



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине:

Введение в искусственный интеллект

Студент	Абидоков Рашид Ширамбиевич
Группа	РК6-11М
Вариант	1
Тема лабораторной работы	Реализация искусственной нейронной сети

Студент	_____	<u>Абидоков Р. Ш.</u>
	<i>подпись, дата</i>	<i>фамилия, и.о.</i>

Преподаватель	_____	<u>Федорук В. Г.</u>
	<i>подпись, дата</i>	<i>фамилия, и.о.</i>

Оценка _____

Москва, 2020 г.

Оглавление

Задание на лабораторную работу	3
Теоретические сведения	3
Примеры работы программы	5
Описание программной реализации.....	6
Процесс обучения.....	8

Задание на лабораторную работу

Разработать, используя язык C/C++, двухслойную нейронную сеть с линейными функциями активации, обеспечить ее обучение для решения задач сжатия данных с потерями.

Теоретические сведения

Искусственная нейронная сеть – сеть, в качестве вершин которой выступают искусственные нейроны. ИНС осуществляет преобразование вектора входных сигналов (воздействий) в вектор выходных сигналов. Выходной сигнал каждого нейрона формируется путем применения *функции активации* к взвешенной сумме входных сигналов.

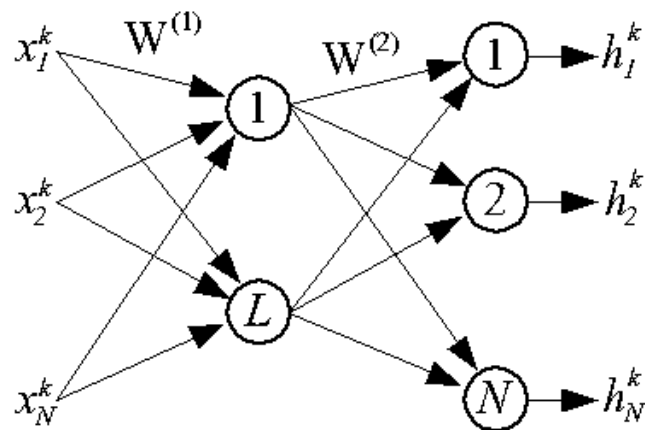


Рис 1. Схема двухслойной ИНС

Для сети с *линейной функцией активации* выход нейрона равен непосредственно взвешенной сумме. Т.е. вектор выходных сигналов каждого слоя формируется путем умножения вектора входных сигналов на матрицу весов. В этом случае для двухслойной ИНС с линейной функцией активации

$$\bar{h}^{(k)} = W_2 W_1 \bar{x}^{(k)}$$

где \bar{x} – вектор входных сигналов k -го объекта, W_1 – матрица весов первого слоя, W_2 – матрица весов второго слоя, \bar{h} – вектор выходных сигналов k -го объекта.

Обучение сети, состоящее в оптимальном подборе весов, составляющих матрицы \mathbf{W}_1 и \mathbf{W}_2 , подразумевает минимизацию целевой функции в виде:

$$E(\mathbf{W}_1, \mathbf{W}_2) = \frac{1}{2} \sum_{k=1}^K \sum_{i=1}^N (h_i - d_i)^2$$

где K – количество объектов выборки, N – размерность выходного вектора.

Поскольку в задаче сжатия изображения стоит задача минимизации отклонения полученного изображения от начального,

$$d_i = x_i$$

и целевая функция принимает вид

$$E(\mathbf{W}_1, \mathbf{W}_2) = \frac{1}{2} \sum_{k=1}^K \sum_{i=1}^N (h_i - x_i)^2$$

Для отыскания минимума используется метод обратного распространения ошибки. В рамках данного метода уточнение весовых коэффициентов производится формулой метода градиента

$$\mathbf{W}_i^{(n+1)} = \mathbf{W}_i^{(n)} - \eta \text{grad} E(\mathbf{W}_i^{(n)})$$

С учетом линейности функции активации выражения для частных производных выходного слоя:

$$\frac{\partial E(\mathbf{W})}{\partial w_{il}^{(2)}} = (h_i - x_i) g_l, \text{ где } g_l - \text{выход первого слоя}$$

Для первого слоя:

$$\frac{\partial E(\mathbf{W})}{\partial w_{lj}^{(1)}} = \sum_{i=1}^N (h_i - x_i) w_{il}^{(2)} x_j$$

Примеры работы программы

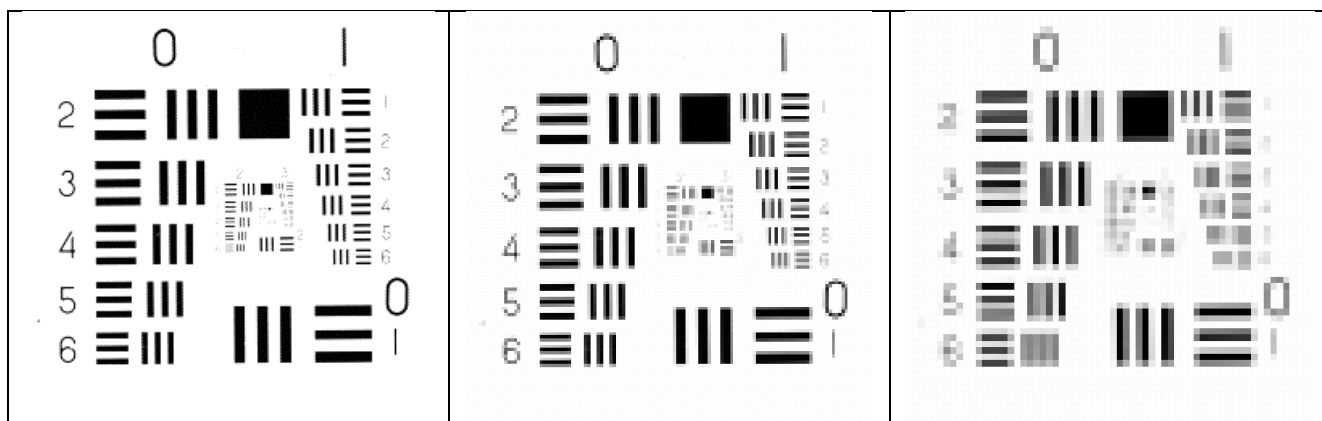
Для работы используются черно-белые изображения в формате .png с разрешением 256x256 пикселей. Исходное изображение разбивается на квадратные части $N \times N$ пикселей, которые пропускаются через сеть, затем собирается выходное изображение. Обучение сети производилось с помощью изображения, приведенного на Рис. 2



