МИНИСТЕРСТВО НАУКИ И ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра прикладной математики**

**КУРСОВАЯ РАБОТА**

**КЛАССИФИКАЦИЯ НЕЙРОННЫМИ СЕТЯМИ НА PYTHON**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.А. Бордонос

(подпись)

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность (профиль) Системный анализ, исследование операций и управление (Математическое и информационное обеспечение экономической деятельности)

Научный руководитель

д-р. техн. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Халафян

(подпись)

Нормоконтролер

канд. физ.-мат. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Г.В. Калайдина

(подпись)

Краснодар

2018

**РЕФЕРАТ**

Работа содержит 26 страницы, 6 рисунков, 7 источников.

НЕЙРОННАЯ СЕТЬ, ГЛУБОКОЕ ОБУЧЕНИЕ, КЛАССИФИКАЦИЯ, ПЕРЦЕПТРОН, НЕЙРОН, SOFTMAX, ЭВОЛЮЦИОННЫЙ АЛГОРИТМ, ОБРАТНОЕ РАСПРОСТРАНЕНИЕ ОШИБКИ, СИНАПТИЧЕСКИЕ ВЕСА

Объектом исследования были нейронные сети классификации, в частности, реализованные на Python. Итог работы – создание сети бинарной классификации на Python.

В результате была получена программа основными функциями которой являются:

– преобразование текстовых данных в форму, пригодную для обработки нейронными сетями;

– построение модели нейронной сети;

– обучение построенной модели;

– бинарная классификация последующих данных.

СОДЕРЖАНИЕ

Введение3

1 Теоретические основы нейронных сетей4

1.1 Условная вероятность и формула Байеса4

1.2 Структура нейронной сети5

1.3 Функции активации7

1.4 SoftMax9

2 Обучение сети10

2.1 Алгоритмы оптимизации10

2.2 Обратное распространение ошибки11

3 Настройка модели16

3.1 Количество нейронов в скрытом слое16

4 Программная реализация классификационной сети на Python17

4.1 Постановка задачи и решение 17

4.2 Результат Rex.py и описание принципа его работы 18

Заключение21

Список использованных источников22

**ВВЕДЕНИЕ**

Обработка информации методами нейронных сетей, также известная как глубокое, или глубинное обучение (англ. Deep Learning) является, одним из самых мощных на данный момент инструментом исследования данных.

Этот био-инспирированный алгоритм, в настоящий момент переживает ренессанс, имеющий в основе две основные причины. Первая, и главная – аппаратное развитие. Если раньше для решения задач, связанных с глубоким обучением (к примеру – распознавание изображений) требовались дни, и даже недели, то сегодня на это требуются считанные минуты. Вторая, во многом является следствием первой – обилие информации для обработки. Интернет делает доступным огромное множество данных для обработки.

Развитие этих методов, также не стоит на месте. Не прекращается разработка оптимизационных алгоритмов, эксперименты с представлением данных, и строением сетей. Все это имеет в конечном итоге одну цель – ускорение обучения.

Внимание к глубокому обучению вполне обосновано. Эти алгоритмы сегодня применяются повсеместно. Для примера – в финансовом анализе, сервисах определения музыки, и даже для управления беспилотными автомобилями. Представленные и другие применения нейронных сетей на практике делают их в высокой степени актуальными.

В данной работе рассмотрены перцептронные нейронные сети классификации.

Цель исследования – изучение технологии разработки нейронных сетей и создание базовой классификационной сети на языке Python и описание ее работы.

**1 Теоретические основы нейронных сетей**

Все существующие на данный момент нейронные сети можно разделить на две категории: регрессионные и классификационные. Регрессионные сети на выходе дают непрерывную случайную величину, классификационные – дискретную случайную величину. Далее будут рассматриваться классификационные сети.

**1.1 Условная вероятность и формула Байеса**

Одним из важнейших принципов работы классификационной нейронной сети является подсчет условной вероятности. Подается исход и нужно вычислить, с какой вероятность некоторое событие послужило ему причиной. Такое соотношение задается формулой Байеса [1].

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

где –

– апостериорная вероятность гипотезы Ak, при выполнении события B.

– априорная вероятность k-ой гипотезы Ak.

– полная вероятность события B.

– вероятность наступления события B, при выполнении гипотезы Ak [2].

