Syntéza neuronových sítí metodou symbolické regrese

Bc. Pavel Vařacha

Diplomová práce
2006

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2005/2006

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Pavel VAŘACHA**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Syntéza neuronových sítí metodou symbolické regrese**

Zásady pro vypracování:

Cílem práce je použít metody symbolické regrese na návrh struktury a učení neuronových sítí. Obsahem práce bude vytvoření programového kódu neuronových sítí a jejich následného použití v evolučním procesu. Jako metoda symbolické regrese bude použito analytické programování.

1. vypracovat přehled problematiky syntézy neuronových sítí
2. vybrat vhodné již řešené příklady
3. vypracovat alternativní řešení pomocí analytického programování a algoritmů SOMA, DE, SA a GA
4. provést závěr

Rozsah práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] ZELINKA, Ivan. Umělá inteligence I. Volume 1. Zlín : Vutium, Brno, 1998. 126 p. ISBN 80-214-1163-5.
[2] ZELINKA, Ivan, Umělá inteligence / kap.6 "Diferenciální evoluce", Academia, 33 p.
[3] Kvasnička V., Pospíchal J., Tiňo P. 2000, Evoluční algoritmy, STU Bratislava, ISBN 85-246-2000, 2000
[4] ZELINKA, Ivan, New Optimization Techniques in Engineering / kap.7 "SOMA – Self Organizingt Migrating Algorithm, Springer-Verlag
[5] Bose B.K., Liang P. 1996, Neural Network Fundamentals with Graphs, Algorithms, and Applications, McGraw-Hill Series in Electrical and Computer Engineering, ISBN 0-07-006618-3, 1996
[6] Masters T. 1993, Practical Neural Networks Recipes in C++, Academic Press, ISBN 0-12-479040-2, 1993
[7] Šnorek M., Jiřina M. 1996, Neuronové sítě a neuropočítače, ČVUT, ISBN 80-01-01455-X, 1996
[8] Bílá J. 1996, Umělá inteligence a neuronové sítě v aplikacích, ČVUT, ISBN 80-01-01275-1, 1996
[9] Zelinka I. 1998, Umělá inteligence I, VUT Brno, ISBN 80-214-1163-5, 1998
[10] Novák M., Faber J., Kufudaki O. 1993, Neuronové sítě a informační systémy živých organismů, Grada, ISBN 80-58424-95-9, 1993

Vedoucí diplomové práce: **doc. Ing. Ivan Zelinka, Ph.D.**
Ústav aplikované informatiky

Datum zadání diplomové práce: **14. února 2006**

Termín odevzdání diplomové práce: **26. května 2006**

Ve Zlíně dne 14. února 2006

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.
①

prof. Ing. Vladimír Vašek, CSc.
pověřený děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Tato práce se zabývá otázkou syntézy umělých neuronových sítí na principu evolučního prohledávání. Základním úkolem je vytvořit algoritmus symbolické regrese na základě analytického programování, který bude schopen syntetizovat vhodné neuronové sítě. Hlavní motivací je automatizace této syntézy a nalezení dosud neznámých řešení. Práce obsahuje především teoretické rozkrytí problému, analýzu úspěšně syntetizovaných řešení a zdrojové kódy vytvořeného algoritmu a provedených experimentů na přiloženém CD.

Klíčová slova: umělá inteligence, neuronová síť, symbolická regrese, analytické programování, evoluční prohledávání, evoluční algoritmy, syntéza sítí.

ABSTRACT

This thesis deals with question of artificial neuron network synthesis using evolutionary searching principle. The very task was to create symbolic regression algorithm based on analytic programming able to synthesize suitable neuron network. Motivation could be seen in computerization of network synthesis and exploration of unknown solutions. Thesis is primarily concern with theoretical background, analysis of successful solutions and experiments. Source codes of used algorithm can be found on attached CD.

Keywords: artificial intelligence, AI, artificial neural network, ANN, symbolic regression, analytic programming, evolutionary searching, evolutionary algorithms, network synthesis.

Na tomto místě bych chtěl především poděkovat panu **doc. Ing. Ivanu Zelinkovi Ph.D.** vedoucímu mé diplomové práce, za laskavé metodické vedení, cenné rady, pomocnou ruku i za důvěru, kterou ve mne neváhal vložit. Jeho pedagogické působení na naší fakultě pro mně bylo neutuchající inspirací po celých pět let mého studia.

V podobném duchu chci také poděkovat všem ostatním akademickým pracovníkům naší fakulty, kteří se spolupodíleli na mém intelektuálním růstu a byli pro mně světlem na cestě za poznáním tak složitého oboru, jakým je inženýrská informatika.

V neposlední řadě patří můj dík mým rodičům, kteří mě po celý můj život zahrnují rodičovskou láskou a bezvýhradnou podporou a umožnili mně tak dosahovat trvalých studijních úspěchů.

OBSAH

ÚVOD

Rozsáhlou renesanci umělých neuronových sítí (*artificial neural networks,* ANN) vystřídalo období stagnace, a to, jak na bázi aplikační, tak i v teoretické rovině. O skutečné složitosti lidského mozku jsme si přitom stále nevytvořili konkrétní představu [1]. Klasickou teorii neuronových sítí [2] nelze již dále rozvíjet, protože její základy vznikaly za pomoci značně omezeného výpočetního výkonu. S nástupem nové generace velmi rychlých mikroprocesorů se však otvírá prostor pro novou oblast umělé inteligence - evoluční algoritmy. Ty se dají s úspěchem použít k rozšíření schopností neuronových sítí [3]. Nikdo je ovšem na tuto oblast ještě neaplikoval tak výrazně, jako si to klade za cíl právě náš současný výzkum.

Jeho základní idea spočívá v myšlence, že počítače budoucnosti nebudou schopny neuronové sítě pouze samostatně učit a optimalizovat, ale také inteligentně syntetizovat - koncipovat jejich strukturu. Jedná se o další logický krok v oblasti neuronální umělé inteligence - tam, kde byl dříve potřeba člověk matematik, vykonává jeho práci chytrý algoritmus - symbolická regrese. Konkrétní metodou symbolické regrese je v našem případě Analytické programování [4], algoritmus dlouhodobě rozvíjený naší fakultou. Tento algoritmus má potenciál nalézt na množině tříd funkcí **F**, třídu funkcí **f*** (nenaučenou neuronovou síť), jíž je možno vhodnou metodou učení redukovat na konkrétní funkci **f** (naučenou neuronovou síť) takovou, že úspěšně řeší daný problém. Tento přístup nám dále umožňuje obohacovat množinu **F** o funkce nikoliv neuronálního charakteru a podnikat tak výzkum zatím zcela neprozkoumaného pomezí mezi ryze neuronálními a čistě matematickými strukturami.

Úkolem naší práce je tedy sestavit algoritmus symbolické regrese, který bude pomocí evolučního prohledávání úspěšně nacházet (syntetizovat) vhodná řešení $\mathbf{f} \in \mathbf{F}$ daného problému *P*. To by ovšem nebylo možné bez adekvátního teoretického pozadí, jak ukáže teoretická část naší práce. Navržené algoritmy i všechny další programové kódy jsou k dispozici na přiloženém CD. Praktická část práce obsahuje mimo jiné ukázky vybraných úspěšně syntetizovaných řešení.

I. TEORETICKÁ ČÁST

1 EVOLUČNÍ PROHLEDÁVÁNÍ

Věta: Mějme množinu všech neuronových sítí s dopředným šířením $\mathbf{ANN_{all}} = \{ANN_1, ANN_2, ..., ANN_i, ...\}$a množinu všech funkcí $\mathbf{F_{all}} = \{\mathbf{f_1}, \mathbf{f_2}, ..., \mathbf{f_k}, ...\}$. Pak pro každou ANNi $\in \mathbf{ANN_{all}}$ existuje funkce $\mathbf{f_k} \in \mathbf{F_{all}}$, eventuelně množina funkcí $\mathbf{F_k} \subset \mathbf{F_{all}}$ taková, že platí $ANN_i \Leftrightarrow \mathbf{f_k}$, eventuelně $ANN_i \Leftrightarrow \mathbf{F_k}$. (Více o funkcionální povaze ANN viz kap. 4.1.)

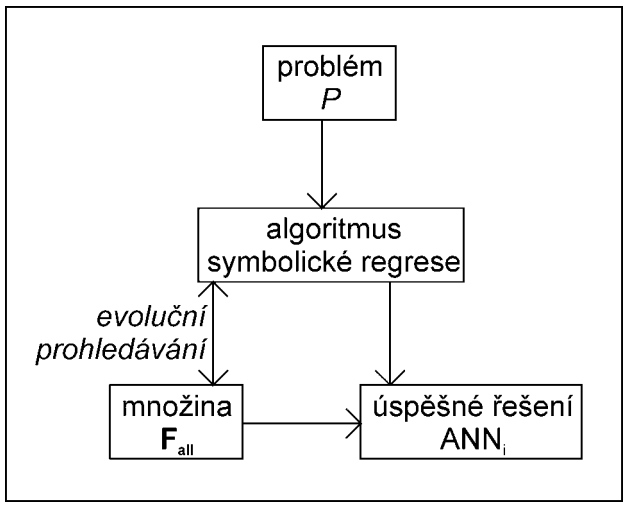
Kolmonogorovův teorém [5] dále ukazuje na platnost obrácené věty: Pro každou spojitou $\mathbf{f_k} \in \mathbf{F_{all}}$ existuje $ANN_i \in \mathbf{ANN_{all}}$ taková, že platí $\mathbf{f_k} \Leftrightarrow$ ANNi.

Úkol 1: Vytvořte algoritmus, který bude pomocí metod symbolické regrese, evolučně prohledávat množinu $\mathbf{F_{all}}$ za účelem nalezení:

a) $\mathbf{f_k} \Leftrightarrow ANN_i$

b) $\mathbf{f_k}$, jejíž alespoň některé subfunkce $\{\mathbf{f_1}, \mathbf{f_2}, ...\} \Leftrightarrow \{ANN_n, ANN_m, ...\}$

takové, že řeší daný problém *P* s globální chybou $E_T < \xi$, kde ξ je uživatelem nastavený práh tolerance.



Obr. 1. Princip evolučního prohledávání

Úkol 2: Proveďte aplikaci tohoto algoritmu na dva vybrané klasifikační problémy *P1* a *P2*.

P1) Lineárně separabilní problém (viz kap. 6.2.1)

P2) Problém XOR (viz kap. 6.3.1)

2 SYMBOLICKÁ REGRESE

Termín **symbolická regrese** reprezentuje proces prokládání předložených dat vhodnou funkcí - matematickou formulí. Po dlouhou dobu byla tato činnost výhradní doménou lidského mozku, ale v několika posledních desetiletích mohou tuto práci vykonávat také počítače. Vedle dvou obecně známých algoritmů Genetického programování (*Genetic programming,* GP) [6] a Gramatické evoluce (*Grammatical Evolution,* GE) [7, 8] prosazuje naše fakulta pod vedením doc. Ing. Ivan Zelinky Ph.D. také vlastní originální řešení, algoritmus s názvem Analytické programování (*Analytic Programming*, AP) [9].

2.1 Analytické programování

Algoritmus AP se úspěšně osvědčil při řešení celé řady problémů symbolické regrese jako jsou: syntéza trigonometrických funkcí [10], polynomiálních funkcí [9], funkce boolovské parity [11], funkce boolovské symetrie [12], řešení diferenciálních rovnic [13] a optimalizace cesty umělého mravence [14]. Tyto úspěchy nás motivovali použít AP také na syntézu ANN. Základní vlastnosti AP, které z něj činí algoritmus k tomuto účelu vhodný, jsou popsány v následujících podkapitolách.

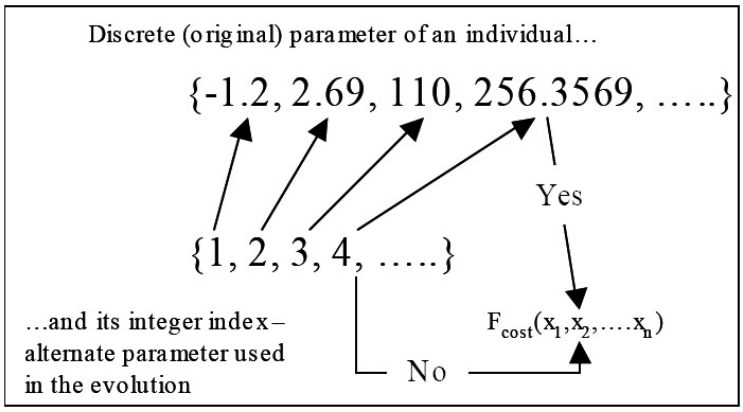
2.1.1 Základní myšlenka AP

AP bylo inspirováno numerickými metodami v Hilbertově funkcionálním prostoru a také GP. Princip AP leží někde mezi těmito dvěma filozofiemi: Z GP přebírá myšlenku evoluční kreace symbolických řešení, zatímco hlavní myšlenka funkcionálního prostoru a budování výsledné funkce pomocí procesu prohledávání je převzata z Hilbertových prostorů. Stejně jako GP či GE je i AP postaveno na množině funkcí, operátorů a takzvaných terminálů, které jsou většinou konstantami či nezávisle proměnnými, například:

- funkce: Exp, Sin, Tan, Tanh, And, Or, ...
- operátory: +, -, ×, /, dt, ...
- terminály: 2.73, 3.14, t, ...

Tyto matematické objekty vytváří množinu, ze které se AP snaží syntetizovat vhodné řešení. Základní princip AP je založen na manipulaci s diskrétními množinami (*Discrete Set Handling*, DSH) [15]. DSH vytváří interface mezi evolučním algoritmem (*Evolutionary*

Algorithm, EA) [16] a problémem, který má být symbolicky vyřešen. Díky tomu může být v AP použít téměř libovolný EA. EA tak vlastně pracuje jako jakýsi motor pohánějící AP.



Obr. 2. Manipulace s diskrétními množinami, DSH

2.1.2 Základní množina - GFS

Množinu všech dostupných matematických objektů - funkce, operátory, terminály - nazýváme základní množinou (*General Function Set,* **GFS**). Prvky **GFS** jsou pak základní funkce **gf**. (Na operátory a terminály můžeme totiž klidně pohlížet jako na funkce.) **GFS** = {$\mathbf{gf}_1$, ... **gf***i,* ... **gf***n*}. Důležitou vlastností AP je, že dále rozděluje **GFS** na několik podmnožin podle počtu argumentů jednotlivých **gf**, jak můžete vidět na následujícím příkladu:

- $\mathbf{GFS_{all}}$ = {+, -, /, Sin, Cos, Exp, Abs, t, x, y, π}
- $\mathbf{GFS_{2arg}}$ = {+, -, /}
- $\mathbf{GFS_{1arg}}$ = {Sin, Cos, Exp, Abs}
- $\mathbf{GFS_{0arg}}$ = {t, x, y, π} - terminály

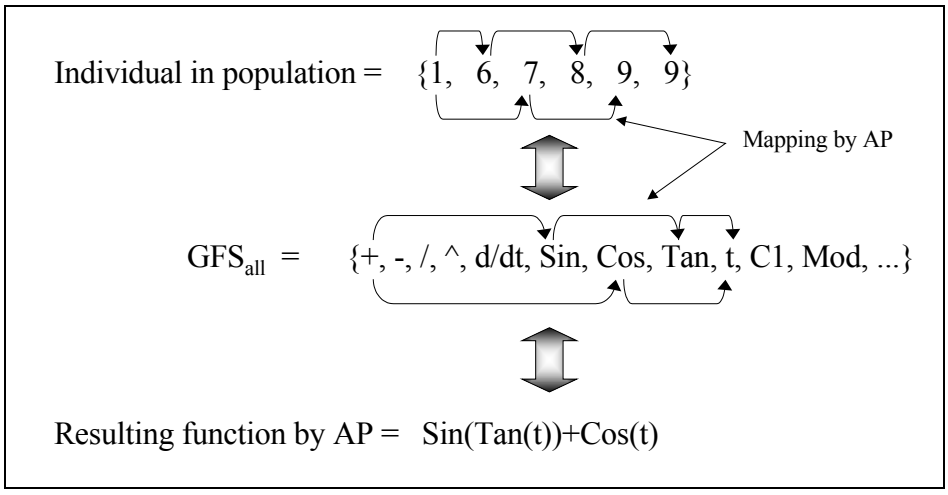
2.1.3 Mapovací operace

Důležitou částí AP je sekvence matematických operací, které jsou použity pro programovou syntézu. Tyto operace transformují jedince (existujícího v EA) na použitelný program (funkci). Matematicky řečeno, jedná se o mapování z prostoru jedinců na prostor programů (funkcí). Takové mapování se skládá ze dvou částí, z DSH a bezpečnostních procedur, které nedovolí vytvářet patologické jedince.

DSH vytvoří celočíselný indexer, který namapuje v podstatě libovolné matematické objekty obsažené v **GFS** na jedince použitelného v EA. Tento indexer je tedy v podstatě vekto-

rem ukazatelů na jednotlivé prvky **GFS**. EA tak vnímá jedince jako vektor diskrétních hodnot a pouze pro potřebu jeho ohodnocení se provede přemapování na konkrétní funkci, která je použita k výpočtu vhodnosti jedince.

AP je tedy v podstatě sérií funkčních mapování. Přitom je ale nezbytné dodržet pravidla, která zajistí, že každý jedinec bude reprezentovat jednoznačnou a nedefektní funkci. Jak takové mapování v praxi funguje nám nejlépe osvětlí následující příklad:



Obr. 3. Mapovací operace v AP

Číslo 1 na pozici prvního parametru znamená, že je použit operátor "+" ∈ $\mathbf{GFS_{all}}$. Protože operátor "+" musí mít právě dva argumenty, ukazatele 6 (pro "Sin" ∈ $\mathbf{GFS_{all}}$) a 7 (pro "Cos" ∈ $\mathbf{GFS_{all}}$) jsou pro tento operátor určeny jako jeho argumenty. Obě tyto funkce Sin i Cos jsou funkcemi jedné proměnné takže další nepoužité ukazatele 8 (pro "Tan" ∈ $\mathbf{GFS_{all}}$) a 9 (pro "t" ∈ $\mathbf{GFS_{all}}$) jsou určeny jako argumenty funkcí Sin a Cos. Protože je jako argument pro Cos zvoleno t, je tato větev výsledné funkce uzavřena (t nemá žádný argument). Zůstává nám funkce jednoho argumentu Tan a jelikož další nepoužitý ukazatel je opět 9, Tan je namapován na "t", které je na 9. pozici v **GFS**.

2.1.4 Evoluční operátory

V průběhu evoluce dané populace možných řešení se používá různých evolučních operátorů, jako jsou například křížení a mutace. AP je ovšem sestaveno tak robustně, že ve své struktuře žádné z těchto operací nepředjímá. Jednotlivé kroky evoluce - mutace, křížení, soutěžení, selekce, aj. - jsou plně v kompetenci použitého EA. EA tak vlastně funguje jako jakýsi „mixér", který mezi sebou míchá jednotlivé prvky **GFS**.

Z tohoto faktu vyplývá, že naši úspěšnost při hledání řešení pomocí AP můžeme na straně AP ovlivnit pouze vhodnou volbou **GFS**. Na straně EA je pak množina uživatelem nastavovaných parametrů odvislá od konkrétního EA. Jeden klíčový argument však musí mít všechny EA stejný. Je jím daná účelová funkce (*Cost Function, Fitness Function*, **CF**), kterou je třeba minimalizovat. Klíčovou dvojicí parametrů AP je tedy **GFS** a **CF**.

