Rozpoznawanie obrazów
Laboratorium nr 5, 6

Tomasz Gałecki

25 kwietnia 2019

Wstęp

Raporty z uczenia, które będę dołączał, mają następującą interpretację:

```
% epoch   duration    terr       trainCorrect  testCorrect
1.00000   92.16672    1.25384    0.82530       0.81370
```

Wszelkie pliki i funkcje *...Old.m* należą do implementacji referencyjnej (z rozdziału 1), zaś wszystkie bez przyrostka - do implementacji poprawionej (sugestiami z artykułu).

Instrukcje w Octave na bazie których powstało to sprawozdanie znajdują się w pliku *mainscript.m*.

Łącznie przeprowadziłem około 20-30 przeuczeń sieci dla różnych parametrów (w tym też sieci o rozmiarach nawet [200 150 100 50] oraz oczywiście mniejszych). Wnioskiem z tych eksperymentów było to, że im więcej warstw w sieci, tym trudniej jest ją poprawnie sparametryzować - takie sieci bardzo agresywnie reagowały na minimalne zmiany współczynnika uczenia i nawet po zastosowaniu poprawek z artykułu takimi sieciami trudno było przebić 88.0% poprawności.

Pewnym zaskoczeniem była sieć [50 40], która osiągnęła przyzwoite wyniki (87.13% poprawności), ale nie udało mi się jej bardziej poprawić i to sieć [100 100] została włączona do sprawozdania.

Dla sieci [100 100 100] udało mi się osiągnąć tylko 87.38% poprawności. Dodatkowo trening takiej sieci był bardzo czasochłonny.

1 Przygotowanie referencyjnej implementacji uczenia sieci

Sama implementacja uczenia przez wsteczną propagację została przygotowana również do współpracy w przetwarzaniu wsadowym (operując na całym zbiorze próbek jednocześnie), ale wyniki uczenia były mizerne i powolne. Bardzo szybko powróciłem do metody stochastycznej. Jednak pociągnęło to konieczność dodania w *ann_training.m* pętli iterującej:

```
terr = 0;
for i=1:rows(tvec)
    [network terrN] = backprop(tvec(i, :), tlab(i, :), network, learningRate);,
    terr += terrN;
endfor
terr /= rows(tvec);
```

1.1 Zastosowane usprawnienia

Implementacja referencyjna została zmodyfikowana do współpracy z sieciami o dowolnej liczbie warstw (analogicznie poprawiony został klasyfikator i generator sieci).

W celu przyśpieszenia uczenia sieci na początkowych stadiach, użyłem agresywniejszej metody doboru współczynnika $\eta_i = \frac{lr}{i}$, gdzie η_i to współczynnik uczenia dla epoki i, a lr to początkowa wartość współczynnika (w tym przypadku 0.1). Jednak ta metoda nie sprawdziła się dobrze, sieć po kilkunastu epokach wpadała w stagnację. Ostatecznie wykorzystywałem stały współczynnik uczenia.

Próbowałem również użyć różnych współcznynników uczenia dla różnych warstw sieci (badałem to nawet na sieci 5 warstowej) - i nie dało to lepszych rezultatów dla przeprowadzonych przeze mnie eksperymentów. Dobór

wielu współczynników jest jeszcze trudniejszy, choć ta metoda mogłaby się dobrze sprawdzić przy bardziej pogłębionych badaniach.

1.2 Wynik

Najlepsze i najbardziej stabilne wyniki (gdzie sieć osiągnęła maksimum po 43 epokach) osiągnąłem dla sieci o 2 warstwach ukrytych w rozmiarach [100 100] i stałym współczynnikiem uczenia (0.1).