В рамках рассматриваемой теории гипотеза – событие B известно – это входные данные и нужно определить максимально вероятную гипотезу Ak для этих данных.

**1.2 Структура нейронной сети**

Элементарной структурой нейронной сети является перцептрон, или как его по-другому называют нейрон.

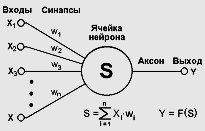


Рисунок 1 – Схема перцептрона

Перцепторн – это такая функция, на вход которой подается вектор (x на рисунке). Компоненты вектора, умноженные на соответствующие значения весов (синапсов) (w на рисунке) суммируются. К полученной сумме применяется так называемая функция активации (аксон). Итоговое значение является выходом.

Существует два варианта написания названия – перцептрон и персептрон, однако оба имеют место быть и использование любого из них – личная предпочтение.

Естественный нейрон действует следующим образом. Он осуществляет преобразование входных сигналов, приходящих в нейрон, которые часто именуются синапсами. Эти сигналы поступают посредством дендритов. Затем в зависимости от весов (отрицательные веса также называют тормозящими, в то время как положительные − активирующим), изменяет сигнал и подает его на один единственный выход, именуемый аксоном, который может быть связан с любым количеством других нейронов. Перцептрон действует по такому-же сценарию.

Их различие состоит в некоторых деталях. Во-первых, простой размер конструкций из них – в человеческом мозге порядка , что не под силу обрабатывать даже самым мощным компьютерам. Во-вторых, нейроны в мозге активны не постоянно, а посылают импульсы, зависящие от возбуждения каждого конкретного нейрона

Нейронная сеть состоит из множества соединённых между собой элементов – перцептронов.

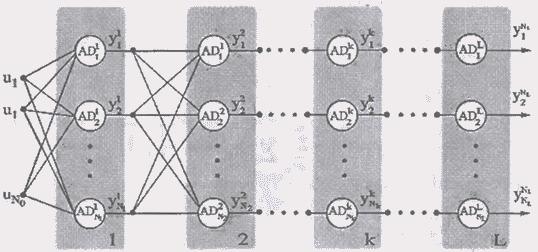


Рисунок 2 – Структура многослойной нейронной сети

Эти нейроны объединяются в слои. Каждый нейрон любого слоя в общем случае является одним из входов нейронов последующего.

Входным слоем в сети является вектор входных данных, координатами которого являются признаки, он и является событием B в указанной ранее формуле Байеса.

Выходной слой определяет вектор вероятностей принадлежности входного элемента к одному из классов.

Подобная модель на основе перцептронов носит название MLP (Multi-Layered Perceptron).

