

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»  
Факультет прикладної математики  
Кафедра прикладної математики

Звіт  
із лабораторної роботи №1  
з дисципліни «Системи нейронних мереж»  
на тему  
«>>

Виконав:  
студент групи КМ-02  
Пилипченко Б. О.

Керівник:  
Терейковський І. А.

## Мета роботи:

Частина 1

Завдання: розробити програмне забезпечення для реалізації класичного нейрону. Передбачити режим навчання на одному навчальному прикладі та режим розпізнавання.

Частина 3 Завдання: розробити програмне забезпечення для реалізації двошарового персептрону із структурою 2-3-1. Передбачити режим навчання «ON-LINE» та режим розпізнавання. Піддослідна функція  $x_1 + x_2 = y$

## Теоретичні відомості:

"Deep learning" Ian Goodfellow, Yoshua Bengio, Aaron Courville The MIT Press

# Теоретичні відомості

При використанні нейронних мереж прямого розповсюдження, що приймає вхід  $X$  і породжує вихід  $O$ , сигнал передається по мережі лише "вперед" - від  $i$ -того шару до  $i+1$ .  $X$  містить початкові дані, дані оброблюються кожним шаром - отримуємо вихід  $O$ . Цей процес називається прямим поширенням сигналу.

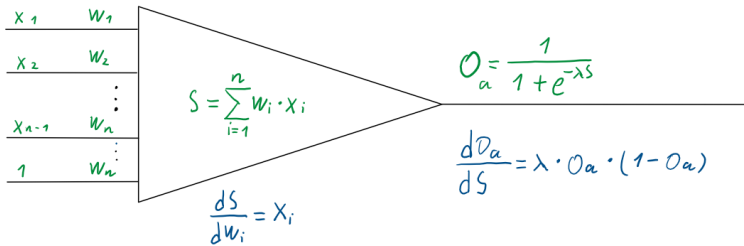
Під час навчання пряме поширення сигналу дозволяє отримати значення функції помилки мережі  $E(W)$  ( $W$  - вагові коефіцієнти мережі).

Алгоритм оберненого поширення обчислює градієнт  $E(W)$  для поточних значень коефіцієнтів  $W$ .

Бібліотеки, подібні до TensorFlow, розраховують градієнт  $E(W)$  будуючи граф обчислення похідних  $dE/dw$  функції помилки для кожного вагового коефіцієнту мережі.

В лабораторній роботі було вручну побудовано граф обчислення похідних  $dE/dw$  для мережі, що складається з 1 нейрону (частина 1) та для мережі з трьома шарами (частина 3).

На графах зеленим кольором позначено пряме розповсюдження сигналу, синім - обернене розповсюдження сигналу.



$$E(W) = \frac{1}{2} (O_a - O_t)^2$$

$$\frac{dE}{dO_a} = O_a - O_t$$

$E(W)$  - функція помилки нейрону від вагових коефіцієнтів

$\alpha$  - крок пошуку мінімуму  $E(W)$

$\lambda$  - параметр згладжування функції активації

$x_1 \dots x_n$  - вхідні сигнали

$W; w_1 \dots w_n$  - вагові коефіцієнти

$O_a$  - вихідний сигнал нейрону

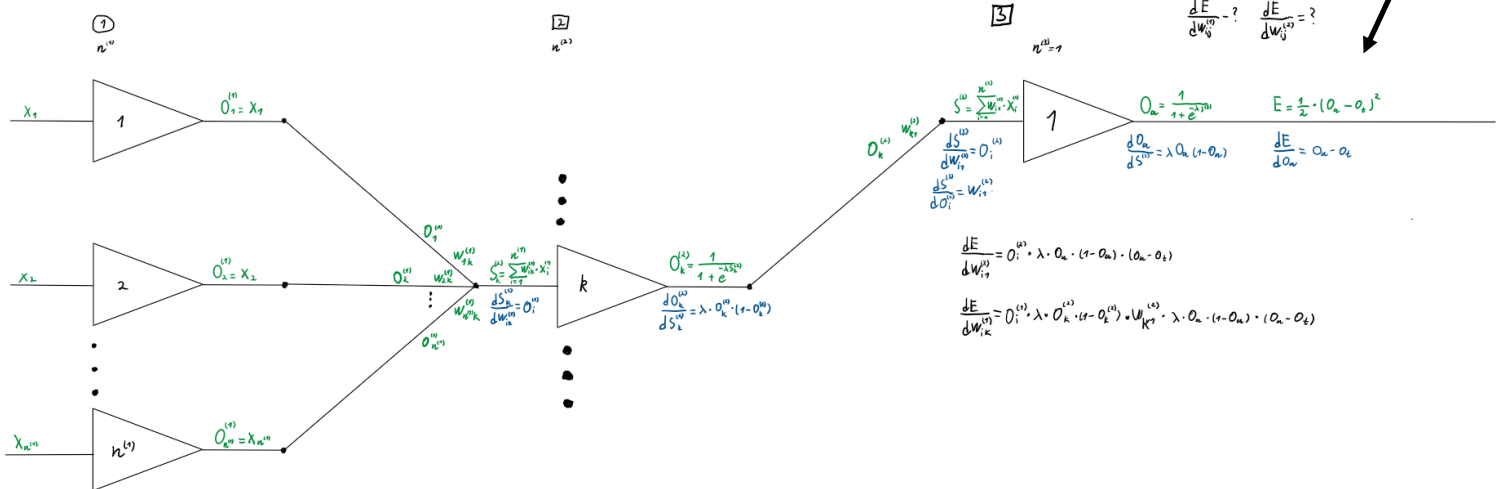
$$\frac{dE}{dw_i} = \frac{dE}{dO_a} \cdot \frac{dO_a}{dS} \cdot \frac{dS}{dw_i} = (O_a - O_t) \cdot \lambda \cdot O_a \cdot (1 - O_a) \cdot x_i$$

