ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»
(СИБГУТИ)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по дисциплине

“Структуры и алгоритмы обработки данных ”

на тему

Нейронные сети с обратным распространением ошибок. Классификация десятичных цифр.

Выполнил

студент Казарцев Евгений Сергеевич

Ф.И.О.

Группы ИВ-521

Работу принял _____ подпись

доцент к.т.н.

О.В. Молдованова

Оценка _____

Новосибирск – 2016

СОДЕРЖАНИЕ

Введение……...3
1 Теоретическая часть……..4
1.1 Искусственный нейрон……..…..4
1.2 Функция активации……..6
1.3 Описание алгоритма……7
2 Экспериментальное исследование эффективности алгоритма……...13
2.1 Организация моделирования……13
2.1.1 Целевая вычислительная машина…..……..……..13
2.1.2 Способ проведения……..13
2.2 Результаты моделирования……...13
2.2.1 Графики……13
2.2.2 Пример работы……18
3 Заключение……..26
4 Литература……...27
5 Приложение…….28

ВВЕДЕНИЕ

Теория нейронных сетей включают широкий круг вопросов из разных областей науки: биофизики, математики, информатики, схемотехники и технологии. Поэтому понятие "нейронные сети" детально определить сложно.

Искусственные нейронные сети (НС) — совокупность моделей биологических нейронных сетей. Представляют собой сеть элементов — искусственных нейронов — связанных между собой синаптическими соединениями. Сеть обрабатывает входную информацию и в процессе изменения своего состояния во времени формирует совокупность выходных сигналов. Работа сети состоит в преобразовании входных сигналов во времени, в результате чего меняется внутреннее состояние сети и формируются выходные воздействия. Обычно НС оперирует цифровыми, а не символьными величинами. Большинство моделей НС требуют обучения. В общем случае, обучение — такой выбор параметров сети, при котором сеть лучше всего справляется с поставленной проблемой. Обучение — это задача многомерной оптимизации, и для ее решения существует множество алгоритмов. Искусственные нейронные сети — набор математических и алгоритмических методов для решения широкого круга задач. [1].

1. Искусственный нейрон

Искусственный нейрон имитирует в первом приближении свойства биологического нейрона. На вход искусственного нейрона поступает некоторое множество сигналов, каждый из которых является выходом другого нейрона. Каждый вход умножается на соответствующий вес, аналогичный синаптической силе, и все произведения суммируются, определяя уровень активации нейрона. На рис. 1.2 представлена модель, реализующая эту идею. Хотя сетевые парадигмы весьма разнообразны, в основе почти всех их лежит эта конфигурация. Здесь множество входных сигналов, обозначенных x1, x2,…, xn, поступает на искусственный нейрон. Эти входные сигналы, в совокупности обозначаемые вектором X, соответствуют сигналам, приходящим в синапсы биологического нейрона. Каждый сигнал умножается на соответствующий вес w1, w2,…, wn, и поступает на суммирующий блок, обозначенный Σ. Каждый вес соответствует «силе» одной биологической синаптической связи. (Множество весов в совокупности обозначается вектором W.) Суммирующий блок, соответствующий телу биологического элемента, складывает взвешенные входы алгебраически, создавая выход, который мы будем называть NET. В векторных обозначениях это может быть компактно записано следующим образом: NET = XW.

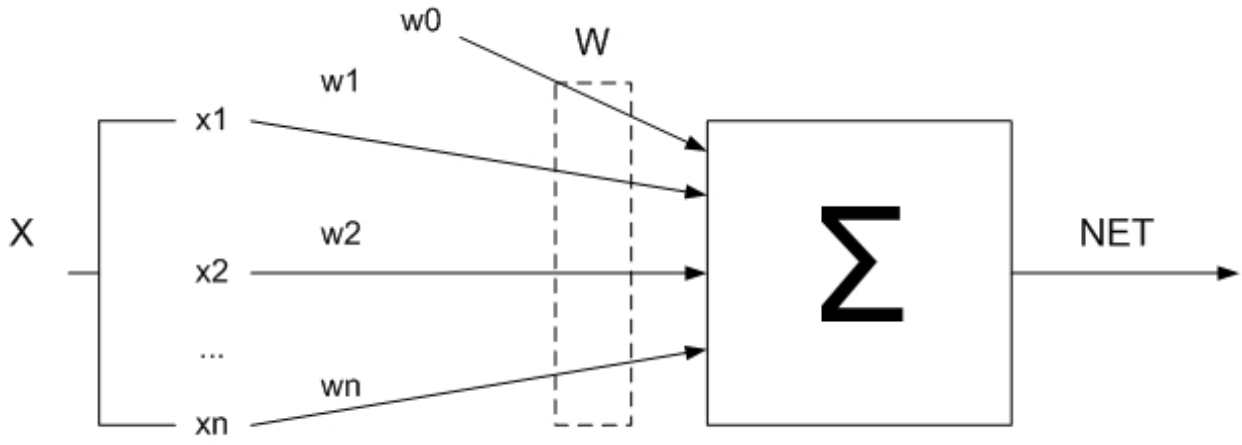

Рис. 1.2. Искусственный нейрон

Данное описание можно представить следующей формулой:

$$NET = \sum_{i=1}^{n} w_i x_i + w_0,$$

где w0 — биас;

wi — вес i- го нейрона;

xi — выход i- го нейрона;

n — количество нейронов, которые входят в обрабатываемый нейрон.

Сигнал w0, который имеет название биас, отображает функцию предельного значения, сдвига. Этот сигнал позволяет сдвинуть начало отсчета функции активации, которая в дальнейшем приводит к увеличению скорости обучения. Этот сигнал добавляется к каждому нейрону, он учится как и все другие весы, а его особенность в том, что он подключается к сигналу +1, а не к выходу предыдущего нейрона.

Полученный сигнал NET как правило обрабатывается функцией активации и дает выходной нейронный сигнал OUT (рис. 1.3)

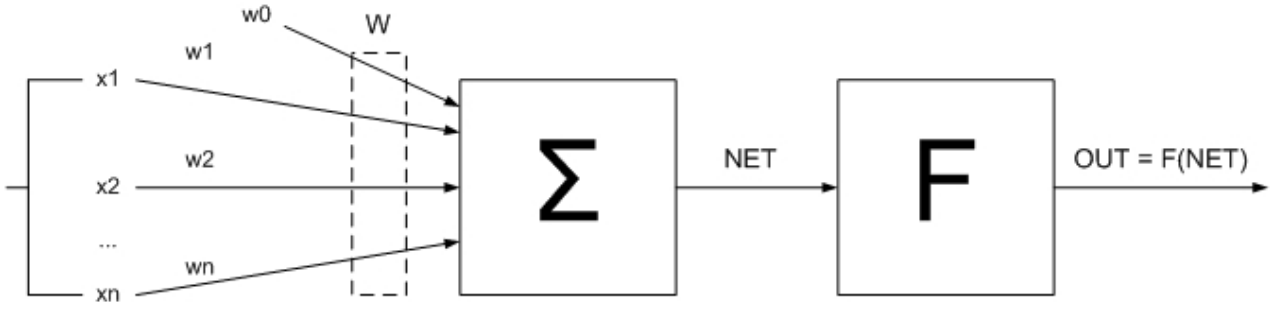

Рис. 1.3 Искусственный нейрон с функцией активации

2. Функция активации

Если функция активации суживает диапазон изменения величины NET так, что при каждом значении NET значения OUT принадлежат некоторому диапазону — конечному интервалу, то функция F называется функцией, которая суживает. В качестве этой функции часто используются логистическая или «сигмоидальная» функция. Эта функция математически выражается следующим образом:

$$OUT = \frac{1}{1 + e^{-NET}}$$

Основное преимущество такой функции — то, что она имеет простую производную и дифференцируется по всей оси абсцисс. График функции имеет следующий вид (рис. 1.4) [2. 3].