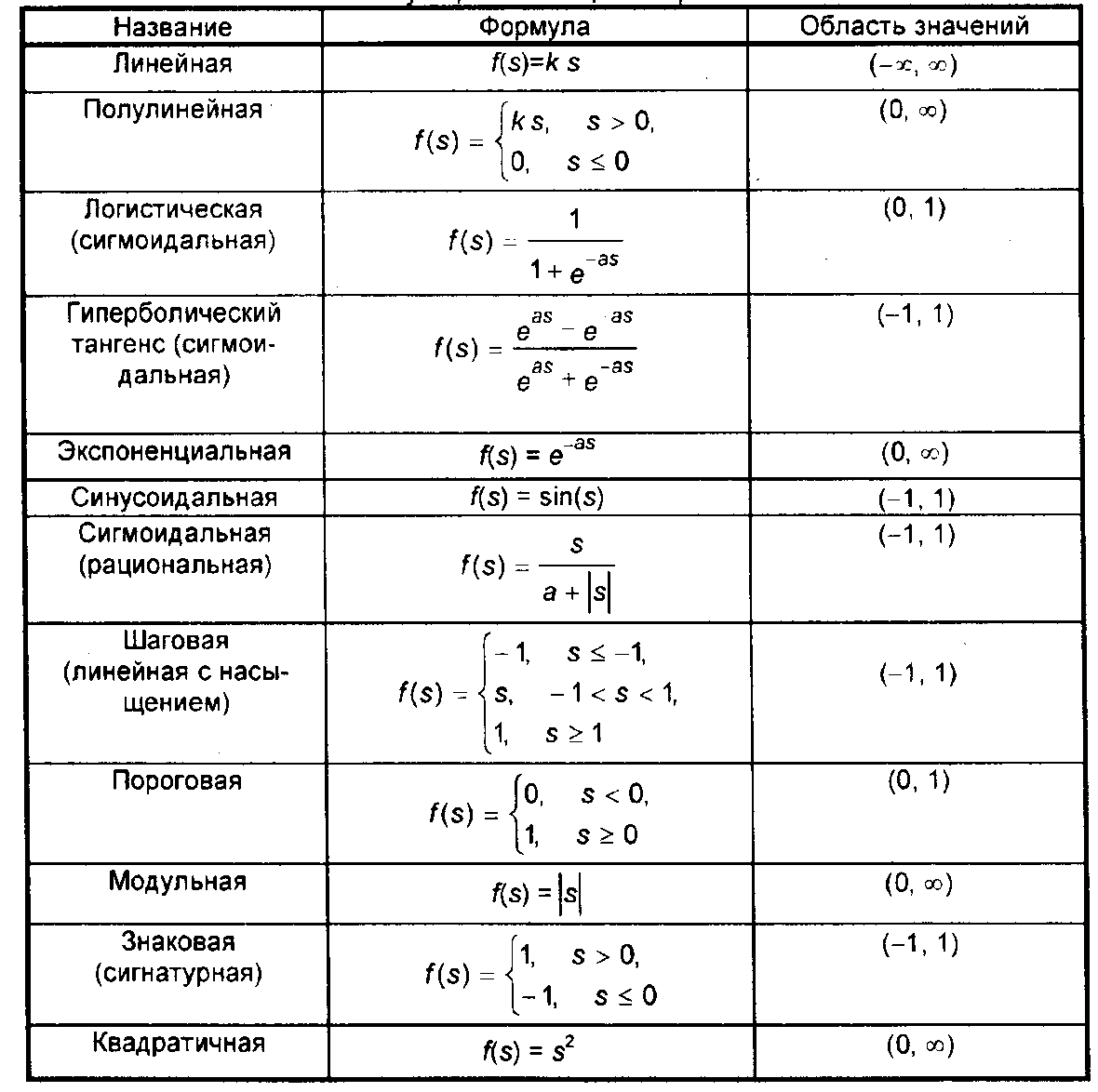
**1.3 Функции активации**

Сумма координат линейной комбинации любого вектора – есть функция линейная. Это порождает необходимость добавления нелинейности в сеть, поскольку иначе приближение итоговой функции может стать задачей невыполнимой за сколь-нибудь приемлемое время.

Основной задачей функций активаций является добавление этой нелинейности. В таблице 1 представлены возможные варианты таких функций. Следует понимать, что указанные ниже функции не все возможные варианты для активации.

Выбор правильной функции позволяет существенно увеличить скорость сходимости во время обучения.

Таблица 1 – некоторые функции активации



В случае если нелинейность не требуется, ничто не запрещает брать линейную функцию для активации.

Для нелинейных задач очень часто всего используют гиперболический тангенс (tanh), функцию ReLU (Rectified Linear Unit) (в переводе: выпрямленный линейный элемент) и им подобные [1].

