

Juraj Holas

PCA: KOMPRESIA OBRAZU

PROJEKT Č.3, NEURÓNOVÉ SIETE, 2014/2015

ÚVOD

Úlohou projektu bolo vytvoriť neurónovú sieť implementujúcu PCA metódu pre kompresiu dát využitím APEX algoritmu. Sieť bude trébovaná a testovaná na obrázkoch v odtieňoch šedej, a bude sa skúmať vplyv počtu vybraných hlavných komponentov na schopnosť rekonštrukcie obrazu. Porovnané budú aj rôzne implementácie PCA: APEX algoritmus, autoasociačný viacvrstvový perceptrón a analytické riešenie.

IMPLEMENTÁCIA

Úloha je implementovaná v jazyku Java, pričom všetky zdrojové súbory sú dostupné v elektronickej prílohe. Jadrom projektu je implementácia triedy `APEX`, ďalšou dôležitou časťou je trieda `MLP` implementujúca autoasociačný viacvrstvový perceptrón. Obe z týchto tried majú obdobné metódy pre vytvorenie a inicializáciu siete, jej trébovanie na vstupnom obrázku, a na kompresiu a následnú rekonštrukciu testovacieho obrázku. V triede `APEX` je navyše možnosť získať naučené hlavné komponenty a im prislúchajúce vlastné čísla, a tiež aj metóda na analytické riešenie PCA.

V hlavnej triede je obsiahnuté načítavanie a ukladanie dát (vrátane rozdelenie obrázku na bloky a spätnej rekonštrukcie), a testy na získavanie výsledkov jednotlivých metód. Nastavujú sa tu tiež parametre behu programu:

- trébovací obrázok
- testovacie obrázky
- šírka a výška blokov
- počet extrahovaných hlavných komponentov
- počet epoch trébovania
- rýchlosť učenia α

Program je prispôsobený tak, aby sa dal použiť na obrázkoch rôznych veľkostí, podmienkou je iba aby šírka každého obrázku bola násobkom šírky bloku, obdobne pre výšku.

POUŽITÉ METÓDY

Implementované boli spomínané tri metódy riešenie PCA, ktoré sú podrobnejšie popísané nižšie. Každú z týchto metód predchádzalo predspracovanie dát – odstránenie priemeru a normovanie smerodajnej odchýlky. Takto normalizované dáta boli potom použité na trébovanie, pričom po výstupe zo siete sa presne opačným postupom denormalizovali. Pri kompresii a rekonštrukcii testovacích dát sa nepočítal priemer a smerodajná odchýlka nanovo, ale použili sa tie z trébovacích dát, nakoľko práve na takto upravené hodnoty bola sieť natrébovaná.

APEX

Ako prvý bol implementovaný algoritmus APEX – *Adaptive Principal components EXtraction*. Tento algoritmus využíva neurónovú sieť s jednou vstupnou a jednou výstupnou vrstvou, ktoré sú medzi sebou kompletne prepojené. Navyše sú prítomné aj laterálne spojenia na výstupnej vrstve, kde každý i -ty výstupný neurón je napojený na všetky predchádzajúce, teda na prvý až $(i - 1)$ -ty neurón. Výstup siete je daný vzorcom:

$$y_i = w_i^T x + u_i^T y_{i-1}$$

kde $u_i = [u_{i,1}, u_{i,2}, \dots, u_{i,i-1}]^T$ a $y_{i-1} = [y_1, y_2, \dots, y_{i-1}]^T$.

Pre učenie dopredných váh sa štandardne Ojovo učiace pravidlo:

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \alpha (y_i x_j - y_i^2 w_{ij}^{(t)})$$

Posledný člen v tomto vzťahu, teda $-y_i^2 w_{ij}^{(t)}$, je tzv. stabilizačný člen, ktorý má zabezpečiť, aby hodnoty váhového vektora (resp. matice) nedivergovali. Experimentálne sa však ukázalo, že aj napriek stabilizačnému členu mali vektory tendenciu divergovať, a to tým rýchlejšie, čím väčší blok bol zvolený (teda čím bola dimenzia vstupného vektora väčšia). Použili sme teda obmenu Ojového pravidla, kde sa po každej úprave váhový vektor explicitne normuje na jednotkovú dĺžku:

$$w_{ij}^{(t+1)} = \frac{w_{ij}^{(t)} + \alpha y_i x_j}{\|w_i^{(t)} + \alpha y x\|}$$

Táto verzia pravidla je síce mierne výpočtovo náročnejšia, no zastáva rovnakú úlohu. Pri použití tejto verzie sa už podarilo udržať váhové vektory pod kontrolou aby nedivergovali.