Maksymalna dokładność to **87.6%** dla zbioru testującego (przy 91.987% dokładności na zbiorze uczącym).

```
>> [networkHistory trrep] = ann_trainingOld(0.1, [100 100], 50);
     1.00000   172.24602     1.26910     0.77695     0.76590
     2.00000   171.67989     1.21866     0.84782     0.83320
     3.00000   172.12110     1.20847     0.85487     0.84000
     4.00000   173.58589     1.20421     0.86190     0.84450
     5.00000   163.63705     1.20163     0.86920     0.85120
     6.00000   159.61460     1.20002     0.87335     0.85320
     7.00000   159.92632     1.19855     0.87353     0.85120
     8.00000   159.61077     1.19725     0.87473     0.85090
     9.00000   159.50308     1.19622     0.87607     0.85210
    10.00000   159.62119     1.19532     0.87655     0.85190
    11.00000   160.00367     1.19454     0.88037     0.85440
    12.00000   159.60899     1.19371     0.88573     0.86000
    13.00000   159.48702     1.19303     0.88738     0.86010
    14.00000   159.54709     1.19234     0.89360     0.86490
    15.00000   159.23509     1.19171     0.89477     0.86340
    16.00000   159.71886     1.19115     0.89400     0.86320
    17.00000   159.23685     1.19073     0.89388     0.86430
    18.00000   159.11666     1.19010     0.89430     0.86400
    19.00000   159.43227     1.18989     0.89742     0.86390
    20.00000   159.36999     1.18958     0.90008     0.86790
    21.00000   159.26313     1.18930     0.89867     0.86460
    22.00000   161.00294     1.18879     0.90025     0.86740
    23.00000   179.70555     1.18833     0.90225     0.86750
    24.00000   179.53357     1.18780     0.90213     0.86800
    25.00000   179.36649     1.18810     0.90330     0.86710
    26.00000   179.35197     1.18752     0.90337     0.86670
    27.00000   179.27783     1.18701     0.90228     0.86520
    28.00000   178.51398     1.18661     0.90755     0.86930
    29.00000   179.42574     1.18619     0.90783     0.86950
    30.00000   176.68362     1.18589     0.90825     0.86980
    31.00000   176.44597     1.18554     0.91163     0.87100
    32.00000   176.75294     1.18516     0.90857     0.86900
    33.00000   176.64640     1.18495     0.91058     0.86740
    34.00000   176.65471     1.18457     0.91075     0.86930
    35.00000   179.44416     1.18455     0.91417     0.87020
    36.00000   177.63567     1.18430     0.91193     0.86930
    37.00000   178.42839     1.18388     0.91295     0.86870
    38.00000   178.17539     1.18398     0.91335     0.86800
    39.00000   177.84848     1.18386     0.91467     0.87000
    40.00000   177.56136     1.18349     0.91655     0.87040
    41.00000   177.26419     1.18343     0.91517     0.87110
    42.00000   177.71169     1.18383     0.91717     0.87400
    43.00000   159.63541     1.18347     0.91987     0.87600 % max
    44.00000   150.92415     1.18306     0.92065     0.87520
    45.00000   150.79917     1.18294     0.91997     0.87100
    46.00000   150.22749     1.18361     0.91957     0.87120
    47.00000   150.69771     1.18313     0.92200     0.87240
    48.00000   150.63733     1.18282     0.92070     0.87390
    49.00000   151.00096     1.18258     0.92472     0.87410
    50.00000   149.12715     1.18217     0.92037     0.87190
```

