МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Теоретической и прикладной информатики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Чубич В.М.
(фамилия, имя, отчество)

_____
(подпись)

«_____»_____ 2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Антонова Сергея Сергеевича
(фамилия, имя, отчество студента – автора работы)

Разработка алгоритмов обучения нейронных сетей простой архитектуры с использованием генетических алгоритмов
(тема работы)

Факультет Прикладной математики и информатики
(полное название факультета)

Направление подготовки 02.03.03. Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки бакалавра)

Руководитель от НГТУ	**Автор выпускной квалификационной работы**
Тимофеев В.С. (фамилия, имя, отчество)	Антонов С.С. (фамилия, И.О.)
д.т.н., доцент (ученая степень, ученое звание)	ФПМИ, ПМИ-71 (факультет, группа)
(подпись, дата)	(подпись, дата)

Новосибирск, 2021 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Теоретической и прикладной информатики
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Чубич В.М.
(фамилия, имя, отчество)

«09» марта 2021 г.

(подпись)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

студенту ***Антонову Сергею Сергеевичу***
(фамилия, имя, отчество студента)

Направление подготовки *02.03.03. Математическое обеспечение и администрирование информационных систем*

Факультет Прикладной математики и информатики

Тема ***Разработка алгоритмов обучения нейронных сетей простой архитектуры с использованием генетических алгоритмов***

Исходные данные (или цель работы):

Цель работы – разработка и исследование нейронной сети с использованием генетическихалгоритмов

Структурные части работы:

В первом разделе определяются общие понятия нейронных сетей, рассматривается премущества и недостатки нейронных сетей,архитектура и разновидности нейронных сетей, также описывается математическое представление функций активатора и потерь. Во втором разделе определяются общие понятия генетических алгоритмов,рассматривается структура генетических алгоритмов и их работу в нейронных сетях, а также математически представляется функция представления .

В третьем разделе приводятся набор данных для исследований, рассматривается структура программы , а также результаты работы.

Задание согласовано и принято к исполнению.

Руководитель от НГТУ	**Студент**
Тимофеев В.С.	*Антонов С.С.*
(фамилия, имя, отчество)	(фамилия, имя, отчество)
д.т.н., доцент	*ФПМИ, ПМИ-71*
(ученая степень, ученое звание)	(факультет, группа)
09.03.2021 г.	*09.03.2021 г.*
(подпись, дата)	(подпись, дата)

Тема утверждена приказом по НГТУ № *969/2* от «*09*» *марта 2021* г.

ВКР сдана в ГЭК № 31, тема сверена с данными приказа

09 марта 2021 г.
(подпись секретаря экзаменационной комиссии по защите ВКР, дата)

Волкова В.М.
(фамилия, имя, отчество секретаря экзаменационной комиссии по защите ВКР)

АННОТАЦИЯ

Отчет 47 с., 3 р., 13 рис., 1 табл.,4 формулы, 11 источников, 1 прил.

ГЕНЕТИЧЕСКИЙ, АЛГОРИТМ, МУТАЦИЯ, СЛИЯНИЕ, ПОПУЛЯЦИЯ, НЕЙРОННЫЕ СЕТИ, ОБУЧЕНИЕ С УЧИТЕЛЕМ,СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ, ФУНКЦИЯ АКТИВАЦИИ.

Объектом исследования являются нейронные сети и их реализация с оптимизацией генетическими алгоритмами.

Цель работы – разработка и исследование нейронной сети с использованием генетических алгоритмов.

В процессе работы была разработана программа, реализующая нейронную сеть с оптимизацией генетическими алгоритмами, проводились исследования влияния параметров генетического алгоритма и нейронных сетей на результат его работы.

В результате исследований были подобраны оптимальные параметры для разработанной программы, реализующей нейронную сеть с генетическими алгоритмами.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Использование нейронных сетей для решения разнообразных задач стало весьма распространено в последние десятилетие.Например задачи классификации.Решая задачу , в первую очередь формируют и обязательно обучают нейронную сеть. Обучение нейронных сетей может быть реализовано при помощи разнообразных алгоритмов, выбор которого основывается от различных критериев.

Генетические алгоритмы обучения считаются довольно мощным методом,которые благополучно используются для обширного класса прикладных задач, в том числе и те, которые труднореализуемы или вовсе нереализуемы другими алгоритмами обучения.Генетические алгоритмы не гарантируют нахождение глобального решения за краткий промежут времени, однако могут использоваться для решения сложных задач.

Цель работы – разработка и исследование нейронной сети с использованием генетических алгоритмов

Для достижения этой цели были поставлены и решены следующие задачи:

Изучение возможностей нейронных сетей и их использование,

Разбор возможностей генетических алгоритмов и их адоптация в нейронные сети,

Разработка программа, реализующую нейронную сеть и оптимизированную генетическим алгоритмом.

По структуре данная работа состоит из введения, трех разделов основного содержания, заключения, списка источников и одного приложения.

В первом разделе определяются общие понятия нейронных сетей, рассматривается премущества и недостатки нейронных сетей,архитектура и разновидности нейронных сетей, также описывается математическое представление функций активатора и потерь.

Во втором разделе определяются общие понятия генетических алгоритмов,рассматривается структура генетических алгоритмов и их работу в

нейронных сетях, а также математически представляется функция представления .

В третьем разделе приводятся набор данных для исследований, рассмотривается структура программы , а также результаты работы.

В приложении А помещен исходный код всех модулей (классов) разработанной программы.

1. НЕЙРОННЫЕ СЕТИ И ИХ ОБУЧЕНИЕ

1.1 Нейронные сети

Исследования по искусственный нейронным сетям связаны с тем, что способ обработки информации человеческим мозгом в корне отличается от методов, применяемых обычными цифровыми компьютерами.Мозг представляет собой чрезвычайно сложный, нелинейный, паралеллельный компьютер(систему обработки информации).Он обладает способностью организовывать свои структурные компоненты, называемые нейронами, так , чтобы они могли выполнять конкретные задачи (такие как распозвнование образов ,обработка сигналов органов чувств, моторные функции) во много раз быстрее, чем могут позволить самые быстродействующие современнные компьютеры. Примером такой задачи информации может служить обычное зрение.В функции зрительной системы входить создание представления окружающего мира в таком виде, который обеспечивает возможность взаимодействия с этим миром.Более точно, мозг последовательно выполняет ряд задач распознавания(например,распознавание знакомого лицав незнакомом окружении).На это у него уходит около 100-200 миллисекунд, в то время как выполнение аналогичных задач даже меньше сложности на компьютере может занять несколько дней.

При рождении мозг имеет совершенную структуру, позволяющую строить собственные правила на основании того, что мы называем “опытом”.Опыт накапливается с течением времени, и особенно масштабные изменения происходят в первые два года жизни человека.В этот период формируется остов обще йструктуры.Однако развитие на этом не прекращается – оно продолжается до последний дней жизни человека.

Понятие развития нейронов связано с понятием пластичности мозга – способности настройки нервной системы в соотвествии с окружающими условиями.Именно пластичность играет самую важную роль в работе нейронов в качестве единиц обработки информации в человеческом мозге.Аналогично, в

искусственных нейронный сетях работа проводится с искусственными нейронами.В общем случаем нейронная сеть представляет собой машину, моделирующую способ обработки мозгом конкретной задачи. Эта сеть обычно реализуется с помощью электронных компонентов или моделируется программой, выполняемой на цифровом компьютере.Предметом рассмотрения этой работы является важдный класс нейронных сетей, осущесвляющих вычисления с помощью процесса обучения.таким образом, можно дать следующее определние нейронных сетей, выступающих в роли адаптивной машины.

Нейронная сеть – это громадный распределительный параллельный процессор, состоящий из элементарных единиц обработки информации, накапливающий экспериментальные знания и предоставляющих их для последующей обработки.

Процедура,используемая для процесса обучения,называется алгоритмом обучения.Эта процедура выстраивает в определенном порядке синаптические веса нейронной сети для обеспечения необходимой структуры взаимосвязей нейронов.

Изменение синакптических весов представляет собой традиционный метод настройки нейронных сетей.Этот подход очень близок к теории линейных адаптивных фильтров, которая уже давно заявила о себе и применяется в различных областях деятельности человека. Это обусловлено тем фактом, что нейроны в человеческом мозге постоянно отмирают, а новые синаптические связи постоянно создаются.

1.1 Приемущества и недостатки нейронных сетей

Существуют различные преимущества нейронных сетей, некоторые из которых обсуждаются ниже:

Хранить информацию по всей сети.Точно так же, как это происходит в традиционном программировании, где информация хранится в сети, а не в базе

данных. Если несколько фрагментов информации исчезают из одного места, это не останавливает работу всей сети.

2) Умение работать с недостаточными знаниями:После обучения ИНС вывод данных может быть неполным или недостаточным. Важность этой недостающей информации определяет отсутствие производительности.

