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.

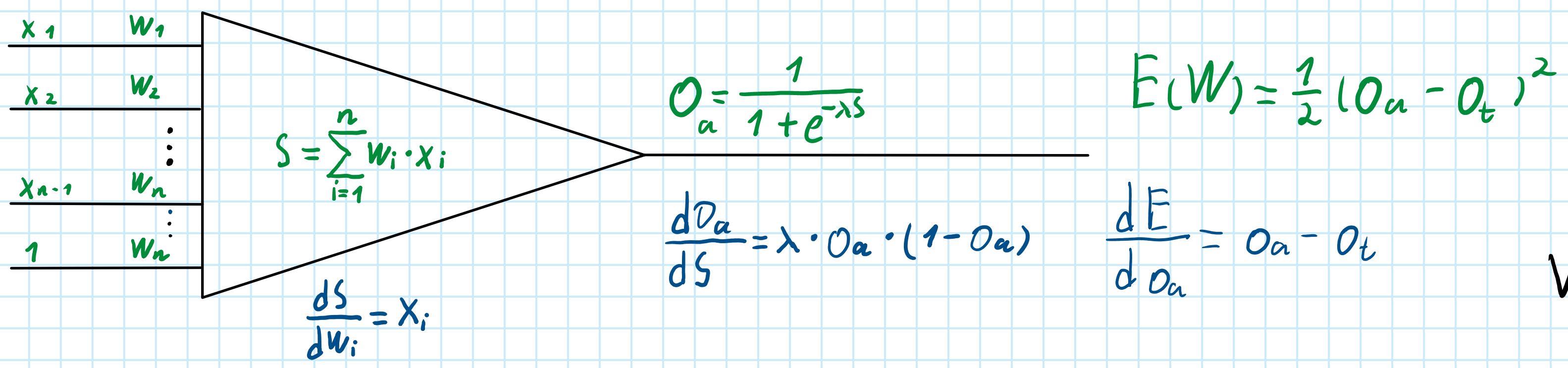
Під час навчання пряме поширення сигналу дозволяє отримати значення функції помилки мережі E(W) (W - вагові коефіцієнти мережі).
Алгоритм оберненого поширення обчислює градієнт E(W) для поточних значень коефіцієнтів Wc.

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."[1]

Граф обчислення похідних для нейронної мережі, що складається з єдиного нейрону(частина 1):



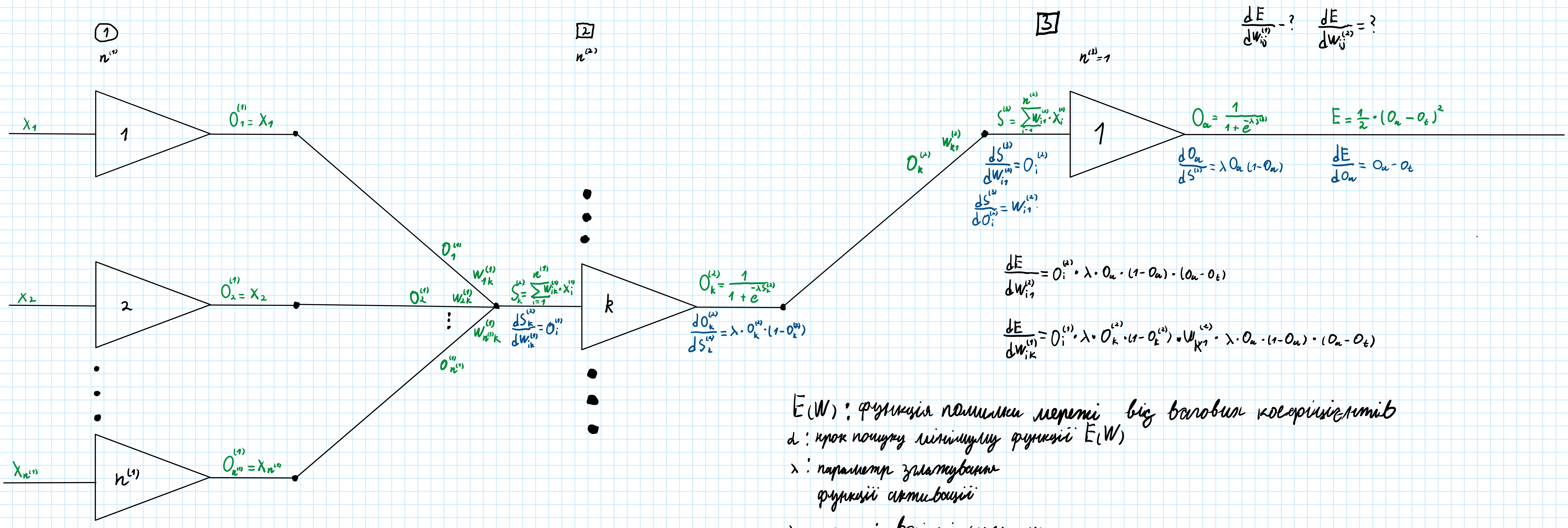
$$\frac{dE}{dw_i}=\frac{dE}{dO_a}\cdot\frac{dO_a}{dS}\cdot\frac{dS}{dw_i}=(O_a-O_t)\cdot\lambda\cdot O_a\cdot(1-O_a)\cdot x_i$$