Macierze błędów:

```
>> [_ ep] = max(trrep(:, 5));
>> clsResTrain = annclsOld(tvec, networkHistory{ep});
>> cfmxTrain = confMx(tlab, clsResTrain)
        cfmxTrain =
           5312      1    159    225     14      4    250      0     34      1      0
             12   5897     12     57      6      1     15      0      0      0      0
             35      0   5265     39    467      1    187      0      6      0      0
             56     10     50   5677    112      0     86      0      8      1      0
              9      4    364    144   5239      1    232      0      7      0      0
              0      0      1      0      0   5911      0     57      6     25      0
            614      2    587    149    353      1   4257      0     37      0      0
              0      0      1      0      0     19      1   5889      2     88      0
             15      1     25     12     11      2     31      8   5895      0      0
              0      0      0      0      0     11      1    136      2   5850      0
>> clsResTest = annclsOld(tstv, networkHistory{ep});
>> cfmxTest = confMx(tstl, clsResTest)
        cfmxTest =
            838      1    35    41     5      1    69      0    10      0      0
              4    965     2    19     5      0     3      0     2      0      0
             11      1   828     9    99      1    50      0     1      0      0
             21      6    22   899    22      0    26      0     4      0      0
              0      0    96    38   813      0    52      0     1      0      0
              0      0     0     0     0    947     0     30     2     21      0
            140      0   140    39    83      0   585      0    13      0      0
              0      0     0     0     0     14     0    965     0     21      0
              1      0    10     5     1      2     9      5   967      0      0
              0      0     0     0     0     12     1     34     0    953      0
>> cfmxTest - (cfmxTrain ./ 6)
    ans = % różnica pomiędzy ilością błędów w zbiorze testowym a treningowym (większe
    ↪  wartości poza przekątną - gorsza klasyfikacja testowego); znormalizowane do różnicy
    ↪  per 1000 próbek
       -47.3     0.8     8.5     3.5     2.7     0.3    27.3     0.0     4.3    -0.2     0.0
         2.0   -17.8     0.0     9.5     4.0    -0.2     0.5     0.0     2.0     0.0     0.0
         5.2     1.0   -49.5     2.5    21.2     0.8    18.8     0.0     0.0     0.0     0.0
        11.7     4.3    13.7   -47.2     3.3     0.0    11.7     0.0     2.7    -0.2     0.0
        -1.5    -0.7    35.3    14.0   -60.2    -0.2    13.3     0.0    -0.2     0.0     0.0
         0.0     0.0    -0.2     0.0     0.0   -38.2     0.0    20.5     1.0    16.8     0.0
        37.7    -0.3    42.2    14.2    24.2    -0.2  -124.5     0.0     6.8     0.0     0.0
         0.0     0.0    -0.2     0.0     0.0    10.8    -0.2   -16.5    -0.3     6.3     0.0
        -1.5    -0.2     5.8     3.0    -0.8     1.7     3.8     3.7   -15.5     0.0     0.0
         0.0     0.0     0.0     0.0     0.0    10.2     0.8    11.3    -0.3   -22.0     0.0
```

Najwięcej błędów wystąpiło podczas klasyfikacji klasy 7. (aż 41.5% elementów tej klasy zostało błędnie sklasyfikowanych), a najwięcej pomyłek nastąpiło z klasą 1. i 3. (po 14% w każdej).

Ponadto - na zbiorze testowym klasa 7 była klasyfikowana gorzej o 12.45 p.p. Przeuczenie sieci nie jest specjalnie widoczne - różnica w klasyfikacji zbioru testowego i treningowego sugeruje raczej, że klasa 7. jest zwyczajnie mało charakterystyczna i podobna do wielu innych klas (głównie do 1., 3., 4., 5.).

2 Poprawki z artykułu

Zastosowałem kilka poprawek z artykułu, które wydały się najbardziej obiecujące.

2.1 *Shuffling*

Zastosowałem zwyczajne losowe przemieszanie próbek zbioru uczącego:

```
M = rows(tvec);
idx = randperm(M);
```

```
tvec = tvec(idx,:);
tlab = tlab(idx,:);
```

2.2 *Normalizing the Inputs*

Wejścia zostały znormalizowane względem wartości ze zbioru uczącego.

```
mn = mean(tvec);
sd = std(tvec);
sd(sd==0) = 1;
tvec = bsxfun(@minus,tvec,mn);
tvec = bsxfun(@rdivide,tvec,sd);
tstv = bsxfun(@minus,tstv,mn);
tstv = bsxfun(@rdivide,tstv,sd);
```

