VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

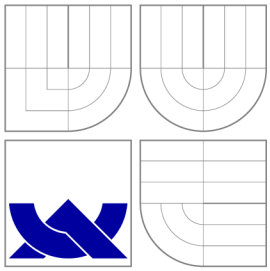
EVOLUČNÍ NÁVRH UMĚLÉ NEURONOVÉ SÍTĚ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

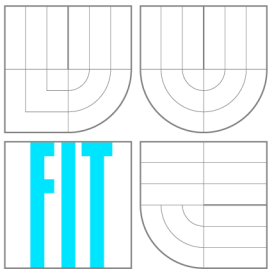
AUTOR PRÁCE
AUTHOR

TOMÁŠ JÍLEK

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ NÁVRH UMĚLÉ NEURONOVÉ SÍTĚ

EVOLUTIONARY DESIGN OF ARTIFICIAL NEURAL NETWORK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

TOMÁŠ JÍLEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR POSPÍCHAL

BRNO 2011

Abstrakt

Tato bakalářská práce je zaměřena na Neuronové sítě, Genetické algoritmy a evoluční návrh. První část popisuje Neuronové sítě, jejich historii, učení, způsob použití a Genetické algoritmy, jejich části, operátory a použití. Další část je věnována predikcícm časových řad, a to konkrétně pomocí Neuronových sítí. Poté následuje samotná implementace a experimenty s evolučně navrhnutými Neuronovými sítěmi pro predikci, v nichž jsou jako časové řady použity kurzy několika měn. V závěru je provedeno shrnutí dosažených výsledků a diskuze nad nimi, stejně tak nad možnostmi dalšího rozvoje práce.

Abstract

Focus of this bachelor thesis is on Neural networks, Genetic algorithms and Evolutionary Design. First part of thesis describes Neural networks, their history, training and ways of use and Genetic algorithms, their components, operators and practical application. Next part is devoted to prediction of time series, specifically prediction with use of Neural networks. This is followed by practical part of work, implementation of experiments with Evolutionary design of Neural networks for prediction in which currency exchange rates of several countries are used as a predicted time series. Results and discussion about further development of thesis are described in last chapter.

Klíčová slova

Neuronové sítě, Evoluční algoritmy, GA, Predikce, Časové řady, Evoluční návrh

Keywords

Neural networks, Evolutionary algorithm, Prediction, Time series, Evolutionary design

Citace

Jílek, T. *Evoluční návrh umělé neuronové sítě.* Brno: FIT VUT v Brně, 2011. Bakalářská práce.

Evoluční návrh umělé neuronové sítě

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Petra Pospíchala

.....
Tomáš Jílek
17. května 2011

Poděkování

Tímto bych rád poděkoval Ing. Petru Pospíchalovi za to, že umožnil vznik této práce svým vstřícným a entuziastickým přístupem, zápalem pro věc a ochotou konzultovat. Také za to, že mi touto prací bylo umožněno hlouběji se zajímat o neuronové sítě a evoluční algoritmy.

© Tomáš Jílek, 2011.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Kapitola 1

Úvod

Informační technologie se vždy vyznačovaly tendencemi inspirovat se v ostatních oborech, a nejinak tomu je s biologií. V rámci mezioborových prací, které poslední dobou získávají na popularitě, je třeba spojovat znalosti, které se mnohým mohou zdát neslučitelné. Takto například vznikal model neuronových sítí, které byly inspirované neurony v lidském mozku. Po úvodní snaze o co největší věrnost své biologické předloze se od ní postupně oprošťoval, vyvíjel se a jeho potenciál byl zkoumán velice intenzivně. Stejně tak genetické algoritmy, součást přístupů k řešení problémů inspirovaných evolucí a genetikou. Důležité je, že takto vznikly a vznikají velice cenné a mocné nástroje informačních technologií, které pomáhají lépe nebo rychleji řešit problémy.

Návrh na základě evolučních přístupů je již úspěšně využíván na celou řadu problémů. Hlavně v situacích, kdy existuje velmi mnoho možností, jsou genetické algoritmy schopné najít v rozumném čase poměrně slušné řešení nebo navrhnout lepší řešení než stávající, navrhnuté člověkem.

Práce se zabývá popisem neuronových sítí a genetických algoritmů a evolučním návrhem neuronové sítě tak, aby se učila lépe či rychleji predikci kurzů měn.

1.1 Členění práce

Práce je členěna do kapitol nazvaných podle tématického okruhu, který popisují.

V první části jsou vcelku podrobně prezentovány neuronové sítě. Od jejich vzniku inspirací biologií, konkrétně lidským mozkem, přes jednoduché matematické a historické modely, až po reálné modely používané k výpočtům. Diskutováno je též učení neuronových sítí, hlavně s učitelem, nicméně nastíněno je i učení bez učitele, samoorganizací.

Druhá část se zabývá genetickými algoritmy. Zpočátku jsou popsány jednotlivé části genetických algoritmů, jejich celková funkčnost a nakonec stručně popsána různá využití. To jak pro statické tak pro dynamické problémy.

V poslední teoretické části je stručně popsán problém predikce, časových řad a použití neuronových sítích právě pro predikci.

Dále jsou prezentovány samotné experimenty, jejichž cílem bylo zrychlit či zlepšit učení sítí evolučním návrhem. Bylo provedeno několik experimentů, na jejichž základě vznikla síť, která se byla schopna rychleji učit predikci kurzů měn.

Následuje kapitola zabývající se implementací. Je krátce popsána knihovna GALib, Flood a dále samotné experimenty.

V závěru jsou poté prezentovány výsledky a celkové shrnutí práce.

Kapitola 2

Neuronové sítě

Neuronové sítě jsou výpočetním modelem inspirovaným biologií, konkrétně neurony v lidském mozku. Vykazuje tyto vlastnosti, kvůli kterým je zkoumán [3, str. 9]:

- Biologické zpracování dat je robustní a k chybám odolné.
- Biologické zpracování dat je také flexibilní. Při přechodu do jiného prostředí není potřeba nijak do mozku zasahovat, člověk se na toto nové prostředí adaptuje, jinými slovy se učí.
- Tyto systémy jsou schopny přirozeně pracovat s fuzzy logikou, zvládat pravděpodobnosti, nejistotu, nekonzistentní a částečná data, což je za současného stavu výpočetní techniky možno jen s velkým množstvím programování, a i tak je to jen pro specciální případy, nikde ne obecně.
- Hardware, který všechny tyto operace provádí, je vysoce paralelní, malý, výkonný a energeticky nenáročný.

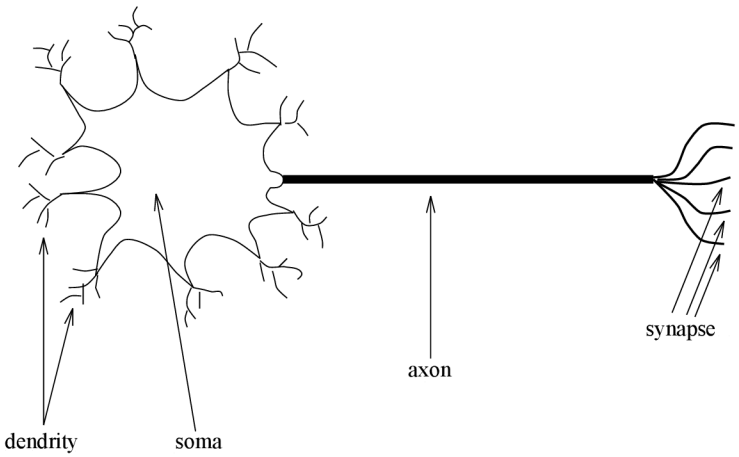
2.1 Historie

2.1.1 Biologický neuron

Na obrázku 2.1 ([5, str. 4]) je zobrazeno zjednodušené schéma biologického neuronu. Tento neuron se skládá z těla (*soma*), krátkých výběžků (*dendrity*) a jednoho dlouhého vlákna (*axon*) na konci rozvětveného do *synapsí*, které jsou napojeny na dendrity ostatních neuronů. Při fungování neuronu vždy po dendritech přicházejí (nebo nepřicházejí) vzruchy od ostatních neuronů, a pokud součet vzruchů překročí určitou *prahovou úroveň* (threshold), neuron začne po svém axonu vysílat sledy impulzů ostatním neuronům, na které je napojen. Dendrit může mít buď vzrušivý (*excitační*) nebo tlumivý (*inhibiční*) účinek. Lze předpokládat, že informace se může šířit celou řadou atributů, jako je rychlost přenosu, frekcence signálu, rozložení signálu v čase, atd. Nicméně funkce a vlastnosti neuronu jsou v biologii, psychologii a chemii stále aktivně zkoumány. Kapitola čerpá z informací v [3] a [4].

2.1.2 Jednoduchý model

Jednoduchý formální model neuronu je ukázán na obrázku 2.2. Informace jsou přijímány vstupním vektorem x, vektor w reprezentuje účinek (váhu) jednotlivých dendritů, která

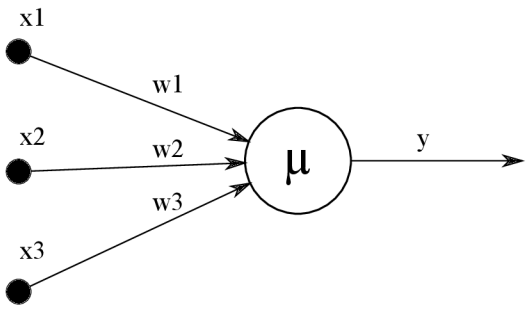


Obrázek 2.1: Biologický neuron

může být různá, a μ je prahová hodnota neuronu. Pro určení výstupu se používá vzoreček 2.1, kde funkce f je tzv. *aktivační funkce*, která je různá u různých modelů.

$$y = f\left(\sum w_i x_i - \mu\right) \tag{2.1}$$

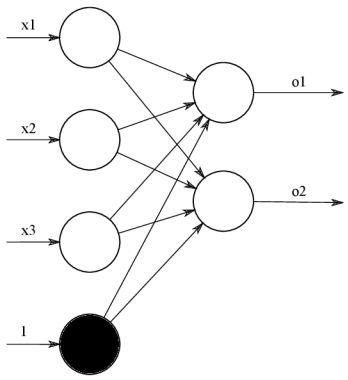
Pro jednoduchý model platí, že $f(x) = x$. Za povšimnutí stojí to, že pokud je váha dendritu kladná, lze tuto synapsi považovat za excitační, a pokud záporná, tak za inhibiční. Prahová úroveň neuronu bývá často pro zjednodušení vyjádřena vstupem $x_0 = -1$, kde pak w_0 vyjadřuje tuto prahovou úroveň. Jak tedy lze vidět, samotný neuron je extrémně jednoduchá výpočetní jednotka a podobně jako u logických členů je výpočtů dosahováno vysokých počtem jednotlivých členů - neuronů.



