategie* (*kompetice*) učení i aktivního režimu – neurony spolu soutěží o to, který z nich bude aktivní, tedy v daném čase je aktivní nejvýše jeden. Pro zajímavost uveďme, že některé oblasti lidského mozku také pracují na soutěžním principu, což bylo experimentálně ověřeno – tvrdí [Šíma a Neruda 96].

Známou aplikací samoorganizace je řešení problému *vektorové kvantizace* (*vector quantization - VQ*), kdy jde o aproximaci rozložení vstupů ve vstupním prostoru pomocí konečně mnoha reprezentantů. Po určení reprezentantů přiřadíme každému vstupu toho reprezentanta, který je mu nejblíže. VQ řeší *Lloydův algoritmus*, jeho on-line variantou je *Kohonenovo učení*, detailní popis obou algoritmů lze nalézt v [Šíma a Neruda 96]. Blíže zde uvedeme až model nazvaný *Kohonenova samoorganizační mapa* (*Self-organizing map, SOM*), jde o zajímavé rozšíření Kohonenova učení.

3. 6. 1 Kohonenova samoorganizační mapa

Organizační dynamika sítě specifikuje (kromě klasické vstupní vrstvy pro distribuci dat) jen vrstvu výstupních neuronů, které hrají roli reprezentantů jednotlivých vstupů. Výstupní neurony jsou u tohoto modely uspořádány do nějaké topologické struktury (odlišnost od původního Kohonenova učení), většinou dvojrozměrné mřížky nebo jednorozměrné řady. Váhy příslušející výstupnímu neuronu určují jeho polohu ve vstupním prostoru, všechny neurony výstupní vrstvy jsou propojeny s každým neuronem vrstvy vstupní.

Aktivní dynamika n-rozměrným reálným vstupům *x* přiřazuje aktivitu právě jednoho výstupního neuronu *j*, což znamená reprezentaci vstupu *x* reprezentantem *j*. Výstupní neurony mívají hodnoty z množiny {0, 1} (nikoli [0, 1]). Výstup každého neuronu určíme podle jeho vzdálenosti od vstupu *x*, kompetici „vyhrává“ ten nejblíže vstupu, konkrétně

$$y_j = \begin{cases} 1 & pro\ j = \arg\min_{j=1..h} \left\{ \| x - w_l \| \right\} \\ 0 & jinak \end{cases},$$

kde *h* značí počet reprezentantů (tedy výstupních neuronů). Vítězný neuron má stav roven jedné, všechny ostatní nule.

Adaptivní dynamika vychází také z principů kompetice, váhy si při každém předložení tréninkového vzoru upraví pouze vítězná jednotka a jednotky z jejího *okolí*. To je možné díky znalosti topologie výstupních neuronů. Pojem *okolí* $N_s(c)$ velikosti *s* neuronu *c* zavedeme jako

$$N_s(c) = \left\{ j \mid d(j,c) \le s \right\}.$$

Vzdálenost *d(j,c)* dvou výstupních neuronů závisí na použité topologické struktuře výstupní vrstvy.

Samotná úprava vah probíhá pro každý tréninkový vzor podle vzorce

$$w_{ji}^{(t)} = \begin{cases} w_{ji}^{(t-1)} + \theta(x_i^{(t)} - w_{ji}^{(t-1)}) & j \in N_s(c) \\ w_{ji}^{(t-1)} & jinak \end{cases},$$

kde $c = \arg\min_{l=1..h} \left\{ \left\| x^{(t)} - w_l \right\| \right\}$, h je počet výstupních neuronů, $s \in N$ velikost okolí, $0 < \theta \leq 1$ reálný parametr určující míru změny vah.

Velikost okolí *s* během učení měníme, nejčastěji na začátku velké (např. polovina velikosti sítě), na konci zahrnuje okolí pouze samotný vítězný neuron.

V [Šíma a Neruda 96] můžeme naleznout několik praktických rad pro učení Kohonenových map, např. počet iterací tréninku by měl být asi pětsetkrát větší než počet neuronů v síti, učení je vhodné rozdělit do dvou fází, *hrubé* s vysokými hodnotami *s* a θ a *dolaďovací*, kdy se oba parametry relativně málo mění a postupně dále klesají k nule. Počáteční hodnoty vah jednotlivých neuronů nehrají velkou roli, pokud zaručíme jejich (vesměs) vzájemnou různost.

3. 6. 2 LVQ pomocí Kohonenových SOM

Kohonenovy samoorganizační mapy lze s úspěchem použít ke *klasifikaci* dat (a na ní např. založit segmentaci obrazu). Jedním z použitelných algoritmů je *učící vektorová kvantizace* (*Learning Vector Quantization, LVQ*), kdy kombinujeme samoorganizaci s klasickým učením s učitelem. *Organizační* i *aktivní dynamika* sítě jsou analogické svým protějškům u SOM, dále upřesníme pouze učení sítě.

Tréninková množina obsahující *k* reálných vzorů pro *adaptivní dynamiku* LVQ má tvar

$$T = \left\{ (x^{(t)}, d^{(t)}) \mid t = 1,..,k \right\}, \quad x^{(t)} \in R, \quad d^{(t)} \in \left\{ C_1,..,C_q \right\},$$

kde *q* je počet kategorií C_i, do kterých mají být zařazovány jednotlivé vzory.

Vlastní učení probíhá ve třech fázích: Nejprve rozmístíme neurony po vstupním prostoru pomocí samoorganizace popsané v kapitole 3.6.1. Poté označíme výstupní neurony kategoriemi a nakonec síť doučíme pomocí jednoho z algoritmů LVQ.

K označení výstupních neuronů kategoriemi projdeme celou tréninkovou množinu a u každého vzoru zjistíme k němu nejbližší neuron. Pro každý výstupní neuron si zapamatujeme, do které kategorie neuronem reprezentovaný vzor patřil, a nejčastěji reprezentovanou kategorii neuronu přiřadíme.

V poslední fázi učení použijeme jeden ze tří algoritmů navržených Kohonenem pro doučení sítě – základní varianta (LVQ1) pouze posílí správnou klasifikaci vítězného neuronu přiblížením vah k tréninkovému vzoru, zatímco špatnou klasifikaci se pokouší napravit odsunutím neuronu od tréninkového vzoru ve vstupním prostoru. Detailní popis LVQ1 spolu s jeho sofistikovanějšími variantami LVQ2 a LVQ3 lze nalézt v [Šíma a Neruda 96].

3. 7 Další modely neuronových sítí

Příkladem dalších modelů použitelných při zpracování obrazu jsou *asociativní neuronové sítě*, které mohou fungovat jako *autoasociativní* nebo *heteroasociativní* paměti. Zájemce o detailní popis zde uvedených modelů odkazuji na [Šíma a Neruda 96], tamtéž se nachází pěkný příklad použití *Hopfieldovy sítě* k rekonstrukci obrazu číslice.

V případě autoasociativity jde o schopnost zúplnit danou informaci, např. rekonstruovat obraz poškozený šumem, rozpoznání číslic či písmen. Tréninkové množiny pro tyto modely mají vstupy rovny odpovídajícím výstupům (tj. $x_k = d_k$ pro všechna k z tréninkové množiny). Příkladem může být model *lineární asociativní paměti* nebo *Hopfieldovy sítě*. S autoasociativní pamětí je spojena adaptace podle *Hebbova zákona* (ve zkratce říká, že změna synaptické váhy mezi dvěma neurony je úměrná jejich souhlasné aktivitě – lze nalézt v [Šíma a Neruda 96]).

Naproti tomu heteroasociativní paměť si k dodanému vstupu vybaví informaci s ním asociovanou, např. jméno osoby rozpoznané na fotografii. Příkladem modelu tohoto typu asociativní paměti může být *Boltzmannův stroj* (stochastická varianta Hopfieldovy sítě se skrytými neurony).

4. Implementace

Řešení problému jsem rozdělil do dvou programů. Prvním je PDP++ verze 3.1 určený pro práci s neuronovými sítěmi, druhým je program Segmentation, který jsem vytvořil v rámci této práce. PDP++ je pro akademické použití volně šiřitelný, licenci k použití přikládám v příloze IV. Program Segmentation provádí segmentaci na základě klasifikace jednotlivých *segmentů*[2] obrazu jako tkáně či pozadí spočtením rysů pro každý segment obrazu a jejich předložením neuronové síti, která pro každý segment rozhodne, co reprezentuje. Výsledky výpočtu neuronové sítě program vizualizuje uživateli, příp. segmentovaný obraz dále předá neuronové síti ke klasifikaci, zda se jedná o obraz zdravého nebo nádorového střeva.

4. 1 Popis programu Segmentation

Program Segmentation je určen pro jednoduché předzpracování vstupního obrazu, výpočet rysů sloužících jako vstupy neuronové síti jak pro trénink, tak pro aktivní režim, a konečně k vizualizaci výsledků výpočtu neuronové sítě na konkrétním obraze.

Program je umístěn na doprovodném CD v adresáři */program/segmentation*, kde se nacházejí zdrojové texty programu. Je vytvořen ve vývojovém prostředí Microsoft Visual Studio .NET Professional (dále jen MS VS .NET) v jazyce C++. Programová hlášení a názvy nabídek (menu) jsou v angličtině. Spustitelných verzí přikládám několik, všechny se jmenují *segmentation.exe* a nacházejí se v následujících podadresářích adresáře s programem:

debug - jedná se o neoptimalizovanou verzi určenou pro ladění, doporučuji spouštět jen ve výjimečných případech

release - optimalizovaná verze používající knihovnu MFC v několika dynamicky připojovaných knihovnách (DLL), které jsou součástí MS VS .NET, bez těchto knihoven ji nebude možné spustit, výhodou je malá velikost

deploy - optimalizovaná verze s již připojenými knihovnami, tuto verzi doporučuji používat.

Další možností spuštění programu je jeho instalace pomocí instalačního programu */program/segmentation-instal/Setup.exe*, nainstaluje verzi *deploy* spolu se zdrojovými texty.