Laterálne váhy sa učia odlišne, a síce anti-Hebbovským inhibičným pravidlom:

$$u_{ij}^{(t+1)} = u_{ij}^{(t)} - \alpha y_i (y_j + y_i u_{ij}^{(t)})$$

Implementáciou týchto pravidiel sa sieť vedela správne naučiť prvých p hlavných komponentov tréningových dát. Dopredné váhy konvergovali k vlastným vektorom korelačnej matice dát, čiže hlavným komponentom, stredná hodnota štvorca výstupov $\langle y^T y \rangle$ konvergovala ku zodpovedajúcim vlastným číslam, a laterálne váhy k nulovým vektorom.

Po každej epoche sa vyhodnocovala chyba ako súčet štvorcov odchýlky rekonštruovaného vstupu od pôvodného, pre každý vstupný vzor v :

$$\begin{aligned} err &= \sum_{v=1}^N \sum_{i=1}^n (x_i^{(v)} - \tilde{x}_i^{(v)})^2 \\ \tilde{x}^{(v)} &= W y^{(v)} \\ W &= [w_1, w_2, \dots, w_p] \end{aligned}$$

Autoasociačný viacvrstvový perceptrón

Druhou skúšanou metódou pre PCA bol autoasociačný viacvrstvový perceptrón (ďalej označovaný MLP). Ide o sieť s jednou skrytou vrstvou a počtami neurónov $n-p-n$ ($p < n$), kde na všetkých vrstvách je použitá lineárna aktivačná funkcia, a pre každý vzor platí $target^{(v)} = x^{(v)}$. Takáto sieť vytvára na strednej (skrytej) vrstve hrdlo, cez ktoré prechádzajú dáta v komprimovanom stave, a následne sa rekonštruujú späť na výstup. V tomto prípade sa na strednej vrstve vytvára rovnaká lineárna redukcia do podpriestoru ako pri APEX či iných PCA algoritmoch, avšak s tým rozdielom, že hlavné komponenty nie sú usporiadané. Tzn. že aktivácia na prvom skrytom neuróne nie je nutne priemetom na prvý hlavný komponent, atď.

Výstup siete aj tréning prebieha podľa štandardných vzťahov pre *back-propagation* (pre lineárnu aktivačnú funkciu):

$$h = W_{hid}x$$

$$y = W_{out}h$$

$$\delta_{out} = target - y$$

$$\delta_{hid} = W_{out}^T \delta_{out}$$

$$W_{out}^{(t+1)} = W_{out}^{(t)} + \alpha \delta_{out} h^T$$

$$W_{hid}^{(t+1)} = W_{hid}^{(t)} + \alpha \delta_{hid} x^T$$

Podobne ako aj pri APEX metóde, aj pri MLP sme narazili na problém divergujúcich váh. Rovnako ako v predchádzajúcom prípade, aj tu pomohlo explicitné normovanie váhovej matice (podľa Forbeniusovej normy: $\|A\| = (\sum_{i,j} a_{ij}^2)^{1/2}$).

$$W_{out}^{(t+1)} = \frac{W_{out}^{(t)} + \alpha \delta_{out} h^T}{\|W_{out}^{(t)} + \alpha \delta_{out} h^T\|}$$

