PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

DIPLOMOVÁ PRÁCE

Metody faktorizace binárních dat:

experimenty a porovnání

2013 Petr Budílek

Anotace

Faktorizace binárních dat slouží k objevení skrytých vzorů v datech, tzv. faktorů. Tyto faktory pak mohou sloužit ke snížení dimensionality dat. Cílem této práce je popsat a experimentálně ověřit metodu faktorizace binárních dat pomocí formálních konceptů a metodu faktorizace binárních dat pomocí atraktorové neuronové sítě.

Děkuji Prof. RNDr. Radimu Bělohlávkovi Ph.D., DSc. za vedení této diplomové práce.

Obsah

Seznam obrázků

1. Úvod

Faktorizace binárních dat slouží k objevení skrytých vzorů v datech, tzv. faktorů. Tyto faktory mohou mimo jiné sloužit ke snížení dimensionality dat. Takovou analýzu lze provádět mnoha způsoby. V této práci se zaměřím na dva z nich. Konkrétně na faktorizaci binárních dat pomocí formální konceptů od autorů Radima Bělohlávka a Viléma Vychodila, představenou v článku [2] a dále pak na faktorizaci binárních dat pomocí atraktorové neuronové sítě od autora Alexandera A. Frolova a kolektivu, představenou v článku [6]. Metody popíši a jejich funkci experimentálně ověřím na reálných datech.

V kapitole 2. formálně popíši problém faktorizace binárních dat a uvedu její možné využití. V kapitole 3. podám základy formální konceptuální analýzy nezbytné pro pochopení metody rozkladu pomocí formálních rozkladů. V kapitole 4. popíši metodu samotnou a uvedu výsledky experimentu na reálných datech. V kapitole 5. seznámím čtenáře se základy neuronových sítí nezbytných pro pochopení metody rozkladu pomocí atraktorové neuronové sítě. V kapitole 6. popíši metodu samotnou spolu s experimenty testujícími různé aspekty této metody a experimentem provedených nad reálnými daty.

2. Faktorizace binárních dat

Při faktorizaci binárních dat (též binární faktorová analýza - BFA) nám jde v podstatě o to, reprezentovat (rozložit) nějakou binární matici symbolizující data charakteru objekt-proměnná pomocí dvou různých matic. Jedna z těchto matic popisuje vztah mezi objekty a skrytými proměnnými - faktory a druhá matice popisuje vztah mezi faktory a originálními proměnnými. Problém můžeme formálně popsat následovně:

Problém 2..1. *Mějme matici I o rozměrech $n \times m$ takovou, že $I_{ij} \in \{0,1\}$. Cílem je získat booleovský maticový produkt $A \circ B$ matice A o rozměrech $n \times k$, kde $A_{il} \in \{0,1\}$ a matice B o rozměrech $k \times m$, kde $B_{lj} \in \{0,1\}$ takový, aby k bylo co nejmenší.*

V booleovské teorii matic bychom uvedený produkt mohli vyjádřit jako

$$A \circ B = \bigvee_{l=1}^{k} A_{il} \cdot B_{lj}$$

kde $\bigvee$ označuje maximum (pravdivostní funkce logické disjunkce) a $\cdot$ označuje obyčejný součin (pravdivostní funkce logické konjunkce). Booleovský maticový produkt potom reprezentuje nelineární vztah mezi objekty, faktory a originálními atributy. Jako příklad si uveďme:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \circ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}.$$

Takový rozklad matice I do produktu $A \circ B$ potom koresponduje s objevením k faktorů, které vysvětlují data reprezentovaná maticí I. Objevení takových faktorů (provedení rozkladu) může poskytnout uživateli nové informace o vztazích mezi originálními proměnnými, redukovat dimenzi vstupních dat, nebo pomocí jen části faktorů potřebných k rozkladu aproximovat původní matici do jistého předem zadaného stupně. Tuto aproximaci lze pak použít například v oblasti ztrátové komprese dat. Informace k této kapitole jsem čerpal z [2].

3. Formální konceptuální analýza

Formální konceptuální analýza (FKA) je metoda analýzy dat, kterou poprvé popsal Bernard Will v roce 1984. FKA operuje nad daty, která popisují vztah mezi určitou množinou objektů a určitou množinou atributů. Výstupem analýzy je hierarchie tzv. formálních konceptů. Formální koncept je určitý shluk, který reprezentuje přirozené lidské koncepty jako např. „listnatý strom“, „nákladní automobil“, „prvočíslo“, atd. Použití FKA je tedy velice široké, stejně tak jako FKA sama. Pro její použití v binární faktorové analýze budeme potřebovat následující matematický aparát. Rád bych poznamenal, že všechny definice, věty a příklady použité v této kapitole, jsou z [1].

3.1. Matematické základy

Začněme definicí tzv. formálního kontextu. Formální kontext v podstatě představuje data, která pak budeme analyzovat.

Definice 3..1 (Formální kontext)**.** Formální kontext *je trojice* $\langle X, Y, I \rangle$*, kde* X *a* Y *jsou neprázdné množiny a* I *je binární relace mezi* X *a* Y*, tj.* $I \subseteq X \times Y$*.*

Prvky x z X nazveme objekty a prvky y z Y nazveme atributy. Dvojice $\langle x, y \rangle \in I$ značí, že objekt x obsahuje atribut y.

Obrázek 1. Tabulka s křížky

I	y_1	y_2	y_3	y_4
x_1	×	×	×	×
x_2	×		×	×
x_3		×	×	×
x_4		×	×	×
x_5	×			

Formální kontext si lze představit jako tabulku s n řádky a m sloupci, kde $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_m\}$ a kde relace I je definována následovně: $\langle X_i, Y_j \rangle \in I$ právě když se na pozici i-tého řádku a j-tého sloupce vyskytuje křížek.

Příklad takové tabulky je možno vidět na obr. 3.1..

Definice 3..2 (Koncept-generující operátory)**.** Formální kontext *Pro formální kontext* $\langle X, Y, I \rangle$*, operátory* $^{\uparrow} : 2^X \to 2^Y$ *a* $^{\downarrow} : 2^Y \to 2^X$ *jsou definovány tak, že pro každé* $A \subseteq X$ *a* $B \subseteq Y$*:*

$$A^{\uparrow} = \{y \in Y \mid \textit{pro každé } x \in A : \langle x, y \rangle \in I\},$$

$$B^{\downarrow} = \{x \in X \mid \textit{pro každé } y \in B : \langle x, y\rangle \in I\}.$$

Operátor $^{\uparrow}$ tedy příslušné množině objektů přiřadí množinu atributů, které jsou společné pro všechny objekty. Naopak operátor $^{\downarrow}$ přiřadí příslušné množině atributů množinu objektů, jež všechny tyto atributy sdílí.

Příklad 3..1 (Koncept-generující operátory)**.** *Pro tabulku 3.1. máme:*

- $\{x_2\}^{\uparrow} = \{y_1, y_3, y_4\}, \{x_2, x_3\}^{\uparrow} = \{y_3, y_4\}$
- $\{x_4, x_5\}^{\uparrow} = \emptyset$
- $\{y_1\}^{\downarrow} = \{x_1, x_2, x_5\}, \{y_1, y_3\}^{\downarrow} = \{x_1, x_2\}$
- $Y^{\downarrow} = \{x_1\}$

Formální koncept je základním pojmem FKA. Jde tedy o určitý shluk v tabulce křížků, definovaný následovně:

Definice 3..3 (Formální koncept)**.** Formální koncept *v* $\langle X, Y, I\rangle$ *je dvojice* $\langle A, B\rangle$*, kde* $A \subseteq X$ *a* $B \subseteq Y$*, taková že* $A^{\uparrow} = B$ *a* $B^{\downarrow} = A$*.*

Jinými slovy, dvojice $\langle A, B\rangle$ je formální koncept, právě když A obsahuje pouze objekty, jež sdílí všechny atributy z B. Pro formální koncept $\langle A, B\rangle$ v $\langle X, Y, I\rangle$ se A nazývá extent a B se nazývá intent.

I	y_1	y_2	y_3	y_4
x_1	×	×	×	×
x_2	×		×	×
x_3		×	×	×
x_4		×	×	×
x_5	×			

Příklad 3..2 (Formální koncepty)**.**

Obrázek 2. Formální koncept

Pro tabulku na obr. 2. pak zvýrazněný obdélník představuje formální koncept:

$$\langle A_1, B_1\rangle = \langle\{x_1, x_2, x_3, x_4\}, \{y_3, y_4\}\rangle$$

protože

$$\{x_1, x_2, x_3, x_4\}^{\uparrow} = \{y_3, y_4\}$$

a

$$\{y_3, y_4\}^{\downarrow} = \{x_1, x_2, x_3, x_4\}.$$

Dalšími koncepty v uvedené tabulce jsou například:

$$\langle A_2, B_2\rangle = \langle\{x_1, x_3, x_4\}, \{y_2, y_3, y_4\}\rangle,$$
$$\langle A_3, B_3\rangle = \langle\{x_1, x_2\}, \{y_1, y_3, y_4\}\rangle.$$

Koncepty lze přirozeně uspořádat pomocí subkoncept-superkoncept relace. Relace subkoncept-superkoncept funguje na pricipu množinové inkluze následovně:

Definice 3..4 (Uspořádání konceptů)**.** *Pro formální koncept* $\langle A_1, B_1\rangle$ *a* $\langle A_2, B_2\rangle$ *z* $\langle X, Y, I\rangle$*, polož* $\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle$ *právě když* $A_1 \subseteq A_2$ *(právě když* $B_2 \subseteq B_1$*).*

Relace subkoncept-superkoncept je zde značena pomocí $\leq$. V případě, že $\langle A_1, B_1\rangle \leq \langle A_2, B_2\rangle$, říkáme, že koncept $\langle A_1, B_1\rangle$ je specifičtější než koncept $\langle A_2, B_2\rangle$. O konceptu $\langle A_2, B_2\rangle$ pak říkáme, že je obecnější než koncept $\langle A_1, B_1\rangle$. V případě `člověk` $\leq$ `organismus`, pak relace $\leq$ dobře podchycuje intuici, že koncept „člověka“ je specifičtější než koncept „organismu“.

Příklad 3..3 (Subkoncept-superkoncept uspořádání)**.** *Nechť*

$$\langle A_1, B_1\rangle = \langle\{x_1, x_2, x_3, x_4\}, \{y_3, y_4\}\rangle,$$

$$\langle A_2, B_2\rangle = \langle\{x_1, x_3, x_4\}, \{y_2, y_3, y_4\}\rangle,$$

$$\langle A_3, B_3\rangle = \langle\{x_1, x_2\}, \{y_1, y_3, y_4\}\rangle$$

jsou formální koncepty z příkladu 3..2. Potom platí: $\langle A_3, B_3\rangle \leq \langle A_1, B_1\rangle, \langle A_2, B_2\rangle \leq \langle A_1, B_1\rangle$.

Množinu konceptů uspořádanou hierarchicky podle relace subkoncept-superkoncept nazveme konceptuálním svazem.

Definice 3..5 (Konceptuální svaz)**.** *Nechť* $\beta(X, Y, I)$ *označuje kolekci všech formálních konceptů v* $\langle X, Y, I\rangle$*, tedy*

$$\mathcal{B}(X, Y, I) = \{\langle A, B\rangle \in 2^X \times 2^Y \mid A^{\uparrow} = B, B^{\downarrow} = A\}$$

$\mathcal{B}(X, Y, I)$ *vybavený subkoncept-superkoncept uspořádáním* $\leq$ *se nazývá* Konceptuální svaz *v* $\langle X, Y, I\rangle$.

Dále pak

$$Ext(X, Y, I) = \{A \in 2^X \mid \langle A, B\rangle \in \beta(X, Y, I) \text{ pro nějaké } B\}$$

a

$$Int(X, Y, I) = \{B \in 2^Y \mid \langle A, B\rangle \in \beta(X, Y, I) \text{ pro nějaké } A\}.$$

Pomocí $Ext(X, Y, I)$ tedy označujeme množinu extentů všech konceptů v konceptuálním svazu, pomocí $Int(X, Y, I)$ pak množinu intentů všech konceptů. Důkaz, že $\langle\mathcal{B}(X, Y, I), \leq\rangle$ je opravdu svaz lze potom nalézt v [1]. Příklad konceptuálního svazu lze též nalézt v [1]. Jak chápeme pojem obdélníku ve formálním kontextu říká následující definice.

Definice 3..6 (Obdélníky v $\langle X, Y, I \rangle$)**.** Obdélník *v* $\langle X, Y, I \rangle$ *je dvojice* $\langle A, B \rangle$ *taková, že* $A \times B \subseteq I$*, to jest: pro každé* $x \in A$ *a* $y \in B$ *máme* $\langle x, y \rangle \in I$*. Pro obdélníky* $\langle A_1, B_1 \rangle$ *a* $\langle A_2, B_2 \rangle$*, nechť* $\langle A_1, B_1 \rangle \sqsubseteq \langle A_2, B_2 \rangle$ *právě když* $A_1 \subseteq A_2$ *a* $B_1 \subseteq B_2$

Díky zavedení uspořádání $\sqsubseteq$ můžeme o některých obdélnících hovořit jako o maximálních. Jinak řečeno maximální obdélník ve formálním kontextu je takový obdélník, který již nelze přidáním dalšího atributu či objektu rozšířit.

Příklad 3..4 (Obdélníky)**.** *V tabulce 3.1.* $\langle \{x_1, x_2\}, \{y_3, y_4\} \rangle$ *je obdélníkem, který není maximální vzhledem k* $\sqsubseteq$*. Naproti tomu obdélník* $\langle \{x_1, x_2, x_3\}, \{y_3, y_4\} \rangle$ *maximální vzhledem k* $\sqsubseteq$ *je.*

Následující věta dává do souvislosti pojem formální koncept a pojem maximální obdélník.

Věta 3..1 (Formální koncepty jako maximální obdélníky)**.** $\langle A, B \rangle$ *je formálním konceptem v* $\langle X, Y, I \rangle$ *právě když* $\langle A, B \rangle$ *je maximálním obdélníkem v* $\langle X, Y, I \rangle$*.*

4. Binární faktorová analýza pomocí FKA

V této části popíši metodu binární faktorové analýzy vytvořenou Radimem Bělohlávkem a Vilémem Vychodilem, a uvedu také několik důležitých poznatků ohledně spojení mezi BFA a FKA. Všechny informace v této části budu čerpat z článku [2]. Toto tedy platí mimo jiné o všech definicích, větách, příkladech a algoritmech zde uvedených.

Tuto část začnu popisem, jak využít FKA k hledání faktorů. Konkrétně uvedu vztah mezi formálními koncepty a faktory. Následně popíši úvahu, která vedla ke konstrukci algoritmů dále popsaných. Na závěr pak podám informace o provedeném experimentu, který zahrnuje test zmíněných algoritmů na reálných datech.

4.1. Formální koncepty jako optimální faktory

Mějme $n \times m$ binární matici I. Nejprve si ukážeme, že matici I můžeme brát jako formální kontext: Objekty jsou reprezentovány řádky matice $X = \{1, \ldots, n\}$, atributy pak jejími sloupci $Y = \{1, \ldots, m\}$ a binární relace I je pak dána maticí I, kde $\langle i, j \rangle \in I$ právě když $I_{ij} = 1$.

Budeme uvažovat rozklad I na produkt $A_{mcF} \circ B_{mcF}$ binárních matic A_{mcF} a B_{mcF} konstruovaných z množiny formálních konceptů $\mathcal{F}$ asociovanou s I. Konkrétně mějme konceptuální svaz $\mathcal{B}(X, Y, I)$. Nechť

$$\mathcal{F} = \{\langle A_1, B_1 \rangle, \ldots, \langle A_k, B_k \rangle\} \subseteq \mathcal{B}(X, Y, I),$$

$\mathcal{F}$ je tedy množina formálních konceptů z $\mathcal{B}(X, Y, I)$. O $\mathcal{F}$ můžeme též mluvit jako o množině faktorových konceptů. Dále označme matice o rozměrech $n \times k$ a $k \times m$ jako $A_\mathcal{F}$ a $B_\mathcal{F}$. Tyto matice definujeme následovně:

$$(A_\mathcal{F})_{il} = \begin{cases} 1 & \text{pokud } i \in A_l, \\ 0 & \text{pokud } i \notin A_l, \end{cases}$$

$$(B_\mathcal{F})_{lj} = \begin{cases} 1 & \text{pokud } j \in B_l, \\ 0 & \text{pokud } j \notin B_l, \end{cases}$$

pro $l = 1, \ldots, k$. Tedy, l-tý sloupec $(A_\mathcal{F})_{\_l}$ matice $A_\mathcal{F}$ je charakteristickým vektorem množiny A_l a l-tý řádek $(B_\mathcal{F})_{l\_}$ matice $B_\mathcal{F}$ se sestává z charakteristického vektoru množiny B_l. Situaci nám pomůže osvětlit následující příklad.

Příklad 4..1 (Dekompozice matic)**.** *Uvažujme matici I:*

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix},$$

v přidruženém konceptuálním svazu nalezneme mimo jiné formální koncepty $\langle A_1, B_1\rangle = \langle\{1,2,3\},\{1,2\}\rangle$ *a* $\langle A_2, B_2\rangle = \langle\{1,2,3,4\},\{1\}\rangle$. *Dále nechť*

$$\mathcal{F} = \langle A_1, B_1\rangle, \langle A_2, B_2\rangle,$$

potom

$$(A_{\mathcal{F}}) = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} a\, (B_{\mathcal{F}}) = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

I když v tomto příkladu se $I \neq A_{\mathcal{F}} \circ B_{\mathcal{F}}$, v obecném případě však pro každou matici I existuje množina formálních konceptů asociovaných s I, pomocí kterých lze I rozložit, viz následující věta:

Věta 4..1 (Univerzalita formálních konceptů jako faktorů)**.** *Pro každou matici* I *existuje* $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$ *takové, že* $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$.

Tvrzení platí díky tomu, že pro každý prvek $I_{ij} = 1$ můžeme nalézt formální koncept $\langle C, D\rangle$, kde $i \in C$ a $j \in D$. Celý důkaz je možno nalézt v [2].

Rozklad pomocí formálních konceptů bude též optimální - počet faktorů bude nejmenší možný:

Věta 4..2 (Optimalita formálních konceptů jako faktorů)**.** *Nechť* $I = A \circ B$ *pro* $n \times k$ *a* $k \times m$ *binárních matic* A *a* B. *Pak existuje množina* $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$ *formálních konceptů v* I *s*

$$|\mathcal{F}| \leq k$$

taková, že pro $n \times |\mathcal{F}|$ *a* $|\mathcal{F}| \times m$ *binární matice* $A_{\mathcal{F}}$ *a* $B_{\mathcal{F}}$ *máme* $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$.

Tvrzení platí díky faktu, že každý rozklad lze zapsat jako $\bigvee$-superpozici, t.j. sjednocení k obdélníků jedniček v původní matici, dále z faktu, že každý takový obdélník je obsažen v nějakém maximálním obdélníku a konečně z faktu, že formální koncepty v I jsou právě maximálními obdélníky. Následující příklad nám pomůže k intuitivnímu pochopení, proč tvrzení platí a také jak tohoto tvrzení využít. Kompletní formální důkaz lze opět nalézt v [2].