$$\vec{W}^{(n)} = \vec{W}^{(n-1)} - \alpha \frac{\vec{grad}(E)}{|\vec{grad}(E)|}$$

$$w_i^{(n)} = w_i^{(n-1)} - \alpha \frac{grad(E)}{|grad(E)|}$$

Граф обчислення градієнту  $E(W)$  для мережі, що складається з 1 нейрону (частина 1)

$O_t$  - очікуваний вихідний сигнал нейрону



Граф обчислення похідних для нейронної мережі з трьома шарами ( $n_1-n_2-1$ ) (частина 3):

# Реалізація

Обчислення, наведені в теоретичних відомостях, було реалізовано мовою C. Також було створено допоміжний скрипт мовою Python для генерації навчальних прикладів.

Посилання на репозиторій (код програм, датасети): <https://github.com/Bohdan628318ylypchenko/MLDL-Lab1.git>

```
PS J:\repos\MLDL\Lab1\x64\Release> .\Neuron.exe
x0 = 0.200000, x1 = 0.700000, x_shift = 1.000000
Iteration 0 | w0 = 0.900000, w1 = 0.100000, w_shift = 0.400000 | ya = 0.657010
Iteration 1 | w0 = 0.908085, w1 = 0.128296, w_shift = 0.440423 | ya = 0.670809
Iteration 2 | w0 = 0.916169, w1 = 0.156592, w_shift = 0.480845 | ya = 0.684319
Iteration 3 | w0 = 0.924254, w1 = 0.184887, w_shift = 0.521268 | ya = 0.697525
Iteration 4 | w0 = 0.932338, w1 = 0.213183, w_shift = 0.561690 | ya = 0.710412
Iteration 5 | w0 = 0.940423, w1 = 0.241479, w_shift = 0.602113 | ya = 0.722968
Iteration 6 | w0 = 0.948507, w1 = 0.269775, w_shift = 0.642536 | ya = 0.735183
Iteration 7 | w0 = 0.956592, w1 = 0.298071, w_shift = 0.682958 | ya = 0.747047
Iteration 8 | w0 = 0.964676, w1 = 0.326367, w_shift = 0.723381 | ya = 0.758555
Iteration 9 | w0 = 0.972761, w1 = 0.354662, w_shift = 0.763803 | ya = 0.769700
Iteration 10 | w0 = 0.980845, w1 = 0.382958, w_shift = 0.804226 | ya = 0.780480
Iteration 11 | w0 = 0.988930, w1 = 0.411254, w_shift = 0.844649 | ya = 0.790892
Iteration 12 | w0 = 0.997014, w1 = 0.439550, w_shift = 0.885071 | ya = 0.800937
Iteration 13 | w0 = 1.005099, w1 = 0.467846, w_shift = 0.925494 | ya = 0.810614
Iteration 14 | w0 = 1.013183, w1 = 0.496142, w_shift = 0.965916 | ya = 0.819927
Iteration 15 | w0 = 1.021268, w1 = 0.524437, w_shift = 1.006339 | ya = 0.828878
Iteration 16 | w0 = 1.029352, w1 = 0.552733, w_shift = 1.046762 | ya = 0.837473
Iteration 17 | w0 = 1.037437, w1 = 0.581029, w_shift = 1.087184 | ya = 0.845716
Iteration 18 | w0 = 1.045521, w1 = 0.609325, w_shift = 1.127607 | ya = 0.853615
Iteration 19 | w0 = 1.053606, w1 = 0.637621, w_shift = 1.168029 | ya = 0.861175
Iteration 20 | w0 = 1.061690, w1 = 0.665916, w_shift = 1.208452 | ya = 0.868405
Iteration 21 | w0 = 1.069775, w1 = 0.694212, w_shift = 1.248875 | ya = 0.875313
Iteration 22 | w0 = 1.077859, w1 = 0.722508, w_shift = 1.289297 | ya = 0.881908
Iteration 23 | w0 = 1.085944, w1 = 0.750804, w_shift = 1.329720 | ya = 0.888199
Iteration 24 | w0 = 1.094029, w1 = 0.779100, w_shift = 1.370143 | ya = 0.894194
Iteration 25 | w0 = 1.102113, w1 = 0.807396, w_shift = 1.410565 | ya = 0.899905
PS J:\repos\MLDL\Lab1\x64\Release>
```

