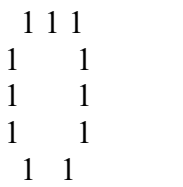
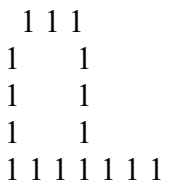
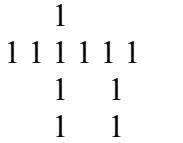
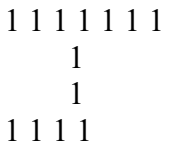
Sprawozdanie – Piotr Tutak
Scenariusz 2 – zagadnienie 1:

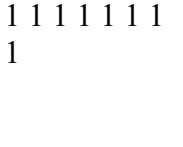
Użyte litery:
c d f h l n q r s y B C D E I K N T Y Z

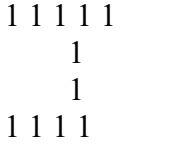


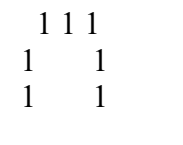












```
 1       1
1 1 1 1 1 1
```

```
1       1
1 1 1 1
1       1
      1
    1
```

```
1     1
1   1   1
1   1   1
1   1   1
 1
```

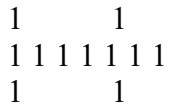
```
1     1 1 1
1   1
 1 1
   1 1 1 1
```

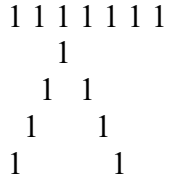
```
1 1 1 1 1 1 1
1     1     1
1     1     1
1     1     1
 1 1   1 1
```

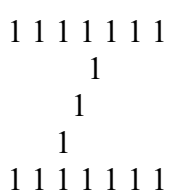
```
 1 1 1 1 1
1         1
1         1
1         1
 1       1
```

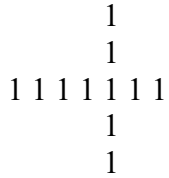
```
1 1 1 1 1 1 1
1         1
1         1
1         1
 1 1 1 1 1
```

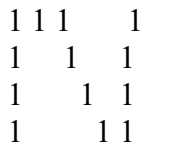
```
1 1 1 1 1 1 1
1     1     1
1     1     1
1     1     1
1         1
```









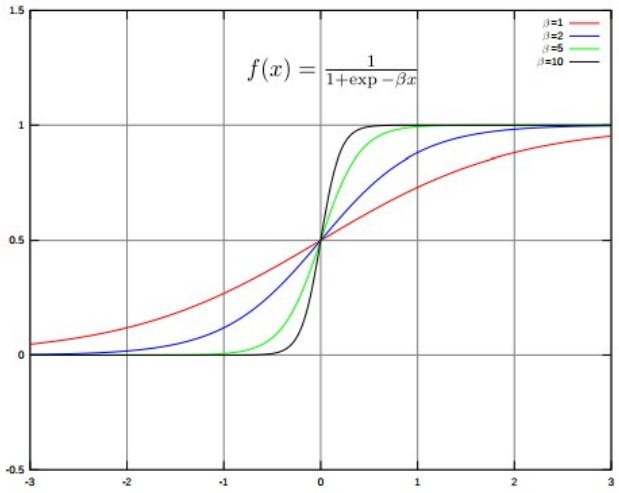


Opis budowy:

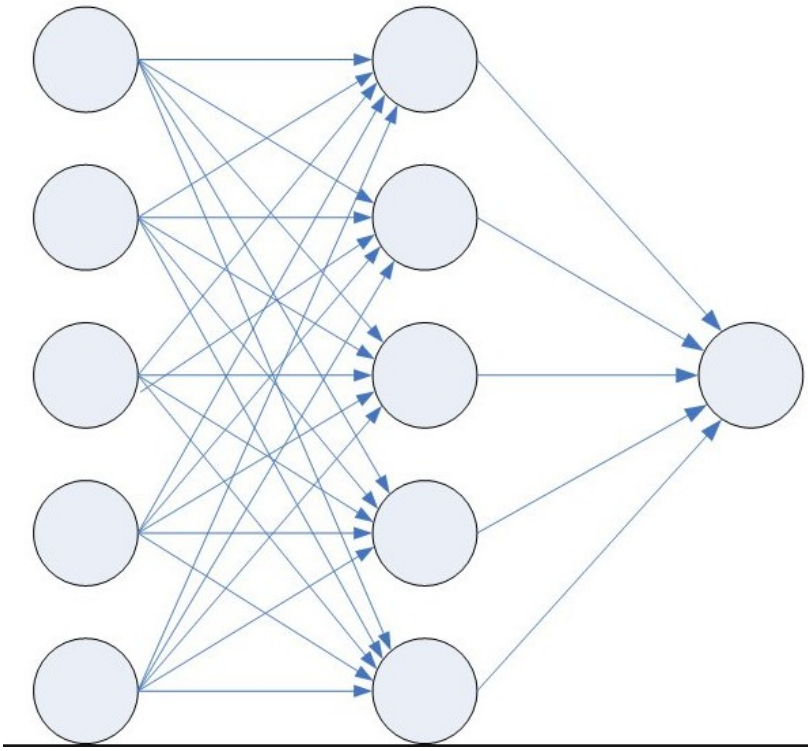
Sieć neuronowa składała się z 15 perceptronów z użytą funkcją aktywacji sigmoidalną ze współczynnikiem beta=1.0:

- Symetryczna sigmoida ($\lim_{x\to-\infty}\phi = -1$)

$$\phi(s) = \frac{1 - \exp(-\beta s)}{1 + \exp(-\beta s)}$$



Rysunek 1.4: Funkcja sigmoidalna z parametrami $\beta = 1$, $\beta = 3$, $\beta = 10$.



Pierwsza warstwa perceptronów służyła jedynie do „wprzęgnięcia” wejścia, nie brała udziału w uczeniu, również ostatnia warstwa była jedynie wyjściem. Łącznie perceptronów w ukrytej warstwie było 15. Perceptronów wejściowych było 35.
Ostatni neuron był dwojakiego rodzaju, był to neuron z funkcją hard one, czyli:

$$f(x) = \begin{cases} 1 \; dla \; x \geq 0 \\ 0 \; dla \; x < 0 \end{cases}$$

Lub z funkcją identycznościową.

Wagi ostatniego neuronu były równe 1.0, współczynnik uczenia był równy 0.0.
Pierwsze i ostatni neuron nie brały udziały w uczeniu sieci.

Algorytm uczenia odbywał się zgodnie ze wzorem:

$w_1 \mathrel{+}= \eta * (d-y) * x_1$
$w_2 \mathrel{+}= \eta * (d-y) * x_2$

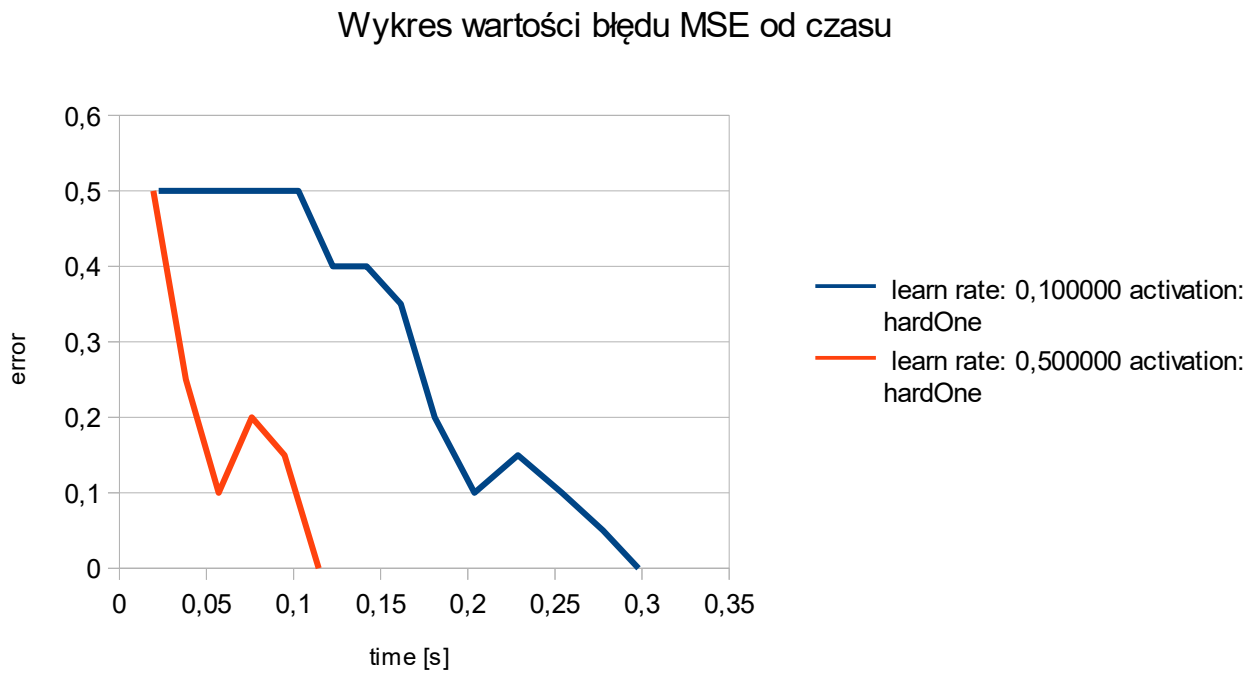
...
$\theta \mathrel{+}= \eta * (d-y)$

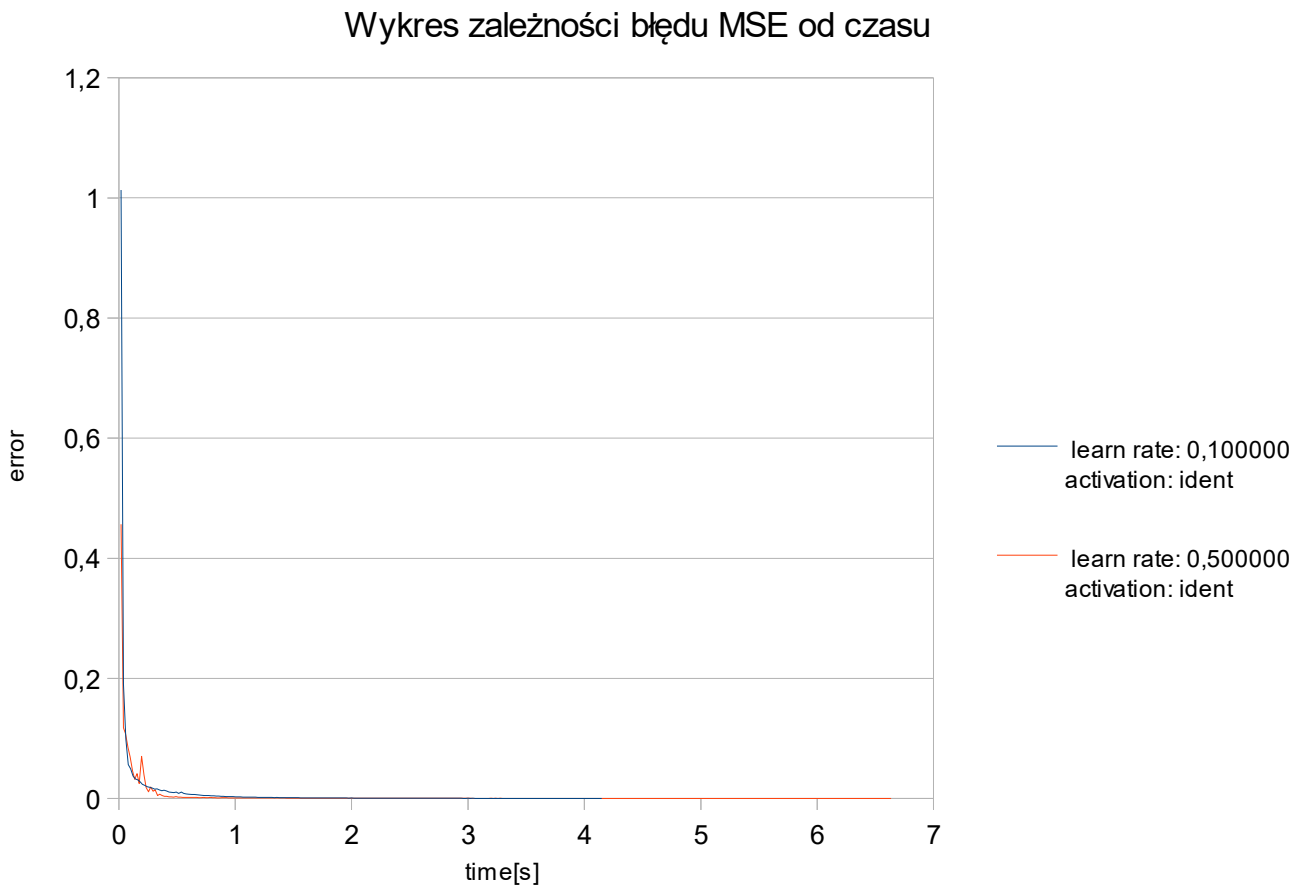
gdzie:

- $w_1, w_{2...}$ – wagi
- $x_1, x_{2...}$ – wartości wejściowe
- η – niewielki współczynnik uczenia ($n > 0$)
- d – oczekiwana odpowiedź
- y – odpowiedź neuronu
- θ – próg wzbudzenia neuronu

Testowane były w sumie 4 sieci neuronowe, po 2 dla współczynnika uczenia 0.1 i po 2 dla współczynnika uczenia 0.5 oraz po 2 dla funkcji końcowej hardOne, i po 2 dla funkcji identycznościowej. W przypadku funkcji identycznościowej uczenie dążyło by sieć zwracała albo wartość najbardziej zbliżoną do 1.0 albo do 0.0 w zależności od podawanej litery. Uczenie było przerywane przy wartości całkowitego błędu MSE dla wszystkich liter mniejszego od 0.0001