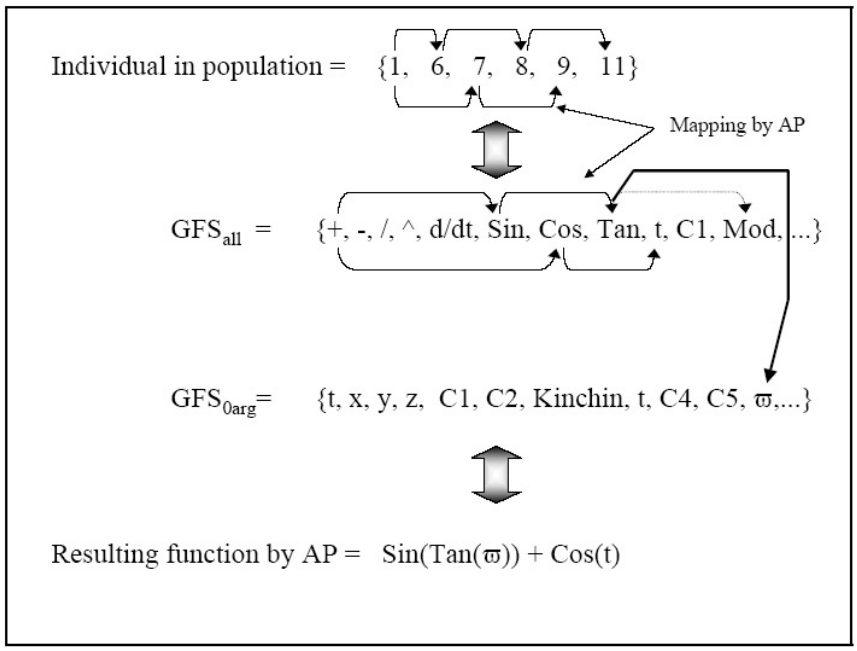
2.1.5 Posílená evoluce

V průběhu evoluce jsou syntetizováni méně či více úspěšní jedinci. Některé jedince přitom můžeme použít k posílení evoluce, abychom získali ještě lepší řešení. Hlavní myšlenka spočívá v přidání již vytvořeného a částečně úspěšného jedince do množiny terminálů. Rozhodnutí, který jedinec bude takto použit je závislé na uživatelem definovaném prahu.

Například pokud je práh nastaven na 5 a všichni jedinci v populaci mají větší hodnotu **CF** než 5, běží evoluce dál na základní **GFS**. Pokud je ovšem nejlepší jedinec v aktuální populaci lepší než 5, pak je kompletně přidán do **GFS** a je označen jako terminál. Od této chvíle pracuje AP s rozšířenou **GFS** obsahující částečně úspěšná řešení. Díky tomu je posílené AP schopno najít vhodné řešení mnohem rychleji než AP bez posílené evoluce. Tento fakt byl verifikován mnoha tisíci úspěšných simulací prováděných na nejrůznějších řešených problémech.

2.1.6 Bezpečnostní procedury

Abychom zabránili syntéze patologických funkcí, musí AP obsahovat několik bezpečnostních opatření. To nejdůležitější je založeno na skutečnosti, že GFS je rozdělena do podmnožin obsahujících **gf** o stejném počtu argumentů. Díky tomu je možné vytvořit speciální bezpečnostní proceduru, která měří jak daleko od konce jedince se AP při mapování nalézá a podle toho volí z vhodné podmnožiny **GFS**, aby tak předešla patologické - neuzavřené - funkci. Přesněji, pokud je funkcí požadováno více argumentů než kolik ukazatelů jedinec obsahuje (blíží se konec jedince). Je vybrána funkce z podmnožiny funkcí s menším počtem argumentů. Stejný ukazatel je přesměrován z $\mathbf{GFS_{all}}$ například do $\mathbf{GFS_{0arg}}$, takovýto případ je zobrazen na následující stránce.



Obr. 4. Bezpečnostní procedura AP

S bezpečností je třeba počítat také při definici **CF** a v případě potřeby definovat další bezpečnostní funkce přímo jako její součást.

2.1.7 Použití nelineárního prokládání v AP

Algoritmus AP byl vytvořen na vysoce efektivní softwarové platformě **Mathematica** vyvíjené společností Wolfram Research, Inc., od roku 1987. [17] (Veškerý zdrojový kód, se kterým jsme v této práci pracovali je tedy na této platformě rovněž závislý.)

Obr. 5. Logo Mathematica

To umožnilo implementovat do AP pokročilé programátorské a matematické techniky, které toto prostředí obsahuje. Především metody nelineárního prokládání (*non-linear fitting, non-linear regress*), jež jsou standardní součástí prostředí **Mathematica** obsaženou ve statistické knihovně - **Statistic`NonLinearFit`**. Použití této metody je velmi jednoduché a z programátorského hlediska je výhodné, že vyžaduje minimální interakci s uživate-

lem. Stačí předložit libovolný matematický model s příslušnými konstantami, které hledáme a data, kterými má být tento model proložen. Přitom je ovšem nutné zajistit, aby měl model na celém intervalu první derivaci a aby počet hledaných konstant, byl větší než počet nezávisle proměnných v modelu. Počítač pak již sám nalezne metodou nejmenších čtverců hodnoty daných konstant. Pokud ovšem řešený problém nepřekročí určitou mez.

V pokročilých verzích AP je tedy upuštěno od definice konkrétních konstant jako terminálů (pokud to není konkrétně vyžadováno) a místo nich jsou definovány obecné konstanty K1 až Kn, jejichž konkrétní hodnota je určena pomocí nelineárního prokládání až těsně před ohodnocením příslušné **CF**. Tento postup je dalším výrazným zefektivněním algoritmu AP.

3 EVOLUČNÍ ALGORITMY

Jak již bylo řečeno EA nám slouží jako srdce pro AP. Pokud nepoužijeme kvalitní (a kvalitně nastavený) EA, nebude nám ani AP pracovat kvalitně. Z tohoto důvodu je v naší práci věnována také zvýšená pozornost dvěma následujícím EA, které jsme oba v různých variantách úspěšně použili v AP pro syntézu ANN.

3.1 SOMA

SamoOrganizujícího se Migračního Algoritmu (*Self-Organizing Migration Algorithm,* SOMA) [3] vytvořil I. Zelinka v roce 1999. SOMA se velmi rychle zařadil mezi ty nejúčinnější a nejrobustnější EA, což ho činí velmi vhodným pro použití v AP.

3.1.1 Princip SOMA

Vznik SOMA byl inspirován soutěživě-komparativním chováním inteligentních jedinců řešících společný problém. Chování tohoto typu lze objevit prakticky kdekoli na světě. Jako příklad lze použít chování smečky lovících vlků, včelího úlu, hejna holubů (*J. Nash*) apod. U těchto příkladů je společným úkolem např. hledání potravy, v rámci níž jedinci spolupracují, ale i, byť nevědomky, soutěží. Ve fázi spolupráce si navzájem jednotliví jedinci sdělují jakou kvalitu hledaného momentálně nalezli a na základě toho se snaží přizpůsobovat své chování. Ve fázi soutěžení (předcházející fázi spolupráce) se každý jedinec snaží vyhrát nad ostatními - snaží se nalézt co nejlepší zdroj potravy apod. Po ukončení fáze soutěže nastane opět fáze spolupráce a jedinci si vymění informace o tom, který z nich má nejlepší zdroj potravy. Ostatní opustí své nalezené zdroje potravy a migrují (fáze soutěžení) směrem k jedinci s nejlepším zdrojem potravy a během této migrace se snaží nalézt ještě lepší zdroj. To se opakuje dokud se všichni nesejdou u nejvydatnějšího zdroje potravy. Na tomto silně zjednodušeném principu funguje i algoritmus SOMA.

Tab. 1. Význam biologické terminologie algoritmu SOMA

Biologická realita	**Počítačová implementace**
členové smečky, společenství	jedinci v populaci, parametr NP
člen společenství s nejlepším zdrojem potravy	Leader, vedoucí aktuálního migračního kola
potrava	vhodnost, hodnota účelové (kriteriální, cenové) funkce, geometricky je to lokální či globální extrém na *N* rozměrné hyperploše
životní prostor společenství	hyperplocha daná účelovou funkcí
migrace členů společenství v životním prostředí	migrační kola v algoritmu SOMA

3.1.2 Fáze SOMA

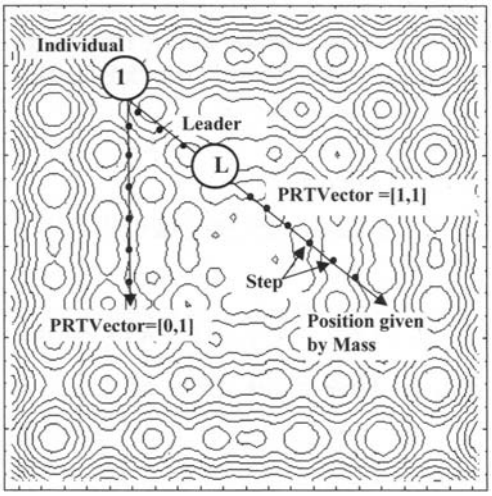
V následujícím textu se podíváme na práci algoritmu SOMA poněkud detailněji. Rozebereme základní verzi SOMA - strategii AllToOnem - ta se skládá z následujících kroků:

1. Definice parametrů. Před startem SOMA je nadefinování řídících a ukončovacích parametrů (Specimen, Step, Mass, Accepted Error, NP, PRT a Migrations). Nezbytně nutné je rovněž nadefinovat účelovou funkci **CF**, která bude optimalizována. Ta slouží pro jedince z populace jako jakési životní prostředí.

2. Tvorba populace. V tomto kroku je vytvořena prvopočáteční populace. Za pomoci specimenu a generátoru náhodných čísel je pro každý parametr jedince generováno náhodné číslo.

3. Migrační kola. Každý jedinec je ohodnocen **CF** a je zvolen Leader (jedinec s nejlepší hodnotou účelové funkce) pro následující migrační kolo. V tomto se začnou ostatní jedinci pohybovat směrem k Leaderovi pomocí skoků, jejichž velikost je dána parametrem Step. Po každém skoku si každý jedinec na takto získané nové pozici přepočítá svou hodnotu **CF** a pokud je lepší nežli předchozí, tak si ji zapamatuje. Pohyb jedince směrem k Leaderovi po skocích pokračuje tak dlouho, dokud není dosaženo pozice, jež je dána parametrem Mass. Po ukončení běhu se jedinec vrací na pozici, kde byla nalezena nejlepší hodnota **CF** během jeho cesty.

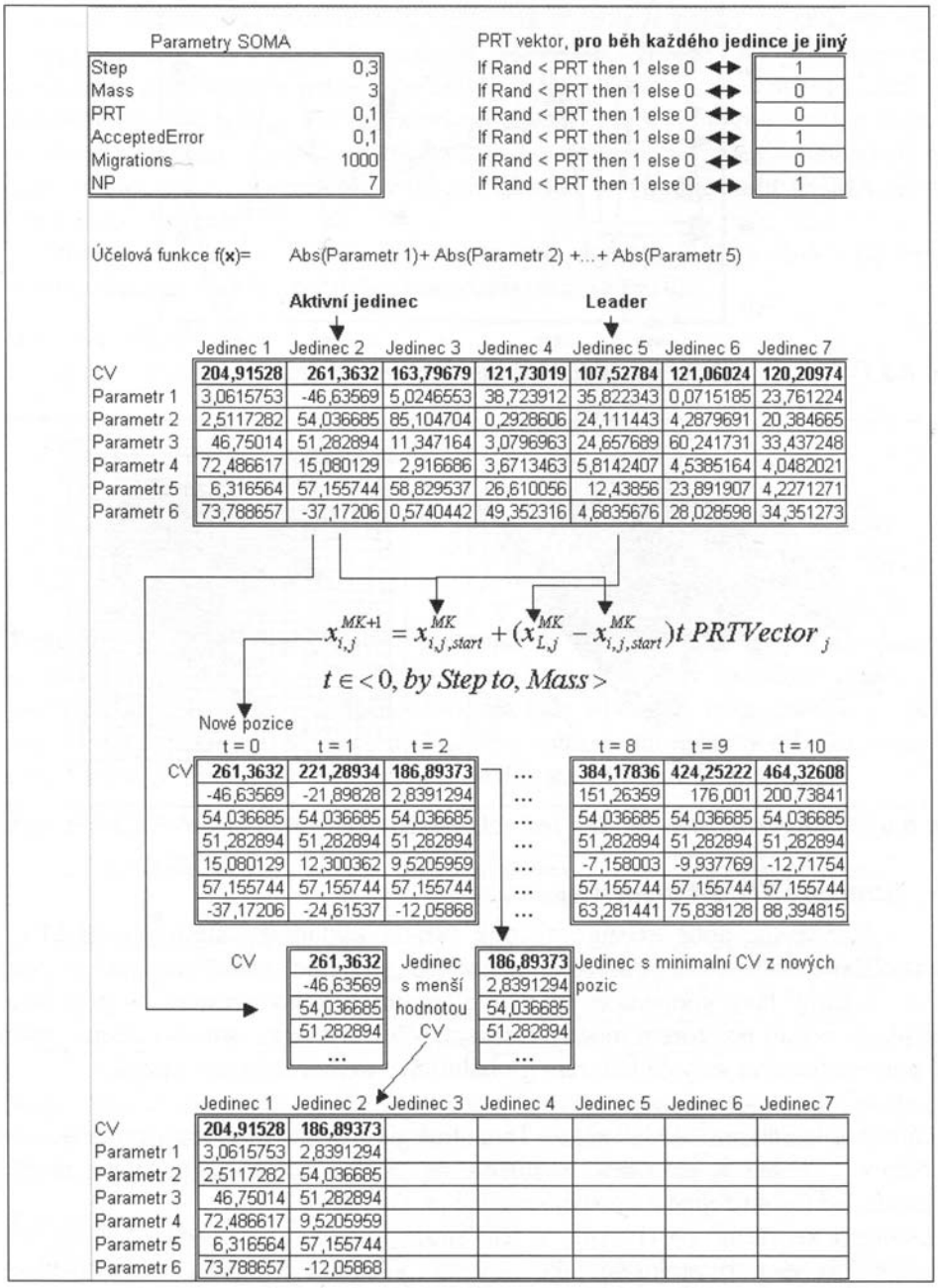
Předtím, nežli daný jedinec započne svou cestu směrem k Leaderovi, je vygenerován prázdný PRTVector o dimenzi = D včetně vygenerované sekvence náhodných čísel, jejichž počet je roven D. Ty jsou porovnány s parametrem PRT. Jestliže je n-té vygenerované číslo větší nežli PRT parametr, pak je n-tý parametr PRTVectoru nastaven na 0 a v opačném případě na 1. Parametry jedince, které jsou takto nastaveny na 0 se nepřepočítávají, jsou zmraženy - snižuje se počet stupňů volnosti pohybu jedince. Tento proces nahrazuje mutaci známou u jiných evolučních algoritmů. Díky tomu se rapidně zvyšuje robustnost SOMA algoritmu ve smyslu nalezení globálního extrému.



Obr. 6. Funkce PRTVectoru

4. Testování naplnění ukončovacích parametrů. V tomto okamžiku je kontrolováno, zda je rozdíl mezi Leaderem (nejlepší jedinec) a nejhorším jedincem menší nežli AcceptedError. Rovněž je kontrolováno, zda byly vykonány migrační cykly v počtu, jenž je dán parametrem Migrations. Pokud není splněna ani jedna podmínka, proces se vrací na krok 3.

5. Stop. Návrat nejlepšího jedince - nejlepší nalezené řešení po posledním migračním kole.



Obr. 7. Princip SOMA: jedno migrační kolo

3.1.3 Strategie SOMA

V současné době existuje několik variací základního algoritmu SOMA, pro jejichž obecné označení se používá rovněž výraz „strategie“.

„Všichni k jednomu“ (AllToOne). Tato strategie již byla popsána v předchozí sekci. Název „Všichni k jednomu“ znamená, že všichni jedinci z populace migrují k Leaderovi.

„Všichni ke všem“ (AllToAll). V této strategii neexistuje Leader. Všichni jedinci migrují ke všem ostatním. I když je tato strategie výpočetně náročnější, je zde vyšší pravděpodobnost, že bude nalezen globální extrém.

„Adaptivně všichni ke všem“ (AllToAllAdaptive). Tato strategie je totožná se strategií „Všichni ke všem“ s tím rozdílem, že aktuálně migrující jedinec se přesouvá do nové pozice po každé aktuálně dokončené migraci. Z této pozice pak provádí migraci k dalším zbývajícím jedincům.

„Všichni k jednomu náhodně“ (AllToOneRand) je strategie ve které se všichni jedinci pohybují opět k jednomu Leaderovi, který ovšem není určen nejhlubší pozicí na hyperploše, ale je pro migraci každého jedince náhodně vybrán z populace.

„Svazky“ (Clusters). SOMA s vytvářením svazků je úprava, která se dá použít na kteroukoliv předchozí strategii. Jedinci účastnící se migračního procesu jsou rozděleni do tzv. svazků. V každém z nich pak probíhá samostatný SOMA. Vzhledem k tomu, že se jedinci pohybují, mohou se svazky spojovat a rozpadat, což ještě více potrhuje synergetický proces tohoto algoritmu.

3.2 Diferenciální evoluce

Diferenciální evoluce (*Differential Evolution*, DE) je stejně jako SOMA poměrně nový typ EA (od r. 1995), který vyvinul a poprvé použil Ken Price a Rainer Storm. Jeho schéma je dost podobné algoritmům genetickým, s nimiž má několik společných rysů, jako je například tvorba potomků (zde však pomocí 4 rodičů a ne 2 jak je tomu u genetických algoritmů), používání tzv. generací apod.

Cílem diferenciální evoluce je v cyklech zvaných „generace“ vyšlechtit co nejlepší populaci jedinců ve smyslu hodnot **CF**, jenž je spojena s každým jedincem. Během každé generace se provádí následující kroky:

1. Stanovení parametrů - jde o parametry, které určují chod evoluce. Jsou to parametry F - mutační konstanta <0, 2>, CR - práh křížení <0, 1>, NP - počet jedinců v populaci, D - rozměr jedince (jsou to v podstatě argumenty účelové funkce). Dále je nutné nadefinovat prototyp jedince - Specimen.

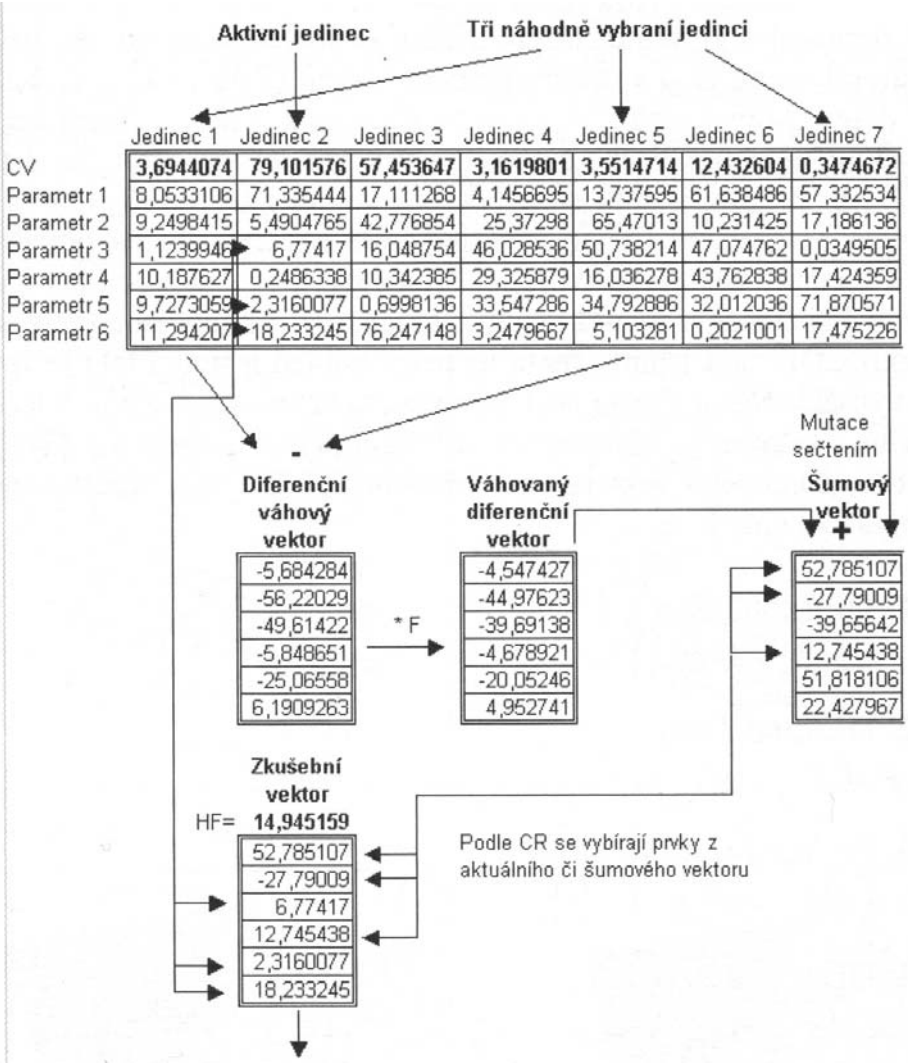
2. Tvorba populace - populace se tvoří vygenerováním množiny jedinců (matice) podle prototypového (Specimen) vektoru. U každého jedince se musí počítat s jedním prvkem navíc a tím je hodnota **CF**.