Pokud nastanou problémy se spouštěním programu, je možné, že je zapotřebí aktualizovat některé knihovny operačního systému (zvláště u starších verzí OS). Na doprovodném CD spusťte soubor */program/dotNetFramework/dotnefx.exe*, což je součást MS VS .NET určená pro aktualizaci operačního systému pro provoz aplikací napsaných v tomto vývojovém prostředí.

Po spuštění programu může uživatel provádět úkony popsané v následujících podkapitolách. Při popisu práce s programem budu používat následující zkratku: *File->Open* bude znamenat výběr nabídky File a její položky Open, podobně tomu bude i pro ostatní nabídky.

[2] *segmentem* nazývám čtvercovou oblast několika bodů v obraze charakterizovanou souřadnicemi centrálního bodu a velikostí oblasti (nastavitelné uživatelem).

4. 1. 1 Segmentace obrazu

i) Příprava dat pro aktivní režim sítě (předzpracování, výpočet rysů)

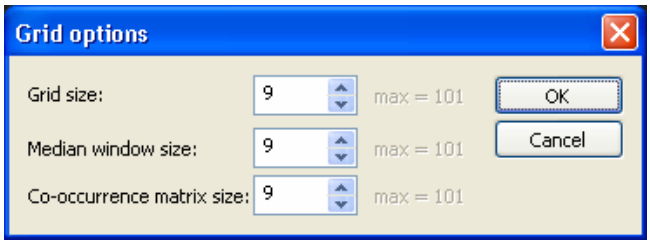
Pro výpočet rysů potřebujeme vstupní šedotónní obraz s intenzitami v rozsahu 0..255. Zvolíme *File->Open* a vybereme vstupní obraz. Pokud barevná hloubka obrazu neodpovídá výše uvedeným podmínkám, převedeme obraz na šedotónní pomocí *Tools->Convert To Grayscale*. Pro výpočet rysů obrazu slouží volba *Analysis->Compute features*. Po jejím zvolení se objeví dialogové okno s možnostmi výpočtu (obr. 5.1):

Grid size - velikost jednoho segmentu mřížky. S ohledem na rychlost všech výpočtů a z důvodu snadnější práce s maskou (viz dále) nabízí program možnost pracovat místo s jednotlivými obrazovými body se čtvercovými oblastmi bodů (segmenty). Segment je pak charakterizován centrálním bodem a velikostí, ta musí být lichá z důvodu snadné manipulace s centrem. Výchozí velikost segmentu je 9x9 bodů.

Median window size - velikost čtvercového okolí centrálního bodu segmentu, ze kterého se počítají rysy medián a unsharp masking.

Co-occurrence matrix size - velikost čtvercového okolí centrálního bodu segmentu, ze kterého se počítá matice sousednosti intenzit použitá pro výpočet Haralickových rysů.

Délka vlastního výpočtu silně závisí na nastavených parametrech, proto nedoporučuji zkoušet příliš vysoké hodnoty.



obr. 5.1

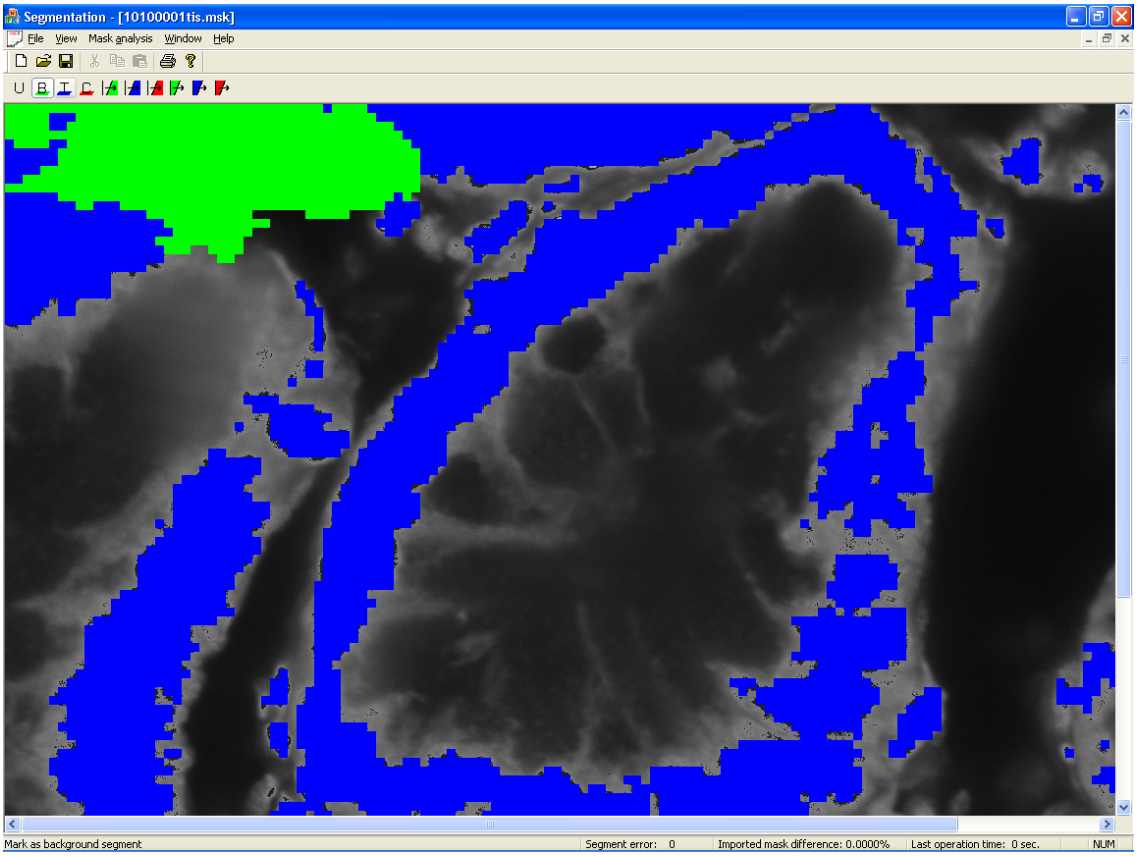
Po skončení výpočtu se objeví okno s tabulkou vypočtených rysů. Každý řádek symbolizuje jeden obrazový segment, první sloupec udává číslo segmentu dané pořadím v průchodu obrazem „po řádcích“, ostatní sloupce pak udávají hodnoty jednotlivých devíti rysů. Vypočtené rysy je možné uložit do souboru v nativním formátu, pro další zpracování je nutný export do textové podoby. Pro předání dat k segmentaci obrazu (aktivní režim sítě) se používá volba *File->Export to PDP+ Environment format*, která uloží rysy do textového souboru čitelného programem PDP++.

Následuje fáze zpracování dat v PDP++, které se budu věnovat v podkapitole 4.2.

ii) Příprava dat pro adaptivní režim sítě, maska obrazu

Pro adaptivní režim sítě potřebujeme kromě vstupních dat (rysů) také správné výsledky, tj. správně segmentované obrazy. Jednotlivé rysy spočítáme stejně jako pro aktivní režim a uložíme je v nativním formátu do souboru. Poté zvolíme nabídku *File->New* pro vytvoření nové masky.

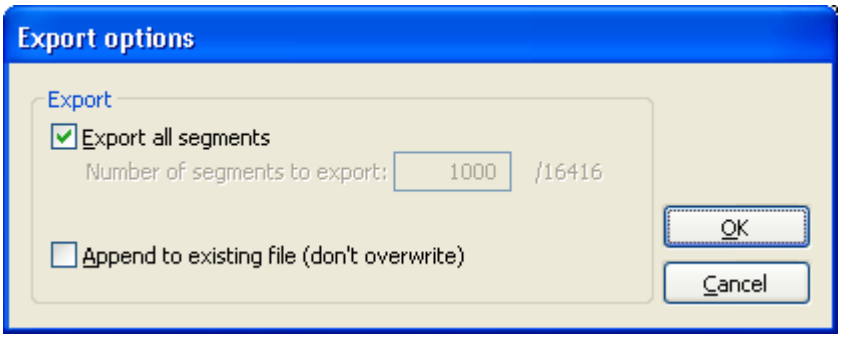
Maska představuje záznam o klasifikaci jednotlivých segmentů obrazu jako tkáň či pozadí. Velikost segmentu masky musí být shodná s velikostí segmentu u vypočtených rysů, navíc počet segmentů by se měl v obou případech shodovat (vstupní i výstupní data sítě reprezentují stejný obraz). Při vytváření nové masky se program zeptá nejprve na velikost mřížky (tj. velikost segmentu), poté na soubor s referenčním obrazem, kterým je myšlen vstupní obraz, jehož maska se má vytvořit. Podle referenčního obrazu se automaticky nastaví velikost masky a obraz se vykreslí na obrazovku překryt mřížkou znázorňující jednotlivé segmenty. Vlastní masku tvoří uživatel vybráním některé z ikon segmentačních tříd na horním okraji okna a vyznačením odpovídajících segmentů myší (obr. 5.2). Segmentační třídou je myšlena třída, do které může být nějaký segment po segmentaci přiřazen, např. pozadí obrazu (*background*), tkáň (*tissue*), rakovinná tkáň (*cancerous tissue*). Neoznačené segmenty se zobrazují jako průhledné (je vidět referenční obraz), pozadí zelenou barvou, tkáň modrou barvou a rakovinná tkáň červenou. Uživatel může masku i mřížku skrýt v nabídce *View*. Třídu rakovinná tkáň jsem při segmentaci nakonec nepoužil, protože pro klasifikaci zdravé a rakovinné tkáně slouží další neuronová síť používající jiné rysy, ale v programu se tato možnost může osvědčit, pokud by uživatel chtěl vyzkoušet jiný přístup.



obr. 5.2 – tvorba masky

Hotovou masku můžeme opět uložit v nativním formátu (pozor, ukládá se pouze odkaz na soubor s referenčním obrazem), pro potřeby tréninku neuronové sítě ale musíme použít jednu z voleb *File->Export as training data to PDP+ Environment* nebo *File->Export as training data to PDP+ with options*, budeme dotázáni na soubor s rysy patřícími k dané masce, tímto způsobem sdružíme vstupní data s odpovídajícími výstupy. Obě volby jsou téměř identické, pouze druhá z nich nabídne dialogové okno s možnostmi pro export masky (obr. 5.3):

Export all segments	- pokud uživatel chce exportovat pouze jistý počet náhodně vybraných segmentů, odškrtne tuto volbu
Number of segments to export	- počet segmentů pro export
Append to existing file	- při tvorbě tréninkových dat je často výhodné zkombinovat do jednoho souboru s daty několik obrazů. Touto volbou je možné přidávat data do již vytvořeného souboru s tréninkovými daty.



obr. 5.3

iii) Vizualizace výsledků segmentace

Výsledky segmentace obdržíme z PDP++ v souboru se záznamem o průběhu aktivního režimu sítě. Tyto výsledky je možné vizualizovat pomocí masky. Nabídkou *File->New* vytvoříme novou masku s odpovídajícím referenčním obrazem (v tomto případě obrazem, ze kterého jsme počítali rysy) i velikostí segmentu. Zobrazí se prázdná maska, do níž můžeme načíst data generované neuronovou sítí pomocí volby *File->Import segmentation results from PDP+ log file*.