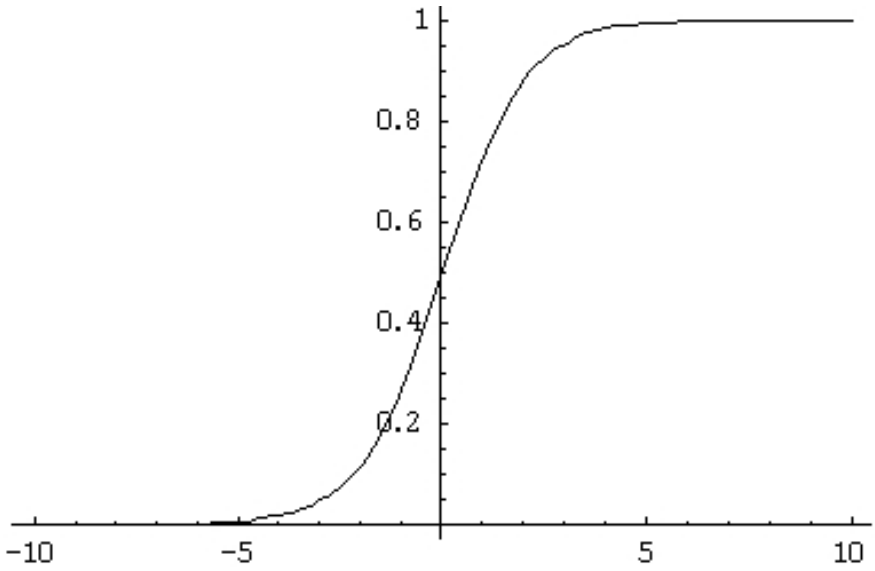

Рис. 1.4 Вид сигмоидной функции активации

3. Описание алгоритма

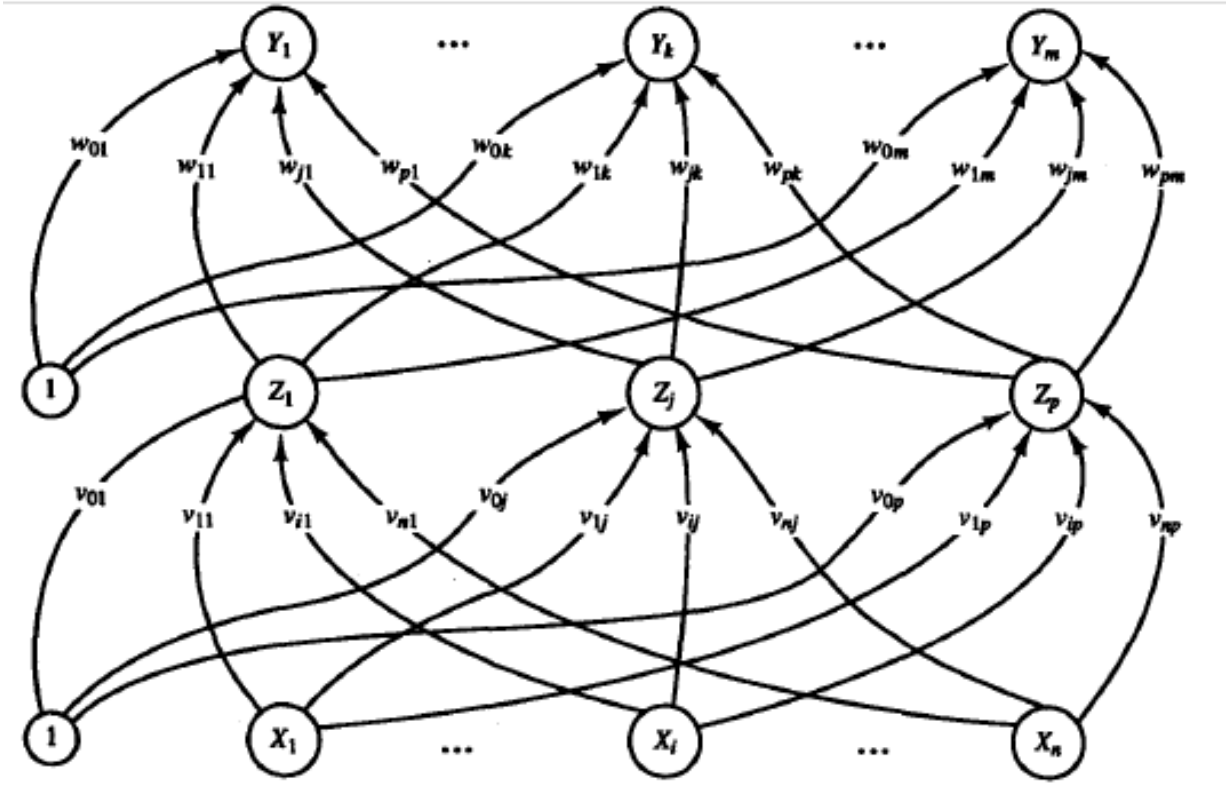

Рис 1.4 Структура сети

Описание алгоритма

Алгоритм, представленный далее, применим к нейронной сети с одним и более скрытым слоем, что является допустимой и адекватной ситуацией для большинства приложений. Как уже было сказано ранее, обучение сети включает в себя три стадии: подача на входы сети обучающих данных, обратное распространение ошибки и корректировка весов. В ходе первого этапа каждый входной нейрон x_i получает сигнал и широковещательно транслирует его каждому из скрытых нейронов $Z_1, Z_2, \ldots, Z_p$. Каждый скрытый нейрон затем вычисляет результат его активационной функции и рассылает свой сигнал z_j всем выходным нейронам. Каждый выходной нейрон y_k, в свою очередь, вычисляет результат своей активационной функции y_k, который представляет собой ничто иное, как выходной сигнал данного нейрона для

соответствующих входных данных. В процессе обучения, каждый нейрон на выходе сети сравнивает вычисленное значение y_k с предоставленным учителем t_k (целевым значением), определяя соответствующее значение ошибки для данного входного шаблона. На основании этой ошибки вычисляется $\sigma_k \ (k=1,2,\dots,m)$. σ_k используется при распространении ошибки от y_k до всех элементов сети предыдущего слоя (скрытых нейронов, связанных с y_k), а также позже при изменении весов связей между выходными нейронами и скрытыми. Аналогичным образом вычисляется $\sigma_j \ (j=1,2,\dots,p)$ для каждого скрытого нейрона Z_j. Несмотря на то, что распространять ошибку до входного слоя необходимости нет, σ_j используется для изменения весов связей между нейронами скрытого слоя и входными нейронами. После того как все σ были определены, происходит одновременная корректировка весов всех связей.

В алгоритме обучения сети используются следующие обозначения:

1.) x Входной вектор обучающих данных $x=(x_1, x_2, \dots, x_i, \dots, x_n)$.

2.) t Вектор целевых выходных значений, предоставляемых учителем $t=(t_1, t_2, \dots, t_k, \dots, t_m)$.

3.) σ_k Составляющая корректировки весов связей w_{jk}, соответствующая ошибке выходного нейрона y_k; также, информация об ошибке нейрона y_k, которая распространяется тем нейронам скрытого слоя, которые связаны с y_k.

4.) σ_j Составляющая корректировки весов связей vij, соответствующая

распространяемой от выходного слоя к скрытому нейрону Z_j информации об ошибке.

5.) α Скорость обучения.

6.) X_i Нейрон на входе с индексом i. Для входных нейронов входной и выходной сигналы одинаковы — x_i.

7.) v_{0j} Смещение скрытого нейрона j.

8.) Z_j Скрытый нейрон j; Суммарное значение подаваемое на вход скрытого элемента Z_j обозначается z_in_j: $z\_in_j = v_{0j} + \sum_i^n x_i * v_{ij}$

9.) Сигнал на выходе Z_j (результат применения к z_in_j активационной функции) обозначается z_j: $z_j = f(z\_in_j)$.

10.) w_{0k} Смещение нейрона на выходе.

11.) Y_k Нейрон на выходе под индексом k; Суммарное значение подаваемое на вход выходного элемента Y_k обозначается y_in_k:

$y\_in_k = w_{0k} + \sum_j z_j * w_{jk}$. Сигнал на выходе Y_k (результат применения к y_in_k активационной функции) обозначается Y_k.

Алгоритм прямого распространения сигнала и обратного распространения ошибки:

Шаг 0.

Инициализация весов (веса всех связей инициализируются случайными небольшими значениями).

Шаг 1.

До тех пор пока условие прекращения работы алгоритма неверно, выполняются шаги 2 — 9.

Шаг 2.

Для каждой пары {данные, целевое значение} выполняются шаги 3 — 8.Распространение данных от входов к выходам:

Шаг 3.

Каждый входной нейрон $(X_i, i = 1,2,\dots,n)$ отправляет полученный сигнал x_i всем нейронам в следующем слое (скрытом).

Шаг 4.

Каждый скрытый нейрон $(Z_j, j = 1,2,\dots,p)$ суммирует взвешенные входящие сигналы: $z\_in_j = v_{0j} + \sum_i^n x_i * v_{ij}$ и применяет активационную функцию: $z_j = f(z\_in_j).$ После чего посылает результат всем элементам следующего слоя (выходного).

Шаг 5.

Каждый выходной нейрон $(Y_k, k = 1,2,\dots,m)$ суммирует взвешенные входящие сигналы: $y\_in_k = w_{0k} + \sum_j z_j * w_{jk}$ и

применяет активационную функцию, вычисляя выходной сигнал:

$$y_k = f(y\_in_k).$$

Обратное распространение ошибки:

Шаг 6.

Каждый выходной нейрон $(Y_k,\ k = 1,2,\dots,m)$ получает целевое значение — то выходное значение, которое является правильным для данного входного сигнала, и вычисляет ошибку: $\sigma_k = (t_k - y_k) * f'(y\_in_k)$, так же вычисляет величину, на которую изменится вес связи w_{jk}:

$\Delta w_{jk} = a * \sigma_k * z_j$. Помимо этого, вычисляет величину корректировки смещения: $\Delta w_{0k} = a * \sigma_k$ и посылает σ_k нейронам в предыдущем слое.

Шаг 7.

Каждый скрытый нейрон $(Z_j,\ j = 1,2,\dots,p)$ суммирует входящие ошибки (от нейронов в последующем слое) $\sigma\_in_j = \sum_{k\in 1}^{n} \sigma_k * w_{jk}$ и вычисляет величину ошибки, умножая полученное значение на производную активационной функции: $\sigma_j = \sigma\_in_j * f'(z\_in_j)$, так же вычисляет величину, на которую изменится вес связи vij:

Помимо этого, вычисляет величину корректировки смещения:

$$v_{0j} = a * \sigma_j$$

Шаг 8. Изменение весов.

Каждый выходной нейрон $(Y_k, k = 1,2,\dots,m)$ изменяет веса своих связей с элементом смещения и скрытыми нейронами:

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$

Каждый скрытый нейрон $(Z_j, j = 1,2,\dots,p)$ изменяет веса своих связей с элементом смещения и выходными нейронами:

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$$

Шаг 9.

Проверка условия прекращения работы алгоритма.

Условием прекращения работы алгоритма может быть как достижение суммарной квадратичной ошибкой результата на выходе сети предустановленного заранее минимума в ходе процесса обучения, так и выполнения определенного количества итераций алгоритма. В основе алгоритма лежит метод под названием градиентный спуск. В зависимости от знака, градиент функции (в данном случае значение функции — это ошибка, а параметры — это веса связей в сети) дает направление, в котором значения функции возрастают (или убывают) наиболее стремительно. [4].