Obrázek 2.2: Jednoduchý model neuronu

2.2 Jednovrstvý perceptron

Nejčasteji používaným modelem neuronové sítě je *perceptron*. Jednoduchý perceptron se dá chápat jako lineární klasifikátor, který transformuje n-rozměrný vstup na m-rozměrný



Obrázek 2.3: Jednovrstvý perceptron

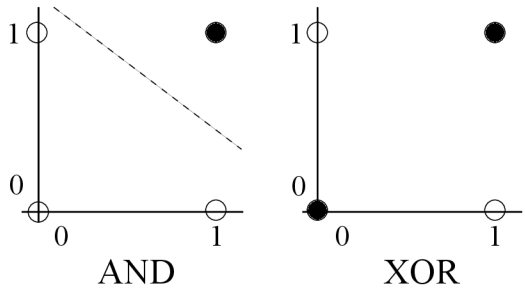
výstup, dle učebních vzorů. Na obrázku 2.3 je příklad takového jednovrstvého pertceptronu, který 3-rozměrný vstup transformuje na 2-rozměrný výstup. Pro perceptron je důležitá aktivační funkce jednotlivých neuronů, hlavně od ní se odvíjí jeho schopnosti.

2.2.1 Aktivační funkce

Každý neuron perceptronu má určitou aktivační funkci, která určuje, jak bude probíhat aktivace neuronu v závislosti na vstupech. Tyto funkce se dají rozdělit do 3 kategorií, a to *Prahová* funkce, *lineární* a *nelineární*.

Prahová aktivační funkce je vyjádřena vzorečkem 2.2, kdy neuron vrací 1 pokud suma vstupů přesáhne prahovou hodnotu. Obecně se počítá s prahem v 0, protože individuální práh je většinou formálně vyjádřen vahou nultého vstupu s pevnou vstupní hodnotou -1.

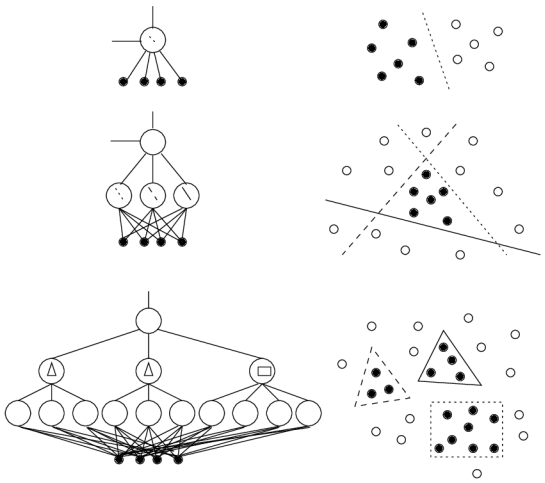
$$g(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{2.2}$$



Obrázek 2.4: Lineární separace

Jediný prahový neuron dokáže n-rozměrný prostor určený svými vstupy rozdělit *nadrovinou* na dva poloprostory (Toto se děje pomocí lineární algebry, kdy se udělá skalární součin $\vec{x}.\vec{w}$ a pokud je větší než 0, neuron zareaguje). Proto samostatný neuron umí řešit jen *lineárně separabilní* problémy, jak je ukázáno na obrázku 2.4 ([5, str. 8]).

Pokud místo jednoho neuronu (či jednovrstvé sítě) bude používa síť dvouvrstvá, lze vymezit libovolný konvexní útvar, kdy neurony první vrstvy vymezí poloprostory, neurony druhé vrstvy jejich průniky. Třívrstvá síť už dokáže vymezit libovolný počet konvexních útvarů. Názorně je to zobrazeno na obrázku 2.5 ([5, str. 9]).



Obrázek 2.5: Vícevrstvé prahové sítě

Lineární aktivační funkce (vzoreček 2.3) provádí lineární zobrazení, a pro neuronové sítě se nepoužívá. Pokud už je použita, není třeba jí učit, jednotlivé váhy lze přímo vypočítat pro potřebné lineární zobrazení.

$$g(x) = x \tag{2.3}$$

Nejčastěji používaným typem aktivační funkce je nelineární, kdy je použita tzv. *saturační funkce*, která v blízkosti nuly stoupá velmi prudce, ale dále od nuly oběma směry stagnuje, nebo má velice pozvolný průběh. Tato vlastnost byla převzata od biologických neuronů a z principu, že pro hladového člověka je jeden rohlík mnohem víc, než pro přejedeného. Jako saturační funkce se nejčastěji používá tzv. *sigmoida* (2.4). Ovšem je možno použít i jiné nelineární funkce, například sinus, který dosahuje mnohem lepších výsledků pro aproximaci periodických funkcí než sigmoida.

$$g(x) = \frac{1}{1 + e^{-\lambda x}} \tag{2.4}$$

Posledním typem používané aktivační funkce, jsou funkce popsané stochasticky - všechny předchozí byly deterministické. Z biologického hlediska se ale neurony nechovají vždy deterministicky, byl pozorován určitý pravděpodobnostní prvek v jejich chování. Pro toto chování není problém vytvořit model, kdy neuron neposílá signál dále hned po přijetí všech vstupních signálů, ale jeho pravděpodobnost poslání signálů v určitý čas závisí na aktuálních vstupech (vzoreček 2.5), kde β určuje sklon pravděpodobnostní funkce.

$$P(o_j^P = \pm 1) = \frac{1}{1 + exp(\mp 2\beta act_i^P)} \tag{2.5}$$

2.2.2 Učení

Ve své nejzákladnější podobě se perceptrony skládají z binárních prahových jednotek. Když uvážíme perceptron s jedním výstupním a N vstupními neurony, dá se říct, že perceptron se musí naučit namapovat $T : \{-1, 1\}^N \to \{-1, 1\}$. Například mapování pro 3 vstupy dané příklady:

$$\begin{aligned} T : (-1, 1, 1) &\longrightarrow \quad 1 \\ T : (1, 1, -1) &\longrightarrow \quad 1 \\ T : (1, -1, -1) &\longrightarrow -1 \end{aligned}$$

Pokud je síť naučena tomuto mapování, dává pro naučené hodnoty správné výsledky. Vlastnost, kvůli které se ale používá je, že vrátí výsledky i pro **podobné** vstupy (s problémem lineární separability zmíněným v kapitole 2.2.1). Sice se nemusí jednat o přesné výsledky, důležité ovšem je, že jsou většinou dost smysluplné. Základní algoritmus tohoto učení, jak jej prezentoval v roce 1959 Rosenblatt, vypadá takto:

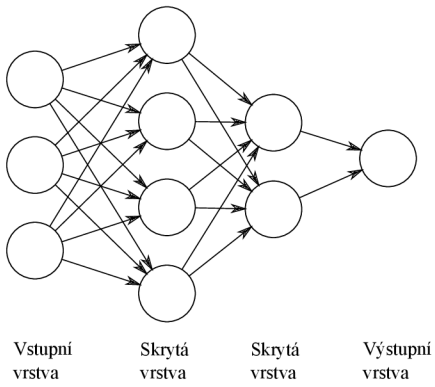
1. Všechny váhy nastav na náhodné číslo mezi -1 a 1
2. Vezmi vstupní vektor x z trénovací množiny
3. Propaguj postupně podle jednotlivých vah aktivaci dopředu, až získáš výstup (o)
4. Pokud $o^P = t^P$ (výstup je správně), pokračuj bodem **2**.
5. Jinak změň váhy podle vzorečku $\Delta w_i = \eta x_i^P (t^P - o^P)$, kde η je malé kladné číslo, kterému se říká *rychlost učení*. Pokračuj bodem **2**.

Váhy upravujeme tak dlouho, dokud síť nedává správné výsledky. Rosenblatt taktéž v roce 1959 dokázal, že algoritmus najde řešení pro mapování T v konečném počtu kroků.

2.3 Vícevrstvý perceptron

V předchozí kapitole byl popsán jednovrstvý perceptron (vstupní vrstva není započítána - neprovádí výpočet, jen přivádí vstupy, proto má dle obrázků *jednovrstvý* perceptron zdánlivě vrstvy dvě), který dokázal provádět výpočty. Nicméně jeho schopnosti jsou velice omezené. Minsky a Papert ukázali v roce 1969, že dvouvrstvá (vstupní, skrytá, výstupní) dopředná neuronová síť dokáže překonat hodně těchto omezení. Nicméně nenašli řešení toho, jak upravovat jednotlivé váhy neuronů ve skryté vrstvě. Do doby než byla nalezena odpoveď nebyly vícevrstvé perceptrony použitelné, tj. do roku 1986, kdy byl prezentován algoritmus *back propagation* [2].

Na obrázku 2.6 je vidět, co je myšleno označením *vícevrstvá dopředná síť*. Síť je uspořádána do vrstev, kdy jednotlivý neuron každé vrstvy přijímá vstupy od všech neuronů předchozí vrstvy, a posílá svůj výstup všem neuronům následující vrstvy. Neexistují žádné spoje mezi neurony jedné vrstvy, ani spoje ob-vrstvu či ještě delší. Ačkoli síť může mít více skrytých vrstev než jednu, v praxi se, kromě speciálních případů, nepoužívají. Bylo dokázáno že jedna skrytá vrstva při nelineární aktivační funkci (nejčastěji sigmoida) dokáže aproximovat libovolnou funkci.



Obrázek 2.6: Perceptron

2.3.1 Algoritmus back-propagation

Mechanismus back-propagation vychází z myšlenky, že síť dostane vstup a transformuje jej na výstup. Pokud se tento výstup nerovná hodnotě, které by se měl rovnat (dle trénovací množiny), spočítá se chyba a síť zpětným šířením upraví své váhy. Algoritmus obecně pak vypadá takto [3, str. 39]:

1. Všechny váhy nastav na náhodná malá čísla
2. Vezmi vstupní vektor x z trénovací množiny
3. Propaguj postupně podle jednotlivých vah aktivaci dopředu, až získáš výstup (o)
4. Vypočítej δs pro výstupní vrstvu $\delta_i^P = (t_i^P - o_i^P) f'(Act_i^P)$
5. Vypočítej δs pro každou skrytou vrstvu $\delta_i^P = \sum_{j=1}^{N} \delta_i^P w_{ji} f'(Act_i^P)$
6. Změň všechny váhy dle $\Delta_p w_{ij} = \gamma \delta_i^P o_i^P$
7. Opakuj kroky 2 až 6 pro každý vstupní vektor

Praktická aplikace tohoto algoritmu je pro názornost uvedena v příloze A

2.3.2 Využití

Vícevrstvé perceptronové sítě lze využít různými způsoby, někdy i dost nečekanými. Jak už bylo řečeno, používají se hlavně v případech, kdy není znám algoritmus, či je ho potřeba vytvořit ze sad příkladů. Následuje pár typických a zajímavých případů využití z [5, str. 15–16].