3) Хорошая толерантность к фолту:на генерацию вывода не влияет повреждение одной или нескольких ячеек искусственной нейронной сети. Это делает сети более устойчивыми к сбоям.

4) Распределенная память:Чтобы искусственная нейронная сеть стала способной к обучению, необходимо набросать примеры и обучить ее в соответствии с желаемым результатом, показывая эти примеры сети. Ход сети прямо пропорционален выбранным экземплярам.

5)Постепенное разложение:Действительно, сеть подвергается относительной деградации и со временем замедляется. Но это не сразу разъедает сеть.

6) Возможность тренировать машину:ИНС извлекает уроки из событий и принимает решения, комментируя похожие события.

7) Возможность параллельной обработки:Эти сети обладают числовой мощью, что позволяет им выполнять более одной функции одновременно.

Из недостатков можно выделить :

1)Этот тип сети подходит только для обученных данных. Когда входные данные уходят далеко от обучающей выборки, результаты часто неудовлетворительны. Это делает приложение ненадежным. Думаю, никто не хочет, чтобы реактивный самолет упал на землю только потому, что облако перед ним выглядит сумасшедшей формы.

2)Этот тип сети создает очень необычные категории, которые используются для распознавания и классификации входных данных. Очень часто они не имеют ничего общего с категориями, используемыми людьми. Возникает проблема, как доказать достоверность результатов, полученных сетью. Нет доверия - нет сделки.

3) Исходные данные, обрабатываемые ИНС, обычно имеют большой размер. Это создает проблемы с производительностью. Беспилотный автомобиль не может долго ждать, пока распознают дорожный знак, потому что он движется быстро. Таким образом, данные уплотняются (свертываются), чтобы алгоритмы могли работать с достаточно небольшими важными данными. Но как выбрать правильную свертку - огромная проблема

4) Типичная сеть создает плоский выходной массив категорий. Это удовлетворительно для некоторых приложений, но для других случаев требуется более сложная модель. Редкие матрицы должны быть заменены иерархией графов по многим причинам, но здесь мы касаемся предела нашего понимания того, как построить такую модель и управлять ею, и как достичь ее соответствия миру, который она описывает. Здесь больше вопросов, чем ответов.

1.2 Архитектура сетей

Нейронные сети - это сложные структуры, состоящие из искусственных нейронов, которые могут принимать несколько входных сигналов для получения одного выходного сигнала[5]. Это основная задача нейронной сети - преобразовывать ввод в осмысленный вывод. Обычно нейронная сеть состоит из входного и выходного слоя с одним или несколькими скрытыми слоями внутри.

В нейронной сети все нейроны влияют друг на друга и, следовательно, все они связаны. Сеть может распознавать и наблюдать каждый аспект имеющегося набора данных, а также то, как различные части данных могут или не могут быть связаны друг с другом. Таким образом нейронные сети могут находить чрезвычайно сложные закономерности в огромных объемах данных.В нейронной сети поток информации происходит двумя способами:

Сети прямого распространения достаточная простая архитектура: в этой модели сигналы распространяются только в одном направлении, к выходному слою. Сети прямого распространения имеют входной слой и единственный выходной слой с нулевым или несколькими скрытыми слоями. Они широко используются при распознавании образов.

Сети обратной связи: в этой модели повторяющиеся или интерактивные сети используют свое внутреннее состояние (память) для обработки последовательности входных данных. В них сигналы могут проходить в обоих направлениях через петли (скрытые слои) в сети. Обычно они используются в задачах временных рядов и последовательных задачах.

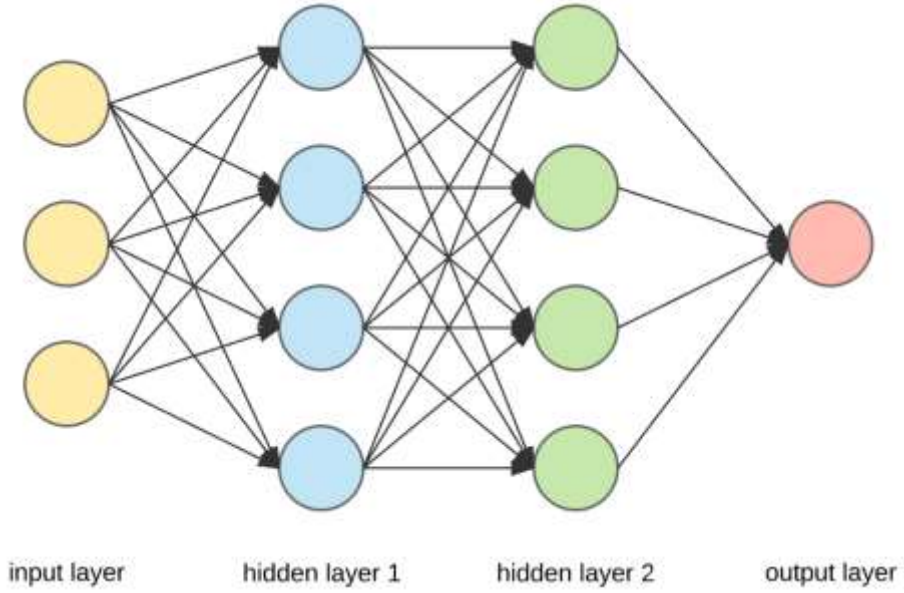

Рис.1 – Пример компоненты нейронной сети

Входные слои, нейроны и веса, на изображении, приведенном выше, внешний желтый слой является входным слоем. Нейрон - это основная единица нейронной сети[11]. Они получают входные данные от внешнего источника или других узлов. Каждый узел связан с другим узлом следующего уровня, и каждое такое соединение имеет определенный вес. Веса назначаются нейрону в зависимости от его относительной важности по сравнению с другими входами.

Когда все значения узлов из желтого слоя умножаются (вместе с их весом) и суммируются, создается значение для первого скрытого слоя. На основе суммарного значения синий слой имеет предопределенную функцию «активации», которая определяет, будет ли этот узел «активирован» и насколько «активным» он будет.

Давайте разберемся в этом с помощью простой повседневной задачи - заваривания чая. В процессе приготовления чая ингредиенты, используемые для приготовления чая (вода, чайные листья, молоко, сахар и специи), являются «нейронами», поскольку они составляют отправные точки процесса. Количество

каждого ингредиента представляет собой «вес». После того, как вы положите чайные листья в воду и добавите в кастрюлю сахар, специи и молоко, все ингредиенты смешаются и перейдут в другое состояние. Этот процесс трансформации представляет собой «функцию активации».

Скрытые слои и выходной слой.Слои, скрытые между входным и выходным слоями, известны как скрытый слой. Он называется скрытым слоем, поскольку он всегда скрыт от внешнего мира. Основные вычисления нейронной сети происходят в скрытых слоях. Итак, скрытый слой принимает все входные данные от входного слоя и выполняет необходимые вычисления для генерации результата. Затем этот результат пересылается на выходной уровень, чтобы пользователь мог просмотреть результат вычисления.

В нашем примере с приготовлением чая, когда мы смешиваем все ингредиенты, рецептура меняет свое состояние и цвет при нагревании. Ингредиенты представляют собой скрытые слои. Здесь нагревание представляет собой процесс активации, который в конечном итоге дает результат - чай.

1.2.1 Модели нейронов

Нейроны - это информация обрабатывающие ячейки.Прежде чем определять функции и процессы в нейроне, мы дадим.приблизительное описание функций нейрона: нейрон - это не что иное, как переключатель с ввода и вывода информации. Выключательбудет активирован при наличии достаточного количества стимулов от других нейронов, попадающих на информационный вход. Затем на выходе информации импульс отправляется, например, на другой нейроны.

Компоненты нейрона. Теперь взглянем на компоненты нейрона. При этом мы будем следовать как электрическая информация попадает внутрь нейрон. Дендриты нейрона получаем информацию по специальным связям, синапсам

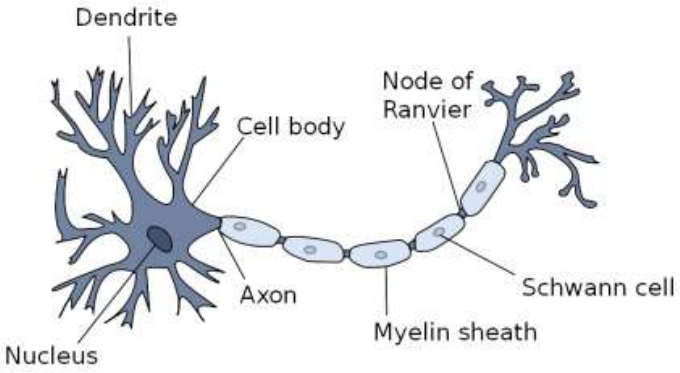

Рис.2 – Модель нейрона

1.2.2 Представление нейронной сети в виде графа

Искусственная нейронная сеть (ИНС) может рассматриваться как направленный граф с взвешенными связями, в котором искусственные нейроны являются узлами.Матрицу весов связей обученной нейронной сети можно отнести к эвристическим моделям представления знаний.Например, если представить мою нейронную сеть ввиде графа, то она будет иметь вид, как показано на рисунке 3.Где каждый круг – это нейрон

На графе можно увидеть ,что архитектура нейронной сети достаточно простая.Сеть прямого распространения с тремя слоями,на входном слое 4 нейрона , на скрытом -6 , а на выходном – 1.