3. Započetí cyklu generace - během každé generace se provádí ještě cyklus, který zabezpečuje postupné evoluční šlechtění každého jedince z populace. V tomto cyklu se postupně vybírá jeden jedinec za druhým až do konce populace a pro každého z nich je proveden následující evoluční cyklus.

4. Evoluční cyklus - v tomto cyklu je prováděna mutace a křížení. Vedle cílového vektoru se náhodně zvolí tři další různé vektory (jedinci) z populace. První dva se od sebe odečtou a získá se tak tzv. diferenční vektor. Ten se vynásobí mutační konstantou „*F*“, která jej tím pádem změní (zmutuje), a získá se „váhovaný diferenční vektor“. Ten se přičte k třetímu náhodně vybranému vektoru a získá se tzv. „šumový vektor“. Poté se připraví tzv. „zkušební vektor“ a z cílového a šumového vektoru se bere postupně jeden prvek za druhým a pro takto vybranou každou dvojici se generuje náhodné číslo v rozsahu 0-1 a porovnává s konstantou CR. Pokud je toto číslo menší než CR, pak se do příslušné pozice v tzv. „zkušebním vektoru“ umístí prvek z vektoru šumového a v opačném případě z vektoru cílového. Tak se získá zkušební vektor, jehož hodnota účelové funkce se porovná s hodnotou účelové funkce cílového vektoru. Na pozici cílového vektoru v nové populaci je vybrán ten vektor (jedinec), který má hodnotu **CF** lepší. Tím je zajištěno, že se do nové generace dostanou jedinci s lepšími nebo stejnými vlastnostmi. Celý evoluční cyklus se opakuje až do vyčerpání populace.

5. Testování naplnění ukončovacích parametrů - diferenciální evoluce je ukončena pouze tehdy, provede-li se uživatelem zadaný počet generací. Jiný ukončovací parametr tento algoritmus nemá.

6. Vyhodnocení - celý proces generací se opakuje, dokud není vyčerpán zadaný počet generací. Během každé generace se uschová hodnota účelové funkce nejlepšího jedince do vektoru historie, který po ukončení znázorňuje průběh evolučního procesu.



Obr. 8. Princip diferenciální evoluce

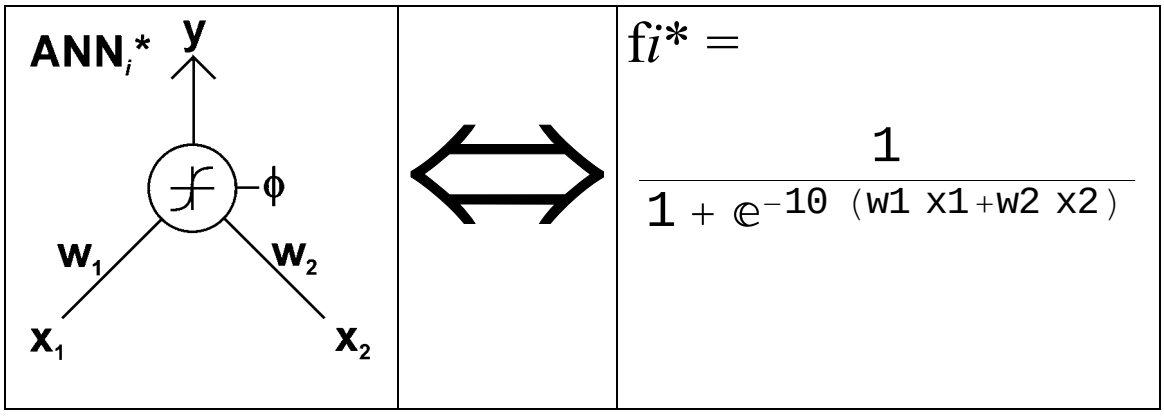
4 SYNTÉZA NEURONOVÝCH SÍTÍ

Abychom dokázali pomocí AP úspěšně syntetizovat ANN, je třeba nejprve vhodně zvolit prvky **GFS** a správně definovat **CF**, příslušnou k řešenému problému. Ke splnění těchto dvou kroků je ovšem nezbytně nutné pochopit vztah ekvivalence mezi ANN a funkcemi - AP totiž samo o sobě pojmu ANN nerozumí, dokáže syntetizovat pouze funkce, eventuelně algoritmy. Následující kapitoly se zabývají právě problematikou přizpůsobení AP k syntéze ANN.

4.1 Vztah mezi ANN a matematickými funkcemi

Spojitost mezi ANN a matematickými funkcemi je v odborné literatuře často zanedbávána, přitom každou ANN lze vyjádřit jako jí ekvivalentní třídu funkcí **f***. Proces syntézy ANN je tedy hledáním vhodné **f*** z množiny tříd funkcí **F**, jež obsahuje třídy vzniklé kombinacemi (základních) bázových funkcí **b** množiny **B**. Složení prvků **B** se pak liší podle jednotlivých typů ANN. Proces učení ANN se dá vyjádřit jako hledání vhodných hodnot konstant, které od sebe odlišují jednotlivé funkce **f** z **f***. Výsledná funkce **f** je potom ekvivalentní s hledanou ANN.

Poznámka: *Zaveďme úmluvu - mluvíme-li obecně o neuronových sítích, které jsou již úspěšně naučeny a nebo není významný fakt jsou-li v dané chvíli naučeny, používáme nadále zkratky* ANN. *Chceme-li ale zdůraznit, že se jedná o zatím nenaučenou síť, používáme zkrácený tvar* ANN*.



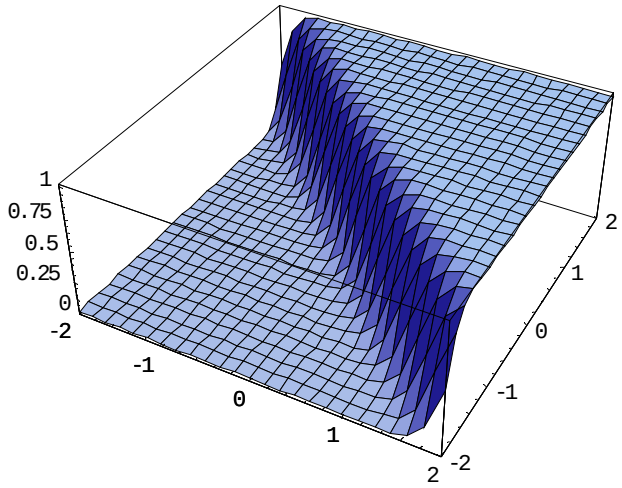
Obr. 9. Ekvivalence mezi ANNi a fi**

Tato vlastnost ANN společně s výše uvedenými vlastnostmi AP, vytváří z AP ideální algoritmus pro syntézu ANN. Nejdůležitějším úkolem je zde zvolit vhodnou **B**, kterou použijeme jako **GFS.**

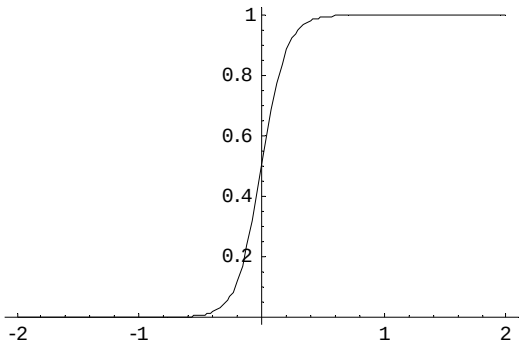
4.2 Koncepce použité GFS

Z předchozího textu jasně vyplývá, že vhodná volba jednotlivých **gf** v námi použité **GFS**, je pro nás naprosto klíčovou otázkou. Jako základní stavební kámen - funkci **gf1** - našich syntetizovaných ANN, jsme zvolili **sigmoidu** - přenosovou funkci dvou vstupních argumentů ***arg1***, ***arg2***:

$$\mathbf{gf1} = \frac{1}{1 + e^{-\lambda\,(\boldsymbol{arg1} + \boldsymbol{arg2})}}$$



Obr. 10. Přenosová funkce sigmoida

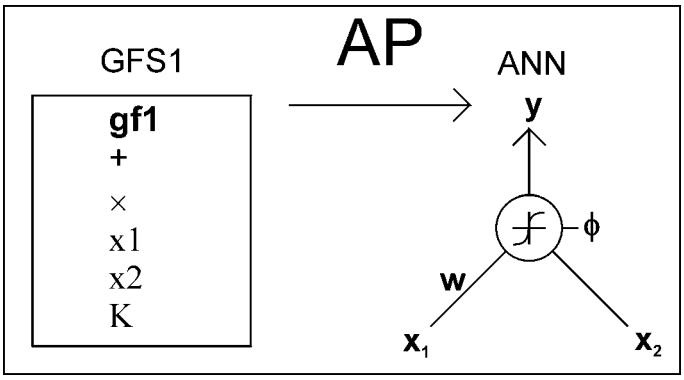


Obr. 11. Řez sigmoidou

Sigmoida je ve světě ANN často používanou přenosovou funkcí [18], která je odvozena z chování skutečných neuronů v lidském mozku [1]. Za parametr λ jsme zvolili konstantu 10, takže se naše sigmoida blíží spíše unipolární binární funkci, která je vhodná k řešení klasifikačních problémů. Oproti binární funkci má ovšem tu výhodu, že má derivaci na celém svém intervalu a na struktury, které díky ní vznikají, lze tedy aplikovat algoritmy na derivaci funkce závislé, jako je třeba právě nelineární prokládání.

Kromě **gf1** musí být nedílnou součástí naší konstituované **GFS** také vstupní nezávislé proměnné x1 a x2, které reprezentují souřadnice jednotlivých bodů na klasifikované ploše, a dále pak konstanty K (K1 až Kn), které budou vystupovat jako prahy ϕ a váhy w sítě. Aby bylo možno váhy a prahy na síť správně navázat, je nutné do **GFS** přidat ještě dvě **gf,** + (plus) a × (krát).

Tímto způsobem jsme konečně získali **GFS1** = {**gf1**, +, ×, x1, x2, K}. Z hlediska našeho problému se jedná o tzv. **úplnou GFS,** tedy **GFS1** má pomocí AP zaručený potenciál řešit syntézu ANN - pro množinu **B,** jejíž kombinací vzniká již známe řešení ANN daného problému, platí **B** $\subseteq$ **GFS1** - tedy obsahuje všechny prvky, ze kterých se skládají příslušné **f**.



*Obr. 12. Princip syntézy ANN z **GFS1***

Taková definice **GFS1** je pro nás atraktivní také tím, že díky ní může AP syntetizovat nejen klasické neuronální struktury, známé z rigorózní teorie ANN, ale také sítě pseudoneuronálního charakteru, eventuelně zcela ne-neuronální funkce. Vlastnosti ze světa neuronových funkcí se tak budou díky AP mimoděk kombinovat s dalšími matematickými transformacemi. Tuto otázku hlouběji objasní následující kapitola.

4.3 Vliv GFS1 na topologii syntetizované ANN

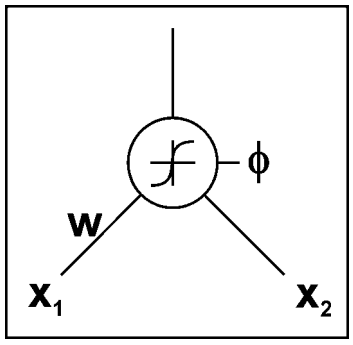
Z množiny **GFS1** mohou za příznivých podmínek v průběhu šlechtění jedince v AP vznikat subfunkce, obsažené ve vítězném jedinci, na něž se můžeme dívat jako na neurony v ANN. Příklady takového uspořádání jednotlivých **gf** $\in$ **GFS1** do vhodné subfunkce jsou uvedeny v následujících třech podkapitolách. Podobným způsobem vznikají všechny neurony, které jsou vyobrazeny jako topologie syntetizovaných ANN v praktické části práce.

4.3.1 Klasický neuron (sigmoida)

Vhodným uspořádáním prvků {**gf1**, +, ×, x1, x2, K } = **GFS1** vznikne subfunkce:

$$\frac{1}{1 + e^{-10(x2+x1K[1]+K[2])}}$$

Takovou subfunkci bychom v rigorózní teorii ANN považovali za klasický neuron a můžeme ji topologicky zobrazit podle následujícího obrázku:



Obr. 13. Klasický neuron

Poznámka: *Ve skutečnosti je na obrázku znázorněna místo funkce sigmoidy funkce hyperbolický tangens. Důvody pro tuto skutečnost jsou ovšem čistě estetické. Platí úmluva, že každý takto zobrazený neuron bude obsahovat jako přenosovou funkci* ***gf1****, tedy sigmoidu.*

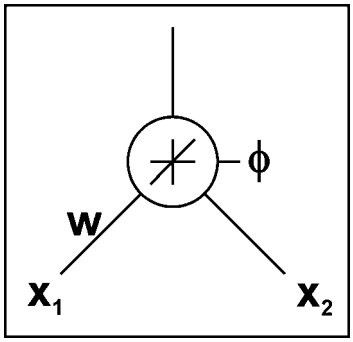
4.3.2 Lineární neuron

Vhodným uspořádáním prvků {+, ×, x1, x2, K } ⊂ **GFS1** vznikne subfunkce:

$$K[1] \times x1 + x2 + K[2]$$

Tuto subfunkci bychom v rigorózní teorii ANN považovali za lineární neuron, neuron jehož přenosovou funkcí je přímka se sklonem 45°. (Sklon a posunutí přímky dále ovlivňují konstanty K.)

Takto vzniklý neuron můžeme topologicky zobrazit podle následujícího obrázku:



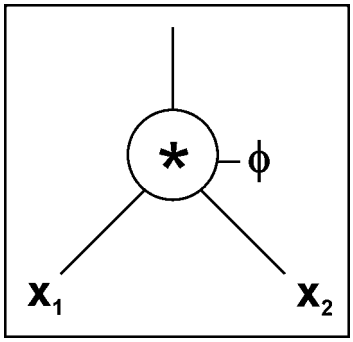
Obr. 14. Lineární neuron

4.3.3 Násobící neuron (pseudoneuronální funkce)

Vhodným uspořádáním prvků {×, x1, x2, K } ⊂ **GFS1** vznikne subfunkce:

$$x1 \times x2 \times K[1]$$

S něčím takovým se v rigorózní teorii ANN nesetkáváme. Jedná se o subfunkci, která provádí operaci násobení, jež v oblasti klasických ANN prostě neexistuje. Takovýto výlet za hranice pravidel konstrukce standartních ANN ovšem nemusí být nutně na škodu. Proto jsme se rozhodli dopustit se heretické myšlenky a zavést v našich topologiích následující násobící neuron. Tedy spíše pseudoneuron, který se v syntetizovaných ANN stará o operaci násobení - jeho přenosovou funkcí je ×, násobení konstantou je pak chápáno jako práh tohoto neuronu.



Obr. 15. Násobící neuron

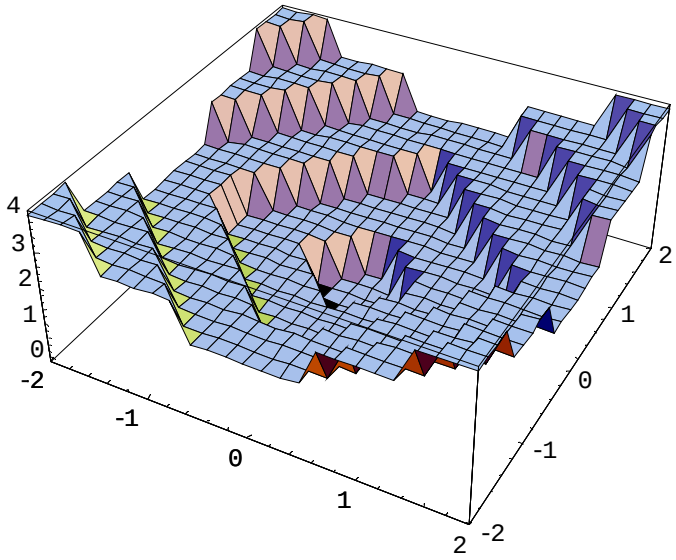
4.4 Definice CF

Správná definice **CF** je unikátní pro každý řešený problém. **CF** v sobě totiž musí obsahovat celou testovací množinu syntetizované ANN. **CF** v tomto případě pracuje na bázi tes-

tování odezvy ANN na jednotlivé prvky testovací množiny. Co zůstává vždy stejné je zvolená filozofie, se kterou příslušnou **CF** vytváříme.

4.4.1 Schodovitý model CF1

Jako základní model CF1 jsme pro naši syntézu zvolili tzv. schodovitou koncepci **CF**. Ta pracuje tak, že na začátku vyhodnocování **CF** příslušného jedince (ANN) platí **CF** = 0. Postupně se prochází všechny prvky testovací množiny. Pokud se u nějakého testovaného prvku odezva ANN neshoduje s požadavky testovací množiny provede se **CF** = **CF** + 1. Tímto způsobem může CF1 nabývat diskrétních hodnot {1,2, ..., n}, kde n je počet prvků testovací množiny. CF1 tedy počítá na kolika prvcích testovací množiny ANN odpověděla nepřípustnou odezvou.



Obr. 16. Příklad potenciální CF1 v řezu

Následuje příklad definice **CF** podle CF1 pro řešení klasifikačního problému XOR (viz kap. 6.3.1). Příklad je zapsán v programovém prostředí **Mathematica**. Přitom se předpokládá, že na výstupu dané ANN je zařazen ještě jeden neuron, který odezvy větší něž 0.8 normuje na 1 a odezvy menší než 0.2 utlumí na 0.

```
CF = Module[{chyba}, chyba = 0;
   If[(ANN /. {x1 → 1, x2 → 1}) < 0.8, chyba++];
   If[(ANN /. {x1 → -1, x2 → -1}) < 0.8, chyba++];
   If[(ANN /. {x1 → -1, x2 → 1}) > 0.2, chyba++];
   If[(ANN /. {x1 → 1, x2 → -1}) > 0.2, chyba++];
   chyba
  ];
```

Model CF1 se osvědčil a byl použit pro syntézu všech ANN uváděných v praktické části práce.

4.4.2 Spojitý model CF2

Vedle modelu CF1 jsme také experimentovali s variantou CF2. Tento postup nepočítá počet špatně ohodnocených prvků testovací množiny. Ale změří vzdálenost odezvy ANN na daný prvek testovací množiny od odezvy požadované. Přitom se, stejně jako u předchozího modelu, předpokládá, že na výstupu dané ANN je zařazen ještě jeden neuron, který odezvy větší něž 0.8 normuje na 1 a odezvy menší než 0.2 utlumí na 0. Například pokud je požadovaná odezva nejméně 0.8 (jednička) a reálná odezva je 0.4, pak jejich vzdálenost je |0.4 - 0.8|. Výslednou **CF** je suma vzdáleností od všech prvků testovací množiny.