Opäť, ako aj pri APEX algoritme, aj tu sme vyhodnocovali chybu po každej epoche rovnakou metódou súčtu štvorcov:

$$err = \sum_{v=1}^N \sum_{i=1}^n (x_i^{(v)} - y_i^{(v)})^2$$

Analytické riešenie PCA

Poslednou implementovanou metódou bolo analytické riešenie pre identifikáciu hlavných komponentov. To vychádza zo vzťahov korelácie medzi jednotlivými zložkami dát. Po vytvorení korelačnej matice R jej vlastné vektory zoradené podľa veľkosti prislúchajúcich vlastných čísel určujú hlavné komponenty dátovej množiny.

$$R = \langle x x^T \rangle = \frac{1}{N} \sum_{v=1}^N x^{(v)} x^{(v)T}$$

$$Ru_i = \lambda_i u_i \quad \wedge \quad \lambda_1 > \lambda_2 > \dots > \lambda_n \Rightarrow u_1, u_2, \dots, u_n \text{ sú hlavné komponenty}$$

Malo by byť rýchlejšie a presnejšie namiesto rátania eig(R) rátať priamo svd(vstupy). Ale porovnávanie s analytickou metódou je určite dobrá vec.

Nakoľko táto metóda je exaktná, používala sa ako referenčné riešenie na porovnávanie úspešnosti zvyšných dvoch postupov. Pri veľkom n a malom p (teda redukcia z veľmi veľa rozmerných dát na málo rozmerov) je však táto metóda menej efektívna, nakoľko sa počítajú všetky vlastné vektory a čísla, i keď sa z nich použije iba prvých p a zvyšok sa zahodí.

VÝSLEDKY

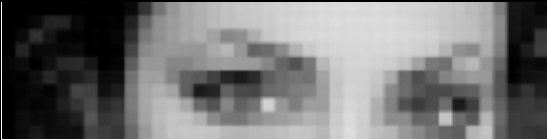

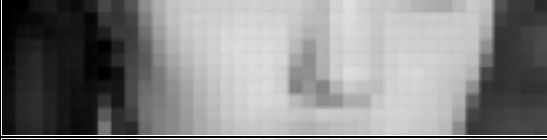

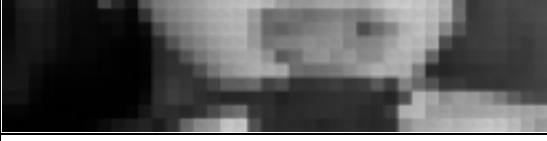





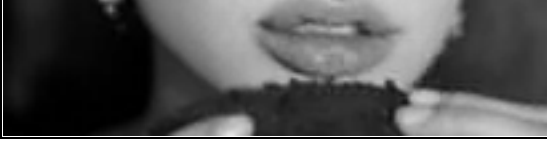
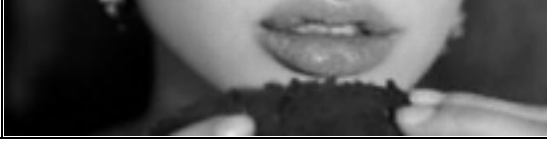
Všetky metódy sme vyskúšali natrénovať na vybranom obrázku *Angelina*, a následne sme pomocou nich skomprimovali a späťne zrekonštruovali jednak obrázok *Angelina*, a jednak aj dva testovacie obrázky *Emma* a *Lenna*.

Pri všetkých testoch boli rovnaké parametre:





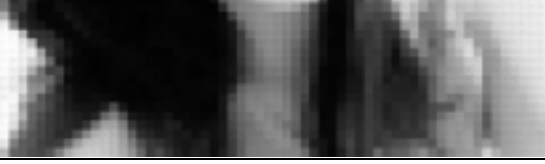







- veľkosť blokov 10×10 px
- extrahovaných 15 hlavných komponentov
- tréovanie dlhé 1000 epoch (pri APEX a MLP)
- rýchlosť učenia $\alpha = 0,001$ (pri APEX a MLP)

V nasledujúcich tabuľkách sú uvedené výsledky jednotlivých metód pre vybrané počty hlavných komponentov: 1, 3, 8 a 15. Všetky výstupné obrázky v plnom rozlíšení môžete nájsť v elektronickej prílohe v priečinkoch `data/run*/`.