Навчання мережі-нейрону на єдиному прикладі ( $x_0=0.2$ ,  $x_1=0.7$ )

Мережа досягла похибки 0.001 за 25 ітерації з параметрами  $\lambda = 1$ ,  $\alpha = 0.05$

Початкові вагові коефіцієнти:

$W_0 = 0.9$  |  $W_1 = 0.1$  |  $W\_Shift = 0.4$

Остаточні вагові коефіцієнти:

$W_0 = 1.10213$  |  $W_1 = 0.807396$  |  $W\_Shift = 1.410565$

```
PS J:\repos\MLDL\Lab1\x64\Release> .\NN231.exe
```

```
Usage:
```

```
[n]ew;
```

```
[t]rain a[l] example-path epoch-count;
```

```
[v]alidate example-path;
```

```
[s]ave path;
```

```
[l]oad path;
```

```
[p]rint;
```

```
[r]un x1 x2;
```

```
[u]sage;
```

```
[e]xit;
```

```
Command: n
```

```
Command: p
```

```
l = 1.0000; a = 0.1000;
```

```
w12:
```

```
| 0.1000 | 0.1000 | 0.1000 |
```

```
| 0.1000 | 0.1000 | 0.1000 |
```

```
| 0.1000 | 0.1000 | 0.1000 |
```

```
w23:
```

```
| 0.1000 | 0.1000 | 0.1000 | 0.1000 |
```

```
Command: v examples2.txt
```

```
oa = -0.564342; ot = 0.174900
```

```
oa = 0.565400; ot = 0.751900
```

```
error = 0.186447
```

```
Command: t 1 4 examples2.txt 100
```

```
Command: p
```

```
l = 4.0000; a = 1.0000;
```

```
w12:
```

```
| -0.8983 | -0.8983 | -0.8983 |
```

```
| -0.1581 | -0.1581 | -0.1581 |
```

```
| 0.0460 | 0.0460 | 0.0460 |
```

```
w23:
```

```
| -0.7563 | -0.7563 | -0.7563 | 0.5185 |
```

```
Command: v examples2.txt
```

```
oa = -0.174901; ot = 0.174900
```

```
oa = 0.751900; ot = 0.751900
```

```
error = 0.000000
```

```
Command: s examples2-learned.nn
```

```
Command: e
```

```
PS J:\repos\MLDL\Lab1\x64\Release> |
```

Створення нової мережі

Новостворена мережа

Валідація нетренованої мережі на 2 прикладах. Загальна квадратична похибка є суттєвою.

Тренування мережі на 2 прикладах (файл examples2.txt)  
 $\alpha = 1, \lambda = 4$ , кількість епох = 100

Мережа після тренування

Валідація тренованої мережі на 2 прикладах.  
Загальна квадратична помилка менша за  $1e-6$ , мережа перенавчилась.

PS J:\repos\MLDL\Lab1\x64\Release> .\NN231.exe

Usage:

[n]ew;  
[t]rain a l example-path epoch-count;  
[v]alidate example-path;  
[s]ave path;  
[l]oad path;  
[p]rint;  
[r]un x1 x2;  
[u]sage;  
[e]xit;

