|  |
| --- |
| Министерство образования и науки Российской Федерации  Федеральное государственное бюджетное образовательное учреждение высшего образования  Российский химико-технологический университет  им. Д.И.Менделеева  Факультет информационных технологий и управления  Кафедра информационных компьютерных технологий |
| Отчет по лабораторному практикуму  по предмету  «Интеллектуальные системы и технологии»  на тему: |
| «Реализация искусственной нейронной сети Кохонена фиксированной структуры» |

|  |
| --- |
| Студентки КС-40  Тереховой Оксаны  2019 |

Оглавление

[Задание 2](#_Toc27313194)

[Цель: 2](#_Toc27313195)

[Задачи: 2](#_Toc27313196)

[Теоретическая часть 3](#_Toc27313197)

[Общее описание нейронных сетей 3](#_Toc27313198)

[Архитектура нейронной сети Кохонена 6](#_Toc27313199)

[Обзор и сравнение возможных средств разработки 8](#_Toc27313200)

[Парадигмы 8](#_Toc27313201)

[Типизация 8](#_Toc27313202)

[Компилятор/интерпретатор 9](#_Toc27313203)

[Управление памятью 9](#_Toc27313204)

[Управление потоком вычислений 9](#_Toc27313205)

[Структуры данных 9](#_Toc27313206)

[Описание выбранного языка - Python 10](#_Toc27313207)

[Практическая часть 11](#_Toc27313208)

[Реализация связи нейрона 11](#_Toc27313209)

[Реализация Нейрона: 11](#_Toc27313210)

[Пример алгоритма работы 12](#_Toc27313211)

[Входные данные 12](#_Toc27313212)

[Результат работы 13](#_Toc27313213)

[Выводы 14](#_Toc27313214)

[Приложение 15](#_Toc27313215)

[main.py 15](#_Toc27313216)

[classes.py 16](#_Toc27313217)

[Список литературы 17](#_Toc27313218)

# Задание

Реализация искусственной нейронной сети Кохонена фиксированной структуры

## Цель:

Изучить структуру самообучающейся нейронной сети Кохонена фиксированной структуры, построить и реализовать алгоритм кластеризации данных.

## Задачи:

* Изучение теории структуры нейронной сети
* Выбор средства разработки
* Определение исходной выборки данных
* Определение количественного и качественного составов входов и выходов
* Обработка и нормализация исходной выборки данных
* Определение структуры нейронной сети
* Настройка параметров нейронной сети
* Обучение нейронной сети
* Тестирование нейронной сети
* Анализ результатов

# Теоретическая часть

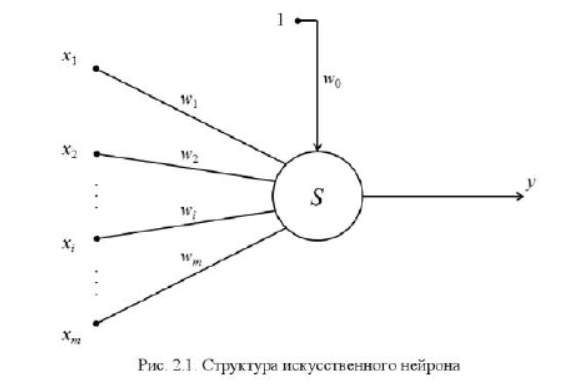
## Общее описание нейронных сетей

Искусственные нейронные сети — это математический аппарат, основанный на аналогиях с функционированием центральной нервной системы и предназначенный для решения широкого круга задач, таких, как распознавание образов, классификация, кластеризация и аппроксимация данных, прогнозирование временных рядов, адаптивное управление и многих других. Круг решаемых задач может быть различен для разных архитектур.

Архитектура нейронной сети — совокупность, объединяющая принципы организации и функционирования нейронной сети и отдельных искусственных нейронов, алгоритмы и расчетные соотношения для ее обучения.