1.3 Разновидности нейронных сетей

Существуют разные типы нейронных сетей. Все они используют разные принципы и определяют свои правила. Существуют различные типы искусственных нейронных сетей, и каждая из них обладает уникальной и особой силой.

1) Нейронная сеть с прямой связью - искусственный нейрон

Простейший тип искусственной нейронной сети[10]. В этом типе данные проходят через различные входные узлы, пока наконец не достигнут выходного узла.Этот тип сетей считается достаточно простой архитектурой.

Проще говоря, данные движутся только в одном направлении. Это также называется фронтальной волной, что обычно достигается путем классификации функции активации. Эта нейронная сеть может иметь только один слой или несколько скрытых слоев.

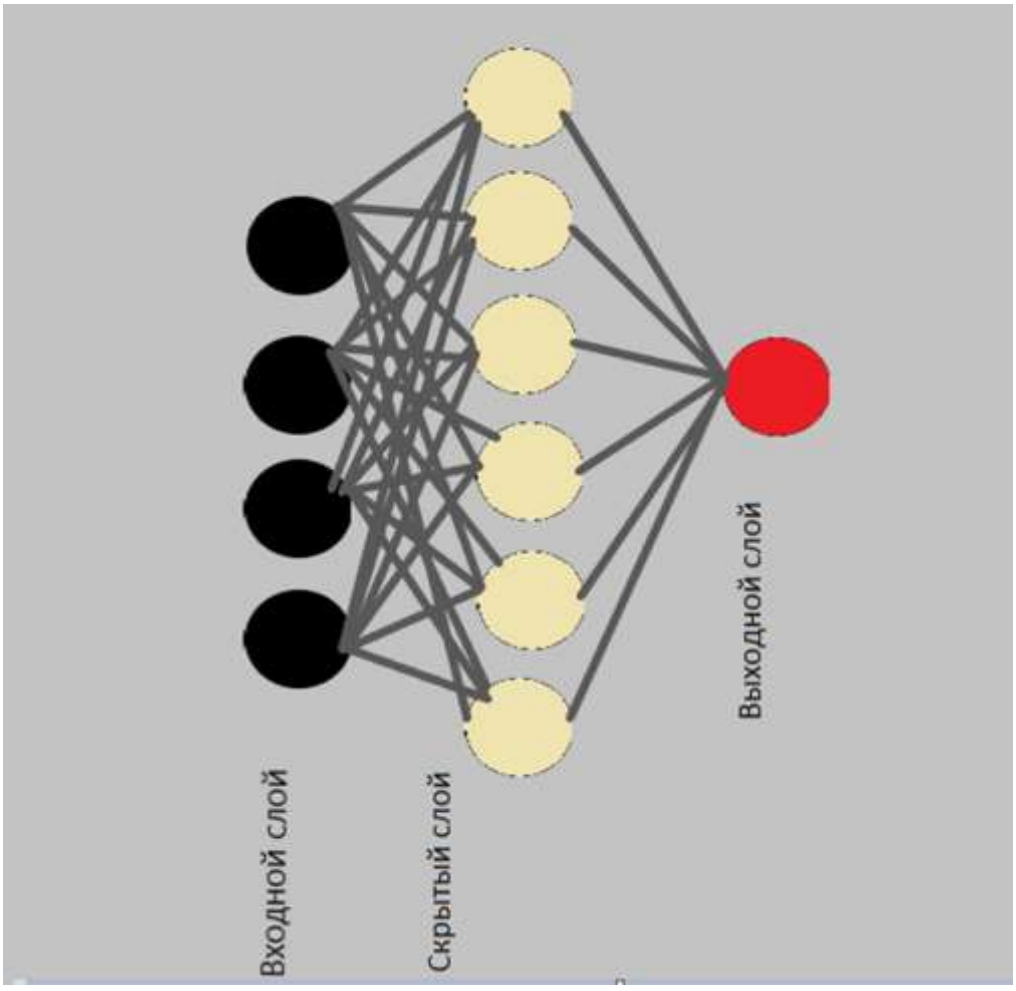

Рис.3 – Представление нейронной сети в виде графа

В нейронной сети с прямой связью вычисляется сумма произведений всех входов и их весов, которые позже передаются на выход.

2) Нейронная сеть с радиальной базисной функцией

Этот тип нейронной сети учитывает расстояние любой определенной точки относительно центра. Эти сети состоят из двух слоев. Во внутреннем слое функции объединены с радиальной базисной функцией. Выход данных функций учитывается, когда тот же результат вычисляется на следующем временном шаге.

Этот тип нейронной сети обычно применяется в системах восстановления питания.

3) Многослойный персептрон

Эта нейронная сеть состоит из трех или более трех слоев. Он в основном используется для классификации данных, которые нельзя разделить линейно. Этот тип искусственной нейронной сети полностью связан, и это потому, что каждый отдельный узел, присутствующий в слое, подключен к узлам на следующем слое.

Он использует нелинейную функцию активации. Многослойная нейронная сеть персептрона применяется в технологиях машинного перевода и распознавания речи .

4) Сверточная нейронная сеть

Этот тип нейронной сети использует разновидность многослойных персептронов. Сверточные нейронные сети содержат один или несколько слоев, которые могут быть объединены или полностью связаны между собой.

Они показывают хорошие результаты при обнаружении перефразирования и семантическом разборе. Они применяются при классификации изображений и обработке сигналов.

5) Рекуррентная нейронная сеть (RNN) - долговременная краткосрочная память

Это тип искусственной нейронной сети, в которой выходные данные определенного слоя сохраняются, а затем возвращаются на вход. Это помогает предсказать результат слоя. Формирование первых слоев такое же, как и в сети прямого распространения.

Рекуррентная нейронная сеть начинается с переднего распространения, но запоминает всю информацию, которая может понадобиться для использования позже.

Искусственная нейронная сеть используется в технологии преобразования текста в речь.

6) Модульная нейронная сеть

Эта нейронная сеть состоит из множества различных сетей, работающих независимо друг от друга и выполняющих подзадачи. Они не взаимодействуют друг с другом в процессе вычислений. Самостоятельно работать для достижения результата.

7) Модели от последовательности к последовательности

Он содержит две повторяющиеся нейронные сети. Присутствует кодировщик, который обрабатывает ввод, а вывод обрабатывает декодер. Кодер и декодер могут использовать похожие или даже разные параметры.

1.4 Обучение нейронных сетей

Существуют разные парадигмы обучения.Обучение - это всеобъемлющий термин. Система обучения изменяется, чтобы адаптироваться апример изменения окружающей среды. Нейронная сеть может многому научиться, но,конечно, всегда будет вопрос, как это реализовать. В принципе,нейронная сеть изменяется, когда меняются ее компоненты. Мы позволяем ашей нейронной сети учиться, изменяя связующие веса в соотвествии с правилами, которые можно сформировать в виде алгоритмов.Поэтому процесс обучения всегда связан с алгоритмимами оптимизации.

Также при обучении нейронных сетей нужно сформировать или использовать готовые обучающие выборки и обучающие и тестовые выборки. В обучающую выборку входит набор данных для обучения нейронной сети.А тестовую выборку для оценки качества работы нейронной сети

Существует две основные парадигмы обучения нейронной сети.Один из них это обучение с учителем.Методы обучения с учителем обеспечивают модели обучения вместе с соответствующие желаемые результаты При обучении с учителем обучающий набор состоит из шаблонов ввода, а также их правильные результаты в виде точной активации всех выходных нейронов. Таким образом, для каждый обучающий набор, который подается в сеть, например, может напрямую сравниваться с правильным решением, и веса сети могут быть изменены в зависимости от их различия. Обучающий набор состоит из шаблонов

ввода с правильные результаты, чтобы сеть могла получить точный вектор ошибки.Второй - обучение без учителя.Обучение без учителя предоставляет сети шаблоны ввода, но нет помощников по обучению.Обучение без учителя - это наиболее приемлемый с биологической точки зрения метод, но он не подходитдля всех проблем. Даны только входные шаблоны; сеть пытается выявить похожие шаблоны и классифицировать их по аналогичным категориям.Обучающий набор состоит только из входныхпаттернов, сеть сама пытается обнаружить сходство и сгенерировать паттерн классы.

