

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

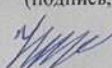
РЕАЛИЗАЦИЯ ГЕНЕТИЧЕСКИХ
АЛГОРИТМОВ И СПОСОБЫ УПРАВЛЕНИЯ
ГЕНЕТИЧЕСКИМИ ОПЕРАЦИЯМИ

Студент

 14.06.2024
(подпись, дата)

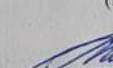
Холодков Данила Валерьевич
(ФИО полностью)

Руководитель

 14.06.2024
(подпись, дата)

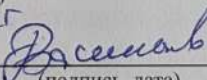
Слепцов Н.В.
(фамилия, инициалы)

Нормоконтролёр

 14.06.2024
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Рецензент

главн. специалист
АО «НПЗ Рубин»  14.06.2024
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 №11)

Заведующий кафедрой


(подпись)

Митрохин М.А.
(фамилия, инициалы)

Работа защищена с отметкой

отлично (протокол заседания ГЭК от 18.06.24 № 26)

Секретарь ГЭК


(подпись)

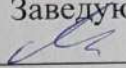
Е.И. Гурин
(фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
 М.А. Митрохин
«13» сентября 2022 г.

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
МАГИСТРА**

1. Студент Холодков Данила Валерьевич гр. 22ВВИм1 факультета вычислительной техники, направления 09.04.01

2. Тема работы «Реализация генетических алгоритмов и способы управления генетическими операциями».

Тема утверждена приказом ПГУ №765/о от "13" сентября 2022г.

3. Руководитель магистерской диссертации к.т.н., доцент Слепцов Н.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: анализ эффективности и конкурентоспособности генетических алгоритмов в качестве метода оптимизации функции ошибки в процессе обучения искусственной нейронной сети для решения задачи классификации, а также сравнительный анализ различных генетических операций.

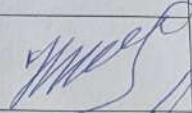
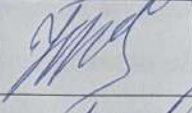
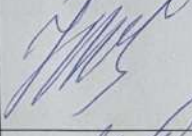
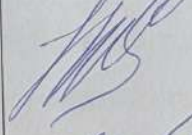
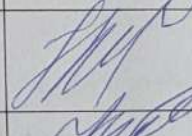
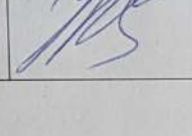
5. Перечень подлежащих разработке вопросов:

1. Анализ предметной области генетических алгоритмов и генетических операций, а также возможностей их применения в различных задачах.

2. Разработка программы реализующей обучение нейронной сети с использованием генетических алгоритмов в качестве метода оптимизации функции ошибки.

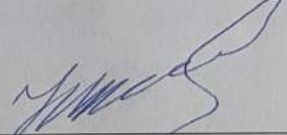
3. Анализ результатов запуска программы и сравнение их с результатами аналогичного оптимизационного метода – градиентным спуском.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области генетических алгоритмов	10%	13.09.2022-13.11.2022	
2.	Анализ предметной области машинного обучения	10%	14.11.2022-14.01.2023	
3.	Разработка программы реализующей обучение нейронной сети с помощью генетических алгоритмов	40%	15.01.2023-25.09.2023	
4.	Разработка программы реализующей обучение нейронной сети с помощью градиентного спуска	20%	26.09.2023-28.01.2024	
5.	Анализ результатов тестовых запусков	10%	29.01.2024-30.03.2024	
6.	Оформление выпускной работы	10%	31.03.2024-31.05.2024	

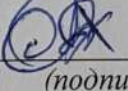
Дата выдачи задания "13" сентября 2022 г.

Руководитель магистерской диссертации


(подпись, дата)

Слепцов Н.В.

Задание к исполнению принял студент


(подпись, дата)

Холодков Д.В.

Работу к защите допустить
Декан факультета


(подпись, дата)

17.06.2024

Фионова Л.Р.

РЕФЕРАТ

Магистерская работа содержит 84 листа, 9 таблиц, 19 рисунков, 26 источников, 3 приложения

ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ, ГЕНЕТИЧЕСКИЕ ОПЕРАЦИИ, МЕТОДЫ ОПТИМИЗАЦИИ, НЕЙРОННЫЕ СЕТИ, МАШИННОЕ ОБУЧЕНИЕ

Цель работы – реализация генетических алгоритмов в качестве метода оптимизации функции ошибки в процессе обучения искусственной нейронной сети для решения задачи классификации, а также анализ и сравнение данного метода с аналогом.

Объектом работы является программа обучения искусственной нейронной сети с применением генетических алгоритмов в качестве метода оптимизации функции ошибки. Для объективного анализа результатов обучения с помощью генетических алгоритмов, была разработана аналогичная программа обучения для той же задачи, но с применением градиентного спуска в качестве метода оптимизации функции ошибки.

В результате проделанной работы были освоены основные понятия генетических алгоритмов и реализована программа обучения искусственной нейронной сети с генетическими алгоритмами в качестве метода оптимизации функции ошибки, что позволило протестировать различные генетические операции на практике.

Результаты данной работы могут быть применены в области машинного обучения для решения широкого круга задач, а также для дальнейших научных исследований в области генетических алгоритмов.

					ПГУ 1.090401.61.001 ПЗ			
Изм.	Лист	№ докум	Подпись	Дата	Реализация генетических алгоритмов и способы управления генетическими операциями	Лит.	Лист	Листов
Разраб.		Холодков Д.В.		14.06.24			2	84
Провер.		Слепцов Н.В.		16.08.24				
Н. Контр.		Бычков А.С.		19.08.24				
Утверд.		Митрохин М.А.						
						ФВТ, каф. ВТ гр. 22ВВИм1		

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ	6
1 Анализ предметной области	8
1.1 Генетические алгоритмы	8
1.2 Описание методов селекции в генетических алгоритмах	10
1.4.1 Метод рулетки	10
1.4.2 Метод стохастической универсальной выборки	12
1.4.3 Ранговый отбор	13
1.4.4 Турнирный отбор	14
1.4.5 Отбор устойчивого состояния	16
1.3 Описание методов скрещивания в генетическом алгоритме	17
1.5.1 Одноточечное скрещивание	17
1.5.2 Двухточечное скрещивание	18
1.5.3 Равномерное скрещивание	18
1.4 Описание методов мутации в генетическом алгоритме	19
1.6.1 Мутация инверсией	19
1.6.2 Мутация обменом	20
1.5 Градиентный спуск	20
1.6 Преимущества генетического алгоритма перед градиентным спуском	22
1.7 Выводы	24
2 Реализация генетических алгоритмов	25
2.1 Используемый язык программирования	25
2.1.1 Возможности реализации генетических алгоритмов на языке программирования C++	25
2.1.2 Возможности реализации генетических алгоритмов на языке программирования Python	26
2.1.3 Сравнительный анализ языков программирования	27
2.2 Используемые библиотеки	29
2.3 Используемая среда разработки	30
2.4 Описание обучающей выборки	31

2.5 Описание структуры нейронной сети	33
2.6 Описание программы генетического алгоритма	36
2.7 Описание программы градиентного спуска	41
2.8 Выводы	44
3 Анализ результатов	45
3.1 Результаты обучения нейронной сети с помощью генетических алгоритмов	45
3.2 Результаты обучения нейронной сети с помощью градиентного спуска	57
3.3 Выводы	59
ЗАКЛЮЧЕНИЕ	60
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	61
ПРИЛОЖЕНИЕ А Листинг программы	64
Приложение А.1 Файл Genetic_algorithm.py	65
Приложении А.2 Файл Gradient_Descent.py.	70
ПРИЛОЖЕНИЕ Б Презентация	76

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей работе применяют следующие сокращения и обозначения

ГА – Генетический алгоритм

ГС – Градиентный спуск

ИНС – Искусственная нейронная сеть

ФП – Функция приспособленности

FPS (Fitness Proportionate Selection) – Отбор пропорциональной приспособленности

CSV (Comma-Separated Values) – Текстовый формат файла, используемый для представления табличных данных

ReLU (Rectified Linear Unit) – Линейный выпрямитель, функция активации нейрона

ВВЕДЕНИЕ

Машинное обучение глубоко укоренилось в различных сферах жизни в последние годы. Развитие и распространение данный инструмент получил в ряде отдельных задач, не требующих абсолютной точности и имеющих низкую стоимость ошибки. Например, такие задачи как прогнозирование, классификация и консультация по каким-либо вопросам. В данном случае результат прогнозирования и консультирования можно расценивать как дополнительную рекомендацию в решении какого-либо вопроса, а не абсолютно верный ответ.

С ростом популярности машинного обучения остро встаёт вопрос оптимизации этого процесса, с целью получить большую эффективность за меньшее время обучения. Большое внимание в данном вопросе имеет метод оптимизации функции потерь. Наряду с градиентного спуском, стохастическим градиентным спуском, адаптивного момента, имитацией отжига, одним из таких методов является генетический алгоритм.

Выбор метода оптимизации функции потерь является очень важным этапом машинного обучения. Каждый метод имеет свои преимущества и недостатки из-за чего нет единого решения и в каждой отдельной задаче тот или иной метод проявляет себя по-разному. В том случае, если специалист решил воспользоваться генетическими алгоритмами в данном вопросе, тогда требуется настроить все параметры генетического алгоритма. Благодаря широкому набору параметров и различных методов генетические алгоритмы позволяют решать более широкий круг задач, чем аналогичные методы оптимизации функции.

В данной работе генетические алгоритмы будут применены в задаче обучения искусственной нейронной сети в качестве метода оптимизации функции потерь. Также будет выполнено сравнение эффективности обучения нейронной сети при использовании различных методов отбора, скрещивания и мутации генетического алгоритма. Выполнив обучение нейронной сети, будет составлена сводная таблица по результатам, позволяющая оценить

значение и влияние на результат каждого метода. Затем будет выполнено обучение идентичной нейронной сети с помощью градиентного спуска для сравнения результатов обучения нейронной сети с помощью генетического алгоритма с градиентным спуском в данной задаче.

1 Анализ предметной области

Первым этапом выполнения поставленных задач является исследование предметной области, а именно исследование основных понятий генетических алгоритмов и изучение методов отбора, скрещивания и мутации. Необходимо ознакомиться с аналогичным методом оптимизации функции потерь в сфере обучения нейронных сетей для проведения сравнения генетических алгоритмов с аналогичными методами.

1.1 Генетические алгоритмы

Генетические алгоритмы — это адаптивные методы поиска, которые в последнее время используются для решения задач оптимизации. В них используются как аналог механизма генетического наследования, так и аналог естественного отбора. При этом сохраняется биологическая терминология в упрощенном виде и основные понятия линейной алгебры [1]. Этот подход к решению задач оптимизации представляет собой разновидность эволюционных алгоритмов и основан на имитации отбора, скрещивания и мутации в популяции особей.

Генетические алгоритмы начинают свою работу с инициализации начальной популяции случайным образом. Каждая особь в популяции представляет собой потенциальное решение задачи оптимизации, представленное в виде набора параметров или генов. Затем осуществляется итерационный процесс, состоящий из нескольких шагов: селекция, кроссовер и мутация.

На этапе селекции особи из текущей популяции выбираются для участия в создании нового поколения в соответствии с их приспособленностью. Чем более приспособленная особь, тем выше вероятность ее выбора для включения в следующее поколение. Распределение вероятности выбора особи зависит от метода отбора. Включение в отбор особей с низкой приспособленностью повышает разнообразие популяции, что может привести к более высоким результатам, при этом увеличивается время схождения алгоритма.

На этапе кроссовера случайным образом выбираются пары особей, которые скрещиваются для создания потомства. В результате скрещивания происходит обмен частями генотипов между родительскими особями, что приводит к появлению новых комбинаций генов.

Шаг с мутацией предназначен для внесения случайных изменений в генотип каждой особи в популяции. Мутации помогают сохранить разнообразие генетического материала и избежать преждевременной сходимости к локальным оптимумам.

После завершения операторов селекции, кроссовера и мутации формируется новая популяция, которая заменяет предыдущее поколение. Процесс повторяется до тех пор, пока не будет достигнуто условие остановки, такое как достижение максимального числа поколений или сходимость к оптимальному решению. На рисунке 1 представлено графическое представление алгоритма.

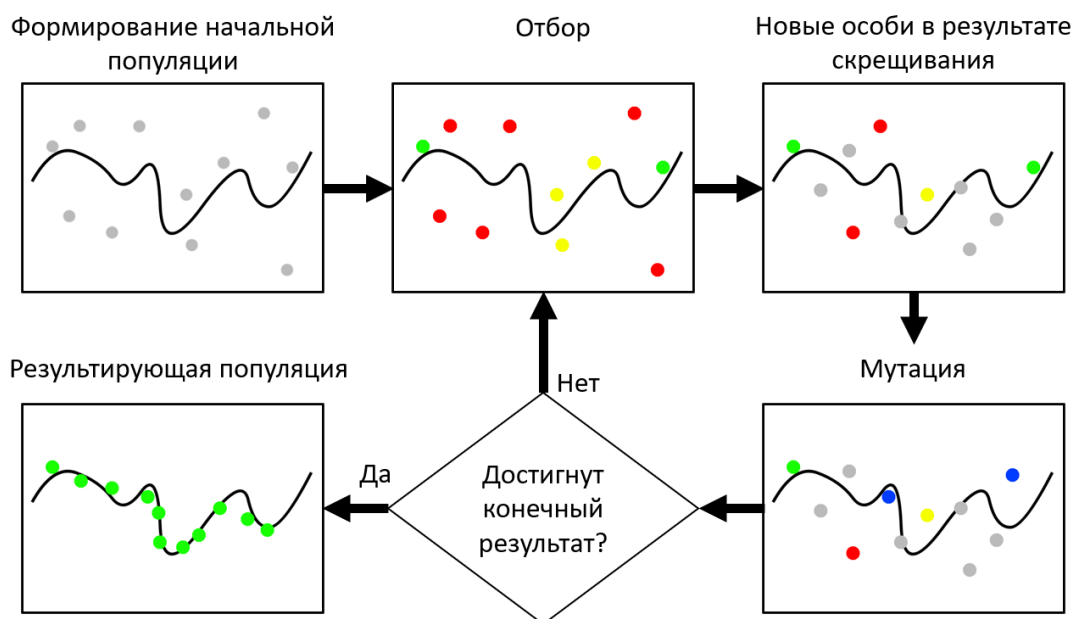


Рисунок 1 – Генетический алгоритм

Применение генетических алгоритмов в задачах машинного обучения весьма разнообразно. Они используются для решения задач оптимизации, таких как подбор параметров моделей, выбор признаков, кластеризация данных и обучение нейронных сетей. Примерами могут служить генетическое программирование для создания оптимальных алгоритмов,

эволюционные стратегии для обучения ИНС, а также генетические алгоритмы для решения задач классификации и регрессии. Их способность к обработке больших пространств параметров и гибкость делают их популярным выбором в контексте машинного обучения.

1.2 Описание методов селекции в генетических алгоритмах

Существуют различные методы селекции в генетических алгоритмах. Чаще всего применяются методы рулетки, турнира и ранговый метод. Есть множество методов, которые представляют собой либо модификации, либо комбинации этих трёх методов. Например, одной из таких модификаций является метод стохастической универсальной выборки, который является модификацией метода рулетки. Также существуют менее популярные методы селекции, такие как случайный отбор и отбор устойчивого состояния. Каждый метод проявляет себя в отдельных задачах по-разному, но можно их охарактеризовать таким понятием как скорость сходимости. Чем чаще в отборе принимают участие более приспособленные особи, тем быстрее происходит схождение к оптимумам. Если целью задачи является поиск большего количества экстремумов, следует применять методы, в которых вероятность выбора менее приспособленной особи повышена, а если цель быстрее найти решение, то больше подходят методы, в которых повышена частота выбора более приспособленных особей при отборе. Например, метод рулетки сходится быстрее, чем ранговый метод из-за того, что особи с большей приспособленностью чаще проходят отбор, однако последний имеет большую вероятность найти больше экстремумов, чем метод рулетки.

1.4.1 Метод рулетки

Метод отбора по правилу рулетки также называется отбором пропорциональной приспособленности или FPS (fitness proportionate selection). Суть метода заключается в случайном выборе особи в популяции, при этом вероятность выбора особи выражается отношением значения функции приспособленности этой особи к сумме значений функции приспособленности всех особей.

Весь процесс напоминает рулетку. В качестве наглядного примера рассмотрим отбор для выборки из таблицы 1.

Таблица 1 – Расчёт вероятности выбора при методе рулетки

Особь	Значение функции приспособленности	Вероятность выбора
А	30	13%
Б	5	2%
В	46	20%
Г	15	6%
Д	27	11%
Е	21	9%
Ж	13	6%
З	10	4%
И	28	12%
К	40	17%

Их можно представить на круговой диаграмме с секторами, размер доли которых соответствует соотношению значения функции приспособленности данной особи к сумме значений функций приспособленности всех особей популяции, графически данный процесс представлен на рисунке 2.