Искусственный нейрон — это элементарная структурная единица искусственной нейронной сети, выполняющая функции по обработке входных сигналов х, поступающих с других нейронов, и представлению результата в форме выходного значения. В общем случае, входные сигналы подвергаются в теле нейрона преобразованию, зачастую нелинейному, с участием весовых (синаптических) коэффициентов w, характеризующих значимость каждой связи, и функции активации. При необходимости для более качественного обучения в некоторых нейросетевых архитектурах вектор синаптических коэффициентов дополняется коэффициентом смещения w0 — весом фиктивного единичного входа.

Структура искусственного нейрона схематично показана ниже:



Множество определенным образом связанных между собой искусственных нейронов формируют искусственную нейронную сеть. Нейроны могут быть сгруппированы в слои. Количество слоев, нейронов в каждом слое, а также количество внешних входных и выходных переменных определяют структуру нейронной сети.

Для практического использования нейронной сети ее необходимо предварительно обучить. Обучение нейронной сети представляет собой процедуру точного или итерационного расчета всех ее весовых коэффициентов, соединяющих внешние входы с нейронами и нейроны между собой.

Под эпохой обучения понимается цикл однократного предъявления (как правило, в случайном порядке) всех примеров обучающей выборки для коррекции весовых коэффициентов при одинаковых настройках самой сети и алгоритма ее обучения.

Поскольку происхождение входных сигналов искусственных нейронов может быть различно: это могут быть как выходы других нейронов, так и входные сигналы из внешней среды (входы нейронной сети), эти сигналы целесообразно использовать в нормализованной форме, то есть изменяющимися в пределах [0, 1] или [-1, 1]. Выбор пределов нормализации зависит, в первую очередь, от планирующейся для использования архитектуры нейронной сети, а во вторую, — от предпочтений самого исследователя, так как отдельные архитектуры сетей позволяют работать только со строго определенными значениями. В зависимости от природы используемых переменных и степени влияния их различных значений могут быть использованы методы линейной и нелинейной нормализации.

Состояние нейрона S определяется как взвешенная с помощью синаптических коэффициентов сумма его входных значений



где m — количество входных сигналов нейрона.

Функция активации преобразует состояние нейрона в значение в форме и масштабе выходного сигнала.



Группа искусственных нейронов, соединенных связями одного уровня с нейронами других групп, входами нейронной сети или друг с другом, образует отдельный слой нейронов.

Слой искусственных нейронов, выходы которого являются выходами всей нейронной сети, называется выходным слоем. Таким образом, количество нейронов выходного слоя должно совпадать с количеством выходов всей нейронной сети. Все остальные слои искусственных нейронов, кроме выходного, называются скрытыми слоями.

Нейронные сети можно классифицировать по принципу обучения. Наиболее распространены следующие разновидности обучения:

* с учителем (когда требуется получить нейросетевую модель, обобщающую примеры, в каждом из которых определенному набору значений входных переменных должен соответствовать набор значений выходных переменных);
* без учителя (когда ведется поиск скрытых закономерностей в массиве обучающих примеров без использования значений выходных переменных).

Еще один классифицирующий признак — тип алгоритма обучения.

* итерационные алгоритмы (когда многократно, в цикле повторяется определенная последовательность действий, предусмотренных алгоритмом, до стабилизации весовых коэффициентов);
  + на детерминированные (когда задана строго определенная последовательность действий по минимизации ошибки обучения);
  + стохастические (в ходе работы которых поиск лучших значений весовых коэффициентов подчиняется случайным процессам).
* точные алгоритмы (когда весовые коэффициенты рассчитываются по обучающему набору с использованием операций линейной алгебры).

По области допустимых значений выходного сигнала нейронные сети классифицируются:

* на бинарные (когда выходная переменная может принимать одно из двух возможных значений);
* аналоговые (когда возможно бесконечное множество значений выходной переменной, принадлежащее пределам ее нормализации).