Tabuľka 1 - tréovací obrázok *Angelina*

	$p = 1$	$p = 3$
analyt. PCA		
APEX		
MLP		
	$p = 8$	$p = 15$
analyt. PCA		
APEX		
MLP		

Tabuľka 2 - testovací obrázok *Emma*

	$p = 1$	$p = 3$
analyt. PCA		
APEX		
MLP		
	$p = 8$	$p = 15$
analyt. PCA		
APEX		
MLP		

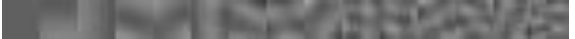

Tabuľka 3 - testovací obrázok *Lenna*

	$p = 1$	$p = 3$
analyt. PCA		
APEX		
MLP		
	$p = 8$	$p = 15$
analyt. PCA		
APEX		
MLP		

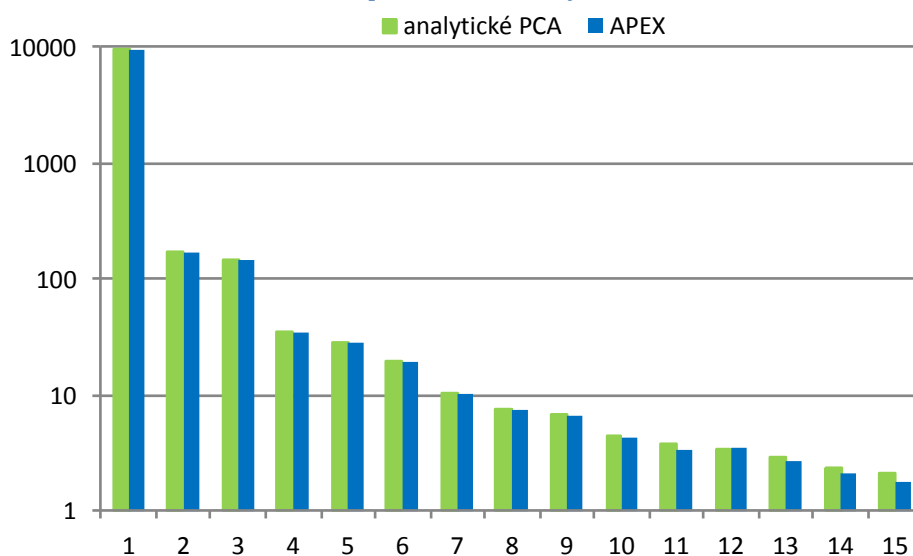
Po vizuálnej stránke môžeme zhodnotiť, že najlepšie výsledky malo, podľa očakávaní, analytické riešenie. O niečo horšie výsledky podal APEX, a z tejto trojice najhoršie výsledky mal MLP. Rozdiely sú však pomerne malé, všetky tri metódy boli schopné obrázky rekonštruovať pomerne vierohodne. Napr. pri $p = 15$ a veľkosti bloku 10×10 px išlo o komprimovanie na 15% pôvodnej veľkosti obrázku, pričom po vizuálnej stránke došlo ku minimálnemu zhoršeniu.

Porovnali sme tiež vlastné čísla a vlastné vektory (hlavné komponenty) naučené pomocou APEX algoritmu v porovnaní s tými vypočítanými analyticky. Hlavné vektory sú znázornené graficky v tabuľke 4, porovnanie vlastných čísel zobrazuje graf 1. Za účelom porovnania boli vlastné čísla preškálované tak, aby boli z približne rovnakých rozsahov. Graf je navyše v logaritmickej mierke kvôli nepomeru veľkosti prvého vlastného čísla a zvyšku.

