ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE
FAKULTA PROVOZNĚ EKONOMICKÁ
KATEDRA INFORMAČNÍHO INŽENÝRSTVÍ

Použití neuronových sítí pro predikci složitosti softwarových produktů

DIPLOMOVÁ PRÁCE

Praha 2009 ©

Vedoucí práce: Ing. Josef Pavlíček, Ph.D.

Autor práce: Jiří Pergl

Poděkování

Nejprve bych rád poděkoval svému vedoucímu diplomové práce Ing. Josefu Pavlíčkovi, Ph.D. za jeho cenné připomínky, odborné vedení, trpělivost a ochotu při vypracování mé diplomové práce. Rovněž bych chtěl poděkovat své rodině a přátelům za rady, trpělivost a za poskytování mnohých užitečných informací.

Prohlášení:

Prohlašuji, že jsem diplomovou práci na téma Použití neuronových sítí pro predikci složitosti softwarových produktů vypracoval samostatně pouze za odborného vedení vedoucího práce.

Dále prohlašuji, že veškeré podklady, ze kterých jsem čerpal, jsou uvedeny v seznamu použité literatury.

V Praze dne 24.3. 2009 Podpis: ……

POUŽÍTÍ NEURONOVÝCH SÍTÍ PRO PREDIKCI SLOŽITOSTI SOFTWAROVÝCH PRODUKTŮ

USING NEURONAL NETWORKS FOR SOFTWARE COMPLEXITY PREDICTION

Souhrn

Tato diplomová práce se zabývá popisem oboru umělé inteligence, její strukturou, historickým vývojem a jejími nejvýznamnějšími osobnostmi. Velká pozornost je věnována neuronovým sítím, kde autor popisuje matematický neuron a jeho zapojení do neuronové sítě. Neuronové sítě jsou v diplomové práci, rozděleny podle druhů a jsou u nich popsány nejvýznamnější charakteristiky a nejčastější možnosti využití.

Druhá část diplomové práce se zabývá návrhem, realizací a popisem softwarového produktu PETRA.

Klíčová slova

Umělá inteligence, neuron, neuronová síť, perceptron, učení s učitelem, algoritmus zpětného šíření chyby, grafické uživatelské rozhraní

Summary

The thesis deals with a description of a field of an artificial intelligence, its structure, historical development and its most important personalities. Great attention is given to neural networks, where the author describes the mathematical neuron and its integration into neural networks. Neural networks are in the thesis divided by species and they describe the most important characteristics and the most common posibility of an usage. The second part of the thesis describes the design, implementation and description of a PETRA software product.

Keywords

Artificial Intelligence, neuron, neural network, perceptron, supervised learning, the algorithm backpropagation of error, graphical user interface.

OBSAH

1 Úvod

Neuronové sítě patří do vědního oboru, který je již znám od počátku poloviny 20. století, avšak díky neúspěchům, které výzkum zpočátku provázel, nedošlo v této době k masivnímu rozšíření. Velký zájem o ně přišel až okolo konce 20. století, kdy se objevila nová vlna zájmu o tuto problematiku. Nebylo to dílem náhody, ale přispělo k tomu hned několik důvodů. Prvním a důležitým krokem byla obnova důvěry v tuto oblast, která byla získána objevením nových učících algoritmů a dalších významných objevů v této oblasti.

Druhým významným vlivem, který dopomohl k zvýšení zájmu o neuronové sítě je hledání odpovědí na požadavky společnosti. S vývojem lidské kultury, dochází i ke zvyšování požadavků na výpočetní stroje a jejich schopnosti. Obecně již společnosti nepostačuje nalézat řešení pro otázky, které jsou přesně popsány. Lidé hledali a hledají východiska, jak vyřešit takové úkoly, které nejsou přesně definovány.

Další důvod, který vedl ke zvýšení zájmu o neuronové sítě je podle názoru autora ten, že člověk chtěl vždy pomocí dostupných prostředků vytvořit tzv. „umělého člověka“, tedy takovou věc, která by byla schopná samostatně uvažovat a vykonávat úkoly bez jeho zásahu. Zde můžeme říci, že neuronové sítě a celý obor umělé inteligence podporuje svými charakteristikami tyto myšlenky. Tento důvod zvýrazňuje ještě fakt, že neurony, tak jak jsou navrženy, se snaží o to, aby se co nejvíce podobaly neuronům v těle živých organismů.

Tyto a ještě další významné důvody jen přispívají ke zvýšené pozornosti o tuto problematiku. Aniž bychom si to uvědomovali, tyto systémy nás začínají obklopovat v každodenním životě. Jako příkladnou ukázku můžeme uvést např. systémy, které slouží pro predikci spotřeby domácností, tepla a energie, pro ekonomické prognózy, pro předpovědi počasí, rozeznání řeči, obrazů a další.

Predikce složitosti softwarových produktů patří svými charakteristikami do oblasti těžko algoritmizovatelných problémů, tedy takových, které nelze přesně matematicky definovat. Je to způsobeno tím, že při vytváření softwarových produktů může vzniknout mnoho různých faktorů, které budou mít vliv na celkovou dobu realizace produktu. Tyto faktory se mohou odlišovat jak svou významností, tak i svou strukturou, vždy pro určitou oblast, ať už podnik či výrobní odvětví.

Pro predikci složitosti softwarových produktů existuje mnoho metod *(např. Borm, COCOMO, Function Points)[1]*, přesto se nám nabízí otázka, zda by nebylo „výhodnější“ a „spolehlivější“ použít neuronový přístup k řešení této problematiky.

2 Cíle práce a metodika

2.1 Cíle práce

Prvním cílem diplomové práce je popsat oblast umělé inteligence, vytyčit její hlavní směry a poukázat na nejdůležitější historické milníky a osobnosti, které se nejvíce podíleli na utváření prostředí v oblasti umělé inteligence.

Druhým cílem je zaměření se na neuronový přístup. V tomto úkolu se autor práce snaží popsat základní teoretická východiska o neuronu a neuronových sítích a shrnuje základní matematické modely používané pro učení neuronových sítí.

Třetím cílem je popsat oblasti, ve kterých je vhodné používat neuronové sítě. Pro tyto oblasti uvést i nejvhodnější typy používaných neuronových sítí.

Čtvrtým cílem diplomové práce je na základě informací získaných z naplnění předešlých cílů práce, najít vhodná východiska pro realizaci programu PETRA.

Pátým cílem této diplomové práce je popsat realizovaný program PETRA, jeho aplikační části a uživatelské rozhraní. V tomto úkolu se autor práce snaží popsat své poznatky a přínosy, které během realizace a základního testovaní programu PETRA zjistil a získal.

Posledním cílem je popsaní možností, které v programu PETRA lze dosáhnout a autorův pohled na neuronové sítě, jejich hodnocení a budoucí vývoj.

2.2 Metodika práce

Pro kapitolu tři autor zvolil tyto dvě metody ekonomického zkoumání:

- metoda abstrakce - její myšlenkou a podstatou je vyloučení náhodných a zbytečných skutečností. Cílem této metody je vznik uceleného souboru skutečností, které dávají do vzájemného vztahu podstatné, a pro určitou informaci, významné souvislosti,

- metoda analýzy - jejím úkolem je rozklad celku na menší části a u těchto částí zachytit věcné prvky, vztahy a vlastnosti, které daný systém, jako celek, nejvíce ovlivňují. Tato metoda se vyskytuje v mnoha rozlišných formách a autor použil pro vypracování této práce klasifikační a systémovou analýzu. Cílem klasifikační analýzy je roztřídění skutečností do skupin podle výrazných a charakteristických znaků. Cílem systémové analýzy je zkoumání složitějších systémů, zachycení jejich cílů, struktury fungování a jejich vztahu k vnějšímu okolí.

Při zpracování diplomové práce byl zvolen následující postup řešení:

- úvod práce je věnován historii umělé inteligence jako celku. Poté je oblast umělé inteligence rozdělena do dílčích oblastí. Jednotlivé dílčí oblasti umělé inteligence jsou popsány dle historického vývoje a jsou zde zmiňovány důležité osobnosti těchto vědních oblastí,
- velká část třetí kapitoly se zabývá neuronem a neuronovými sítěmi. Jelikož neuron a neuronové sítě jsou základním pilířem pro kapitolu čtyři, je výše uvedené problematice věnována velká pozornost. Autor popisuje neuron podle matematického hlediska a hledá podobnosti s fyziologickými hledisky neuronů v živých organismech. Po popsání matematického modelu neuronu jsou dále rozebírány možnosti učení neuronů a jejich zapojení do neuronových sítí. Neuronové sítě jsou dále pak rozděleny podle základních charakteristik, které vychází především z jejich topologií. Autor práce popisuje základní druhy sítí a u nich se zabývá jejich topologií a způsobem učení,
- závěr třetí kapitoly je věnován praktickému využití neuronových sítí. V této části práce jsou popsány základní oblasti, kde byly úspěšně nasazeny neuronové sítě. U každé z těchto oblastí se autor práce snažil popsat známý systém, který je v dané oblasti používán,
- na úvod čtvrté kapitoly je představen program PETRA a autor udává důvody, proč se rozhodl právě pro tento program. Dále jsou popsány základní cíle při tvorbě programu PETRA, které byly autorovi zadány. Zbytek této kapitoly se zabývá realizací programu PETRA. Autor nejdříve popisuje aplikační část jazyk ve kterém byl program vytvořen, znalostní bázi a základní parametry programu. Velkou část

pozornosti věnuje grafickému uživatelskému rozhraní. Z grafického uživatelského rozhraní jsou pak vybrány základní oblasti. U jednotlivých oblastí jsou popsány základní ovládací prvky. Kde je to vhodné, autor přidal příklady s názornými ukázkami.

- závěr práce je věnován programu PETRA (PErcepTRon Artifical inteligence). Autor práce uvádí možnosti nasazení programu PETRA v praxi a přidává vlastní hodnocení tohoto softwarového řešení (dále uvedeno pod pojmem program). Poté autor hodnotí neuronové sítě jako celek a přidává svůj vlastní úsudek, který si při realizaci neuronových sítí vytvořil. Konec kapitoly je věnován možnému budoucímu vývoji neuronových sítí.

3 Úvod do umělé inteligence a zhodnocení jejího praktického využití

3.1 Historie umělé inteligence

Ve výzkumu a vývoji inteligentních systémů lze od počátku jejího historického vývoje sledovat dva hlavní směry. První směr je inspirován novými biologickými a fyziologickými poznatky o fungování neuronové soustavy živých organismů a především lidského mozku. Do tohoto směru patří především výzkum neuronových sítí. Druhý směr se zaměřuje na analýzu způsobu lidského uvažování. Často je nazýván logickým přístupem. Začíná zkoumáním pojmové reprezentace světa a logické dedukce. Do tohoto směru můžeme zařadit klasickou logiku a neklasickou logiku.

3.1.1 Historie neuronových sítí

Za počátek vzniku oboru neuronových sítí je považována práce Warrena McCullocha a Waltera Pittse z roku 1943, kteří vytvořili velmi jednoduchý matematický model neuronu, což je základní buňka nervové soustavy. Číselné hodnoty parametru v tomto modelu byly převážně bipolární, tj. z množiny {-1,0,1}. Ukázali, že nejjednodušší typy neuronových sítí mohou v principu počítat libovolnou aritmetickou nebo logickou funkci. Ačkoliv nepočítali s možností bezprostředního praktického využití svého modelu, jejich článek měl velký vliv na ostatní badatele. Článkem se nechal inspirovat, např. zakladatel kybernetiky Norbert Wiener při studiu podobnosti činnosti nervové soustavy a systémů výpočetní techniky nebo autor amerického projektu elektronických počítačů John von Neuman. Ten napsal práce, ve kterých navrhoval výzkum počítačů, které by byly inspirovány činností mozku. Tyto návrhy, přestože byly hojně citovány, nepřinesly zpočátku očekávané výsledky.

V roce 1949 napsal Donald Hebb knihu „The Organization of Behaviour“, ve které navrhl učící pravidlo pro synapse neuronů (mezineuronové rozhraní). Toto pravidlo bylo inspirováno myšlenkou, že podmíněné reflexy, které jsou pozorovatelné u všech živočichů, jsou vlastnostmi jednotlivých neuronů.

Ve 40. a 50. letech 20. století, však zatím nebyly učiněny ještě zásadní pokroky v oblasti neurovýpočtů. Typickým příkladem výzkumu v tomto období byla v roce 1951 konstrukce prvního neuropočítače Snark, u jehož zrodu stál Marvin Minsky. Snark byl sice úspěšný z technického hlediska, dokonce již automaticky adaptoval váhy (tj. míra synaptické propustnosti), ale ve skutečnosti nebyl nikdy využit k řešení nějakého zajímavého praktického problému. Jeho architekturou se později inspirovali další konstruktéři neuropočítačů.

Za přelomový rok je označován rok 1957, kdy Frank Rosenblatt vynalezl tzv. perceptron, který je zobecněním McCullochova a Pittsova modelu neuronu pro reálný číselný obor parametrů. Pro tento model navrhl učící algoritmus, o kterém matematicky dokázal, že pro daná tréninková data nalezne, po konečném počtu kroků odpovídající váhový vektor parametrů nezávisle na jeho počátečním nastavení. Jedinou omezující podmínkou bylo to, že daný vektor musí existovat. Rosenblatt také napsal jednu z prvních knih o neurovýpočtech „Principles of Neurodynamics“. Na základě tohoto výzkumu Rosenblatt spolu s Charlesem Wightmanem a dalšími spolupracovníky sestrojil během let 1957 a 1958 první úspěšný neuropočítač, který nesl jméno „Mark I Perceptron“. Protože původním odborným zájmem Rosenblatta bylo rozpoznávání obrazců, „Mark I Perceptron“ byl navržen pro rozpoznávání znaků.

Díky úspěšné prezentaci uvedeného neuropočítače se neurovýpočty, které byly alternativou ke klasickým výpočtům realizovaným na von neumannovské architektuře počítače, staly novým předmětem výzkumu. Frank Rossenblatt je proto dodnes některými odborníky považován za zakladatele tohoto nového oboru.

Krátce po objevu perceptronu Bernard Widrow se svými studenty vyvinul další typ neuronového výpočetního prvku, který nazval „ADALINE“ (ADAptive LInear NEuron). Tento model byl vybaven novým výkonným učícím pravidlem, které se dodnes nezměnilo. Widrow se svými studenty demonstroval funkčnost „ADALINE“ na mnoha jednoduchých typových příkladech. Widrow také založil první firmu (Memistor Corporation) orientovanou na hardware neuropočítačů, která v první polovině 60. let vyráběla a prodávala neuropočítače a jejich komponenty.

Na přelomu 50. a 60. let dochází k úspěšnému rozvoji neurovýpočtů v oblasti návrhu nových modelů neuronových sítí a jejich implementací. Například Karel Steinbuch

vyvinul model binární asociativní sítě. Roger Barron a Lewey Gilstrap založil v roce 1960 první firmu zaměřenou na aplikace neurovýpočtů.

I přes značné úspěchy, které byly dosaženy v tomto období se obor neuronových sítí potýkal se dvěma problémy. První problém byl ten, že většina badatelů přistupovala k neuronovým sítím z experimentálního hlediska a zanedbávala analytický výzkum neuronových modelů. Druhým problémem bylo to, že nadšení některých výzkumných pracovníků vedlo k velké publicitě neopodstatněných prohlášení (např. za několik málo let bude vyvinut umělý mozek). Tyto skutečnosti a prohlášení diskreditovaly neuronové sítě v očích odborníků z jiných oblastí a odradily vědce a inženýry, kteří se o neurovýpočty zajímali. Navíc se samostatný obor neuronových sítí vyčerpal a další krok kupředu v této oblasti by býval požadoval radikálně nové myšlenky a postupy. Nejlepší odborníci tak oblast neuronových sítí začali opouštět a začali se zabývat příbuznými obory umělé inteligence. Velký vliv na úpadek zkoumání neuronových sítí v tomto období měla kampaň, vedená Marvinem Minským a Seymourem Papertem. Tito odborníci využili svůj vliv na to, aby zdiskreditovali výzkum neuronových sítí, nacházející se v krizi, ve snaze přenést finanční zdroje z této oblasti na jiný výzkum v oblasti umělé inteligence. Tuto kritiku uvedli v knize pod názvem „Perceptrons“. V této knize Minsky a Papert využili pro svou argumentaci známého triviálního faktu, že jeden perceptron nemůže počítat jednoduchou logickou funkci, tzv. vylučovací disjunkci (XOR). Tento problém lze sice vyřešit vytvořením dvouvrstvé sítě se třemi neurony, ale pro vícevrstvý perceptron nebyl v této době znám učící algoritmus. Autoři z toho nesprávně vyvodili, že takový algoritmus, vzhledem ke komplikovanosti funkce, kterou vícevrstvá síť počítá, snad ani není možný. Jejich tvrzení bylo všeobecně přijato a považováno za matematicky dokázané. Kampaň Minského a Paperta byla úspěšná, výzkum neuronových sítí nebyl již déle dotován a neurovýpočty byly považovány za neperspektivní.

Počátkem 80. let se badatelé v oblasti neurovýpočtů rozhodli ve svém bádání výrazně pokračovat a začali podávat vlastní grantové projekty zaměřené na vývoj neuropočítačů a jejich aplikace. Zásluhou programového manažera Ira Skurnicka začala v roce 1983 americká grantová agentura DARPA (Defense Advanced Research Projects Agency)

finančně podporovat výzkum neuronových sítí a jejího příkladu v krátké době následovaly i jiné organizace podporující základní i aplikovaný výzkum.

Další zásluhu na renesanci oboru neuronových sítí měl světově uznávaný fyzik John Hopfield, který se v této době zabýval neurovýpočty. Své výsledky publikoval v roce 1982 a 1984. Ukázal souvislost některých modelů neuronových sítí s fyzikálními modely magnetických materiálů. *V roce 1982 vydává práci, v níž zavádí obecněji propojenou síť než byla síť perceptronová a na které ukazuje, že tato síť může zastávat funkci asociativní paměti. V další práci v roce 1986 Hopfield spolu s Tankem ukázali, že tato síť může být použita k řešení optimalizačních úloh typu obchodního cestujícího.[2]* Sítě s touto architekturou se dnes nazývají podle jejich autora Hopfieldovy sítě. K dalším významným úspěchům v oblasti učení neuronových sítí dochází Kohonen, který v letech 1982-1990, zavadí tzv. učení bez učitele(unsupervised learning).

V roce 1986 publikovali své výsledky badatelé z tzv. „PDP skupiny“ (Parallel Distributed Processing Group). Ve svých pracích popsali učící algoritmus zpětného šíření chyby (backpropagation) pro vícevrstvou neuronovou síť a vyřešili tak problém, který se Minskému a Pappertovi v 60. letech jevil jako nepřekonatelná překážka pro využití a další rozvoj neuronových sítí. Tento algoritmus je doposud nejpoužívanější učící metodou neuronových sítí a jeho publikováním dosáhl zájem o neuronové sítě svého vrcholu. Výsledky praktických aplikací v nejrůznějších oborech, jako např. při rozpoznávání písma a řeči, v rozhodovacích modelech, při aproximaci funkcí, predikci časových řad a v expertních systémech, byly velmi povzbudivé.

3.1.2 Historie oblasti logických přístupů

Tato oblast chápe inteligenci jako schopnost reprezentace světa abstraktními symboly (pojmy) a schopnost správně s nimi manipulovat. Základním rámcem popisu světa se stala již od počátku predikátová logika a její fragment výroková logika. První program pro automatické dokazování vět v predikátovém počtu vytvořili již koncem padesátých let Davies a Putnam (1960). *Přelomový krok v technice dokazování vět byl umožněn po uveřejnění jedné ze stěžejních prací v oblasti umělé inteligence a tou byla práce Robinsona (1965), ve které byl zaveden dedukční systém založený na rezolučním*

principu. Další práce pak na tuto práci navazovaly a stanovily efektivní rezoluční strategii (např. Kowalski a Hayes v roce 1969, Anderson v roce 1967), která umožnila konstrukci efektivních systémů pro dokazování vět teorií formulovaných v predikátové logice. Za typického představitele těchto systémů je považován interpret jazyka Prolog, který umožňuje automatickou dedukci.

Reprezentace světa v predikátové logice je dnes základním způsobem, jak zachytit naše znalosti o vnějším světě, ale při modelování běžného usuzování je nedostačující, jelikož pracuje pouze s nedostačně strukturovanými pojmy. Proto byly navrženy formální systémy obsahující efektivní prostředky pro modelování bohatě strukturovaných pojmů. Jedná se především o sémantické sítě, rámce a skripty.[3]

Z důvodu, že při usuzování nejde jen o pravdivost či nepravdivost výpovědi, ale že výpověď zpravidla obsahuje i další aspekty, které nelze v klasické logice modelovat, vzniká odpověď na tyto požadavky v podobě neklasické logiky (např. modální logika, default logika, fuzzy logika).

3.2 Dělení okruhů v umělé inteligenci

Dělení umělé inteligence vychází z jejího historického vývoje. Na její rozdělení měl především vliv neúspěch tradiční umělé inteligence v 80. a 90. letech 20. století, který přispěl k hledání alternativních přístupů a ke změně úhlu pohledu na umělou inteligenci. Tyto alternativní přístupy jsou označovány jako Soft Computing a pro klasické přístupy zůstal název „Klasická umělá inteligence“.

Dělení:

Klasická umělá inteligence

- řešení úloh,
- robotika,
- automatické dokazování vět,
- formalizace znalostí.

Soft Computing

- neuronové sítě,
- evoluční algoritmy,

- aproximativní uvažování (fuzzy logika).

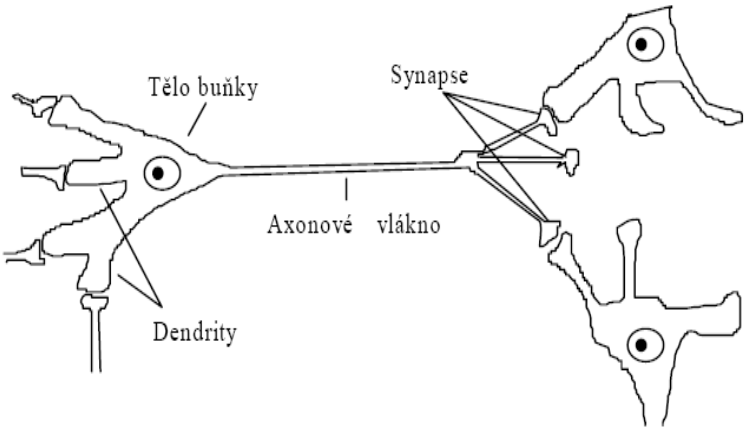
3.3 Neuron a jeho matematický model

Základním kamenem neuronové sítě je formální neuron. Jeho chování a konstrukce vychází z biologických poznatků o neuronových buňkách živých tvorů. Tak jako biologické neuronové buňky jsou ovlivněny prostředím, ve kterém jsou umístěny, tak i popis matematického modelu neuronu a jeho adaptace je ovlivněno prostředním, do kterého je zasazen.

3.3.1 Biologický model neuronu

Základní stavební jednotkou nervové soustavy živých organismů je neuronová buňka. Neurony jsou specializované buňky, které jsou určené k přenosu, zpracování a uchování informací. Jsou nezbytné pro realizaci životních funkcí organismu. V těle organismů můžeme nalézt několik druhů neuronových buněk, které se rozlišují podle vlastností, kterých jsou schopny. Neuronové buňky, které přenáší signál z vnějšího prostředí do organismu jsou receptory (např. tyčinky v oku). Neuronové buňky, které mají výstup na svalová vlákna a převádí nervový vzruch na fyzický pohyb svalu, se nazývají motorické neurony.

obrázek 1: Biologický neuron



Pramen: VOLNÁ, Eva. Neuronové sítě 1. Ostrava: Ostravská univerzita v Ostravě, 2002

Struktura neuronové buňky je následující:

- **tělo neuronu** (somat) – obsahuje základní součásti, které musí mít každá buňka živého organismu (např. buněčné jádro),
- **dendrity** vybíhají z těla buňky a větví se. Na dendritech jsou umístěny synapse, přes které do neuronové buňky vstupují vstupní signály,
- **axon** je mohutnější výběžek než jsou dendrity. Stejně jako dendrity se může větvit. Na konci axonu jsou umístěny synapse, kterými je axon spojen s dendrity nebo tělem jiného neuronu. Prostřednictvím axonu je elektrický signál, který vznikne excitací neuronu, veden směrem k synapsím. Tyto excitace obvykle vznikají v místě, kde axon vystupuje z těla neuronu,
- **synapse** vznikají v místě, kde se axon dostává do bezprostřední blízkosti jiného neuronu. Prostřednictvím synapsí je elektrický signál, který se šíří po axonu presynaptického neuronu, přenesen na postsynaptický neuron (tj. následující neuron). Z funkčního hlediska lze synapse rozdělit na excitační, které umožňují rozšíření vzruchu v nervové soustavě a na inhibiční, které způsobují jeho útlum.