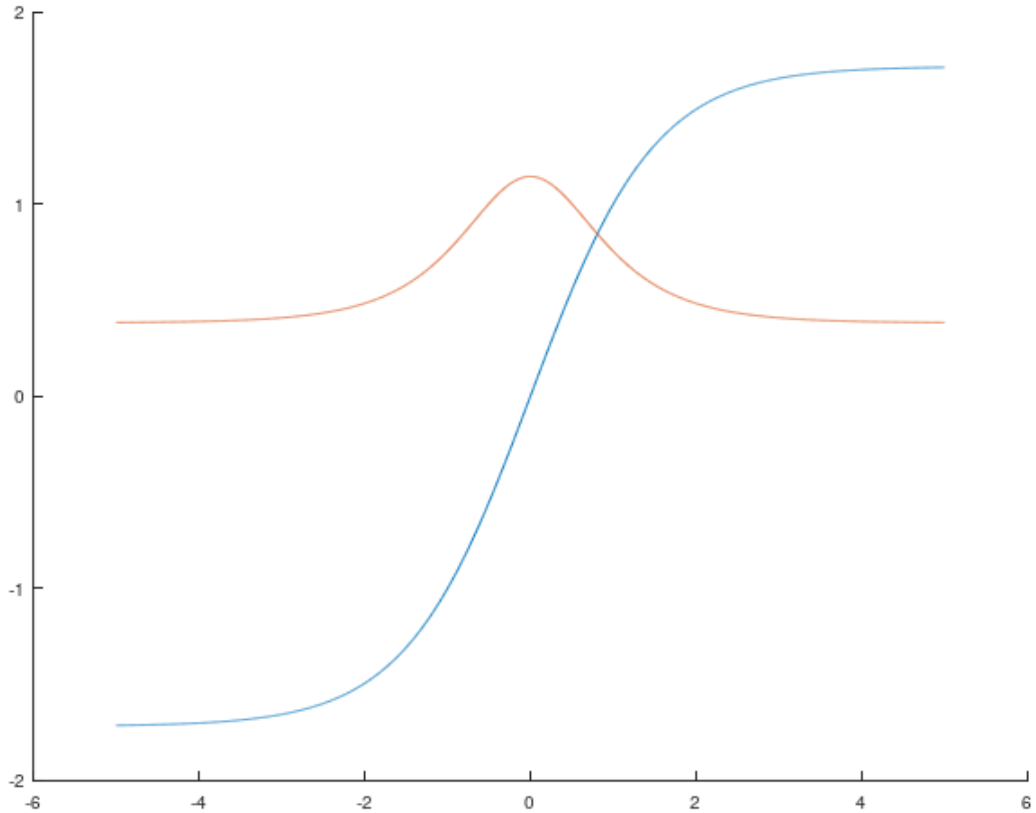
2.3 *The sigmoid, Target values*

Funkcje aktywacji i jej pochodnej zostały zamienione na te sugerowane w artykule. Ponadto w programie *backprop.m* zostały zaktualizowane wartości oczekiwane (zmienione na {*-1, 1*}) zgodnie z nową funkcją.

Ponadto nowa funkcja aktywacji mocno wpłynęła na proces uczenia (w zakresie (-1, 1) jest praktycznie liniowa), przez co stała uczenia również została zmodyfikowana (0.1 powodowało całkowitą destabilizację, najbardziej obiecującą wartością okazało się 0.0005).

```
function res = actf(tact)
        res = 1.7159 * tanh(2/3 * tact) +
↪  0.0001*tact;
end
```

```
function res = actdf(tact)
        res = 1.14393 * (sech(2/3 * tact)).^2
↪  + 0.0001;
end
```



Rysunek 1: Wykres nowej funkcji aktywacji (niebieska) i jej pochodnej (czerwona).