Příklad 4..2. *Uvažujme následující rozklad*

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \circ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

a odpovídající obdélníky $J_1, \ldots, J_4$:

$$\begin{pmatrix}1&1&0&0&0\\1&1&0&0&0\\1&1&0&0&0\\0&0&0&0&0\end{pmatrix},\begin{pmatrix}0&0&0&0&0\\0&0&0&0&0\\0&0&1&1&0\\0&0&0&0&0\end{pmatrix},\begin{pmatrix}0&0&0&0&0\\1&0&0&0&1\\0&0&0&0&0\\1&0&0&0&1\end{pmatrix},\begin{pmatrix}0&1&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{pmatrix}.$$

Dále uvažujme formální koncepty

$$\begin{aligned}\langle A_1, B_1\rangle &= \langle\{1,2,3\},\{1,2\}\rangle,\\ \langle A_2, B_2\rangle &= \langle\{3\},\{1,2,3,4\}\rangle,\\ \langle A_3, B_3\rangle &= \langle\{2,4\},\{1,5\}\rangle\end{aligned}$$

v I. Každý z obdélníků $J_1,\ldots,J_4$ je pak obsažen v nějakém maximálním obdélníku, který odpovídá $\langle A_1,B_1\rangle,\langle A_2,B_2\rangle$ nebo $\langle A_3,B_3\rangle$. Položením $\mathcal{F}=\{\langle A_1,B_1\rangle,\langle A_2,B_2\rangle,\langle A_3,B_3\rangle\}$ pak dostaneme $I=A_\mathcal{F}\circ B_\mathcal{F}$, t.j.

$$\begin{pmatrix}1&1&0&0&0\\1&1&0&0&1\\1&1&1&1&0\\1&0&0&0&1\end{pmatrix}=\begin{pmatrix}1&0&0\\1&0&1\\1&1&0\\0&0&1\end{pmatrix}\circ\begin{pmatrix}1&1&0&0&0\\1&1&1&1&0\\1&0&0&0&1\end{pmatrix}$$

Rozklad $I=A_\mathcal{F}\circ B_\mathcal{F}$ se tedy podařilo realizovat pomocí stejného nebo menšího počtu faktorů než v původním rozkladu.

Rozklad $I=A_\mathcal{F}\circ B_\mathcal{F}$ můžeme dále rozepsat jako $I=(A_\mathcal{F})_{\_1}\circ(B_\mathcal{F})_{1\_}\vee(A_\mathcal{F})_{\_2}\circ(B_\mathcal{F})_{2\_}\vee(A_\mathcal{F})_{\_3}\circ(B_\mathcal{F})_{3\_}$, což tvoří $\bigvee$-rozklad I do maximálních obdélníků. V našem případě pak

$$\begin{pmatrix}1&1&0&0&0\\1&1&0&0&1\\1&1&1&1&0\\1&0&0&0&1\end{pmatrix}=\begin{pmatrix}1&1&0&0&0\\1&1&0&0&0\\1&1&0&0&0\\0&0&0&0&0\end{pmatrix}\vee\begin{pmatrix}0&0&0&0&0\\0&0&0&0&0\\1&1&1&1&0\\0&0&0&0&0\end{pmatrix}\vee\begin{pmatrix}0&0&0&0&0\\1&0&0&0&1\\0&0&0&0&0\\1&0&0&0&1\end{pmatrix}.$$

4.2. Povinné faktory

Ukázalo se, že některé formální koncepty jsou *povinné*. Povinné v tom smyslu, že musí být přítomny v každém $\mathcal{F}$ pro které $I=A_\mathcal{F}\circ B_\mathcal{F}$. Označme nyní $\mathcal{O}(X,Y,I)$ jako množinu všech objektových konceptů a $\mathcal{A}(X,Y,I)$ jako množinu všech atributových konceptů. Množiny definujeme následovně:

$$\begin{aligned}\mathcal{O}(X,Y,I)&=\{\langle\{x\}^{\uparrow\downarrow},\{x\}^{\uparrow}\rangle\mid x\in X\},\\ \mathcal{A}(X,Y,I)&=\{\langle\{y\}^{\downarrow},\{y\}^{\downarrow\uparrow}\rangle\mid y\in Y\}.\end{aligned}$$

Formální koncepty, které jsou přítomny jak v $\mathcal{O}(X,Y,I)$, tak v $\mathcal{A}(X,Y,I)$ jsou povinné, viz následující tvrzení:

Věta 4..3 (Povinné faktory)**.** *Pokud* $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$ *pro nějaké* $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$ *pak* $\mathcal{O}(X, Y, I) \cap \mathcal{A}(X, Y, I) \subseteq \mathcal{F}$*.*

Pokud bychom jako $\langle C, D\rangle$ označili koncept takový, že $\langle C, D\rangle \in \mathcal{O}(X, Y, I) \cap \mathcal{A}(X, Y, I)$, pak maximální obdélník, který odpovídá $\langle C, D\rangle$, je jediným maximálním obdélníkem, který pokrývá jedničky na průsečíku řádku x a sloupce y, kde $x \in C$ a $y \in D$.

4.3. Redukce na optimalizační problém množinového pokrytí

Přejděme nyní k poznatkům o složitosti nalezení optimálního binárního rozkladu. Nejprve však uveďme problém množinového pokrytí.

Definice 4..1 (Optimalizační problém množinového pokrytí)**.** *Mějme množinu* $\mathcal{U}$ *(univerzum) a systém množin* $\mathcal{S} \subseteq 2^{\mathcal{U}}$*, kde* $\bigcup \mathcal{S} = \mathcal{U}$*. Úkolem je najít systém množin* $\mathcal{C} \subseteq \mathcal{S}$ *takový, že* $\mathcal{U} = \bigcup \mathcal{C}$ *a zároveň je* $|C|$ *co nejmenší.*

Jinými slovy máme za úkol najít systém množin $\mathcal{C} \subseteq \mathcal{S}$, kde je počet jeho množin co nejmenší a zároveň jejich sjednocení je rovno $\mathcal{U}$. Problém, tak jak je výše definovaný, je NP-těžký a odpovídající rozhodovací problém je NP-úplný. Naštěstí však existuje efektivní aproximační žravý algoritmus s aproximačním poměrem $\leq ln(|\mathcal{U}|)+1$. To znamená, že nalezené řešení bude v nejhorším případě $(ln(|\mathcal{U}|) + 1)\times$ horší než řešení optimální.[2]

Problém nalezení rozkladu $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$ je potom redukovatelný na optimalizační problém množinového pokrytí. [2] To znamená, že pokud nalezneme řešení optimalizačního problému množinového pokrytí, pak toto řešení můžeme v polynomickém čase převést na řešení problému nalezení rozkladu $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$.

Redukce pak v tomto případě vypadá následovně: $\mathcal{U} = \{\langle i, j\rangle \mid I_{ij} = 1\}$ a $\mathcal{S} = \{C \times D \mid \langle C, D\rangle \in \mathcal{B}(X, Y, I)\}$. Snažíme se zde tedy pokrýt univerzum $\mathcal{U}$, které se skládá z dvojic, reprezentujících pozice jedniček v I, „obdélníkovými množinami“(ze systému $\mathcal{S}$) odpovídajících formálních konceptů.

Z výše uvedeného však nevyplývá do jaké třídy tedy vlastně problém nalezení rozkladu $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$ patří. Na to nám odpoví následující věta:

Věta 4..4. *Problém nalezení rozkladu* $I = A \circ B$ *binární* $n \times m$ *matice* I *do binární* $n \times k$ *matice* A *a binární* $k \times m$ *matice* B *takového, že* k *je co nejmenší, je NP-těžký a přidružený rozhodovací problém je NP-úplný.*

Takže pokud $P \neq NP$, pak neexistuje žádný algoritmus, který by v polynomickém čase vyřešil faktorizační problém.

4.4. Algoritmy

V této části si předvedeme dva aproximační algoritmy pro nalezení řešení problému rozkladu binární matice I. Díky větě 4..2 hledáme nejmenší množinu $\mathcal{F}$ faktorových konceptů v I. Každý z algoritmů ji hledá trochu jiným způsobem.

4.4.1. Algoritmus první

První z algoritmů těží z již zmíněné redukce na problém nalezení nejmenší množiny $\mathcal{F}$ faktorových konceptů na optimalizační problém množinového pokrytí. Můžeme tedy použít žravý aproximační algoritmus pro nalezení již výše zmíněného pokrytí. Žravý v tom smyslu, že množinu pro pokrytí vybíráme takovou, která v daném kroku pokryje z univerza nejvíce. Algoritmus můžeme dále urychlit přidáním povinných faktorových konceptů hned na začátku.

input : I
output: $\mathcal{F}$

1 **set** $\mathcal{S}$ **to** $\mathcal{B}(X, Y, I)$
2 **set** $\mathcal{U}$ **to** $\{\langle i, j\rangle \mid I_{ij} = 1\}$
3 **set** $\mathcal{F}$ **to** $\emptyset$
4 **foreach** $\langle i, j\rangle \in \mathcal{S}$ **do**
5 | **if** $\langle C, D\rangle \in \mathcal{O}(X, Y, I) \cap \mathcal{A}(X, Y, I) \subseteq \mathcal{F}$ **then**
6 | | **add** $\langle C, D\rangle$ **to** $\mathcal{F}$
7 | | **remove** $\langle C, D\rangle$ **from** $\mathcal{F}$
8 | | **foreach** $\langle i, j\rangle \in C \times D$ **do**
9 | | | **remove** $\langle i, j\rangle$ **from** $\mathcal{U}$
10 **while** $\mathcal{U} \neq \emptyset$ **do**
11 | **select** $\langle C, D\rangle \in \mathcal{S}$ **that maximizes** $(C \times D) \cap \mathcal{U}$:
12 | | **add** $\langle C, D\rangle$ **to** $\mathcal{F}$
13 | | **remove** $\langle C, D\rangle$ **from** $\mathcal{S}$
14 | | **foreach** $\langle i, j\rangle \in C \times D$ **do**
15 | | | **remove** $\langle i, j\rangle$ **from** $\mathcal{U}$
16 **return** $\mathcal{F}$

Algorithm 1: Algoritmus první

Řádek 1. - 3. definuje již zmíněnou redukci. Množina $\mathcal{B}(X, Y, I)$ je spočítána pomocí algoritmu NextClosure, který může čtenář nalézt v [1]. Řádek 4.-9. přidá do množiny $\mathcal{F}$ množinu povinných konceptů. A konečně řádek 10. - 15. postupně hledá koncepty, které pokrývají co nejvíce jedniček z univerza $\mathcal{U}$. Při každém přidání konceptu $\langle C, D\rangle$ do množiny $\mathcal{F}$ je tentýž koncept z množiny $\mathcal{S}$ odstraněn a „obdélník jedniček“, jenž daný koncept $\langle C, D\rangle$ pokrýval, je z univerza vymazán.

Uvedený algoritmus produkuje dobré výsledky v tom smyslu, že počet faktorů je běžně blízko optima. Kvůli nutnosti spočítat množinu všech konceptů (jež

může být veliká) a nutnosti jejího následného procházení při hledání konceptu, který pokrývá co nejvíce jedniček z univerza, může být algoritmus pro velká data pomalý.

4.4.2. Algoritmus druhý

Zásadní rozdíl prvního algoritmu oproti druhému algoritmu 2 je v hledání kandidáta na faktorový koncept. Algoritmus 2 nepočítá a následně neprochází množinu všech konceptů, nýbrž hledá kandidáta na faktorový koncept pomocí metody postupného přidávání „slibných sloupců“. Tato změna má velký dopad na časovou a paměťovou efektivitu algoritmu.

Metoda přidávání „slibných sloupců“ je založena na faktu, že pro každý formální koncept $\langle C, D\rangle$, D lze vyjádřit jako $D = \bigcup_{y\in D}\{y\}^{\downarrow\uparrow}$ a dále na pozorování, že pokud $y \notin D$, pak $\langle (D \cup \{y\})^{\downarrow}, (D \cup \{y\})^{\downarrow\uparrow}\rangle$ je formální koncept, pro který $D \subset (D\cup\{y\})^{\downarrow\uparrow}$. V tomto případě tak vybíráme $y \in Y$ takové, které maximalizuje

$$D \oplus j = ((D \cup \{y\})^{\downarrow} \times (D \cup \{y\})^{\downarrow\uparrow}) \cap \mathcal{U}.$$

Modifikací prvního algoritmu pak dostaneme algoritmus, který neprochází všechny formální koncepty, ale pouze přidává sloupce (atributy), které maximalizují hodnotu konstruovaného faktorového konceptu ve smyslu pokrytí největšího počtu jedniček z $\mathcal{U}$. Algoritmus druhý pak obecně produkuje jiný rozklad než algoritmus první.

input : I
output: $\mathcal{F}$

set $\mathcal{U}$ **to** $\{\langle i, j\rangle \mid I_{ij} = 1\}$
set $\mathcal{F}$ **to** $\emptyset$
while $\mathcal{U} \neq \emptyset$ **do**
  set D **to** $\emptyset$
  set V **to** 0
  while there is $j \notin D$ **such that** $|D \oplus j| > V$ **do**
    select $j \notin D$ **that maximizes** $D \oplus j$
      set D **to** $(D \cup \{j\})^{\downarrow\uparrow}$
      set V **to** $|(D^{\downarrow} \times D) \cap \mathcal{U}|$
  set C **to** $D^{\downarrow}$
  add $\langle C, D\rangle$ **to** $\mathcal{F}$
  foreach $\langle i, j\rangle \in C \times D$ **do**
    remove $\langle i, j\rangle$ **from** $\mathcal{U}$
return $\mathcal{F}$

Algorithm 2: Algoritmus druhý

4.5. Experiment

Pro otestování výše uvedených algoritmů (1, 2) jsem se rozhodl použít reálná binární data `mushrooms`[3]. Data obsahují 119 atributů. Oba algoritmy rozložili data pomocí 112 faktorů. Co se týče času potřebného pro provedení rozkladu, je algoritmus druhý, jak již bylo nastíněno v předchozí kapitole, mnohem rychlejší než algoritmus první. Konkrétně čas potřebný k úplnému rozkladu byl u prvního algoritmu 52 minut a 35 vteřin, u algoritmu druhého pak pouhých 38 vteřin.

5. Umělé neuronové sítě

Umělá neuronová síť (dále jen neuronová síť) je výpočetním modelem inspirovaným funkcí běžných biologických neuronových sítí. Neuronová síť je tvořena množinou neuronů (dále jen neurony) spojených synapsemi. Podoba spojení je běžně definována ve fázi učení, kdy jsou síti předkládány vzory z učící množiny. Neuronové sítě se obvykle používají k podchycení složitých vztahů nebo k nalezení skrytých vzorů v datech. Matematicky vzato neuronová síť definuje nějaké zobrazení $f : X \to Y$. Informace použity v celé této části jsem čerpal z [4] a [5].

5.1. Model umělého neuronu

Umělý neuron je jednotka, která z několika vstupních kanálů přijímá signály, ty zpracuje a na výstupní kanál vysílá výsledek. Kanály označujeme jako synapse. Úrovně signálů reprezentujeme reálnými čísly. Vstupem je tedy vektor $\mathbf{x} = (x_1, \ldots, x_N) \in \mathbb{R}^N$, kde N označuje počet vstupních synapsí. Dále je každé synapsi přiřazeno reálné číslo - váha. Vektor $\mathbf{w} = (w_1, \ldots, w_N) \in \mathbb{R}^N$ pak označuje hodnoty vah všech vstupních synapsí.

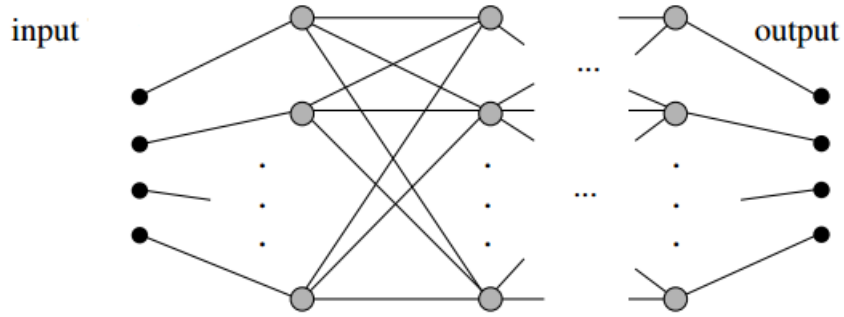
Samotný výpočet neuronu se skládá ze dvou částí. V první části je potřeba agregovat všech n vstupů do jedné hodnoty zvané „potenciál“. Toto se nejčastěji provádí tak, že vynásobíme vstupní hodnotu každé synapse její vahou a následně tyto hodnoty sečteme. Popsaný postup lze též vyjádřit jako skalární součin $\mathbf{xw}$.

$$p(\mathbf{x}, \mathbf{w}) = \sum_{i=1}^{N} x_i w_i = \mathbf{xw}$$

Ve druhé části se pak na potenciál aplikuje tzv. aktivační funkce $\delta = \mathbb{R} \to \mathbb{R}$. Aktivační funkce může být u různých neuronů různá. Aplikací aktivační funkce na potenciál dostaneme aktivaci neuronu označovanou jako $\delta(p)$.

5.2. Neuronová síť

Neuronovou síť si lze s jistou rezervou představit jako orientovaný graf, kde uzly jsou tvořeny neurony a hrany synapsemi. Výstupní synapse jednoho neuronu může být vstupní synapsí jiného neuronu, tím je zajištěna komunikace mezi neurony. Synapse, jež postrádají výchozí neuron, se nazývají vstupní a signál v nich tekoucí je vstupem celé sítě. Naproti tomu synapse, jež postrádají koncový neuron, se nazývají výstupní, a signál v nich tekoucí je součástí výstupu sítě. Na neuronovou síť o N vstupních synapsích a M výstupních synapsích se můžeme také dívat jako na výpočetní jednotku počítající funkci $f : \mathbf{R}^N \to \mathbf{R}^M$. Schéma neuronové sítě můžeme vidět na obr. 3.. Váhy synapsí mezi neurony potom ukládáme do matice $\mathbf{W}$, kde $\mathbf{w_{ij}}$ je hodnota váhy synapse vedoucí z neuronu i do neuronu j.



Obrázek 3. Schéma neuronové sítě [5]

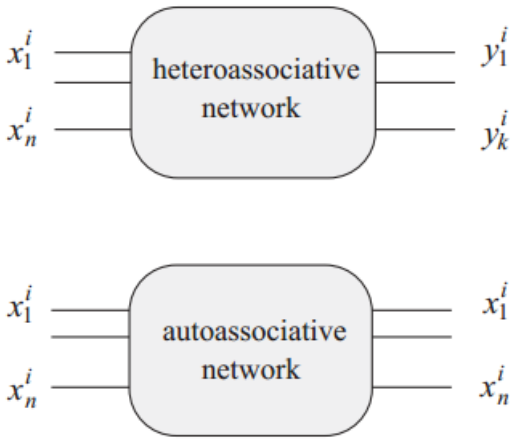
Tvar sítě, tedy způsob spojení jednotlivých neuronů mezi sebou, je označován jako *topologie* sítě. Topologie sítí dělíme na *dopředné* a *rekurentní*. V dopředné síti se nevyskytují cykly, neuron vypočítá svoji aktivaci v okamžiku, kdy jsou známy hodnoty signálu všech jeho vstupních synapsí. Na druhé straně stojí rekurentní sítě. Jak už ze samého názvu plyne, tento typ sítě cykly obsahuje. V tomto typu sítí též nelze jednoznačně definovat pravidlo pro pořadí výpočtu aktivací jednotlivých neuronů. Neuron se totiž jednoho ze svých vstupů nemusí nikdy dočkat. O výpočtu sítě zde mluvíme jako o výpočtu v čase, t.j. v každé časové jednotce jsou spočítány aktivace všech neuronů. Síť potom produkuje výsledek v každé takové časové jednotce. Rozdíl mezi topologiemi je tedy mimo jiné v tom, že první z uvedených produkuje pouze jeden výsledek, zatímco druhá z uvedených produkuje výsledků nekonečně mnoho.