Tabuľka 4 - porovnanie vlastných vektorov

analytické PCA	
APEX	

Graf 1 - porovnanie vlastných čísel

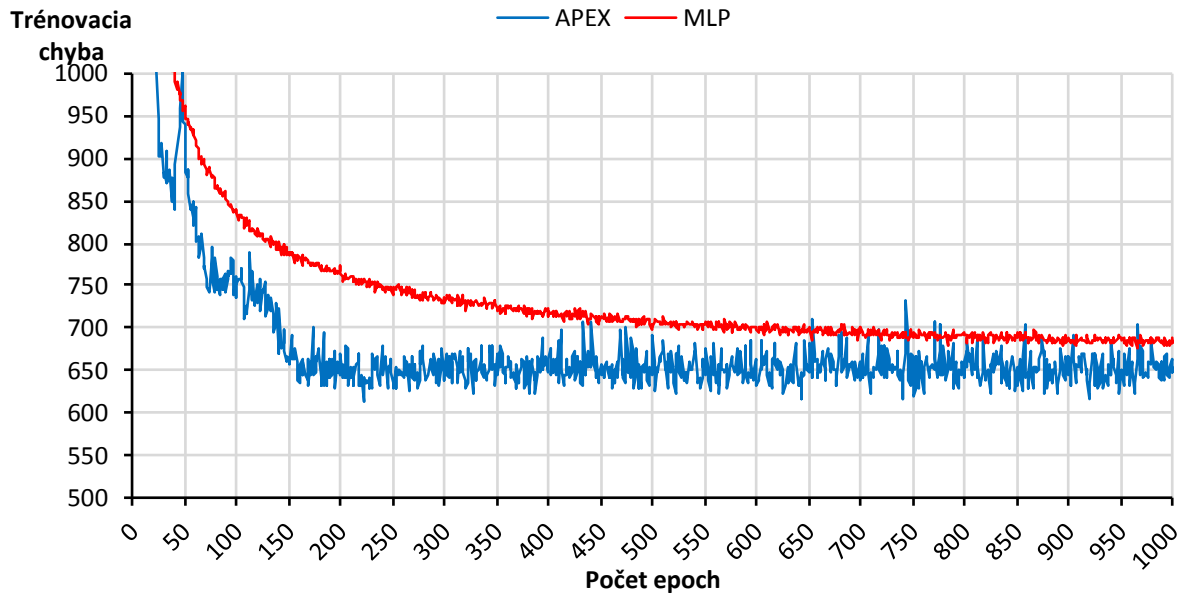


Vidíme, že APEX dosahuje výsledky veľmi podobné referenčnému analytickému riešeniu ako pri vlastných vektoroch, tak aj pri im prislúchajúcich číslach. (Pripomeňme, že pri počítaní vlastných vektorov sú u_i a $-u_i$ ekvivalentné výsledky. Preto napr. tretí či piaty vlastný vektor vyšiel v jednej metóde ako negácia výsledku druhej metódy.)

Ako posledné sme porovnávali rýchlosť naučenia neurónovej siete pri použití APEX a MLP. Za týmto účelom sme sledovali vývoj trénovacej chyby po každej epoche pri oboch metódach. Ako stav naučenia sme brali moment, od ktorého sa trénovacia chyba už výrazne nezlepšovala. Kompletne výsledky zobrazuje graf 2. Prvých zhruba 50 hodnôt, ktoré dosahujú hodnoty 1000 až 35000 nie sú z hľadiska porovnávania rýchlosti naučenia zaujímavé, preto boli vynechané z mierky.

Na výsledkoch vidíme, že chyba APEX začala stagnovať po zhruba 160 epochách. MLP si po celý čas držal väčšiu chybu, t.j. učí sa pomalšie, avšak chyba ani po 1000 epochách klesať neprestala. Po 350 epochách už bol však pokles veľmi pomalý, takmer minimálny. Môžeme teda skonštatovať, že APEX sa vo všeobecnosti učí rýchlejšie ako MLP.

Graf 2 - porovnanie vývoja trénovacej chyby



ZÁVER

Počas tohto projektu sme implementovali tri rôzne algoritmy pre PCA – APEX, autoasociačný viacvrstvový perceptrón a analytické riešenie. Ich použitie sme demonštrovali na úlohe kompresie obrázkov, a následne sme porovnali ich výsledky. Zistili sme, že APEX poskytuje lepšiu rekonštrukciu obrazu ako MLP, nie však natoľko dobrú ako analytické riešenie, i keď od nej nie je d'aleko. Tiež sme porovnali, že APEX algoritmus sa dokáže naučiť rýchlejšie než MLP.