В программе смоделирована нейронная сеть с одним скрытым слоем размером 20 нейронов, входным слоем размером 15 нейронов, выходным слоем размером 10 нейронов. В качестве функции активации выбрана сигмоида с коэффициентом нелинейности 1. Начальное значение весов всех синоптических связей случайное в промежутка [-0.05; 0.05].

2.Экспериментальное исследование эффективности алгоритма

2.1 Организация моделирования

2.1.1. Целевая вычислительная машина

Все эксперименты проводились на сервере:

Память: 3 Gb 667 MHz, SATA III HDD.

Процессор: Intel® Core™ 2 Duo CPU E7300 2.66GHz × 2.

Операционная система: Ubuntu 16.04 Server , 64-bit.

2.1.2. Способ проведения экспериментов.

Построение графиков зависимости величины среднеквадратичной ошибки на одном примере от количества итераций обучения.

Пример работы: на обучающей выборке и на зашумлённых данных.

2.2 Результаты моделирования

2.2.1. Графики:

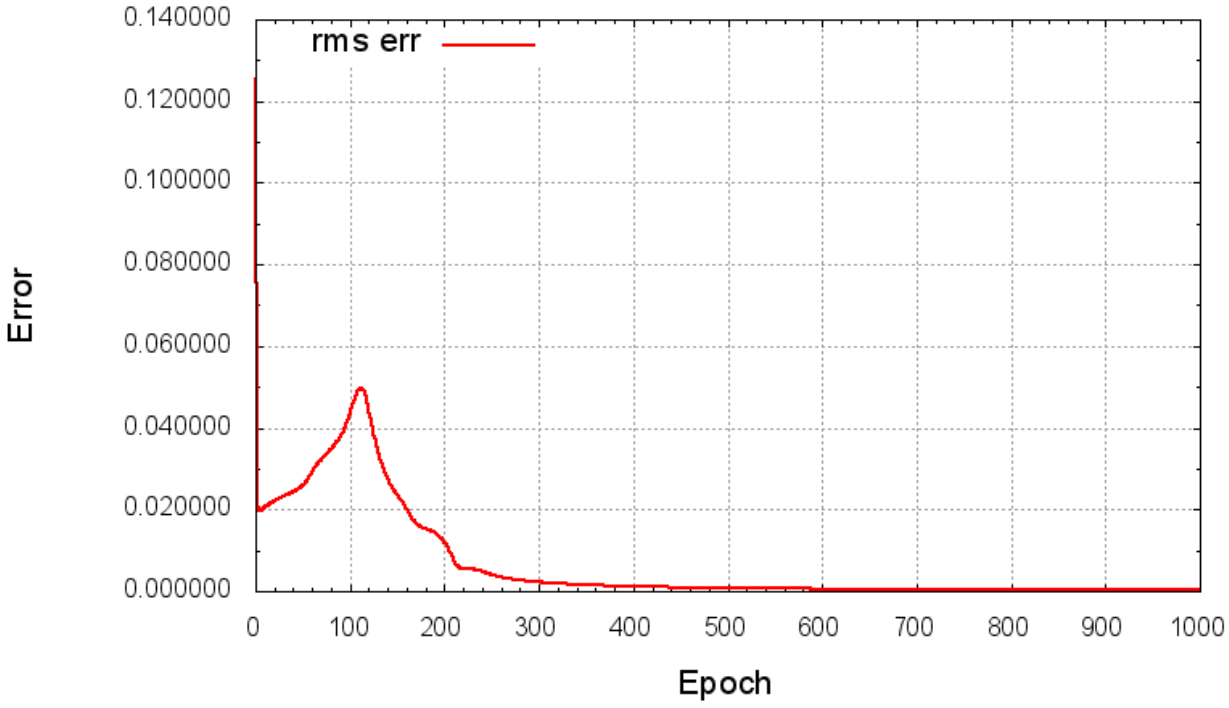

Рис. 2.1 Среднеквадратичная ошибка цифра «0».

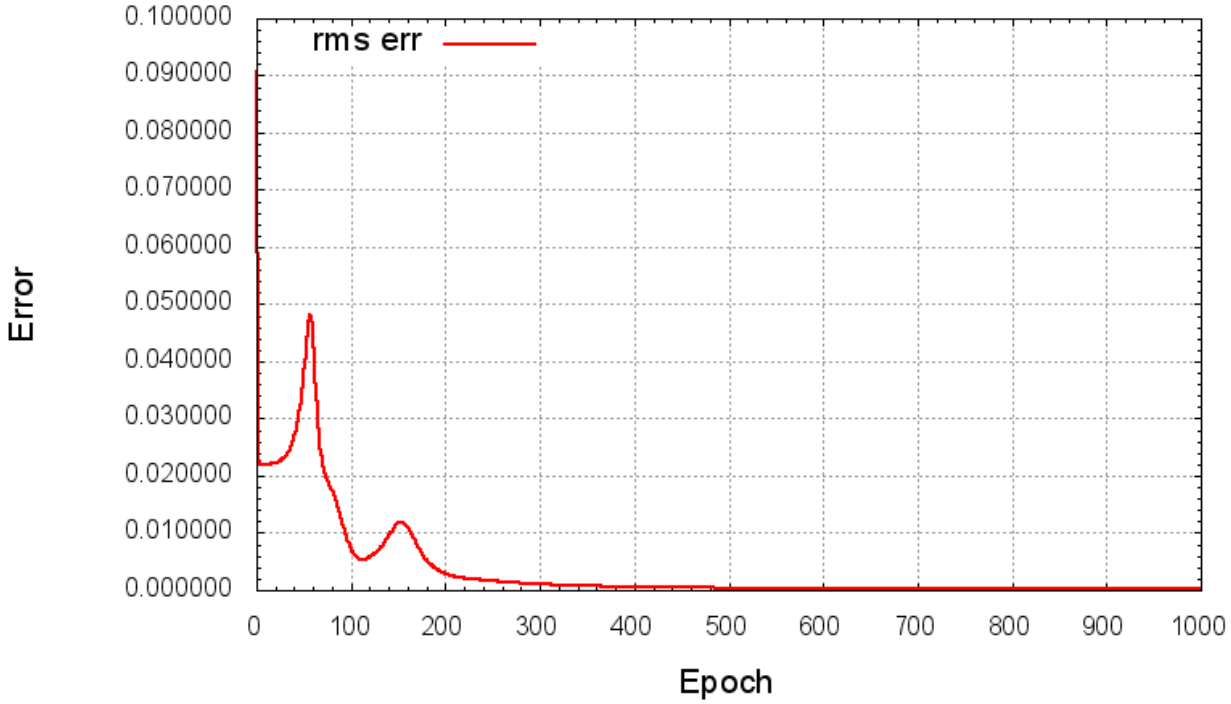

Рис. 2.2 Среднеквадратичная ошибка цифра «1».

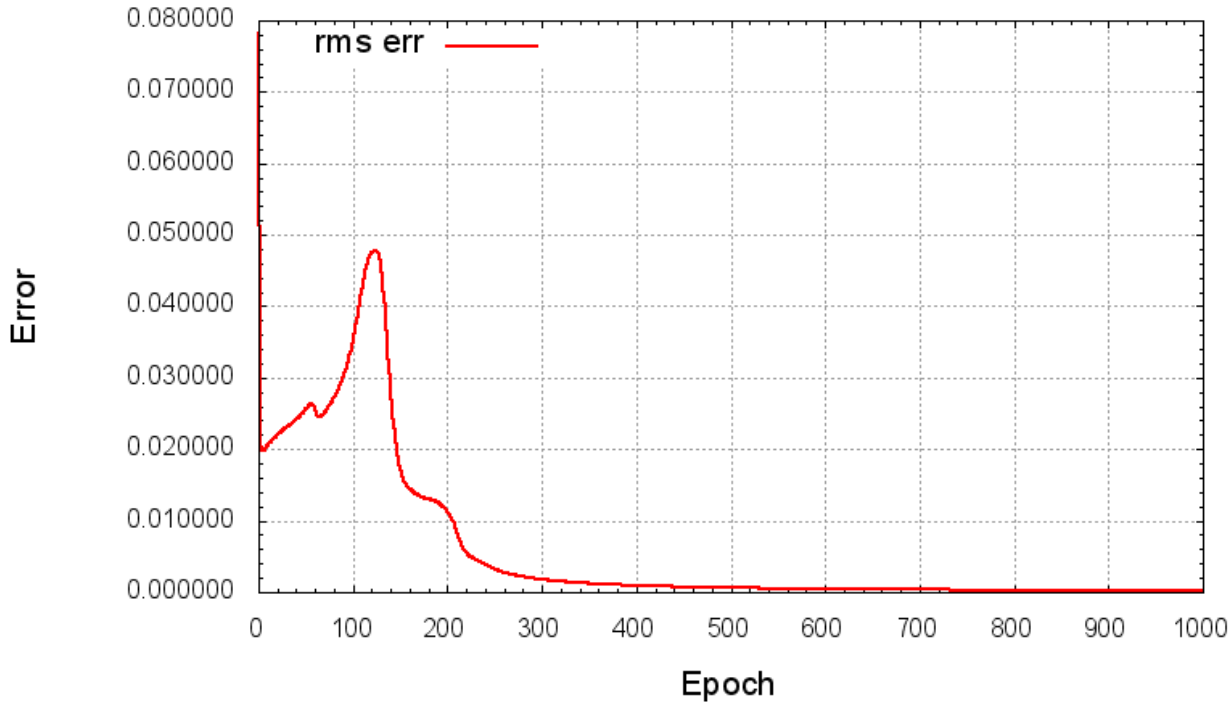

Рис. 2.3 Среднеквадратичная ошибка цифра «2».