Pokud bychom chtěli v prostředí **Mathematica** definovat **CF** pro klasifikační problém XOR (viz kap. 6.3.1) podle modelu CF2, mohl by jeho programový kód vypadat například takto:

```
CF = Module[{chyba}, chyba = 0;
   If[(ANN /. {x1 → 1, x2 → 1}) < 0.8, chyba += Abs[(ANN /. {x1 → 1, x2 → 1}) - 0.8] ];
   If[(ANN /. {x1 → -1, x2 → -1}) < 0.8, chyba += Abs[(ANN /. {x1 → -1, x2 → -1}) - 0.8]];
   If[(ANN /. {x1 → -1, x2 → 1}) > 0.2, chyba += (ANN /. {x1 → -1, x2 → 1}) - 0.2];
   If[(ANN /. {x1 → 1, x2 → -1}) > 0.2, chyba += (ANN /. {x1 → 1, x2 → -1}) - 0.2];
   chyba
  ];
```

Výsledná **CF** pak nenabývá pouze několik diskrétních hodnot, ale **CF** $\in (0, \infty)$. Zatímco za použití CF1 mohlo v EA existovat více jedinců se stejnou hodnotou **CF**. V případě CF2 je to silně nepravděpodobné (samozřejmě s výjimkou hodnoty 0). CF2 tak ovšem pro EA vytvoří podstatně složitější prohledávanou hyperplochu.

S modelem CF2 jsme také experimentovali a prokázali jsme, že dokáže nalézat kvalitativně stejná řešení jako CF1. Ale protože při hledání řešených problémů (viz praktická část práce) se CF1 prokázal jako rychlejší, rozhodli jsme se nadále v této práci používat pouze CF1.

4.5 Redukce GFS1 na GFS2

Protože **GFS1** často vede k syntéze neneuronálních prvků (viz kap. 4.3.3) rozhodli jsme se vytvořit alternativu, která bude pro srovnání syntetizovat sítě bez těchto aditivních prvků (viz kap. 6.4). Toho dosáhneme tím, že sloučíme **gf3** a **gf5** z **GFS1** = {**gf1**, +, ×, x1, x2, K}. Vznikne tak nová **gf:** ×K. To znamená, že jedním z argumentů funkce krát je povinně konstanta K. Díky tomu nemohou v syntetizovaných ANN vznikat násobící pseudoneurony. Nová **gf** může tak sloužit pouze k syntéze vah uvnitř ANN. Vzniká tak **GFS2** = {**gf1**, +, ×K, x1, x2}.

II. PRAKTICKÁ ČÁST

5 VÝZKUM METOD UČENÍ NEURONOVÝCH SÍTÍ

Zásadní otázkou úspěšné syntézy ANN pomocí AP je problematika jejich učení. Pokud nalezneme ANN*, která je potenciálně velmi vhodnou k řešení daného problému, ale nedokážeme tuto ANN* úspěšně naučit na ANN, je nám potom ANN* k ničemu a nezbývá něž ji vyřadit jako nepoužitelnou. Ba co hůř, dokonce se o takové ANN* ani nedozvíme do jaké míry pro nás byla potenciálně vhodnou.

Klasické ANN s dopředným šířením jsou většinou úspěšně učeny pomocí algoritmu Back-propagation [19]. Tento rigorózní algoritmus je ovšem v našich podmínkách zcela nepoužitelný, protože nestandardní topologie, jež AP syntetizuje, nám neumožňují jeho aplikaci. Chceme-li se syntézou ANN uspět, musíme hledat takový algoritmus, který dokáže s vysokou účinností naučit ANN libovolné topologie.

V úvahu připadají dva známé přístupy, jednak učení ANN pomocí EA, přístup, který již dříve prokázal svoji robustnost [3], a jednak učení ANN pomocí metod nelineárního prokládání (*non-linear fitting, non-linear regress*), tedy postupem, který je v AP standardně implementován pro nalezení ideálních hodnot syntetizovaných konstant.

Hlavní otázka, kterou je třeba v této kapitole zodpovědět, tedy zní: Jak složitou ANN aplikovanou na jak složitý problém budeme pomocí nelineárního prokládání, nebo EA, schopni naučit? Následující popsané experimenty jsou dostupné a opakovatelné na přiloženém CD.

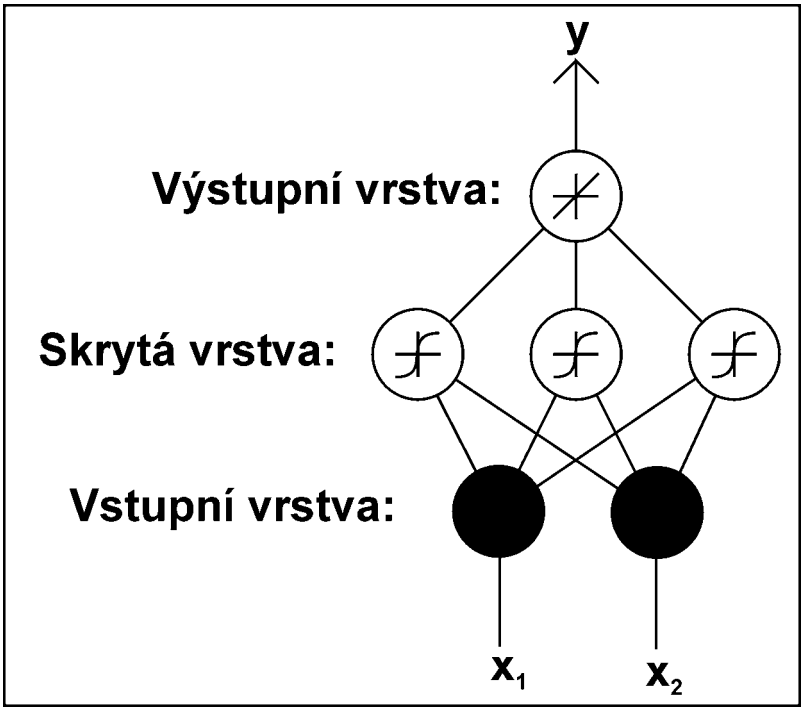
5.1 Použité sítě

Rozhodli jsme se provádět pokusy na dvou běžně používaných ANN. Obě ANN jsou použity na problém funkční aproximace funkcí dvou nezávislých proměnných x1 a x2. ANN(2-3-1) má 3 neurony v jedné skryté vrstvě. ANN(2-3-2-1) má 3 a 2 neurony ve dvou skrytých vrstvách. Neurony ve vstupních vrstvách slouží pouze k distribuci vstupních hodnot na skrytou vrstvu. Neurony ve skrytých vrstvách mají jako přenosovou funkci sigmoidu. Neuron ve výstupní vrstvě obsahuje lineární přenosovou funkci. Všechny neurony mají své nastavitelné prahy a spoje mezi všemi neurony jsou zatíženy vahami. Topologie a funkce těchto pokusných sítí je popsána dále.

5.1.1 Síť s jednou skrytou vrstvou ANN(2-3-1)

Funkční tvar ANN(2-3-1) obsahuje třináct konstant (vah a prahů) K1 až K13, které je potřeba naučit:

$$K1 + \frac{K10}{1 + e^{K11+K12\,x1+K13\,x2}} + \frac{K2}{1 + e^{K3+K4\,x1+K5\,x2}} + \frac{K6}{1 + e^{K7+K8\,x1+K9\,x2}}$$

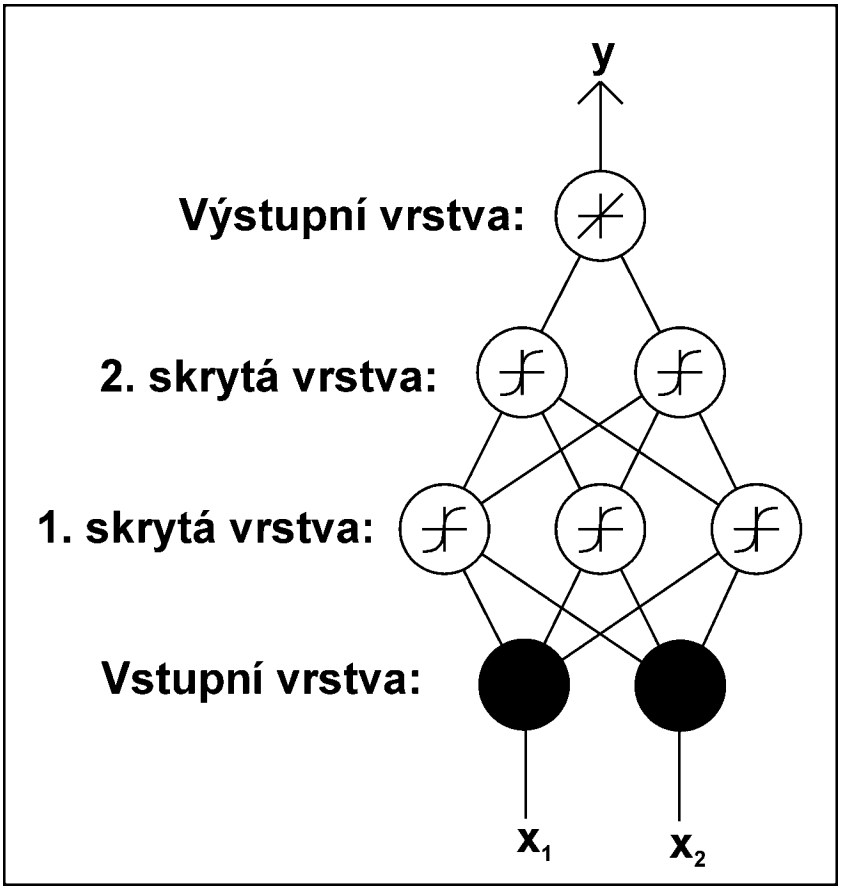


Obr. 17. Topologie ANN(2-3-1)

5.1.2 Síť se dvěma skrytými vrstvami ANN(2-3-2-1)

Funkční tvar ANN(2-3-2-1) obsahuje dvacet konstant (vah a prahů) K1 až K20, které je potřeba naučit:

$$K18 + \frac{K19}{1 + e^{K1+\frac{K10}{1+e^{K11+K12\,x1+K13\,x2}}+\frac{K2}{1+e^{K3+K4\,x1+K5\,x2}}+\frac{K6}{1+e^{K7+K8\,x1+K9\,x2}}}} + \frac{K20}{1 + e^{K14+\frac{K15}{1+e^{K3+K4\,x1+K5\,x2}}+\frac{K16}{1+e^{K7+K8\,x1+K9\,x2}}+\frac{K17}{1+e^{K11+K12\,x1+K13\,x2}}}}$$



Obr. 18. ANN(2-3-2-1)

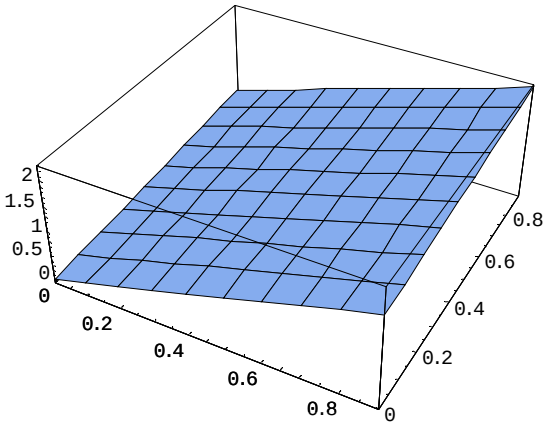
5.2 Testovací funkce

K naučení výše popsaným ANN jsme předložili tři následující problémy funkční aproximace odstupňované podle obtížnosti. Trénovací množina byla vždy sestavena z hodnot aproximované funkce v bodech x1, x2 $\in$ {0; 0.1; 0.2; 0.3; 0.4; 0.5; 0.6; 0.7; 0.8; 0.9}, kombinací x1 a x2 tedy v trénovací množině získáme 81 bodů.

Záměrně jsme zvolili problém funkční aproximace, oproti klasifikačním problémům, které se snažíme později syntetizovat, abychom při učení výrazně obtížnějšího aproximačního problému získali silné výsledky, jejichž stochasticky zatížené závěry pak budeme moci z jistotou aplikovat na klasifikaci.

5.2.1 Lineární funkce

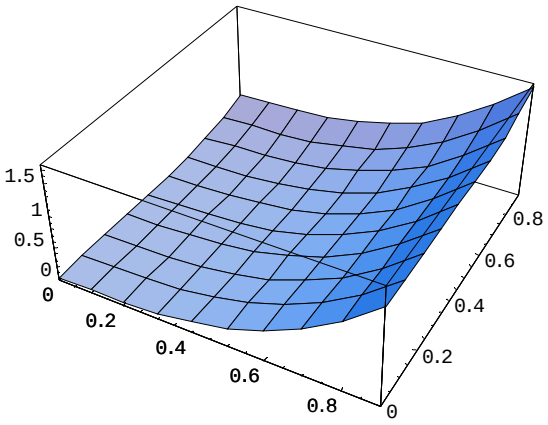
$$f_{lin}(x1,x2) = 0.5\ x1 + 2.0\ x2$$



Obr. 19. Lineární funkce

5.2.2 Polynomiální funkce

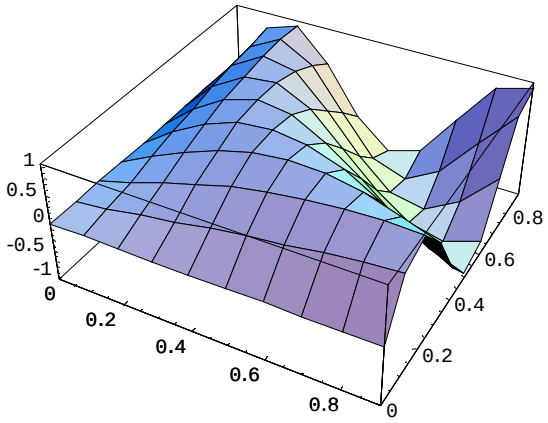
$$f_{pol}(x1,x2) = 0.5\ x1^6 + 2.0\ x2^3$$



Obr. 20. Polynomiální funkce

5.2.3 Goniometrická funkce

$$f_{gon}(x1,x2) = \sin(10\ x1\ x2)$$



Obr. 21. Goniometrická funkce

5.3 Výsledky učení

Za výsledek učení pokusné ANN považujeme průměrnou chybu, které se ANN dopustí při jednom odhadu prvku z testovací množiny, přičemž testovací množina je shodná s množinou trénovací. Hodnotu této chyby má smysl v následujících tabulkách uvádět pouze řádově, protože se jedná o stochasticky zatíženou veličinu.

5.3.1 Učení pomocí Back-propagation

Pro srovnání jsme testovali také, jakou průměrnou chybu dosáhneme na jednotlivých problémech při použití algoritmu Back-propagation - ve 20 iteracích. Časová náročnost této metody se dá označit vzhledem k ostatním výpočetním nárokům AP za zanedbatelnou.

Tab. 2. Učení Back-propagation

	ANN(2-3-1)	**ANN(2-3-2-1)**
$\mathbf{f_{lin}}$	10^-6	10^-5
$\mathbf{f_{pol}}$	10^-3	10^-3
$\mathbf{f_{gon}}$	10^-1	10^-1

5.3.2 Učení pomocí nelineárního prokládání

Nelineární prokládání poskytuje v přibližně stejné časové náročnosti (jedná se však až o 1000 iterací) srovnatelnou hodnotu průměrné chyby jako Back-propagation, ovšem na slo-

žitější problém řešení f_{gon} pomocí ANN(2-3-2-1) již nestačí a zcela selhává ve sledování trendu této aproximované funkce.

Tab. 3. Učení nelineárním prokládáním

	ANN(2-3-1)	**ANN(2-3-2-1)**
$\mathbf{f_{lin}}$	10^-7	10^-5
$\mathbf{f_{pol}}$	10^-3	10^-4
$\mathbf{f_{gon}}$	10^-1	**X**

5.3.3 Učení pomocí SOMA

Z různých EA jsme si k experimentu vybrali algoritmus SOMA, přesněji verzi AllToOne {NP = 30, Migrations = 1000, PRT = 0.1, Step = 0.3, Mass = 3}, s jehož aplikací máme z dřívějška bohaté pozitivní zkušenosti. Rozsah hledaných K se pohyboval v intervalu <-10;10>. Algoritmus vždy rychle konvergoval k hledanému řešení a poté již jeho hodnotu vylepšoval jen velmi pomalu ovšem trvale (viz grafy na přiloženém CD). Použití SOMA přináší řádově horší, ale stále uspokojivé a také robustnější výsledky, než nelineární prokládání. Značnou nevýhodou je ovšem zvýšená výpočetní náročnost - řádově v jednotkách minut, která se může na použití AP negativně promítnout. Jednalo by se pak vlastně o zapouzdření jednoho SOMA do druhého - což může časovou náročnost daného výpočtu neúměrně protáhnout.

Tab. 4. Učení SOMA

	ANN(2-3-1)	**ANN(2-3-2-1)**
$\mathbf{f_{lin}}$	10^-2	10^-2
$\mathbf{f_{pol}}$	10^-2	10^-2
$\mathbf{f_{gon}}$	10^-1	10^-1

5.4 Extrahované poznatky

Míra úspěšnosti, se kterou algoritmus učí danou ANN, závisí jednak na obtížnosti aproximovaného problému a jednak také na struktuře ANN, respektive počtu neznámých kon-

stant (vah a prahů) v ANN. Protože při syntéze ANN pomocí AP je struktura ANN silně variabilní, nelze úspěšnost použitého algoritmu jednoznačně předvídat.

Nejdůležitější ovšem je, že jsme jednoznačně prokázali, že nelineární prokládání je dostatečně silným nástrojem pro řešení jednodušších problémů, jako je otázka dvouhodnotové klasifikace, kterou se budeme zabývat v následující kapitole.

6 SYNTÉZA NEURONOVÝCH SÍTÍ

V naší práci jsme si vybrali dva základní, ale také zásadní, problémy dvojhodnotové klasifikace, pro které pomocí AP hodláme syntetizovat vhodné ANN. Jedná se o lineárně separabilní problém a problém XOR. Jako metodu učení ANN* jsme použili nelineární prokládání, které se v předchozích experimentech prokázalo být dostatečně vhodné. K syntéze bylo dále užito CF1 a **GFS1**, tak jak jsou tyto popsány v teoretické části práce. Pouze v kap. 4.4 je **CFS1** nahrazena **CFS2**. Jako EA použitý v AP vystupuje vždy SOMA „AllTo-One“. Přitom za úspěšně nalezené řešení je považována ANN, která úspěšně odpoví na všechny podněty z testovací množiny (ta je shodná s množinou trénovací), tedy $E_T = \xi = 0$.

6.1 Popis použitých formulí a grafů

Struktura ANN* - vhodná struktura ANN nalezená pomocí AP s konstantami K připravenými k učení, vyjádřená jako příslušná **f***.

Struktura ANN - struktura ANN, kde konstanty K (váhy a prahy) byly determinovány (naučeny) pomocí metod nelineárního prokládání (*nonlinear fitting*), vyjádřená jako příslušná **f**.

Substituovaný tvar ANN - přenosová funkce neuronu se dvěma vstupy - sigmoida - je nahrazena tvarem **Neuron[*arg1*, *arg2*]**, pro lepší zobrazení závislosti jednotlivých funkcí **b***i* obsažených v dané ANN. Při zobrazování takové formule jsme museli volit mezi dvěma zly, buď formuli zalomit na dva řádky a nebo ji vytisknout v malém fontu. Rozhodli jsme se, že použití malých písmenek bude menším znepřehledněním, než zalomení na více řádků.

Hranice mezi třídami ANN - zobrazuje klasifikační hranici ANN, pokud je odezva větší než 0.5 spadá klasifikace do třídy 1 v opačném případě do třídy 0.

Pásmo necitlivosti ANN - zobrazuje pásmo (šedou plochu), ve kterém odezva ANN na podnět leží v intervalu (0.2, 0.8), ANN zde neodpovídá na podnět citlivě - ostrou klasifikací.