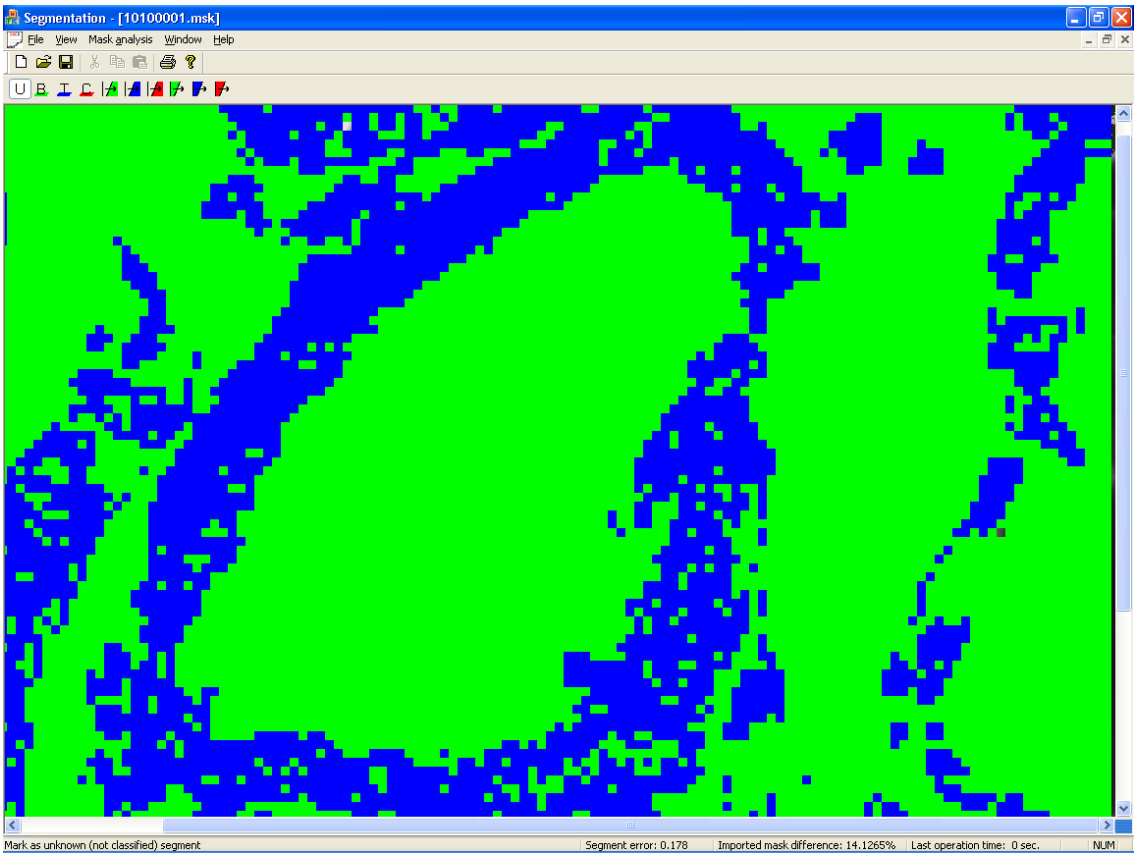
Soubor obsahuje stavy výstupních neuronů pro jednotlivé segmenty, každý neuron odpovídá jedné segmentační třídě. V ideálním případě odpovídá stav výstupního neuronu roven jedné přiřazení odpovídající segmentační třídy segmentu, jehož rysy byly předloženy vstupním neuronům. Ostatní výstupní neurony by měly mít svůj stav roven nule. Může se ale stát, že si síť s daným vstupem nebude vědět rady. To se může projevit např. tím, že ani jeden z výstupních neuronů nebude mít hodnotu blízkou jedné. Proto pokud program tuto situaci zjistí, označí výsledný segment jako neznámý (unknown), takové segmenty se v masce zobrazují jako průhledné (do této segmentační třídy také spadají všechny segmenty po vytvoření nové masky).

Ve stavovém řádku okna se nachází položka *Segment error*, jejíž hodnota odpovídá vzdálenosti hodnoty odpovídajícího výstupního neuronu od jedničky pro segment, nad kterým se nachází ukazatel myši.

Další položkou stavového řádku je *Imported mask difference*, tedy rozdíl mezi prázdnou maskou, kterou jsme vytvořili pro vizualizaci výsledků, a maskou vzniklou z výsledků výpočtu sítě. Tato veličina má smysl v případě, že místo prázdné masky si na začátku otevřeme např. vzorovou masku vytvořenou k danému obrazu člověkem, po načtení výsledků ze souboru uvidíme, v kolika procentech počtu segmentů se tyto dvě masky liší, a tedy jak se síť přiblížila optimálnímu výsledku.

Masku lze také exportovat jako obrazový soubor, což je vhodné zejména pro další práci s výsledky segmentace. Slouží k tomu volba *File->Export mask as image*. Podporované jsou zatím pouze formáty *Enhanced metafile* (.emf) a *Windows metafile* (.wmf), naštěstí převod do běžnějších formátů není problematický.

Ukázka vizualizace výsledků viz obr. 5.4.



obr. 5.4 – ukázka vizualizace výsledků segmentace

4. 1. 2 Klasifikace segmentovaného obrazu

Klasifikace již segmentovaného obrazu probíhá podobně jako samotná segmentace. Nejprve potřebujeme vypočítat rysy. Ty se tentokrát určují z celé masky (všechny jsou globální), takže odpadají volby velikosti mřížky či velikosti oblasti pro výpočet matice sousednosti intenzit. Rysy masky spočteme (při aktivním dokumentu s maskou, jejíž rysy chceme získat) pomocí nabídky *Mask analysis->Compute features for classification*.

i) Data pro aktivní režim

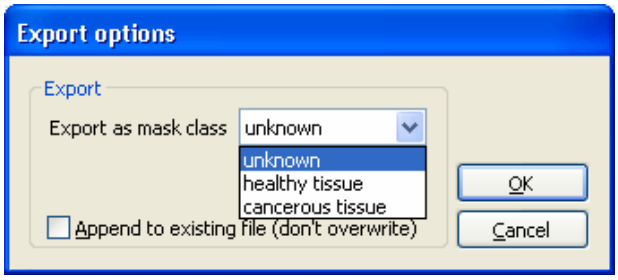
Spočtené rysy můžeme podobně jako rysy ještě nesegmentovaného obrazu převést do formátu pro aktivní režim neuronové sítě pomocí *File->Export to PDP+ Environment format*.

ii) Data pro adaptivní režim

Rysy ve formátu pro adaptivní režim získáme volbou *File->Export as training data to PDP+ format*, zobrazí se dialogové okno s možnostmi (obr. 5.5):

Export as mask class - klasifikační třída segmentovaného obrazu (masky), může nabývat hodnot zdravá tkáň (*healthy tissue*) nebo rakovinná tkáň (*cancerous tissue*). Hodnota neznámá (*unknown*) není přípustná pro učení sítě.

Append to existing file - Touto volbou je možné přidávat data do již vytvořeného souboru s tréninkovými daty.

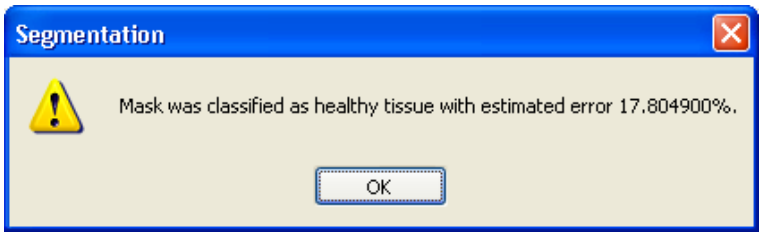


obr. 5.5

iii) Zobrazení výsledků klasifikace

Po volbě *File->Import classification results from PDP+ log file* se zobrazí dialogové okno s informací, do jaké třídy (neznámá, zdravá tkáň, rakovinná tkáň) byla maska podle svých rysů zařazena a s jakou eventuální chybou (počítá se stejně jako u segmentace, tj. vzdálenost aktivního výstupního neuronu od jedničky, pouze je uváděna v procentech). Viz obrázek 5.6.

Při načítání výsledků klasifikace se program zeptá, zda uživatel chce načtenou hodnotu nastavit pro aktivní dokument s rysy masky. Nastavená hodnota se zobrazuje ve stavovém řádku, při exportu dat je nabízena jako výchozí a ukládá se společně s dokumentem.



obr. 5.6 – příklad zobrazení výsledků klasifikace

4. 2 Popis programového balíku PDP++

Jedná se o balík programů pro návrh a testování modelů neuronových sítí, pro tuto práci jsem použil verzi 3.1. Autory projektu uvádím spolu s adresou serveru pro stažení balíku v [PDP 04]. Volně dostupné jsou také zdrojové texty v jazyce C++.