1.6 Математическое представление активатора и функции потерь

Функция активации — это один из самых мощных инструментов, который влияет на силу, приписываемую нейронным сетям. Отчасти, она определяет, какие нейроны будут активированы, другими словами и какая информация будет передаваться последующим слоям.

Без функций активации глубокие сети теряют значительную часть своей способности к обучению. Нелинейность этих функций отвечает за повышение степени свободы, что позволяет обобщать проблемы высокой размерности в более низких измерениях. В программе использованы две функции активации :

- **Сигмоидальная функция**

На самом деле существует целое семейство сигмоидальных функций, некоторые из которых применяют в качестве функции активации в искусственных нейронах.Все эти функции обладают некоторыми очень полезными свойствами, ради которых их и применяют в нейронных сетях.

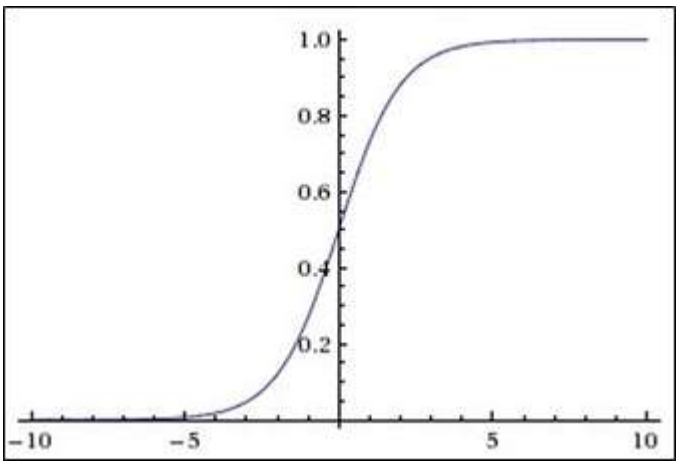

Рис.4 – График сигмоидальной функции

График этой функции выглядит достаточно просто. Если присмотреться, то можно увидеть некоторое подобие английской буквы S, откуда и пошло название семейства этих функций.

А вот так она записывается аналитически:

$$f(s)=\frac{1}{1+e^{x}} \qquad (1)$$

Где s – веса поступающие для активации,

А e – число Эйлера

- **Гиперболический тангенс**

Однако есть и еще одна сигмоида – гиперболический тангенс. Он применяется в качестве функции активации биологами для более реалистичной модели нервной клетки.Такая функция позволяет получить на выходе значения разных знаков (например, от -1 до 1), что может быть полезным для ряда сетей.

Функция записывается следующим образом:

$$f(s)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad (2)$$

Где s – веса поступающие для активации

А e – число Эйлера

А вот так выглядит график этой функции

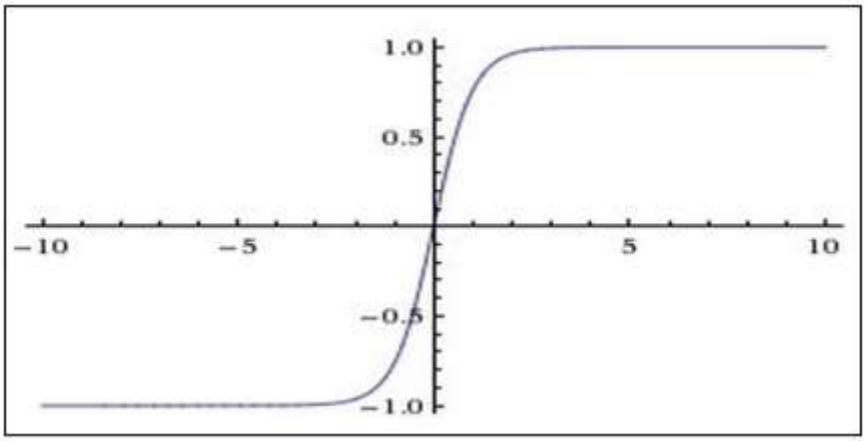

Рис.5 – График гипорболического тангенса

Функция потерь (Loss Function, Cost Function, Error Function; J) – фрагмент программного кода, который используется для оптимизации Алгоритма (Algorithm) Машинного обучения (Machine Learning). Значение, вычисленное такой функцией, называется «потерей».

Среднеквадратичная ошибка (MSE), возможно, является самой простой и наиболее распространенной функцией потерь, которую часто преподают на вводных курсах машинного обучения. Чтобы рассчитать MSE, вы берете разницу между предсказаниями вашей модели и основополагающей правдой, возводите ее в квадрат и усредняете ее по всему набору данных.MSE никогда не будет отрицательным, так как всегда возводится в квадрат ошибки. MSE формально определяется следующим уравнением:

$$MSE=\frac{1}{N}\sum_{i=1}^{N}(yi-yj)^2 \qquad (3)$$

Где yi –фактический ожидаемый результат

А yj – это прогноз модели.

2. ИСПОЛЬЗОВАНИЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ПРИ ОБУЧЕНИИ НЕЙРОННЫХ СЕТЕЙ

2.1 Генетические алгоритмы

Позаимствовавший идею у теории эволюции Чарльза Дарвина, один из самых удивительных методов решения задач заслуженно получил название «эволюционные вычисления». Самыми известными и широко распространенными представителями этого семейства являются генетические алгоритмы.

Генетические алгоритмы – это семейство поисковых алгоритмов, идеи которых подсказаны принципами эволюции в природе. Имитируя процессы естественного отбора и воспроизводства, генетические алгоритмы могут находить высококачественные решения задач, включающих поиск, оптимизацию и обучение[1]. В то же время аналогия с естественным отбором позволяет этим алгоритмам преодолевать некоторые препятствия, встающие на пути традиционных алгоритмов поиска и оптимизации, особенно в задачах с большим числом параметров и сложными математическими представлениями.

Особенности генетических алгоритмов определяют их преимущества по сравнению с традиционными алгоритмами поиска.

Ниже перечислены основные преимущества генетических алгоритмов:

- способность выполнять глобальную оптимизацию;
- применимость к задачам со сложным математическим представлением;
- применимость к задачам, не имеющим математического представления;
- устойчивость к шуму;
- поддержка распараллеливания и распределенной обработки;
- пригодность к непрерывному обучению.

Гипотеза структурных элементов, лежащая в основе генетических алгоритмов, заключается в том, что оптимальное решение задачи может быть собрано из небольших структурных элементов, и чем их больше, тем ближе мы подходим к оптимальному решению. Индивидуумам, которые содержат

некоторые из желательных структурных элементов, назначается более высокая оценка. Повторные операции отбора и скрещивания приводят к появлению все лучших индивидуумов, передающих эти структурные элементы следующему поколению, возможно, в сочетании с другими успешными структурными элементами. Тем самым создается генетическое давление, направляющее популяцию в сторону появления все большего числа индивидуумов, обладающих структурными элементами, образующими оптимальное решение. В результате каждое поколение оказывается лучше предыдущего и содержит больше индивидуумов, близких к оптимальному решению.

Можно сказать, что генетические алгоритмы лучше применять для решения следующих задач:

Задачи со сложным математическим представлением. Поскольку генетическим алгоритмам нужно знать только значение функции приспособленности, их можно использовать для решения задач, в которых целевую функцию трудно или невозможно продифференцировать, задач с большим количеством параметров и задач с параметрами разных типов.

Задачи, не имеющие математического представления. Генетические алгоритмы не требуют математического представления задачи, коль скоро можно получить значение оценки или существует метод сравнения двух решений.

Задачи с зашумленной окружающей средой. Генетические алгоритмы устойчивы к зашумленным данным, например прочитанным с датчика или основанным на оценках, сделанных человеком.

Задачи, в которых окружающая среда изменяется во времени. Генетические алгоритмы могут адаптироваться к медленным изменениям окружающей среды, поскольку постоянно создают новые поколения, приспосабливающиеся к изменениям.

С другой стороны, если для задачи известен специализированный способ решения традиционным или аналитическим методом, то вполне вероятно, что он окажется эффективнее.

2.2 Структура генетических алгоритмов

Гипотеза структурных элементов, лежащая в основе генетических алгоритмов, заключается в том, что оптимальное решение задачи может быть собрано из небольших структурных элементов, и чем их больше, тем ближе мы подходим к оптимальному решению. Индивидуумам, которые содержат некоторые из желательных структурных элементов, назначается более высокая оценка. Повторные операции отбора и скрещивания приводят к появлению все лучших индивидуумов, передающих эти структурные элементы следующему поколению, возможно, в сочетании с другими успешными структурными элементами. Тем самым создается генетическое давление, направляющее популяцию в сторону появления все большего числа индивидуумов, обладающих структурными элементами, образующими оптимальное решение. В результате каждое поколение оказывается лучше предыдущего и содержит больше индивидуумов, близких к оптимальному решению Например, если имеется популяция четырехзначных двоичных строк и требуется найти строки с максимальной суммой цифр, то цифра 1 в любой из четырех позиций является хорошим структурным элементом. В процессе работы алгоритм будет определять решения, содержащие такие структурные элементы, и объединять их. В каждом поколении будет больше индивидуумов, содержащих 1 в различных позициях, а в конечном итоге получится строка 1111, объединяющая все четыре желательных структурных элемента.