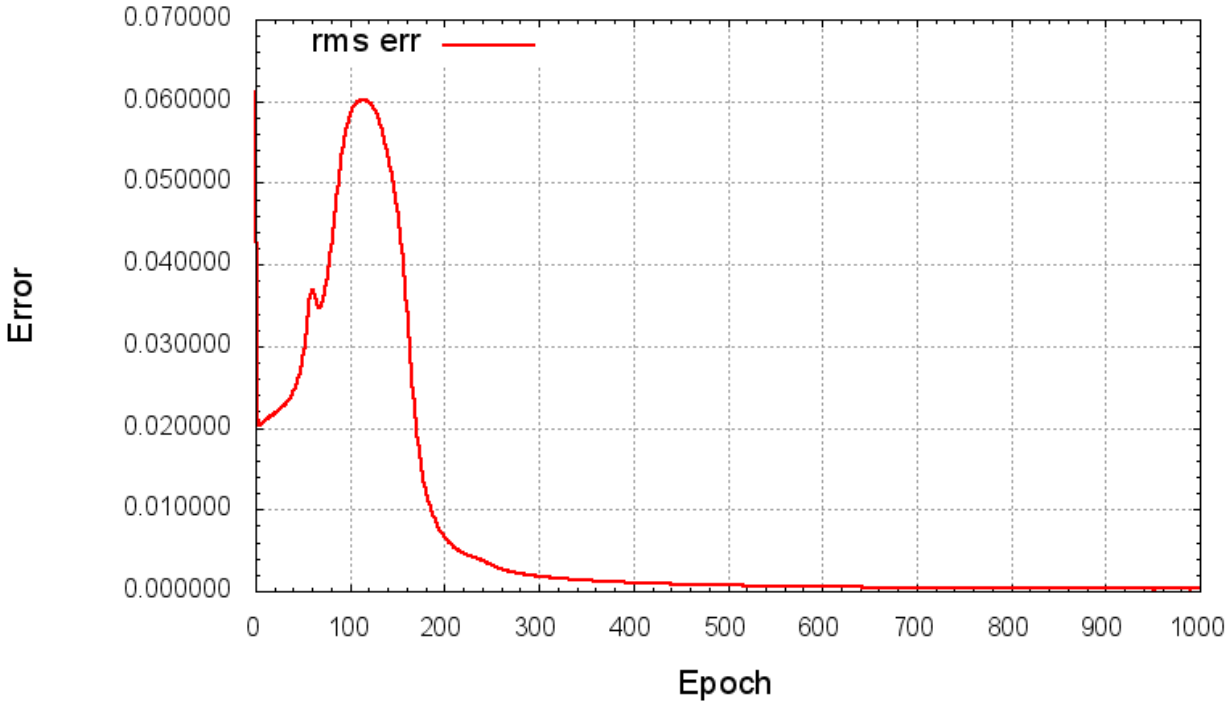

Рис. 2.4 Среднеквадратичная ошибка цифра «3».

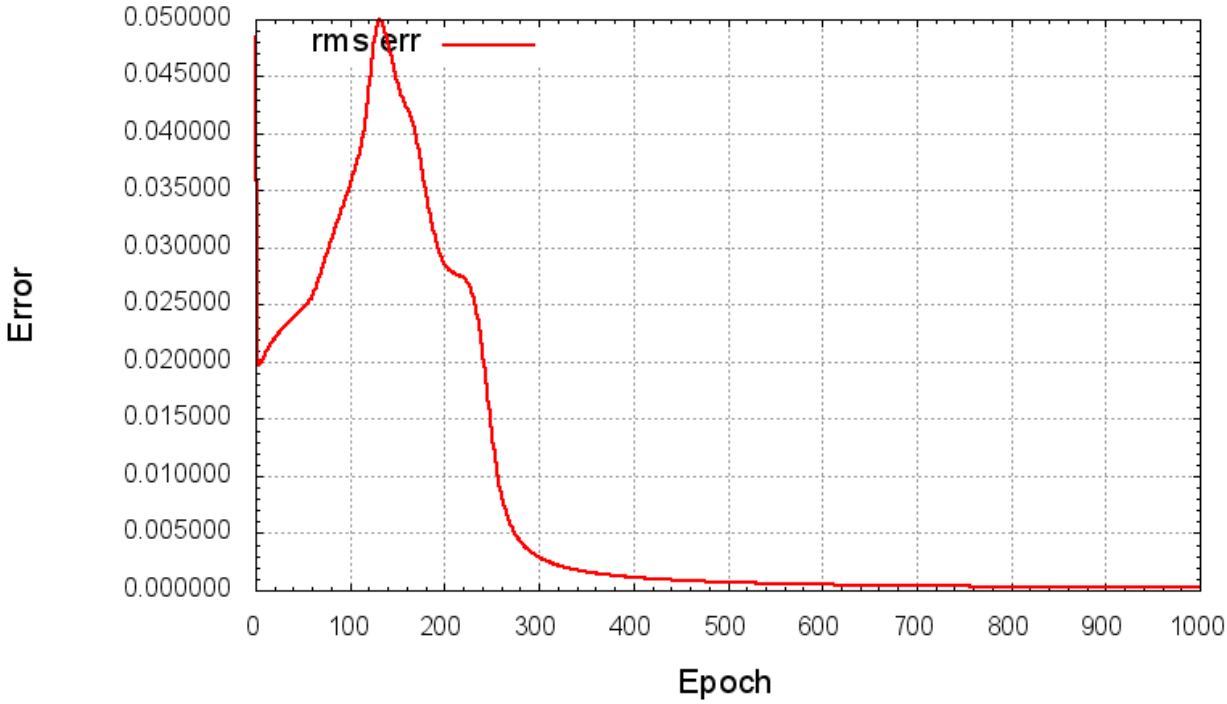

Рис. 2.5 Среднеквадратичная ошибка цифра «4».

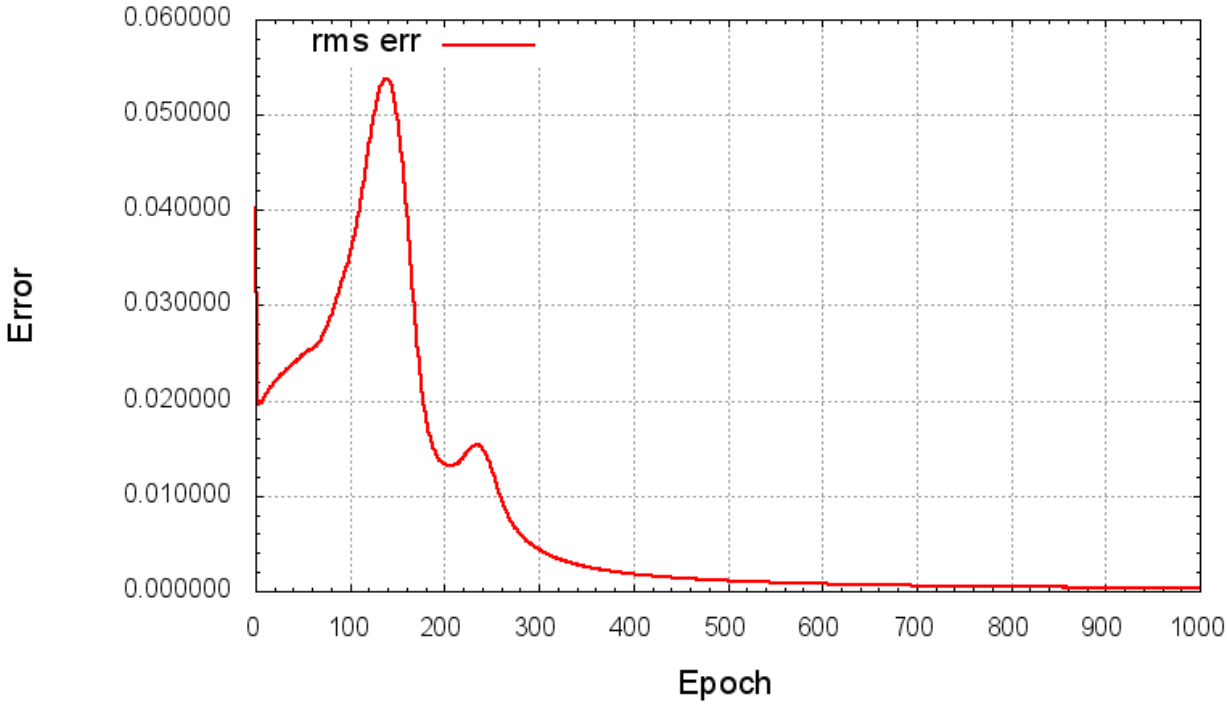

Рис. 2.6 Среднеквадратичная ошибка цифра «5».