Paměťová stopa v nervové soustavě vzniká pravděpodobně zakódováním synaptických vazeb na cestě mezi receptorem (čidlem orgánu) a efektorem (výkonným orgánem). Šíření informace je umožněno tím, že tělo neuronu i axon jsou obaleny membránou, která má schopnost za jistých okolností generovat elektrické impulsy. Tyto impulsy jsou z axonu přenášeny na dendrity jiných neuronů synaptickými branami, které svou propustností určují intenzitu podráždění dalších neuronů. Takto podrážděné neurony při dosažení určité hraniční meze (tzv. prahu) samy generují impuls a zajišťují tak šíření příslušné informace. Po každém průchodu signálu se synaptická propustnost mění, což je předpokladem paměťové schopnosti neuronů. Také propojení neuronů prodělává během života organismu svůj vývoj - v průběhu učení se vytváří nové paměťové stopy a při zapomínání se synaptické spoje přerušují. Nervová soustava člověka je velmi složitý systém, který je stále předmětem zkoumání. Uvedené velmi zjednodušené neurofyziologické principy nám však v dostatečné míře stačí k formulaci matematického modelu neuronové sítě.

3.3.2 Matematický model neuronu

Původním cílem výzkumu neuronových sítí byla snaha pochopit a modelovat způsob, jakým myslíme a způsob, jak funguje lidský mozek. Neurofyziologické poznatky umožnily vytvořit zjednodušené matematické modely, které se dají využít pro neurovýpočty a při řešení praktických úloh z oblasti umělé inteligence. Pro lepší orientaci bude autor označovat matematický neuron jako formální neuron.

Formální neuron je jednotka mající **n** vstupů ($x_1 ... x_n$), tyto vstupy mají charakterizovat dendrity. Vstupy jsou ohodnoceny reálnými čísly, které tvoří váhy(w_{1j}, ..., w_{nj}) na jednotlivých spojeních a které určují jejich propustnost a jsou shodné se synapsemi. Jednotlivé váhy mohou nabývat i záporných hodnot a tím docílit chování podobného jako u nervových inhibičních synaptických spojeních. U matematických neuronů můžeme nalézt ještě jeden vstup, který se váže pouze na vlastní neuron a tudíž není propojen s žádným neuronem ve svém okolí. Takový vstup nazýváme Bias (b_j), nebo může být označován jako (w_0). Bias nám slouží k nastavení vlastního stavu neuronu. Každý neuron má jeden výstup (y_j), který má podobnost v axonu nervové buňky. Vnitřní hodnota formálního neuronu (in_y_j) je dána:

$$in\_y_j = w_{0j} + \sum_{i=0} x_i * w_{ij}$$

Po vypočítání vnitřní hodnoty neuronu můžeme určit výstupní hodnotu neuronu (y_j), která charakterizuje stav neuronu a která modeluje elektrický impuls axonu. Hodnota (y_j) ze získá jako:

$$y_j = f(in\_y_j)$$

3.3.3 Základní typy přenosových funkcí

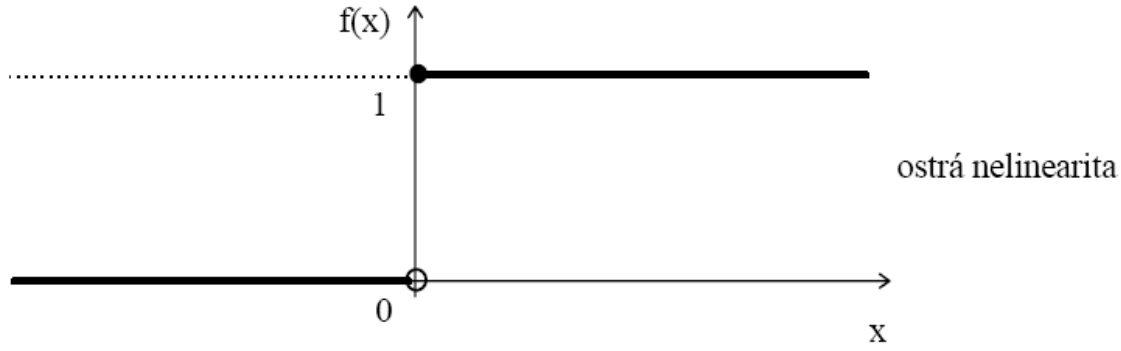
Ostrá nelineární funkce

Nejjednodušším typem přenosové funkce je ostrá nelinearita, která má pro j-tý neuron tvar:

$f(in\_y_j) = 1$, pokud $in\_y_j \geq 0$

$f(in\_y_j) = 0$, pokud $in\_y_j < 0$

graf 1: Nelineární funkce

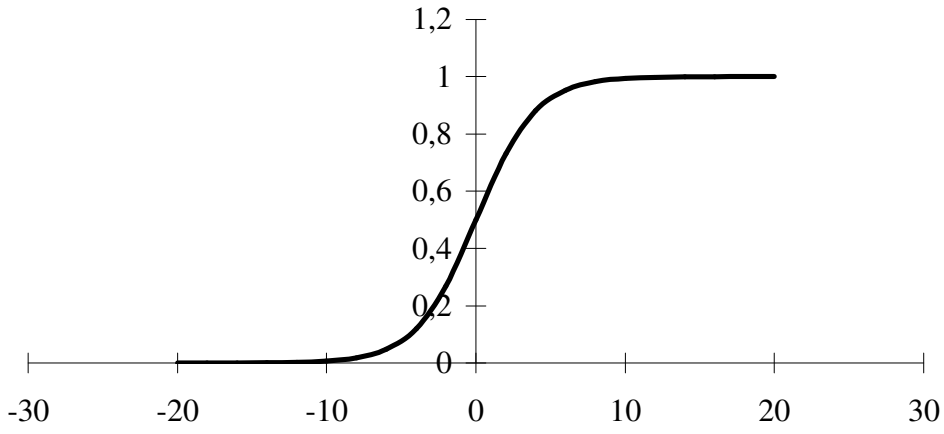


Logická funkce

Tato funkce patří mezi spojité aproximovatelné funkce. Pro j-tý neuron má tvar:

$$f(x)=\frac{1}{1+e^{-xk}}$$, pro k = 1

graf2: Logická funkce

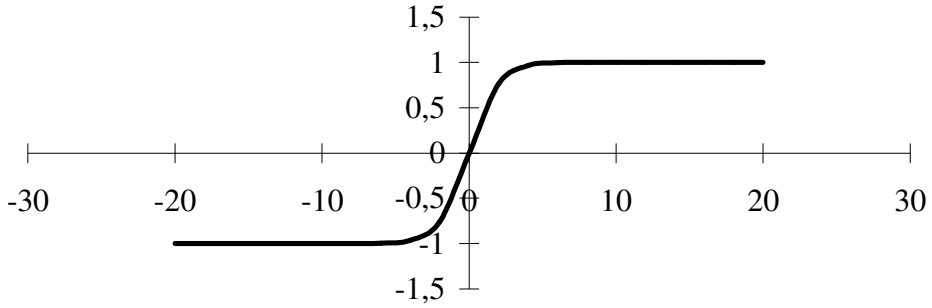


Hyperbolický tangens

Tato funkce patří mezi spojité aproximovatelné prahové funkce a její matematický tvar je:

$$g(x)=\frac{1-e^{-2kx}}{1+e^{-2kx}}$$, pro k = 0,5

graf 3: Hyperbolický tangens



3.3.4 Adaptace lineárního neuronu

Chování neuronu je ovlivněno hned několika aspekty. Jsou to vstupy neuronu, na které působí prostředím, ve kterém se neuron nachází. Vstupy postupně nabývají určitých hodnot a neuron na tyto hodnoty reaguje. Výsledkem reakce neuronu je hodnota, která se objeví na výstupu. Časové zpoždění, reakce mezi vstupy a výstupem neuronu bude autor pro tento případ ignorovat. Prostředí, ve kterém se neuron nachází, může být popsáno:

- deterministickým modelem,
- stochastickým modelem.

3.3.5 Deterministický model klasifikace

Při deterministické klasifikaci je dán prostor $\mathbf{X^n}$**.** Obsahující prvky $\mathbf{x} \in \mathbf{X^n}$, dále prvky **x** mohou patřit do podmnožin (kategorií) $\mathbf{C_1, \dots , C_m \subset X^n}$**.**

Cílem klasifikačního zařízení je na základě znalosti vstupního vektoru **x** určit, do kterých z těchto kategorií vektor **x** patří. *Podmínkou pro úspěšnou klasifikaci je, že se předpokládá, že vektor* $\boldsymbol{X^n}$ *je n-rozměrný eukleidovský prostor* $\boldsymbol{E^n}$ *nebo nějaký jeho podprostor např.* $\{0,1\}^n$ *všech uspořádaných n-tic 0 a 1. Dále musíme předpokládat, že nadrovina v prostoru* $\boldsymbol{E^n}$ *je dána rovnicí:*

$w_n x_n + w_{n-1} x_{n-1} + \ldots + w_1 x_1 + w_0 = 0$, *při čemž vektor* $(w_n, \ldots, w_1)$ *je vektor kolmý k této nadrovině. Za takto daných podmínek nám nadrovina rozděluje* E^n *na dva poloprostory. Pro body* $\mathbf{k} = (k_1, \ldots . k_n)$ *z jednoho poloprostoru platí:*

$$w_n k_n + w_{n-1} k_{n-1} + \ldots + w_1 k_1 + w_0 > 0$$

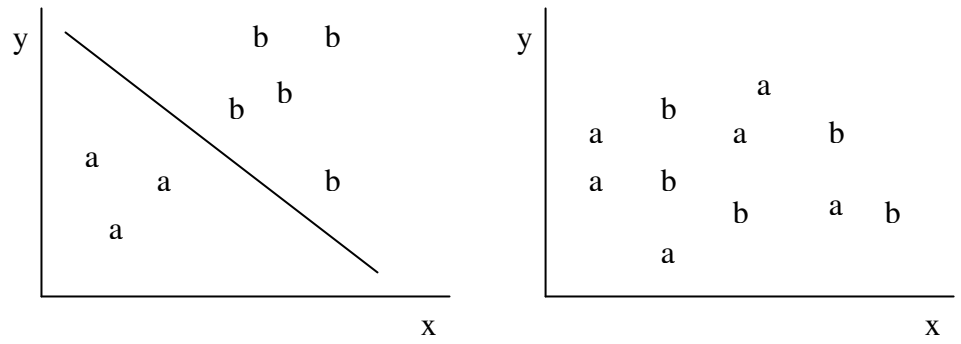
a pro body $\mathbf{k} = (k_1, \ldots, k_n)$ *z druhého poloprostoru platí:*

$$w_n k_n + w_{n-1} k_{n-1} + \ldots + w_1 k_1 + w_0 < 0.$$

Pokud tyto poloprostory označíme jako podmnožinu C^+*, pro body* $\mathbf{k}$ *náležící do první rovnice*$(y_1 > 0)$ *a podmnožinu* C^-*, pro body* $\mathbf{k}$ *náležící do druhé rovnice*$(y_1 < 0)$*, můžeme z nich určit další vlastnost.[4]* Pokud podmnožiny $C^+ \subset E^n$ a $C^- \subset E^n$ a zároveň jejich průnik $C^+ \cap C^-$ je prázdná množina, můžeme kategorii označit jako lineárně separabilní.

Jestliže je daná kategorie lineárně separabilní, potom existuje lineární neuron, který je schopen tuto kategorii klasifikovat s nulovou chybou.

obrázek 2: Lineárně separabilní a neseparabilní zobrazení



Kategorie C^+ a C^- jsou obvykle zadány konečnou množinou vektorů $\boldsymbol{x}$, o kterých je známo do jaké kategorie náleží. V tomto kontextu hovoříme o tzv. trénovací množině nebo učebním souboru. Trénovací množina je konečná množina dvojic $\{(x_i,d_i)\}$, kde hodnota d_i určuje, do které kategorie vektor x_i patří.

Chování neuronu je definováno jeho váhovým vektorem. V řešení úloh chceme docílit toho, že na základě učebního souboru stanovujeme váhový vektor tak, aby prvky učebního souboru byly klasifikovány správně. V případě, že se jedná o úlohu lineárně

neseparabilní, budeme se snažit určit váhový vektor tak, aby klasifikoval úlohu s co nejmenší dosažitelnou chybou. K řešené úloze lze přistupovat dvěma přístupy. Prvním postupem je řešení pomocí soustavy lineárních nerovnic. Rovnice musí být splněny tak, aby jednotlivé vektory učebního souboru ležely v jim odpovídajících poloprostorech a soustava je řešena numericky. Tento přístup je s rostoucím počtem prvků učebního souboru těžko zvládnutelný.

Druhý přístup je nalézt neuron klasifikující jednotlivé kategorie C^+ a C^- za pomoci metody učení. Tato metoda spočívá v tom, že na počátku nastavíme složky váhového vektoru neuronu na libovolné náhodné hodnoty, zpravidla blízké nule, nebo je položíme rovny 0. V dalším kroku na vstup neuronu vkládáme vektory učebního souboru $x_1, ..., x_n$ a sledujeme, jak neuron tyto vektory klasifikuje. Po klasifikaci vstupního vektoru neuronem změníme velikost složek váhového vektoru o malé hodnoty podle předem známého a definovaného způsobu. V této činnosti pokračujeme tak dlouho, dokud správnost klasifikace předkládaných vstupních vektorů není dostačující. Tato metoda adaptace váhového vektoru neuronu se nazývá **učení** nebo **trénování s učitelem (supervised learning).**

Základní podmínkou úspěšnosti této metody učení je nalezení vhodného algoritmu pro adaptaci váhového vektoru neuronu. První takový algoritmus navrhl v 50. letech Rosenblatt ve své knize o perceptronech. Proto se tento algoritmus nazývá perceptronový algoritmus adaptace vah.[4]

3.3.5.1 Perceptronový algoritmus adaptace vah

Je-li učební soubor C^+ a C^- lineárně separabilní, potom perceptronový algoritmus provede pouze konečný počet změn váhového vektoru. Po poslední provedené změně váhového vektoru bude neuron klasifikovat učební soubor C^+ a C^- s nulovou chybou.[5]

Pro perceptronový algoritmus adaptace vah je nutný předpoklad, že vektory **x** patří do jedné ze dvou kategorií C^+ a C^-. Je k dispozici učební soubor $C^+ \cup C^-$ obsahující konečný počet reprezentantů kategorií C^+ a C^-. Z učebního souboru jsou postupně, nebo náhodně vybírány vstupní vektory **x** a jsou vkládány na vstup. Po vyhodnocení odpovědi od neuronu je adaptován váhový vektor neuronu. Tím je realizován jeden

učební krok algoritmu. Na počátku učebního cyklu jsou jednotlivé váhové složky neuronu nastaveny jako náhodná čísla velmi blízká nule nebo jsou zvoleny jako 0.

Perceptronový algoritmus postupuje v následujících krocích:

1. *Start: Vyber libovolnou hodnotu vektoru* $\boldsymbol{w}$ *(např.* $\boldsymbol{w = 0}$*).*
2. *Test: Vyber* $\boldsymbol{x \in C^+ \cup C^-}$ *a vypočti* $\boldsymbol{xw}$*:*
   - *Je-li* $\boldsymbol{x \in C^+}$ *a zároveň* $\boldsymbol{xw > 0}$*, jdi na Test.*
   - *Je-li* $\boldsymbol{x \in C^+}$ *zároveň* $\boldsymbol{xw \leq 0}$*, jdi na Sečti.*
   - *Je-li* $\boldsymbol{x \in C^-}$ *a zároveň* $\boldsymbol{xw < 0}$*, jdi na Test.*
   - *Je-li* $\boldsymbol{x \in C^-}$ *a zároveň* $\boldsymbol{xw \geq 0}$*, jdi na Odečti.*
3. *Sečti:* $\boldsymbol{w = w + x}$
   - *jdi na Test.*
4. *Odečti :* $w = w - x$
   - *jdi na Test.[6]*

Využití perceptronového algoritmu adaptace vah neuronu bylo hlavně v počátku výzkumu neuronových sítí a neuropočítačů. Jako první úspěšný neuropočítač pracující na perceptronovém přístupu je považován „Mark I Perceptron“ navržený Roseblattovou skupinou.

3.3.5.2 Fungování a popis „Mark I Perceptron“

Neuropočítač „Mark I Perceptron“ fungoval následujícím způsobem: znak byl promítán na světelnou tabuli, ze které byl snímán polem 20x20 fotovodičů. Intenzita 400 obrazových bodů byla vstupem do neuronové sítě perceptronů, jejímž úkolem bylo klasifikovat, o jaký znak se jedná (např. „A“, „B“ apod.). „Mark I Perceptron“ měl 512 adaptovatelných váhových parametrů, které byly realizovány polem 8x8x8 potenciometrů. Hodnota odporu u každého potenciometru, která právě odpovídala příslušné váze, byla nastavována automaticky samostatným motorem. Ten byl řízen analogovým obvodem, který implementoval perceptronový učící algoritmus. Jednotlivé perceptrony bylo možné spojit se vstupy libovolným způsobem. Zpravidla bylo použito

náhodné zapojení, aby se ilustrovala schopnost perceptronu učit se požadované vzory bez přesného zapojení drátů v protikladu ke klasickým programovatelným počítačům.

3.3.5.3 Widrow-Hoffův algoritmus adaptace vah

Tento algoritmus vhodný pro adaptaci lineárního neuronu poprvé použili autoři Widrow a Hoff, odtud také plyne název tohoto algoritmu. Widrow-Hoffův algoritmus sice nezaručuje v lineárně separabilní situaci nalezení řešení po konečném počtu kroků, ale v případě, které lineárně separabilní nejsou, dosahuje obvykle lepších výsledků než perceptronový algoritmus. K odvození tohoto algoritmu se používá gradientní metoda hledání minima chybové funkce. Podobným způsobem se odvozuje algoritmus pro adaptaci vícevrstvých neuronových sítí, známý jako algoritmus zpětného šíření chyby. Gradientní metoda využitá pro jeden neuron je daleko jednodušší než v případě vícevrstvých neuronových sítí.

Předpoklad pro využití Widrow-Hoffova algoritmu je, že výstupní funkce $\mathbf{y = f(xw)}$ *je spojitá. Další podmínkou je, že neuron bude klasifikovat vkládané vstupní vektory* $\mathbf{x \in X^n}$ *na dvě kategorie* $C^+ \subset E^n$ *a* $C^- \subset E^n$ *následujícím způsobem:*

1. $\mathbf{x \in C^+}$ *pokud* $\mathbf{f(xw) \geq 0}$
2. $\mathbf{x \in C^-}$ *a zároveň* $\mathbf{f(xw) < 0}$

Za těchto daných podmínek nastavíme váhy neuronu tak, že bude platit:

1. $\mathbf{f(xw)} = d_1$, $d_1 \geq 0$ *pro* $\mathbf{x \in C^+}$
2. $\mathbf{f(xw)} = d_2$, $d_2 < 0$ *pro* $\mathbf{x \in C^-}$

Pokud docílíme výše uvedeného stavu, bude neuron klasifikovat kategorie C^+ *a* C^- *s nulovou chybou. Hodnoty* d_1 *a* d_2 *se obvykle volí 1 a -1. Chybová funkce při výše uvedeném nastavení vah bude vypadat takto:*

$E = \frac{1}{2}\sum_x (f(xw) - d)^2$, *kde* $d = d_1$ *pro* $x \in C^+$ *a* $d = d_2$ *pro* $x \in C^-$, *nabývá hodnoty 0.*

Jelikož funkce E nabývá pouze nezáporných hodnot a tvoří v prostoru váhových vektorů paraboloid, má právě jedno lokální minimum, které je zároveň globálním minimem. Toto minimum lze hledat gradientní metodou, jejíž algoritmus je následující:

1. *Počáteční hodnoty váhového vektoru w^0 zvolíme náhodně.*
2. *V n-tém kroku vypočteme gradient*

   $$grad\ E = \left(\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n} \right) = grad\ E = \sum_x (f(xw) - d) f'(xw) x.$$

   Pokud budeme počítat gradient pro jednotlivé předkládané vstupní vektory bude vypadat takto:

   $$grad\ E = (x^n w^{n-1} - d) x^n = \Delta x^n .$$

   *Znak **Δ** označuje odchylku mezi skutečnou a požadovanou hodnotou výstupu neuronu.*

3. *Pozměníme váhový vektor $\mathbf{w^{n-1}}$ na váhový vektor $\mathbf{w^n}$, při čemž*

   $\mathbf{w^n = w^{n-1} - \varepsilon\ grad\ E}$ *, kde $\varepsilon > 0$ je konstanta definující velikost změny váhového vektoru v jednotlivém kroku.*

Princip Widrow-Hoffova algoritmu s využitím gradientní metody, vychází z libovolně zvolené hodnoty váhového vektoru a po té postupně měníme váhový vektor o malou hodnotu v opačném směru ke směru gradientu. V tomto směru se velikost funkce E zmenšuje nejvíce a váhový vektor se blíží k hodnotě, pro kterou funkce E nabývá hodnotu, která je jejím lokálním minimem.[7]

Popsanému způsobu adaptace vah neuronu se říká adaptace vah podle **Δ-pravidla.** Toto pravidlo použili poprvé právě Widrow-Hoff u svého modelu neuronu, který nazvali ADALINE, a proto se také někdy toto pravidlo nazývá Widrow-Hoffovo pravidlo, v některých případech používají autoři pro něj název **LMS-pravidlo.**

Na začátku kapitoly autor zmiňoval předpoklad, že pro řešení problému je potřeba mít spojitou výstupní funkci. My však můžeme daný problém řešit i pro funkci, která spojitá nebude. V tomto případě, budeme nuceni oddělit excitaci neuronu **e(n)** a výstup neuronu **o(n).** Pro nastínění problému můžeme jako příklad použít nespojitou funkci *sing(x)* a vzorec by vypadal následujícím způsobem:

$$e(n) = \sum_j w_j x_j - \vartheta = xw ,$$

$$y(n) = sign(e(n)),$$

zbytek algoritmu již zůstane beze změny. Po nalezení minima funkce budou vektory $x \in C^+$ *mít hodnotu excitace* xw *blízkou hodnoty* $d_1 > 0$ *a jejich výstupní hodnota* $y(n) = 1$*. Vektory* $x \in C^-$ *budou mít hodnotu excitace* xw *blízko hodnoty* $d_2 < 0$ *a jejich výstupní hodnota* $y(n) = -1$*. Tím jsme docílili, že daný neuron bude schopen rozlišovat mezi kategoriemi* C^+ *a* C^-*.[7]*

3.3.6 Pravděpodobnostní model klasifikace

Při pravděpodobnostním modelu klasifikaci je dán prostor $\mathbf{X^n}$**.** Obsahující prvky $\mathbf{x} \in \mathbf{X^n}$, dále prvky **x** mohou patři do podmnožin (kategorií) $\mathbf{C_1, \dots, C_m} \subset \mathbf{X^n}$**.** *Oproti deterministickému modelu je navíc přidána na kartézském součinu* $X * C$*,* $C = \{C_1, \dots, C_n\}$ *pravděpodobnostní distribuce* $P(x, C)$*. Jelikož můžeme určit pravděpodobnostní distribuci, můžeme také ze vztahu odvodit i podmíněnou pravděpodobnost* $P(Ci \mid x)$*. Cíl pravděpodobnostního klasifikačního modelu je stejný jako v případě modelu deterministického, tedy na základě znalosti vstupního vektoru* x *určit, do kterých kategorií* $C_1, \dots, C_m \subset X^n$ *vektor* x *patří. K tomu, abychom daný vstupní vektor správně zařadili do odpovídající kategorie, využijeme bayesovské klasifikační pravidlo:*

$P(C_i \mid x) \geq \max_j P(C_j \mid x)$*, vektor* x *je zařazen do té kategorie* C_i*, pro který je splněna tato podmínka. Chyba, která při této bayesovské klasifikaci vzniká, se nazývá bayesovská chyba. Bayesovská chyba je nejmenší dosažitelnou chybou, s kterou lze v daném modelu vektor* x *klasifikovat. Pokud pro kategorie* $C_1, \dots, C_m \subset X^n$ *platí, že* $C_1 \cap C_j$ *je prázdná množina a* $i, j = 1, \dots, n \; ; \; i \neq j$*, můžeme prohlásit o těchto kategoriích, že jsou separabilní. Pokud je splněn předpoklad a dané kategorie jsou separabilní, pak bayesovská chyba je nulová.[8]*

Problém této pravděpodobnostní klasifikaci podle výše popsaného vzorce nastává v praktických aplikacích, kde např. při odhadu $\mathbf{X^n = \{0,1\}^{200}}$ musíme odhadnout 2^{200} podmíněných pravděpodobností, což ze statického materiálu, který obsahuje zpravidla tisíce až desetitisíce realizací vektoru **x** nelze. Tento problém můžeme řešit pomocí teorií bayesovských sítí. Nejjednodušší způsob jak daný problém řešit, je předpokládat, že jednotlivé složky vektoru **x** jsou v rámci každé kategorie $\mathbf{C_i}$ navzájem nezávislé.