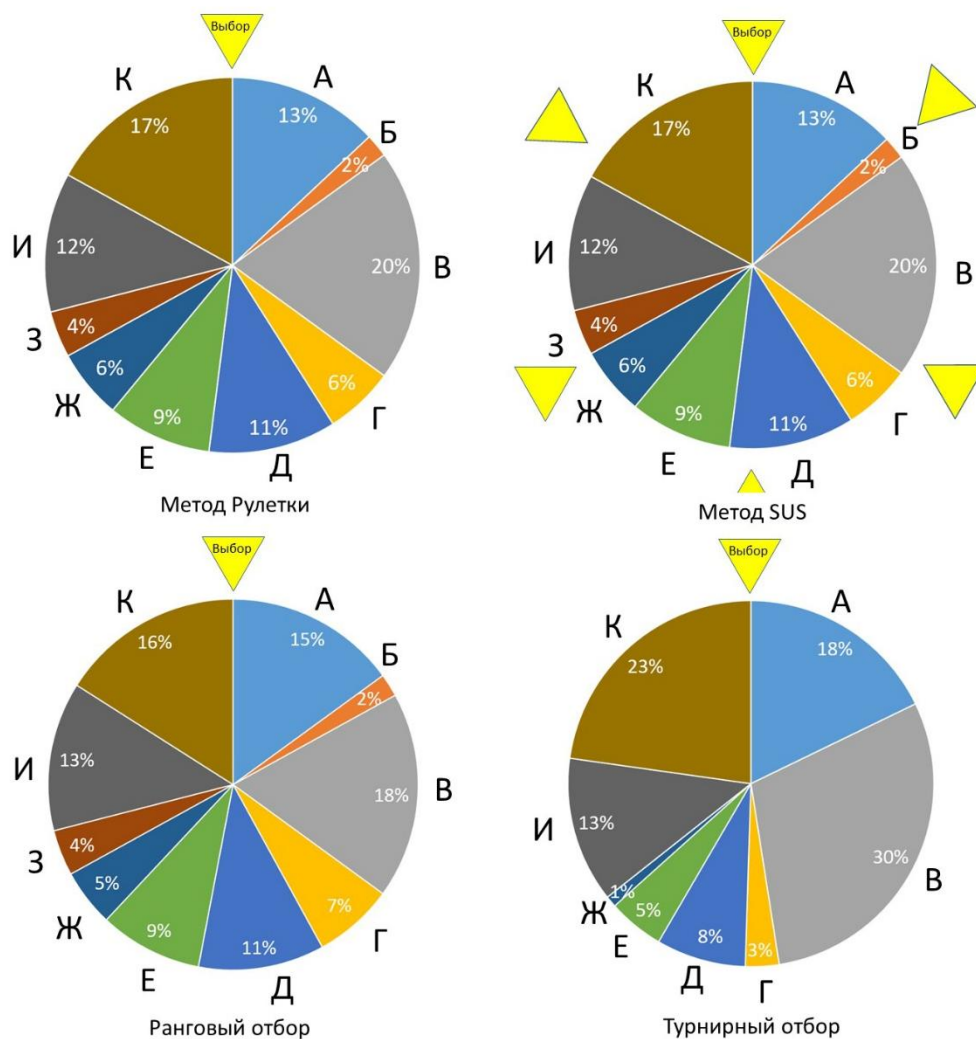


Рисунок 2 – Выбор в разных методах отбора

Следующим шагом мы случайным образом выбираем одну из особей. Таким образом, чаще будут отбираться особи с наибольшим значением функции приспособленности, так как у них больше сектор. При этом сохраняется вероятность выбора менее приспособленных особей, что сохраняет разнообразие популяции.

1.4.2 Метод стохастической универсальной выборки

Метод стохастической универсальной выборки или SUS (Stochastic universal sampling) – это усовершенствованный метод рулетки. Суть данного метода абсолютно идентична с методом рулетки, за исключением того, при использовании метода рулетки применяется одна точка отбора на колесе, а в методе стохастической выборки их несколько. При отборе, выбираются сразу несколько особей из разных участков колеса выбора. Колесо отбора

стохастической универсальной выборки для таблицы 1 представлено на рисунке 3.

Относительно метода рулетки, популяция при использовании метода стохастической универсальной выборки более разнообразна.

1.4.3 Ранговый отбор

Метод рангового отбора схож с методом рулетки. Отличием является способ распределения вероятности выбора особи. Если в методе рулетки вероятность отбора особи заключается в соотношении значения функции приспособленности данной особи к сумме значений функции приспособленности всех особей популяции, то в ранговом отборе вероятность выбора распределяется иначе. Все особи популяции сортируются по значению функции приспособленности в порядке возрастания. Таким образом наименее приспособленная особь получает индекс 1, а наиболее приспособленная особь получает последний индекс. Данные индексы считаются значением ранга особи. Таким образом, вероятность выбора особи рассчитывается как соотношение значения ранга данной особи к сумме рангов особей всей популяции.

Для примера реализации рангового метода отбора, рассчитаем вероятности выбора и размер доли на колесе выбора для популяции из таблицы 1. Расчёты вероятности выбора в ранговом отборе представлены в таблице 2.

Таблица 2 – Расчёт вероятности выбора в ранговом отборе

Особь	Значение функции приспособленности	Ранг	Вероятность выбора
А	30	8	15%
Б	5	1	2%
В	46	10	18%
Г	15	4	7%
Д	27	6	11%

Продолжение таблицы 2

Особь	Значение функции приспособленности	Ранг	Вероятность выбора
Е	21	5	9%
Ж	13	3	5%
З	10	2	4%
И	28	7	13%
К	40	9	16%

Если сравнить распределение вероятностей отбора особей в методе рулетки и в ранговом отборе, то можно заметить, что в ранговом отборе у менее приспособленных особей больше вероятность пройти отбор. Данная особенность обеспечивает большее разнообразие в популяции при ранговом отборе, чем в методе рулетки.

1.4.4 Турнирный отбор

В сути данного метода селекции лежит эффективная и простая идея. Сначала выбираются случайным образом кандидаты в турнире. Их должно быть от двух и более. Затем из них выбирается особь, имеющая наибольшее значение функции приспособленности. Выбранный индивид попадает в новую популяцию. Это повторяется до тех пор, пока число особей в новой популяции не сравняется с числом особей в исходной популяции.

Данный метод показывает высокую скорость выполнения и достаточную эффективность. Он не требует трудоёмких вычислений, потому что в алгоритме задействован только генератор случайных чисел и операция сравнения значения функции приспособленности. Из минусов следует отметить низкое разнообразие в популяции из-за сильного перевеса вероятности выбора в сторону более приспособленных особей в сравнении с ранее описанными методами. На рисунке 3 представлено графическое представление алгоритма турнирного отбора.



Рисунок 3 – Турнирный отбор

В качестве примера рассчитаем вероятность выбора особи при ранговом отборе для популяции из таблицы 1.

Если вероятность прохождения турнирного отбора это P , тогда $P = P_k * P_n$, где P_k – это вероятность для особи стать кандидатом в турнире, а P_n – это вероятность победы в турнире.

При 3 случайно выбираемых кандидатах для участия в турнире с общим количеством особей равным 10, вероятность участия в турнире P_k одинакова для каждой особи и равна 0.3.

P_n для особи можно вычислить как вероятность события, при котором два оставшихся кандидата будут иметь меньшее значение функции приспособленности. То есть для особи Г есть ещё 3 особи, у которых меньше значение функции приспособленности, а всего особей 9, если не считать саму особь Г, потому что она уже выбрана как кандидат турнира. В таком случае вероятность события, при котором два оставшихся кандидата в турнире будут иметь меньшее значение функции приспособленности, чем у особи Г, равняется 0.08333. Тогда P для особи Г равняется 0.03 или 3%. Это довольно простой и эффективный метод, учитывая отсутствие вычислений, однако если посчитать вероятность прохождения отбора для особей Б и З, то можно заметить, что у них нет шансов пройти отбор. Если Б и З имеют наименьшую приспособленность, то при выборе 3 кандидатов всегда будет третий

кандидат, у которого больше. Вероятности прохождения турнирного отбора для особей из таблицы 1 представлены в таблице 3.

Таблица 3 – Вероятность выбора особи в турнирном отборе

Особь	Значение функции приспособленности	Вероятность выбора
А	30	18%
Б	5	0%
В	46	30%
Г	15	3%
Д	27	8%
Е	21	5%
Ж	13	1%
З	10	0%
И	28	13%
К	40	23%

В турнирном отборе, как и в ранговом отборе, имеет значение не само значение ФП, а порядковый номер особи в отсортированном по возрастанию списке значений ФП для данной популяции. При этом алгоритм турнирного отбора подразумевает, что при двух кандидатах отбор никогда не пройдёт одна самая не приспособленная особь, при трёх кандидатах отбор никогда не пройдут две самые не приспособленные особи и т.д. Данная особенность приводит к меньшему разнообразию популяции и более скорому схождению алгоритма.

1.4.5 Отбор устойчивого состояния

Метод отбора устойчивого состояния или SSS (Steady state selection) менее распространён и часто уступает ранее упомянутым методам. Суть метода в том, что в каждом поколении отбирается несколько особей с высокой приспособленностью для создания нового потомства. Затем

некоторые особи с низкой приспособленностью удаляются и на их место помещается новое потомство. Остальная часть населения сохраняется в новом поколении.

1.3 Описание методов скрещивания в генетическом алгоритме

Скрещивание или кроссовер заключается в создании новых решений на основании отобранных родительских решений. Многие методы скрещивания позволяют создавать более чем одного потомка, но большое количество потомков от одной пары родителей приводит к созданию скудной своим разнообразием популяции, при этом возрастает время обработки потомства, что является доводом в пользу использования алгоритма скрещивания для создания одного-трех потомков.

Существует множество методов скрещивания. В данной работе будут исследованы следующие методы:

- одноточечное скрещивание,
- двухточечное скрещивание,
- равномерное скрещивание.

1.5.1 Одноточечное скрещивание

В данном случае речь идёт о классическом методе скрещивания генов для создания потомка. Суть метода заключается в случайном определении точки разделения родительских хромосом и создание потомка путём соединения части генов первой родительской особи до точки разделения и части генов после точки разделения от второй родительской особи. На рисунке 4 представлено графическое представление одноточечного скрещивания.

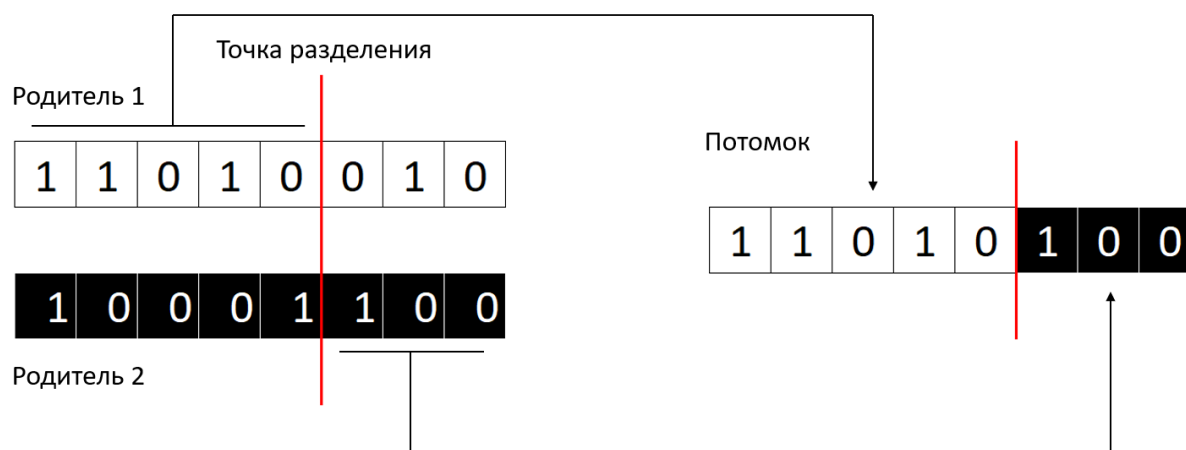


Рисунок 4 – Одноточечное скрещивание

1.5.2 Двухточечное скрещивание

Идея двухточечного скрещивания, как и любого k -точечного скрещивания заключается в повторении одноточечного скрещивания k раз. К примеру, при двухточечном скрещивании родительские гены делятся на три участка двумя случайными точками, а гены потомка получаются соединением первой части от первого родителя, второй части от второго родителя и третьей части снова от первого родителя.

Двухточечное скрещивание показывает лучшую эффективность по сравнению с одноточечным методом, однако дальнейшее увеличение количества точек не оказывают такого положительного результата как двухточечное скрещивание, но однозначно, что большее количество точек увеличивает пространство исследования генетического алгоритма.

1.5.3 Равномерное скрещивание

При равномерном скрещивании каждый ген потомка случайно берётся или от матери, или от отца. На практике генерируется маска генов, по которой составляются гены потомка. Если в маске стоит 0, тогда потомок получает ген от первого родителя, а если 1, то от второго. При использовании равномерного распределённого генератора случайных чисел мы можем ожидать равномерное перемешивание генов родителей при создании потомка. Пример реализации равномерного скрещивания изображён на рисунке 5.



Рисунок 5 – Равномерное скрещивание

Можно заметить, что данным методом можно сгенерировать большое количество потомков с уникальной хромосомой, сгенерировав множество масок. Однако, как уже было сказано ранее, потомки получаются генетически схожи, поэтому для разнообразия популяции эффективнее будет создание одного потомка от одной пары родителей.

1.4 Описание методов мутации в генетическом алгоритме

Этап мутации необходим для создания новых решений и расширения области исследования генетического алгоритма. В результате мутации появляется уникальная особь, которая не могла появиться в результате скрещивания особей текущей популяции, и, возможно, станет самой приспособленной особью в популяции.

Все методы мутаций заключаются в случайном изменении одного или нескольких генов, а это значит, что можно придумать сколько угодно собственных правил мутации и в решении каждой отдельной задачи они будут проявлять себя по-разному. В данной работе будут исследованы два метода мутации: мутация инверсией и мутация обменом.

1.6.1 Мутация инверсией

Самым стандартным методом мутации является инверсия, когда на этапе мутации, случайный бит хромосомы инвертируется. На рисунке 6 представлено графическое представление мутации инверсией.



Рисунок 6 – Мутация инверсией

Это самый простой вариант мутации, но при этом хорошо себя проявляющий в широком круге задач.

1.6.2 Мутация обменом

Метод обменом отличается от инверсии тем, что выбираются два гена хромосомы и их значения меняются местами. Данный метод мутации эффективнее мутации инверсией, когда ген кодирует порядок в каком-либо множестве, а не численное значение. На рисунке 7 представлено графическое представление мутации обменом.



Рисунок 7 – Мутация обменом

1.5 Градиентный спуск

Градиентный спуск (ГС) — это один из наиболее распространенных и эффективных методов оптимизации, который применяется в машинном обучении для нахождения минимума (или максимума) функции. Основная идея метода заключается в том, чтобы последовательно изменять параметры модели в направлении, противоположном градиенту функции потерь. Градиент функции потерь указывает направление наискорейшего возрастания функции, поэтому движение в противоположном направлении

позволяет приближаться к минимуму. На рисунке 8 представлено графическое представление градиентного спуска.

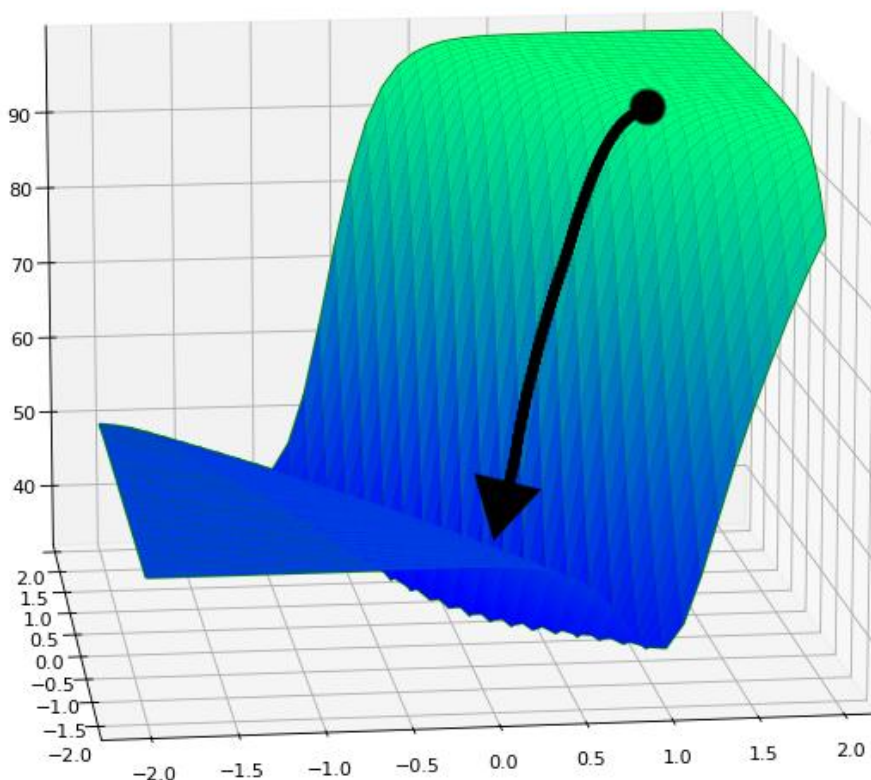


Рисунок 8 – Градиентный спуск

Процесс работы метода градиентного спуска можно представить следующим образом: начиная с некоторого начального значения параметров модели, на каждом шаге алгоритм вычисляет градиент функции потерь по отношению к этим параметрам. Затем параметры модели обновляются в направлении, обратном градиенту, с некоторым коэффициентом, называемым скоростью обучения. Этот процесс повторяется до достижения условия остановки, такого как достижение определенного значения функции потерь или выполнение заданного числа итераций.