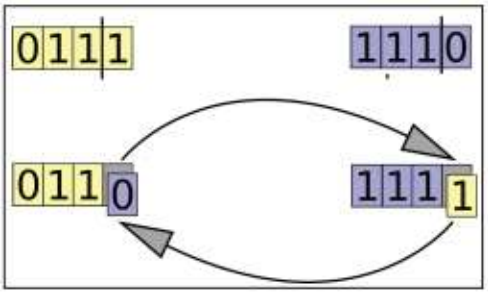

Рис.6 – Демонстрация операции скрещивания

Здесь мы видим, как два индивидуума, представляющих хорошие решения задачи (в каждом три бита равны 1), порождают потомка, дающего наилучшее решение (четыре единичных бита), после того как операция скрещивания

объединяет желательные элементы обоих родителей[2]. На следующей блок-схеме показаны основные этапы типичного генетического алгоритма. Управляемыми параметрами в генетическом алгоритме являются:

- длина хромосомы;
- наполнение хромосомы (локусы и аллели);
- параметры оператора кроссовера;
- параметры оператора мутации;
- параметры оператора инверсии;
- параметры выбора лучших особей;
- параметры генерации начальной и последующих популяций и т.д.

2.2.1 Популяция

В любой момент времени генетический алгоритм хранит популяцию индивидуумов – набор потенциальных решений поставленной задачи. Поскольку каждый индивидуум представлен некоторой хромосомой, эту популяцию можно рассматривать как коллекцию хромосом. Популяция всегда представляет текущее поколение и эволюционирует со временем, когда текущее поколение заменяется новым.

2.2.2 Метод отбора

После того как вычислены приспособленности всех индивидуумов в популяции, начинается процесс отбора, который определяет, какие индивидуумыбудут оставлены для воспроизводства, т. е. создания потомков образующихследующее поколение.Процесс отбора основан на оценке приспособленности индивидуумов.

Рис.7 – Базовая структура генетического алгоритма

Те,чья оценка выше, имеют больше шансов передать свой генетический материал следующему поколению.Плохо приспособленные индивидуумы все равно могут быть отобраны,но с меньшей вероятностью. Таким образом, их генетический материал неполностью исключен. Отбор выполняется в начале каждой итерации цикла генетического алгоритма, чтобы выбрать из текущей популяции тех индивидуумов, которые станут родителями индивидуумов в

следующем поколении. Отбор носит вероятностный характер, причем вероятность выбора индивидуума зависит от его приспособленности, так что у более приспособленных индивидуумов шансы отобраться выше.

Метод отбора по правилу рулетки, или отбор пропорционально приспособленности (fitness proportionate selection – FPS)[3], устроен так, что вероятность отбора индивидуума прямо пропорциональна его приспособленности.Тут можно провести аналогию с вращением колеса рулетки, где каждому индивидууму соответствует сектор, стоимость которого равна приспособленности индивидуума. Шансы, что шарик остановится в секторе индивидуума,пропорциональны размеру этого сектора. Пусть, например, имеется популяция из шести индивидуумов с такими значениями приспособленности, как в таблице ниже. По этим значениям вычисляются доли, занимаемые секторами каждого индивидуума.

Таблица 1– Пример данных

Индивидуум	Приспособленность	Доля
А	8	7 %
Б	12	10 %
В	27	23 %
Г	4	3 %
Д	45	39 %
Е	17	15 %

После каждого запуска рулетки отбор индивидуума из популяции производится в точке отбора. Затем рулетка запускается еще раз для выбора следующего индивидуума, и так до тех пор, пока не наберется достаточно индивидуумов для образования следующего поколения. В результате один и тот же индивидуум может быть выбран несколько раз[9]. На рисунке 8 изображена соответствующая рулетка.

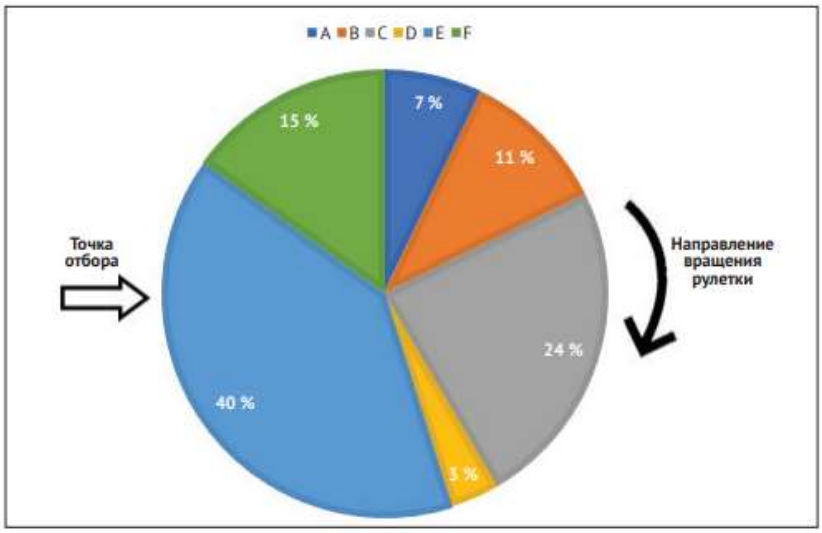

Рис.8 – Пример отбора по правилу рулетки

2.2.3 Метод скрещивания

Для создания пары новых индивидуумов родители обычно выбираются из текущего поколения, а части их хромосом меняются местами (скрещиваются), в результате чего создаются две новые хромосомы, представляющие потомков. Эта операция называется скрещиванием, или рекомбинацией. Оператор скрещивания (или рекомбинации) соответствует биологическому скрещиванию при половом размножении[8]. Он используется для комбинирования генетической информации двух индивидуумов, выступающих в роли родителей, в процессе порождения потомков (обычно двух). Как правило, оператор скрещивания применяется не всегда, а с некоторой (высокой) вероятностью. Если скрещивание не применяется, то копии обоих родителей переходят в следующее поколение без изменения. Позиция в хромосомах обоих родителей выбирается случайным образом. Эта позиция называется точкой скрещивания, или точкой разреза. Гены одной хромосомы, расположенные справа от этой точки, обмениваются с точно так же расположенными генами другой хромосомы. В результате мы получаем двух потомков, несущих генетическую информацию обоих родителей. На рисунке 9 показано одноточечное скрещивание пары двоичных хромосом, когда точка скрещивания находится между пятым и шестым геном.

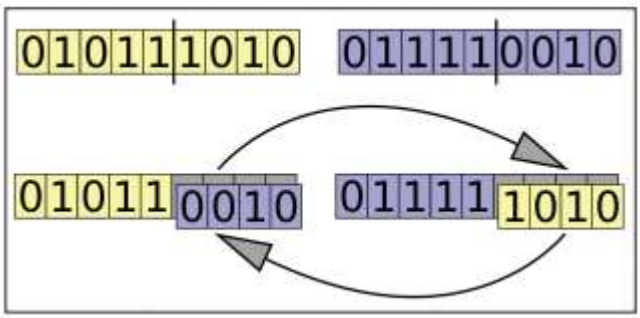

Рис.9 – Пример скрещивания

2.2.4 Метод мутации

Цель оператора мутации – периодически случайным образом обновлять популяцию, т. е. вносить новые сочетания генов в хромосомы, стимулируя тем самым поиск в неисследованных областях пространства решений[4]. Мутация может проявляться как случайное изменение гена. Мутации реализуются с помощью внесения случайных изменений в значения хромосом, например инвертирования одного бита в двоичной строке. Мутация – последний генетический оператор, применяемый при создании нового поколения[7]. Он применяется к потомку, созданному в результате операций отбора и скрещивания. Операция мутации вероятностная, обычно она выполняется изредка,с очень низкой вероятностью, поскольку может ухудшить качество индивидуума, к которому применена. В некоторых вариантах генетических алгоритмов вероятность мутации постепенно увеличивается, чтобы предотвратить стагнацию и повысить разнообразие популяции. С другой стороны,если частота мутации слишком велика, то генетический алгоритм выродится в случайный поиск.

При применении этого метода к двоичной или целочисленной хромосоме выбирается случайная последовательность генов, и порядок генов в ней меняется на противоположный, как показано на рисунке 10.

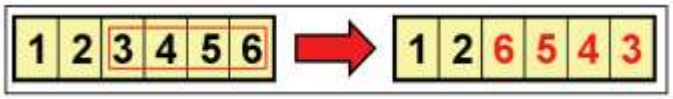

Рис.10 – Пример мутации

2.4 Генетические алгоритмы в нейронных сетях

Существует обширное множетсво архитектур нейронных сетей и их методов обучения.Самая популярная связка – многослойный персептрон в работе с алгоритмом обраного распространения ошибок.