Rozpoznávání obrazu

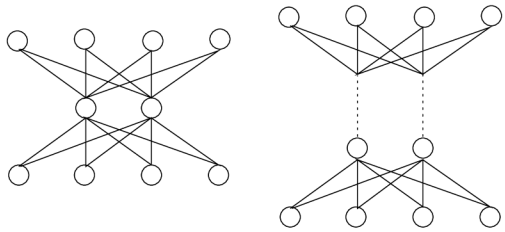
Neuronovou síť je možno použít na rozpoznávání obrazu. Například na poštách – na rozpoznávání PSČ – byl aplikován systém s takovouto sítí, u které nějakou dobu seděl pracovník, a učil jí tím, že sám rozpoznával písmo, a zadával síti cílový výstup. Také se používá na klasifikaci obrazu z kamery robotů, a naučení kam mají jezdit, zatáčet, zpomalovat. Byl i případ, kdy se neuronová síť použila v ponorkách na rozpoznávání nevybuchlých min z druhé světové války.

Predikce časových řad

Umělá neuronová síť se velmi často používá pro predikci číselných řad, kdy je na vstup přivedeno n předchozích hodnot (plovoucí okno), a síť předpoví další hodnotu. Konkrétně toto použití je demonstrováno v následujících kapitolách.

Komprese dat

Zajímavé využití neuronových sítí je na kompresi dat. Ta se provádí tak, že se vezme síť se stejným počtem vstupů a výstupů a jednou skrytou vrstvou, ve které je nižší počet neuronů než na vstupu/výstupu (obrázek 2.7) a je učena na vracení na výstup identického se vstupem. Síť je poté rozdělena tak, aby první polovina transformovala vstup do skryté vrstvy, a druhá polovina skrytou vrstvu na výstup. Jako komprimovaná data jsou přenesena data o skryté vrstvě, která má méně neuronů než vrstva vstupní.



Obrázek 2.7: Komprese pomocí neuronové sítě

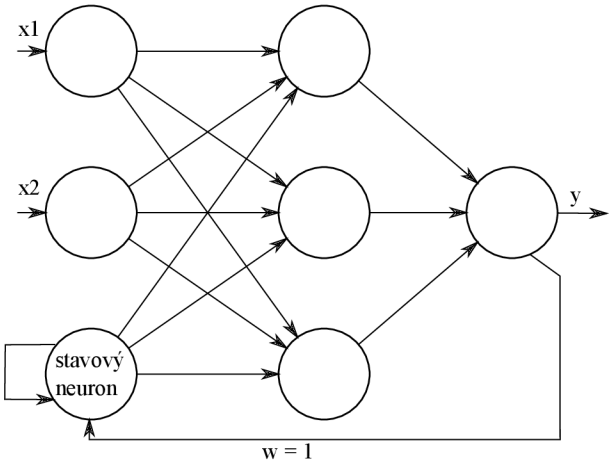
2.4 Rekurentní sítě

Rekurentní sítě ([4, str. 47-56]) jsou neuronové sítě, které vzniknou přidáním nějakého *zpětného* spoje do dopředné vícevrstvé sítě. Ačkoli se schopnosti sítě nezvýší, lze takto redukovat její komplexitu či velikost (což může být někdy velice významný faktor, ovlivňující jak cenu, tak výkon).

Pro učení těchto sítí lze jednodušše použít algoritmus *back-propagation* pro dopředné sítě, ale je poté potřeba vyřešit, co se bude dít při zpětném šíření chyby – díky zpětnému spoji je možno pokračovat donekonečna, či skončit po dosažení určitého ustáleného stavu. Typický případ pro použití rekurentní sítě je situace, kdy máme jako vstupy po sobě jdoucí členy časové řady $(x(t), x(t-1), x(t-2), ..., x(t-n))$. To můžeme řešit dvěma způsoby:

1. Vytvořit vstupy $x_1, x_2, ..., x_n$ na které budou přivedeny jednotlivé členy časového okna.
2. Vytvořit vstupy $x, x', x''...$, kdy kromě vstupu x(t) se budou zpátky vracet již vypočítané hodnoty z minulých hodnot.

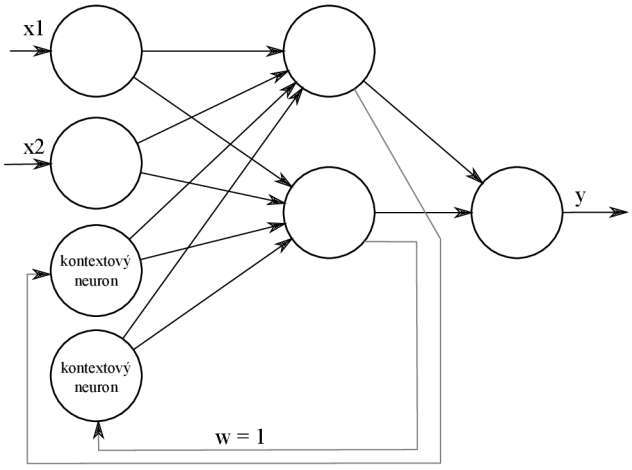
Výhodou druhé metody je velké snížení velikosti sítě. Pro velké časové okno by se síť navržená první metodou stala obrovskou, což by snižovalo výkon. Proto je vhodné použít Jordanovu nebo Elmanovu rekurentní síť.



Obrázek 2.8: Jordanova síť

2.4.1 Jordanova a Elmanova síť

Jednou z prvních rekurentních neuronových sítích byla síť Jordanova (1986). Příklad takové sítě je na obrázku 2.8, kde výstup ze všech výstupních neuronů je také veden zpátky do speciálních *stavových* neuronů. Tyto neurony mají také reflexivní spoj samy na sebe. Váhy těchto spojení jsou nastaveny konstantně na 1 a neuplatňuje se na ně učení. Proto je možné použít back-propagation beze změny (změny vah se aplikují na spoje, které spojují stavový neuron s neurony ze skryté vrstvy).



Obrázek 2.9: Elmanova síť

Elmanova síť (obrázek 2.9) byla představena v roce 1990. Tato síť je podobná Jordanově síti, nicméně jsou zde podstatné rozdíly.

1. Stavové neurony se jmenují *Kontextové*

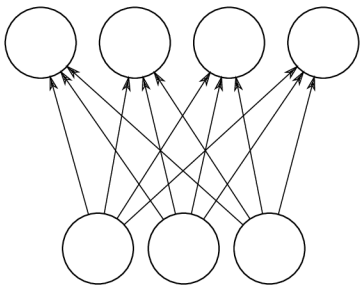
2. Místo z výstupních neuronů jsou do kontextové vrstvy vedeny výstupy (poslední) skryté vrstvy

3. Kontextové neurony nemají reflexivní spojení samy na sebe

2.5 Sítě s učením bez učitele

Prozatím byly popisovány sítě, které byly trénovány, aby se naučily určité mapování pomocí příkladů. Existují ovšem situace, kdy neexistují ke vstupům správné výstupy na trénování a veškeré informace je třeba vytěžit pouze ze vstupů. Příkladem takových situací je třeba shlukování, kdy je potřeba zjistit, zda se ve vstupech vyskytují shluky a identifikovat je.

Příkladem učení bez učitele je konkurenční učení, jehož jenoduchý příklad je na obrázku 2.10. Každý vstup je spojen s každým výstupem přes váhy. Konkurenční proto, že oproti perceptronům je aktivován vždy jen jediný výstupní neuron, který značí příslušnost vstupu do určitého shluku. Síť je pak trénována k tomu, aby byl vstup zařazen k nejbližšímu shluku. Počet shluků, na které umí síť vstupy rozdělit, je dán počtem výstupních neuronů, a tak se tento problém obvykle řeší dynamickým přidáváním/odebíráním neuronů za chodu podle potřeby (ne vždy je známý počet shluků/kategorií v datech).



Obrázek 2.10: Síť pro konkurenční učení

Kapitola 3

Genetické algoritmy

Genetické algoritmy jsou stochastické metody ze skupiny evolučních algoritmů (mezi které patří mimo jiné například genetické programování, evoluční strategie, evoluční programování) používané převážně pro optimalizaci. Metoda je inspirována biologií, konkrétně evolucí (Darwin, genetika, přirozený výběr, ...), kdy existuje množina různě dobrých řešení (populace), která jsou určitým způsobem zakódována (genetický kód). Poté jsou v populaci zvýhodňována lepší (silnější, životaschopnější) řešení oproti horším a vhodný genetický materiál je pomocí genetických operátorů mixován a přenášen do další generace. Tato metoda je vhodná pro prohledání velkého stavového prostoru v relativně krátkém čase s použitelnými výsledky (ve většina případů nezaručuje optimálnost řešení). Kapitola čerpá z informací z [11] a [9].

3.1 Části GA

Pokud je pomocí GA řešen určitý problém či optimalizace, je nutné sestavit tzv *účelovou funkci*, což je zakódování problému obecně do N parametrů. Poté se GA snažíme najít globální extrém (minimum či maximum) této N-rozměrné funkce. *Fitness funkce* je funkce hodnotící úspěšnost jedince v populaci, a to tak, že nejlepší jedinec má nejvyšší[1] hodnotu fitness funkce. Občas je používána nebo přímo vyžadována ještě *normalizovaná fitness funkce*, což je fitness funkce přeškálovaná do intervalu $x \in < 0; 1 >$. Celkovou normalizaci lze aplikovat pouze pokud známe obor hodnot fitness funkce, jinak pouze v rámci jedné generace. Algoritmus obecného GA by pak vypadal v pseudokódu následovně [11, str. 13]:

1. Nastav $t = 0$, náhodně generuj počáteční populaci $P(0)$ s mohutností N.
2. Proveď ohodnocení jedinců populace $P(t)$ fitness funkcí.
3. Generuj populaci potomků $O(t)$ s mohutnosti $M \leq N$ použitím operátorů křížení (3.1.3) a mutace (3.1.4).
4. Vytvoř novou populaci $P(t+1)$ nahrazením části populace $P(t)$ jedinci z $O(t)$.
5. Nastav $t \leftarrow t + 1$.
6. Pokud není splněna podmínka ukončení algoritmu (3.1.5), jdi na **2**.

[1] Záleží na směru konvergence. Vyšší hodnota fitness funkce pro lepší jedince je stanovena dohodou. Proto při opačné potřebě se používá přepočítání ve stylu $1/x$.

3.1.1 Zakódování do chromozomu

Zakódování problému do genů a následně do chromozomu je klíčová část GA. Je třeba jednotlivá řešení zakódovat tak, aby mohla vystupovat jako jedinci populace. Obvykle je řešení, či optimalizační problém charakterizován vektorem parametrů, a jeho zakódování by mělo ideálně vykazovat následující vlastnosti [9, str. 9]:

- **kauzalita** - jeden konkrétní chromozom představuje právě jedno řešení problému
- **legalita** - všechny možné kombinace zakódování by měly být platná a legální řešení daného problému (například nemělo by se vyskytnout řešení nepatřící do definičního oboru funkce)
- **kompletnost** - zakódování by mělo pokrývat všechna možná řešení problému
- **neredundance** - ne přímo nutná, ale žádoucí vlastnost, neexistují dvě různá zakódování stejného řešení.