Широкое использование градиентного спуска в задачах машинного обучения объясняется его простотой и эффективностью. Он применяется в обучении нейронных сетей, линейной и логистической регрессии, а также во многих других моделях машинного обучения. Градиентный спуск позволяет обучать модели на больших объемах данных и решать сложные задачи оптимизации.

Однако у метода градиентного спуска есть и недостатки. Он может столкнуться с проблемой затухающего градиента, особенно в глубоких нейронных сетях, что замедляет скорость обучения. Также выбор правильной скорости обучения может быть нетривиальной задачей и может потребовать настройки вручную. Кроме того, градиентный спуск может застрять в локальных минимумах функции потерь, не достигнув глобального минимума. Также стоит отметить важность точки инициализации. Результат алгоритма полностью зависит от точки инициализации, потому что после вычисления производной в данной точке происходит вычисление градиента и алгоритм сразу же движется в обратную сторону от возрастания производной. Таким образом алгоритм градиентного спуска всегда попадает в самый ближайший локальный минимум.

В целом, градиентный спуск остается одним из наиболее эффективных методов оптимизации в машинном обучении, но его применение требует внимательного анализа и настройки для достижения оптимальных результатов.

1.6 Преимущества генетического алгоритма перед градиентным спуском

Генетические алгоритмы представляют собой мощный инструмент в области машинного обучения, который обладает рядом преимуществ перед методом градиентного спуска, особенно в контексте определенных задач.

Генетические алгоритмы проявляют свою эффективность в решении задач оптимизации в сложных пространствах параметров. В то время как метод градиентного спуска может застрять в локальных оптимумах функции потерь, генетические алгоритмы способны находить глобальные оптимумы благодаря своей способности исследовать большие области поиска. Это особенно важно в задачах, где функция потерь имеет множество локальных минимумов или не является гладкой. Также стоит отметить о возможности распараллеливания вычислений в генетическом алгоритме, что является ещё одним достоинством перед градиентным спуском.

Кроме того, генетические алгоритмы показывают высокую эффективность при работе с не дифференцируемыми и мультимодальными функциями. ГА не требуют никакой информации о дифференцируемости и непрерывности функции [1]. В отличие от градиентного спуска, который требует гладкости и дифференцируемости функции потерь, генетические алгоритмы могут эффективно обрабатывать функции, не имеющие градиента или имеющие множество локальных экстремумов. Это делает их особенно полезными в задачах, связанных с оптимизацией сложных нелинейных моделей.

Например, генетические алгоритмы часто применяются в задачах подбора параметров моделей машинного обучения, где требуется настройка большого числа гиперпараметров для достижения оптимальной производительности модели. Они также успешно применяются в задачах, связанных с оптимизацией структуры нейронных сетей, выбором признаков и кластеризацией данных.

Таким образом, генетический алгоритмы представляет собой мощный и эффективный метод оптимизации в области машинного обучения, который позволяет решать сложные задачи оптимизации в больших пространствах параметров и обрабатывать не дифференцируемые и мультимодальные функции. Их способность находить глобальные оптимумы делает их привлекательным выбором для решения широкого спектра задач в машинном обучении.

Перспективы развития и дальнейшие направления исследований в области использования генетических алгоритмов в машинном обучении представляют собой интересную область для исследований. Дальнейшее углубление в изучение принципов работы генетических алгоритмов, разработка новых операторов и стратегий адаптации, а также исследование их применения в различных задачах машинного обучения поможет расширить возможности и повысить эффективность этого метода оптимизации.

1.7 Выводы

В рамках работ, выполненных в данной главе, был проведён анализ предметной области, выполнено исследование операторов отбора, скрещивания и мутации в генетических алгоритмах, сравнение генетического алгоритма с градиентным спуском в качестве метода оптимизации функций. Можно сделать вывод, что генетические алгоритмы представляют собой важный и перспективный инструмент в области машинного обучения, который может успешно применяться в различных задачах оптимизации и обладает потенциалом для дальнейшего развития и улучшения его эффективности.

2 Реализация генетических алгоритмов

Вторым этапом выполнения поставленных задач является реализация программы обучения нейронной сети с использованием генетических алгоритмов в качестве метода оптимизации функции потерь. Перед написанием непосредственно программы следует провести выбор инструментов для реализации генетических алгоритмов. Также необходимо подготовить обучающую выборку и определиться со структурой обучаемой нейронной сети.

2.1 Используемый язык программирования

Перед тем как приступить к реализации обучения нейронной сети с использованием генетических алгоритмов, требуется выбрать используемый язык программирования.

2.1.1 Возможности реализации генетических алгоритмов на языке программирования C++

Язык программирования C++ является высокоуровневым компилируемым языком программирования общего назначения со статической типизацией, который подходит для создания программ применяемых в широком круге задач. От разработки серверной части приложений и драйверов операционной системы, до обучения нейронных сетей можно найти реализации программ на данном языке. На сегодняшний день C++ является одним из самых популярных и распространённых языков.

Существует также множество крупномасштабных фреймворков для машинного обучения, реализованных на C++, включая Caffe, TensorFlow, OpenNN и другие, что подтверждает возможность применения языка в области машинного обучения.

Компилируемость и относительная низкоуровневость языка позволяет программам написанным на C++ выполняться очень быстро. Это крайне важное отличие от других языков, когда речь идёт о трудоёмких задачах, например глубоком обучении, обработке огромных баз данных или визуализации. В то же время компилируемость языка программирования

замедляет разработку и отладку. Для внесения небольшого изменения в программе, чтобы протестировать её, требуется компилировать код всей программы, поэтому для небольших проектов, не требующих высокой производительности, данный язык является не лучшим выбором.

Он имеет избыточный синтаксис, из-за чего программы иногда становятся сложно читаемыми. Исследование и дальнейшее использование требуемых постулатов C++ может отвлекать от задач и целей программы. К примеру, зачастую появляются трудности применения списков, для верного использования которых важны шаблоны и область видимости. Для работы с данным языком программирования необходимо всегда применять ручное выделение и освобождение памяти, от чего повышается риск ошибки [2]. Все эти минусы приводят к высокому порогу вхождения и сильно увеличивают время и стоимость разработки программ.

2.1.2 Возможности реализации генетических алгоритмов на языке программирования Python

Python — это высокоуровневый интерпретируемый язык программирования, отличающийся эффективностью, простотой и универсальностью использования. Он широко применяется в разработке серверной части приложений и сайтов, а также в машинном обучении и анализе данных. За счет простого и интуитивно понятного синтаксиса является частым выбором новичков, решивших изучить язык программирования.

Программы на языке Python медленнее выполняются относительно программ на таких языках как C++ или Java [3]. Данный факт ограничивает сферу применения программ на языке Python. В трудоёмких задачах, требующих высокой скорости выполнения, Python будет плохим выбором. Интерпретируемость языка позволяет выполнять программу построчно, что ускоряет разработку и отладку, потому что нет необходимости ждать компиляции всей программы. Это положительное качество высоко ценится в области анализа данных. При анализе данных, как правило, от программы

требуется отобразить графики и вычислить метрики, а выполнение каждой из этих задач происходит отдельно, что позволяет работать с отдельными блоками кода независимо.

Данный язык является лидером в сфере анализа данных. Его популярность привела к развитию и распространению таких библиотек как `numpy`, `pandas`, `scikit-learn`, `TensorFlow` и `PyTorch`, знание которых стало негласным стандартом множества специалистов в области анализа данных.

Сфера машинного обучения неразрывно связана с анализом данных. Использование Python и соответствующих библиотек в анализе данных привело к применению языка в сфере машинного обучения. Это породило ошибочное мнение о его лидерстве в том числе и в машинном обучении. Когда дело касается коммерческих приложений, выполняющих обработку большого объёма данных предприятия в реальном времени, либо глубокого обучения, Python уступает C++ и Java. Таким образом данный язык в сфере машинного обучения больше подходит для создания прототипов, небольших проектов и исследований.

2.1.3 Сравнительный анализ языков программирования

Сравнение языков программирования для разработки будет произведено по следующим критериям:

- скорость разработки,
- наличие документации к библиотекам для работы с генетическими алгоритмами.

В программе будут реализованы генетические алгоритмы и способы управления генетическими операциями для исследовательских целей, поэтому важна скорость разработки, а скорость выполнения программы не является важным критерием. Программы на C++ требуют много внимания к выделению памяти. При этом Python имеет простой синтаксис и не требует компиляции всего приложения, что позволяет работать с отдельными частями кода как с отдельной программой.

Из-за небольшой распространённости использования генетических алгоритмов, относительно градиентных методов оптимизации, количество библиотек ограничено. В особенности это касается библиотек с открытой документацией. Из доступных библиотек для языка C++ можно отметить openGA [4], GAlib [5]. Также можно отметить библиотеку для работы с ГА в системе Matlab.

Из документации по библиотеке openGA имеется описание в сети интернет [6] и книга Араша Мохамеда «OpenGA, a C++ Genetic Algorithm Library» [7] описывающую работу с данной библиотекой.

По библиотеке GAlib у разработчика имеется сайт с описанием работы всех функций [8]. Также была найдена русскоязычная статья, описывающая работу с библиотекой [9].

Для работы с генетическими алгоритмами на языке Python доступны библиотеки: EasyGA [10], DEAP [11], PyGAD [12].

К этим библиотекам имеются следующие доступные материалы. По работе с библиотекой EasyGA доступна документация от разработчиков [13] и один видеоролик по работе с библиотекой [14].

По работе с библиотекой DEAP разработчик составил подробную документацию по каждой функции с примерами [15].

Можно отметить библиотеку PyGAD для Python. Доступна подробная документация от разработчиков и множество примеров [16]. Также доступна статья автора библиотеки Ahmed Fawzy Gad «PyGAD: An Intuitive Genetic Algorithm Python Library», с подробным разбором [17].

В ходе сравнительного анализа языков программирования для разработки был выбран язык программирования Python. Первым фактором, повлиявшим на выбор, является то, что разработка программы на языке Python происходит быстрее благодаря простому синтаксису и интерпретируемости программного кода. Интерпретируемость языка позволит в процессе разработки запускать и тестировать код по частям, что крайне удобно при подборе параметров обучения нейронной сети и

генетического алгоритма, потому что не требуется компиляция всего кода, запускается только та часть кода, которая была изменена. При этом такой минус как относительно медленная скорость работы программы является несущественным. Вторым фактором выбора Python является наличие для языка программирования Python доступной подробной документация сразу по двум библиотекам, DEAP и PyGAD, что даёт возможность освоить и опробовать более широкий функционал используя обе библиотеки.

2.2 Используемые библиотеки

Для работы с генетическими алгоритмами будет использована библиотека PyGAD. Данная библиотека имеет подробную документацию [18] и широкий набор параметров для более точной настройки работы генетического алгоритма. В конструктор класса `pygad.GA` можно передать метод отбора, метод скрещивания и метод мутации, а также количество поколений и функцию приспособленности. В данной библиотеке имеется модуль `pygad.gann` [19], в котором реализовано обучение нейронной сети с помощью генетических алгоритмов. В конструктор класса `GANN` модуля `pygad.gann` передаются параметры нейронной сети, которую мы собираемся обучать, такие как количество слоёв, количество нейронов в слоях и функция активации каждого слоя. Данная библиотека поддерживает все необходимые функции, необходимые для обучения ИНС с использованием генетических алгоритмов.

Для чтения файла формата CSV с базой данных пациентов для обучения нейронной сети и дальнейшей работы с обучающей выборкой были использованы библиотеки `Pandas` [20] и `NumPy` [21].

Для нормализации значений обучающей выборки была использована функция `StandardScaler()`, а для разбиения выборки на обучающую и тестовую была использована функция `train_test_split()` библиотеки `scikit-learn` [22].

2.3 Используемая среда разработки

В качестве среды разработки было решено использовать Spyder. Spyder — это бесплатная среда разработки с открытым исходным кодом, написанная на языке программирования Python. На главном сайте Spyder указано, что среда была разработана учеными, инженерами и аналитиками данных, то есть специалистами, которые и будут использовать её. Данная среда имеет уникальные функции при работе с данными, гибкую настройку интерфейса и в среду встроена поддержка популярных библиотек для работы с данными NumPy, Pandas и Matplotlib [23]. В интерфейс среды встроено специальное окно для отображения графиков библиотеки Matplotlib. В данном окне сохраняются графики, выведенные с каждого запуска кода, что особенно удобно при тестировании запуска кода с разными параметрами, а к примеру в другой популярной среде разработки PyCharm графики отображаются отдельным окном, и для продолжения работы кода требуется закрыть его, то есть в среду не встроена возможность автоматического сохранения отображаемых графиков библиотеки Matplotlib. На рисунке 9 представлен интерфейс среды разработки Spyder.

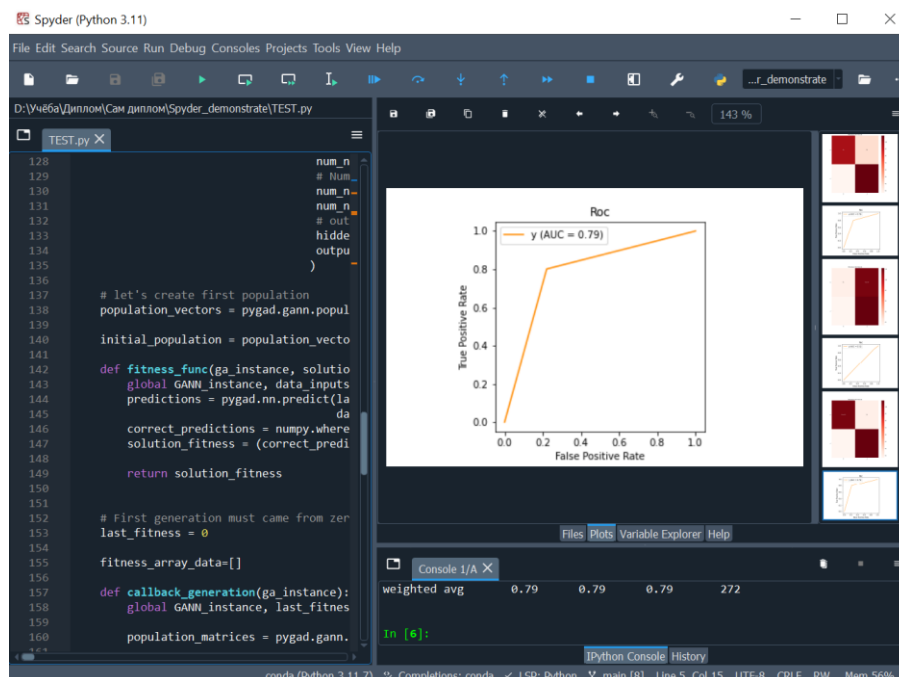


Рисунок 9 – Интерфейс используемой среды разработки Spyder

2.4 Описание обучающей выборки

Предметной областью, в которой было решено обучать нейронную сеть было выбрано прогнозирование диабета по показателям здоровья пациентов. Были найдены данные пациентов женского пола. Обучающая выборка содержит данные о 904 пациентах женского пола, обследованных на наличие диабета. В выборке присутствуют следующие поля:

- количество беременностей,
- уровень глюкозы,
- кровяное давление,
- толщина кожи,
- инсулин,
- индекс массы тела,
- DPF,
- возраст,
- результат.

Поле «DPF» содержит значение специальной функции, которая оценивает вероятность диабета на основе семейного анамнеза, а поле «Результат» содержит диагноз «Болен» если значение 1, и диагноз «Здоров» если значение 0.

На рисунке 10 представлены 5 экземпляров данных из данной выборки в качестве примера.

Беременностей	Глюкоза	Давление	Толщина_Кожи	Инсулин	ИМТ	DPF	Возраст	Результат
6.0	148.0	72.0	35.00	155.61	33.6	0.63	50.0	1.0
1.0	85.0	66.0	29.00	155.61	26.6	0.35	31.0	0.0
8.0	183.0	64.0	28.99	155.61	23.3	0.67	32.0	1.0
1.0	89.0	66.0	23.00	94.00	28.1	0.17	21.0	0.0
0.0	137.0	40.0	35.00	168.00	43.1	2.29	33.0	1.0

Рисунок 10 – Пример данных в выборке

Для каждого параметра были вычислены и отображены на рисунке 11 среднее значение, стандартное отклонение, минимальное значение, максимальное значение, а также 25, 50 и 75 процентиля.