5.3. Asociativní neuronové sítě

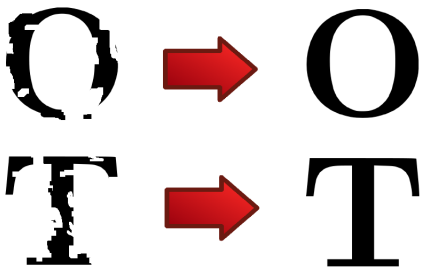
Cílem asociativních sítí je zapamatovat si vazbu nějakého vstupního vektoru $\mathbf{x}$ s nějakým výstupním vektorem $\mathbf{y}$, mluvíme zde o tzv. asociativní paměti. Síť by si pak měla vybavit $\mathbf{y}$ i v případě, že se na vstupu vyskytl vektor $\mathbf{x}'$, který se od původního $\mathbf{x}$ mnoho neliší. Toto chování je motivováno funkcí lidské asociativní paměti. Například pro lidi obvykle není problémem poznat známého člověka i po letech, kdy se jeho obličej trochu změnil.

Paměti dělíme do dvou typů, *heteroasociativní* a *autoasociativní*, viz obr. 4.. Heteroasociativní paměť se chová tak, jak je popsáno výše. Autoasociativní paměť je pak speciálním typem heteroasociativní paměti, kde $\mathbf{y} = \mathbf{x}$. Síť pak mapuje $\mathbf{x}$ na sebe sama. Druhá ze jmenovaných nachází využití hlavně v korekci poškozených vzorů - odstranění šumu. Síť naučíme nějakou množinu vzorů. Po předložení poškozeného vzoru $\mathbf{x}'$ si síť vybaví původní vzor $\mathbf{x}$, viz obr. 5..

Asociativní paměť můžeme implementovat buď se zpětnou vazbou či bez ní.



Obrázek 4. Typy asociativních pamětí [5]

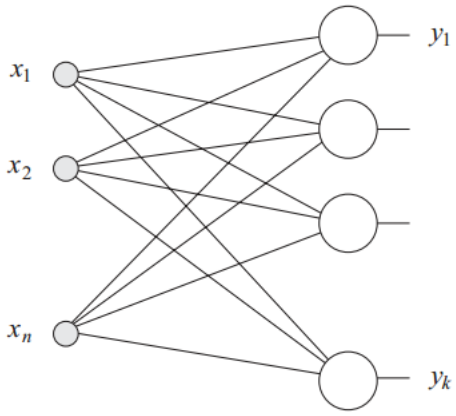


Obrázek 5. Korekce poškozeného vzoru

Síť bez zpětné vazby si vybaví výsledek v jednom kroku a tím její výpočet končí. Síť se zpětnou vazbou spočítá po předložení vstupního vzoru odpověď-výstup. Tuto odpověď však nevrátí, nýbrž ji znovu použije jako svůj vstup v další iteraci. Výpočet pak končí v momentě, kdy se síť ustálí - dostane se do pevného bodu, tj. vstupní vzor se rovná výstupnímu. O síti se zpětnou vazbou mluvíme také jako o síti rekurentní. Heteroasociativní síť bez zpětné vazby je vyobrazena na obr. 6., autoasociativní síť se zpětnou vazbou potom na obr. 7..

5.4. Hopfieldova síť

Model Hopfieldovy sítě lze zařadit do třídy autoasociativních pamětí. Model byl navržen v 80. letech minulého století fyzikem Johnem Hopfieldem. Každý neuron se chová jako nezávislá jednotka spolupracující se všemi ostatními jednotkami v síti, není zde nutná žádná synchronizace. Takto složitý systém Hopfield



Obrázek 6. Heteroasociativní síť bez zpětné vazby [5]

připodobnil k dynamickému systému ve fyzice, díky čemuž dobře analyzoval jeho chování.

5.4.1. Architektura

Hopfieldova síť je tedy plně spojená rekurentní neuronová síť. Každý neuron je synapsemi spojený se všemi ostatními neurony, krom se sebou samým. Synapse jsou symetrické, signál tedy může putovat oběma směry. Analogie s orientovaným grafem však stále platí, stačí nahradit symetrickou synapsi dvěma orientovanými hranami se stejnou vahou. Matice vah $\mathbf{W}$ je tedy symetrická a s nulovou diagonálou. Dále evidujeme stavy neuronů, které lze nastavit buď pomocí vstupního vektoru, nebo podle jejich aktivace z předchozího kroku. Počet neuronů v síti potom odpovídá dimenzi prostoru vstupních vzorů.

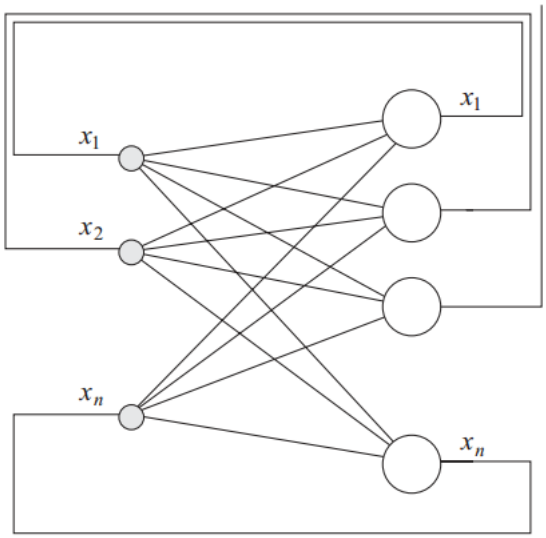
Vstupní vektory jsou obvykle kódovány bipolárně, což znamená, že složky vstupního vektoru mohou nabývat buď hodnoty -1 nebo 1, jinými slovy pro vektor x platí: $x \in \{-1, 1\}^N$. Každému neuronu je přiřazen nějaký práh θ. Aktivační funkce neuronu potom vypadá následovně:

$$\delta(x) = \begin{cases} 1 & x > \theta \\ -1 & x \leq \theta \end{cases}$$

.

5.5. Vybavování Hopfieldovy sítě

Přejděme nyní k algoritmu 3 pro vybavovaní Hopfieldovy sítě. Nechť N je počet neuronů v síti. Nechť x_i symbolizuje stav i-tého neuronu. Vstupem algoritmu je nějaký vzor $\mathbf{x}$, výstupem pak odezva sítě.



Obrázek 7. Autoasociativní síť se zpětnou vazbou [5]

Řádek 1. - 2. slouží k nastavení složek vstupního vzoru $\mathbf{x}$ jako stavů neuronů. Na řádku 4. - 5. vybereme náhodně nějaký neuron a na základě jeho aktivace změníme jeho stav. Speciální funkci v algoritmu plní množina U, která slouží ke zjištění ustálenosti neuronů, tedy zda jsme se již nedostali do pevného bodu dané sítě. Do množiny si ukládáme pořadová čísla neuronů, jejichž stav se po spočítání jejich aktivace v posledním kroku nezměnil. V případě že se nějakému neuronu aktivace změní, celá tato množina se vymaže. Pamatujeme si tak vlastně spojitou sekvenci kroků, při kterých se žádnému neuronu stav nezměnil. Pokud tato množina obsahuje pořadová čísla všech neuronů sítě, víme, že jsme skončili v pevném bodě dané sítě.

5.5.1. Hopfieldova síť jako dynamický systém

Hopfieldovu síť lze též chápat jako dynamický systém. Nejprve nastavíme počáteční stav systému (nastavení počátečního stavu neuronů), následně se systém dynamicky v čase mění (pomocí algoritmu se mění stavy neuronů), dokud se nedostane do tzv. stabilního stavu (výše uvedený pevný bod). Stabilních stavů může být obecně více. Do kterého z nich se systém dostane pak záleží na počátečním nastavení. Stabilnímu stavu říkáme *atraktor*, z anglického „to attract“ = přitahovat. Stabilní stav totiž k sobě přitahuje stavy ze svého okolí (v tom smyslu, že tyto stavy do něj přechází). Atraktor je v kontextu Hopfieldovy sítě charakterizován vektorem stavů neuronů. Pro lepší čitelnost textu budu od nynějška i tento vektor nazývat atraktorem. V souvislosti s Hopfieldovou sítí se mluví o tzv. *asynchronní dynamice* systému. Tento pojem znamená, že neurony

input : vstupní vzor $\mathbf{x}^{in}$
output: odezva sítě

for $i \leftarrow 1$ **to** N **do**
  set x_i **to** x_i^{in}
while $U \neq \{1, \ldots, N\}$ **do**
  pick j **at random such that** $j \in \{1, \ldots, N\}$
  set x_j **to** $\delta(\sum_{i=1}^{N} x_i w_{ji})$
  if x_j **changed state then**
    set U **to** $\emptyset$
  else
    set U **to** $U \cup \{j\}$
return x

Algorithm 3: Vybavování Hopfieldovy sítě

nejsou při výpočtu nijak synchronizovány jako tomu může být u jiných typů sítí, ale neuron, jehož aktivace se bude počítat, je vybrán náhodně.

Přejděme nyní k definici energetické funkce. Taková funkce slouží k vystižení stavu nějakého složitého dynamického systému pomocí jedné skalární hodnoty. Díky výše popsané analogii budeme moci takto vystihnout i to, v jakém stavu se nachází Hopfieldova síť.

Definice 5..1. *Uvažujeme-li Hopfieldovu síť s N neurony, váhovou maticí $\mathbf{W}$ a vektorem prahů neuronů θ, je* energetická funkce *zobrazení: $E : \{-1, 1\}^N \Rightarrow \mathbf{R}$.*

$$E(x) = -\frac{1}{2} \sum_{j=1}^{N} \sum_{i=1}^{N} w_{ij} x_i x_j + \sum_{i=1}^{N} \theta_i x_i$$

kde $\mathbf{x}$ je stav sítě (vektor stavů neuronů). Energetickou funkci lze též zapsat v maticovém zápisu:

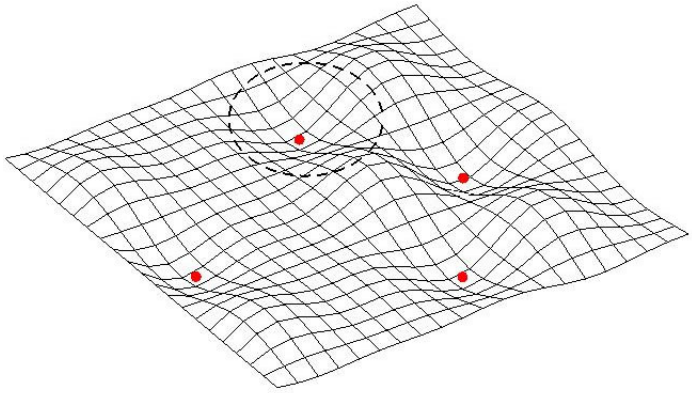
$$E(\mathbf{x}) = -\frac{1}{2} \mathbf{x} \mathbf{W} \mathbf{x}^T + \theta \mathbf{x}$$

Následující věta říká, že energetická funkce má lokální minima ve stabilních stavech (atraktorech) Hopfieldovy sítě, a navíc, že proces vybavovaní je vždy konečný.

Věta 5..1. *Hopfieldova síť s asynchronní dynamikou, která začíná z libovolného stavu, se vždy dostane do stabilního stavu (do lokálního minima energetické funkce).*

Důkaz této věty je možno nalézt v [5].

Možný průběh energetické funkce lze vidět na obrázku 8.. Předpokládáme zde, že vektor stavů neuronů lze nějakým způsobem zobrazit do roviny. Zvlnění



Obrázek 8. Možný průběh energetické funkce [11]

povrchu pak představuje hodnoty energetické funkce. Na obrázku jsou červenými puntíky vyobrazeny lokální minima takové funkce (tedy atraktory dané sítě). Kruh vyobrazený přerušovanými čarami symbolizuje tzv. *atraktor basin*. Jde v podstatě o množinu počátečních stavů, které po provedení algoritmu 3. přejdou do daného atraktoru.

5.6. Učení v Hopfieldově síti

Pokud bychom po síti chtěli, aby si vybavovala do ní „nahranou“ množinu M vzorů $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_M}$ dimenze N, tedy bychom chtěli aby se nahrané vzory staly atraktory dané sítě, potřebujeme najít adekvátní váhy pro synapse mezi jednotlivými neurony, tedy matici vah $\mathbf{W}$.

Jedním z jednodušších a zároveň hojně využívaným způsobem jak získat matici vah $\mathbf{W}$ je tzv. Hebbovské učení. Učení je založené na jednoduchém pravidle navrženém v roce 1949 psychologem Donaldem Hebbem a lze ho shrnout do jedné věty:

„Vazba mezi neurony, které jsou společně aktivovány, by se měla posílit, zatímco vazba mezi neurony, které nejsou společně aktivovány, by se měla oslabit.“

Pravidlo lze algebraicky vyjádřit následovně:

$$\Delta w_{ij} = \eta x_i x_j$$

kde x_i a x_j je i-tá a j-tá složka vstupního vektoru, η je pak parametr určující rychlost učení.

Jak probíhá Hebbovo učení v Hopfieldově síti je popsáno následujícím algoritmem 4.

Vstupem algoritmu je množina T složená z M vzorů, kde každý vzor obsahuje

input : $T = \{\mathbf{x}^k\}_{k=1}^{M},\ \mathbf{x}^k \in \{-1, 1\}^N$
output: matice vah $\mathbf{W}$

1 **set** W_{ij} **to** 0
2 **for** $k \leftarrow 1$ **to** M **do**
3 | **set** W_{ij} **to** $W_{ij} + x_i{}^k x_j{}^k, \qquad i, j = 1, \ldots, N;\ i \neq j$
4 **return** $\mathbf{W}$

Algorithm 4: Hebbovo učení Hopfieldovy sítě

N složek. Algoritmus na 1. řádku nastaví všechny váhy na nulu a poté pro každý vzor danou váhu podle výše zmíněného Hebbova pravidla upraví. V základním modelu asociativní paměti budeme uvažovat neurony s nulovými prahy. Algoritmus si lze představit jako hlasování mezi neurony, w_{ij} je pak rozdíl mezi počtem souhlasných a nesouhlasných stavů x_i^k, x_j^k.

Vyjádříme-li výše uvedený algoritmus „pomocí matic“, dostaneme

$$\mathbf{W}_k = \mathbf{x}^k \diamond \mathbf{x}^k - \mathbf{I}$$

kde, symbol $\diamond$ slouží pro označení vnějšího vektorového součinu.

5.7. Úvaha o vhodnosti Hebbova učení

V případě, že tedy hodláme Hopfieldovu síť použít jako autoasociativní paměť, je dobré ověřit, za jakých podmínek a zda vůbec si bude síť nahrané vzory vybavovat, za jakých podmínek budou nahrané vzory jejími atraktory.

Uvažujme nejprve případ, kdy po síti chceme, aby si pamatovala pouze jeden vzor x^1. Matice vah příslušející x^1 vypadá následovně:

$$\mathbf{W}^1 = \mathbf{x}^1 \diamond \mathbf{x}^1 - \mathbf{I}$$

Energetická funkce nyní vypadá následovně:

$$E(\mathbf{x}) = -\frac{1}{2}\mathbf{x}\mathbf{W}^1\mathbf{x}^T = -\frac{1}{2}(\mathbf{x}\mathbf{x}^1 \cdot \mathbf{x}^1\mathbf{x} - \mathbf{x}\mathbf{x}) = -\frac{1}{2}(||\mathbf{x}\mathbf{x}^1||^2 - n)$$

Nyní již snadno vidíme, že lokální minimum se opravdu nachází v $\mathbf{x}^1$. K minimalizaci $E($**x** dochází maximalizací výrazu $\mathbf{x}\mathbf{x}^1$ a ten je za použití bipolárního kódování v $\mathbf{x}^1$ opravdu maximální. Nahraný vzor $\mathbf{x}^1$ je tedy stabilním stavem sítě.

V případě, že do sítě chceme uložit více vzorů, tedy $\mathbf{x}^1, \ldots \mathbf{x}^M$, bude váhová matice vypadat takto:

$$\mathbf{W}^1 = (\mathbf{x}^1 \diamond \mathbf{x}^1 - \mathbf{I}) + \cdots + (\mathbf{x}^M \diamond \mathbf{x}^M - \mathbf{I})$$

Za předpokladu, že síť inicializujeme vektorem $\mathbf{x}^1$, pak vektor potenciálů všech neuronů v síti spočítáme následovně:

$$\mathbf{v} = \mathbf{x}^1\mathbf{W} = \mathbf{x}^1(\mathbf{x}^1 \diamond \mathbf{x}^1) + \cdots + \mathbf{x}^1(\mathbf{x}^M \diamond \mathbf{x}^M) - M\mathbf{x}^1\mathbf{I} = (N-M)\mathbf{x}^1 + \sum_{j=2}^{M}(\mathbf{x}^1\mathbf{x}^j)\mathbf{x}^j$$

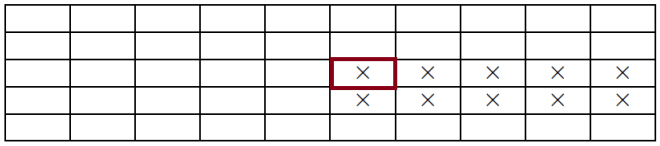
Vektor $\mathbf{x}^1$ je potom stabilní, pokud $sgn(\mathbf{v}) = sgn(x_1)$. Z výše vypočítaného vektoru potenciálů lze vidět, že toto nastává právě když $M < N$ a výraz $\sum_{j=2}^{M}(\mathbf{x}^1\mathbf{x}^j)\mathbf{x}^j$, který označujeme jako *crosstalk* je dostatečně malý. Crosstalk lze interpretovat jako míru interference mezi vzory, v tomto případě jde o míru překryvu mezi vzory. Crosstalk je nulový, pokud jsou nahrané vzory (vektory) vzájemně ortogonální.

Jinými slovy, pro správné vybavovaní vzorů je nutné, aby počet vzorů do sítě nahraných byl menší než počet neuronů v síti a zároveň aby si nahrané vzory byly co nejméně podobné. V praxi pak samozřejmě má vliv také to, jaký je překryv mezi vstupním vzorem a ostatními nahranými vzory mimo vzoru, který se snažíme vybavit.

Situaci si lze představit také tak, že čím „méně“ jsou výše uvedená kritéria splněna, tím větší je šance, že se síť ustálí v atraktoru, který neodpovídá žádnému z nahraných vzorů.

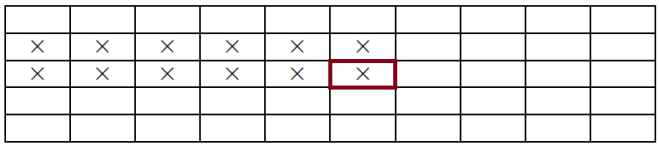
5.8. Příklad 1.

Řekněme, že do paměti (sítě) chceme nahrát vzory $\mathbf{x}_1, \mathbf{x}_2 \in \{-1, 1\}^{50}$ (obr. 9., obr. 10.). Tedy chceme, aby tyto dva vzory byly zároveň atraktory této sítě. K tomu použijeme algoritmu Hebbova učení 4.

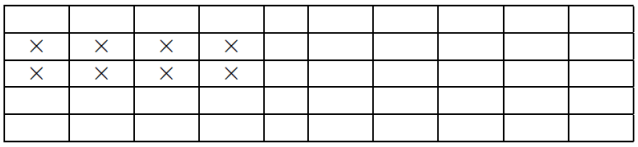


Obrázek 9. Příklad 1, vzor $\mathbf{x}_1$

Nyní přejděme k vybavovaní. Síti na vstup předložíme vzor $\mathbf{x}^{in}$ (obr. 11.), který se od vzoru $\mathbf{x}_2$ mnoho neliší. Očekáváme tedy, že tento vstupní vzor přejde po provedení algoritmu 3 do atraktoru $\mathbf{x}_2$. Pro toto očekávání máme dobré důvody. Počet nahraných vzorů (2 vzory) je opravdu menší než počet neuronů v síti (50 neuronů). Dále překryv mezi nahranými vzory (mezi $\mathbf{x}_1$ a $\mathbf{x}_2$) je nízký a tedy i výsledný crosstalk bude v tomto případě nízký. Zároveň i překryv mezi vstupním vzorem $\mathbf{x}^{in}$ a ostatními vzory v paměti mimo vzor $\mathbf{x}_2$ je nízký, v tomto případě konkrétně žádný.



