Peter Csiba 13.05.2011

Neurónové siete – PCA implementované algoritmom GHA

Úvod

Pri spracovaní vysokorozmerných dát vzniká často potreba zníženia ich dimenzie (objemu), pričom požadujeme minimálne znížení kvality. Jednou z jednoduchších metód takej transformácie dát je metóda hlavných komponentov (principal component analysis, PCA). Metódu PCA je možné implementovať ako klasický výpočtový algoritmus, alebo ako model samoorganizujúcej sa neurónovej siete, ktorej učenie funguje na Hebbovskom princípe. V oboch prípadoch je v pozadí používaná lineárna algebra.

My sme si vybrali práve Hebbovské učenie na samoorganizujúcej sa sieti. Táto metóda sa na nazýva GHA. V projekte popisujeme trénovanie siete na obrázku lena.png a naučenú sieť potom testujeme na obrázkoch moon.png a elaine.png.

Prezentujeme výsledky pre rôzne úrovne kompresie 8x8 podobrázkov, ktoré zodpovedajú 64 rozmernému vektoru. Vstupný obrázok sme rozložili na 32x32 takýchto vektorov, ktoré sme natrénovanou neurónovou sieťou komprimovali. V nasledujúcom texte označujeme p (resp. P) počet hlavných komponentov (=výstupných neurónov siete).

Na implementáciu algoritmu sme použili interpretovaný jazyk PHP a jej knižnice. Musíme povedať, že to bola nevhodná voľba. Hlavným problémom PHP je jeho rýchlosť. Podobný kód v jazyku C počítal rádovo 10-15 krát rýchlejšie. Jediná výhoda zvoleného postupu je, že daný program je jednoducho poskytnuteľný ako webová služba na väčšine webserverov, respektíve programátorské pohodlie samotného autora.

Predspracovanie dát

Podľa štandardného postupu sme dáta najprv centralizovali. To znamená, že od všetkých vstupných dát sme odčítali priemerný vstupný vektor (čím sme dosiahli E(X)=0). Okrem toho sme rozsah hodnôt (-128, 128) znížili na štandardný rozsah (-1, 1) predelením.

Po dekódovaní dát z neurónovej siete je nutné spraviť inverzný proces k hore uvedenej normalizácií. Čo znamená vynásobiť výsledky číslom 128 a pričítať vektor priemernej hodnoty.

Trénovací postup

Ako sme už spomínali, vstupný obrázok sme rozdelili na podobrázky veľkosti 8x8 ktoré sme transformovali do 64 rozmerných vektorov. Sieť sme trénovali v epochách. V každej epoche sme permutovali vstupné dáta. Pri spracovaní konkrétneho vstupu sme najprv vypočítali forward pass a následne sme upravili váhy podľa pravidla.

Poznámka: váhový vektor $W je indexovaný opačne ako štandardne: $W[output][input]

```
//Učiaca funkcia
//http://en.wikipedia.org/wiki/Generalized_Hebbian_Algorithm
function applyrule(&$X, $i, $j){
  global $alpha, $W, $Y;
  $s = 0;
  for($k=0; $k<=$j; $k++) $s += $W[$k][$i] * $Y[$k];
  return $alpha * $Y[$j] * ($X[$i] - $s);
}

//Trénovanie jednej epochy
for($in_id=count($train)-1; $in_id >= 0 ; $in_id--){
    //FORWARD PASS
    for($Y_id=0; $Y_id<$p; $Y_id++){
      $Y[$Y_id]=0;
      for($X_id=0; $X_id<$n; $X_id++){
        $Y[$Y_id] += $W[$Y_id][$X_id] * $train[$in_id][$X_id];
      }
    }

    //foreach weight
    for($X_id=0; $X_id<$n; $X_id++){
      for($Y_id=0; $Y_id<$p; $Y_id++){
        //online
        //$W[$Y_id][$X_id] += applyrule($train[$in_id], $X_id, $Y_id);
        //batch
        $plusW[$Y_id][$X_id] = applyrule($train[$in_id], $X_id, $Y_id);
      }
    }
    //batch
    for($i=0; $i<$n; $i++) for($j=0; $j<$p; $j++)
      $W[$j][$i] += $plusW[$j][$i];
  }
  shuffle($train);
```

Kódovanie

Podľa teórie za algoritmom vieme, že váhové vektory reprezentujú maticu transformácie súradníc do priestoru hlavných komponentov. A teda kódovanie spočíva v jednoduchom forward pass-e pre každý vektor zo vstupu. Všetky výstupy siete si pamätáme a spolu s popisom siete ich považujeme za kód vstupu.

Dekódovanie

Aby sme kódové vektory transformali do pôvodného priestoru, je nutné vykonať inverznú transformáciu. Z vlastností váhových vektor vieme, že v tomto prípade je inverzná matica ekvivalentná transponovej matici. To znamené, že pre dekódovanie nám stačí vykonať backward pass (pre každý výstupný vektor).