	Беременностей	Глюкоза	Давление	Толщина_Кожи	Инсулин	ИМТ	DPF	Возраст	Результат
count	904.00	904.00	904.00	904.00	904.00	904.00	904.00	904.00	904.0
mean	4.08	125.49	72.84	29.31	159.66	32.84	0.50	33.81	0.5
std	3.37	31.35	11.66	8.05	85.15	6.54	0.33	11.36	0.5
min	0.00	44.00	24.00	7.00	14.00	18.20	0.08	21.00	0.0
25%	1.00	102.00	65.95	26.00	130.00	28.28	0.26	24.07	0.0
50%	3.00	121.58	72.36	28.99	155.61	32.65	0.41	31.00	0.5
75%	6.05	146.00	80.00	33.00	155.61	36.80	0.65	41.00	1.0
max	17.00	199.00	122.00	63.00	846.00	59.40	2.42	81.00	1.0

Рисунок 11 – Сводная информация о параметрах в выборке

Данная выборка была заранее обработана с целью устранения выбросов в данных или некорректных значений. В некоторых экземплярах данных встречались нулевые значения параметров, то есть пропуски. Все пропуски были заменены на среднее значение параметра, вычисленное по всей выборке.

Все параметры имеют численные значения, кроме параметра «Результат», потому что данный параметр кодирует категорию «Болен» как число 1 и «Здоров» как число 0. Все значения численных параметров выборки в процессе выполнения программы будут нормализованы, что требуется для наиболее качественного обучения нейронной сети. Если отказаться от нормализации значений, тогда параметры имеющие относительно крупные значения будут влиять на результат обучения больше, чем более мелкие значения. К примеру, из рисунка 11 видно, что в выборке инсулин в среднем имеет значение 159.66, а толщина кожи в среднем 29.31. Это означает, что из-за величины значимость параметра «Инсулин» на результат будет больше, чем у параметра «Толщина кожи». Многими исследователями была отмечена значимость нормализации данных для повышения точности алгоритмов машинного обучения [24].

Результатом вывода ИНС является значение от 0 до 1. Вывод ИНС будет сравниваться с полем «Результат» обучающей выборки, и корректироваться в соответствии с разницей между выводом и действительным значением выборки. После обучения, для приведения вывода ИНС к бинарному формату в соответствии с обучающей выборкой,

было решено, что если ИНС выводит значение больше 0.5, считать, что значение равно 1, а если меньше 0.5, то считать, что значение равно 0.

На основе данной выборки будет происходить обучение нейронной сети с помощью генетических алгоритмов.

2.5 Описание структуры нейронной сети

Для решения задач классификации и аппроксимации применимы нейронные сети прямого распространения. Главной моделью построения таких нейронных сетей является многослойный персептрон.

Данная модель широко используется в трудно формализуемых задачах, для которых не существует алгоритма нахождения решения. Нейронная сеть также не предполагает нахождение верного решения, а лишь даёт возможность находить приближённые решения в ряде сложных задач. В условиях низкой стоимости ошибки и отсутствии иных решений, данное решение, показывающее лишь вероятное предположение, является оптимальным. Многослойный персептрон зарекомендовал себя как универсальный аппроксиматор функций, с успехом использующийся в построении регрессионных моделей. В данной работе обучаемая нейронная сеть будет решать задачу бинарной классификации, а классификацию можно рассматривать как частный случай регрессии. Это означает, что многослойный персептрон подходит для выполнения данной работы.

Прежде чем обучить искусственную нейронную сеть с помощью генетических алгоритмов, следует определиться со структурой нейронной сети. Многослойный персептрон состоит минимум из трёх слоёв: входной, скрытый и выходной. При этом скрытых слоёв может быть сколько угодно.

Количество нейронов входного слоя напрямую зависит от размерности исходного пространства входных данных [25]. В нашей выборке 8 входных параметров, а это значит, что входной слой модели будет иметь 8 нейронов.

В задаче классификации количество нейронов выходного слоя зависит от метода кодирования тех классов, которым соответствует входной вектор параметров [25]. Так как в данной работе рассматривается задача бинарной

классификации, есть два оптимальных способа кодирования классов: двумя нейронами или одним нейроном. В случае использования двух нейронов, выходные значения каждого нейрона можно оценивать как вероятности соответствия входного вектора к тому или иному классу, тогда конечным результатом является наиболее вероятный класс. В случае использования одного нейрона в выходном слое, вывод нейрона можно оценивать как вероятность принадлежности к одному из классов. Для интерпретации результатов определяется граничное значение. Если вывод нейрона больше граничного значения, тогда определён один класс, а если меньше граничного значения, тогда определён другой класс.

В данной работе было принято решение использовать в выходном слое один нейрон, так как перед нами стоит задача бинарной классификации и выводом сети является «0» – здоров и «1» – болен, то нам достаточно вероятности принадлежности к одному из классов и нет потребности вычислять дополнительно вероятность принадлежности к другому классу. Стоит уточнить, что функцией активации выходного слоя будет логистическая функция – сигмоида. В таком случае выводом нейронной сети будет значение от 0 до 1. Граничным было решено задать значение 0.5. Все результаты нейронной сети, которые меньше значения 0.5, соответствуют классу «Здоров», а те, что больше или равны значению 0.5, соответствуют классу «Болен».

Больше всего проблем возникает на этапе выбора количества скрытых слоёв искусственной нейронной сети и количества нейронов в них. До сих пор нет однозначных правил определения этих параметров. Существуют различные методы подбора этих параметров, которые в большинстве своём основаны на оценке тестовых запусков обучения с использованием меньшей исходной выборки для экономии времени. Из работы Загинайло М.В. «Влияние количества нейронов искусственной нейронной сети на эффективность её работы» известно, что если количество скрытых слоёв и нейронов в них недостаточно, то сеть не обучается. При этом чрезмерно

большое количество скрытых слоёв и нейронов в них не только сильно замедляет процесс обучения, но и приводит к переобучению и явлению, в результате которого значения весов в результате коррекции становятся очень большими, а производная сжимающей функции слишком мала, из-за чего обучение парализуется не достигнув цели [26].

В результате тестовых запусков обучения нейронной сети с различным количеством скрытых слоёв и нейронов в них была определена оптимальная структура нейронной сети: 8 нейронов во входном слое, один скрытый слой с 12 нейронами и 1 нейрон в выходном слое. Данную структуру можно описать как 8-12-1. Предварительные тестовые запуски программы обучения показали эффективность данной структуры перед такими вариантами как 8-16-1, 8-8-16-1 и 8-4-1.

Для скрытого слоя в качестве функции активации была использована функция ReLU, а для выходного слоя была использована сигмоидная функция. Уравнения данных функций активации представлены на рисунке 12.

ReLU	Сигмоидная функция активации
$f(x) = \begin{cases} 0, & \text{если } x < 0 \\ x, & \text{если } x \geq 0 \end{cases}$	$f(x) = \frac{1}{1 + e^{-x}}$

Рисунок 12 – Функции активации

Графическое изображение конечной структуры нейронной сети представлено на рисунке 13.

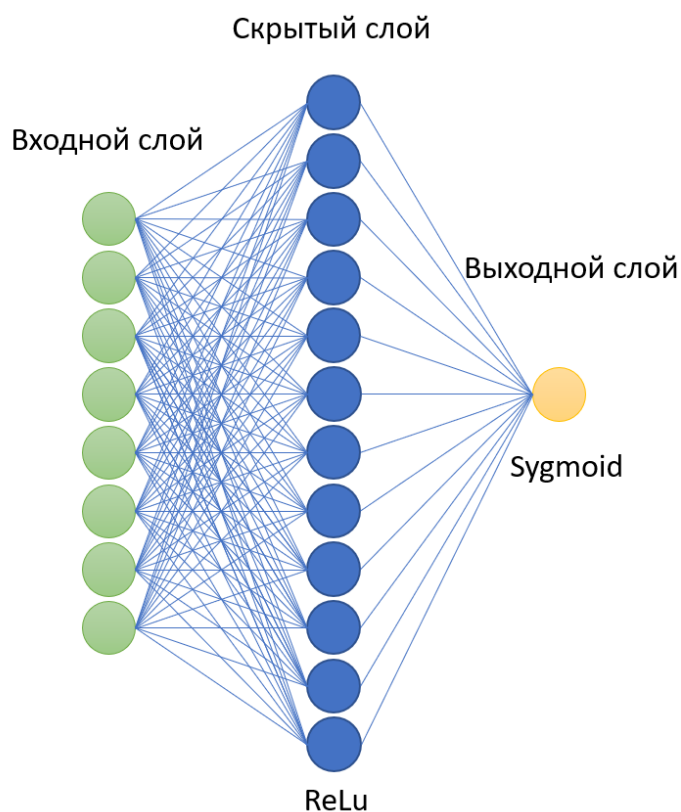


Рисунок 13 – Используемая структура нейронной сети

2.6 Описание программы генетического алгоритма

Код программы начинается с импортирования необходимых библиотек. Используемые библиотеки были описаны в пункте 2.2. Далее происходит подготовка выборки. Сначала выполнено чтение данных из файла, имеющего формат CSV, а затем разбиение выборки на обучающую и тестовую в соотношении 7:3 соответственно. Затем выполняется нормализация данных для приведения численных значений к единому виду. Нормализация была выполнена с помощью `StandardScaler()` библиотеки `scikit-learn`.

После подготовки данных для обучения, выполняется блок инициализации всех необходимых переменных, которые будут использоваться при обучении нейронной сети с помощью библиотеки `PyGAD`. В процессе обучения нейронной сети с помощью генетических алгоритмов, требуется определиться с тем, что из себя будут представлять особи популяции и какую информацию будут кодировать гены. Перед генетическими алгоритмами стоит задача оптимизации функции ошибки

нейронной сети, значит гены будут кодировать веса нейронной сети, а особью будет являться набор весов отдельной нейросети. Таким образом процессе отбора, скрещивания и мутации будут порождаться новые варианты весов нейросети (особи), которые, возможно, будут выдавать лучшие результаты.

В соответствии с документацией библиотеки PyGAD, сначала требуется создать начальную популяцию [18]. Наша популяция представляет из себя множество нейросетей, точнее весов нейросетей. В библиотеке PyGAD имеется специальный модуль для создания популяции нейросетей, который называется `pygad.gann`, где «gann» означает «genetic algorithms neural networks». Это специальный модуль для применения ГА в нейронных сетях, в конструктор класса которого передаются количество особей в популяции и параметры нейронной сети. В качестве параметров нейронной сети передаются такие параметры как функции активации каждого слоя, количество нейронов входного и выходного слоя, а также количество скрытых слоёв и нейронов в них. На основе данных параметров инициализируется популяция с помощью модуля `pygad.gann`:

```
# Инициализация популяции
GANN_instance = pygad.gann.GANN(
    num_solutions=100,          # Кол-во особей
    num_neurons_input=data_inputs.shape[1], # Кол-во нейронов
    входного слоя
    num_neurons_hidden_layers=[12], # Кол-во скрытых слоёв и
    нейронов в них
    num_neurons_output=2, # Кол-во выходных нейронов
    hidden_activations=["relu"], # Функция активации скрытого
    слоя
    output_activation="sigmoid" # Функция активации выходного
    слоя
)
```

После инициализации популяции требуется создать функцию приспособленности, с помощью которой особь будет оцениваться. Так как

мы оптимизируем результаты вывода нейронной сети, то очевидным решением в оценке эффективности будет пробный запуск нейронной сети на обучающей выборке с весами, которые были закодированы в генах особи. Таким образом значением функции приспособленности для особи будет процент правильных предсказанных ответов нейронной сети использующей веса данной особи. Функция приспособленности:

```
def fitness_func(ga_instance, solution, sol_idx):
    global GANN_instance, data_inputs, data_outputs
    pred = pygad.nn.predict(last_layer =
GANN_instance.population_networks[sol_idx], _inputs =
data_inputs)
    correct_pred = np.where(pred == data_outputs)[0].size
    fitness = (correct_pred/data_outputs.size)*100
    return fitness
```

В реализации генетических алгоритмов библиотеки PyGAD после каждого поколения требуется обновить веса в популяции нейронных сетей, для чего создаётся отдельная функция:

```
# Функция для промежуточных вычислений после каждого
поколения
def update_population(ga_instance):
    global GANN_instance, last_population_fitness
    population_vector_to_matrices =
pygad.gann.population_as_matrices(population_networks=
GANN_instance.population_networks,
    population_vectors = ga_instance.population)
    GANN_instance.update_population_trained_weights(
population_trained_weights = population_vector_to_matrices)
    fitness_array_data.append(
ga_instance.best_solution()[1].copy())
    last_population_fitness =
ga_instance.best_solution()[1].copy()
```

Ключевым классом библиотеки PyGAD является класс GA. В нём реализованы генетические алгоритмы. Для того чтобы воспользоваться им, следует передать в конструктор класса все необходимые параметры:

```
num_generations = 200 # Кол-во поколений
num_parents_mating = 2 # Кол-во родительских особей
участвующих в скрещивании
K_tournament=3 # Кол-во участников турнира в турнирном
отборе
parent_selection_type="tournament"
crossover_type="uniform"
mutation_type="swap"
ga_instance = pygad.GA(
    num_generations=num_generations,
    num_parents_mating=num_parents_mating,
    initial_population=initial_population,
    fitness_func=fitness_func,
    K_tournament=K_tournament,
    parent_selection_type=parent_selection_type,
    crossover_type=crossover_type,
    mutation_type=mutation_type,
    on_generation=callback_generation
)
```

Конструктор класса принимает в качестве аргументов следующие параметры:

- num_generations – количество поколений,
- num_parents_mating – количество родительских особей участвующих в скрещивании,
- initial_population – начальная популяция,
- fitness_func – функция оценки приспособленности,
- K_tournament – количество особей, выбираемых в качестве кандидатов в турнирном отборе,
- parent_selection_type – метод отбора,
- crossover_type – метод скрещивания,

- `mutation_type` – метод мутации,
- `on_generation` – функция, выполняемая после каждого поколения.

Обозначения методов отбора передаваемых в конструктор класса `pygad.GA`:

- `"tournament"` – турнирный отбор
- `"rws"` – метод рулетки,
- `"sus"` – метод стохастической универсальной выборки,
- `"rank"` – ранговый отбор,
- `"sss"` – отбор устойчивых состояний.

Обозначения методов скрещивания передаваемых в конструктор класса `pygad.GA`:

- `"single_point"` – одноточечное скрещивание,
- `"two_points"` – двухточечное скрещивание,
- `"uniform"` – равномерное скрещивание.

Обозначения методов мутации передаваемых в конструктор класса `pygad.GA`:

- `"inversion"` – мутация инверсией,
- `"swap"` – мутация обменом.

Запуск обучения нейронной сети выполняется короткой командой `ga_instance.run()`. В процессе обучения создаются 100 особей с различными генами. В генах закодированы веса для нейронной сети. Таким образом после обучения мы имеем 100 различных нейронных сетей. Конечным результатом принимается одна из особей популяции, как правило особь с наибольшим значением функции приспособленности.

После обучения выполняется проверка нейронной сети на тестовой выборке. Происходит вычисление результатов и запись их в файлы. Для удобства дальнейшей обработки результатов всех 30 тестов, было решено организовать вывод не в консоль, а в файлы чтобы после запуска была возможность обработать результаты и нарисовать графики используя сторонние программы. Для анализа результатов была собрана информация о

времени обучения, общем количестве верных указанных ответов и номер поколения, в котором был достигнут максимальный результат. Также для анализа результатов была вычислена матрица неточности, позволяющая оценить количество верных положительных ответов и верных отрицательных ответов нейронной сети. Это позволит узнать, насколько хорошо нейронная сеть способна определить тот или иной класс. Пример записи данных в файл:

```
# Запись в файл времени обучения в секундах и минутах
File_time = open(f'{COUNT_TEST}_T_{name_test}.txt', 'w')
File_time.write(f'{time_learning_sec}\n{time_learning_min}\n{time_learning_min_sec}')
File_time.close()
```

Листинг всей программы представлен в Приложении А.1 Файл Genetic_algorithm.py.

2.7 Описание программы градиентного спуска

От генетических алгоритмов градиентный спуск отличается большей скоростью нахождения экстремума и отсутствием какого-либо исследования функции, что приводит к застреванию в локальных минимумах. Градиентный спуск от точки инициализации сразу же направляется в сторону минимума значения функции ошибки. Однако данный метод слишком сильно зависит от точки инициализации, так как в зависимости от точки инициализации, градиентный спуск может попасть в локальный оптимум. В задачах с большим количеством локальных оптимумов применение градиентного спуска требует многочисленного повторного запуска в разных точках, для того чтобы найти глобальный оптимум или хотя бы более подходящий локальный оптимум.