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

$$w_i^{(n)}=w^{(c)}-\alpha\,\overrightarrow{grad}(E)$$

$E(W)$: функція помилки нейрону від вагових коефіцієнтів
α : крок пошуку мінімуму функції $E(W)$
λ : параметр згладжування функції активації
$x_1 \cdots x_{n-1}$: вхідні сигнали
W; $w_1 \cdots w_n$: вагові коефіцієнти нейрону
O_a : вихідний сигнал нейрону
O_t : очікуваний вихідний сигнал нейрону
$w^{(c)}$: поточне значення вагового коефіцієнту
$w^{(n)}$: оновлене значення вагового коефіцієнту

Граф обчислення похідних для нейронної мережі з трьома шарами (n1-n2-1) (частина 3):



$$\frac{dE}{dw_{i1}^{(2)}}=O_i^{(2)}\cdot\lambda\cdot O_a\cdot(1-O_a)\cdot(O_a-O_t)$$

$$\frac{dE}{dw_{ik}^{(1)}}=O_i^{(1)}\cdot\lambda\cdot O_k^{(2)}\cdot(1-O_k^{(2)})\cdot w_{k1}^{(2)}\cdot\lambda\cdot O_a\cdot(1-O_a)\cdot(O_a-O_t)$$

$E(W)$: функція помилки мережі від вагових коефіцієнтів
α : крок пошуку мінімуму функції $E(W)$
λ : параметр згладжування функції активації
$x_1 \cdots x_{n-1}$: вхідні сигнали
W; $w_1^{(j)} \cdots w_n^{(j)}$: вагові коефіцієнти між шарами (j ; j+1)
O_a : вихідний сигнал мережі
O_t : очікуваний вихідний сигнал мережі
$O_i^{(j)}$: вихідний сигнал i-того нейрону j-того шару

[1]: "Deep learning" Ian Goodfellow, Yoshua Bengio, Aaron Courville The MIT Press

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 (x0=0.2, x1=0.7)
Мережа досягла похибки 0.001 за 25 ітерації з параметрами lambda = 1, alpha = 0.05
Остаточні вагові коефіцієнти:
W0 = 1.10213 | W1 = 0.807396 | W_Shift = 1.410565
Початкові вагові коефіцієнти:
W0 = 0.9 | W1 = 0.1 | W_Shift = 0.4

```
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 мережі
Похибка суттєва
Навчаємо мережу
Мережа після навчання
Похибка майже 0, мережа перенавчилась

Навчання мережі 2-3-1 на датасеті examples2.txt (2 приклади).

```
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
```

⋮

```
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
```

$$average\ error = \left(\sum_{i=1}^{example\_count} (o_a - o_t)^2\right) / example\text{-}count$$

Валідація мережі 2-3-1, навченої на датасеті examples100.txt, на датасеті examples1000.txt (1000 прикладів, жоден не був присутній у навчальній вибірці)

```
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
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>
```

Валідація мережі на examples2.txt (не входить в навчальну вибірку)

Валідація на навчальній вибірці

Навчена мережа

Навчання мережі 2-3-1 на датасеті examples100.txt (100 прикладів)

Код реалізації мережі-нейрону

Реалізація нейрону

neuron.h

```c
#pragma once

/// <summary>
/// Calculates neuron output.
/// </summary>
/// <param name="n"> Neuron dimension: weight count / input count (include shift). </param>
/// <param name="w"> Weights array (include shift. </param>
/// <param name="x"> Input array (include shift). </param>
/// <param name="l"> Activation function smoothing coefficient. </param>
/// <returns> Neuron output. </returns>
double neuron_activate(unsigned long n, double * w, double * x, double l);