4. 2. 1 Instalace

Již přeložená verze PDP++ je dostupná pro několik platforem, konkrétně Linux, CygWin (pro MS Windows), Darwin (Mac OS-X), Sun, Sgi a další. Pro bližší informace o instalaci a kompletní seznam podporovaných platforem odkazuji na 2. kapitolu manuálu k PDP++ [Man 04]. Použil jsem verzi pro operační systém MS Windows, k jejíž instalaci stačí spustit soubor pdp++_3.1_setup.exe. Podrobnosti opět viz 2. kapitola [Man 04].

4. 2. 2 Možnosti práce s PDP++

Celý balík se dělí na několik spustitelných souborů, knihoven, zdrojových textů a demonstračních projektů. Spustitelné soubory se liší modelem neuronové sítě, který implementují:

‘*bp++.exe‘* - implementace modelu backpropagation

‘cs++.exe’ - implementace tzv. constraint satisfaction modelu

‘so++.exe’ - model učení pomocí samoorganizace

‘bpso++.exe’ - kombinace backpropagation a samoorganizačních modelů

a další.

Grafické uživatelské rozhraní (GUI) je pro všechny modely jednotné, jednotlivé implementace se liší „pouze“ tím, co umí, což je výhodné pro rychlejší osvojení si práce s programem. Jednotlivé implementace lze také spustit dávkově, pomocí skriptu napsaného v jazyce CSS [CSS 04], případně použít knihovny dodané s PDP++ přímo ve svém programu (viz závěrečná kapitola).

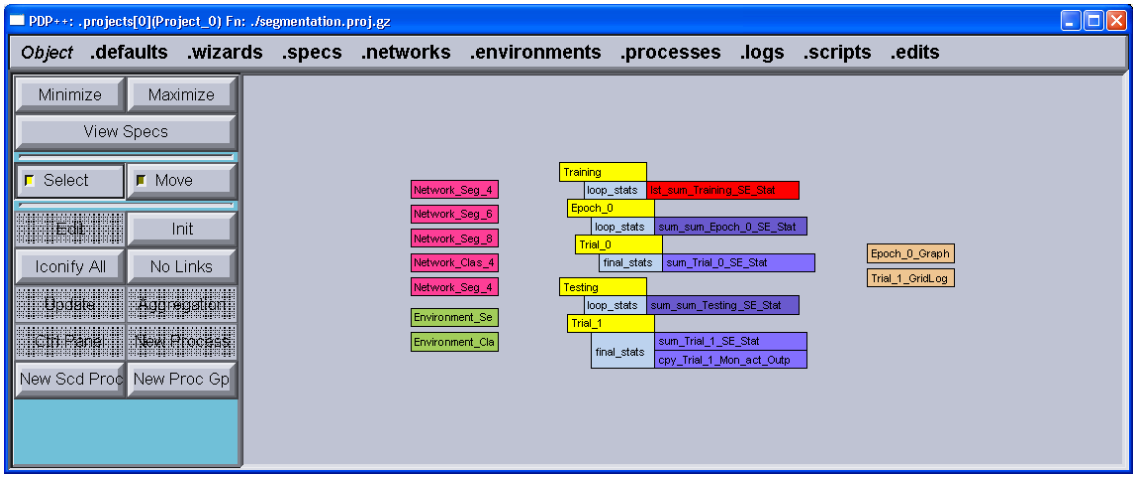
4. 2. 3 Koncepce práce s PDP++

Balík PDP++ je navržen a implementován podle principů objektově orientovaného programování. Tento přístup přináší své výhody zejména díky snadnému rozšiřování stávajících modelů. Ve většině případů stačí vytvořit potomka správné třídy a soustředit se pouze na odlišnosti přidávaného prvku.

Celý balík se snaží dosáhnout co největší variability práce s jednotlivými prvky, aby bylo možné kombinovat téměř „každý s každým“. Nespornou výhodou tohoto přístupu je právě ona

variabilita, na druhou stranu ji vyvažuje nutnost na začátku se zorientovat v tom, které prvky použít a jak.

Práce v PDP++ je organizována do projektů (viz obr. 5.7). Každý projekt může obsahovat neuronové sítě, prostředí, procesy a záznamy (logs). Každou z těchto částí lze uložit samostatně do zvláštního souboru, při práci s projektem jako celkem se vždy ukládají všechny jeho části do souboru projektu. Následuje základní popis jednotlivých částí projektu, detaily viz [Man 04]. Všechny uváděné příklady se týkají modelu dopředné sítě backpropagation.



obr. 5.7 – okno projektu v PDP++, obsahuje pět sítí, dvě prostředí, dva procesy (se svými podprocesy a statistikami chyby) a konečně dva objekty pořizující záznamy

i) Neuronová síť

Základem projektu je neuronová síť. Vytvoříme ji nejsnadněji v okně projektu pomocí *.networks->New,* zobrazí se editor sítě.

Nejprve potřebujeme vrstvy, což jsou jakési kontejnery na neurony. Zpracování probíhá postupně vrstvu po vrstvě, takže budeme potřebovat např. jednu vstupní vrstvu, jednu skrytou a jednu výstupní. V levé části editoru zvolíme tlačítko *New Layer(s)* a jako počet vytvářených vrstev zadáme tři. Nyní zvolíme v levé horní části režim *ReShape*, pomocí myši rozšíříme vygenerované vrstvy, aby v nich bylo postupně od vstupní k výstupní místo na 8, 4 a 3 neurony. Poté pomocí tlačítka *Build All* necháme vygenerovat odpovídající neurony.

Nyní potřebujeme jednotlivé neurony spojit, nejsnadněji pomocí tzv. projekcí. Projekce je něco podobného pro spojení jako vrstva pro neurony. Jedna projekce funguje vždy mezi dvěma vrstvami a v rámci projekce jsou teprve definovaná jednotlivá spojení mezi neurony. Pro tvorbu projekcí se přepneme do výběrového režimu tlačítkem *Select*. Nyní označíme výstupní vrstvu a skrytou vrstvu, např. pomocí myši a stisknutého přepínače *Shift*. Je nutno zachovat pořadí označení, projekce se vždy vytváří od cílové vrstvy ke zdrojové (ze které vycházejí spojení). Mezi výstupní a skrytou vrstvou by se měla vytvořit prázdná šipka. Podobným způsobem vytvoříme projekci mezi skrytou vrstvou a vstupní vrstvou. Pro naplnění projekcí spojeními každého neuronu s každým (v rámci vrstev projekce) stiskneme tlačítko *Connect All*, prázdné šipky by se měly vyplnit. Nyní máme hotovou jednoduchou neuronovou síť se třemi vrstvami.

V levém horním okraji okna se nachází přepínač *Display*, určuje, zda se má zobrazovat vybraná charakteristika jednotlivých neuronů. Tuto charakteristiku lze zvolit v levé spodní části okna, přičemž z nejpoužívanějších se jedná o aktivaci (stav) neuronu – proměnná *act*, váhy spojení přicházejících do neuronu – *r.wt* a váhy spojení vycházejících z neuronu – *s.wt*. Hodnoty se zobrazují také při aktivním režimu *View* (tlačitko vlevo nahoře) a jsou znázorněny vybarvením příslušných neuronů korespondující barvou. Škála barev se zobrazuje v pravé části okna, tamtéž lze nastavit i zobrazování číselných hodnot. Při výpočtu sítě doporučuji přepínač *Display* vypnout, protože neustálé zobrazování stavu všech neuronů výpočet znatelně zpomaluje.

Ve výběrovém režimu (*Select*) lze nastavit vlastnosti jednotlivých projekcí či vrstev. Jednoduše myší označíme požadovaný objekt a v levé části okna stiskneme tlačítko *Edit Layers(s)*, příp. *Edit Prjn(s)* – pro projekce. Nejdůležitějšími atributy vrstev či projekcí jsou specifikace (*specs*). Specifikace určují chování daného objektu, např. aktivační funkci neuronu, způsob propojení neuronů v rámci projekce nebo parametry učení spoje dvou neuronů. Specifikace lze upravovat samostatně v okně projektu pod nabídkou *.specs->Edit*.

Vlastnosti neuronu určuje tzv. *UnitSpec*, v případě backpropagation *BpUnitSpec*, ovlivňuje zejména:

act range - rozsah hodnot aktivační funkce

bias con type - typ biasu

bias spec - specifikace typu spojení s biasem

sig - parametry aktivační funkce sigmoidy (*offset* a *gain*)

err fun - typ chybové funkce

Pomocí tlačítka *Graph Act Fun* lze pohodlně zobrazit graf aktivační funkce.

