ZÁKLADY SOFTCOMPUTINGU

URČENO PRO VZDĚLÁVÁNÍ V AKREDITOVANÝCH STUDIJNÍCH PROGRAMECH

EVA VOLNÁ

ČÍSLO OPERAČNÍHO PROGRAMU: CZ.1.07
NÁZEV OPERAČNÍHO PROGRAMU:
VZDĚLÁVÁNÍ PRO KONKURENCESCHOPNOST
OPATŘENÍ: 7.2
ČÍSLO OBLASTI PODPORY: 7.2.2

INOVACE VÝUKY INFORMATICKÝCH PŘEDMĚTŮ VE STUDIJNÍCH PROGRAMECH OSTRAVSKÉ UNIVERZITY

REGISTRAČNÍ ČÍSLO PROJEKTU: CZ.1.07/2.2.00/28.0245

OSTRAVA 2012

Tento projekt je spolufinancován Evropským sociálním fondem a státním rozpočtem České republiky

Recenzent: RNDr. Martin Kotyrba, Ph.D.

Název: Základy softcomputingu
Autor: doc. RNDr. PaedDr. Eva Volná, PhD.
Vydání: první, 2012
Počet stran: 152

Jazyková korektura nebyla provedena, za jazykovou stránku odpovídá autor.

OBSAH

Úvod pro práci s textem pro distanční studium

Cíl předmětu

V posledních letech se v praxi můžeme setkat s přístupy a principy, které jsou založeny na poměrně nové vědní disciplíně, která je označována zkratkou SC - Soft Computing. Cílem předmětu *Základy softcomputingu* je seznámit studenty s jeho základními principy, metodami a aplikacemi.

Softcomputing se zabývá symbiózou různých výpočetních postupů, jejichž společným jmenovatelem je odklon od klasického modelování založeného na booleovské logice, analytických modelech, ostré klasifikaci a deterministickém prohledávání. V názvu uvedené slovo „soft", vyjadřující lexikálně „měkkost, mírnost", tady znamená „měkké" požadavky na přesnost popisovaných jevů. Mezi hlavní zástupce SC zahrnujeme fuzzy logiku (FL), neuronové sítě (NS) a genetické algoritmy (GA).

Fuzzy logika spočívá v rozšíření logických operátorů na fuzzy množiny.

Neuronové sítě jsou výpočetní struktury, které mají schopnost učení.

Genetické algoritmy provádějí náhodné prohledávání prostoru za pomoci imitace živé přírody. V tomto postupu probíhá modelová evoluce od vzniku jedinců přes jejich selekci a křížení až po jejich nahrazení dokonalejšími jedinci.

Po prostudování textu budete znát:

Tyto učební texty jsou určeny studentům informatiky pro *Základy softcomputingu*. Úvodní kapitoly jsou věnovány základům fuzzy logiky a fuzzy řízení. Další kapitoly představují úvod do problematiky strojového učení. Dále následují kapitoly věnované umělým neuronovým sítím a evolučním algoritmům, jež lze zařadit mezi přírodou inspirované metody strojového učení. Závěrečné kapitoly presentují aplikace SC metod.

V textu jsou dodržena následující pravidla:

- je specifikován cíl lekce (tedy co by měl student po jejím absolvování umět, znát, pochopit)
- výklad učiva
- důležité pojmy
- úkoly a otázky k textu
- korespondenční úkoly (mohou být sdruženy po více lekcích)

Úkoly

Vyberte si jeden korespondenční úkol a vypracujte na toto téma semestrální projekt, jehož obhajoba proběhne v dohodnutém termínu. Podrobné informace obdržíte na úvodním tutoriálu.

Pokud máte jakékoliv věcné nebo formální připomínky k textu, kontaktujte autora (eva.volna@osu.cz).

1. Hard-Computing vs. Soft-Computing

V této kapitole se dozvíte:

- Jaké jsou hlavní rysy hard-computingu.
- Jaké jsou hlavní rysy soft-computingu.

Po jejím prostudování byste měli být schopni:

- Vymezit rozdíly mezi hard-computingem a soft-computingem.
- Uvést, jaké oblasti soft-computing zastřešuje.

Klíčová slova této kapitoly:

Hard-computing, soft-computing, výpočetní inteligence.

Průvodce studiem

Vymezme si dva pojmy, a sice Hard-Computing a Soft-Computing. Jedná se o dva přístupy k řešení problémů (abstrahujme od konkrétních metod). Celá kapitola je napsána podle (Štěpnička 2010) a si vymezíme v ní rozdíly mezi oběma zmíněnými přístupy.

1.1 Hard-Computing

Hlavní rysy Hard-Computingu jsou:

1. používáme model reálné situace (např. fyzikální model, ekonomický model);
2. je hledáno přesné řešení (např. fyzikální model vede na soustavu parciálních diferenciálních rovnic, tj. rovnic, kde neznámou jsou funkce, pak je hledáno analytické řešení - čili analytický zápis hledaných funkcí);

3. analytické řešení je hledáno jen na teoretické úrovni, v praxi je stejně často nenalezitelné (alespoň ne v požadovaném čase) a musíme přistoupit k hledání přibližného řešení pomocí numerických metod;
4. při hledání přibližného řešení je hledáno co nejlepší možné řešení z předem daného prostoru přípustných řešení, tj. nastupuje problém optimalizace.

Z výše uvedených rysů vyplývá nebezpečí následujících negativ při použití Hard-Computingu:

- nutná znalost fyzikálních, ekonomických či jiných mechanismů (např. zmiňovaných parciálních diferenciálních rovnic popisujících dané procesy);
- i při snaze o přesnost a exaktnost dochází k chybám, zanedbávání mnoha vlivů, zaokrouhlování apod.;
- každý model je do jisté míry idealizací reálné situace a je tedy opět nepřesným modelem (např. na středních školách učený fyzikální model volného pádu pracuje s vakuem, nikoliv odporem vzduchu, který bude v různých podmínkách také různý);
- přístup si vynucuje vysoké výpočetní nároky;
- není vždy zajištěna existence řešení;
- nedostatek robustnosti (při drobné změně údajů či parametrů na vstupu může být již získaný model nefunkční).

Výsledkem výše uváděného je obecně přijímaný fakt, že nikoliv v každé situaci je Hard Computing opodstatněný.

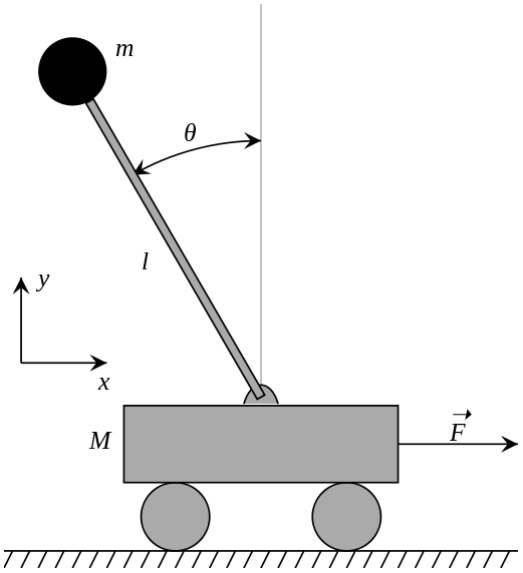
1.2 Soft-Computing

Hlavní rysy Soft-Computingu jsou oproti Hard-Computingu následující:

1. není důležitý model mechanismů, kterými se chování studovaného systému řídí (např. fyzikální model či ekonomický model ve formě rovnic popisujících daný mechanismus - tedy „vnitřní znalosti");

2. důležitá je existence „znalostí vnějších”a nikoliv vnitřních, tedy existence znalostí o chování systému, nikoliv znalostí o příčinách chování.

Nyní si uvedené přiblížíme na typickém příkladu invertovaného kyvadla. Invertované kyvadlo je zjednodušeně kyvadlo směrem vzhůru. Kdybychom chtěli invertované kyvadlo uřídit pohyby automaticky řízeného vozíku (obrázek 1), na kterém by kyvadlo bylo postaveno, stojí před námi nelehký úkol.



Obrázek 1: Invertované kyvadlo na automaticky řízeném vozíku (Zdroj http://en.wikipedia.org/wiki/Inverted pendulum)

Řešení pak lze směřovat cestou Hard-Computingu, anebo Sof-Computingu. V případě hledání řešení cestou Hard-Computingu si nastudujeme fyziku, popíšeme pomocí parciálních diferenciálních rovnic daný proces, implementujeme do PC numerický algoritmus přibližného výpočtu a pak získáme řídící funkci pro daný vozík. V případě hledání řešení cestou Soft-Computingu je jedno, jaké fyzikální zákony balancování řídí. Uvědomíme si však vnější znalosti, tj. znalosti, které nepopisují příčiny chování, ale chování samotné. Tyto znalosti budou např. ve formě IF-THEN (jestliže-pak) pravidel: *Jestliže je kyvadlo mírně vychýleno doprava, jemně posunu vozíkem také doprava*. Tedy pravidel

popisujících logickou závislost mezi odchylkami a reakcemi. A již máme první možnost, jak problém řešit. Použitím logiky můžeme IF-THEN pravidla symbolicky reprezentovat ($A \Rightarrow B$) a automaticky pomocí dedukčního pravidla a naměřených odchylek kyvadla z osy (A) odvozovat tzv. akční zásahy čili pohyby vozíku (B). Tím získáme input / output body, kterými proložíme křivku a získáme funkcionální závislost mezi odchylkami a akčními zásahy. Tento přístup je používán neuronovými sítěmi, které ze začátku „nic neumí", stejně jako v případě člověka po narození, ale časem (s více a více pokusy) se „učí", tj. hledají správné nastavení, které pak již mají zafixované.

Z výše uvedeného lze tedy shrnout, že v případě Soft Computingu platí následující:

- iluzorní přesnost není základem;
- stejně tak zde již není nutnost znalosti fyzikálních, ekonomických či jiných mechanismů;
- ani optimalita není základem, protože bychom se stejně snažili optimalizovat pouze idealizovaný model;
- hovoří se o praktické optimalizaci, tedy o stavu, kdy není dosaženo teoretického optima, ale z praktického hlediska se jedná o již tak marginální rozdíl, že by další vylepšování ničemu příliš nepomohlo;
- základními kritérii jsou jednoduchost, implementovatelnost a robustnost (např. při změně délky a vyvážení tyče je fyzikální model již nefunkční, nicméně IF-THEN pravidla nevycházejí z fyzikální podstaty a popisují vnější znalost, která zůstává víceméně stejná).

Pojem *Soft-Computing* zavedl Lotfi A. Zadeh (Zadeh, 1973) jako název zastřešující tři základní oblasti, a sice *fuzzy* (ať už logiku či modelování, chápání pojmu fuzzy logika v širším slova smyslu se již poměrně vzdálilo formální logice a měli bychom spíše hovořit o fuzzy

modelování), *neuronové sítě* a *evoluční algoritmy*. Pochopitelně se jedná o otevřenou definici, takže sem řadíme jak hybridní systémy kombinující dva a více přístupů z těchto tří oblastí, a dále nově vznikající příbuzné oblasti.

Pojem Soft-Computing je sám o sobě diskutabilní, protože obsahuje také vysoce teoretické podoblasti, které mohou slovo „computing" jemně degradovat. Navíc byl pojem definován L.A. Zadehem, který měl na mysli hlavně tři základní oblasti (fuzzy, neuro, evoluční) a jejich hybridní kombinace. Nicméně při striktním držení se tohoto původního vymezení by se komunita Soft-Computingu stala poměrně uzavřenou. Proto se poslední dobou prosazuje používat spíše pojem *výpočetní inteligence* (*Computational Intelligence*), zkráceně CI. Pojem CI nám tedy označuje oblasti umělé inteligence zahrnující Soft Computing, nicméně otevřené novým příbuzným oblastem (inteligence hejna, fraktály, strojové učení apod.).

Kontrolní otázky:

1. Jaké jsou hlavní rysy hard-computingu?
2. Jaké jsou hlavní rysy soft-computingu?

Shrnutí obsahu kapitoly

V této kapitole byly vymezeny pojmy Hard-Computing a Soft-Computing. Jedná se o dva přístupy k řešení problémů. Celá kapitola je napsána podle (Štěpnička 2010)

Pojmy k zapamatování

- hard-computing
- soft-computing
- výpočetní inteligence

2 Základní pojmy fuzzy logiky

V této kapitole se dozvíte:

- Co je to fuzzy logika.
- Na jakém principu pracují fuzzy množiny.
- Co je to funkce příslušnosti.
- Jaké existují operace s fuzzy množinami.
- Jaké jsou metody defuzzifikace.

Po jejím prostudování byste měli být schopni:

- Znát základní rozdíl mezi klasickou a fuzzy množinou.
- Definovat pojem funkce příslušnosti.
- Znát základní operace s fuzzy množinami a metody defuzzifikace.

Klíčová slova této kapitoly:

Fuzzy logika, fuzzy množina, funkce příslušnosti, jazyková (lingvistická) proměnná, fuzzy pravidla, defuzzifikace.

Průvodce studiem

V této kapitole si osvojíte základní informace o fuzzy logice a fuzzy množinách. Získáte přehled o důležitých pojmech, jako funkce příslušnosti a jak se tvoří fuzzy pravidla. Dále se seznámíte se základními operacemi s fuzzy množinami a metodami defuzzifikace.

2.1 Význam fuzzy logiky

Celou kapitolu týkající se fuzzy logiky začneme netradičně výrokem slavného génia Alberta Einsteina *„Pokud matematika popisuje realitu, není přesná. A pokud je přesná, nepopisuje realitu“.* Tento výrok vystihuje podstatu nepřesné formulace a to, jaký význam nabývá přesný popis daného problému.

Fuzzy logika se poprvé objevila v roce 1965 v článku, jehož autorem byl profesor Lotfi A. Zadeh. Tehdy byl definován základní pojem fuzzy logiky, a to fuzzy množina. Slovo fuzzy znamená neostrý, matný, mlhavý, neurčitý, vágní. Odpovídá tomu i to, čím se fuzzy teorie zabývá: snaží se pokrýt realitu v její nepřesnosti a neurčitosti (Zadeh, 1973).

V klasické teorii množin prvek do množiny buďto patří (úplné členství v množině) nebo nepatří (žádné členství v množině). (Pokorný, 2004)

Fuzzy množina je množina, která kromě úplného nebo žádného členství připouští i členství částečné. To znamená, že prvek patří do množiny s jistou mírou příslušnosti (stupeň příslušnosti). Funkce, která každému prvku universa přiřadí stupeň příslušnosti, se nazývá funkce příslušnosti.

Striktní popis vede k popisu skutečnosti pouze pomocí dvouprvkové množiny {0,1}. Pokud problém nelze jednoznačně určit, rozkládá se na menší podproblémy a opět lze použít jen dvouprvkovou množinu. V případech, kdy je již nemožné nebo neúnosné takto problém rozdělit, dopouštíme se jisté chyby a tím je dán odklon od reality (Pokorný, 2004). S tím souvisí i princip inkompatibility, který vyslovil v roce 1973 L. A. Zadech: *„S rostoucí složitostí sytému klesá naše schopnost formulovat přesné a významné vlastnosti o jeho chování, až je dosáhnutá hranice, za kterou je přesnost a relevantnost prakticky vzájemně se vylučující jevy.“*

2.2 Fuzzy množiny

V klasické teorii množin je možno množinu popsat několika způsoby:

a) výčtem prvků množiny *M* = {*x1, x2, x3, x4*}

b) pravidlem, kterému musí prvky vyhovovat

c) charakteristickou funkcí $m_M(x)$, pro kterou platí (1)

$$m_M(x) = \begin{cases} 1 & if \quad x \in M \\ 0 & if \quad x \notin M \end{cases} \tag{1}$$

Prvek *x* v klasické teorii množin do množiny bud patří, nebo nepatří, protože jeho charakteristická funkce nabývá hodnot 1 nebo 0. Hovoříme pak o ostrých množinách - ostrém rozlišení při rozhodování o příslušnosti. Pokud ale uvádíme stupeň, s jakým prvek do množiny patří, pak tyto množiny označujeme jako množiny neostré - fuzzy množiny.

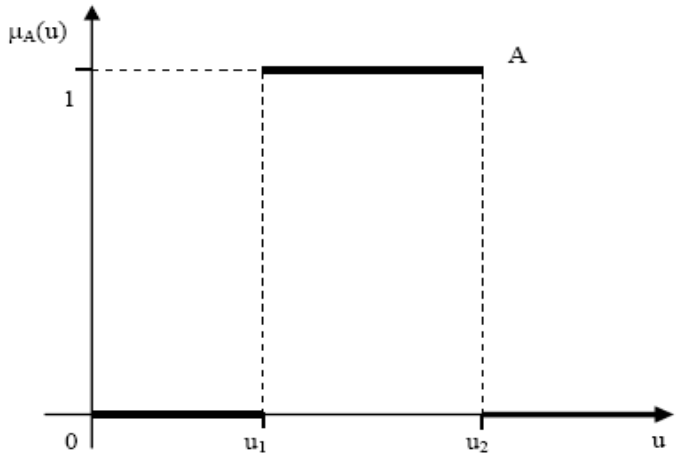
Stupeň příslušnosti (náležení) prvku univerza *U* do klasické množiny A (Mařík, 2003), (Berka, 2010) je dán funkcí $\gamma_A : U \to [0,1]$.

Její velikost může nabývat dvou hodnot:

- stupeň příslušnosti 0 – prvek do množiny A (plně) nenáleží
- stupeň příslušnosti 1 – prvek do množiny A (plně) náleží.

Jak jsme již dříve konstatovali, v převážné většině praktických případů lze jen obtížně tvrdit, že určitý prvek do určité množiny náleží či nenáleží. I když prvek nese dominantní znaky vlastností prvků určité množiny A, může také v menší míře vykazovat znaky vlastností množiny B. Rozhodnutí o přiřazení prvku do množiny A je pak nejednoznačné. Situaci může efektivně řešit přístup, v němž kromě pojmu absolutního náležení či nenáležení prvku do určité množiny zavedeme pojem částečného náležení prvku do množiny. Jde o zobecnění pojmů stupně přípustnosti „1“, kdy rozšíříme definiční obor jeho hodnot ze dvou diskrétních {0,1} na uzavřený interval ⟨0,1⟩, tj. $\mu_F : U \to \langle 0,1 \rangle$.

Množiny, které umožňují definovat velikost stupně náležení prvků, se nazývají fuzzy množiny. Fuzzy množina F je definována jako přiřazení, které každému prvku *u* univerza *U* přiřazuje hodnotu funkce jeho příslušnosti do fuzzy množiny *F* rovnou $\mu_F(u)$, tj. $F = \{(\mu_F(u))/u \in U\}$ (Pokorný, 2004)

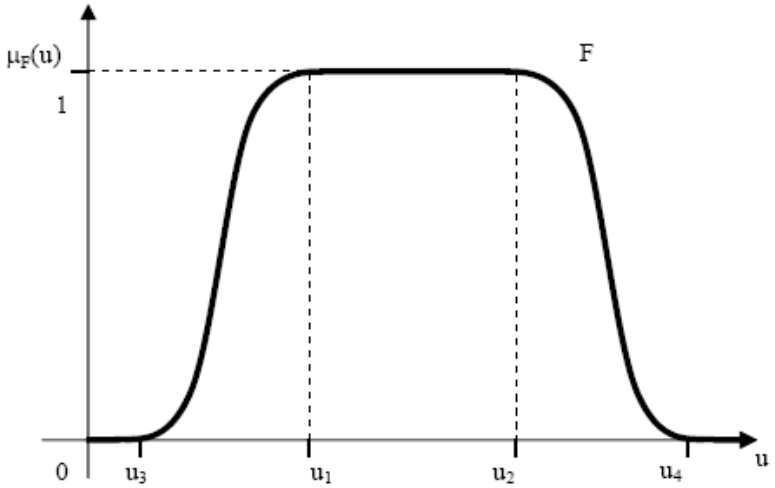


Obrázek 2: Průběhu funkce příslušnosti $\mu_A(u)$ klasické množiny A

V praxi je fuzzy množina F prvků *u* ztotožněná s její funkci příslušnosti $(\mu_F(u))$. Při popisu grafického průběhu funkce příslušnosti $(\mu_F(u))$ vyjdeme nejprve z průběhu funkce příslušnosti $(\mu_F(u))$ klasické množiny A, nakresleného na obrázku 2.

Prvky $u \in U$ z uzavřeného intervalu $\langle u_1, u_2 \rangle$ do množiny A zcela jistě patří, ostatní prvky univerza do množiny A zcela jistě nepatří.

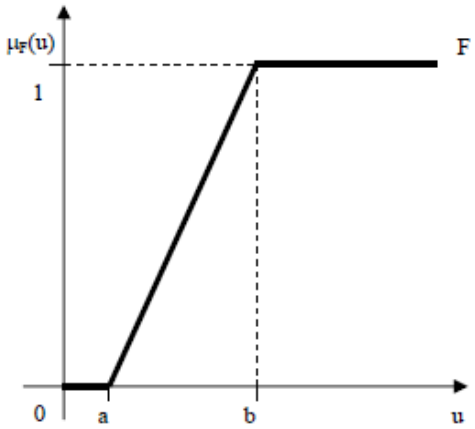
Na obrázku 3 je nakreslena funkce příslušnosti fuzzy množiny *F*, definované opět na univerzu *U*. Prvky $\langle u_1, u_2 \rangle$ opět do fuzzy množiny *F* zcela jistě patří, prvky *u* z intervalu (-∞,u_3) a prvky *u* z intervalu (u_4, +∞) do fuzzy množiny *F* zcela jistě nepatří (Mařík, 2003).



Obrázek 3: Funkce příslušnosti fuzzy množiny *F*

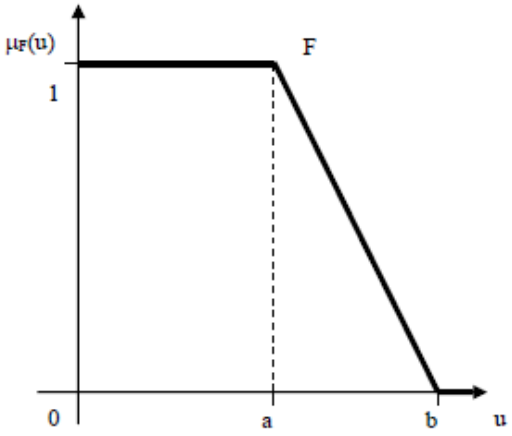
Prvkům z intervalů (u_3,u_1) a (u_2,u_4) je funkcí $(\mu_F(u))$ přiřazena hodnota jejich příslušnosti do fuzzy množiny *F* reálným číslem z intervalu (0,1) a vyjadřuje tedy jejich příslušenství částečné.

Praktické použití fuzzy množin vyžaduje analytické vyjádření funkce $(\mu_F(u))$. V praxi používáme nejčastěji jejich aproximaci lomenými přímkami. Příklady takových aproximací a jejich analytické odpovídající aproximaci jsou uvedeny na obrázcích 4 až 7. Funkce příslušnosti jsou parametrizovány jejich čtyřmi body zlomu, tedy hodnotami [a, b, c, d]. (Pokorný, 2004).



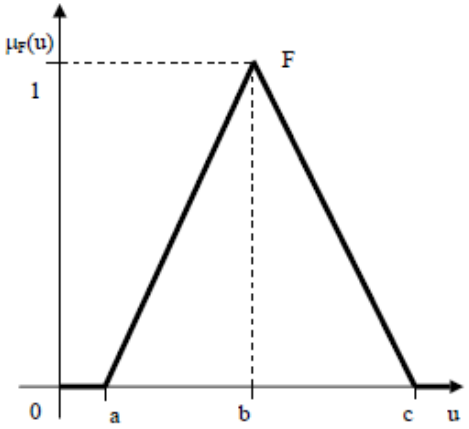
$$\Gamma(u,a,b)=\begin{cases}0 & u<a\\ (u-a)/(b-a) & a\le u\le b\\ 1 & u>b\end{cases}$$

Obrázek 4: Aproximace funkce příslušnosti fuzzy množiny lomenými přímkami



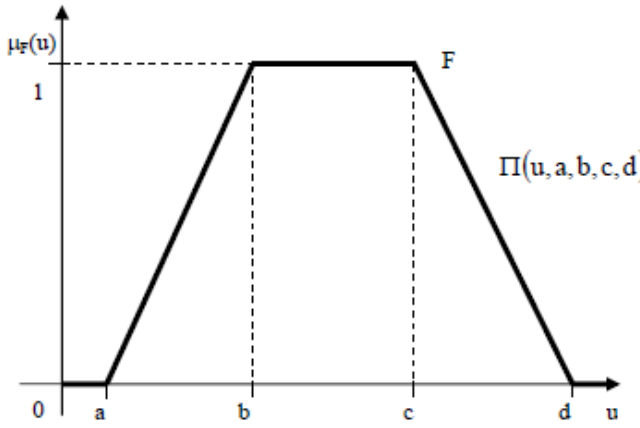
$$L(u,a,b)=\begin{cases}1 & u<a\\ (b-u)/(b-a) & a\le u\le b\\ 0 & u>b\end{cases}$$

Obrázek 5: Aproximace funkce příslušnosti fuzzy množiny lomenými přímkami



$$\Lambda(u,a,b,c)=\begin{cases}0 & u<a\\ (u-a)/(b-a) & a\le u\le b\\ (c-u)/(c-b) & b\le u\le c\\ 0 & u>c\end{cases}$$

Obrázek 6: Aproximace funkce příslušnosti fuzzy množiny lomenými přímkami

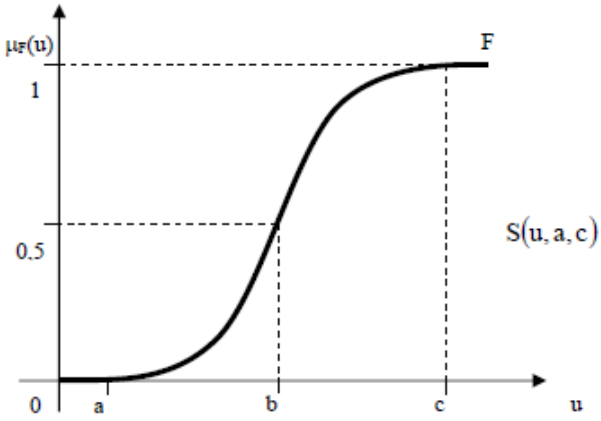


$$\Pi(u,a,b,c,d)=\begin{cases}0 & u<a\\ (u-a)/(b-a) & a\le u\le b\\ 1 & b\le u\le c\\ (d-u)/(c-d) & c\le u\le d\\ 0 & u>d\end{cases}$$

Obrázek 7: Aproximace funkce příslušnosti fuzzy množiny lomenými přímkami

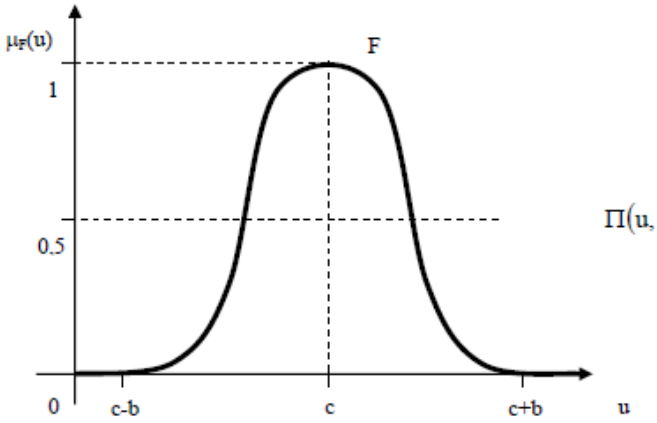
Případ fuzzy množiny, jejíž funkce příslušnosti má trojúhelníkový tvar (obrázek 6), má zvláštní důležitost. Taková fuzzy množina totiž formalizuje ne zcela přesné, tzv. fuzzy číslo, v daném případě reprezentované jazykovým výrazem „asi b“. Fuzzy čísla (vedle čísel obyčejných, ostrých – zde „přesně b“ mají velký praktický význam. Uvědomíme si, že ostré číslo můžeme rovněž reprezentovat fuzzy množinou (normální fuzzy množina definovaná na jediném prvku univerza). Takovou fuzzy množinu pak nazýváme singleton. Singletonem je tedy fuzzy množina, odpovídající obyčejnému ostrému číslu.

Aproximace funkce příslušnosti fuzzy množiny spojitou funkcí je uvedena na obrázcích 8 a 9. Křivky jsou parametrizovány dvěma hodnotami – [a,c] resp. [b,c]. (Pokorný, 2004)



$$S(u,a,c)=\begin{cases}0 & u\le a\\ 2(u-a)^2/(c-a)^2 & a<u\le b\\ 1-2(u-c)^2/(c-a)^2 & b<u\le c\\ 1 & u>c\end{cases}$$

Obrázek 8: Aproximace funkce příslušnosti fuzzy množiny spojitou funkcí



$$\Pi(u,b,c)=\begin{cases}S(u,c-b,c) & u\le c\\ 1-S(u,c,c+b) & u\ge b\end{cases}$$

Obrázek 9: Aproximace funkce příslušnosti fuzzy množiny spojitou funkcí

2.3 Jazykové proměnné

Jazyková (lingvistická) proměnná je taková proměnná, jejíž hodnoty jsou výrazy nějakého jazyka. Hodnotu Jazykové proměnné můžeme interpretovat jako fuzzy množiny. Množina jazykových hodnot se označuje jako množina termů, jež jsou definovány na univerzu (univerzální množina).

Ilustrační příklad:

Při regulaci teploty lázně můžeme teplotu kapaliny chápat jako jazykovou proměnnou s názvem „Teplota lázně“. Teplotu měříme ve stupních Celsia. Kvantitativní vyjádření teploty lázně v hovorovém jazyce však nemusí být vyjádřeno jen ve stupni, ale v běžně používaných výrazech jako: lázeň je LEDOVÁ, STUDENÁ, VLAŽNÁ, TEPLÁ atd.

Jako *hodnotu jazykové proměnné* „Teplota lázně“ pak můžeme označit prvek z množiny teplot: {*ledová* (L), *studená* (S), *vlažná* (v), *teplá* (T), *horká* (H)}.

Takto zavedená jazyková kvantifikace teplot (např. *studená*) představuje term, který označuje neostrou množinu, pro kterou je možno definovat funkcí příslušnosti $\mu_S(u)$.

Jako příklad funkcí příslušnosti uvádíme neostré množiny *studená* a *vlažná* a jejich funkce příslušnosti $\mu_S(u)$ a $\mu_V(u)$. Každý z termů studená a vlažná je definován funkcí příslušnosti na určitém intervalu teplot (univerza) ve stupních Celsia.

Funkci příslušnosti neostré množiny osvětlíme na následujícím příkladu (Modrlák, 2002):

- naměříme-li teplotu *u* = 20°C, pak $\mu_S(u) = 0$ a jistě tato naměřená teplota nepatří do termu – jazykové hodnoty *studená*.

- naměříme-li teplotu *u* = 0°C, pak $\mu_S(u) = 0.5$, což indikuje, že tato naměřená teplota patří do termu - jazykové hodnoty *studená* stupněm příslušnosti 0.5
- naměříme-li teplotu *u* = - 10°C, pak $\mu_S(u) = 1$ a je zřejmé, že tato naměřená teplota patří do termu - jazykové hodnoty *studená* stupněm příslušnosti 1
- naměříme-li teplotu *u* = - 20°C, pak $\mu_S(u) = 1$ a také tato naměřená teplota patří do termu – jazykové hodnoty *studená* se stupněm příslušnosti 1
- naměříme-li teplotu *u* = +15°C, pak je $\mu_S(u) = 0.25$ a tato naměřená teplota patří do termu – jazykové hodnoty *studená* se stupněm příslušnosti 0.25. Ale pozor, funkce příslušnosti $\mu_V(u) = 1$ z čehož plyne, že tato naměřená teplota patří také do množiny-termu *vlažná* se stupněm příslušnosti 1.

Proces přiřazování měřených hodnot vstupních veličin do fuzzy množin pomocí funkcí příslušností se označuje jako *fuzzyfikace*.

Pro označování hodnot jazykové proměnné se používá standardní označení. Typické označení termů – fuzzy hodnot a jejich zkratek, včetně anglického označení, je v tabulce 1 (Modrlák, 2002).

Význam	*Ozn. Česky*	*Ozn. Anglicky*
Hodnota velká záporná	ZV	NL
Hodnota střední záporná	ZS	NM
Hodnota malá záporná	ZM	NS
Hodnota záporná blízká nule	ZN	NZ
Hodnota nulová	NU	Z
Hodnota kladná blízká nule	KN	PZ
Hodnota malá kladná	KM	PS
Hodnota střední kladná	KS	PM
Hodnota velká kladná	KV	PL

Tabulka 1: Typické označení termů – fuzzy hodnot a jejich zkratek

2.4 Operace s fuzzy množinami

Uveďme nejprve další možné formy zápisu funkce příslušnosti fuzzy množiny *F*. Je-li fuzzy množina *F* definována na diskrétním univerzu s *n*- prvky, lze ji určit výčtem dvojic $\mu_F(u)/u$ ve tvaru zápisu:

$$F = \{\mu_F(u_1)/u_1, \mu_F(u_2)/u_2, ..., \mu_F(u_n)/u_n\}.$$

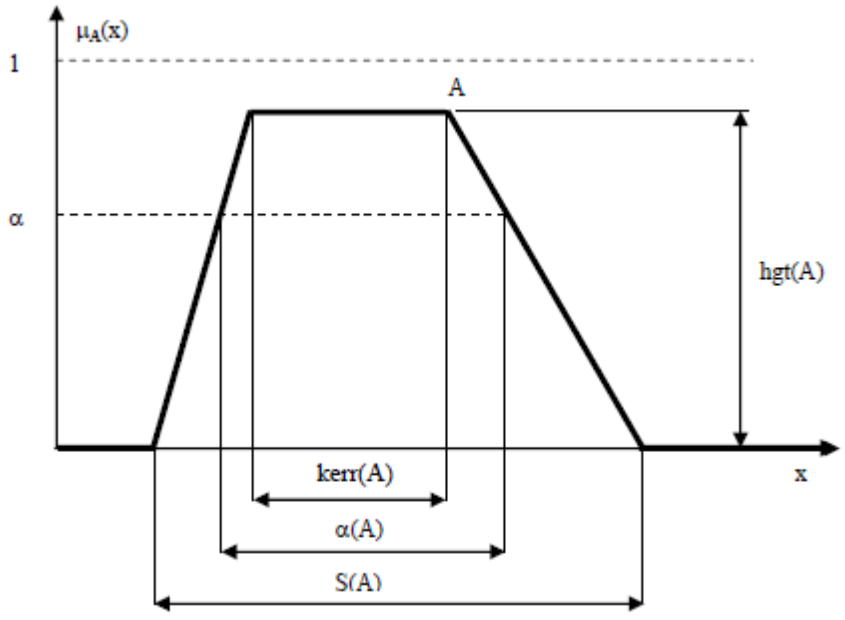
V tomto zápisu se někdy používá znaménko + a konečnou množinu F lze zapsat ve tvaru:

$$F = \mu_F(u_1)/u_1 + \mu_F(u_2)/u_2 + ... + \mu_F(u_n)/u_n.$$

Fuzzy množinu *F*, definovanou na diskrétním konečném nebo spočetném univerzu *U* můžeme formalizovat zápisem:

$$F = \sum_{u \in U} \mu_F(u)/u\,.$$

v případě univerza spojitého nebo nespočetného pak formálně použijeme symbol integrálu $\int_U \mu_F(u)/u$.

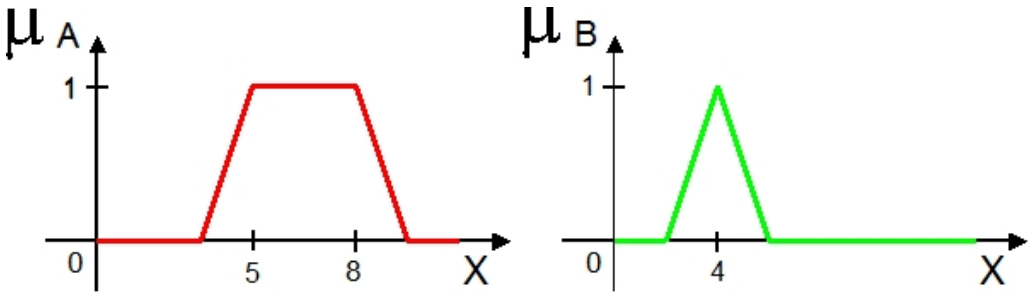


Obrázek 10: Parametry fuzzy množiny A

Pro kvantitativní vyjádření tvaru funkce příslušnosti fuzzy množiny A použijeme následující parametry, jejichž význam plyne z obrázku 10 (Pokorný, 2004):

- *nosič fuzzy množiny* A: S(A) = {x / $\mu_A(x) > 0$}
- *jádro fuzzy množiny* A: kerr(A) = {x $\in X$ / $\mu_A(x) = 1$}
- *výšku fuzzy množiny* A: $\mathrm{hgt}(A) = \sup_{x \in X}(\mu_A(x))$
- *α- řez fuzzy množiny* A: α(A) = {x $\in X$ / $\mu_A(x) \geq \alpha$}

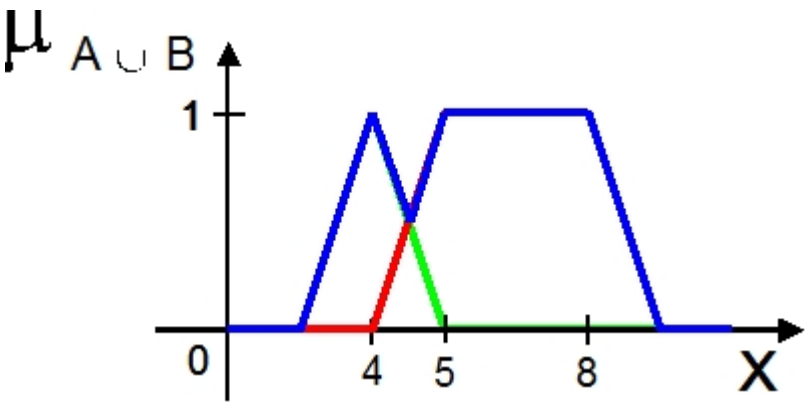
Při pojednání o fuzzy množinových operacích vyjdeme z operací množin klasických (ostrých). Budeme tedy hovořit o operaci průniku, sjednocení a doplňku. Uvažujme dvě fuzzy množiny A a B z obrázku 11 definované na jediném univerzu X, popsané svými funkcemi příslušnosti.