В текущей работе будет изучена эффективность работы генетических алгоритмов в сравнении с градиентным спуском в области обучения нейронной сети для решения задачи бинарной классификации. Для того чтобы оценить эффективность генетических алгоритмов была написана программа обучения нейронной сети с идентичной структурой (8 нейронов в

входном слое, 12 нейронов в 1 скрытом слое, 1 нейрон в выходном слое) с использованием градиентного спуска для оптимизации функции ошибки.

В разработке программы градиентного спуска не применялись сторонние библиотеки и весь алгоритм был написан самостоятельно. Для этого был создан класс MLGS, в методы которого добавлены все необходимые функции активации, прямого прохода, обратного прохода, обучения, а также производные функция активации. Для создания экземпляра класса, конструктор класса MLGS принимает в качестве параметров количество нейронов во входном, скрытом и выходном слоях.

В качестве функции активации, как и в тестах генетических алгоритмов, использовалась функция ReLU для скрытого слоя и сигмоидная функция для выходного слоя:

```
# Функция сигмоиды
def sigmoid(self, x):
    return 1 / (1 + np.exp(-x))

# Функция ReLU
def relu(self, x):
    a=[]
    for i in x:
        b=[]
        for j in i:
            if j<0:
                b.append(0)
            else:
                b.append(j)
        a.append(b)
    return np.array(a)
```

В процессе вычисления градиента слоя используются производные функций активации, поэтому для сигмоидной функции и функции ReLU в класс MLGS были добавлены методы вычисления производных от функций:

```
# Функция вычисления производной от сигмоиды
def derivative_sigmoid(self, x):
    return self.sigmoid(x) * (1 - self.sigmoid(x))
```

```

# Функция вычисления производной от ReLU
def derivative_relu(self,x):
    a=[]
    for i in x:
        b=[]
        for j in i:
            if j<0:
                b.append(0)
            else:
                b.append(1)
        a.append(b)
    return np.array(a)

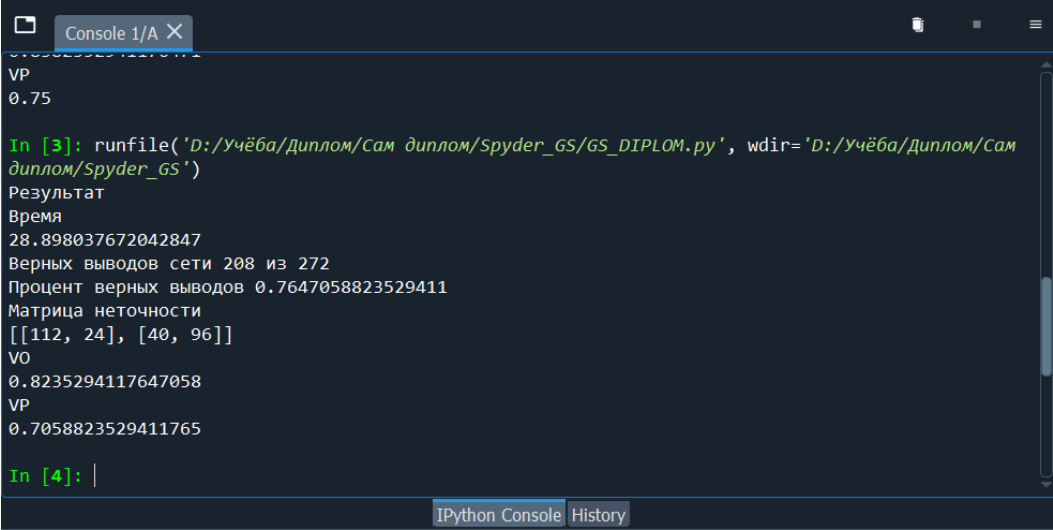
```

Функция прямого прохода выполняет перемножение весов на входные сигналы слоя и передаёт результаты в функцию активации слоя, а функция обратного прохода содержит в себе вычисления ошибки и градиента слоя с дальнейшим пробросом ошибки на предыдущий слой.

Сам процесс обучения состоит из выполнения прямого и обратного прохода последовательно, где за количество повторений данного процесса отвечает количество итераций градиентного спуска. В текущей работе будет выполнено 6 запусков программы с количеством итераций равным 100, 500, 1000, 5000, 10000 и 50000. Ещё одним важным параметром является шаг алгоритма. После всех вычислений требуется обновить веса нейронной сети, и величина, на которую изменятся веса за одну итерацию градиентного спуска, зависит от шага. В данной работе шагом градиентного спуска было задано число 0.0001. Листинг всей программы представлен в Приложении А.2 Файл Gradient_Descent.py.

После обучения следует тестирование обученной нейронной сети на тестовой выборке и дальнейший анализ результатов. Если в процессе выполнения тестов генетических алгоритмов результаты записывались в файл, чтобы потом обработать в сторонних программах, то в данном случае из-за небольшого количества тестов, было принято решение выводить результаты в консоль. Для каждого запуска программы вычислялись

следующие показатели: процент верных выводов, время обучения, процент верных положительных выводов и процент верных отрицательных выводов. Пример вывода программы градиентного спуска представлен на рисунке 14.



```
VP
0.75

In [3]: runfile('D:/Учёба/Диплом/Сам диплом/Spyder_GS/GS_DIPLOM.py', wdir='D:/Учёба/Диплом/Сам диплом/Spyder_GS')
Результат
Время
28.898037672042847
Верных выводов сети 208 из 272
Процент верных выводов 0.7647058823529411
Матрица неточности
[[112, 24], [40, 96]]
VO
0.8235294117647058
VP
0.7058823529411765

In [4]: |
```

Рисунок 14 – Пример вывода в консоль

2.8 Выводы

В рамках текущей главы был проведён сравнительный анализ языков программирования, также были выбраны и другие инструменты для реализации генетических алгоритмов такие как среда разработки и используемые библиотеки. Затем, в качестве подготовки к реализации генетических алгоритмов, была изучена и подготовлена обучающая выборка и выбрана структура нейронной сети. В результате была разработана программа обучения нейронной сети с использованием генетических алгоритмов в качестве метода оптимизации функции потерь. Для более объективного анализа результатов полученных в тестах генетических алгоритмов была также разработана программа обучения нейронной сети с использованием градиентного спуска в качестве другого метода оптимизации функции потерь. Результаты использования иного метода позволят провести сравнение методов.

3 Анализ результатов

Завершающим этапом выполнения поставленных задач является анализ данных полученных в результате запуска программ обучения нейронной сети с использованием в качестве метода оптимизации функции потерь генетические алгоритмы и градиентного спуска.

3.1 Результаты обучения нейронной сети с помощью генетических алгоритмов

Всего выполнено 30 запусков программы с каждой комбинацией методов отбора, скрещивания и мутации. Каждый тест со своей комбинацией методов имеет собственный номер в соответствии с таблицей 4.

В таблице использованы следующие сокращения:

- ТО – турнирный отбор,
- МР – метод рулетки,
- МСУВ – метод стохастической универсальной выборки,
- РО – ранговый отбор,
- ОУС – отбор устойчивого состояния,
- ОС – одноточечное скрещивание,
- ДС – двухточечное скрещивание,
- РС – равномерное скрещивание,
- МИ – мутация инверсией,
- МО – мутация обменом.

Таблица 4 – Соответствие номера теста с используемыми методами.

Номер теста	Отбор	Скрещивание	Мутация
1	ТО	ОС	МИ
2	ТО	ОС	МО
3	ТО	ДС	МИ
4	ТО	ДС	МО
5	ТО	РС	МИ

Продолжение таблицы 4

Номер теста	Отбор	Скращивание	Мутация
6	ТО	РС	МО
7	МР	ОС	МИ
8	МР	ОС	МО
9	МР	ДС	МИ
10	МР	ДС	МО
11	МР	РС	МИ
12	МР	РС	МО
13	МСУВ	ОС	МИ
14	МСУВ	ОС	МО
15	МСУВ	ДС	МИ
16	МСУВ	ДС	МО
17	МСУВ	РС	МИ
18	МСУВ	РС	МО
19	РО	ОС	МИ
20	РО	ОС	МО
21	РО	ДС	МИ
22	РО	ДС	МО
23	РО	РС	МИ
24	РО	РС	МО
25	ОУС	ОС	МИ
26	ОУС	ОС	МО
27	ОУС	ДС	МИ
28	ОУС	ДС	МО

Продолжение таблицы 4

Номер теста	Отбор	Скрещивание	Мутация
29	ОУС	РС	МИ
30	ОУС	РС	МО

Для оценки работы генетического алгоритма при использовании разных методов селекции, скрещивания и мутации были выведены следующие показатели выполнения алгоритма. Все результаты теста были занесены в таблицу 5.

Время обучения в секундах засекалось непосредственно перед началом обучения и после окончания обучения нейронной сети.

Перед началом обучения исходная выборка была разделена на обучающую и тестовую. 70 процентов значений выборки стала обучающей, а 30 процентов оставшихся значений стали тестовой. После обучения было выполнено прогнозирование обученной нейронной сетью на тестовом наборе данных. Соотношение верно спрогнозированных ответов нейронной сети добавлены в таблицу 5 в колонку «Результат». Также на основании прогнозирования сети были вычислены процент верных положительных (ВП) и верных отрицательных (ВО) предсказаний нейронной сети. Благодаря показателям ВП и ВО мы можем оценить вероятность определения положительных и отрицательных результатов.

В данном исследовании было строго ограничено количество поколений. Алгоритм работал до тех пор, пока количество поколений не достигнет 200 и колонка в таблице 5 «Время» отражает затраченное время на обработку всех 200 поколений. Однако в большинстве случаев, решение находилось гораздо раньше. Знание количества поколений для достижения лучшего решения позволяет оценить скорость используемых методов отбора, скрещивания и мутации. В колонке «Поколение» таблицы 5 обозначен номер поколения, в котором было найдено лучшее решение.

Таблица 5 – Результаты тестов генетических алгоритмов

Номер теста	Время, с	Результат, %	ВП, %	ВО, %	Поколение
1	1336	76.84	73.53	80.15	150
2	975	73.9	72.06	75.74	168
3	1404	73.9	68.38	79.41	52
4	809	75.0	77.94	72.06	83
5	1391	74.26	66.18	82.35	112
6	962	75.0	87.5	62.5	60
7	1422	73.9	67.65	80.15	65
8	1083	71.69	69.12	74.26	173
9	1410	78.31	77.21	79.41	20
10	1120	75.0	88.24	61.76	193
11	1341	75.0	70.59	79.41	197
12	884	75.74	73.53	77.94	80
13	1206	70.96	75.0	66.91	17
14	937	76.1	74.26	77.94	35
15	1360	77.94	77.94	77.94	40
16	1305	74.63	63.97	85.29	132
17	1439	75.74	76.47	75.0	196
18	796	76.84	84.56	69.12	84
19	1407	73.9	68.38	79.41	29
20	609	47.43	52.94	41.91	0
21	1337	74.26	80.15	68.38	86
22	685	44.12	53.68	34.56	0

Продолжение таблицы 5

Номер теста	Время, с	Результат, %	ВП, %	ВО, %	Поколение
23	1399	74.26	69.85	78.68	45
24	842	58.82	82.35	35.29	0
25	1415	79.41	83.82	75.0	49
26	1072	76.47	70.59	82.35	19
27	1402	75.74	74.26	77.21	91
28	1142	70.59	87.5	53.68	16
29	1396	73.9	66.91	80.88	200
30	1087	74.26	66.18	82.35	30

После получения результатов, на основании таблицы 5 были вычислены новые данные. Исходные данные были сгруппированы по операторам отбора, скрещивания и мутации, а затем для каждого поля были вычислены сводные данные, а именно минимальное значение, максимальное значение и среднее значение. Строке с вычисленными минимальными значениями соответствуют строки с значением «Мин» в столбце «Функция», строке с вычисленными максимальными значениями соответствуют строки с значением «Макс» в столбце «Функция», а строке с вычисленными средними значениями соответствуют строки с значением «Средн» в столбце «Функция». Все сводные данные представлены в таблице 6.

Таблица 6 – Сводная информация по тестам генетических алгоритмов

Оператор	Функция	Время, с	Результат, %	ВП, %	ВО, %	Поколение
РО	Мин	609	44	53	35	0
	Макс	1407	74	82	79	86
	Средн	1046	62	68	56	26,67
МР	Мин	884	72	68	62	20

Продолжение таблицы 6

Оператор	Функция	Время, с	Результат, %	ВП, %	ВО, %	Поколение
МР	Макс	1422	78	88	80	197
	Средн	1210	75	74	75	121
ОУС	Мин	1072	71	66	54	16
	Макс	1415	79	88	82	200
	Средн	1252	75	75	75	67
МСУВ	Мин	796	71	64	67	17
	Макс	1439	78	85	85	196
	Средн	1173	75	75	75	84
ТО	Мин	809	74	66	63	52
	Макс	1404	77	88	82	168
	Средн	1146	75	74	75	104
ОС	Мин	609,00	0,47	0,42	0,53	0,00
	Макс	1422,00	0,79	0,82	0,84	173,00
	Средн	1146,20	0,72	0,73	0,71	70,50
ДС	Мин	685,00	0,44	0,35	0,54	0,00
	Макс	1410,00	0,78	0,85	0,88	193,00
	Средн	1197,40	0,72	0,69	0,75	71,30
УС	Мин	796,00	0,59	0,35	0,66	0,00
	Макс	1439,00	0,77	0,82	0,88	200,00
	Средн	1153,70	0,73	0,72	0,74	100,40
МИ	Мин	1206	71	66	67	17
	Макс	1439	79	84	82	200

Продолжение таблицы 6

Оператор	Функция	Время, с	Результат, %	ВП, %	ВО, %	Поколение
МИ	Средн	1377	75	73	77	89
МО	Мин	609	44	53	35	0
	Макс	1305	77	88	85	193
	Средн	953	70	74	66	71
Сводка по всем	Мин	609	44	53	35	0
	Макс	1439	79	88	85	200
	Средн	1165	72	73	72	80

Из данных таблицы 6 можно заметить очевидное превосходство по времени методов мутации обменом перед методом мутации инверсией. Среднее время выполнения тестов с использованием для мутации метода инверсии равно 1377.7 секунд, а для тестов использующих для мутации метода мутации обменом равно 953.87 секунд, а это означает, что обучение нейронной сети с использованием метода обмена, в качестве метода мутации примерно на 31% быстрее реализации обучения нейронной сети при использовании метода мутации инверсией. При этом средний процент верных выводов сети с использованием мутации инверсией больше на 5, чем с мутацией обменом.

Для оценки скорости прогрессирования при нахождении решения и общего поведения алгоритма, в каждом поколении вычислялась особь с самым высоким значением функции приспособленности. В результате были построены графики со значением функции приспособленности самой приспособленной особи в качестве оси Y и номером её поколения в качестве оси X. Графики по тестам отображены на рисунках 15-19.