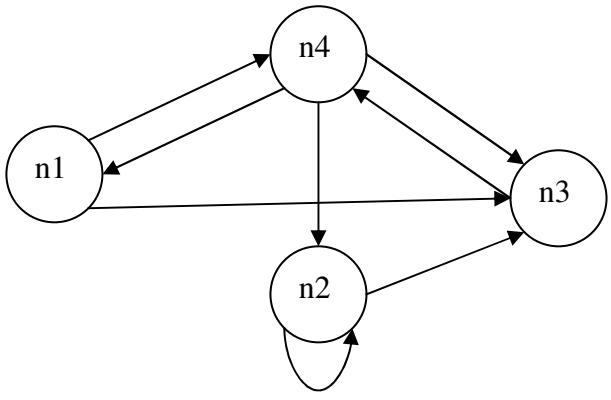
Pokud je tento předpoklad splněn, lze na daný vektor použít tzv. naivní bayesovo pravidlo, které podmíněné pravděpodobnosti **P(x | Ci)** vyjadřuje takto:

$P(x \mid C_i) = \prod_j P(x_j \mid C_i)$ *viz. [8]*, při takto upraveném vzorci pro klasifikaci se nám podstatně snižuje počet odhadů podmíněných pravděpodobností ze statistického materiálu. Ve výše uvedeném příkladu místo 2^{200} odhadů by byl nový počet odhadů 200. Pokud ale, předpoklad nezávislosti neplatí, tak potom nebudeme rozhodovat podle bayesovského kriteria a chyba rozhodování bude větší než je optimální bayesovská chyba.

3.4 Neuronové sítě

Za neuronovou síť můžeme označit propojení dvou a více neuronů. Neurony jsou mezi sebou propojeny tak, že výstup jednoho neuronu je vstupem pro jeden nebo i více neuronů. Výstup neuronu může také tvořit vlastní vstup do neuronu samého. Počet neuronů a jejich vzájemné propojení určuje tzv. topologii sítě. Graficky můžeme tuto topologii znázornit orientovaným grafem, jehož uzly budou reprezentovat jednotlivé neurony a hrany budou jednotlivá propojení neuronů.

obrázek 3: Neuronová síť



Neuronová síť se obvykle skládá z lineárních neuronů. Každá propojení mezi vstupem i - tého neuronu s výstupem j-tého neuronu (tedy hrany grafu) jsou popsána váhovou složkou $\mathbf{w_{ij}}$. Jelikož obvykle mají neurony navíc ještě fiktivní vstup (Bias), který je identický hodnotě 1, nese zpravidla tento vstup označení $\mathbf{w_{i0}}$. Takto popsaný orientovaný graf s váhovými složkami u jednotlivých hran se nazývá konfigurací (strukturou) sítě.

Topologie neuronové sítě se nazývá úplnou, pokud jednotlivé neurony sítě jsou propojeny maximálním možným způsobem.

Jsou dva základní typy architektury sítě:

- cyklická (rekurentní) síť,
- acyklická (dopředná) síť.

Cyklická síť je taková síť, kde v grafu topologie sítě existuje uzavřená smyčka, tedy že v síti existuje skupina neuronů, které tvoří kruh. Při manipulaci s neurony v cyklických sítích musíme brát v úvahu to, že jednotlivé neurony mění svůj stav v diskrétních časových intervalech. Výsledný výstup neuronu pak lze popsat funkcí takto:

$y_{(t+1)} = f(x_{(t)}w_{(t)})$.

Acyklická síť je taková síť, kde v grafu topologie sítě neexistuje uzavřená smyčka. Acyklickou síť můžeme rozdělit do vrstev. Tyto vrstvy pak můžeme uspořádat tak, že spoje mezi neurony vedou z vyšších vrstev do nižších. Při manipulaci s neurony u acyklických sítí můžeme předpokládat, že neurony reagují na změnu hodnot na svém vstupu okamžitě a při popisu jejich chování není třeba uvažovat o jejich závislosti na čase. Výstup neuronu pak lze popsat funkcí takto:

$y = f(xw)$.

Z hlediska využití rozlišujeme v síti vstupní, skryté a výstupní neurony. Na vstupní neurony přicházejí podněty z vnějšího prostředí, zpravidla jsou vyjádřeny reálným nebo binárním číslem. Tyto podněty nejsou vstupními neurony nijak zpracovány a vstupní neurony je předávají na své výstupy. Takto se chovajícím neuronům říkáme receptory.

Skryté neurony se vyskytují u speciálního případu acyklické struktury tzv. vícevrstvé neuronové sítě. Topologie sítě je následující:

- nejvyšší vrstvu tvoří vstupní neurony,
- nejnižší vrstvu tvoří výstupní neurony,
- ostatní vrstvy, které tvoří spojení mezi nejvyšší a nejnižší vrstvou, tvoří tzv. skryté vrstvy.

Konfiguraci neuronové sítě s **n** neurony a s fiktivními vstupy lze jednoznačně vyjádřit maticí **M** typu n + 1 ∗ n, pro jejíž prvky $\mathbf{m_{ij}}$ platí:

1. Pokud existuje spojení mezi výstupem i-tého neuronu a vstupem j-tého neuronu a hrana, která tento spoj reprezentuje, je ohodnocena vahou $\mathbf{w_{ij}}$, pak platí, že $\mathbf{m_{ij} = w_{ij}}$.
2. Pokud spojení mezi i-tým neuronem a j-tým neuronem neexistuje, pak **mij** = 0.
3. $\mathbf{m_{i0} = w_{i0}}$, **i = 1, ..., j.**

Za splnění těchto předpokladů dostaneme následující matici:

M

$$\begin{bmatrix} \mathbf{w_{10}} & \cdots & \mathbf{w_{1j}} \\ \vdots & \vdots & \vdots \\ \boldsymbol{w_{i0}} & \cdots & \boldsymbol{w_{ij}} \end{bmatrix}$$

3.4.1 Režimy práce neuronové sítě

Režimy práce neuronové sítě se dají rozdělit podle toho, k jakým dochází změnám v síti během její činnosti. V neuronové síti se může během činnosti měnit:

1. stav neuronů sítě,
2. konfigurace váhy $\mathbf{w_{ij}}$,
3. topologie sítě.

Od těchto změn se odvíjí tři režimy práce neuronové sítě:

1. aktivní (změna stavu),
2. adaptační (konfigurace vah),
3. organizační (topologie sítě).

V aktivním režimu práce je vstup sítě nejdříve nastaven na určité hodnoty. Poté dochází k postupným změnám stavů neuronů v síti, označovaný také jako tzv. výpočet nebo relaxace sítě. Zpravidla je požadavkem, aby se po konečném počtu kroků stav

neuronů sítě již dále neměnil. Hodnoty na výstupních neuronech pak představují výsledek výpočtu sítě.

V adaptačním režimu se mění váhy neuronu tak, aby po adaptaci byla síť schopná provádět příslušný výpočet. Změna váhových složek, při nichž v adaptaci dochází, se označuje jako učení nebo též trénování neuronové sítě. Učení neuronové sítě se provádí na základě učebního (trénovacího) souboru.Učení neuronových sítí se provádí dvěma způsoby:

- učení s učitelem (supervised learning),
- učení bez učitele (unsupervised learning).

V případě učení neuronových sítí s učitelem obsahuje učební soubor konečný počet dvojic ($\mathbf{x}_i$, $\mathbf{d}_i$), kde $\mathbf{x_i}$ je představitelem jednoho z možných vstupů neuronové sítě a $\mathbf{d_i}$ představuje požadovaný výstup sítě. Učení probíhá tak, že na vstup sítě se vloží vstupní vektor $\mathbf{x_i}$. Síť provede výpočet a výsledek porovná s očekávaným výstupem $\mathbf{d_i}$, dále pak na základě předem známého algoritmu dochází k úpravě váhových složek jednotlivých neuronů. Tento algoritmus se nazývá adaptační algoritmus sítě. Tyto algoritmy bývají zpravidla inkrementační, tj. změna vah se vždy provádí při předložení jednoho vstupního vektoru. Vstupní vektory se z učebního souboru mohou vybírat podle pořadí, tj. jak jsou v učebním souboru seřazeny nebo k jejich výběru může docházet náhodně. Předložením jednoho vstupního vektoru a následná změna váhových složek neuronu představuje jeden krok učebního algoritmu. Pokud jsme z učebního souboru již vybrali všechny prvky $\mathbf{x_1, ..., x_n}$, splnili jsme tzv. jednu učební epochu neuronové sítě.

Při učení bez učitele obsahuje učební soubor pouze vstupní vektory $\mathbf{x_i}$. Ke změně vah dochází opět adaptačním algoritmem, ale pouze na znalosti vstupu $\mathbf{x_i}$. Tímto způsobem naučené neuronové sítě dokáží klasifikovat vstupní vektory do kategorií tak, že v každé kategorii jsou vektory, které jsou si navzájem velmi podobné. Jiné algoritmy dokáží neuronové sítě adaptovat tak, že při dokončení učení lze sítě využít jako asociativní paměť. Učení bez učitele se také často nazývá jako samoorganizace neuronové sítě.

Při organizačním režimu mění neuronová síť svou topologii. Tento režim však nebývá častý a síť se do tohoto režimu nedostává. Zpravidla se topologie sítě nastaví na počátku adaptace a dále se už nemění. Organizační režim se používá v případech, kdy podle analýzy výsledku testování v učební epoše dojde ke konfiguraci sítě

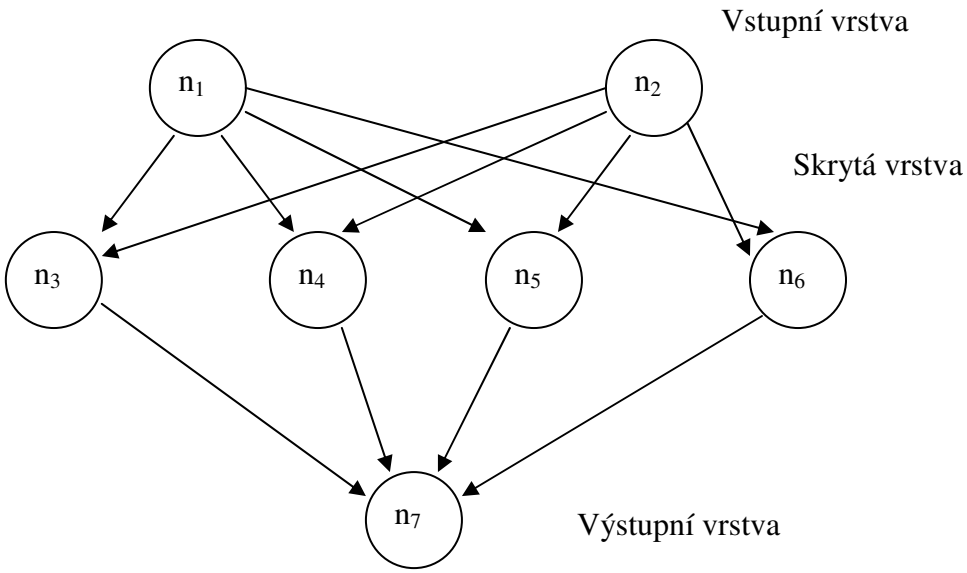
(např. ubráním neuronu, který se na celkovém výsledku podílí zanedbatelnou měrou, atd.).

3.4.2 Vícevrstvé neuronové sítě

Vícevrstvé neuronové sítě se skládají z:

- vstupní vrstvy – složené z receptorů,
- jedné nebo více skrytých vrstev,
- výstupní vrstvy.

obrázek 4: Vícevrstvé neuronové sítě



Vícevrstvé neuronové sítě jsou acyklické sítě, kde výstupy neuronů dané vrstvy jsou spojeny se vstupy neuronů vrstvy následující. Jelikož se jedná o síť acyklickou, budeme předpokládat, že vstup ze vstupní vrstvy se přenese ihned na vrstvu výstupní a tedy nebude v tomto případě brán v úvahu čas. Dále budeme předpokládat, že neurony, ze kterých je síť složená, jsou lineárními neurony s fiktivním vstupem nebo jejich sigmoidálními aproximacemi. Speciální skupinou jsou neurony vstupní vrstvy (receptory). Tyto neurony jsou tvořeny speciálním degenerovaným řešením lineárních neuronů. Mají pouze jeden vstup a hodnota na něm je rovna hodnotě na jejich výstupu. Hodnoty na výstupních neuronech jsou ovlivněny nejen vnitřní sítí, ale také jejich vlastní funkcí.

Na základě teoretických výsledků a na základě zkušeností z praktických aplikací, se v praxi ustálila následující pravidla použití vrstvených sítí:

1. *Používají se vrstvené sítě s jednou nebo dvěma skrytými vrstvami.*
2. *Ve skrytých vrstvách se používají neurony* $y = f(\boldsymbol{x}.\boldsymbol{w})$.
3. *Ve výstupní vrstvě se použijí buď neurony* $y = f(\boldsymbol{x}.\boldsymbol{w})$ *nebo* $y = \boldsymbol{x}.\boldsymbol{w}$. *Pokud je třeba aproximovat funkce s funkčními hodnotami mimo interval (0, 1), použijí se neurony* $y = \boldsymbol{x}.\boldsymbol{w}$.
4. *Počet neuronů ve vstupní a výstupní vrstvě je dán typem úlohy, která se neuronovou sítí řeší.*
5. *Počet neuronů první skryté vrstvy za vstupní vrstvou se volí zhruba dvakrát větší než počet neuronů vstupní vrstvy. Pokud má síť ještě další skrytou vrstvu, zvolí se počet jejich neuronů tak, aby byl větší než počet neuronů vstupní vrstvy a menší než počet neuronů první skryté vrstvy.[9]*

Důležitým prvkem u vícevrstvých neuronových sítí je najít takový algoritmus, který zabezpečí jednoduché a kvalitní naučení vícevrstvé neuronové sítě. Jedním z nejpoužívanějších takových algoritmů je algoritmus zpětného šíření chyby (Backpropagation of error).

3.4.2.1 Algoritmus zpětného šíření chyby

Pro lineární neuron je algoritmus znám již od 50. let 20. století. Pro vícevrstvou síť byl přesně formulován až v polovině 80. let výzkumným týmem vedeným Rumelhartem.[9] Objev tohoto algoritmu znamenal velké oživení zájmu o problematiku neuronových sítí a dovolil podstatně rozšířit oblast jejich aplikací. K odvození algoritmu zpětného šíření chyby byla použita gradientní metoda hledání minima funkce:

$$E(w_{ij}) = \frac{1}{2}\sum_{x}\sum_{\alpha} c_{\alpha}(O_{\alpha} - D_{\alpha})^2 \text{. viz. [10]}$$

Tento algoritmus v přibližně 80 % všech aplikací neuronových sítí. Samotný algoritmus obsahuje tři etapy:

- dopředné (*feedforward*) šíření vstupního signálu tréninkového vzoru,

- zpětné šíření chyby,
- aktualizace váhových hodnot na spojeních.

Během dopředného šíření signálu obdrží každý neuron ve vstupní vrstvě (X_i, $i = 1,..., n$) vstupní signál $\mathbf{x_i}$ a zprostředkuje jeho přenos ke všem neuronům vnitřní vrstvy ($Z_1, ..., Z_j$). Každý neuron ve vnitřní vrstvě vypočítá svou aktivaci $\mathbf{z_j}$ a pošle tento signál všem neuronům ve výstupní vrstvě. Každý neuron ve výstupní vrstvě ($Y_1, ..., Y_k$) vypočítá svou aktivaci $\mathbf{y_k}$, která odpovídá jeho skutečnému výstupu (*k-tého* neuronu) po předložení vstupního vzoru. Tímto způsobem získáme odezvu od neuronové sítě na vstupní podnět daný excitací neuronů vstupní vrstvy.

K popsání algoritmu bude použito následující značení:

$\boldsymbol{x}$ *- vstupní vektor,* $x \in (x_1, ..., x_i)$,

$\boldsymbol{t}$ *- výstupní tréninkový vektor,* $t \in (t_1, ..., t_m)$,

$\boldsymbol{\delta k}$ *- částečné váhové korekce pro* $\boldsymbol{w_{jk}}$ *příslušející chybě na spojeních vedoucích k neuronu* $\boldsymbol{Y_k}$ *ve výstupní vrstvě,*

$\boldsymbol{\delta j}$ *- částečné váhové korekce pro* $\boldsymbol{v_{ij}}$ *příslušející chybě na spojeních vedoucích k neuronu* $\boldsymbol{Z_j}$ *ve skryté vrstvě,*

$\boldsymbol{\alpha}$ *- koeficient učení,*

$\boldsymbol{X_i}$ *– i-tý neuron ve vstupní vrstvě. Pro neurony ve vstupní vrstvě je hodnota vstupního i výstupního signálu stejná =* $\boldsymbol{x_i}$,

$\boldsymbol{v_{0j}}$ *- bias j-tého neuronu ve skryté vrstvě,*

$\boldsymbol{Z_j}$ *– j-tý neuron ve skryté vrstvě. Hodnota vstupního signálu pro* $\boldsymbol{Z_j}$ *je* $\boldsymbol{z\_in_j}$*:*

$z\_in_j = v_{0j} + \sum_{i=1} x_i * v_{ij}$ *a tedy hodnota vstupního signálu pro* $\boldsymbol{Z_j}$ *je* $\boldsymbol{z_j = f(z\_in_j)}$,

$\boldsymbol{w_{0k}}$ *- bias k-tého neuronu ve výstupní vrstvě,*

$\boldsymbol{Y_k}$ *– k-tý neuron ve výstupní vrstvě. Hodnota vstupního signálu pro* $\boldsymbol{Y_k}$ *je* $\boldsymbol{y\_in_k}$*:*

$y\_in_k = w_{0k} + \sum_j z_j * w_{jk}$ *a tedy hodnota vstupního signálu pro* $\mathbf{Z_j}$ *je* $z_j = y_k$ $= f(y\_in_k)$.

Algoritmus adaptace vah podle metody zpětného šíření chyby :

1. *Váhové hodnoty a bias jsou inicializovány jako malá náhodná čísla blízká 0. Dále se přiřadí inicializační hodnota koeficientu učení* α.
2. *Dokud není splněna podmínka ukončení výpočtu, budou opakovány body (3 - 10).*
3. *Pro každý (bipolární) tréninkový pár* ***s-t*** *provádět kroky (4 - 9).*

- ***fáze feedforward:***

4. *Aktivace vstupních neuronů* $\mathbf{X_i}$, *kde* $(i = 1, \ldots, n)$ *a současně* $x_i = s_i$.
5. *Vypočítání vstupních hodnot vnitřních neuronů* $\mathbf{Z_j}$, *kde* $(j = 1, \ldots, m)$, *podle funkce :*

   $z\_in_j = v_{0j} + \sum_{i=1} x_i * v_{ij}$ *a stanovení výstupních hodnot vnitřních neuronů* $z_j = f(z\_in_j)$.
6. *Stanovení skutečných výstupních hodnot signálu neuronové sítě* $\mathbf{Y_k}$, *kde* $(k = 1, \ldots, m)$, *podle funkce:*

   $y\_in_k = w_{0k} + \sum_j^p z_j * w_{jk}$, *kde* $y_k = f(y\_in_k)$.

- ***fáze backpropagation:***

7. *Ke každému neuronu ve výstupní vrstvě* $\mathbf{Y_k}$, *kde* $(k = 1, \ldots, m)$, *je přiřazena hodnota očekávaného výstupu pro vstupní tréninkový vzor. Poté je vypočteno:*

   $\delta_k = (t_k - y_k) f'(y\_in_k)$, *které je součástí váhové korekce* $\Delta w_{jk} = \alpha \delta_k z_j$ *i korekce biasu* $\Delta w_{0k} = \alpha \delta_k$.

8. *Ke každému neuronu ve vnitřní vrstvě Zj, kde (j = 1, ... , m) je přiřazena sumace jeho delta vstupů (tj. neuronů, které se nacházejí v následující vrstvě),*

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk} .$$

Vynásobením hodnot δ_in_j *a derivací její aktivační funkce obdržíme:* $\delta_k = \delta\_in_j * f'(z\_in_k)$, *které je součástí váhové korekce* $\Delta v_{ij} = \alpha\delta_j x_i$ *a i korekce biasu* $\Delta v_{0j} = \alpha\delta_j$.

- ***aktualizace vah a prahů:***

9. *Každý neuron ve výstupní vrstvě* $\mathbf{Y_k}$, *kde* $\mathbf{(k = 1, ... , m)}$, *aktualizuje na svých spojeních váhové hodnoty včetně svého biasu* $\mathbf{(j=0, ..., n)}$ *a to takto:*

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk} .$$

10. *Každý neuron ve vnitřní vrstvě* $\mathbf{Z_j}$, *kde* $\mathbf{(j = 1, ... , m)}$, *aktualizuje na svých spojeních váhové hodnoty včetně svého biasu (i=0, ..., n):*

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij} .$$

11. *Algoritmus je ukončen, pokud již nenastávají žádné změny váhových hodnot nebo pokud již bylo vykonáno maximálně definované množství váhových změn. Jestliže tyto podmínky nebyly splněny, dochází k pokračování algoritmu.[10]*

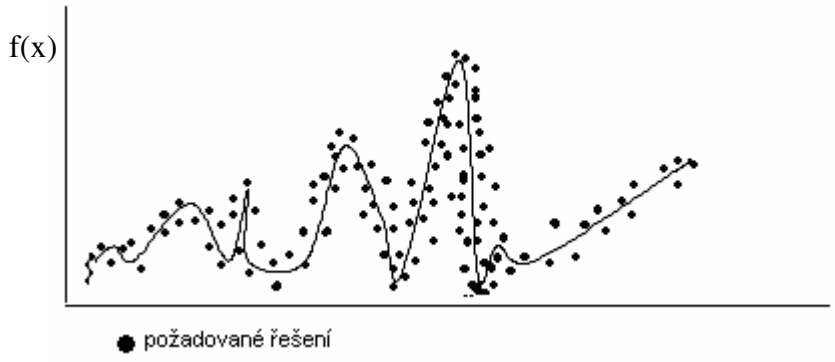
Ačkoliv vlastní popis učícího algoritmu backpropagation je formulován pro klasický von Neumannovský model počítače, přesto je zřejmé, že jej lze implementovat distribuovaně. Pro každý tréninkový vzor probíhá nejprve aktivní režim pro jeho vstup tak, že informace se v neuronové síti šíří od vstupu k jejímu výstupu (feedforward). Potom na základě externí informace učitele o požadovaném výstupu, tj. o chybě u jednotlivých vstupů se počítají parciální derivace chybové funkce tak, že se signál šíří zpět od výstupu ke vstupu (backpropagation). Výpočet sítě při zpětném chodu probíhá sekvenčně po vrstvách, přitom v rámci jedné vrstvy může probíhat paralelně (adaptace vah).

3.4.2.2 Komplikace u vícevrstvých neuronových sítí

Při praktických aplikacích se zjistilo, že u vícevrstvých neuronových sítí může docházet k problému s vhodnou volbou topologie sítě. Z teoretických výsledků je znám fakt, že k realizaci jakékoliv libovolné funkce stačí pouze vícevrstvá neuronová síť s jednou skrytou vrstvou.[11] Při řešení problémů se však v praxi ukazuje, že lze někdy získat lepší výsledky při použití dvou skrytých vrstev. Na základě tohoto poznatku dochází při volbě vhodné topologie sítě k experimentování, kdy se volí buď síť s jednou nebo dvěmi skrytými vrstvami. Při volbě dvou skrytých vrstev se postupuje s volbou počtu neuronů podle pravidel pro vícevrstvé neuronové sítě

Další komplikace, se kterou se můžeme setkat u vícevrstvých neuronových sítí, je tzv. přeučení (overfitting). K tomuto jevu dochází, když zvolíme zbytečně velký počet neuronů. Výsledná síť se ve fázi učení příliš věrně naučí vyhodnocovat dané vstupy a ztrácí jednu z důležitých výhod neuronového přístupu a tím je schopnost abstrakce. Při ztrátě abstrakce již daná síť nemusí správně abstrahovat tvar funkce nebo jen v omezené míře.

obrázek 5: Přeučení sítě (overfitting)



3.4.2.3 Lineární asociativní neuronové sítě

Lineární asociativní neuronové sítě patří mezi sítě s jednoduchou architekturou. Skládají se z vrstvy receptorů a z vrstvy výstupních neuronů. Receptory, jako u ostatních sítí, jsou speciální degenerované lineární neurony, které přenášejí hodnoty ze vstupu na jejich výstup. Výstupní neurony jsou lineární neurony s výstupní funkcí $\mathbf{y = (xw)}$, kde $\mathbf{x = (x_1, \ldots, x_n)}$ a $\mathbf{w = (w_1, \ldots w_n)}$. Zvláštností je, že u lineárních asociativních

neuronových sítí nemají neurony fiktivní vstup. Lineární asociativní neuronová síť má zpravidla **n** receptorů a **m** výstupních neuronů, u nichž se předpokládá, že každý receptor je propojen s každým výstupním neuronem. Konfiguraci sítě lze popsat maticí **W,** jejíž řádky tvoří jednotlivé váhové vektory výstupních neuronů. Dále pak tu budeme mít dvě matici typu **n** x **1,** které budou reprezentovat vstupní vektor **x** a výstupní vektor **y**.

$$\mathbf{W} = \begin{bmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \vdots & \vdots \\ w_{m1} & \cdots & w_{mn} \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

Výstup neuronové sítě lze potom popsat maticovou rovnicí: **y = Wx.**

*Učební algoritmus pro změnu konfigurace sítě **W** vypadá takto:*

1. $W^0 = 0$,
2. $W^k = W^{k-1} + d_k x_k^T$, *pro* $(k = 1,..., n)$.

