Нижегородский государственный университет им. Н.И. Лобачевского Институт информационных технологий, математики и механики Кафедра Математического обеспечения и суперкомпьютерных технологий

**Образовательный курс «Методы глубокого обучения для решения задач компьютерного зрения»**

**Лабораторная работa №1:**

**Реализация метода обратного распространения ошибки для двуслойной полностью связанной нейронной сети**

Выполнил:

Чугунов Е. А.

Преподаватель:

Кустикова В. Д.

Нижний Новгород

2017

Оглавление

[Постановка задачи 3](#_Toc501807324)

[Вывод необходимых формул 3](#_Toc501807325)

[Функция активации 3](#_Toc501807326)

[Кросс-энтропия 3](#_Toc501807327)

[Метод обратного распространения ошибки 4](#_Toc501807328)

[Алгоритм работы программы 5](#_Toc501807329)

[Данные для обучения 5](#_Toc501807330)

[Результаты работы 6](#_Toc501807331)

[Расчет ошибки 6](#_Toc501807332)

[Результаты тестирования 6](#_Toc501807333)

[Программная реализация 7](#_Toc501807334)

[Заключение 8](#_Toc501807335)

[Литература 9](#_Toc501807336)

[Приложения 10](#_Toc501807337)

[Код функции train 10](#_Toc501807338)

[Код функции test 11](#_Toc501807339)

# Постановка задачи

Необходимо написать программу, реализующую двуслойную полностью связанную нейронную сеть. Проверить работоспособность нейронной сети на задаче классификации рукописных цифр из набора данных MNIST. В качестве функции активации на втором слое используется функция softmax. В качестве функции ошибки используется кросс-энтропия.

# Вывод необходимых формул

## Функция активации

Когда задача классификации требует распознавания для более чем двух классов, в качестве функции активации обычно применяется softmax для результирующего слоя. Softmax функция предоставляет способ для предсказания распределения вероятности между классами. В качестве функции ошибки используется кросс-энтропия.

Softmax функция активации для i-того результата:

, где - выходные значения слоя. – число нейронов на слое.

Функция активации на скрытом слое – сигмоида:

## Кросс-энтропия

Кросс энтропия в качестве функции ошибки для распознавания на более чем два класса:

, где – желаемый результирующий вектор, – полученный в результате вычисления результат.

Вычисление градиента:

Градиент для весов на последнем слое:

Градиент для нейронов скрытого слоя:

## Метод обратного распространения ошибки

1. Подать значения интенсивностей на вход и подсчитать значения на скрытом слое.
2. Применить функцию активации к значениям на скрытом слое:
3. Рассчитать значения на выходном слое.
4. Применить функцию активации к выходным значениям на последнем слое:
5. Рассчитать градиент на последнем слое:
6. Рассчитать градиент на скрытом слое:
7. Пересчитать веса всех ребер:

# Алгоритм работы программы

1. Считывание данных для обучения
2. Задание начальных значений – число нейронов скрытого слоя (), скорость обучения (), генерируются малые случайные значения для весов ребер ().
3. Обучение. Для каждого изображения из обучающей выборки повторить:
   1. Подать значения интенсивностей на вход и подсчитать значения на последнем слое.
   2. Применить функцию активации к выходным значениям на последнем слое:
   3. Рассчитать градиент на последнем слое:
   4. Рассчитать градиент на скрытом слое:
   5. Пересчитать веса ребер:
4. Тестирование. Для каждого изображения из тестовой выборки повторить:
   1. Подать значения интенсивностей на вход и подсчитать значения на последнем слое.
   2. Выбрать класс, имеющий максимальное значение на выходном слое.
   3. Сравнить выбранный класс с правильным ответом. Если значения не совпадают, увеличить счетчик ошибок.
5. Расчет процента ошибки.

# Данные для обучения

Для обучения персептрона используется набор из 60000 изображений рукописных цифр MNIST. Сеть обучается одним полным последовательным проходом по всем данным из обучающей выборки. При увеличении числа проходов по данным незначительно увеличивается точность.

# Результаты работы

## Расчет ошибки

Для проверки работоспособности нейронной сети используется набор из 10000 тестовых изображений MNIST. В качестве результата классификатора выбирается класс, имеющий наибольшее значение после применения функции softmax к выходу последнего слоя. Полученный результат сравнивается с истинным значением. При несоответствии результата увеличивается счетчик ошибок. Полученное после полного тестирования количество ошибок делится на общее количество тестовых изображений для получения доли ошибки. Формула расчета вероятности ошибки в процентах:

Формула для расчета точности классификатора:

## Результаты тестирования

Для тестирования работы персептрона необходимо установить параметры обучения: скорость обучения и число нейронов на скрытом слое . В данном случае скорость обучения = 0.1. Число нейронов скрытого слоя = 300. Данные параметры можно указать в качестве параметров запуска.

В результате работы нейронной сети из 10000 изображений 490 были классифицированы неправильно. Таким образом, шанс на ошибку составил 4.9%. Точность классификатора – 0.951.

Так же был проведен эксперимент, когда обучение проводилось тремя проходами по тестовым данным. В таком случае ошибка составила 4.04%, точность классификатора – 0.9596.