Obrázek 10. Příklad 1, vzor $\mathbf{x}_2$

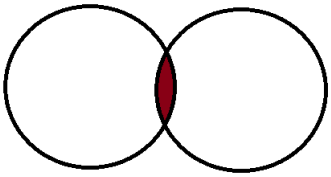


Obrázek 11. Příklad 1, vstupní vzor $\mathbf{x}^{in}$

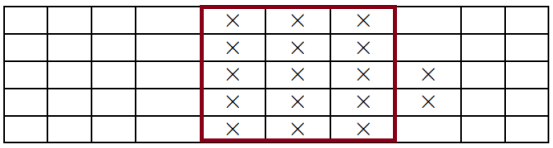
Na obrázku 12. je vidět, jak by mohly vypadat basiny jednotlivých atraktorů vzoru $\mathbf{x}_1$ a vzoru $\mathbf{x}_2$. S trochou nepřesnosti můžeme říci, že míra překryvu mezi basiny jejich atraktorů odpovídá míře překryvu mezi samotnými atraktory. Překryv jednotlivých basinů je vyznačen červeně.

5.9. Příklad 2.

Podívejme se nyní na ne již tolik optimistický příklad. Opět chceme nahrát vzory $\mathbf{x}_1, \mathbf{x}_2 \in \{-1, 1\}^{50}$ (obr. 13., obr. 14.). Jak je vidět, crosstalk mezi nahranými vzory je vysoký, překryv mezi vzory je oblast vyznačená červeně, stejně tak je vysoký i překryv jejich atraktorových basinů (obr.16.). Pokud takové síti předložíme jako vstup vzor $\mathbf{x}^{in}$ (obr.15.), nemáme vůbec žádnou jistotu o tom, v kterém z atraktorů se síť ustálí. V takovémto patologickém případě se dokonce může stát, že se síť neustálí ani v jednom z atraktorů odpovídajícímu některému ze vstupních vzorů.



Obrázek 12. Příklad 1, atraktorové basiny



Obrázek 13. Příklad 2, vzor $\mathbf{x}_1$

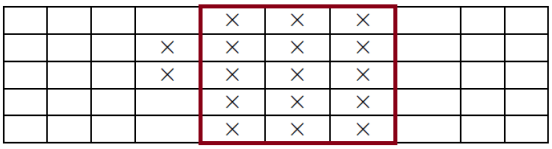
6. Booleovská faktorová analýza pomocí atraktorové neuronové sítě

V této části bych rád představil metodu booleovské faktorové analýzy pomocí atraktorové neuronové sítě (dále jen BFAANN) vyvinutou Alexandrem Frolovem a kolektivem v prvním desetiletí 21. století. V této části budu vycházet zejména z článku [6], dále pak také z [7]. V původním článku [6] je metoda popsána pouze slovně a poměrně těžko stravitelně. Její popsání jednodušší a přehlednější formou, tak aby ji čtenář byl schopen bez větších potíží eventuálně implementovat, je též jedním z cílů této diplomové práce.

Tuto kapitolu začnu úvodem do problému booleovské faktorové analýzy (6.1.), tak jak ho chápou autoři BFAANN, pokračovat budu mojí vlastní úvahou (6.2.), jak by čtenář obeznámený s funkcí Hopfieldovy sítě jako asociativní paměti, mohl dojít k nápadu, jak takovou síť využít ke hledání faktorů v datech.

V průběhu popisu metody se dále budou vyskytovat hypotézy, jak které části metody BFAANN korespondují s metodou hledání faktorů pomocí FKA (dále jen BFAFKA). Jinými slovy se budu snažit uvádět, jaký by mohl být vztah mezi těmito dvěma metodami.

Závěr této kapitoly pak obsahuje několik experimentů, které demonstrují, že se moje implementace metody opravdu chová správně v tom smyslu, že pro stejná data produkuje stejné výsledky, jaké autoři publikovali v již zmíněných článcích. Toto by též mělo jistým způsobem podtrhnout pravdivost mého popisu metody BFAANN a také toho, že jsem metodu pochopil a implementoval správně.



Obrázek 14. Příklad 2, vzor $\mathbf{x}_2$

Obrázek 15. Příklad 2, vstupní vzor $\mathbf{x}^{in}$

6.1. Úvod a značení

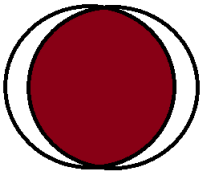
Tato kapitola slouží k tomu, aby čtenáře seznámila s tím, jak o binární faktorové analýze uvažují autoři BFAANN. Značení jsem se rozhodl zachovat takové, jaké používají autoři v článcích, a to z toho důvodu, aby čtenář nebyl zmaten, kdyby si chtěl zmíněné články sám přečíst. Toto platí i o dalších kapitolách a pojmech používaných v dalších kapitolách. Bohužel ani tak se čtenář jistému zmatení nevyhne, sami autoři totiž často používají pro stejné věci v různých článcích různého značení.

Binární faktorovou analýzu potom autoři charakterizují následovně: každý originální signál (vzor v oblasti asociativních pamětí, objekt v oblasti FKA), vektor $\mathbf{x} \in \{0,1\}^N$ (kde N je dimenzí prostoru signálů) může být reprezentován jako booleovská suma vážených binárních faktorů:

$$\mathbf{x}^T = \mathbf{s}^T\mathbf{F} = \bigvee_{l=1}^{L} s_l \mathbf{f}^l$$

kde $\mathbf{F}$ je matice $L \times N$ loading faktorů (v originále faktor loadings, jde o atributy přítomné v daném faktoru), L faktorů jsou zde uloženo jako řádky této matice, tato matice je tedy ekvivalentní s faktor-atribut maticí v BFAFKA, kde dále vektor $\mathbf{s}$ o počtu složek L značí skóre faktorů (v originále factor scores). Složky vektoru $\mathbf{s}$ nám říkají, z jakých všech faktorů se originální signál skládá. Jde tedy o řádek matice objekt-faktor v BFAFKA.

Pro obecně M signálů pak situace vypadá následovně. Signály jsou uloženy v matici $\mathbf{X}$ o rozměrech $M \times N$, signály jsou tedy uloženy v řádcích matice $\mathbf{X}$. Dále pak jednotlivé skóry faktorů jsou uloženy v řádcích matice $\mathbf{S}$ o rozměrech



Obrázek 16. Příklad 2, atraktorové basiny

$M \times L$.

Na binární faktorovou analýza můžeme též hledět jako na proceduru, která zobrazuje originální signály do prostoru faktorů. Zobrazení z originálního prostoru do prostoru faktorů potom znamená, že signály budeme reprezentovat vektory $\mathbf{s}^m$ (řádky matice $\mathbf{S}$) namísto vektory $\mathbf{x}^m$ (řádky matice $\mathbf{X}$).

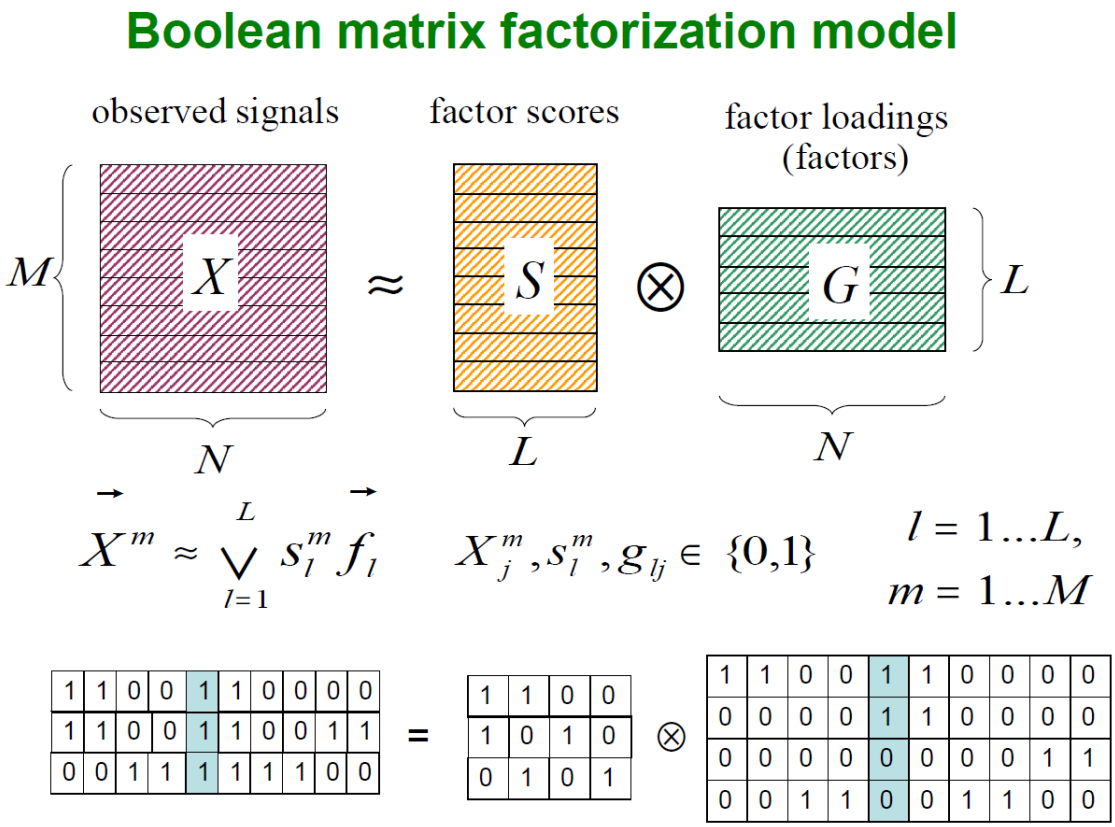
Situaci i s příkladem lze vidět na obr. 17. z prezentace jednoho z autorů Pavla Polyakova [9]. Autor použil mírně jiného značení. Konkrétně matici loading faktorů, kterou jsem zde označil jako $\mathbf{F}$, označil autor jako $\mathbf{G}$.

6.1.1. Omezení

Pro funkci metody předpokládáme několik omezení, co se týče podoby jak hledaných faktorů, tak vstupních signálů $\mathbf{X}$. Za prvé předpokládáme, že faktory jsou kódovány řídce. Tím je myšleno, že počet jedniček ve faktoru (n) je mnohem nižší než dimenze prostoru signálů N, jde tedy o poměr $p = \frac{n}{N}$. Za druhé autoři předpokládají, že ačkoli počet faktorů namixovaných do jednoho signálů může být poměrně vysoký, musí být stále mnohem nižší, než celkový počet faktorů. Průměrný počet faktorů namixovaných do jednoho signálu nazývají autoři v článcích komplexitou signálu a označují ji jako C. Důvody pro tato omezení uvedu později (6.11.).

6.2. Od asociativní paměti k hledání faktorů

Vraťme se však ještě nyní na okamžik k příkladu z kapitoly 5.9.. Jediné, co o tomto příkladu můžeme s relativní jistotou říci je, že atraktor, ve kterém se síť ustálí, bude velice pravděpodobně obsahovat oblast překryvu mezi vzory $\mathbf{x}_1$ a $\mathbf{x}_2$. Intuitivně to můžeme očekávat z ideje Hebbova učení, kdy se snažíme utužit vazby mezi neurony aktivovanými v procesu učení společně. Vazby neuronů v již zmíněné oblasti překryvu budou zcela jistě spojeny silnější vazbou, než neurony mimo ni. Tedy pokud aktivujeme nějakou podmnožinu neuronů z této oblasti, máme velice vysokou šanci, že v průběhu vybavovacího procesu budou postupně aktivovány i zbylé neurony z této oblasti. Dalo by se říci, že přítomnost



Obrázek 17. Binární faktorová analýza

této relativní jistoty vede k základní myšlence za tím, jak využít Hopfieldovu síť k hledání faktorů v datech.

Obecně vzato, tyto překryvy mezi vzory, o kterých jsme doposud hovořili, potom, co tyto vzory naskládáme pod sebe jako řádky matice, nejsou totiž v konečném důsledku nic jiného, než právě maximálními obdélníky v dané matici.

Na situaci se můžeme podívat též tak, že nám v podstatě jde o opak, než nám šlo v případě využití sítě jako autoasociativní paměti. K vybavování vzorů je zapotřebí, aby vzory mezi sebou interferovaly co nejméně, ale nyní, při hledání faktorů, se této interference snažíme využít a těžíme z její maximalizace. Pokud tedy chceme Hopfieldovu síť použít pro hledání faktorů, musíme nějakým způsobem zařídit, aby síť již déle neměla atraktory v nahraných vzorech, ale měla je ve zmíněných překryvech mezi vzory. Takový atraktor by pak v příkladu 5.9., při zakreslení celé situace do roviny, ležel někde v oblasti překryvu (vyznačené červeně) mezi basiny původních atraktorů, viz obr. 16..

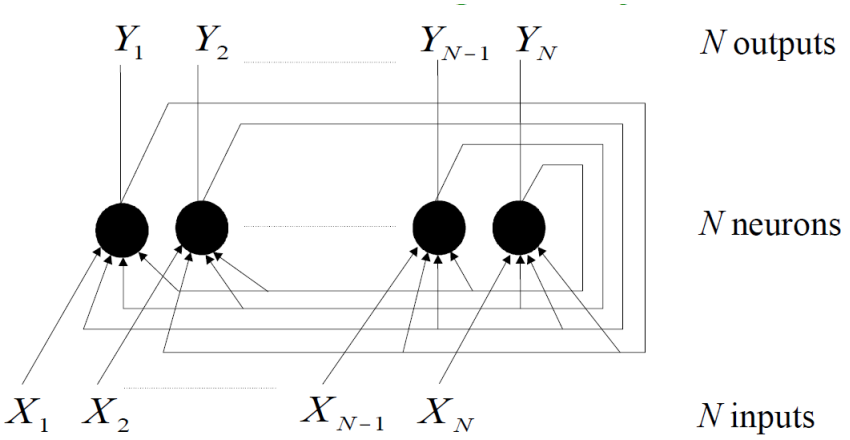
K tomu, jak zajistit, aby Hopfieldova síť měla atraktory v popsaných překryvech, je zapotřebí upravit jak běžně používané pravidlo, tak samotný proces vybavování (dynamiku sítě). Atraktory o nichž jsem mluvil doposud, budu v dalším

textu označovat jako *pravé atraktory*. Termín *atraktor* si ponechám pro lokální atraktor - tedy atraktor v nějakém stupni aktivity k, viz kapitola 6.6.1..

6.3. Atraktorová neuronová síť se zvyšující se aktivitou

Síť, kterou zde budeme používat k objevování faktorů, autoři v původním článku ([6]) nazvali „Atraktorová neuronová síť“ v originále pak „Attractor Neural Network“ (ANN), v pozdějších článcích ([7],[8]) změnili její jméno na „Atraktorová neuronová síť se zvyšující se aktivitou“, v originále „Attractor Neural Network with Increasing Activity“ (ANNIA).

Jde tedy o úplně spojenou rekurentní neuronovou síť Hopfieldovského typu. Síť se skládá z N neuronů, N vstupů a N výstupů. Každý neuron tedy koresponduje s jednou složkou vstupního signálu. Váhy synapsí jsou uloženy v matici $\mathbf{J}$, která je symetrická a s nulovou diagonálou. Situaci můžeme vidět na obr. 18..



Obrázek 18. ANNIA

Z pohledu FCA koresponduje každý neuron s nějakým atributem. Vektor stavů neuronů $\mathbf{x}$ je pak v podstatě charakteristickým vektorem nějaké množiny atributů $A^{\mathbf{x}}$ v tom smyslu, že aktivní neurony symbolizují atributy nacházející se v dané množině. O této množině ($A^{\mathbf{x}}$) se budu dále vyjadřovat jako o množině indukované vektorem $\mathbf{x}$.

V následujícím popisu algoritmu budu označovat proměnnou *network* uspořádanou dvojici matice vah $\mathbf{J}$ a vektoru stavů neuronů $\mathbf{x}$, tedy $network = \langle \mathbf{J}, \mathbf{x} \rangle$, kde $\mathbf{x} \in \{0,1\}^N$. Přistupovat k prvkům této dvojice budu pomocí tečky. Tedy například matici vah dostanu pomocí příkazu $network.\mathbf{J}$.

6.4. Hlavní smyčka

V této a v dalších kapitolách postupně popíši, jak metoda BFAAN pracuje.

Každou její část též popíši algoritmem.

Vstupem celé metody je tedy matice $\mathbf{X}$ vstupních signálů. Výstupem je pak matice $\mathbf{F}$ faktorů a matice $\mathbf{S}$ skórů jednotlivých faktorů. Hlavní smyčka celé metody (funkce $BFAANN(\mathbf{X})$ je popsána algoritmem 5.

input : matrix of signals $\mathbf{X}$
output: matrix of factors $\mathbf{F}$ and matrix of factor scores $\mathbf{S}$

```
1  SetConstants()
2  set F_ij to 0
3  set S_ij to 0
4  set failedTrialsCount to 0
5  set J to ComputeWeights(X)
6  repeat
7      set attractor to Trial(network)
8      if FactorP(attractor, network) then
9          UnlearnAttractor(network, attractor)
10         set F_l_ to attractor.x
11         set S_l to SignalsContainingFactor(X, attractor.x)
12         increment l
13         set failedTrialsCount to 0
14     else
15         increment failedTrialsCount
16     return F and F
17 until X = S · F or failedTrialsCount = maximumFailedTrialsCount
```

Algorithm 5: Funkce $BFAANN(\mathbf{X})$

V algoritmu 5 nejprve pomocí funkce $SetConstants()$ určíme hodnoty různých konstant, nastavení těchto konstant pak nějakým způsobem ovlivňuje vybavovaní faktorů v síti. To, k čemu takové konstanty slouží a jaké jsou jejich obvyklé hodnoty vždy ve vhodném místě uvedu. Na řádku (2. - 3.) pak inicializujeme všechna políčka matice $\mathbf{F}$ a $\mathbf{S}$ na nulu. Na 5. řádku spočteme matici vah $\mathbf{J}$. Na řádku 7. provedeme tzv. trial (pokus), kapitola 7. Výsledkem pokusu je vždy nějaký atraktor s určitým počtem aktivovaných neuronů (později také referovaným jako atraktor v nějakém stupni aktivity sítě), jestli tento atraktor označíme jako faktor se rozhodneme podle nějakého kritéria, zde symbolizovaného predikátovou funkcí $FactorP(attractor, network)$ na řádku 8, kapitola 6.10.. Pokud se opravdu jedná o faktor, pak je potřeba jej ze sítě odučit, aby v dalších iteracích nepřekážel v hledání ostatních faktorů, kapitola 6.9.. Nalezený faktor si uložíme do matice $\mathbf{F}$. Následně spočítáme pomocí $ComputeScore(\mathbf{X}, \mathbf{x}^f)$ vektor značící, které všechny signály tento nově nalezený faktor obsahují, tento pak uložíme do matice $\mathbf{S}$ jako její sloupec, kapitola 6.10.. Množina indukovaná vektorem $\mathbf{x}$ je

v podstatě intentem konceptu korespondujícího faktoru v rétorice FCA. To, co funkce $ComputeScore(\mathbf{X}, \mathbf{x}^f)$ potom počítá by se dalo nazvat extentem konceptu korespondujícího faktoru.

Řádky 8. - 14. provádíme do té doby, dokud buď nenalezneme všechny faktory potřebné pro rozklad matice, nebo síť přestane vracet rozumné výsledky. To znamená, že síť skončila v atraktoru, který se neukázal býti faktorem, více než $maximumFailedTrialsCount$ v řadě, přičemž $maximumFailedTrialsCount$ je nějaká námi zvolená konstanta.

6.5. Učení

Jak jsem již zmínil, cílem algoritmu učení sítě pro její použití jako asociativní paměti 5.3. bylo, aby se nahrávané vzory (nyní signály, tedy řádky matice $\mathbf{X}$) staly atraktory sítě.