Pokud budou vektory učebního souboru ortonormální, tedy bude platit:
$x_i^T x_j = 0$ *pro* $i \neq j$ *a současně* $x_i^T x_j = 1$ *pro* $i = j$, *tak v tomto případě, pokud na vstup naučené sítě vložíme jeden ze vstupních vektorů* $\boldsymbol{x_r}$, *se objeví na výstupu sítě s ním v učebním souboru asociovaný vektor* $\boldsymbol{d_r}$. *Toto chování sítě lze využít k vytvoření asociativní paměti.*

U lineárních asociativních neuronových sítí se můžeme setkat ještě s několika speciálními sítěmi. Je to tzv. heteroasociativní síť a autoasociativní síť. Heteroasociativní síť vzniká, když asociujeme vektory $\boldsymbol{x_k}$ *a vzniká nám při tom tento vztah* $d_k \neq x_k$. *Autoasociativní síť vzniká, když asociujeme vektory* $\boldsymbol{x_k}$ *samy se sebou, tedy vzniká nám vztah* $d_k = x_k$.

Učení lineárních asociativních neuronových sítí náleží do kategorie učení bez učitele. Naučit a odnaučit síť další asociaci je jednoduché, děje se na základě přičítání (naučení asociace) a odečítání (odnaučení asociace) matic. Velkou nevýhodou asociativní sítě je příliš silný požadavek na bezchybné vybavování sítě, tj. požadavek ortonormality vstupních vektorů učebního souboru.[11]

3.4.2.4 Hopfieldova síť

Hopfieldovy sítě se zařazují do architektury cyklických sítí, a proto při operacích s těmito sítěmi musíme brát v úvahu čas. Hopfieldovy sítě jsou tvořeny lineárními neurony. Architektura Hopfieldovy sítě je tvořena neurony tak, že jsou spojeny se vstupy všech ostatních neuronů. Výstup vlastního neuronu již jako vstup pro ten samý neuron nevzniká. Spoje mezi jednotlivými neurony jsou symetrické, tudíž zde můžeme najít tento vztah $\mathbf{w_{ij} = w_{ji}}$. Tento vztah nám říká, že výstup i-tého neuronu se vstupem j - tého neuronu se rovná výstupu j-tého neuronu do vstupu i-tého neuronu. Síť můžeme jako v předešlých případech neuronových sítí znázornit maticí. Pokud neuronová síť bude mít **n** neuronů, pak výsledná konfigurační matice **M** bude typu **n** x **n,** kde na diagonále budou samé 0. Chování jednotlivých neuronů oproti předešlým uvedeným případům je však jiné, jelikož budeme u těchto neuronů muset brát v úvahu faktor času. Chování neuronů Hopfieldovy sítě lze popsat následujícím vztahem:

$$h_k(t) = \sum_{i=1}^{n} y_i(t) w_{ki}(t),$$

$y_k(t + 1) = sign(h_k(t))$, *pro* $h_k(t) \neq 0$,

$y_k(t + 1) = y_k(t)$, *pro* $h_k(t) = 0$,

kde $\boldsymbol{h_k(t)}$ *je postsynaptický potenciál k-tého neuronu v čase* ***t,*** $\boldsymbol{y_k(t)}$ *je výstup k-tého neuronu v čase* ***t*** *a* $\boldsymbol{w_{ki}(t)}$ *je váha i-tého neuronu v čase* ***t.*** *Vstupy neuronů jsou označeny jako* y_i*, změna tohoto značení je dána tím, že jednotlivé výstupy neuronů jsou zároveň vstupy pro neurony ostatní. V některých případech se můžeme setkat, že neurony mají navíc ještě excitační práh, potom popis jejich chování bude následující:*

$y_k(t + 1) = sign(h_k(t - \vartheta))$, *pro* $h_k(t) \neq \vartheta$,

$y_k(t + 1) = y_k(t)$, *pro* $h_k(t) = \vartheta$. *viz. [12]*

Vlastnosti chování neuronů při použití excitace jsou podobné jako chování neuronů, když excitace použita není.

Učení, někdy uváděné pod pojmem relaxace sítě, probíhá tak, že na počátku jsou sítí vnuceny hodnoty. Následně pak již síť jiné hodnoty z vnějšího prostředí nepřijímá a výpočty jsou prováděny v rámci sítě izolovaně. Stavy neuronů se začnou měnit v diskrétních časových okamžicích **(t, t + 1, ... , t + m).** Tyto stavy se mohou měnit

dvojím způsobem. Buď se v daném okamžiku změní stavy všech neuronů, pak se daný jev nazývá synchronní relaxace sítě, nebo se v daném okamžiku změní stav pouze jednoho neuronu, pak se daný jev nazývá asynchronní relaxace sítě.

Při relaxaci sítě dochází ke změně pouze výstupů jednoho nebo všech neuronů v ohledu na relaxaci sítě. Dále předpokládáme,že konfigurace sítě ***M*** *se nemění. Hopfield při svém výzkumu cyklických neuronových sítí definoval ještě jeden důležitý parametr a tím je energetická hodnota funkce* ***E(y).*** *Energetická hodnota funkce určuje energii (potenciál), která přísluší ke každému stavu sítě v daném časovém okamžiku. Výpočet energetické hodnoty sítě je následující:*

$$E(y) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}y_i y_j$$

Z tohoto výrazu může odvodit jednu důležitou větu a to:

Při asynchronní relaxaci sítě je její (potenciál) energie sítě klesající funkcí v závislosti na čase tj. platí E(y(t + 1) < E(y(t)).

Při synchronní relaxaci sítě však energie neuronové sítě klesat nemusí a neuronová síť může oscilovat mezi dvěma stavy. [12]

Relaxaci sítě můžeme sledovat vždy v určitém časovém okamžiku a v určitém stavu. Během relaxace sítě se tento stavový vektor sítě **y(t)** pohybuje od počátečního stavu $\mathbf{y(t_0)}$ až po konečný stav $\mathbf{y(t_m)}$ a v prostoru vytváří určitou trajektorii relaxace sítě. Koncový stav sítě se nazývá atraktorem. Pokud budeme uvažovat o asynchronní relaxaci sítě, kdy energie v každém dalším časovém okamžiku **t** klesá, pak v konečném časovém okamžiku $\mathbf{t_m}$, kdy dosáhneme koncového stavu sítě $\mathbf{y(t_m)}$, budou atraktory představovat stavy s nízkou energetickou hodnotou. Když vhodně nakonfigurujeme síť tak, aby určitý stav sítě $\mathbf{y(t_m)}$ byl atraktorem sítě, potom daná síť bude z různých počátečních stavů $\mathbf{y(t_0)}$ končit ve stavu $\mathbf{y(t_m)}$**.** Této vlastnosti lze využít při navrhování asociativní paměti, pokud vektory, které si má síť zapamatovat, budou atraktory sítě.

Další využití Hopfieldových sítí je v optimalizačních úlohách, ve kterých se hledá minimum určité funkce, jelikož Hopfieldovy sítě se při relaxaci chovají podobně jako sítě, které využívají gradientní metody. Jako příklad praktického využití Hopfieldovy sítě v optimalizačních úlohách je úloha obchodního cestujícího.

3.4.2.5 Kompetiční sítě

Kompetiční sítě jsou speciální neuronové sítě, které jsou tvořeny dvěma vrstvami neuronů. První vrstvu tvoří receptory, které jsou propojeny s každým výstupním neuronem. Výstupní neurony jsou tvořeny lineárními neurony postrádajícími fiktivní vstup. Kompetiční sítě se odlišují od ostatních výše uvedených neuronových sítí tím, že je ve výstupní vrstvě aktivní pouze jeden neuron a to ten, který má nejvyšší hodnotu excitace. Na základě tohoto chování jsou někdy tyto sítě nazývány jako sítě se sobeckými neurony nebo také sítě typu „vítěz bere vše".

Relaxace těchto sítě probíhá takto:

1. *Na vstup sítě vložíme vektor* $\boldsymbol{x}$.
2. *Vyhodnotí se postsynaptický potenciál každého výstupního neuronu, tj. pro každý výstupní j-tý neuron se vypočte* $\boldsymbol{xw_j}$.
3. *Zaktivuje se ten výstupní neuron, jehož postsynaptický potenciál je největší, tj.* $\boldsymbol{xw_j \geq xw_n}$ *, pro všechna* $\boldsymbol{j \neq n}$. *Pokud je podmínka splněna pro více neuronů, je zaktivován ten neuron, jehož index je nejmenší. Současně je tedy zaktivován pouze jeden neuron, který se též někdy nazývá jako vítěz (winner) a ostatní výstupní neurony jsou inhibovány (zneaktivněny).*

Problém s takto popsanou sítí je, že nesplňuje obecné paradigma neuronových sítí. Existuje však způsob, jak kompetiční síť zapsat, aby obecné paradigma bylo splněno. Kompetiční síť můžeme popsat jako cyklickou neuronovou síť, jejíž výstupní vrstvy jsou typu $\boldsymbol{y = f(xw)}$, *kde* $0 \leq \boldsymbol{f(x)} \leq 1$ *a kde* $\boldsymbol{f(x)}$ *je sigmoidální funkce. Výstupní neurony nebudou obsahovat fiktivní vstup a budou tvořit navíc mezi sebou inhibiční spojení. Tato inhibiční spojení budou vyjádřena zápornými váhami. Dále pak budou výstupní neurony spojeny se svými vstupními neurony excitačními spoji* $\boldsymbol{v_{ij}}$, *které jsou vyjádřeny kladnými hodnotami. Jelikož se jedná o cyklickou neuronovou síť, musíme při operacích s touto sítí počítat s faktorem času vyjádřeným v diskrétních časových intervalech.*

Výpočet takto navržené kompetiční sítě, bude probíhat následujícím způsobem:

1. *V čase* $\boldsymbol{t}$ *vložíme na vstup sítě vektor* $\boldsymbol{x}$ *a vypočteme hodnoty neuronů výstupní vrstvy.*

2. *Odebereme vektor* $\boldsymbol{x}$ *ze vstupu tím, že výstupy receptorů nastavíme na 0.*
3. *V dalších časových okamžicích* $\boldsymbol{t+1, ..., t+n}$*, síť postupně mění svůj stav.*
4. *Na konci výpočtu bude pouze jeden neuron aktivní, tj. bude mít hodnotu rovnou 1. Bude to ten neuron, pro který po vložení vstupního vektoru* $\boldsymbol{x}$ *platí, že* $\boldsymbol{xw_j \geq xw_n}$ *, pro všechna* $\boldsymbol{j \neq n}$*. Ostatní neurony mají hodnotu 0.*

Během výpočtu se excitační váhy $\boldsymbol{v_{ij}}$ *nastaví následujícím způsobem:*

$\boldsymbol{v_{ij} = 1}$*, pro* $\boldsymbol{i = j}$*,*

$\boldsymbol{v_{ij} = -\vartheta}$*, pro* $\boldsymbol{i \neq j}$*,* $\vartheta < \frac{1}{m}$*, kde* $1 \leq \boldsymbol{i, j} \leq \boldsymbol{m}$ *a kde* $\boldsymbol{m}$ *- je počet výstupních neuronů.*

Při takto zadaném výpočtu bude síť konvergovat a na konci budeme schopni určit vítězný neuron.

V praktických příkladech se obvykle nenasazuje kompetiční síť jako cyklická neuronová síť, ale předešlý ekvivalent. Kompetenční cyklická síť má význam především při fyzikální realizaci této sítě v elektronické podobě.[13]

Kompetenční sítě se používají pro automatickou klasifikaci a jsou využívány pro shlukové analýzy. U těchto sítí je využito té vlastnosti, že daná síť dokáže zastupovat jednotlivé kategorie jedním neuronem. Tohoto výsledku se docílí naučením sítě tak, že daná síť se učí zařazovat vstupní vektory do jednotlivých kategorií tak, aby jednotlivé prvky v rámci jedné kategorie si byly navzájem podobné. Učení, jak je z popisu patrné, probíhá na základě učebního souboru, který obsahuje pouze vzory jednotlivých kategorií a jedná se proto o učení bez učitele.

3.4.2.6 Kohonenovy samoorganizační mapy

Kohonenovy samoorganizační mapy vznikly na počátku 80. let 20. století, ale jejich první koncepty můžeme datovat do 70. let 20 století, autorům von der Malsburga a Amariho. Kohonenovy mapy jsou v některých bodech podobné kompetičním sítím. Kohonenova síť je tvořena vrstvou receptorů a vrstvou výstupních neuronů. Rozdíl od kompetičních sítí je v tom, že u Kohonenovy mapy jsou výstupní neurony uspořádány do n-rozměrného pole. Zpravidla se jedná o dvourozměrné pole. Každý výstupní neuron je spojen s každým receptorem. Výstupní neurony mají charakteristiku

sobeckých neuronů. Z toho plyne, že aktivní může být v určitém okamžiku pouze jeden neuron a to ten, jehož postsynaptický potenciál je nejvyšší. Výpočet Kohonenovy sítě probíhá stejným způsobem jako výpočet sítě kompetiční. Základním principem Kohonenových map je zobrazení vstupních vektorů do množiny neuronů.

Hlavním účelem Kohonenových map je zachytit při zobrazení topologii a pravděpodobnostní distribuci vstupních dat při zachování následujících vlastností:

1. Jsou-li vstupní vektory $\mathbf{x_1}$ a $\mathbf{x_2}$ podobné, tj. jestli platí, že $\| \mathbf{x_1} - \mathbf{x_2} \|$ je malé číslo, pak také neurony $\mathbf{n_1}$ a $\mathbf{n_2}$, na které se vstupní vektory $\mathbf{x_1}$ a $\mathbf{x_2}$ zobrazí, musí být blízko sebe.
2. Jestliže oblast $\mathbf{X_1}$ hodnot vstupních vektorů se vyskytuje na vstupu sítě s větší pravděpodobností než vektory jiné oblasti $\mathbf{X_2}$. Potom taková oblast $\mathbf{X_1}$ musí být reprezentována větším počtem neuronů než oblast $\mathbf{X_2}$.

Na základě znalostí, které máme o rozložení dat, můžeme ke konfiguraci sítě přistupovat dvěma způsoby. Pokud známe pravděpodobnostní rozložení dat v souboru, pak můžeme konfiguraci sítě provést analytickými metodami. Pokud ale toto pravděpodobnostní rozložení neznáme, je nutné, aby si síť toto pravděpodobnostní rozložení určila sama v procesu učení. Učební soubor, tak jako u kompetičních sítí, bude obsahovat pouze vstupní vektory, a proto se bude jednat o učení bez učitele. Algoritmus adaptace vah při konfiguraci Kohonenovy sítě, v případě učení bez učitele, zavedl Kohonen v 80. letech 20. století.

Algoritmus učení pro Kohonenovu mapu bude následující:

Předpokládejme, že budeme pracovat s neurony, které budou organizovány do dvourozměrného pole $\mathbf{n_{ij}}$. Pro další kroky výpočtu je nutné zjistit a definovat vzdálenost dvou výstupních neuronů a okolí výstupního neuronu. Musíme určit tzv. sousední neurony. Sousední neurony jsou takové neurony, které se liší v indexu o 1 s indexem výstupního neuronu. Pokud propojíme všechny sousední neurony, získáme dvourozměrnou mřížku. Při takto sestavené Kohonenově mapě můžeme dále určit vzdálenost dvou neuronů **i** a **j**. Za vzdálenost **d(i,j)** mezi neurony **i** a **j**, bude považován minimální počet hran mřížky, které je třeba propojit, abychom se dostali z neuronu **i**

do neuronu **j**. Dále pak můžeme určit okolí neuronu $\mathbf{N_s(c)}$, kde **c** je velikost a **s** bude znamenat množinu všech neuronů **j,** jejichž vzdálenost do neuronu $\mathbf{c \leq s}$.

1. *Váhy* $\boldsymbol{w_{ij}^0}$ *zvolíme jako malé hodnoty blízké 0.*
2. *V n-tém kroku změníme váhové vektory tak, aby platilo:*

$|| x^n - w_i^{n-1} || \leq || x^n - w_j^{n-1} ||$ *, pro všechna* $i \neq j$.

3. *Pokud je splněn bod 2., tak získáváme i-tý neuron, který zvolíme jako aktivní.*
4. *Budeme adaptovat všechny váhové vektory všech neuronů* $\boldsymbol{k}$ *z jeho okolí* $\boldsymbol{N_{s(n)}(i)}$*, tímto způsobem:*

$\boldsymbol{w_k^n} = w_k^{n-1} + \alpha(n)(x^n - w_k^{n-1})$*, kde* $\alpha(n)$ *je funkce klesající s počtem kroků.*

5. *Jestliže jsme vyčerpali předem zvolený počet kroků, ukončíme algoritmus. Pokud tato podmínka splněna není, přejdeme k bodu 2.*

V některých případech, jako je rozpoznávání obrazců a písma, je nutné používat u Kohonenových map učení s učitelem, tj. každému předloženému vektoru $\boldsymbol{x_i}$ *je předložen odpovídající výstup* $\boldsymbol{d_i}$*. [13]*

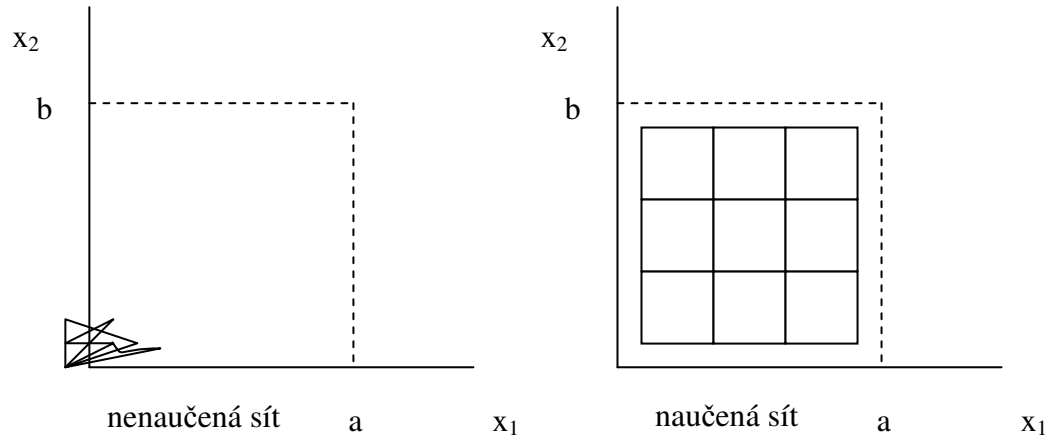
Učení Kohonenovy mapy bude probíhat v následujících krocích:

1. Nejprve proběhne učení bez učitele podle algoritmu uvedeného výše.
2. Označí se výstupní neurony kategoriemi.
3. Doučení sítě:

- **první fáze:** učení sítě probíhá podle standardního algoritmu pro Kohonenovy samoorganizační mapy, kdy jsou síti předkládány pouze vektory $\mathbf{x_i}$. Na tomto základě se vytvoří **m** typických reprezentantů $\mathbf{w_i}$.
- ***druhá fáze:*** *přiřadíme reprezentantům* $\boldsymbol{w_i}$ *kategorie* $\boldsymbol{C_1, \ldots, C_n}$*, toto přiřazení se označí jako* $\psi(w_i)$*. Přiřazení se provádí tak, že projdeme celý učební soubor a zaznamenáme, kolik prvků daný reprezentant* $\boldsymbol{w_i}$ *zastupuje.*
- ***třetí fáze:*** *podle algoritmu LVQ provedeme doučení sítě. LVQ algoritmy jsou navržené Kohonenem a fungují na principu, že v každém kroku po předložení vektoru* $\boldsymbol{x_i}$ *je modifikován pouze jeden váhový vektor. Jestliže je vektor* $\boldsymbol{x_i}$

nejblíže k váhovému vektoru, který reprezentuje danou kategorii, je váhový vektor posunut k vektoru $\boldsymbol{x_i}$. *Pokud je ale vektor* $\boldsymbol{x_i}$ *nejblíže váhovému vektoru, který reprezentuje jinou kategorii, než do které vektor* $\boldsymbol{x_i}$ *patří, je váhový vektor této jiné kategorie odsunut směrem od vektoru* $\boldsymbol{x_i}$. *[13]*

obrázek 6: Učení Kohonenovy sítě



3.5 Použití neuronových sítí

Neuronový přístup je charakteristický následujícími rysy:

- schopnost učení,
- schopnost abstrakce,
- paralelní zpracování,
- robustnost.

Tyto základní rysy neuronového přístupu jsou velkou výhodou a umožňují nasazení neuronového přístupu v mnoha oborech jako:

- syntéza řeči,
- rozpoznání řeči,
- rozpoznání obrazců,
- řízení složitých zařízení,
- predikce časových řad,
- komprese dat,
- expertní systémy.

Neuronové sítě se v praxi používají tam, kde je složité, (v některých případech ani není explicitně možné), odvodit algoritmus chování systému. Na druhou stranu tam, kde je znám efektivní algoritmus je zbytečné a často i kontraproduktivní nasazovat nějaký druh neuronové sítě. V případech, kdy je znám efektivní algoritmus, je výhodnější použít počítače s klasickou von Neumannovskou architekturou. Jednou z dalších velkých nevýhod, které brání ještě masivnějšímu rozšíření neuronových sítí do praxe, je vlastní nedostatek těchto sítí. Tento nedostatek je v tom, že sítě nám poskytnou správné řešení, ale nejsou schopny nám zároveň předložit zdůvodnění svého řešení. V některých oborech toto zdůvodnění nemá velký význam, ale jsou obory jako např. lékařství, kde sama podstata problému klade důraz na to, proč jsme toto řešení předložili jako správné.

3.5.1 Syntéza řeči

Tato oblast patří mezi první oblasti, kde byla úspěšně nasazena aplikace, pracující na neuronovém přístupu. První taková aplikace byla NETtalk vyvinutá Sejnovským a Rosenbergem. Pracovala na principu algoritmu zpětného šíření chyby a jednalo se o vícevrstvou neuronovou síť s jednou skrytou vrstvou. Síť obsahovala 203 receptorů, 80 neuronů bylo umístěno do skryté vrstvy a 29 neuronů tvořilo výstupní vrstvu. Jejím úkolem bylo naučit se vyslovovat psaný anglický text. Tento úkol znamenal, že na základě znalosti písmen se musela síť naučit přiřazovat těmto písmenům fonémy. Síť rozeznávala 29 znaků, z čehož bylo 26 písmen, dále pak mezeru a dva interpunkční znaky. Výsledkem sítě byl kód fonému, který byl veden do syntezátoru. V celkovém souhrnu dosáhla síť 95 % úspěšnosti při vyslovování znaků na učebním souboru a 80 % úspěšnosti při použití na testovacím souboru.

3.5.2 Rozpoznání řeči

V této oblasti největší úspěch z neuronových sítí zaznamenaly Kohonenovy sebeorganizační mapy. Za jednu z prvních aplikací v této oblasti, pracující na principu Kohonenových sebeoragnizačních map, je považována aplikace na rozeznávání finštiny. Tato aplikace, která zaznamenala i úspěšnou realizaci, byla rozdělena do několika fází:

- odfiltrování šumu,

- převod analogického signálu do digitální podoby,
- Fourierova transformace – přetransformování řečového signálu do frekvenčních oblastí,
- získané vektory z Fourierovy transformace jsou vedeny na vstup Kohonenovy sebeorganizační mapy,
- podle heuristických pravidel je výstupu z Kohonenovy mapy přiřazen psaný text.

Aplikace zaznamenala při přepisu pro plynulou řeč s neomezeným slovníkem přesnost 92 – 97 %. Přesnost přepisu pro izolovaná slova a s omezeným slovníkem s počtem 1000 slov, dosahovala síť přesnosti 96 – 98 %. Aplikace byla zkoušena za podmínek zachování stejného mluvčího. Kdybychom uvažovali o změně mluvčího, museli bychom celou aplikaci znovu naučit.

