Binární faktorová analýza a komprese obrazu pomocí neuronových sítí

Mgr. Aleš Keprt

Katedra informatiky, FEI, VŠB - Technická Univerzita Ostrava
ales.keprt@vsb.cz

Abstrakt. Tento text stručně shrnuje mou práci za uplynulý školní rok. První kapitola se týká binární (booleovské) faktorové analýzy. Popisuje optimalizovaný algoritmus referenčního faktorizátoru, který se snaží najít globálně nejlepší řešení, aniž by zkoušel všechny možnosti. Zmíněna je také paralelní implementace programu.
Druhá kapitola představuje použití analýzy hlavních komponent (Principal Component Analysis, PCA) pro kompresi obrazu a řešení za pomoci lineární neuronové sítě. Převážná část textu je věnována shrnutí dosažených výsledků vlastní práce.

Klíčová slova: Binární faktorizace, neuronové sítě, analýza hlavních komponent

1 Binární (booleovská) faktorová analýza

1.1 Definice problému

Jedná se o faktorovou analýzu dichotomických (binárních) dat. Tato analýza se liší od klasické faktorové analýzy (viz [5]) tím, že pracujeme pouze s binárními hodnotami (0, 1) a používáme booleovskou aritmetiku (and, or). Binární faktorizace má široké využití v medicíně, chemii a dalších oblastech, doposud však nejsou známy žádné opravdu dobré algoritmy k jejímu řešení.

Binární faktorizace je zřejmě NP-úplný problém. Její řešení je tedy buď suboptimální (tedy nepřesné), nebo velmi pomalé. Výsledkem mé práce je referenční algoritmus, který jde cestou nalezení opravdu nejlepšího řešení. Vzniklý program má sloužit pro kontrolu dalších (již suboptimálních) algoritmů.

1.2 Optimalizovaný algoritmus

Základem řešení je postupné zkoušení různých možností. Nezkouší se však všechny, naopak je kladen důraz na optimalizace v podobě nalezení co největších podprostorů v prostoru možných řešení, kde hledané řešení určitě nebude.

Nulové řádky a **nulové sloupce** datové matice jsou pro faktorovou analýzu bezcenné, proto je možné je zcela vynechat. (Pro jedničkové sloupce a řádky to však neplatí.) Toto jsou spíše okrajové optimalizace, daleko významnější úspora času spočívá ve **slučování duplicitních řádků**. Místo duplicit stačí zapamatovat si jen jejich počet, jako multiplicitu neboli váhu řádku.

Optimalizace výpočtu matice A (factor loadings)
Ukázalo se, že je výhodné provést tzv. **seřazení faktorových vektorů**, tj. řádků matice A (matice factor loadings). Pro první řádek zkoušíme všechny možnosti, pro každý další řádek zkoušíme ty, které nebyly zkoušeny o řádek výše. Výsledkem je dvojnásobné zrychlení.

I přes všechny doposud uvedené optimalizace je stále rozsah běžných úloh nad rámec možností programu. **Stanovení pevného počtu jedniček na řádek** A je omezující prvek, ale pro použití k zamýšleným účelům se program nadále hodí a složitost řešení se extrémně sníží.

Optimalizace výpočtu matice F (factor scores)
Faktorizace je hledání rozkladu matice X na součin matic $F \cdot A$. Zjistil jsem, že při znalosti A je možno hledat F po řádcích, čímž se extrémně snižuje složitost celého výpočtu. Řešení, ke kterému výpočet F vede, obvykle není tím hledaným nejlepším řešením a při výpočtu F po řádcích na to můžeme přijít mnohem dříve, než vůbec vypočítáme celou F, a zbytek vůbec nepočítat.

1.3 Paralelní verze programu

Program jsem realizoval i paralelně, s využitím běžných nástrojů pro paralelní programování. Výsledný program jsem s úspěchem odzkoušel síti s Windows a Linuxem a to s vynikajícími výsledky. Základem úspěchu paralelní implementace je samotný algoritmus, který se pro paralelní výpočet vyloženě hodí.

Na testovacích počítačích vždy byl při paralelních výpočtech Linux vždy efektivnější než Windows. Provedl jsem řadu testů s jasným výsledkem - zatímco na Linuxu je zhruba 90% efektivita bez ohledu na počet výpočetních uzlů, ve Windows je efektivita výrazně menší a hodnoty hodně kolísají. Další variantou paralelního výpočtu, kterou jsem rozpracoval, je použití multithreadingu jako pomocníka pro ještě vyšší efektivitu výpočtu v paralelním prostředí.

Shrnutí: Paralelní verze programu může běžet současně na několika počítačích a výpočetní čas se tak zkracuje úměrně s růstem počtu počítačů. Samotná realizace ve Windows je zajímavá, neboť vědeckotechnické výpočty jsou obvykle realizovány pouze na Linuxu.