Binární kódování

Binární kódování patří mezi nejstarší a nejpoužívanější. Hlavní výhodou je přirozená implementace v počítači, a tak lze nad nimi implementovat rychlé a efektivní operátory. Každý gen tvoří posloupnost binárních číslic, načež chromozom je tvořen za sebou vyskládanými binárními posloupnostmi jednotlivých genů, jak je znázorněno na obrázku 3.1. Pro toto kódování je velice vhodný inverzní mutační operátor.

$$\begin{aligned} g_1 &= 00011111_b = 31_d \\ g_2 &= 00100000_b = 32_d \\ &\Downarrow \\ ch &= 0001111100100000 \end{aligned}$$

Obrázek 3.1: Binární zakódování

Jiná kódování

V určitých případech lze úspěšně použít jiné, lépe využitelné kódování. Například *reálné kódování*, kdy jsou jednotlivé geny zakódovány jako čísla s plovoucí desetinnou čárkou (datový typ float) či *permutační kódování*, kdy se chromozom skládá z fixního počtu číslic reprezentujících pořadí provádění určitých akcí (výrobní linka, problém obchodního cestujícího). Tato kódování obvykle vyžadují změny v mutačních a křížících operátorech.

3.1.2 Selekce

Operátor výběru vytváří novou populaci $P(t+1)$, kde se mohou, dle nastavení algoritmu buď vyskytovat jedinci z předchozí populace ($P(t)$), nebo nemohou. Existuje několik používaných metod výběru, které se používají v různých situacích, a významně ovlivňují konvergenci a rychlost GA. Je potřeba na jednu stranu dostatečně zvýhodňovat jedince s vyšší fitness funkcí, ale na druhou stranu je potřeba neztrácet diverzitu populace. V prvním případě by

algoritmus velice pomalu konvergoval, a v druhém by uvízl v nejbližším lokálním extrému. Proto byl zavedem pojem *selekční tlak*, který je vyjádřen vztahem 3.1, kde $\overline{M}$ označuje průměrnou hodnotu fitness funkce v populaci před selekcí, $\overline{M^*}$ průměrnou fitness po selekci a $\overline{\sigma}$ rozptyl fitness funkcí před selekcí.

$$I = \frac{\overline{M^*} - \overline{M}}{\overline{\sigma}} \tag{3.1}$$

Proporcionální selekce (roulette wheel selection)

Tzv. ruletový výběr byl prvním algoritmem používaným pro selekci, kdy je pravděpodobnost výběru přímo úměrná fitness funkci. Název je odvozen od podobnosti s koláčovými grafy a ruletou, kde mají jedinci větší výsek, na kterém je možno se zastavit, a tudíž i větší šanci aby na nich bylo zastaveno. Toto však může působit i negativně na celý GA, pokud se v populaci vyskytne jedinec s výrazně vyšší fitness funkcí, než ostatní. V takovém případě dojde časem k nahrazení celé populace tímto jedním jedincem. Proto se nejčastěji používají dvě metody, aby se zmenšil rozdíl mezi nejlepším a nejhorším jedincem populace.

1. **Komprimace fitness funkce:** $f'(i) = f(i) + \beta'$, kde β je nejhorší fitness v aktuální generaci
2. **Sigma škálování:** $f'(i) = max(f(i) - (< f > -c * \sigma_f), 0.0)$, kde c je konstanta, obvykle 2.0, a $< f >$ je střední hodnota fitness funkce

Lineární a exponenciání uspořádání (ranking)

Lineární uspořádání vyžaduje seřazenou populaci dle fitness funkce sestupně. Poté je pravděpodobnost výběru dána vztahem 3.2 kde s označuje selekční tlak. Exponenciální se liší pouze rozložením pravděpodobnosti v populaci.

$$P(i) = \frac{2 - s}{N} + \frac{2i(s - 1)}{N(N - 1)} \tag{3.2}$$

Turnajová selekce

Tato metoda má výsledky srovnatelné s předchozí metodou, největším přínosem je však absence požadavku na seřazení populace. Z tohoto důvodu je tato metoda poměrně často používána. Z populace je vždy vybráno x jedinců, kteří spolu simulovaně svedou souboj (záleží na implementaci fitness funkce. Tato metoda dokáže pracovat i s realativně určinou fitness funkcí), a do další generace postoupí vždy vítězný jedinec.

3.1.3 Křížení

Druhý charakteristický operátor pro GA je operátor křížení (crossing-over), který přestavuje primární operátor pro evoluci populace. Je to proces, při kterém z genomů dvou jedinců z populace vzniká další jedinec (jedinci) s rozdílným genomem, dle užité metody namixovaným od obou rodičů. Existují také GA bez tohoto operátoru, či s určitou pravděpodobností jeho provedení. Pro binárně zakódované chromozomy se používají metody křížení, kdy jsou genomy rodičů rozděleny obecně na N místech a genom potomka je vždy v každém místě brán od jiného rodiče. Podle počtu rozdělení rozlišujeme křížení (znázorněná na obrázku 3.2):

- **Jednobodové** - $N = 1$
- **Dvoubodové** - $N = 2$
- **Vícebodové** - $N > 2$
- **Uniformní** - $N = random()$, každý bit potomka je vzat náhodně z jednoho nebo druhého rodiče.

Žádné	$\{X, X, X, X, X, X, X\}$	$\{O, O, O, O, O, O, O\}$
Jednobodové	$\{X, X, X, O, O, O, O\}$	$\{O, O, O, X, X, X, X\}$
Dvoubodové	$\{O, X, X, O, O, O, O\}$	$\{X, O, O, X, X, X, X\}$
Uniformní	$\{X, O, X, X, O, O, X\}$	$\{O, X, O, O, X, X, O\}$

Obrázek 3.2: Druhy křížení

3.1.4 Mutace

Další charakteristický operátor je operátor mutace. Jedná se o operátor s velmi malou pravděpodobností výskytu (nejčastěji $p_m \in < 0.0005 - 0.01 >$), ale s velkým významem. Přináší do populace nové bloky informací. Při velké pravděpodobnosti mutace začne být GA nestabilní, při malé či žádné bude pomalu, a nebo zcela přestane, konvergovat. Pro binárně zakódované chromozomy se nejčastěji používá:

- **běžná mutace** - je vybrán náhodný index bitu, a ten je invertován.
- **inverzní mutace** - Podobně jako u jednobodového křížení je vybráno místo, a od tohoto místa jsou všechny bity invertovány. Lze použít i vícebodovou inverzní mutaci.

Běžná mutace	$\{1, 1, 1, 1, 1, 1, 1\}$	$\Longrightarrow$	$\{0, 0, 0, 1, 0, 0, 0\}$
Jednobodová inverzní mutace	$\{1, 1, 1, 1, 1, 1, 1\}$	$\Longrightarrow$	$\{1, 1, 1, 0, 0, 0, 0\}$
Dvoubodová inverzní mutace	$\{1, 1, 1, 1, 1, 1, 1\}$	$\Longrightarrow$	$\{1, 0, 0, 0, 1, 1, 1\}$

Obrázek 3.3: Druhy mutací

3.1.5 Ukončující podmínky

Jako ukončující podmínku/y GA lze zvolit například jednu z těchto možností:

- **počet generací** - udáme počet generací po jehož dosažení GA skončí, a za řešení prohlásíme nejlepšího jedince poslední generace.
- **účelová funkce nejlepšího jedince** - pokud ohodnocení nejlepšího jedince v jakékoli generaci překročí danou hranici, algoritmus končí a řešením je tento jedinec. Za jistých podmínek ovšem nemusí algoritmus skončit.

- **minimální změna účelové funkce** - algoritmus končí, když průměrná změna účelové funkce v populaci dosáhne určité hranice (používají se různé statistiky).
- **časové omezení** - algoritmus končí po uplynutí určeného času.

3.2 Využití GA

Kromě klasických problémů popsaných v předchozí kapitole, lze GA úspěšně využít ještě v dalších typech úloh a použít různé modifikace.

3.2.1 Multimodální problémy

Mnohdy má problém jedno globální maximum a více různých lokálních. U klasické GA mají jednotlivci tendence uváznout v těchto lokálních maximech, a proto se používají modifikace GA pro zlepšení chování v těchto situacích. Obecným smyslem všech používaných metod je udržet velkou diverzitu populace, aby se neshlukovala jen kolem jednoho maxima, které může, ale také nemusí být globálním.

Jedna z takových metod je *Sdílená fitness funkce*, která je inspirovaná omezenými zdroji v určitém prostředí. Pokud se tedy v jednom místě sejde více jedinců, všem se sníží fitness funkce. To má za následek lepší rozložení jedinců ve stavovém prostoru. Nicméně existují dva zásadní problémy. Kvadratická náročnost výpočtu, a poté nutnost přibližné znalosti počtu lokálních extrémů k vhodnému nastavení „zdrojů“ v rámci shluků.

3.2.2 Dynamické problémy

Existují také optimalizační úlohy, které nejsou statické. Může se měnit účelová funkce, cílové parametry, či omezující podmínky prostředí. Jedná se například o vypadnutí výrobní linky a dynamické přeplánování výroby, aby se zminimalizoval efekt její nefungující části. Pro tyto případy se většinou používají 3 přístupy či jejich kombinace:

1. GA běží standardně, ale jakmile je zaznamenána změna prostředí, jsou podniknuty kroky ke zvýšení diverzity populace.
2. Násilné oddalování konvergence, aby populace zůstávala poměrně hodně divergentní.
3. Přidána paměť, kdy si buď jedinec či celý GA zachovává užitečné informace o minulosti. Velice vhodné, pokud se podmínky mění periodicky.

Restart

Nejjednodušší řešení je znovu spustit celý GA, s případným zahrnutím několika jedinců předchozí populace (kdyby nové optimum bylo poblíž starého). Používá se hlavně při změně kódování problému či chromozomu.

Adaptace mutačního operátoru

Vychází z myšlenky, že při změně prostředí je třeba přinést více nových informací do populace, tudíž dočasně zvýšit pravděpodobnost mutačního operátoru. Sem spadá i metoda náhodných imigrantů, což lze považovat za velice silnou mutaci na části populace.

Paměť

Použití metody s pamětí se velice osvědčuje v periodicky měnících se prostředích. Při použití explicitní paměti je založena jedna nebo více pamětí pro předky každého jedince. Po změně prostředí jsou ohodnoceni i archivovaní jedinci, a pokud je některý lepší než aktuální jedinec, nahradí se jím. Speciální případ použití paměti jsou *stráže*, což jsou pravidelně rozmístění jedinci ve statovém prostoru, jejichž genotyp se nemění. V případě objevení se optima blízko nich jsou pak schopni populaci rychle přitáhnout.

Multipopulační techniky

Tyto techiky jsou založeny na několika víceméně oddělených populacích. Například po objevení lokálního extrému je v něm zanechána subpopulace, a zbytek dále prohledává stavový prostor. Případně opačný postup, kdy hlavní populace je soustředěna okolo dosavadního nejlepšího řešení, a subpopulace jsou rozmístěny v lokálních optimech.