Specifikace jednotlivých spojení mezi neurony je schována pod tzv. *ConSpec*, v našem případě *BpConSpec*:

md - typ inicializace vah – rozsah, pravděpodobnostní rozložení

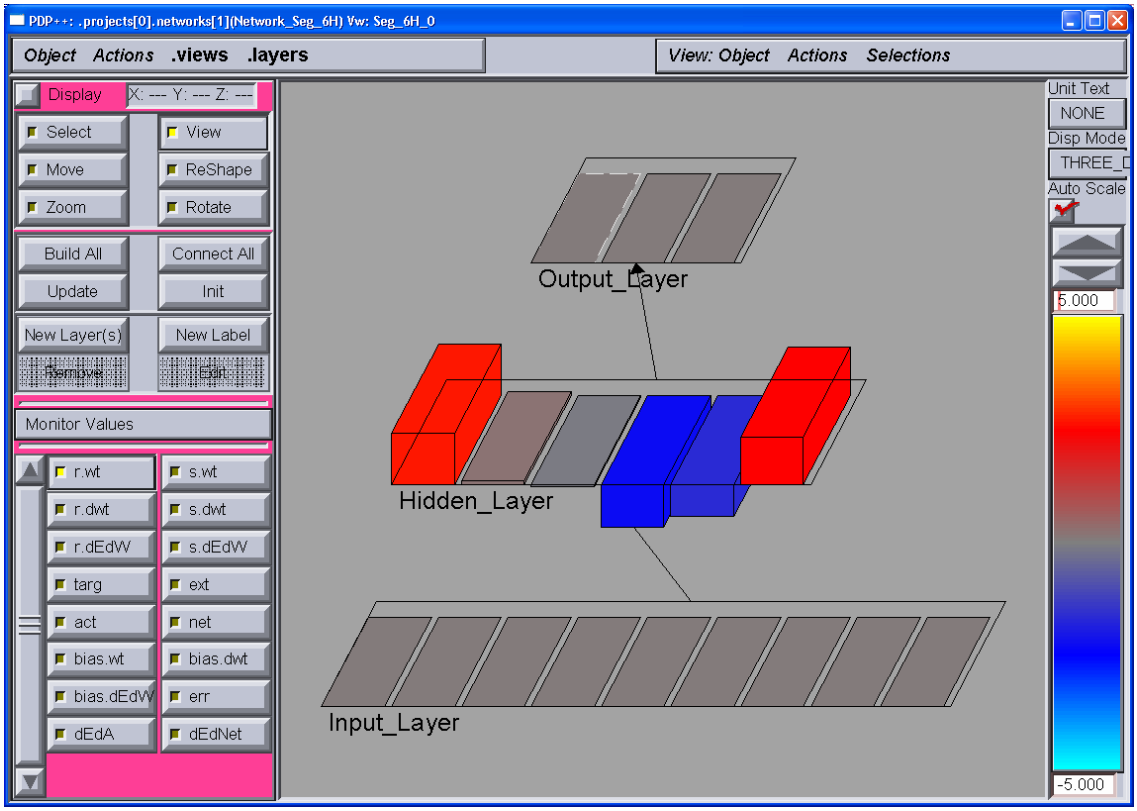
wt limits - určuje, zda mají být hodnoty vah omezeny, příp. stanoví jejich rozsah

lrate - tzv. *learning rate*, rychlost učení (tato a následující vlastnosti viz 3. kapitola)

momentum - určuje spolu s *momentum type* tzv. moment učení sítě

decay - spolu s *decay fun* určuje hodnotu (většinou poměrnou část váhy), o kterou se zmenší chyba při zpětném šíření chyby v adaptivním režimu

Nejčastější specifikací typu projekce je *FullPrjnSpec* definující spojení neuronů každý s každým.



obr. 5.8 – prohlídka vah prvního neuronu výstupní vrstvy – zvolen režim *View*, zobrazovaná hodnota *r.wt* a vybrán první neuron výstupní vrstvy

ii) Prostředí

Prostředí (Environment) odpovídá jakémusi kontejneru na tréninková i testovací data. Autoři tuto část nazývají prostředím, aby se více přiblížili představě, že neuronová síť reprezentuje organismus žijící v určitém prostředí. Prostředí obsahuje události (events) reprezentující jednotlivá data, skládající se z hodnot pro vstupy i výstupy sítě. Vstupy a výstupy sítě se v terminologii PDP++ nazývají vzory (patterns). Prostředí samo o sobě nedefinuje pořadí, v jakém budou jednotlivé události předkládány neuronové síti (jak pro trénink, tak pro aktivní režim), toto je záležitost odpovídajícího procesu (viz dále).

Po vytvoření nového prostředí pomocí *.environments->New* můžeme buď jednotlivé události zadat ručně, nebo je načíst z textového souboru pomocí *Object->Read Text*. Tuto možnost jsem použil pro načítání dat připravených programem Segmentation. Při použití více sítí s různými počty vstupních nebo výstupních neuronů je zapotřebí pro správné načtení dat definovat tzv. specifikaci události (event spec). Toho můžeme docílit pomocí tlačítka *Edit Specs* v okně prostředí a pomocí nabídky *.event_specs*. Zpět k úpravě událostí se vrátíme pomocí *Edit Events*, případně můžeme použít nabídky *.events*, např. pro odstranění všech událostí z prostředí (pomocí *.events->Remove->All*). Samotné přiřazení hodnot jednotlivým událostem (pokud je nenačítáme ze souboru) probíhá po vytvoření události (*.events->New*) pomocí barevné reprezentace hodnot, podobně se znázorňují např. hodnoty aktivace jednotlivých neuronů. Ve spodní levé části okna prostředí vybereme událost, jejíž hodnoty chceme upravit. Jednotlivé vzory vybrané události se zobrazí v centrální části okna jako obdélníky, každému z nich

přiřadíme barvu odpovídající námi požadované hodnotě. Přesné hodnoty pro jednotlivé barvy lze samozřejmě nastavit, konkrétně ve spodní části okna prostředí.

Při tvorbě nového prostředí máme na výběr z několika typů, spíše pro zajímavost uvádím např. typ *FromFileEnv*, který umožňuje postupné vybírání jednotlivých událostí ze souboru na disku.

iii) Proces

Mezi procesy patří jakékoli aktivity prováděné neuronovou sítí v daném prostředí. Proces řídí např. i to, jestli se budou váhy sítě během běhu procesu upravovat, tj. zda se jedná o učení nebo o aktivní režim. Výsledky a průběh procesu mohou být uchovány a posléze zpracovány pomocí záznamů (logs), kterým se budu věnovat v následující části.

Procesy vytvoříme pomocí volby *.processes->New*, kdy dostaneme na výběr z několika typů procesu. V PDP++ procesy tvoří následující hierarchii:

Training – učení neuronové sítě, cyklí přes epochy

Epoch – „epocha“ vývoje, jedná se o prezentaci všech[3] událostí prostředí

Trial – předložení jednoho vzoru (vstupních a výstupních hodnot)

Díky této hierarchii je možné nastavovat zvlášť parametry každé úrovni procesu v hierarchii. Nejdříve vytvoříme **proces učení** neuronové sítě (adaptivní režim).

V nabídce *.processes->New* vybereme typ *TrainProcess* a v následujícím dialogovém okně ponecháme zaškrtnutou volbu vytvořit podprocesy (*CreateSubProcs*). V okně projektu se zobrazí vytvořený proces spolu s hierarchií svých podprocesů. Kliknutím pravým tlačítkem myši na příslušný podproces můžeme nastavovat jeho vlastnosti, společné vlastnosti se propagují od nejvyšší úrovně po nejnižší. Podívejme se na nejdůležitější vlastnosti jednotlivých procesů:

Trénink (training)

network - zde vybereme síť, která se má učit tímto procesem

environment - prostředí, jehož události budou předkládány síti

epoch - max - maximální počet epoch tréninku, pokud nebylo dosaženo stop kritérium (viz dále)

step - určuje jeden krok tréninku, tj. epochu, jako jeden krok můžeme zadat i více epoch

[3] ne nutně všech událostí, pokud procházíme prostředí metodou *random*, některé události nemusejí být předloženy vůbec, jiné mohou být síti prezentovány vícekrát (viz dále)

Epocha (epoch)

Dialogové okno obsahuje již výše uvedené možnosti (pouze *step* se nyní týká procesu *Trial*) a navíc tyto možnosti:

order - způsob, v jakém pořadí budou síti předkládány jednotlivé události, lze si vybrat z následujících možností:

- *sequential* - události budou předkládány postupně v takovém pořadí, v jakém se nacházejí v prostředí
- *permuted* - události budou prezentovány každou epochu v jiném pořadí, ale vždy každá událost právě jednou
- *random* - události budou předkládány v náhodném pořadí, přičemž každá může být předložena vícekrát nebo vůbec

wt update - určuje způsob aktualizace vah sítě, opět si můžeme vybrat:

- *test* - váhy se nemění vůbec, jedná se vlastně o aktivní režim sítě, proces testování sítě popíši později
- *on line* - váhy se adaptují po každé události
- *small batch* - adaptace vah po předložení *batch n* událostí
- *batch* - adaptace vah po průběhu epochy

Okno s vlastnostmi procesu *Trial* nepřináší žádné nové důležité vlastnosti.

Vpravo od každého procesu v hierarchii se nacházejí statistiky použité pro výpočet chyby sítě v příslušném procesu, standardně se chyba procesu na vyšší úrovni počítá jako součet hodnot chyby procesu o úroveň níž. Na úrovni procesu *Trial* se chyba určuje jako rozdíl hodnoty sítě od správného výsledku (daném v prostředí) umocněný na druhou. Ve vlastnostech chyby na úrovni tréninkového procesu (té nejvyšší) můžeme nastavit tzv. *stop kritérium*, které, pokud je splněno, znamená konec tréninku i před dosažením maximálního počtu epoch. *Stop kritérium* nastavíme zaškrtnutím volby *stopcrit* u položky *se* (sum error). Poté stiskneme tlačítko *Apply*, zobrazí se pole pro vložení hodnoty spolu s polem *cnt* (count), jenž značí nutný počet splnění stop kritéria pro konec tréninku (většinou jedna).

Nyní vytvoříme **proces testování** sítě (aktivní režim) pomocí *.processes->New*, kde vybereme typ *EpochProcess*. Opět necháme vygenerovat příslušné podprocesy zaškrtnutím volby *CreateSubProcs* a ve vlastnostech epochy nového procesu nastavíme režim aktualizace vah na *test* a pořadí na *sequential* – časté charakteristiky aktivního režimu sítě.