Область допустимых значений выходного сигнала определяется выбранной активационной функцией соответствующего выходного нейрона. В общем случае возможно одновременное использование бинарных значений для одних и аналоговых значений для других выходов одной и той же нейронной сети.

По структуре связей среди нейронных сетей можно выделить:

* сети прямого распространения (когда сигналы распространяются непосредственно от входов к выходам);
* сети с обратными связями (когда выходной сигнал нейрона может возвращаться на его же вход, входы других нейронов того же слоя или входы нейронов предшествующих слоев).

Если выходные сигналы нейрона распространяются по обратным связям ко всем остальным нейронам без исключения, такая сеть называется полносвязной.

Нейронные сети могут быть разделены на классы по способу изменения состояний нейронов:

* на асинхронные (у которых в определенный момент времени только один нейрон меняет свое состояние);
* синхронные (у которых в определенный момент времени сразу несколько нейронов (как правило, целый слой) меняют свои состояния).

## Архитектура нейронной сети Кохонена

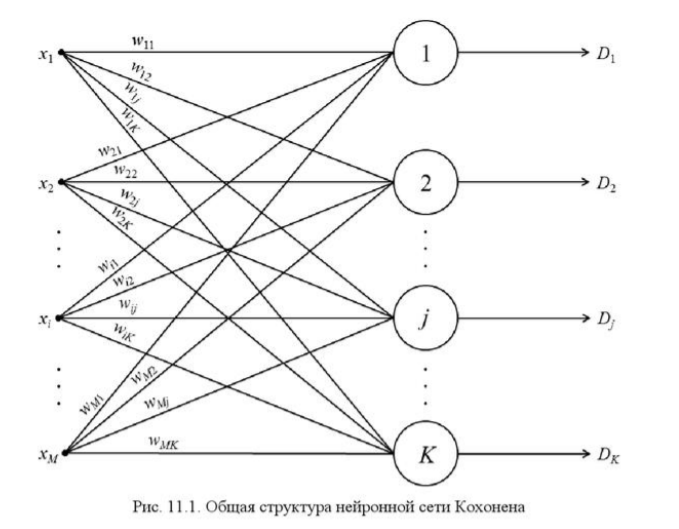
Нейронная сеть Кохонена - нейросетевая архитектура, обучающейся без учителя.

Перечень решаемых задач: кластеризация данных или прогнозирование свойств, может использоваться с целью уменьшения размерности данных с минимальной потерей информации.

В рассматриваемой архитектуре сигнал распространяется от входов к выходам в прямом направлении. Структура нейронной сети содержит единственный слой нейронов (слой Кохонена) без коэффициентов смещения (рис. 11.1). Общее количество весовых коэффициентов рассчитывается как произведение

Nw=МК

Количество нейронов равно количеству кластеров, среди которых происходит распределение и последующее перераспределение обучающих примеров. Количество входных переменных нейронной сети равно числу признаков, характеризующих объект исследования и на основе которых происходит отнесение его к одному из кластеров.



Нормализация входных переменных выполняется в пределах [-1, 1] или [0, 1]

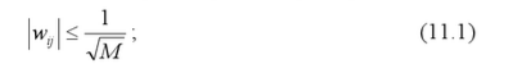
Для жизненного цикла нейронных сетей данной архитектуры характерны три основных стадии жизненного цикла: обучение, кластерный анализ и практическое использование.

Алгоритм обучения:

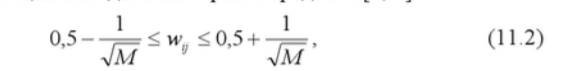
1. Задание структуры сети (количества нейронов слоя Кохонена) (К).