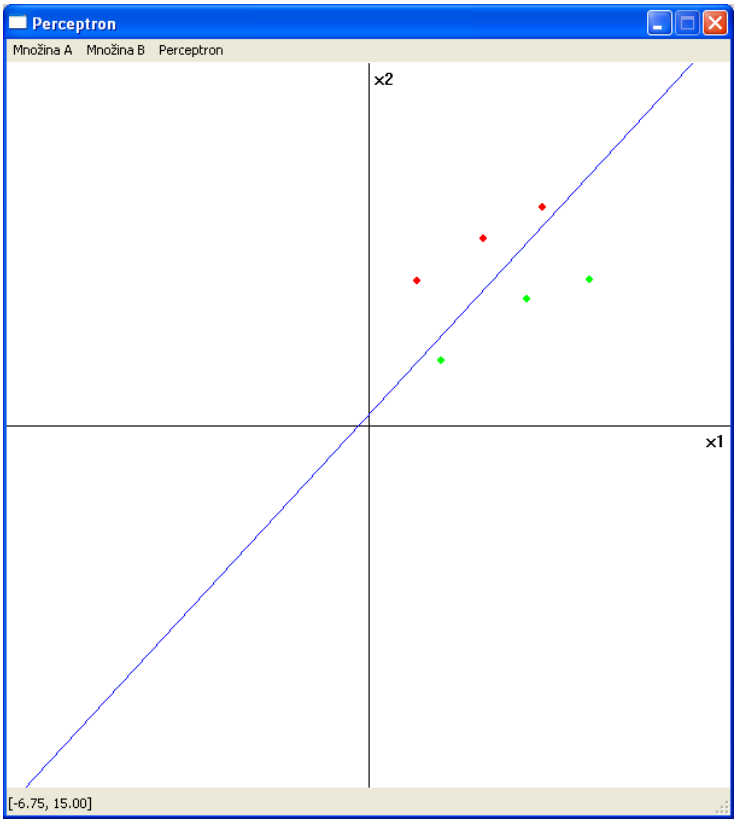
3D graf funkce ANN - zobrazuje ANN, jako funkci v $\mathbb{E}^3$.

Topologie ANN - zobrazuje topologické vztahy **b** v ANN, převedených do neuronálních tvarů - kde jednotlivé K jsou reprezentovány, jako váhy w a prahy ϕ, bez pořadového číslování.

6.2 Řešení lineálně separabilního problému

Lineárně separabilní klasifikační problém, je takový problém, v němž lze použít jako hranici klasifikovaných tříd přímku. Jedná se o vůbec základní problém, který lze řešit pomocí jednoho jediného neuronu a který nutně musíme vyřešit i my, máme-li pomýšlet na řešení obtížnějších úloh. Skutečnost, že na řešení tohoto problému stačí právě jeden neuron, nám pomůže k posouzení složitosti syntetizovaných řešení. Dále uvádíme čtyři z mnoha variant úspěšně syntetizovaných ANN, ANN1 až ANN4.

Řešení lineárně separabilních problémů nám demonstruje také volně šiřitelný program s názvem Perceptron, který naprogramoval autor této práce (Bc. Pavel Vařacha). Otevřený programový kód i spustitelná verze, jsou k dispozici jako příloha P II na přiloženém CD. Program používá k učení mononeuronální sítě perceptron jednoduchou metodu fixního přírůstku. [19]

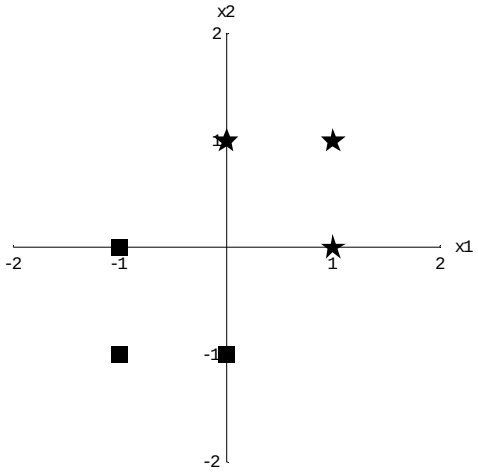


Obr. 22. Program Perceptron

6.2.1 Definice lineárně separabilního problému

Tab. 5. Lineární klasifikace

x1	x2	třída
1	1	**1**
1	0	**1**
0	1	**1**
-1	-1	**0**
-1	0	**0**
0	-1	**0**



Obr. 23. Lineárně separabilní problém

6.2.2 Příklad řešení ANN1

Struktura ANN1*:

$$\frac{1}{1 + e^{-10\,(x1 + K[1] + x2\,K[2])}}$$

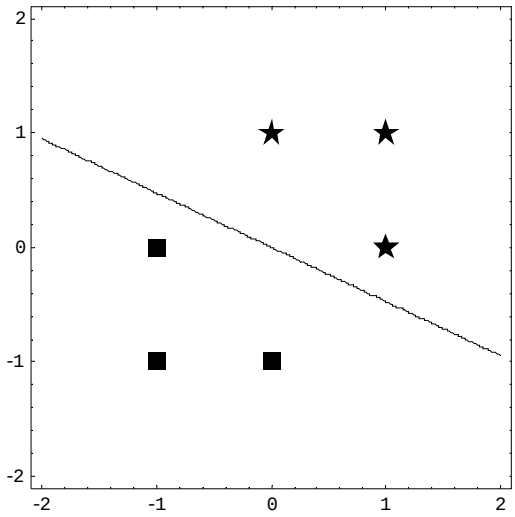
Struktura ANN1:

$$\frac{1}{1 + e^{-10\,(-3.28444 \times 10^{-9} + x1 + 2.11403\,x2)}}$$

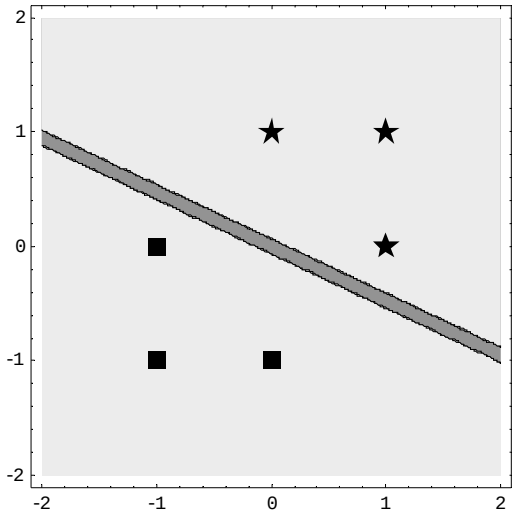
Substituovaný tvar ANN1*:

$$\text{Neuron}[x1,\ K[1] + x2\,K[2]]$$

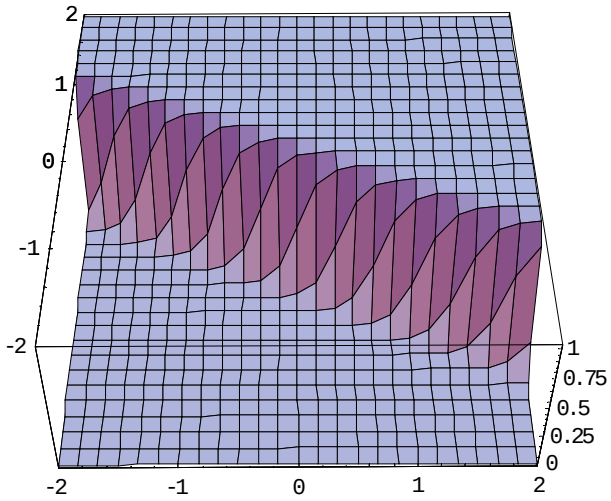
Grafické zobrazení ANN1:



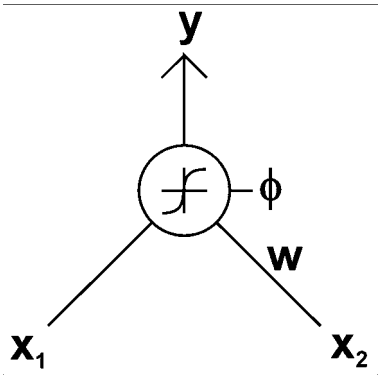
Obr. 24. Hranice mezi třídami ANN1



Obr. 25. Pásmo necitlivosti ANN1



Obr. 26. 3D graf funkce ANN1



Obr. 27. Topologie ANN1

6.2.3 Příklad řešení ANN2

Struktura ANN2*:

$$\frac{1}{1+e^{-10\,(x2+K[1]+x1\,K[2])}}+x1\,x2^2$$

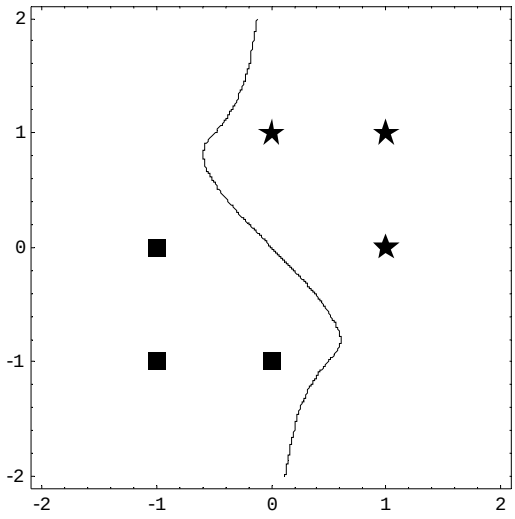
Struktura ANN2:

$$\frac{1}{1+e^{-10\,(9.1771\times10^{-6}+0.999986\,x1+x2)}}+x1\,x2^2$$

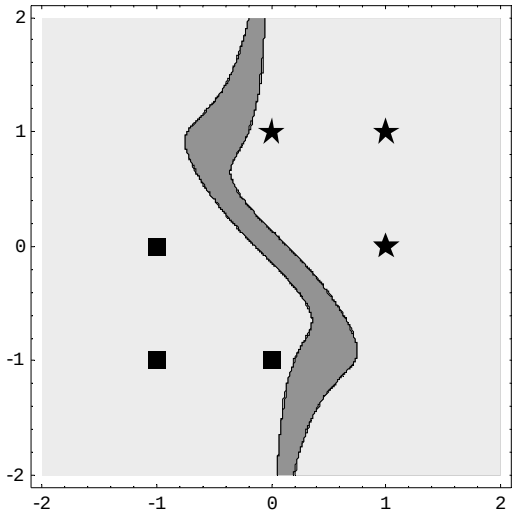
Substituovaný tvar ANN2*:

$$x1\,x2^2+\text{Neuron}[x2,\,K[1]+x1\,K[2]]$$

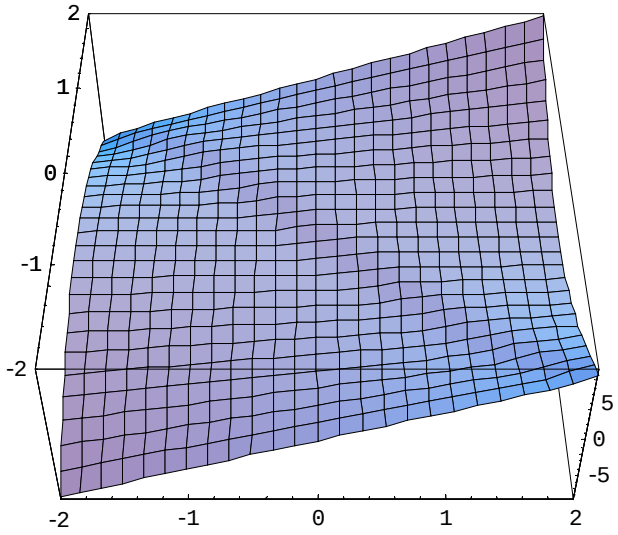
Grafické znázornění ANN2:



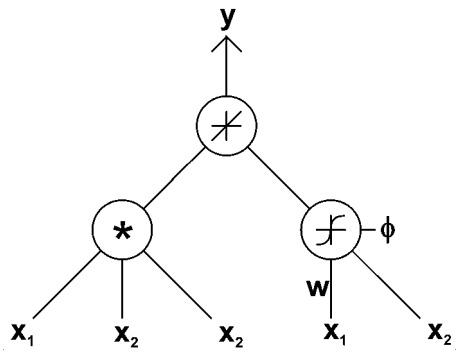
Obr. 28. Hranice mezi třídami ANN2



Obr. 29. Hranice mezi třídami ANN2



Obr. 30. 3D graf funkce ANN2



Obr. 31. Topologie ANN2

6.2.4 Příklad řešení ANN3

Struktura ANN3*:

$$\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-10\left(x1+x1\left(\frac{1}{1+e^{-10(2x1+K[1])}}+\frac{x1}{1+e^{-10(x2+K[2])}}+x2\right)\left(\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-20x2}}+\frac{x1x2}{1+e^{-10(x1+x2)}}\right)}}+2x2+K[3]\right)\right)}}+x2+K[4]\right)}}$$

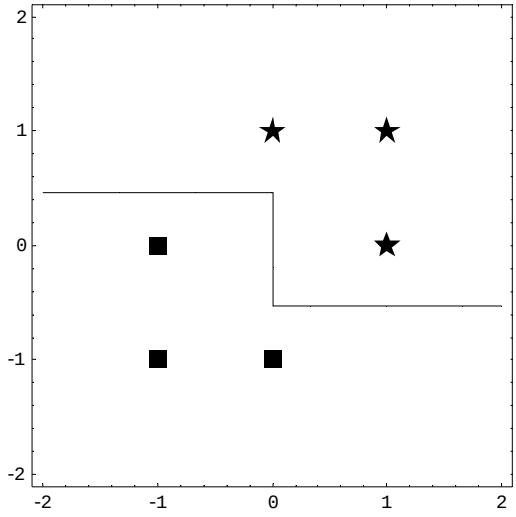
Struktura ANN3*:

$$\frac{1}{1+e^{-10\left(-0.466912+\frac{1}{1+e^{-10\left(x1+x1\left(\frac{1}{1+e^{-10(7.45942\times10^{9}+2x1)}}+\frac{x1}{1+e^{-10(-1.33843\times10^{6}+x2)}}+x2\right)\left(1.11316\times10^{6}+\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-20x2}}+\frac{x1x2}{1+e^{-10(x1+x2)}}\right)}}+2x2\right)\right)}}+x2\right)}}$$

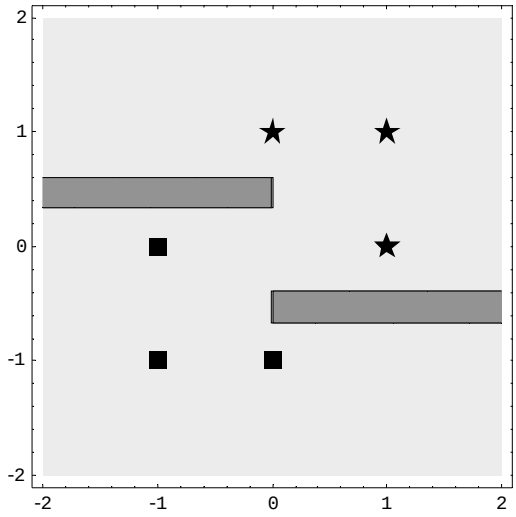
Substituovaný tvar ANN3*:

$$\text{Neuron}\left[\text{Neuron}\left[x1,\ x1\ (x2+\text{Neuron}[2\,x1,\ K[1]]+x1\,\text{Neuron}[x2,\ K[2]])\ \left(2\,x2+\text{Neuron}\left[\frac{1}{1+e^{-20\,x2}},\ x1\,x2\,\text{Neuron}[x1,\ x2]\right]+K[3]\right)\right],\ x2+K[4]\right]$$

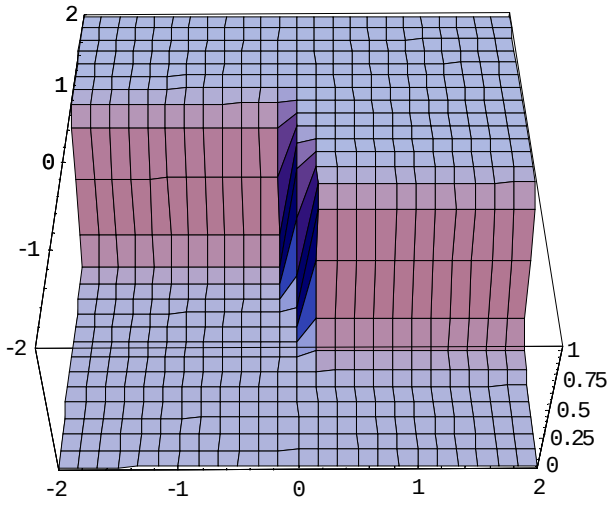
Grafické zobrazení ANN3:



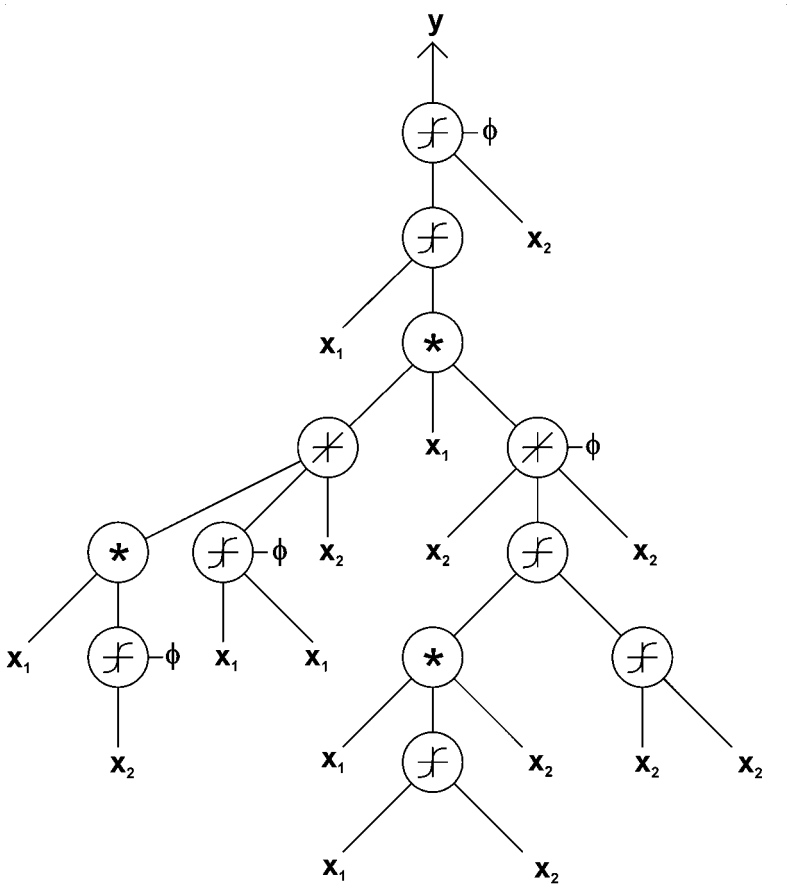
Obr. 32. Hranice mezi třídami ANN3



Obr. 33 Pásmo necitlivosti ANN3



Obr. 34. 3D graf funkce ANN3



Obr. 35. Topologie ANN3

6.2.5 Příklad řešení ANN4

Struktura ANN4*:

$$(x1 + K[\![1]\!])\,(x1 + x2 + K[\![2]\!] + x2\,K[\![3]\!])\left(x2 + K[\![4]\!] + \frac{x2\left(x2 + K[\![6]\!] + \frac{K[\![8]\!]}{1+e^{-10\left(\frac{1}{1+e^{-20\,x1}} + K[\![7]\!]\right)}}\right)}{1 + e^{-10\left(x1 + \frac{K[\![5]\!]}{1+e^{-20\,x2}}\right)}}\right)$$