Command: n

Command: p

l = 1.0000; a = 0.1000;

w12:

| 0.1000 | 0.1000 | 0.1000 |

| 0.1000 | 0.1000 | 0.1000 |

| 0.1000 | 0.1000 | 0.1000 |

w23:

| 0.1000 | 0.1000 | 0.1000 | 0.1000 |

Command: t 2 2 examples100.txt 5000

Command: v examples2.txt

oa = 0.224212; ot = 0.174900

oa = 0.791838; ot = 0.751900

error = 0.004027

Command: v examples100.txt

oa = 0.255825; ot = 0.241600

oa = 0.698727; ot = 0.681800

oa = 0.937071; ot = 0.966300

oa = 0.825084; ot = 0.785100

oa = 0.339915; ot = 0.358300

oa = 0.895158; ot = 0.905100

oa = 0.428279; ot = 0.450500

oa = 0.852852; ot = 0.826800

oa = 0.573037; ot = 0.565800

oa = 0.933016; ot = 0.959400

oa = 0.364517; ot = 0.385500

oa = 0.417787; ot = 0.432100

oa = 0.276596; ot = 0.266600

oa = 0.840169; ot = 0.805900

oa = 0.920098; ot = 0.938500

oa = 0.751951; ot = 0.710900

oa = 0.833323; ot = 0.801000

oa = 0.808864; ot = 0.789700

oa = 0.940257; ot = 0.972200

oa = 0.629626; ot = 0.631900

oa = 0.564007; ot = 0.560400

oa = 0.838031; ot = 0.796000

oa = 0.701951; ot = 0.671500

oa = 0.923312; ot = 0.933200

oa = 0.746713; ot = 0.715800

oa = 0.542394; ot = 0.547700

oa = 0.924862; ot = 0.941000

oa = 0.884941; ot = 0.884200

oa = 0.838833; ot = 0.812600

oa = 0.893520; ot = 0.870500

oa = 0.930405; ot = 0.949200

oa = 0.645329; ot = 0.636800

oa = 0.812338; ot = 0.785900

oa = 0.820195; ot = 0.777900

oa = 0.255152; ot = 0.235200

oa = 0.853108; ot = 0.844200

oa = 0.369517; ot = 0.385300

oa = 0.453891; ot = 0.473700

oa = 0.558886; ot = 0.571100

oa = 0.288356; ot = 0.290300

oa = 0.605635; ot = 0.611700

oa = 0.646359; ot = 0.624300

oa = 0.489536; ot = 0.513500

oa = 0.408918; ot = 0.424500

oa = 0.941663; ot = 0.970200

oa = 0.866503; ot = 0.845700

oa = 0.790077; ot = 0.761200

oa = 0.726966; ot = 0.705800

oa = 0.255762; ot = 0.239000

Створення нової мережі

новостворена  
мережа

Навчання мережі на  
examples100.txt (100  
прикладів)  
 $\alpha = 2$ ,  $\lambda = 2$ , кількість  
epoch = 5000

Валідація навченої мережі на  
examples2.txt (жоден приклад не  
входив у навчальну вибірку).  
Результат задовільний.

Валідація навченої мережі на навчальній вибір-  
ці. Загальна квадратична помилка - 0.08, мережа  
навчилась.

Навчена мережа

oa = 0.351962; ot = 0.370900  
oa = 0.823048; ot = 0.808400  
oa = 0.837929; ot = 0.813700  
oa = 0.484255; ot = 0.509100  
oa = 0.478233; ot = 0.503000  
oa = 0.554537; ot = 0.553600  
oa = 0.909155; ot = 0.922700  
oa = 0.422471; ot = 0.441700  
oa = 0.489384; ot = 0.507600  
oa = 0.585302; ot = 0.583900  
oa = 0.416572; ot = 0.436200  
oa = 0.943739; ot = 0.997600  
oa = 0.658655; ot = 0.650000  
oa = 0.818678; ot = 0.778000  
oa = 0.805133; ot = 0.767600  
oa = 0.917606; ot = 0.911000  
oa = 0.944328; ot = 0.983400  
oa = 0.196579; ot = 0.114700  
oa = 0.777249; ot = 0.737100  
oa = 0.470080; ot = 0.492100  
oa = 0.609126; ot = 0.604500  
oa = 0.902361; ot = 0.902400  
oa = 0.478039; ot = 0.489500  
oa = 0.885459; ot = 0.880100  
oa = 0.678544; ot = 0.662000  
oa = 0.476832; ot = 0.488600  
oa = 0.720659; ot = 0.694400  
oa = 0.278224; ot = 0.278900  
oa = 0.796841; ot = 0.773200  
oa = 0.396525; ot = 0.412200  
oa = 0.400869; ot = 0.431600  
oa = 0.662659; ot = 0.645800  
oa = 0.868025; ot = 0.859200  
oa = 0.937996; ot = 0.978400  
oa = 0.816928; ot = 0.780000  
oa = 0.253956; ot = 0.236400  
oa = 0.164920; ot = 0.018200  
oa = 0.670622; ot = 0.650900  
oa = 0.479737; ot = 0.497900  
oa = 0.923031; ot = 0.945800  
oa = 0.824475; ot = 0.808300  
oa = 0.827163; ot = 0.796600  
oa = 0.644453; ot = 0.623600  
oa = 0.916554; ot = 0.919900  
oa = 0.559305; ot = 0.568500  
oa = 0.300073; ot = 0.305600  
oa = 0.557101; ot = 0.553000  
oa = 0.935792; ot = 0.981100  
oa = 0.914030; ot = 0.924300  
oa = 0.757533; ot = 0.740500  
oa = 0.948061; ot = 0.996600  
error = 0.081400