2. Случайная инициализация весовых коэффициентов значениями, удовлетворяющими одному из следующих ограничений:

— при нормализации исходной выборки в пределах [-1, 1]:

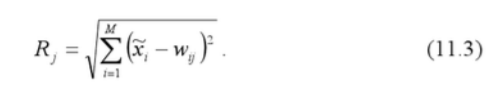


— при нормализации и выборки в пределах [0, 1]:

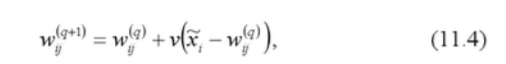


где М — количество входных переменных сети - характеристических признаков объекта исследования.

3. Подача на входы сети случайного обучающего примера текущей эпохи обучения и расчет евклидовых расстояний от входного вектора до центров всех кластеров



4. По наименьшему из значений К, выбирается нейрон-победитель j, в наибольшей степени близкий по значениям с входным вектором. Для выбранного нейрона (и только для него) выполняется коррекция весовых коэффициентов:



где у- коэффициент скорости обучения.

5. Цикл повторяется с шага 3 до выполнения одного или нескольких условий окончания:

* исчерпано заданное предельное количество эпох обучения;
* не произошло значимого изменения весовых коэффициентов в пределах заданной точности на протяжении последней эпохи обучения;
* исчерпано заданное предельное физическое время обучения.

Коэффициент скорости обучения может задаваться постоянным из пределов (0, 1] или переменным значением, постепенно уменьшающимся от эпохи к эпохе.

## Обзор и сравнение возможных средств разработки

### Парадигмы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Возможность | Языки | | | | |
| С | С++ | Java | Python | Delphi |
| ООП | Средства библиотек | + | + | + | + |
| Рефлексия | - | Ограничено | Ограничено | + | Ограничено |
| Обобщенное программирование | + (void) | Ограничено (шаблоны) | + | + | + |

**ООП**: Основана на представлении всего в виде объектов, являющихся экземплярами того или иного класса и воплощает применение концепции абстрагирования. Объект при этом соединяет внутри себя как данные, так и методы, их обрабатывающие. Как правило, поддерживаются характерные возможности: наследование, инкапсуляция и полиморфизм.

В информатике отражение или **рефлексия** означает процесс, во время которого программа может отслеживать и модифицировать собственную структуру и поведение во время выполнения.

**Обобщённое программирование** позволяет записывать алгоритмы, принимающие данные любого типа.

### Типизация

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Возможность | Языки | | | | |
| С | С++ | Java | Python | Delphi |
| Статическая типизация | + | + | + | - | + |
| Динамическая типизация | - | - | - | + | Тип Variant |
| Явная типизация | + | + | + | Частично | + |
| Неявная типизация | - | Статическая | - | + | - |
| Неявное приведение типов с потерей данных | + | + | - | - | + |

**Статическая типизация** - переменные и параметры методов/функций связываются с типами в момент объявления и не могут быть изменены позже.

**Динамическая типизация** - переменные и параметры методов/функций связываются с типами в момент присваивания значения (или передачи параметра в метод/функцию), а не в момент объявления переменной или параметра. Одна и та же переменная в разные моменты может хранить значения разных типов.

**Явная типизация** - типы переменных и параметров указываются явно.

**Неявная типизация** - типы переменных и параметров не указываются явно. Неявная типизация может быть и статической, в таком случае типы переменных и параметров вычисляются компилятором.

**Неявное приведение типов** в таких ситуациях, где может произойти потеря данных — например, использование числа с плавающей точкой там, где предполагалось использование целого числа.

### Компилятор/интерпретатор

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Возможность | Языки | | | | |
| С | С++ | Java | Python | Delphi |
| Оpen-source | + | + | + | + | + |
| Интерпретатор командной строки | CINT | CINT | - | + | PascalScript |
| Условная компиляция | + | + | - | Не применимо к языку | + |

Наличие полноценного **open-source** компилятора/интерпретатора.

**Интерпретатор командной строки** - возможность вводить инструкции языка строка за строкой с их немедленным выполнением.