3.2.3 Omezující podmínky

Pokud při kódování chromozomu není splněna podmínka legality z kapitoly 3.1.1 nebo jí nelze splnit, i tak lze použít GA. Pro tyto případy se používají nejčastěji 3 metody:

- **Pokutové funkce** - pokud je řešení nepřípustné, je jeho fitness funkce "pokutována", tj. snížena, a to tím více, čím více porušuje podmínky.
- **Opravné procedury** - při nepřípustném řešení je vyhledán nejbližší přípustný jedinec, a dále je buď původní jedinec nahrazen tímto nejbližším přípustným, nebo je ponechán a od nejbližšího je převzata pouze hodnota jeho fitness funkce.
- **Problémově specifické zakódování** - záměrem je vyhnout se všem nepřípustným jedincům návrhem zakódování problému a operátorů tak, aby v žádné jejich kombinaci nemohlo dojít k nepřípustnému řešení. Vyžaduje to ovšem znalost problému do hloubky a ne vždy je to možné.

Kapitola 4

Predikce časových řad

Předpověď kurzů měn, teplot, či jiných časově závislých řad je důležitý a používaný způsob pro podporu rozhodování. Předpověď budoucího členu či členů řady nazýváme *predikce*. Nejedná se o přesnou předpověď, protože nejsou vzaty v potaz všechny souvislosti, na kterých je řada závislá. Nicméně je možnost z minulých hodnot odhadnout budoucí vývoj. Čím delší budoucnost, tím nepřesnější odhad. Je tedy možno řadu brát jako stochastický proces a vytvořit model (kvantitativní metoda), nebo vzít v úvahu expertní znalosti (kvalitativní metoda). U kvantitativních metod vše záleží na předpokladu, či spíše pravděpodobnosti, že model řady zůstane stejný i v budoucnosti, nebo aspoň výrazně podobný. U kvalitativních zase záleží na dotyčných expertech, tj na subjektivních měřítkách. Kapitola je založena na informacích z [8] a [1].

4.1 Časová řada

Časová řada je chronologicky uspořádaná posloupnost hodnot určitého statistického ukazatele. Tento ukazatel musí být v čase vymezen věcně a prostorově shodně. Matematicky řečeno, časová řada je posloupnost jednotlivých funkčních hodnot funkce, která závisí na čase, v určitých bodech. Mezi těmito body je zpravidla shodná vzdálenost. Typickým případem by například byl kurz měny, teplota místa či výška hladiny v určitém časovém období.

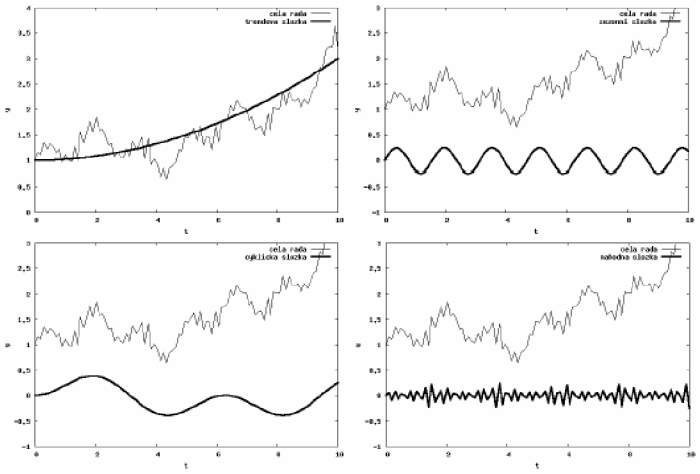
Časové řady se dají dělit dle několika hledisek, zejména dle již zmíněné vzdálenosti hodnot. Pokud jsou vzdálenosti stejné, řada se nazývá *ekvidistantní*, jinak *neekvidistantní*. Další dělení je podle informací, které máme o systému. Pokud známe funkci, dle které je řada generována, a můžeme kdykoli přesně rekonstruovat řadu v jakémkoli období, jedná se o řadu *deterministickou*. Pokud taková funkce neexistuje nebo jí neznáme, musíme popisovat *stochasticky*. Existují další dělení a kategorizace řad, ty jsou však důležité až ve speciálních případech.

4.2 Analýza časových řad

Klasická analýza časových řad spočívá v konstrukci modelu časové řady, který jí generuje, a následně předpověď (*predikci*) dalších možných hodnot této řady. Základní přístup k takovéto analýze je *dekompozice* na čtyři složky, které je pak jednodušší analyzovat. Tyto složky jsou a znázorněné na obrázku 4.1:

- **Trendová složka** - Dlouhodobá tendence řady, ve smyslu „řada má tendence mírně stoupat“.

- **Sezónní složka** - Cyklická složka s pravidelnou periodou, menší než je délka řady.
- **Cyklická složka** - Dlouhodobá cyklická složka s proměnlivou délkou periody. Někdy zahrnuta v trendu, když není tak lehce pozorovatelná.
- **Reziduální složka** - Složka, která „zbyde“ po dekompozici na tři předešlé, někdy se jí též říká šumová složka.



Obrázek 4.1: Trendová, sezónní, cyklická a reziduální složka časové řady [1, str. 5]

Pokud v řadě chybí některé hodnoty, je většinou třeba tyto doplnit - další metody zpracování často dokáží pracovat jen s úplnými řadami, které mají konstantní krok. Dopočítat další hodnoty je též v takových případech nutno u řad s nekonstatním krokem pro další zpracování. Pro takové situace se používají metody, které chybějící hodnoty nahradí nulami, průměrem, interpolací atd.

4.3 Kvalita predikce

Podstatnou roli v predikci hraje určování míry chybovosti, ať už pro určení spolehlivosti předpovědi, či porovnávání úspěšnosti různých metod. Největším zdrojem chyby bývá, při dobře provedené analýze, reziduální složka řady, i když velká chyba může být způsobena chybným modelem ostatních, nereziduálních, složek. Chyba (ε) obecně je definována jako rozdíl předpovězené (b) a skutečné (a) hodnoty (vzoreček 4.1), kterou většinou v danou chvíli neznáme. Proto je možné posuzovat kvalitu předpovědi až zpětně, či použít známou množinu dat a předpovídat už známé hodnoty.

$$\varepsilon = |a - b| \tag{4.1}$$

Existuje více statistických metod určování chyby, a pro časové řady se nejčastěji používají metody popsané níže. Všechny jsou úspěšně použitelné při porovnávání metod predikce na stejných datech, a konkrétní volba záleží na tom, jaké chyby se hledají. Hledat lze menší počet větších chyb, nebo naopak velký počet relativně malých chyb. Pro porovnávání na různých datech je však třeba použít relativní míru.

- **odmocnina ze střední čtvercové chyby** (Root Mean Squared Error)

$$RMS = \sqrt{\sum_{t=1}^{n} \frac{\varepsilon_t^2}{n}} \tag{4.2}$$

- **střední čtvercová chyba** (Mean Squared Error)

$$MSE = \sum_{t=1}^{n} \frac{\varepsilon_t^2}{n} \tag{4.3}$$

- **součet čtvercových chyb** (Sum of Squared Error)

$$SSE = \sum_{t=1}^{n} \varepsilon_t^2 \tag{4.4}$$

- **střední absolutní odchylka** (Mean Absolute Deviation)

$$MAD = \sum_{t=1}^{n} \frac{|\varepsilon_t|}{n} \tag{4.5}$$

4.4 Predikce pomocí neuronových sítí

Další možný přístup ke kvantitativní predikci časových řad je použití neuronových sítí, které implicitně provedou analýzu řady, a naleznou i skryté nelineární závislosti. Toho se využívá pokud z nějakého důvodu nelze řadu analyzovat výše popsaným způsobem. Takovouto síť je možno navrhnout dvěma způsoby:

- **dedukcí** - Na počátku je vybrána struktura modelu, a poté se chování sítě přizpůsobuje časové řadě úpravou koeficientů.
- **indukcí** - Na počátku máme základní stavební jednotky, a struktura sítě se postupně vytváří na základě učení tak, aby co nejlépe odpovídala modelované řadě.

Jako jednoduchý příklad predikce pomocí neuronových sítí pravděpodobně velice intuitivně vypadá rekurentní síť, která si „pamatuje“ předchozí vstupy. Nicméně vzhledem k problémům s učením rekurentních sítí je ještě jednodušší použití klasické dopředné vícevrstvé sítě, která ovšem nebude mít jeden vstup, ale tolik vstupů, jak velké bude časové okno pro predikci.

4.4.1 Časové okno

V případě použití dopředné neuronové sítě se data upravují způsobem, který jim umožní pracovat s predikcí časových řad. Při predikci je použito plovoucí časové okno o velikosti N, které se posouvá po vzorcích. Na vstup sítě je tedy dodán vektor obsahující N za sebou jdoucích hodnot. Jako výsledek je pak brána hodnota (či více hodnot) následující. Toto se děje buď za běhu, nebo se data předem upraví do požadovaného formátu.

$$
\begin{aligned}
t_1 &: \overline{x_1, x_2, x_3}, x_4, x_5, x_6, x_7, x_8, x_9 \\
t_2 &: x_1, \overline{x_2, x_3, x_4}, x_5, x_6, x_7, x_8, x_9 \\
t_3 &: x_1, x_2, \overline{x_3, x_4, x_5}, x_6, x_7, x_8, x_9 \\
t_4 &: x_1, x_2, x_3, \overline{x_4, x_5, x_6}, x_7, x_8, x_9 \\
t_5 &: x_1, x_2, x_3, x_4, \overline{x_5, x_6, x_7}, x_8, x_9 \\
t_6 &: x_1, x_2, x_3, x_4, x_5, \overline{x_6, x_7, x_8}, x_9 \\
t_7 &: x_1, x_2, x_3, x_4, x_5, x_6, \overline{x_7, x_8, x_9}
\end{aligned}
$$

Obrázek 4.2: časové okno pro řadu o prvcích $x_1 - x_9$ pro časy $t_1 - t_7$

Kapitola 5

Experimenty

Cílem experimentů bylo navrhnout neuronovou síť, která bude typické finanční časové řady (kurzy měn) predikovat s lepší přesností, či se predikci lépe a rychleji učit. Nejdříve se navrhla strukturu sítě, velikost a časové okno pro predikci. Poté jsou popsány experimenty s různým nastavením koeficientů inicializované neuronové sítě, protože to, jak se při náhodné inicializaci nastaví jednotlivé váhy, má podstatný vliv na konvergenci učení neuonové sítě.

5.1 Experimentální data

Jako experimentální data byly vybrány různé řady o délce 254, což byl počet záznamů kurzu měny za jeden rok na stránkách ČNB. V simulacích je použito vždy jen **200** údajů, zbylé jsou rezervovány pro vytváření časového okna[1]. Data lze rozdělit do několika kategorií. V první kategorii je několik řad reálných kurzů měn, pro které je cílem navrhnout optimální neuronovou síť pro predikci. V dalších dvou jsou vygenerované hodnoty několika známých funkcí a náhodných šumů, pro které bude následně srovnávána predikce sítí, naučených na kurzech měn a opačně. To kvůli srovnání, zda je síť opravdu navrhnuta, aby lépe predikovala kurzy oproti jiným typům řad. Grafy použitých řad jsou v příloze B.