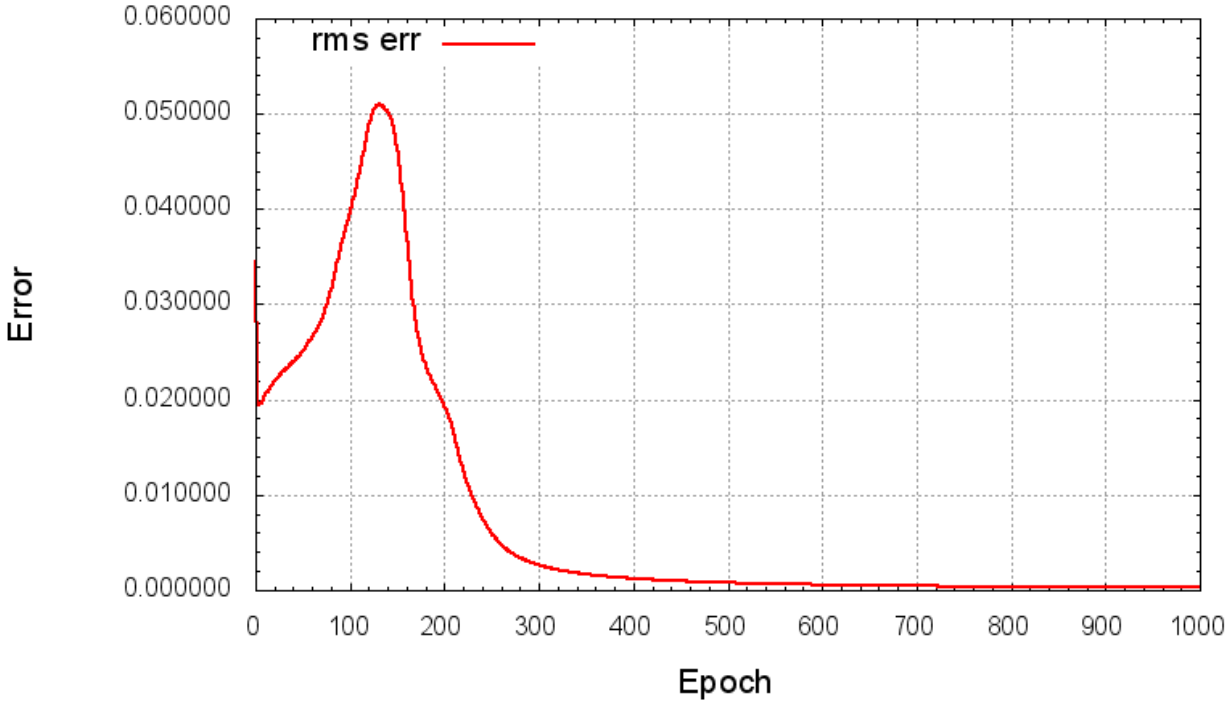

Рис. 2.7 Среднеквадратичная ошибка цифра «6».

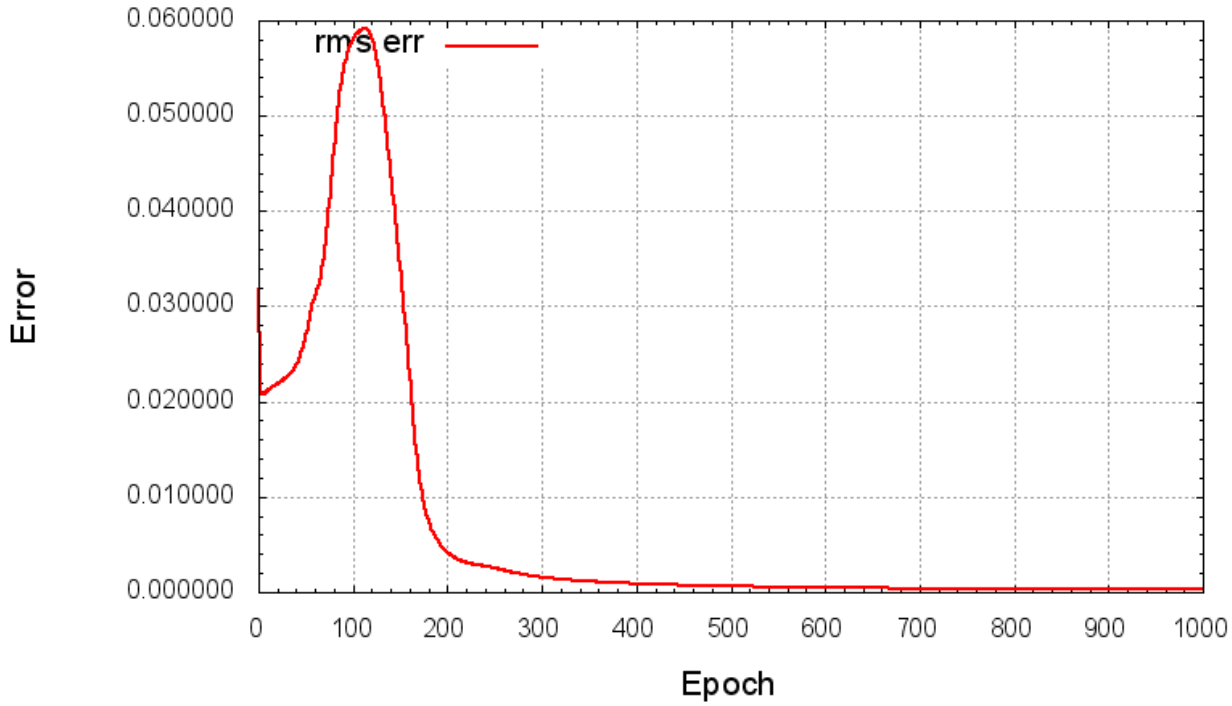

Рис. 2.8 Среднеквадратичная ошибка цифра «7».

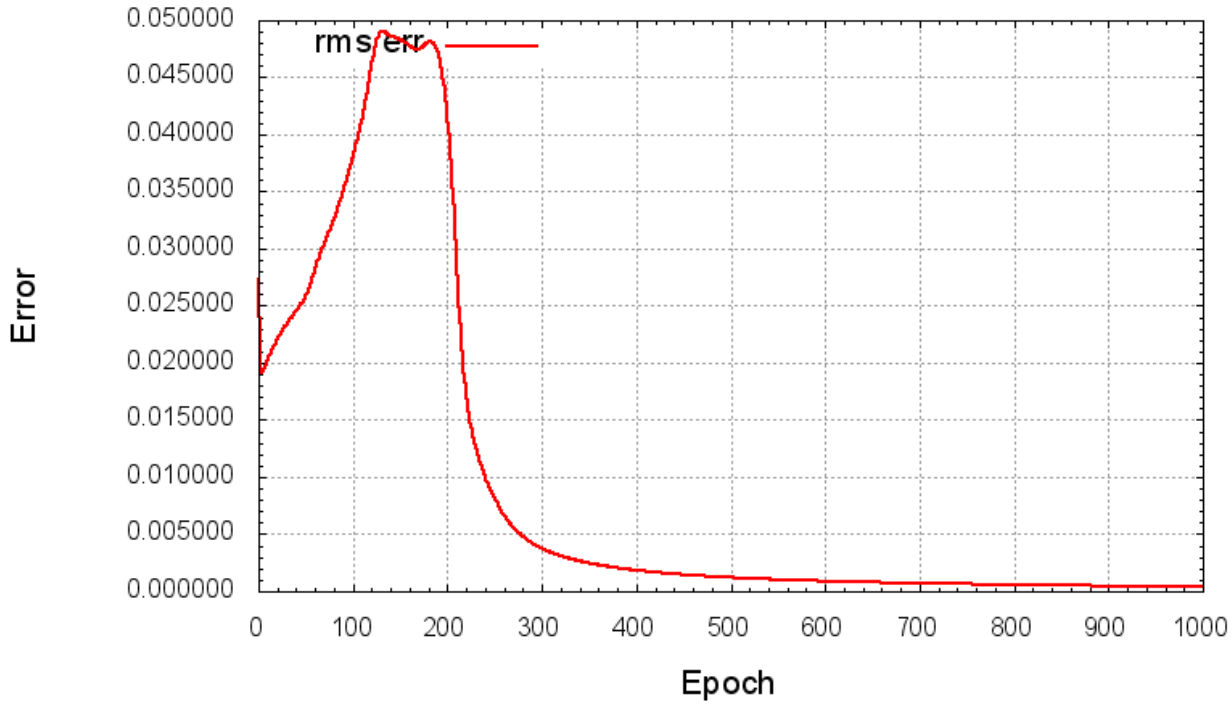

Рис. 2.9 Среднеквадратичная ошибка цифра «8».

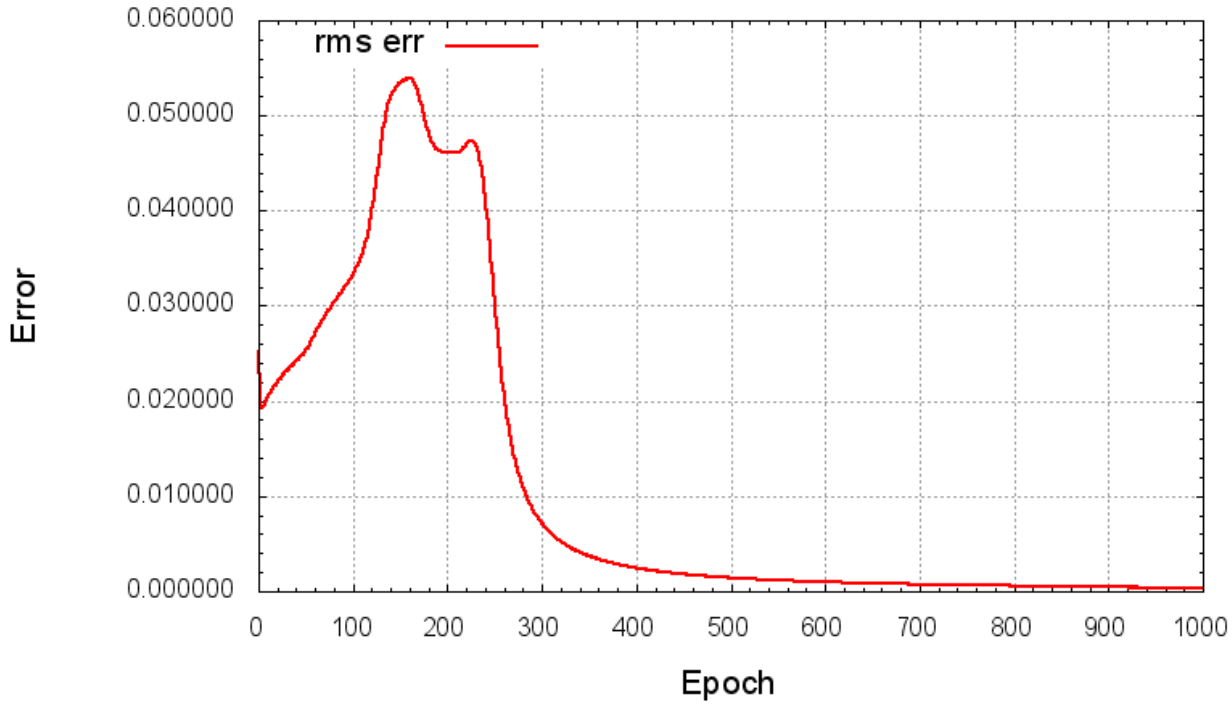

Рис. 2.10 Среднеквадратичная ошибка цифра «9»