**Условная компиляция** - возможность включать/выключать части кода в зависимости от значения символов условной компиляции.

### Управление памятью

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Возможность | Языки | | | | |
| С | С++ | Java | Python | Delphi |
| Ручное управление памятью | + | + | - | - | + |
| Сборка мусора | - | Для интеллектуальных указателей | + | + | - |

**Ручное управление памятью** - возможность явного выделения и освобождения памяти в куче (например, с помощью операторов new и delete в C++).

**Сборка мусора** - возможность использовать автоматический процесс сборки мусора (освобождения памяти в куче, занятой неиспользуемыми объектами).

### Управление потоком вычислений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Возможность | Языки | | | | |
| С | С++ | Java | Python | Delphi |
| GoTo | + | + | - | - | + |
| Break | + | + | + | + | - |
| Try/catch | - | + | + | + | + |
| Try/else | - | - | + | + | + |

### Структуры данных

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Возможность | Языки | | | | |
| С | С++ | Java | Python | Delphi |
| Кортежи | - | +- | - | + | - |
| Многомерные массивы | + | + | Массив массивов | Массив массивов | + |
| Динамические массивы | - | + | Vector | List | + |
| Целые числа произвольной длины | - | - | + | + | - |

## Описание выбранного языка - Python

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Плюсы:

* интерпретируемый язык – выполнение кода происходит даже при наличии ошибок в коде
* модификация кода во время выполнения
* интерпретатор присутствует на многих операционных системах (включая windows и ubuntu)
* наличие большого кол-ва модулей и библиотек
* высокая скорость разработки и отладки
* не требуется время на компиляцию
* динамическая типизация - не требуется описание переменных
* простой синтаксис
* динамические массивы заменены списками
* оптимальная работа с классами и указателями
* уменьшение кол-ва кода в несколько раз в силу минималистичного синтаксиса

Минусы:

* скорость выполнения программы, но для исполнения поставленной задачи – не актуальная проблема, тк падение скорости незначительное.

# Практическая часть

## Реализация связи нейрона

|  |
| --- |
| **class** Link:  Input = 0  weight = 1 *# весовой коэффициент связи* **def** \_\_init\_\_(self, weight): *# создание связи с заданным весовым коэффициентом* self.weight = weight   **def** set\_input(self, inp): *# переопределение входного значения связи* self.Input = inp |

## Реализация Нейрона:

|  |
| --- |
| **class** Neuron:  input\_count = 1 *# кол-во связей нейрона* Links = [Link] *# массив связей* Output = 1 *# выходное значение нейрона* **def** \_\_init\_\_(self, M): *# создание нейрона с заданым кол-вом связей* self.input\_count = M *# задание кол-ва связей нейрона* self.Links = [] *# обнуление массива связей* a = 0.5 - 1 / math.sqrt(M) *# нижний предел весового коэффициента (для нормализации в пределах [0;1]* b = 0.5 + 1 / math.sqrt(M) *# верхний предел весового коэффициента (для нормализации в пределах [0;1]* **for** i **in** range(M): *# для каждой связи от 1 до М* w = round(random.uniform(a, b), 1) *# округление рандомного весового коэффициента до 1 знака после запятой* self.Links.append(Link(w)) *# добавление связи с весом w в массив связей нейрона* **def** print\_info(self): *# вывод информации о нейроне* print(**'Neuron.'**)  print(**'Inputs: '**, self.input\_count)  **for** i **in** range(self.input\_count):  print(i+1, **') '**, self.Links[i].Input, **' '**, self.Links[i].weight)  print(**'Outputs: '**, self.Output)   **def** getR(self, InputList): *# получение информации о расстояние нейрона до входных параметров* summ = 0  **for** i **in** range(self.input\_count):  summ = summ + (InputList[i] - self.Links[i].weight)\*\*2  self.Output = math.sqrt(summ)  **return** self.Output   **def** correction(self, u, InputList): *# корректировка весовых коэффициентов* err = 0  **for** i **in** range(self.input\_count):  w = self.Links[i].weight  self.Links[i].weight = self.Links[i].weight + u\*(InputList[i]-self.Links[i].weight)  err = err + (w - self.Links[i].weight)  **return** math.fabs(err) |