Wyniki:


Wykres wartości błędu MSE od czasu
0,6
0,5
0,4
0,3
0,2
0,1
0
error
0
0,05
0,1
0,15
0,2
0,25
0,3
0,35
time [s]
learn rate: 0,100000 activation: hardOne
learn rate: 0,500000 activation: hardOne


Wykres zależności błędu MSE od czasu
1,2
1
0,8
0,6
0,4
0,2
0
error
0
1
2
3
4
5
6
7
time[s]
learn rate: 0,100000 activation: ident
learn rate: 0,500000 activation: ident

Testowanie sieci:
Dane zaszumione w liczbie 5 pikseli

```
1 1
1   1 1 1
1   1   1   1
1 1 1   1
1 1 1 1     1
```

```
  1 1 1
1       1 1
1       1   1
1     1 1 1
1 1 1   1 1 1
```

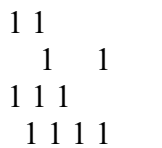
```
1 1   1 1 1
        1   1
      1 1
1 1 1 1
          1
```

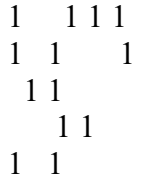
```
  1 1       1

1 1 1 1 1   1
1   1       1
```

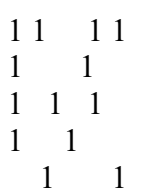
```
            1
    1 1 1
1   1 1 1 1 1
1
```

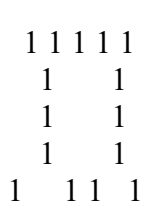
```
  1   1
1       1   1
1 1     1
1       1
  1 1 1 1   1
```

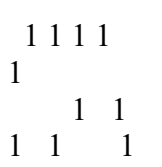
```
1 1 1 1 1 1
          1
1 1       1

  1 1 1 1

  1   1 1   1
```

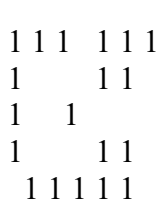


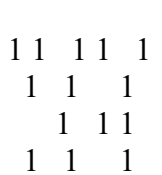


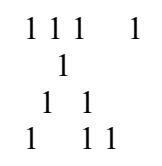












```
1   1       1


1 1   1 1 1
1   1       1
1
1           1
   1 1 1 1


 1     1 1
     1     1
1     1 1   1
1         1 1
 1 1 1   1


1 1   1   1
1       1
 1 1 1 1
1
 1 1 1 1 1 1


1   1     1 1
1   1   1
1     1
   1   1
1 1 1     1 1


       1 1 1
1     1     1
 1 1       1

       1 1 1
```

Multilayer:
learnRate:0.10000
activFunc:hardOne
iter number: 14 ; time taken[s]:0.2980411653328474
errors:
(letter, result, expected):
('h', 1.0, 0.0)
('y', 1.0, 0.0)
('C', 0.0, 1.0)
('W', 0.0, 1.0)
number of errors: 4
error value MSE: 0.2

Multilayer:
learnRate:0.50000
activFunc:hardOne
iter number: 6 ; time taken[s]:0.11453784536843159
errors:
(letter, result, expected):
('h', 1.0, 0.0)
('y', 1.0, 0.0)
('W', 0.0, 1.0)
number of errors: 3
error value MSE: 0.15

Multilayer:
learnRate:0.10000
activFunc:ident
iter number: 201 ; time taken[s]:4.147680291568122
errors:
(letter, result, expected):
('a', 0.4003029132427893, 0.0)
('d', -0.0705466487190698, 0.0)
('h', 0.4249609164157855, 0.0)
('i', 0.9666831097365207, 0.0)
('l', 1.3219890469741231, 0.0)
('o', 0.2558906662628524, 0.0)
('p', -0.198042866868412, 0.0)
('w', 0.3341675070257659, 0.0)
('y', -0.05393175824373353, 0.0)
('z', 0.01715547055474631, 0.0)
('A', 1.2286040470108994, 1.0)
('C', 0.8191240746979747, 1.0)
('D', 1.1269092496421913, 1.0)
('F', 1.1820399676636804, 1.0)
('K', 0.8204157734561974, 1.0)
('O', 0.7876219049752287, 1.0)
('S', 1.1447620792745914, 1.0)
('W', 0.3743409644100484, 1.0)
('X', 1.1634058172152315, 1.0)
('Y', 1.333845963136409, 1.0)
number of errors: 20
error value MSE: 0.20048247147660062