/// <summary>
/// Adjusts neuron weights by gradient of Error from w values.
/// </summary>
/// <param name="n"> Neuron dimension: weight count / input count (include shift). </param>
/// <param name="w"> Weights array (include shift). </param>
/// <param name="x"> Input array (include shift). </param>
/// <param name="ya"> Neuron output. </param>
/// <param name="yt"> Expected output. </param>
/// <param name="l"> Activation function smoothing coefficient: o(s) = 1 / (1 + e ^ (-l * s)) </param>
/// <param name="a"> Weight adjustment length coefficient: wn = wc - a * ngrad </param>
void neuron_adjust_weights(unsigned long n,
                                        double * w, double * x,
                                        double ya, double yt, double l, double a);
```

neuron.c

```c
#include "neuron.h"

#define _USE_MATH_DEFINES
#include <math.h>

#include <stdlib.h>

static void _grad(unsigned long n, double * x,
                        double ya, double yt, double l,
                            double * grad);

static void _norm(unsigned long n, double * v);

/// <summary>
/// Calculates neuron output.
/// </summary>
/// <param name="n"> Neuron dimension: weight count / input count (include shift). </param>
/// <param name="w"> Weights array (include shift. </param>
/// <param name="x"> Input array (include shift). </param>
/// <param name="l"> Activation function smoothing coefficient. </param>
/// <returns> Neuron output. </returns>
double neuron_activate(unsigned long n, double * w, double * x, double l)
{
    // Activation function argument
    double s = 0;
    for (unsigned long i = 0; i < n; i++)
        s += w[i] * x[i];

    // Activation function
    double ya = 1.0 / (1.0 + exp(-1.0 * l * s));

    // Returning
    return ya;
}

/// <summary>
/// Adjusts neuron weights by gradient of Error from w values.
/// </summary>
/// <param name="n"> Neuron dimension: weight count / input count (include shift). </param>
/// <param name="w"> Weights array (include shift). </param>
/// <param name="x"> Input array (include shift). </param>
/// <param name="ya"> Neuron output. </param>
/// <param name="yt"> Expected output. </param>
/// <param name="l"> Activation function smoothing coefficient: o(s) = 1 / (1 + e ^ (-l * s)) </param>
/// <param name="a"> Weight adjustment length coefficient: wn = wc - a * ngrad </param>
void neuron_adjust_weights(unsigned long n,
                                        double * w, double * x,
                                double ya, double yt, double l, double a)
{
    // Calculate grad
    double * grad = (double *)malloc(n * sizeof(double));
    _grad(n, x, ya, yt, l, grad);

    // Normalize
    _norm(n, grad);

    // Adjust weight
    for (unsigned long i = 0; i < n; i++)
        w[i] -= a * grad[i];

    // Free resources
    free(grad);
}

/// <summary>
/// Calculates gradient of E(W).
/// Result is stored in grad.
/// </summary>
/// <param name="n"> Dimension. </param>
/// <param name="x"> Neuron input. </param>
/// <param name="ya"> Neuron output. </param>
/// <param name="yt"> Expected output. </param>
/// <param name="l"> Activation function smoothing coefficient. </param>
/// <param name="grad"> Array to store gradient coordinates in. </param>
static void _grad(unsigned long n, double * x,
                            double ya, double yt, double l,
                            double * grad)
{
    for (unsigned long i = 0; i < n; i++)
        grad[i] = (ya - yt)        // dE/dya
       * l * ya * (1 - ya) // dya/ds
            * x[i];           // ds/dwi;
}

/// <summary>
/// Normalizes given vector.
/// </summary>
/// <param name="n"> Vector dimension. </param>
/// <param name="v"> Vector values. </param>
static void _norm(unsigned long n, double * v)
{
    // vector module
    double m = 0;
    for (unsigned long i = 0; i < n; i++)
        m += v[i] * v[i];
    m = sqrt(m);

    // normalize
    for (unsigned long i = 0; i < n; i++)
        v[i] /= m;
}
```

Код реалізації мережі 2-3-1

Реалізація мережі

nn.h

```c
#pragma once

#include <stdio.h>

#define L1COUNT 2
#define L2COUNT 3

/// <summary>
/// NN activation implementation.
/// </summary>
/// <param name="x"> Input signal vector. </param>
/// <param name="w12"> layer1->layer2 weights as matrix. </param>
/// <param name="w23"> layer2->layer3 weights as vector. </param>
/// <param name="l"> NN lambda parameter. </param>
/// <param name="o1"> Vector to write layer1 output. </param>
/// <param name="o2"> Vector to write layer2 output. </param>
/// <param name="oa"> Pointer to write layer3 output. </param>
void nn_activate(double x[L1COUNT],
                                double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1], double l,
                                double o1[L1COUNT + 1], double o2[L2COUNT + 1], double * oa);