Command: p

l = 2.0000; a = 2.0000;

w12:

| 1.1734 | 1.1734 | 1.1734 |

| 1.1288 | 1.1288 | 1.1288 |

| -1.2425 | -1.2425 | -1.2425 |

w23:

| 2.0146 | 2.0146 | 2.0146 | -

1.2944 |

Command: s examples100-  
trained.nn

Command: e

PS J:\repos\MLDL\Lab1\x64  
\Release>

```

PS J:\repos\MLDL\Lab1\x64\Release> .\NN231.exe
Usage:
[n]ew;
[t]rain a l example-path epoch-count;
[v]alidate example-path;
[s]ave path;
[l]oad path;
[p]rint;
[r]un x1 x2;
[u]sage;
[e]xit;
Command: l examples100-trained.nn
Command: p
l = 2.0000; a = 2.0000;
w12:
| 1.1734 | 1.1734 | 1.1734 |
| 1.1288 | 1.1288 | 1.1288 |
| -1.2425 | -1.2425 | -1.2425 |
w23:
| 2.0146 | 2.0146 | 2.0146 | -1.2944 |
Command: v examples1000.txt

```

Завантаження мережі, навченої на examples100.txt

Завантажена мережа

Валідація мережі, навченої на examples100.txt, на examples1000.txt (1000 прикладів), жоден з валідаційних прикладів не був присутній у навчальній вибірці.

```

oa = 0.785319; ot = 0.747400
oa = 0.239995; ot = 0.209300
oa = 0.898389; ot = 0.888300
oa = 0.768722; ot = 0.728700
oa = 0.847976; ot = 0.818000
oa = 0.625345; ot = 0.619600
oa = 0.890756; ot = 0.868000
error = 0.709704
average_error = 0.000710

```

Середня квадратична помилка мережі на examples1000.txt є задовільною, мережа успішно апроксимувала функцію  $y = x1 + x2$ .

# Selected files

## 6 printable files

Neuron\neuron.c  
 Neuron\neuron.h  
 NN231\nn\_io.c  
 NN231\nn.c  
 NN231\nn.h  
 NN231\examples.py

### Neuron\neuron.c

```

1  #include "neuron.h"
2
3  #define _USE_MATH_DEFINES
4  #include <math.h>
5
6  #include <stdlib.h>
7
8  static void _grad(unsigned long n, double * x,
9                  double ya, double yt, double l,
10                 double * grad);
11
12 static void _norm(unsigned long n, double * v);
13
14 /// <summary>
15 /// Calculates neuron output.
16 /// </summary>
17 /// <param name="n"> Neuron dimension: weight count / input count (include shift). </param>
18 /// <param name="w"> Weights array (include shift). </param>
19 /// <param name="x"> Input array (include shift). </param>
20 /// <param name="l"> Activation function smoothing coefficient. </param>
21 /// <returns> Neuron output. </returns>
22 double neuron_activate(unsigned long n, double * w, double * x, double l)
23 {
24     // Activation function argument
25     double s = 0;
26     for (unsigned long i = 0; i < n; i++)
27         s += w[i] * x[i];
28
29     // Activation function
30     double ya = 1.0 / (1.0 + exp(-1.0 * l * s));
31
32     // Returning
33     return ya;
34 }
35
36 /// <summary>
37 /// Adjusts neuron weights by gradient of Error from w values.
38 /// </summary>
39 /// <param name="n"> Neuron dimension: weight count / input count (include shift). </param>
40 /// <param name="w"> Weights array (include shift). </param>
41 /// <param name="x"> Input array (include shift). </param>