2.2.2. Пример работы

Классификация обучающей выборки:

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/0.bmp"
1 1 1
1 0 1
1 0 1
1 0 1
1 1 1
output: 0 = 0.971312
3.522164
output: 1 = 0.000007
-11.895808
output: 2 = 0.000692
-7.275922
output: 3 = 0.000000
-16.132543
output: 4 = 0.003458
-5.663673
output: 5 = 0.000001
-14.000535
output: 6 = 0.000357
-7.938051
output: 7 = 0.014219
-4.238832
output: 8 = 0.021155
-3.834520
output: 9 = 0.001649
-6.405930
Ответ сети цифра: 0 точность: 0.971312
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.11 Классификация цифры «0»

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/1.bmp"
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
output: 0 = 0.000000
-16.181165
output: 1 = 0.969992
3.475811
output: 2 = 0.001892
-6.268009
output: 3 = 0.009503
-4.646650
output: 4 = 0.000001
-14.372059
output: 5 = 0.017992
-3.999695
output: 6 = 0.009705
-4.625369
output: 7 = 0.000293
-8.136037
output: 8 = 0.000000
-15.736994
output: 9 = 0.000206
-8.488017
Ответ сети цифра: 1 точность: 0.969992
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.12 Классификация цифры «1»

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/2.bmp"
1 1 1
0 0 1
1 1 1
1 0 0
1 1 1
output: 0 = 0.000198
-8.529218
output: 1 = 0.004304
-5.443873
output: 2 = 0.970530
3.494482
output: 3 = 0.015084
-4.178916
output: 4 = 0.000448
-7.709578
output: 5 = 0.000001
-14.254108
output: 6 = 0.003508
-5.649095
output: 7 = 0.000862
-7.055960
output: 8 = 0.019400
-3.922886
output: 9 = 0.000102
-9.190969
Ответ сети цифра: 2 точность: 0.970530
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.13 Классификация цифры «2»

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/3.bmp"
1 1 1
0 0 1
0 1 1
0 0 1
1 1 1
output: 0 = 0.000001
-14.295741
output: 1 = 0.013974
-4.256473
output: 2 = 0.016502
-4.087612
output: 3 = 0.970877
3.506654
output: 4 = 0.005091
-5.275157
output: 5 = 0.012805
-4.345014
output: 6 = 0.000007
-11.861185
output: 7 = 0.016022
-4.117619
output: 8 = 0.000009
-11.623689
output: 9 = 0.003391
-5.683142
Ответ сети цифра: 3 точность: 0.970877
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.14 Классификация цифры «3»

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/4.bmp"
1 0 1
1 0 1
1 1 1
0 0 1
0 0 1
output: 0 = 0.002675
-5.921246
output: 1 = 0.000008
-11.729532
output: 2 = 0.000020
-10.814048
output: 3 = 0.003951
-5.529924
output: 4 = 0.972192
3.554228
output: 5 = 0.000381
-7.872403
output: 6 = 0.000000
-16.549464
output: 7 = 0.004828
-5.328431
output: 8 = 0.017112
-4.050740
output: 9 = 0.021952
-3.796706
Ответ сети цифра: 4 точность: 0.972192
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.15 Классификация цифры «4»

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/5.bmp"
1 1 1
1 0 0
1 1 1
0 0 1
1 1 1
output: 0 = 0.000011
-11.385634
output: 1 = 0.013858
-4.264925
output: 2 = 0.000021
-10.747777
output: 3 = 0.013599
-4.284035
output: 4 = 0.000333
-8.006575
output: 5 = 0.972928
3.581811
output: 6 = 0.019302
-3.928067
output: 7 = 0.000205
-8.494307
output: 8 = 0.000041
-10.102560
output: 9 = 0.020058
-3.888849
Ответ сети цифра: 5 точность: 0.972928
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.16 Классификация цифры «5»

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/6.bmp"
1 1 1
1 0 0
1 1 1
1 0 1
1 1 1
output: 0 = 0.000234
-8.361779
output: 1 = 0.010232
-4.571935
output: 2 = 0.002527
-5.978293
output: 3 = 0.000029
-10.459291
output: 4 = 0.000009
-11.616704
output: 5 = 0.018658
-3.962661
output: 6 = 0.971383
3.524733
output: 7 = 0.000003
-12.761203
output: 8 = 0.019524
-3.916393
output: 9 = 0.000193
-8.550337
Ответ сети цифра: 6 точность: 0.971383
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.17 Классификация цифры «6»

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/7.bmp"
1 1 1
0 0 1
0 0 1
0 0 1
0 0 1
output: 0 = 0.017550
-4.024990
output: 1 = 0.004956
-5.302158
output: 2 = 0.007184
-4.928665
output: 3 = 0.020190
-3.882181
output: 4 = 0.004835
-5.326965
output: 5 = 0.000233
-8.363247
output: 6 = 0.000000
-14.582768
output: 7 = 0.971821
3.540605
output: 8 = 0.000000
-14.566872
output: 9 = 0.011978
-4.412669
Ответ сети цифра: 7 точность: 0.971821
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.18 Классификация цифры «7»

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/8.bmp"
1 1 1
1 0 1
1 1 1
1 0 1
1 1 1
output: 0 = 0.018765
-3.956793
output: 1 = 0.000055
-9.799388
output: 2 = 0.015649
-4.141575
output: 3 = 0.000040
-10.136352
output: 4 = 0.007872
-4.836481
output: 5 = 0.000008
-11.718379
output: 6 = 0.019458
-3.919831
output: 7 = 0.000030
-10.407940
output: 8 = 0.972470
3.564559
output: 9 = 0.013914
-4.260814
Ответ сети цифра: 8 точность: 0.972470
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.19 Классификация цифры «8»

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/8.bmp"
1 1 1
1 0 1
1 1 1
1 0 1
1 1 1
output: 0 = 0.018765
-3.956793
output: 1 = 0.000055
-9.799388
output: 2 = 0.015649
-4.141575
output: 3 = 0.000040
-10.136352
output: 4 = 0.007872
-4.836481
output: 5 = 0.000008
-11.718379
output: 6 = 0.019458
-3.919831
output: 7 = 0.000030
-10.407940
output: 8 = 0.972470
3.564559
output: 9 = 0.013914
-4.260814
Ответ сети цифра: 8 точность: 0.972470
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.20 Классификация цифры «9»

Классификация зашумлённых данных:

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/8.bmp"
1 1 1
1 1 1
1 1 1
1 1 1
1 1 1
output: 0 = 0.008386
-4.772797
output: 1 = 0.000174
-8.653698
output: 2 = 0.014838
-4.195637
output: 3 = 0.000014
-11.184055
output: 4 = 0.001961
-6.232530
output: 5 = 0.000005
-12.170168
output: 6 = 0.043701
-3.085693
output: 7 = 0.000013
-11.276545
output: 8 = 0.924870
2.510429
output: 9 = 0.002642
-5.933620
Ответ сети цифра: 8 точность: 0.924870
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.21 Классификация цифры «8»

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/2.bmp"
1 1 1
0 1 1
1 1 1
1 1 0
1 1 1
output: 0 = 0.000103
-9.180279
output: 1 = 0.016401
-4.093847
output: 2 = 0.964997
3.316677
output: 3 = 0.006352
-5.052537
output: 4 = 0.000129
-8.956448
output: 5 = 0.000001
-14.500050
output: 6 = 0.010775
-4.519661
output: 7 = 0.000474
-7.653434
output: 8 = 0.010640
-4.532476
output: 9 = 0.000018
-10.899945
Ответ сети цифра: 2 точность: 0.964997
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.22 Классификация цифры «2»

```
root@ubuserv:/dsk2/git/Neuro/build# ../bin/Neuro "../digits/3.bmp"
1 1 1
0 1 1
0 1 1
0 1 1
1 1 1
output: 0 = 0.000000
-14.965829
output: 1 = 0.096368
-2.238249
output: 2 = 0.029275
-3.501304
output: 3 = 0.922017
2.470075
output: 4 = 0.001152
-6.764821
output: 5 = 0.004980
-5.297327
output: 6 = 0.000021
-10.791898
output: 7 = 0.010977
-4.500950
output: 8 = 0.000006
-12.038663
output: 9 = 0.001934
-6.246309
Ответ сети цифра: 3 точность: 0.922017
root@ubuserv:/dsk2/git/Neuro/build#
```

Рис. 2.23 Классификация цифры «3»