3.5.3 Rozpoznávání obrazců

V této oblasti bylo zaznamenáno jedno z prvních použití neuronových sítí. V případě rozpoznávání obrazců se zpravidla jedná o dvourozměrné obrazce, zaznamenané na dvourozměrné mřížce. K tomuto úkolu se nejčastěji používají vícevrstvé neuronové sítě s využitím algoritmu zpětného šíření chyby. Na základě vícevrstvých neuronových sítí, v této oblasti, vznikají aplikace na rozpoznávání tištěného nebo rukou psaného slova, automatické čtení směrových čísel na dopisních obálkách, ověřování správnosti podpisu na šecích, identifikace objektů získaných z radarů a automatické vyhodnocování EKG a krevního tlaku.

3.5.4 Řízení složitých zařízení

V této oblasti se uplatňují neuronové sítě tam, kde je vzhledem ke složitosti systému odvození výsledného algoritmu složité nebo v některých případech i nemožné. Dobré uplatnění zde našly vícevrstvé neuronové sítě na principu algoritmu zpětného šíření chyby. Jedna z nejznámějších a nejúspěšnějších aplikací pracující na principu vícevrstvé neuronové sítě je aplikace ALVINN, která byla vyvinuta počátkem 90. let 20. století. Aplikace má za úkol řídit vozidlo v běžném dopravním provozu. Skládá se ze dvou systémů, z nichž první a nejdůležitější část tvořila vícevrstvá neuronová síť s jednou

skrytou vrstvou. ALVINN ujel při testovací jízdě, bez zásahu řidiče, v plném provozu 21,2 míle při průměrné rychlosti 55 mil/hodinu.

Neuronový přístup je dále často nasazován v oblastech robotiky, jako např. ve výrobních procesech pro řízení pohybu výrobních robotů.

3.5.5 Predikce časových řad

V oblasti predikce časových řad je nasazení neuronového přístupu asi nejčastější. U predikce časových řad chceme dosáhnout předpovědi časového vývoje nějaké veličiny, např. ekonomické (ceny akcií, hrubého národního produktu atd.) a dále se používá k předpovědi počasí, spotřeby elektrické energie, plynu atd. U tohoto neuronového přístupu se nejčastěji používají vícevrstvé neuronové sítě s jednou nebo dvěma skrytými vrstvami a síť se adaptuje pomocí algoritmu zpětného šíření chyby. U predikce časových řad lze také použít klasické statistické metody a jejich výsledky vychází stejně, nebo o něco hůře, než výsledky za použití neuronových sítí.

3.5.6 Komprese dat

Nasazení neuronového přístupu v této oblasti je jednoduchý. Používají se zde nejčastěji vícevrstvé sítě s jednou skrytou vrstvou. Skrytá vrstva má za úkol komprimovat data, při čemž vstupní vrstva slouží pouze pro vstup dat a výstupní již dekóduje data předaná skrytou vrstvou. Jednou z úspěšných aplikací nasazených v této oblasti byla aplikace na komprimaci digitálního obrazu. Bylo zde dosaženo komprese 4:1. Zajímavostí bylo, že síť byla původně naučena pouze na jednu fotografii lidské tváře. Když autoři posílali i jiné portréty, síť nevykazovala viditelné zhoršení přenášených fotek oproti předchozí variantě s jednou fotografií.

3.5.7 Expertní systémy

Expertní systémy, v nichž se využívá neuronový přístup, se nazývají expertní systémy s neuronovou architekturou. Neuronový přístup nabídl v oblasti expertních systémů alternativu, jak dojít k řešení problému bez nutnosti znalosti báze znalostí, kterou jiné přístupy požadují. Neuronová síť se totiž naučí řešení problémů na předem známých řešeních a vytváří si automaticky vlastní reprezentaci problému ve fázi učení.

Neuronové sítě tak ušetří čas vytvářením báze znalostí, u které by byla nutnost spolupráce s celou řadou vysoce kvalifikovaných expertů dané oblasti. *Typickou aplikací, zastupující oblast expertních systémů byla aplikace využívaná v lékařství pro diagnostiku infarktu. V souhrnu síť diagnostikovala infarkt správně s 92 % úspěšností (lékaři s 88 %). V souvislosti s nasazením aplikace v oblasti lékařství můžeme poukázat na jednu s největších nevýhod neuronového přístupu.[14]* Neuronové sítě dokáží spolehlivě a s dostačující přesností klasifikovat a řešit dané problémy, ale jejich velkou nevýhodou je explicitní zdůvodnění svého rozhodnutí. Tato nevýhoda tak zatím braní k masivnímu rozšíření expertních systémů v oblastech jako je např. lékařství, kde je požadováno zdůvodnění výsledku.

4 Odhad složitosti softwaru s využitím umělé inteligence PETRA

S problematikou odhadu složitosti softwaru za použití neuronových sítí se autor práce poprvé seznámil na konferenci Objektů 2006, kde se setkal se svým garantem předkládané práce Ing. Josefem Pavlíčkem, Ph.D. (dále jen Pavlíček), který autorovi nabídl spolupráci na jeho navrženém programu PETRA (PErcepTRon Artifical inteligence). Program PETRA používá jako hlavní prostředek pro řešení problému neuronový přístup.

Metody používané pro odhad složitosti softwaru jsou následující:

- COCOMO,
- Funkčních jednic,
- Use Case point,
- odhad složitosti programových modulů,
- složitost na základě analýzy grafu řízení,
- složitost na základě přehledného hierarchického rozkladu,
- složitost v objektovém prostředí.

Uvedené metody již autor práce nebude dále teoreticky rozebírat, jelikož to není záměrem této diplomové práce. Metody jsou již podrobně teoreticky rozebrány v disertační práci Pavlíčka a autor uvede pouze metodu Use Case Point. Tato metoda je využívána v programu PETRA a poskytuje nám data ve fázi učení neuronových sítí programu PETRA. Metoda je citována z uvedené disertační práce, aby se vycházelo ze stejných teoretických znalostí.

4.1 Metoda Use Case Point

Kritika metody FP a jejích odvozenin vedla v roce 1993 Gustava Karnera k vyvinutí metody založené na tzv. "Use Case Points". Use Case je anglický název pro případ užití. Případ užití je způsob jak zachytit požadavek na funcionalitu budoucího systému za dodržení pravidel definovaných v metodě Unified Process. Use Case Points lze zjednodušeně přeložit jako body případů užití.

Výsledná metoda byla diplomovou prací Gustava Karnera na švédské University of Linköping. Práva k užívání a kopírování na ní vlastní firma Rational Software. Tato firma je mimo jiné aktivním rozšiřovatelem sjednoceného (unifikovaného) modelovacího jazyka UML a tvůrcem Rational Unified Process. Karner založil metodu odhadu bodů případů užití na základě předpokladu, že funkčnost systému vychází z uživatelem definovaných případů užití – Use Cases. Tyto případy užití jsou základem pro odhad velikosti informačního (nebo jiného softwarového) systému.[15]

4.1.1 Matematický model UCP

Pro výpočet odhadu složitosti na základě bodů případů užití je nejprve potřeba zjistit počet aktorů a počet případů užití. Viz Tabulka 3 je jedním z možných případů, jak funkční body případů užití počítat. Slouží jako vzorový příklad.

tabulka 1: ***Příklad případu užití***

Role	**Případ užití**	**Počet případů užití**
Administrátor	Administrace_1	1
Uživatel 2	Zadávání dat_1, Čtení sestav_1	2
Manager	Čtení sestav_1,Čtení sestav_2	1
Celkem		**4**

Pramen: PAVLÍČEK, Josef. Odhad manažerských charakteristik vývoje IS v etapě specifikace požadavků.(Doktorská disertační práce). Praha: Česká zemědělská univerzita v Praze, 2006.

V tabulce jsou uvedeni aktoři a jejich role v jednom daném ukázkovém příkladu. Popis rolí není pro výpočet bodů případů užití důležitý. Je ale důležité vědět, kolik aktorů má vazbu na jeden případ užití. Body jsou případu užití udělovány na základě počtu aktorů, které jej využívají. V našem případě má případ užití Čtení_sestav_1 dva aktory. To znamená, že je patrně složitější než ostatní případy užití. Karner navrhuje klasifikovat vznikající případy užití za pomoci ordinální stupnice s běžně využívanými hodnotícími stupni:

tabulka 2: ***Příklad přiřazení vah***

Typ případu užití	**Váha w**
Jednoduchý	1
Středně složitý	3
Složitý	6

Pramen: PAVLÍČEK, Josef. Odhad manažerských charakteristik vývoje IS v etapě specifikace požadavků.(Doktorská disertační práce). Praha: Česká zemědělská univerzita v Praze, 2006.

Podle "složitosti" se pak jednotlivému případu užití přidělují váhy obdobně jako na vzorovém příkladu v Tabulce 2. Dále, podobně jako v metodě FP, je nutné ohodnotit jednotlivé dílčí technické faktory prostředí od 0 – nemá vliv, do 5 – má zásadní vliv. Po dosazení do obdobných vzorců získáme počet bodů případu užití. Karner dále navrhuje zavést míru – pracnost v člověkohodinách na jeden bod případu užití. Karner doporučuje tuto míru rovnou 20. Z vlastních zkušeností při vedení projektů i dle zahraniční literatury navrhuji zvolit hodnotu v intervalu <15, 25> na jeden bod případu užití. Nakonec je potřeba vynásobit počet bodů případu užití zvolenou mírou a sečíst výsledky. Vyjde celková odhadovaná složitost pro softwarový produkt.[16]

4.2 Topologie neuronové sítě použité v programu PETRA

PETRA (PErcepTRon Artifical inteligence) je program, který využívá perceptronovou neuronovou síť. Neuronová síť se skládá ze vstupní vrstvy tvořené receptory, které přijímají vstupní vektor **x.** Neuronová síť dále obsahuje skrytou vrstvu, kterou tvoří dvojnásobný počet neuronů, než je ve vrstvě vstupní. Poslední vrstvu představuje jeden neuron, který reprezentuje výsledek dané sítě a využívá sigmoidální funkce. Neuronová síť PETRA se učí podle algoritmu zpětného šíření chyby navrženého vědeckým týmem vedeným Rumelhartem.

4.3 Základní požadavky na neuronovou síť v programu PETRA

Základní požadavky na neuronovou síť vycházejí z požadavků, pro které je neuronová síť určena. Jelikož síť je používána pro odhad složitosti softwaru za použití metody Use Case Point a obsahující další faktory navržené Pavlíčkem, které tvoří vstupní vektor sítě, jsou minimální předpoklady pro architekturu sítě následující:

- 30 neuronů ve vstupní vrstvě,
- skrytá vrstva bude obsahovat 60 neuronů,
- jeden neuron bude reprezentovat výstupní vrstvu,
- před zahájením učení budou váhy sítě malá čísla blízké 0.

Důležitým bodem, který by měla síť splňovat, je její konfigurovatelnost vstupních neuronů. Tento požadavek je odvozen z důvodu toho, aby mohla být síti rozšířena vstupní vrstva o další neurony a tím umožnila rozšíření o další faktory vystupujících ve fázi učení sítě.

4.4 Znalostní báze pro program PETRA

Znalostní báze neuronové sítě v programu PETRA tvoří vstupní vektor **x** a její požadovaný výstup tvoří vektor **d**. Vektor $\mathbf{x_n}$ popisuje jednotlivé realizované softwarové produkty. Jednotlivé složky vektoru $\mathbf{x_n}$ jsou tvořeny údaji získanými z metody Use Case Point a dalšími faktory navrženými Pavlíčkem.

Vstupní vektor je tvořen následujícími údaji:

- *ZSP* – 1 položka vektoru,
- *FSk* - 10 položek vektoru,
- *FSo* - 19 položek vektoru.

ZSP - základní složitost softwarového produktu, která se získá jako:

$ZSP = \sum_{i=1}^{n} ZS_i$, kde **ZS** je základní složitost diagramu užití.

Základní složitost diagramu užití (**ZSP**) vychází z diagramu užití. Diagram užití slouží k tomu, abychom jednoduchou formou popsali důležité a podstatné požadavky

na systém a jeho vztahy k uživatelům. Jelikož zpravidla většina produktů nebývá tak jednoduchá, aby nám stačilo k jejich popsání jednoho grafického diagramu užití, musíme si vypomoci proměnnou **ZSP**, která tento souhrn vyjádří.

FSk - faktory složitosti klasifikovatelné. Tento pojem zavedl Pavlíček ve své disertační práci. Tato proměnná má za úkol určitým způsobem klasifikovat, jaký vliv má na diagram užití typ aktorů, počet jeho vazeb a počet případů užití konkrétního aktoru. Aktor můžeme chápat jako určitou entitu v systému (uživatel, databáze atd.).Vazba je spojnice mezi aktorem a případem užití. Případ užití můžeme chápat jako interakci, např. přidej uživatele, vyhledej uživatele atd., mezi dvěma a více aktory. Ve zvláštním případě může teoreticky případ užití vzniknout i mezi jedním aktorem, který by musel obsahovat dvě vazby na stejný případ užití. Faktor složitosti klasifikovatelný se snaží přiřadit (klasifikovat) aktoru, podle jeho známých vazeb a případů užití, určitou roli. Aktory klasifikované stejnou rolí jsou si navzájem podobné. Všechny role, které byly Pavlíčkem navrženy nesou souhrnně označení - **Manažerské charakteristiky zadání požadavků na IS.**

Manažerské charakteristiky zadání požadavků na IS popisuje následující tabulka:

tabulka 3 : Manažerské charakteristiky zadání požadavků na IS

Třída rolí aktorů		
Číslo role	Jméno role	Popis role (jednotlivé role je možné upravit dle potřeby vznikajícího softwarového produktu)
1	Uživatel	Tato role je přiřazena každému aktoru, který má v systému základní práva typu - číst veřejně dostupné dokumenty, vyhledávat nad nimi atd. Zástupce této role je například běžný uživatel operačního systému Windows bez práv instalace programů na disk s předpřipravenými programy k použití.
2	Uživatel s rozšířenými právy	Tato role je přiřazena každému aktoru, který má povolený částečný nebo úplný přístup k datům, která vyžadují nějakou autorizaci. Zástupce této role je například uživatel operačního systému Windows, který vlastní přístupová práva k některým programům, jako je elektronická pošta atd.

3	Administrátor s omezenými právy	Tato role je přiřazena každému aktoru, který má povoleno udržovat a spravovat vybranou část informačního systému. Nemá však práva administrovat celý informační systém. Zástupcem této role je například uživatel systému Windows, který má práva instalovat soubory na disk nebo přidávat jiné uživatele pod svůj uživatelský účet. Zde může nastavovat přístup jiným uživatelům nižší nebo stejná přístupová práva jako má. on.
4	Administrátor	Tato role je přiřazena každému aktoru, který má absolutní práva správy informačního systému. Zástupce této role je například Administrátor v operačním systému Windows, který má úplná práva číst, mazat, zapisovat a přidělovat práva ostatním uživatelům v celém operačním systému.
5	Relační databáze	Tato role je přiřazena každému programu - aktoru, který zastupuje relační systém správy dat.
6	Jiný druh databáze	Tato role je přiřazena každému programu – aktoru, který zastupuje jiný než relační systém pro správu dat.
7	File systém	Tato role je přiřazena každému programu – aktoru, který pracuje (otevírá, přepisuje, ukládá, maže) se soubory ve stromové struktuře.
8	Jiný program	Tato role je volně využitelná pro specifické účely vznikajícího softwarového produktu a je jí přiřazen jakýkoliv jiný program – aktor, kterému nelze přiřadit některou z navržených rolí.
9	Operační systém	Tato role zastupuje program – aktora, který je operačním systémem.
10	Nespecifikovaná role	Tato role je volně využitelná pro specifické účely vznikajícího softwarového produktu. Je možné ji využít dle potřeby odhadu.

Pramen: PAVLÍČEK, Josef. Odhad manažerských charakteristik vývoje IS v etapě specifikace požadavků.(Doktorská disertační práce). Praha: Česká zemědělská univerzita v Praze, 2006.

FSo – faktory složitosti obtížně klasifikovatelné. Tento pojem také vychází z disertační práce Pavlíčka. **FSo** popisuje faktory, které se podílejí na celkové složitosti produktu, ale které jsou z pohledu firmy těžko klasifikovatelné. Tyto faktory nemůžeme vyčíst z diagramu užití a mohou to být faktory jako:

- prostředí firmy, která projekt realizuje,
- typ softwaru, který vzniká,
- použitý programovací jazyk,
- druh vývojářského týmu,
- požadavek na stabilitu vyvíjeného softwaru.

Pro tyto faktory je důležité zavést také určitou třídu, kterou Pavlíček nazývá **Manažerské charakteristiky prostředí vývoje IS.** Jednotlivé prvky této třídy pak budou zastupovat jeden charakteristický faktor obtížně klasifikovatelný. U těchto faktorů je problém v tom, že každá firma může tuto třídu vnímat jinak. K tomuto problému lze přistupovat několika způsoby a záleží na konkrétním užití PETRY v daném prostředí. Jednotlivé faktory ovlivňuje nejen množstvím informací, které má firma k dispozici o daném faktoru, ale také realistické návrhy jednotlivých faktorů firmou. Jako univerzální variantu FSo navrhuje Pavlíček použít následující třídu:

tabulka 4: Třída faktorů FSo

Třída faktorů FSo	
Číslo faktoru	Název faktoru
Faktory lidského typu	
1	Analýzu prováděl zkušený analytik
2	Tým má zkušenosti s podobným projektem
3	Kapacita týmu pro realizaci požadavků je dostatečná
4	Programátoři umí programovat v daném programovacím jazyce
5	Fluktulace jednotlivých členů je značná
6	Členové týmu mají dostatek zkušeností s potřebnými nástroji
Faktory produktového typu	
7	Požadavek na spolehlivost software je vysoký
8	S použitou databází jsou dostatečné zkušenosti
9	Požadavek na znuvupoužitelnost softwaru je vysoký

10	Požadavek na programovou dokumentaci je vysoký
11	Požadavek na rychlost odezvy je vysoký
Faktory nasazení	
12	Software musí zabírat co nejméně místa na discích
13	Software musí zabírat co nejméně místa v paměti
14	Operační systém, pod jehož řízením bude software nasazen, je dostatečně stabilní
15	Software musí být multiplatformní
Faktory projektu	
16	Požadavky na funkčnost softwaru se často mění
Další, nespecifikované faktory	
17	Faktor 1 (v případě potřeby lze doplnit)
18	Faktor 2 (v případě potřeby lze doplnit)
19	Faktor 3 (v případě potřeby lze doplnit)

Pramen: PAVLÍČEK, Josef. Odhad manažerských charakteristik vývoje IS v etapě specifikace požadavků.(Doktorská disertační práce). Praha: Česká zemědělská univerzita v Praze, 2006.

Faktory 1 až 16 tvoří tedy navrženou skupinu univerzálních faktorů, které mohou ovlivnit celkovou složitost při vzniku produktu. Abychom tyto faktory mohli nějakým způsobem interpretovat dané neuronové síti ve fázi učení a testování sítě, je potřeba tyto obtížně klasifikovatelné faktory správně vyjádřit. Jako jedno z možných řešení se nabízí použít ordinální stupnici navrženou Pavlíčkem. Tato ordinální stupnice je následující:

- 0 - Faktor neexistuje nebo nemá vliv.
- 1 – Faktor existuje, má střední vliv.
- 2 – Faktor existuje, jeho vliv je zásadní.

Faktory 17 až 19 slouží k volbě dalších nespecifických faktorů, které se podílejí na tvorbě výsledného produktu v konkrétní firmě a které nejsou již zahrnuty v bodech 1 až 16. Faktory 17 až 19 jsou specifické ještě tím, že mohou zastupovat takové faktory, které budou obtížně klasifikovatelné ordinální stupnicí použitou v bodech 1- 16. Pokud tato situace nastane, můžeme použít pro každý faktor jinou klasifikační stupnici, aniž bychom porušili pravidla pro odhad složitosti softwaru. Podmínkou však je, abychom vždy klasifikovali daný faktor stejnou klasifikační stupnicí a zachovali jednotnost v klasifikaci. Jako příklad využití jednoho specifického

faktoru může být označení analytika, který se podílel na vzniku jednotlivých případů užití popisujících funkční požadavky na budoucí software. Pak můžeme daný problém klasifikovat tak, že zavedeme stupnici, kde každý z analytiků dostane přiděleno své číslo.

Faktory 1 – 19 netvoří uzavřenou skupinu a pokud to situace dané firmy vyžaduje, mohou být kdykoliv rozšířeny o další faktory. Musíme však počítat s tím, že pokud tuto třídu rozšíříme o příliš velký počet faktorů, bude nám celková délka a složitost při učení sítě narůstat. V konečném výsledku však můžeme dojít k daleko přesnějším odhadům. V případě opačném, tedy kdybychom prvky odebírali, může nastat situace opačná. V této fázi záleží tedy na tvůrci znalostí báze, jak zvolí velkou skupinu **FSo** a jaké faktory do ní zařadí.

4.5 Celková složitost softwarového produktu

Výpočet celkové složitosti softwarového produktu je problematický. Je to z důvodu, že nemůžeme přesně určit stálou funkci, která by nám celkovou složitost softwarového produktu popisovala. Důvodem je, že daná funkce je funkcí základní složitosti softwarového produktu **ZSP** a faktorů složitosti **FS**. Jelikož z výše uvedených kapitol víme, že určení některých faktorů a jejich klasifikace záleží na tvůrci znalostí báze, můžeme také z tohoto důvodu odvodit, proč je tak složité nalézt obecně platný vzorec celkové složitosti softwarového produktu. Jako vhodné řešení se nám zde nabízí možnost k získání funkce celkové složitosti softwarového produktu (**CSP)** využít neuronovou síť PETRA. Tato funkce se získá tak, že budeme chápat celkovou složitost softwarového produktu jako funkci **CSP = f(ZSP,FS).** Pak sestavení funkce pomocí neuronové sítě bude probíhat takto:

1. Nastavíme neuronové síti váhové složky $\mathbf{w_{ij}}$ jako náhodná čísla blízké 0.
2. Ve fázi učení sítě ji budeme předkládat z učebního souboru vždy vektor vstupních dat obsahující složky **ZSP, FSk** a **FSo.** Tyto složky budou vždy jednotné pro celý učební soubor, tedy nebude se měnit pořadí faktorů ani jejich způsob klasifikace pro jednotlivé vstupní vektory.

3. Učení bude probíhat způsobem učení s učitelem (supervised learning) a neuronové síti budou předkládány vstupní vektory v cyklech.
4. Učební fáze bude trvat tak dlouho, dokud se nevyčerpá počet učebních cyklů nebo pokud nedojde ke splnění podmínky naučení sítě. Učební cykl je v souvislosti s použitím neuronové sítě PETRA chápán jako předložení všech vstupních vektorů z učebního souboru síti .

4.6 Realizace programu PETRA

Program PETRA mohl navázat již na základní program, který navrhl Pavlíček jako součást disertační práce. Tento program byl napsán v Javě a již obsahoval třídy pro vytvoření neuronů a třídu obsahující učební algoritmus. Program neměl hotové GUI - grafické uživatelské rozhraní (Graphical User Interface) a komunikovalo se s ním pomocí řádkového výstupu, které dovolovalo programovací prostředí. Po domluvě s garantem práce měl autor volnost ve výběru programovacího jazyka a společně se došlo k závěru, že pro kvalitní zpracování úkolů a jejich lepší pochopení by bylo vhodné, aby autor vytvořil svou vlastní verzi programu PETRA od začátku. Toto rozhodnutí se v budoucnu projevilo jako správné, jelikož náročnost úkolů vyžadovala plnou orientaci v kódu programu a plné pochopení metody Zpětného šíření chyby (Backpropagation of Error).

4.7 Volba programovacího prostředí

Prvním a nezanedbatelným úkolem, který autor práce musel řešit, byl výběr programovacího prostředí. Autor si uvědomoval, že v současné době je k dispozici celá řada programovacích jazyků, které pracují na rozdílných programovacích přístupech. Na konec si autor zvolil programovací jazyk Java v programovacím prostředí NetBeans a to z několika důvodů.

Neuronová síť je složena ze dvou a více neuronů, jak již bylo uvedeno v kapitolách o neuronových sítích. Podstatné na tomto faktu je to, že každý neuron lze chápat jako jedinečný objekt, který má několik předem daných vlastností (vlastní jádro, vstupy a výstupy, atd.). Neuron jako objekt je pak spojen s dalšími neurony (dalšími objekty)

a podle předem daných pravidel vytváří neuronovou síť. Tento pohled na problém zúžil autorův výběr programovacího jazyka na programovací jazyky objektově orientované (úplně nebo částečně). Tyto jazyky se dokáží nejblíže přiblížit znázornění biologických neuronů právě svou objektovou orientací a jsou i blízké myšlení autora práce.

Dalším důvodem, který rozhodl ve prospěch programovacího jazyka Java, byla možnost konzultací s vedoucím práce o možných problémech a jejich řešení, jelikož vedoucí práce má s tímto programovacím jazykem mnoho zkušeností.