Vytvořené procesy spustíme volbou *.processes->Control Panel* a vybráním požadovaného procesu v nabídce. Zobrazí se ovládací panel procesu (viz obr. 5.9), většina výchozích hodnot je již nastavena podle vlastností procesu. Následuje popis základních funkcí ovládacího panelu.

network - síť, pro kterou bude proces spuštěn

environment - prostředí s událostmi předkládanými síti

step - označuje krok procesu, díky této volbě můžeme předdefinovat např. více typů epoch a při tréninku sítě mezi nimi pohodlně vybírat až při spouštění

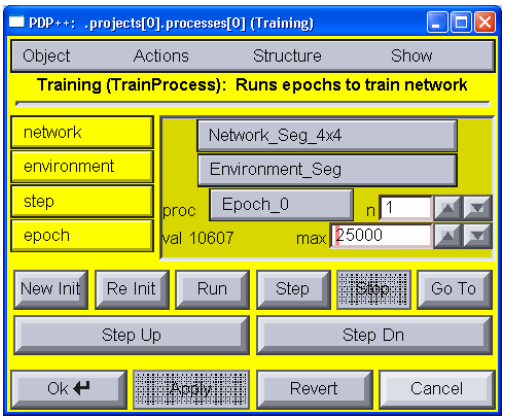
epoch/trial - číslo právě probíhající epochy pro trénink, pro aktivní režim sítě zobrazuje číslo předkládané události

New Init - nastaví čítač aktuální epochy/předkládané události na nulu a vygeneruje novou inicializační hodnotu pro generátor náhodných čísel (tzv. random seed), např. pro náhodný výběr vzorů během epochy; pokud se jedná o tréninkový proces, nastaví počáteční hodnoty vah sítě

Re Init - to, co předchozí volba, pouze použije původní inicializační hodnotu pro generátor náhodných čísel, pokud žádná dosud není vytvořena, vygeneruje ji

Run/Stop - spustí/zastaví proces

Step - spustí proces a zastaví jej po *step n* krocích



obr. 5.9 – ovládací panel tréninkového procesu

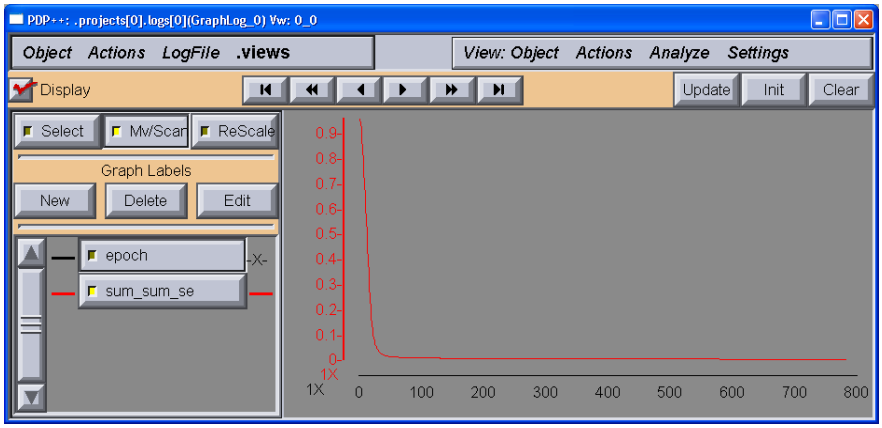
iv) Záznam

Záznamy podávají informace o průběhu procesů, lze je ukládat v textové podobě do odděleného souboru na disku, čehož jsem využil jako možnosti předávání výsledků výpočtů sítě programu Segmentation. Základními dvěma typy záznamů v PDP++ jsou grafický záznam (*GraphLog*) a tabulkový záznam (*GridLog*). Grafický záznam lze s výhodou použít pro sledování průběhu učení sítě, zatímco hodnoty v tabulce se hodí pro zpracování výsledků výpočtu sítě v aktivním režimu programem Segmentation. Podívejme se nejprve na grafický záznam.

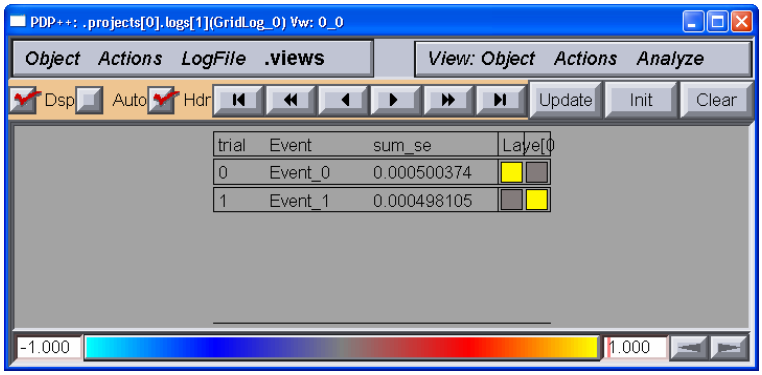
Nový objekt pořizující grafický záznam činnosti procesů vytvoříme (v okně projektu) pomocí nabídky *.logs->New*, jako typ objektu zvolíme *GraphLog* a ponecháme zaškrtnutou volbu *AddUpdater*. Díky ní se ihned po vytvoření objekt sám zeptá, ze kterého procesu má číst data. Pro záznam o průběhu učení sítě zvolíme proces epochy (např. *Epoch_0*) patřící procesu *Training*. Pokud nyní spustíme tréninkový proces, můžeme sledovat průběh chyby sítě v jednotlivých epochách tréninku v grafu (obr. 5.10). Kromě chyby sítě lze nechat grafický záznam zobrazovat i jiné veličiny, v případě zájmu viz [Man 04].

Tabulkový záznam vytvoříme v okně projektu nabídkou *.logs->New* a zvolením typu *GridLog*. Opět ponecháme zaškrtnutou volbu *AddUpdater*, jako zdroj dat zvolíme proces *Trial* v procesu testování sítě (např. *Trial_1*). V této chvíli bude tabulkový záznam zobrazovat ke každé předkládané události pouze chybu sítě, pro zpracování výsledků bychom potřebovali spíše hodnoty aktivační funkce neuronů ve výstupní vrstvě. V okně editoru sítě zvolíme v levé části tlačítkem *Act* zobrazování hodnot aktivace a myší vybereme výstupní vrstvu. Poté pomocí tlačítka *Monitor Values* a volby *New* přidáme hodnoty aktivace vybraných neuronů ke sledování procesu *Trial* patřícího testovacímu procesu (*Trial_1*). V okně tabulkového záznamu zvolíme *Actions->Get Headers*, což aktualizuje hlavičku tabulky, a po spuštění testovacího procesu uvidíme výsledky výpočtu aktivního režimu sítě v tabulce (obr. 5.11). Při zaznamenávání dlouhých výpočtů do souboru doporučuji vypnout zobrazování tabulky na obrazovku, dosáhneme tak znatelného zrychlení.

Každý záznamový objekt nabízí kromě zobrazení příslušných dat také možnost ukládat průběžně přicházející hodnoty do textového souboru. Stačí v okně záznamu zvolit nabídku *LogFile->Set Save File* a zadat jméno souboru. Údaje lze také přidávat na konec již existujícího souboru, např. pokud byl trénink z nějakého důvodu přerušen, a to volbou *LogFile->Set Append File*. Pokud přestaneme pořizovat záznam, ale PDP++ ještě nekončíme, je vhodné soubor uzavřít, opět v nabídce *LogFile*.



obr. 5.10 – ukázka záznamu grafu průběhu chyby sítě při učení, výpočet skončil až v 784. epoše díky poměrně silnému stop kritériu



obr. 5.11 – ukázka tabulkového záznamu výpočtu sítě v aktivním režimu v prostředí se dvěma událostmi

5. Konkrétní řešení a výsledky

K řešení problému segmentace obrazu jsem v PDP++ vytvořil čtyři vícevrstvé dopředné sítě typu backpropagation s pevnými topologiemi 9-4-3, 9-6-3, 9-8-3 a 9-4-4-3. Klasifikaci řešily dvě sítě lišící se od segmentačních pouze v topologiích – 6-4-2, 6-10-10-2. Specifikace spojení obsahovala parametry rychlost učení (*learning rate*) 0,0001 a moment 0,5. Proces učení aktualizoval váhy způsobem *batch*, tedy na konci každé epochy tréninku. Jako aktivační funkce neuronů byla použita standardní sigmoida se strmostí (λ) rovnou jedné, neurony propojeny každý s každým. Podobné hodnoty byly s úspěchem použity v [Pelikan et al. 99] pro podobný problém, proto jsem z nich vycházel.

Programem Segmentation jsem vytvořil dva soubory tréninkových dat z části výřezů vstupních obrazů, každý o velikosti 23804 záznamů s mřížkou (velikostí segmentu) 9x9 bodů. Záznamy se lišily parametry použitými při výpočtu rysů, konkrétně velikostí okolí, ze kterého se počítala matice sousednosti intenzit, a velikostí okolí pro výpočet mediánu (v obou případech se jednalo o okolí 9x9 nebo 11x11 bodů). Jednotlivé záznamy byly postupně předkládány síti, každou epochu tréninku v jiném pořadí (metodou *permuted*), aby se znemožnilo síti zapamatovat si polohu segmentu v obraze, což by mohlo vést ke špatné generalizaci. Vývoj chyby vybraných sítí při tréninku zachycují grafy v příloze II, konkrétní hodnoty chyby po 10000. iteraci (epoše) tréninku ukazuje tabulka 5.1. Pro úplnost dodávám, že chyba sítě je určena jako součet chyb jednotlivých výstupních neuronů, jejichž chybu určuje druhá mocnina rozdílu výstupu neuronu a požadované hodnoty (tzv. *sum of squares error*) – viz kapitola 3. 4. Trénink jedné neuronové sítě je časově dosti náročný, možnostem jeho zrychlení se budu věnovat v závěrečné kapitole. Na počítači s procesorem Intel Celeron 433 Mhz, 384 MB RAM trval výpočet 10000 iterací tréninku přibližně devět hodin.

Chyba sítě po 10000. iteraci tréninku

topologie sítě velikost okolí	9-4-3	9-6-3	9-8-3	9-4-4-3
9x9	5162,82	5091,70	5115,99	5073,14
11x11	4902,35	4578,99	4567,54	4698,54

tabulka 5.1 – při celkovém počtu 23804 vzorů ukazuje pohyb chyby okolo 20%

K otestování sítě jsem vytvořil dva soubory testovacích dat (obsahující každý 18489 vzorů) analogicky jako pro trénink, lišily se pouze v použití jiných výřezů původních obrazů. Naměřené hodnoty ukazuje tabulka 5.2.