```



```

42  /// <param name="ya"> Neuron output. </param>
43  /// <param name="yt"> Expected output. </param>
44  /// <param name="l"> Activation function smoothing coefficient:  $o(s) = 1 / (1 + e^{-1 * s})$ 
    </param>
45  /// <param name="a"> Weight adjustment length coefficient:  $w_n = w_c - a * ngrad$  </param>
46  void neuron_adjust_weights(unsigned long n,
47                             double * w, double * x,
48                             double ya, double yt, double l, double a)
49  {
50      // Calculate grad
51      double * grad = (double *)malloc(n * sizeof(double));
52      _grad(n, x, ya, yt, l, grad);
53
54      // Normalize
55      _norm(n, grad);
56
57      // Adjust weight
58      for (unsigned long i = 0; i < n; i++)
59          w[i] -= a * grad[i];
60
61      // Free resources
62      free(grad);
63  }
64
65  /// <summary>
66  /// Calculates gradient of E(W).
67  /// Result is stored in grad.
68  /// </summary>
69  /// <param name="n"> Dimension. </param>
70  /// <param name="x"> Neuron input. </param>
71  /// <param name="ya"> Neuron output. </param>
72  /// <param name="yt"> Expected output. </param>
73  /// <param name="l"> Activation function smoothing coefficient. </param>
74  /// <param name="grad"> Array to store gradient coordinates in. </param>
75  static void _grad(unsigned long n, double * x,
76                    double ya, double yt, double l,
77                    double * grad)
78  {
79      for (unsigned long i = 0; i < n; i++)
80          grad[i] = (ya - yt) // dE/dya
81                  * l * ya * (1 - ya) // dya/ds
82                  * x[i]; // ds/dwi;
83  }
84
85  /// <summary>
86  /// Normalizes given vector.
87  /// </summary>
88  /// <param name="n"> Vector dimension. </param>
89  /// <param name="v"> Vector values. </param>
90  static void _norm(unsigned long n, double * v)
91  {
92      // vector module
93      double m = 0;
94      for (unsigned long i = 0; i < n; i++)
95          m += v[i] * v[i];
96      m = sqrt(m);

```

```

97
98     // normalize
99     for (unsigned long i = 0; i < n; i++)
100         v[i] /= m;
101 }
102

```

## Neuron\neuron.h

```

1  #pragma once
2
3  /// <summary>
4  /// Calculates neuron output.
5  /// </summary>
6  /// <param name="n"> Neuron dimension: weight count / input count (include shift). </param>
7  /// <param name="w"> Weights array (include shift). </param>
8  /// <param name="x"> Input array (include shift). </param>
9  /// <param name="l"> Activation function smoothing coefficient. </param>
10 /// <returns> Neuron output. </returns>
11 double neuron_activate(unsigned long n, double * w, double * x, double l);
12
13 /// <summary>
14 /// Adjusts neuron weights by gradient of Error from w values.
15 /// </summary>
16 /// <param name="n"> Neuron dimension: weight count / input count (include shift). </param>
17 /// <param name="w"> Weights array (include shift). </param>
18 /// <param name="x"> Input array (include shift). </param>
19 /// <param name="ya"> Neuron output. </param>
20 /// <param name="yt"> Expected output. </param>
21 /// <param name="l"> Activation function smoothing coefficient:  $o(s) = 1 / (1 + e^{(-1 * s)})$  </param>
22 /// <param name="a"> Weight adjustment length coefficient:  $w_n = w_c - a * ngrad$  </param>
23 void neuron_adjust_weights(unsigned long n,
24                             double * w, double * x,
25                             double ya, double yt, double l, double a);
26

```

## NN231\nn\_io.c

```

1  #include "nn.h"
2
3  /// <summary>
4  /// Writes neural network to stream in binary format.
5  /// </summary>
6  /// <param name="l"> NN lambda parameter. </param>
7  /// <param name="a"> NN alpha parameter. </param>
8  /// <param name="w12"> Weights of layer1 -> layer2 as matrix. </param>
9  /// <param name="w23"> Weights of layer2 -> layer3 as vector. </param>
10 /// <param name="f"> Output stream. </param>
11 void nn_fwrite(double l, double a,
12                double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
13                FILE * f)
14 {
15     fwrite(&l, sizeof(double), 1, f);