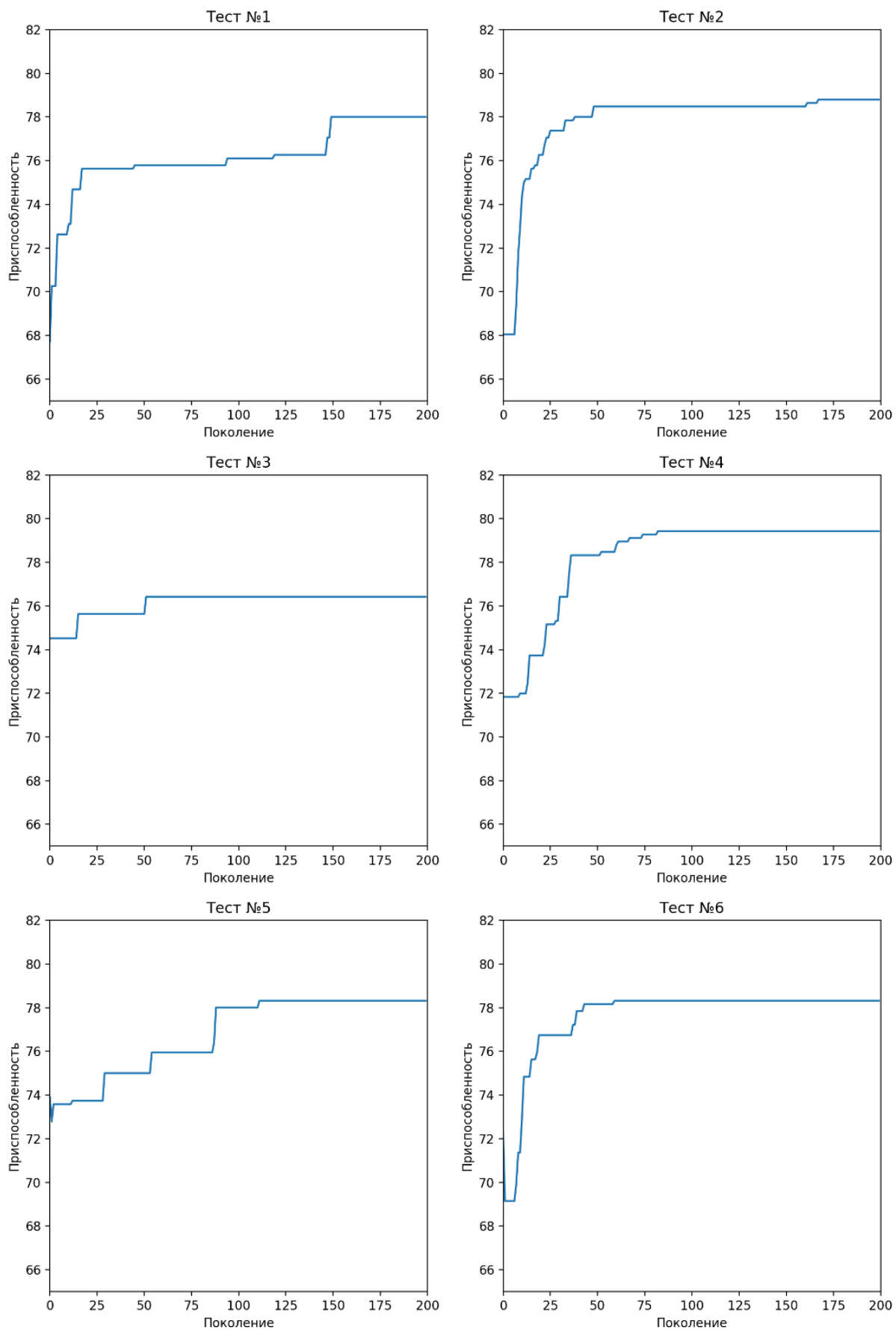


Рисунок 15 – Графики приспособленности для тестов 1-6

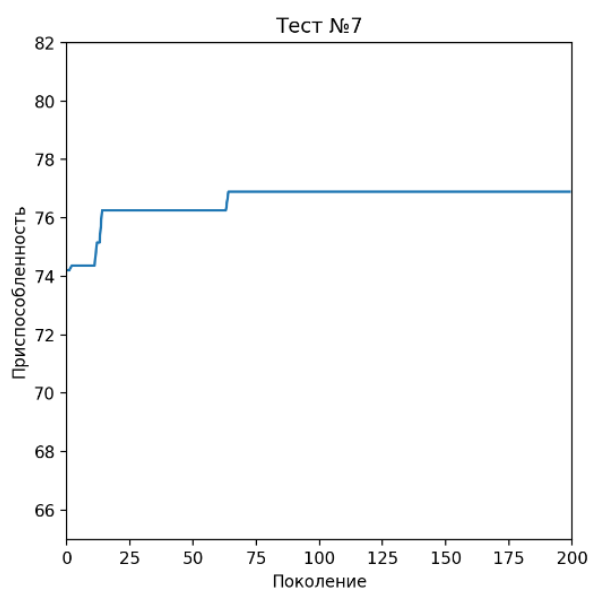
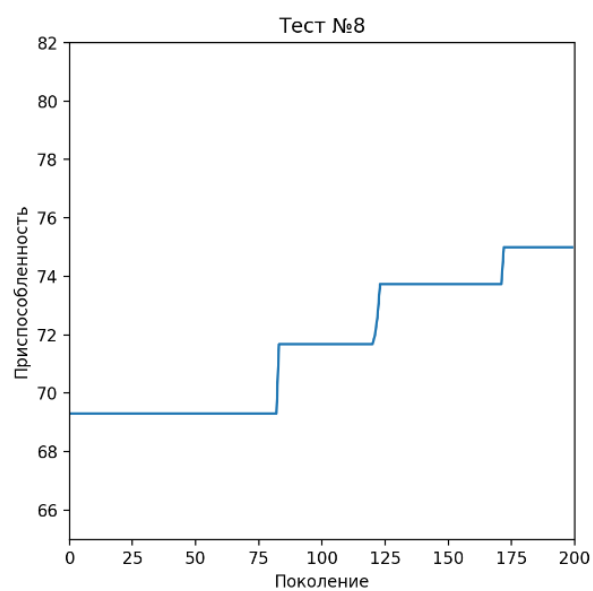
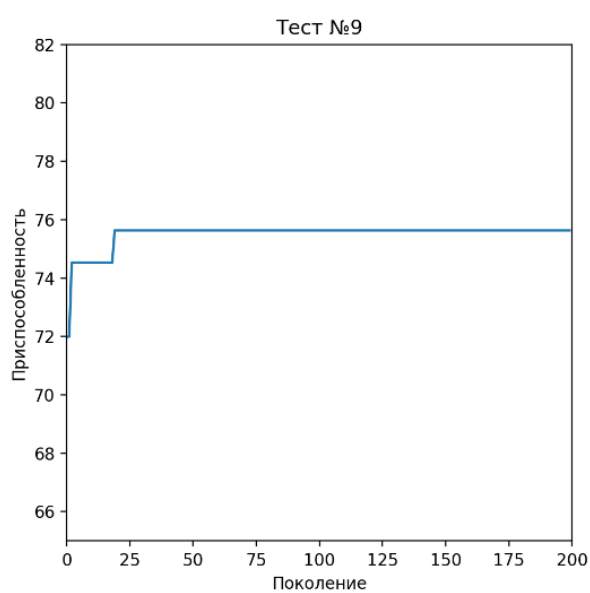
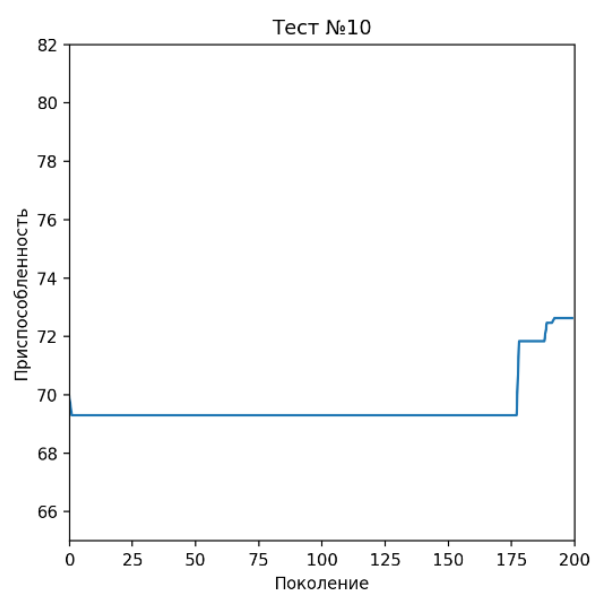
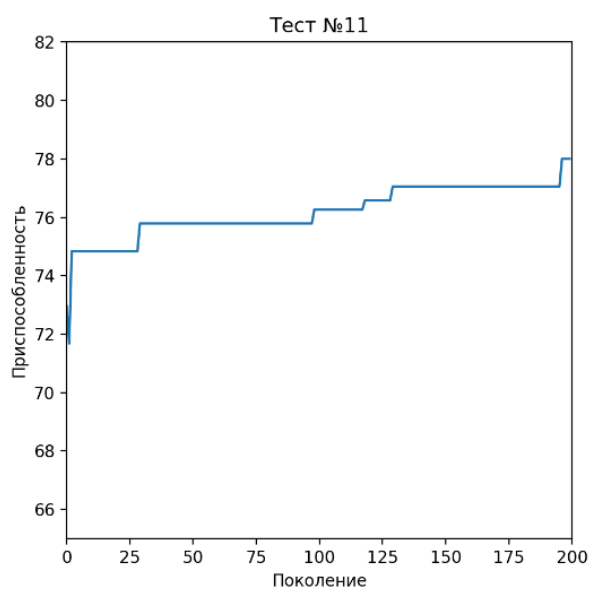
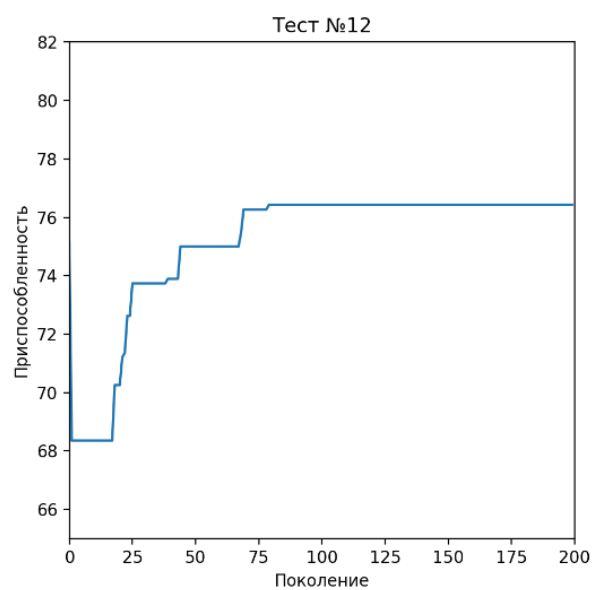


Рисунок 16 – Графики приспособленности для тестов 7-12

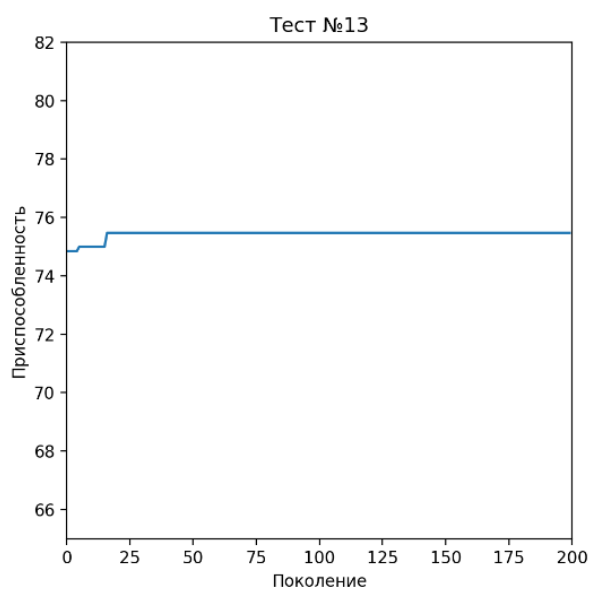
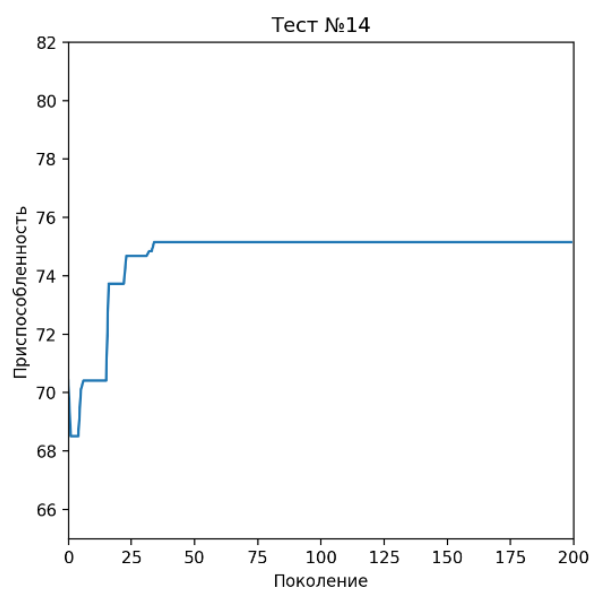
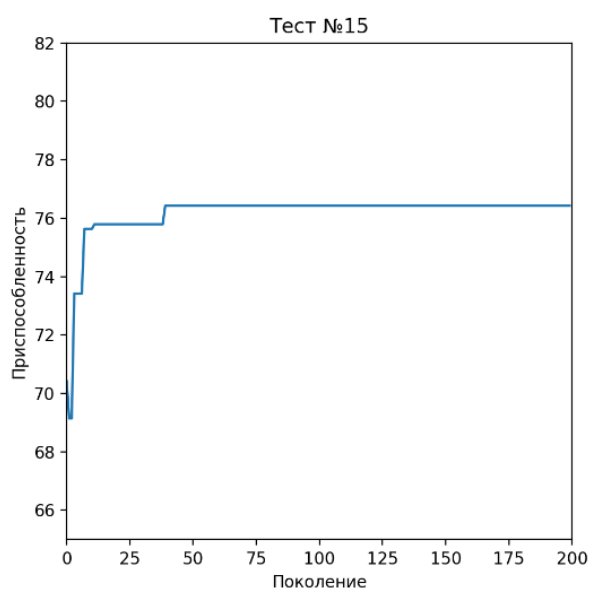
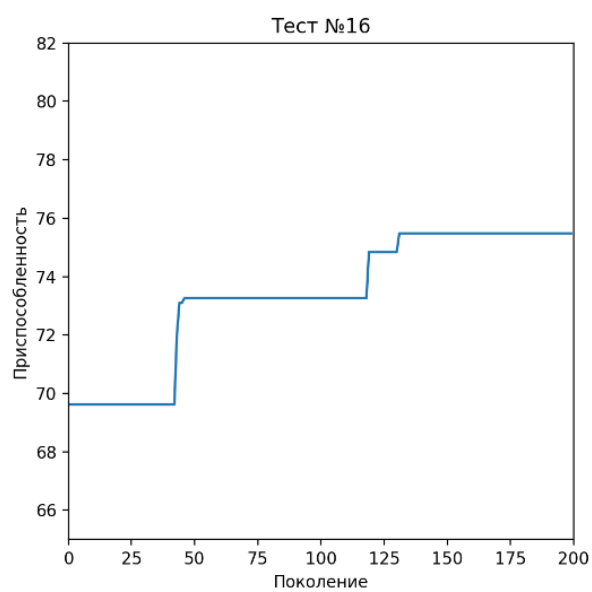
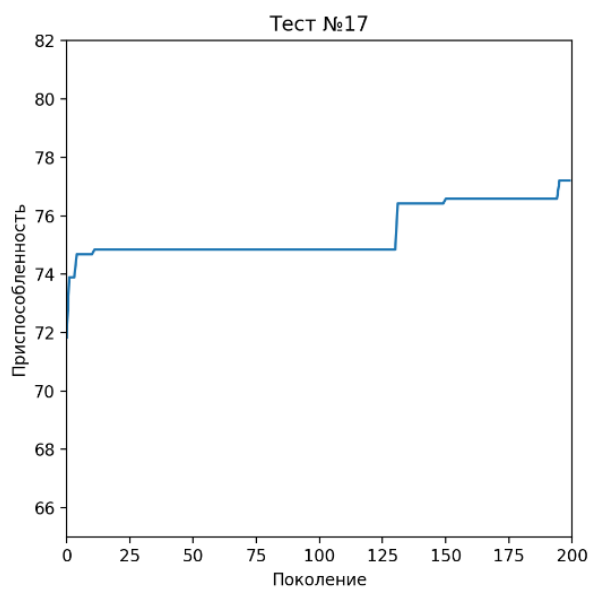
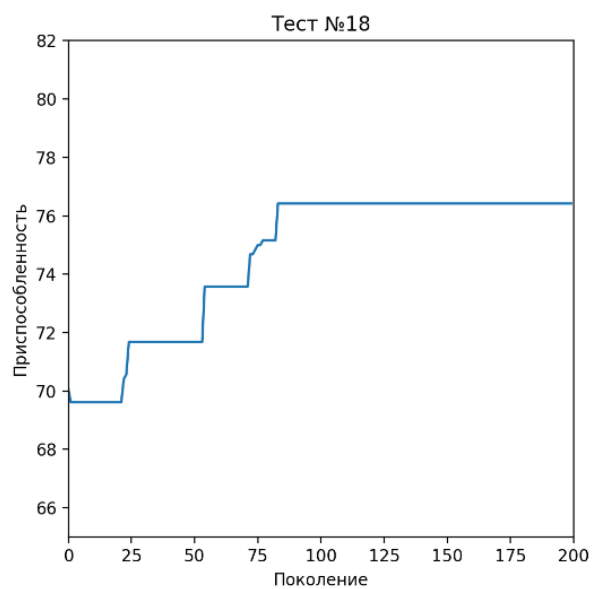