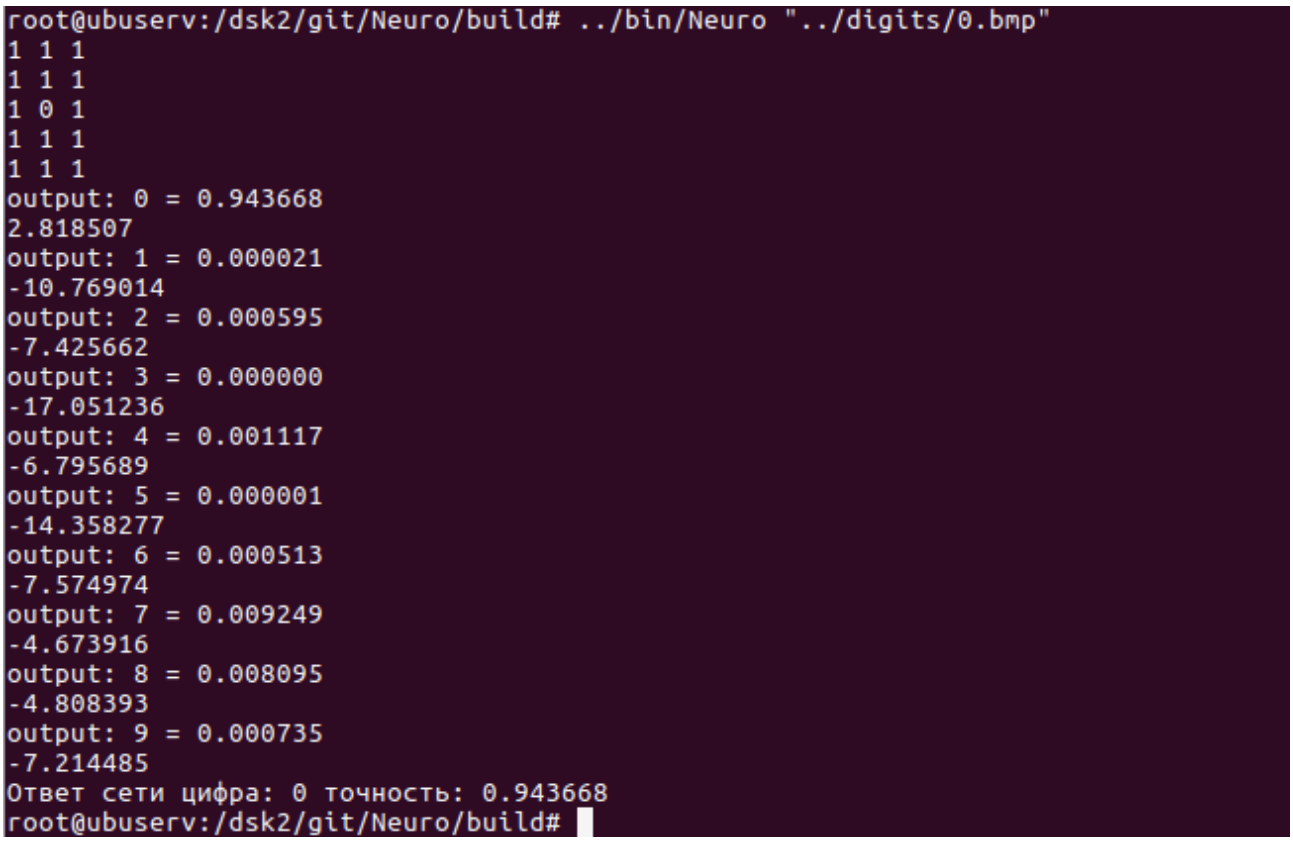

Рис. 2.24 Классификация цифры «0»

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была разработана и исследована такая структура данных, как искусственная нейронная сеть с обратным распределением ошибки .

Осуществлено моделирование нейронной сети и её обучение, что подтверждается графиками среднеквадратичной ошибки на обучающей выборке (произошла минимизация функции ошибки на каждом обучающем примере).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Заенцев И.В. Нейронные сети - основные модели (1999) — 3 стр.
2. Ф. Уоссермен. Нейрокомпьютерная техника: теория и практика. Перевод на русский язык Ю. А. Зуев, В. А. Точенов, 1992
3. И. В. Заенцев. Нейронные сети: основные модели. Учебное пособие к курсу “Нейронные сети”
4. https://habrahabr.ru/post/198268/

ПРИЛОЖЕНИЕ

ИСХОДНЫЙ ТЕКСТ ПРОГРАММЫ

Main.c (app)

```
#include <stdbool.h>
#include "neuro.h"
#include "./lib/qdbmp.h"


#define N_nodes_h_layer 20
#define N_nodes_o_layer 10

int main(int argc, char **argv)
{
  if (argc != 2) {
    printf("Error\n");
    return EXIT_FAILURE;
  }

  const char *input_path = argv[1];

  BMP *image = NULL;
  image = BMP_ReadFile(input_path);

  const char *err_str = NULL;

  err_str = BMP_GetErrorDescription();
  if (err_str != NULL) {
    printf("%s\n", err_str);
    return EXIT_FAILURE;
  }

  UINT width = BMP_GetWidth(image);
  UINT height = BMP_GetHeight(image);

  double **mass = calloc(height, sizeof(double *));
  for (int i = 0; i < height; i++) {
    mass[i] = calloc(width, sizeof(double));
  }


  UCHAR r, g, b;
  for (UINT y = 0; y < height; y++) {
    for (UINT x = 0; x < width; x++) {
      BMP_GetPixelRGB(image, x, y, &r, &g, &b);
      if ((r <= 30) && (g <= 30) && (b <= 30)) {
        mass[y][x] = 1;
      }
      else {
        mass[y][x] = 0;
      }
    }
  }

  err_str = BMP_GetErrorDescription();
  if (err_str != NULL) {
    printf("%s\n", err_str);
    return EXIT_FAILURE;
  }
/*
  //noise
  mass[1][1] = 1;
  mass[3][1] = 1;
*/
  int N_nodes_i_layer = (int) ((int) width * height);

  for(int i = 0; i < height; i++) {
    for(int j = 0; j < width; j++) {
      printf("%d ", (int) mass[i][j]);
    }
    printf("\n");
  }

  Neuron **input_layer = layer_create(N_nodes_i_layer, 1);
  Neuron **hidden_layer = layer_create(N_nodes_h_layer, N_nodes_i_layer);
  Neuron **output_layer = layer_create(N_nodes_o_layer, N_nodes_h_layer);

  tie_layers(input_layer, hidden_layer, N_nodes_i_layer, N_nodes_h_layer);
  tie_layers(hidden_layer, output_layer, N_nodes_h_layer, N_nodes_o_layer);

  srand((unsigned int) time(NULL));
  init_input_layer(input_layer, mass, (int) width, (int) height);
  init_next_layer(hidden_layer, N_nodes_h_layer, N_nodes_i_layer, "../src/memory/(w)i-h.txt");
  init_next_layer(output_layer, N_nodes_o_layer, N_nodes_h_layer, "../src/memory/(w)h-o.txt");

  input_layer_calc(input_layer, N_nodes_i_layer);
```

```
                next_layer_calc(hidden_layer, N_nodes_h_layer,
N_nodes_i_layer);
                next_layer_calc(output_layer, N_nodes_o_layer,
N_nodes_h_layer);

        double most;
        int digit;
        most = -2;
        digit = -1;
        for (int i = 0; i < N_nodes_o_layer; i++) {
          double tmp = output_layer[i]->output;
          if (tmp > most) {
            most = tmp;
            digit = i;
          }
        }

        for (int i = 0; i < N_nodes_o_layer; i++ ) {
          printf("output: %d = %f\n", i, output_layer[i]->output);
          printf("%f\n", output_layer[i]->sum_imp);
        }

         printf("Ответ сети цифра: %d точность: %f \n", digit,
most);

        return EXIT_SUCCESS;
      }
```

Neuro.h

```
      #ifndef NEURO_H
      #define NEURO_H

      #include <stdlib.h>
      #include <math.h>
      #include <time.h>
      #include <unistd.h>
      #include <stdio.h>
      #include <string.h>

      #define alf 1 //коэфициент наклона сигмоидной функции
      #define speed 1 //скорость обучения сети

      typedef struct {
        double **input; //массив входных импульсов
          double sum_imp; //суммарное значемни импульсов на
входе
          double act_weight; //смещение нейрона на выходе(вес
активаци)
        double output; //выходной импульс нейрона

          double *weight; //вес связи между нейроном данного
слоя и нейроном предыдущего слоя
          double err; //велечина ошибки
          double sum_err; //суммарная величина ошибки
           double *del_weight; //величина на которую изменяется
вес
      }Neuron;

      Neuron *neuron_create(int relations);
      Neuron **layer_create(int number_nodes, int relations);
      void tie_layers(Neuron **before, Neuron **after, int
n_nodes_before, int n_nodes_after);
      void init_input_layer(Neuron **neuron, double **mass, int
width, int height);
      void init_next_layer(Neuron **neuron, int number_nodes, int
number_nodes_before, char *file);
      double f_rand(double fMax, double fMin);
      void input_layer_calc(Neuron **neuron, int number_nodes);
      void next_layer_calc(Neuron **neuron, int number_nodes_l,
int number_nodes_before_l);
      void calc_err_and_delweight_out_layer(Neuron **after,
Neuron **before, int n_nodes_after, int n_nodes_before, double
*answer);
      void calc_err_and_delweight_prew_layer(Neuron **out,
Neuron **hid, Neuron **inp, int n_nodes_out, int n_nodes_hid, int
n_nodes_inp);
      void change_weight(Neuron **neuron, int n_nodes, int
n_nodes_before);
      double activation_func(double x);
      double derivative_activation_func(double x);
      double rms_err(Neuron **neuron, int n_nodes);
      void write_weigh(char *name, Neuron **neuron, int
n_nodes, int n_nodes_before);

      #endif
```