## Пример алгоритма работы

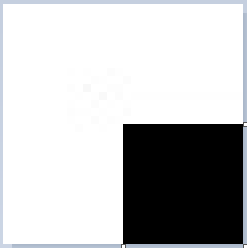
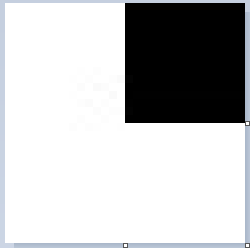
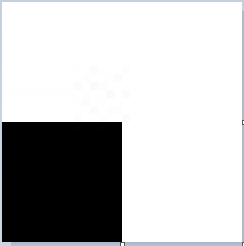
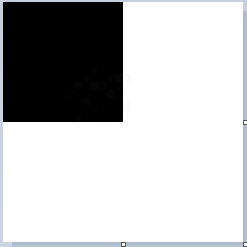
main.py

|  |
| --- |
| **for** i **in** range(50,1,-2):  u=i\*0.01  print(**'Epoha - '**, u)  eps = 0  list\_sort = [] *# список выигравших нейронов для каждой картинки* **for** data **in** data\_image\_list:  Distance = []  **for** j **in** range(K):  Distance.append(Neurons[j].getR(data)) *#заполнение массива расстояний для каждого нейрона* list\_sort.append(Distance.index(min(Distance))) *# определение выигр. нейрона* eps = eps + Neurons[Distance.index(min(Distance))].correction(u, data) *# накопление суммарной ошибки эпохи*  **if** eps < 0.001:  **break** |

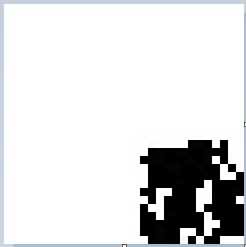
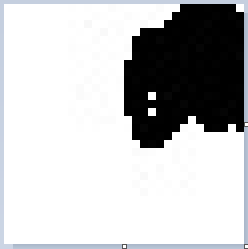
## Входные данные

Подается набор картинок 30х30 пикселей. На каждой закрашена одна из областей: один из четырех углов.

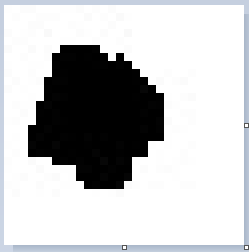
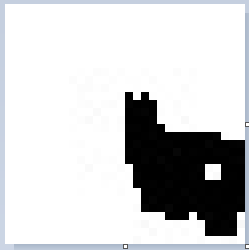
Для обучения сначала подаются картинки с четким делением размеченных углов:

Далее на вход подаются картинки с различным шумом, но четким разделением на области, например:

  и т.п.

Для тестирования работы подаются случайно закрашенные области, например:

Всего на вход поступает 15 изображений

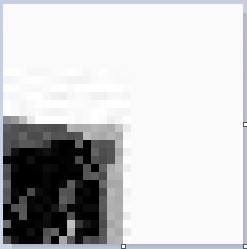
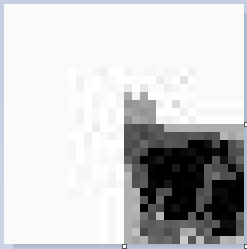
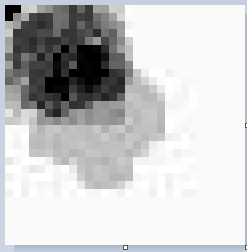
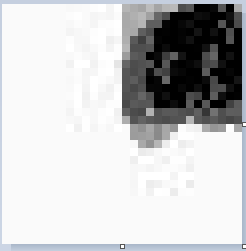
## Результат работы

По ходу программы в рабочей папке создается папка sort, в которой по кол-ву кластеров находятся папки от 0 до К-1. В данном случае от 0 до 3 (тк углов 4). Внутри каждой папки находятся изображения отнесенные к данному кластеру.