Chyba sítě na testovacích datech

topologie sítě velikost okolí	9-4-3	9-6-3	9-8-3	9-4-4-3
9x9	3997,28	3710,02	3583,81	3580,72
11x11	3385,97	3505,41	3539,70	3589,02

tabulka 5.2 – chyba při celkovém počtu vzorů 18489

Výsledky vypovídají o nižší chybě pro rysy založené na mediánu a matici sousednosti intenzit spočtených z okolí 11x11 bodů, zatímco topologie sítě má na chybu poměrně malý vliv – ve většině případů chyba s větším počtem neuronů nepatrně klesne. To se dalo očekávat, neboť větší počet neuronů dává síti větší kapacitu pro zapamatování si vztahů mezi vstupy – to však nevypovídá o míře generalizace dosažené při učení. Zajímavé je porovnání výsledků sítě 9-4-4-3 oproti síti 9-8-3, jenž se liší uspořádáním neuronů do další skryté vrstvy.

Pro klasifikaci segmentovaných obrazů jsem vytvořil soubor testovacích a tréninkových dat, tentokrát se rysy počítaly vždy z celého obrazového výřezu, takže odpadlo rozlišení typu použitých rysů. V praxi se bohužel ukázalo, že tréninková data založená na výřezech původních obrazů nejsou pro síť vhodná, sítě se na těchto datech nepodařilo naučit. Vhodnějšími daty by mohly být celé vstupní obrazy, protože při klasifikaci obrazu střeva soustředíme pozornost na části zachycující pokud možno co největší úsek tkáně krypty, už nestačí „jakékoli“ (i dostatečně velké) výřezy. Vstupních obrazů je ale pro vytvoření tréninkových i testovacích dat málo.

6. Závěr a doporučení

Nastínili jsme problematiku segmentace a klasifikace obrazu, představili výhody a nevýhody přístupu používajícího intenzity jednotlivých bodů jako vstupů neuronové sítě a přístupu využívajícího tzv. rysy. Vybrali jsme některé modely neuronových sítí zajímavé z hlediska zpracování obrazu, důkladněji popsali model *backpropagation* použitý při řešení problému. Bylo navrženo konkrétní řešení a uvedeny experimentální výsledky.

Na doprovodném CD je umístěn program Segmentation, který lze ve spojení s PDP++ použít k řešení problému, případně jej dále rozvíjet, např. doprogramovat nové rysy. Tyto dva programy lze využít jako výchozí bod pro další experimenty či vývoj v této oblasti.

Experimentální výsledky odhalují některé nedostatky v návrhu řešení, kterým bych se chtěl závěrem věnovat. Pokud by tato práce někoho inspirovala a chtěl na ni navázat, odstraněním dále uvedených nedostatků zcela jistě dojde k příznivějším výsledkům.

Grafy průběhu učení sítí (viz příloha) vykazují, že chyba po jisté době tréninku již neklesá[4]. Teoreticky se může jednat o lokální minimum chybové funkce, což lze řešit jinou inicializací vah, ale v tomto případě to nepomáhalo. Proto docházím k závěru, že zvolené rysy nedokáží rozlišit některé segmenty patřící do různých segmentačních tříd a *chybu sítě* by bylo možné dále *snížit* doplněním rysů o nějakou vhodnou charakteristiku.

S výběrem rysů souvisí i fakt, že některé segmenty nelze jednoznačně zařadit do segmentační třídy – pomohla by konzultace s odborníkem a vytýčení jednoznačného kritéria pro klasifikaci segmentu. Pokud se takové kritérium nepodaří nalézt, mohlo by pomoci vyloučení sporných segmentů z tréninkové množiny, což lze provést např. jejich označením jako neznámé.

Učení sítě můžeme *zrychlit* zmenšením tréninkové množiny. Můžeme sice náhodně vybrat její podmnožinu (např. rovnou v programu Segmentation), ale lepší metodou by mohlo být vyřazení příliš podobných vzorů podle nějakého definovaného kritéria.

PDP++ lze použít i v podobě knihovny připojené k uživatelskému programu. Dalším vylepšením je tedy *integrace neuronové sítě* přímo *do programu Segmentation*, což by bylo jak uživatelsky pohodlnější, tak i výrazně rychlejší. Na druhou stranu, pro experimentální použití může být výhodnější prostředí PDP++ nabízející vysokou míru flexibility pro hledání optimální neuronové sítě.

Výsledek segmentace by bylo možné ještě *dodatečně zpracovat* a odstranit některé chyby. Nabízí se např. možnost odstranit „díry“ o velikosti jednoho či dvou segmentů v tkáni, v dostupných obrazech se tak malá „díra“ v tkáni nevyskytovala.

Rychlost výpočtu rysů můžeme zvýšit optimalizací výpočtu matice sousednosti intenzit, která je symetrická, protože se počítá pro všechny směry, zatímco stávající algoritmus byl navržen pro výpočet obecné matice sousednosti.

[4] případy, kdy se chyba pohybuje i v 10000. iteraci (tj. na konci grafu), jsem učil i dále a chyba se v drtivé většině případů později ustálila.

Z experimentálních výsledků vychází výhodnější použít rysy založené na GLCM a mediánu spočtených z okolí 11x11 bodů než z okolí 9x9. Při volbě většího okolí bohužel znatelně (kvadraticky) klesá rychlost výpočtu lokálních rysů, nalezení nějakého rychlého algoritmu by umožnilo rozumně pracovat i s náročnějšími rysy.

Tato práce může fungovat jako inspirace pro člověka hledajícího stručný úvod do problematiky využití neuronových sítí při zpracování obrazu, v případě praktických experimentů i jako upozornění na možná úskalí. Zájemce o další informace odkazuji na seznam použité literatury.

Reference

[Murphy et al. 03] R.F. Murphy, M. Velliste, G. Porreca – Robust numerical features for description and classification of subcellular location patterns in fluorescence microscope images, VLSI Sig Proc, vol. 35, 2003

[Bishop 95] Christopher M. Bishop – Neural Networks for Pattern Recognition, Oxford University Press, 1995

[Scott et al. 98] M.J.J. Scott, M. Niranjan, R.W. Prager – Feature subset selection in variable cost domains, CUED/F-INFENG/TR. 323, červen 1998

[Sonka et al. 95] M. Sonka, V. Hlavac, R. Boyle - Image Processing, Analysis and Machine Vision, Chapman & Hall, 1995, ISBN 0412455706

[Haralick et al. 73] R.M. Haralick, K. Shanmugam, I. Dinstein – Textural Features for Image Classification, IEEE Trans. on Systems, Man and Cybernetics, Vol. SMC-3, No. 6, listopad 1973, pp. 610-621

[Pelikan et al. 99] E. Pelikan, M. Egmont-Petersen – Detection of bone tumours in radiographs using neural networks, Pattern Analysis and Applications, Vol. 2, No. 2, pp. 172-183, 1999

[Petersen et al. 02] M. Egmont-Petersen, D. de Ridder, H. Handels – Image processing with neural networks – a review, Pattern Recognition 35, 2002

[Šíma a Neruda 96] R. Neruda, J. Šíma – Teoretické otázky neuronových sítí, Praha: Matfyzpress, 1996, 390 s.

[PDP 04] Charles K. Dawson, Randall C. O'Reilly, James L. McClelland - PDP++ software verze 3.1, umístění (URL ověřeno v lednu 2004):

URL: ftp://cnbc.cmu.edu/pub/pdp++,
ftp://grey.colorado.edu/pub/oreilly/pdp++,
ftp://unix.hensa.ac.uk/mirrors/pdp++

CD: /PDP++

[Man 04] Manuál programového balíku PDP++, umístění:

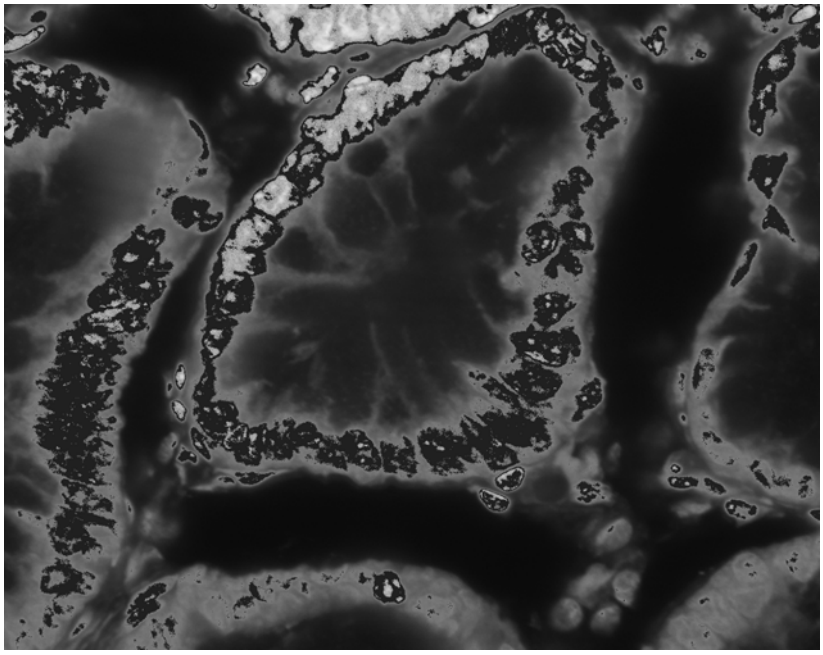
CD: /PDP++/manual/pdp-user.pdf,

po instalaci PDP++ v manual/pdp-user.pdf.