Multilayer:
learnRate:0.50000
activFunc:ident
iter number: 336 ; time taken[s]:6.635513787529703
errors:
(letter, result, expected):
('a', 0.7257317199848472, 0.0)
('d', -0.12398614084171711, 0.0)
('h', 0.6381558490452499, 0.0)

('i', 0.9421650016375748, 0.0)
('l', 0.8483597710343757, 0.0)
('o', -0.16071214628551567, 0.0)
('p', 0.27503213663414994, 0.0)
('w', 0.1512795117896355, 0.0)
('y', 0.5891148212548346, 0.0)
('z', 0.6114580244965981, 0.0)
('A', 1.038346055271147, 1.0)
('C', 0.8251904125100259, 1.0)
('D', 1.0046504620890355, 1.0)
('F', 1.1541203117080607, 1.0)
('K', 0.934210322829342, 1.0)
('O', 0.9392473891438692, 1.0)
('S', 1.0090556078334254, 1.0)
('W', 0.8251102425317566, 1.0)
('X', 0.9762899350559486, 1.0)
('Y', 0.9814794670483669, 1.0)
number of errors: 20
error value MSE: 0.17486905999303942

Analiza:

Jak widać sieć z funkcją aktywacji hardOne uczy się znacznie szybciej niż sieć z funkcją aktywacji sigmoidalną. Również zwiększenie współczynnika uczenia z 0.1 do 0.5 wpływa pozytywnie na szybkość uczenia sieci. Sieć z funkcją hardOne jest w stanie ostatecznie dobrze rozpoznać ponad 75% przypadków zaszumionych prawidłowo, natomiast sieć neuronowa z funkcją sigmoidalną nie rozpoznaje tych przypadków dokładnie a jedynie w sposób przybliżony.
Ostatecznie jednak wartość błędu w obu przypadkach sieci i dla obu współczynników uczenia jest bardzo zbliżona i plasuje się na poziomie 0.15-0.20 jeśli chodzi o błąd MSE.

Sieć neuronowa z funkcją progową potrzebuje ok 0.15[s] by nauczyć się wszystkich przypadków i rozpoznawać je bezbłędnie. Sieć z funkcją ciągłą potrzebuje nawet do 7s by rozpoznawać przypadki z maksymalnym błędem równym 0.0001 (MSE), co jest znaczącą różnicą jeśli chodzi o zysk czasowy uczenia sieci – ok. 50krotny – natomiast procent rozpoznawania danych zaszumionych pozostaje na podobnym poziomie w obu przypadkach.

Wnioski:

- Jak widać najbardziej na czas uczenia i ilość potrzebnych przebiegów ma wybrana funkcja zbierająca wyniki i obliczająca wartość błędu:
  - w przypadku użycia funkcji granicznej (hardOne) sieć uczy się bardzo szybko i osiąga bardzo dobre wyniki w testach przy obu współczynnikach uczenia, zarówno 0.1 jak i 0.5
  - w przypadku użycia zwykłej sumy i funkcji identycznościowej sieć uczy się znacznie wolniej
- Sieć z funkcją końcową hardOne bardzo dobrze radzi sobie z klasyfikacją liter zaszumionych, natomiast dla sieci z funkcją identycznościową jest to trudniejsze zadanie, choć całkowita wartość błędu MSE jest ostatecznie bardzo zbliżona w przypadku obu sieci
- Zwiększenie współczynnika uczenia wpłynęło pozytywnie na oba rodzaje sieci pod względem testów na danych zaszumionych. Natomiast dla sieci z funkcją końcową identycznościową znacznie wydłużyło czas uczenia, dla sieci z funkcją hardOne – skróciło.