Poslední důvod, který vedl autora k tomuto rozhodnutí, bylo programovat v nových trendech, kde se programování s objektovou orientací dostává na popředí programovacích stylů.

4.8 Základní požadavky na realizovaný program PETRA

Základní požadavky, které se vytvořily, měly nejen určovat cíl nové verze PETRY, ale jejich volba měla autora práce postupně provést od základů programování v Javě, až po pochopení metody učení neuronové sítě a měla končit autorovým plnohodnotným přispěním na vznikajícím produktu PETRA. Proto zde uvedené cíle nejsou konečné a jsou tu jen ty, které se zatím autorovi podařilo splnit. Některé cíle během vytváření nové verze PETRY byly upravovány a rozšiřovány podle zjištěných nových možností, které podle autora a vedoucího práce mohly přispět k celkovému zlepšení produktu PETRA.

Základní cíle byly navrženy takto:

1. Seznámení s programovacím jazykem Java.
2. Vytvořit metody pro práci se soubory.
3. Vytvoření neuronu a jeho zařazení do neuronové sítě.
4. Vytvoření XML souborů a z těchto souborů umět načítat informace a podobné informace zase zpět do nich ukládat.
5. Vytvoření metody Zpětného šíření chyby a na ní navazujících metod.
6. Vytvořit základní GUI – grafické uživatelské rozhraní a toto rozhraní propojit s vytvořenými třídami.

7. Vytvořit možnost paralelního učení více sítí na základě programovacích prostředků využívající možností tvorby vláken (Threading).
8. Doplnění PETRY o grafický výstup popisující fázi učení sítě.
9. Tento bod zastupuje ještě neuskutečněné cíle, které budou vytvořeny v nových verzích programu nebo ještě budou doplněny do verze programu stávajícího.

4.9 Soubory xml obsahující data pro program PETRA

Program PETRA nevyužívá klasické databáze jako je sql apod., ale všechna potřebná data ukládá do souborů ve formátu xml. Výběr tohoto formátu nebyl náhodný, jelikož má tento formát následující vlastnosti:

- otevřený formát, který není úzce svázán s nějakou platformou,
- je určen především pro výměnu dat mezi aplikacemi,
- je určen pro publikování dokumentů,
- umožňuje popsat strukturu dokumentu z hlediska věcného obsahu jednotlivých částí,
- jazyk XML nemá žádné předdefinované značky tagy (tj. názvy jednotlivých prvků).

Na základě těchto vlastností jsme zvolili právě formát xml, jako nejvhodnější formát pro ukládání našich dat. Program PETRA tedy využívá databázi, která pracuje se soubory v xml formátu. Tuto databázi můžeme ještě rozdělit na tři základní zdroje, které se liší svojí funkcí a to:

1. Soubory obsahující znalostní bázi dat, tj. soubory obsahující učební složky, které jsou důležité ve fázi učení sítě.
2. Soubory obsahující neuronové sítě.
3. Soubor obsahující řešený problém. To jsou soubory obsahující složky, které má naučená síť za úkol analyzovat a řešit.

4.9.1 Soubory obsahující znalostní bázi dat

Neuronové sítě používané programem PETRA ve fázi učení sítě se učí pomocí učitele (supervised learning). Proto data, která jsou poskytována při učení sítě, musejí kromě vektoru **x** obsahovat i vektor **d.** Vektor **d** obsahuje k příslušné složce vektoru $\mathbf{x_i}$ složku $\mathbf{d_i}$ popisující odpovídající výsledek. Jako příklad uvádím učební soubor obsahující dva učební řádky.

```
<TEACHFILE>
<INPUT>
1.0 1.0
</INPUT>
<OUTPUT>
1
</OUTPUT>
<INPUT>
0.0 1.0
</INPUT>
<OUTPUT>
0
</OUTPUT>
</TEACHFILE>
```

Tag:

- TEACHFILE určuje začátek a konec učebního souboru,
- INPUT určuje začátek a konec jednoho učebního řádku,
- OUTPUT určuje začátek a konec výstupu pro odpovídající učební řádek.

4.9.2 Soubory obsahující neuronové sítě

Tyto soubory mohou být dvojího typu a to:

- soubory obsahující nenaučené sítě,
- soubory obsahující naučenou síť.

Hlavní rozdíl mezi těmito dvěma typy souborů je to, že soubor obsahující nenaučené sítě obsahuje jednu až více neuronových sítí a není v něm zaznamenána žádná funkce, podle které se síť učila. Naproti tomu soubor obsahující naučenou síť obsahuje právě jednu naučenou síť a funkci, podle které se síť učila. Jako příklad uvedu oba typy souborů, které jsou ve zkrácené formě a mají jen poukázat na strukturu a důležité tagy.

Soubory se sítěmi:

```
<GROUPNEURONALNETS>
  <FUNCTION>null</FUNCTION>
  <NEURONOVASIT>
<NE>
                <POSITION>2</POSITION>
                <WEIGHT>0.22143128698170778</WEIGHT>
                <WEIGHT>0.7677433959705671</WEIGHT>
                <WEIGHT>0.9428553477450046</WEIGHT>
                <LAYER>1</LAYER>
        </NE>
  </NEURONOVASIT>
</GROUPNEURONALNETS>
```

Tag:

- GROUPNEURONNETS – určuje začátek a konec souboru s neuronovými sítěmi,
- FUNCTION – určuje začátek a konec funkce. V případě, že by se jednalo o naučenou síť, je mezi tímto tagem uzavřen název použité funkce. Jinak je v tomto tagu uložena hodnota null.
- NEURONOVASIT – určuje začátek a konec jedné neuronové sítě,
- NE – určuje začátek a konec jednoho neuronu,
- POSITION – určuje pozici neuronu v neuronové síti,
- WEIGHT – určuje jednotlivé váhy $\mathbf{w_{ij}}$ neuronu,
- LAYER – určuje vrstvu, ve které se neuron nachází.

4.9.3 Soubor obsahující analyzovaný problém

I tuto část můžeme rozdělit na dvě skupiny a to:

- soubory obsahující zadání (hodnoty k hledanému problému),
- soubory obsahující 1, ... n vektorů $\mathbf{x_n}$.

Soubor obsahující zadání slouží v programu PETRA pro hledání řešení již na naučených sítích. Tedy tyto soubory obsahují to, co má PETRA ve výsledné fázi hodnotit.

Druhým typem souboru jsou soubory obsahující podklady pro analýzu. Změna xml souboru oproti variantě, kdy hledáme řešení, je v tom, že soubor obsahuje 1 až n

vstupních vektorů **x.** Tedy v popisovaném xml souboru by tento soubor obsahoval více SOURCE tagů. Každý tento tag by označoval začátek a konec jednoho učebního řádku $\mathbf{x_i}$ z učebního vektoru **x.**

```
<TEACHFILE>
        <FUNCTION>
                LOGICAL
        </FUNCTION>
        <SOURCE>
                <PARAMETR_0>
                1.0
                </PARAMETR_0>
                <PARAMETR_1>
                1.0
                </PARAMETR_1>
        </SOURCE>
</TEACHFILE>
```

Tag:

- TEACHFILE – určuje začátek a konec souboru,
- FUNCTION – určuje pole, ve kterém se nachází použitá funkce,
- SOURCE – určuje začátek a konec jednoho analyzovaného řádku,
- PARAMETR_0,1, ... , n – určují pořadí faktoru.

4.10 Součásti programu PETRA

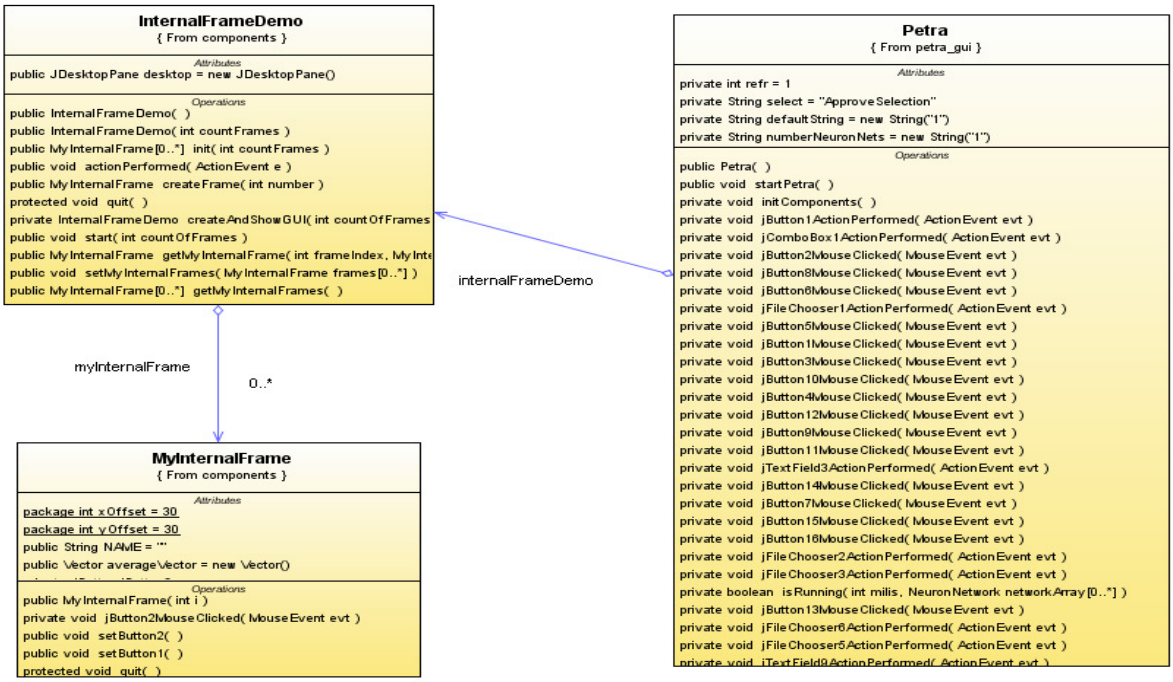
Program PETRA, tak jak je naprogramován, lze rozdělit na dvě hlavní oblasti a to:

1. Třídy využívané pro grafické uživatelské rozhraní GUI (Graphical User Interface)
2. Třídy, které zajišťují realizační část (tj. operace s neuronovými sítěmi a soubory, atd.).

Následující podkapitoly se budou zabývat nejdůležitějšími třídami programu PETRA a pro jejich grafickou interpretaci je využit **Class Diagram**, kde jednotlivé třídy budou uvedeny ve zkrácené formě a diagram bude rozdělen na dvě části podle hlavních oblastí.

4.10.1 Třídy grafického uživatelského rozhraní programu PETRA

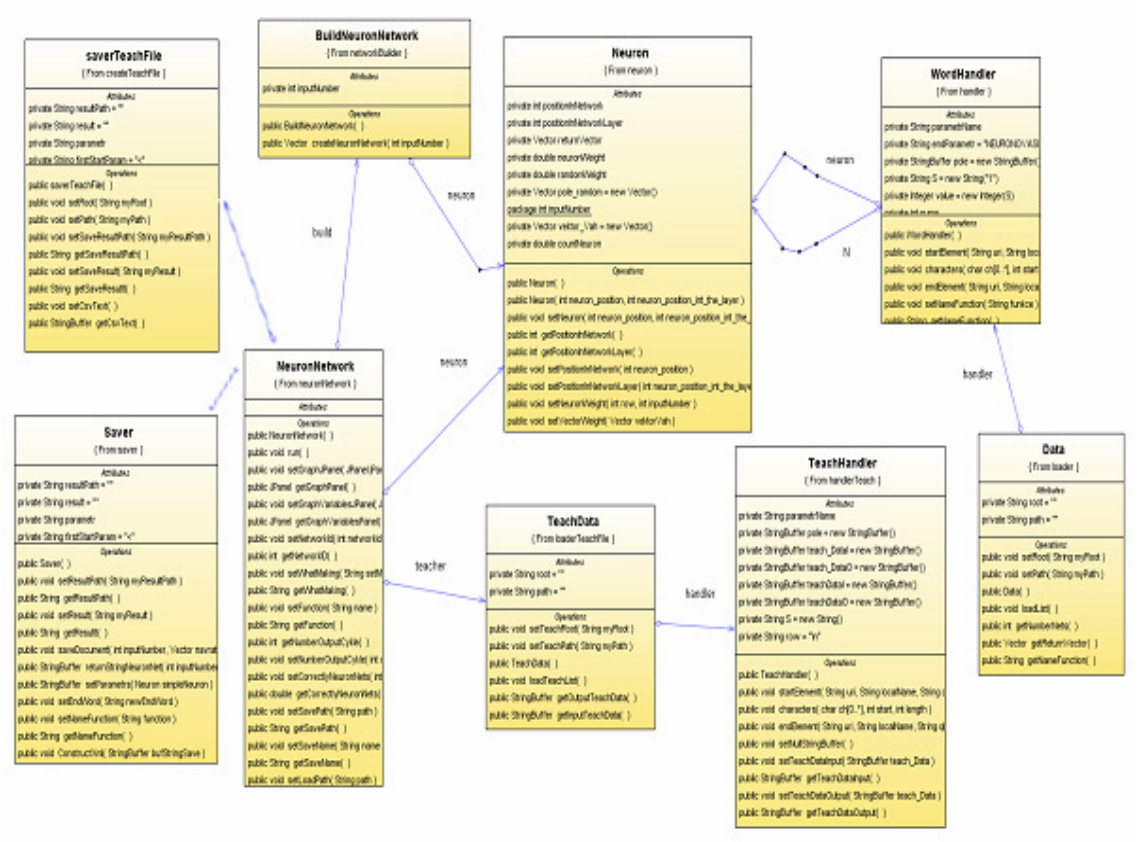
obrázek 7: Třídy grafického uživatelského rozhraní programu PETRA



Tato oblast se skládá ze tří tříd, kde je nejvýznamnější třída **Petra.** Třída **Petra** je hlavní třídou celého programu a tvoří spojnici mezi grafickou a realizační oblastí. V třídě **Petra** se vytváří hlavní komponenty grafického uživatelského rozhraní celého programu. Následující dvě třídy **InternalFrameDemo** a **MyInternalFrame** tvoří pouze doplňkovou část programu PETRA a jsou využívané pro grafické výstupy, kde realizují grafické zobrazení grafů.

4.10.2 Třídy realizující operace v programu PETRA

obrázek 8: Třídy realizující operace v programu PETRA



Oblast realizační části programu PETRA se skládá z devíti tříd. Nejdůležitější třídou je třída **NeuronNetwork**, která zabezpečuje komunikaci mezi GUI a realizační částí a dále „řídí“ činnost celého programu.

1. **Třída NeuronNetwork**

   Tvoří jádro celého programu a zajišťuje plnění požadavků, které přicházejí od uživatelů. Úkolem této třídy je zajištění:

   - ukládání a načítání souborů pomocí tříd,
   - vytvoření neuronové vrstvy,
   - učení sítě probíhá ve vlastní třídě NeuronNetwork a dále k tomu využívá třídu Neuron,
   - třída vlastními metodami zajišťuje učení neuronových sítí na základě metody Zpětného šíření chyby.

2. **Třída BuilderNeuronNetwork**

   Třída má za úkol vytvořit **x** neuronových sítí, které budou splňovat požadavky, které zadal uživatel. K tomuto úkolu využívá třídu **Neuron.**

3. **Třída Neuron**

   Tato třída má za úkol vytvářet jednotlivé neurony a poskytovat informace o těchto neuronech.

4. **Třídy Saver, SaverTeachFile a TeachData**

   Tyto třídy mají za úkol zajišťovat operace jako načítání a ukládaní xml souborů, podle požadavků třídy **NeuronNetwork.**

4.11 Realizace neuronů a neuronových sítí programem PETRA

Neurony, které třída vytváří, musejí mít už takové vlastnosti, abychom je mohli vložit do neuronové sítě a dál pak s nimi provádět operace. Při realizaci neuronů autor vycházel z teoretických poznatků o vícevrstvých neuronových sítích. Na základě těchto znalostí musel autor realizovat neurony jako objekty, které budou znát neurony ve svém okolí, budou znát svou pozici v neuronové síti a budou umět na základě příchozích podnětů (tj. hodnot) určit svou vlastní hodnotu.

Neurony jsou realizovány tak, že je vždy vytvořena instance (objekt) neuron. Tato instance, aby splňovala vlastnosti neuronů, obsahuje metody pro vytvoření váhových spojení s ostatními neurony, ale i jeho vlastní váhovou složku Bias. Dále jsou tu metody, které buď nastaví tyto váhové složky jako náhodná čísla a nebo je dokáží nastavit podle požadavků programu. Instance Neuron obsahuje také metodu, která vyjádří hodnotu neuronu a která se získá jako:

$in\_y_j = w_{0j} + \sum_{i=0} x_i * w_{ij}$, kde $\mathbf{x_i}$ představuje hodnotu neuronu z vyšší vrstvy nebo receptor a kde $\mathbf{w_{ij}}$ je váhová složka. Hodnota neuronu je klíčová ve fázích učení sítě, v testování a analyzování problémů, které má síť za úkol řešit. Pro správné

fungování neuronů a jejich zařazení do neuronových sítí, obsahují neurony metody, které zařazují neuron do vrstvy a nastavují mu pozici ve vrstvě.

Jako vhodné řešení pro vyjádření třívrstvé neuronové sítě pro program PETRA je používáno dynamické pole vektor. Toto pole obsahuje další tři dynamická pole typu vektor. Každé z těchto tří polí znázorňuje jednu z vrstev sítě a jsou v něm zařazeny jednotlivé neurony. Funkce jednotlivých vektorů je následující:

- první vektor zastupuje vrstvu receptorů, počet neuronů zde určuje uživatel,
- druhý vektor znázorňuje skrytou vrstvu sítě, je velikostně dvakrát větší než první vektor, tedy obsahuje dvakrát víc neuronů než vektor první,
- třetí vektor je zastoupen jedním neuronem a představuje výstupní vrstvu sítě.

4.12 Grafické uživatelské rozhraní programu PETRA

Pokud program PETRA má být úspěšně nasazován v podnicích, musí vytvořit takové prostředí, které bude schopno komunikovat s uživatelem. Pro tento účel nám slouží GUI – grafické uživatelské rozhraní. Toto rozhraní tvoří spojnici mezi uživatelem a programovým jádrem.

V první verzi programu PETRA se autor práce spíše zaměřil na jednotlivé komponenty programu než na celkový design. Cílem bylo vytvořit takové GUI, které nabídne uživateli mnoho možností jak konfigurovat a ovládat neuronové sítě v jakékoli fázi životního cyklu sítě.

4.12.1 Úvodní okno programu PETRA

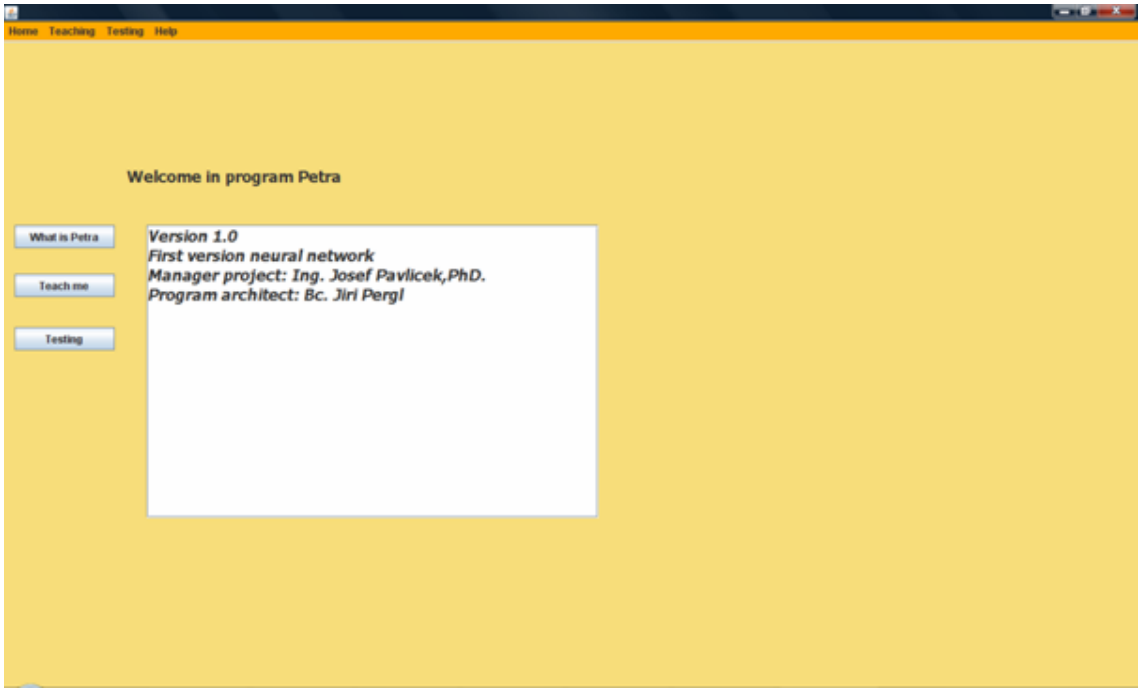
Po spuštění programu PETRA má uživatel možnost ovládat program pomocí tlačítek umístěných po levé straně nebo se může použít navigační panel na horní liště.

Z úvodního okna má uživatel možnost vstoupit do třech hlavních oblastí programu:

- první oblast poskytuje uživateli základní informace o Programu PETRA a neuronových sítích,
- druhá oblast slouží pro účely vytváření neuronových sítí, učebních souborů a učení sítí,

- třetí oblast slouží k hledání řešení analyzovaných problémů a k určování významnosti jednotlivých parametrů.

obrázek 9: Úvodní okno programu PETRA



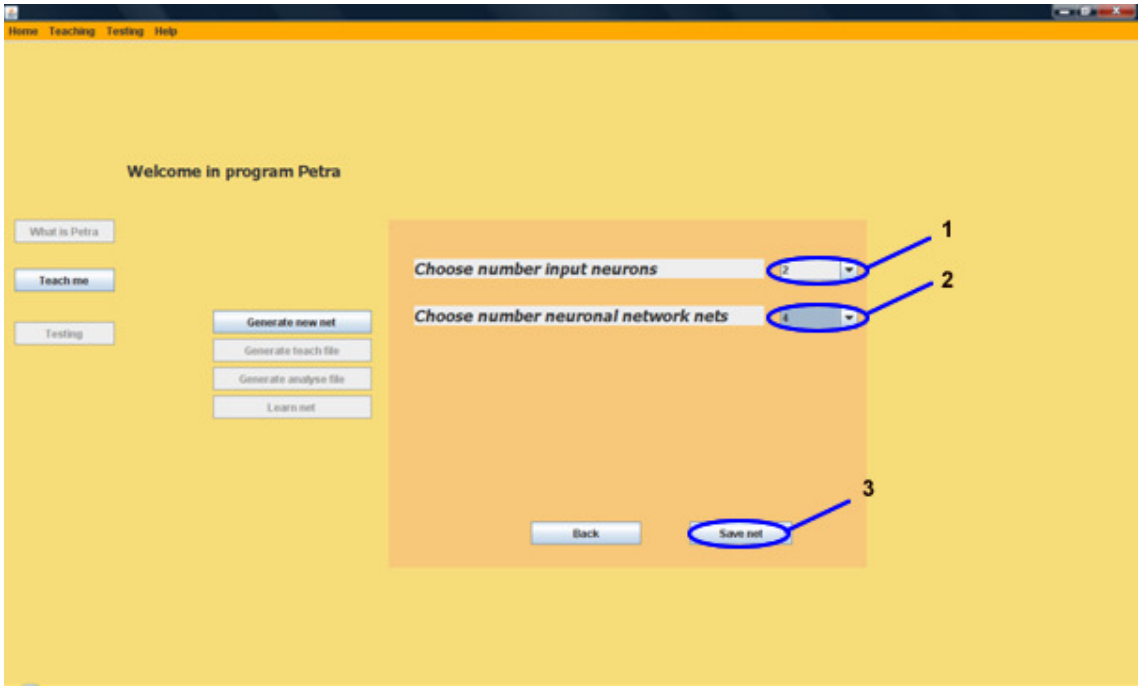
4.12.2 Oblast vytváření neuronových sítí a učebních souborů a učení sítí

Tato oblast je asi nejdůležitější oblastí programu PETRA, lze ji rozdělit na části:

- oblast vytváření neuronových sítí,
- oblast vytváření učebních souborů,
- oblast vytváření souborů pro analýzu,
- oblast učení neuronových sítí,
- oblast výstupu programu PETRA.

4.12.2.1 Oblast vytváření neuronových sítí

obrázek 10: Vytváření neuronových sítí



Při vytváření neuronových sítí má uživatel možnost zvolit počet vstupních neuronů (receptorů), které ovlivní velikost celé sítě a schopnost sítě řešit zadané problémy.V této fázi je vždy zapotřebí zvolit stejný počet vstupních neuronů jako má vstupní vektor v učebním souboru (např. logická funkce AND by byla realizována dvěma vstupními neurony). K zadávání počtu vstupních neuronů nám slouží komponenta označená na obrázku číslem 1.

