



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

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.

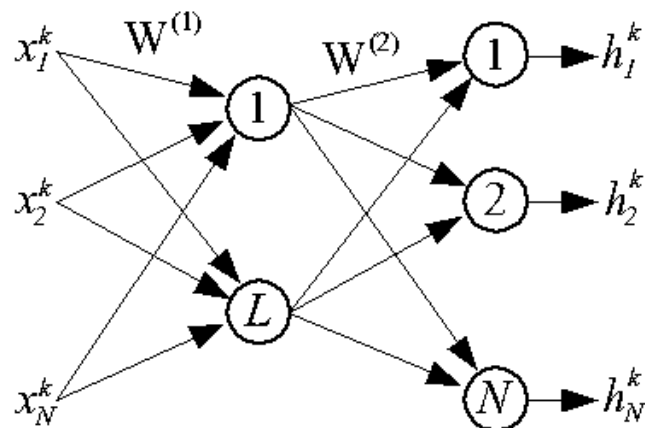


Рис 1. Схема двухслойной ИНС

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 прямоугольные части, которые пропускаются через сеть, затем собирается выходное изображение. Обучение сети производилось с помощью изображения, приведенного на Рис. 2






Рис 2. Исходное изображение

Пример работы при различных размерах ячейки

$N_h = 1, N_v = 1, L = 1$	$N_h = 2, N_v = 2, L = 1$	$N_h = 4, N_v = 4, L = 1$
		

Пример работы при различном количестве нейронов

$N_h = 4, N_v = 4, L = 1$	$N_h = 4, N_v = 4, L = 4$	$N_h = 4, N_v = 4, L = 16$
		

Описание программной реализации

Программа написана на языке C++ с использованием компилятора g++. Для работы с изображениями используется библиотека `img`.

Алгоритм работы программы (см. Листинг 1):

1. С помощью функции `load_images` загружаются изображения с диска, формируется вектор изображений
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 $N_h \times N_v$, где N_h, N_v – заданные числа, а затем данные матрицы переразбиваются в вектора размерности $N_h * N_v \times 1$ для последующей подачи в нейронную сеть
4. Создается нейронная сеть, по первому изображению в выборке происходит обучение
5. Все изображения пропускаются через сеть
6. Пункты 1-3 выполняются в обратном порядке, собираются выходные изображения

Алгоритм обучения нейронной сети (см. Листинг 2):

1. Веса W_1, W_2 инициализируются малыми случайными числами из интервала $(0.01, \frac{0.5}{\sqrt{N_h * N_v}})$ – интервал подобран эмпирически
2. Значение квадратичной ошибки на предыдущем проходе задается очень большим числом
3. Совершается проход по обучающему объекту (представляющему собой изображение в виде набора векторов размерности $N_h * N_v \times 1$) – для каждого вектора вычисляется выход сети, находятся ошибки на втором и первом слоях, веса модифицируются по формулам, приведенным ранее
4. Вычисляется значение квадратичной ошибки на текущем проходе, сравнивается с предыдущим – если разница меньше определенной или количество проходов достигло максимального, обучение завершается. Иначе – совершается еще один проход.

```

45 //=====
46 unsigned SIDE = 256;
47 unsigned Nh = 2;
48 unsigned Nv = 2;
49 unsigned L = 1;
50
51 v int main(int argc, char* argv[]) {
52     // Если передали размер ячейки
53 v     if (argc == 4) {
54         Nh = stoi(argv[1]);
55         Nv = stoi(argv[2]);
56         L = stoi(argv[3]);
57     }
58     std::cout << "Nh = " << Nh << " | Nv = " << Nv << " | L = " << L << std::endl;
59     // Загружаем изображения с диска
60     auto img_vec = load_images("./img_in/", 4);
61     std::cout << "load_images done" << std::endl;
62     // Превращаем изображения в матрицы
63     auto img_m_vec = images_to_matrices(img_vec, SIDE);
64     std::cout << "images_to_matrices done" << std::endl;
65     // Переразбиваем каждую матрицу в несколько одномерных векторов
66     auto img_cv_vec = matrices_to_cell_vecs(img_m_vec, Nh, Nv, SIDE);
67     std::cout << "matrices_to_cell_vecs done" << std::endl;
68     // Сама сеть
69     network n(Nh*Nv, L, Nh*Nv);
70     std::cout << "network create" << std::endl;
71     std::cout << "W1.n: " << n.W1_.get_n() << std::endl;
72     std::cout << "W1.m: " << n.W1_.get_m() << std::endl;
73     std::cout << "W2.n: " << n.W2_.get_n() << std::endl;
74     std::cout << "W2.m: " << n.W2_.get_m() << std::endl;
75     // Обучаемся по первой картинке
76     unsigned n_iters_max = 40;
77     unsigned n_iters = n.fit(img_cv_vec[0], n_iters_max)
78     // Прогоняем всё через сеть
79     auto img_cv_vec_out = n.out(img_cv_vec);
80     std::cout << "n.out done" << std::endl;