/// <summary>
/// Does 1 nn weights adjustment based on layer1, layer2, layer3 output
/// and expected NN output.
/// </summary>
/// <param name="oa"> Layer3 output. </param>
/// <param name="ot"> Expected NN output. </param>
/// <param name="w12"> layer1->layer2 weights as matrix. </param>
/// <param name="w23"> layer2->layer3 weights as vector. </param>
/// <param name="l"> NN lambda parameter. </param>
/// <param name="a"> NN alpha parameter. </param>
/// <param name="o1"> Layer1 output. </param>
/// <param name="o2"> Layer2 output. </param>
void nn_adjust(double oa, double ot,
                        double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1], double l, double a,
                        double o1[L1COUNT + 1], double o2[L2COUNT + 1]);

/// <summary>
/// Writes neural network to stream in binary format.
/// </summary>
/// <param name="l"> NN lambda parameter. </param>
/// <param name="a"> NN alpha parameter. </param>
/// <param name="w12"> Weights of layer1 -> layer2 as matrix. </param>
/// <param name="w23"> Weights of layer2 -> layer3 as vector. </param>
/// <param name="f"> Output stream. </param>
void nn_fwrite(double l, double a,
                        double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
                        FILE * f);

/// <summary>
/// Reads neural network from stream.
/// </summary>
/// <param name="l"> Pointer to read NN lambda parameter in. </param>
/// <param name="a"> Pointer to read NN alpha parameter in. </param>
/// <param name="w12"> Matrix to read layer1->layer2 weights in. </param>
/// <param name="w23"> Vector to read layer2->layer3 weights in. </param>
/// <param name="f"> Output stream. </param>
void nn_fread(double * l, double * a,
                        double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
                        FILE * f);

/// <summary>
/// Writes neural network to stream in text format.
/// </summary>
/// <param name="l"> NN lambda parameter. </param>
/// <param name="a"> NN alpha parameter. </param>
/// <param name="w12"> Weights of layer1 -> layer2 as matrix. </param>
/// <param name="w23"> Weights of layer2 -> layer3 as vector. </param>
/// <param name="f"> Output stream. </param>
void nn_fprint(double l, double a,
                        double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
                        FILE * stream);
```

nn.c

```c
#include "nn.h"

#define _USE_MATH_DEFINES
#include <math.h>

#define SIGMA(x, l) 1.0 / (1.0 + exp(-1.0 * l * x))

/// <summary>
/// NN activation implementation.
/// </summary>
/// <param name="x"> Input signal vector. </param>
/// <param name="w12"> layer1->layer2 weights as matrix. </param>
/// <param name="w23"> layer2->layer3 weights as vector. </param>
/// <param name="l"> NN lambda parameter. </param>
/// <param name="o1"> Vector to write layer1 output. </param>
/// <param name="o2"> Vector to write layer2 output. </param>
/// <param name="oa"> Pointer to write layer3 output. </param>
void nn_activate(double x[L1COUNT],
                                double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1], double l,
                                double o1[L1COUNT + 1], double o2[L2COUNT + 1], double * oa)
{
        // 1st layer output + shift
        for (int i = 0; i < L1COUNT; i++)
                o1[i] = x[i];
        o1[L1COUNT] = 1.0;

        // Sum for 2nd layer
        double s2[L2COUNT] = { 0.0, 0.0, 0.0 };
        for (int i = 0; i < L2COUNT; i++)
                for (int j = 0; j < L1COUNT + 1; j++)
                        s2[i] += o1[j] * w12[j][i];

        // 2nd layer output + shift
        for (int i = 0; i < L2COUNT; i++)
                o2[i] = SIGMA(s2[i], l);
        o2[L2COUNT] = 1.0;

        // Sum for 3rd layer
        double s3 = 0;
        for (int i = 0; i < L2COUNT + 1; i++)
                s3 += o2[i] * w23[i];

        // Final output
        *oa = SIGMA(s3, l);
}

/// <summary>
/// Does 1 nn weights adjustment based on layer1, layer2, layer3 output
/// and expected NN output.
/// </summary>
/// <param name="oa"> Layer3 output. </param>
/// <param name="ot"> Expected NN output. </param>
/// <param name="w12"> layer1->layer2 weights as matrix. </param>
/// <param name="w23"> layer2->layer3 weights as vector. </param>
/// <param name="l"> NN lambda parameter. </param>
/// <param name="a"> NN alpha parameter. </param>
/// <param name="o1"> Layer1 output. </param>
/// <param name="o2"> Layer2 output. </param>
void nn_adjust(double oa, double ot,
                        double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1], double l, double a,
                        double o1[L1COUNT + 1], double o2[L2COUNT + 1])
{
        // Common part
        double cdelta = l * oa * (1.0 - oa) * (oa - ot);