Кроме этого, в каждой папке создается изображение с тем же названием, что и папка, которое показывает финальное значение весовых коэффициентов в серых цветах.

Например, для данного примера:

# Выводы

В результате выполнения работы была разработана программа, содержащая модель искусственной нейронной сети Кохонена фиксированной структуры, предназначенной для кластеризации данных.

При написании использовался язык программирования Python и объектный подход, что позволило снизить время разработки системы и значительно уменьшить размер исходного кода программы.

Программа поддерживает универсальную возможность элементарного расширения по кол-ву кластеров (константа К) и размеру получаемого изображения для кластеризации (константы а и b), что позволяет использовать ее для разных задач.

Тестирование проводилось на изображениях 30х30 и 4 кластерах - по расположению закрашенного угла. По полученным результатам работы нейронной сети можно сделать выводы, что самообучение прошло корректно. Задача кластеризации объектов выполнена, результат удовлетворительный.

# Приложение

## main.py

|  |
| --- |
| **import** classes **from** PIL **import** Image **import** glob **import** os **import** shutil   K = 4 *# кол-во нейронов = кол-ву кластеров* a = 30 b = 30 M = a\*b *# кол-во связей в 1 нейроне = кол-ву входных значений (пикселей)* Neurons = [] **for** j **in** range(K):  Neurons.append(classes.Neuron(M)) *# массив из нейронов (заполнение)* image\_list = [] image\_name = [] **for** filename **in** glob.glob(**'image/\*.jpg'**):  image\_name.append(filename)  image = Image.open(filename) *# считать изображение* image\_list.append(image) *# сохранить массив изображений* print(len(image\_name)) print(image\_name)  data\_image\_list = [] **for** im **in** image\_list:  blackImage = im.convert(**'L'**) *# конвертировать изображение в серые тона* bw = blackImage.point(**lambda** x: 0 **if** x < 128 **else** 1, **'1'**) *# преобразование в черно-белое изображение* data = list(bw.getdata()) *# преобразование изображения в список значений: 1-белый, 0-черный* data\_image\_list.append(data)  **for** i **in** range(50,1,-2):  u=i\*0.01  print(**'Epoha - '**, u)  eps = 0  list\_sort = [] *# список выигравших нейронов для каждой картинки* **for** data **in** data\_image\_list:  Distance = []  **for** j **in** range(K):  Distance.append(Neurons[j].getR(data)) *# заполнение массива расстояний для каждого нейрона* list\_sort.append(Distance.index(min(Distance))) *# определение выигр. нейрона* eps = eps + Neurons[Distance.index(min(Distance))].correction(u, data) *# накопление суммарной ошибки эпохи  # print(list\_sort)* print(eps)  **if** eps < 0.001:  **break  if** os.path.exists(**'sort'**): *# проверка на существование папки* shutil.rmtree(**'sort'**) *# удаление папки* os.mkdir(**'sort'**) *# создание папки* **for** i **in** range(K):  os.mkdir(**'sort/'** + str(i)) *# создание папок для каждого нейрона* **for** i **in** range(len(list\_sort)):  shutil.copy(image\_name[i],**'sort/'**+str(list\_sort[i])) *# сортировка изображений копированием* **for** n **in** range(K):  im = Image.new(**'L'**, (a, b)) *# создание изображения для каждого нейрона* **for** i **in** range(a):  **for** j **in** range(b):  pix = round(250 \* Neurons[n].Links[i\*b+j].weight) *# преобразование data в число от 0 до 255 (250, чтобы видеть границы)* im.putpixel((j, i), (pix))  im.save(**'sort/'** + str(n) + **'/'** + str(n)+**'.jpg'**, **"JPEG"**) *# сохранение изображения в папке с нейроном* |