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

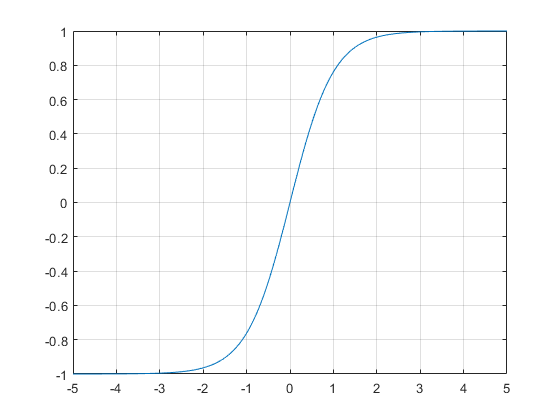


Рисунок 3 – Гиперболический тангенс

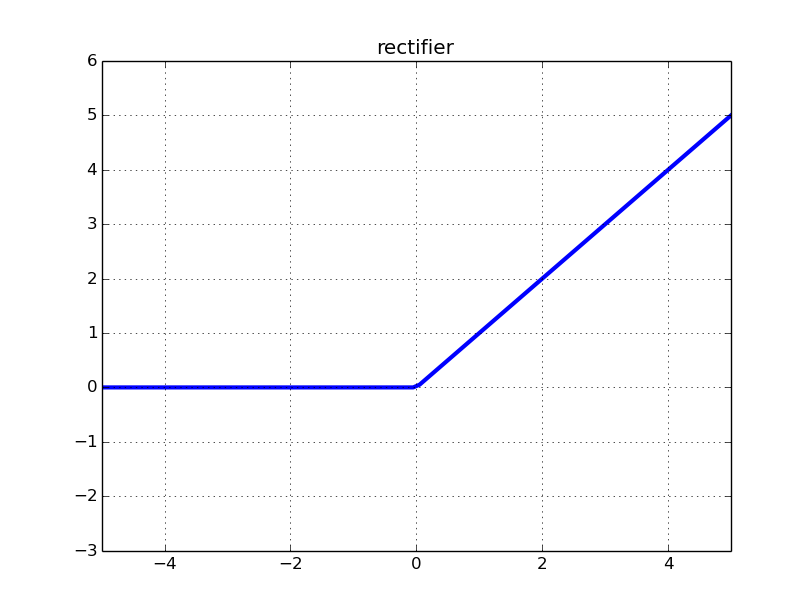


Рисунок 4 – ReLU

**1.4 SoftMax**

Для нейронов выходного слоя используется функция SoftMax. Эта функция переводит входной вектор в вероятность.

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

где

*–* вектор входов

*–* веса выхода [3].

**2 Обучение сети**

Перед рассмотрением алгоритмов оптимизации необходимо определить функцию, которую будем минимизировать. В сетях классификации такой функцией выступает расстояние Кульбака – Лейблера.

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

где

P и Q - дискретные случайные велечины на дискретном множестве X=()

и – вероятности исхода [1].

**2.1 Алгоритмы оптимизации**

В нейронных сетях используют два алгоритма оптимизации:

– эволюционный;

– алгоритм обратного распространения ошибки.

Эволюционный алгоритм, берет свое начало из Дарвиновской теории. В его основе лежат те же принципы наследственности, изменчивости и естественного отбора. В начале у родителей создаются несколько потомков, с некоторыми мутациями. Затем выбираются наиболее приспособленные, и так-же оставляют потомство. Процесс будет продолжатся до тех пор, пока не получатся оптимальные варианты. Такой алгоритм, к сожалению не обеспечивает достаточную быстроту сходимости, поэтому используется относительно редко [1].

Ему в противовес идет алгоритм обратного распространения ошибки. В процессе его выполнения значение ошибки идет из выхода вверх по сети, попутно пересчитывая значения весов в нейронах. Сам пересчет может выполняться достаточно большим разнообразием алгоритмов оптимизации [4].

**2.2 Обратное распространение ошибки**

Указанные ниже обозначения иллюстрированы на рисунках 1 и 2.

Рассмотрим многослойную нейронную сеть. В каждом слое расположено  нейронов, , обозначаемых , , с нелинейной функцией (сигмоидой, либо гиперболическим тангенсом) на выходе. Исследуемая сеть имеет  входов , куда подаются значения , записываемые в векторной форме:

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

Выходной сигнал -го нейрона в -м слое обозначается ,

, .

На рисунке 1 показана структура -го нейрона в -м слое.

Нейрон  имеет  входов, которые образуют вектор

|  |  |  |
| --- | --- | --- |
|  | , | (7) |

причем  для  и . Следует обратить внимание на тот факт, что входной сигнал нейрона  связан с выходным сигналом -ого слоя следующим образом:

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

На рисунке 1  обозначает синаптический вес связи -го нейрона, , находящегося в -м слое, который соединяет этот нейрон с -м входным сигналом . Вектор весов нейрона  обозначим

|  |  |  |
| --- | --- | --- |
|  | . | (9) |

Выходной сигнал нейрона в -й момент времени,   
  определяется как

|  |  |  |
| --- | --- | --- |
|  | , | (10) |

причем

|  |  |  |
| --- | --- | --- |
|  |  | (11) |

Отметим, что выходы нейронов в -м слое задаются так

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

одновременно являются выходами всей сети. Они сравниваются с так называемыми эталонными сигналами сети

|  |  |  |
| --- | --- | --- |
|  |  | (13) |

в результате чего получается погрешность

|  |  |  |
| --- | --- | --- |
|  | . | (14) |

Можно сформулировать меру погрешности, основанную на сравнении сигналов (12) и (13), в виде суммы квадратов разностей (14), т.е.