Рисунок 17 – Графики приспособленности для тестов 13-18

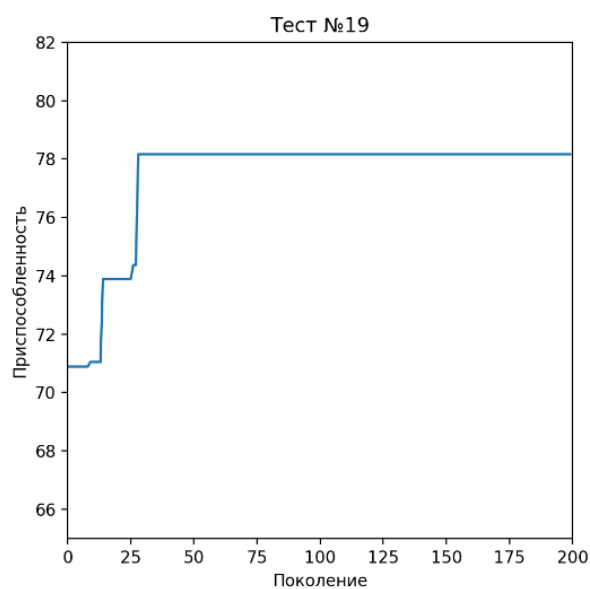
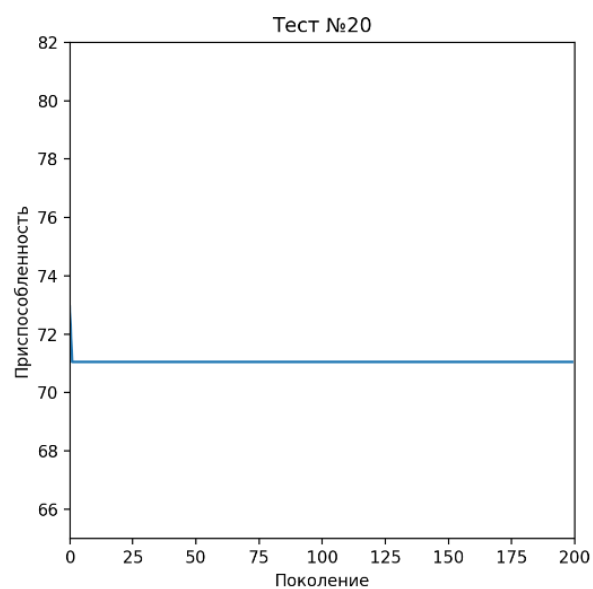
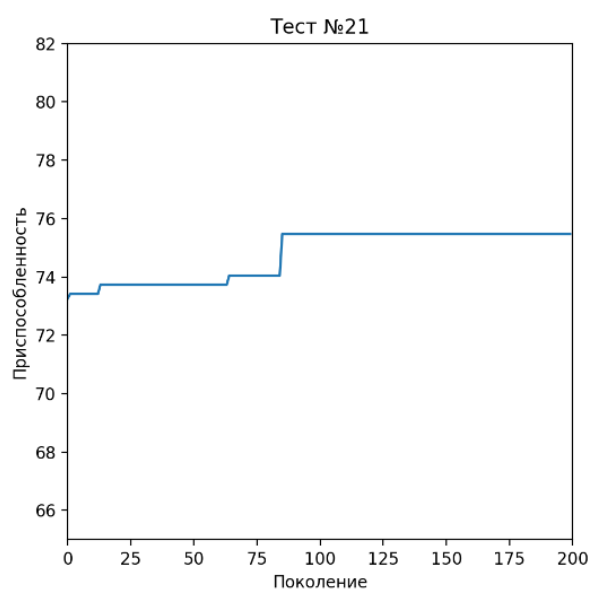
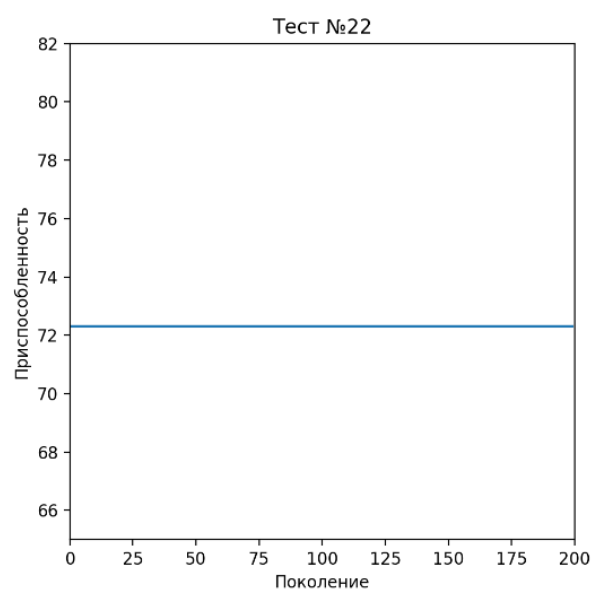
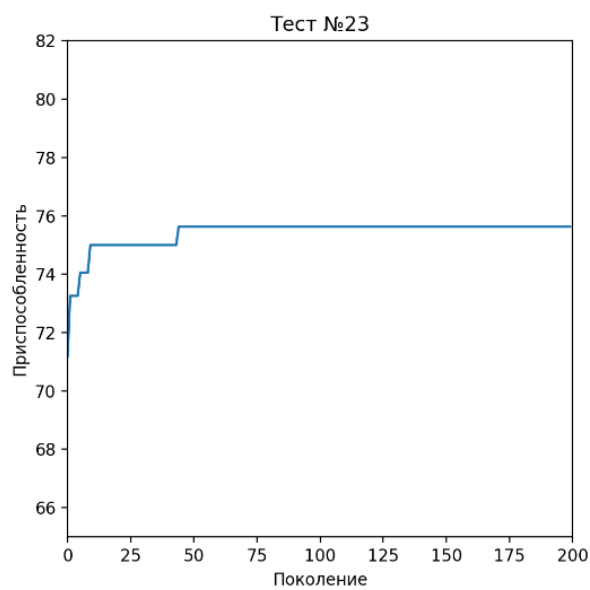
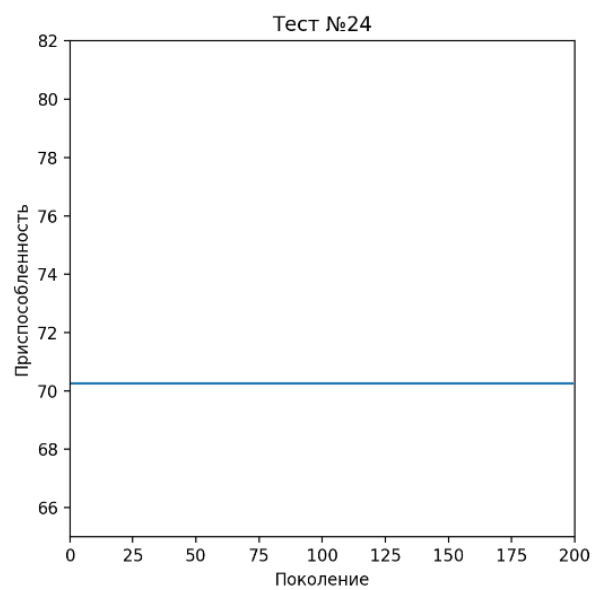


Рисунок 18 – Графики приспособленности для тестов 19-24

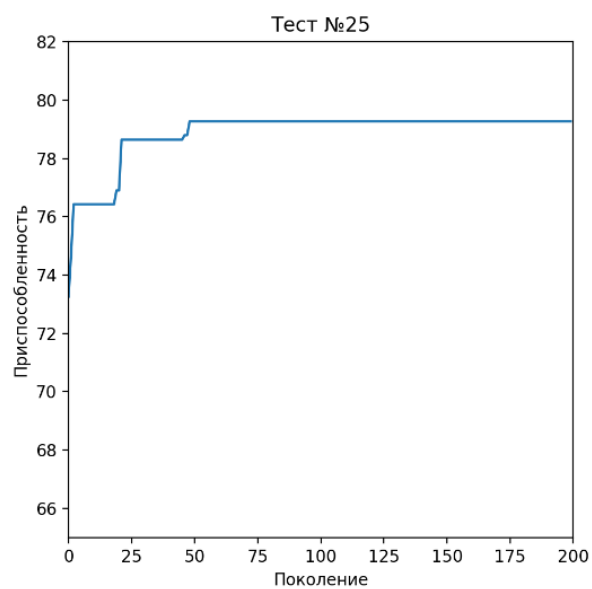
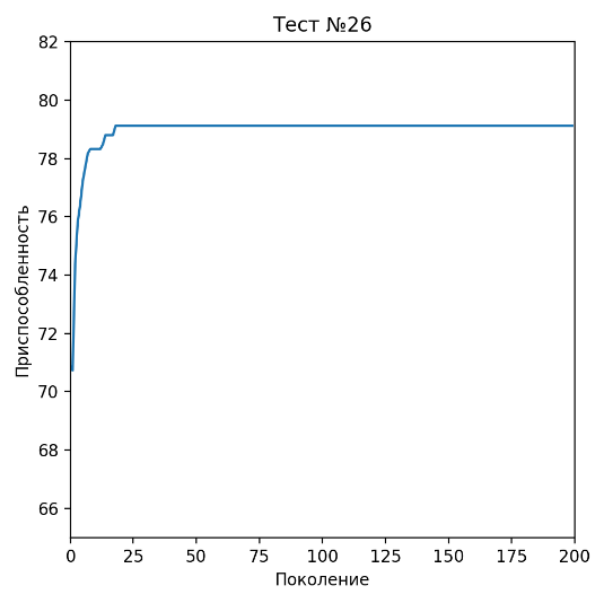
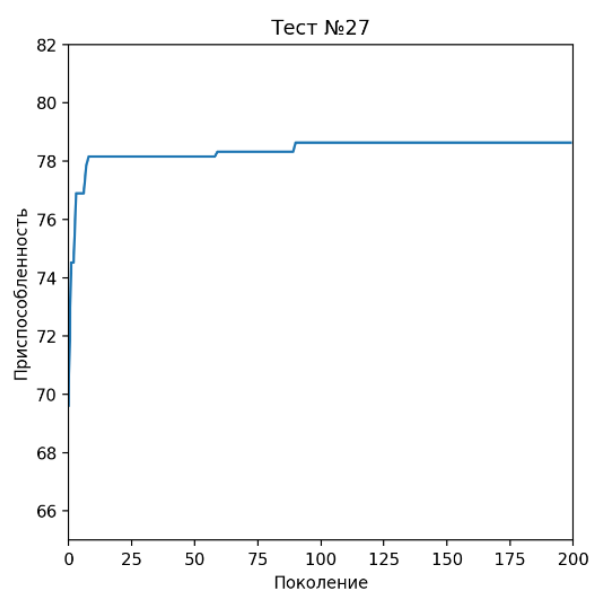
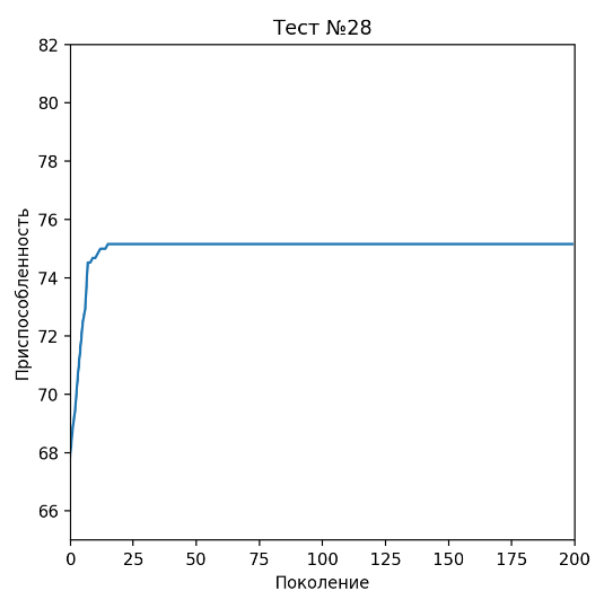
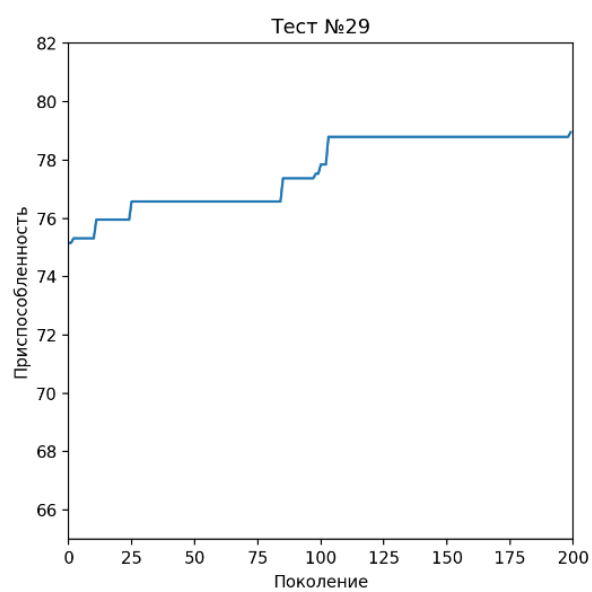
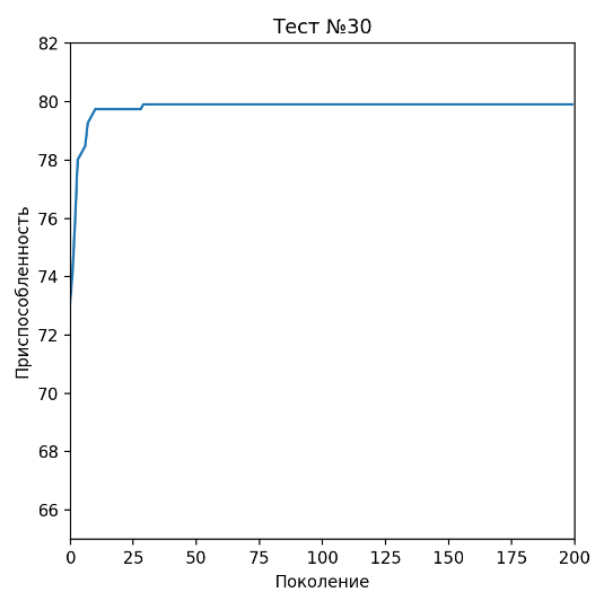


Рисунок 19 – Графики приспособленности для тестов 25-30

В результате тестов было выяснено, что для данной задачи реализация генетических алгоритмов библиотеки PyGAD является подходящим решением вне зависимости от выбранных параметров, за исключением тестов 20, 22 и 24. Если учитывать, что методы отбора, скрещивания и мутации используемые в тестах 20, 22, 24 были применены в других тестах, и результаты были в пределах нормы, то можно предположить, что в реализации генетических алгоритмов библиотеки PyGAD имеется ошибка, либо, что методы используемые в тестах 20, 22, 24 в комбинации приводят к неоднозначным результатам.

Также по графикам приспособленности тестов можно заметить, что в тестах, в которых использовался метод мутации обменом, чаще генерировались особи с большим значением функции эффективности, чем при использовании метода мутации инверсией.

3.2 Результаты обучения нейронной сети с помощью градиентного спуска

Было выполнено 6 запусков программы обучения нейронной сети с использованием градиентного спуска в качестве метода оптимизации функции потерь с разным количеством итераций. Запуск выполнялся с 100, 500, 1000, 5000, 10000 и 50000 итерациями соответственно.

По сравнению с тестами генетического алгоритма, в данном случае не будет отслеживаться номер итерации, на которой было достигнуто лучшее решение. Для работы генетического алгоритма хранение информации о лучшем решении было встроенной функцией библиотеки PyGAD, которую нельзя отключить, из-за чего отслеживание данного параметра не влияло на скорость его работы. Однако добавление такой функции в алгоритм градиентного спуска существенно замедлит работу алгоритма, что исказит данные о скорости его работы. Результаты тестов градиентного спуска представлены в таблице 9.

Таблица 9 – Результаты тестов градиентного спуска

Номер теста	Итераций	Время, с	Результат, %	ВП, %	ВО, %
1	100	0.54	63,97	59,81	68,31
2	500	3.08	69,49	68,01	71,01
3	1000	5.82	74,27	71,32	77,21
4	5000	27.90	74,26	72,06	76,47
5	10000	57.02	79,04	75	83,09
6	50000	289.98	77,57	75	80,15

При запуске программы с генетическим алгоритмом, было зафиксировано, что загрузка процессора была равна 14%, а при запуске программы с градиентным спуском загрузка процессора поднялась до 56%. Таким образом есть возможность выполнить запуск параллельно до 7 программ с генетическими алгоритмами без ущерба скорости выполнения, а для градиентного спуска такой возможности нет.

Из тестов 1-3 видно, что, используя шаг 0.0001, алгоритм не успевает достигнуть минимума за установленное количество итераций. Однако в промежутке между 1000 и 10000 был найден минимум. При этом в тесте №4 и №6 алгоритм явно попал в локальный оптимум, то есть стартовая точка инициализирована была таким образом, что алгоритм попадает в локальный минимум функции и застревает в нём. В подобном случае не имеет значения сколько итераций выбрать, что подтверждает тест №6, в котором 50000 итераций, ведь тест №5 достиг лучшего результата за 10000 итераций, что в 5 раз меньше, благодаря более удачной точке инициализации.

После выполнения тестов был достигнут максимальный процент верных ответов обученной нейронной сети, равный 79% в тесте генетических алгоритмов №25 и тесте градиентного спуска №5.

3.3 Выводы

В ходе тестирования было выяснено, что обучение с помощью генетического алгоритма занимает больше времени чем при использовании градиентного спуска, но при этом имеет большую вероятность нахождения более подходящего локального, а возможно и глобального минимума. Если проводить аналогию, то запуск программы генетического алгоритма можно сравнить с многократным запуском программы градиентного спуска в различных точках, из-за чего повышается вероятность нахождения лучшего решения. Кроме того, тестирование обоих методов проводилось в однопоточном режиме, тогда как к генетическим алгоритмам есть возможность применить распараллеливание вычислений.