```

```

16     fwrite(&a, sizeof(double), 1, f);
17     for (int i = 0; i < L1COUNT + 1; i++)
18         fwrite(w12[i], sizeof(double), L2COUNT, f);
19     fwrite(w23, sizeof(double), L2COUNT + 1, f);
20 }
21
22 /// <summary>
23 /// Reads neural network from stream.
24 /// </summary>
25 /// <param name="l"> Pointer to read NN lambda parameter in. </param>
26 /// <param name="a"> Pointer to read NN alpha parameter in. </param>
27 /// <param name="w12"> Matrix to read layer1->layer2 weights in. </param>
28 /// <param name="w23"> Vector to read layer2->layer3 weights in. </param>
29 /// <param name="f"> Output stream. </param>
30 void nn_fread(double * l, double * a,
31              double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
32              FILE * f)
33 {
34     fread(l, sizeof(double), 1, f);
35     fread(a, sizeof(double), 1, f);
36     for (int i = 0; i < L1COUNT + 1; i++)
37         fread(w12[i], sizeof(double), L2COUNT, f);
38     fread(w23, sizeof(double), L2COUNT + 1, f);
39 }
40
41 /// <summary>
42 /// Writes neural network to stream in text format.
43 /// </summary>
44 /// <param name="l"> NN lambda parameter. </param>
45 /// <param name="a"> NN alpha parameter. </param>
46 /// <param name="w12"> Weights of layer1 -> layer2 as matrix. </param>
47 /// <param name="w23"> Weights of layer2 -> layer3 as vector. </param>
48 /// <param name="f"> Output stream. </param>
49 void nn_fprint(double l, double a,
50               double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
51               FILE * stream)
52 {
53     fprintf(stream, "l = %.4lf; a = %.4lf;\n", l, a);
54
55     fprintf(stream, "w12:");
56     for (int i = 0; i < L1COUNT + 1; i++)
57     {
58         fputs("\n|", stream);
59         for (int j = 0; j < L2COUNT; j++)
60         {
61             fprintf(stream, " %.4lf |", w12[i][j]);
62         }
63     }
64
65     fprintf(stream, "\nw23:\n|");
66     for (int i = 0; i < L2COUNT + 1; i++)
67     {
68         fprintf(stream, " %.4lf |", w23[i]);
69     }
70     fputc('\n', stream);
71 }

```

72 |

## NN231\nn.c

```

1  #include "nn.h"
2
3  #define _USE_MATH_DEFINES
4  #include <math.h>
5
6  #define SIGMA(x, l) 1.0 / (1.0 + exp(-1.0 * l * x))
7
8  /// <summary>
9  /// NN activation implementation.
10 /// </summary>
11 /// <param name="x"> Input signal vector. </param>
12 /// <param name="w12"> layer1->layer2 weights as matrix. </param>
13 /// <param name="w23"> layer2->layer3 weights as vector. </param>
14 /// <param name="l"> NN lambda parameter. </param>
15 /// <param name="o1"> Vector to write layer1 output. </param>
16 /// <param name="o2"> Vector to write layer2 output. </param>
17 /// <param name="oa"> Pointer to write layer3 output. </param>
18 void nn_activate(double x[L1COUNT],
19                 double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1], double l,
20                 double o1[L1COUNT + 1], double o2[L2COUNT + 1], double * oa)
21 {
22     // 1st layer output + shift
23     for (int i = 0; i < L1COUNT; i++)
24         o1[i] = x[i];
25     o1[L1COUNT] = 1.0;
26
27     // Sum for 2nd layer
28     double s2[L2COUNT] = { 0.0, 0.0, 0.0 };
29     for (int i = 0; i < L2COUNT; i++)
30         for (int j = 0; j < L1COUNT + 1; j++)
31             s2[i] += o1[j] * w12[j][i];
32
33     // 2nd layer output + shift
34     for (int i = 0; i < L2COUNT; i++)
35         o2[i] = SIGMA(s2[i], l);
36     o2[L2COUNT] = 1.0;
37
38     // Sum for 3rd layer
39     double s3 = 0;
40     for (int i = 0; i < L2COUNT + 1; i++)
41         s3 += o2[i] * w23[i];
42
43     // Final output
44     *oa = SIGMA(s3, l);
45 }
46
47 /// <summary>
48 /// Does 1 nn weights adjustment based on layer1, layer2, layer3 output
49 /// and expected NN output.
50 /// </summary>
51 /// <param name="oa"> Layer3 output. </param>

```

```

52  /// <param name="ot"> Expected NN output. </param>
53  /// <param name="w12"> layer1->layer2 weights as matrix. </param>
54  /// <param name="w23"> layer2->layer3 weights as vector. </param>
55  /// <param name="l"> NN lambda parameter. </param>
56  /// <param name="a"> NN alpha parameter. </param>
57  /// <param name="o1"> Layer1 output. </param>
58  /// <param name="o2"> Layer2 output. </param>
59  void nn_adjust(double oa, double ot,
60                double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1], double l, double a,
61                double o1[L1COUNT + 1], double o2[L2COUNT + 1])
62  {
63      // Common part
64      double cdelta = l * oa * (1.0 - oa) * (oa - ot);
65
66      // 21 -> 31 deltas
67      double deltas23[L2COUNT + 1];
68      for (int i = 0; i < L2COUNT + 1; i++)
69          deltas23[i] = o2[i] * cdelta;
70
71      // 11 -> 21 deltas
72      double deltas12[L1COUNT + 1][L2COUNT];
73      for (int i = 0; i < L1COUNT + 1; i++)
74          for (int j = 0; j < L2COUNT; j++)
75              deltas12[i][j] = o1[i] * l * o2[j] * (1 - o2[j]) * w23[j] * cdelta;
76
77      // 21 -> 31 adjustment
78      for (int i = 0; i < L2COUNT + 1; i++)
79          w23[i] -= (a * deltas23[i]);
80
81      // 11 -> 21 adjustment
82      for (int i = 0; i < L1COUNT + 1; i++)
83          for (int j = 0; j < L2COUNT; j++)
84              w12[i][j] -= (a * deltas12[i][j]);
85  }
86