## classes.py

|  |
| --- |
| **import** random **import** math  **class** Link:  Input = 0  weight = 1 *# весовой коэффициент связи* **def** \_\_init\_\_(self, weight): *# создание связи с заданым весовым коэффициентом* self.weight = weight   **def** set\_input(self, inp): *# переопределение входного значения связи* self.Input = inp   **class** Neuron:  input\_count = 1 *# кол-во связей нейрона* Links = [Link] *# массив связей* Output = 1 *# выходное значение нейрона* **def** \_\_init\_\_(self, M): *# создание нейрона с заданым кол-вом связей* self.input\_count = M *# задание кол-ва связей нейрона* self.Links = [] *# обнуление массива связей* a = 0.5 - 1 / math.sqrt(M) *# нижний предел весового коэффициента (для нормализации в пределах [0;1]* b = 0.5 + 1 / math.sqrt(M) *# верхний предел весового коэффициента (для нормализации в пределах [0;1]* **for** i **in** range(M): *# для каждой связи от 1 до М* w = round(random.uniform(a, b), 1) *# округление рандомного весового коэффициента до 1 знака после запятой* self.Links.append(Link(w)) *# добавление связи с весом w в массив связей нейрона* **def** print\_info(self): *# вывод информации о нейроне* print(**'Neuron.'**)  print(**'Inputs: '**, self.input\_count)  **for** i **in** range(self.input\_count):  print(i+1, **') '**, self.Links[i].Input, **' '**, self.Links[i].weight)  print(**'Outputs: '**, self.Output)   **def** getR(self, InputList): *# получение информации о расстояние нейрона до входных параметров* summ = 0  **for** i **in** range(self.input\_count):  summ = summ + (InputList[i] - self.Links[i].weight)\*\*2  self.Output = math.sqrt(summ)  **return** self.Output   **def** correction(self, u, InputList): *# корректировка весовых коэффициентов* err = 0  **for** i **in** range(self.input\_count):  w = self.Links[i].weight  self.Links[i].weight = self.Links[i].weight + u\*(InputList[i]-self.Links[i].weight)  err = err + (w - self.Links[i].weight)  **return** math.fabs(err) |

# Список литературы

1. Дударов С. П., Папаев П. Л. Теоретические основы и практическое применение искусственных нейронных сетей: учеб. Пособие

М.: РХТУ им. Д. И. Менделеева, 2014. – 104 с.

1. Сайт Wikipedia [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B0%D0%B2%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5\_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2\_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F#cite\_note-11 (дата обращения: 15.12.2019).
2. Сайт Wikipedia [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Python (дата обращения: 15.12.2019).
3. Сайт Программирование на Python, раздел Перевод документации Python 3.x [Электронный ресурс]. – Режим доступа: https://pythoner.name/documentation (дата обращения: 30.11.2019).
4. Сайт библиотеки Pillow (PIL Fork), раздел Документации, модуль Image [Электронный ресурс]. – Режим доступа: https://pillow.readthedocs.io/en/stable/reference/Image.html (дата обращения: 16.11.2019).
5. Сайт Python 3 для начинающих, Модуль glob [Электронный ресурс]. – Режим доступа: https://pythonworld.ru/moduli/modul-glob.html (дата обращения: 15.11.2019).
6. Сайт Python 3 для начинающих, Модуль os [Электронный ресурс]. – Режим доступа: https://pythonworld.ru/moduli/modul-os.html (дата обращения: 17.11.2019).
7. Сайт Python 3 для начинающих, Модуль shutil [Электронный ресурс]. – Режим доступа: https://pythonworld.ru/moduli/modul-shutil.html (дата обращения: 16.11.2019).