|  |  |  |
| --- | --- | --- |
|  | . | (15) |

Из выражений (10) и (11) следует, что мера погрешности (15) − это функция от весов сети. Обучение сети основано на адаптивной коррекции всех весов  таким образом, чтобы минимизировать ее значение. Для коррекции произвольного веса можно использовать правило градиентного спуска, которое принимает вид

|  |  |  |
| --- | --- | --- |
|  |  | (16) |

где константа  определяет величину шага коррекции, также называемой скоростью обучения. Обратим внимание, что

|  |  |  |
| --- | --- | --- |
|  |  | (17) |

Если ввести обозначение

|  |  |  |
| --- | --- | --- |
|  |  | (18) |

то получим равенство

|  |  |  |
| --- | --- | --- |
|  |  | (19) |

При этом алгоритм (16) принимает вид

|  |  |  |
| --- | --- | --- |
|  |  | (20) |

Способ расчета значения , заданного выражением (18), зависит от номера слоя. Для последнего слоя получаем

|  |  |  |
| --- | --- | --- |
|  |  | (21) |
|  |  | (22) |
|  |  | (23) |

Для произвольного слоя  получим

|  |  |  |
| --- | --- | --- |
|  |  | (24) |
|  |  | (25) |
|  |  | (26) |

Найдем погрешность в -м (не последнем) слое для -го нейрона в виде

|  |  |  |
| --- | --- | --- |
|  |  | (27) |

Если подставить выражение (27) в формулу (26), то получим

|  |  |  |
| --- | --- | --- |
|  |  | (28) |

В результате алгоритм [4] обратного распространения ошибки записывается в виде:

|  |  |  |
| --- | --- | --- |
|  |  | (29) |
|  |  | (30) |
|  |  | (31) |
|  |  | (28) |
|  |  | (32) |

**3 Настройка моделей**

Следует понимать, что на результат обучения сети могут сильно влиять параметры алгоритма обучения. Окончательный выбор зависит от требуемой скорости выполнения. Как правило, чем меньше сеть, − тем дольше она обучается и тем быстрее она работает. Наибольшее влияние оказывает число нейронов в промежуточном слое [5].

Кроме того необходимо учитывать размерность самих данных, поскольку все значения одного признака должны обладать одним порядком [6].

**3.1 Количество нейронов в скрытом слое**

Для приблизительной оценки числа нейронов в скрытом слое однородной нейронной сети с сигмоидальными передаточными функциями воспользовались формулой, являющейся следствием из теоремы Арнольда−Колмогорова−Хета−Нильсона [7]

|  |  |  |
| --- | --- | --- |
|  |  | (33) |

где

– размерность входного слоя;

– размерность выходного слоя;

– размер обучающей выборки;

 − число синаптических связей.

Число нейронов в скрытом слое нейронной сети рассчитывается из равенства

|  |  |  |
| --- | --- | --- |
|  |  | (34) |

**4 Программная реализация классификационной сети на Python**

Прежде чем продолжить, необходимо ответить на вопрос: почему был выбран именно Python? На это есть две причины.

Первая – впечатляющая скорость разработки. Это определяется двумя факторами – простотой синтаксиса, существенно уменьшающей количество вводимого кода, и вместе с этим, отсутствие этапа компиляции этого кода. Оба фактора значительно уменьшают время, затрачиваемое на реализацию кода.

Вторая – Python, является стандартом языка для работы с данными. Его выделяет простота использования, сравнимая с предметно - ориентированными скриптовыми языками типа MATLAB. В то-же время он сравним по универсальности и мощности с объектно-ориентированными языками типа C++.

На Python реализовано множество библиотек работы с данными. В том числе для статистических вычислений и визуализации данных, обработки изображений и звуков и прочие [1]. Но самое важное в контексте данной работы – библиотеки для машинного обучения, такие как TensorFlow, или scikit-learn.

Далее будет представлена простейшая реализация нейронной сети на Python версии 3.6.5. Исходный код представлен в приложении А . Были использованы библиотеки numpy и scikit-learn версий 1.14.3 и 0.19.1 соответственно.