Другим похожим методом обучения является использование генетических алгоритмов для поиска весов синаптических свзяей.

Опишем стркутуру генетических алгоритмов в совокупности с нейронными сетями:

1) В первую очередь определяем константы генетического алгоритма, в данной программе это :

   Размер популяции равный 100,

   Коэффициент мутации равный 0.1,

   Коэффициент скрещивания равный 0.8.

2)Затем заполняем популяцию особями.В данном случаем особями являются случайные связующие веса, созданные нейронной сетью с отрезком [-2.2].

3)Затем для каждой особи в популяции считается пригодность с помощью функции приспособленности.В данном случае функцией приспособленности служит функция потерь MSE.

4)Далее методом рулетки,когда после каждого запуска рулетки отбор индивидуума из популяции производится в точке отбора , затем рулетка запускается еще раз для выбора следующего индивидуума, и так до тех пор, пока не наберется достаточно индивидуумов для образования следующего поколения оценивается пригодность каждой особи.

5)Затем выбирается пара особей для скрещивания из особей отобранных методом рулетки , для этого каждой особи присваивается случайное десятичное значение и если это значение меньше коэффициента скрещивания, то особь отправляется для срещивания.

6) Затем производится скрещивание выбранный особей методом одноточечного скрещивание пары двоичных хромосом, когда точка скрещивания находится между пятым и шестым геном

7)Затем особи мутируют, это происходит следующим образом: каждой особи присваивается случайное десятичное число и если это число оказывается меньше заданного в начале коэффициента мутации , то особи присваивается новый вес интервале [-2,2].Это нужно для того, чтобы переодически случайным образом обновлять популяцию и исследовать области, которые были нераскрыты.

8)В заверешние алгоритма формируется новая популяция, состоящия из скрещенных и мутировавших особей.И эта новая популяция заменяет старую.

Так выглядит один цикл работы генетических алгоритмов.В случае реального обучения нейронных сетей определяется число эпох (N) и этот цикл повторяется N число раз.После этого выдает нейронной сети лучшее поколение, которое уже тестируется нейронной сетью и определяется точность.

9) Повторить 3-8 пункты N раз

2.5 Математическое представление функции приспособленности

На каждой итерации алгоритма индивидуумы оцениваются с помощью функции приспособленности (или целевой функции). Это функция, которую мы стремимся оптимизировать, или задача, которую пытаемся решить. Индивидуумы, для которых функция приспособленности дает наилучшую оценку, представляют лучшие решения и с большей вероятностью будут отобраны для воспроизводства и представлены в следующем поколении. Со временем качество решений повышается, значения функции приспособленности растут, а когда будет найдено удовлетворительное значение, процесс можно остановить.

$$\text{Fitness}=\sum_{i=0}^{N-1} \text{individ}[i] \qquad (4)$$

3. ПРОГРАММНЫЙ МОДУЛЬ ОБУЧЕНИЯ НЕЙРОННОЙ СЕТИ ГЕНЕТИЧЕСКИМ АЛГОРИТМОМ И РЕЗУЛЬТАТЫ ИССЛЕДОВАНИЙ

3.1 Среда разработки

Для разработки программы была выбран язык программирования Python.Разработка велась в интегрированной среде Microsoft Visual Studio Community 2019 Версия 16.8.5 в операционной системе Windows 10.

Выбор языка объясняется тем, что в нем находятся много бибилотек, облегчающий работу с нейронной сетью и генетическим алгоритмом.Например для нейронных сетей есть такие библиотека как keras(это библиотека глубокого обучения, представляющая из себя высокоуровневый API, написанный на Python и способный работать поверх TensorFlow, Theano или CNTK. Он был разработан с расчетом на быстрое обучение.),sklearn(самый распространенный выбор для решения задач классического машинного обучения. Она предоставляет широкий выбор алгоритмов обучения с учителем и без учителя.).А для генетического алгоритма присутствует библиотека deap(это новая эволюционная вычислительная платформа для быстрого прототипирования и тестирования идей. Она стремится сделать алгоритмы явными, а структуры данных-прозрачными. Он работает в полной гармонии с такими механизмами распараллеливания, как многопроцессорность).

3.2 Описание структуры нейронной сети

1)Из входных данных в наш класс с нейронной сетью постаупает информация о нейронроннах на каждом слое

2)Инициализируем слои

3)Создаем веса

4)Определяем функцию активации

5)Считаем потери

6)Суммируем потери

7)Передаем вес с одного слоя на другой

8) Считаем веса

9) Предсказываем верный результат и сравниваем с правильным

3.3 Набор данных для тестирования

Набор данных о цветках ириса или Набор данных Fisher's Iris - это многомерный набор данных , представленный британским статистиком , евгенистом и биологом. Рональд Фишер в своей статье 1936 г. Использование комбинации этих четырех характеристик, Фишер разработал линейную дискриминантную модель, позволяющую отличать виды друг от друга.

На основе линейной дискриминантной модели Фишера этот набор данных стал типичным тестовым примером для многих методов статистической классификации в машинном обучении такие как машины поддержки векторов .

Использование этого набора данных в кластерном анализе , однако, не является обычным явлением, поскольку набор данных содержит только два кластера с довольно очевидным разделением. Один из кластеров содержит Iris setosa, а другой кластер содержит как Iris virginica, так и Iris versicolor, и множественных измерений в таксономических задачах в качестве примера линейного дискриминантного анализа . Иногда его называют набором данных ириса Андерсона , потому что Эдгар Андерсон собрал данные для количественной оценки морфологической вариации цветков ириса трех родственных видов. . Два из трех видов были собраны на полуострове Гаспе «все с одного пастбища, собраны в один день и измерены в одно и то же время одним и тем же человеком с помощью одного и того же прибора». Статья Фишера была опубликована в журнале Annals of Eugenics , что вызвало споры о продолжающемся использовании набора данных Iris для обучения статистическим методам сегодня.

Набор данных состоит из 50 образцов каждого из трех видов ириса (Iris setosa , Iris virginica и Iris versicolor). Для каждого образца измеряли четыре элемента : длину и ширину чашелистиков и лепестков в сантиметрах. Основываясь на его невозможно разделить без информации о видах, которую

использовал Фишер. Это делает набор данных хорошим примером для объяснения разницы между контролируемыми и неконтролируемыми методами в интеллектуальном анализе данных : линейную дискриминантную модель Фишера можно получить, только если известны виды объектов: метки классов и кластеры не обязательно являются то же самое.

Тем не менее, все три вида Ириса отделимы в проекции на нелинейную и ветвящуюся главную составляющую. Набор данных аппроксимируется ближайшим деревом с некоторым штрафом за чрезмерное количество узлов, изгибов и растяжений. Затем строится так называемая «карта метро». Точки данных проецируются на ближайший узел. Для каждого узла составляется круговая диаграмма прогнозируемых точек. Площадь круговой диаграммы пропорциональна количеству проецируемых точек. Из диаграммы (слева) видно, что абсолютное большинство образцов разных видов ирисов принадлежат разным узлам. Лишь небольшая часть Iris-virginica смешана с Iris-versicolor (смешанные сине-зеленые узлы на диаграмме). Следовательно, три вида ириса (Iris setosa, Iris virginica и Iris versicolor) можно разделить с помощью неконтролирующих процедур нелинейного анализа главных компонентов . Чтобы различать их, достаточно просто выбрать соответствующие узлы на главном дереве.

Набор данных содержит набор из 150 записей по пяти атрибутам - длине чашелистика, ширине чашелистика, длине лепестка, ширине лепестка и виду.

3.4 Описание алгоритма работы программы

1)Считываем описанные библиотеки (random,nympy,time)

2)Инициализираем функции активиции tanh и sigmoid и их производные

3)Запускаем команду из библиотеки time для подсчета времени работы программы

Рис.11 – Iris setosa

Рис.12 – Iris versicolor

Рис.13 – Virginica

4)Передаем константы для генетического алгоритма и слоев

5) Инициализируем набор данных iris

6) Предварительно обрабатываем данные iris

7)Инициализуем слои и создаем веса ,определяем функцию активации и передаем в популяцию

8)Далее в генетическом алгоритме для каждой особи добавляем 1/сумму (MSE) индивидуума в список смещения

9)Считаем пригодность

10) Оцениваем пригодность каждой хромосомы методом рулетки

11) Выбираем двух родителей

12) Скрещиваем

13)Мутируем

14) Заменяем старую популяцию новой и сортируем в порядке убывания пригодности

15) Повторяем 9-14 пункты пока не достигнем заданного количества эпох

16)Заканчиваем подсчет времени

17)Тестируем нейронную сеть с наиболее приспособленными весами

18) Выводим результат

3.5 Резльтаты работы программы

В ходе работы было выяснено, что ключевыми параметрами для результата программы являются : размер популяции, число эпох и число нейронов на слоях.Далее я проводил исследования и подобрал оптимальные параметры.