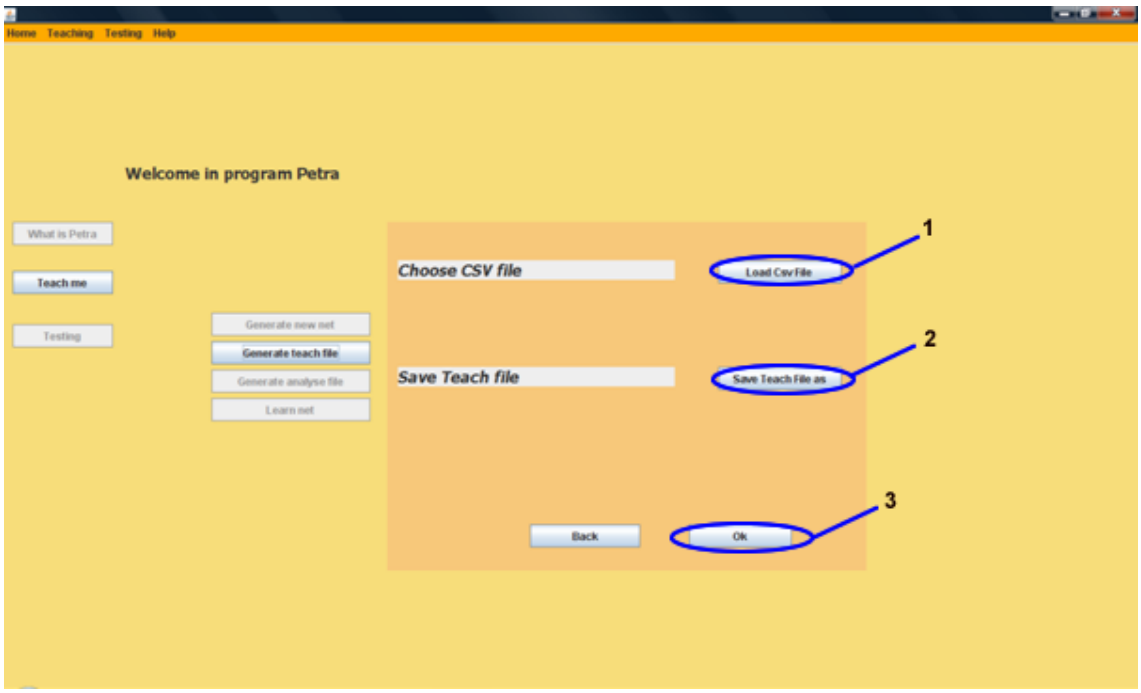
Jelikož na počátku jsou váhovým složkám v neuronové síti nastavena náhodná malá čísla blízká nule, můžeme z toho pozorovat, že každá síť má jinou schopnost učení. Tato počáteční náhodná čísla ovlivňují rychlost učení sítě, kvalitu naučení sítě, a v některých případech mohou způsobit neschopnost sítě se daný problém naučit. Tento jev je podobný, jako kdybychom měli skupinu lidí, kteří řeší zadaný úkol. V této skupině bychom našli jedince, kteří mají schopnost se rychle naučit zadaný úkol, pak jedince, kteří se učí pomaleji, ale úkol plní kvalitněji a pak jsou zde jedinci, kteří se daný úkol nenaučí řešit vůbec. Pokud bychom tuto skupinu lidí postavili před jiný úkol, můžeme dosáhnout opačného výsledku u každého jedince.

Na základě těchto skutečností jsme zvolili možnost vytvořit více neuronových sítí se stejným počtem vstupních neuronů v jednom souboru. Od tohoto kroku jsme očekávali možnost volby kvalitnější sítě pro zadaný úkol a celkové urychlení ve fázi učení sítí, než kdybychom museli nastavovat parametry pro každou síť jednotlivě. Nevýhodou více neuronových sítí v jednom souboru je zvolení příliš velkého počtu těchto sítí na jeden soubor. Tato skutečnost může mít za následek příliš velké výpočetní zatížení pro PC a i zvětšení doby ve fázi učení sítí. Komponenta na volbu počtu neuronových sítí je označena bodem 2.

Komponenta označená číslem 3 slouží k potvrzení vytvoření neuronových sítí a vyvolaní okna s možností volby cesty a jména souboru.

4.12.2.2 Oblast vytváření učebního souboru a analyzovaného řešení

obrázek 11: Vytváření učebního souboru



Tato oblast slouží k vytvoření souboru xml, který bude obsahovat učební soubor. Jak už autor popsal v předešlých kapitolách, program PETRA používá jako databázové úložiště soubory xml, které mají předem danou formu. Abychom usnadnili uživateli práci s programem PETRA, vytvořili jsme oblast, kde uživatel má možnost převést svá data do učebního xml souboru. Převod se provádí automaticky a tím odpadá uživateli

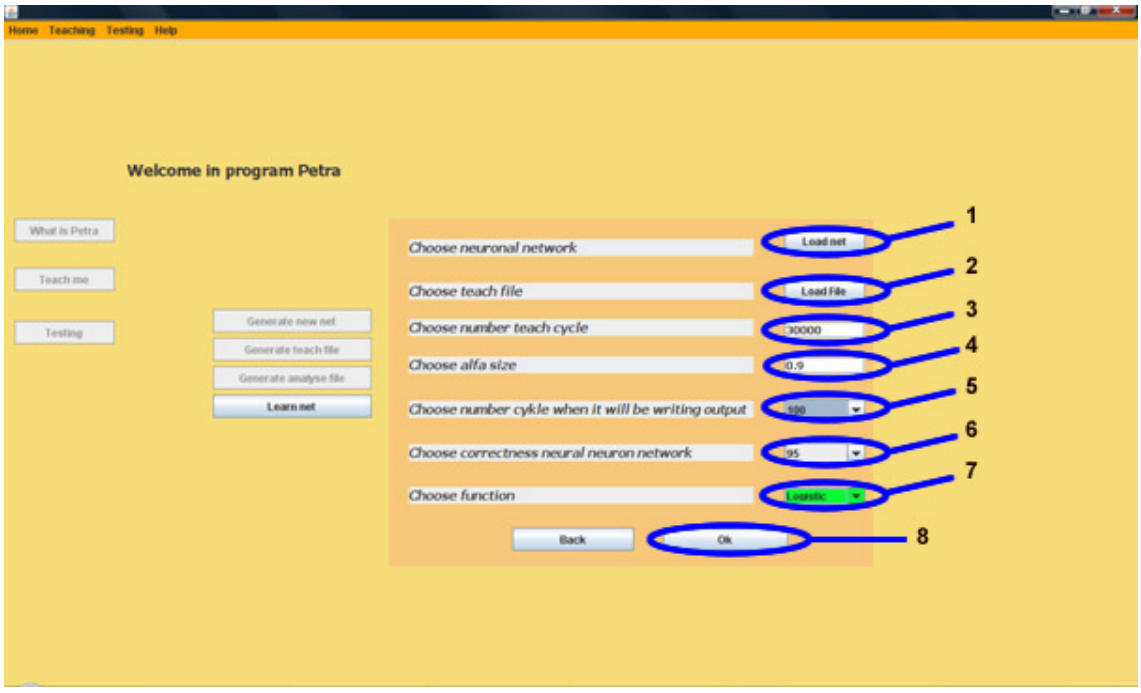
náročná práce s vytvářením tohoto xml souboru. Podmínkou tohoto převodu je, že data musí být uložena v csv formátu, soubor nesmí obsahovat názvy sloupců a jednotlivé hodnoty jsou rozděleny středníkem. Tyto hodnoty musí obsahovat pouze číselný zápis. Program PETRA je pak schopný ještě provést základní kontrolu dat, která spočívá v odstranění desetinných čárek, které nahradí tečkami a mezer v číselném zápise.

Tato oblast obsahuje tři základní komponenty. Komponenta označená číslem 1 je tlačítko, které vyvolá okno s možností volby cesty a jména csv souboru. Komponenta označená číslem 2, vyvolá okno s možností zadání cílového umístění a názvu učebního souboru. Celý proces se pak potvrdí komponentou označenou číslem 3.

Podobný ovládací panel, jako pro ukládání učebního souboru je při operaci vytváření analyzovaného zadání. Předpokladem je, že dané zadání bude uloženo v souboru ve formátu csv. Tento soubor csv je pak uložen do xml souboru. Chování a ovládací prvky jsou stejné jako na obrázku popisujícího vytváření učebního souboru. Dochází jen ke změně popisku tlačítek a polí.

4.12.3.3 Oblast učení neuronových sítí

obrázek 12 : Učení neuronových sítí



Správné naučení neuronových sítí je klíčový faktor, který určuje úspěšnost v řešení a analyzování požadovaných problémů (požadavků). Z tohoto důvodu je tato oblast nejdůležitější a klade největší nároky na komunikaci s uživatelem a na uživatelovu znalost neuronových sítí, jelikož obsahuje mnoho důležitých parametrů, které ovlivňují rychlost a kvalitu učení neuronových sítí. Komponenty označené čísly 1 a 2 vyvolají podobné akce, vždy se objeví nové okno, kde si uživatel navolí cestu a název učebního souboru nebo souboru s neuronovými sítěmi. Zajímavější jsou pak komponenty 3 až 7, které ovlivňují učení sítí. V komponentě 3 si uživatel zadá číslo, které představuje počet cyklů, které má síť provést. Jedním cyklem je zde chápáno předložení všech vstupních vektorů a jím odpovídajících výstupů z učebního souboru xml. Je jasné, že čím je číslo větší, tím se zvyšuje čas a výpočetní náročnost na PC a program PETRA. V této souvislosti je zde ještě jeden parametr, který může zastavit program, aniž by došlo ke splnění podmínky počtu provedených cyklů, které zadal uživatel. Tento parametr je zde proto, aby buď nedošlo k přeučení sítě a také zbytečnému zatěžování programu PETRA a PC a nebo zbytečnému časovému zatížení, kdy uživatel čeká na naučení sítě. Tato komponenta je označena číslem 6 a označuje procentní korektnost naučení sítě. Hodnota korektnosti sítě se zjistí jako:

$$C = \left(1 - \frac{\sum_{i=1}^{n} E_i(x_i, d_i)}{n} \right) * 100$$, kde **n** je počet učebních řádků v souboru a funkce $E_i(x_i, d_i)$ určuje chybu pro jeden učební řádek v souboru. Můžeme zde nalézt podobnost jako u předešlé komponenty, kde může hrozit, že uživatel zadá příliš velkou hodnotu. U této komponenty může být situace podobná v případě, když uživatel zadá příliš velký požadavek na korektnost sítě. Při požadavku na vyšší korektnost bude síť nucena se učit déle a tím budou stoupat požadavky na PC a program PETRA. Proto tyto dvě komponenty tvoří hranici (omezení) jedna pro druhou, kdy při splnění podmínky buď při naplnění učebních cyklů nebo splnění korektnosti sítě dochází k zastavení učení sítí.

Komponenta číslo 4 slouží pro zadání čísla alfa. Číslo alfa je koeficient učení. Autor práce by toto číslo označil jako „magické číslo“. Jedná se o číslo, které není předem

definováno. Všeobecně je známo, že koeficient učení by se měl nalézt v určitých intervalech, ale i tyto intervaly jsou rozdílné, např. *Veselý viz. [4]* udává interval $0.01 \leq \alpha \leq 0.1$, naproti tomu *Volná viz. [10]* udává koeficient $0 \leq \alpha \leq 1$, autor práce se spíše přiklání k druhému intervalu, který by ještě rozšířil na interval $0 \leq \alpha \leq 2$. Tento interval autor získal z pokusů, které prováděl s neuronovými sítěmi. Koeficient učení - „magické číslo“, je důležitým prvkem ve fázi učení sítě, jelikož určuje velikost váhové změny složek v neuronové síti. Pokud koeficient učení bude příliš vysoký budou váhové změny v síti veliké a může dojít k přeskočení minima funkce, které je hledáno. Při opačné možnosti, když bude koeficient malé číslo, váhové změny v síti budou malé a síť se bude učit dlouho. Optimální velikost koeficientu učení nelze určit, jelikož je pro každou úlohu, dokonce i pro každou síť jiný. Jako důkaz tohoto tvrzení autor práce udává dva příklady:

1. Při pokusech s koeficientem učení na logických funkcích autor zjistil, že pokud našel hodnotu koeficientu učení blízkou optimální hodnotě pro konkrétní síť, např. 0.9 pro logickou funkci OR, tak učení sítě trvalo 33 cyklů, než splnila podmínku korektnosti 95%. Po změně koeficientu učení na 1.2, tedy o tři desetiny, narostl počet cyklů na 300 pro stejnou síť.
2. Ještě výraznější příklad byl při testování jiné sítě na stejnou logickou funkci OR, kde optimální hodnota koeficientu učení byla blízká číslu 1.35 a síť se naučila daný problém v 41. cyklu. Při pokusech se hodnota koeficientu zvýšila o jednu setinu, tedy na číslo 1.36, a tato skutečnost vedla k tomu, že síť se nebyla schopna v rozmezí 100 000 cyklů daný příklad vůbec naučit. Nakonec až při změně koeficientu na 1.39 dosáhla naučení, při zachování 95 procentní korektnosti, v 37 375. cyklu, což oproti původnímu řešení byl dost značný nárůst.

Z těchto dvou příkladů je patrné, jak je významný koeficient učení. Nalezení koeficientu učení blízkého optimální hodnotě je pak klíčovým faktorem v úspěchu učení sítí.

Dalším důležitým faktorem, který ovlivňuje učení sítě, je druh funkce použitý na výstupu neuronu. Volbu funkce zajišťuje komponenta číslo 7 a nabízí v současné době dvě možnosti, buď použít logickou, nebo logistickou funkci. Zvolení funkce není sice v některých případech klíčový faktor pro naučení sítí, ale vhodná volba může

urychlit celkové učení sítě. Jako příklad můžeme uvést testování na neuronové síti uvedené v příkladu 2, kde při koeficientu učení 1.35 síť dosáhla naučení, při korektnosti 95 %, v 41. cyklu. V tomto příkladě byla využita logická funkce na výstupu. Pro názornou ukázku autor ponechal všechny parametry stejné a změnil pouze výstupní funkci na logistickou a získal následující výsledek. Síť se naučila na 95 % korektnost v 115. cyklu. Z tohoto příkladu je patrné, že volba sítě dokáže uspíšit celkové učení sítě. Autor také zjistil, že pokud učil sítě, kde výsledná hodnota $\mathbf{d_i}$ nebyla hodnota 0 nebo 1, je volba funkce dost podstatná. Jako příklad autor uvede, že se pokoušel naučit síť hodnocení hodů šestistěnné kostky, kde bylo možných šest výsledků (1 až 6). Při učení sítě došel k závěru, že logická funkce je nepostačující a nedokáže se tomuto problému dostatečně přizpůsobit. Na základě tohoto problému zavedl do učebního cyklu funkci logistickou, která se dokáže lepé přizpůsobit výsledkům, které neodpovídají jen hodnotám 0 nebo 1. Na druhou stranu je lepší nasazení logické funkce tam, kde je správné řešení 0 nebo 1. Proto je výhodnější nasazení logické funkce tam, kde budeme učit síť logické funkce.

Jelikož učebních cyklů může být řádově statisíce až miliony, je do programu PETRA zařazena komponenta označená číslem 5, ve které si uživatel vybírá číslo výpisu cyklu. Toto číslo udává, v jakém cyklu má dojít k výpisu textového výstupu nebo zaznamenání do grafu. V první verzi programu PETRA je důležité volit číslo výpisu rozumně k počtu zadaných celkových cyklů, které mají proběhnout, jelikož grafy jsou omezeny pamětí a může dojít k jejich přeplnění. Pokud k této události dojde, pak uživateli se daný grafický výstup nezobrazí.

Jestliže uživatel zadal všechny parametry správně, dojde k učení sítí.

4.12.2.4 Oblast výstupu programu PETRA

Po správném zadání všech parametrů v oblasti učení sítí se uživateli objeví nové okno, které má v sobě podoblasti zobrazené jako další okna, která reprezentují každou síť, kterou učil. Tyto „vnořená“ okna nabídnou pro každou síť tři výstupy a to:

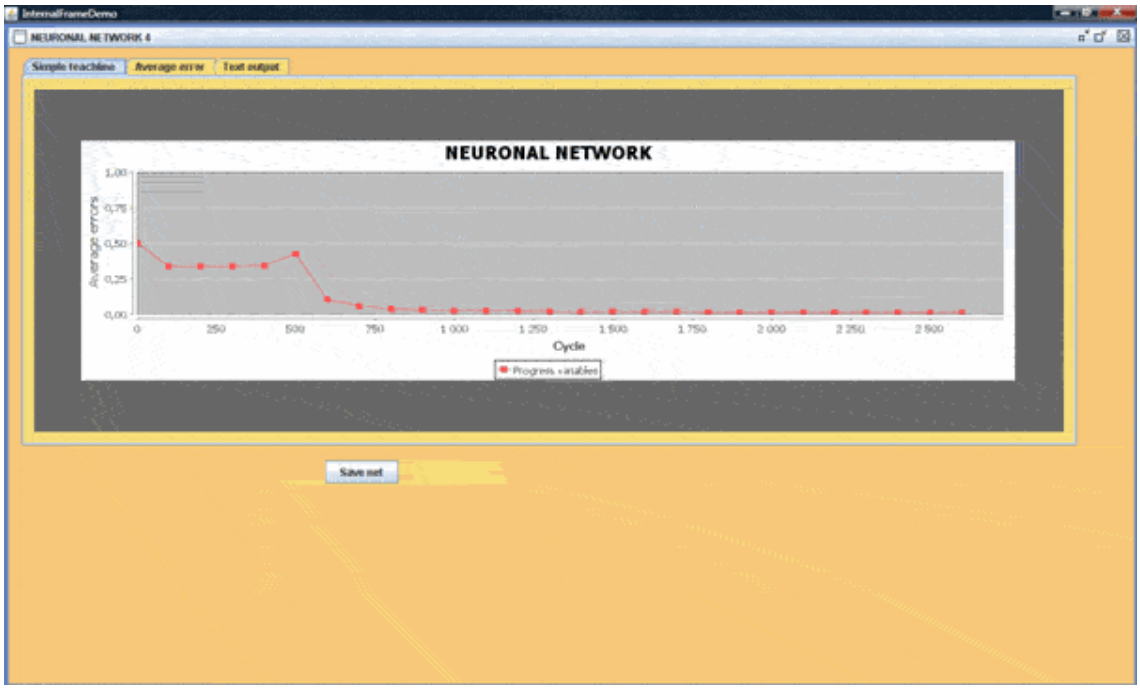
- grafický výstup znázorňující celkovou průměrnou chybu,
- grafický výstup znázorňující chybu pro konkrétní učební řádek,
- textový výstup.

Každý z grafických výstupů lze ještě upravovat (např. změna barvy spojnic), zvětšovat a zmenšovat atd. Pro realizaci řešení je zde využito knihovny **JFreeChart**, která je dostupná jako openSource verze a poskytuje základ pro tvorby grafů.

Grafický výstup znázorňující celkovou průměrnou chybu

Tento výstup znázorňuj, jaké chyby se síť dopustila v rámci jednoho učebního cyklu. Uživatel zde může vidět, jak se daná síť učila a v jakém cyklu dosáhla prahu korektnosti. Na základě tohoto výstupu pak může volit velikost koeficientu učení, počet cyklů, druh funkce, korektnost a doplňkové funkce (např. v kolikátém cyklu má dojít k výpisu).

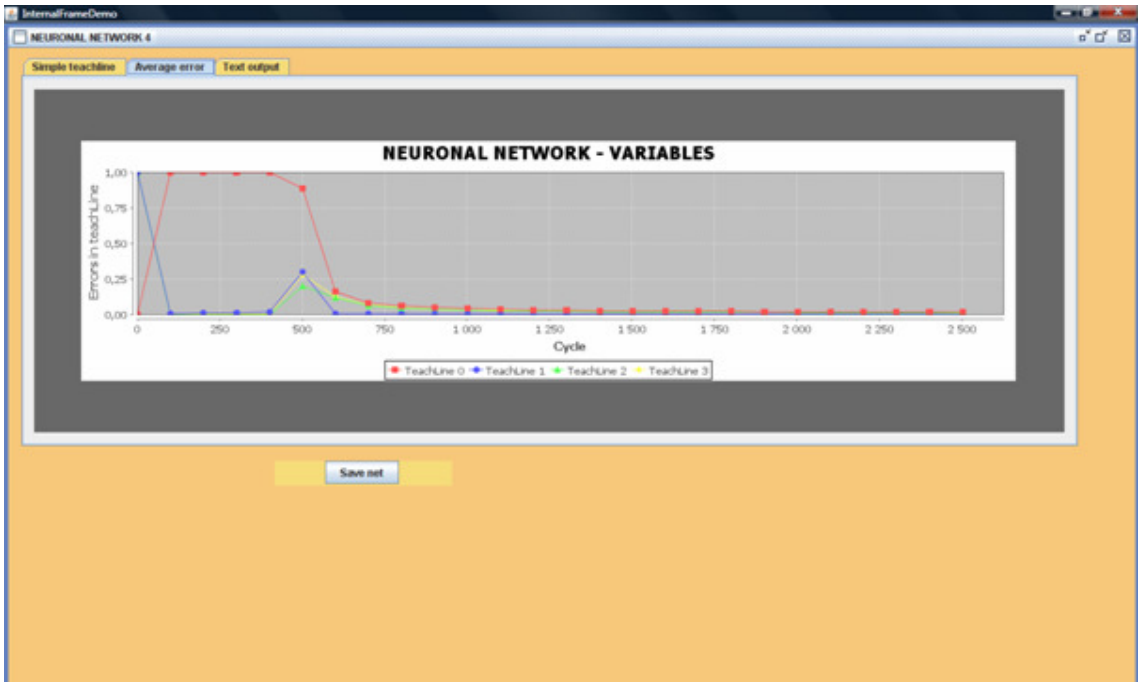
obrázek 13: Grafický výstup průměrné chyby



Grafický výstup znázorňující chybu pro konkrétní učební řádek

Tento výstup zachycuje reakci sítě na konkrétní učební řádek v konkrétním učebním cyklu. Uživatel může vidět, jak daný učební řádek dělal, nebo naopak nedělal síti problém se ho naučit. V grafu jsou zaznamenány velikosti chyb, kterých se síť na daném učebním řádku v daném cyklu dopustila. Tento grafický výstup byl do programu zařazen proto, abychom uživateli pomohli ještě více optimalizovat učební soubor a tím zrychlit učení sítě. Uživateli se tak naskytne možnost daný problematický učební řádek prozkoumat. Pokud usoudí, že není pro učení sítě významný, může ho z učebního souboru odstranit. Tento výstup lze tak využít s funkcí analyzování významnosti jednotlivých faktorů. Funkce analyzovaní významnosti jednotlivých faktorů bude vysvětlena v diplomové práci později.

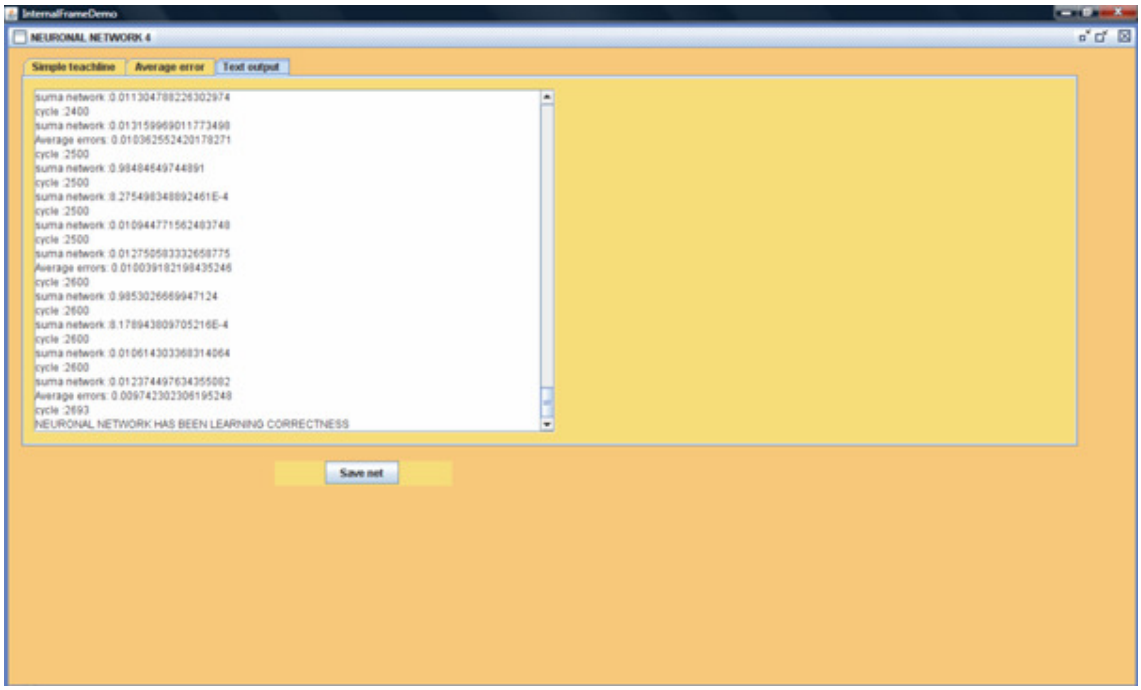
obrázek 14: Grafický výstup chyby v učebním řádku



Textový výstup

Textový výstup popisuje detailní výsledky neuronové sítě. Do textového výstupu se zapisuje konkrétní hodnota pro učební řádek v konkrétním cyklu. Uživateli se tak naskytne možnost sledovat velikost skutečné hodnoty jako odpovědi na daný vstup.

obrázek 15: Grafický výstup textového výstupu



Poslední komponentou, pro všechny tři výstupové bloky společnou, je tlačítko, které umožňuje danou síť uložit ve formátu xml.