**4.1 Постановка задачи и решение**

Выборка состоит из 150 человек. Данные необходимо классифицировать по двум категориям «здоров» и «болен». Вычисления производятся на основе десяти признаков: Q19/Q2, Q20/Q2, Q21/Q2, Q13/2, Q9/Q6, Q12/Q6, Q18/Q6, Q10/Q6, Q8/Q3 и Q14/Q3. Для использованного набора данных равна 7.3,а равна 418. Все значения признаков одной размерности, что не добавляет необходимости преобразовывать их. Поскольку размерность выборки недостаточно большая – было принято решение разбивать данные на тренировочные и тестовые как 120 на 30.

Программа получила название Rex.py

**4.2 Результат Rex.py и описание принципа его работы**

При запуске программа выдает следующие результаты:

– точность на тренировочном наборе: 95.33 процента;

– точность на тестовом наборе: 90 процентов.

Подробные результаты можно рассмотреть в приложении Б.

При запуске скрипта прежде всего запускается метод \_\_init\_\_ класса DataClassNP, являющегося наследным для класса DataClass. Оба класса идентичны, за исключением метода list\_to\_matrix, переводящего тип list в тип numpy.array. Метод необходим, поскольку scikit-learn принимает для вычислений данные именно этого типа. Сам метод \_\_init\_\_ открывает файл data.txt с выборкой и считывает её в два списка объекта (словаря data\_dict), список ‘features’, содержащий признаки, и ‘targets’– соответствующие им значения. Далее они преобразуются в указанную ранее матрицу.



Рисунок 5 – класс DataClassNP и вызов его метода list\_to\_matrix

Полученный ранее массив затем разбивается на тренировочный и тестовый, посредством функции train\_test\_split. Параметр train\_size определяет размер тренировочного набора данных. В данном случе он равен 120, что определяет разбиение выборки на обучающую и тестовую, как четыре к одному. random\_state фиксируется, для стабильности результата.

Далее создается модель классификатора MLPClassifier. Аргумент solver определяет алгоритм оптимизации. hidden\_layers\_sizes –количество и размерность скрытых слоев. В данной программе он имеет значение 8, что соответствует указанному ранее диапазону. Это значение было получено экспериментально, и дает наилучшую производительность для данных.

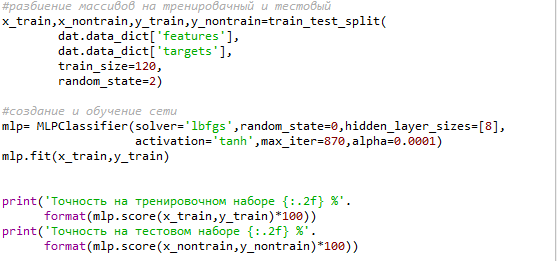


Рисунок 6 – Разбиение массивов данных, создание и обучение сети

Помимо прочего, в этой функции присутствует параметр alpha, являющийся коэффициентом обучаемости, его значение по умолчанию равно 0.0001. При alpha,к примеру равном 1 будет получен результат:

– точность на тренировочном наборе: 70 процентов;

– точность на тестовом наборе: 63.33 процента.

Итог говорит сам за себя. В большинстве случаев его лучше оставлять без изменений. Параметр activation передает функцию активации нейронов. Параметр max\_iter задает предельное значение итераций. Стоит заметить, что оно будет достигаться далеко не всегда.

Далее идет метод модели – fit. Он и обучает сеть, на основе массивов x\_train и y\_train.

**ЗАКЛЮЧЕНИЕ**

В данной работе были рассмотрены основные принципы работы нейросетей, ее структура и методы обучения, а также разобран метод обратного распространения ошибки для поиска весов. Была представлена базовая сеть классификации на Python.

На сегодняшний день алгоритмы на основе нейронных сетей являются очень многообещающими. Они позволят решать весьма сложные классификационные задачи и приближать очень сложные функции. По большей части эта тематика остается достаточно малоизученной, но при этом развивается весьма бурными темпами. Вполне возможно, что исследования в этом направлении позволят создать машинный разум который по уровню абстракции будет близок к человеческому. Рассмотренная проблема была лишь малой частью из того многообразия задач, которые задает нам окружающий мир, однако, очевидно, что подобные алгоритмы будут оставаться актуальными достаточно долго.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 Николенко С., Кадурин А., Архангельская Е. Глубокое обучение погружение в мир нейронных сетей. М.: СПб.:Питер, 2018г. – 480с.