```

## NN231\nn.h

```

1  #pragma once
2
3  #include <stdio.h>
4
5  #define L1COUNT 2
6  #define L2COUNT 3
7
8  /// <summary>
9  /// NN activation implementation.
10 /// </summary>
11 /// <param name="x"> Input signal vector. </param>
12 /// <param name="w12"> layer1->layer2 weights as matrix. </param>
13 /// <param name="w23"> layer2->layer3 weights as vector. </param>
14 /// <param name="l"> NN lambda parameter. </param>
15 /// <param name="o1"> Vector to write layer1 output. </param>
16 /// <param name="o2"> Vector to write layer2 output. </param>
17 /// <param name="oa"> Pointer to write layer3 output. </param>

```

```

18 void nn_activate(double x[L1COUNT],
19                 double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1], double l,
20                 double o1[L1COUNT + 1], double o2[L2COUNT + 1], double * oa);
21
22 /// <summary>
23 /// Does 1 nn weights adjustment based on layer1, layer2, layer3 output
24 /// and expected NN output.
25 /// </summary>
26 /// <param name="oa"> Layer3 output. </param>
27 /// <param name="ot"> Expected NN output. </param>
28 /// <param name="w12"> layer1->layer2 weights as matrix. </param>
29 /// <param name="w23"> layer2->layer3 weights as vector. </param>
30 /// <param name="l"> NN lambda parameter. </param>
31 /// <param name="a"> NN alpha parameter. </param>
32 /// <param name="o1"> Layer1 output. </param>
33 /// <param name="o2"> Layer2 output. </param>
34 void nn_adjust(double oa, double ot,
35               double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1], double l, double a,
36               double o1[L1COUNT + 1], double o2[L2COUNT + 1]);
37
38 /// <summary>
39 /// Writes neural network to stream in binary format.
40 /// </summary>
41 /// <param name="l"> NN lambda parameter. </param>
42 /// <param name="a"> NN alpha parameter. </param>
43 /// <param name="w12"> Weights of layer1 -> layer2 as matrix. </param>
44 /// <param name="w23"> Weights of layer2 -> layer3 as vector. </param>
45 /// <param name="f"> Output stream. </param>
46 void nn_fwrite(double l, double a,
47               double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
48               FILE * f);
49
50 /// <summary>
51 /// Reads neural network from stream.
52 /// </summary>
53 /// <param name="l"> Pointer to read NN lambda parameter in. </param>
54 /// <param name="a"> Pointer to read NN alpha parameter in. </param>
55 /// <param name="w12"> Matrix to read layer1->layer2 weights in. </param>
56 /// <param name="w23"> Vector to read layer2->layer3 weights in. </param>
57 /// <param name="f"> Output stream. </param>
58 void nn_fread(double * l, double * a,
59               double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
60               FILE * f);
61
62 /// <summary>
63 /// Writes neural network to stream in text format.
64 /// </summary>
65 /// <param name="l"> NN lambda parameter. </param>
66 /// <param name="a"> NN alpha parameter. </param>
67 /// <param name="w12"> Weights of layer1 -> layer2 as matrix. </param>
68 /// <param name="w23"> Weights of layer2 -> layer3 as vector. </param>
69 /// <param name="f"> Output stream. </param>
70 void nn_fprint(double l, double a,
71               double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
72               FILE * stream);
73

```

NN231\examples.py

```
1 import random
2
3 OUTPUT = "examples.txt"
4 EXAMPLE_COUNT = 30
5
6 examples = []
7 while (len(examples) != EXAMPLE_COUNT):
8     a = round(random.random(), 4)
9     b = round(random.random(), 4)
10    if a + b <= 1.0:
11        examples.append((a, b, round(a + b, 6)))
12
13 with open(OUTPUT, 'w') as file:
14     file.write(f"{EXAMPLE_COUNT}\n")
15     for item in examples:
16         file.write(f"{item[0]} {item[1]} {item[2]}\n")
17
```