Teď je ale naší snahou aby se faktory, ve smyslu maximálních obdélníků v matici $\mathbf{X}$ (matice I v BFAFKA), staly atraktory sítě. Ačkoli nám v tomto snažení napomáhá již fakt, že při BFA se očekává nějaký překryv mezi signály (na rozdíl od asociativní paměti) a je tedy dost dobře možné, že některé faktory se stanou atraktory sítě již za použití algoritmu vybavování 3 (což by, opět, bylo v případě asociativní paměti nežádoucí). Stále je však toto chování (faktory jako atraktory) třeba umocnit. Tohoto umocnění je dosaženo pomocí několika úprav v algoritmu učení. Primárně hlavně úpravou Hebbova pravidla.

Matice synaptických spojení $\mathbf{J}'$ pro síť ANNIA vypadá následovně:

$$J'_{ij} = \sum_{m=1}^{M} (x_i^m - q^m)(x_j^m - q^m), \qquad i, j = 1, \ldots, N; \quad i \neq j; \quad J'_{ii} = 0$$

kde $\mathbf{x}^m$ je vektor vstupního signálu - řádek matice $mathbfX$ a kde

$$q^m = \frac{\sum_{i=1}^{N} x_i^m}{N}$$

je celková aktivita m-tého signálu. Tato forma biasu (v češtině je v tomto kontextu pojmu nejblíže asi pojem „ovlivnění“), tedy že globální inhibice je proporcionální celkové aktivitě daného neuronu, je podle autorů z pohledu biologie, přesvědčivá.

Takto prezentoval matici synaptických spojení Pavel Polyakov ve své přednášce [9]. Dále ji takto prezentovali autoři v článku [8].

6.5.1. Inhibiční neuron

V článku, který označuji jako původní [6], však autoři představili ještě další formu inhibice, tzv. inhibiční neuron. Inhibiční neuron je neuron spojený se všemi

základními N neurony sítě (neurony představeny v kapitole 18.) pomocí obousměrných synapsí. Neuron je aktivován během prezentace každého signálu z učící množiny. Vektor **j**, definovaný jako

$$j_i'' = \sum_{m=1}^{M}(x_i^m - q^m) = M(q_i - q), \qquad i = 1, \ldots, N$$

kde $q_i = \frac{\sum_{m=1}^{M} X_i^m}{M}$ a $q = \frac{\sum_{i=1}^{N} q_i}{N}$,

označuje váhy synapsí mezi inhibičním neuronem a každým základním neuronem sítě. Zároveň předpokládáme, že excitabilita tohoto inhibičního neuronu klesá inverzně vzhledem k velikosti učící množiny a je $1/M$ potom, co jsou všechny signály do sítě nahrány. Aktivita inhibičního neuronu je po skončení učení takováto:

$$A(t) = (1/M)\sum_{i=1}^{N} J_i'' X_i(t) = (1/M)\mathbf{J}''^T\mathbf{X}(t)$$

Takováto globální inhibice je užitečná ze dvou důvodů:

- úplně potlačuje dva globální falešné atraktory,
- pokud je komplexita signálů C výrazně vyšší než 1 (každý signál je složen z výrazně více jak jednoho faktoru), pak globální inhibice způsobí, že velikost basinů jednotlivých atraktorů je nyní nezávislá na komplexitě signálu C.

Toto je samozřejmě výhodné pro samotné vybavování sítě. Ve skutečnosti platí, že přidání inhibičního neuronu je ekvivalentní s nahrazením běžné matice $\mathbf{J}'$ maticí $\mathbf{J} = \mathbf{J}' - M\mathbf{q} \diamond \mathbf{q}^T$, kde **q** je vektor o složkách $q_i - q$. Argument za tímto tvrzením je možno nalézt v [6].

6.5.2. Motivace za pravidlem

Motivace za popsanými pravidly pro učení sítě ANNIA je následující:

- synaptické spojení mezi i-tým a j-tým neuronem je tím silnější, čím vyšší je frekvence vzájemného výskytu atributu i a j ve vstupních datech,
- množina vzájemně silně spojených neuronů pak reprezentuje faktor. [9]

Na obrázku 19. vidíme matici vstupních dat vyobrazenou podobně jako vyobrazujeme formální kontext. Jak lze vidět frekvence vzájemného výskytu atributů $A = \{a_2, a_3, a_4, a_5, a_6\}$ je vzhledem k možné jiné podmnožině daných atributů vysoká. Podle prvního bodu výše zmíněné motivace by tedy spojení mezi korespondujícími neurony mělo být silné a podle druhého bodu pak takové silné spojení reprezentuje faktor.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
x_1		×						×	×	
x_2					×	×			×	
x_3		×	×	×	×	×				
x_4	×	×	×	×	×	×				
x_5		×	×	×	×	×				
x_7		×	×	×	×	×	×	×		
x_8		×	×	×	×	×	×	×		
x_9		×	×	×	×	×	×			×
x_{10}						×	×	×		×
x_{11}		×			×		×			×
x_{12}			×	×	×		×	×		

Obrázek 19. Matice vstupních dat

Není pravděpodobně žádnou náhodou, že z hlediska FCA je výše zmíněná množina atributů A intentem konceptu $\langle\{x_3, x_4, x_5, x_6, x_7, x_8, x_9\}, \{a_2, a_3, a_4, a_5, a_6\}\rangle$.

6.5.3. Algoritmus

Proces učení, tedy implementaci funkce $ComputeWeights(\mathbf{X})$, můžeme vidět v algoritmu 6.5.. Symboly q^m a $\mathbf{q}$ označují proměnné již dříve definované v této kapitole. Za zmínku zde stojí nepovinný 5. řádek, jehož presence či absence rozhodne o tom, zda použijeme či nepoužijeme inhibiční neuron.

input : matrix of signals $\mathbf{X}$
output: matrix of weights $\mathbf{J}$

1 **set** J'_{ij} **to** 0
2 **for** $m \leftarrow 1$**to**M **do**
3 | **set** J'_{ij} **to** $J'_{ij} + (x_i^m - q^m)(x_j^m - q^m), \qquad i, j = 1, \ldots, N; , i \neq j$
4 **set J to J′**
5 **set J to J** $- M\mathbf{q} \diamond \mathbf{q}^T$
6 **return J**

Algorithm 6: Funkce $ComputeWeights(\mathbf{X})$

6.6. Dynamika sítě, vybavování faktorů

K úspěšnému vybavování faktorů, namísto vybavování nahraných vzorů (signálů), musíme dále pozměnit klasickou dynamiku Hopfieldovy sítě - její vybavovací proces.

6.6.1. Průběh pokusu (Trial)

V následujícím popisu algoritmu budeme jako atraktor ve stupni uvažovat uspořádanou šestici:

$$attractor = \langle \mathbf{x}, \lambda, T, R, R', Sim \rangle$$

kde $\mathbf{x}$ je vektorem stavů neuronů v daném atraktoru, o zbylých prvcích této šestice bude řeč později. Dále budeme potřebovat množinu atraktorů $Attractors$ definovanou následovně:

$$Attractors = \{\langle k, attractor \rangle \mid k \in \{k_{in}, \ldots, k_{fin}\}\}$$

.

Množina obsahuje jako své prvky uspořádané dvojice, kde se na první pozici nachází číslo k a na druhé výše uvedený atraktor. Číslo k značí stupeň aktivity pro daný atraktor, ale o tom až později. K atraktoru ve stupni aktivity k uloženém v množině $Attractors$ se budu v algoritmu odkazovat pomocí $Attractors[k]$.

V dalším textu budu pro zjednodušení uvažovat, že síť skončí vždy pouze v bodovém atraktoru. V reálné situaci pak může síť skončit ještě v cyklickém atraktoru. Taková situace je popsána v článku [6]. Mnou implementovaný „framework“ pro experimentování se sítí ANNIA pak pracuje i s případem, že síť může skončit v cyklickém atraktoru.

input : *network*
output: *bestAttractor* - atraktor, který by potenciálně mohl reprezentovat faktor

1 **randomly activate** k_{in} neurons of $\mathbf{x}$
2 **for** $k \leftarrow k_{in}$ **to** k_{fin} **do**
3 | **set** *attractor* **to** $EvolveToAttractor(k, network)$
4 | **store** $\langle k, attractor \rangle$ **in** $Attractors$
5 $ComputeDerivativesOfR(Attractors)$
6 $ComputeSim(Attractors)$
7 **set** *bestAttractor* **to** $PossibleFactor(Attractors)$
8 **return** *bestAttractor*

Algorithm 7: Funkce $Trial(\mathbf{X})$

V algoritmu 7 nejprve náhodně vybereme a aktivujeme k_{in} neuronů (3. řádek algoritmu). Jinými slovy, náhodně vybrané složky vektoru $network.\mathbf{x}$ (vektor reprezentuje stavy neuronů sítě) nastavíme na hodnotu 1. Tedy vlastně náhodně vybereme nějakou podmnožinu atributů o velikosti k_{in}, tedy $A_{k_{in}} \subseteq A_{all}$.

Poté necháme síť pomocí funkce $EvolveToAttractor(k, network)$ ustálit - necháme ji přejít do atraktoru daného „stupně aktivity“, kapitola 6.6.3.. Stupněm

aktivity je zde myšlen počet aktivovaných neuronů v atraktoru, v algoritmu jde o proměnnou k, která postupně v průběhu cyklu (řádek 2.- 4.) nabývá hodnot k_{in} až k_{fin}. Jeden takový krok cyklu(řádek 4.- 6.) označují autoři v článku [6] jako „externí krok“.

Funkce po zavolání tedy vrátí atraktor, kde je aktivovaných právě k neuronů (5. řádek). Funkce obsahuje vedlejší efekt v tom smyslu, že změní stavy neuronů sítě. Vektor stavů neuronů sítě $network.\mathbf{x}$ je po vykonání funkce shodný s vektorem stavů neuronů v právě navráceném atraktoru($attractor.x$). Navrácený atraktor si následně spolu se stupněm aktivity uložíme do množiny *Attractors* (6. řádek).

S trochou nepřesnosti můžeme říci, že výše uvedenou množinu $A_{k_{in}}$ postupně „rozšiřujeme“ o další „vhodné“ atributy. Taková představa je nepřesná proto, že my nemáme zaručeno, že v následujícím kroku dostaneme nadmnožinu v předchozím kroku vybrané množiny atributů. Proto je termín „rozšíření“ v uvozovkách. Nicméně, jak uvidíme později (6.7.3.), je pro nalezení faktoru důležité, abychom množinu opravdu rozšiřovali. Dále jsem do uvozovek umístil i termín „vhodné“, a to z důvodu, že nám sice jde o postupné rozšiřování vhodnými atributy, ale pouze do určitého okamžiku - určité hodnoty k, kdy k by mělo zcela jistě být ostře menší než k_{fin}.

Pro další popis algoritmu se potřebujeme seznámit s pojmem Lyapunovovy funkce.

6.6.2. Lyapunovova funkce

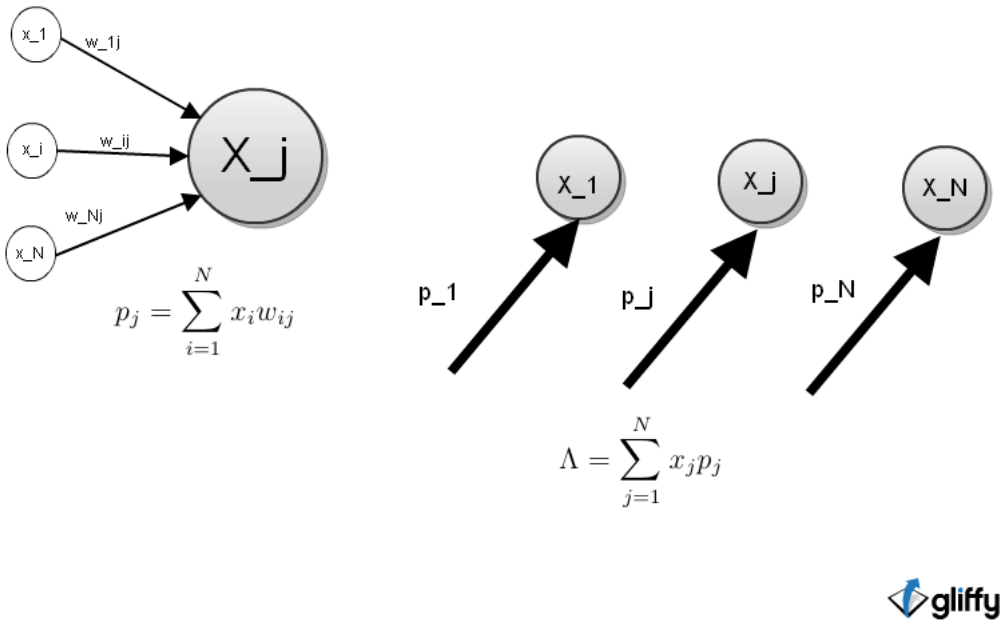
Zásadní roli v při hledání faktorů hraje tzv. *Lyapunovova funkce* a od ní dále odvozená *relativní Lyapunovova funkce*.

Lyapunovovy funkce jsou důležité zejména v oblasti teorie stability a teorie řízení [10]. Funkce tohoto typu se mimo jiné používají pro popis stavu nějakého složitého dynamického systému pomocí skalární hodnoty. Jak jsme si již řekli, na vybavovací proces Hopfieldovy sítě se můžeme dívat jako na dynamický systém a proto k popisu toho, v jakém stavu se právě vybavovací proces nachází, můžeme použít funkce tohoto typu. V případě použití Hopfieldovy sítě jako asociativní paměti bychom mohli zmíněnou energetickou funkci E označit jako Lyapunovovu [12]. V případě sítě ANNIA označíme jako Lyapunovovu funkci následující:

$$\Lambda(\mathbf{x}) = \mathbf{x}^T\mathbf{J}\mathbf{x}$$

kde $\mathbf{x}$ je stav sítě v bodovém atraktoru (definici Lyapunovovy funkce pro cyklický atraktor může čtenář opět nalézt v [6]. Na Lyapunovovu funkci se též můžeme dívat jako na sumu potenciálů aktivních neuronů, viz obr. 20.. Takto definovaná funkce potom monotonicky roste vzhledem k počtu aktivních neuronů (vektor $\mathbf{x}$).

Relativní Lyapunovovu funkci definujeme potom následovně:



Obrázek 20. Lyapunovova funkce jako suma potenciálů aktivních neuronů

$$\lambda(\mathbf{x}) = \frac{\Lambda(\mathbf{x})}{k}$$

kde k je počet aktivních neuronů. Na tuto funkci můžeme též hledět jako na průměrný potenciál aktivních neuronů. Atraktor, ve kterém má tato funkce maximum, by pak měl být hledaným faktorem. Optikou FCA pak můžeme na λ pohlížet jako na míru vyjadřující obsah obdélníku v matici vstupních dat($\mathbf{I}$), obdélníku definovaného množinou aktivovaných neuronů - vybraných atributů a objektů, jež tyto atributy sdílí.

6.6.3. Ustálení

Nyní přejděme k popisu procesu ustálení sítě ve stupni aktivity k, popsaném algoritmem 8. Pro ustálení sítě v atraktoru daného stupně aktivity musíme nejprve spočítat potenciály jednotlivých neuronů (v článku [6] označované jako synaptické excitace), toto se děje na 6. řádku, kde si jednotlivé potenciály uložíme do vektoru $\mathbf{h}$, algebraicky $\mathbf{h} = \mathbf{Jx}$. Na řádku 7. deaktivujeme všechny neurony a následně na 8. řádku vybereme k neuronů s nejvyšším potenciálem a ty aktivujeme. Aktivované neurony si uložíme do množiny *activate* na řádku 10. Tato množina, spolu s množinou *previouslyActivated* slouží ke zjištění ustálení. Tento proces (řádek 5. - 10.) opakujeme do té doby, dokud se množina vybraných neuronů neustálí, řádek 11.

input : číslo k, *network*
output: *atrractor* - atraktor daného stupně aktivity k

1 **set** J **to** *network*.J
2 **set** **x** **to** *network*.**x**
3 **set** *activated* **to** $\emptyset$
4 **set** *previouslyActivated* **to** $\emptyset$
5 **repeat**
6 **set** previouslyActivated **to** activated
7 **set** vector **h such that** $h_j = \Sigma_{i=1}^{N} J_{ij} x_i$
8 **set** **x** **to** **0**
9 **choose** k neurons with greatest h_j **and activate** them
10 **set** activated **to** $\{j \in \{1, \ldots, N\} \mid x_j = 1\}$
11 **until** *previouslyActived* = *activated*
12 **set** Λ **to** $\mathbf{x}^T\mathbf{h}$
13 **set** *attractor*.λ **to** $\frac{\Lambda}{k}$
14 **set** *attractor*.T **to** $min(\{h_i \mid x_i = 1\})$
15 **set** *attractor*.R **to** $R(k) = \frac{\lambda(k)}{(k-1)} - \frac{T(k)}{k}$
16 **return** *attractor*

Algorithm 8: Funkce $EvolveToAttractor(k, network)$

Poté, co se síť ustálila v nějakém atraktoru, si potřebujeme zaznamenat hodnotu relativní Lyapunovovy funkce a hodnotu prahu v daném atraktoru. Jako hodnota prahu je zvolen nejmenší z potenciálů vybraných neuronů. Nakonec si ještě uložíme hodnotu R, o které bude řeč později v kapitole 6.7.2..

Z povahy právě popsaného algoritmu - postupně vybíráme neurony s nejvyšším potenciálem, a z povahy Lyapunovy funkce(Λ) - suma potenciálů všech aktivních neuronů, můžeme o Λ říci, že po ustálení sítě v atraktoru (dokončení smyčky na řádku 4.-10.) je její hodnota maximální. Tato funkce má tedy lokální maxima v atraktorech sítě pro daný stupeň aktivity. Zbývá ještě zmínit, že autoři v článku [6] provedení řádků 5. - 9. nazývají interním krokem.

Právě popsaný algoritmus lze též vyjádřit algebraicky následovně:

$$x_i(t+1) = \Theta(h_i(t) - T(t)), \quad i = 1, \ldots, N$$
$$x_i(0) = x_i^{in}$$

kde $\mathbf{x}^{in}$ je stav neuronů na začátku výpočtu funkce $EvolveToAttractor(k, network)$, Θ je Heavisidova funkce (funkce pro hodnotu argumentu ≥ 0 vrátí 1, pro každou jinou hodnotu pak 0), $h_i(t)$ značí aktuální potenciál neuronu i, $T(t)$ je pak aktuální aktivační práh. Tento práh je vybrán tak, aby bylo aktivováno pravě k neuronů. Nakonec $x_i(t+1)$ pak značí

nový stav neuronu i. Jak lze z předchozího popisu vidět, dynamika procesu ustálení sítě ANNIA je *synchronní*. Synchronní proto, že potenciály všech neuronů jsou nejdříve spočteny a až potom jsou příslušné neurony aktivovány. Na rozdíl od Hopfieldovy sítě, kde byl neuron aktivován buďto ihned po spočtení jeho potenciálu, a nebo vůbec a kde se tudíž nečekalo na spočtení potenciálů ostatních neuronů.

6.7. Zjištění polohy možného faktoru

6.7.1. Průběh Lyapunovovy a prahové funkce

Vraťme se nyní k algoritmu funkce Trial 7. Po úplném provedení smyčky na řádku 2. máme v *Attractors* uloženo $k_{fin} - k_{in}$ atraktorů, což mimo jiné znamená, že máme k dispozici $k_{fin} - k_{in}$ hodnot relativní Lyapunovy funkce(λ), dále pak stejný počet hodnot prahu (T). Analýzou průběhu těchto hodnot se budeme snažit nalézt atraktor, který by mohl být faktorem.