# Программная реализация

В данной работе реализован класс Perceptron. В классе имеются 5 методов:

* public Perceptron(string datapath, int \_s, float \_speed) - конструктор, считывающий входные данные для обучения сети, задающий начальные значения весов. datapath – путь к папке с данными, s – число нейронов скрытого слоя, speed – скорость обучения.
* public void Train() – функция, реализующая проход по обучающей выборке, вызывает функцию train(unsigned char\* input, char output) для каждого изображения из данной выборки.
* private double\* train(unsigned char\* input, char output) – функция, реализующая метод обратного распространения ошибки. Возвращает результат применения функции Softmax на последнем слое.
* public void Test() – функция, реализующая проход по тестовой выборке и подсчет ошибки, вызывает функцию test(unsigned char\* input, char output) для каждого изображения из данной выборки.
* private bool test(unsigned char\* input, char output) – функция, определяющая класс для заданного входного слоя и сравнивающая его с ожидаемым истинным значением. Возвращает true, если класс определен верно.

Для считывания обучающей и тестовой выборки используется класс MNIST,

которых хранит в себе пути к наборам обучающих и тестовых изображений, а так же к наборам классов (labels) к которым относится каждое изображение. Содержит методы:

* unsigned char\*\* read\_mnist\_train\_images(int& number\_of\_images, int& image\_size) – функция, возвращающая набор входных данных для обучения сети.
* unsigned char\*\* read\_mnist\_test\_images(int& number\_of\_images, int& image\_size) – функция, возвращающая набор входных данных для тестирования сети.
* unsigned char\* read\_mnist\_train\_labels(int& number\_of\_labels) – функция, возвращающая набор классов входных изображений.
* unsigned char\* read\_mnist\_test\_labels(int& number\_of\_labels) – функция, возвращающая набор классов для проверки точности классификатора.

Также написан класс Functions, реализующий используемые функции активации:

* void sigm(int s, double\* a) – функция активации для скрытого слоя (сигмоида).
* void softmax(int k, double\* a) – функция активации для выходного слоя.

Обучение и тестирование нейронной сети реализуется следующим образом:

Perceptron\* perceptron = new Perceptron(datapath, s, speed);

perceptron->Train();

perceptron->Test();

# Заключение

Написанный персептрон способен успешно справиться с задачей классификации рукописных цифр из набора данных MNIST. Параметры сети можно настроить для увеличения точности определения. Также для увеличения точности классификации можно производить обучение в несколько проходов по тестовой выборке.

# Литература

1. MNIST dataset [http://yann.lecun.com/exdb/mnist].
2. OpenCV [http://opencv.org].
3. Материалы Летней межвузовской школы 2016 [https://github.com/itseez-academy/itseez-ss-2016theory], [https://github.com/itseez-academy/itseez-ss-2016-practice].

# Приложения

## Код функции train

double\* Perceptron::train(unsigned char\* input, char output)

{

double\* hidden = new double[s];

int\* out = new int[k];

double\* result = new double[k];

double\* deltaout = new double[k];

double\* deltahid = new double[s];

Functions\* func = new Functions();

for (int i = 0; i < k; i++)

{

if (i == output) out[i] = 1; else out[i] = 0;

}

for (int j = 0; j < s; j++)

{

hidden[j] = 0;

for (int i = 0; i < n; i++)

{

hidden[j] += input[i] \* w0[i\*s + j] / 255;

}

}

func->sigm(s, hidden);

for (int j = 0; j < k; j++)

{

result[j] = 0;

for (int i = 0; i < s; i++)

{

result[j] += hidden[i] \* w1[i\*k + j];

}

}

func->softmax(k, result);

for (int i = 0; i < k; i++)

{

deltaout[i] = result[i] - out[i];

}

for (int i = 0; i < s; i++)

{

deltahid[i] = 0;

for (int j = 0; j < k; j++)

{

deltahid[i] += deltaout[j] \* w1[i\*k + j] \* (hidden[i] \* (1 - hidden[i]));

}

}

bool tmp = false;

for (int i = 0; i < n; i++)

{

for (int j = 0; j < s; j++)

{

deltaw0[i\*s + j] = speed \* deltahid[j] \* input[i] / 255;

w0[i\*s + j] -= deltaw0[i\*s + j];

}

}

for (int i = 0; i < s; i++)

{

for (int j = 0; j < k; j++)

{

deltaw1[i\*k + j] = speed \* deltaout[j] \* hidden[i];

w1[i\*k + j] -= deltaw1[i\*k + j];

}

}

delete hidden;

delete result;

delete out;

delete deltaout;

delete deltahid;

return result;

}

## Код функции test

bool Perceptron::test(unsigned char\* input, char output)

{

double\* hidden = new double[s];

double\* result = new double[k];

Functions\* func = new Functions();

for (int j = 0; j < s; j++)

{

hidden[j] = 0;

for (int i = 0; i < n; i++)

{

hidden[j] += input[i] \* w0[i\*s + j] / 255;

}

}

func->sigm(s, hidden);

for (int j = 0; j < k; j++)

{

result[j] = 0;

for (int i = 0; i < s; i++)

{

result[j] += hidden[i] \* w1[i\*k + j];

}

}

func->softmax(k, result);

int maxI = 0;

double maxR = result[0];

for (int i = 1; i < k; i++)

{

if (result[i] > maxR)

{

maxI = i;

maxR = result[i];

}

}

delete hidden;

delete result;

if (maxI != output) return false;

return true;

}