Obrázek 11: Fuzzy množiny A a B

Logický součet:

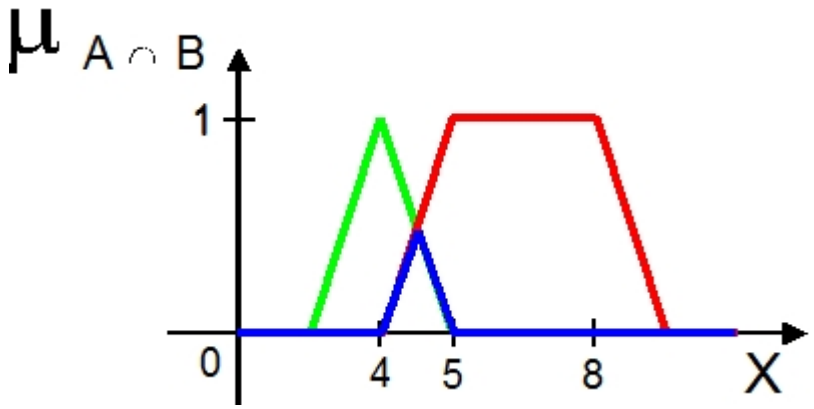
sjednocení A a B: $A \cup B : \mu_{A \cup B} = max\{\mu_A(x); \mu_B(x)\}\ pro\ \forall x \in X$



Obrázek 12: Fuzzy sjednocení

Logický součin

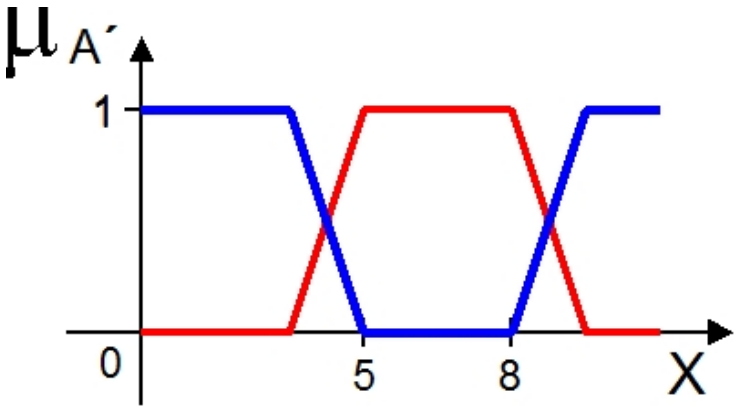
průnik A a B: $A \cap B : \mu_{A \cap B} = min\{\mu_A(x); \mu_B(x)\}\ pro\ \forall x \in X$



Obrázek 13: Fuzzy průnik

komplement (doplněk) A:

$A' : \mu_{A'} = 1 - \mu_A(x)\ pro\ \forall x \in X$



Obrázek 14: Fuzzy doplněk (komplement)

Neplatí zde zákon protikladu a vyloučení třetího ($A' \cup A \neq X, A' \cap A \neq \theta$)

2.5 Fuzzy pravidla

Obecně můžeme rozhodovací pravidla zapsat ve formě *JESTLIŽE…PAK.*

Ve fuzzy logice je podmínka vyjádřena formou implikace dvou fuzzy výroků většinou jako *JESTLIŽE <fuzzy výrok> PAK <fuzzy výrok>*,

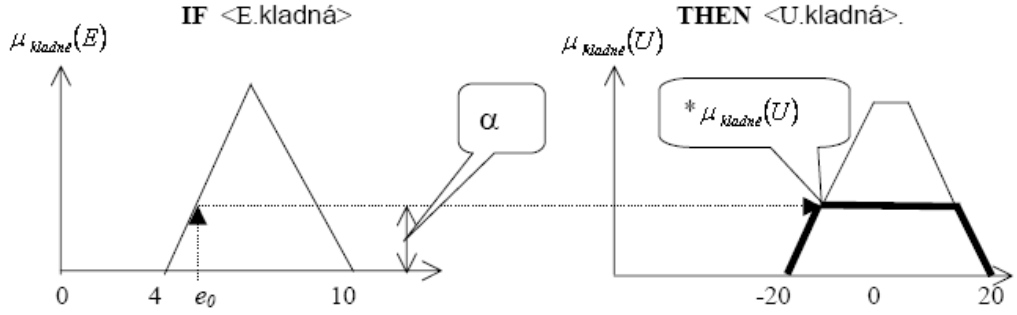
V anglické verzi pak *IF <fuzzy výrok> THEN <fuzzy výrok>*.

Tato podmínka je označována jako *produkční pravidlo*. První fuzzy výroková množina, kterou je často složený výrok, se nazývá *antecedent*, kde jednotlivé části výroku jsou vázány logickými spojkami. Druhý fuzzy výrok je *konsekvent*.

Ilustrační příklad: (Modrlák, 2002)

Uvažujme jednoduchý fuzzy výrok: *IF <E.kladná> THEN <U.kladná>*

V rozhodovacím pravidle je v antecedentu jazyková proměnná *E* (regulační odchylka), jejíž hodnota je „kladná“ a má funkci příslušnosti $\mu_{kladné}(E)$. Konsekvent obsahuje jazykovou proměnnou *U* (akční veličinu) s hodnotou „kladná“, jejíž funkce příslušnosti je $\mu_{kladné}(U)$, viz obrázek 15



Obrázek 15: Mamdaniho metoda na jednosměrné závislosti

Změříme-li ostrou hodnotu regulační odchylky e_0, pak můžeme na obrázku 15 pomocí funkce příslušnosti $\mu_{kladné}(E)$ odečíst stupeň příslušnosti *α*, s jakým naměřená hodnota přísluší k množině hodnot *E.kladná*. Naším úkolem je však nalézt pro naměřenou ostrou hodnotu odpovídající fuzzy množinu konsekventu. Nejčastější postup, jak určit tuto množinu vychází z logického předpokladu, že konsekvent může mít maximálně stupeň příslušnosti jako má antecedent. Stupeň příslušnosti naměřené „ostré“ hodnoty určuje e_0 tedy hladinu, která nám ořízne výstupní fuzzy množinu *U* konsekventu. Funkce příslušnosti konsekventu je pak $*\mu_{kladné}(U)$ (obrys tučně).

Zobecnění tohoto principu pro dvourozměrný případ vyjadřuje Mamdaniho metoda:

IF <x.PS> AND <y.PM> THEN <u.NM>,

tj. jestliže X je kladné střední a zároveň Y je kladné střední pak U je záporné střední

Mamdaniho metoda definuje funkci příslušnosti konsekventu jako $\mu_{MI}(x_1, x_2) = \min\{\mu_A(x_1), \mu_B(x_2)\}$. Minimalizací se vyjadřuje skutečnosti, že důsledek (konsekvent) může mít maximálně stupeň příslušnosti, jako má podmínka (antecedent).

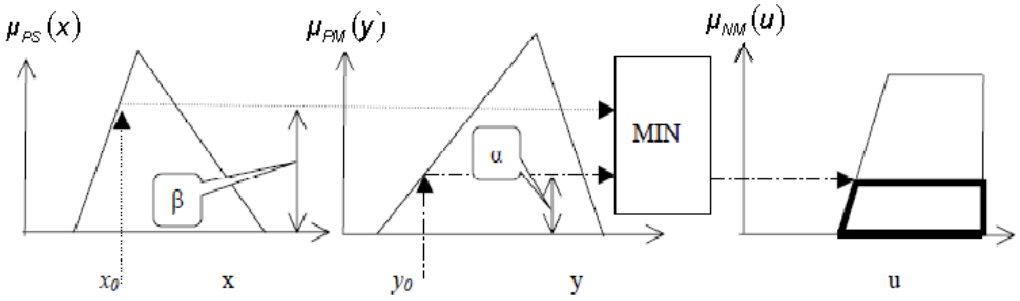
Nalezení výstupní množiny pro jedno pravidlo a dvourozměrnou závislost se liší dle použití operátoru AND nebo OR.

Je-li vazba AND, tak použitím Mamdaniho metody obdržíme funkci příslušnosti konsekventu jako minimum z antecedentu a jeho projekce do osy *μ.*, Což znamená oříznutí funkce příslušnosti konsekventu na hladině α, která odpovídá minimu ze stupňů příslušnosti pro obě vstupní ostré hodnoty x_0 a y_0, viz. obrázek 16 (Modrlák, 2002).

$$\alpha = \mu_{PS}(x) \wedge \mu_{PM}(y) = \min\{\mu_{PS}(x), \mu_{PM}(y)\}$$

Pro funkci příslušnosti konsekventu obdržíme:

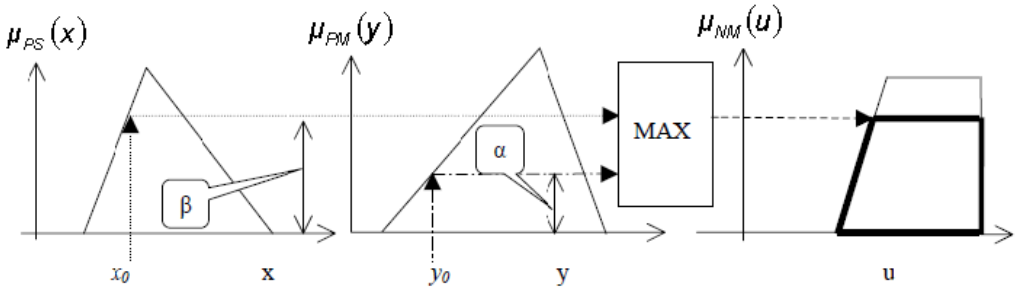
$$^{*}\mu_{NM}(u) = \alpha \wedge \mu_{NM}(u) = \min\{\alpha, \mu_{NM}(u)\}$$



Obrázek 16: Nalezení výstupní množiny pro jedno pravidlo a dvourozměrnou závislost s operátorem AND

Je-li vazba OR, tak vybíráme maximum z odpovídajících funkcí příslušnosti, viz. obrázek 17 (Modrlák, 2002). Oříznutí funkce

příslušnosti konsekventu na hladině β, která odpovídá maximu z obou funkcí příslušnosti vstupních hodnot, tj. *IF <x.PS> OR <y.PM> THEN <u.NM>.*



Obrázek 17: Nalezení výstupní množiny pro jedno pravidlo a dvourozměrnou závislost s operátorem OR

Nalezení výstupní množiny pro dvě rozhodovací pravidla, dvourozměrnou závislost a Mamdaniho metodu je zobrazeno na obrázku 18 (Modrlák, 2002):

IF <x.PS> AND <y.PS> THEN <u.PS> ELSE

IF <x.PM> AND <y.PS> THEN <u.PM>

Pro dvě rozhodovací pravidla jsou nejprve určeny funkce příslušnosti dvou výstupních jazykových proměnných - termů, pro které platí:

$$\alpha_1 = \mu_{PS}(x) \wedge \mu_{PS}(y) = \min\{\mu_{PS}(x), \mu_{PS}(y)\}$$
$$\alpha_2 = \mu_{PM}(x) \wedge \mu_{PS}(y) = \min\{\mu_{PM}(x), \mu_{PS}(y)\}$$

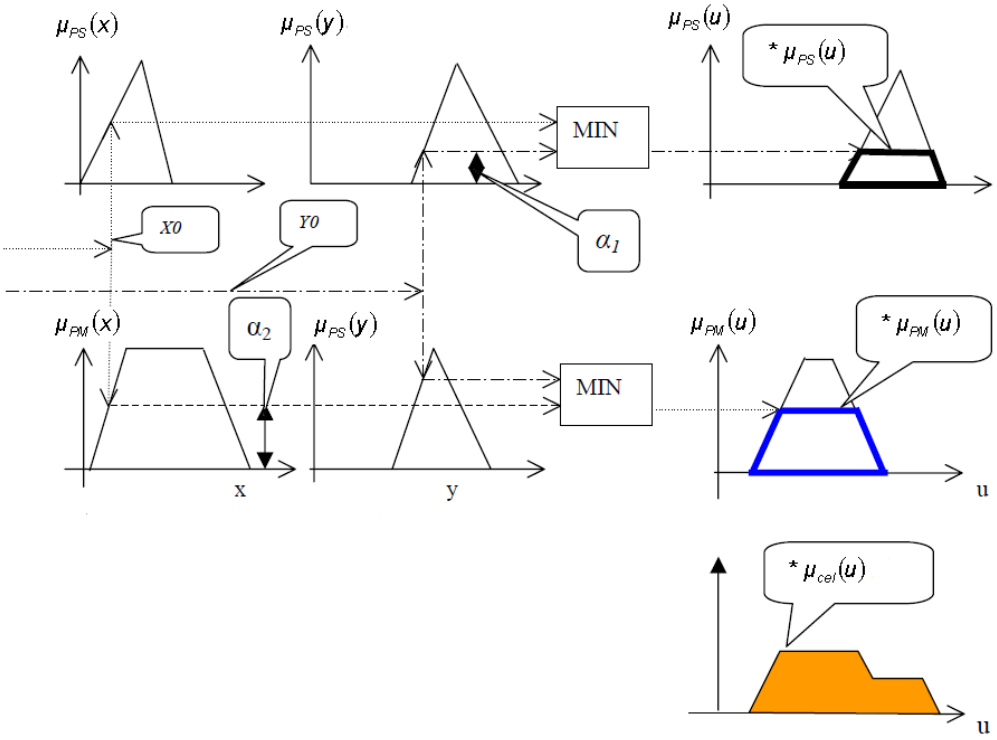
Pro konsekventy obou implikací dostaneme:

$$*\,\mu_{PS}(u) = \alpha_1 \wedge \mu_{PS}(u) = \min\{\alpha_1, \mu_{PS}(u)\}$$
$$*\,\mu_{PM}(u) = \alpha_2 \wedge \mu_{PM}(u) = \min\{\alpha_2, \mu_{PM}(u)\}$$

Konsekventy obou implikací $*\,\mu_{PS}(u)$ a $*\,\mu_{PM}(u)$ určují jejich dílčí podíly na velikosti akční veličiny. Intuitivně se nabízí možnost interpretovat účinek obou dílčích výstupních termů jako jejich logický součet. Pak pro výstupní fuzzy množinu obou účinků dostaneme:

$$*\,\mu_{cel}(u) = \max\{\min\{\alpha_1, \mu_{PS}(u)\}, \min\{\alpha_2, \mu_{PM}(u)\}\}$$

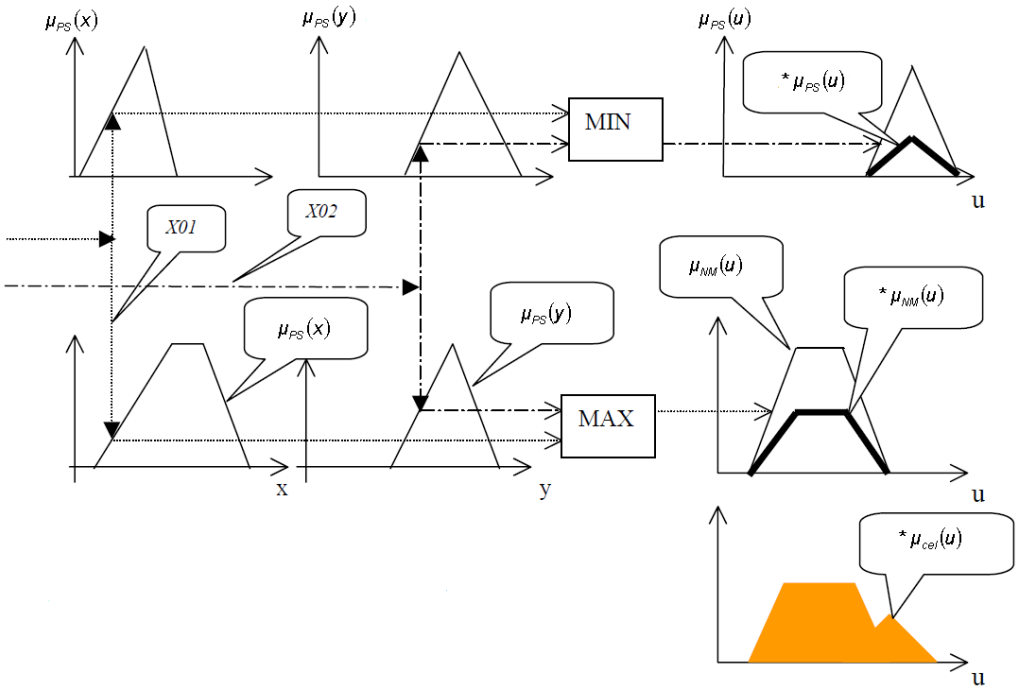
Toto pravidlo je možno rozšířit na libovolný počet rozhodovacích pravidel.



Obrázek18: Nalezení výstupní množiny pro dvě pravidla a dvourozměrnou závislost s použitím Mamdaniho metoda

Použijeme-li Larsenovu metodu, pak výstupní množina pro dvě rozhodovací pravidla nebude oříznuta hladinou α, β, ale vynásobená těmito hladinami jak je vidět na obrázku19. (Modrlák, 2002)

Pro dvourozměrnou funkční závislost jazykových proměnných *X*, *Y* inferenční pravidla tvoří dvojice, které patří do množiny $A \times B$, která je dána kartézským součinem: $P = \{(x, y) / x \in A, y \in B\}$.



Obrázek 19: Nalezení výstupní množiny pro dvě pravidla a dvourozměrnou závislost s použitím Larsenovy metody

2.6 Defuzzifikace

Výsledkem činnosti bloku rozhodovacích pravidel je soubor funkcí příslušnosti pro jednotlivé termy výstupních jazykových proměnných. Funkce příslušnosti výstupní množiny je dána sjednocením oříznutých (Mamdaniho metoda) nebo zmenšených funkcí příslušnosti (Larsenova metoda), viz obrázky 18 a 19. Pro praktické provedení akčních zásahů je třeba přiřadit výstupním jazykovým proměnným ostrou hodnotu akční veličiny v přípustném rozsahu. Tento proces „aproximace neostrých termů“ ostrou hodnotou akční veličiny se nazývá *defuzzifikace*. Existuje celá řada metod defuzzifikace, které vycházejí z empirického ověření až po heuristické přístupy.

Při volbě metody defuzzifikace můžeme zvolit buď metody, které hodnotu akční veličiny určí výpočtem jako nejlepší kompromis (metody těžiště) nebo metody hledající přijatelné řešení (metody nejvýznamnějšího maxima).

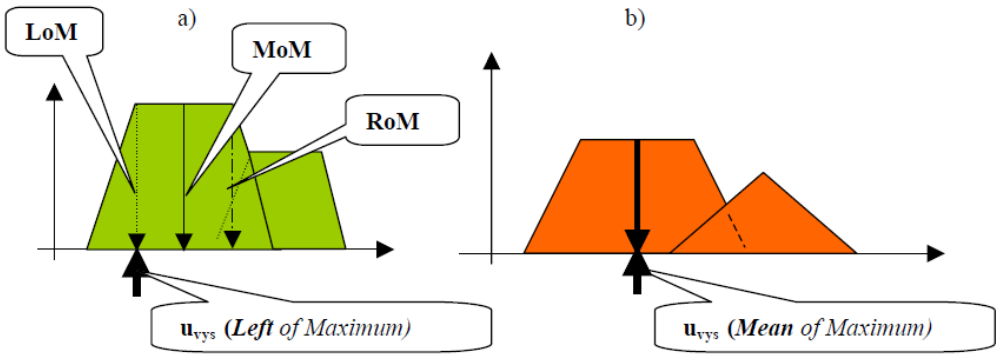
Mean of maximum - metoda nejvýznamnějšího maxima

U metod tohoto typu hledáme tzv. přijatelné řešení, které vyhovuje podmínkám v rozhodovacích pravidlech. Ze všech termů vybereme term s největší hodnotou funkce příslušnosti a nalezneme maximální hodnotu funkce příslušnosti, která pak svým umístěním v závislosti na zvolené metodě určí ostrou hodnotu výstupní veličiny.

Mezi tyto metody patří (Modrlák, 2002):

- Left of Maximum (LoM) : výsledkem je nejvíce vlevo položená hodnota z největší hodnoty funkce příslušnosti
- Mean of Maximum (MoM) : výsledkem je ve středu položená hodnota
- největší hodnoty funkce příslušnosti
- Right of Maximum (RoM) : výsledkem je nejvíce vpravo položená hodnota z největší hodnoty funkce příslušnosti

Na obrázku 20a je příklad určení akční veličiny u_{vys} metodou Left of Maximum a na obrázku 20b metodou Mean of Maximum. Protože se hledá jen maximum, vyznačují se tyto metody vysokou výpočetní rychlostí. Jejich nevýhodou je, že akční veličina se může měnit nespojitě.



Obrázek 20: Funkce příslušnosti pro jednotlivé termy výstupních jazykových proměnných

Metody těžiště

Metody těžiště z průběhů výstupních termů určí ostrou výstupní proměnnou jako jejich těžiště. Existují dva základní přístupy

a) **Center of Maximum** (těžiště singltonů) - nahrazuje funkční závislost každého výstupního termu jeho typickou hodnotou a ostrou výstupní veličinu $u_{výs}$ určí, jako jejich těžiště viz obrázek 21.

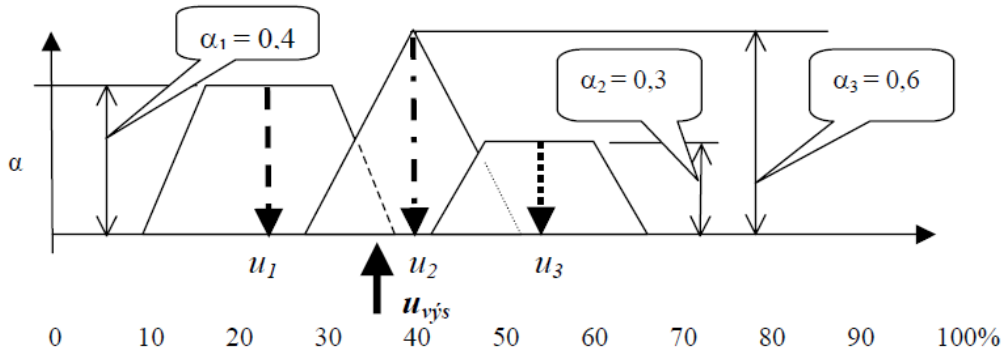
$$u_{výs} = \frac{\sum_{k=1}^{r} \alpha_k * u_k}{\sum_{k=1}^{r} \alpha_k},$$

kde

$u_{výs}$ …je výsledná hodnota výstupní veličiny

α_k …je hodnota příslušnosti k-tého termu

u_k … je souřadnice výstupní veličiny k-tého termu.

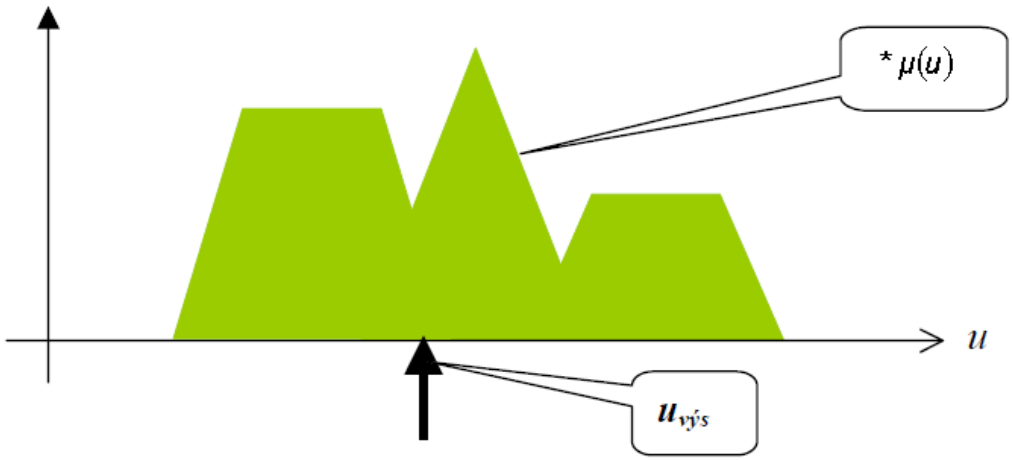


Obrázek 21: Výpočet akční veličiny metodou Center of Maximum

b) **Centre of Gravity** (těžiště plochy) - hledáme těžiště plochy funkce příslušnosti výstupní veličiny. Výslednou hodnotu akční veličiny určíme jako souřadnici těžiště plochy vzniklé sjednocením dílčích ploch, které jsou určeny ohraničením funkcí výstupních termů s nenulovými hodnotami funkce příslušnosti, viz obrázek 22.

$$u_{výs} = \frac{\int * \mu(u) \cdot u \, du}{\int * \mu(u) \, du},$$

kde $* \mu(u)$ je průběh funkce příslušnosti výsledné plochy.



Obrázek 22: Center of Gravity

Pro defuzzifikaci je možno použít ještě celou řadu metod, kterými se již nebudeme podrobněji zabývat. Je zřejmé, že každá metoda poskytuje trochu odlišné defuzzifikované výstupy a proto jejich použití se volí podle druhu aplikace.

Kontrolní otázky:

1. Co je to fuzzy množina?
2. K čemu slouží funkce příslušnosti?
3. Jaké znáte operace s fuzzy množinami?
4. Jaké znáte metody defuzzifikace?

Úkoly k zamyšlení:

Pokuste se zamyslet nad tím, k čemu vlastně slouží fuzzy množiny a proč by si jakýkoli druh modelování nevystačil s klasickými množinami.

Korespondenční úkol:

Popište pomocí fuzzy množiny základní věkové rozdělení, které znají lidé. Jedná se o věkové typy: malí, mladí, střední, starší, staří, nejstarší. Zobrazte tabulkou i graficky

Shrnutí obsahu kapitoly

Fuzzy množina je množina, která kromě úplného nebo žádného členství připouští i členství částečné. To znamená, že prvek patří do množiny s jistou mírou příslušnosti (stupeň příslušnosti). Funkce, která každému prvku universa přiřadí stupeň příslušnosti, se nazývá funkce příslušnosti. V praxi je fuzzy množina F prvků *u* ztotožněná s její funkci příslušnosti $(\mu_F(u))$. Při popisu grafického průběhu funkce příslušnosti $(\mu_F(u))$ vyjdeme nejprve z průběhu funkce příslušnosti $(\mu_F(u))$ klasické množiny A. Praktické použití fuzzy množin vyžaduje analytické vyjádření funkce $(\mu_F(u))$. V praxi používáme nejčastěji jejich aproximaci lomenými přímkami. Dále jste se v této kapitole seznámili se základními operacemi s fuzzy množinami a metodami defuzzifikace.

Pojmy k zapamatování

- fuzzy logika
- fuzzy množina
- funkce příslušnosti
- jazyková (lingvistická) proměnná
- fuzzy pravidla
- defuzzifikace

3 Fuzzy řízení

V této kapitole se dozvíte:

- Jaký je rozdíl mezi PID regulátorem, D regulátorem a I regulátorem.
- Jaké jsou základní principy fuzzy řízení.

Po jejím prostudování byste měli být schopni:

- Vysvětlit princip fuzzy řízení.
- Vysvětlit, jak se fuzzy regulátor liší od klasického regulátoru.

Klíčová slova této kapitoly:

PID regulátor, D regulátor, I regulátor, fuzzy regulátor, fuzzy řízení.

Průvodce studiem

V této kapitole se seznámíte se základními pojmy teorie regulace: PID regulátorem, D regulátorem a I regulátorem. Dále pak se základními principy fuzzy řízení a fuzzy regulace. V závěru bude prakticky představen jednoduchý fuzzy regulátor typu PID.

3.1 Základní pojmy teorie regulace

Základní metodou řízení v teorii regulace je·řízení podle odchylek. Konkrétněji: označme w_t požadovanou hodnotu, tj. hodnotu, kterou má výstup y_t regulovaného (řízeného) procesu dosáhnout. Pak odchylka v okamžiku t je dána vztahem: $e_t = w_t - y_t$

Dále uvažujeme *změnu odchylky*: $\Delta e_t = e_t - e_{t-1}$,

kde *t - 1* označuje předchozí časový vzorek a u_t nám udává akční zásah v okamžiku *t*.

PID regulátor patří mezi spojité regulátory, složený z proporcionální (P), integrační (I) a derivační (D) části, kde proporcionální složka reaguje na okamžitou velikost regulační odchylky, integrační složka reaguje na dobu trvání nenulové regulační odchylky, tj. výstup se mění tak dlouho, dokud regulační odchylka není nulová a derivační složka reaguje na okamžitou rychlost změny regulační odchylky, tj. výstup je nenulový pouze tehdy, když se regulační odchylka mění. V systémech řízení se regulátory řadí před řízenou soustavu. Do regulátoru vstupuje regulační odchylka *e(t)* a vystupuje akční veličina *x(t)*.

P regulátor, je prostý zesilovač. Regulační odchylka je přímo úměrná akční veličině: $x(t) = r_0 e(t)$, kde r_0 je činitel zesílení.

I regulátor je takový regulátor, kdy akční veličina je přímo úměrná integrálu regulační odchylky (2). r_i je zesílení integračního regulátoru.

$$x(t) = r_i \int_0^t e(t)dt + x(0) \qquad (2)$$

D regulátor, je takový regulátor, kdy akční veličina je přímo úměrná derivaci regulační odchylky. Vzhledem k tomu, že "čistá" derivace není technicky realizovatelná, mluvíme o ideálním D regulátoru.

$$x(t) = r_d \frac{de(t)}{dt} \qquad (3)$$

Volba typu regulátoru je v (Kadlec, 2010) uvedena následovně:

P pro méně náročné aplikace, snadno se nastavuje, pracuje s trvalou regulační odchylkou

PI nejběžnější, pro středně náročné aplikace se středně rychlými změnami regulované veličiny a statické soustavy, nastavuje se obtížněji, pracuje bez trvalé regulační odchylky

PID pro nejnáročnější aplikace s rychlými změnami regulované veličiny a astatické soustavy, poměrně obtížně se nastavuje, pracuje bez trvalé regulační odchylky

PD málo používaný, pro aplikace s rychlými změnami regulované veličiny, pracuje s trvalou regulační odchylkou

3.2 Fuzzy regulátory

U fuzzy řízení není základem řízená soustava a její model, ale pozornost je zaměřena na chování člověka (tzv. experta), který umí soustavu řídit na základě pravidel typu „jestliže ukazatel teploty hodně pomalu klesá, stačí o trochu pootočit ventilem doprava“. Fuzzy řízení je vhodný prostředek pro řízení soustav, u nichž neznáme matematický model, ale které dovede člověk řídit. Lze určit hodnotu výstupu, aniž známe vzorce mezi vstupem a výstupem. Každé fuzzy pravidlo přispívá jen částí ke konečnému výsledku. Takový postup je odolnější proti chybám než obvyklý algoritmus. K nevýhodám fuzzy regulátorů patří složitý řídící algoritmus, možnost oscilací v ustáleném stavu a zatím malé zkušenosti s jeho návrhem. Fuzzy regulátor přiřazuje zvoleným vstupním veličinám jazykovou hodnotu. To se provede nejlépe pomocí funkcí příslušnosti, jež bývají obvykle voleny ve tvaru lichoběžníka (obrázek 7) či trojúhelníka (obrázek 6). Tato etapa je označována jako fuzzyfikace. V dalším kroku určí fuzzy regulátor na základě znalostí experta slovní hodnoty akčních veličin (např. regulační odchylka bude záporná malá, když....). V závěrečném kroku převede slovní vyjádření na konkrétní číselné hodnoty veličin – tzv. defuzzifikaci (Janošek 2012).

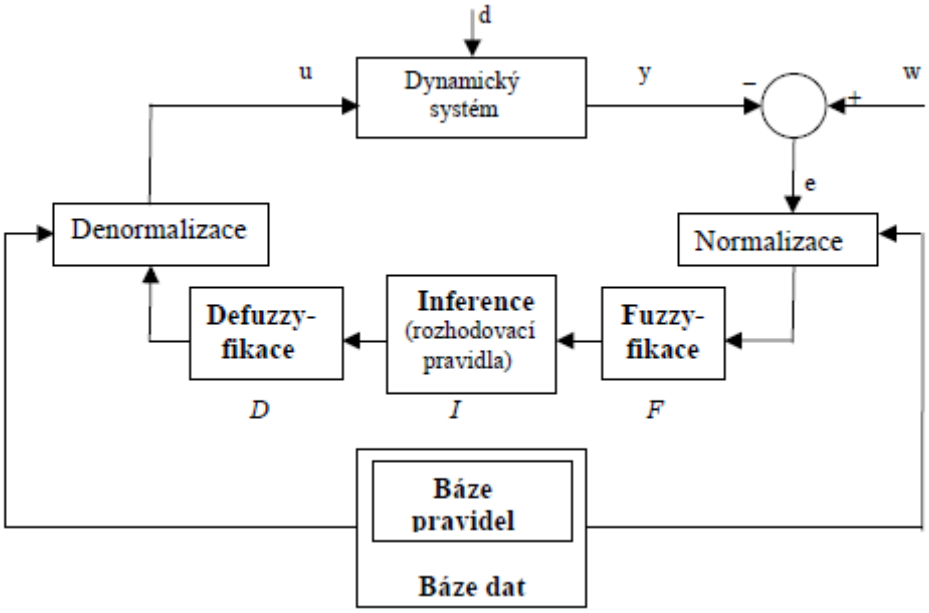
Charakteristickým znakem fuzzy řízení je možnost bezprostředního použití empirických znalostí člověka - operátora o řízeném procesu, které označujeme jako bázi znalostí. Bázi znalostí tvoří (Modrlák, 2002):

a) *informace o stacionární stavech*, intervalech, ve kterých se pohybují hodnoty vstupních a výstupních veličin, jejich mezní hodnoty, atd. Rozšíříme-li tato data o funkce příslušnosti všech

vstupních a výstupních fuzzy množin, pak všechny tyto informace o procesu se v bázi znalostí označují jako báze dat.

b) *kvantitativně formulované zkušenosti* včetně slovně definované strategie řízení, pomocí kterých je možno realizovat řízení, tj. generovat akční veličinu. Takto zkušeností získané strategie řízení označujeme jako bázi pravidel.

Struktura fuzzy regulátoru je na obrázku 22 a jeho ústřední člen tvoří tři základní bloky: fuzzyfikace *F*, inference *I* a blok defuzzifikace *D*. V bloku fuzzyfikace se převádí ostrá data, která jsou naměřena nebo zadána, na fuzzy data. Bloku fuzzyfikace může předcházet blok normalizace, kde se fyzikální hodnoty naměřených či zadaných hodnot převedou na normalizovanou množinu - univerzum. V bloku inference, který tvoří ústřední část regulátoru, se realizuje inferenční mechanismus z rozhodovacích pravidel, pomocí kterého získáváme ze vstupních fuzzy množin výstupní množiny. Blok defuzzifikace umožňuje přiřadit výstupní fuzzy množině určitou ostrou výstupní veličinu. Za blokem defuzzifikace může následovat blok denormalizace, kde se provede denormalizace výstupní veličiny - přepočet na fyzikální výstupní veličiny.



Obrázek 22: Struktura fuzzy regulátoru

3.3 Jednoduchý fuzzy regulátor typu PID

Ilustrační příklad: (Modrlák, 2002)

Výstup číslicového PI regulátoru v přírůstkovém tvaru, který zajišťuje nulovou regulační odchylku, je (4)

$$u(k) = u(k-1) + \Delta u(k);$$
$$\Delta u(k) = q_0 e(k) + q_1 e(k-1) \qquad (4)$$

Výstup číslicového PD regulátoru, který ovšem nezajišťuje nulovou regulační odchylku, je ve tvaru (5):

$$u(k) = K_p e(k) + K_D \Delta e(k) \qquad (5)$$

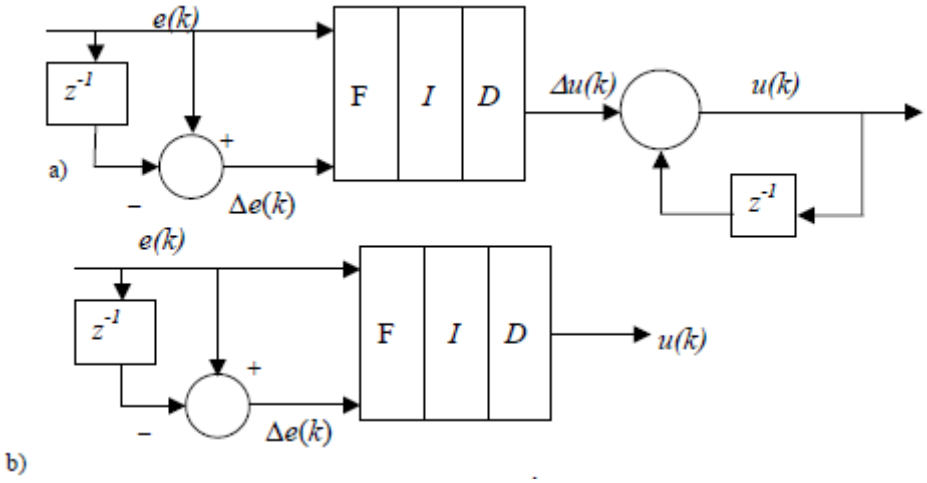
Hovoříme-li o jednoduchém fuzzy regulátoru a chceme-li ho porovnávat s PI regulátorem nebo s PD regulátorem, pak vstupem těchto regulátorů je $e(k)$ a $\Delta e(k)$. Výstup je pak nelineární funkcí, která závisí na fuzzyfikaci, inferenci a defuzzifikaci. Takže pro fuzzy regulátor typu PI bude platit (6)

$$u(k) = u(k-1) + \Delta u(k);$$
$$\Delta u(k) = F_{PI}(e(k)). \qquad (6)$$

Fuzzy regulátor typu PD dostáváme jako nelineární funkci ve tvaru

$$\Delta u(k) = F_{PD}(e(k), \Delta e(k)). \qquad (7)$$

Struktura fuzzy regulátoru typu PI a PD je na obrázku 23 a, b.



Obrázek 23: Struktura jednoduchých fuzzy regulátorů typu PI a PD

Jednoduchý fuzzy regulátor s vlastnostmi PI - PD regulátoru vytvoříme nejjednodušším způsobem tak, že tyto dva regulátory paralelně propojíme viz (Vysoký 1997) (Pivoňka 2000)

3.4 Tvorba báze pravidel

Bázi pravidel je možno vytvořit:

1. na základě empirických znalostí obsluhy
2. na základě obecně platných metapravidel.

Pro jednoduchý fuzzy regulátor typu PI, PD je možno odvodit bázi pravidel pomocí tří základních metapravidel, která uvedeme:

- MP1: Jestliže regulační odchylka *e(k)* a její změna Δ*e(k)* je nulová nebo blízká nule, pak by měl být přírůstek akční veličiny Δ*u(k)* – akční zásah nulový nebo blízký nule.

- MP2: Jestliže regulační odchylka *e(k)* klesá k nule nebo se blíží nule s dostačující rychlostí, pak je vhodné také neměnit akční veličinu.