Neuro.c

```
      #include "neuro.h"

      Neuron *neuron_create(int relations)
      {
        Neuron *neuron = NULL;
        neuron = malloc(sizeof(*neuron));
        if (neuron != NULL) {
          neuron->input = malloc(relations * sizeof(double *));
                    neuron->del_weight = malloc(relations *
sizeof(double));
          neuron->weight = malloc(relations * sizeof(double));
          neuron->output = 0;
        }
        return neuron;
      }
```

```
Neuron **layer_create(int number_nodes, int relations)
{
  Neuron **layer = NULL;
  layer = malloc(number_nodes * sizeof(*layer));
  if (layer != NULL) {
    for (size_t i = 0; i < number_nodes; i++) {
      layer[i] = neuron_create(relations);
      if (layer[i] == NULL)
        return NULL;
    }
  }
  return layer;
}

void tie_layers(Neuron **before, Neuron **after, int n_nodes_before, int n_nodes_after)
{
  for (int i = 0; i < n_nodes_after; i++) {
    for (int j = 0; j < n_nodes_before; j++) {
      after[i]->input[j] = &before[j]->output;
    }
  }
}

void init_input_layer(Neuron **neuron, double **mass, int width, int height)
{
  int place = 0;
  for (int i = 0; i < width; i++) {
    for (int j = 0; j < height; j++) {
      neuron[place]->input[0] = &mass[i][j];
      place++;
    }
  }
}

void init_next_layer(Neuron **neuron, int number_nodes, int number_nodes_before, char *file)
{
  FILE *inp = NULL;
  inp = fopen(file, "r");
  if (inp == NULL) {
    for (int i = 0; i < number_nodes; i++) {
      for (int j = 0; j < number_nodes_before; j++) {
        neuron[i]->weight[j] = f_rand(0.05, -0.05);
      }
      neuron[i]->act_weight = f_rand(0.05, -0.05);
    }
  }
  else {
    float w;
    for (int i = 0; i < number_nodes; i++) {
      for (int j = 0; j < number_nodes_before; j++) {
        fscanf(inp, "%f", &w);
        neuron[i]->weight[j] = w;
      }
    }
    for (int i = 0; i < number_nodes; i++) {
      fscanf(inp, "%f", &w);
      neuron[i]->act_weight = w;
    }
    fclose(inp);
  }
}

double f_rand(double fMax, double fMin)
{
  double f = (double) rand() / RAND_MAX;
  return fMin + f * (fMax - fMin);
}

void input_layer_calc(Neuron **neuron, int number_nodes)
{
  for (int i = 0; i < number_nodes; i++) {
    neuron[i]->output = *neuron[i]->input[0];
  }
}

void next_layer_calc(Neuron **neuron, int number_nodes_l, int number_nodes_before_l)
{
  for (int i = 0; i < number_nodes_l; i++) {
    for (int j = 0; j < number_nodes_before_l; j++) {
      neuron[i]->sum_imp += ((*neuron[i]->input[j]) * (neuron[i]->weight[j]));
    }
    neuron[i]->sum_imp += neuron[i]->act_weight;
    neuron[i]->output = activation_func(neuron[i]->sum_imp);
  }
}

void calc_err_and_delweight_out_layer(Neuron **after, Neuron **before, int n_nodes_after, int n_nodes_before, double *answer)
{
  for (int i = 0; i < n_nodes_after; i++) {
    after[i]->err = (answer[i] - after[i]->output) * derivative_activation_func(after[i]->sum_imp);
    after[i]->act_weight += speed * after[i]->err;
    for (int j = 0; j < n_nodes_before; j++) {
      after[i]->del_weight[j] = speed * after[i]->err * before[j]->output;
    }
  }
}
```

```
void calc_err_and_delweight_prew_layer(Neuron **out, Neuron **hid, Neuron **inp, int n_nodes_out, int n_nodes_hid, int n_nodes_inp)
{
  for (int i = 0; i < n_nodes_hid; i++) {
    for (int j = 0; j < n_nodes_out; j++) {
      hid[i]->sum_err += out[j]->err * out[j]->weight[i];
    }
    hid[i]->err = hid[i]->sum_err * derivative_activation_func(hid[i]->sum_imp);
  }
  for (int i = 0; i < n_nodes_inp; i++) {
    for (int j = 0; j < n_nodes_hid; j++) {
      hid[j]->del_weight[i] = speed * hid[j]->err * inp[i]->output;
      hid[j]->act_weight += speed * hid[j]->err;
    }
  }
}

void change_weight(Neuron **neuron, int n_nodes, int n_nodes_before)
{
  for (int i = 0; i < n_nodes; i++) {
    for (int j = 0; j < n_nodes_before; j++) {
      neuron[i]->weight[j] += neuron[i]->del_weight[j];
    }
  }
}

double activation_func(double x)
{
  double y = 1 / (1 + exp((-1) * x * alf));
  return y;
}

double derivative_activation_func(double x)
{
  double z = activation_func(x);
  double y = alf * z * (1 - z);
  return y;
}

double rms_err(Neuron **neuron, int n_nodes)
{
  double sum_sq = 0;
  for (int i = 0; i < n_nodes; i++) {
    sum_sq += pow(neuron[i]->err, 2);
  }
  return sqrt(sum_sq/n_nodes);
}

void write_weigh(char *name, Neuron **neuron, int n_nodes, int n_nodes_before)
{
  FILE *out = NULL;
  out = fopen(name, "w");
  for (int i = 0; i < n_nodes; i++) {
    for (int j = 0; j < n_nodes_before; j++) {
      fprintf(out, "%f\n", neuron[i]->weight[j]);
    }
  }
  for (int i = 0; i < n_nodes; i++) {
    fprintf(out, "%f\n", neuron[i]->act_weight);
  }
  fclose(out);
}
```

Main.c (training)

```
#include "../app/neuro.h"
#include "../app/lib/qdbmp.h"
#include <string.h>

#define N_nodes_h_layer 20
#define N_nodes_o_layer 10

int main()
{
  char *input_path;
  for (size_t j = 0; j < 1000; j++) {
    for (size_t i = 0; i <= 9; i++) {
      input_path = return_path(i);
      //printf("%s\n", input_path);
      BMP *image = NULL;
      image = BMP_ReadFile(input_path);

      const char *err_str = NULL;

      err_str = BMP_GetErrorDescription();
      if (err_str != NULL) {
        printf("%s\n", err_str);
        return EXIT_FAILURE;
      }

      UINT width = BMP_GetWidth(image);
      UINT height = BMP_GetHeight(image);

      double **mass = calloc(height, sizeof(double *));
      for (int i = 0; i < height; i++) {
        mass[i] = calloc(width, sizeof(double));
      }


      UCHAR r, g, b;
      for (UINT y = 0; y < height; y++) {
        for (UINT x = 0; x < width; x++) {
```

```
                BMP_GetPixelRGB(image, x, y, &r, &g, &b);
                if ((r <= 30) && (g <= 30) && (b <= 30)) {
                    mass[y][x] = 1;
                }
                else {
                    mass[y][x] = 0;
                }
            }
        }

        err_str = BMP_GetErrorDescription();
        if (err_str != NULL) {
            printf("%s\n", err_str);
            return EXIT_FAILURE;
        }

        int N_nodes_i_layer = (int) ((int) width * height);

        /*for(int i = 0; i < height; i++) {
            for(int j = 0; j < width; j++) {
                printf("%d ", (int) mass[i][j]);
            }
            printf("\n");
        }*/

                                    Neuron **input_layer = layer_create(N_nodes_i_layer, 1);
                                    Neuron **hidden_layer = layer_create(N_nodes_h_layer, N_nodes_i_layer);
                                    Neuron **output_layer = layer_create(N_nodes_o_layer, N_nodes_h_layer);

                                    tie_layers(input_layer, hidden_layer, N_nodes_i_layer, N_nodes_h_layer);
                                    tie_layers(hidden_layer, output_layer, N_nodes_h_layer, N_nodes_o_layer);

        srand((unsigned int) time(NULL));
            init_input_layer(input_layer, mass, (int) width, (int) height);
                init_next_layer(hidden_layer, N_nodes_h_layer, N_nodes_i_layer, "../src/memory/(w)i-h.txt");
                init_next_layer(output_layer, N_nodes_o_layer, N_nodes_h_layer, "../src/memory/(w)h-o.txt");

            double **exp_response = calloc(10, sizeof(double *));
        for (int i = 0; i < 10; i++) {
            exp_response[i] = calloc(10, sizeof(double));
        }
        for (int i = 0; i < 10; i++) {
            exp_response[i][i] = 1;
        }
        double most;
        int digit;

        input_layer_calc(input_layer, N_nodes_i_layer);
            next_layer_calc(hidden_layer, N_nodes_h_layer, N_nodes_i_layer);
            next_layer_calc(output_layer, N_nodes_o_layer, N_nodes_h_layer);

        most = -2;
        digit = -1;
        for (int i = 0; i < N_nodes_o_layer; i++) {
            double tmp = output_layer[i]->output;
            if (tmp > most) {
                most = tmp;
                digit = i;
            }
        }
            //printf("Ответ сети цифра: %d точность: %f \n", digit, most);
            calc_err_and_delweight_out_layer(output_layer, hidden_layer, N_nodes_o_layer, N_nodes_h_layer, exp_response[i]);
        double rms;
        rms = rms_err(output_layer, N_nodes_o_layer);
        printf("epoch %ld\n", j);
        printf("rms_err %f\n", rms);
        FILE *tb;
        char *out = calloc(30, sizeof(char));
        char *name = "../testData/rms_err";
        char *type = ".dat";
        strcat(out, name);
        char n[1];
        itoa(i, n);
        strcat(out, n);
        strcat(out, type);
        tb = fopen(out, "a");
        fprintf(tb, "%ld %.6f\n", j, rms);
        fclose(tb);
        if (digit != i || rms > 0.0003) {
            printf("неправильно\n");
            calc_err_and_delweight_prew_layer(output_layer, hidden_layer, input_layer, N_nodes_o_layer, N_nodes_h_layer, N_nodes_i_layer);
            change_weight(output_layer, N_nodes_o_layer, N_nodes_h_layer);
            change_weight(hidden_layer, N_nodes_h_layer, N_nodes_i_layer);
                write_weigh("../src/memory/(w)i-h.txt", hidden_layer, N_nodes_h_layer, N_nodes_i_layer);
                write_weigh("../src/memory/(w)h-o.txt", output_layer, N_nodes_o_layer, N_nodes_h_layer);
        }
        else {
```

```
    printf("правильно\n");
}


free(input_path);
        }
    }
    return 0;
}
```