2.4 Wyniki

Maksymalna dokładność wyniosła **88.28%** dla zbioru testującego przy jednoczesnej poprawności dla zbioru uczącego 93.74% w 22 epoce. Daje to poprawę o **0.68 p.p.** (0.78% względem starej wartości) dla zbioru testowego.

```
>> [networkHistory trrep] = ann_training(0.0005, [100 100], 50);
1.0000000000      165.8724470139       0.0000104808       0.8531333333       0.8377000000
2.0000000000      168.1958780289       0.0000076952       0.8698333333       0.8518000000
3.0000000000      167.7478060722       0.0000069068       0.8790333333       0.8586000000
4.0000000000      168.1000859737       0.0000064345       0.8860333333       0.8642000000
5.0000000000      167.2721951008       0.0000061005       0.8919833333       0.8663000000
6.0000000000      167.6253159046       0.0000058331       0.8967333333       0.8701000000
7.0000000000      168.0562880039       0.0000056040       0.9011333333       0.8719000000
8.0000000000      192.7898459435       0.0000054008       0.9041500000       0.8733000000
9.0000000000      191.9035689831       0.0000052152       0.9068166667       0.8758000000
10.0000000000     192.0988531113       0.0000050420       0.9098166667       0.8765000000
11.0000000000     167.9884738922       0.0000048792       0.9125500000       0.8764000000
12.0000000000     167.8665699959       0.0000047251       0.9151666667       0.8771000000
13.0000000000     167.4276819229       0.0000045783       0.9175500000       0.8773000000
14.0000000000     168.3704679012       0.0000044383       0.9197666667       0.8777000000
15.0000000000     167.4815249443       0.0000043045       0.9220500000       0.8782000000
16.0000000000     167.3235931396       0.0000041768       0.9243000000       0.8785000000
17.0000000000     167.3794181347       0.0000040547       0.9271000000       0.8788000000
18.0000000000     167.9950377941       0.0000039378       0.9298833333       0.8790000000
19.0000000000     166.9083158970       0.0000038257       0.9318333333       0.8802000000
20.0000000000     167.0793540478       0.0000037182       0.9337500000       0.8809000000
21.0000000000     168.2310860157       0.0000036150       0.9354833333       0.8820000000
22.0000000000     169.0779151917       0.0000035160       0.9374166667       0.8828000000 %
↪  max
23.0000000000     167.9284381866       0.0000034202       0.9396666667       0.8828000000
24.0000000000     167.1154417992       0.0000033279       0.9413333333       0.8819000000
25.0000000000     167.6396069527       0.0000032386       0.9428166667       0.8822000000
26.0000000000     166.9714648724       0.0000031522       0.9445500000       0.8825000000
27.0000000000     167.1393561363       0.0000030682       0.9461166667       0.8825000000
28.0000000000     168.4908540249       0.0000029872       0.9472000000       0.8823000000
29.0000000000     167.5231142044       0.0000029092       0.9486333333       0.8820000000
30.0000000000     167.3776700497       0.0000028343       0.9498333333       0.8816000000
31.0000000000     167.3946719170       0.0000027618       0.9507666667       0.8802000000
32.0000000000     168.6968829632       0.0000026918       0.9516833333       0.8800000000
33.0000000000     167.6593780518       0.0000026244       0.9526166667       0.8796000000
34.0000000000     167.5037500858       0.0000025593       0.9534666667       0.8790000000
35.0000000000     168.1177380085       0.0000024963       0.9543000000       0.8786000000
36.0000000000     167.8174290657       0.0000024352       0.9552000000       0.8786000000
37.0000000000     167.7404749393       0.0000023761       0.9559000000       0.8781000000
38.0000000000     168.6633160114       0.0000023190       0.9566666667       0.8778000000
39.0000000000     168.5381288528       0.0000022630       0.9574166667       0.8774000000
40.0000000000     167.7314989567       0.0000022091       0.9579666667       0.8778000000
41.0000000000   26065.4816408157       0.0000021575       0.9585500000       0.8773000000 %
↪  anomalia czasu trwania spowodowana uśpieniem komputera
42.0000000000     168.8981740475       0.0000021079       0.9596500000       0.8773000000
43.0000000000     170.5344560146       0.0000020624       0.9600833333       0.8776000000
44.0000000000     168.4020500183       0.0000020179       0.9608500000       0.8775000000
45.0000000000     168.0848789215       0.0000019748       0.9616666667       0.8765000000
46.0000000000     168.4456710815       0.0000019324       0.9627166667       0.8765000000
47.0000000000     171.2700810432       0.0000018932       0.9631333333       0.8769000000
48.0000000000     162.6052420139       0.0000018547       0.9636500000       0.8762000000
49.0000000000     156.1211490631       0.0000018177       0.9645666667       0.8756000000
50.0000000000     157.1613371372       0.0000017794       0.9653500000       0.8754000000
```