4.12.3 Oblast testování a hledání řešení zadaných problémů

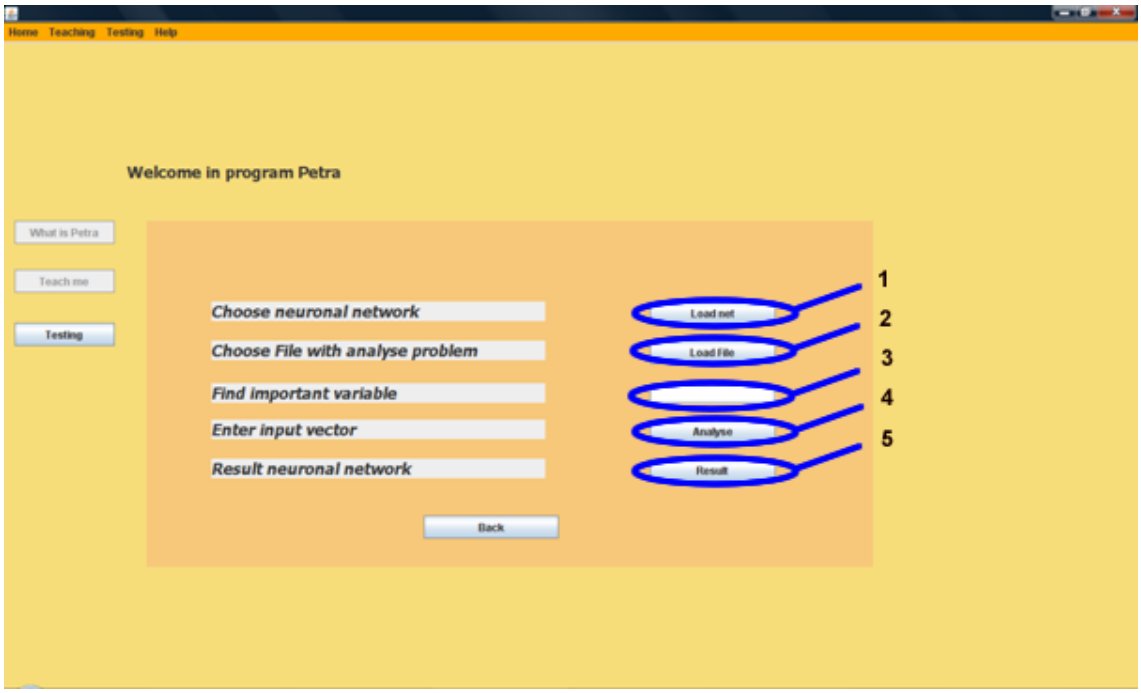
Pokud již máme naučenou neuronovou síť, můžeme v oblasti testování provést dvě různé operace a to:

- zadat naučené síti konkrétní problém a očekávat výsledky,
- analyzovat síť a hledat nevýznamné parametry.

Jestliže zvolíme první variantu a budeme se pouze pokoušet nalézt řešení na zadaný problém, stačí pouze načíst naučenou síť. Tento úkol plní komponenta označená číslem 1. Poté co uživatel vybral neuronovou síť, má dvě možnosti, jak zadat vstupní parametry pro síť. První a asi jednodušší metodou je využít komponentu číslo 2 a načíst parametry z xml souboru. Druhou možností je zadat parametry do vstupního pole.

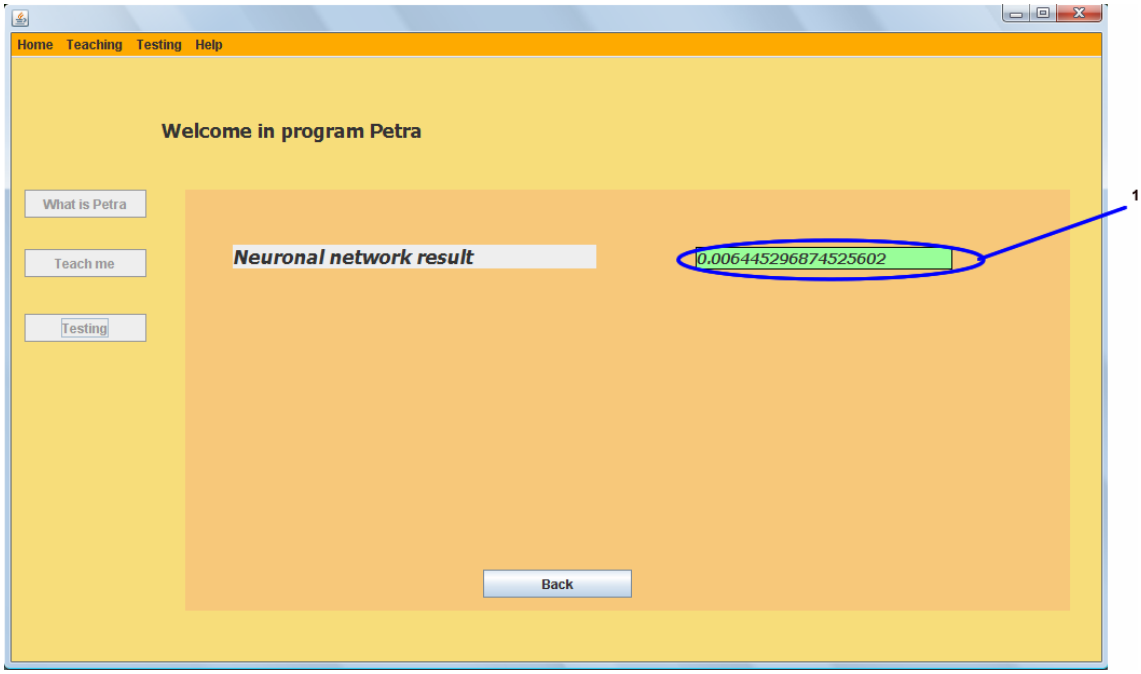
Druhá možnost je při větším počtu parametrů pro uživatele časově náročnější a zvyšuje riziko, že uživatel v zápise provede chybu nebo zadá nějaký z parametrů v chybném tvaru. Vstupní pole určené k zadaní vstupních parametrů je označené číslem 3.

obrázek 16: Testování a hledání řešení analyzovaných problémů



Po vybrání neuronové sítě a načtení vstupních parametrů stačí použít komponentu označenou číslem 5. Poté program provede požadovaný výpočet a výsledek výpočtu se zobrazí na novém panelu (obrázek č. 17), který obsahuje pole s výslednou hodnotou. Pole je označeno číslem 1.

obrázek 17: Výsledek řešení sítě



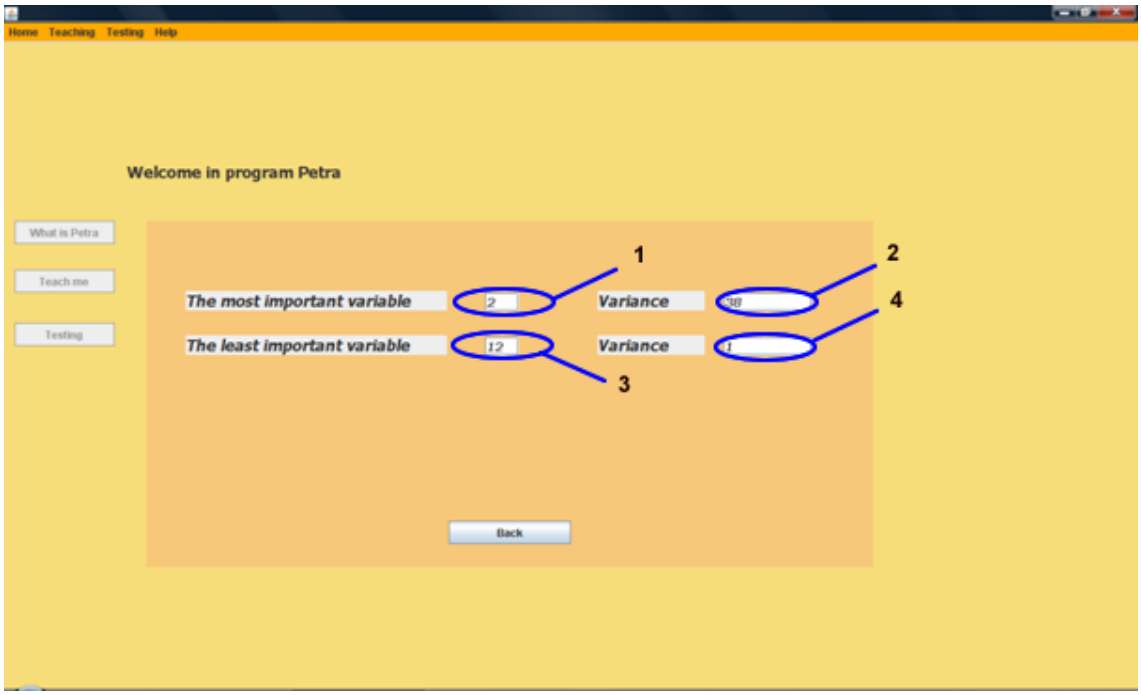
Při druhé variantě má uživatel možnost spustit na naučené neuronové síti analýzu významnosti parametrů. V tomto případě je podmínkou to, aby uživatel načetl učební soubor, ale i soubor obsahující vstupní parametry z xml souborů. V načítání xml souborů postupuje stejným způsobem jako v první variantě.

Analýza spočívá v tom, že určitým způsobem dochází k váhové modifikaci v již naučené síti. Této síti je předložena jedna složka ze vstupního vektoru **x** a zkoumá se, k jak velkému rozdílu ve výsledku došlo oproti hodnotě výsledku, která byla určena před váhovou modifikací. Váhová modifikace proběhne vždy jen pro jednu ze složek vstupního vektoru $\mathbf{x_i}$. V analýze se program snaží o to, aby určil ty složky vektoru $\mathbf{x_i}$, které jsou nejvýznamnější, tedy při změně vyvolaly největší rozdíl oproti původnímu výsledku. Na druhé straně hledáme i opačné složky, které jsou nejméně významné, tedy ty, které při testování vykazovaly malý rozdíl oproti původnímu řešení. Pokud málo významná složka vykazovala velmi malý rozdíl, můžeme ji po objektivní úvaze i z modelu odstranit, aniž bychom ovlivnili výsledek sítě. Při odstranění této složky z modelu pak ale musí dojít k novému naučení sítě, jelikož jsme odstranili jeden parametr ze vstupního vektoru. Přínos odstranění 1 až n nevýznamných složek pro model může být v tom, že síť v průběhu učení nezatěžujeme dalšími nedůležitými faktory. V konečném výsledku může síť zvýšit citlivost na ostatní faktory, které nám

vyšly v analýze jako důležitější, než odstraňované nevýznamné složky. Uživatel by se měl vždy při odstraňování nevýznamných složek nejdříve podívat na učební soubor jako na celek, jelikož může dojít k tomu, že odstraněním nevýznamného faktoru dojde k narušení struktury učebního souboru. Takto narušený soubor pak bude ve výsledku poskytovat špatný základ pro učení sítí. Dané sítě by pak vykazovaly menší spolehlivost a přesnost ve výsledku.

Spuštění analýzy zajišťuje komponenta označená číslem 4. Po skončení analýzy se uživateli zobrazí následující okno:

obrázek 18: Přehled důležitosti parametrů



Toto okno popisuje výsledky analýzy. Komponenta označená číslem 1, určuje nejvýznamnější faktor, který nejvíce ovlivnil výsledek sítě. Komponenta označená číslem 2 určuje, k jaké došlo odchylce při změně faktoru od výsledku sítě bez změny faktoru.

Komponenta označená číslem 3 určuje nejméně významný faktor a jeho odchylku. Odchylka faktoru je označena číslem 4. Uživatel má tedy možnost posoudit, jaký faktor je nejvýznamnější, který je nejméně významný a jak se tyto faktory podílejí na celkovém výsledku sítě. Na základě toho má uživatel další pomoc v případných změnách a optimalizacích v učebním souboru.

5 Závěr

Při analýze neuronových sítí autor došel k závěrům, že neuronové sítě jsou velmi dobře propracované. Můžeme na nich sledovat, jak historickým vývojem dochází i k přizpůsobování se požadavkům, jejich rozvoji a zdokonalování. Přelomové změny byly spjaty s vyřešením jejich nedostatků, za které byly kritizovány. Neuronové sítě v podobě, v jaké je vidíme dnes, vznikly především zásluhou vědeckých týmů vedených Rosenblattem, Minským, Widrowem, Hopfieldem a Kohonenem. Tyto týmy nepracovaly pouze na jedné variantě neuronových sítí, ale připravily nám mnoho různých variant. U všech typů sítí však nalezneme vlastnosti, které jsou pro všechny sítě společné a díky nim jsou tak vyhledávané. Je to:

- schopnost učení,
- schopnost abstrakce,
- paralelní zpracování,
- robustnost.

Tyto čtyři základní vlastnosti vytvořily z neuronových sítí unikátní oblast, která je využívána pro prognózy, rozpoznávání objektů, řečí a řízení složitých systémů. Často je použití těchto sítí jediným reálným východiskem v dané problematice. Tak jak se vytvářely různé alternativy neuronových sítí, tak se i zvyšují nároky na podrobnou znalost různých obměn a odlišností mezi jednotlivými sítěmi. Každý typ je totiž vhodný pro jiný typ problematiky a volba sítě je tak základním krokem k úspěšnému řešení.

Důležitou vlastnost, kterou však všechny varianty neuronových sítí postrádají, je schopnost vysvětlit, proč přijmuly dané řešení jako správné. Tento nedostatek brzdí rozvoj sítí v oblastech (např. lékařství), kde právě poukázání na prvky, které ovlivnily řešení, je klíčovým faktorem úspěchu. Do budoucna autor předpokládá, že se podaří nalézt takové metody a přístupy, který tento nedostatek potlačí nebo úplně odstraní.

Při realizaci programu PETRA se autor práce seznámil s vícevrstvými neuronovými sítěmi využívajícími k učení algoritmus zpětného šíření chyby. Ze zkušeností, které se mu podařilo získat, odvozuje hned několik závěrů. Implementace, tedy realizace neuronů a jejich zapojení a učení v síti je ve fázi programování velmi složité.

Díky objektově orientovanému programování byla tato fáze ulehčena hlavně při vytváření neuronů a jejich zapojení do sítě. Hlavní problém se pak ukázal při programování procesu učení sítě, kde autor musel vyhledat další konzultace s odborníky v této oblasti (doc. Ing. Arnoštem Veselým, CSc.). Avšak po překonání všech problémových částí, tak vznikl univerzální prostředek, který je schopen vytvářet neuronové sítě s různou velikostí a modifikací.

Po třech letech se podařilo zkompletovat program PETRA do podoby, která je již schopná plnit požadavky, pro které byla určena. Je zde vytvořeno grafické uživatelské rozhraní, které dovoluje uživateli snazší komunikaci s neuronovou sítí. Uživateli je umožněno ovlivnit síť ve fázi učení sítě a tím dosáhnout rychlejšího a i kvalitnějšího učení sítí. Pro snadnější orientaci v celém programu, autor práce vytvořil uživatelský manuál, který uživatele provede všemi kroky od instalace až po testovaní a učení sítě.

Důležitý doplněk, který by mohl částečně odstranit výše zmiňovaný nedostatek sítí, tedy neschopnost sítí poukázat na prvky, které ovlivnily řešení, je hledání významnosti parametrů.

Program PETRA je v době vypracování této diplomové práce připraven na testování problematiky odhadu složitosti softwaru, tak jak je popsané v disertační práci Ing. Josefa Pavlíčka, PhD.. Toto testování bude uskutečněno v následujících měsících. Program PETRA je svými vlastnostmi schopen plnit nejen problematiku odhadu složitosti softwaru, ale i úkoly z oblastí, ve kterých se používají vícevrstvé neuronové sítě s učením s učitelem a využívající metodu algoritmu zpětného šíření chyby.

6 Seznam literatury

1. PAVLÍČEK, Josef. Odhad manažerských charakteristik vývoje IS v etapě specifikace požadavků.(Doktorská disertační práce). Praha: Česká zemědělská univerzita v Praze, 2006.
2. VOLNÁ, Eva. Neuronové sítě 1. Ostrava: Ostravská univerzita v Ostravě, 2002. s. 10.
3. VESELÝ, Arnošt. Úvod do umělé inteligence. Praha: Česká zemědělská univerzita v Praze, 2005. s. 16.
4. VESELÝ, Arnošt. Úvod do umělé inteligence. Praha: Česká zemědělská univerzita v Praze, 2005. s. 66-67.
5. ROSSENBLATT, Frank. Principles of neurodynamics, Washington: Spartan Books, 1962.
6. VESELÝ, Arnošt. Úvod do umělé inteligence. Praha: Česká zemědělská univerzita v Praze, 2005. s. 68.
7. VESELÝ, Arnošt. Úvod do umělé inteligence. Praha: Česká zemědělská univerzita v Praze, 2005. s. 74.
8. VESELÝ, Arnošt. Úvod do umělé inteligence. Praha: Česká zemědělská univerzita v Praze, 2005. s. 80.
9. VESELÝ, Arnošt. Úvod do umělé inteligence. Praha: Česká zemědělská univerzita v Praze, 2005. s. 91-92.
10. VOLNÁ, Eva. Neuronové sítě 1. Ostrava: Ostravská univerzita v Ostravě, 2002. s. 38-41.
11. VESELÝ, Arnošt. Úvod do umělé inteligence. Praha: Česká zemědělská univerzita v Praze, 2005. s. 97-100.
12. VOLNÁ, Eva. Neuronové sítě 1. Ostrava: Ostravská univerzita v Ostravě, 2002. s. 72-76.

13. VESELÝ, Arnošt. Úvod do umělé inteligence. Praha: Česká zemědělská univerzita v Praze, 2005. s. 107 -109.

14. VESELÝ, Arnošt. Úvod do umělé inteligence. Praha: Česká zemědělská univerzita v Praze, 2005. s. 131.

15. PAVLÍČEK, Josef. Odhad manažerských charakteristik vývoje IS v etapě specifikace požadavků.(Doktorská disertační práce). Praha: Česká zemědělská univerzita v Praze, 2006. s.36.

16. PAVLÍČEK, Josef. Odhad manažerských charakteristik vývoje IS v etapě specifikace požadavků.(Doktorská disertační práce). Praha: Česká zemědělská univerzita v Praze, 2006. s.37-39.

Seznam tabulek, obrázků a grafů

Přílohy

Příloha č. 1

Ukázka podkladových údajů pro odhad složitosti softwarových produktů

Jmeno_projektu	ZSP	FSk_1	Fsk_2	Fsk_3	Fsk_4	Fsk_5	Fsk_6	Fsk_7	Fsk_8	Fsk_9	Fsk_10
OW	9	0	0	0	0	0	0	0	0	0	9
Slid_Button	5	0	0	0	0	0	0	0	0	0	5
CommonSTL	16	0	0	0	0	0	0	0	0	0	16
CodeNavigator	5	0	0	0	0	0	0	0	0	0	5
ProgressIndication	26	0	0	0	0	0	0	0	0	0	26
JunitTest	4	0	0	0	0	0	0	0	0	0	4
NewNameForRefac	3	0	0	0	0	0	0	0	0	0	3
SafeDelete	3	0	0	0	0	0	0	0	0	0	3
NavigateAndSource	8	0	0	0	0	0	0	0	0	0	8
http://refactoring.netbeans.org/nonav/refactorings/introducevariable.html	5	0	0	0	0	0	0	0	0	0	5
http://refactoring.netbeans.org/nonav/refactorings/convertanonymoustonested.html	5	0	0	0	0	0	0	0	0	0	5
http://refactoring.netbeans.org/nonav/refactorings/convertinnertotoplevel.html	4	0	0	0	0	0	0	0	0	0	4
http://core.netbeans.org/proposals/toolbarcustomizer-ui-spec.html	5	0	0	0	0	0	0	0	0	0	5
http://ui.netbeans.org/docs/ui/update/manager.htm	2	0	0	0	0	0	0	0	0	0	2
http://ui.netbeans.org/nonav/docs/hi/promoF/vcs-spec.html	25	0	0	0	0	0	0	0	0	0	25
http://refactoring.netbeans.org/nonav/refactorings/extractinterface.html	5	0	0	0	0	0	0	0	0	0	5
http://ui.netbeans.org/docs/ui/editor/find_dialog/find_dialog.html	6	0	0	0	0	0	0	0	0	0	6
http://ui.netbeans.org/docs/ui/editor/highlight_line/highlight-line.html	1	0	0	0	0	0	0	0	0	0	1
http://ui.netbeans.org/docs/ui/editor/gutter/glyphgutter-foldingbar.html	3	0	0	0	0	0	0	0	0	0	3
http://j2ee.netbeans.org/docs/promoe/enterprise-app-project-ui-spec-promoe.html	4	0	0	0	0	0	0	0	0	0	4
http://projects.netbeans.org/nonav/buildsys/build-sys-ui-spec-promoe.html	7	0	0	0	0	0	0	0	0	0	7
http://projects.netbeans.org/nonav/buildsys/j2se-project-ui-spec-promoe.html	8	0	0	0	0	0	0	0	0	0	8
http://projects.netbeans.org/nonav/buildsys/freeform-project-ui-spec-promoe.html	5	0	0	0	0	0	0	0	0	0	5

FSo_1	Fso_2	Fso_3	Fso_4	Fso_5	Fso_6	Fso_7	Fso_8	Fso_9	Fso_10	Fso_11	Fso_12	Fso_13	Fso_14	Fso_15	Fso_16	Fso_17	Fso_18	Fso_19	Dny
1	2	2	2	0	0	2	0	0	0	2	0	0	2	1	0	0	0	0	60
1	2	2	2	0	0	2	0	0	0	2	0	0	2	2	1	0	0	0	40
1	2	2	2	0	0	2	0	0	0	2	0	0	2	1	0	0	0	0	82
1	2	2	2	0	2	2	0	0	0	2	0	0	2	1	1	1	0	0	65
1	2	2	2	0	2	7	0	0	0	2	0	0	2	1	0	2	1	8	147
1	2	2	2	0	2	2	0	0	0	2	0	0	2	2	0	4	9	0	119
1	2	2	2	0	2	2	0	0	0	2	0	0	2	1	0	4	0	0	5
0	0	2	2	0	1	2	0	0	0	2	0	0	2	0	1	5	0	0	94
2	2	2	2	0	2	2	0	0	0	2	0	0	2	0	0	3	0	0	50
0	2	2	2	0	1	2	0	0	0	2	0	0	2	0	0	5	0	0	46
1	2	2	2	0	1	2	0	0	0	2	0	0	2	0	0	5	0	0	106
0	2	2	2	0	1	2	0	0	0	2	0	0	2	0	1	5	0	0	95
1	2	2	2	0	2	2	0	0	0	2	0	0	2	2	1	6	0	0	50
1	2	2	2	0	1	2	0	0	0	2	0	0	2	0	0	1	0	0	30
2	2	2	2	0	2	2	0	0	0	2	0	0	2	1	1	3	7	0	357
1	2	2	2	0	1	2	0	0	0	2	0	0	2	0	0	5	0	0	82
1	2	2	2	0	1	2	0	0	0	2	0	0	2	1	0	4	0	0	26
1	2	2	2	0	1	2	0	0	0	2	0	0	2	1	0	4	0	0	25
1	2	2	2	0	1	2	0	0	0	2	0	0	2	1	0	4	0	0	5
1	2	2	2	0	2	2	0	0	0	2	0	0	2	1	2	12	13	3	46
2	2	2	2	0	2	2	0	0	0	2	0	0	2	0	1	3	0	0	177
2	2	2	2	0	2	2	0	0	0	2	0	0	2	0	1	3	0	0	107
2	2	2	2	0	2	2	0	0	0	2	0	0	2	0	1	3	0	0	142