- MP3: Jestliže se regulační odchylka *e(k)* nekoriguje sama, potom je třeba akční veličinu změnit a akční zásah Δ*u(k)* bude nenulový. Jeho velikost a znaménko závisí na znaménku a velikosti regulační odchylky *e(k)* a její změny Δ*e(k)*.

Ilustrační příklad: (Modrlák, 2002)

Podle těchto metapravidel byla pro jednoduchý fuzzy regulátor sestavena báze pravidel, která je uvedena tabulce 2:

Regulační odchylka	*e*	*{ NL, NM, NS, Z, PS, PM, PB}*
Změna regulační odchylky	*Δe*	*{ NL, NM, NS, Z, PS, PM, PB}*
Akční zásah	*Δu*	*{ NL, NM, NS, Z, PS, PM, PB}*

kde

NL - hodnota velká záporná
NM - hodnota střední záporná
NS - hodnota malá záporná
Z - hodnota nulová
PS - hodnota malá kladná
PM - hodnota střední kladná
PL - hodnota velká kladná

e \ *Δe*	NL	NM	NS	Z	PS	PM	PL
NL	NL	NL	NL	NL	NM	NS	Z
NM	NL	NL	NL	NM	NS	Z	PS
NS	NL	NL	NM	NS	Z	PS	PM
Z	NL	NM	NS	Z	PS	PM	PL
PS	NM	NS	Z	PS	PM	PL	PL
PM	NS	Z	PS	PM	PL	PL	PL
PL	Z	PS	PM	PL	PL	PL	PL

Tabulka 2: Báze pravidel pro jednoduchý fuzzy regulátor

V bázi pravidel je možno rozlišit pět skupin pravidel.

Skupina 1

Tato skupina pravidel se použije tehdy, jestliže regulační odchylka *e(k)* a její změna Δ*e(k)* je nulová nebo blízká nule. Znamená to, že regulovaná soustava je v ustáleném stavu nebo v jeho blízkosti.

Akční veličina se nemá měnit, čili změna akční veličiny je nulová nebo blízká nule.

Skupina 2.

Pro aplikaci pravidel této skupiny platí, že regulační odchylka *e(k)* je záporná (velká nebo střední) a její změna Δ*e(k)* je kladná nebo blízká nule. Důsledkem toho je, že regulační odchylka

e(k) se **zmenšuje nebo se nemění**.

Akční zásah má zrychlit nebo zpomalit přibližování k ustálené hodnotě.

Skupina 3.

Pro tuto skupinu platí, že regulační odchylka *e(k)* je kladná (blízká nule, střední, veliká). Změna Δ*e(k)* je kladná velká nebo střední, což znamená, že regulovaná veličina **se bude vzdalovat** od žádané hodnoty - ustáleného stavu.

Kladnou změnou akční veličiny Δu(k) je třeba zajistit přibližování k ustálenému stavu.

Skupina 4.

Pro aplikaci pravidel této skupiny je charakteristické, že regulační odchylka *e(k)* je kladná (velká nebo střední) a její změna Δ*e(k)* je záporná nebo nulová. To znamená, že regulační odchylka *e(k) se* **zmenšuje nebo se nemění**. (Porovnej se skupinou 2)

Akční zásah má zrychlit nebo zpomalit přibližování k ustálené hodnotě.

Skupina 5.

Pro tuto skupinu platí, že regulační odchylka *e(k)* je záporná (blízká nule, střední, veliká). Změna Δ*e(k)* je **záporná velká nebo střední**. To znamená, že regulovaná veličina se bude **vzdalovat** od žádané hodnoty - ustáleného stavu. (Porovnej se skupinou 3)

Zápornou změnou akční veličiny Δu(k) je třeba zajistit přibližování k ustálenému stavu.

Velká většina jednoduchých fuzzy regulátorů má bázi pravidel založenou na použití zadaných pravidel. Báze pravidel lze snadno modifikovat pro jiný počet termů regulační odchylky a její změny. Odezvy regulačních obvodů s fuzzy regulátorem závisí na bázi rozhodovacích pravidel a na bázi dat. Součástí fuzzy regulátoru je získání či vytvoření báze rozhodovacích pravidel, zadání funkcí příslušnosti pro jednotlivé vstupní a výstupní proměnné, včetně volby metod fuzzyfikace a defuzzifikace. Vlastní implementace těchto bázických znalostí pro daný řídící systém nebo produkt se realizuje softwarově.

Kontrolní otázky:

1. Jaký je rozdíl mezi PID regulátorem, D regulátorem a I regulátorem.
2. Jaké jsou základní principy fuzzy řízení.

Shrnutí obsahu kapitoly

V této kapitole se seznámíte se základními pojmy teorie regulace: PID regulátorem, D regulátorem a I regulátorem. Dále pak se základními principy fuzzy řízení a fuzzy regulace. V závěru bude prakticky představen jednoduchý fuzzy regulátor typu PID.

Pojmy k zapamatování

- PID regulátor
- D regulátor
- I regulátor
- fuzzy regulátor
- fuzzy řízení

4 Úvod do problematiky strojového učení

V této kapitole se dozvíte:

- Co je to strojové učení?
- Jaké jsou interakce mezi učením, výkonem, znalostmi a prostředím.
- Jaké jsou základní pojmy strojového učení.

Po jejím prostudování byste měli být schopni:

- Charakterizovat strojové učení.
- Vysvětlit základní pojmy strojového učení.

Klíčová slova této kapitoly:

Strojové učení, prostor řešení, trénovací množina, nulová hypotéza, intenzionální definice, extenzionální definice.

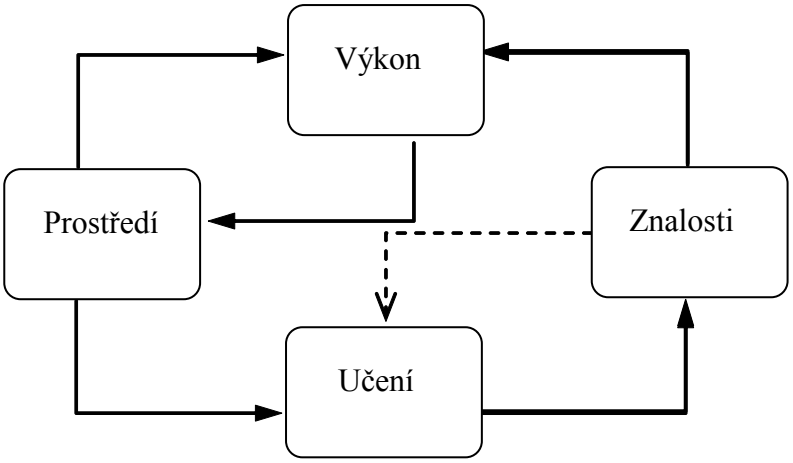
Průvodce studiem

Strojové učení je jádrem současné umělé inteligence a je svázané s oblastmi statistiky, data-miningem a s přírodou inspirované metodami modelování, jejichž představitelem jsou zejména umělé neuronové sítě. Učením v daném kontextu rozumíme takovou změnu vnitřního stavu systému, která zefektivní schopnost přizpůsobení se změnám okolního prostředí. Strojové učení je hlavně věda o algoritmech.

4.1 Co je to strojové učení

Strojové učení je podoblastí umělé inteligence, zabývající se algoritmy a technikami, které umožňují počítačovému systému 'učit se'. Učením v daném kontextu rozumíme takovou změnu vnitřního stavu systému, která zefektivní schopnost přizpůsobení se změnám okolního prostředí.

Vzájemné interakce mezi učením, prostředím, výkonem a znalostmi ilustruje obrázek 24. Prostředí logicky reprezentuje zkušenost. Přerušovaná čára představuje interakci, která se nemusí při učení vyskytovat. Avšak učení s touto interakcí je optimální. Jde o případ, kdy naučené znalosti ovlivňují zpětně proces učení. Učení samozřejmě ovlivňuje znalosti, protože je vytváří. Naučené znalosti mohou ovlivnit výkon. Podobně ovlivňují výkon zkušenosti z prostředí. Je žádoucí, aby zvyšovaly výkon. Prostředí samozřejmě ovlivňuje proces učení.



Obrázek 24: Interakce mezi učením, výkonem, znalostmi a prostředím

Jedním z cílů strojového učení je modelování mechanismů, které tvoří základ lidského učení. V tomto rámci jsou vyvíjeny učící algoritmy, které jsou konzistentní s lidským způsobem poznávání a architekturou, kterou člověk používá pro ukládání znalostí a vztahů mezi nimi. Tyto algoritmy jsou navrhovány tak, aby vysvětlily specifika pozorovaných návyků učení. Některé metody vysvětlují tyto návyky na kvalitativní úrovni, zatímco jiné pracují v kvantitativní rovině, např. nakládají s rozsahem chyby a reakčním časem lidských subjektů. Protože tyto metody vytvářejí predikce návyků učení, jsou potenciálně použitelné v návrhu instruktivních materiálů pro vzdělávání lidských subjektů.

Ke studiu strojového učení lze přistupovat i empiricky. Empirický přístup představuje snahu objevit obecné principy, které se vztahují na charakteristiky adaptačních algoritmů a obecné zásady domén, v jejichž rámci tyto algoritmy operují. Experimentální studie ze strojového učení

vyprodukovali velké množství empirických dat o alternativních metodách, které poukazovaly:

- na použitelnost jednotlivých tříd algoritmů pro řešení různých typů úloh;
- na slabosti zkoumaných algoritmů;
- na možnosti jejich vylepšení;
- na zdroje obtížnosti úkolů.

Se strojovým učením lze nakládat také jako s matematickou oblastí s cílem formulovat a dokazovat teorémy o zpracovatelnosti celých tříd problémů učení a algoritmů navržených k řešení těchto problémů.

Ke strojovému učení lze přistupovat i aplikačně a zaměřit se zejména na:

- formulaci problému v termínech strojového učení,
- návrh reprezentace trénovacích příkladů, jakož i naučených znalostí,
- shromáždění trénovacích příkladů,
- generování báze znalostí použitím strojového učení.

Co spojovalo všechny tyto přístupy, byl zájem o rozvoj, porozumění a ohodnocení studujícím algoritmů. ***Strojové učení je hlavně věda o algoritmech.*** Algoritmy strojového učení lze podle způsobu učení rozdělit do následujících kategorii (Mařík, 2003):

- učení s učitelem
- učení bez učitele
- kombinace učení s učitelem a bez učitele
- zpětnovazebné učení (učení posilováním)

4.2 Základní pojmy strojového učení

Strojové učení se uskutečňuje nad prostorem pojmů. Prostor pojmů se někdy nazývá také *prostorem řešení* (pojem představuje řešení úkolu: naučit se pojem). Tento prostor je zpravidla uspořádaný a učení spočívá v jeho procházení. Pojmy, které se nacházejí v daném prostoru na nejnižší úrovni, obecně představují jednotlivá pozorování, na jejichž základě proběhne proces učení. Tímto pozorováním obvykle míníme typické příklady, jež jsou charakteristické tím, že každý atribut příkladu je dán svou konkrétní hodnotou. Typické příklady jsou vstupy učících algoritmů. Prostor typických příkladů se někdy nazývá i *problémovým prostorem* nebo *trénovací množinou*. Všechny ostatní pojmy uspořádaného prostoru představují možná řešení. Tedy mohou představovat pojem, který je výsledkem učení. Na nejvyšší úrovni obecně se nachází takzvaná *nulová hypotéza*, která pokrývá všechny pojmy prostoru a zároveň neříká nic konkrétního. Přidáním dalších typických příkladů se může prostor pojmů zvětšovat nejen horizontálně ale i vertikálně.

Učící algoritmy mají na vstupu zpravidla množinu typických příkladů a na výstupu definici učeného pojmu. Typický příklad je reprezentován množinou *n* atributů, které představují vlastnosti daného příkladu. Poslední *n*-tý atribut může reprezentovat třídu, do níž je typický příklad zařazen, pokud je taková informace součástí daného pozorování (kontrolované učení). V takovém případě má typický příklad datovou část a rozhodovací část. Datová část představuje soubor *n-1* hodnot atributů. Rozhodovací část představuje informace o třídě, do níž je příklad zařazen.

Podle toho, jaké hodnoty atributy nabývají, rozlišujeme různé typy atributů:

- *binární atributy*, v nichž každá hodnota může specifikovat buď přítomnost, nebo absenci dané vlastnosti. Obecněji řečeno, nabývají hodnoty z dvouprvkové množiny, např. {ANO / NE}.

- *nominální atributy* připouštějí více než dvě hodnoty atributu. Nabývají hodnoty z konečné neuspořádané množiny, např.: {země, oheň, voda, vzduch}. Je možná transformace nominální reprezentace na binární formalismus, když se všechny hodnoty nominálního atributu sdruží do dvou skupin, které budou vystupovat jako dvě hodnoty binárního atributu.
- *numerické atributy* mohou nabývat reálné či celočíselné uspořádány hodnoty atributů. Pokud máme daných *k* numerických atributů, trénovací příklad může reprezentovat bod v *k*-dimenzionálním prostoru, přičemž jednotlivé atributy definují souřadné osy. Někdy se takový prostor nazývá problémovým prostorem.
- *ordinální atributy* - nabývají hodnoty z konečné uspořádané množiny např. {slabá, mírná, silná}. Často uváděným příkladem je uspořádání barev podle vlnových délek.
- *hierarchické atributy* – hodnoty atributů nejsou nezávislé, ale jsou součástí nějaké hierarchické struktury. Jednotlivé hodnoty nemusí být disjunktní, např. {Ovál, kruh, elipsa, mnohoúhelník, trojúhelník, čtyřúhelník, čtverec, obdélník, pětiúhelník, ...}.

Množina typických příkladů je zpravidla udávána ve formě tabulky, kde řádky představují jednotlivá pozorování resp. typické příklady a sloupce jednotlivé atributy s jejich hodnotami, poslední sloupec obsahuje třídu. Cílem *induktivního učení* je z takové tabulky indukovat popis hledaného pojmu. Rozhodovací procedura pak pro datovou část příkladu najde příslušnou rozhodovací část, tedy třídu. Pokud jsou známy všechny typické příklady, pak je rozhodovací procedura jednoznačná. To je ale v praxi velmi vzácné. Pokud nejsou známy všechny typické příklady, obvykle se množina typických příkladů rozdělí na množinu trénovacích příkladů a množinu testovacích příkladů. *Trénovací množina* slouží k naučení pojmu. *Testovací množina* se používá k ověření stupně

přesnosti klasifikace. Pojem trénovací příklad je ve strojovém učení běžnější než pojem typický příklad.

Na výstupu učícího algoritmu je naučený pojem. Hledáme zpravidla takový popis pojmu, který je konzistentní se zadanými údaji. Pojem může být reprezentován *extenzionálně* nebo *intenzionálně*. Extenzionální definice představuje naivní přístup k reprezentaci pojmu, který jednoduše považuje všechny trénovací příklady za pojmy. Tedy extenzionální definice je reprezentována vyjmenováním všech příkladů, které pojem reprezentují. Taková definice pouze sumarizuje předchozí zkušenosti. Není schopna vykonávat predikci zařazení neznámých trénovacích příkladů, které mohou pojem také reprezentovat. Intenzionálna definice je kompaktnější a obecnější, protože se získává generalizací množiny trénovacích příkladů. Je schopna predikovat zařazení nových trénovacích příkladů.

Mějme *interpret*, jehož úkolem je rozhodnout, který z pojmů pokrývá daný příklad a je potřebný k tomu, aby interpretoval výsledek učení reprezentovaný intenzionálné. Nalezena intenzionální definice pojmu pak může být použita pro klasifikaci nových příkladů a to tak, že se definice porovná s definicí příkladu. Pokud příklad vyhovuje všem podmínkám v definici, je označen za pozitivní příklad daného výrazu. Jinak je označen za negativní příklad pojmu. Extenzionální reprezentace se dá přímo použít, protože se jednoduše porovná s definicí nového příkladu. Interpret může být použit k transformaci intenzionální definice na extenzionální definici jednoduše tím, že je aplikován na všechny možné příklady, tedy:
Intenzionální definice + *Interpreter* = *Extenzionální definice*

Naučené pojmy mohou být různé *interpretovány*. Každá interpretace má vlastní reprezentaci pojmu. Nejčastěji se vyskytují následující interpretace pojmů:

- *Logická interpretace*. Pojmy jsou reprezentovány logickými kombinacemi hodnot atributů. Proces pokrývání je úplný a musí platit buď všechny podmínky, nebo žádná.
- *Prahová interpretace*. Interpreter obstarává částečné pokrývání, což vyžaduje specifikaci nějaké prahové hodnoty, která určuje akceptovatelný stupeň shody.
- *Soutěžní interpretace*. Interpreter zajišťuje částečné pokrývání. Namísto určení prahové hodnoty, vypočítá stupeň pokrývání pro jednotlivé alternativní soutěžící pojmy a vybírá nejlepšího kandidáta.

Dalšími činiteli, jež ovlivňují učení, jsou *aspekty okolí*. Nejběžnější aspekt okolí je komplexnost cílových znalostí, které mají být získány. Tato komplexnost závisí na:

- *komplexnosti popisu pojmu*. Např. pojem, který obsahuje mnoho atributů nebo podmínek, je těžší na naučení než ten, který obsahuje méně atributů, resp. podmínek. Tento aspekt zohledňuje generování rozhodovacího stromu pomocí minimální délky popisu.
- *struktury problémového prostoru*. Např., Pokud pojem obsahuje mnoho irelevantních atributů, učící systém může mít potíže s jejich odlišením od relevantních atributů.
- *výskytu šumu*. V kontrolovaném učení existují dva druhy šumu. Klasifikační šum, tj. šum při určování klasifikační třídy. Představuje chybu zpětné vazby. Druhý typ je tzv. atributový šum, který zahrnuje chybu v popisu atributů trénovacího příkladu. Většinou se jedná o to, že u konkrétních trénovacích příkladů chybí některé hodnoty atributů. Chybějící hodnoty atributů mohou být nahrazeny (např. nejčastěji vyskytovanou hodnotou v rámci daného atributu) nebo mohou být vyloučeny trénovací příklady s touto chybou z trénovací množiny. Větší množství šumu má tendenci dělat učení složitějším.

Kontrolní otázky:

1. Co je to strojové učení?
2. Jaké jsou interakce mezi učením, výkonem, znalostmi a prostředím?
3. Vysvětlete základní pojmy strojového učení.

Shrnutí obsahu kapitoly

V této kapitole jsme se seznámili se základními principy strojového učení. Důraz zde byl kladen vysvětlení základních pojmů strojového učení: prostor řešení, trénovací množina, nulová hypotéza, intenzionální definice pojmu, extenzionální definice pojmu apod.

Pojmy k zapamatování

- strojové učení;
- trénovací množina;
- nulová hypotéza;
- intenzionální definice;
- extenzionální definice.

5 Rozhodovací stromy

V této kapitole se dozvíte:

- Co je to rozhodovací strom?
- Na jakém principu pracuje metoda TDIDT?
- Co jsou to rozhodovací pravidla?

Po jejím prostudování byste měli být schopni:

- Popsat rozhodovací strom.
- Vysvětlit princip metody TDIDT.

Klíčová slova této kapitoly:

Rozhodovací strom, TDIDT (Top Down Induction Of Decision Trees), rozhodovací pravidla.

Průvodce studiem

Při strojovém učení používáme především rozhodovací stromy jako způsobu reprezentování znalostí, což je dobře známo i z řady jiných oblastí. Vzpomeňme jen nejrůznějších „klíčů k určování“ různých živočichů nebo rostlin známých z biologie. Indukce rozhodovacích stromů patří k nejznámějším algoritmům z oblasti symbolických metod strojového učení. Právě těmto tématům se bude věnovat tato kapitola.

5.1 Reprezentace rozhodovacích stromů

Při strojovém učení používáme především rozhodovací stromy jako způsobu reprezentování znalostí. Hlavní důvod spočívá v jejich přehlednosti a snadné interpretovatelnosti, která umožňuje uživatelům rychle a lehce vyhodnocovat získané výsledky, identifikovat klíčové položky a vyhledávat zajímavé segmenty případů. Cílem použití

rozhodovacích stromů je identifikovat objekty, popsané různými atributy, do tříd. Představit si je můžeme jako řádky v tabulce, kde jednotlivé sloupce jsou atributy (barva očí, délka ocasu apod.). Jelikož se jedná o strom, algoritmus je velmi rychlý. *Rozhodovací strom* vytvoříme z množiny daných objektů, které musí někdo (učitel, jiný algoritmus) zařadit do skupin (skupina se obvykle označuje jako závislý atribut a zapisuje se do tabulky do posledního sloupce). Jedná se tedy o *učení s učitelem*.

Každý uzel stromu představuje jednu (vybranou) vlastnost objektů, z tohoto uzlu vede konečný počet hran. Proto je nutné vlastnosti nejdříve diskretizovat (např. z reálných čísel do konečného počtu intervalů), protože strom musí co nejlépe objekty od sebe odlišit; zejména pro kořenový uzel se vybírá takový atribut, který objekty od sebe odliší maximálně. Vytváření stromů je dobře algoritmicky popsáno. Rozhodovací stromy dělí prostor atributu na (mnohorozměrné) hranoly rovnoběžné s osami souřadné soustavy.

Rozhodovací strom představuje reprezentaci pojmu, která je velmi blízká reprezentaci pojmu produkčním pravidlem. Konečně je velmi snadné transformovat rozhodovací strom na produkční pravidla. Rozhodovací strom má výhodu v tom, že velmi názorně ilustruje proces učení. Proto je tato reprezentace vhodná na řešení úkolů, které vyžadují spolupráci s odborníky jiných vědních oborů a tedy ve strojovém učení. Používají se také v znalostních systémech na automatické generování bází znalostí, v objevování znalostí a dalších oblastech.

Rozhodovací strom představuje reprezentaci rozhodovací procedury pro klasifikaci příkladů do příslušných tříd. Je to grafová struktura ve formě stromu obsahující kořenový uzel, mezilehlé a listové uzly. Uzly rozhodovacího stromu reprezentují třídy a jeho hrany reprezentují hodnoty testovacího atributu. Z každého uzlu vychází tolik hran, kolik hodnot má testovací atribut v daném uzlu.

Na začátku generování rozhodovacího stromu se nacházejí všechny trénovací příklady, (celá trénovací množina) v kořenovém uzlu. Pro každý nelistový uzel (kořenový a mezilehlá uzly) se pak vybere nejvhodnější atribut. Říkáme, že každý nelistový uzel reprezentuje specifický test. V rámci mezilehlých uzlů se trénovací množina člení na podmnožiny resp. strom se větví do podstromu pro každý výsledek specifického testu, tj. pro každou hodnotu zvoleného atributu. Obecně test může být i něco složitější. Každý nelistový uzel rozhodovacího stromu tedy obsahuje testovací podmínku, která rozděluje prostor příkladů podle možného výsledku testu a reprezentuje klasifikační třídu. Klasifikace nového příkladu se pak uskutečňuje takto: Pokud nový příklad splňuje všechny testy na cestě ve stromě od kořene po listový uzel (podmínky podmínkové části pravidla), pak je daný příklad zařazen do třídy v listové uzlu (v závěru pravidla).

Z výše uvedeného vyplývá, že pro definici rozhodovacího stromu je třeba určit:

- pro *listový uzel* název třídy, do níž budou klasifikovány všechny typické příklady, asociované s daným listovým uzlem
- pro *nelistový uzel* testovací atribut, podle kterého se rozhoduje větvení na jednodušší podstromy pro každou hodnotu zvoleného zkušebního atributu.

5.2 Induktivní tvorba rozhodovacích stromů

Rozhodovací stromy jsou jednou z data-miningových technik. Výhody použití rozhodovacích stromů při strojovém učení je následující (Mařík, 2003):

- příklady jsou reprezentovány hodnotami atributu;
- úkolem je klasifikovat příklady do konečného (malého) počtu tříd;
- hledaný popis konceptu muže být tvořen disjunkcemi;

- trénovací data mohou být zatížena šumem;
- trénovací data mohou obsahovat chybějící hodnoty.

Metoda TDIDT (*Top down induction of decision trees*), tj. metoda specializace v prostoru hypotéz – stromu (postup shora dolu, počínaje prázdným stromem) v úlohách klasifikace objektu do tříd. Použití rozhodovacích stromů pro klasifikaci odpovídá analogii s klíči k určování rostlin nebo živočichů. Od kořene stromu se na základě odpovědí na otázky (umístěné v nelistových uzlech) postupuje příslušnou větví stále hlouběji, až do listového uzlu, který odpovídá zařazení příkladu do třídy. Trénovací data se zde postupně rozdělují na menší a menší podmnožiny tak, aby v těchto podmnožinách převládaly příklady jedné třídy. Cílem je nalézt nějaký strom konsistentní s trénovacími daty. Přitom se dává přednost menším stromům.

TDIDT algoritmus:

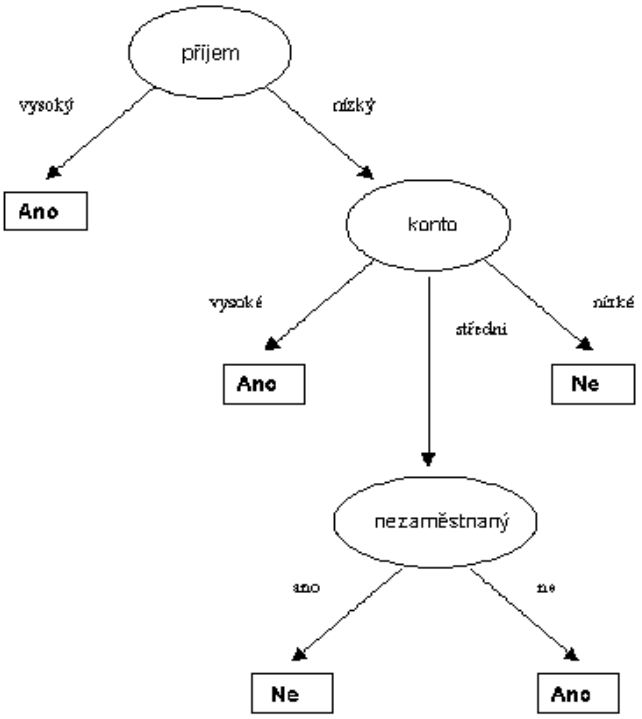
1 Vezmi jeden atribut jako kořen dílčího stromu.

2 Rozděl data na podmnožiny podle hodnot tohoto atributu.

3 Nepatří-li všechna data v podmnožině do téže třídy, pro tuto podmnožinu opakuj postup od bodu 1.

Ilustační příklad (Berka, 2010)**:**

Úplný rozhodovací strom je pak zobrazen na obrázku 25.



Obrázek 25: Úplný rozhodovací strom

Data pro tvorbu stromu jsou uvedena v tabulce 3:

klient	**příjem**	**konto**	**pohlaví**	**nezaměstnaný**	**úvěr**
k1	vysoký	vysoké	žena	ne	ano
k2	vysoký	vysoké	muž	ne	ano
k3	nízký	nízké	muž	ne	ne
k4	nízký	vysoké	žena	ano	ano
k5	nízký	vysoké	muž	ano	ano
k6	nízký	nízké	žena	ano	ne
k7	vysoký	nízké	muž	ne	ano
k8	vysoký	nízké	žena	ano	ano
k9	nízký	střední	muž	ano	ne
k10	vysoký	střední	žena	ne	ano
k11	nízký	střední	žena	ano	ne
k12	nízký	střední	muž	ne	ano

Tabulka 3: Data pro tvorbu stromu

Rozhodovací pravidla *If-then* konstrukce se používají i v běžné mluvě. Není tedy divu, že pravidla s touto syntaxí patří, vedle stromů k nejčastěji používaným prostředkům pro reprezentaci znalostí, ať už získaných od expertů, nebo vytvořených automatizovaně z dat.

Jedním z nejznámějších algoritmů pro tvorbu pravidel je algoritmus pokrývání množin pracující metodou „odděl a panuj“ (separate and conquer). Při pokrývání množin jde totiž o to nalézt pravidla, která pokrývají příklady téže třídy a oddělit je od příkladů třídy jiné.

Ilustrační príklad (pokračování)**:**

Pro naše data lze nalézt pravidla, jejichž použití pro rozhodování o novém klientovi je následující: Nalezneme první pravidlo, jehož předpokladům klient vyhovuje. Závěr tohoto pravidla pak určí, zda dostane úvěr či ne.

Strom převedený na pravidla:

If příjem(vysoký) *then* úvěr(ano)

If příjem (nízký) $\wedge$ konto(vysoké) *then* úvěr(ano)

If příjem (nízký) $\wedge$ konto(střední) $\wedge$ nezaměstnaný(ano) *then* úvěr(ne)

If příjem(nízký) $\wedge$ konto(střední) $\wedge$ nezaměstnaný(ne) *then* úvěr(ano)

If příjem(nízký) $\wedge$ konto(nízké) *then* úvěr(ne)

V případě **asociačních pravidel** není žádný atribut vyčleněn jako cíl klasifikace. Asociační pravidla hledají „všechny zajímavé“ asociace (implikace, ekvivalence) mezi hodnotami různých atributů. K výše uvedeným (rozhodovacím) pravidlům tak mohou přibýt např. pravidla:

If nezaměstnaný(ano) *then* příjem(nízký)

If příjem(vysoký) *then* nezaměstnaný(ne)

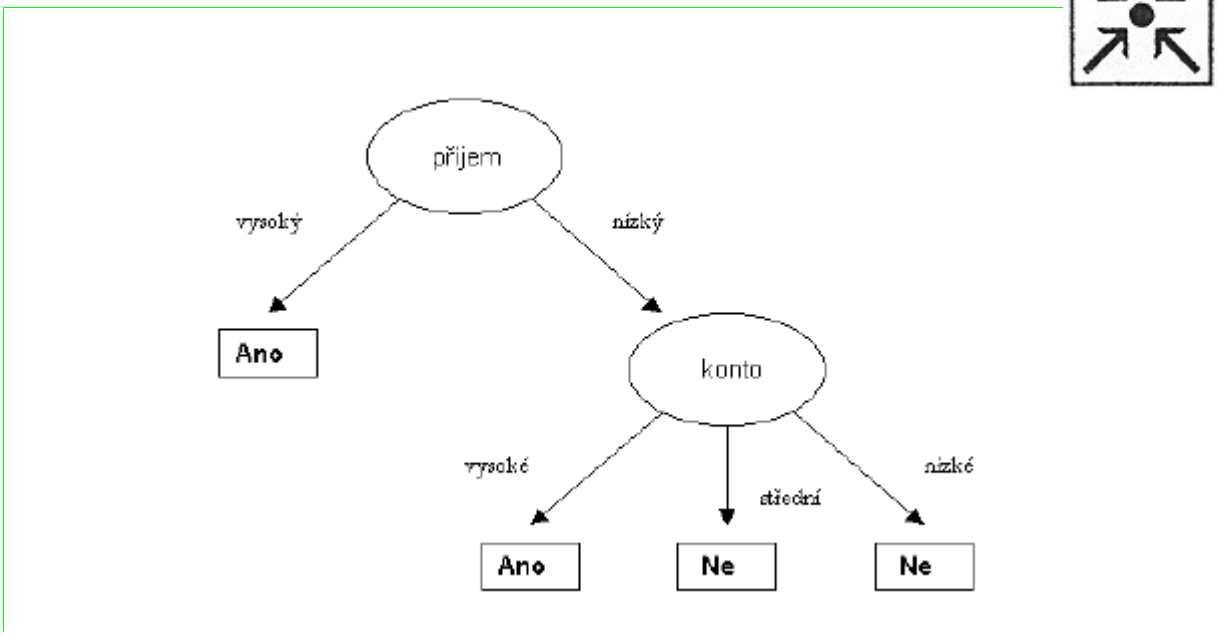
Prořezávání stromů, tj. redukce stromu, aby v listovém uzlu „převažovaly“ příklady jedné třídy. Algoritmus prořezávání lze zapsat takto:

1 Převeď strom na pravidla.

2 Generalizuj pravidlo odstraněním podmínky z předpokladu, pokud dojde ke zlepšení odhadované přesnosti:

3 Uspořádej prořezaná pravidla podle odhadované přesnosti; v tomto pořadí budou pravidla použita pro klasifikaci.

Prořezaný rozhodovací strom je uveden na obrázku 26



Obrázek 26: Prořezaný rozhodovací strom z obrázku 25

Kontrolní otázky

1. 1. Co je to strojové učení?
2. Co je to rozhodovací strom?
3. Na jakém principu pracuje metoda TDIDT?
4. Co jsou to rozhodovací pravidla?

Úkoly k zamyšlení:

Pro vytváření rozhodovacích stromů a pravidel byla vyvinuta celá řada jiných algoritmů. Mezi nejznámější patří CART, CLS, ID3, GID3, C4.5, AID, TREEDISC, QUEST a CHAID. Prostudujte si některý z nich a srovnejte s metodou TDIDT.

Korespondenční úkol:

Vytvořte rozhodovací strom a převeďte jej na pravidla pro data zadaná tabulkou následující tabulkou a na získaný strom aplikujte algoritmus prořezávání.

Hrát tenis / golf?

Den	Obloha	Teplota	Vlhkost	Vítr	Tenis/golf
D1	Slunečno	Vysoká	Vysoká	Slabý	Ne
D2	Slunečno	Vysoká	Vysoká	Silný	Ne
D3	Zataženo	Vysoká	Vysoká	Slabý	Ano
D4	Déšť	Střední	Vysoká	Slabý	Ano
D5	Déšť	Nízká	Normální	Slabý	Ano
D6	Déšť	Nízká	Normální	Silný	Ne
D7	Zataženo	Nízká	Normální	Silný	Ano
D8	Slunečno	Střední	Vysoká	Slabý	Ne
D9	Slunečno	Nízká	Normální	Slabý	Ano
D10	Déšť	Střední	Normální	Slabý	Ano
D11	Slunečno	Střední	Normální	Silný	Ano
D12	Zataženo	Střední	Vysoká	Silný	Ano
D13	Zataženo	Vysoká	Normální	Slabý	Ano
D14	Déšť	Střední	Vysoká	Silný	Ne

Shrnutí obsahu kapitoly

V této kapitole jsme se seznámili se základními principy strojového učení. Důraz zde byl kladen na induktivní tvorbu rozhodovacích stromů. Velká pozornost byla věnována demonstraci metody *Top down induction of decision trees* (TDIDT) pro tvorbu rozhodovacích stromů.

Pojmy k zapamatování

- rozhodovací strom;
- metoda TDIDT (Top Down Induction Of Decision Trees);
- rozhodovací pravidla
- asociační pravidla.

6 Učící algoritmy

V této kapitole se dozvíte:

- Jak pracuje algoritmus AQ (Covering Algorithm) – pokrývání zdola nahoru?
- Co je to reinforcement learning?
- Jak pracuje algoritmus učení z neklasifikovaných příkladů?

Po jejím prostudování byste měli být schopni:

- Vysvětlit práci algoritmu AQ (Covering Algorithm).
- Vysvětlit, co je to reinforcement learning.
- Vysvětlit práci algoritmu učení z neklasifikovaných příkladů.

Klíčová slova této kapitoly:

Učení z klasifikovaných příkladů, algoritmus AQ, protokoncepty, učení z neklasifikovaných příkladů, reinforcement learning, Q-learning.

Průvodce studiem

V této kapitole si ozřejmíme princip učení z klasifikovaných příkladů (učení s učitelem), neklasifikovaných příkladů (učení bez učitele) a učení odměnou a trestem (reinforcement learning). V případě učení z klasifikovaných příkladů se stroji předkládají objekty se známou klasifikací, které si „zapamatuje“ a pak je schopen podle nich zařadit neznámý objekt do definovaných tříd, např. tohle je kruh, tohle je elipsa apod. Zatímco v případě učení z neklasifikovaných příkladů se stroji musí v parametrickém prostoru odhalit třídy objektů. Učení odměnou a trestem je svou podstatou nekontrolované učení, přičemž zpětná vazba zde sice nechybí úplně, ale je k dispozici až na konci celého procesu učení.

6.1 Dělení strojového učení

Strojové učení v zásadě dělíme na *induktivní* a *deduktivní*. Induktivní učení z informací na nejnižší úrovni (tj. na úrovni trénovacích příkladů) indukuje, resp. generuje obecnější *znalosti*. Deduktivní učení naopak ze znalostí na vysoké úrovni obecně dedukuje znalost méně obecnou a složitou, která je lépe uzpůsobena pro řešení konkrétního druhu problémů.

Další faktor, který má vliv na učení je *stupeň kontroly*. Podle stupně kontroly rozeznáváme kontrolované a nekontrolované učení, jež se používají hlavně na řešení klasifikačních a sekvenčních úloh. Při klasifikaci se jedná o rozhodnutí zařazení objektu do třídy. Příklad, který je definován hodnotami atributů resp. vlastností, klasifikujeme - tedy zařadíme do třídy. Při sekvenčních úlohách je třeba určit posloupnost kroků, která povede k žádoucímu řešení. *Kontrolované* učení disponuje zpětnou vazbou o úspěšnosti učení. To znamená, že při klasifikaci se předpokládá, že pro každý trénovací příklad je předem určena třída a cílem učení je indukovat popis, resp. definici pojmu, který přesně predikuje tuto třídu pro každý nový příklad (bez třídy). Indukovaný popis může mít formu: logické konjunkce, prahového pojmu, etalonu, rozhodovacího stromu nebo rozhodovacího seznamu. Při sekvenčních úkolech tutor, nebo expert navrhne korektní krok v každém stavu v prohledávacím, nebo vysvětlovacím procesu. Na druhé straně při *nekontrolovaném* učení neexistuje zpětná vazba o vhodnosti výkonu. Tedy klasifikační úlohy nemají předem specifikovanou třídu. Proto při nekontrolovaném učení se příklady pouze shlukují do shluků podle nějakého kritéria, nejčastěji podobnosti. Při sekvenčních úlohách učící se subjekt musí odlišit vhodný krok od nežádoucího. Například když hraje šachy a nemá učitele, který by hned zpětně okomentoval každý jeho krok z hlediska vhodnosti. Vzniká podúloha a to určení kreditu, který by pomohl učícímu se subjektu rozlišit kroky vedoucí k úspěchu (učení odměnou a trestem).

Na základě *způsobu*, jakým se získávají trénovací příklady, dělíme učící úlohy na úlohy typu *online* a *offline*. Úkoly typu online získávají trénovací příklady jednotlivě a postupně (inkrementální učení). Úkoly, které řeší člověk, mají většinou online povahu, protože lidé existují v časovém světě. Úkoly typu offline disponují všemi trénovacími příklady současně (neinkrementální učení). I u člověka se může výjimečně vyskytnout takový případ, kdy je konfrontován s masou shromážděných znalostí, např. když student přechází do nové oblasti. Kompromisem mezi úlohami typu online a offline je učící úloha, která získá sadu trénovacích příkladů jednou za čas.