2 Komprese obrazu pomocí neuronové sítě řešící PCA

2.1 Úvod

PCA je jedním z možných přístupů ke kompresi obrazu a obecně jakýchkoliv vícerozměrných dat. Komprese pomocí PCA je založena na principu redukce dimenze vektorového prostoru. Aplikace na kompresi klasických bitmapových obrazů je přitom pouze jednou z mnoha možností, jak lze PCA využít.

PCA chápe každý bitmapový obraz jako množinu vektorů ve vícerozměrném prostoru. Tato množina vzniká tak, že obrázek je *nakostičkován* na útvary navzájem stejného tvaru, které zaplňují celou bitmapu. Obrázek ukazuje několik možných tvarů kostek. U šikmých hran není možné přesné mapování kostek na pixely, ale obvykle lze najít takové přibližné mapování, aby výsledek pokryl beze zbytku celou plochu.

Útvary vyplňující plochu - čtverec, obdélník, šestihran, roh, trojúhelník

Každá kostka je dále zpracovávána již bez ohledu na její tvar a polohu v původním obrazu. Při nakostičkování vstupních dat tedy dochází k abstrakci topologie, jelikož dále v PCA jsou data zpracovávána jako čistě abstraktní n-rozměrné vektory, kde n je počet pixelů v jedné kostičce.

2.2 PCA

Pomocí PCA lze pro dané n a m spočítat takové lineární zobrazení redukující prostor z dimenze n na dimenzi m, které je vzhledem k MSE[1] nejlepší.

Výsledkem výpočtu PCA je ortonormální matice typu $m \times n$, která definuje zobrazení z prostoru dimenze n do prostoru dimenze m. Vynásobením vstupního n-rozměrného vektoru touto maticí provedeme žádanou redukci dimenze (kompresi), výsledkem této operace je tedy m-rozměrný vektor.

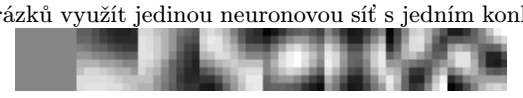
Důležitou vlastností je právě ortonormalita této transformační matice (zaručuje rovnost inverzní a transponované matice). Díky tomu je provedení inverzního zobrazení, které slouží ke zpětné transformaci (dekompresi), velmi snadné. (Toto je velice důležitá vlastnost.)

2.3 GHA - Výpočet PCA pomocí neuronové sítě

Výpočet PCA je velmi obtížný, založený na určení vlastních vektorů korelační matice vstupních dat. Neuronové sítě poskytují alternativu k řešení. Generalized Hebb(ian) Algorithm (GHA) je způsob, jak spočítat nejdůležitější vlastní vektory korelační matice bez výpočtu celé matice. V konečném důsledku tedy jde o neúplné řešení PCA v podobě výhodné pro kompresi.

Pro n-rozměrná vstupní data může GHA spočítat 1 až n vlastních vektorů. Tyto pak tvoří ortonormální bázi kompresovaného prostoru. V reálných datech jsou totiž obvykle určité vzájemné vztahy (korelace) mezi jednotlivými složkami prostoru, které PCA přirozeně „vyhledává“. Chyba vzniklá snížením dimenze prostoru je tak minimalizována (vůči MSE).

Výsledkem učení GHA-sítě o n vstupech a m neuronech je prvních m vlastních vektorů korelační matice, čili prvních m hlavních komponent. Následující obrázek ukazuje podobu prvních 8 vektorů pro kostky o rozměrech 8×8. Při kompresi většího počtu obrázků stejného typu (např. databáze rentgenových snímků) lze bez problémů pro kompresi všech obrázků využít jedinou neuronovou síť s jedním konkrétním naučením.



[1] MSE = Mean Square Error, střední kvadratická chyba - použitá vektorová metrika

3 Vlastní výsledky

Základním výsledkem mé práce v této oblasti je implementace neuronové sítě pro řešení PCA, která je velmi efektivní (rychlá) a umožňuje různá nastavení, díky kterým jsem mohl zkoumat problematiku z různých aspektů.

3.1 Zajímavé vlastnosti vektorů báze

Ve všech testech vyšel vždy první vlastní vektor jako víceméně jednobarevná kostka, určuje tedy průměrný jas kostky. (Důkaz empiricky. Je to pouze obecná vlastnost, teoreticky nemusí vždy platit.) Také na dalších významných vektorech je možno pozorovat určité snadno popsatelné barevné přechody, jako horizontální, vertikální nebo diagonální (viz obrázek výše).