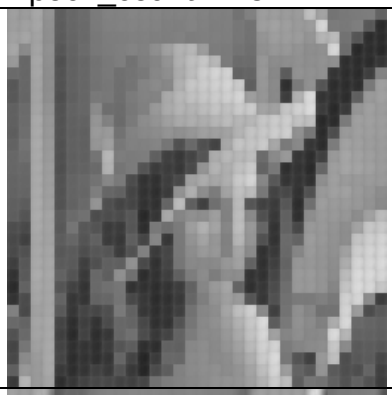
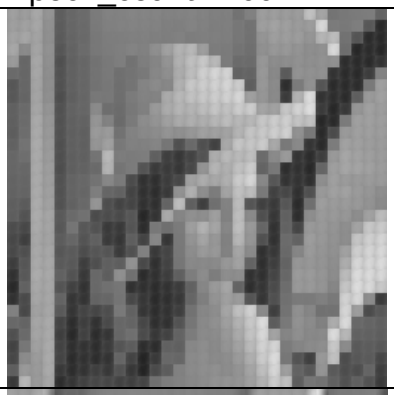
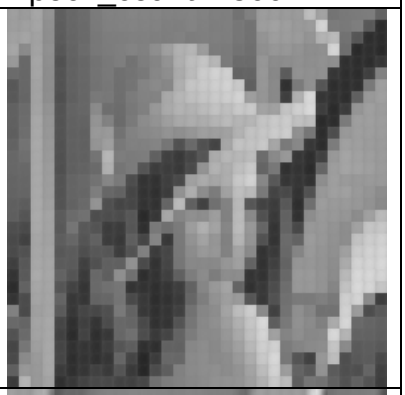
Ak chceme výsledky reprezentovať obrázkom, je nutné vykonať denormalizáciu, ktorú sme spomínali v odseku “Predspracovanie dát“.

Výsledky

Obrázky a bázové vektory pre jednotlivé hodnoty vstupných parametrov

Pri testovaní samotného programu sme určili ideálnu alphu ako 0,001. Pre menšiu alphu bol obrázok veľmi kontrastný a pre menšiu alphu nebolo vidieť zlepšenie, respektíve program konvergoval pomerne pomali. (Takúto alphu sme zvolili napriek tomu, že Haykin odporúča optimálnu alphu 0,0001.)

V nasledujúcej tabuľke znázorňujeme výsledky pre rôzne vstupné parametre. Vo všetkých sme používali alpha = 0,001.

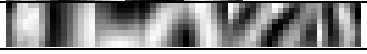
	Epoch_count = 25	Epoch_count =100	Epoch_count = 500
			
P=2			
P=4			
P=8			

P=16			
Originál			

V nasledujúcej tabuľke je vypočítaný priemer štandardnej odchylky komprimovaného obrázku od originálneho obrázku (pre každú dvojicu vstupného a výstupného pixelu sme pripočítali ich rozdiel na druhú a celú sumu predelili počtom pixelov).

	Epoch_Count=25	Epoch_Count=100	Epoch_Count=500
P=1	493.707366943	493.618225098	493.599380493
P=2	294.272521973	297.277374268	293.851242065
P=4	225.055389404	185.9921875	185.08555603
P=8	137.60043335	107.113082886	95.3569488525
P=16	114.505462646	62.0952758789	44.4181518555

Nasledujúca tabuľka zobrazuje bázové vektory, ktoré sú naškálované tak, aby boli dobre vidieť rozdieli na ich jednotlivých súradniciach. Sú zoradené podľa dôležitosti a nie sú oddeľované.

	Epoch_Count=25	Epoch_Count=100	Epoch_Count=500
P=1			
P=2			
P=4			
P=8			
P=16			

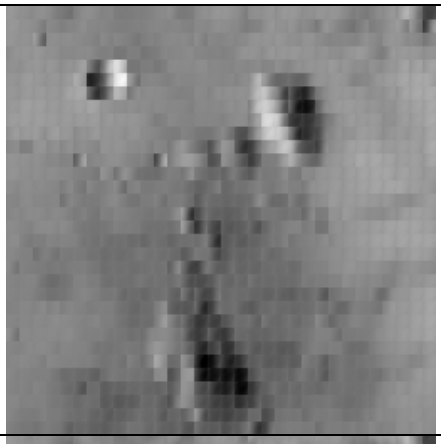
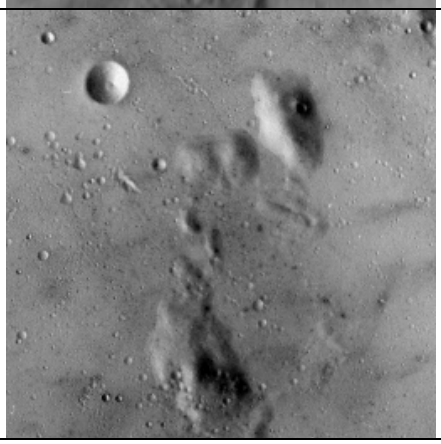
Kovariančná matica a jej vlastné čísla

Nevedeli sme zistiť zo všeobecnej literatúry jednoznačné definovanie kovariančnej matice. Na Wikipédií definujú kovariančnú maticu rovnako ako korelačnú maticu definuje pán Farkaš v slide-och. A preto len vypíšeme hodnoty kovariančnej matice pre p=8, epoch_count=100. Vidíme, že vektory transformácie sú skoro ortonormálne (sú návzojom na seba kolmé (ich skalárny súčin je rovný 0) a ich dĺžka je 1 (skalárny súčin samého so sebou je rovný 1)).