Algoritmy, které se používají na řešení studujících úkolů (typu online i offline) dělíme stejným způsobem na *inkrementální* a *neinkrementální*. Inkrementální učení je učení pomocí algoritmu, který zpracovává jeden trénovací příklad za druhým. Po každém příkladu algoritmus poskytuje řešení. Inkrementální učení odpovídá online výukového úkolům. Neinkrementální učení je učení pomocí algoritmu, který zpracovává celou množinu trénovacích příkladů. Odpovídá offline učícím úkolům. Nicméně je možné aplikovat neinkrementálne učení i na online úkoly, a to tak, že se předchozí trénovací příklady zdrží v paměti a každý nový příklad se přidá k této množině.

6.2 Učení z klasifikovaných příkladů

Pro učení z klasifikovaných příkladů popsaných pomocí atributů je určen algoritmus AQ (Covering Algorithm) – pokrývání zdola nahoru. Jeho výstupem je soubor pravidel popisujících všechny pozitivní příklady z trénovací množiny. Algoritmus lze shrnout do následujících bodů (Michalski, 1998):

1 Rozděl množinu příkladů na dvě podmnožiny: podmnožinu **P** pozitivních příkladů a podmnožinu **N** negativních příkladů.

2 Vyber z množiny **P** jeden příklad a označ jej *s* (*seed* neboli jádro).

3 Nalezni všechny maximální generalizace popisu jádra *s* tak, že jimi nesmí být pokryt žádný negativní příklad.

4 Podle vhodného preferenčního kritéria vyber nejlepší z těchto popisů a zařaď jej do množiny popisů. Odstraň z množiny **P** všechny příklady pokryté tímto popisem.

5 Je-li množina **P** prázdná, ukonči práci (výsledným popisem je disjunkce všech nalezených popisů). V opačném případě se vrať na bod 2.

Možná zdokonalení algoritmu AQ:

- *Případ více tříd.* Základní podoba algoritmu uvažuje dvě třídy. Rozšíření se obvykle provede tak, že pro každou třídu c se za pozitivní považují příklady této třídy a za negativní všechny ostatní. Jinou možností je vytvářet pravidla ke všem třídám současně.
- *Data zatížená šumem*. Algoritmus můžeme upravit tak, že v kroku 3 nepožadujeme, aby pravidlo pokrývalo příklady pouze jedné třídy.
- *Rozhodovací seznam.* Algoritmus AQ vytváří tzv. neuspořádaný soubor pravidel. Opakem je uspořádaný soubor pravidel, neboli rozhodovací seznam (decision list), ve kterém jsou pravidla propojena pomocí ELSE. Podmínka za ELSE IF tedy implicitně obsahuje negaci podmínek všech předcházejících pravidel.

Maximální generalizace popisu příkladu (jádra *s*) je taková generalizace, která pokrývá, co největší počet příkladů. Představme si, že máme k dispozici jeden pozitivní a jeden negativní příklad:

příklad	at1	at2	at3	klasifikace
p1	*x*	*r*	*m*	+
p2	*y*	*r*	*n*	-

Dále nalezneme atribut, v němž se pozitivní a negativní příklad od sebe liší. Jakýkoliv popis, který obsahuje jinou hodnotu tohoto atributu, než jaká byla u negativního příkladu, může být považována za přípustnou generalizaci pozitivních příkladů. Za nejobecnější popis (maximální generalizace) považujeme ten popis, o němž se domníváme, že vyhovuje co největšímu množství pozitivních příkladů, a to i těch, které nebyly předloženy k učení.

První maximální generalizace je tedy: $at1 \neq y$.

Druhá maximální generalizace je: $at3 \neq n$.

Preferenční kritérium může být založeno na nákladové funkci tak, že přednost je dávána těm atributům, jejichž hodnoty jsou snadno dostupné. Relativní dostupnost jednotlivých atributů může být součástí znalostí, které mají usnadňovat učení.

Ilustrujme nyní celý algoritmus na příkladu. Předpokládejme, že máme k dispozici tři pozitivní a tři negativní příklady nějakého pojmu, tak jak jsou uvedeny v tabulce 4:

příklady		*at1*	*at2*	*at3*	*klasifikace*
PE	*o1*	*x*	*r*	*m*	+
	o2	*y*	*r*	*n*	+
	o3	*y*	*s*	*n*	+
NE	*o4*	*x*	*s*	*m*	-
	o5	*z*	*t*	*n*	-
	o6	*z*	*r*	*n*	-

Tabulka 4: Ilustrační příklad - data

Dále předpokládejme, že náš deskripční jazyk nezná negaci. Potom analýza pojmu „+“ pomocí algoritmu AQ proběhne takto:

První jádro: náhodně jsme zvolili *o2*. Nejobecnějším popisem tohoto jádra v daném deskripčním jazyku je (*at1=y*). Tento popis pokrývá příklady *o2* a *o3*., ale nepokrývá *o1* ani žádný z negativních příkladů.

Druhé jádro (vybrané z dosud nepokrytých pozitivních příkladů): o1. Z hlediska obecnosti existují dvě rovnocenné maximální generalizace: $(at1 = x) \wedge (at2 = r)$, $(at2 = r) \wedge (at3 = m)$.

Nechť podle nějakého kritéria vyhovuje lépe první z těchto dvou popisů (např. proto, že atribut *at1* se snáze měří). Pojem „+“ je proto nejlépe popsán takto: $(at1 = y) \vee [(at1 = x) \wedge (at2 = r)]$.

6.3 Učení z neklasifikovaných příkladů

Podstatou metody založené na učení z neklasifikovaných příkladů je hledání vhodných generalizací jader a snaha o co nejdokonalejší pokrytí celého souboru příkladů (Mařík, 2003).

1. Z množiny všech příkladů náhodně vybereme *k* jader (číslo *k* zadává uživatel). Označme je $e_1,\ldots,e_k$.
2. Pro každé jádro hledáme dostatečně obecný popis, který by jej odlišil od všech ostatních jader. Každý takový popis nazveme protokonceptem. Máme tedy tolik protokonceptů, kolik je jader.
3. Provedeme optimalizaci protokonceptů tak, aby byly disjunktní. Jako výsledek dostaneme množinu konceptů. Zpravidla existuje několik různých řešení (množin konceptů), z nichž pomocí vyhodnocovací funkce vybereme to nejlepší.
4. Není-li splněno zvolené kritérium ukončení, vybereme nová jádra a přejdeme zpět ke kroku 2. Jádra volíme tak, že z každého stávajícího protokonceptu vybereme jedno.

Vyhodnocovací funkce je součástí znalostí zadaných uživatelem a slouží k porovnání kvality různých řešení, tedy různých sad protokonceptů (shluků). Může zahrnovat celou řadu kritérií, jako je míra

rozdílnosti mezi jednotlivými shluky, počet proměnných, které umožňují diskriminovat mezi všemi shluky, jednoduchost výsledného řešení, přítomnost určitých struktur v řešení apod. Výslednou funkci pak můžeme chápat jako posloupnost párů (c_1, t_1), (c_2, t_2),…, kde c_i je kritérium a t_i je práh tolerance vyjádřený v procentech. Na počátku jednotlivá řešení vyhodnotíme (a seřadíme) pomocí prvního kritéria a ponecháme pouze prvních t_1 procent. Poté přejdeme k druhému kritériu a opět ponecháme pouze t_2 procent nejúspěšnějších řešení a tak pokračujeme, až dokud nezůstane pouze jediné řešení, aneb dokud se nevyčerpá seznam kritérií. Ukončovací kritérium je zadáno uživatelem a jeho podstatou je sledování, zda současné řešení je lepší (podle vyhodnocovací funkce) než minulé řešení. Jestliže po několik iterací (tento počet je zadán uživatelem) nepozorujeme žádné zlepšení, proces ukončíme a vybereme to řešení, které je nejlépe ohodnoceno.

Celou proceduru si podrobněji rozebereme na příkladu. Předložíme učícímu se programu neklasifikované příklady z tabulky 5:

příklad	*at1*	*at2*	*at3*
o1	*a*	2	110
o2	*a*	4	100
o3	*b*	2	9
o4	*b*	3	10
o5	*c*	5	20
o6	*c*	4	15
o7	*a*	5	200
o8	*b*	4	50

Tabulka 5: Ilustrační příklad - řešení

Kromě příkladů jsou k dispozici znalosti usnadňující generalizaci popisů. Tyto znalosti říkají, že atribut *at1* může nabývat pouze symbolických hodnot *a*,*b*,*c*, atribut *at2* nabývá celočíselných hodnot od 2 do 6 a atribut *at3* nabývá celočíselných hodnot od 1 do 300, přičemž

hodnoty pod 30 jsou považovány za malé, hodnoty nad 150 jsou považovány za velké a všechny ostatní jsou střední. Máme vytvořit dva shluky. Náhodně tedy vybereme dva jádra, např. *o2* a *o5*. Provedeme maximální generalizaci jejich popisu, tak aby popis *o2* nepokrýval *o5* a naopak. Dostaneme dva protokoncepty:

protokoncept(*o2*): (*at1* $\neq$ *c*) $\vee$ (*at2* $\neq$ 5) $\vee$ (*at3* $\neq$ 20),

protokoncept(*o5*): (*at1* $\neq$ a*c*) $\vee$ (*at2* $\neq$ 4) $\vee$ (*at3* $\neq$ 100).

Vidíme, že všechny zbývající příklady *o1*, *o3*, *o4*, *o6*, *o7* a *o8* vyhovují oběma těmto popisům, oba protokoncepty se tedy silně překrývají. Proto vytvoříme z prototypů disjunktní koncepty. Utvořme seznam *L* ze všech příkladů, které jsou pokryty více než jedním protokonceptem. Vezměme první prvek *L*, jímž je *o1*, vyjměme jej z tohoto seznamu a vložme jej nejprve do prvního shluku, který tedy nyní obsahuje prvky *o1* a *o2*. Nyní vytvoříme nový, co nejobecnější popis tak, aby pokrýval prvky *o1* a *o2*, ale aby se nepřekrýval s druhým protokonceptem. Nyní máme k dispozici tyto popisy:

koncept vytvořený kolem *o1* a *o2*: (*at1* = *a*)$\wedge$(*at2* $\neq$ 2..4)$\vee$(*at3* = střední),

koncept(*o5*): (*at1* $\neq$ a*c*) $\vee$ (*at2* $\neq$ 4) $\vee$ (*at3* $\neq$ 100).

Podobně vložíme příklad *o1* do druhého protokonceptu místo do prvního, dostaneme následující popisy:

koncept(*o2*): (*at1* $\neq$ *c*) $\vee$ (*at2* $\neq$ 5) $\vee$ (*at3* $\neq$ 20),

koncept vytvořený kolem *o1* a *o5*:
(*at1* = *a*$\vee$*c*) $\wedge$ (*at2* $\neq$ 2..5) $\wedge$ (*at3* = male $\vee$střední),

Předpokládejme, že podle vyhodnocovacích kritérií lépe vyhovuje první dvojice popisů. Začleníme tedy *o1* do tohoto konceptu a celou proceduru opakujeme s příkladem *o4* a tak dále, dokud nevyprázdníme seznam *L*. Ve chvíli, kdy se seznam vyprázdní, máme dva koncepty, z nichž jeden pokrývá *o1*, *o2*, *o7* a druhý *o3*, *o4*, *o6*, *o8*. Toto řešení

uložíme do paměti. Řešení pokračuje následovně. Vybereme nová dvě jádra, každé z jiného konceptu vytvořeného v minulé iteraci, např. *o1* a *o6*. Nyní opakujeme celou proceduru a získáme jiné řešení. Porovnáme toto řešení s předchozím pomocí vyhodnocovací funkce. Je-li toto řešení lepší než to minulé, uložíme jej do paměti místo minulého řešení a pokračujeme další iterací a to tak dlouho, dokud není splněno kritérium ukončení.

6.4 Učení odměnou a trestem

Učení odměnou a trestem bylo motivované snahou o řešení úloh sekvenčního typu. Jsou to úkoly, v nichž je daný počáteční stav a konečný stav. Úkolem je najít resp. naučit systém optimální cestu od počátečního do koncového stavu. Učení odměnou a trestem je svou podstatou nekontrolované učení, protože učící se objekt nemá k dispozici okamžitou zpětnou vazbu o vhodnosti svého rozhodnutí po každém kroku. Zpětná vazba nechybí úplně, ale je k dispozici až na konci celého procesu učení ve formě výsledku snažení (vyhrál šachovou partii resp. nevyhrál). Anglický ekvivalent *reinforcement learning* (RL) se často používá bez překladu.

Abychom definovali náš problém, musíme určit, co je dáno a co chceme získat. *Dané jsou*: částečné znalosti problémové domény, zkušenosti s prohledáváním problémového prostoru. *Získáváme*: znalosti, které umožňují přesné rozhodnutí v každém stavu. Tyto znalosti se používají resp. aplikují na sekvenční roli učení, kdy učení spočívá ve vylepšování rozhodování. Tuto úlohu je možné považovat i za klasifikační úlohu, kdy každý stav se klasifikuje do nějaké třídy, přičemž třídy reprezentují možná rozhodnutí. Kvalitu získaného řešení můžeme posoudit na základě efektivnosti hledaného řešení, spolehlivost plánů po realizaci v externím světě či kvality návrhu. Jakýkoliv typ vylepšení silně závisí na provedení přesných rozhodnutí v každém stavu. Řešitel úkolu se dozví, zda jeho rozhodnutí bylo správné (nesprávné) resp. přesné

(nepřesné) až dlouho poté co ho provedl. To vede k dvěma základním jistotám v učení tohoto typu:

- přiřazení odměny (kreditu) dobrým rozhodnutím nebo akcím
- přiřazení pokuty nevhodným rozhodnutím nebo akcím

Přiřazení odměny nebo pokuty je třeba kvůli absenci okamžité zpětné vazby. Dané preference se realizují pomocí *ohodnocovací funkce*, která přiděluje stavům odměnu resp. trest. Odměna (reward) se přiděluje více žádaným stavům - čím vyšší odměna, tím žádanějším stav. Na druhé straně je zde trest (negative reward), který se přiděluje méně žádaným nebo nežádoucím stavům. Učení představuje získání řídicí strategie, která vždy povede ke stavu s nejvyšší odměnou. Řídící strategie se získává ze zkušeností s odměnami nebo tresty, které se vyskytly ve zkoumaných sekvencích stavů, získaných aplikací operátorů.

Ilustrační příklad:

Tabulkový přístup je nejjednodušší z použitelných přístupů k danému typu učení. Tento přístup ukládá řídící znalosti v tabulce 6.

STAV	**OPERÁTOR**	**ODMĚNA**
(kostka **a**)(kostka **b**)(kostka **c**)(stůl **t**)(**b** na **a**) (**a** na **c**)(**c** na **t**)(prázdné **b**)(prázdné rameno)	(slož **b** z **a**)	0.1
(kostka **a**)(kostka **b**)(kostka **c**)(stůl **t**)(**a** na **c**) (**c** na **t**)(prázdné **a**)(drží **b**)	(polož **b** na **t**) (polož **b** na **a**)	0.2 0.0
. . .	. . .	. . .
(kostka **a**)(kostka **b**)(kostka **c**)(stůl **t**)(**b** na **c**) (**c** na **t**)(prázdné **b**)(drží **a**)	(polož **a** na **b**) (polož **a** na **t**)	0.9 0.0

Tabulka 6: Tabulkový přístup - řešení

Každý vstup tabulky popisuje možný pár stav - akce, společně s očekávanou odměnou. Odměna reprezentuje vhodnost provedení

dotyčné akce v tom stavu. Například doména *s* počtem stavů **p** a počtem akcí **a** bude vytvářet tabulku, jejíž velikost je dána součinem **p** a **a**. Pokud se v daném stavu akce nedá použít, příslušná buňka tabulky bude prázdná. Takovou tabulku je možné znázornit jako orientovaný ohodnocený graf, jehož uzly znázorňují stavy a hrany znázorňují akce. Jednotlivé hrany jsou ohodnoceny. Hodnotící skóre (odměna nebo trest) je to co se učí. Při řešení sekvenčních úloh se používá typicky dopředné zřetězené prohledávání a to vyžaduje:

- hledání tabulkových vstupů pro aktuální stav
- výběr akce s nejvyšším ohodnocením
- aplikování vybrané akce k dosažení nového stavu.

Tento proces založený na třech krocích se opakuje, dokud se nedosáhne požadovaný stav, nebo není splněna jiná ukončovací podmínka. Na podporu průzkumu alternativních cest některé systémy používají stochastické schéma, které vybírá akce pomocí pravděpodobnostní funkce jejich predikované odměny.

Algoritmus učení mění predikovanou odměnu uloženou v tabulce stavů a akcí na základě zkušenosti. Vstupy obvykle startují s libovolnými nebo přiměřenými (odhadnutými na základě apriorních znalostí o problému) hodnotami. Skóre pro každý pár *stav-akce* (**s**, **a**) se mění pokaždé, když řešitel úlohy aplikuje akci **a** ve stavu **s**. To vytvoří novou výslednou odměnu, která se použije k aktualizaci daného vstupu. Například, pokud je k dispozici rada učitele a ten vyznačí sekvenci operátorů, která představuje vhodné řešení, pak se všechny odměny na dané cestě zvýší.

Q-learning (Machová, 2002) je algoritmus učení odměnou a trestem, jehož základem je následující aktualizační schéma. Tento režim definuje způsob, jakým se mění ohodnocení akcí v jednotlivých stavech: $\Delta Q(\mathbf{s},\mathbf{a}) \leftarrow \beta[r(\mathbf{s},\mathbf{a}) + \gamma U(\mathbf{s}') - Q(\mathbf{s},\mathbf{a})]$

kde:

- 0 <γ <1 je redukční faktor
- 0 <β <1 je faktor rychlosti učení
- *Q*(**s**,**a**) je *interní odměna* resp. tabulkový vstup pro stav **s** a akci **a**. Je předmětem procesu učení.
- **s'** je výsledný stav po aplikaci akce **a** ve stavu **s**
- *r*(**s**,**a**) je *externí odměna*, vyplývající z aplikace **a** ve stavu **s**. Externí odměna může být dána učitelem, nebo může být získána z trénovacích příkladů. Představuje a priori znalosti o řešené úloze. Není předmětem učení. Zadává se na počátku, před spuštěním procesu učení a nemusí být přiřazena každému páru (**s**,**a**). Nejčastěji se externí odměna přiřadí ke stavu, ve kterém končí sekvence operátorů (akcí), z níž se právě učí. Pokud to byla úspěšná sekvence, pak je odměna kladná. Jinak může být i záporná.
- *U*(**s'**) je maximum z očekávaných odměn *Q*(**s'**,**a'**) pro všechny akce **a'**, které mohou být aplikovány ve stavu **s'**.

Tedy, jak z výše uvedeného vyplývá, algoritmus používá dvě tabulky: *r*(**s**,**a**) tabulku a *Q*(**s**,**a**) tabulku. Strategie *Q-learning* vypočítává součet externí odměny *r*(**s**,**a**) a redukované maximální odměny, očekávané v následujícím stavu γ*U*(**s'**). Od tohoto součtu odečte aktuální vstup *Q*(**s**,**a**). Výsledek násobí faktorem rychlosti učení a získanou hodnotou aktualizuje zpracovávaný vstup takto: $Q(\mathbf{s},\mathbf{a}) \leftarrow Q(\mathbf{s},\mathbf{a}) + \Delta Q(\mathbf{s},\mathbf{a})$. Faktor rychlosti učení β určuje rozsah, v němž můžeme revidovat tabulkové vstupy. Redukční faktor γ specifikuje význam poměru aktuální versus očekávaná odměna. Při dostatečném počtu trénovacích příkladů výše uvedená aktualizační schéma konverguje k následujícímu výrazu: $Q(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma U(\mathbf{s}')$.

Cílem učení je transformovat počáteční ohodnocovací funkci (externí odměnu) na revidovanou ohodnocovací funkci (interní odměnu, která roste monotónně). Pro každý příklad (sekvenci operátorů) je nutné mnohonásobné opakování učení, jímž se jednotlivé hodnoty interních odměn vhodně rozprostřou podél dané cesty v grafu. Po dostatečném množství experimentů se může učící algoritmus přemístit do nejžádanějšího stavu z jakéhokoliv místa v problémovém prostoru. Učení je možné urychlit změnou faktoru rychlosti učení β během učení. Začíná se s velkou hodnotou faktoru rychlosti učení, čímž se umožní hrubé přibližování v počáteční fázi učení. Postupně se tento parametr snižuje, čímž se umožní v posledních etapách učení přesnější ladění. Strategie Q-learning navazuje na metody dynamického programování.

Tento způsob učení má řadu atraktivních vlastností, ale i nevýhod. **K výhodám patří:**

- nepožaduje znalosti o účinnosti operátorů
- dokáže zvládnout neurčité a zašuměné domény
- může spolupracovat s vnějším světem.

Mezi nevýhody patří:

- tendence k nízké rychlosti učení, zvláště při dlouhých řešících cestách
- závislost na postupném šíření odměn zpětně podél hledané cesty
- požadavek velkého množství trénovacích cest k dosažení smysluplných vstupů pro každý pár (**s**,**a**)
- nutnost uchovávat alespoň jeden vstup pro každý stav způsobuje exponenciální nárůst velikosti potřebné paměti v případě narůstání délky řešící cesty, resp. složitosti úkolu.

Jednou z odpovědí na tyto problémy je řešení, ve kterém programátor rozdělí problémový prostor na smysluplné segmenty a trénuje učící systém odděleně nad každým segmentem. Individuální cesty v jednotlivých segmentech jsou pak přirozeně kratší, z čehož vyplývá rychlejší dosažení odměny.

Kontrolní otázky:

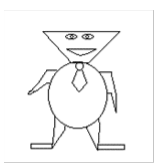
1. Jak pracuje algoritmus AQ (Covering Algorithm) – pokrývání zdola nahoru?
2. Jak pracuje algoritmus učení Q-learning?
3. Jak pracuje algoritmus učení z neklasifikovaných příkladů?

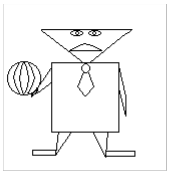
Úkoly k zamyšlení:

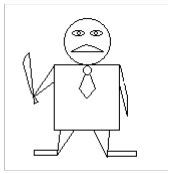
Zamyslete se nad typickými úlohami pro použití učení z klasifikovaných příkladů, učení z neklasifikovaných příkladů a učení odměnou a trestem.

Korespondenční úkol:

Vyřešte následující úlohu. *Můžeme se naučit roboty rozlišit na základě krátké zkušenosti?* Vytvořte tabulku s atributy: Klasifikace (přítel/nepřítel), usmívá se (ano/ne), kravata (ano/ne), tělo (kruh, trojúhelník, čtverec), hlava (kruh, trojúhelník, čtverec), co má v ruce (nic, míč, květ, meč).

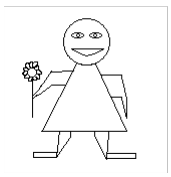


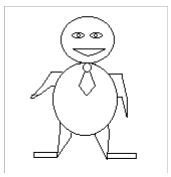


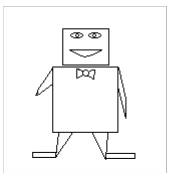


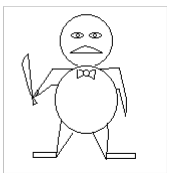
přátelství

nepřátelství









Shrnutí obsahu kapitoly

Při učení z klasifikovaných příkladů (učení s učitelem) se stroji předkládají objekty se známou klasifikací, které si „zapamatuje“ a je schopen podle nich zařadit neznámý objekt do definovaných tříd. Je to poměrně časově náročná činnost a navíc s rizikem, že učitel přiřadí objekt do špatné třídy, tj. chyba v trénovacích datech.

Při učení z neklasifikovaných příkladů (učení bez učitele) se stroji musí v parametrickém prostoru odhalit třídy objektů. Například pomocí vyhledávání shluků v parametrickém prostoru.

Učení odměnou a trestem je svou podstatou nekontrolované učení, přičemž zpětná vazba zde sice nechybí úplně, ale je k dispozici až na konci celého procesu učení.

Pojmy k zapamatování

- učení z klasifikovaných příkladů;
- algoritmus AQ;
- reinforcement learning;
- Q-learning.

7 Přírodou inspirované metody strojového učení

V této kapitole se dozvíte:

- Co je to deduktivní a induktivní modelování.
- Co jsou to umělé neuronové sítě.
- Jaké jsou hlavní rysy evolučních algoritmů.

Po jejím prostudování byste měli být schopni:

- Vysvětlit podstatu deduktivního a induktivního modelování.
- Pochopit základní pojmy z oblasti umělých neuronových sítí.
- Uvést hlavní rysy evolučních algoritmů.

Klíčová slova této kapitoly:

Deduktivní modelování, induktivní modelování, umělé neuronové sítě, přenosová funkce neuronu, evoluční algoritmy, křížení, mutace.

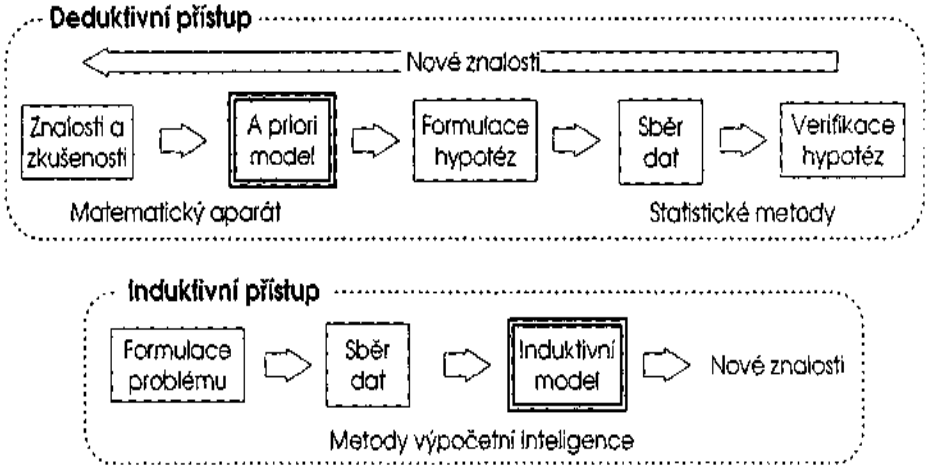
Průvodce studiem

Motivací pro metody, které dále v této kapitole popíšeme, je hledání nejlepších řešení metodami inspirovanými přírodou. Nejlepší řešení (v našem případě) závisí na „nějakých " spojitých parametrech a právě jejich optimální hodnoty hledáme. V této kapitole se budeme zejména věnovat základním pojmům z oblasti umělých neuronových sítí a evolučních algoritmů, které byly převážně převzaty ze (Zelinka, 1999).

7.1 Deduktivní a induktivní modelování

Při hledání optimálního modelu systému (např. pro predikci, aproximaci, klasifikaci atd.) lze postupovat dvěma způsoby (Mařík, 2007).

- *Dedukcí*: na počátku na základě zkušeností a poznatků vytvoříme obecný model, ten pak ověříme na reálných datech, popř. přizpůsobíme jeho parametry, aby lépe vystihoval chování reálného systému.
- *Indukcí*: na počátku máme několik základních stavebních jednotek, ze kterých se během učící fáze snažíme vytvořit (odvodit) model, který nejlépe požadované chování realizuje. Máme tedy k dispozici pouze data, nevyužíváme žádné externí znalosti o povaze modelovaného systému. Tento přístup je typický například pro umělé neuronové sítě.



Obrázek 27: Postup induktivního a deduktivního přístupu hledání modelů pro vytěžování dat

Indukční modelování spočívá ve vytváření modelů chování reálných systémů, u kterých známe jejich vnější projevy (reakce výstupů na vstupy). Přitom je nám ale vnitřní struktura těchto reálných systémů neznámá, nebo je poznatelná jen velmi obtížně. Hledání optimálního modelu induktivním přístupem lze např. připodobnit přírodnímu principu výběru nejsilnějšího jedince. Zde se za nejsilnějšího jedince považuje model, který je „nejlepší" ze všech modelů, které během hledání (učení) vytváříme. Matematicky lze selekci nejlepšího modelu popsat následujícím způsobem (8):

$$m = \arg\min CR(m), \text{ kde } m \& M \text{ a } CR(m) = j(C, \sigma, T) \qquad (8)$$

kde m představuje optimální (hledaný) model, M množinu všech zkoumaných modelů a CR externí kritérium, na základě kterého se posuzuje kvalita modelu. Parametry externího kritéria jsou v tomto případě složitost modelu C, rozptyl šumu v datech a σ typ transformace dat T. Externí kritérium CR musí nabývat minima pro modely, jejichž složitost jednak odpovídá složitosti modelovaných závislostí v datech (složitost T), a zároveň bere v úvahu rozptyl šumu v datech (σ). Tak lze zabránit přespecifikování neboli přeučení modelů (overfitting).

7.2 Umělé neuronové sítě – základní pojmy

Neuronová síť je jeden z výpočetních modelů používaných v umělé inteligenci. Jejím vzorem je chování odpovídajících biologických struktur. Umělá neuronová síť je struktura určená pro distribuované paralelní zpracování dat. Skládá se z umělých (nebo také formálních) neuronů, jejichž vzorem je biologický neuron. Neurony jsou vzájemně propojeny a navzájem si předávají signály a transformují je pomocí tzv. přenosových funkcí. Neuron má libovolný počet vstupů, ale pouze jeden výstup.

Rozdíl mezi klasickým počítačem a neuronovou sítí je ten, že při používání klasického PC musíme vytvořit program, který řeší daný problém. Do tohoto programu obvykle zahrneme ve formě podmínek a rozhodovacích instrukcí veškeré dostupné informace. Co se však stane, jestliže je náš program postaven před problém, který patří do třídy známých problémů, ale je dost odlišný? Obvykle je buď ignorován, nebo v lepším případě je obsluha jen upozorněna, že se vyskytl neznámý případ, který byl odložen bokem. V takovém případě musí opět nastoupit programátor a program upravit.

Co se však stane, použije-li se neuronová síť? Nemusí se vymýšlet žádný algoritmus a v případě vhodné konfigurace a dobrého učení daná

neuronová síť zareaguje správným způsobem a novou informaci zařadí s velkou pravděpodobností do správné třídy. Není potřeba žádné úpravy sítě. Samozřejmě, že nic není neměnné, takže jak vzrůstá počet nových informací, je pravděpodobné, že nám vzniknou i nové třídy informací, na které se daná síť musí doučit a případně pozměnit svou konfiguraci konfiguraci. Ale to se dá obejít bez přítomnosti člověka.

Jako názorný příklad lze uvést dva programátory, kteří se neznají a oba dva dostanou stejný úkol. Musí udělat algoritmus, který má umět rozlišit podle vstupních informací, do jaké třídy patří daný dopravní prostředek. Kolo do třídy kol, auto do třídy aut, letadlo do třídy letadel, atd. Klasický programátor to bude asi řešit tak, že vytvoří kriteriální filtry typu „jestliže objekt má dvě kola, řetězový převod na zadní kolo, řídítka, sedátko pro jednu osobu, atd. pak jej zařaď do třídy kol" a totéž udělá pro všechny možné třídy. Samozřejmě, že chytřejší programátor bude popis dělat v proporcích a ne v absolutních rozměrech z jednoduchého důvodu - i malé dětské kolo je kolo. Co se však stane, jestliže má jeho program vyhodnotit kolo z minulého století (obří přední kolo s maličkým vzadu a navíc bez převodu - pedály byly přímo na předním kole), nebo zařadit trojkolku či nový futuristický model kola? Pravděpodobně ho vyřadí a v lepším případě dá vědět obsluze, že došlo k vyřazení neznámého případu - programátor musí udělat úpravu. To stojí čas a peníze. To však neplatí v případě neuronové sítě. Ta by s velkou pravděpodobností provedla správné vyhodnocení - zařadila by nový objekt (kolo) do správné třídy.

Několik důležitých rozdílů mezi klasickým PC a neuronovou sítí je uvedeno v následující tabulce 7 (Zelinka, 1999).

Neuronová síť	Počítač
Je učena nastavováním vah, prahů a struktury	Je programován instrukcemi, (if, then, go to,...)
Paměťové a výkonné prvky jsou uspořádány spolu	Proces a paměť pro něj jsou separovány
Paralelismus	Sekvenčnost
Tolerují odchylky od originálních informací	Netolerují odchylky
Samoorganizace během učení	Neměnnost programu

Tabulka 7: Rozdíly mezi neuronovou sítí a PC

Typy umělých neuronových sítí.

Všechny typy neuronových sítí jsou složeny ze stejných stavebních jednotek - neuronů. Mohou obsahovat různé přenosové funkce, spojení mezi sebou a adaptivní - učící algoritmus. To vše pak určuje, o jakou síť se jedná.

Umělé neuronové sítě se dělí podle několika kritérií (Zelinka, 1999).

Podle počtu vrstev:

- s jednou vrstvou (Hopfieldova síť, Kohonenova síť, ...)
- s více vrstvami. (ART síť, Perceptron, klasická vícevrstvá síť s algoritmem Backpropagation)

Dělení podle počtu vrstev znamená, že rozlišujeme z kolika vrstev se daná síť skládá. Existují sítě s jednou vrstvou, se dvěma, třemi a více vrstvami. Sítě s jednou či dvěma vrstvami bývají většinou speciální sítě jako např. Hopfieldova, Kohonenova či ART síť, které mají svůj speciální učící algoritmus a topologii, zatímco pro sítě se třemi a více vrstvami se obvykle používá topologie klasické vícevrstvé sítě a adaptační algoritmus Backpropagation. Pro topologii sítě obvykle platí pravidlo, že každý neuron bývá spojen s každým neuronem ve vyšší vrstvě, obrázek 29. Zvláštností je např. Hopfieldova síť, ve které je

spojen každý neuron se všemi ostatními. Každé spojení mezi neurony je ohodnoceno váhami, které mohou nabývat různých hodnot a vyjadřují, jaký význam tento spoj má pro daný neuron. To ovšem neznamená, že spoje s malou vahou lze zanedbat, protože nikdy není jasné, jaký vliv má tento vstup na celkovou činnost sítě.

Podle typu algoritmu učení:

- s učitelem (síť s algoritmem Backpropagation,...)
- bez učitele (Hopfieldova síť,...)

Učení s učitelem znamená, že se síť snaží přizpůsobit svou odezvu na vstupní informace tak, aby se její momentální výstup co nejvíce podobal požadovanému originálu. Je to stejné, jako když učíme dítě číst. Ukážeme mu písmeno (vstup) a vyslovíme ho např. „áááá" (požadovaný výstup). Pokud je dítě pozorné, pak se nás snaží napodobit - učí se. Učitelem zde není myšlen onen dotyčný dospělý, ale princip předložení fonetického vzoru a vyžadování jeho zopakování. Dotyčný dospělý pouze zajišťuje vykonávání těchto operací.

Učení bez učitele je proces, ve kterém síť vychází z informací, které jsou obsaženy ve vstupních vektorech. Je to stejné, jako když vám někdo vysype na jednu hromadu různé tvary jako krychličky, pyramidky, kuličky, tyčinky a jejich zdeformované variace a dá vám za úkol tuto hromadu roztřídit na předem neurčený počet hromádek podle tvaru – zde nebyly stanoveny „podmínky" učitele (tj. kolik hromádek jakých typů máte udělat) a tudíž se jedná u učení bez učitele. Průměrně inteligentní člověk začne třídit hromadu na hromádky podle typu (podobnosti) tvaru, tzn. hromádku krychliček, kuliček, atd. Pokud se vyskytne zdeformovaný tvar (např. krychlička se zdeformovanými stranami či useknutým rohem) tak jej pravděpodobně zařadí mezi krychličky. Pokud bude takhle zdeformovaných krychliček vice, určitě vytvoří třídu „zdeformované krychličky”. V případě že budou deformace příliš velké a tudíž nebude možné rozpoznat, co je to za tvar, bude určitě vytvořena třída „nerozpoznané". Pokud se čtenáři nezdá příklad s člověkem

vhodný, opak je pravdou. Mozek je velmi složitý neurosystém, který pomocí zraku, sluchu, hmatu a dalších vstupů zpracovává informace takto získané - v podstatě se takto adaptuje na své okolí.

Podle stylu učení na sítě s učením:

- deterministickým (např. algoritmus Backpropagation) stochastickým
- (náhodné nastavování vah)

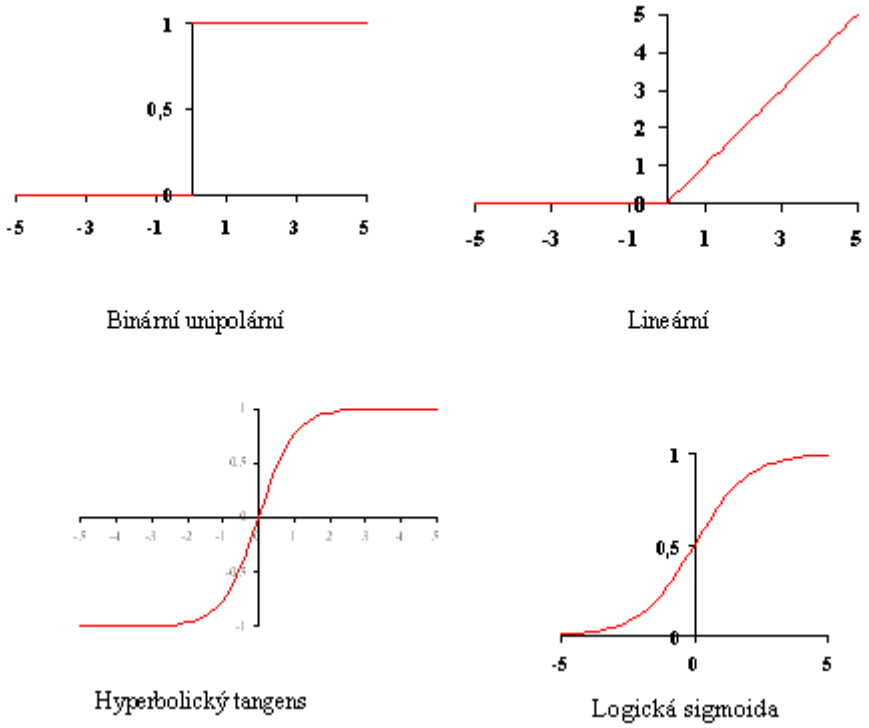
Styl učení v podstatě znamená, jak se přistupuje k nastavení vah sítě. V případě, že se jedná o nastavení vah výpočtem, pak mluvíme o deterministickém učení. Jestliže jsou však váhy získávány pomocí generátoru náhodných čísel, pak mluvíme o stochastickém stylu učení. Tento způsob získání vah sítě se obvykle používá jen při startu sítě (inicializaci sítě).

Přenosová funkce neuronu

Přenosová funkce neuronu je funkce, která transformuje vstupní signál na signál výstupní. Tato funkce může být skoková, či spojitá, ale musí být monotónní - tzn. že přiřazení odezev výstupu na vstup je jednoznačné.