        // 2l -> 3l deltas
        double deltas23[L2COUNT + 1];
        for (int i = 0; i < L2COUNT + 1; i++)
                deltas23[i] = o2[i] * cdelta;

        // 1l -> 2l deltas
        double deltas12[L1COUNT + 1][L2COUNT];
        for (int i = 0; i < L1COUNT + 1; i++)
                for (int j = 0; j < L2COUNT; j++)
                        deltas12[i][j] = o1[i] * l * o2[j] * (1 - o2[j]) * w23[j] * cdelta;

        // 2l -> 3l adjustment
        for (int i = 0; i < L2COUNT + 1; i++)
                w23[i] -= (a * deltas23[i]);

        // 1l -> 2l adjustment
        for (int i = 0; i < L1COUNT + 1; i++)
                for (int j = 0; j < L2COUNT; j++)
                        w12[i][j] -= (a * deltas12[i][j]);
}
```

nn_io.c

```c
#include "nn.h"

/// <summary>
/// Writes neural network to stream in binary format.
/// </summary>
/// <param name="l"> NN lambda parameter. </param>
/// <param name="a"> NN alpha parameter. </param>
/// <param name="w12"> Weights of layer1 -> layer2 as matrix. </param>
/// <param name="w23"> Weights of layer2 -> layer3 as vector. </param>
/// <param name="f"> Output stream. </param>
void nn_fwrite(double l, double a,
                        double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
                        FILE * f)
{
        fwrite(&l, sizeof(double), 1, f);
        fwrite(&a, sizeof(double), 1, f);
        for (int i = 0; i < L1COUNT + 1; i++)
                fwrite(w12[i], sizeof(double), L2COUNT, f);
        fwrite(w23, sizeof(double), L2COUNT + 1, f);
}

/// <summary>
/// Reads neural network from stream.
/// </summary>
/// <param name="l"> Pointer to read NN lambda parameter in. </param>
/// <param name="a"> Pointer to read NN alpha parameter in. </param>
/// <param name="w12"> Matrix to read layer1->layer2 weights in. </param>
/// <param name="w23"> Vector to read layer2->layer3 weights in. </param>
/// <param name="f"> Output stream. </param>
void nn_fread(double * l, double * a,
                        double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
                        FILE * f)
{
        fread(l, sizeof(double), 1, f);
        fread(a, sizeof(double), 1, f);
        for (int i = 0; i < L1COUNT + 1; i++)
                fread(w12[i], sizeof(double), L2COUNT, f);
        fread(w23, sizeof(double), L2COUNT + 1, f);
}

/// <summary>
/// Writes neural network to stream in text format.
/// </summary>
/// <param name="l"> NN lambda parameter. </param>
/// <param name="a"> NN alpha parameter. </param>
/// <param name="w12"> Weights of layer1 -> layer2 as matrix. </param>
/// <param name="w23"> Weights of layer2 -> layer3 as vector. </param>
/// <param name="f"> Output stream. </param>
void nn_fprint(double l, double a,
                        double w12[L1COUNT + 1][L2COUNT], double w23[L2COUNT + 1],
                        FILE * stream)
{
        fprintf(stream, "l = %.4lf; a = %.4lf;\n", l, a);

        fprintf(stream, "w12:");
        for (int i = 0; i < L1COUNT + 1; i++)
        {
                fputs("\n|", stream);
                for (int j = 0; j < L2COUNT; j++)
                {
                        fprintf(stream, " %.4lf |", w12[i][j]);
                }
        }

        fprintf(stream, "\nw23:\n|");
        for (int i = 0; i < L2COUNT + 1; i++)
        {
                fprintf(stream, " %.4lf |", w23[i]);
        }
        fputc('\n', stream);
}
```

Генератор навчальних прикладів

```python
import random

OUTPUT = "examples.txt"
EXAMPLE_COUNT = 30

examples = []
while (len(examples) != EXAMPLE_COUNT):
   a = round(random.random(), 4)
   b = round(random.random(), 4)
   if a + b <= 1.0:
      examples.append((a, b, round(a + b, 6)))

with open(OUTPUT, 'w') as file:
   file.write(f"{EXAMPLE_COUNT}\n")
   for item in examples:
      file.write(f"{item[0]} {item[1]} {item[2]}\n")
```