Użyta sieć:

```
Multilayer:
    Layer(inputNumber:1, perceptronNumber:35, activFunc:hardOne, activFuncDeriv:zero, learnRate:0.00000)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)
        Perceptron(weights:[ 1.00000],bias:-0.50000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:zero)

    Layer(inputNumber:35, perceptronNumber:15, activFunc:signSigm(1.000), activFuncDeriv:signSigmDeriv(1.000),
learnRate:0.10000 or 0.50000)
        Perceptron(weights:[ 0.26928, 0.47171,-0.05028, 0.53446, 0.35564, 0.43071,-0.08592,-0.04487, 0.30404,-
0.03695, 0.74482, 0.51806, 0.32947, 0.27136, 0.34840, 0.61350, 0.89527, 0.68548, 0.68858, 0.80617, 0.50709,
0.63335, 0.01967, 0.52462, 0.58060, 0.46207, 0.78445, 0.53450, 0.54260, 0.87067, 0.59758, 0.45320, 0.78246,
0.45270, 0.69962],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))
        Perceptron(weights:[ 0.47296, 0.40760, 0.41446, 0.66916, 0.88234, 0.17341, 0.50630, 0.14931, 0.60761,
0.63388,-0.01685, 0.37176, 0.46359, 0.27394, 0.65958,-0.09879, 0.58900, 0.46092, 0.61665, 0.21694, 0.14834,
0.40678, 0.06988, 0.39908, 0.52846, 0.53784, 0.15097, 0.10613, 0.37145, 0.63362, 0.09567, 0.16386, 0.23138,
0.27838, 0.05637],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))
        Perceptron(weights:[ 0.66682, 0.56927, 0.20252, 0.44448, 0.17188, 0.38047, 0.02411, 0.67469, 0.16549, 0.11777,
0.71727, 0.52862, 0.83467, 0.02065, 0.45466, 0.67310, 0.21865, 0.58927, 0.45410, 0.24048, 0.80033, 0.87806,
0.27702, 0.15158, 0.20151, 0.57086, 0.53620, 0.12908, 0.47518, 0.53320, 0.29060, 0.25854, 0.17780, 0.54619,
0.39209],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))
        Perceptron(weights:[ 0.10542, 0.07674, 0.06210, 0.46111, 0.46845, 0.36920,-0.08161, 0.11412, 0.84573, 0.31813,
0.02558, 0.52174, 0.17791, 0.59835, 0.76412, 0.00779, 0.54646,-0.02385, 0.37480, 0.54650, 0.54510, 0.44242,-
0.01535, 0.62962, 0.61960, 0.58061, 0.51062, 0.39925, 0.32894, 0.10734, 0.12116, 0.11953, 0.72100, 0.61240,-
0.01731],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))
        Perceptron(weights:[ 0.55929, 0.19702, 0.69074, 0.70197, 0.34683, 0.46725,-0.06530, 0.33487, 0.00729, 0.22335,
0.05247, 0.36323, 0.54757, 0.32954, 0.25776, 0.46357, 0.20992, 0.56626, 0.56381, 0.60283, 0.56874,-0.01359,
0.24210, 0.33736, 0.48661, 0.69776, 0.27684, 0.66737, 0.10313, 0.17909, 0.02002, 0.34999, 0.41943, 0.70063,
0.11012],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))
```

Perceptron(weights:[0.41971, 0.88702, 0.40966, 0.19770, 0.32976, 0.84578, 0.43859, 0.33020, 0.61625, 0.76607, 0.58785, 0.62340, 0.40442, 0.80887, 0.10292, 0.54376, 0.15530, 0.53487, 0.16066, 0.50325, 0.27430, 0.41400, 0.69553, 0.42108, 0.45592, 0.36584,-0.09547, 0.40517, 0.54755, 0.79923, 0.39672, 0.49726, 0.66151, 0.48122, 0.77263],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))

Perceptron(weights:[0.43871, 0.39380, 0.44571, 0.32012, 0.44937, 0.42731, 0.29211, 0.78879, 0.53685, 0.33691, 0.71153, 0.82298, 0.03724, 0.14652, 0.65097, 0.07829, 0.37188, 0.69802, 0.18192, 0.44675, 0.44993, 0.25178, 0.40501, 0.87293,-0.09003, 0.65962, 0.50981, 0.45167, 0.37036, 0.29617,-0.09205,-0.06841, 0.02751, 0.32266, 0.69456],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))

Perceptron(weights:[0.58740, 0.46967, 0.76585, 0.30557, 0.61193, 0.24260, 0.53565, 0.13956, 0.35912, 0.15549, 0.80632, 0.31711, 0.53110, 0.60966, 0.69272, 0.65411, 0.11553, 0.57990, 0.89998, 0.24193, 0.11648, 0.04221, 0.11433, 0.67785, 0.33605, 0.45848, 0.85537, 0.82075, 0.13886, 0.73275,-0.06366, 0.69217, 0.56413, 0.73300, 0.73863],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))

Perceptron(weights:[0.69036, 0.72290, 0.80859, 0.14907, 0.06003, 0.54275, 0.66674, 0.34913, 0.71693, 0.24548, 0.43617, 0.56011,-0.07397, 0.37389, 0.51001, 0.49779, 0.34339, 0.58519, 0.65757, 0.31347, 0.02250, 0.33408, 0.75345, 0.04484, 0.21513, 0.37272, 0.23030, 0.60043, 0.86723,-0.09276,-0.01826, 0.06949, 0.46738, 0.87370, 0.44459],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))