Pro správný chod neuronu a neuronových sítí je důležité, jakou přenosovou funkci zvolíme. Přenosová funkce udává, jaká bude odezva výstupních neuronů na vstupní podnět. Jsou různé druhy funkcí, u kterých obecně platí, že jejich výstupní hodnota musí být z oboru hodnot dané funkce (sigmoida, hyperbolický tangens, binární funkce apod.). Nejpoužívanější přenosové funkce jsou uvedeny na obrázku 28 (Zelinka, 1999)

Volba funkce závisí na problému, který chceme řešit. Např. pokud chceme klasifikovat, zda je výrobek dobrý či špatný, pak nám stačí binární funkce. Pokud bychom použili funkci spojitou, pak musíme rozhodnout, jaká její hodnota (0.7, 0.8,...) znamená dobrý a jaká špatný výrobek.



Obrázek 28: Přenosové funkce a neuronů

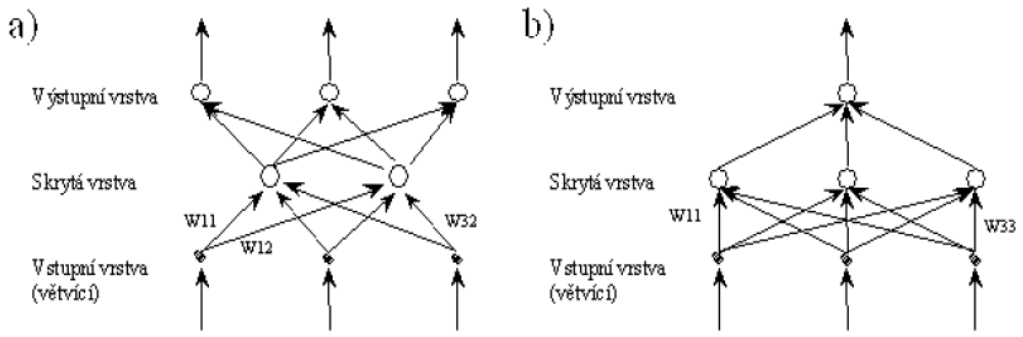
Obecné schéma neuronové sítě

Pod pojmem topologie (někdy také struktura) sítě, rozumíme způsob, jakým jsou mezi sebou spojeny jednotlivé neurony, vrstvy a kolik vstupů a výstupů síť má. Do topologie sítě lze zahrnout i typ přenosové funkce neuronů. Mezi parametry neuronové sítě dále patří parametr učení, momentum, atd.

V technickém slova smyslu si lze představit neuronovou síť jako síť složenou z jedné či několika vrstev, přičemž každá vrstva se skládá z „libovolného“ počtu neuronů. Libovolného samozřejmě jen teoreticky, protože jsme limitováni technickými podmínkami. Vlastní funkce

neuronové sítě je v podstatě transformační funkce, která přiřazuje jistému vstupnímu obrazu obraz výstupní. Důkaz matematického charakteru o takovéto funkci nebyl dlouho k dispozici. Kolmogororův teorém o řešení třináctého Hilbertova problému (Zelinka, 1999) aplikovaného na neuronové sítě vedl k poznatku, že k aproximaci libovolné funkce neuronovou sítí stačí, aby měla minimálně tři vrstvy s odpovídajícím počtem neuronů v každé z nich. Bohužel tento důkaz již nespecifikuje počet neuronů v těchto vrstvách, jež by byl z hlediska řešení daného problému optimální.

U vícevrstvých sítí platí, že první vrstva je vždy větvící, což znamená, že neurony ve vstupní vrstvě pouze distribuují vstupní hodnoty do další vrstvy. Vzhledem k tomu, že se jedná obecně o vícebodový vstup do sítě, tak hovoříme o vstupních (výstupních) vektorech informací. Jak je z obrázku 29 (Zelinka, 1999) vidět, tak počet neuronů v jednotlivých vrstvách je variabilní a záleží na řešeném problému.



Obrázek 29: Různé topologie neuronových sítí

Proč se síť učí

Učící schopnost neuronových sítí spočívá právě v možnosti měnit všechny váhy v síti podle vhodných algoritmů na rozdíl od sítí biologických, kde je schopnost se učit založena na možnosti tvorby nových spojů mezi neurony. Fyzicky jsou tedy obě schopnosti se učit založeny na rozdílných principech, nicméně z hlediska logiky ne.

V případě vzniku nového spoje - vstupu u biologického neuronu je to stejné, jako když v technické síti je spoj mezi dvěma neurony ohodnocen vahou s hodnotou 0, a tudíž jako vstup pro neuron, do kterého vstupuje, neexistuje. V okamžiku, kdy se váha změní z 0 na libovolné číslo, tak se daný spoj zviditelní - vznikne.

Jak síť funguje

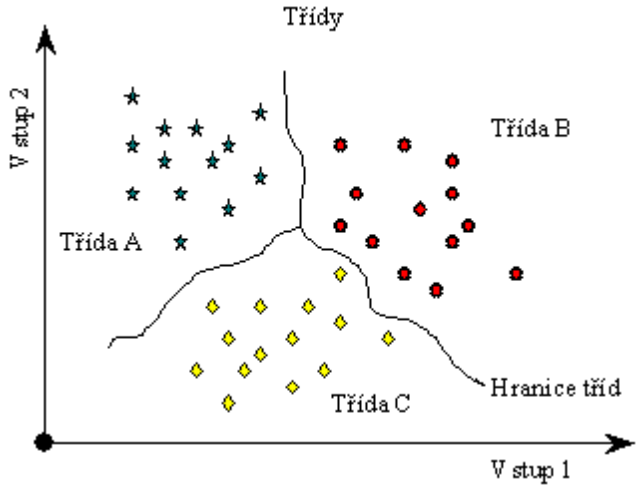
Jeden ze základních předpokladů pro funkci sítě je její naučení - adaptace na daný problém, během které se nastavují váhy sítě. Adaptační - učící proces se skládá ze dvou fází - adaptační a aktivační.

Nově vytvořená, ale také jakákoliv nenaučená neuronová síť neumí řešit žádný problém. Aby se mohla síť používat, musí být naučena stejně, jako kterýkoliv živý jedinec (samozřejmě, že zde se množství informací a délka učení nedá porovnávat). Z tohoto důvodu byly vyvinuty algoritmy, pomocí kterých se příslušná síť dokáže naučit na danou množinu informací. Algoritmus se obvykle dělí na dvě fáze, a to na fázi aktivační (vybavovací) a adaptační (učící), které ke své činnosti potřebují trénovací množinu. Trénovací množina je skupina vektorů obsahujících informace o daném problému pro učení. Pokud učíme síť s učitelem, pak jsou to dvojice vektorů vstup - výstup. Jestliže učíme síť bez učitele, pak trénovací množina obsahuje jen vstupní vektory. Pokud používáme jen fázi aktivační, pak mluvíme o vybavování. Tuto fázi používáme samostatně jen tehdy, když je síť naučena. Cyklické střídání obou fází je vlastní učení.

Aktivační fáze je proces, při kterém se předložený vektor informací na vstup sítě přepočítá přes všechny spoje včetně jejich ohodnocení vahami až na výstup, kde se objeví odezva sítě na tento vektor ve formě výstupního vektoru. Při učení se tento vektor porovná s vektorem originálním (požadovaným, výstupním) a rozdíl (lokální odchylka - chyba) se uloží do paměťové proměnné.

Adaptační fáze je proces, při kterém je minimalizována lokální chyba sítě tak, že se přepočítávají váhy jednotlivých spojů směrem z výstupu na vstup za účelem co největší podobnosti výstupní odezvy s originálním vektorem. Poté se opět opakuje aktivační fáze. Další získaný rozdíl (lokální odchylka) se přičte k předchozímu, atd. Pokud se tímto postupem projde celá trénovací množina, je dokončena jedna epocha. Celé sumě odchylek za jednu epochu se říká globální odchylka - chyba. Pokud je globální odchylka menší než námi požadovaná chyba, pak proces učení skončí.

Z výše popsaného je vidět, že proces učení není nic jiného, než přelévání informací ze vstupu na výstup a naopak. Při učení se v tzv. vstupním prostoru vytvářejí shluky bodů, které představují jednotlivé členy tříd, přičemž každý shluk představuje jednu třídu (obrázek 30) (Zelinka, 1999).



Obrázek 30: Třídy a jejich hranice

Například, mějme několik tříd: třída kol, třída aut, třída lodí, atd. Z každé z nich se vybere množina reprezentativních zástupců (vzorů pro učení) a všechny se popíší vhodným číselným způsobem ve formě vektorů. Pro každou množinu vektorů jedné třídy se vytvoří vzorový vektor, který bude zastupovat mateřskou třídu z reálného světa. V tomto okamžiku jsou vytvořeny skupiny vektorů popisující jednotlivé členy a jim příslušející představitele tříd ve formě vektorů. Například máme vektory,

které popisují vybrané členy z třídy kol a vektor, který říká „já jsem třída kol“. V tomto případě učení znamená to, že se učící algoritmus snaží najít takovou kombinaci vah, která umožní přiřazení vektoru třídy jejím členům. Jinak řečeno, hledá se taková kombinace vah, že pokud položíme ve vybavovací fázi na vstup vektor popisující objekt např. kolo, pak by se na výstupu měl objevit vektor, který říká „já jsem třída kol“. Mějme tedy například 5 různých tříd a pro každou třídu 20 reprezentativních členů, dohromady tedy 100 vstupních vektorů. Při učení se snažíme najít takové váhy, aby odezva sítě pro daný vstupní vektor - člen byla co nejvíce podobná vektoru jeho třídy. Vzhledem k tomu, že se neuronová síť učí vždy s nějakou chybou, nebude odezva odpovídat vždy přesně originálu a tím pádem dostaneme shluk bodů, okolo pozice originální třídy (obrázek 30).

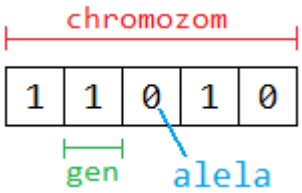
To, zda se naše síť naučí správným odezvám na dané podněty, závisí na více okolnostech, a to na množství vektorů a jejich velikosti, topologii sítě, odlišnosti charakteristických vlastností jednotlivých tříd, přípravě trénovací množiny, a jiných. Schopnost sítě přiřazovat jednotlivé vstupní členy daným třídám je dána tím, že síť v podstatě počítá vzdálenost daného členu od členů již přiřazených a na základě toho usuzuje, do jaké třídy daný vektor patří. To samozřejmě může někdy znamenat problém, který se dá odstranit tzv. pravděpodobnostní sítí (Barnsley, 1993).

7.3 Evoluční algoritmy – základní pojmy

Evoluční algoritmy jsou odvozeny podle evolučních procesů probíhajících v přírodě již po milióny roků. Evoluční proces funguje na principu, který odhalil v minulém století britský přírodozpytec (tehdy oficiální název pro přírodovědce) Charles Darwin. Princip spočívá v tom, že se cyklicky tvoří tzv. generace nových jedinců a z nich přežívají a vítězí v konkurenci při tvorbě nových potomků jen ti nejlepší. Tím je zajištěn vývoj daného druhu. Tyto principy byly „zmatematizovány" a používají se na řešení mnohých a mnohdy také

obtížně řešitelných problémů. Jeden z nejstarších evolučních algoritmů - *genetické algoritmy* jsou jejich typickým představitelem. Genetické algoritmy byly odvozeny na základě biologické genetiky a teorie evoluce, která ovlivňuje vývoj všeho živého na této planetě. Při vývoji jednotlivých druhů mají geny veliký význam. Základem všeho je DNA - deoxyribonukleová kyselina (obrázek 32), ve které je zakódován kompletní popis daného jedince. DNA je dlouhý molekulární řetězec tvořený čtyřmi odlišnými složkami. Uspořádání těchto složek reprezentuje genetický kód. V problematice genetických algoritmů se setkáváme s následujícími pojmy (Zelinka, 1999).

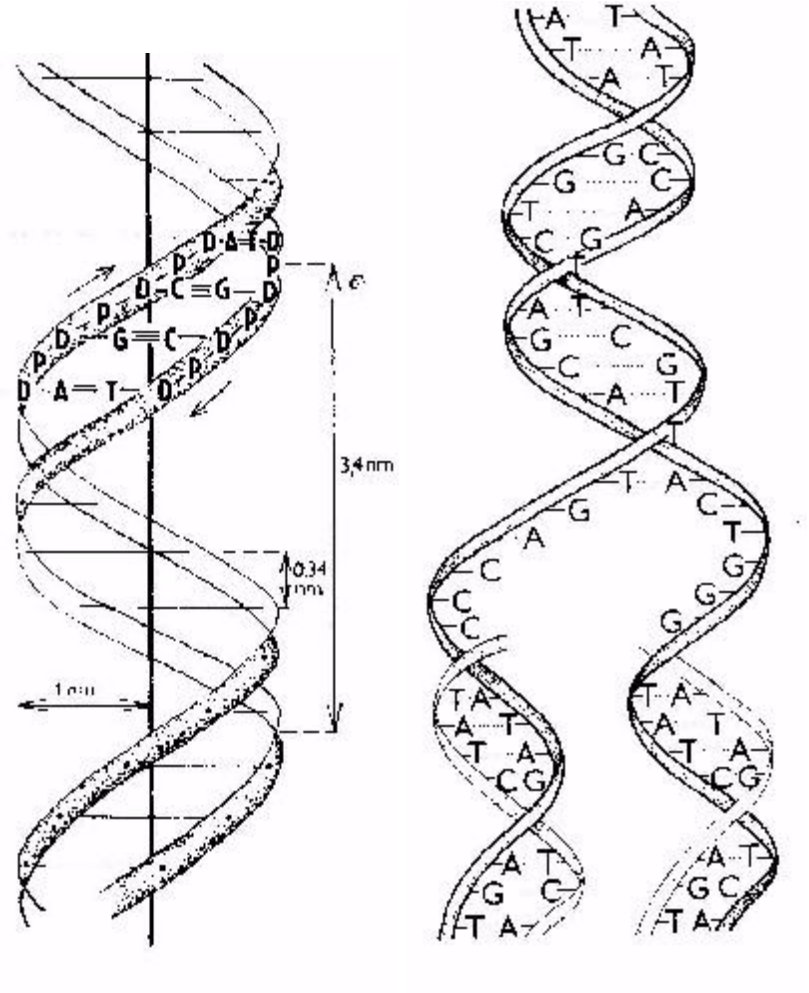
Chromozóm - je část DNA (obrázek 32a), která je stočena do záhybů. *Gen* - jsou jednotlivé části chromozómu. Volba jak řešení reprezentovat často závisí na samotném problému. Pro některé problémy je vhodné řešení reprezentovat jako binární řetězec, u jiných problémů jako pole reálných hodnot, jako textové řetězce či celé grafy. Této reprezentaci jedinců se říká *genotyp*. V souvislosti s genotypem se ještě uvádí i tzv. *fenotyp*, který je v podstatě fyzickým popisem genotypu. Např. jestliže je v binárním pojetí genotyp „0101“, pak fenotyp je jeho dekadická hodnota, a to 5. Geny mohou nabývat pouze jistých hodnot, jejichž obecné označení je *alela*. Vztah chromozom – gen – alela je uveden na obrázku 31.



Obrázek 31: Chromozom, gen, alela

V přírodě hraje DNA důležitou roli. Např. při rozmnožování člověka musí mít potomek 46 chromozómů - od každého z rodičů dostane polovinu s tím, že výsledná kombinace určuje pohlaví i ostatní vlastnosti. Potomek sdílí po svých rodičích genovou výbavu a s tím částečně i jejich schopnosti. Vzhledem k tomu, že v přírodě přežívají

jen ti nejschopnější, umírají nevyhovující potomci velmi rychle, dříve než stačí předat svým potomkům nevyhovující genetickou výbavu. Zde je na místě podotknout, že jejich výbava je nevýhodná jen pro aktuální okolní prostředí. V jiném prostředí by mohli být úspěšní. Vezměme si např. různé druhy vyhynulých pravěkých zvířat, která vyhynula jen proto, že se nestačila přizpůsobit okolním podmínkám. Kdyby se podmínky nezměnily, asi by žila dodnes.



Obrázek 32: a) DNA b) dělení DNA

Kritériem pro přežití druhu je mimo jiné to, zda se jedná o rozmnožování bezpohlavní či pohlavní (použito v genetických algoritmech v PC). Na první pohled se zdá, že bezpohlavní rozmnožování je jednodušší a méně náročné. Při pohlavním rozmnožování musí jedinci daného druhu vynaložit hodně energie na to, aby našli vhodného partnera a obstáli v konkurenci ostatních, což

činí tento způsob na první pohled náročným a nevhodným. Jak je tedy možné, že na světě je rozšířeno převážně pohlavní dělení? Odpověď je jednoduchá. Představte si, že ve stejném životním prostředí existují dva druhy (A a B), přičemž A se rozmnožuje nepohlavně (dělením) a B pohlavně. V případě A dostane nový jedinec kompletní genetickou kopii od svého rodiče i s případnou mutací. V případě B je zapotřebí ke vzniku nového jedince dvou rodičů. Každý z nich mu předá pouze část své genetické výbavy (je to ovlivněno způsobem dělení genů), díky čemuž se potomek od rodičů liší. Takto nově vzniklý jedinec může díky nové genetické výbavě lépe vyhovovat okolním podmínkám, což může po dlouhé době vést ke kvalitativním skokům ve srovnání s případem A, který je pak druhem B vytlačen.

Na stejném principu jako B pracují i genetické algoritmy v počítačových aplikacích. Náhodně je vygenerována množina chromozómů, ze které jsou vybírány dvojice (rodiče) na základě toho, jak dobře splňují kriteriální funkci. Potomci jsou tvořeni chromozómy z každého rodiče a nahrazují své rodiče. Stejně jako v přírodě, tak i v technických aplikacích hrají důležitou roli náhodné změny - mutace. To se v technice obvykle řeší pomocí generátoru náhodných čísel. Při používání genetických algoritmů v technických aplikacích se používají ještě další pojmy jako (Zelinka, 1999):

Objektivní funkce – je to funkce, kterou chceme minimalizovat. *Vhodnost (fitness)* – je číslo, které nám udává vhodnost nového potomka z hlediska kriteriální funkce (je to v podstatě matematický popis životního prostředí daného jedince), obvykle to bývá převrácená hodnota objektivní funkce. Čím je větší, tím je vhodnější daný potomek pro dané okolní podmínky. *Schéma* - množina genů v chromozómu, které mají jisté specifické hodnoty. V aplikacích se takováto množina chápe jako skupina genů, které je schopna za určitých podmínek vytvořit žádaný efekt. Obsah genu je variabilní. Nejjednodušší je, když je obsah v binární podobě, např. „01011101“, ale lze použít i dekadický zápis.

Vlastní algoritmus genetické optimalizace je cyklus, ve kterém jsou vytvářeni noví potomci, kteří reprezentují rodiče v dalším cyklu. Po každém cyklu se vyhodnocuje vhodnost potomka a na základě toho se buď pokračuje ohodnocením každého jedince v populaci, nebo výpočet končí. Cyklus neboli epocha se zde nazývá „generace“. Vlastní schéma genetického algoritmu je následující (Zelinka, 1999):

1) Navržení genetické struktury
2) Inicializace
3) Ohodnocení
    3.1) Konverze genotypu na fenotyp
    3.2) Ohodnocení objektivní funkce
    3.3) Konverze objektivní funkce na vhodnost
    3.4) Konverze vhodnosti na selekci rodičů
4) Volba rodičů
5) Reprodukce
6) Mutace
7) zpět na 3) nebo konec

Navržení genetické struktury.

Při navrhování genetické struktury je důležité, jak budou reprezentovány jednotlivé alely a jak budou v jednotlivých chromozomech rozmístěny. Reprezentace alel a rozmístění v genech ovlivňuje výkonnost příslušného algoritmu. Při volbě toho, jak budou v genu reprezentovány jednotlivé alely, se obvykle vychází z více možností. Nejrozšířenějším přístupem je reprezentace pomocí binárních hodnot. V takovém případě je chromozom tvořen řetězem alel a výpočet fenotypu je počítán jako dekadická hodnota binárního genu - řetězce. Tento přístup má mimo jiné dvě výhody. Tou první je, že binární kódování je vlastní všem PC a tudíž genetické algoritmy jsou poněkud jednodušší, tou druhou jsou teorémy, které pojednávají

o různých algoritmech a jsou lépe dokazatelné. V případě použití dekadických číslic je to trochu složitější. Lze však použít i různé jiné přístupy např. že každá alela bude nabývat jen alfanumerických parametrů.

Další nezanedbatelnou věcí je pozice genů v chromozomu. Standardní metoda je dělení chromozómů na dvě části. Každý potomek pak obsahuje po jedné části obou rodičů. Pokud jsou geny, které zastupují důležité kooperující parametry daleko od sebe, dochází při jejich dělení k znehodnocení jejich celkové účinnosti. Představte si, že oba rodiče mají geny, jejichž struktura popisuje chytrost potomka. Jednotlivé geny popisující „chytrost", jsou v chromozómu rozházeny a při jejich dělení půl na půl, může dojít k jejich rozdělení a výsledkem jsou průměrně chytří potomci. Pokud by však byly geny blízko sebe, pak by mohl jeden potomek obdržet „chytré“ geny a druhý bohužel jen ten zbytek. Měli bychom dva potomky, jeden by byl asi geniální a ten druhý hloupoučký. Z toho všeho plyne, že je vhodné umístnit důležité geny tak, aby pravděpodobnost jejich rozdělení byla co nejmenší.

Inicializace

Je to v podstatě jednoduchý a dá se říci i primitivní proces, kdy se náhodným způsobem (pomocí generátoru náhodných čísel), vygeneruje populace prvotních rodičů.

Ohodnocení

Tento krok se skládá ze čtyř etap, a to (Zelinka, 1999):

Konverze genotypu na fenotyp - je to primitivní proces, kdy se např. při použití binárních genů provede přepočet na dekadickou hodnotu. Pokud tedy máme chromozom „00001010“, pak jeho fenotyp je 12.

Ohodnocení objektivní funkce - v tomto kroku se počítá výsledek objektivní funkce. Obvykle se předpokládá, že daná funkce je

ovlivňována všemi geny zhruba stejně významně. Pokud by se vyskytl nějaký gen či skupina genů s vysokou „prioritou“ ovlivňování, pak by daný genetický algoritmus měl horší výkonnost.

Konverze objektivní funkce na vhodnost - je krok, který se skládá ze dvou operací. Když získáme objektivní funkci, pak ji musíme zkonvertovat na tzv. hrubou vhodnost, která se dá počítat mnoha způsoby. Například v **(**Barnsley, 1993) je použita rovnice (9):

$$f\left(E_{globalni}\right) = e^{-K\,E_{globalni}} \tag{9}$$

Tento způsob se ukázal být vhodný pro většinu problémů. Aby byl účinný, musí konstanta K nabývat vhodné hodnoty. Takto získanou hrubou vhodnost musíme zkonvertovat na tzv. jemnou vhodnost z následujícího důvodu. Pokud bychom použili hrubou vhodnost, dojde k tomu, že z prvotní vygenerované populace budou vítězové jen ti potomci, jejichž vhodnost významně převyšuje vhodnost ostatních, a tudíž by došlo k tvorbě pouze silných potomků. To v konečném důsledku znamená konec přirozeného výběru, protože by nebylo z čeho vybírat. Úpravou hrubé vhodnosti na jemnou se dosáhne toho, že jedincům s příliš vysokou vhodností je tato hodnota snížena, zatímco průměrné hodnoty zůstávají nedotčeny. S počtem jedinců v populaci souvisí ještě jeden problém - „vadní jedinci“. Pokud generujeme prvopočáteční množinu jedinců, pak je pravděpodobné, že se v této množině vyskytnou i jedinci naprosto nevyhovující, kteří při „páření“ mohou znehodnotit i materiál partnera, který je vložen do potomka. Abychom se tomuto vyhnuli, můžeme takovéto jedince nepřipustit k páření. Tím se však dostaneme do situace, která již byla popsána - zůstanou jen silní, vhodní jedinci a možnost výběru je pryč. Tomu se vyhneme tak, že při potřebě např. populace o 200 jedincích, vygenerujeme pro prvotní křížení 250 - 300 jedinců, tyto pokřížíme a z nich vybereme nejlepších 200. Tímto navýšením vlastně snížíme možnost vzniku a výběru defektního jedince a navíc v takto vybrané

části budou zcela jistě velké rozdíly mezi nejlepším a nejhorším jedincem a tak bude zajištěna možnost výběru.

Konverze vhodnosti na selekci rodičů - je operace, při které se určí, který jedinec a kolikrát bude vybrán k vytvoření potomka. Obvykle se tato operace provádí podle vztahu (10):

$$selekce = \frac{JV}{prumer\ JV} \tag{10}$$

kde *JV* znamená jemnou vhodnost.

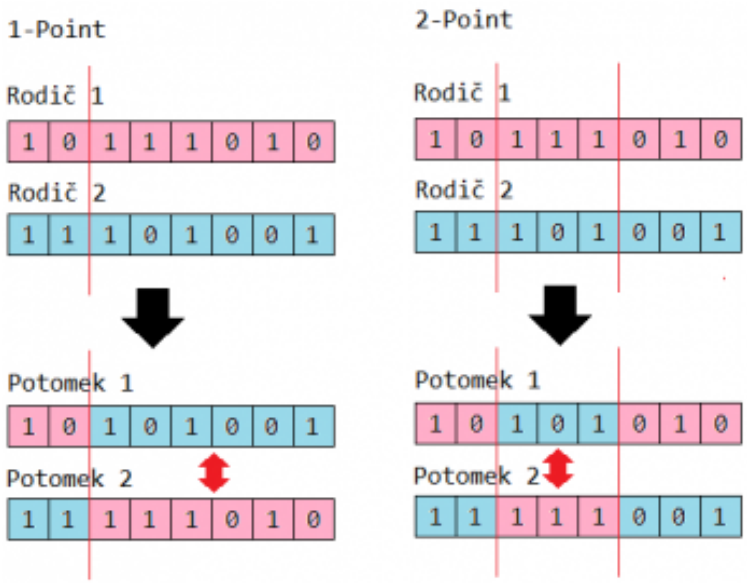
Volba rodičů

Tento problém byl zpočátku řešen tak, že se vytvořila ruleta s pozicemi, jejichž pravděpodobnosti jsou úměrné vhodnosti a poté probíhal náhodný výběr. Tato metoda se ukázala být nevhodnou, protože zde hrála velkou roli náhoda. Díky tomu docházelo k degradaci účinnosti vlastního genetického algoritmu. Mnohem lepší přístup je vytvořit pole, které bude nejprve vyplněno rodiči s nejvyšší selekcí, poté rodiči s menší selekcí až po jedince se selekcí menší jak 1, kteří jsou do pole vybíráni náhodně. Z tohoto pole se pak budou pářením vytvářet noví potomci - budoucí rodiče.

Reprodukce. Při reprodukci dochází k dělení chromozomů a vytváření potomků. Jak už bylo dříve řečeno, je dobré mít skupiny vhodných genů pohromadě a tím snížit pravděpodobnost jejich rozdělení a tak zkvalitnit jak genetický proces, tak množinu jedinců - budoucích rodičů.

Jedním z nejčastějších způsobů reprodukce je *křížení*. Většinou se provádí s jedním či se dvěma body, které se v chromozomu náhodně vyberou a geny se mezi těmito body vzájemně prohodí. Tento způsob křížení ilustruje obrázek 33. Dalším způsobem je např. aritmetické křížení, kdy hodnota v potomkovi je dána aritmetickou operací nad alelami obou rodičů. V případě binárních genů lze použít operátory

AND, OR, XOR. V případě genů s reálnými čísly se dá také použít křížení průměrem, který hodnoty potomků vypočte jako střední hodnotu aritmetickou či geometrickou.



Obrázek 33: Křížení (Zdroj http://www.root.cz/clanky/biologicke-algoritmy-3-evolucni-algoritmy/)

Mutace. Volba míry a pravděpodobnosti mutace genů je velmi ošemetný problém, který při necitlivém řešení může způsobit katastrofu v genetické množině. V podstatě se dají zvolit dvě základní cesty, jak přistupovat k mutaci. První cestou jsou mutace s malým účinkem, ale s vysokou četností výskytu. Protože tento typ mutací přináší malé změny a požadovány jsou změny větší (je potřeba nový materiál v genetické množině), je tento přístup nevhodný. Aby byl vliv mutací na průběh genetického algoritmu důrazný, je potřeba volit jejich menší četnost výskytu s větším účinkem (druhá metoda). Rozpoznání toho, jaká míra a četnost mutací genů je výhodná a jaká je škodlivá, je problém. Jako vodítko by mohl posloužit postup, při kterém se po každé mutaci vyhodnotí vhodnost a pokud je tato hodnota neustále klesající, je lepší se vrátit k množině chromozomů z „předmutačního“ období.

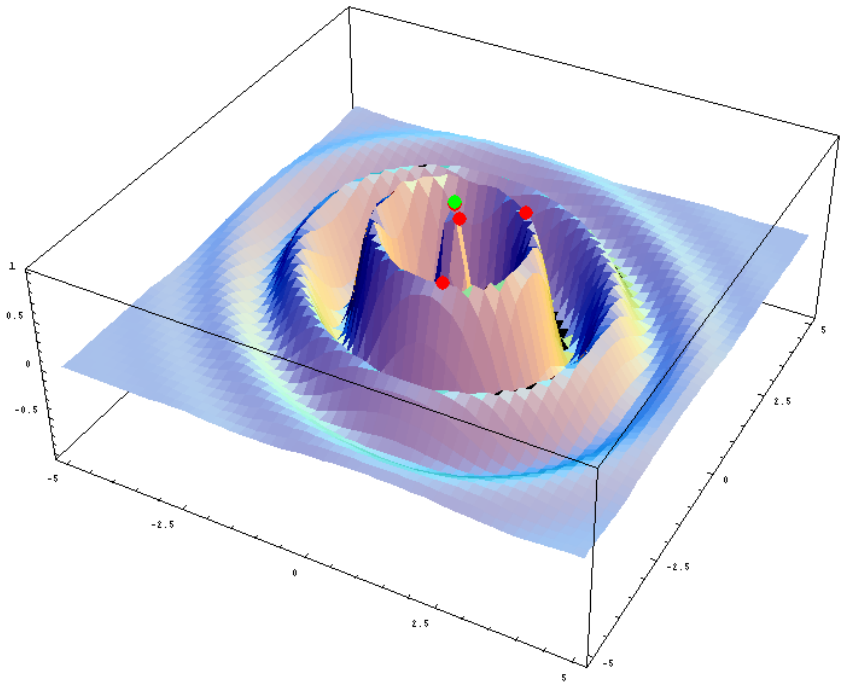
Vlastní mutace se provádí v cyklu pomocí náhodného generátoru čísel tak, že se porovnává náhodně vygenerované číslo s požadovanou četností výskytu mutací. Pokud je náhodné číslo menší, pak dojde k mutaci genu (jedná se o součin již vygenerovaného náhodného čísla

a dalšího náhodného čísla s tím, že se tento součin přičte k již existujícímu genu - gen mutuje). To lze samozřejmě modifikovat.

V oblasti genetických algoritmů existuje mnoho dalších přístupů k řešení mutací, výběru rodičů, atd. Vážnějším zájemcům bych doporučovala další literaturu jako (Davis, 1991; Goldberg, 1989, Back, 1997; Zelinka, 2008 aj).

Ilustrační příklad (Zelinka, 1999):

Jednoduchým a klasickým příkladem použití genetických algoritmů je při hledání extrému funkce (viz obrázek 34). Zde byly obě souřadnice reprezentovány jedním chromozomem s tím, že populaci tvořilo 50 rodičů. Tvorba potomků probíhala po dobu 30-ti generací. Vlastní chromozóm měl délku 15 bitů a mutační práh byl 0.5. V tomto případě bylo hledáno maximum, které bylo úspěšně nalezeno (zelený bod na vrcholu funkce v počátku).



Obrázek 34: Hledání extrému funkce pomocí genetických algoritmů

Kontrolní otázky:

1 Jaký je rozdíl mezi deduktivním a induktivním modelování?
2 Uveďte rozdíly mezi klasickým PC a umělou neuronovou sítí.
3 Jaké jsou základní typy umělých neuronových sítí? Vezměte v úvahu dělení podle různých kritérií.
4 Vysvětlete význam přenosové funkce neuronu.
5 Jak probíhá učení neuronové sítě?
6 Jaké jsou hlavní rysy evolučních algoritmů?
7 Jaké jsou základní evoluční operátory a jak pracují?

Úkoly k textu:

Zopakujte si všechny pojmy týkající se oblasti umělých neuronových sítí a evolučních algoritmů. V následujících kapitolách s nimi budeme pracovat.

Shrnutí obsahu kapitoly

Tato kapitola představuje úvod do problematiky přírodou inspirovaných metod řešení. Postupně se zde věnujeme induktivnímu a deduktivnímu modelování zejména základním pojmům z oblasti umělých neuronových sítí a evolučních algoritmů. Následující kapitoly se pak již jen na všechny zde vysvětlené pojmy odkazují.

Pojmy k zapamatování

- induktivní modelování;
- deduktivní modelování;
- umělé neuronové sítě;
- přenosová funkce neuronu;
- evoluční algoritmy
- křížení;
- mutace.

8 Princip řešení evolučními algoritmy

V této kapitole se dozvíte:

- Jaký je princip řešení úloh evolučními algoritmy.
- Jak lze graficky reprezentovat vývoj populace v čase.

Po jejím prostudování byste měli být schopni:

- Aplikovat evoluční algoritmy na řešení jednoduchých úloh.
- Graficky reprezentovat průběh řešení.

Klíčová slova této kapitoly:

Optimalizační algoritmy, evoluční algoritmy, fitness - kvalita jedince.

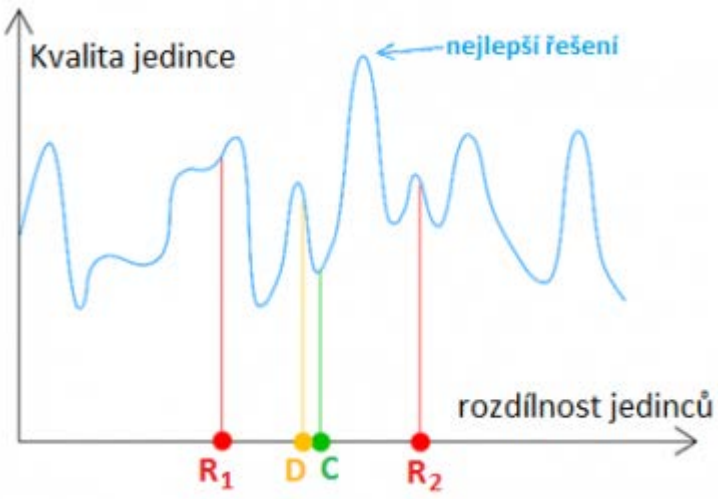
Průvodce studiem

Tato kapitola demonstruje praktické použití evolučních algoritmů při řešení úloh. Převážná část textu je převzata z (Chalupník, 2012).

Evoluční algoritmy patří mezi globálně optimalizační algoritmy. Principem jejich řešení je prohledávání prostoru za účelem najít optimální řešení – tedy takové řešení, které maximalizuje nebo minimalizuje známou funkci. Je-li prostor možných řešení omezený a malý, lze jednotlivá řešení vyhodnotit hrubou silou a vybrat z nich to nejlepší. Ovšem ve většině případů velikost prostoru možných řešení roste velmi rychle s velikostí problému. U takových problémů je nemožné prohledávat celý prostor řešení a tudíž je i nemožné najít optimální řešení v krátkém čase. Evoluční algoritmy tento problém umožňují řešit.

Každé možné řešení je v evolučních algoritmech reprezentováno jedincem v populaci. Populace je tedy množina jedinců – množina různých řešení. Postupem času se populace vyvíjí. Špatní jedinci vymírají, objevují se lepší jedinci, kteří je nahrazují. Počkáme-li dostatečně dlouho, populace bude tvořena dobrými jedinci, kteří reprezentují správná řešení problému.

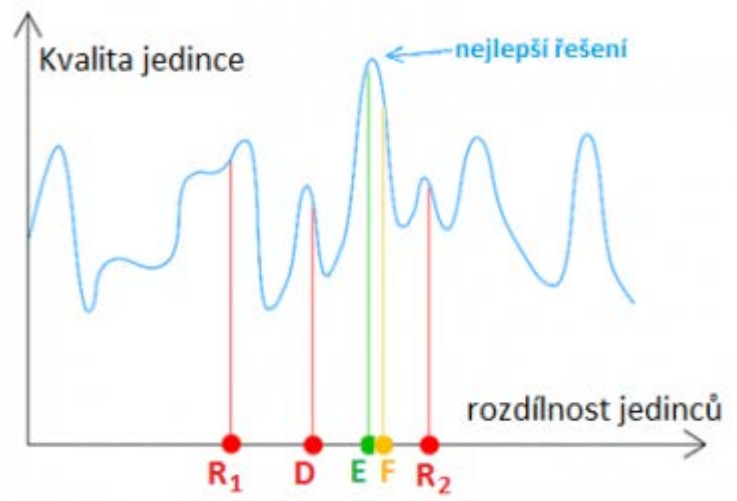
Aby byla evoluce funkční, jsou nezbytné tři věci. Za prvé je nezbytné umět ze dvou existujících řešení vytvořit nové „zprůměrované“, této operaci se říká křížení. Za druhé je nezbytné umět vytvořené řešení náhodně pozměnit, této operaci se říká mutace. A za třetí je nezbytné pro vybraného jedince vybrat vhodného jedince ke křížení, této operaci se říká selekce nebo přirozený výběr.



Obrázek 35: Zdroj http://www.root.cz/clanky/biologicke-algoritmy-1-evolucni-algoritmy/

Nyní se podívejme na graf na obrázku 35. Vodorovná osa reprezentuje rozdílnost jedinců (jejich genetický kód). Rozdílní jedinci jsou od sebe dále. Na svislé ose je jejich kvalita. Každý jedinec tak má jednoznačně přiřazenou svoji kvalitu. V grafu je také vyznačeno nejlepší možné řešení, jedná se o globální maximum celého prostoru. Jedinec $\mathbf{R_1}$ si vybere jedince $\mathbf{R_2}$, se kterým se chce zkřížit. Jejich zkřížením by měl vzniknout „zprůměrovaný“ jedinec mezi nimi **C**. Ten však nahodile zmutuje (posune se některým směrem) a vznikne tak v populaci nový

potomek **D** se svojí kvalitou. Míra mutace je základním parametrem evolučních algoritmů, což si dále ukážeme.