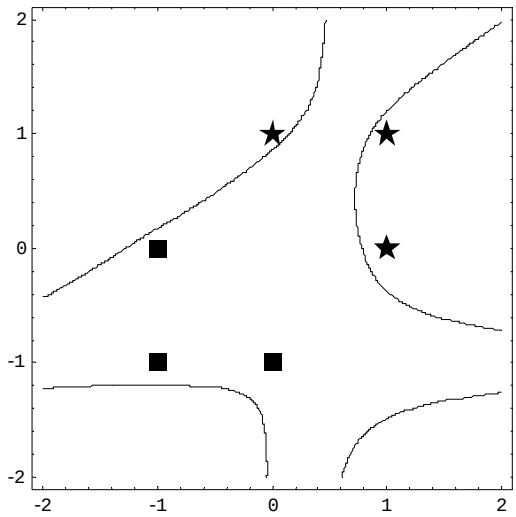
Struktura ANN4:

$$(-0.500001 + x1)\,(1. + x1 - 1.5\,x2)\left(1. + x2 + \frac{x2\left(0.500007 + \frac{0.500007}{1+e^{-10\left(0.999999 + \frac{1}{1+e^{-20\,x1}}\right)}} + x2\right)}{1 + e^{-10\left(\frac{0.999797}{1+e^{-20\,x2}} + x1\right)}}\right)$$

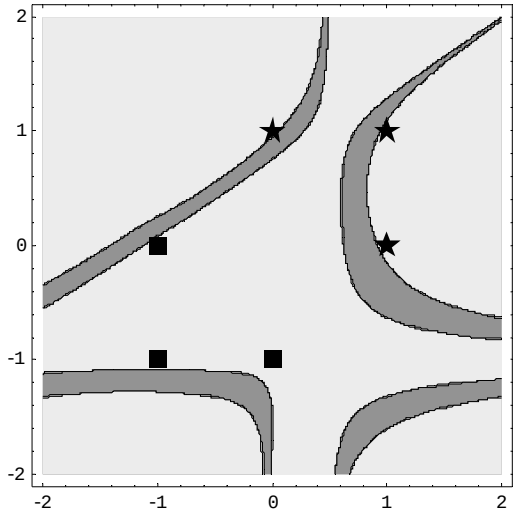
Substituovaný tvar ANN4*:

$$(x1 + K[\![1]\!])\,(x1 + x2 + K[\![2]\!] + x2\,K[\![3]\!])\left(x2 + K[\![4]\!] + x2\,\text{Neuron}\left[x1, \frac{K[\![5]\!]}{1 + e^{-20\,x2}}\right]\left(x2 + K[\![6]\!] + \text{Neuron}\left[\frac{1}{1 + e^{-20\,x1}}, K[\![7]\!]\right]K[\![8]\!]\right)\right)$$

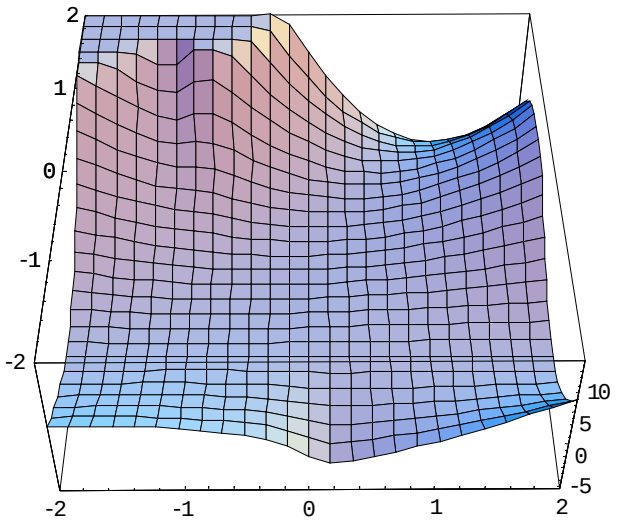
Grafické zobrazení ANN4:



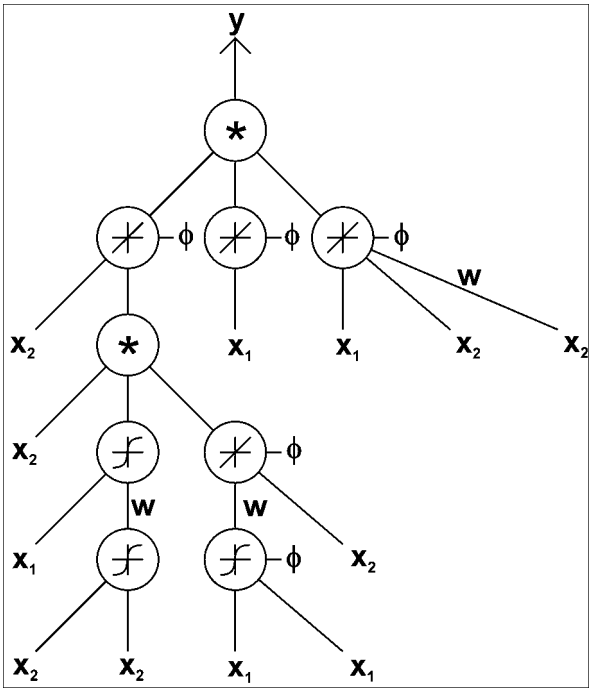
Obr. 36. Hranice mezi třídami ANN4



Obr. 37. Pásmo necitlivosti ANN4



Obr. 38. 3D graf ANN4



Obr. 39. Topologie ANN4

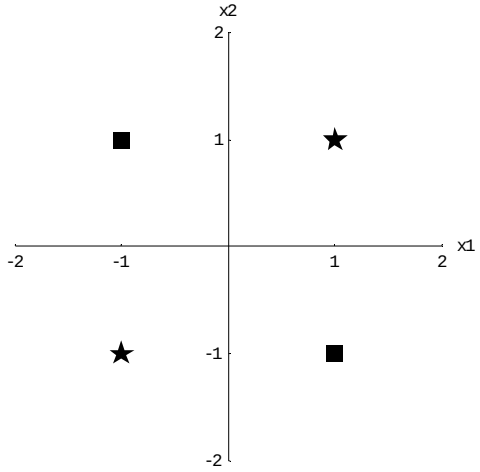
6.3 Řešení problému XOR

Problém klasifikace podle logické funkce XOR, neblaze proslul zbrzděním vývoje neuronových sítí v 60. letech 20. stol. [19] Logická funkce XOR je charakteristická tím, že její prvky nelze lineárně separovat - k jejímu řešení jsou třeba minimálně dva sériově zapojené neurony. Pokud mají být námi syntetizované ANN klasifikačně funkční, je nezbytně nutné, aby byly schopny daný problém řešit. Následující příklady ANN5 až ANN8 reprezentují vybrané úspěšně syntetizované řešení problému XOR.

6.3.1 Definice problému XOR

Tab. 6. Klasifikace XOR

x1	x2	třída
1	1	**1**
-1	-1	**1**
1	-1	**0**
-1	1	**0**



Obr. 40. Problém XOR

6.3.2 Příklad řešení ANN5

Struktura ANN5*:

$$\frac{1}{1+e^{-10\left(x1\left(\frac{1}{1+e^{-10\left(x2+\left(\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-10(x1+K[2]^2)}}+x1+x2+2K[3]\right)}}+x2\right)(2x2+K[4])\left(\frac{1}{1+e^{-20K[5]}}+x1+x1K[6]+x2K[7]\right)\right)}}+2x2\right)+x1K[1]^2\right)}}$$

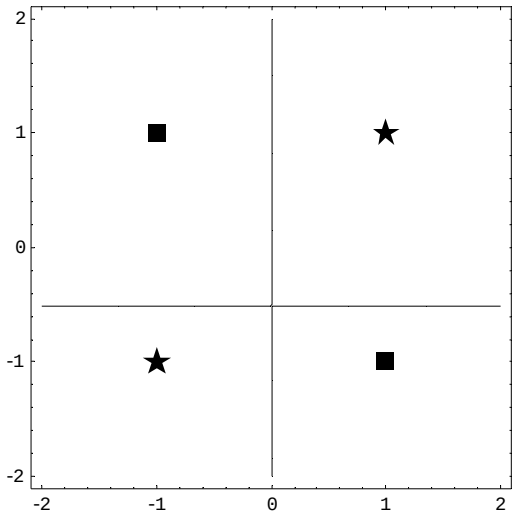
Struktura ANN5:

$$\frac{1}{1+e^{-10\left(1.x1+x1\left(\frac{1}{1+e^{-10\left(x2+\left(\frac{1}{1+e^{-10\left(2.+\frac{1}{1+e^{-10(1.+x1)}}+x1+x2\right)}}+x2\right)(1.+2.x1+1.x2)(1.+2x2)\right)}}+2x2\right)\right)}}$$

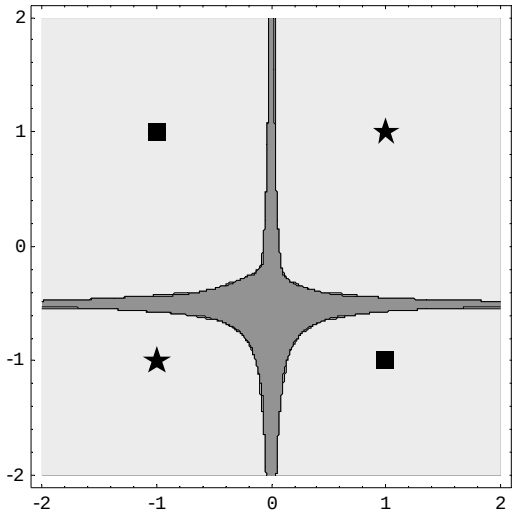
Substituovaný tvar ANN5*:

$$\text{Neuron}\left[x1\left(2x2+\text{Neuron}\left[x2,\ (x2+\text{Neuron}[\text{Neuron}[x1,K[2]^2],x1+x2+2K[3]])(2x2+K[4])\left(\frac{1}{1+e^{-20K[5]}}+x1+x1K[6]+x2K[7]\right)\right]\right),x1K[1]^2\right]$$

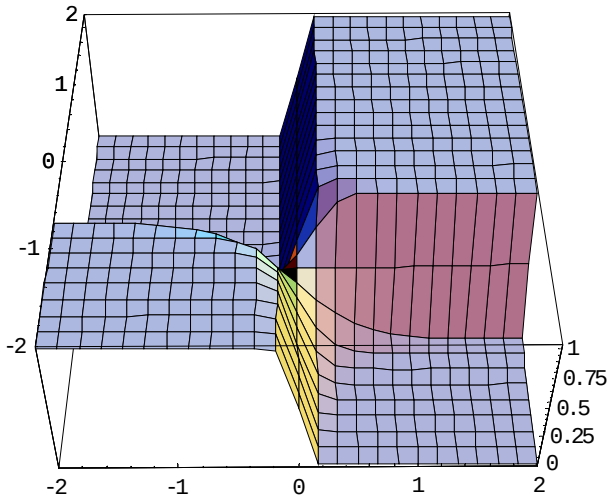
Grafické zobrazení ANN5:



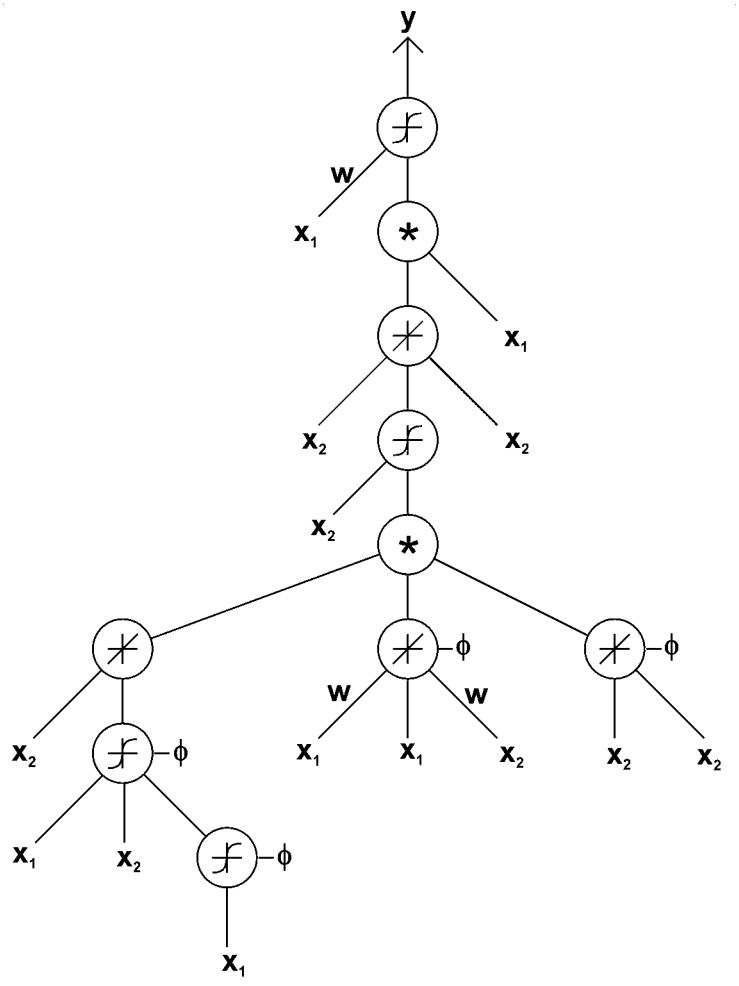
Obr. 41. Hranice mezi třídami ANN5



Obr. 42. Hranice mezi třídami ANN5



Obr. 43. 3D graf funkce ANN5



Obr. 44. Topologie ANN5

6.3.3 Příklad řešení ANN6

Struktura ANN6*:

$$x1\left(x1+x2+x1\,x2\,K[\![1]\!]^2\left(\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-10\left(\frac{x1}{1+e^{-10\,(x1+K[\![3]\!])}}+K[\![2]\!]\right)}}+K[\![4]\!]\right)}}+K[\![5]\!]\right)(x2+K[\![6]\!]+x1^2\,K[\![7]\!])\right)$$

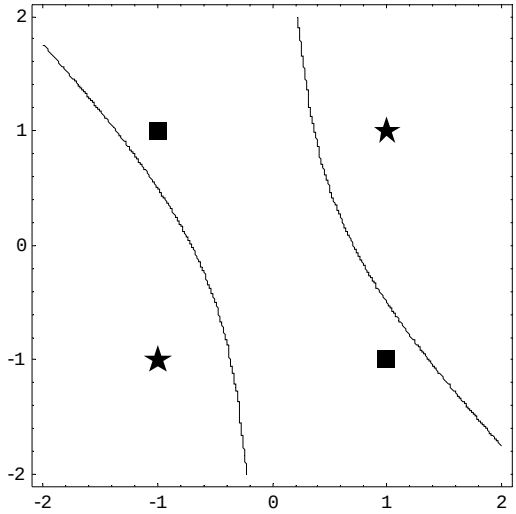
Struktura ANN6:

$$x1\left(x1+x2+2.6485\times10^{-26}\left(0.674245+\frac{1}{1+e^{-10\left(1.+\frac{1}{1+e^{-10\left(1.+\frac{x1}{1+e^{-10\,(1.+x1)}}\right)}}\right)}}\right)x1\,x2\,(0.783028+0.783028\,x1^2+x2)\right)$$

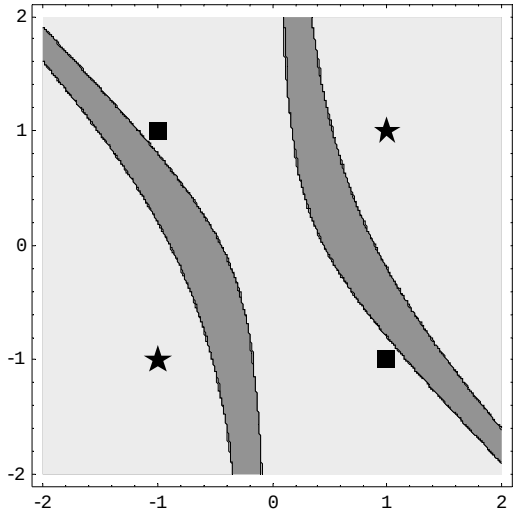
Substituovaný tvar ANN6*:

```
x1 (x1 + x2 + x1 x2 K[[1]]^2 (Neuron[Neuron[x1 Neuron[x1, K[[3]]], K[[2]]], K[[4]]] + K[[5]]) (x2 + K[[6]] + x1^2 K[[7]]))
```

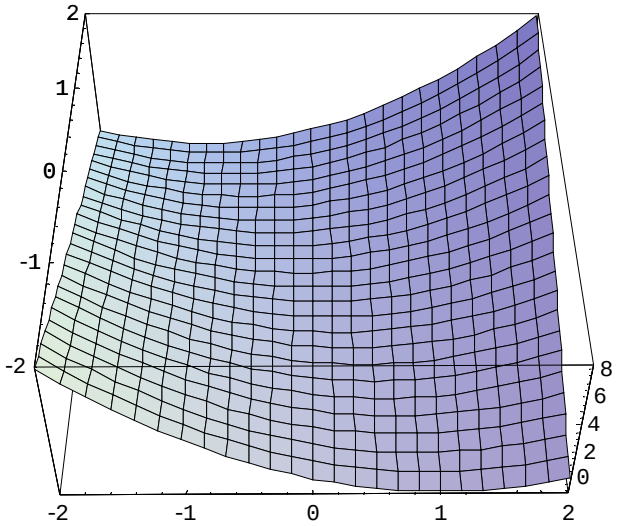
Grafické znázornění ANN6:



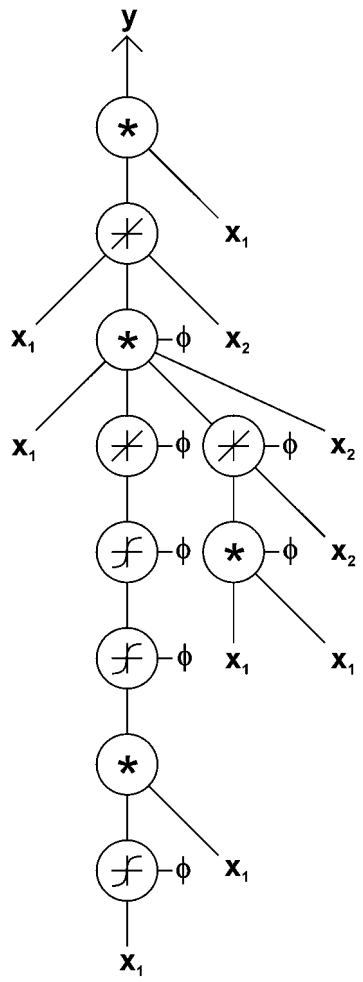
Obr. 45. Hranice mezi třídami ANN6



Obr. 46. Pásmo necitlivosti ANN6



Obr. 47. 3D graf funkce ANN6



Obr. 48. Topologie ANN6

6.3.4 Příklad řešení ANN7

Struktura ANN7*:

$$\frac{1}{1+e^{-10\left(K[1]+x1\,x2\,K[2]\left(x2+x1\,x2^{3}+K[3]+\frac{K[5]}{1+e^{-20K[4]}}+K[6]^{2}+\frac{x1\,K[9]}{1+e^{-10\left(\frac{1}{1+e^{-10}}+x1\right)}}\right)\right)}}$$

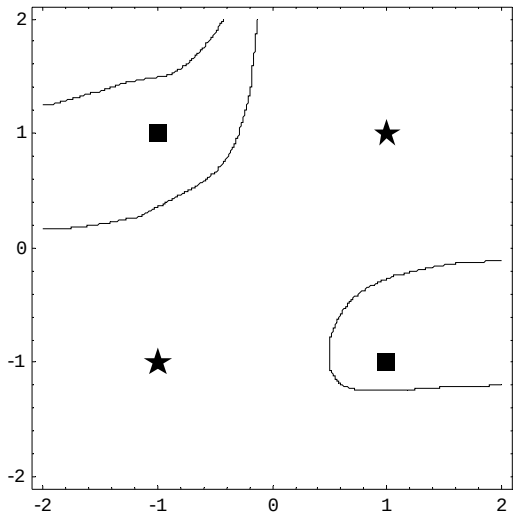
Struktura ANN7:

$$\frac{1}{1+e^{-10\left(1.+1.\,x1\,x2\left(3.+\frac{1.\,x1}{1+e^{-10\left(\frac{1}{1+e^{-10\left(1.+\frac{x2}{1+e^{-10\left(\frac{1}{\left(1+e^{-10\,(1.+x1)}\right)\left(1+e^{-10\,(x1+x2)}\right)}+x1\right)}}\right)}}+x1\right)}}+x2+x1\,x2^3\right)}}$$