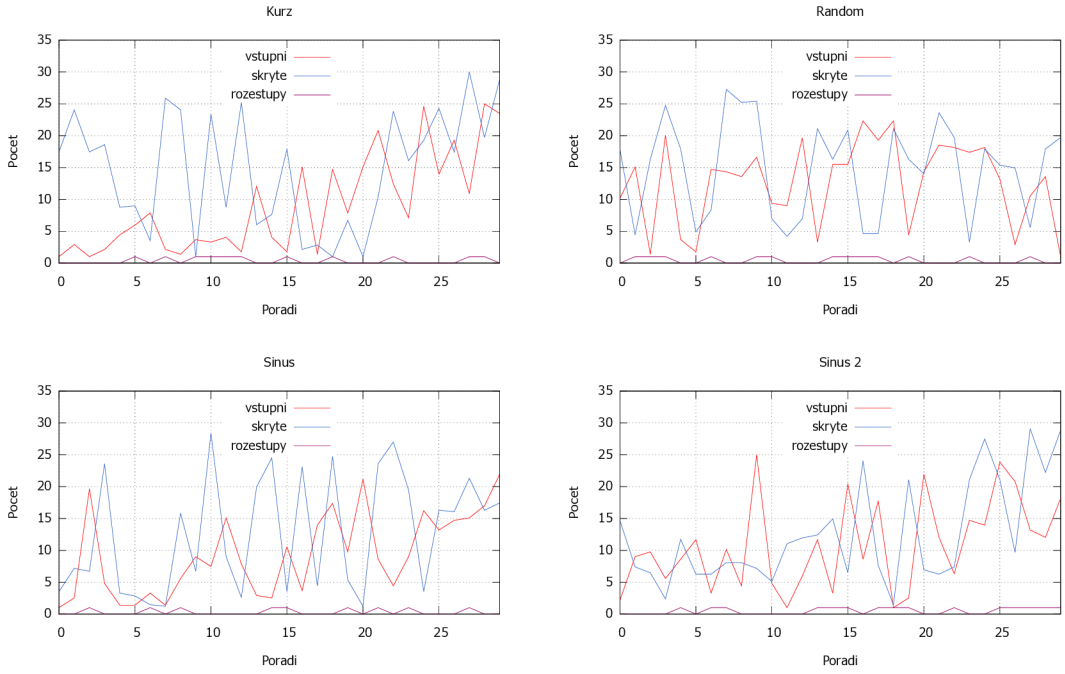
5.2 Velikost neuronové sítě

V tomto experimentu se zkoumalo, jaká velikost neuronové sítě (počet vstupních a skrytých neuronů) by byla vhodná pro predikci řad. Prvně bylo třeba získat alespoň přibližnou představu o chování systému, načež mohl být proveden podrobnější experiment. Provedeny byly tedy jednotlivé experimenty na čtyřech různých vzorcích dat (kurz britské libry, náhodný šum, jedna perioda funkce sinus, dvě periody funkce sinus) s následujícími parametry:

- Neuronová síť: **třívrstvý perceptron**
- Data: **kurz Velké Británie**, **náhodný šum**, **jedna perioda funkce sinus**, **dvě periody funkce sinus**
- Počet trénovacích/testovacích vzorků: **150**/**50**

[1] Aby všechna řešení v experimentu s velikostí měla stejné podmínky, tj. stejný počet trénovacích dat, nejde udělat trénovací data ze všech 254 vzorků. Při různé velikosti časového okna (vstupů) by se počet lišil. Proto bylo zvoleno 200 vzorků, za prvé je to přirozeně kulaté číslo, a za druhé nebude experimentováno s časovým oknem větším než 54.

- Velikost populace: **30**
- Počet generací: **30**
- Pravděpodobnost mutace: **0.1**
- Pravděpodobnost křížení: **0.9**
- Fenotyp: vstupní neurony **1–25**, skryté neurony **1–30**, rozestupy **0–1**
- Selekce: **Ruletový výběr**
- Mutace: **Jednobodové inverzní křížení**
- Aktivační funkce skryté vrstvy: **Hyperbolický tangens**
- Aktivační funkce výstupní vrstvy: **Lineární**
- Počáteční nastavení vah: **Normální rozložení**
- Maximální počet trénovacích epoch: **200**
- Maximální doba učení: **5** s



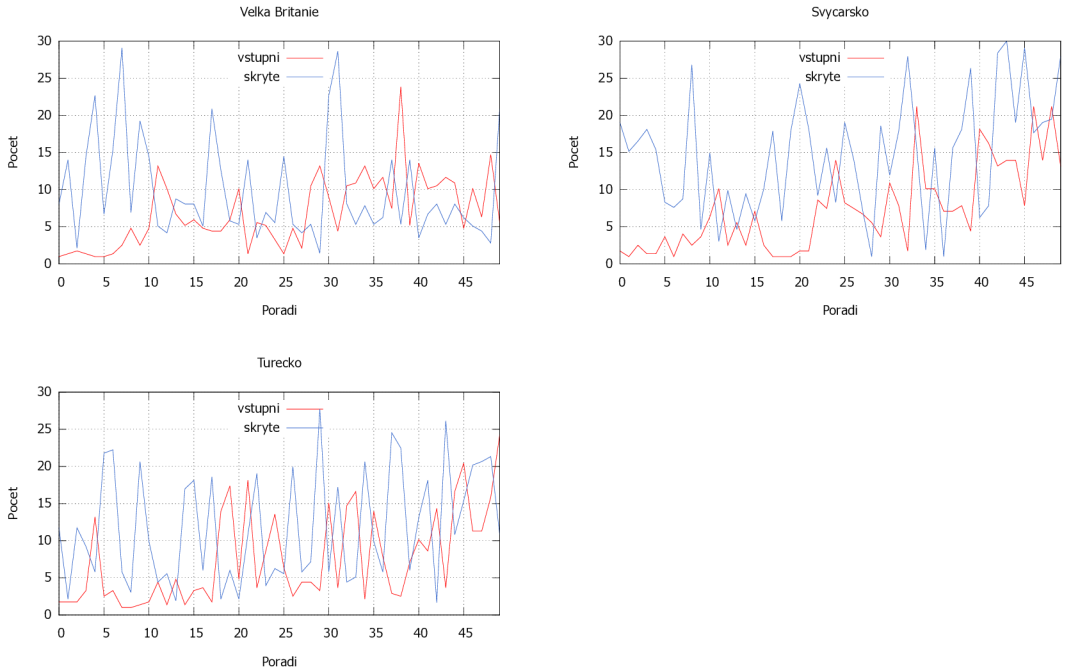
Obrázek 5.1: Výpis řešení v populaci od nejlepšího po nejhorší

Výsledky experimentu jsou zobrazeny na grafech 5.1, kde na ose X je pořadí řešení v populaci, a různé barvy čar znázorňují parametry těchto řešení. Je vidět, že na předních pozicích u kurzu libry se umístila řešení poměrně s malým počtem vstupních neuronů a poměrně velkým počtem skrytých. Naopak populace u náhodného šumu je přibližně stejná na

všech pozicích, což jen potvrzuje intuitivní závěr, že pro predikci náhodného šumu je v podstatě jedno, jak velké je časové okno. U predikce funkce sinus jsou také patrná spíše řešení s malým počtem vstupních neuronů na předních pozicích populace, nicméně už s ne tak velkým počtem skrytých neuronů. Parametr rozestupy nevykazuje žádné výrazné chování, a proto bude z dalších experimentů vynechán.

Pro hlavní fázi experimentu byl tedy vypuštěn parametr *rozestupy* a některé parametry změněny.

- Data: **kurz Velké Británie**, **kurz Švýcarska**, **kurz Turecka**
- Velikost populace: **50**
- Počet generací: **50**



Obrázek 5.2: Výpis řešení v populaci od nejlepšího po nejhorší pro experiment s kurzy

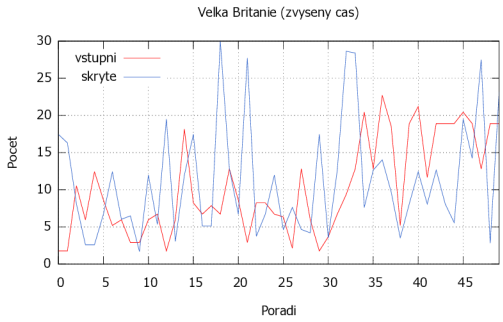
Dle výpisu výsledných populací (obrázek 5.2) lze celkem s jistotou soudit, že úspěch měly sítě s malým počtem vstupních neuronů (1–3). Počet neuronů ve skryté vrstvě už tak jednoznačný není, a zřejmě záleží na konkrétní podobě predikované řady. Dobrá hodnota pro obecnou řadu kurzu měny by se mohla pohybovat kolem 10–15 skrytých neuronů.

5.3 Vliv omezeného času na učení

Při tomto experimentu byl zvýšen čas učení na dvojnásobek, a otestováno na kurzu Velké Británie. Ostatní parametry zůstaly stejné, jako u předchozího experimentu.

- Data: **kurz Velké Británie**
- Maximální doba učení: **10**s

Na výsledném grafu populace (obrázek 5.3) není už jasně patrný úspěch řešení s malým počtem vstupních a velkým počtem skrytých neuronů, jako na předchozích. I když takový jedinec zaujal první místo v populaci. Nicméně trend zaujímání lepších pozic v populaci řešeními, se spíše menším počtem vstupních neuronů, je patrný stále. Proto v dalších experimentech bude uvažováno místo se 2 vstupními neurony se 3, a počet skrytých bude také upraven, a to na 10.