Рис 2.

Пример работы на некоторых изображениях

Исходное изображение	$N=2$	$N=4$
		
		



Описание программной реализации

Программа написана на языке C++ с использованием компилятора g++. Для работы с изображениями используется библиотека `stimg`. Алгоритм работы программы (см. Листинг 1):

1. С помощью функции `load_images` загружаются изображения с диска, формируется вектор объектов `CImg`
2. Вектор изображений функцией `images_to_matrices` преобразуется в вектор матриц размерности 256×256
3. Каждая из матриц функцией `matrices_to_cell_vecs` преобразуется в набор матриц размерности $N \times N$, где N – заданное число, а затем данные матрицы переразбиваются в вектора размерности $(N \times N) \times 1$ для последующей подачи в нейронную сеть
4. Создается объект нейронной сети, веса инициализируются малыми числами
5. По первому изображению в выборке происходит обучение, затем все изображения пропускаются через сеть
6. Пункты 1-3 выполняются в обратном порядке, собирается выходное изображение

```
30 int main() {
31     size_t SIDE = 256;
32     size_t CELL = 4;
33     // Загружаем изображения с диска
34     auto img_vec = load_images("src\\img\\", 3);
35     std::cout << "load_images done" << std::endl;
36     // Превращаем изображения в матрицы
37     auto img_m_vec = images_to_matrices(img_vec, SIDE);
38     std::cout << "images_to_matrix done" << std::endl;
39     // Переразбиваем каждую матрицу в несколько одномерных векторов
40     auto img_cv_vec = matrices_to_cell_vecs(img_m_vec, CELL, SIDE);
41     std::cout << "reshape done" << std::endl;
42     // Сама сеть
43     network n(CELL*CELL, CELL/2*CELL/2, CELL*CELL);
44     std::cout << "network create" << std::endl;
45     std::cout << "W1.n: " << n.W1_.get_n() << std::endl;
46     std::cout << "W1.m: " << n.W1_.get_m() << std::endl;
47     std::cout << "W2.n: " << n.W2_.get_n() << std::endl;
48     std::cout << "W2.m: " << n.W2_.get_m() << std::endl;
49     // Обучаемся по первой картинке
50     n.fit(img_cv_vec[0]);
51     std::cout << "fit done" << std::endl;
52     // Прогоняем всё через сеть
53     auto img_cv_vec_out = n.out(img_cv_vec);
54     std::cout << "n.out() done" << std::endl;
55     // Одномерные вектора обратно в матрицы
56     auto img_m_vec_out = cell_vecs_to_matrices(img_cv_vec_out, CELL, SIDE);
57     std::cout << "reshape2 done" << std::endl;
58     // Конвертим обратно в изображения
59     auto img_vec_out = matrices_to_images(img_m_vec_out, SIDE);
60     // Смотрим пробную картинку
61     auto test = img_vec_out[2];
62     test.display("2");
63     return 0;
64 }
```





Процесс обучения

Функция обучения сети приведена в Листинге 2

Листинг 2. Метод network.fit

```
61 void network::fit(const vector< matrix<double>> orig) {
62     // Количество проходов
63     for (int k = 0; k < 5; k++) {
64         // Проходимся по всем кусочкам и суммируем ошибку
65         int counter = 0;
66         for (auto input : orig) {
67             // Выход первого слоя
68             matrix<double>& out1_temp = W1_ * input;
69             // Выход второго слоя
70             matrix<double>& out2_temp = W2_ * out1_temp;
71             // Ошибки на выходе второго слоя
72             matrix<double>& errors2 = out2_temp - input;
73             // Прогоняем в обратную сторону
74             matrix<double>& W2_transp = W2_.transpose();
75             matrix<double>& errors1 = W2_transp * errors2;
76             // Корректируем веса
77             double l_c = 0.1; // Коэффициент обучения
78             // Второй слой
79             for (size_t i = 0; i < W2_.get_n(); i++) {
80                 for (size_t l = 0; l < W2_.get_m(); l++) {
81                     W2_[i][l] += -l_c * errors2[i][0] * out1_temp[l][0];
82                 }
83             }
84             delete &out1_temp;
85             delete &out2_temp;
86             delete &W2_transp;
87             delete &errors1;
88             delete &errors2;
89             counter++;
90         }
91     }
92 }
```


Для наглядности приведены первые пять итераций обучения

$i = 1$	
$i = 2$	
$i = 3$	
$i = 4$	

$i = 5$