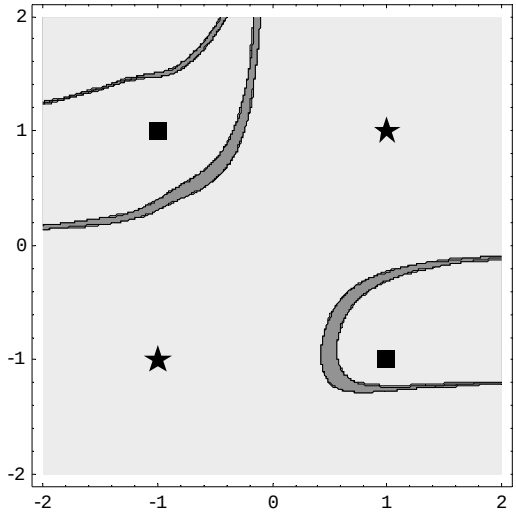
Substituovaný tvar ANN7*:

$$\text{Neuron}\left[K[1],\ x1\,x2\,K[2]\left(x2+x1\,x2^3+K[3]+\frac{K[5]}{1+e^{-20K[4]}}+K[6]^2+x1\,\text{Neuron}[\text{Neuron}[x2\,\text{Neuron}[\text{Neuron}[x1,\ x2]\ \text{Neuron}[x1,\ K[8]],\ x1],\ K[7]],\ x1]\ K[9]\right)\right]$$

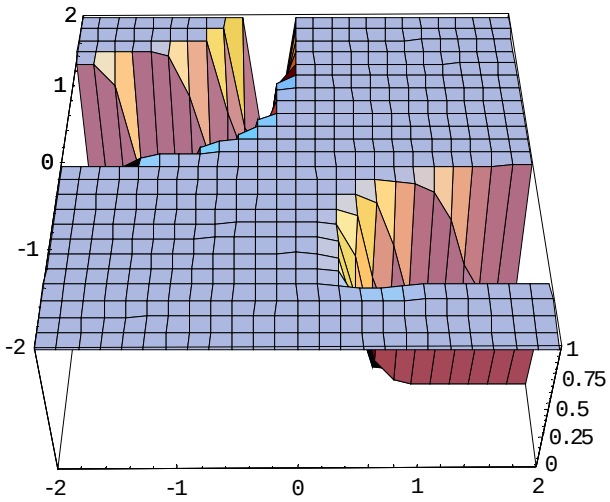
Grafické znázornění ANN7:



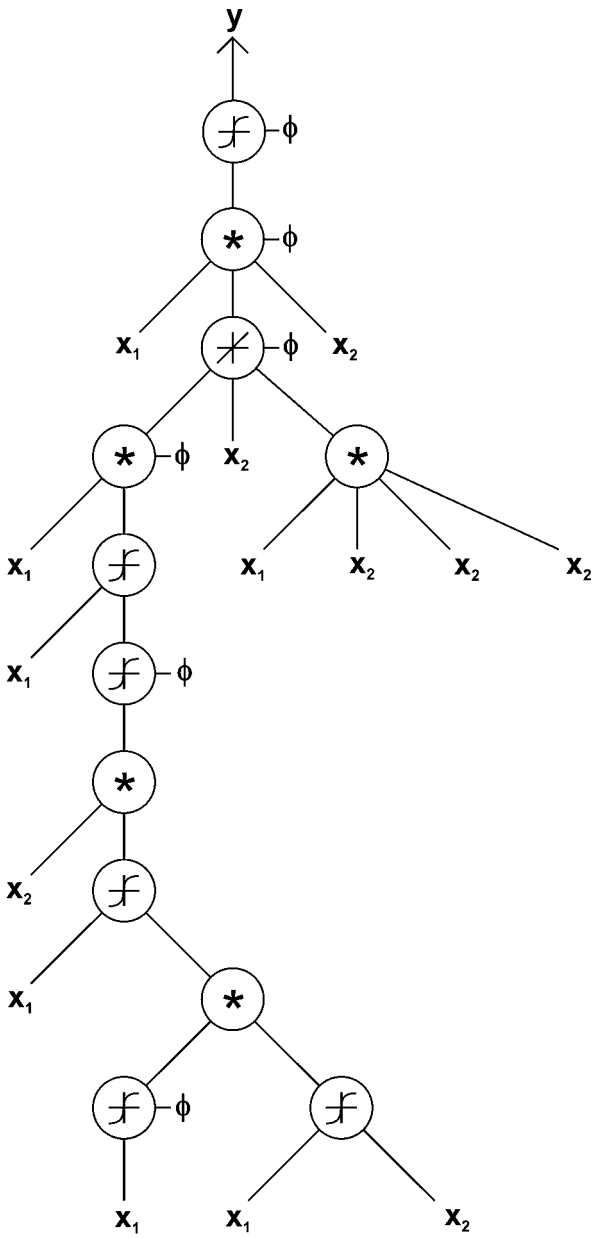
Obr. 49. Hranice mezi třídami ANN7



Obr. 50. Pásmo necitlivosti ANN7



Obr. 51. 3D graf funkce ANN7



Obr. 52. Topologie ANN7

6.3.5 Příklad řešení ANN8

Struktura ANN8*:

$$\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-10(x1+x2+K[1])}}+\frac{x12}{1+e^{-10(x2+K[3])}}+x1K[2]2+x1(x2+K[4])\right)}}+\frac{x1}{(1+e^{-10(x1+K[5])})(1+e^{-10(x1+x2+K[6])})}\right)}}+\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-10(x1+x2+K[7])}}+2x1+x2K[7]\right)}}+x1\,x2$$

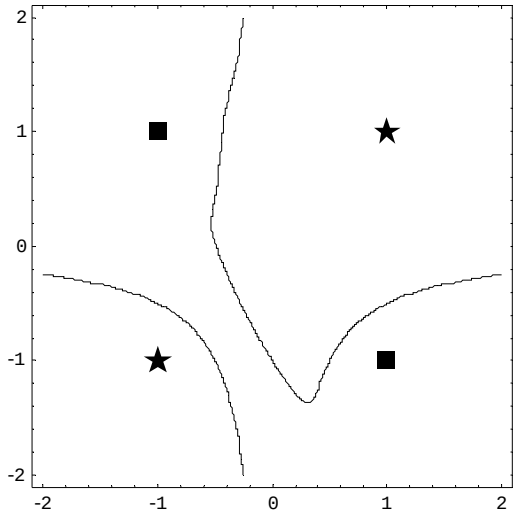
Struktura ANN8:

$$\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-10(1.+x1+x2)}}+1.\,x1+\frac{x1^2}{1+e^{-10(1.+x2)}}+x1\,(1.+x2)\right)}}+\frac{x1}{(1+e^{-10(1.+x1)})\,(1+e^{-10(1.+x1+x2)})}\right)}}+x1\,x2$$

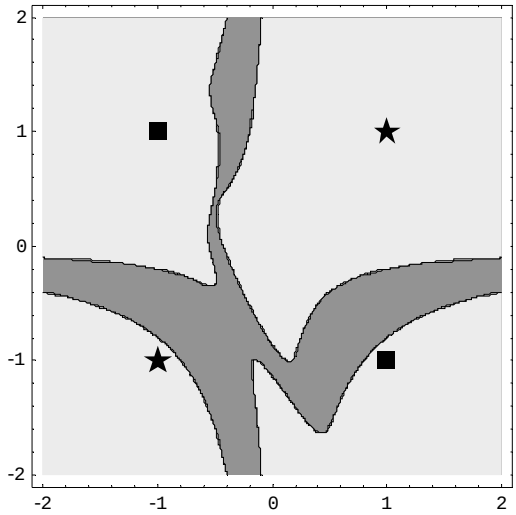
Substituovaný tvar ANN8*:

$$x1\,x2+\text{Neuron}[\text{Neuron}[\text{Neuron}[\text{Neuron}[x1,\,x2+K[1]],\,x1^2\,\text{Neuron}[x2,\,K[3]]+x1\,K[2]^2+x1\,(x2+K[4])],\,x1\,\text{Neuron}[x1,\,K[5]]\,\text{Neuron}[x1,\,x2+K[6]]],\,2\,x1+x2\,K[7]]$$

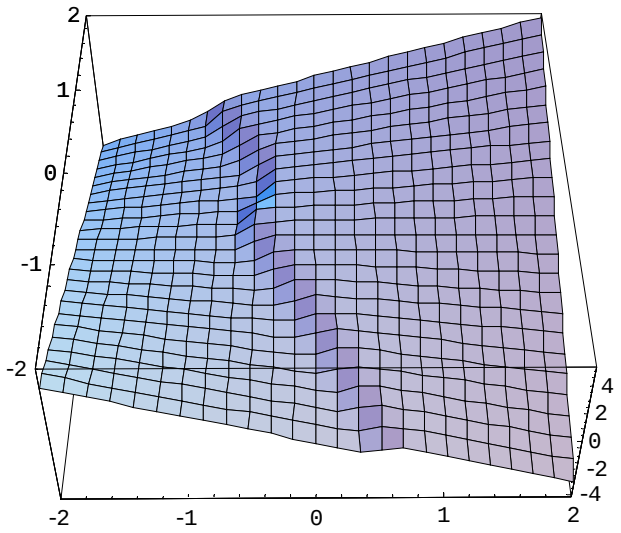
Grafické znázornění ANN8:



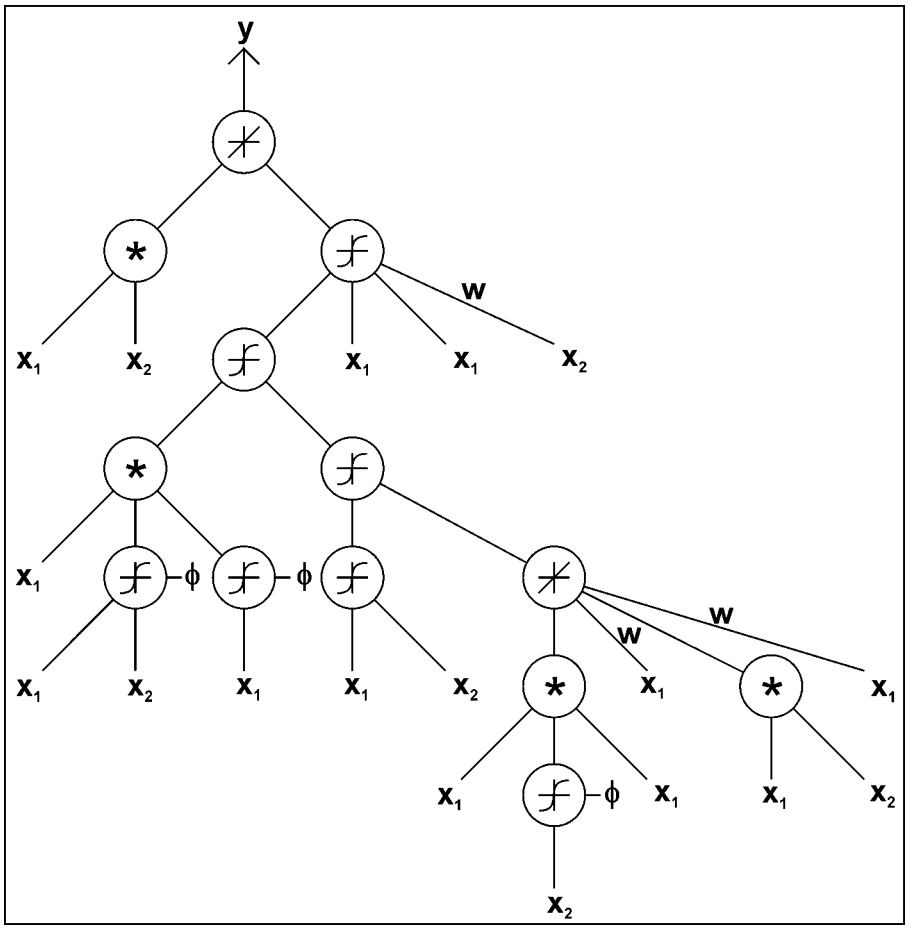
Obr. 53. Hranice mezi třídami ANN8



Obr. 54. Pásmo necitlivosti ANN8



Obr. 55. 3D graf funkce ANN8



Obr. 56. Topologie ANN8

6.4 Řešení lineárního problému pomocí GFS2

Následující příklady ANN9 a ANN10 se vracejí zpět k lineárně separabilnímu problému a demonstrují jeho řešení pomocí redukované množiny GFS2.

6.4.1 Příklad řešení ANN9

Struktura ANN9*:

$$\frac{1}{1 + e^{-10\,(K[1]+x1\,K[2]+x2\,K[3])}}$$

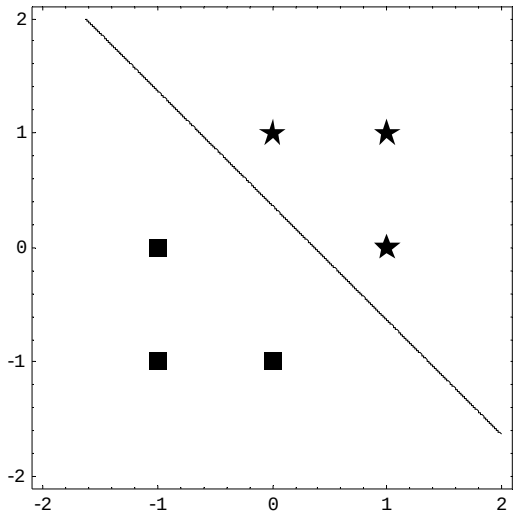
Struktura ANN9:

$$\frac{1}{1 + e^{-10\,(-2.1027+5.81557\,x1+5.81557\,x2)}}$$

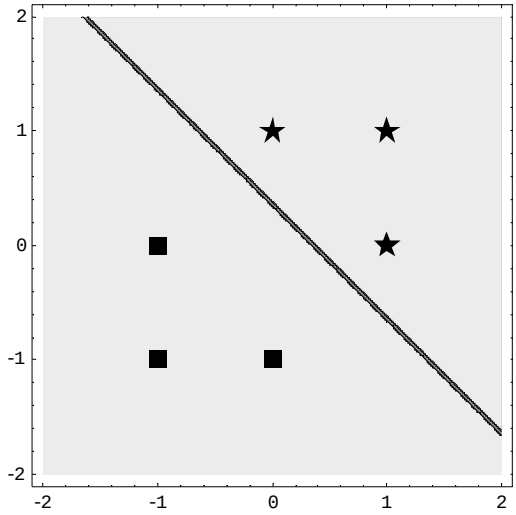
Substituovaný tvar ANN9*:

```
Neuron[K[1], x1 K[2] + x2 K[3]]
```

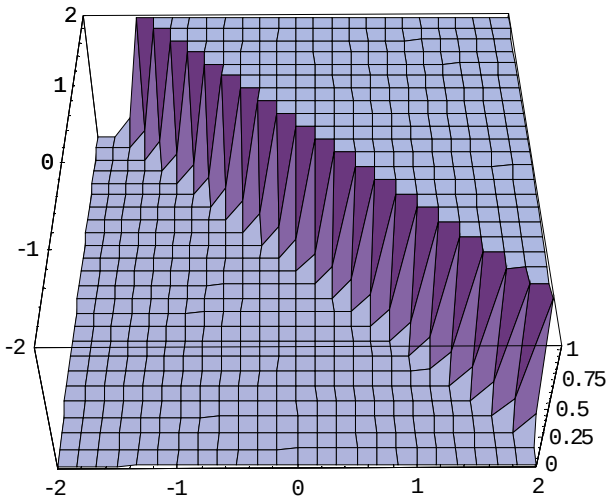
Grafické znázornění ANN9:



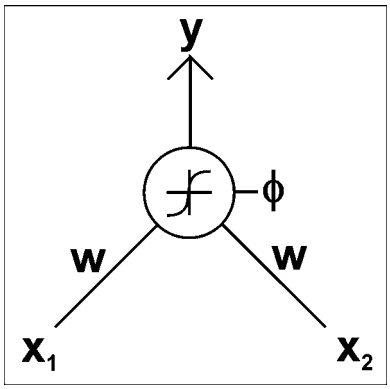
Obr. 57. Hranice mezi třídami ANN9



Obr. 58. Pásmo necitlivosti ANN9



Obr. 59. 3D graf funkce ANN9



Obr. 60. Topologie ANN9

6.4.2 Příklad řešení ANN10

Struktura ANN10*:

$$\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-10\left(\frac{1}{1+e^{-10(x2+K[1])}}+x1+\frac{K[3]}{1+e^{-10(x1+K[2])}}\right)}}+K[4]\right)}}$$

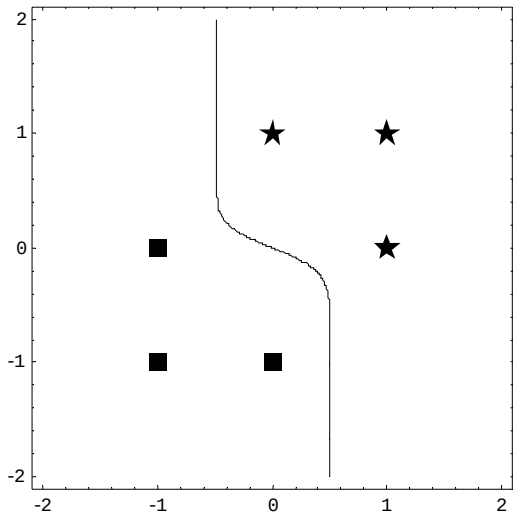
Struktura ANN10:

$$\frac{1}{1+e^{-10\left(-0.5+\frac{1}{1+e^{-10\left(-\frac{0.500013}{1+e^{-10(2.49709+x1)}}+\frac{1}{1+e^{-10(0.000928763+x2)}}+x1\right)}}\right)}}$$

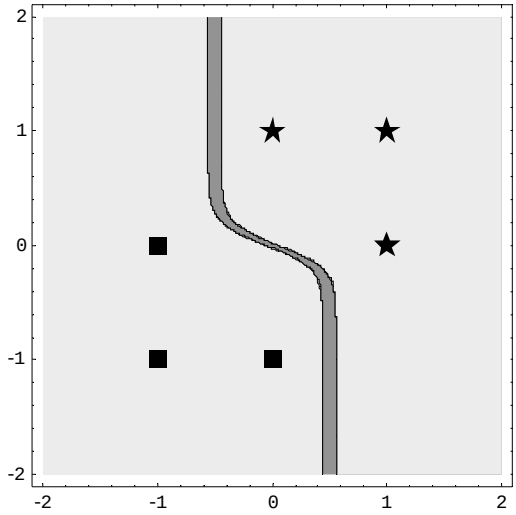
Substituovaný tvar ANN10*:

```
Neuron[Neuron[Neuron[x2, K[[1]]], x1 + Neuron[x1, K[[2]]] K[[3]]], K[[4]]]
```

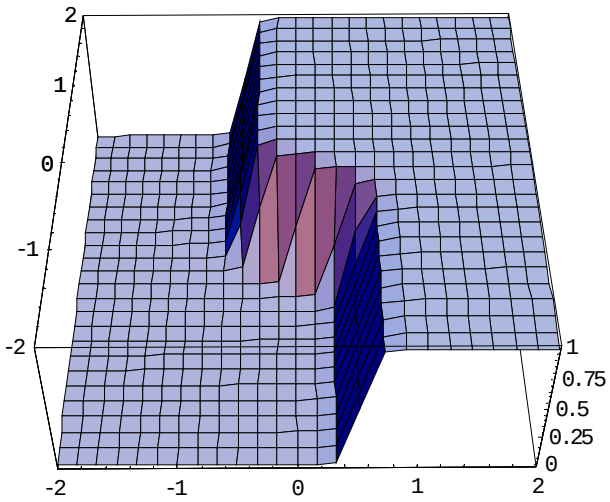
Grafické znázornění ANN10:



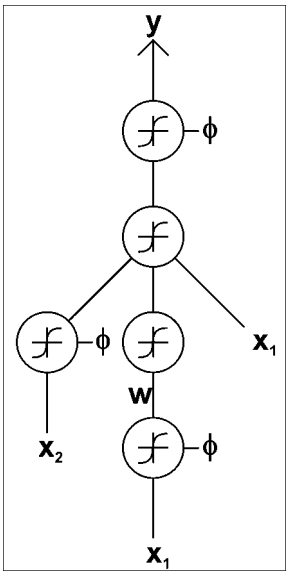
Obr. 61. Hranice mezi třídami ANN10



Obr. 62. Pásmo necitlivosti ANN10



Obr. 63. 3D graf funkce ANN10



Obr. 64. Topologie ANN10

6.5 Analýza získaných řešení

Tato kapitola popisuje získaná řešení z hlediska jejich rozlehlosti, tvaru klasifikační hranice a struktury topologie. Výsledná tvrzení jsou dokumentována na předcházejících ukázkách ANN1 až ANN10.