После подбора оптиматльного числа нейронов и популяции оставалось только проводить исследования при ранзом числе эпох.Например, при заданных параметрах :

- Популяция - 100
- Число эпох - 30
- Число нейронов на слоях- 4,6,1

Проведем несколько тестов, чтобы отследить средний результат при заданных параметрах.Видно, что программа считает приблезительно 50 секунд , а точность ответа колеблится от 84% до 93%.Этой точности нам недостаточно, поэтому продолжаем тестирование.

При заданных параметрах:

- Популяция - 100
- Число эпох - 300
- Число нейронов на слоях- 4,6,1

После проведения множества тестов с этими условиями, становится видно, что программа считает ориентировочно 8 минут, тогда как точность уже имеет меньший разброс от 93% до 97%.Можно сделать вывод, что сеть стала более стабильной, но еще есть куда расти, поэтому попробуем подобрать более оптимальные параметры.

При заданных параметрах:

- Популяция - 100
- Число эпох - 1000
- Число нейронов на слоях- 4,6,1

Снова проводим множество тестов при этих условиях и видим,что точность стала в среднем 97% иногда удавалось добиться 100% времени для вычисления требовалось в среднем 27 минут, что не может не радовать. Проанализировав результаты тестирования при данных параметрах можно сделать выводы, что эти параметры могут являться одними из наилучших.Попробуем еще увеличить число эпох.

При заданных параметрах:

- Популяция - 100
- Число эпох - 3000
- Число нейронов на слоях- 4,6,1

Результатом работы программы стала средняя точность 97%.Время для вычисления составило в среднем 72 минуты.Из этого видно, точность немного уменьшилась, но все еще остается высокой.Для того,чтобы сделать точный вывод попробуем проведем тест при большем числе эпох.

При заданных параметрах:

- Популяция - 100
- Число эпох - 7000
- Число нейронов на слоях- 4,6,1

При данных параметрах программа выполнялась примерно 150 минут, со средней точностью 94-97%. Исходя из этого можно сделать вывод, что параметры завышены для наилучшего результата

Исходя из исследований можно сделать вывод, что оптимальное число эпох 1000-3000 , при некоторых тестах можно получить даже 100% процент предсказаний.При больших параматрах точность классификации снижается, но все равно остается высокой.

ЗАКЛЮЧЕНИЕ

Результатом данной работы является нейронная сеть оптимизированная генетическим алгоритмом.

В ходе тестов и исследований влияния различных параметров результат работы были выявлены наилучшие параметры.При подобранных параметрах, используя 11 нейронов ,размер популяции в 100 особей и 100 эпох удалось добиться точности определения 97-100%.С увеличением числа эпох точность возрастает.

Эту программу можно использовать для более крупных данных , в которых генетический алгоритм более наглядно покажет свои приемущества .

СПИСОК ЛИТЕРАТУРЫ

1. Генетические алгоритмы на Python / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 286 с.: ил. Стр. 165

2. Использование генетических алгоритмов для обучения нейронных сетей/Шумков Е.А. стр 78.

3. Применение генетических алгоритмов для обучения нейронных сетей / Федяев О.И., Соломка Ю.И. 2004 стр.56

4. Генетические алгоритмы / Панченко Т.В, Тарасевич Ю.Ю. 2007

5. Нейронные сети.Полный курс, 2-е издание/Пер. с англ. -М : Издательский дом “Вильямс”,2006. – 1104с. Хайкин С. Стр. 287

6. Моделирование генетических популяций с биохимическими свойствами/ Розенберг Р.С.., Мичиганский университет, 1967. Стр. 193

7. Многоцелевая оптимизация с использованием генетических алгоритмов: Учебное пособие. Надежность и безопасность системы/ Конак А., Койт Д. В., Смит А. Е., 2006 г. Стр. 312

8. Оптимизация множественных целей с использованием недоминирующей сортировки в генетических алгоритмах. Эволюционные вычисления/ Сринивас Н., Деб К.1994 г. Стр. 502

9. Эволюционные алгоритмы решения многоцелевых задач/ Коэльо К.А., Ламонт Г.Б., Ван Вельдхейзен Д.А. 2007 г. Стр. 325

10. Искусственные нейронные сети и приложения: учеб. пособие / Ф.М. Гафаров, А.Ф. Галимянов. – Казань: Изд-во Казан. ун-та, 2018. –121 с.

11. Pattern Recognition and Machine Learning (Information Science and Statistics)/ Christopher M. Bishop 2006 г. Стр. 234

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

Iris_dataset.py

```
import pandas as pd
import numpy as np

"""
 набор данных iris, всего 150 случаев, разделенных на три категории: iris-setosa, iris-
versicolor, iris-virginica.
"""


def read_data():
    IRIS_TRAIN_URL = 'iris_training.csv'
    IRIS_TEST_URL = 'iris_test.csv'

    names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'species']
    train = pd.read_csv(IRIS_TRAIN_URL, names=names, skiprows=1)
    test = pd.read_csv(IRIS_TEST_URL, names=names, skiprows=1)

    x_train = train.drop('species', axis=1)
    x_test = test.drop('species', axis=1)

    y_train = train.species
    y_test = test.species

    for i in range(0, 3):
        y_train = y_train.replace(i, i - 1)
        y_test = y_test.replace(i, i - 1)

    return x_train, x_test, y_train, y_test


def pre_processing(x_train, x_test, y_train, y_test):
    x_train_list = np.array(x_train).tolist()
    x_test_list = np.array(x_test).tolist()

    y_train_one_list = np.array(y_train).tolist()
    y_train_list = []
    for i in y_train_one_list:
        y_train_list.append([i])

    y_test_one_list = np.array(y_test).tolist()
    y_test_list = []
    for i in y_test_one_list:
        y_test_list.append([i])

    iris_train_data = []
    for item in list(zip(x_train_list, y_train_list)):
        iris_train_data.append(list(item))

    iris_test_data = []
    for item in list(zip(x_test_list, y_test_list)):
        iris_test_data.append(list(item))

    return iris_train_data, iris_test_data
```