Perceptron(weights:[0.33890, 0.39709, 0.21708, 0.80742, 0.37789, 0.53115, 0.40360, 0.22832, 0.58172, 0.69949, 0.19768, 0.14247, 0.11534, 0.87104, 0.39576, 0.49242, 0.08769, 0.34599, 0.59007,-0.00253, 0.12450, 0.52743, 0.54953, 0.82678, 0.54162, 0.58454,-0.08552, 0.19365, 0.18150, 0.29599, 0.61460, 0.40497, 0.31029, 0.06644, 0.51998],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))

Perceptron(weights:[0.39178,-0.03502, 0.31493, 0.70809, 0.72055, 0.20872, 0.84911, 0.48142, 0.23622, 0.75280, 0.27212,-0.06354, 0.41919, 0.69369, 0.46894, 0.49077, 0.30816, 0.43258, 0.14590, 0.53129, 0.00903, 0.49146, 0.00444,-0.00425, 0.37452, 0.18276, 0.10754, 0.37232, 0.50623, 0.48598, 0.54841, 0.22722, 0.25636, 0.47142, 0.42938],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))

Perceptron(weights:[0.48094,-0.00479, 0.51667, 0.27046, 0.40603, 0.02767, 0.05975, 0.68226, 0.57310, 0.80570, 0.29536, 0.26770, 0.82653, 0.20734, 0.81729, 0.89375, 0.44354, 0.72123, 0.67369, 0.25213, 0.41401, 0.78174, 0.51586,-0.00217, 0.57164, 0.34430, 0.38046, 0.67067, 0.84532, 0.36054, 0.59864, 0.84984, 0.34213, 0.82002, 0.41749],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))

Perceptron(weights:[0.67687, 0.35784, 0.53619, 0.49974, 0.09618, 0.23936, 0.11645, 0.53015, 0.52998, 0.37464,-0.06536, 0.06022, 0.58508, 0.57802, 0.15645, 0.47753, 0.11059, 0.20298, 0.86913, 0.33099, 0.40552, 0.30734, 0.61961, 0.05526, 0.43911, 0.67211, 0.43692, 0.61044, 0.50294, 0.31203, 0.33174,-0.09629, 0.46809, 0.67428, 0.52284],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))

Perceptron(weights:[-0.00816, 0.80433, 0.55585, 0.16349, 0.42838, 0.32982, 0.08756, 0.25956, 0.01074, 0.64941, 0.15871, 0.56169, 0.84784, 0.83748, 0.16358, 0.55715, 0.24708, 0.45867, 0.00318, 0.40684, 0.25625, 0.74984, 0.41258, 0.47078, 0.37182, 0.59524, 0.16693, 0.39714, 0.36841, 0.52461, 0.41698, 0.02624, 0.38357, 0.31631, 0.62249],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))

Perceptron(weights:[0.57968, 0.39193, 0.51424, 0.54423, 0.56927, 0.27404, 0.67974, 0.41023, 0.35466, 0.07853, 0.39482, 0.22319, 0.36074, 0.61903, 0.11176, 0.60789,-0.00130, 0.25307, 0.63136, 0.57944, 0.46828, 0.56321, 0.54962, 0.42840, 0.78501, 0.07578, 0.11028, 0.49377,-0.07297, 0.82753, 0.53444, 0.64362, 0.28032, 0.49553, 0.32910],bias:-0.05000,learnRate:0.10000,activFunc:signSigm(1.000),activFuncDeriv:signSigmDeriv(1.000))

Layer(inputNumber:15, perceptronNumber:1, activFunc: ident or hardOne, activFuncDeriv:one, learnRate:0.00000)

Perceptron(weights:[1.00000, 1.00000, 1.00000, 1.00000, 1.00000, 1.00000, 1.00000, 1.00000, 1.00000, 1.00000, 1.00000, 1.00000, 1.00000, 1.00000, 1.00000],bias:
0.00000,learnRate:0.00000,activFunc:hardOne,activFuncDeriv:one)