3.2 Vliv velikosti kostky na kvalitu komprese

Testy jsem prováděl na různě velkých kostkách, vždy tvaru čtverce o sudých rozměrech. Kompresi jsem zvolil 1:4 a sledoval jsem kvalitu. Srovnávat kvalitu u různých kostek nelze pomocí vektorové MSE. Proto jsem počítal MSE přes celý obraz, tj. čtverce rozdílů jasů všech pixelů / počet pixelů.

Shrnutí výsledků: Čím větší jsou kostky, tím jsou výsledky kvalitnější. Tento vztah je jednoznačný, bohužel výpočet PCA pro velké kostky je velmi pomalý.

3.3 Vliv tvaru kostky na kvalitu komprese

Základní tvar je čtverec, testoval jsem však i jiné. Aby byly naměřené hodnoty vůbec srovnatelné, použil jsem výhradně kostky o 48 pixelech, které jsem opět kompresoval $1 : 4$, tj. za použití 12 vlastních vektorů.

Rozbor výsledků byl komplikovaný. Ukázalo se, že nestandardní tvary kostek nejsou vhodné, alespoň u testovaných obrazů dávají kostky ve tvaru čtverce vždy lepší výsledky než kostky jiných tvarů. Zajímavé také je, že nekonvexní „roh“ dává lepší výsledky než konvexní pravidelný trojúhelník a šestihran (viz obrázek výše). Další výhodou kostek ve tvaru čtverce je jejich jednodušší zobecnění do podoby vícerozměrných hyperkrychlí. Další testy jsem prováděl opět na kostkách o 48 pixelech, ovšem tentokrát šlo o srovnání různých obdélníků.

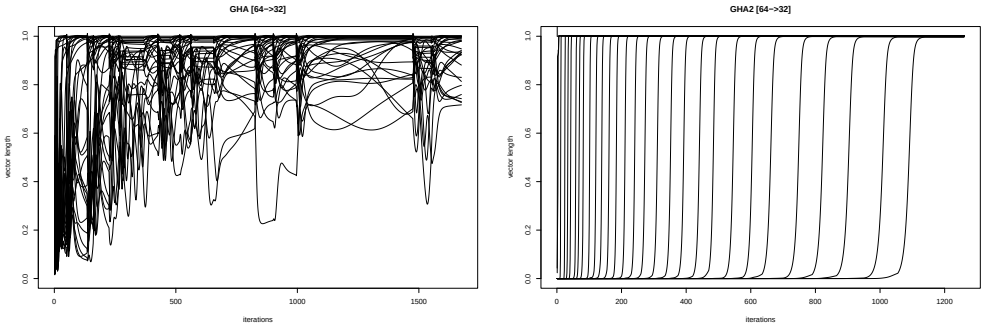
Shrnutí výsledků: I tyto pokusy ukázaly, že neobvyklé tvary kostek nejsou vhodné. Jednoznačně nejlepších výsledků bylo dosaženo u čtverců. Čím více je tvar kostky vzdálený od čtverce, tím jsou výsledky horší.

3.4 Automatizace učícího procesu

Zde jedno překvapení - všechny testy a výpočty jsem ve skutečnosti prováděl dosti modifikovanou verzí GHA. Modifikací jsem zkoušel a nakonec i použil hned několik a jejich cílem bylo zejména zrychlit proces učení.

Další obrázek ukazuje průběh učení sítě originálním algoritmem GHA (vlevo). Je vidět, že ani velký počet učících cyklů nevede ke zvlášť přesnému řešení. Vpravo je pro srovnání zobrazen průběh učení sítě zde popsaným upraveným algoritmem, nazvaným

GHA2. Tyto dva obrázky jsou natolik výmluvné, že nepotřebují žádný další komentář (úplné naučení neuronu je při $y = 1$).



Kvalitu naučení lze změřit pomocí dvou vlastností, které mají vlastní vektory splňovat: Jsou ortogonální, jsou jednotkové. Automatický systém učení, který jsem implementoval, průběžně testuje obě uvedené vlastnosti. Jakmile se délka vektoru přibližuje k 1.000 a skalární součin s každým z předchozích vektorů se blíží k nule, neuron náležející k tomuto vektoru je považován za naučený.

Vzhledem k tomu, že neuronová síť za daných podmínek dává pouze suboptimální řešení, mohou teoreticky nastat situace, kdy neuron nesplňuje kritérium naučení, avšak použitým algoritmem GHA již nelze v dané situaci dosáhnout lepšího naučení. To je způsobeno zejména numerickými chybami a lze to detekovat právě tak, že neuron je již ustálený. Je-li neuron ustálený během několika učících cyklů, považuje se za naučený bez ohledu na jeho délku.