1.0006	0.0054	-0.0113	0.0057	-0.0019	0.0039	-0.0025	-0.0006
0.0054	1.0027	-0.0010	0.0086	-0.0035	0.0003	-0.0051	-0.0028
-0.0113	-0.0010	1.0028	-0.0011	0.0002	-0.0025	0.0139	0.0009
0.0057	0.0086	-0.0011	1.0019	0.0229	-0.0068	0.0266	-0.0069
-0.0019	-0.0035	0.0002	0.0229	1.0023	0.0277	-0.0992	0.0140
0.0039	0.0003	-0.0025	-0.0068	0.0277	0.9911	0.3280	-0.0897
-0.0025	-0.0051	0.0139	0.0266	-0.0992	0.3280	0.5741	0.0990
-0.0006	-0.0028	0.0009	-0.0069	0.0140	-0.0897	0.0990	0.9704

Generalizácia siete

V oboch prípadoch generalizácie sme použili sieť natrénovanú na obrázku lena.png s 500 epochami a alphou 0,001. V prvých troch riadkoch sú výsledky pre p=2, p=4 a p=8. V poslednom riadku sú znárone né originálne obrázky. Vidíme, že z väčšej diaľky pre p=8 je rekonštruovaný obrázok len ťažko rozlíšiteľný od originálu. Najväčšie rozdieli vidieť na ostrých farebných prechodoch obrázku elaine.png.

P = 2		
P = 4		
P=8		
Originál		

Porovnanie výsledkov GHA s viacvrstvovou neurónovou sieťou typu n-p-n

Ako implementáciu n-p-n sme použili program z minulého projektu backpropagation. V tomto prípade sme nehladali najlepší model, ale natvrdo sme určili hodnoty konštánt tak, aby boli čo najpodobnejšie konštantám použitých v GHA. Presnejšie parametre siete backpropagation:

-jedna skrytá vrstva s 8 neurónmi (zodpovedajú p=8).
-*alpha*=0,001 ; *mi*=0,003 ;
-použili sme *k-cross validation* pri trénovaní s k=5 (a vybrali najlepší z týchto modelov)
-*tangent activation function* (kvôli potenciálnym záporným hodnotám)
-*stopping criteria*: buď bolo dosiahnutých 400 epôch, alebo sa výstup dlho nezlepšoval

Do tejto backpropagation siete sme zadali vstupy centrované a normované (rovnako ako v GHA). Pred behom algoritmu sme ich náhodne permutovali (kvôli k-cross validation – aby validačná vzorka bola náhodná podmnožina vstupu). Sieť sme nechali trénovať na normalizovanej množine podobrázkov, pričom požadované výstupy boli rovnaké ako vstupy. To znamená, že sieť n-p-n v prvej vrstve komprimuje a v druhej vrstve sa snaží čo najlepšie dekomprimovať. Dekomprimované výsledky sme porovnali s originálnym obrázkom pomocou štandardnej odchýlky použitej už pri vyhodnocavní výsledkov GHA. Pre vizuálne rozlíšenie prikladáme aj výstupný obrázok z n-p-n a originálny obrázok.

Error= 1374.00141907

Vidíme, že model GHA je rádovo vhodnejší ako n-p-n backpropagation. Je možné, že v implementácií (resp. vstupoch) algoritmu backpropragation je nejaká chyba. Bol ale použitý rovnaký model ako v predošlom projekte, ktorý dával dobré výsledky.

Záver

Zobrazovali sme výsledky jednoduchej implementácie algoritmu GHA pre prístup PCA. Výsledky sú uspokojúce a korelujú so všeobecnými výsledkami .

Z chýb pri rôznych vstupných parametrov vidíme, že viac epôch má zmysel hlavne pri väčších hodnotách *p*.

Súbory

gha.php – algoritmus gha implementovaný v jazyku php
gha_transform.php – prekladá vstupný obrázok do číselných vektorov a naopak

Zdroje

-http://ii.fmph.uniba.sk/~farkas/Courses/NeuralNets/ns-proj3a.pdf
-http://en.wikipedia.org/wiki/Principal_component_analysis
-http://en.wikipedia.org/wiki/Generalized_Hebbian_Algorithm
- Neural Networks A Comprehensive Foundation – Simon Haykin