Konkrétně autoři tvrdí, že v místě, kde by hledaný faktor měl ležet, bychom měli v případě $\lambda(k)$ zaznamenat jakýsi vrchol - globální maximum (autoři se o tomto vyjadřují jako o „kink“), v případě $T(k)$ bychom také měli zaznamenat globální maximum následované prudkým poklesem $T(k)$ (prudším, než v případě $\lambda(k)$). Průběh $\lambda(k)$ můžeme vidět na obr. 22., průběh $T(k)$ pak můžeme vidět na obr. 23..

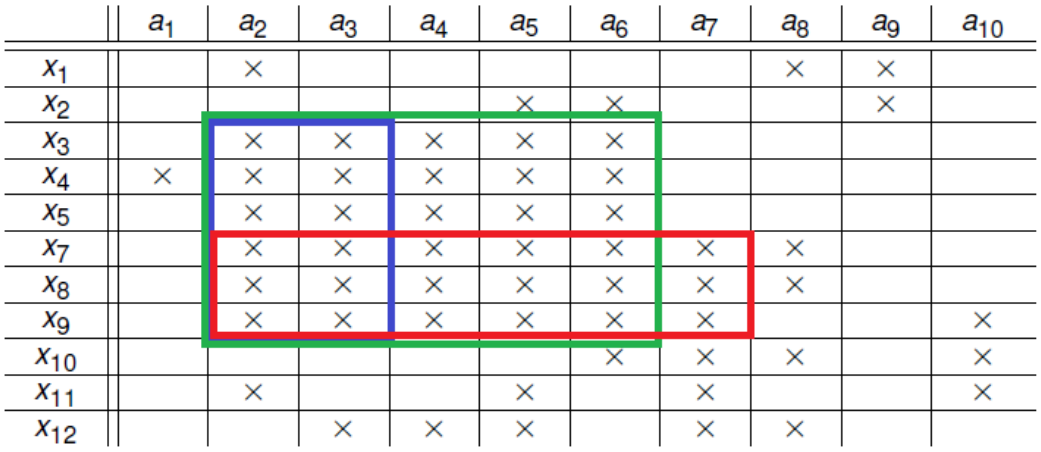
Nyní uvedu důvody, proč by tomu tak mělo být. Nejprve si vysvětleme důvod proč by hledaný faktor měl ležet na pozici globálního maxima $\lambda(k)$.

Předpokládejme, že $\mathbf{x}^{in}$ byl zvolen tak, že spadá do basinu nějakého atraktoru reprezentujícího faktor $\mathbf{x}^{f}$ (ležící na pozici k_f, v tomto odstavci dále jen faktor). Pro zjednodušení dále předpokládejme, že jsme aktivovali fragment výsledného faktoru. Postupnou aktivací dalších neuronů faktoru se průměrná hodnota potenciálu aktivovaných neuronů zvedá, a to díky algoritmu učení (sekce), jehož motivací bylo, aby neurony faktoru byly silně spojeny - aby váhy synaptických spojení byly vyšší než mezi neurony netvořící faktor.

Pokud po aktivaci všech neuronů faktoru aktivujeme ještě další neuron, jež do faktoru nepatří, měla by průměrná hodnota potenciálu všech v tuto chvíli aktivních neuronů klesnout, a to právě proto, že tento poslední aktivovaný neuron je s neurony faktoru spojen slabě. Pokles průměrné hodnoty potenciálu znamená tedy i pokles $\lambda(k)$, a tedy již zmíněnou presenci globálního maxima $\lambda(k)$ v bodě k_f.

Přítomnost globálního maxima $T(k)$ v k_f a následného prudkého poklesu $T(k)$ je z podobného důvodu jako je tomu v případě $\lambda(k)$. Vycházíme z toho, jak v každém atraktoru na pozici k volíme hodnotu T. Protože hodnotu T volíme jako nejmenší potenciál aktivních neuronů, i tato hodnota při postupné aktivaci neuronů faktoru roste. Potom, co zvolíme hodnotu prahu pro atraktor na pozici

k_{f+1}, tedy přidáme neuron, jež do faktoru nepatří (tento je tedy opět s neurony faktoru spojen jen slabě), jeho potenciál je tedy oproti potenciálu silně spojených neuronů faktoru výrazně nižší, musí i hodnota $T(k)$ výrazně klesnout.



Obrázek 21. Tabulka s obdélníky

Příklad Jako příklad nyní analyzujme situaci na obrázku 21.. Vidíme zde faktor $A^{\mathbf{x}^f} = \{a_2, a_3, a_4, a_5, a_6\}$ vyznačený zeleně. Nyní předpokládejme, že vybavovací proces zahájíme v atraktoru $\mathbf{x}^{in}$ s aktivovanými atributy $A^{\mathbf{x}^{in}} = \{a_2, a_3\}$, tento startovní atraktor je na obrázku vyznačen modře. Postupnou aktivací dalších neuronů faktoru $\mathbf{x}^f$ rozšiřujeme jím indukovanou množinu atributů (postupně o atributy a_4, a_5, a_6). Můžeme vidět, že tyto atributy sdílí poměrně mnoho objektů, což nás přesvědčuje o tom, že neurony korespondující s těmito atributy jsou opravdu silně spojeny. Po aktivaci posledního neuronu faktoru (a_6), potom aktivujeme neuron, řekněme a_7, abychom následně zjistili, že jeho korespondující atribut nesdílí s atributy faktoru mnoho objektů. Je s nimi tedy díky motivaci pro algoritmus učení spojen slabě. Po aktivaci tohoto neuronu klesne hodnota jak $\lambda(k)$, tak $T(k)$. Vidíme také, že obsah obdélníku (na obrázku vyznačeného červeně) definovaného atributy $\{a_2, a_3, a_4, a_5, a_6, a_7\}$ a jejich společnými objekty je nižší, než obsah obdélníku korespondujícího s faktorem $\mathbf{x}^f$.

Jak jsem již řekl, na $\lambda(k)$ se můžeme též dívat jako na vyjádření obsahu obdélníku korespondujícího s právě aktivovanými neurony atraktoru $\mathbf{x}(k)$, tedy obdélníku nacházejícího se v matici vstupních dat a korespondujícího s množinou vybraných atributů a objektů, jež tyto atributy sdílí. V příkladu pak vidíme, že se obsah takového obdélníku postupnou aktivací neuronů faktoru z zvětšuje, v momentě kdy jsou všechny neurony faktoru z aktivovány, je největší a při aktivaci

dalšího neuronu, jenž do faktoru nepatří (např. a_7), pak obsah korespondujícího obdélníku klesne.

6.7.2. Výpočet R'

Pro zkoumání polohy faktoru by se nám hodilo výše uvedené průběhy funkcí $\lambda(k)$ a $T(k)$ nějak agregovat do jedné funkce $R(k)$. Tohoto autoři docílili následovně:

$$R(k) = \frac{\lambda(k)}{(k-1)} - \frac{T(k)}{k}$$

Pro ještě přesnější identifikaci polohy faktoru budeme analyzovat derivaci $R(k)$:

$$R'(k) = R(k) - R(k-1),\ kde\ k \in \{k_{in}+1, \ldots, k_{fin}\}$$
$$R'(k_{in}) = 0.$$

Zajímá nás tedy místo největší změny v $R(k)$. Derivaci $R(k)$ počítá v algoritmu7 funkce $ComputeDerivativesOfR(Attractors)$. Implementaci této funkce vidíme v algoritmu 9.

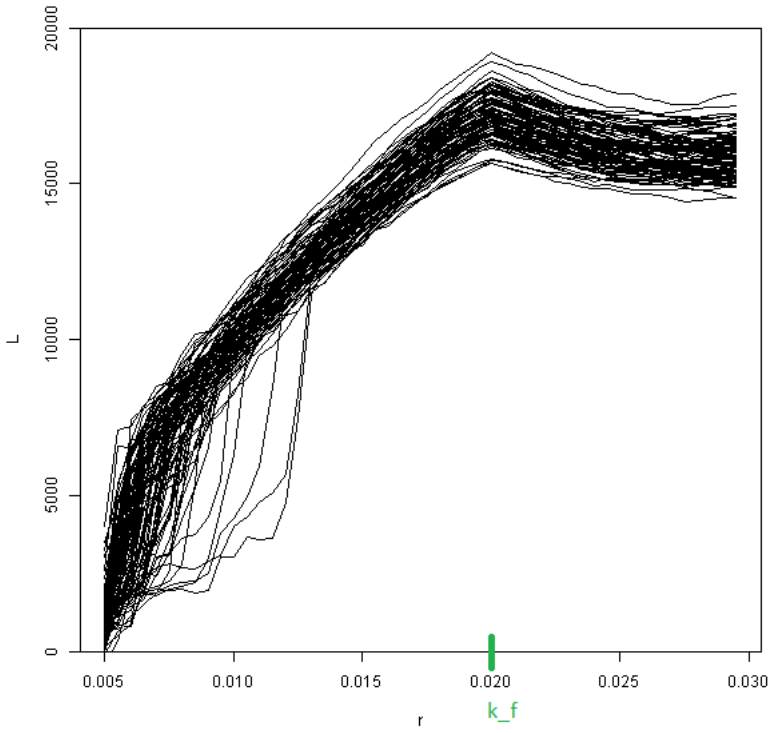
input : $Attractors$
output: $Attractors$, kde každý atraktor má vyplněnu položku R'

```
1 Attractors[k_in].R' = 0
2 for k ← k_in + 1 to k_fin do
3 |   Attractors[k].R' = Attractors[k].R − Attractors[k − 1].R
4 return Attractors
```

Algorithm 9: $ComputeDerivativesOfR(Attractors)$

Na obrázku 22. můžeme vidět průběh funkce $\lambda(k)$, dále na obrázku 23. průběh $T(k)$, a konečně na obrázku 24. můžeme vidět průběh funkce $R(k)'$. Na všech těchto obrazcích je poloha hledaného faktoru vyznačena jako k_f. Obrázky pochází z 2. experimentu, viz kapitola 6.11..

Funkce $R(k)'$ by potom měla nabývat svého globálního maxima v bodě $(k_f + 1)$, v místě největší změny $R(k)$. Na obrázku 24. lze však vidět, že tomu tak bohužel není. Důvod pro to je takový, že se občas může stát, hlavně u hodnot blízkých k_{in}, že se aktivita sítě přesune do atraktoru $\mathbf{x}^{attr}(k)$ hodně vzdáleného od atraktoru z předchozího kroku $\mathbf{x}^{attr}(k-1)$. Situace popsaná v předchozí větě je v podstatě taková, že množina aktivních neuronů v $\mathbf{x}^{attr}(k)$ se výrazným způsobem změnila, oproti množině aktivních neuronů v $\mathbf{x}^{attr}(k-1)$. Takovýto přesun aktivity - *skok* potom může též vytvořit vrchol, dokonce globální maximum, funkce $R'(k)$.



Obrázek 22. Průběh $\lambda(k)$

6.7.3. Podobnostní kritérium

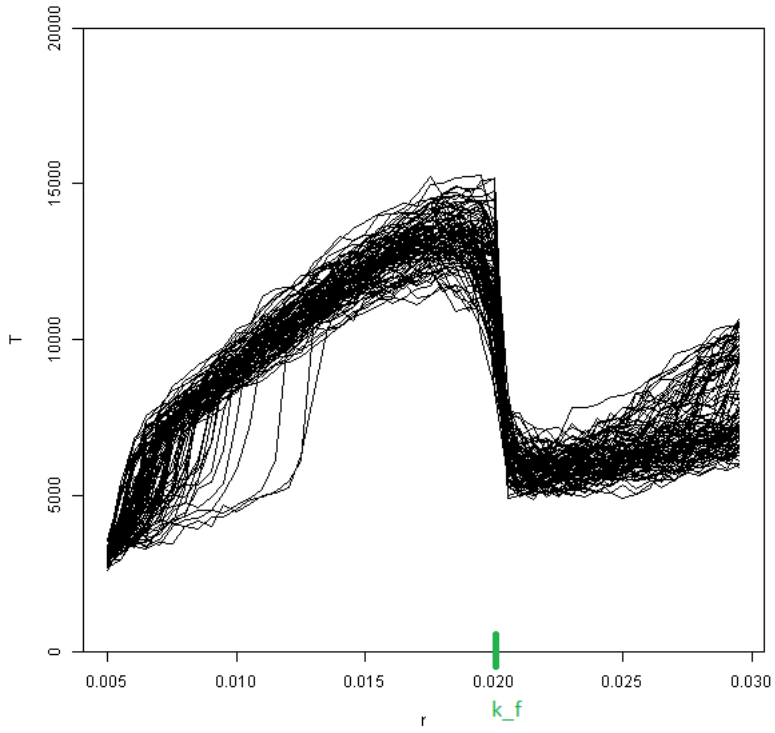
K tomu, abychom podle vrcholů, vniknuvších díky velkým přesunům aktivity v síti tak, jak je popsáno v samotném závěru předchozí kapitoly, chybně neurčili polohu hledaného faktoru, vymysleli autoři kritérium podobnosti $Sim(k)$ mezi dvěma sousedními atraktory následovně:

$$Sim(k) = \frac{a - (k-1)k/N}{(k-1)(1-k/N)}$$

kde a je počet společně aktivovaných neuronů (počet společných jedniček) atraktoru $X^{attr}(k)$ a $X^{attr}(k-1)$[8].

Pokud je množina aktivních neuronů v $\mathbf{x}^{attr}(k)$ nadmnožinou množiny aktivních neuronů v $\mathbf{x}^{attr}(k-1)$, pak $Sim(k) = 1$. Pokud jsou $\mathbf{x}^{attr}(k)$ a $\mathbf{x}^{attr}(k-1)$ nezávislé, pak se $Sim(k)$ rovná v průměru nule[8].

Autoři usoudili, že vrchol $R'(k)$ vznikl skokem do vzdáleného atraktoru, pokud kritérium $Sim(k) < Sim_{thr}$, kde $Sim_{thr} = 0.8$. A tedy že globální maximum $R'(k)$ určuje pozici potenciálního faktoru pouze, pokud jsme se do korespondujícího atraktoru $\mathbf{x}^{attr}(k)$ nedostali takovýmto skokem. Abychom toto mohli správně



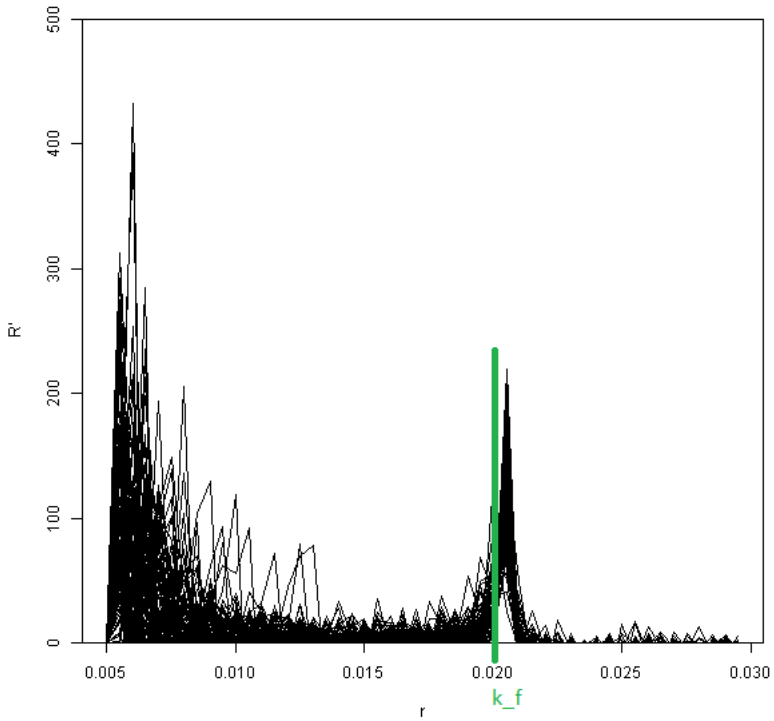
Obrázek 23. Průběh $T(k)$

ověřit, musíme si v každém atraktoru zapamatovat podobnost s atraktorem předchozím. Toto v algoritmu 7 zajišťuje funkce $ComputeSim(Attractors)$ na řádku 6. Implementaci takové funkce můžeme vidět v algoritmu 10.

Nyní konečně můžeme popsat proces zjištění polohy faktoru, tedy uvést implementaci funkce $PossibleFactor(Attractors)$, algoritmus 11. Výběr atraktoru, jež je potenciálním faktorem, probíhá následovně: nejprve vybereme pouze ty atraktory, jež splňují kritérium podobnosti, tedy že $attractor.Sim \geq Sim_{thr}$, v algoritmu množina $nonjumpAttractors$. Ponechme teď stranou extrémní případ, že tato množina může být potenciálně prázdná. Z těchto atraktorů pak jako potenciální faktor označíme ten, jehož hodnota R' je maximální ze všech atraktorů $nonjumpAttractors$. Tento atraktor je tedy výsledkem funkce $PossibleFactor(Attractors)$.

6.8. Ověření možného faktoru

Nyní jsme tedy zjistili polohu atraktoru, který by mohl být faktorem. Přistupme nyní k tomu, jak zjistit, jestli tento atraktor opravdu je faktorem či není. Lépe řečeno, uveďme si kritérium podle kterého budeme atraktor považovat za



Obrázek 24. Průběh R')

faktor. Čtenář by se nyní mohl podivit nad tím, proč vůbec takové kritérium zavádět, vždyť nemělo by snad každé volání funkce $Trial(network)$ skončit úspěšným nalezením faktoru? Bohužel se může stát, že síť se ve vybavovacím procesu dostane ne pouze k *pravým atraktorům*, jež jsou faktory (ve článcích referovaných jako „true attractors“, ale také k *atraktorům falešným* (ve článcích referovaným jako „spurious attractors“), jež nejsou faktory a jsou vzdáleny od všech pravých atraktorů. Z pohledu atraktor basinů můžeme říct, že síť se dostane do nějakého pravého atraktoru pouze tehdy, pokud její počáteční stav (výše reprezentovaný vektorem $\mathbf{x}_{in}$) spadne do atraktorového basinu některého z pravých atraktorů, v opačném případě se síť dostane do některého z atraktorů falešných. Naštěstí jsou ale takové falešné atraktory (v optimálním případě, jak uvidíme později) snadno rozlišitelné od pravých atraktorů. Hodnota Lyapunovovy funkce (Λ) falešných atraktorů je totiž mnohem nižší, než atraktorů pravých - faktorů.

To, že je Λ pravých atraktorů je zřejmé z definice toho, co považujeme za faktor (množinu silně spojených k neuronů) a definice Lyapunovovy funkce (suma potenciálů aktivních neuronů). Faktory tedy musí mít vyšší hodnotu Lyapunovovy funkce než nějaká náhodně vybraná množina k neuronů, jež bude velice pravděpodobně spojena slabě. Náhodně vybraná množina k neuronů by potom

input : $Attractors$
output: $Attractors$, kde každý atraktor má vyplněnu položku S'

1 $Attractors[k_{in}].Sim = 0$
2 **for** $k \leftarrow k_{in} + 1$ **to** k_{fin} **do**
3 | **set** a **to** number of common ones in $Attractors[k].\mathbf{x}$ and $Attractors[k-1].\mathbf{x}$
4 | $Attractors[k].Sim = \frac{a-(k-1)k/N}{(k-1)(1-k/N)}$
5 **return** $Attractors$

Algorithm 10: $ComputeSim(Attractors)$

input : $Attractors$
output: $possibleFactor$

1 $Attractors[k_{in}].Sim = 0$
2 **set** $nonjumpAttractors$ **to** $\{attractor \in Attractors \mid attractor.Sim \geq Sim_{thr}\}$
3 **set** $possibleFactor$ **to** $attractor \in nonjumpAttractors$ **with maximal** R' **among** $nonjumpAttractors$
4 **return** $possibleFactor$

Algorithm 11: $PossibleFactor(Attractors)$

měla být spojena slabě proto, že uvažujeme pouze faktory kódované řídce 6.1.1. a tedy je velice nepravděpodobné, že by se nám při náhodné aktivaci několika neuronů povedlo „trefit“ do nějakého faktoru. Pokud se síť dostala do falešného atraktoru, stalo se tak nějakým nedopatřením, jde v podstatě o nechtěný stav a množina neuronů, jež jsou v takovém stavu aktivní, byla vybrána z našeho pohledu náhodně.