Macierze błędów dla zbioru testowego i uczącego dla 22. epoki:

```
>> [_ ep] = max(trrep(:, 5));
>> clsResTrain = anncls(tvec, networkHistory{ep});
>> cfmxTrain = confMx(tlab, clsResTrain)
    cfmxTrain =
       5683      2     40     59     16      1    181      1     17      0      0
         16   5866      7     79      8      2     19      0      2      1      0
         92      2   5320     42    339      0    193      1     11      0      0
         76     10     27   5742     80      0     60      1      4      0      0
         12      3    258    159   5270      0    290      1      7      0      0
          3      0      2      0      0   5892      0     92      4      7      0
        584      5    285     86    156      0   4870      0     13      1      0
          0      0      0      1      0     17      0   5874      4    104      0
         23      0     20     21     10      1     43     14   5867      1      0
          0      0      0      0      0      7      0    131      1   5861      0
>> clsResTest = anncls(tstv, networkHistory{ep});
>> cfmxTest = confMx(tstl, clsResTest)
    cfmxTest =
        869     0    13    21     4     0    85     1     6     1     0
          6   957     1    24     3     0     9     0     0     0     0
         24     0   807    12    80     0    77     0     0     0     0
         28     3    12   911    18     1    23     0     4     0     0
          3     1    85    41   799     0    66     0     5     0     0
          0     0     1     1     0   936     0    39     2    21     0
        149     0    91    25    57     0   669     1     8     0     0
          0     0     1     0     0    13     0   959     0    27     0
          4     0     8     4     5     5    15     4   955     0     0
          0     0     0     0     0     7     1    32     0   960     0
>> cfmxTest - (cfmxTrain ./ 6)
    ans = % różnica pomiędzy ilością błędów w zbiorze testowym a treningowym (większe
    ↪  wartości poza przekątną - gorsza klasyfikacja testowego); znormalizowane do różnicy
    ↪  per 1000 próbek
      -78.2    -0.3     6.3    11.2     1.3    -0.2    54.8     0.8     3.2     1.0     0.0
        3.3   -20.7    -0.2    10.8     1.7    -0.3     5.8     0.0    -0.3    -0.2     0.0
        8.7    -0.3   -79.7     5.0    23.5     0.0    44.8    -0.2    -1.8     0.0     0.0
       15.3     1.3     7.5   -46.0     4.7     1.0    13.0    -0.2     3.3     0.0     0.0
        1.0     0.5    42.0    14.5   -79.3     0.0    17.7    -0.2     3.8     0.0     0.0
       -0.5     0.0     0.7     1.0     0.0   -46.0     0.0    23.7     1.3    19.8     0.0
       51.7    -0.8    43.5    10.7    31.0     0.0  -142.7     1.0     5.8    -0.2     0.0
        0.0     0.0     1.0    -0.2     0.0    10.2     0.0   -20.0    -0.7     9.7     0.0
        0.2     0.0     4.7     0.5     3.3     4.8     7.8     1.7   -22.8    -0.2     0.0
        0.0     0.0     0.0     0.0     0.0     5.8     1.0    10.2    -0.2   -16.8     0.0
```

Najwięcej błędów wystąpiło podczas klasyfikacji klasy 7. (aż 33.1% elementów tej klasy zostało błędnie sklasyfikowanych vs. 41.5% w implementacji referencyjnej), a najwięcej pomyłek nastąpiło z klasą 1 (14.9%) oraz 3. (9.1%).

Ponadto - na zbiorze testowym klasa 7 była klasyfikowana gorzej o 14.2 p.p. (co sugeruje przeuczenie sieci, która próbowała się dostosować do bardzo trudnej i niecharakterystycznej klasy 7). Ten argument popiera też to, że dużo klas (1, 3, 5) były błędnie klasyfikowane jako 7.

Kolejną rzeczą jest różnica w skali pomyłek pomiędzy zbiorem uczącym a testowym - tutaj jeszcze silniej objawił się efekt źle ocenianej klasy 7. Ponadto - różnica w poprawności na zbiorze treningowym i testowym wynosiła w implementacji referencyjnej 12.45 p.p., co jest wartością niższą niż osiągnięta dla poprawionej wersji sieci (a przecież poprawność oceny klasy 7 wzrosła o 8.4 p.p.).

Sposobem na poprawienie działania sieci (poza dalszymi optymalizacjami jej działania, dobierania współczynników), byłoby częstsze odrzucanie decyzji klasyfikacyjnej, która wskazywałaby na klasę 7. z niskim pobudzeniem.