main_gann.py

```
from operator import itemgetter
```

```
import random
import matplotlib.pyplot as plt
import numpy as np
from iris_dataset import read_data, pre_processing
import time


# Гиперболическая функция
def tanh(x):
    return np.tanh(x)


# Производная гиперболической функции
def tanh_derivate(x):
    return 1.0 - np.tanh(x) * np.tanh(x)


# Сигмовидная функция
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


# Производная сигмовидной функции
def sigmoid_derivate(x):
    return sigmoid(x) * (1 - sigmoid(x))



def calculate_fit(loss):
    total, fitnesses = sum(loss), []
    for i in range(len(loss)):
        fitnesses.append(loss[i] / total)
    return fitnesses


# берет совокупность объектов NetWork
def pair_pop(iris_data, pop):
    weights, loss = [], []

    # для каждой особи
    for individual_obj in pop:
        weights.append([individual_obj.weights_input, individual_obj.weights_output])
        # добавить 1 / сумму (MSE) индивидуума в список смещения
        loss.append(individual_obj.sum_loss(data=iris_data))

    # пригодность - это часть общей погрешности
    fitnesses = calculate_fit(loss)

    del pop

    # Вес становится элементом [0], а фитнес [1], таким образом, фитнес соединяется со своим
весом в кортеже.
    return zip(weights, loss, fitnesses)


def roulette(fitness_scores):
    """Оценка пригодности является частью, и ее сумма равна 1. У подходящих хромосом больше
очков.."""
    cumalative_fitness = 0.0
    r = random.random()
    # Оценка пригодности для каждой хромосомы
    for i in range(len(fitness_scores)):
        # Очки фитнеса добавляются для каждой хромосомы, чтобы повысить шансы
        cumalative_fitness += fitness_scores[i]
        # Колориметрический индекс возвращается, если совокупная пригодность больше r
```

```
            if cumalative_fitness > r:
                return i


    def iterate_pop(ranked_pop):
        ranked_weights = [item[0] for item in ranked_pop]
        fitness_scores = [item[-1] for item in ranked_pop]
        new_pop_weight = [eval(repr(x)) for x in ranked_weights[:int(pop_size * 0.15)]]

        # Воспроизведение двух случайно выбранных, но разных хромосом, пока не будет достигнуто
    значение pop_size.
        while len(new_pop_weight) <= pop_size:
            ch1, ch2 = [], []
            index1 = roulette(fitness_scores)
            index2 = roulette(fitness_scores)
            while index1 == index2:
                # Убедимся, что для разведения используются разные хромосомы.
                index2 = roulette(fitness_scores)
            # index1, index2 = 3,4
            ch1.extend(eval(repr(ranked_weights[index1])))
            ch2.extend(eval(repr(ranked_weights[index2])))
            if random.random() < crossover_rate:
                ch1, ch2 = crossover(ch1, ch2)
            mutate(ch1)
            mutate(ch2)
            new_pop_weight.append(ch1)
            new_pop_weight.append(ch2)
        return new_pop_weight


    def crossover(m1, m2):
        # ni*nh+nh*no  = общий вес
        r = random.randint(0, (nodes_input * nodes_hidden) + (nodes_hidden * nodes_output))
        output1 = [[[0.0] * nodes_hidden] * nodes_input, [[0.0] * nodes_output] * nodes_hidden]
        output2 = [[[0.0] * nodes_hidden] * nodes_input, [[0.0] * nodes_output] * nodes_hidden]
        for i in range(len(m1)):
            for j in range(len(m1[i])):
                for k in range(len(m1[i][j])):
                    if r >= 0:
                        output1[i][j][k] = m1[i][j][k]
                        output2[i][j][k] = m2[i][j][k]
                    elif r < 0:
                        output1[i][j][k] = m2[i][j][k]
                        output2[i][j][k] = m1[i][j][k]
                    r -= 1
        return output1, output2


    def mutate(m):
        # Можно включить константу, чтобы контролировать, насколько резко изменился вес.
        for i in range(len(m)):
            for j in range(len(m[i])):
                for k in range(len(m[i][j])):
                    if random.random() < mutation_rate:
                        m[i][j][k] = random.uniform(-2.0, 2.0)


    def rank_pop(new_pop_weight, pop):

        loss, copy = [], []
        pop = [NeuralNetwork(nodes_input, nodes_hidden, nodes_output) for _ in range(pop_size)]
        for i in range(pop_size):
            copy.append(new_pop_weight[i])

        for i in range(pop_size):
```

```
            # Каждому назначается вес, сгенерированный предыдущей итерацией.
            pop[i].assign_weights(new_pop_weight, i)
            pop[i].test_weights(new_pop_weight, i)

        for i in range(pop_size):
            pop[i].test_weights(new_pop_weight, i)

        # Рассчитаем пригодность этих весов и изменим их весами.
        paired_pop = pair_pop(iris_train_data, pop)

        # Вес отсортирован в порядке убывания пригодности (наиболее подходящий).
        ranked_pop = sorted(paired_pop, key=itemgetter(-1), reverse=True)
        loss = [eval(repr(x[1])) for x in ranked_pop]
        return ranked_pop, eval(repr(ranked_pop[0][1])), float(sum(loss)) / float(len(loss))


    def randomize_matrix(matrix, a, b):
        for i in range(len(matrix)):
            for j in range(len(matrix[0])):
                matrix[i][j] = random.uniform(a, b)


    class NeuralNetwork(object):
        def __init__(self, nodes_input, nodes_hidden, nodes_output, activation_fun='tanh'):
            # слои
            self.nodes_input = nodes_input
            self.nodes_hidden = nodes_hidden
            self.nodes_output = nodes_output


            self.activations_input = [1.0] * self.nodes_input
            self.activations_hidden = [1.0] * self.nodes_hidden
            self.activations_output = [1.0] * self.nodes_output

            # создавать веса
            self.weights_input = [[0.0] * self.nodes_hidden for _ in range(self.nodes_input)]
            self.weights_output = [[0.0] * self.nodes_output for _ in range(self.nodes_hidden)]
            randomize_matrix(self.weights_input, -0.1, 0.1)
            randomize_matrix(self.weights_output, -2.0, 2.0)

            # определить функцию активации
            if activation_fun is 'tanh':
                self.activation_fun = tanh
                self.activation_fun_deriv = tanh_derivate
            elif activation_fun is 'sigmoid':
                self.activation_fun = sigmoid
                self.activation_fun_deriv = sigmoid_derivate

        def sum_loss(self, data):
            loss = 0.0
            for item in data:
                inputs = item[0]
                targets = item[1]
                self.feed_forward(inputs)
                loss += self.calculate_loss(targets)
            inverr = 1.0 / loss
            return inverr

        def calculate_loss(self, targets):
            loss = 0.0
            for k in range(len(targets)):
                loss += 0.5 * (targets[k] - self.activations_output[k]) ** 2
            return loss

        def feed_forward(self, inputs):
```

```
        if len(inputs) != self.nodes_input:
            print('неправильное число слоев')

        for i in range(self.nodes_input):
            self.activations_input[i] = inputs[i]

        for j in range(self.nodes_hidden):
            self.activations_hidden[j] = self.activation_fun(
                sum([self.activations_input[i] * self.weights_input[i][j] for i in
range(self.nodes_input)]))
        for k in range(self.nodes_output):
            self.activations_output[k] = self.activation_fun(
                sum([self.activations_hidden[j] * self.weights_output[j][k] for j in
range(self.nodes_hidden)]))
        return self.activations_output

    def assign_weights(self, weights, I):
        io = 0
        for i in range(self.nodes_input):
            for j in range(self.nodes_hidden):
                self.weights_input[i][j] = weights[I][io][i][j]
        io = 1
        for j in range(self.nodes_hidden):
            for k in range(self.nodes_output):
                self.weights_output[j][k] = weights[I][io][j][k]

    def test_weights(self, weights, I):
        same = []
        io = 0
        for i in range(self.nodes_input):
            for j in range(self.nodes_hidden):
                if self.weights_input[i][j] != weights[I][io][i][j]:
                    same.append(('I', i, j, round(self.weights_input[i][j], 2),
round(weights[I][io][i][j], 2),
                                 round(self.weights_input[i][j] - weights[I][io][i][j], 2)))

        io = 1
        for j in range(self.nodes_hidden):
            for k in range(self.nodes_output):
                if self.weights_output[j][k] != weights[I][io][j][k]:
                    same.append((('O', j, k), round(self.weights_output[j][k], 2),
round(weights[I][io][j][k], 2),
                                 round(self.weights_output[j][k] - weights[I][io][j][k],
2)))
        if same:
            print(same)

    def test(self, data):
        results, targets = [], []
        for d in data:
            inputs = d[0]
            rounded = [round(i) for i in self.feed_forward(inputs)]
            if rounded == d[1]:
                result = '√ Верно '
            else:
                result = '× Ошибка'
            print('{0} {1} {2} {3} {4} {5} {6}'.format(
                'Входные данные:', d[0], '-->', str(self.feed_forward(inputs)).rjust(65),
'определен', d[1],
                result))
            results += self.feed_forward(inputs)
            targets += d[1]
        return results, targets
```

```
start = time.clock()

graphical_error_scale = 300
max_epoch = 30
pop_size = 100
mutation_rate = 0.1
crossover_rate = 0.8
nodes_input, nodes_hidden, nodes_output = 4, 6, 1
x_train, x_test, y_train, y_test = read_data()
iris_train_data, iris_test_data = pre_processing(x_train, x_test, y_train, y_test)

# Сортировать случайную совокупность в первый раз
pop = [NeuralNetwork(nodes_input, nodes_hidden, nodes_output) for i in range(pop_size)]

paired_pop = pair_pop(iris_train_data, pop)

ranked_pop = sorted(paired_pop, key=itemgetter(-1), reverse=True)


epoch = 0
tops, avgs = [], []

while epoch != max_epoch:


    new_pop_weight = iterate_pop(ranked_pop)
    ranked_pop, toperr, avgerr = rank_pop(new_pop_weight, pop)

    tops.append(toperr)
    avgs.append(avgerr)
    epoch += 1

end = time.clock()
print("генерации генетических общих затрат времени: " + str(end - start))

# протестировать NN с наиболее приспособленными весами
tester = NeuralNetwork(nodes_input, nodes_hidden, nodes_output)
fittestWeights = [x[0] for x in ranked_pop]
tester.assign_weights(fittestWeights, 0)
results, targets = tester.test(iris_test_data)
x = np.arange(0, 150)
#title2 = 'Test after ' + str(epoch) + ' epoch'
#plt.title(title2)
#plt.ylabel('Node output')
#plt.xlabel('Instances')
#plt.plot(results, 'xr', linewidth=0.5)
#plt.plot(targets, 's', color='black', linewidth=3)
#lines = plt.plot(results, 'sg')
#plt.annotate(s='Target Values', xy=(110, 0), color='black', family='sans-serif',
size='small')
#plt.annotate(s='Test Values', xy=(110, 0.5), color='red', family='sans-serif',
size='small', weight='bold')
#plt.show()

print('max_epoch', max_epoch, 'pop_size', pop_size, 'pop_size*0.15', int(
    pop_size * 0.15), 'mutation_rate', mutation_rate, 'crossover_rate', crossover_rate,
      'nodes_input, nodes_hidden, nodes_output', nodes_input, nodes_hidden, nodes_output)
```