Tato úvaha vedla autory k navržení kritéria pro rozlišení pravých faktorů od falešných. Pokud jsme tedy zjistili lokaci nějakého atraktoru ve stupni aktivity k, potenciálně pravého, zjistíme, jak si vede jeho hodnota Λ vzhledem k průměrné hodnotě Λ u zcela jistě falešných atraktorů ve stupni aktivity k, tedy vlastně náhodně aktivované množině k neuronů.

Rozlišení pravých atraktorů od falešných zajišťuje v algoritmu 5 funkce $FactorP(attractor, network)$. Její implementaci je možno vidět v algoritmu 12.

Vstupem algoritmu 5 je tedy atraktor s k aktivními neurony. Algoritmus začíná tím, že v síti náhodně aktivujeme k neuronů, poté necháme síť ustálit ve stupni aktivity k a zjistíme hodnotu $\Lambda(k)$, tuto hodnotu si zaznamenáme do množiny *spuriousLambdas* (řádek 4. - 6.). Tento proces 100× zopakujeme. Množina *spuriousLambdas* tedy nyní obsahuje 100 hodnot Λ, jež lze považovat za hodnoty Λ falešných atraktorů ve stupni aktivity k, viz úvaha výše. Nyní spočítáme aritmetický průměr a standardní odchylku těchto hodnot a obojí agregujeme do

input : *attractor* který je potenciálním faktorem, *network*
output: $true \times false$ hodnota určující, zda je atraktor faktorem či nikoli

```
1  set spuriousLambdas to ∅
2  set k to number of ones in attractor.x
3  for i ← 1 to 100 do
4  |  randomly activate k neurons of network.x
5  |  set spuriousAttractor to EvolveToAttractor(k, network)
6  |  store spuriousAttractor.Λ in spuriousLambdas
7  set mean to Mean(spuriousLambdas)
8  set standardDeviation to σ(spuriousLambdas)
9  set h_max to mean + 2 * standardDeviation
10 if attractor.Λ > h_max then
11 |  return true
12 else
13 |  return false
```

Algorithm 12: $FactorP(attractor, network)$

hodnoty h_{max} tak, jak je uvedeno na řádku 9. Pokud Λ vstupního atraktoru je vyšší než h_{max} pak o tomto atraktoru řekneme, že je pravý, v opačném případě řekneme, že je falešný. Tedy vlastně prohlásíme vstupní atraktor za pravý právě tehdy, když je jeho Λ mnohem vyšší než průměrná hodnota Λ falešných atraktorů.

6.9. Odučení nalezeného faktoru

Vybavovací proces má sklon největší faktory (faktory, jež korespondují k největším obdélníkům v matici vstupních dat) nacházet nejdříve. To z toho důvodu, že atraktorové basiny takových faktorů (přesněji řečeno pravých atraktorů) jsou největší a tedy šance, že při náhodné volbě inicializačního stavu, vektoru $\mathbf{x}^{in}$, tento stav spadne do nějakého z takových basinů, je největší. Též Lyapunova funkce takových atraktorů bude největší. Pro to, aby síť neustále nevracela jen několik nejsilnějších faktorů, je nutné tyto faktory ze sítě tzv. odučit, tedy smazat.

Autoři metody takovéto smazání faktoru $\mathbf{x}$ provádí pomocí odečtení ΔJ_{ij} od každého synapse J_{ij}, kde

$$\Delta J_{ij} = \bar{J}[(x_i - r)(x_j - r)], \quad j \neq i, \quad DeltaJ_{ii} = 0$$

, kde $\bar{J} = \frac{\lambda}{rN}$ je průměrná váha synapse mezi neurony faktoru a kde $r = \frac{k_f}{N}$ je opět (tentokrát ale trochu jiné) vyjádření stupně aktivity v daném faktoru. Dané pravidlo platí opět pouze pro bodový atraktor, pravidlo pro cyklický atraktor nalezne čtenář v [6]. Implementaci funkce $UnlearnAttractor(network, attractor)$ vidíme v algoritmu 13.

Zde popsané odučení by se dalo připodobnit k odmazání obdélníku korespondujícího s formálním konceptem z matice **I** v BFAFKA.

input : $attractor$, $network$
output:

set $\mathbf{x}$ **to** $attractor.\mathbf{x}$
set k_f **to** number of ones in $\mathbf{x}$
set r **to** $\frac{k_f}{N}$
set $\bar{J}$ **to** $\frac{attractor.\lambda}{rN}$
$\Delta J_{ij} = \bar{J}[(x_i - r)(x_j - r)], \quad j \neq i, \quad DeltaJ_{ii} = 0$
$J_{ij} = J_{ij} - \Delta J_{ij}$

Algorithm 13: Funkce $UnlearnAttractor(network, attractor)$

6.10. Zjištění signálů obsahujících faktor

Zjištění signálů obsahujících nalezený faktor vyřešili autoři zavedením podobnostní Yulovy Q míry mezi původním signálem $\mathbf{x}$ a nalezeným faktorem $\mathbf{x}^f$ [7]. Míru Q zavedli následovně:

$$Q(\mathbf{x}, \mathbf{x}^f) = (ad - bc)/(ad + bc)$$

kde

- a je počet společných jedniček mezi $\mathbf{x}$ a $\mathbf{x}^f$,
- b je počet jedniček ve faktoru $\mathbf{x}^f$ koincidujících s nulami v signálu $\mathbf{x}$,
- c je počet jedniček v signálu $\mathbf{x}$ koincidujících s nulami ve faktoru $\mathbf{x}^f$,
- d je počet společných nul mezi $\mathbf{x}$ a $\mathbf{x}^f$.

Pokud signál $\mathbf{x}$ obsahuje daný faktor $\mathbf{x}^f$ úplně (množina atributů indukovaná $\mathbf{x}^f$ je podmnožinou množiny indukované $\mathbf{x}$), pak $b = 0$ a $Q(\mathbf{x}, \mathbf{x}^f) = 1$. Nicméně autoři považují faktor $\mathbf{x}^f$ za součást $\mathbf{x}$ už v případě, kdy $Q(\mathbf{x}, \mathbf{x}^f) > Q_{thr}$, kde $Q_{thr} = 0.5$. Toto mimo jiné znamená, že tato metoda nemusí poskytnout přesný rozklad v tom smyslu, že nemusí platit $\mathbf{X} = \mathbf{SF}$. Zjištění, které signály obsahují nalezený faktor, zajišťuje v algoritmu 5 funkce $SignalsContainingFactor(\mathbf{X}, \mathbf{x}^f)$ 14.

V článku [8] autoři použili jiné kritérium, které je nicméně v konečném důsledku zde uvedenému kritériu velice podobné.

input : matice $\mathbf{X}$, faktor $\mathbf{x}^f$
output: vektor $\mathbf{z} \in \{0,1\}^M$ značící, které signály obsahují $\mathbf{x}^f$

```
set z to 0
for i ← 0 to M do
    set x^m to X_{m_}
    if Q(x^m, x^f) > Q_thr then
        set z_m to 1
    else
        set z_m to 0
return z
```

Algorithm 14: Funkce $SignalsContainingFactor(\mathbf{X}, \mathbf{x}^f)$

6.11. Experimenty

V této kapitole bych chtěl popsat několik experimentů, které jsem s metodou provedl. V každém z prvních třech experimentů jsem nejprve podle nějakého pravidla nechal vygenerovat množinu o L faktorech, které jsem pak vždy podle nějakého jiného pravidla namixoval do jednotlivých signálů a vytvořil tím tak množinu o M vstupních signálech. Poté jsem v prvních třech experimentech analyzoval průběh relativní Lyapunovovy funkce, průběh prahové funkce a též funkce R' (definované v kapitole 6.7.2.) pro každý pokus o vybavení faktoru (volání funkce $Trial(\mathbf{X})$ v algoritmu 5). Protože, jak jsem popsal výše, byla množina faktorů apriori známá, mohl jsem porovnat (a na obrázcích barevně rozlišit) průběhy těchto funkcí pro pokusy, které skončily nalezením faktoru a pro pokusy, jež nalezením faktoru neskončily (dále již jen úspěšný × neúspěšný pokus). Toto platí zejména pro experiment 6.11.1.. V experimentu 6.11.2. pak uvedu, jak byla výše zmíněná kritéria pro zjištění polohy možného faktoru a pro rozlišení pravých a falešných atraktorů úspěšná. V experimentu 6.11.3. se zaměřím na pravidlo odučení nalezených faktorů. Na závěr popíši, jak dopadl experiment na reálných datech `mushrooms` [3] (experiment 6.11.4.).

Dále bych rád uvedl poznámku k výše zmíněné apriori znalosti faktorů. Obecně nemůžeme mnoho říci o optimalitě rozkladu matice vstupních signálů pomocí takových faktorů. Po zmíněném namixování se totiž může stát, že vzniknou nové zajímavé shluky dat, které potenciálně vysvětlují data lépe, než naše apriori zvolená množina faktorů. Toto je však, vzhledem k povaze faktorů a pravidel pro mixování zmíněných dále, velice nepravděpodobné. Dopustíme se tedy v následujícím textu zjednodušení a budeme předpokládat, že množinu faktorů opravdu apriori známe.

Než přejdeme k samotným experimentům musím uvést ještě druhou poznámku, která se týká značení osy x na grafech zobrazujících průběhy funkcí $(\lambda(k), T(k), R'(k))$ u jednotlivých experimentů. Osa x na nich totiž nabývá hodnot r namísto doposud používaného k, kde vztah mezi nimi je $k = rN$. Tedy r je

poměr mezi aktivními neurony sítě a celkového počtu neuronů v síti. Průběhy jednotlivých funkcí budu tedy v této kapitole značit postupně jako $\lambda(r), T(r), R'(r)$.

Vygenerované faktory v prvních třech experimentech byly dále uniformně rozloženy v

$$B_n^N = \{\mathbf{X} | X_i \in \{0,1\}^N, \Sigma_{i=1}^N X_i = n\}.$$

Vybavovací proces každého z prvních tří experimentů potom začal v bodě $r_{in} = 0.005$ (tedy s rN aktivovanými neurony) a skončil v bodě $r_{fin} = 0.03$.

6.11.1. Experiment 1

Tento experiment měl za cíl potvrdit vztah mezi průběhy jednotlivých funkcí $(\lambda(r), T(r), R'(r))$ pokusů, jež skončily nalezením faktoru a pokusů jež neskončily nalezením faktoru (dále jen úspěšný $\times$ neúspěšný pokus). Experiment je inspirován experimentem provedeného autory v článku [6].

Dimenze prostoru faktorů byla $N = 3000$, každý faktor dále obsahoval $n = pN$ jedniček a $(1-p)N$ nul, kde $p = 0.02$. Počet faktorů byl potom $L = 2100$. V tomto příkladu byl každý vstupní signál zároveň jedním z faktorů. Průběhy (trajektorie) funkcí $(\lambda(r), T(r), R'(r))$ u úspěšných pokusů jsou na jednotlivých obrázcích vyznačeny černě, průběhy funkcí u neúspěšných pak červeně.

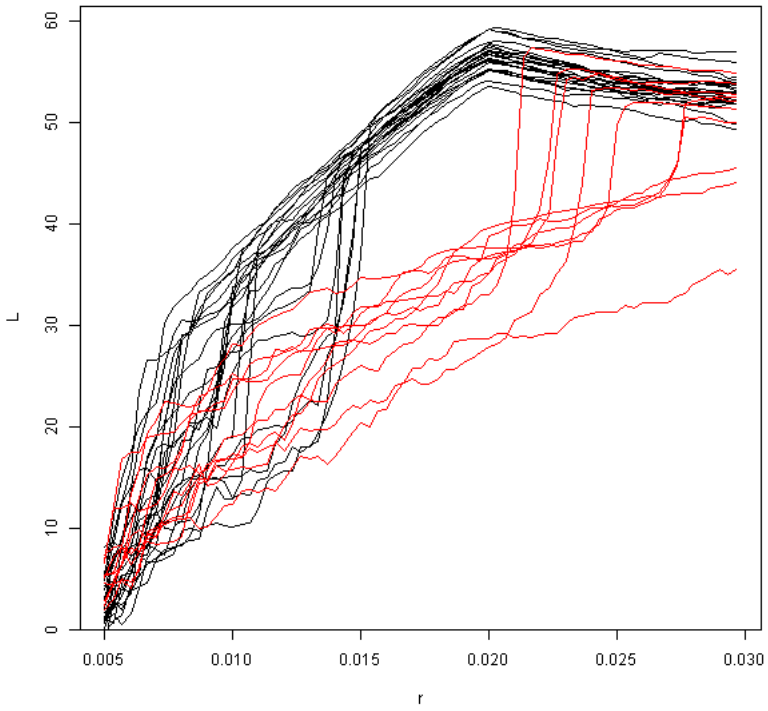
Na obr. 25. je vyobrazen průběh $\lambda(r)$ funkce. Je zde jasně vidět, že v bodě $p = 0.02$ (pozice faktorů) nabývá hodnota funkce $\lambda(r)$, v případě úspěšných pokusů, opravdu globálního maxima, zatímco u neúspěšných faktorů tomu tak není. Dále si můžeme povšimnout, že hodnota funkce $\lambda(r)$ je v bodě $p = 0.02$ v případě úspěšných pokusů mnohem vyšší než v případě neúspěšných pokusů. Toto odpovídá výše popsanému 6.8. a tedy, že pravé atraktory - faktory, mají vyšší hodnotu $\lambda(r)$ než falešné atraktory.

Na obr. 26. je pak vyobrazen průběh $T(r)$ funkce. V bodě $p = 0.02$ u úspěšných pokusů opět vidíme globální maximum nyní následované prudkým poklesem, zatímco u neúspěšných pokusů nic takového nepozorujeme.

Konečně na obr. 27. vidíme průběh $R'(r)$ funkce, kde si znovu můžeme povšimnout globálního maxima v bodě $p = 0.02$ pro úspěšné pokusy.

6.11.2. Experiment 2

Další z experimentů měl za cíl zjistit, zda se průběhy funkcí $\lambda(k), T(k), R'(k)$ chovají podle očekávání i potom, co do jednoho signálů namixujeme více jak jeden faktor. Navíc, a to je důležitější, měl experiment prozradit i to, jak účinná jsou kritéria pro detekci pozice možného faktoru a dále, jak úspěšné je kritérium pro rozhodnutí, zda se opravdu jedná o faktor či nikoli. Experiment je inspirován jedním z experimentů provedeného autory v článku [7]. Nejprve však úvod do situace. Dimenze prostoru faktorů byla tentokrát $N = 2000$, každý faktor opět obsahoval $n = pN$ jedniček a $(1-p)N$ nul, kde $p = 0.02$. Počet faktorů byl



Obrázek 25. Průběh $\lambda(r)$, experiment 1

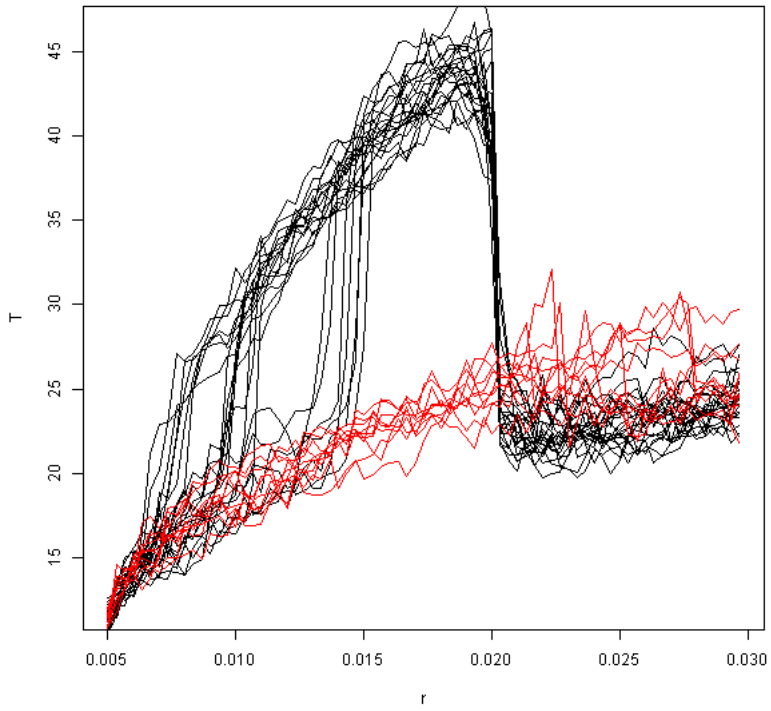
potom $L = 1000$. Vstupních signálů bylo 50000 a do každého vstupního signálu bylo namixováno 20 faktorů.

Z přiložených obrázků průběhu funkcí $\lambda(r), T(r), R'(r)$ (obr. 28., 29., 30.). vidíme, že se průběhy chovají opravdu podle očekávání.

Co se týče úspěšnosti zmíněných kritérií, tak v 81 ze 100 pokusů se podle, v algoritmu popsaných, kritérií podařilo správně najít atraktor v bodě $r = 0.02$. Ve všech 81 případech byl pak atraktor správně identifikován jako faktor pomocí kritéria popsaného v 6.8.. Čtenář by se nyní mohl podivit nad tím, jak je možná tak vysoká úspěšnost detekce správné pozice, když globální maximum funkce $R'(r)$ je v drtivé většině provedených pokusů někde kolem bodu $r = 0.006$. Možné to je právě díky dodatečnému kritériu podobnosti 6.7.3., kdy díky chaotickému přesunu aktivity na začátku vybavovacího procesu a tedy následném nesplnění takového kritéria, nejsou tyto globální maxima brána v potaz.

6.11.3. Experiment 3

Poslední z řady experimentů nad uměle vytvořenými daty měl za cíl ověřit, jak účinné je pravidlo odučení nalezeného faktoru 6.9.. Experiment je opět inspirován



Obrázek 26. Průběh $T(r)$, experiment 1

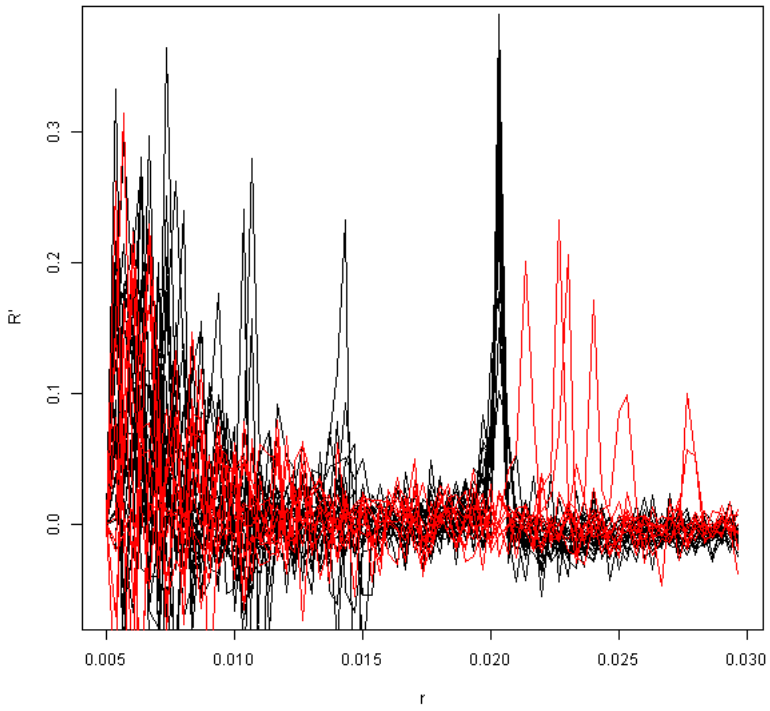
jedním z experimentů provedeného autory v článku [7].