2 Халафян А. А., Калайдина Г. В., Пелипенко Е. Ю. Теория вероятностей и математическая статистика: учебное пособие. М.: Краснодар: Кубанский гос. ун-т, 2018 г – 184 с.

3 The Softmax Function, Simplified URL – https://towardsdatascience.com/softmax-function-simplified-714068bf8156 (20.12.2018).

4 Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. М.: Горячая линия - Телеком, 2006. – 452 с.

5 Пелипенко Е. Ю. Разработка систем поддержки принятия решений для слабоструктурированных задач классификации многомерных объектов: дис… канд. техн. наук: 05.13.01 /Пелипенко Е.Ю. Кубан. гос. ун-т. – Краснодар, 2016. – 164 с.

6 Мюллер А., Гвидо С. Введение в машинное обучение с помощью Python М.: Москва, 2016. – 393 с.

7 Widrow B., Lehr M.A. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation // Proceedings of the IEEE, vol. 78, No. 9, September, 1990, p. 1415-1442.

**ПРИЛОЖЕНИЕ А**

**Исходный код Rex**

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.neural\_network import MLPClassifier

class DataClass():

data\_dict={'target\_types':{'здоров':1, 'болен':0},

'features':[], 'targets':[]}

def \_\_init\_\_(self,file\_name='data.txt'):

#открытие файла

with open(file\_name,'r') as file:

#построчное чтение из файла

for line in file:

#преобразование сторки в список слов

lst=line.split('\t')

#преобразование второго и последующих слов в числа

for i in range(1,len(lst)):

lst[i]=float(lst[i])

#преобразование превого слова в коэффициент

#из словоря result\_types и запись его в targets

if lst[0] in self.data\_dict['target\_types']:

self.data\_dict['targets'].append(

self.data\_dict['target\_types'][lst[0]])

else:

self.data\_dict['targets'].append(-1)

Рисунок А.1 – Исходный код часть 1

self.data\_dict['features'].append(lst[1:])

class DataClassNP(DataClass):

def list\_to\_matrix(self):

self.data\_dict['features']=np.array( self.data\_dict['features'])

self.data\_dict['targets']=np.array( self.data\_dict['targets'])

#ввод и составление массива данных

dat=DataClassNP()

dat.list\_to\_matrix()

#разбиение массивов на тренировачный и тестовый

x\_train,x\_nontrain,y\_train,y\_nontrain=train\_test\_split(

dat.data\_dict['features'],

dat.data\_dict['targets'],

train\_size=120,

random\_state=2)

#создание и обучение сети

mlp= MLPClassifier(solver='lbfgs',random\_state=0,hidden\_layer\_sizes=[8],

activation='tanh',max\_iter=870)

mlp.fit(x\_train,y\_train)

print('Точность на тренировочном наборе {:.2f} %'.

format(mlp.score(x\_train,y\_train)\*100))

print('Точность на тестовом наборе {:.2f} %'.

format(mlp.score(x\_nontrain,y\_nontrain)\*100))

Рисунок А.2 – Исходный код часть 2

**ПРИЛОЖЕНИЕ Б**

**Результаты работы нейронной сети**

Таблица Б.1 – Итоговые результаты для тренировочной выборки

|  |  |  |  |
| --- | --- | --- | --- |
|  | c/h-болен | c/h-здоров | всего |
| Все | 60 | 60 | 120 |
| Правильно | 57 | 58 | 115 |
| Неправильно | 3 | 2 | 5 |
| Правильно (%) | 95 | 96,676 | 95,833 |
| Неправильно (%) | 5 | 3,333 | 4,617 |

Таблица Б.2 – Итоговые результаты для тестовой выборки

|  |  |  |  |
| --- | --- | --- | --- |
|  | c/h-болен | c/h-здоров | всего |
| Все | 15 | 15 | 30 |
| Правильно | 14 | 13 | 27 |
| Неправильно | 1 | 2 | 3 |
| Правильно (%) | 93,333 | 86,667 | 90 |
| Неправильно (%) | 6,667 | 13,333 | 10 |