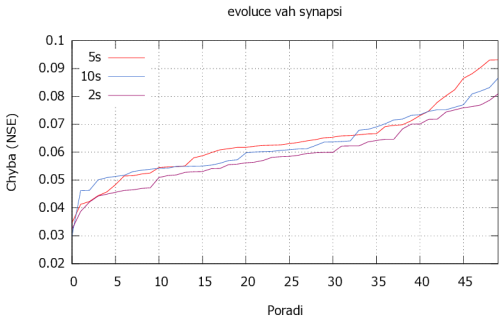
Obrázek 5.3: Výpis řešení v populaci od nejlepšího po nejhorší pro experiment s omezeným časem

5.4 Váhové koeficienty

V tomto experimentu byla použita velikost sítě navrhnutá v předchozích experimentech (3 vstupní, 10 skrytých, 1 výstupní neuron), u které byly vyvíjeny váhy jednotlivých synapsí, nastavených při inicializaci sítě před začátkem učení. Každé řešení v populaci bylo otestováno na všech třech kurzech tak, že u každého byla síť trénována. Souhrn změněných parametrů experimentu:

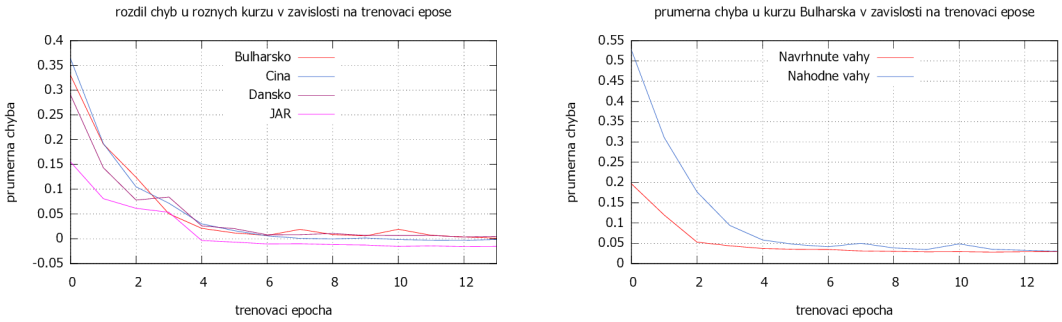
- Data: **kurz Velké Británie**, **kurz Švýcarska**, **kurz Turecka**
- Velikost populace: **50**
- Počet generací: **100**
- Fenotyp: pole hodnot **(-1.0; 1.0)**
- Mutace: **Dvoubodová inverzní mutace**
- Počáteční nastavení vah: **Uniformní rozložení**
- Struktura sítě: **3/10/1**
- Maximální počet trénovacích epoch: **5/10/20**

Po proběhnutí experimentu bylo nejlepšími jedinci v populaci dosáhnuto chyby až 0.03. Jak ukazuje graf 5.4, počet trénovacích epoch neměl podstatný efekt na výsledky. Pro dané kurzy by se mělo jednat o nejlepší možná počáteční nastavení koeficientů pro učení, a další snižování chyby už nelze docílit pomocí koeficientů.



Obrázek 5.4: Chyba jednotlivých řešení v populaci pro evoluci vah

Následující grafy (5.5) jsou výsledkem testování dosažených výsledků v tomto experimentu. Bylo provedeno padesát učení sítě s navrhnutými váhami a padesát s náhodnými koeficienty. Na levém grafu je tedy rozdíl průměrů těchto hodnot. (náhodná - navrhnutá), na pravém pak absolutní chyba pro jeden případ. Je jasně vidět, že navrhnutá síť dosahuje rychlejší konvergence k její minimální možné chybě (která je daná velikostí sítě). Sítě s náhodnými vahami dosahovaly konvergence okolo čtvrté epochy, s náhodnými později.



Obrázek 5.5: Otestování výkonu navrhnutých koeficientů na různých kurzech

Kapitola 6

Implementace

Pro experimenty byl použit jazyk **C++** v prostředí Microsoft Visual Studio 2010 Ultimate, dostupném zdarma pro studenty FIT ke studijním účelům v rámci *MSDN Academic Alliance* [7]. Hlavním důvodem pro tuto volbu byly již existující knihovny pro genetické algoritmy a neuronové sítě v tomto jazyce. GAlib [12] pro genetické algoritmy a Flood [6] pro neuronové sítě.

Na dokumentaci byl použit LaTeX (MiKTeX pro Windows s editorem TeXnicCenter) spolu s programem *gnuplot* pro tvorbu grafů a Inkscape pro tvorbu diagramů a obrázků. Také byla použita norma citací z [10].

6.1 Aplikace

Vzhledem k experimentální povaze práce nebylo třeba vytvářet složitý model aplikace, proto se experimenty spouští pomocí direktivy pro preprocesor, podle jejíž hodnoty se spustí patřičný experiment. Dodatečné parametry experimentu se určují přímo v kodu. Pro druhý experiment už je dostupná struktura s nastavením parametrů, protože bylo třeba je přenastavovat a měnit.

```
struct weightsExperimentParams {
  int input;
  int hidden;
  int output;
  Flood::Vector<double> data;
  Flood::Vector<double> data1;
  Flood::Vector<double> data2;
  int popsize;
  int ngen;
  float pmut;
  float pcross;
};
```

6.2 Genetické algoritmy

Pro genetické algoritmy z knihovny GAlib je typický tento přístup k řešení:

1. Vybrat a nastavit jak bude vypadat genom a zakódvat problém.
2. Specifikovat fitness funkci, která bude ze zadaného genomu vracet jeho ohodnocení.

3. Vytvořit objekt genetického algoritmu a nastavit parametry.

4. Spustit evoluci a poté případně uložit/vyhodnotit/vypsat výsledky.

```
GABin2DecPhenotype map;
GABin2DecGenome genome(map, SizeExperimentObjective);

GASimpleGA ga(genome);
ga.populationSize(100);
ga.nGenerations(100);
ga.pMutation(0.05);
ga.pCrossover(0.9);
ga.evolve();

float SizeExperimentObjective(GAGenome& g){
  return x;
}
```

Fitness funkce pro každý experiment je v základu podobná. Je v ní volána neuronová síť s parametry z genomu, která vrátí chybu. A čím menší chyba je vrácena neuronovou sítí, tím větší ohodnocení má daný jedinec. Následující příklad ilustruje funkci z prvního experimentu, kde je volána síť na základě dvou argumentů z genomu (definice genomu nad funkcí):

```
GABin2DecPhenotype map;
map.add(6, 1, 25);
map.add(7, 1, 30);

float SizeExperimentObjective(GAGenome& g){
  GABin2DecGenome & genome = (GABin2DecGenome &)g;
  double b = simple_neuron(int(genome.phenotype(0)),0,int(genome.phenotype(1)));
  float a = static_cast<float>(b);
  return 1/a;
}
```

Pro druhý experiment je použit jiný typ genů (`GARealGenome`), který obsahuje pole reálných čísel z intervalu $< -1, 1 >$. Toto pole je pak ve funkci rozděleno na dvě matice s váhami a dva vektory s prahy a posláno neuronové síti (zajišťuje funkce `getWeightsFromVector` ze souboru `neuron.cpp`).

```
GAAlleleSet<float> alleles(-1.0,1.0);
GARealGenome genome(X, alleles, WeightsExperimentObjective);
```

6.3 Neuronové sítě

Obecný postup jak v knihovně Flood vytvořit a natrénovat vícevrstvý perceptron je:

1. Vytvořit datový objekt a načíst do něj patřičná data (případně provést nad daty další úpravy).

2. Vytvořit objekt perceptronu s požadovaným počtem vstupů, výstupů a skrytých neuronů (počet vstupů musí souhlasit s počtem vstupních proměnných datového objektu).

3. Vytvořit cílové kritérium (čeho má síť dosáhnout).

4. Vytvořit trénovací metodu a případně specifikovat dodatečné parametry pro trénování.

5. Spustit trénink a poté dle potřeby vyhodnotit výsledky.

```
InputTargetDataSet itds(200, X, 1);
itds.set_data(data);

MultilayerPerceptron mlp(X, Y, Z);

NormalizedSquaredError nse(&mlp, &itds);

QuasiNewtonMethod qnm(&nse);
qnm.set_maximum_epochs_number(10);
qnm.train();
```

U všech experimentů je pro trénování sítě použito 75% dat a zbylých 25% je použit pro testování. Chyba, kterou funkce poté vrátí, je chyba na této testovací množině, na které nebyla síť trénována (je jasné, že pro data, na kterých byla trénována, bude dosahovat dobrých výsledků). Proto je potřeba vytvořit novou datovou množinu, pouze z původních testovacích dat (`itds1`), a na ní otestovat chybu (`calculate_objective()`).

```
itds.split_random_indices(0.75,0,0.25);
...
Matrix<double> test = itds.get_testing_data();
itds1.set_data(test);
NormalizedSquaredError nse1(&network, &itds1);
return nse1.calculate_objective();
```

6.4 Úprava dat

Datové struktury z knihovny *Flood* používají formát dat takový, kde pro vstup na neuronovou síť s N vstupními neurony je potřebný vektor jednotlivých vstupů o velikosti N, ovšem časové řady jsou uspořádány jako jednotlivé řádky buď v jednom sloupci, či ve dvou, pokud je přítomen údaj o čase. V práci byly použity řady s konstatními časovými rozestupy jeden pracovní den. Proto je vždy před načtením dat do datových struktur potřeba upravit formát. Dle zvoleného časového okna (počtu vstupů neuronové sítě) je vždy vytvořena matice, která má počet sloupců rovný $N + 1$, kde N je velikost časového okna a 1 je následující hodnota (cílová hodnota predikce). Dále je ještě použit parametr **rozestupy**, který určuje jaká je mezera mezi vstupy. Při hodnotě 0 je bráno N po sobě jdoucích vzorků v časovém okně, při hodnotě 1 jsou vzorky brány ob jeden. Toto zajišťuje funkce `getSpecificData` v souboru `neuron.cpp`.

t0	t1	t2	t4	t0	t2	t4	t5
t2	t3	t4	t5	t1	t3	t5	t6
t3	t4	t5	t6	t2	t4	t6	t7
t4	t5	t6	t7	t3	t5	t7	t8
t5	t6	t7	t8	t4	t6	t8	t9

Obrázek 6.1: Ukázka formátu vstupních dat pro $N = 3$ a rozestup= 0/1

V datovém objektu je navíc potřeba před použitím všechny sloupce přeškálovat a získat údaje (minimum, maximum,...), které vyžaduje neuronová síť. Tyto informace jsou pak také použity pro zpětný převod dat a výsledků z přeškálovaných na původní.

```
variables_statistics = itds.scale_variables_mean_standard_deviation();
mlp.set_variables_statistics(variables_statistics);
```

6.5 Modifikace knihovny Flood

Na několika místech souboru `MultilayerPerceptron.cpp` z knihovny Flood bylo třeba modifikovat zdrojový kód. Tyto změny se týkaly míst podobných, jak je ukázáno v příkladu dole. Zřejmě došlo vlivem refaktorování kódu automatickými nástroji k nezáměrnému překrytí proměnné i, která byla předávána jako parametr funkci, a zároveň jako proměnná cyklu `for`. Proto byla proměnná cyklu změněna na j. Flood sice obsahuje vlastní Unit testy, nicméně nastavení jejich parametrů tuto, zřejmě chybu, neodhalilo. V době odevzdání práce byly na oficiálním webu stále chybné zdrojové kódy.

```
int MultilayerPerceptron::get_hidden_layer_parameters_number(int i){
   int hidden_parameters_number = 0;
   for(int i = 0; i < hidden_layers_size[i]; i++) {
      hidden_parameters_number += hidden_layers[i][i].get_parameters_number();
   }
}
```

Kapitola 7

Závěr

Byla provedena studie neuronových sítí a evolučních algoritmů, na jejímž základě se realizovaly experimenty s predikcí časových řad, jako ukázka evolučního návrhu umělé neuronové sítě. Těmito experimenty byla navrhnuta síť, která je schopná učit se predikci obecných kurzů měn rychleji, než síť s náhodným nastavením vah. Jedná se ovšem o průměr, při jednotlivém pokusu není jisté, jaké váhy se vygenerují. Proto pro jednotlivý pokus je téměř vždy výhodné použít navrhnuté váhy, které se zdají být použitelné pro predikci kurzů obecně. Pokud by bylo použito více sítí, může se stát, že náhodné váhy budou vygenerovány vhodně pro daný případ. Zadání bylo splněno ve všech bodech, jak teoretická, tak praktická část.