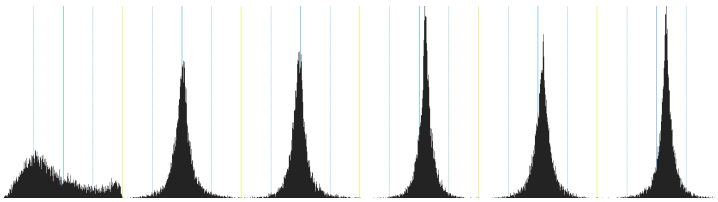
Laterální inhibice Algoritmus GHA používá laterální inhibici, jen díky ní je zajištěna ortogonalita vektorů. Vzhledem k tomu, že na každý neuron působí laterální inhibicí všechny předchozí neurony, nemá smysl neuron i učit, když kterýkoliv z neuronů 0 až $i-1$ ještě není alespoň částečně naučený.

Tato vlastnost mě přivedla k zásadní změně algoritmu GHA: Učí se v danou chvíli vždy pouze tři neurony. Jakmile je splněna podmínka naučení prvního neuronu, ten je zablokován a dále se již neučí. Místo toho se začne s učením čtvrtého neuronu. Takto se postupuje až do naučení celé neuronové sítě. Tento princip velmi výrazně zrychluje proces učení a přitom výsledky jsou totožné s originálním GHA.

Testy srovnávající kvalitu naučení originální GHA se zde popsaným upraveným algoritmem: Obojí dává neměřitelně podobné výsledky, nový algoritmus je však mnohem rychlejší.

3.5 Další zpracování zkompresovaných dat

Jednoduchými testy jsem zjistil, že zkompresovaná data, která jsou uložena pomocí lineární skalární kvantizace, lze ještě dále kompresovat běžným kompresním programem (zip, rar apod.) nebo aritmetickým kódováním. To značí, že v datech je určitá korelace, ačkoliv již jiného typu než ta, kterou jsme odstranili pomocí PCA. Je to velmi výhodná vlastnost umožňující další kompresi. Pozn.: Pro tyto testy jsem realizoval nový typ grafu - histogram kompresovaných dat po složkách. Tento graf dobře ukazuje některé skryté vlastnosti kompresovaných vektorů, čímž přímo nabízí konkrétní možnosti další (sekundární) komprese dat. (Detaily jsou nad rámec tohoto textu.)



3.6 Bitová hloubka (počet barev) obrázku

Provedl jsem sadu testů, kde jsem snižoval počet odstínů šedi postupně z 8 na 1 bit. Výsledky jsou nečekané - 16 vektorů, které jsem nechal počítat, vyšlo vždy velmi podobně, zcela bez ohledu na barevnou hloubku. Shrnutí: Na bitové hloubce obrazu při výpočtu PCA prakticky nezáleží. Snížení bitové hloubky však „jen“ šetří paměť, nijak nezrychluje výpočty.

3.7 Porovnávání obrazů pomocí PCA komprese

Kompresi na bázi PCA je možno využít také pro porovnávání obrazů, což je důležitá součást každého vyhledávacího algoritmu. V testech jsem použil celkem 7 různých metrik na sadách fotografií s různými stupni vzájemné podobnosti. Celkem 238 provedených testů přineslo celou řadu zajímavých výsledků, pozitivních i negativních. Některé metriky se v některých konkrétních situacích osvědčily více než jiné, avšak celkově se žádné „univerzálně nejlepší“ řešení nevyskytlo.

Reference

1. D.Húsek, A.Keprt, H.Řezanková, V.Snášel: *Nelineární booleovská faktorová analýza.* Konference ITAT, Szliezsky Dom, Vysoké Tatry, Slovensko, 2003.

2. *BMDP (Bio-Medical Data Processing).* Balík statistických programů. SPSS. `http://www.spss.com/`
3. A.A.Frolov, A.M.Sirota, D.Húsek, I.P.Muraviev, P.A.Polyakov: *Binary factorization in Hopfield-like neural networks: Single-step approximation and computer simulations*, 2003.
4. M.Oravec: *Neuronové siete pre extrakciu príznakov, kompresiu a rozpoznávanie obrazu.* Habilitační práce, Katedra telekomunikácií FEI STU, Bratislava, 2001.
5. K.Überla: *Faktorenanalyse* (2.vydání). Springer-Verlag, Berlin-Heidelberg, 1971. Slovenský překlad: Alfa, Bratislava, 1974.

Annotation:

Binary Factor Analysis and Image Compression Using Neural Networks

First part of this article is dedicated to binary (boolean) factorization. It presents my work on reference algorithm to be used for measuring quality of other algorithms. It is generally based on mindless searching of the optimal solution, and is advanced by several optimizations. Its parallel implementation is presented too. The second part of the article presents the usage of Principal Component Analysis (PCA) in image compression, and neural network (NN) based solver. Main portion of the article summarizes my findings in the area of PCA and NN.