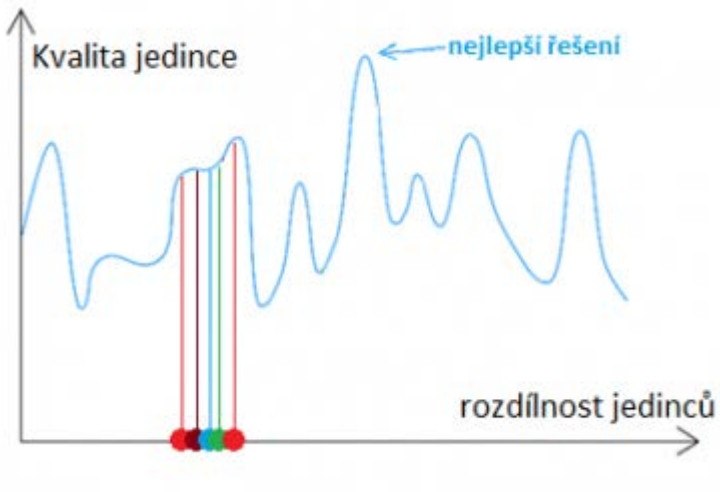
Obrázek 36: Zdroj http://www.root.cz/clanky/biologicke-algoritmy-1-evolucni-algoritmy/

V následující generaci se proces opakuje, viz obrázek 36. Dejme tomu, že jedinec **D** se chce zkřížit s jedincem $\mathbf{R_2}$. Zkřížením by vznikl jedinec **E**, náhodnou mutací se však toto řešení posune a vznikne nový jedinec v populaci **F** se svojí kvalitou. Zapojíme-li do tohoto procesu selekci, aby si jedinci vybírali ke křížení kvalitnější partnery, bude celá populace konvergovat k optimálnímu řešení. Špatní jedinci budou mít nižší pravděpodobnost, že si najdou partnera ke křížení a předají své geny.

V přírodě se za kvalitu považuje schopnost přežít. Jedinci si vybírají ke křížení takové partnery, kteří mají vyšší šanci přežít. Vlivem změny životního prostředí se uvedená křivka znázorňující kvalitu deformuje. Náhlá změna životního prostředí tak může vést k tomu, že kvalitní populace se v novém prostředí stane nekvalitní a vyhyne, anebo se populace, která již dosáhla maxima a dlouhou dobu se příliš nevyvíjela, náhle přesune k novému optimálnímu maximu. Nové druhy proto vznikají často při náhlých změnách životního prostředí. Kvalitu jedinců obvykle označujeme pojmem fitness.

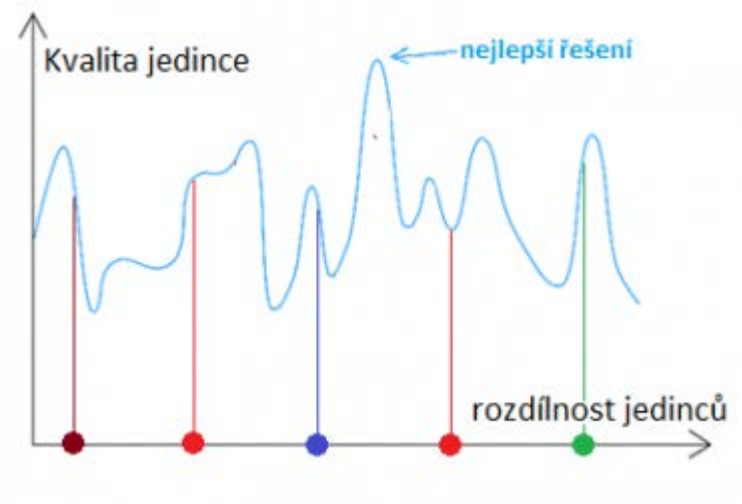
Cílem evoluční optimalizace je v krátkém čase dosáhnout optimálního řešení. Přitom však může docházet ke dvěma nežádoucím jevům. Za prvé populace může uváznout v lokálním maximu – nikoliv globálním. Tato situace je znázorněna na obrázku 37. Všichni jedinci v populaci jsou si geneticky velmi blízcí a pokrývají pouze rozsah odpovídající lokálnímu maximu. Dostat se z této pasti a nalézt globální maximum bude velmi obtížné, protože vlivem křížení mohou vznikat pouze „zprůměrovaní“ jedinci mezi svými rodiči.

Jedinou cestou jak se dostat z této pasti je vysoká mutace, která populaci rozšíří, přidá nové geny a posune nové potomky mimo tuto past. K tomuto nežádoucímu jevu dochází, nastavíme-li míru mutace příliš nízkou. V přírodě se tento problém řeší tak, že při křížení geneticky blízkých jedinců je zvýšena pravděpodobnost mutace. Jsou-li rodiče vzájemně příbuzní, míra mutace potomků je vyšší. Je nezbytné zajistit v populaci heterogenitu, protože vývoje lze dosáhnout pouze v heterogenní populaci!



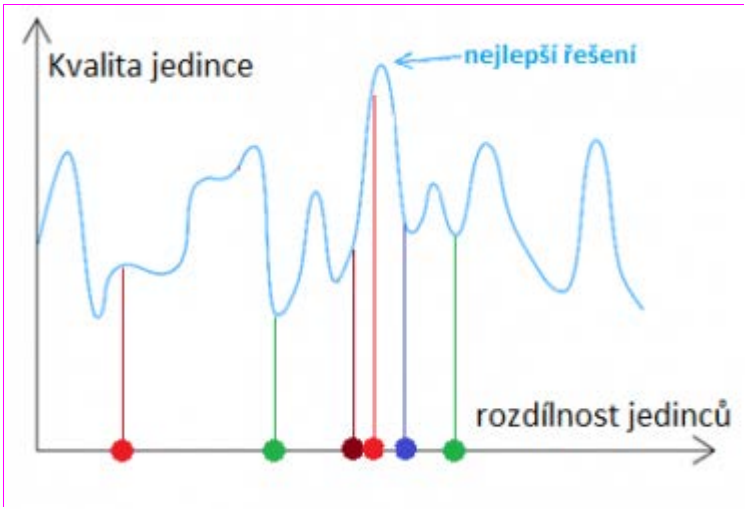
Obrázek 37: Uváznutí v lokálním maximu (zdroj http://www.root.cz/clanky/biologicke-algoritmy-1-evolucni-algoritmy/)

Druhým problémem je nahodilé nebo uniformní rozmístění populace. Najít optimální řešení bude opět velmi obtížné, je totiž málo pravděpodobné, že vznikne dobrý jedinec. Tato situace je znázorněna na obrázku 38. K tomuto jevu dochází, pokud je mutace příliš vysoká.



Obrázek 38: Zdroj http://www.root.cz/clanky/biologicke-algoritmy-1-evolucni-algoritmy/)

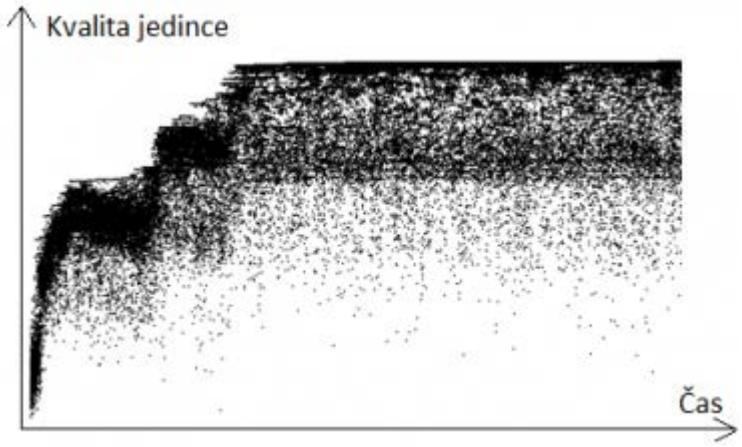
Optimální situace je znázorněna na obrázku 39. Hustota populace směrem k maximu roste, vzdálenějších jedinců je méně. Pravděpodobnost, že nastane v genu mutace, obvykle volíme mezi 2 - 4 %.



Obrázek 39: Zdroj http://www.root.cz/clanky/biologicke-algoritmy-1-evolucni-algoritmy/

Nyní se podívejme na graf na obrázku 40, který znázorňuje typický vývoj populace v čase. Na vodorovné ose je čas, na svislé ose je kvalita jedinců. Každý bod v grafu znázorňuje jedince. Populaci v konkrétním čase získáme svislým řezem v grafu. Na začátku jsou v populaci nekvalitní jedinci. Postupem času se populace vyvíjí. V okamžiku, když se objeví nový kvalitnější jedinec, zbytek populace k němu konverguje.

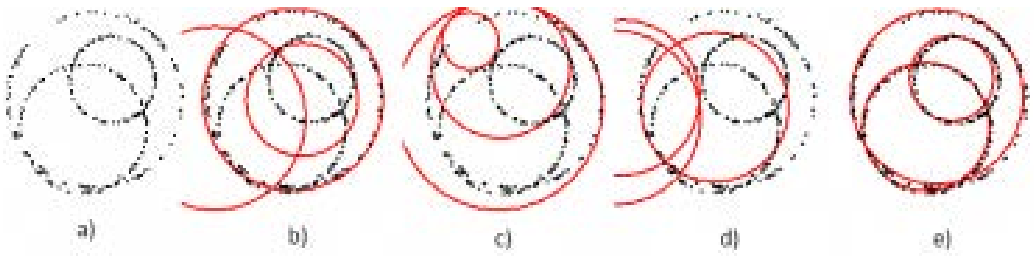
Populace se často vyvíjí skokově. Pokud po delší dobu nedochází ke zlepšení populace, signalizuje to, že populace uvízla v lokálním extrému.



Obrázek 40: (převzato z http://www.root.cz/clanky/biologicke-algoritmy-1-evolucni-algoritmy/)

Ilustrační příklad (Chalupník, 2012)

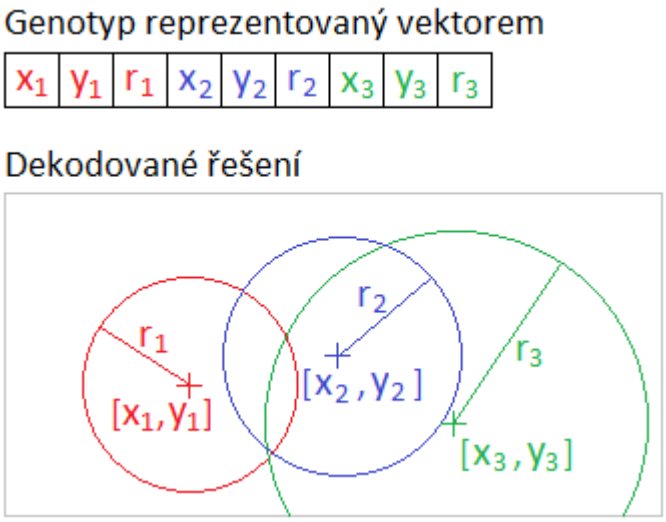
Na vstupu máme 300 bodů v dvourozměrném prostoru. Tyto body jsou znázorněny na obrázku 41 a). Cílem je najít tři kružnice, které nejlépe odpovídají těmto bodům. Každá kružnice je definována třemi parametry – souřadnice *x* a *y* a poloměr *r*. Pro tři kružnice je tak každé řešení dáno 9 parametry. Různá řešení jsou uvedena na obrázku 41 b) – e)**.** Některá řešení jsou lepší, jiná horší. Velmi dobré řešení je znázorněno na obrázku e)**.**



Obrázek 41: Zobrazení různých řešení příkladu

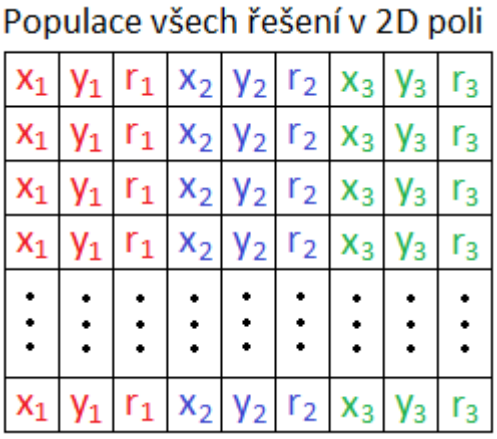
Reprezentace řešení

Trojici kružnic můžeme reprezentovat vektorem o devíti hodnotách – viz obrázek 42.



Obrázek 42: Genotyp a fenotyp (dekódované řešení)

Všechny jedince v populaci lze reprezentovat v 2D poli tak, jak je to uvedeno na obrázku 43.



Obrázek 43: Populace jedinců

Výpočet fitness

Dále potřebujeme umět vyhodnotit kvalitu řešení. K tomuto účelu definujeme funkci fitness následovně: Na vstupu je vektor

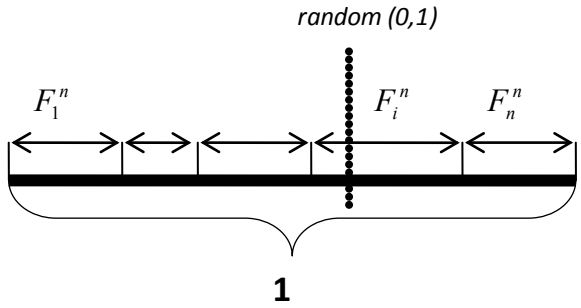
o 9 hodnotách, který obsahuje genotyp jedince. Výstupem je kvalita jedince. Pro každý bod se vypočte vzdálenost od 1. kružnice *d1*, vzdálenost od 2. kružnice *d2* a vzdálenost od 3. kružnice *d3*. Z těchto vzdáleností se vybere nejmenší hodnota. Najdeme tak ke každému bodu vzdálenost od nejbližší kružnice. Je-li vzdálenost od kružnice nulová, přičte se hodnota 1.0, se zvyšující se vzdáleností se bude přičítat nižší hodnota. V ideálním případě – když všech 300 bodů bude ležet na kružnicích – bude mít fitness svoji maximální hodnotu 300.0.

Výběr rodičů

Z praktických důvodů je vhodné, aby numerické hodnoty fitness byly z intervalu (0,1), proto se zavádí tzv. normalizovaná fitness. Normalizovaná fitness F_i^n má pro *i.* jedince z populace obsahující celkem *n* jedinců tvar (11):

$$F_i^n = \frac{F_i}{\sum_{i=1}^{n} F_i}, \qquad (11)$$

kde F_i je vypočítaná fitness *i.* jedince. Výběr rodičů pak probíhá následovně (Kvasnička, 2000), viz obrázek 44.



Obrázek 44: Výběr rodičů.

Jednotková úsečka je rozdělena na úseky podle velikosti normalizovaných fitness hodnot jedinců z populace. Náhodně vygenerované číslo z intervalu (0,1) představuje polohu na úsečce (je reprezentované hrubou přerušovanou čarou) a podle této polohy určuje chromozóm. Z konstrukce této „rulety“ vyplývá: čím větší je fitness

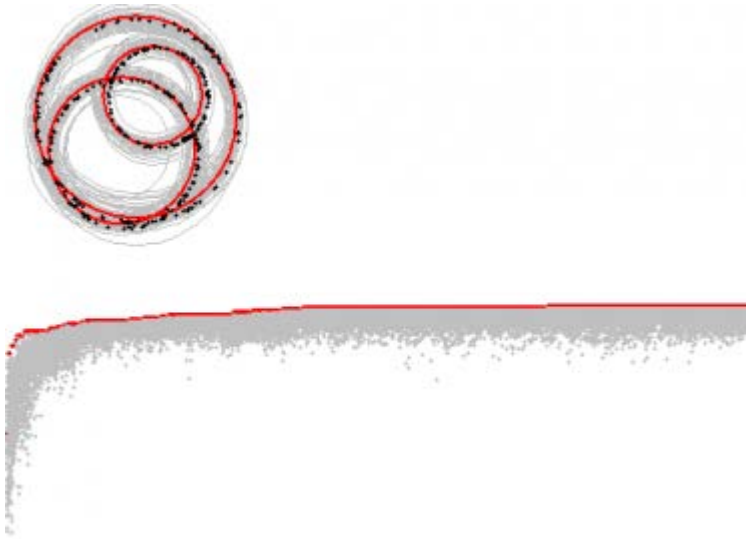
chromozómu, tím větší je i pravděpodobnost jeho výběru. Lepší jedinci tak budou mít vyšší šanci, že budou vybráni.

Průběh algoritmu, křížení a mutace

Nejprve se populace inicializuje náhodnými hodnotami od 0 do 100. Pro všechny jedince vypočteme jejich fitness a znormalizujeme je. Tím kvalitu každého jedince dostaneme do rozsahu od 0 do 1. V dalším cyklu budeme vytvářet novou populaci, která nahradí stávající. V každé iteraci vybereme náhodné 2 jedince ke křížení a vytvoříme místo nich 2 potomky. S pravděpodobností CROSS_RATE (hodnotu obvykle volíme mezi 0.7 – 1.0) provedeme křížení, jinak pouze zkopírujeme do nové populace již existující vybrané jedince. Křížení provedeme tak, že každou hodnotu genotypu potomka vybereme náhodně buď z prvního rodiče, nebo druhého rodiče. Po křížení se provede mutace obou potomků. V každém potomku procházíme jeho genotyp (9 hodnot) a s pravděpodobností MUTATION_RATE hodnotu v genotypu změníme tak, že přičteme náhodnou hodnotu danou normálním rozdělením. S vyšší pravděpodobností dojde k malé změně hodnoty a s nižší pravděpodobností dojde k vyšší změně hodnoty.

Po vytvoření nové populace se provede operace, která se v evolučních algoritmech označuje jako elitismus. Je to umělé zachování nejlepších jedinců. Zaručuje, že nejlepší jedinec se v populaci zachová. To umožňuje rychleji nalézt optimální řešení. Do nové populace na první řádek zkopírujeme nejlepšího jedince v minulé generaci.

Vykreslování průběhu řešení je uvedeno na obrázku 45. Nejprve se šedou barvou vykreslí všechny kružnice všech řešení. Potom se červenou barvou vykreslí z nich nejlepší řešení, které je uloženo v populaci jako první řádek 2D pole. Dále se vykreslí černou barvou všech 300 bodů a nakonec graf vývoje populace v čase. Jak je z obrázku 45 vidět, kružnice byly nalezeny správně. Velmi rychle (během pár generací) bylo dosaženo dobrého řešení, které se postupem času ještě zpřesňovalo.



Obrázek 45: Průběh řešení

Korespondenční úkol:

Pomocí evolučních algoritmů implementujte vybraný problém. Jeho zadání nejprve konzultujte s vyučujícím.

Shrnutí obsahu kapitoly

Tato kapitola demonstruje praktické použití evolučních algoritmů při řešení úloh.

Pojmy k zapamatování

- optimalizační algoritmy
- evoluční algoritmy
- fitness - kvalita jedince

9 Logické neurony McCullocha a Pittse

V této kapitole se dozvíte:

- Jaké Boolovské funkce mohou simulovat sítě složené z logických neuronů.
- Jak se chová neuronová síť s logickými neurony.

Po jejím prostudování byste měli být schopni:

- Vysvětlit, jaké Boolovské funkce mohou simulovat logické neurony.
- Vysvětlit, jak se chová neuronová síť s logickými neurony.

Klíčová slova této kapitoly:

Logické neurony, syntaktický strom, Boolovské funkce, třívrstvá neuronová síť.

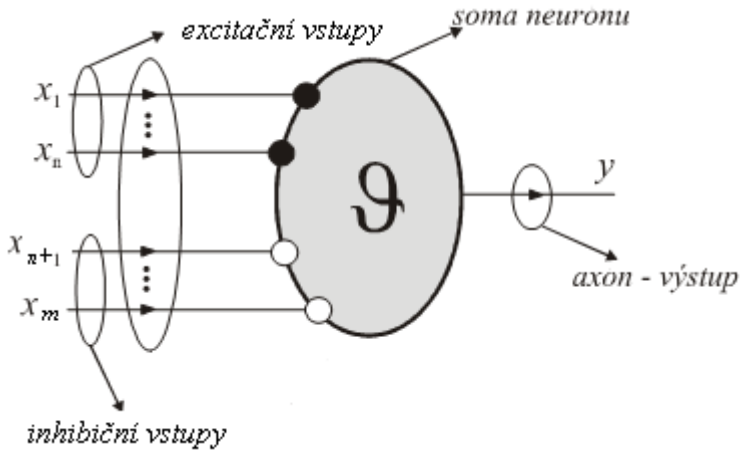
Průvodce studiem

V této kapitole si ukážeme, že neuronové sítě jsou mocným modelovým prostředkem, například, že jsou schopné simulovat libovolnou Boolovu funkci. (Kvasnička, 1997). Logické neurony jsou schopny simulovat logické spojky, které jsou charakterizovány jako lineární oddělitelné (např. disjunkce, konjunkci, implikaci a negaci). Logické spojky, které nejsou lineární oddělitelné (např. ekvivalence a exkluzivní disjunkce XOR) nemohou být simulované logickým neuronem. Tato skutečnost naznačuje, že logický samotný neuron není univerzální výpočetní zařízení, existují úlohy (např. logická spojka XOR), které nejsou řešitelné pomocí logického neuronu.

9.1 Logický neuron

V publikaci Warrena McCullocha a Waltera Pittsa (McCulloch1943) „*A logical calculus of the ideas immanent to nervous activity*" z r. 1943 bylo poprvé uvedeno, že neuronové sítě jsou mocným modelovým prostředkem, např. sítě složené z logických neuronů mohou simulovat Boolovské funkce.

Elementární jednotkou McCullochových a Pittsových neuronových sítí je logický neuron (výpočetní jednotka), přičemž stav neuronu je binární (tj. má dva stavy, 1 nebo 0). Takový logický neuron lze interpretovat jako jednoduché elektrické zařízení - relé. Předpokládejme, že dendritický systém logického neuronu obsahuje jak excitační vstupy (popsáno binárními proměnnými x_1, x_2, ..., x_n, které zesilují odezvu), tak inhibiční vstupy (popsáno binárními proměnnými x_{n+1}, x_{n+2}, ..., x_m, které zeslabují odezvu), viz obrázek 46.



Obrázek 46: Znázornění McCullochova a Pittsova neuronu, který obsahuje dendritický systém pro vstupní (excitační nebo inhibiční) aktivity, axon pro výstup neuronové aktivity. Soma (tělo neuronu) je charakterizována prahovým koeficientem ϑ.

Aktivita logického neuronu je jednotková, pokud vnitřní potenciál neuronu definovaný jako rozdíl mezi sumou excitačních vstupních aktivit a inhibičních vstupních aktivit je větší nebo roven prahu - ϑ, v opačném případě je nulová (12)

$$y = \begin{cases} 1 & \left(x_1 + \ldots + x_n - x_{1+n} - \ldots - x_m \geq -\vartheta\right) \\ 0 & \left(x_1 + \ldots + x_n - x_{1+n} - \ldots - x_m < -\vartheta\right) \end{cases} \tag{12}$$

Pomocí skokové funkce *s* můžeme aktivitu vyjádřit takto (13):

$$y = s\left(\underbrace{x_1 + \ldots + x_n - x_{1+n} - \ldots - x_m}_{\xi} + \vartheta\right) \tag{13}$$

Tento vztah pro aktivitu logického neuronu můžeme rovněž interpretovat tak, že excitační aktivity vstupují do neuronu přes spoje, které jsou ohodnoceny jednotkovým váhovým koeficientem (*w*=1), zatímco inhibiční aktivity vstupují do neuronu přes spoje se záporným jednotkovým váhovým koeficientem (*w*=-1). Potom aktivitu logického neuronu můžeme vyjádřit takto (14):

$$y = s\left(\underbrace{w_1 x_1 + \ldots + w_m x_m}_{\xi} + \vartheta\right) = s\left(\sum_{i=1}^{m} w_i x_i + \vartheta\right) \tag{14}$$

Jednoduchá implementace elementárních Booleovských funkcí disjunkce, konjunkce, implikace a negace je znázorněna na obrázku 47.

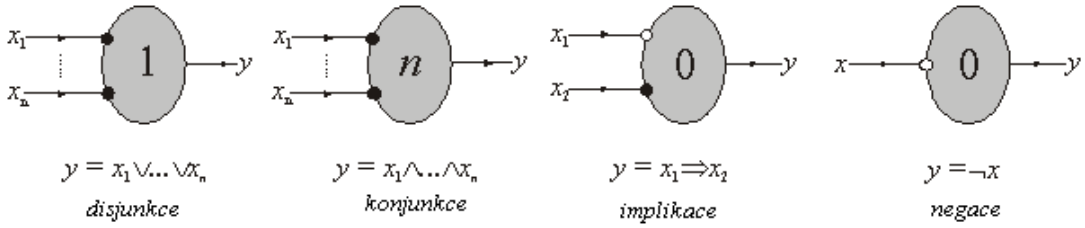
Jako ilustrační příklad uvedeme funkci disjunkce pro *n* = 2, použitím vzorců z definice logického neuronu dostaneme (15):

$$y_{OR}\left(x_1, x_2\right) = s\left(x_1 + x_2 - 1\right) \tag{15}$$

#	x_1	x_2	$y_{OR}(x_1,x_2)$	$x_1 \vee x_2$
1	0	0	$s(-1)$	0
2	0	1	$s(0)$	1
3	1	0	$s(0)$	1
4	1	1	$s(1)$	1

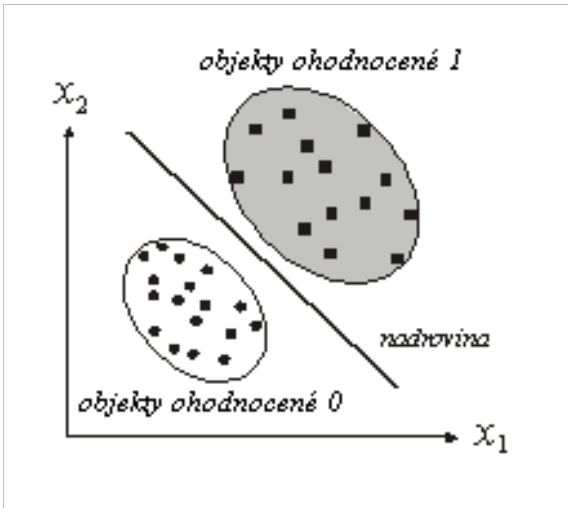
Tabulka 8: Binární Booleova funkce disjunkcie

Z tabulky 8 vyplývá, že Booleova funkce y_{OR} simuluje Booleovu funkci disjunkcie.



Obrázek 47 Logické neurony pro implementaci Booleovských funkcí disjunkce, konjunkce, implikace a negace. Excitační spoje jsou znázorněny plným kroužkem, inhibiční prázdným kroužkem.

Můžeme si položit otázku, jaké Boolove funkce je schopen logický neuron vyjádřit? Tato otázka se dá poměrně jednoduše vyřešit pomocí geometrické interpretace výpočtu probíhajícího v logickém neuronu. Výpočetní funkce logického neuronu rozděluje prostor vstupů na dva poloprostory pomocí roviny $w_1x_1 + w_2x_2 + ... + w_nx_n = \vartheta$ pro koeficienty $w_i \in \{0, +1, -1\}$. Říkáme, že Booleova funkce $f(x_1, x_2, ..., x_n)$ je *lineárně separovatelná*, pokud existuje taková rovina $w_1x_1 + w_2x_2 + ... + w_nx_n = \vartheta$ která separuje prostor vstupních aktivit tak, že v jedné části prostoru jsou vrcholy ohodnoceny 0, zatímco v druhé části prostoru jsou vrcholy ohodnoceny 1 (viz obrázek 48).

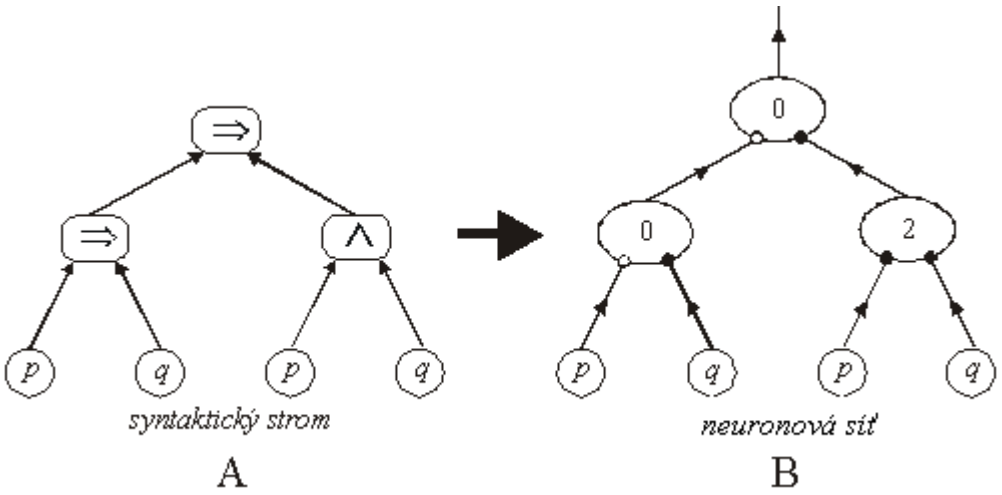


Obrázek 48: Schematické znázornění pojmu lineární separovatelnost, kde kulaté a čtvercové objekty jsou separované nadrovinou $w_1x_1+...+w_nx_n- \vartheta = 0$ tak, že v jednom poloprostoru jsou objekty jednoho druhu, zatímco ve druhém poloprostoru jsou objekty druhého druhu.

Logický neuron je schopen simulovat jen takové Boolovské funkce, které jsou lineární separovatelné.

9.2 Neuronová síť s logickými neurony

Každou Booleovskou funkci lze reprezentovat pomocí syntaktického stromu, který reprezentuje její rekurentní výstavbu inicializovanou Booleovskými proměnnými, viz obrázek 49, diagram A. Syntaxe stromu je důležitá, pokud hledáme podformule dané formule, každý uzel stromu specifikuje podformuli: nejníže položené uzly reprezentují triviální podformule *p* a *q*, následující dva vrcholy reprezentují podformule $p \Rightarrow q$ a $p \wedge q$, nejvýše položený vrchol - kořen stromu - reprezentuje samotnou formuli $(p \Rightarrow q) \Rightarrow (p \wedge q)$.



Obrázek 49: (A) Syntaktický strom Booleovské funkce (výrokové formule) $(p \Rightarrow q) \Rightarrow (p \wedge q)$. Koncové vrcholy stromu reprezentují Booleovské proměnné (výrokové proměnné) *p* a *q*, vrcholy z následujících vrstev jsou přiřazeny spojkám implikace a konjunkce. Vyhodnocování tohoto stromu probíhá postupně zdola nahoru. (B) Neuronová síť obsahující logické neurony spojek, které se vyskytují v příslušném sémantickém stromu diagramu A.

Pomocí syntaktického stromu snadno sestrojíme neuronovou síť tak, že jednotlivé vrcholy reprezentující Booleovské elementární funkce (logické spojky) nahradíme příslušnými logickými neurony podle obrázku 47. Na obrázku 49 je znázorněna tato konstrukce neuronové

sítě pro formuli $(p \Rightarrow q) \Rightarrow (p \wedge q)$. Tento postup konstrukce neuronové sítě shrneme pomocí následující věty:

Každá Booleovská funkce může být vyjádřena pomocí „neuronové sítě" složené z logických neuronů vyjadřujících logické spojky.

Tato věta patří mezi základní výsledky McCullocha a Pittse (McCulloch, 1943), kteří ukázali, že pomocí několika jednoduchých logických neuronů je možné sestrojit neuronovou síť, která simuluje danou Booleovu funkci. Můžeme tedy hovořit o tom, že neuronové sítě s logickými neurony mají univerzální charakter v doméně Booleovských funkcí.

Aktivita *i*-tého neuronu v neuronové síti je vyjádřená zobecněním vzorce (14)

$$x_i = t\left(\sum_j w_{ij} x_j + \vartheta_i\right) \tag{16}$$

kde sumace probíhá nad všemi neurony, které v neuronové síti předcházejí *i*-tý neuron a váhové koeficienty w_{ij} jsou definované takto (17):

$$w_{ij} = \begin{cases} 1 & (spoj\ j \to i\ má\ exitačný\ charakter) \\ -1 & (spoj\ j \to i\ má\ inhibičný\ charakter) \\ 0 & (spoj\ j \to i\ neexistuje) \end{cases} \tag{17}$$

To znamená, že v neuronové síti jsou váhové koeficienty fixně stanoveny a jsou určeny topologií syntaktického stromu specifikující Booleovu funkci. Architektura neuronové sítě, která je sestrojena pro danou Booleovu funkci může být podstatně zjednodušená na tzv. 3 - vrstvou neuronovou síť, tj. obsahuje vrstvu vstupních neuronů (které jen kopírují vstupní aktivity, nejsou výpočetními jednotkami), vrstvu skrytých neuronů a poslední vrstva obsahuje výstupní neuron. Tato architektura je minimalistická a už zřejmě nemůže být zjednodušena. Ukážeme jak sestrojit takovou neuronovou síť pro danou Booleovu funkci.

Jednoduchým zobecněním logických spojek lze ukázat, že logický neuron je schopen simulovat i Booleovu funkci, která obsahuje konjunkce proměnných nebo jejich negace $x_1 \wedge ... \wedge x_n \wedge \neg x_{n+1} \wedge ... \wedge \neg x_m$, viz obrázek 50. Tato Booleova funkce se rovná 1 pouze pro $x_1 = ... = x_n = 1$ a $x_{n+1} = ... = x_m = 0$, ve všech ostatních pravdivostních kombinacích argumentů je její hodnota 0 (nepravdivý výrok)

$$val_\tau\left(x_1 \wedge ... \wedge x_n \wedge \neg x_{n+1} \wedge ... \wedge \neg x_m\right) = \begin{cases} 1 & \left(pre\ \tau = \tau_0\right) \\ 0 & \left(pre\ \tau \neq \tau_0\right) \end{cases} \qquad (18)$$

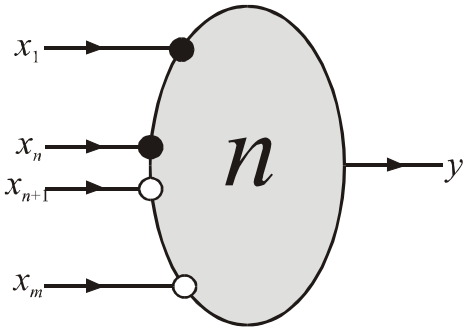
kde $\tau_0 = \left(x_1/1, ..., x_n/1, x_{n+1}/0, ..., x_m/0\right)$ je specifikace pravdivostních hodnot proměnných. Snadno se přesvědčíme o tom, že tato klauzule je simulována logickým neuronem znázorněným na obrázku 50; jeho výstupní aktivita je určena formulí (19):

$$y = s\left(x_1 + ... + x_n - x_{n+1} - ... - x_m - n\right) \qquad (19)$$

Funkční hodnota této funkce se rovná 1 pouze tehdy, když

$$x_1 + ... + x_n - x_{n+1} - ... - x_m \geq n \qquad (20)$$

Tato podmínka je dosažena pouze tehdy, když prvních *n* vstupních (excitačních) aktivit se rovná 1 a dalších (*m* - *n*) vstupních (inhibičních) aktivit se rovná 0.



Obrázek 50: Logický neuron simulující klauzuli, která obsahuje konjunkci konečného počtu výrokových proměnných nebo jejich negací, $y = x_1 \wedge ... \wedge x_n \wedge \neg x_{n+1} \wedge ... \wedge \neg x_m$.

Ilustrační příklad:

V teorii Booleovských funkcí je dokázána důležitá vlastnost, podle níž každá Booleovská funkce může být přepsáno do ekvivalentního disjunktivního tvaru (Kvasnička 2006). Jako ilustrační příklad použití této vlastnosti, studujme Booleovu funkci, jejíž funkční hodnoty jsou zadány tabulkou 9.

#	x_1	x_2	x_3	$y = f(x_1, x_2, x_3)$	*klauzula*
1	0	0	0	0	-
2	0	0	1	0	-
3	0	1	0	1	$\overline{x}_1 \wedge x_2 \wedge \overline{x}_3$
4	0	1	1	1	$\overline{x}_1 \wedge x_2 \wedge x_3$
5	1	0	0	0	-
6	1	0	1	1	$x_1 \wedge \overline{x}_2 \wedge x_3$
7	1	1	0	0	-
8	1	1	1	0	-

Tabulka 9: Zadané funkční hodny

V této tabulce v řádcích 3, 4 a 6 jsou jednotkové funkční hodnoty (funkce je pro tyto tři hodnoty proměnných pravdivá). Tvar funkce určené pouze tabulkou její funkčních hodnot je pak následující (21):

$$y = f(x_1, x_2, x_3) = (\overline{x}_1 \wedge x_2 \wedge \overline{x}_3) \vee (\overline{x}_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \overline{x}_2 \wedge x_3) \quad (21)$$

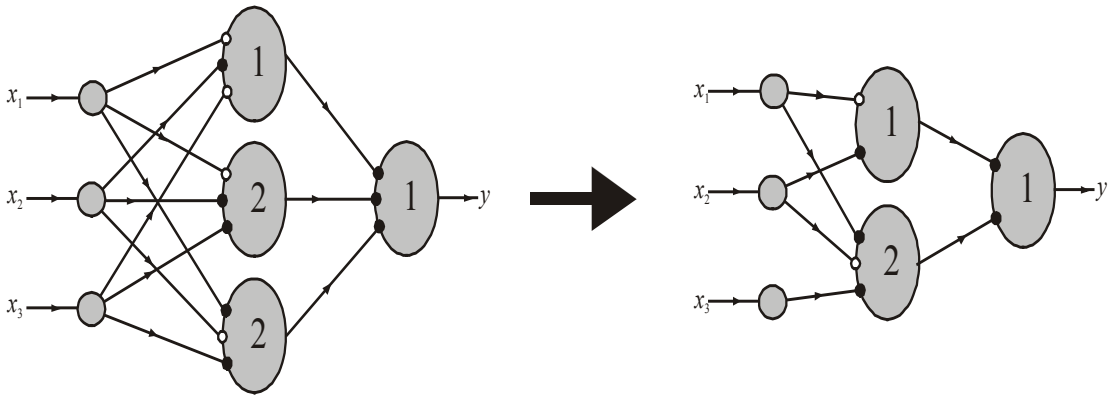
Tato Booleova funkce může být zjednodušená tak, že se zjednoduší první a druhá klauzule (22)

$$(\overline{x}_1 \wedge x_2 \wedge \overline{x}_3) \vee (\overline{x}_1 \wedge x_2 \wedge x_3) = \overline{x}_1 \wedge x_2 \wedge \underbrace{(\overline{x}_3 \vee x_3)}_{1} = \overline{x}_1 \wedge x_2 \qquad (22)$$

Potom

$$y = f(x_1, x_2, x_3) = (\overline{x}_1 \wedge x_2) \vee (x_1 \wedge \overline{x}_2 \wedge x_3) \qquad (23)$$

Klauzuli $x_1^{(\tau)} \wedge x_2^{(\tau)} \wedge ... \wedge x_n^{(\tau)}$ můžeme vyjádřit jedním logickým neuronem, viz obrázek 50. Výstupy z těchto neuronů spojíme do disjunkce pomocí neuronu reprezentujícího disjunkce (viz obrázek 47). Pak neuronová síť, reprezentující Booleovu funkci (21) má tvar znázorněný na obrázku 51.