Listing kodu:

```
# -*- coding: utf-8 -*-
"""
Created on Sun Oct 22 11:20:52 2017

@author: PiotrTutak
"""

from perceptron import *
from operator import itemgetter
import random
import time
import sys
import copy
#funkcja wypisująca zawartosc listy z zadaną precyzją
def listWithPrec(listA,prec):
    ret="["
    formatStr="{0: "+str(int(prec+3))+"."+str(int(prec))+"f}"
    for x in listA:
        ret+=formatStr.format(x)
        ret+=","
    ret=ret[:-1]+']'
    return ret

#funkcje liczace wartosci błędów
def MSE(results,expected):
    sum=0.0
    for i in range(len(results)):
        sum+=(results[i]-expected[i])**2
    return sum/len(results)

def MAPE(results,expected):
    sum=0.0
    for i in range(len(results)):
        sum+=abs((expected[i]-results[i])/expected[i])
    return 100*sum/len(results)

#przekierowanie wyjscia do pliku
STDOUT=sys.stdout
f=open('results.txt','w');
sys.stdout=f


literyLow=dict()
literyHigh=dict()
with open('litery.txt','r') as f:
    for l in f:
        x=l.strip().split('=')
        if x[0].islower():
            literyLow[x[0]]=[float(a) for a in x[1]]
        else:
            literyHigh[x[0]]=[float(a) for a in x[1]]

#utworzenie danych uczacych i testujacych
litery20=sorted(random.sample(list(literyLow.items()),10),key=itemgetter(0))
litery20.extend(sorted(random.sample(list(literyHigh.items()),10),key=itemgetter(0)))
litery20Expected=[[1.0] if x >9 else [0.0] for x in range(20)]
litery20ExpectedTest=[0.0 if x<=9 else 1.0 for x in range(20)]

litery20Stirred=copy.deepcopy(litery20)

#liczba blednych pikseli w literze
```

```
NUMBER_OF_STIRR=5

for x in litery20Stirred:
   change=random.sample(set(range(35)),NUMBER_OF_STIRR)
   for c in change:
      if x[1][c]==0.0:
         x[1][c]=1.0
      else:
         x[1][c]=0.0

print('użyte litery:')
print(*list(x[0] for x in litery20),sep=' ')
print('\n')

for x in litery20:
   for j in range(5):
      for i in range(7):
         if x[1][j*7+i]:
            print(str(int(x[1][j*7+i]))+' ',end='')
         else:
            print('  ',end='')
      print('')
   print('\n')


listPerc=[]
RES_NUMBER=4

#przyjeta liczba perceptronow w warstwie
HIDDEN_LAYER_PERCEP_NUMB=15
learnRate=0.5

multilayerOrig=Multilayer(
       [35,HIDDEN_LAYER_PERCEP_NUMB,1],
       [hardOne,SignSigm()(1.0),hardOne],
       [zero,SignSigm().derivative(1.0),one],
       [[1.0],None,[1.0 for x in range(HIDDEN_LAYER_PERCEP_NUMB)]],
       [0.0,0.1,0.0],
       [-0.5,-0.05,0.0]
       )

#uczenie poszczegolnych warstw
while(len(listPerc)<RES_NUMBER):
   multilayer=copy.deepcopy(multilayerOrig)
   print(multilayer)
   if len(listPerc)==1:
      multilayer[1]['learnRate']=0.5
   elif len(listPerc)==2:
      multilayer[2]['activFunc']=ident
   elif len(listPerc)==3:
      multilayer[2]['activFunc']=ident
      multilayer[1]['learnRate']=0.5

   i=0
   run=True
   start=time.clock()
   print('start learning:')
   print('iteration;  time  ;  error   ; learn rate: %f;'% (multilayer[1]['learnRate']))
   while(run):
      samples=list(litery20)
      while(run and samples):
         inp=random.sample(samples,1).pop(0)
         samples.remove(inp)
```

```
            multilayer.learn(inp[1],litery20Expected[litery20.index(inp)])

        results=[]
        for inp in litery20:
            results.extend(multilayer.process(inp[1]))

        error=MSE(results,litery20ExpectedTest)
        if error<0.0001:
            run=False
        i+=1
        print("{0:9};{1: 8.5f}".format(i,time.clock()-start),error,sep=';')
    listPerc.append((multilayer,i,error,time.clock()-start))
    print("iter number:{0:8}".format(i),"; time taken[s]:{0:8}".format(time.clock()-start))
    print('')


print('\n\nTestowanie sieci:')
print('Dane zaszumione w liczbie %d pikseli' % NUMBER_OF_STIRR)

for x in litery20Stirred:
    for j in range(5):
        for i in range(7):
            if x[1][j*7+i]:
                print(str(int(x[1][j*7+i]))+' ',end='')
            else:
                print('  ',end='')
        print('')
    print('\n')

#testowanie warstw na danych zaszumionych
for m in listPerc:
    res=[]
    errorCompute=[]
    print(repr(m[0]),end='')
    print("iter number:{0:8}".format(m[1]),"; time taken[s]:{0:8}".format(m[3]))
    print('errors:\n(letter, result, expected):')
    for s in litery20Stirred:
        res.append(m[0].process(s[1]))
        errorCompute.extend(m[0].process(s[1]))
    res=[(x[0],*y,*z) for x,y,z in zip(litery20,res,litery20Expected) if z[0]!=y[0]]
    print(*res,sep='\n')
    print('number of errors:',len(res))
    print('error value MSE:',MSE(errorCompute,litery20ExpectedTest))
    print('\n')


f.close()
sys.stdout=STDOUT
```