6.5.1 Rozlehlost syntetizovaných ANN

Oba dva klasifikační problémy (viz kap. 6.2.1 a kap. 6.3.1) se nám podařilo úspěšně řešit širokou škálou syntetizovaných ANN. Nutno přiznat, že struktura většiny nalezených řešení byla podstatně složitější, než nejmenší možná rigorózně konstruovaná ANN, která řeší identický problém. Někdy se jedná opravdu o výrazně delší ovšem stále plně funkční řešení např. ANN8 (viz kap. 6.3.5).

Existují ovšem také výjimky. Například v ANN9 (viz kap. 6.4.1) se nám podařilo vyšlechtit pro účely řešení lineárně separabilního problému, síť o velikosti právě jednoho klasického neuronu. Počítačem nalezené řešení se tak shoduje s klasickým minimálním řešením.

ANN1 dokonce nabízí řešení stejného problému pomocí ještě jednoduší sítě než je ANN9, tím, že vypustí váhu w ze vstupu x1. Přitom kvalita klasifikační plochy je u ANN1 a ANN9 identická. Lze totiž ukázat, že podobně jako přímka:

$$y = w1\ x1 + w2\ x2 + \phi = 0$$

$$x1 = (w2 / w1)\ x2 + (\phi / w1)$$

která reprezentuje zjednodušenou představu o hranici mezi třídami ANN9, může za pomoci vhodného nastavení konstant w1, w2, ϕ protínat libovolné dva body v rovině. Tak i přímka:

$$y = x1 + w2\ x2 + \phi = 0$$

$$x1 = w2\ x2 + \phi$$

která reprezentuje zjednodušenou představu o hranici mezi třídami ANN1, může za pomoci vhodného nastavení konstant w2, ϕ protínat dva libovolné body v rovině.

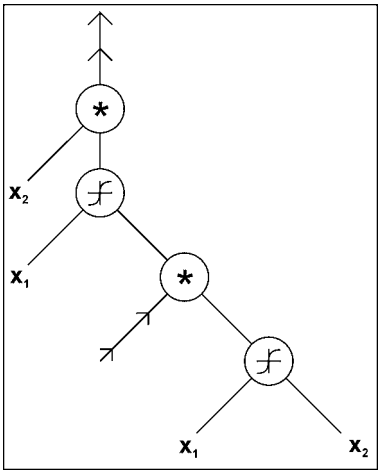
6.5.2 Tvar vyšlechtěné hranice mezi třídami

Jednotlivé tvary hranice mezi separovanými třídami jsou u vyšlechtěných ANN velmi různorodé. Přitom tvar této hranice patří mezi základní kritéria kvality syntetizované ANN. Je

ovšem otázkou jakou metodou toto kritérium hodnotit. Obecně se dá říci, že existují **vhodné** průběhy dané hranice a **defektní** průběhy (ANN4, ANN7, ANN8). Vhodné průběhy můžeme dále rozdělit na průběhy **přímé** (ANN1, ANN9), **oblé** (ANN2, ANN6, ANN10) a **ostré** (ANN3, ANN5). Zvláštností je, že v ostrém průběhu je vždy hranice mezi třídami zalomena přesně pod úhlem 95°. Přitom za všechny ostré ale i defektní průběhy jsou odpovědné právě násobné neurony (viz kap. 4.3.3).

6.5.3 Struktura syntetizované topologie

Obecně lze říci, že topologie syntetizovaných ANN je výrazně odlišná od standardních ANN (viz např. kap. 5.1.1 a kap. 5.1.2). Zatímco v běžných ANN je na malém prostoru (počtu neuronů) „nahuštěn“ velký počet ovládacích prvků (vah a prahů), v ANN, které vytváří náš algoritmus jsou tyto prvky spíše vzácné. Přitom se častěji syntetizuje práh neuronu, než váha na některém ze spojů v síti. Vyšlechtěným ANN dominují spíše pevné transformace. Tedy subfunkce uvnitř ANN, které jsou pouze jistou matematickou transformací vstupního signálu, ale na které nemá učení sítě žádný vliv. Tento fakt ovšem není syntetizovaným ANN na škodu a nijak jim nebrání v úspěšném řešení daného problému. Také nelze zaručit, že výstupním neuronem takové ANN bude vždy obsahovat sigmoidu, proto je nutné na výstupu sítě automaticky počítat ještě s jedním neuronem, který normuje výstupní signál do intervalu (0,1). Důvod proč je syntetizována topologie právě takového charakteru spočívá v povaze operací prováděných na ANN algoritmem AP (viz kap. 2.1).



Obr. 65. Pevná transformace v ANN7

6.5.4 Vliv GFS2 na výsledné ANN

Použití **GFS2** místo **GFS1** (viz kap. 6.4) také úspěšně řešilo zadaný problém, nicméně výrazně ovlivnilo výsledky syntézy. Takto získané ANN byly většinou jednoduší s větším počtem vah a prahů. Struktura jejich hranice mezi třídami byla stabilnější - nevytvářely se defektní průběhy. Ovšem množina takto získaných řešení byla ochuzena o celou škálu velmi zajímavých ostrých průběhů klasifikační hranice (viz kap. 6.5.2).

ZÁVĚR

Podařilo se nám úspěšně dosáhnout všech cílů, které jsme si na začátku naší práce vytýčili. Pomocí AP jsme vytvořili účinný algoritmus k syntéze ANN. Tímto procesem jsme dále syntetizovali širokou škálu různých ANN vhodných k řešení základních problémů. Diskutovali jsme strukturu a kvalitu nalezených řešení.

Tak jako každý správný vědecký výzkum, i ten náš za každou zodpovězenou otázku odhaluje tři další otázky, které je třeba zodpovědět. Opět jsme prokázali vysokou použitelnost analytického programování a přidali tak další aplikaci do bohaté sbírky tímto algoritmem úspěšně řešených problémů. Naučili jsme se syntetizovat neuronové a pseudoneuronové sítě s dopředným šířením, které úspěšně řeší klasifikační problémy do dvou tříd. Otevřeli jsme však také mnoho dalších oken, kterými se může ubírat následný výzkum. Základní otázkou, kterou zbývá zodpovědět je, podle jakých kritérií zhodnotit kvalitu nalezené ANN, máme-li celou množinu sítí s identickou reakcí na trénovací i testovací množinu.

Položili jsme platformu, kterou je třeba rozvíjet směrem k rozsáhlejším aplikacím - vícehodnotová klasifikace, aproximace, predikce, minimalizaci použitých ANN...

Směřujeme k formulaci nového pohledu na oblast ANN, ke křížení klasických ANN s ostatními funkcemi a transformacemi známými v matematice.

SEZNAM POUŽITÉ LITERATURY

[1] NOVÁK, M., FABER, J. a KUFUDAKI, O. 1993. *Neuronové sítě a informační systémy živých organismů*. Praha : Grada, 1993, 272 s. ISBN 8-85424-95-9.

[2] BOSE, B.K., LIANG, P. 1996. *Neural Network Fundamentals with Graphs, Algorithms, and Aplications*. McGraw-Hill Series in Electrical and Computer Engineering, 1996, 478 s. ISBN 0-07-006618-3.

[3] ZELINKA, I. 2002. *Umělá inteligence v problémech globální optimalizace*. Praha : BEN, 2002, 189 s. ISBN 80-7300-069-5.

[4] OPLATKOVÁ, Z. 2003. *Analytic Programming*, Zlín : UTB-FT, 2003, 73 s. (Diplomová práce)

[5] ŠNOREK, I. 2002 *Neuronové sítě a neuropočítače*, Praha : ČVUT, 2002, 156 s.

[6] KOZA J. R. 1998, *Genetic Programming II*, MIT Press.

[7] O'SULLIVAN J., CONOR R. 2002, *An Investigation into the Use of Different Secarch Strategies with Grammatical Evolution* Proceedings of 5th European Conference on Genetic Programming, p.268 - 277, Springer-Verlag London, UK.

[8] O'NEILL M. AND RYAN C. 2002, *Grammatical Evolution. Evolutionary Aromatic Pro-gramming in an Arbitrary Language*, Kluwer Academic Publisher.

[9] ZELINKA I., OPLATKOVÁ Z. 2003, *Analytic programming – Comparative Study*. CIRAS'03, The second International Conference on Computational Intelligence, Robotics, and Autonomous Systems, Singapore, 2003, ISSN 0219-6131.

[10] ZELINKA I. 2002, *Analytic programming by Means of Soma Algorithm. Mendel '02*, In: Proc. 8th International Conference on Soft Computing Mendel'02, Brno, Czech Republic, 2002, 93-101., ISBN 80-214-2135-5.

[11] ZELINKA I., OPLATKOVÁ Z. 2004, *Boolean Parity Function Synthesis by Means of Arbitrarry Evolutionary Algorithms - Comparative Study*, In: 8th World Multicon-ference on Systemics, Cybernetics and Informatics (SCI 2004), Orlando, USA, in July 18-21, 2004.

[12] ZELINKA I., OPLATKOVÁ Z., NOLLE L. 2004, *Boolean Symmetry Function Synthesis by Means of Arbitrarry Evolutionary Algorithms - Comparative Study*,

In: 18th European Simulation Multiconference (ESM 2004), Magdeburg, Germany, June 13-16, 2004, ISBN 3-936150-35-4.

[13] ZELINKA I. 2002, *Analytic programming by Means of Soma Algorithm*. ICICIS'02, First International Conference on Intelligent Computing and Information Systems, Egypt, Cairo, 2002, ISBN 977-237-172-3

[14] OPLATKOVÁ Z., ZELINKA I. 2006, *Investigation on Artificial Ant using Analytic Programming*, GECCO 2006, Seatle, WA, USA, 8-12.7.2006, 11th conference

[15] ZELINKA I. 2004, *SOMA - Self Organizing Migrating Algoritm*, kap. 7, str. 33, in B.V. Batu, G. Onwubolu (eds), New Optimization Techniques in Engineering, Springer-Verlag.

[16] KVASNIČKA V., POSPÍCHAL J., TIŇO P. 2002 *Evoluční algoritmy*, STU Bratislava, 2000, ISBN 85-246-2000.

[17] CHRAMCOV B. 2005, *Základy práce v prostředí Mathematica*, Zlín : UTB-FAI, 2005, 122 s. ISBN 80-7318-268-8.

[18] MASTER T., 1993, *Practical Neural Networks Recipes in C++,* London : Academi Press, 1993, 490 s. ISBN 0-12-479040-2.

[19] ZELINKA I., 2005, *Umělá inteligence*, Zlín : UTB-FAI, 2005, 127 s. ISBN 80-73318-277-7

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ANN	umělá neuronová síť (*Artificial Neuronal Network*)
ANN*	nenaučená ANN
$\mathbf{ANN_{all}}$	množina všech ANN
AP	Analytické programování (*Analytic Programming*)
arg	argument funkce
B	množina základních (bázových) funkcí **b** z nichž se skládá ANN
b	funkce $\mathbf{b} \in \mathbf{B}$
CD	přiložené záznamové médium
CF	účelová funkce (*Cost Function, Fitness Function*) použitá v EA
D	dimenze (rozměr jedince), počet argumentů **CF**
DE	Diferenciální evoluce (*Differential Evolution*)
EA	evoluční algoritmus (*Evolutionary Algorithm*)
E_T	globální chyba ANN
f*	třída funkcí $\mathbf{f^*} \in \mathbf{F}$ reprezentují nenaučenou ANN
f	funkce **f** z třídy **f*** reprezentující naučenou ANN
F	množina všech přípustných tříd **f*** vzniklých kombinací prvků množiny **B**
$\mathbf{F_{all}}$	množina všech funkcí
F	mutační konstanta DE
GE	Gramatická evoluce (*Grammatical Evolution*)
GFS	základní množina AP (*General Function Set*)
gf	základní funkce AP $\mathbf{gf} \in \mathbf{GFS}$
GP	Genetické programování (*Genetic programming*)
K	konstantní funkce $\mathbf{gf}_i \in \mathbf{GFS}$, $\mathbf{gf}_i = K_i$, kde K_i je později determinováno, např. pomocí metod nelineárního prokládání

NP	počet jedinců v populaci SOMA nebo DE
P	úloha (problém) předložený k řešení ANN
PRT	parametr SOMA
SOMA	Samoorganizující se migrační algoritmus (*Self-organizing Migration Algorithm*)
w	váha na spoji v ANN
XOR	logická funkce
ϕ	práh neuronu v ANN
ξ	maximální přípustná E_T
$\in$	a $\in$ **B**, a je prvkem **B**
$\subset$	A $\subset$ B, **A** je podmnožinou **B**
$\subseteq$	**A** $\subseteq$ **B**, **A** je podmnožinou, nebo identickou množinou s **B**
$\mathbb{E}^3$	Euklidův trojdimenzionální prostor
3D	zobrazení v $\mathbb{E}^3$

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: ADREÁŘOVÁ STRUKTURA PŘILOŽENÉHO CD

Protože stromová struktura přiloženého CD může být pro nezasvěceného čtenáře značně nepřehledná. Rozhodli jsme se přiložit její stručný popis, který má za cíl usnadnit eventuelnímu zájemci orientaci. Vždy nejprve uvádíme cestku ke konkrétnímu adresáři a poté stručný popis jeho obsahu.

- \perceptron

Obsahuje zdrojové soubory a spustitelnou verzi programu Perceptron.

- \prace

Obsahuje text této diplomové práce.

- \synteza\gfs1\lin\algorimus\cf1

Obsahuje zdrojový kód algoritmu AP řešící problém *P1* pomocí **GFS1** a CF1.

- \synteza\gfs1\lin\algorimus\cf2

Obsahuje zdrojový kód algoritmu AP řešící problém *P1* pomocí **GFS1** a CF2.

- \synteza\gfs1\lin\priklady

Obsahuje použité příklady ANN1 až ANN4.

- \synteza\gfs1\xor\algorimus\cf1

Obsahuje zdrojový kód algoritmu AP řešící problém *P2* pomocí **GFS1** a CF1.

- \synteza\gfs1\xor\algorimus\cf2

Obsahuje zdrojový kód algoritmu AP řešící problém *P2* pomocí **GFS1** a CF2

- \synteza\gfs1\xor\priklady

Obsahuje použité příklady ANN5 až ANN8.

- \synteza\gfs1\lin\algorimus\cf1

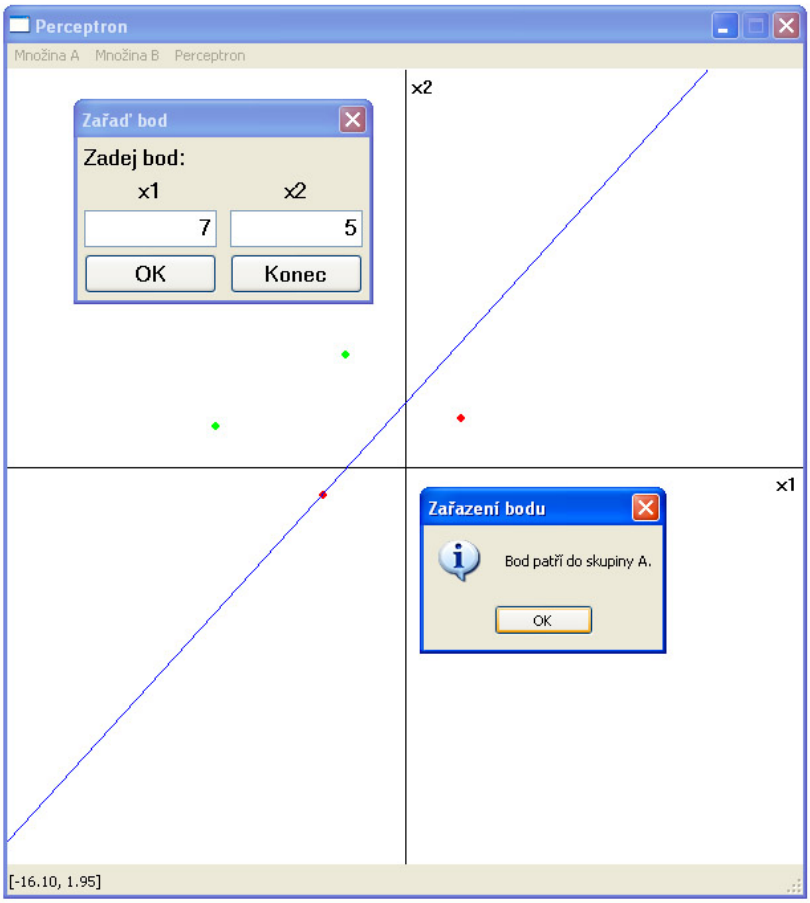
Obsahuje zdrojový kód algoritmu AP řešící problém *P1* pomocí **GFS2** a CF1

- \synteza\gfs2\lin\priklady

Obsahuje použité příklady ANN9 a ANN10.

PŘÍLOHA P II: PROGRAM PERCEPTRON

Všechny algoritmy použité v této práci jsou navrženy pro programové prostředí **Mathematica**. Protože zvídavý čtenář, který se teprve seznamuje s problematikou neuronových sítí nemusí toto nákladné prostředí nutně vlastnit. Rozhodl jsme se (Vařacha P.) naprogramovat alespoň malý příklad použití neuronu ke klasifikaci a uvolnit jeho zdrojové kódy k volnému použití. CD obsahuje také spustitelnou verzi tohoto programu s názvem **Perceptron**, která je zkompilována pro operační systém Windows. Nicméně zdrojové kódy tohoto programu jsou sestaveny multiplatformě a je možné je překompilovat téměř pro libovolný operační systém. Tato skutečnost je umožněna použitím vynikající *open source* multiplatformní knihovny **wxWidget** (http://www.wxwidgets.org/). Celý program vznikl za pomoci *freeware* nástrojů **wx-Devcpp** (http://wxdsgn.sourceforge.net/) a **wxGlade** (http://wxglade.sourceforge.net/). Domnívám se, že program je příkladem vhodného použití wxWidged a objektového programování.



Ovládání programu Perceptron je velmi jednoduché, nejprve je třeba navrhnout dvě separované třídy (množinu prvků **A** a **B**). To můžeme udělat například klikáním levého a pra-

vého tlačítka myši na souřadnicovou plochu hlavního okna. Potom zvolíme menu *Perceptron - Nauč perceptron*. Jestliže jsme před tím nadefinovali lineárně separabilní problém, jediný neuron sítě programu Perceptron (v okně reprezentovaný modrou přímkou) bude úspěšně naučen metodou fixních přírůstků. Máme-li naučený neuron, můžeme jej pomocí menu *Perceptron - Zařaď prvek...* nechat klasifikovat uživatelem vybrané prvky (body z plochy).