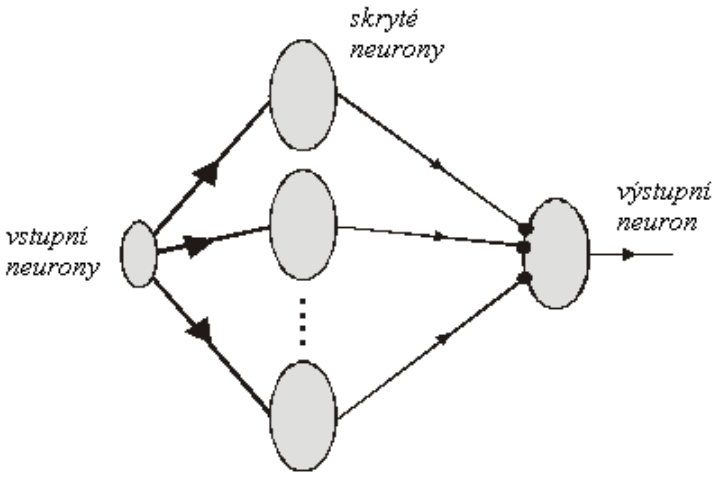
Obrázek 51: Třívrstvá neuronová síť, která simuluje Booleovu funkci zadanou primárně tabulkou 9. Skryté neurony reprezentují jednotlivé klauzule z tabulky 9, jejich disjunkce je realizována pomocí výstupního neuronu. Tato Neuronová síť může být zjednodušená tak, že první dvě klauzule se spojí do jedné jednodušší klauzule, viz (22) a (23).

Výsledky z tohoto ilustračního příkladu shrneme pomocí následující věty.

Libovolnou Booleovskou funkci f lze simulovat pomocí 3 vrstvé neuronové sítě.

Obecný tvar 3-vrstvé neuronové sítě je znázorněný na obrázku 52. Třívrstvé neuronové sítě obsahující logické neurony lze tedy považovat za univerzální výpočetní zařízení pro doménu Booleovských funkcí. Tento výsledek avizoval moderní výsledek neuronových sítí z počátku 90. let minulého století, podle něhož mají třívrstvé dopředné neuronové sítě se spojitou aktivační funkcí vlastnost univerzálního aproximátoru. Byl zde sice předveden jednoduchý postup, jak tuto síť systematickým způsobem sestrojit pro libovolnou Booleovu funkci. Bohužel, problém minimálnosti takto sestrojené sítě není tímto řešený. Obecně může

existovat síť, která pro danou Booleovu síť obsahuje méně logických (skrytých) neuronů než ta, která byla sestrojena navrženým postupem.



Obrázek 52: Schématické znázornění 3-vrstvé neuronové sítě - zleva doprava. První je vstupní vrstva obsahující vstupní neurony, které nejsou výpočtové elementy, ale jen formálně reprezentují vstupní aktivity. Druhá vrstva obsahuje skryté neurony, které reprezentují jednotlivé klauzule dané Booleovy funkce. Třetí (poslední) vrstva obsahuje výstupní neuron, který provádí disjunkce aktivit ze skrytých neuronů.

Kontrolní otázky:

1. Jaké Boolovské funkce mohou simulovat logické neurony?
2. Jaká je topologie McCullochova a Pittsova neuronu?
3. Jaké Boolovské funkce je schopen simulovat logický neuron?
4. Jaké funkce je schopna simulovat třívrstvá neuronová síť?

Korespondenční úkol:

Uveďte logické neurony pro implementaci Booleovských funkcí disjunkce, konjunkce, implikace a negace. Excitační spoje znázorněte plným kroužkem, inhibiční prázdným kroužkem. Řešte numericky i graficky. Uveďte definice logického neuronu pro implementaci uvedených Booleovských funkcí.

Shrnutí obsahu kapitoly

V této kapitole jsme si ukázali, že logické neurony jsou schopny simulovat logické spojky, které jsou charakterizovány jako lineární oddělitelné (např. disjunkce, konjunkci, implikaci a negaci). Libovolnou Booleovskou funkci lze simulovat pomocí třívrstvé neuronové sítě.

Pojmy k zapamatování

- logické neurony
- syntaktický strom
- Boolovské funkce
- třívrstvá neuronová síť

10 Princip činnosti umělého neuronu

V této kapitole se dozvíte:

- Jaká je struktura biologického neuronu a jak jej modelujeme - formální neuron.
- Jaký je princip adaptačního algoritmu perceptronu.

Po jejím prostudování byste měli být schopni:

- Vysvětlit model formálního neuronu s biasem.
- Vysvětlit, jaký je princip adaptačního algoritmu perceptronu

Klíčová slova této kapitoly:

Biologický neuron, formální neuron, vnitřní potenciál, bias, , perceptron, učení s učitelem.

Průvodce studiem

V této kapitole se stručně seznámíte se základním matematickým modelem biologického neuronu, tj. formálním neuronem. Z tohoto modelu budeme dále vycházet, a proto je nutné, abyste jeho pochopení věnovali zvýšenou pozornost.

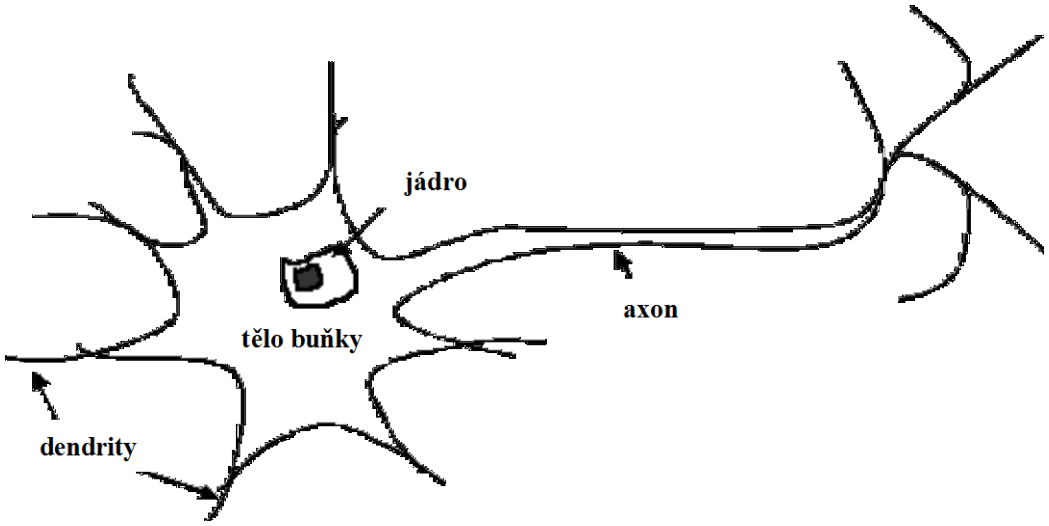
Dále je zde představen perceptron, který lze považovat za nejjednodušší neuronovou síť s jedním pracovním neuronem. Na jeho adaptačním algoritmu si vysvětlíme proces učení s učitelem.

Celá kapitola je převážně zprecována podle (Fausett, 1994).

10.1 Biologický neuron

Základním stavebním funkčním prvkem nervové soustavy je nervová buňka, *neuron*. Neurony jsou samostatné specializované buňky, určené

k přenosu, zpracování a uchování informací, které jsou nutné pro realizaci životních funkcí organismu. Struktura neuronu je schematicky znázorněna na obrázku 53.



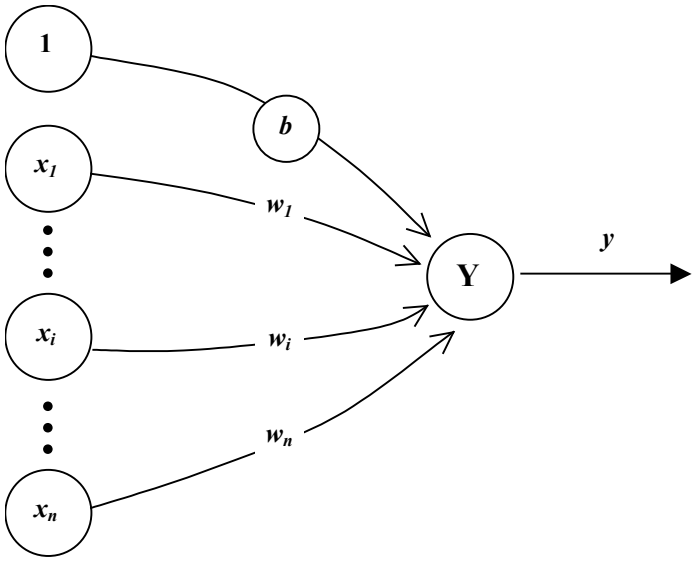
Obrázek 53: Biologický neuron

Neuron je přizpůsoben pro přenos signálů tak, že kromě vlastního těla (somatu), má i vstupní a výstupní přenosové kanály: dendrity a axon. Z axonu odbočuje řada větví (terminálů), zakončených blánou, která se převážně stýká s výběžky (trny), dendritů jiných neuronů. K přenosu informace pak slouží unikátní mezineuronové rozhraní, synapse. Míra synaptické propustnosti je nositelem všech význačných informací během celého života organismu. 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 soma 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 nebo při zapomínání se synaptické spoje přerušují.

10.2 Formální neuron

Základem matematického modelu neuronové sítě je *formální neuron*. Jeho struktura je schematicky zobrazena (Fausett, 1994) na obrázku 54. Formální neuron *Y* (dále jen neuron) má obecně *n* reálných *vstupů,* které modelují dendrity a určují vstupní vektor $x = (x_1, ... , x_n)$. Tyto vstupy jsou ohodnoceny reálnými *synaptickými váhami* tvořícími vektor $w = (w_1, ... , w_n)$. Ve shodě s neurofyziologickou motivací mohou být synaptické váhy i záporné, čímž se vyjadřuje jejich inhibiční charakter.



Obrázek 54: Formální neuron s biasem

Vážená suma vstupních hodnot *y_in* představuje *vnitřní potenciál* neuronu *Y* (24):

$$y\_in = \sum_{i=1}^{n} w_i x_i \qquad (24)$$

Bias může být do vztahu včleněn přidáním komponent $x_0 = 1$ k vektoru $\boldsymbol{x}$, tj. $\boldsymbol{x} = (1, x_1, x_2, ..., x_n)$. Bias je dále zpracováván jako jakákoliv jiná

váha, tj. $w_0 = b$. Vstup do neuronu *Y* je pak dán následujícím vztahem (25):

$$y\_in = \sum_{i=0}^{n} w_i x_i = w_0 + \sum_{i=1}^{n} w_i x_i = b + \sum_{i=1}^{n} w_i x_i \qquad (25)$$

Hodnota vnitřního potenciálu *y_in* po dosažení hodnoty *b* indukuje *výstup (stav) y* neuronu *Y*, který modeluje elektrický impuls axonu. Nelineární nárůst výstupní hodnoty $y = f(y\_in)$ při dosažení hodnoty potenciálu *b* je dán *aktivační* (*přenosovou*) *funkcí f*. Nejjednodušším typem přenosové funkce je *ostrá nelinearita*, která má pro neuron *Y* tvar (26):

$$f(y\_in) = \begin{cases} 1 & \text{pokud } y\_in \geq 0 \\ 0 & \text{pokud } y\_in < 0 \end{cases} \qquad (26)$$

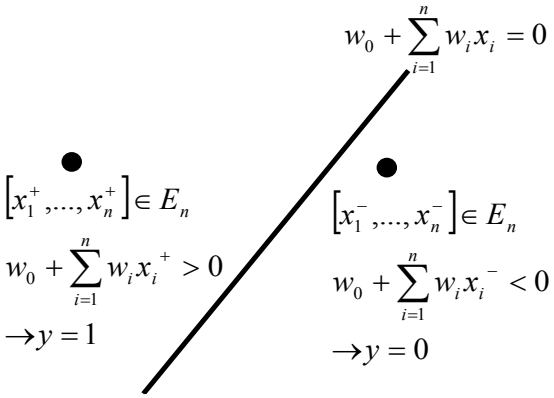
Pokud místo váhového biasu, pracujeme s fixním prahem θ pro aktivační funkci, pak má přenosové funkce *ostrá nelinearita* pro neuron *Y* tvar (27):

$$f(y\_in) = \begin{cases} 1 & \text{pokud } y\_in \geq \theta \\ 0 & \text{pokud } y\_in < \theta \end{cases}, \text{ kde } y\_in = \sum_{i=1}^{n} w_i x_i . \qquad (27)$$

Ilustrační příklad:

K lepšímu pochopení funkce jednoho neuronu nám pomůže geometrická představa načrtnutá na obrázku 55. Vstupy neuronu chápeme jako souřadnice bodu v *n - rozměrném Euklidovském vstupním prostoru* E_n. V tomto prostoru má rovnice nadroviny (v E_2 přímka, v E_3 rovina) tvar: $w_0 + \sum_{i=1}^{n} w_i x_i = 0.$ Tato nadrovina dělí vstupní prostor na dva poloprostory. Souřadnice bodů $[x_1^+,...,x_n^+]$, které leží v jednom poloprostoru, splňují následující nerovnost: $w_0 + \sum_{i=1}^{n} w_i x_i^+ > 0.$ Body $[x_1^-,...,x_n^-]$ z druhého poloprostoru pak vyhovují relaci s opačným relačním znaménkem: $w_0 + \sum_{i=1}^{n} w_i x_i^- < 0.$ Synaptické váhy $\boldsymbol{w} = (w_0,..., w_n)$

chápeme jako koeficienty této nadroviny. Neuron *Y* tedy klasifikuje, ve kterém z obou poloprostorů určených nadrovinou leží bod, jehož souřadnice jsou na vstupu, tj. realizuje *dichotomii* vstupního prostoru. Neuron *Y* je *aktivní*, je-li jeho stav *y = 1* a *pasivní*, pokud je jeho stav *y = 0*.



Obrázek 55: Geometrická interpretace funkce neuronu

10.3 Perceptron

Autorem této nejjednodušší neuronové sítě je Frank Rosenblatt (r. 1957). Za typický perceptron je považována jednoduchá neuronová síť s *n* vstupy ($x_1, x_2, ..., x_n$) a jedním pracovním neuronem (obrázek 54) spojeným se všemi svými vstupy. Každému takovému spojení je přiřazena váhová hodnota ($w_1, w_2, ..., w_n$). Signál přenášený vstupními neurony je buď binární (tj. má hodnotu 0 nebo 1), nebo bipolární (tj. má hodnotu -1, 0 nebo 1). Výstupem z perceptronu je pak *y = f* (*y_in*). Aktivační funkce *f* má tvar (28):

$$f(y\_in) = \begin{cases} 1 & pokud\ y\_in > \theta \\ 0 & pokud - \theta \le y\_in \le \theta \\ -1 & pokud\ y\_in < -\theta \end{cases}, \qquad (28)$$

kde θ je libovolný, ale pevný práh aktivační funkce *f.*

Váhové hodnoty jsou adaptovány podle adaptačního pravidla perceptronu tak, aby diference mezi skutečným a požadovaným výstupem byla co nejmenší. Adaptační pravidlo perceptronu je mnohem silnější než Hebbovo adaptační pravidlo.

Adaptační algoritmus perceptronu (Fausett, 1994)

Krok 0. Inicializace vah w_i *(i = 1* až *n)* a biasu *b malými náhodnými čísly.*

Přiřazení inicializační hodnoty koeficientu učení α *(0* $< \alpha \leq$ *1).*

Krok 1. Dokud není splněna podmínka ukončení výpočtu, opakovat kroky (2 až 6).

Krok 2. Pro každý tréninkový pár **s:**t (tj. vstupní vektor **s** a příslušný výstup **t)**, provádět kroky (3 až 5).

Krok 3. Aktivuj vstupní neurony:

$x_i = s_i$.

Krok 4 Vypočítej skutečnou hodnotu na výstupu:

$$y\_in = b + \sum_i x_i w_i;.$$

$$y = \begin{cases} 1 & pokud\ y\_in > \theta \\ 0 & pokud - \theta \leq y\_in \leq \theta \\ -1 & pokud\ y\_in < -\theta \end{cases}$$

Krok 5 Aktualizuj váhové hodnoty a bias pro daný vzor

jestliže $y \neq t$,

$w_i(new) = w_i(old) + \alpha\, t\, x_i$ *(i = 1* až *n).*

$b(new) = b(old) + \alpha\, t.$

jinak

$w_i(new) = w_i(old)$

$b(new) = b(old)$

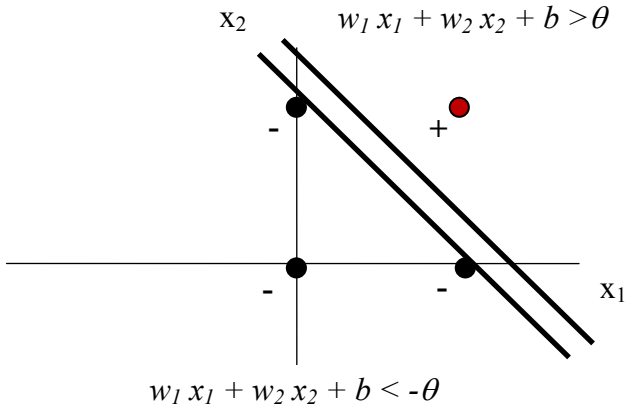
Krok 6. Podmínka ukončení:

jestliže ve *2. kroku* již nenastává žádná změna váhových hodnot, stop; jinak, pokračovat.

Aktualizaci podléhají pouze ty váhové hodnoty, které neprodukují požadovaný výstup *y*. To znamená, že čím více tréninkových vzorů má korektní výstupy, tím méně je potřeba času k jejich tréninku. Práh aktivační funkce je pevná nezáporná hodnota θ. Tvar aktivační funkce pracovního neuronu je takový, že umožňuje vznik pásu pevné šířky (určené hodnotou θ) oddělujícího oblast pozitivní odezvy od oblasti negativní odezvy na vstupní signál. Předcházející analýza o zaměnitelnosti prahu a biasu zde nemá uplatnění, protože změna θ

mění šířku oblasti, ne však její umístění. Místo jedné separující přímky tedy máme pás určený dvěma rovnoběžnými přímkami (viz obrázek 56):

1. Přímka separující oblast pozitivní odezvy od oblasti nulové odezvy na vstupní signál; tato hraniční přímka má tvar: $w_1 x_1 + w_2 x_2 + b > \theta$.
2. Přímka separující oblast nulové odezvy od oblasti negativní odezvy na vstupní signál;. tato hraniční přímka má tvar: $w_1 x_1 + w_2 x_2 + b < -\theta$.



Obrázek 56: Hraniční pás pro logickou funkci „AND“ po adaptaci algoritmem perceptronu.

Úkoly k zamyšlení:

Vytvořte geometrickou interpretaci funkce jednoho neuronu pro Booleovské funkci OR ve 2 rozměrném Euklidovském prostoru. Vstupy neuronu jsou souřadnice bodu v E_2.

Korespondenční úkol:

Vytvořte počítačový program pro realizaci adaptačního algoritmu perceptronu.

Shrnutí obsahu kapitoly

V této kapitole jsme se stručně seznámíte se základním matematickým modelem biologického neuronu, tj. formálním neuronem. Dále je zde představen perceptron, který lze považovat za nejjednodušší neuronovou síť s jedním pracovním neuronem. Na jeho adaptačním algoritmu jsme si vysvětlili proces učení s učitelem.

Pojmy k zapamatování

- biologický neuron
- formální neuron
- perceptron
- vnitřní potenciál
- bias
- učení s učitelem.

11 Rozpoznávání vzorů a klasifikace

V této kapitole se dozvíte:

- Co je podstatou klasifikace objektů do tříd.
- Proč je klasifikace jednou z oblastí, kde se neuronové sítě velmi dobře uplatňují.
- Jaký je princip dvouhodnotové a vícehodnotové klasifikace.
- Jaké metody soft-computingu lze uplatnit při klasifikaci.

Po jejím prostudování byste měli být schopni:

- Vysvětlit, jaké metody soft-computingu lze uplatnit při klasifikaci.
- Vysvětlit, jaké jsou možnosti klasifikace za použití umělých neuronových sítí.

Klíčová slova této kapitoly:

Klasifikace, dvouhodnotová klasifikace, klasifikátor.

Průvodce studiem

V této kapitole se budeme zabývat problematikou rozpoznávání vzorů a klasifikace. Zaměříme se především na metody soft-computingu a možnosti jejich klasifikace. Důraz zde bude kladen na uplatnění umělých neuronových sítí. Kapitola je z velké části napsána podle (Zelinka, 1999).

11.1 Klasifikace

Klasifikace je činnost, při které se posuzované objekty zařazují do příslušných tříd. Z matematického hlediska lze tyto činnosti nazývat funkční aproximací (Zelinka, 1999).

Klasifikace je jednou z oblastí, kde se neuronové sítě velmi dobře uplatňují. Klasifikace znamená v podstatě ohodnocení daného problému a jeho zařazení do příslušné třídy. Je to aktivita, kterou člověk provádí dnes a denně, aniž by si to uvědomoval. Například když v obchodě vybíráme mezi dvěma výrobky, musíme rozhodnout, který je horší a který lepší (nyní není podstatné z jakého hlediska). Pokud chceme jít do kina, obvykle předem oklasifikujeme film a na základě tohoto výsledku se rozhodneme. Pokud se nad otázkou klasifikace v našem životě zamyslíte, jistě sami najdete mnoho podobných příkladů.

Klasifikace není vždy jednoduchá záležitost. Mnohdy se hranice, na jejichž základě se zařazují klasifikované členy do tříd, překrývají. Proč se tedy na klasifikaci používají neuronové sítě? Důvodů je hned několik. Neuronová síť svá rozhodnutí provádí prakticky ihned i v případě, že vstupních informací popisující daný problém je poměrně dost. Co to znamená „... dost vstupních informací ..." ? Představte si, že máme na základě vstupních informací binárního charakteru rozhodnout, kam zařadíme daný výrobek. Pokud nám zmíněný výrobek popisují dvě vstupní proměnné, pak máme $2^2 = 4$ kombinace, na základě kterých můžeme tento výrobek ohodnotit. Ale co když náš výrobek popisují ne dvě, ale tři (8 kombinací), čtyři (16 kombinací), pět (32 kombinací) a více vstupních proměnných? Jistě každý z vás najde ve svém životě situaci, kdy se nemohl rozhodnout nebo se rozhodl špatně a přitom vycházel z malého počtu vstupních proměnných. To neuronové síti nehrozí, pokud je dobře natrénována, neboť její výhodou je, že zpracuje vše, co jí předložíme. To znamená, že pokud předložíme síti pro klasifikaci např. 30 či více vstupů popisující daný objekt na základě např. sonarového odrazu, pak je síť zpracuje. Co ale udělá člověk? Začne vypouštět některé vstupní proměnné, a pokud není dostatečné zkušený, pak může vypustit i proměnné, které jsou pro klasifikaci důležité. Jistě nyní namítáte, že neuronová síť vlastně jen zpracuje vše, co se jí předloží. Není to tak zcela pravda, protože neuronové sítě umí pomocí tzv. neurální citlivostní analýzy určit, která informace je vhodná

pro zpracování a která není (Barnsley, 1993). Zatím jsme ale pouze hovořili o binárním rozhodování. Představte si, že by každá z výše uvedených 30 možností mohla nabývat hodnot ne 0 či 1, ale z intervalu <0, 1>. Tím se náš problém klasifikace podstatně zkomplikuje. Klasifikaci tedy můžeme provádět pomocí binární či spojité funkce pro zařazení do jedné a více tříd. V případě použití neuronových sítí se tedy problém klasifikace dělí na klasifikaci dvouhodnotovou a vícehodnotovou. Pro klasifikaci jsou „důležité“ výstupní neurony, protože právě ty nám určují, do jaké třídy patří momentálně hodnocený objekt.

11.1.1 **Dvouhodnotová klasifikace (binární)**

Dvouhodnotová klasifikace patří mezi nejjednodušší klasifikace vůbec. V jejím případě se v podstatě rozhoduje mezi „černou“ a „bílou“, „ano“ a „ne“ či 1 a 0. Tento typ klasifikace lze realizovat tak, že neurony v neuronové síti mají skokovou funkci, která umožňuje nastavení výstupu neuronu pouze ve dvou výše zmíněných hodnotách. Na realizaci tohoto typu klasifikace v podstatě stačí jeden výstupní neuron (pouze v případě, že pracujeme s jednou třídou), který nabývá hodnot 0 a 1. Dvouhodnotovou klasifikaci lze také provést pomocí logistické funkce, která je sice kontinuální, nicméně není problém naučit síť tak, aby výstupní neurony dosahovaly binárních hodnot. V případě použití jakékoliv kontinuální funkce musíme počítat s tím, že výstupní neurony nikdy nedosáhnou přesně požadovaných hodnot, ale že se vždy naučí s nějakou chybou. Jinými slovy, místo výstupní hodnoty 1 nám daný neuron poskytne např. 0.9, 0.88, a naopak. I tyto výsledky jsou jistě akceptovatelné, nicméně, při vhodných či spíše nevhodných podmínkách, nám může síť odpovědět hodnotou 0.7, 0.65,... Z těchto důvodů je dobré, určit prahy, které jasně vymezují co je 0 a co 1. Existence prahů nabývá při vícehodnotové klasifikaci na významu ještě více.

11.1.2 **Vícehodnotová klasifikace**

Tato klasifikace je složitější než klasifikace binární. Při ní, jak už z názvu vyplývá, existuje možnost zařadit klasifikovaný objekt do více než dvou tříd. To lze realizovat jak spojitou funkcí (obrázek 28), tak funkcí binární. V případě, že použijeme spojitou funkci, pak pro zařazení objektu do dané třídy stačí, když na výstupu sítě je jen jeden neuron, jehož funkce je rozdělena což znamená, že např. třídě A přiřadíme hodnotu 0 - 0.3, třídě B 0.3 - 0.7 a třídě C 0.7 - 1. Toto dělení může být i hrubé či jemné podle potřeb uživatele. Mějme však na paměti, že díky nepřesnosti učení sítě může dojít k špatnému zařazení objektu, což v některých aplikacích nemusí mít vážné následky, zatímco v jiných aplikacích to může mít následky katastrofální. Tomu se lze vyhnout pomocí tzv. „vícenásobné“ dvouhodnotové klasifikace. Při této klasifikaci už nevystačíme s jedním výstupním neuronem (ten je schopen nabýt pouze dvou hodnot), ale musíme použít více neuronů, obvykle tolik, kolik je tříd. V tomto případě hodnota 1 znamená, že daný objekt patří do třídy zastoupené příslušným neuronem a hodnota 0, že tam nepatří.

Pokud použijeme pro vícehodnotovou klasifikaci binární neurony, pak lze pro zařazení objektu do dané třídy použít dvojí způsob kódování - přímý a kombinovaný. Přímý způsob je přiřazení jedné třídy jednomu neuronu, tj. pokud je stav neuronu 0, pak daný objekt do třídy nepatří a pokud jeho stav 1, pak daný objekt do třídy patří. Jiné třídě se přiřadí další neuron. Kombinované kódování je založeno na binární povaze hodnot neuronu - je to v podstatě přepočet z binárních hodnot. Jinými slovy, pokud máme např. 8 tříd, pak pro jejich realizaci stačí 3 neurony (2^3=8) na rozdíl od přímého kódování, kde bychom potřebovali celkem 8 neuronů. Na druhou stranu větší počet neuronů je nevýhodný, protože znamená delší čas učení a tím i větší finanční náklady.

Pokud však máme hodnotit objekt, který nepatří do žádné regulérní třídy, pak abychom se tomuto problému vyhnuli, je dobré přidat další třídu „zamítnutí“ a do trénovací množiny přidat představitele této třídy, aby síť dokázala v budoucím rozhodování na takové objekty reagovat.

Pokud bychom tak neučinili, pak to, že všechny neurony jsou v 0 ještě nemusí nutně znamenat, že hodnocený objekt nepatří do žádné třídy. Může to být např. člen některé okrajové třídy, jehož charakteristický vzorek nebyl v trénovací množině zastoupen.

11.2 Metody soft computingu pro rozpoznávání vzorů a klasifikaci

Pro účely rozpoznávání vzorů (Pattern Recognition) či klasifikaci může být uplatněna spousta algoritmů, jež lze obecně rozdělit do dvou kategorií (Ranawana, 2006): algoritmy využívající pravidla (*rule-based*) a algoritmy vycházející z metod umělé inteligence (*computational intelligence based*). Klasifikátory využívající pravidla jsou obvykle konstruovány expertem, jenž definuje pravidla na základě možné kombinace vstupů. To je v kontrastu s klasifikátory postavenými na bázi umělé inteligence, kde expert pouze vytvoří základní rámec pro interpretaci dat. Adaptační algoritmy takovýchto systémů jsou pak zodpovědné za vytvoření pravidel pro správné zařazení vstupů. Metody umělé inteligence pro rozpoznávání vzorů lze charakterizovat třemi směry vývoje: *fuzzy rozhodování*, *umělé neuronové sítě* a *samoorganizace*. V kombinaci s preprocesingem, metrikami a lineárními transformacemi pak můžeme získat plně funkční rozpoznávací systémy. Ke zpracování signálů lze s výhodou využít postupy z *fuzzy logiky* a *teorie fuzzy množin*. Tím jsou míněny především algebry pro podporu fuzzy rozhodování, jejich operátory a funkce. Z teorie fuzzy množin využíváme transformaci reálných hodnot na stupně příslušnosti k fuzzy množinám a hovoříme o tzv. *fuzzifikaci*. Fuzzy systémy jsou obvykle založeny na elementárních konjunkcích, které jsou chápány jako příčiny a ze kterých usuzujeme na následky pomocí pravdivosti fuzzy pravidel. I při pevné struktuře pravidel dosahujeme mnohotvárnosti navrženého řešení, protože právě jednotlivé pravdivosti pravidel jsou vhodné parametry pro doladění systému.

Druhým silným nástrojem pro realizaci rozpoznávání vzorů jsou *neuronové sítě* se spojitým chováním. Ty historicky vznikly rozšířením nespojitých prahových lineárních obvodů při splnění oprávněného požadavku na spojitost a ohraničenou citlivost. Výsledkem je vícevrstvá perceptronová síť (*MLP – Multi Layer Perceptron*), která je univerzálním nástrojem pro modelování vztahů mezi vstupem a výstupem libovolného statického systému. Druhá cesta k realizaci umělé neuronové sítě vede přes vzdálenosti od etalonů a definování vhodné funkce pro podobnost vektorů. Vycházíme přitom z představy, že čím bližší jsou si vektory, tím mají větší podobnost, ale jejich vzdálenost je de facto menší. Mezi podobností a vzdáleností tedy existuje nepřímá závislost. Toho se využívá při konstrukci skryté vrstvy neuronových sítí s radiální bází (*RBF – Radial Basis Function*). Signály vzniklé ve skryté vrstvě pouze popisují, do jaké míry je vstupní obraz podobný jednotlivým vzorovým obrazům. Přitom se již netrvá na určení „nejpodobnějšího“ obrazu, ale podobnost nabývá hodnot v intervalu od nuly od jedné. Takto normalizovaný signál ze skryté vrstvy je pak pouze vhodně lineárně kombinován k dosažení požadovaných vlastností rozpoznávacího systému. Tímto způsobem lze modelovat libovolný vztah mezi vstupem a výstupem systému. Uvedená úvaha o dvou typech neuronových sítí nám v podstatě ukazuje, jaké jsou možnosti používání lineárních kombinací a vzdáleností v praxi. Bez přídavných nelinearit vyhrává samozřejmě vzdálenost, při použití nelineárních prvků ve skryté vrstvě naopak vyhrávají lineární kombinace.

Samostatný směr v úlohách rozpoznávání obrazu tvoří *samoorganizace*. Při ní systému předložíme obvykle rozsáhlou skupinu obrazů a požadujeme od něj, aby sám určil, co do které třídy patří. Uvedený postup bychom pak mohli opakovaně použít i na obrazy, se kterými se teprve seznámí. Je samozřejmé, že takovému systému musíme zadat přinejmenším počet tříd, do kterých chceme obrazy zařadit, což je to jediné know-how, které od nás vyžaduje. K samoorganizaci využíváme nejen klasické nástroje (shluková analýza), ale i moderní nástroje s vyšší užitnou hodnotou (SOM).

Pokud klasifikační systém nemá předem žádné konkrétní znalosti, ale pouze obecnou schopnost rozpoznávat, je nutné jej nejprve seznámit se vzorovými objekty různých typů a požadovat, aby si sám vytvořil konkrétní pravidla pro jejich rozpoznávání. Moderní klasifikační systémy si nejen pamatují vzorové situace, ale umějí z nich učinit obecný závěr. Při této příležitosti se snaží vytvořit vhodný model pro klasifikaci objektů do tříd. K jeho realizaci se nabízí celá nástrojů: *fuzzy systémy*, *umělé neuronové sítě*, *lineární a nelineární regrese* aj. Přitom se musíme vyrovnat s potížemi, které vzniknou při hledání optimálních parametrů uvedených modelů. U rozpoznávacích systémů je velmi důležité, aby se nezabývaly drobnými detaily. V této souvislosti hovoříme buď o robustním chování (náhodné chyby ve vstupním vektoru příliš neovlivňují výstupní vektor), nebo o invariantním chování (systém zcela ignoruje i velké odchylky, pokud mají stejnou příčinu). Robustního chování docílíme nejsnáze tím, že do celého systému zařadíme takové prvky, které zcela ignorují extrémní hodnoty a pokud možno velmi málo vnímají každou jednotlivou hodnotu. Mnohem zajímavější je požadavek na invarianci systému vůči některým operacím. Nejčastěji požadujeme invarianci systému vůči posunu v prostoru či čase, vůči rotaci v prostoru, změně měřítka v prostoru či čase a změně škálování intenzity signálu.

Celá problematika rozpoznávání obrazů leží na hranici mezi *informatikou*, *matematikou* a *umělou inteligencí*. Rozpoznávání obrazů se zdaleka netýká jen práce s 2D obrazem snímaným optickou cestou. Jde o celou třídu postupů, které slouží ke zpracování 1D, 2D a 3D signálů přicházejících z libovolného senzoru. Takto vzniklé vektory je třeba vhodným způsobem zpracovat, jinak nebude mít rozpoznávací systém užitnou hodnotu. Pokud připustíme, že vektor může vznikat i jiným způsobem než měřením veličin, pak dostává rozpoznávání obrazů ještě mnohem širší význam pro praktické aplikace. V posledních deseti letech jsou velmi rozšířeny průmyslové aplikace, které využívají jak optické senzory, tak speciální diagnostické postupy k tomu, aby

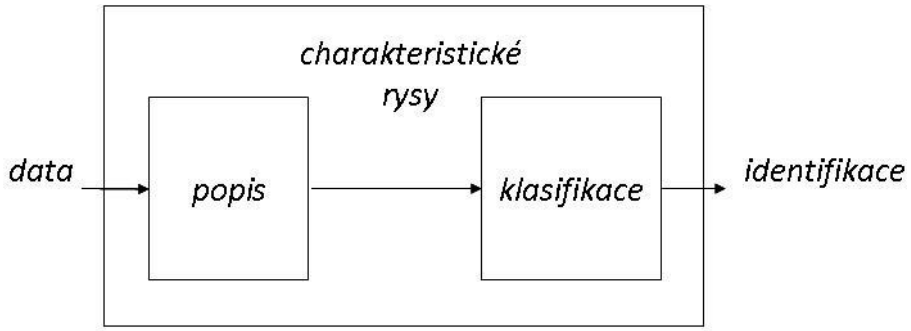
poskytly co nejchytřejší řešení dané technické nebo medicínské úlohy. Více diskutováno v (Bishop, 1995) (Bishop, 2006).

11.3 Proces extrakce hlavních rysů ve vzorech

Metody pracující přímo s obrazcem vzoru bychom mohli označit za přímé, ale v praxi nejsou tyto metody samostatně pouze pro rozpoznávání vzorů používány. Jsou zařazeny spíše na konci nějakého řetězce metod a jejich vstupem nebývají samotné vzory, ale jejich extrahované vlastnosti. Metody pro extrakci vlastností bývají zpravidla zaměřeny na ošetření rozdílů ve vzorech způsobené například jejich otočením, posunutím, deformací, poškozením nebo šumem. Problémy identifikace tvoří velice důležitou podmnožinu aplikací pro rozpoznávání vzorů z důvodu, že není obecně stanoveno, jaké vlastnosti klasifikovaného objektu je třeba vzít v úvahu. Jinými slovy, pouze strukturální tvar je nepostačující pro úplný popis vlastností objektu, proto je potřeba mezi strukturální informace zařadit i doménu znalostí o daném objektu. Cílem tohoto přístupu však není extrakce znalostí, ale poskytnout uživatelům snadný nástroj k provedení prvního datového screeningu.

Typické schéma klasifikátoru vypadá následovně (obrázek 57). Obecně proces klasifikace či rozpoznávání vzorů probíhá ve dvou krocích: nejprve je nutné určit charakteristický rys objektů a potom podle něj objekty klasifikovat. Výběr hlavních rysů závisí zejména na typu dat, jež souvisí s danou aplikací. Velkou roli zde hrají zkušenosti a intuice experta, neboť žádné obecné pravidlo neexistuje. Máme-li *n* navzájem různých vstupů, můžeme použít jim odpovídajících *n* různých symbolů a vytvořit z nich množinu, kterou nazýváme charakteristický vektor (*feature vector*). Tento *n* - rozměrný vektor charakterizuje *n* - rozměrný charakteristický prostor (*feature space*). Podstatou správného rozpoznávání je určení vhodného tvaru tzv. rozlišovací funkce (*discriminant function*). Rozlišovací funkce je matematicky popsána rovnicí nadroviny. Určení takovéto hranice nebývá většinou v praxi tak

jednoduché. Existuje mnoho způsobů, jak stanovit její tvar, např. metodou klasifikace s užitím nejbližších sousedů (*nearest neighbour classification*), lineární klasifikací (*linear classifiers*), statistickými technikami (*Bayesian classification*) apod. Lineární metody klasifikace jsou nejbližší aplikacím teorie umělých neuronových sítí. Jedním ze způsobů, jež umožňuje nalézt rovnici oddělující nadroviny je *algoritmus adaptace perceptronu* (kapitola 10).