```

Листинг 1 (продолжение). Функция main

```
81 // Одномерные вектора обратно в матрицы
82 auto img_m_vec_out = cell_vecs_to_matrices(img_cv_vec_out, Nh, Nv, SIDE);
83 std::cout << "cell_vecs_to_matrices done" << std::endl;
84 // Конвертим обратно в изображения
85 auto img_vec_out = matrices_to_images(img_m_vec_out, SIDE);
86 std::cout << "matrices_to_images done" << std::endl;
87 // Записываем в файлы
88 write_images(img_vec_out, "./img_out/");
89 std::cout << "write out images done" << std::endl;
90 // Получаем картинки для шагов обучения
91 auto fit_steps_m_vec = cell_vecs_to_matrices(n.fit_steps, Nh, Nv, SIDE);
92 auto fit_steps_vec = matrices_to_images(fit_steps_m_vec, SIDE);
93 write_images(fit_steps_vec, "./fit_steps/");
94
95 std::cout << "write fit steps done" << std::endl;
96 std::cout << "======" << std::endl;
97 std::cout << "n_iterations: " << n_iters << std::endl;
98 return 0;
99 }
```


Листинг 2. Метод network.fit

```

75 unsigned network::fit(const vector< matrix<double>> orig, unsigned n_iters) {
76     // Коэффициент обучения
77     double l_c = 0.0000003;
78     // Проходы
79     double mse1 = 0, mse2 = 0;
80     double mse2_prev = 10000000;
81     for (unsigned k = 0; k < n_iters; k++) {
82         vector<matrix<double> > step_k;
83         // Проходимся по всем кусочкам и суммируем ошибку
84         int counter = 0;
85         for (auto input : orig) {
86             // Выход первого слоя
87             matrix<double>& out1_temp = W1_ * input;
88             // Выход второго слоя
89             matrix<double>& out2_temp = W2_ * out1_temp;
90             // Сохраняем в текущий шаг обучения
91             step_k.push_back(out2_temp);
92             // Ошибки на выходе второго слоя
93             matrix<double>& errors2 = out2_temp - input;
94             // Прогоняем в обратную сторону
95             matrix<double>& W2_transp = W2_.transpose();
96             matrix<double>& errors1 = W2_transp * errors2;
97             // Корректируем веса
98             // Второй слой
99             for (unsigned i = 0; i < W2_.get_n(); i++) {
100                 for (unsigned l = 0; l < W2_.get_m(); l++) {
101                     W2_[i][l] -= l_c * errors2[i][0] * out1_temp[l][0];
102                 }
103             }
104             // Первый слой
105             for (unsigned l = 0; l < W1_.get_n(); l++) {
106                 for (unsigned j = 0; j < W1_.get_m(); j++) {
107                     W1_[l][j] -= l_c * errors1[l][0] * input.ij(j, 0);
108                 }
109             }

```