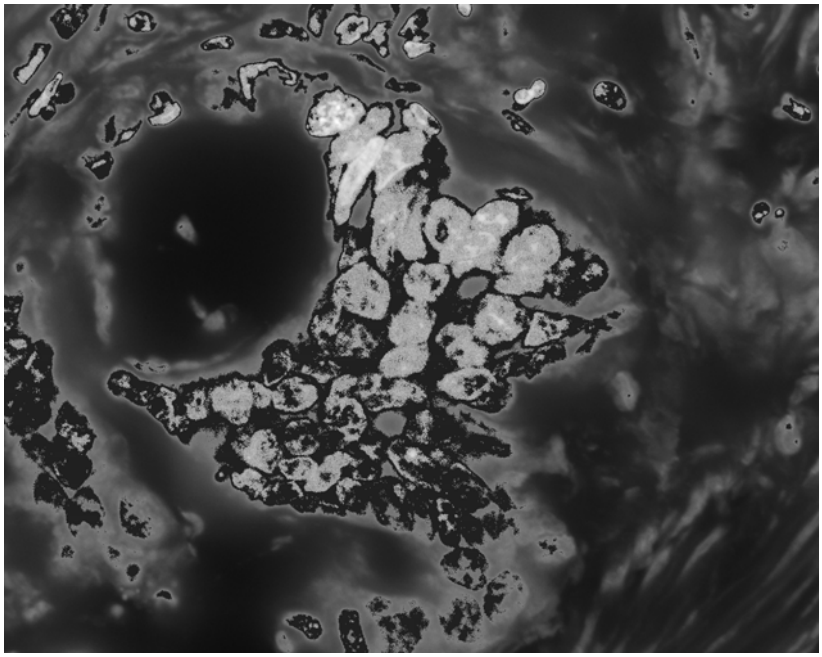
[CSS 04] TypeAccess/CSS manuál ke skriptovacímu jazyku použitému pro PDP++. Je umístěn na doplňkovém CD v /PDP++/manual/ta_css.pdf, po instalaci PDP v manual/ta_css.pdf.

Přílohy

I. Ukázka vstupních dat



10100001.tga – obraz zdravého střeva

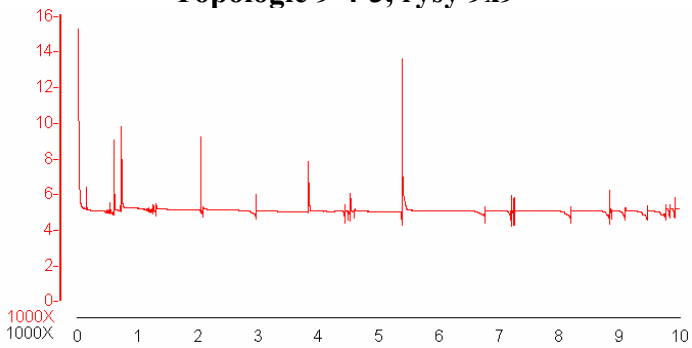


16100001.tga – obraz nádorového střeva

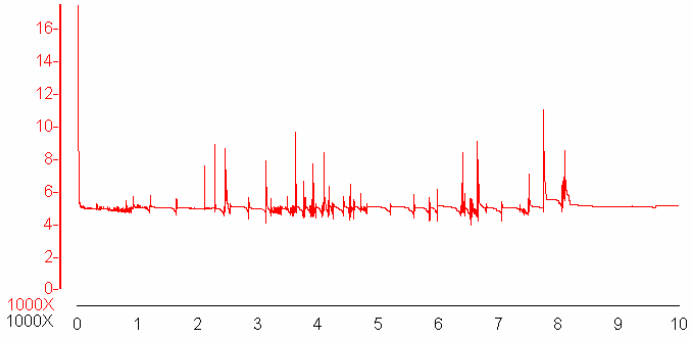
II. Grafy průběhů učení neuronových sítí

Všechny grafy znázorňují průběh chyby sítě pro 10000 epoch učení. Vodorovná osa reprezentuje epochu tréninku, svislá osa hodnotu chyby (tzv. *sum of squares error*). Tréninková množina obsahovala 23804 vzorů. Nadpis grafu znamená topologii sítě a velikost okolí (9x9 nebo 11x11 bodů) pro výpočet rysů, ze kterých byla složena vstupní data.

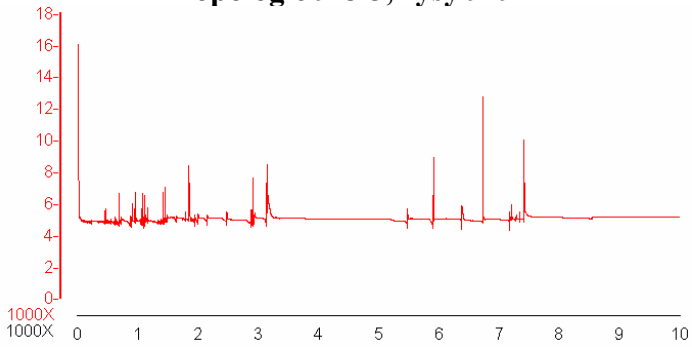
Topologie 9-4-3, rysy 9x9



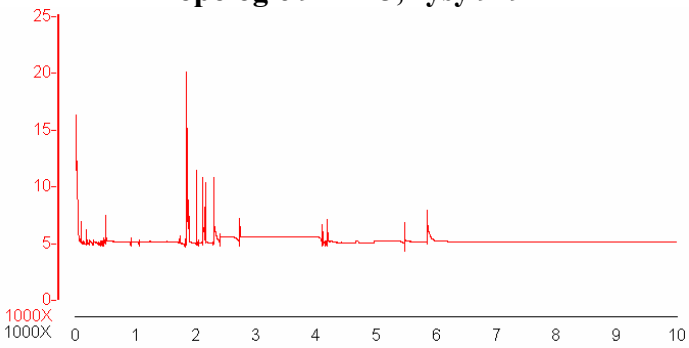
Topologie 9-6-3, rysy 9x9



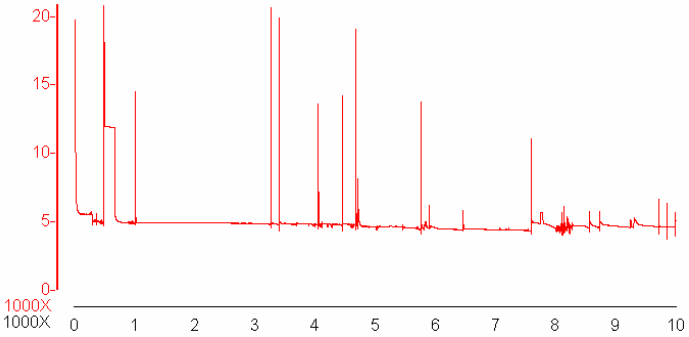
Topologie 9-8-3, rysy 9x9



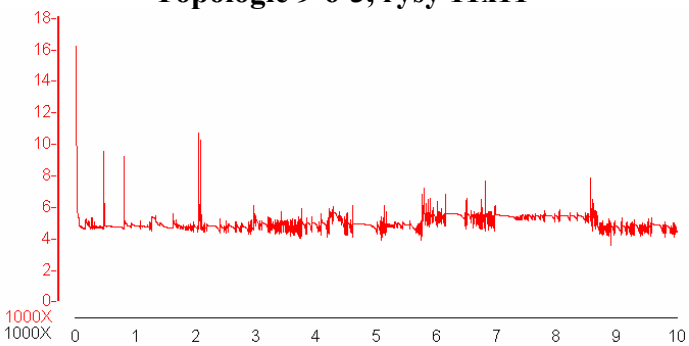
Topologie 9-4-4-3, rysy 9x9



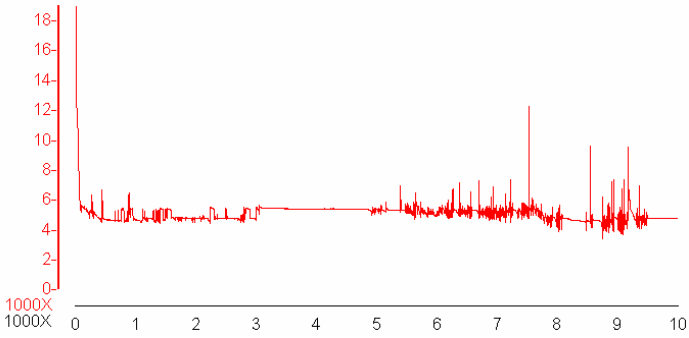
Topologie 9-4-3, rysy 11x11



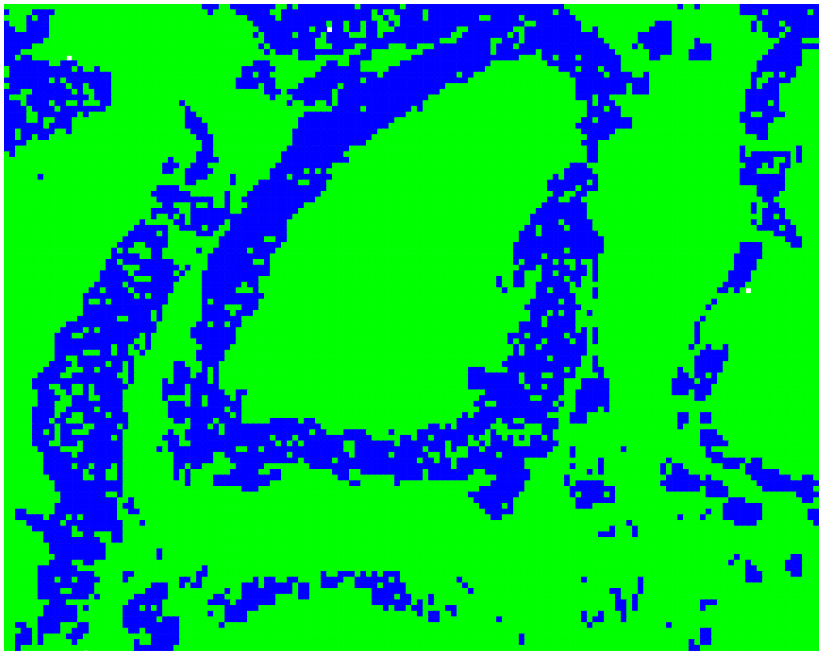
Topologie 9-6-3, rysy 11x11



Topologie 9-4-4-3, rysy 11x11



III. Ukázka segmentovaného obrazu



výsledek segmentace obrazu 10100001.tga

IV. Licence k použití programu PDP++

This file is part of the PDP++ software package.