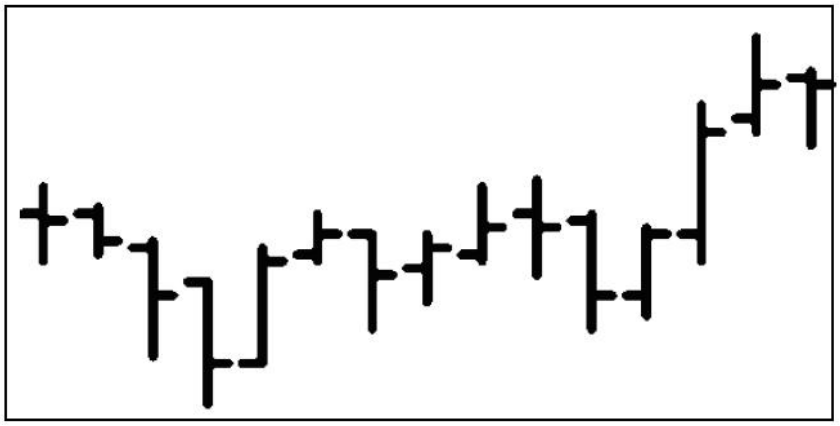


Obrázek 57: Obecné schéma klasifikátoru

Vstupní data mohou být systému předkládána v různé podobě.
V zásadě můžeme rozlišit dvě základní možnosti:

- Číselné vyjádření sledovaných parametrů
- Obrazová data - využití metod tzv. strojového vidění

Na obrázcích 58 a 59 jsou ukázky obrazového a číselného vyjádření jednoho konkrétního úseku ekonomických dat (OHLC). Obrazová reprezentace vzoru obsahuje pouze informace z 3.-6. sloupce tabulky (obrázek 59). Přesto je velikost vzoru (počet pixelů) rovna 7440. Naproti tomu jeho tabulkové vyjádření s 15 řádky a 7 sloupci vystačí při rozlišení 16 bitů na číslo s 1680 bity.



Obrázek 58: Obrazová reprezentace vzoru

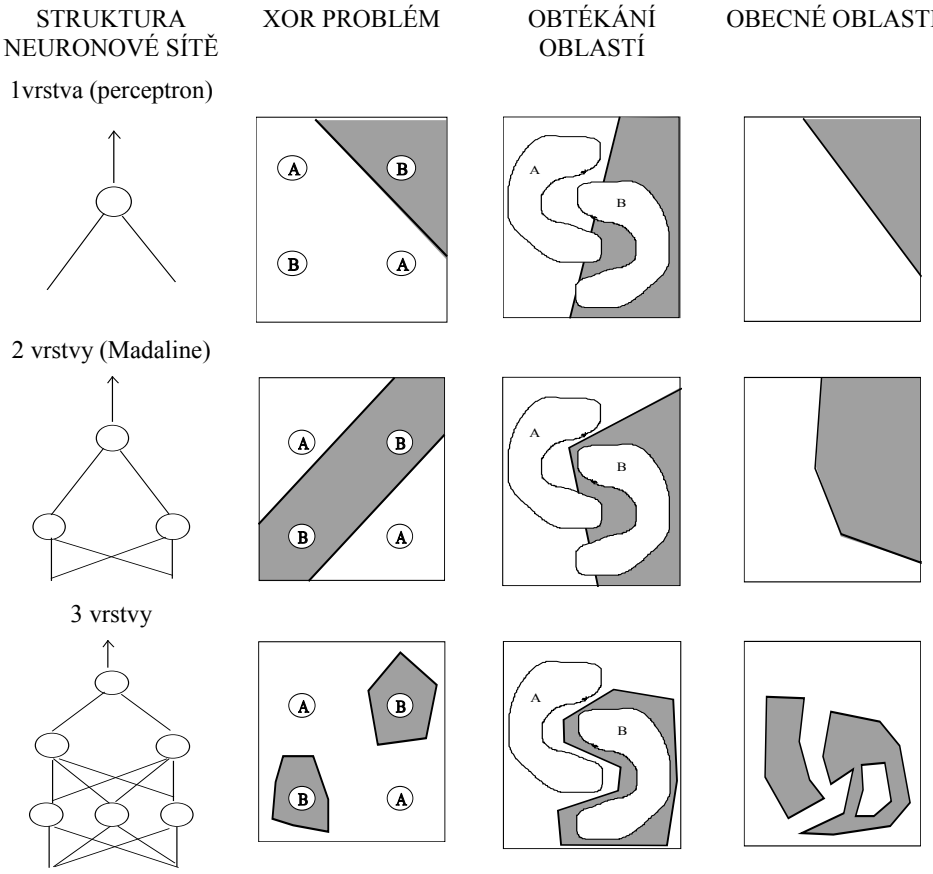
2010.06.02	08:15	1.22220	1.22260	1.22140	1.22210	107
2010.06.02	08:20	1.22220	1.22230	1.22150	1.22170	76
2010.06.02	08:25	1.22160	1.22170	1.21990	1.22090	71
2010.06.02	08:30	1.22110	1.22110	1.21910	1.21970	85
2010.06.02	08:35	1.21980	1.22160	1.21970	1.22140	78
2010.06.02	08:40	1.22150	1.22220	1.22140	1.22190	61
2010.06.02	08:45	1.22180	1.22190	1.22030	1.22120	84
2010.06.02	08:50	1.22130	1.22180	1.22070	1.22160	71
2010.06.02	08:55	1.22150	1.22260	1.22140	1.22200	61
2010.06.02	09:00	1.22220	1.22270	1.22120	1.22200	91
2010.06.02	09:05	1.22210	1.22220	1.22030	1.22090	53
2010.06.02	09:10	1.22080	1.22200	1.22050	1.22190	87
2010.06.02	09:15	1.22180	1.22390	1.22140	1.22350	90
2010.06.02	09:20	1.22370	1.22500	1.22350	1.22430	87
2010.06.02	09:25	1.22440	1.22450	1.22330	1.22430	77

Obrázek 59: Tabulková reprezentace vzoru

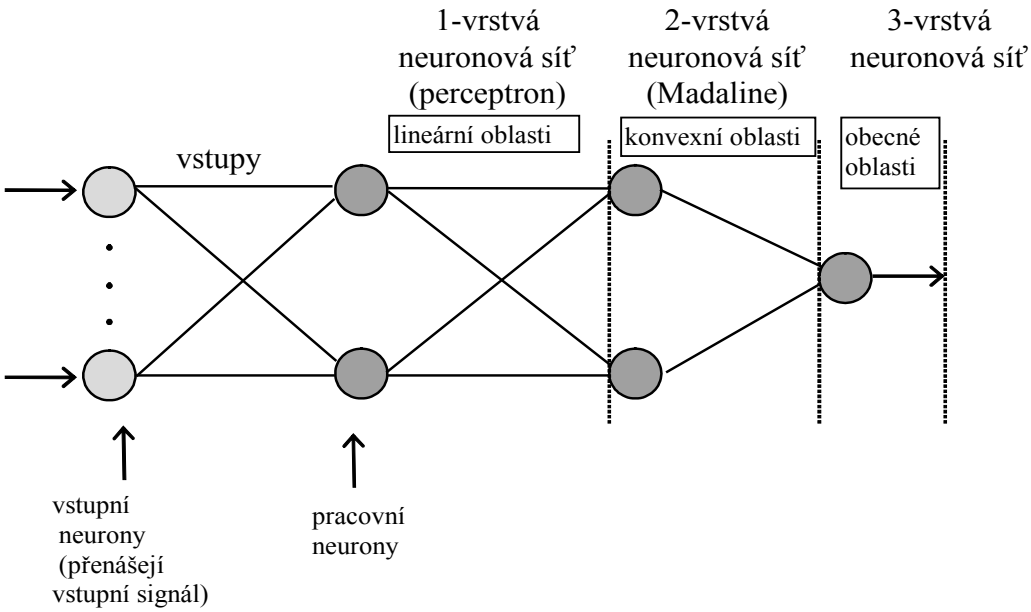
Výhodou obrazových dat je, že lépe korespondují s intuitivní lidskou představou o rozpoznávání vzorů. Dále je potřeba mít na paměti, že i tabulkové údaje musí být zpravidla před zpracováním převedeny do binární (v podstatě obrazové) podoby. Obrazová data jsou vždy dvourozměrná. Tabulkové vzory mohou mít obecně více rozměrů. Grafická reprezentace OHLC dat na obrázku 58 je pěknou ukázkou, projekce vícerozměrných dat do jednoho dvourozměrného obrazce. Jedná se o zobrazení změn hodnot 4-rozměrného vektoru v čase. Celkem jde tedy o vyjádření 5 rozměrů. Lze říci, že intuitivnímu pojmu „vzor" odpovídají zejména dvourozměrné tvary, což umožňuje zobrazit např. průběh skalární proměnné, ale grafické zobrazení již není triviální v případě, že systém má více než jeden parametr.

11.4 Neuronové sítě a jejich možnosti klasifikace

Klasifikace je jednou z hlavních aplikačních oblastí pro umělé neuronové sítě (Bishop, 1995). Na následujících dvou obrázcích 60 a 61 (Beale, 1992) jsou souhrnně zobrazeny různé typy neuronových sítí (tj. neuronové sítě s různým počtem vnitřních vrstev) a jejich možnosti klasifikace.



Obrázek 61: Neuronové sítě s různým počtem vnitřních vrstev a jejich možnosti klasifikace.



Obrázek 61: Mezní oblasti rozpoznávané neuronovou sítí s různým počtem vnitřních vrstev.

Použití klasifikátorů na bázi umělých neuronových sítí

V mnoha případech se jedná o systémy, kde klasifikace reprezentuje pouze součást řešení a celkové řešení využívá řadu dalších metod.

- OCR programy, které musí na digitalizovaném dokumentu rozpoznat jednotlivá písmena (OCR software se dodává k většině scannerů).
- kontrola dopravy, kde se klasifikace používá při detekci a rozpoznávání poznávacích značek aut
- průmyslová kontrola kvality výrobků
- rozpoznání textury povrchů
- predikce trendu časové řady (resp. prognóz vývoje ekonomických ukazatelů poskytuje managementu podniků důležité informace pro strategické plánování)
- automatické řízení systému v reálném čase (adaptace systému na změny během jeho provozu v reálném čase)

a další.

Kontrolní otázky:

1. Co je podstatou klasifikace objektů do tříd?
2. Proč je klasifikace jednou z oblastí, kde se neuronové sítě velmi dobře uplatňují?
3. Jaký je princip dvouhodnotové a vícehodnotové klasifikace?
4. Jaké metody soft-computingu lze uplatnit při klasifikaci?

Shrnutí obsahu kapitoly

V této kapitole jsme se zabývali problematikou rozpoznávání vzorů a klasifikace. Zaměřili jsme se především na metody soft-computingu a na možnosti jejich klasifikace. Důraz zde byl kladen na uplatnění umělých neuronových sítí při řešení úloh z oblasti rozpoznávání vzorů a při klasifikaci.

Pojmy k zapamatování

- klasifikace
- dvouhodnotová klasifikace
- klasifikátor

12 Soft-Computing v aplikacích

V této kapitole se dozvíte:

- Jaké jsou aplikační oblasti fuzzy logiky.
- Jaké jsou oblasti použití umělých neuronových sítí.
- Jaké jsou oblasti použití evolučních algoritmů.

Po jejím prostudování byste měli být schopni:

- Uvést, jaké jsou aplikační oblasti fuzzy logiky.
- Uvést, jaké jsou aplikační oblasti umělých neuronových sítí.
- Uvést, jaké jsou aplikační oblasti evolučních algoritmů.

Klíčová slova této kapitoly:

Fuzzy logika, umělé neuronové sítě, evoluční algoritmy.

Průvodce studiem

V této závěrečné kapitole se postupně zamyslíme nad třemi tématy: (1) aplikace fuzzy logiky, (2) aplikace umělých neuronových sítí a (3) aplikace evolučních algoritmů.

12.1 Aplikace fuzzy logiky

Teorie fuzzy logiky je oborem inteligentních systémů, který dospěl do stadia, kdy může být rutinně využíván i na úrovni standardních automatizačních prostředků, např. programovatelných automatů. Brání tomu ale nedostatečné znalosti a chybějící metodika pro programátory i nedostatečný zájem investorů. Chování fuzzy systémů se nejčastěji zadává jako soubor příkazů (fuzzy pravidel). Může být ale popsáno i jako soubor zobecněných logických výrazů nebo jim odpovídajícího

zobecněného schématu (logického či blokového). Ve vícehodnotové logice lze navíc využít datové objekty (tabulky, mapy nebo složitější datové struktury). Ve formě pravdivostních map je často zobrazován i soubor pravidel pro fuzzy systémy (mapy pravidel).

V souvislosti s fuzzy systémy se obvykle nerozlišují pojmy *kombinační a sekvenční logické funkce* (*statické a dynamické systémy*). Může tak vzniknout mylný dojem, že fuzzy logikou jsou realizovány jen kombinační funkce. Sekvenční (dynamický) fuzzy systém je ale vytvářen vždy, když je třeba zpracovat fuzzy proměnné různých úrovní z časové posloupnosti (např. současný vzorek spolu s minulým), aktuální hodnotu veličiny spolu s jejím přírůstkem (diferencí oproti minulému kroku) nebo např. hodnotu polohy spolu s údajem o její rychlosti a zrychlení. Sekvenční systém vzniká i při vytváření zpětných vazeb, což je pro fuzzy logiku ještě riskantnější postup než při realizaci asynchronních logických obvodů v booleovské logice. Může být zdrojem nečekaného (obvykle nežádoucího) chování. I osvědčený synchronní postup realizace sekvenčních obvodů je pro fuzzy logiku problematický, překvapivě se může chovat už pouhá fuzzy analogie paměťového členu, např. obvodu R-S. Fuzzy systém obvykle začne „žít vlastním životem“ a chová se podobně jako číslicový dynamický systém; fuzzy logika tak překvapivě sbližuje obě kategorie systémů, dosud při popisu a výkladu oddělované.

Významné a pro další vývoj rozhodující bylo použití fuzzy logiky pro automatické řízení podzemní dráhy v Sendai blízko Tokia (1987). Regulační systém firmy Hitachi dosáhl ve srovnání s klasickými regulačními technikami velmi dobré výsledky: zkrácení dobu cestování, vyšší bezpečnost, větší pohodlí cestování, plynulejší příjezd a brzdění, snížení spotřeby energie. Fuzzy regulace tak potvrdila životaschopnost teorie. Pozoruhodná výkonnost systému byla tak působivá, že v průběhu jednoho roku pracovalo na vývoji fuzzy technologie více než 50 japonských společností. Japonské ministerstvo pro mezinárodní obchod a průmysl (MITI) založilo s podporou předních japonských firem v březnu 1989 institut LIFE (Laboratory for International Fuzzy

Engineering Research). Kromě financování základního výzkumu bylo jeho cílem podpořit mezinárodní spolupráci a kontakty mezi vědou a hospodářstvím. Již v roce 1992 bylo v Japonsku registrováno více než 2000 fuzzy patentů. Proč je fuzzy logika taková úspěšná? Její základní předností je schopnost matematicky podchytit informace vyjádřené slovy. Čili umožňuje pracovat s nejednoznačnými pojmy jako „nízký", „blízko", „daleko", „asi", „zhruba", „málo", „velmi", „mnoho", a s hromadou dalších slov, hojně používaných v lidské řeči. Co pro počítač znamená „blízko" 50 metrů, 100 metrů nebo kilometr? A pokud je 50 metrů blízko, 51 metrů už není? Množství nejednoznačností v běžném jazyce představuje problém, který není zvládnutelný standardní logikou. A právě tento problém je třeba překonat, pokud máme počítače „naučit" lidské řeči. Fuzzy logika nabízí řešení, které umožní počítači „pochopit", co znamená „středně vysoký" na základě výše budov na dané ulici a co znamená „blízko" vzhledem k délce ulice a poloze banky. Potom spojením zjištěných údajů určí, které budovy připadají v úvahu. Každou budovu zároveň ohodnotí, čímž stanoví pořadí prohledávání budov. Tedy stejně jako člověk začne prohledávat budovy v pořadí, které si sestaví podvědomě. Obešli bychom se i bez takových slov? Pokud se jen trochu zamyslíme, zjistíme, že bez nejednoznačných slov by bylo dorozumívání mezi lidmi velmi obtížné. Jsou však situace, kdy má s určením jejich významu problémy i člověk. Zejména zpracování velkých objemů dat v počítačových databázích vyžaduje inteligentní nástroje, umožňující člověku používat pojmy, jejichž pomocí dokáže tak dobře vyjádřit své požadavky. Předností fuzzy nástrojů je i vyhledávání informací na základě nesprávných nebo neúplných údajů a s možností najít i chybně uložené informace. Velmi populární jsou tzv. systémy na podporu rozhodování, kde úloha fuzzy logiky spočívá ve výběru nejvhodnějších alternativ. Oblastí využití může být výběr obchodního partnera, vyhodnocení konkurzu, nebo efektivně zasílání reklamních poštovních zásilek. I seznam může být podkladem pro analýzu sortimentu výrobků, odebíraných určujícími zákazníky. Naznačený problém zvládnutí běžného jazyka je přirozeně, složitější.

Pokud však mají v budoucnu běžní lidé masově používat počítače, pak je nezbytné, aby počítače rozuměli lidskému jazyku.

Nejednoznačnost v lidské řeči se přenáší i do běžného lidského uvažování. Např. pokud se při řízení blížíme k zatáčce, podvědomě uvažujeme: „pokud je zatáčka ostrá a auto jede příliš rychle, tak je třeba prudce přibrzdit", „pokud je zatáčka mírná a auto jde rychle, tak je třeba trochu ubrat plyn", atd. Pro danou zatáčku a rychlost náš mozek vyhodnotí tato „pokud-tak" pravidla a my podle toho nakládáme s brzdou a plynem. Právě na tom je založena podstata fuzzy logiky, která poskytuje metody pro přibližné „uvažování". Uplatnění fuzzy logiky v regulaci vychází z předpokladu, že máme takové znalosti ve tvaru „jestliže-tak" pravidel, která jsou východiskem pro tzv. fuzzy pravidla. Jako příklad nám dobře poslouží fuzzy vysavač. Fuzzy systém se sestává z malého mikročipu. Infračervené senzory zjišťují typ koberce a množství nasávané nečistoty. Tyto údaje vstupují do fuzzy pravidel: „pokud je podklad plyšový koberec, tak vysávej silně", „pokud je podklad keramická dlažba, vysávej slabě", „pokud je koberec velmi špinavý, vysávej velmi silně atd. Mikročip na výstupu průběžně udává sílu sání motoru. Fuzzy vysávání je elegantní a šetří energii.

Komfort a úspornost jsou pro fuzzy regulaci typické. Fuzzy pračka po naplnění „přezkoumá" senzory obsah bubnu („mnoho" prádla, „málo" špinavé, „jemná" látka), vybere nejvhodnější z 600 pracích programů a nastaví jen nezbytné množství pracího prášku a vody. Významnější úspory energie přináší klimatizace, která navíc zlepšuje zachování teploty při rušivých vlivech (otevření dveří nebo okna). Mitsubishi a Samsung uvádějí, že jejich fuzzy modely dosahují až 40% úspory energie. Většina japonských a korejských automobilek vyvinula a patentovala několik fuzzy systémů. Například Nissan vlastní patent na ABS, který zabrání smyku na základě rychlosti a akcelerace auta. Firmy samozřejmě neprozrazují přesný typ a počet fuzzy pravidel, protože právě v nich je „ukrytá" veškerá znalost problematiky.

Obzvlášť dobře se fuzzy logika osvědčuje v případech s aktivním vstupem člověka, jako je například rozpoznávání rukopisu. Počítače tak

mohou odlišovat jednotlivé znaky podle délky, směru a vzájemné polohy čar, které znak tvoří. Při rozpoznávání ručně napsaného znaku přiřadí správný znak (vzor), i když jsou oproti vzoru čáry trochu delší či kratší, strmější nebo navzájem posunuté. Čili je tolerantní, podobně jako člověk, k odchylkám a nepřesnostem, které má každý rukopis.

Pozoruhodné je vylaďování fuzzy pravidel pomocí umělých neuronových sítí. Pokud nezkušený řidič dostává v zatáčkách smyk v důsledku rychlé jízdy, postupně vylaďuje svou představu o „rychlosti" auta a „ostrosti" zatáček v pravidlech tak, aby smyku předešel, a současně se učí nová pravidla pro zvládnutí smyku.

Některá komerční použití fuzzy technologie:

- Automatická převodovka (Honda, Nissan,) určující převodový poměr na základě zatížení motoru, stylu jízdy a stavu vozovky.
- Myčka nádobí Matushita upravuje cyklus čištění - máchání a strategii mytí v závislosti na počtu nádobí a typu a množství zvyků potravy na nádobí.
- Řízení výtahů (Fujitec, Mitsubishi Electric, Toshiba) zmenšuje dobu čekání při přivolávání výtahu na základě provozu pasažérů.
- Fotoaparát (Canon, Minolta) nalezne osobu kdekoliv na snímek a určí centrální bod pro zaostření.
- Mikrovlnná trouba (Hitachi, Sanyo, Sharp, Toshiba) nastavuje výkon a přizpůsobuje strategii tepelné úpravy.
- Chladnička (Sharp) stanoví dobu odmrazování a dobu zchlazování na základě užívání. Neuronová síť se učí zákazníkovy zvyky používání a podle toho ladí fuzzy pravidla.
- Televizor (Goldstar, Hitachi, Samsung, Sony) upravuje barvu obrazovky pro každý jednotlivý obraz a stabilizuje hlasitost podle polohy diváka v místnosti.

- Vysavač (Hitachi, Matsushita, Toshiba) určuje strategii sání motoru na základě množství nečistoty a typu odkladu.
- Videokamera (Canon, Sanyo) vyrovnává chvění způsobené držením v rukou a přizpůsobuje zaostření.
- Topinkovač (Sony) nastavuje čas opékání a stanovuje strategii záření pro jednotlivé druhy chleba.
- Automatická pračka (Daewoo, Goldstar, Hitachi, Matsushita, Samsung, Sanyo, Sharp) přizpůsobuje strategii praní na základě pořízeného úrovně znečištění, typu látky, množství prádla a teploty vody. Některé modely používají neuronové sítě na adaptaci fuzzy pravidel vzhledem k zákazníkovy potřebám.

a další.

12.2 Aplikace umělých neuronových sítí

Text kapitoly je převážně převzat z (Šíma 1996). Neuronové sítě v současnosti patří mezi významnou část počítačově orientované umělé inteligence, kde zaujali postavení univerzálního matematicko-informatického přístupu ke studiu a modelování procesů učení. Kromě umělé inteligence mají neuronové sítě nezastupitelné uplatnění také v kognitivní vědě, lingvistice, neurovědě, řízení procesů, přírodních a společenských vědách, kde se pomocí nich modelují nejen procesy učení a adaptace, ale i široké spektrum různých problémů klasifikace objektů a také problémů řízení složitých průmyslových systémů. Původním cílem výzkumu neuronových sítí byla snaha pochopit a modelovat způsob, jakým myslíme a způsob, jak funguje lidský mozek. Při vytváření modelů umělých neuronových sítí nám nejde o vytvoření identických kopií lidského mozku, ale napodobujeme zde pouze některé jeho základní funkce. Neurofyziologie zde slouží jen jako zdroj inspirací a navržené modely umělých neuronových sítí jsou dále rozvíjeny bez ohledu na to, zda modelují lidský mozek, či nikoliv. Nejvýznamnější oblasti použití umělých neuronových sítí jsou následující:

Kromě použití neuronových sítí v úlohách rozpoznávání a klasifikace (popsané v předchozí kapitole) je jejich další možnou aplikační oblastí *řízení* složitých zařízení v dynamicky se měnících podmínkách. V úvodní kapitole jsme uvedli motivační příklad z této oblasti, a to balancování koštěte. Dalším příkladem řídicího systému popsaného v literatuře je autopilot automobilu, který se v počítačové simulaci pohybuje na dvouproudé dálnici spolu s auty jedoucími stejným směrem. Auto řízené neuronovou sítí určovalo na základě vzdálenosti a rychlosti nejbližších aut v obou pruzích svou vlastní rychlost a změnu pruhu. Dále neuronová síť ovládala volant podle zakřivení dálnice, polohy auta v pruhu a aktuálního úhlu volantu. Je zajímavé, že neuronová síť se kromě úspěšného řízení vozidla (bez kolizí) včetně předjíždění naučila i různé zvyky a styl jízdy (např. riskantní rychlá jízda a časté předjíždění nebo naopak opatrná pomalá jízda) podle řidičů - trenérů, od kterých byly získány tréninkové vzory.

Jinou důležitou aplikační oblastí neuronových sítí je *predikce* a příp. následné *rozhodování*. Typickými příklady z této oblasti jsou předpověď počasí, vývoj cen akcií na burze, spotřeba elektrické energie apod. Např. při předpovědi počasí jsou vstupem neuronové sítě odečty základních parametrů (teplota, tlak apod.) v čase a učitelem je skutečný vývoj počasí v následujícím období. Uvádí se, že u předpovědi počasí v rozpětí několika dnů byla síť úspěšnější než meteorologové.

Jiným příkladem uplatnění neuronových sítí je *analýza signálů* jako např. EKG, EEG apod. Spojitý signál je vzorkován ve stejných časových intervalech a několik posledních diskrétních hodnot úrovně signálu slouží jako vstup do např. dvouvrstvé neuronové sítě. Naučená neuronová síť je schopna identifikovat specifický tvar signálu, který je důležitý pro diagnostiku. Např. neuronová síť s topologií 40 - 17 - 1 byla použita pro klasifikaci EEG signálů se specifickými α-rytmy.

Další oblastí aplikace neuronových sítí je *transformace signálů*, jehož příkladem je systém NETtalk, určený pro převod anglicky psaného textu na mluvený signál. Tento systém je založen na neuronové síti s topologií 203 - 80 - 26 s 7×29 vstupními neurony pro zakódování

kontextu 7 písmen psaného textu odpovídá jeden neuron, který je při jejich výskytu aktivní, 80 skrytými neurony v mezilehlé vrstvě 26 výstupními neurony reprezentují fonény odpovídajícího mluveného signálu. Funkce sítě je následující: vstupní text se postupně přesouvá u vstupních neuronů po jednom písmenu zprava doleva a přitom je aktivní právě ten výstupní neuron, který reprezentuje fonén odpovídající prostřednímu ze 7 písmen vstupního textu. V našem příkladě se čte prostřední písmeno „C“ v anglickém slově „CONCEPT“ s výslovností [konsept], kterému odpovídá fonén [s]. Stejné písmeno „C“ na začátku tohoto slova však v daném kontextu odpovídá fonénu [k]. Úspěšná implementace systému NETtalk vedla ke snaze vytvořit systém založený na neuronové síti s obrácenou funkcí, která by převáděla mluvený jazyk do psané formy (tzv. fonetický psací stroj).

Další možností využití neuronových sítí je *komprese dat* např. pro přenos televizního signálu, telekomunikaci apod. Pro tento účel byla vyvinuta technika použití neuronové sítě se dvěma vnitřními vrstvami a s topologií *n - n/4 - n/4 - n* (tj. *n* neuronů ve vstupní vrstvě, *n/4* neuronů ve vnitřních vrstvách a n neuronů ve výstupní vrstvě). Počet neuronů ve vnitřních vrstvách je výrazně menší než je počet neuronů ve vstupní a výstupní vrstvě. Počet neuronů ve vstupní i výstupní vrstvě je stejný, protože obě vrstvy reprezentují stejný obrazový signál. Tato neuronová síť se učí různé obrazové vzory tak, že vstup i výstup tréninkových vzorů představují totožný obraz. Síť tak pro daný obrazový vstup odpovídá přibližně stejným výstupem. Při vlastním přenosu je pro daný obrazový signál u vysílače nejprve vypočten stav skrytých neuronů a takto komprimovaný obraz je přenášen informačním kanálem k příjemci, který jej dekóduje výpočtem stavů výstupních neuronů. Tímto způsobem je získán téměř původní obraz. Při vlastním experimentu se ukázalo, že kvalita přenosu (srovnatelná s jinými způsoby komprese dat) závisí na tom, zda jsou přenášené obrazy podobné tréninkovým vzorům, na které se síť adaptovala.

Posledním oborem aplikace neuronových sítí, který zde uvedeme, jsou *expertní systémy.* Velkým problémem klasických expertních systémů

založených na pravidlech je vytvoření báze znalostí, která bývá časově velmi náročnou záležitostí s nejistým výsledkem. Neuronové sítě představují alternativní řešení, kde reprezentace znalostí v bázi vzniká učením z příkladových inferencí. V tomto případě aktivní režim neuronové sítě zastupuje funkci inferenčního stroje. Na druhou stranu implicitní reprezentace znalostí neumožňuje pracovat s neúplnou informací a neposkytuje zdůvodnění závěrů, což jsou vlastnosti, bez kterých se prakticky použitelný expertní systém neobejde. Tento problém částečně řeší univerzální neuronový expertní systém EXPSYS, který obohacuje vícevrstvou neuronovou síť o intervalovou aritmetiku pro práci s nepřesnou informací a o heuristiku analyzující síť, která umožňuje jednoduché vysvětlení závěrů. Systém EXPSYS byl úspěšně aplikován v energetice a medicíně. Např. v lékařské aplikaci jsou zakódované příznaky onemocnění a výsledky různých vyšetření vstupem do neuronové sítě a diagnózy, popř. doporučená léčba jsou jejím výstupem. Tréninkovou množinu lze získat z kartotéky pacientů.

12.3 Aplikace evolučních algoritmů

Kdy jsou evoluční algoritmy vhodné?

Tímto způsobem je vhodné řešit obecně obtížné problémy, u kterých je velký prostor řešení. Dále je tímto způsobem možné řešit obtížně formulovatelné problémy. Např. víme, jak by měl vypadat výstup, ale nevíme, jakým způsobem takový výstup dostat. Příkladem může být návrh elektronických schémat. Víme, jaký požadujeme od schématu výstup, ale nevíme, jak zapojit součástky, abychom tento výstup dostali. Možností, jaké součástky a jak je ve schématu zapojit, je neomezeně mnoho. Evoluční algoritmy dokážou prohledat různé možnosti a najít nejvhodnější zapojení. Tímto způsobem se např. podařilo nalézt patentovaná schémata. Evoluční algoritmy jsou schopné také najít více optimálních řešení, jsou schopny najít celočíselná řešení i reálná. Velké uplatnění nacházejí evoluční algoritmy také při vytváření rozvrhů práce pro stroje v továrnách, v teorii her, v managementu, pro řešení

optimalizačních problémů multimodálních funkcí, při řízení robotů, v rozpoznávacích systémech, v úlohách umělého života, použití evolučních algoritmů pro optimalizaci neuronových sítí apod. Z hlediska dobývání znalostí z databází je zajímavé využití genetických algoritmů přímo pro učení se konceptům, kdy genetický algoritmus provádí paralelní náhodné prohledávání prostoru hypotéz. Jeho schopnost učit se koncepty je srovnatelná s algoritmy pro tvorbu rozhodovacích stromů i rozhodovacích pravidel.

Nevýhody evolučních algoritmů

U obtížných problémů jsou tyto algoritmy schopny najít pouze přibližná řešení. I když složitost roste pouze lineárně s velikosti populace a počtem generací, výpočet nemusí být rychlý vzhledem k tomu, že vyhodnotit řešení bývá časově náročné.

Evoluční algoritmy jsou používané jak pro optimalizaci diskrétních, tak i pro optimalizaci reálných proměnných. Je zřejmé, že fungují pomaleji než jakékoliv jiné heuristické přístupy a pokud nemáme předem zadané podmínky pro globální extrém, nikdy nevíme, zda jsme jej již dosáhli a zda máme optimalizaci zastavit. Mají však i podstatné výhody (Pospíchal, 1996): (1) jsou velmi obecně definované, a proto aplikovatelné téměř na jakýkoliv problém; (2) dokážou se dostat z lokálního extrému. Evoluční proces prohledávání prostoru potenciálních řešení je pak rovnováhou dvou cílů (Pospíchal, 1996): (1) co nejrychleji nalézt nejbližší (většinou lokální) optimum v malém okolí výchozího bodu; (2) co nejlépe prohledat prostor všech možných řešení. Jednotlivé evoluční metody se liší právě svým zaměřením k těmto dvěma cílům. Zatím neexistuje jednoznačně přijímaný přístup, jak analyzovat řešený problém tak, abychom určili, který optimalizační algoritmus k jeho řešení použít. V současné době se jedná při používání jednotlivých druhů evolučních algoritmů spíše o zvyk než o důkladnou analýzu problému před volbou algoritmu pro jeho řešení. Evoluční algoritmy mají také velké množství parametrů pro nastavení a

mnoho modifikací. Zatím se pro výběr algoritmu obvykle používá metoda pokusu a omylu, stejně tak i pro nastavení jeho parametrů. Existují sice různé „metapřístupy“ při kterých se optimalizuje výběr použité metody včetně nastavení jejích parametrů, ale tyto přístupy jsou velmi výpočetně náročné. Pokud se skutečně nejedná o problém, který se chystáme řešit opakovaně na podobných typech funkcí, je tento přístup neefektivní. V poslední době se stále častěji používají hybridní metody, které přebírají a kombinují několik základních přístupů do jednoho.

Kontrolní otázky:

1 Jaké jsou aplikační oblasti fuzzy logiky?
2 Jaké jsou oblasti použití umělých neuronových sítí?
3 Jaké jsou oblasti použití evolučních algoritmů?

Korespondenční úkol:

Vypracujte seminární práci na téma „Použití metod soft-computingu v …“ (oblast použití si zvolte sami). Informace hledejte především na www-stránkách. Zaměřte se pouze na jednu oblast, v níž použití vybraných soft-computingových metod rozeberete podrobněji.

Shrnutí obsahu kapitoly

V této kapitole byly postupně uvedeny některé aplikační oblast fuzzy logiky, umělých neuronových sítí a evolučních algoritmů.

Pojmy k zapamatování

- fuzzy logika
- umělé neuronové sítě
- evoluční algoritmy

Literatura

(Back, 1997) Back T., Fogel D. B., Michalewicz Z. (1997), *Handbook of evolutionary algorithms*, Oxford University Press, , ISBN 0750303921

(Barnsley, 1993) Barnsley M.F., *Fractals Everywhere*, Academic Press Professional 1993, ISBN 0-12-079061-0

(Beale, 1992) Beale, R. - Jackson, T. (1992) Neural Computing: An Introduction. J W Arrowsmith Ltd, Bristol, Greit Britain.

(Berka, 2010) Berka, P. Aplikace systémů dobývání znalostí pro analýzu medicínských dat. Dostupné z http://euromise.vse.cz/kdd/index.php (květen 2010)

(Bishop, 1995) Bishop, C. M.: Neural Networks for Pattern Recognition. Oxford: Calderon Press, 1995. ISBN 0-19-853864-2.

(Bishop, 2006) Bishop, C. M.: Pattern Recognition and Machine Learning. Springer, 2006. ISBN 0-387-31073-8.

(Davis, 1991) Davis, L. (1991) Handbook of genetic algorithms. Van Nostrand Reinhold, New York.

(Chalupník, 2012) Chalupník, V. Biologicky inspirované algoritmy. [online http://www.root.cz/serialy/biologicky-inspirovane-algoritmy/] (Citováno 10.5.2012)

(Goldberg, 1989) Goldberg, D. E. Genetic algorithm in search optimalization and machine learning. Addison-Wesley, Reading., Massachusets 1989.

(Fausett, 1994) Fausett, L. V. (1994) Fundamentals of Neural Networks. Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Janošek, M (2011) Fuzzy řízení. Distanční studijní opora. Ostravská univerzita v Ostravě.

(Kadlec, 2010) K. Měřící technika 2010. [on line http://web.vscht.cz/kadleck/] (Citováno 21.12.2012)

(Kvasnička, 1997) Kvasnička, V., Beňušková, L., Pospíchal, J., Farkaš, I., Tiňo, P., Kráľ, A. (1997) Úvod do teórie neurónových sietí. IRIS, Bratislava.

(Kvasnička, 2000) Kvasnička, V., Pospíchal, J., Tiňo, P. (2000) Evolučné algoritmy. STU Bratislava.

(Kvasnička 2006) Kvasnička, V., Pospíchal, J.: *Matematická logika*, Vydavateľstvo STU, Bratislava, 2006.

(Machová, 2002) Machová, K. *Stojové učenie. Princípy a algoritmy*. FEI TUKE Košise, 2002.

(Mařík, 2003) Mařík, V., Štěpánková, O., Lažanský, J.: Umělá inteligence (1), Academia, Praha 1993.

(Mařík, 2007) Mařík, V., Štěpánková, O., Lažanský, J. *Umělá inteligence* (5), Academia, Praha 2007.

(McCulloch1943) McCulloch, W. S., Pitts, W. H.: A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, **5** (1943), 115-133.

(Michalski, 1998) Michalski, R., Bratko, I., Kubat, M.: *Machine Learning and Data Mining, Methods and Applications.* Addison Wesley 1998.

(Modrlák, 2002) Modrlák O., *Fuzzy řízení a regulace – Teorie automatického řízení II*, Fakulta mechatroniky a mezioborových inţenýrských studií, 2002, 25 s. Skripta. Technická univerzita v Liberci

(Pivoňka 2000) Pivoňka, P.: Analysis and Design of Fuzzy Controller. In: Fuzzy Control. Theory and Praxis, Physica-Verlag, 2000, ISBN 3 - 7908-1327-3.

(Pokorný, 2004) Pokorný, Miroslav. *Expertní systémy*. Ostravská univerzita v Ostravě, 2004. 103 s. Skripta. Ostravská univerzita v Ostravě.

(Pospíchal, 1996) Pospíchal, J. Evolučné optimalizačné algoritmy. Habilitační práce. Bratislava, 1996.

(Ranawana, 2006) Ranawana R, Palade V (2006) Multi-classifier systems: review and a roadmap for developers, In-ternational Journal of Hybrid Intelligent Systems, 3 (1) 35–61

(Šíma 1996) Šíma, J., Neruda, J. (1996) Teoretické otázky neuronových sítí. Matfyzpress, Praha.

(Štěpnička 2010) Štěpnička, M., Vavříčková, L. Matematické metody pro umělou inteligenci. 2010.

(Vysoký 1997) Vysoký, P.: Fuzzy řízení. Skripta, ČVUT Praha, 1997.

(Zadeh, 1973) ZADEH, L.A.: *Outline of a New Approach to the Analysis of Complex Systems and Decision Processes*. IEEE Trans. Syst. Man. Cybern., 1, 1973, s. 28-44.

(Zelinka, 1999) Zelinka, I. Aplikovaná informatika. Učební texty VUT v Brně. 1999

(Zelinka, 2008) Zelinka, I., Oplatková, Z., Ošmera, P., Šeda, M., Včelař, F. (2008) Evoluční výpočetní techniky - principy a aplikace. BEN - technická literatura, Praha, , ISBN 80-7300-218-3.