Dimenze prostoru faktorů byla znovu $N = 2000$, každý faktor v sobě ale nyní obsahoval $n = p_i N$ jedniček a $(1 - p_i)N$ nul, kde $p_i \in \{0.01, 0.015, 0.02$ je vybráno náhodně (rozložení p_i je opět uniformní). Počet faktorů byl potom $L = 1000$. Vstupních signálů bylo 50000 a do každého vstupního signálu bylo namixováno 20 faktorů.

Na obrázku 31. jsou černě vyznačeny průběhy funkce $\lambda(r)$ faktorů na pozici 0.02, tedy faktory s $0.02N$ jedniček. Zeleně jsou potom vyznačeny průběhy stejné funkce pro faktory na pozici 0.015. Jak je vidět tak $\lambda(r)$ pro faktory na pozici 0.015 je nižší než pro faktory na pozici 0.02.

Po tom, co ze sítě odučíme všechny faktory stupně aktivity $0.02N$, začne síť postupně objevovat faktory stupně aktivity 0.015. Na obrázku 32. jsou tentokrát zobrazeny černě faktory stupně aktivity 0.015 a zeleně faktory stupně aktivity 0.01.

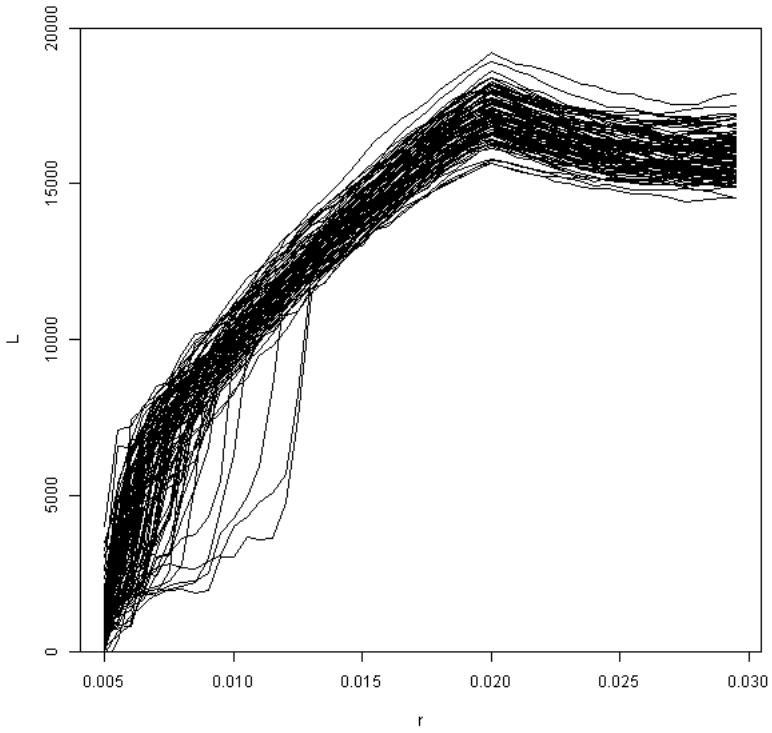
Po odučení všech faktorů stupně aktivity $0.015N$, začne síť postupně objevovat faktory stupně aktivity 0.01. Situace je vyobrazena na obr. 33.. Jak je vidět, pravidlo pro odučení faktorů funguje spolehlivě.



Obrázek 27. Průběh R', experiment 1

6.11.4. Experiment 4

Poslední experiment měl za cíl otestovat metodu na reálných datech `mushrooms` [3]. Bohužel metoda na těchto datech selhává. Selhává v tom smyslu, že není běžně schopná, podle kritérií pro nalezení pozici možného faktoru 6.7., nalézt pozici atraktoru s počtem aktivovaných neuronů větších jak 2. Dále pak oprávněně selhává i kritérium pro ověření, zda je nalezený atraktor opravdu faktorem 6.8.. Toto chování připisuji tomu, že uvedená data jsou pro metodu „malá“, nesplňující omezení zmíněná v kapitole 6.1.1.. Data totiž obsahují pouze 119 atributů, což je zhruba 20× méně, než počet atributů v datech, na nichž testovali metodu autoři v článcích ([6], [7], [8]). Basiny jednotlivých atraktorů mají pak mezi sebou pravděpodobně velké překryvy, což způsobuje chaotický proces vybavování. Dále kritérium uvedené v kapitole 6.8. selhává z toho důvodu, že rozdíl Lyapunovovy funkce mezi vráceným atraktorem ve stupni aktivity k a náhodně aktivovanou množinou k neuronů je zanedbatelný a to opět z důvodu nízké dimensionality dat a z ní částečně plynoucí vysoké „hustoty“ dat (průměrný počet jedniček v jednom signálu).



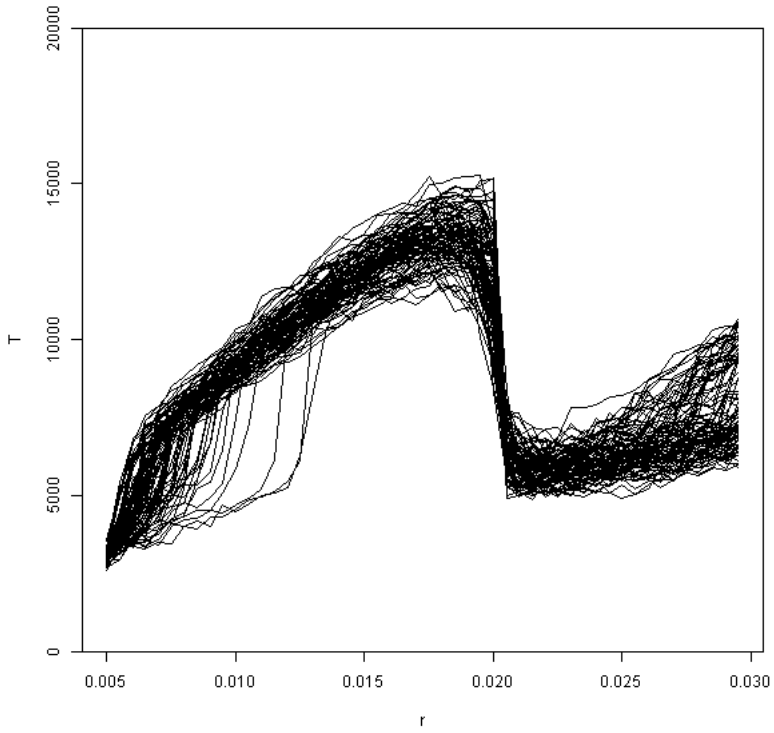
Obrázek 28. Průběh $\lambda(r)$, experiment 2

Závěr

Cílem této práce bylo popsat a experimentálně ověřit dvě metody faktorizace binárních dat. Metodu faktorizace pomocí formálních konceptů a metodu faktorizace pomocí atraktorové neuronové sít, přičemž se detailněji zaměřit hlavně na druhou ze jmenovaných.

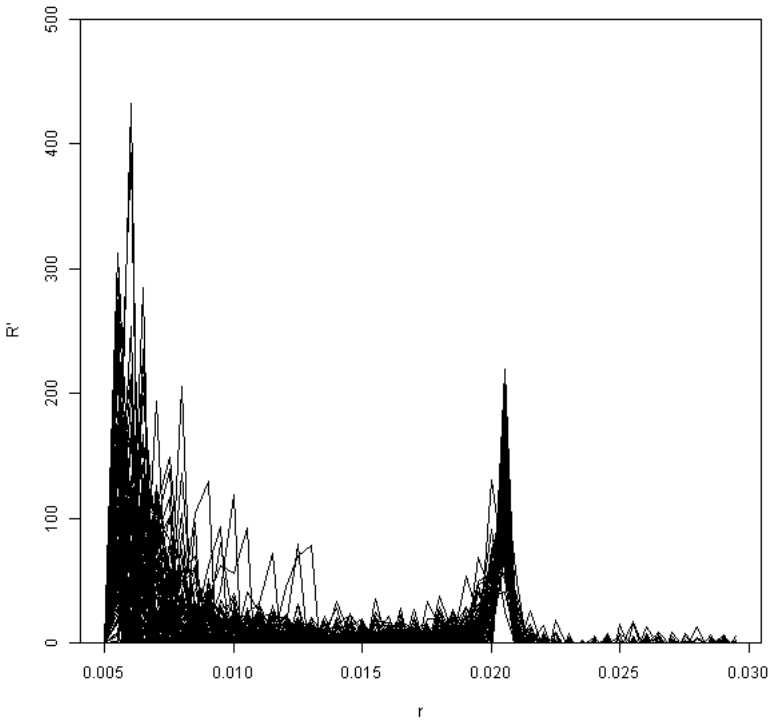
Metody jsem ověřil na reálných datech `mushrooms` získaných z [3]. Zatímco metoda faktorizace pomocí formálních konceptů faktorizuje data bez problému v přiměřeném čase, metoda faktorizace pomocí neuronové sítě na těchto datech selhává. Toto selhání je způsobeno hlavně tím, že tato metoda je stavěna na data o dimenzi v řádech tisíců, ne jedné stovky, jako tomu bylo v případě `mushrooms` dat. Odpovídající reálná binární data takto vysoké dimenze se mi bohužel nepodařilo získat. Na uměle vytvořených datech o vysoké dimenzi, takových, jakých použili sami autoři v uvedených článcích ([6],[7],[8]), si potom „neuronová“ metoda vede dobře.

V průběhu popisu metody faktorizace dat pomocí neuronové sítě jsem se dále snažil uvést hypotézy o tom, jak které její části souvisí s metodou faktorizace



Obrázek 29. Průběh $T(r)$, experiment 2

dat pomocí formálních konceptů. Další výzkum by mohl být věnován formálnímu ověření těchto hypotéz a případnému nalezení dalších vztahů mezi těmito metodami.



Obrázek 30. Průběh R', experiment 2

Reference

[1] Radim Bělohlávek *Introduction to Formal Concept Analysis* Olomouc, 2008

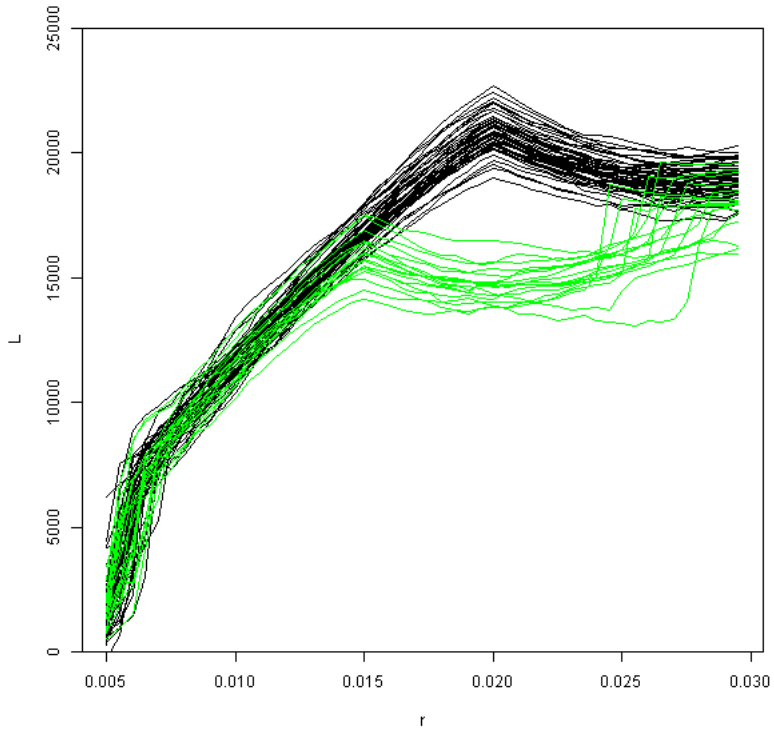
[2] Radim Bělohlávek, Vilém Vychodil *Discovery of optimal factors in binary data via a novel method of matrix decomposition* Journal of Computer and System Sciences, 2010

[3] *UCI Machine Learning Repository, 2013, [Online].* `http://archive.ics.uci.edu/ml/datasets.html`

[4] Petr Osička *Neuronové sítě - poznámky k přednáškám* Olomouc, 2011

[5] Raul Rojas *Neural Networks: A Systematic Introduction* Springer, 1996

[6] Alexander A. Frolov, Dusan Husek, Igor P. Muraview, Pavel Yu. Polyakov. *Boolean Factor Analysis by Attractor Neural Network* IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL.18, NO.3, MAY 2007



Obrázek 31. Průběh $\lambda(r)1$ a, experiment 3

[7] Alexander A. Frolov et al. *Recurrent-Neural-Network-Based Boolean Factor Analysis and Its Application to Word Clustering* IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL.20, NO.7, JULY 2009

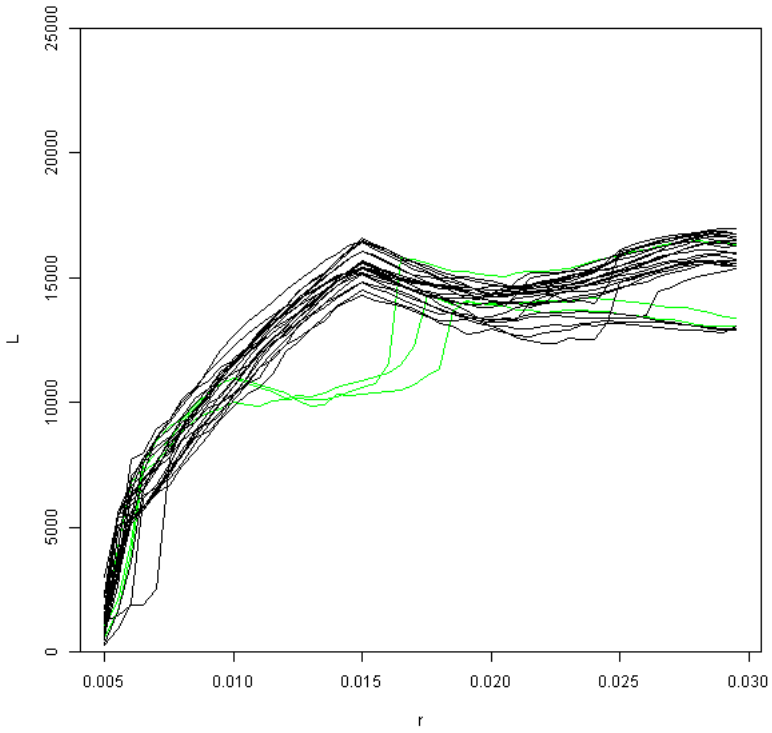
[8] Alexander A. Frolov et al. *Analysis and Evaluation of Different Methods for Bar Problem Solving - technical report No. 1082* Institute of Computer Science, Academy of Sciences of the Czech Republic

[9] Pavel Yu. Polyakov *Boolean factor analysis by Attractor Neural Network with Increasing Activity (ANNIA) - presentation slides* prezentace použitá pro představení BFAANN v Praze v únoru 2011

[10] *Lyapunov function - Wikipedia, the free encyclopedia. 2013, [Online].* `http://en.wikipedia.org/wiki/Lyapunov_function`

[11] *Attractor - Scholarpedia, 2013, [Online].* `http://www.scholarpedia.org/article/Attractor`

[12] *Hopfield network - Scholarpedia, 2013, [Online].* `http://www.scholarpedia.org/article/Hopfield_network`



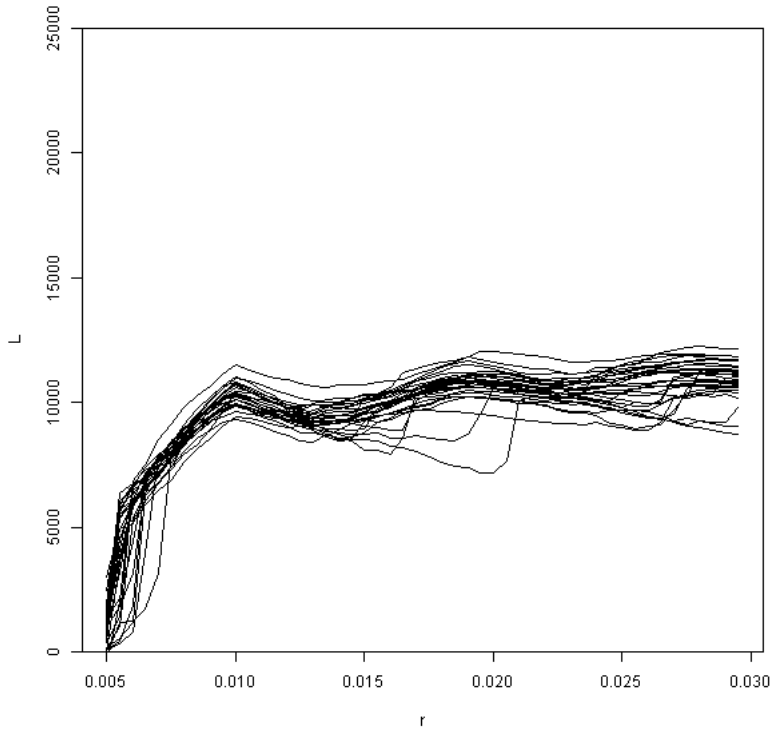
Obrázek 32. Průběh $\lambda(r)$ b, experiment 3

A. Dodatky

A.1. Implementace metod

Obě metody jsou sice implementovány v kompilovaném jazyce C♯, nicméně já jsem tento jazyk využíval téměř jako by byl interpretovaný. Tím mám na mysli, že jsem zkompilovaný program sám o sobě mimo debugger v podstatě nikdy nepustil, program tedy nedisponuje žádným uživatelským rozhraním. Pro každou z metod jsem vyvinul jakýsi „framework“ umožňující s ní pohodlně pracovat. Samotné experimenty jsou pak spíše sekvence příkazů (něco jako skript) pracující s daným „frameworkem“ (nastavení příslušných stavů, vyvolaní příslušných programových metod, atd.). Tímto způsobem bude s „frameworky“ předpokládám zacházet i jejich případný další uživatel. Jinými slovy, pokud by si případný uživatel například v učení metody BFAANN 6.5. přál potlačit vliv inhibičního neuronu na matici vah, musel by zavolat k tomu příslušnou metodu (ve smyslu metoda v objektovém programování).

Vzhledem k vyčerpávajícímu popisu obou zmíněných metod v tomto textu a dále díky tomu, že zdrojový kód je psán v souladu se zde užívanými pojmy



Obrázek 33. Průběh $\lambda(r)$ c, experiment 3

a konečně z faktu, že v kritických místech je kód dostatečně komentovaný, není zde, myslím, další popis takového zdrojového kódu nutný.

A.2. Lokace experimentů na CD

Zdrojové kódy „frameworků“, pomocí nichž byly jednotlivé experimenty provedeny, se na CD nacházejí v adresáři `src/`. Dále pak pro jednotlivé metody:

BFAFKA Experiment na datech `mushrooms` byl proveden pomocí zdrojových kódů v adresáři `MDFCA` a jeho dalších podadresářích.

BFAANN Experimenty byly provedeny pomocí zdrojových kódů v adresáři `MDHN` a jeho dalších podadresářích. Konkrétně jména jednotlivých podadresářů symbolizují, které experimenty byly, pomocí v nich obsažených zdrojových kódů, provedeny.

B. Obsah přiloženého CD

`doc/`
: Adresář obsahuje text diplomové práce ve formátu PDF spolu se soubory, jež jsou nutné pro jeho sestavení.

`src/`
: Kompletní zdrojové kódy a to jak pro metodu faktorizace dat pomocí formálních konceptů, tak pro metodu faktorizace binárních dat pomocí atraktorové neuronové sítě. Viz lokace A.1..

`data/`
: Obsahuje dataset `mushrooms` z [3].

`literature/`
: Prezentační slajdy Pavla Polyakova [9].