K experimentům byl použit jazyk C++ s knihovnami `GALib` (evoluční algoritmy) a `Flood` (neuronové sítě) s pomocí `Microsoft Visual Studio 2010`. Analýza i zpracování dat a výsledků probíhala pomocí programů jako `GNUPLOT`, `PYTHON`. Pro sazbu dokumentace posloužil LaTeXa pro kreslení diagramů a obrázků `INKSCAPE`.

7.1 Možnosti dalšího rozvoje

Při praktickém použití by stálo za zvážení najít pro evoluční návrh více různých typů kurzů, nebo se snažit vybrat lepší, obecnější vzorky. Například na všechny dostupné kurzy použít metodu shlukování pomocí neuronové sítě a učení bez učitele (kapitola 2.5), a tím najít potřebné, typické vzorky.

Další zajímavé rozšíření by bylo spojit predikci kurzů s dolováním znalostí z expertních databází, či hledáním závislostí mezi kurzy jednotlivých států. V tu chvíli by se predikovalo nejen z minulých hodnot daného kurzu, ale z více kurzů. Případně ještě zahrnout různé burzovní ukazatele, indexy, či jiné hodnoty závislé na čase.

Literatura

[1] Bouška, J. *Neuronové sítě pro predikci časových řad.* Praha: České vysoké učení technické v Praze, 1995. Diplomová práce.

[2] Bryson, A. E. a Ho, Y.-C. *Applied optimal control: optimization, estimation, and control.* 1969.

[3] Fyfe, C. *Artificial Neural Networks.* 1996. Dostupné na: <http://www.scribd.com/doc/38060181/Artificial-Neural-Networks-Colin-Fyfe>.

[4] Kröse, B. a Smagt, P. van der. *An introduction to Neural Networks.* 1996.

[5] Kuba, M. *Neuronové sítě.* Brno: FI MU v Brně, 1995. Diplomová práce.

[6] Lopez, R. *Neural Networks for Variational Problems in Engineering.* Barcelona: Technical University of Catalonia, 2008. Disertační práce. Dostupné na: <http://www.cimne.com/flood/docs/PhDThesis.pdf>.

[7] Microsoft. *MSDN Academic Alliance.* 2011. Dostupné na: <http://msdn.microsoft.com/en-us/academic/default>.

[8] Plummer, E. A. *Time series forecasting with feed-forward neural networks: guidelines and limitations.* Laramie, Wyoming: The University of Wyoming, 2000. Diplomová práce. Dostupné na: <http://www.karlbranting.net/papers/plummer/Paper_7_12_00.htm>.

[9] Pospíchal, P. *Akcelerace genetického algoritmu s využitím GPU.* Brno: FIT VUT v Brně, 2009. Diplomová práce. Dostupné na: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=6540>.

[10] Pyšný, R. *BIBTEX STYL PRO ČSN ISO 690 A ČSN ISO 690-2.* Brno: FIT VUT v Brně, 2009. Bakalářská práce. Dostupné na: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=7848>.

[11] Schwarz, J. a Sekanina, L. *Aplikované evoluční algoritmy, Studijní opora předmětu EVO.* 2006. FIT VUT v Brně.

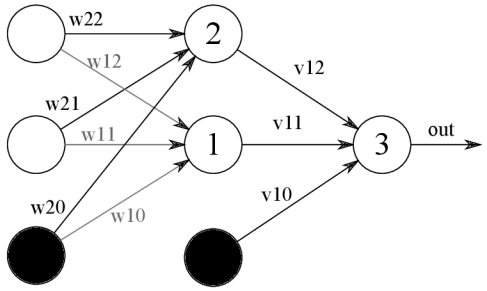
[12] Wall, M. a MIT. *Stránky knihovny GAlib.* 1999. Dostupné na: <http://lancet.mit.edu/ga/>.

Seznam příloh

Příloha A

Příklad XOR

Příklad řešení problému XOR dopřednou vícevrstvou sítí s učícím algoritmem back-propagation [3, str. 39–40]. Topologie sítě je znázorněna na obrázku A.1. Černě vybarvené neurony jsou neurony simulující práh jednotlivých neuronů, a tudíž je jejich hodnota (in) vždy 1.



Obrázek A.1: Topologie sítě pro příklad

Inicializace

- Nastav váhy (w,v) na náhodná malá čísla.
- Nastav míru učení η například na 0.001.
- Vyber aktivační funkci, například $tanh()$.

Vybrání vzoru

V této fázi je vybrán vzor na učení sítě, pro XOR jsou zde 4 možné vzory:

$$T : (0,0) \longrightarrow 0$$
$$T : (0,1) \longrightarrow 1$$
$$T : (1,0) \longrightarrow 1$$
$$T : (1,1) \longrightarrow 0$$

Propagace

V této fázi je aktivace neuronů propagována dopředu mezi vrstvami. Pro každou vrstvu jsou napřed sečteny vstupy a poté provedena transformace aktivační funkcí.

$$\begin{aligned} in_1 &= w_{10} + w_{11}x_1 + w_{12}x_2 \\ in_2 &= w_{20} + w_{21}x_1 + w_{22}x_2 \\ o_1 &= tanh(in_1) \\ o_2 &= tanh(in_2) \\ in_3 &= v_{10} + v_{11}o_1 + v_{12}o_2 \\ o_3 &= tanh(in_3) \end{aligned}$$

Určení chyby

Nyní se spočítají chyby jednotlivých neuronů, kde t je hodnota, které by se o_3 mělo rovnat dle vzoru:

$$\begin{aligned} \delta_3 &= (t - o_3)f'(o_3) = (t - o_3)(1 - o_3^2) \\ \delta_1 &= \delta_3 v_{11} f'(o_1) = \delta_3 v_{11}(1 - o_1^2) \\ \delta_2 &= \delta_3 v_{12} f'(o_2) = \delta_3 v_{12}(1 - o_2^2) \end{aligned}$$

Změna vah

Všechny váhy v síti se nakonec změní dle těhto vzorečků. Jinými slovy, upraví se o trochu tak, aby lépe odpovídala vzoru, který je v tuto chvíli učen. Jak moc velký skok se provede záleží na hodnotě η.

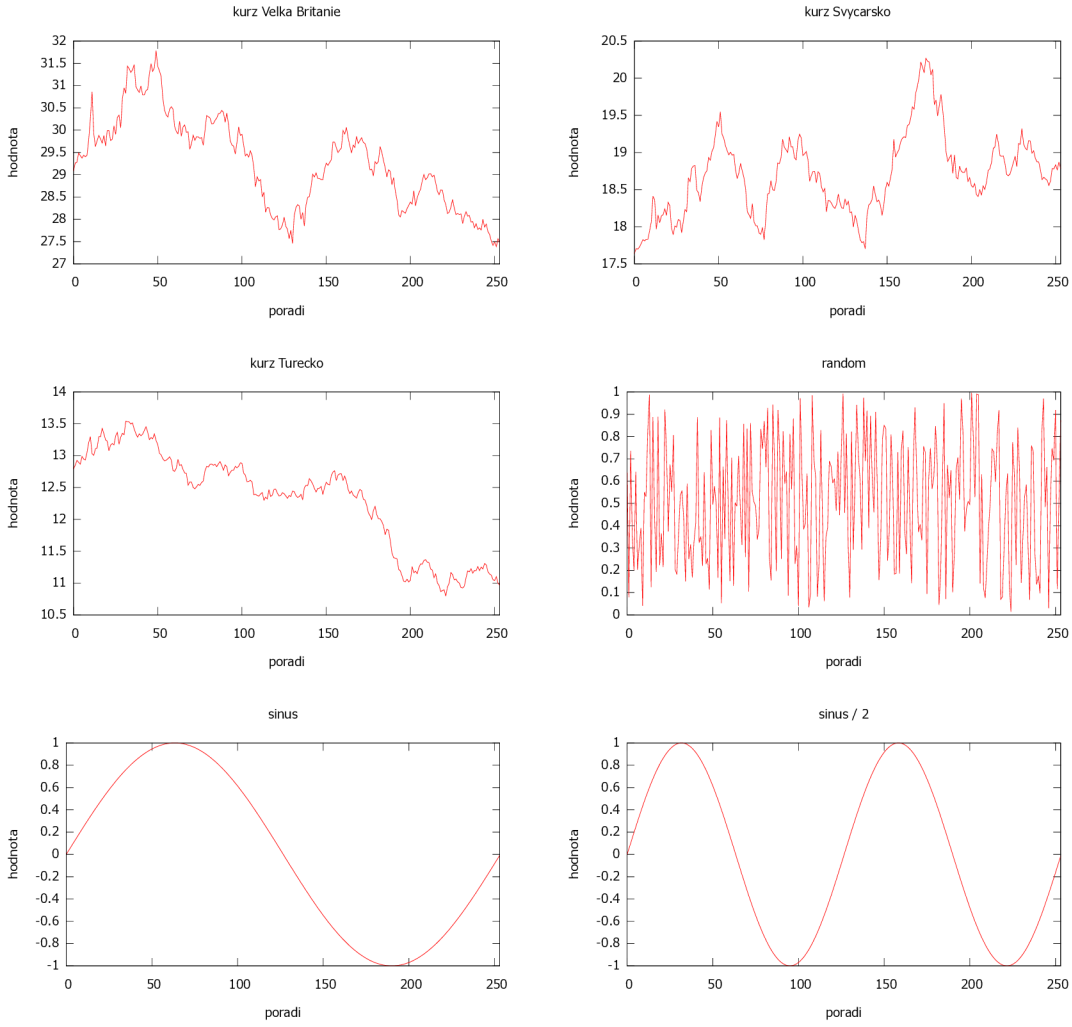
$$\begin{aligned} \Delta v_{10} &= \eta * \delta_3 * 1 \\ \Delta v_{11} &= \eta * \delta_3 * o_1 \\ \Delta v_{12} &= \eta * \delta_3 * o_2 \\ \Delta w_{10} &= \eta * \delta_1 * 1 \\ \Delta w_{11} &= \eta * \delta_1 * x_1 \\ \Delta w_{12} &= \eta * \delta_1 * x_2 \\ \Delta w_{20} &= \eta * \delta_2 * 1 \\ \Delta w_{21} &= \eta * \delta_2 * x_1 \\ \Delta w_{22} &= \eta * \delta_2 * x_2 \end{aligned}$$

Opakování

Toto se opakuje vícekrát (dle potřeby) pro každý vzor v trénovací množině, dokud síť nedává uspokojivé výsledky. [3] [4]

Příloha B

Grafy použitých řad



Obrázek B.1: Grafy řad