В тестах оба метода достигли максимального процента верных ответов обученной нейронной сети равного 79%, но при этом время обучения генетических алгоритмов в тесте №25 составляет 1415 секунд, а в тесте градиентного спуска №5 57 секунд. Если учесть, что наилучшее решение в тесте генетических алгоритмов №25 было найдено в 49 поколении, а 1415 секунд обрабатывалось 200 поколений, то при среднем времени обработки одного поколения 7.075 секунд, алгоритм нашёл решение примерно за 347 секунд, что всё ещё значительно больше, чем в тесте градиентного спуска. Можно сделать вывод, что в рамках данной задачи эффективнее использовать градиентный спуск.

ЗАКЛЮЧЕНИЕ

В рамках данной выпускной квалификационной работы были проведены исследования генетических алгоритмов и способов управления генетическими операциями. Исследование проводилось на примере реализации генетических алгоритмов в качестве метода оптимизации функции потерь в процессе обучения искусственной нейронной сети. Было выполнено 30 тестовых запусков, и каждому тесту соответствовала собственная комбинация методов отбора, скрещивания и мутации. Собирая информацию о различных генетических операциях, можно сделать выводы о скорости и эффективности того или иного метода. Для более объективного анализа результатов, полученных в ходе тестирования генетических алгоритмов, был выполнен запуск программы обучения с использованием аналогичного метода – градиентного спуска. В результате выполненной работы было доказано, что генетические алгоритмы – это конкурентноспособный инструмент для оптимизации функции потерь, по эффективности обучения искусственной нейронной сети не уступающий градиентному спуску, но проигрывающий в плане скорости обучения.

Применение генетического алгоритма в обучении нейронных сетей обосновывается его преимуществами по сравнению с другими оптимизационными методами:

- обширная область поиска,
- устойчивость к локальным оптимумам,
- имеется возможность распараллелить процесс,
- область применения шире, чем у аналогичных методов.

Полученные результаты обладают научной ценностью. Ценность заключается в перспективе дальнейших исследований в области генетических алгоритмов. Поставленные в работе научные и практические задачи выполнены полностью.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Панченко, Т. В. Генетические алгоритмы [Текст]: учебно-методическое пособие / под ред. Ю. Ю. Тарасевича. — Астрахань: Издательский дом «Астраханский университет», 2007. — 87 [3] с.
2. Тищенко Е. Н., Мушта Е. Е., Жилина Е. В. Анализ особенностей объективно-ориентированного программирования C++. Актуальные исследования. 2022. №50 (129). Ч.I. С. 60-62. URL: <https://apni.ru/article/5180-analiz-osobennostej-obektno-orientirovannogo>
3. Давыдов А.В., Жусупова А.К., Салыкова О.С. Сравнение различных языков программирования в машинном обучении. Международный научный журнал «ВЕСТНИК НАУКИ» №2 (59) Т.3.
4. openGA [Электронный ресурс] – URL: <https://github.com/neal-nie/openGA/tree/master>. (дата обращения: 24.03.2024).
5. GAlib. A C++ Library of Genetic Algorithm Components [Электронный ресурс] – URL: <http://lancet.mit.edu/ga/>. (дата обращения: 24.03.2024).
6. Welcome to openGA's documentation! [Электронный ресурс] – URL: <https://openga.readthedocs.io/en/latest/>. (дата обращения: 24.03.2024).
7. Arash Mohammadi, Houshyar Asadi, Shady Mohamed, Kyle Nelson, Saeid Nahavandi. «openGA, a C++ Genetic Algorithm Library». 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
8. GAlib Documentation. [Электронный ресурс] – URL: <http://lancet.mit.edu/galib-2.4/>. (дата обращения: 24.03.2024).
9. Бессонов, Д. В. C++ библиотека компонентов генетических алгоритмов. Молодой ученый. — 2014. — № 6 (65). — С. 73-77. — URL: <https://moluch.ru/archive/65/10815/>. (дата обращения: 24.03.2024).
10. EasyGA [Электронный ресурс] – URL: <https://github.com/danielwilczak101/EasyGA>. (дата обращения: 24.03.2024).
11. DEAP EasyGA [Электронный ресурс] – URL: <https://github.com/deap/deap>. (дата обращения: 24.03.2024).

12. PyGAD - Python Genetic Algorithm! [Электронный ресурс] – URL: <https://pygad.readthedocs.io/en/latest/index.html>. (дата обращения: 24.03.2024).

13. EasyGA [Электронный ресурс] – URL: <https://github.com/danielwilczak101/EasyGA/wiki/Introduction>. (дата обращения: 24.03.2024).

14. Exploring EasyGA - Genetic algorithms in Python made easy? Let's find out ... [Электронный ресурс] – URL: <https://www.youtube.com/watch?v=XYBhgCxPhUQ>. (дата обращения: 24.03.2024).

15. DEAP documentation [Электронный ресурс] – URL: <https://deap.readthedocs.io/en/master/>. (дата обращения: 24.03.2024).

16. pygad Module [Электронный ресурс] – URL: <https://pygad.readthedocs.io/en/latest/pygad.html>. (дата обращения: 24.03.2024).

17. Ahmed Fawzy Gad. PyGAD: An Intuitive Genetic Algorithm Python Library. School of Electrical Engineering and Computer Science.

18. PyGAD - Python Genetic Algorithm! [Электронный ресурс] – URL: <https://pygad.readthedocs.io/en/latest/index.html>. (дата обращения: 24.03.2024).

19. pygad.gann Module [Электронный ресурс] – URL: <https://pygad.readthedocs.io/en/latest/gann.html>. (дата обращения: 24.03.2024).

20. pandas - Python Data Analysis Library [Электронный ресурс] – URL: <https://pandas.pydata.org>. (дата обращения: 24.03.2024).

21. NumPy [Электронный ресурс] – URL: <https://numpy.org>. (дата обращения: 24.03.2024).

22. scikit-learn: machine learning in Python [Электронный ресурс] – URL: <https://scikit-learn.org>. (дата обращения: 24.03.2024).

23. Spyder [Электронный ресурс] – URL: <https://www.spyder-ide.org>. (дата обращения: 24.03.2024).

24. Naeini, A. A. Assessment of normalization techniques on the accuracy of hyperspectral data clustering / A. A. Naeini, M. Babadi, S. Homayouni // Intern.

Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences.
– 2017. – Vol. 42. – P. 27–30.

25. Годунов А. И., Шишков С. В., Баламян С. Т., Аль Сафтли Ф. Х. Разработка алгоритма оптимизации обучения нейронной сети при определении количества нейронов в скрытом слое в целях повышения вероятности распознавания образов наземной цели. 2021. № 4. С. 106–118. doi:10.21685/2307-4205-2021-4-12

26. Загинайло М.В. Влияние количества нейронов искусственной нейронной сети на эффективность ее работы // E-Scio. 2019. №6 (33). URL: <https://cyberleninka.ru/article/n/vliyanie-kolichestva-neyronov-iskusstvennoy-neyronnoy-seti-na-effektivnost-ee-raboty>. (дата обращения: 24.03.2024).

Приложение А

(Обязательное)

Реализация генетических алгоритмов и способы управления генетическими
операциями

Листинг программы

Приложение А.1 Файл Genetic_algorithm.py

```
import pandas as pd
import numpy as np
import time
import pygad
import pygad.nn
import pygad.gann
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
#Чтение данных из файла
data_diabetes=pd.read_csv('diabetes.csv')
data_diabetes.drop(data_diabetes.columns[[0]], axis= 1 ,
inplace= True)
print(data_diabetes.head())
print(data_diabetes.describe())
print(data_diabetes.shape)
y = data_diabetes.Outcome
X = data_diabetes.drop(columns="Outcome")
#Разделение выборки на обучающую и тестовую
x_train, x_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    stratify
= y,

test_size = 0.3,

random_state = 42)
#Нормализация данных
scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
# Инициализация параметров для функции ГА
data_inputs = x_train
data_outputs = y_train.values
```

```

num_inputs = data_inputs.shape[1] # Кол-во входных
параметров
num_classes = 2 # Кол-во классов
# Методы отбора передаваемые в функцию ГА
# "tournament" - Турнирный отбор
# "rws" - Метод рулетки
# "sus" - Метод стохастической универсальной выборки
# "rank" - Ранговый отбор
# "sss" - Отбор устойчивых состояний
parent_selection_type="tournament"
# Методы скрещивания передаваемые в функцию ГА
# single_point - Одноточечное скрещивание
# two_points - Двухточечное скрещивание
# uniform - Равномерное скрещивание
crossover_type="uniform"
# Методы мутации передаваемые в функцию ГА
# inversion - Мутация инверсией
# swap - Мутация обменом
mutation_type="swap"
num_generations = 200 # Кол-во поколений
num_parents_mating = 2 #Кол-во родительских особей
участвующих в скрещивании
K_tournament=3 # Кол-во участников турнира в турнирном
отборе
# Инициализация популяции
GANN_instance = pygad.gann.GANN(num_solutions=100, # Кол-во
особей

num_neurons_input=data_inputs.shape[1], # Кол-во нейронов
входного слоя

num_neurons_hidden_layers=[12], # Кол-во скрытых слоёв и
нейронов в них

num_neurons_output=2, # Кол-
во выходных нейронов

```

```

hidden_activations=["relu"],

# Функция активации скрытого слоя

output_activation="sigmoid"

# Функция активации выходного слоя

)

population_vectors = pygad.gann.population_as_vectors(
population_networks=GANN_instance.population_networks)
initial_population = population_vectors.copy()
# Функция оценки приспособленности
def fitness_func(ga_instance, solution, sol_idx):
    global GANN_instance, data_inputs, data_outputs
    pred = pygad.nn.predict(
last_layer=GANN_instance.population_networks[sol_idx],
data_inputs=data_inputs)
    correct_pred = np.where(pred == data_outputs)[0].size
    fitness = (correct_pred/data_outputs.size)*100
    return fitness
last_population_fitness = 0
fitness_array_data=[] # Массив хранящий приспособленность
лучшей особи поколения
# Функция для промежуточных вычислений после каждого
поколения
def update_population(ga_instance):
    global GANN_instance, last_population_fitness
    population_vector_to_matrices =
pygad.gann.population_as_matrices(population_networks=GANN_insta
nce.population_networks,

population_vectors=ga_instance.population)
    GANN_instance.update_population_trained_weights(population_t
rained_weights=population_vector_to_matrices)
    fitness_array_data.append(ga_instance.best_solution()[1].cop
y())
    last_population_fitness =
ga_instance.best_solution()[1].copy()
# Инициализация генетического алгоритма

```

```

ga_instance = pygad.GA(num_generations=num_generations,

num_parents_mating=num_parents_mating,

initial_population=initial_population,
                        fitness_func=fitness_func,
                        K_tournament=K_tournament,
                        parent_selection_type=
parent_selection_type,
                        crossover_type=crossover_type,
                        mutation_type=mutation_type,
                        on_generation=update_population
                        )

# Запуск обучения нейронной сети с помощью ГА
time_before=time.time()
ga_instance.run()
time_after=time.time()
# Вычисление времени обучения в секундах и минутах
time_learning_sec=int(time_after-time_before)
time_learning_min=time_learning_sec//60
time_learning_min_sec=time_learning_sec%60
# Извлечение после обучения лучшей особи и данных о ней
best_solution, best_solution_fitness, best_match_idx =
ga_instance.best_solution()
    if ga_instance.best_solution_generation != -1:
        print("Best fitness value reached after
{best_solution_generation}
generations.".format(best_solution_generation=ga_instance.best_s
olution_generation))
        name_test=f'{parent_selection_type}_{crossover_type}_{mutati
on_type}'
        COUNT_TEST=5
        # Запись в файл времени обучения
        File_time = open(f'{COUNT_TEST}_T_{name_test}.txt','w')

```

```

File_time.write(f'{time_learning_sec}\n{time_learning_min}\n
{time_learning_min_sec}')
File_time.close()
# Запись в файл наибольшее значения функции
приспособленности каждого поколения
File_fitness = open(f'{COUNT_TEST}_F_{name_test}.txt','w')
File_fitness.write('\n'.join(map(str,fitness_array_data)))
File_fitness.close()
# Запись в файл номера поколения в котором был достигнут
максимум функции приспособленности
File_max_generation=
open(f'{COUNT_TEST}_MG_{name_test}.txt','w')
File_max_generation.write(str(ga_instance.best_solution_gene
ration))
File_max_generation.close()
# Применение нейронной сети к обучающей выборке и тестовой
y_pred_train =
pygad.nn.predict(last_layer=GANN_instance.population_networks[be
st_match_idx],
                    data_inputs=x_train)
y_pred_test =
pygad.nn.predict(last_layer=GANN_instance.population_networks[be
st_match_idx],
                    data_inputs=x_test)
# Вычисление матрицы неточности
conf_matr=[[0,0],[0,0]]
for i in range(0,len(y_pred_test)):
    if (y_pred_test[i]==0 and y_test.values[i]==0):
        conf_matr[0][0]=conf_matr[0][0]+1
    if (y_pred_test[i]==1 and y_test.values[i]==1):
        conf_matr[1][1]=conf_matr[1][1]+1
    if (y_pred_test[i]==0 and y_test.values[i]==1):
        conf_matr[1][0]=conf_matr[1][0]+1
    if (y_pred_test[i]==1 and y_test.values[i]==0):
        conf_matr[0][1]=conf_matr[0][1]+1
conf_matrix_vector=np.ravel(conf_matr)

```

```

print("Матрица неточности")
print(conf_matrix_vector)

#Запись матрицы неточности в файл
File_conf_matr= open(f'{COUNT_TEST}_CM_{name_test}.txt','w')
File_conf_matr.write('\n'.join(map(str,conf_matrix_vector)))
File_conf_matr.close()

# Вычисление и запись в файл процента верных ответов
нейронной сети для тестовой выборки
score_GA=(conf_matrix_vector[0]+conf_matrix_vector[3])/(conf
_matrix_vector[0]+conf_matrix_vector[1]+conf_matrix_vector[2]+co
nf_matrix_vector[3])

File_Score= open(f'{COUNT_TEST}_S_{name_test}.txt','w')
File_Score.write(str(score_GA))
File_Score.close()

```

Приложении А.2 – Файл Gradient_Descent.py.

```

import pandas as pd
import numpy as np
import time as time

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

class MLGS:

    def __init__(self, inputSize, outputSize,
learning_rate=0.1, hiddenSizes = 12):

        # Инициализация весов нейронной сети

        self.weights = [

            # Веса скрытого слоя

            np.random.uniform(-2, 2,
size=(inputSize,hiddenSizes)),

            # Веса выходного слоя

            np.random.uniform(-2, 2,
size=(hiddenSizes,outputSize))

        ]

```

```

        self.learning_rate = learning_rate

        self.layers = None

# Функция сигмоиды
def sigmoid(self, x):

    return 1 / (1 + np.exp(-x))

# Производная функции сигмоиды
def derivative_sigmoid(self, x):

    return self.sigmoid(x) * (1 - self.sigmoid(x))

# Функция ReLU
def relu(self, x):

    a=[]

    for i in x:

        b=[]

        for j in i:

            if j<0:

                b.append(0)

            else:

                b.append(j)

        a.append(b)

    return np.array(a)

# Производная функции ReLU
def derivative_relu(self, x):

    a=[]

    for i in x:

        b=[]

        for j in i:

            if j<0:

```

```

        b.append(0)

    else:

        b.append(1)

    a.append(b)

    return np.array(a)

# Прямой проход
def feed_forward(self, x):

    input_ = x # Входные сигналы

    hidden_ = self.relu(np.dot(input_, self.weights[0]))
# Выход скрытого слоя = ReLU(Входные сигналы * Веса скрытого
# слоя)

    output_ = self.sigmoid(np.dot(hidden_,
self.weights[1]))# Выход сети = Сигмоида(выход скрытого слоя *
веса выходного слоя)

    self.layers = [input_, hidden_, output_]

    return self.layers[-1]

# Обратный проход
def backward(self, target):

    # Вычисление ошибки сети

    err = (target - self.layers[-1])

    # Выходной слой

    err_delta = err *
self.derivative_sigmoid(self.layers[2])

    # Проброс ошибки на предыдущий слой

    err = np.dot(err_delta, self.weights[2 - 1].T)

    # Ошибка слоя * Производную функции активации *
Входные сигналы слоя

    dw = np.dot(self.layers[2 - 1].T, err_delta)

    # Обновление весов слоя

    self.weights[2 - 1] += self.learning_rate * dw

    # Скрытый слой

```

```

        err_delta = err *
self.derivative_relu(self.layers[1])

        # Проброс ошибки на предыдущий слой

        err = np.dot(err_delta, self.weights[1 - 1].T)

        # Ошибка слоя * Производную функции активации *
Входные сигналы слоя

        dw = np.dot(self.layers[1 - 1].T, err_delta)

        # обновляем веса слоя

        self.weights[1 - 1] += self.learning_rate * dw

        # Функция обучения выполняет подряд прямой и обратный
проход

    def train(self, x_values, target):

        self.feed_forward(x_values)

        self.backward(target)

        # Функция предсказания возвращает выход последнего слоя

    def predict(self, x_values):

        return self.feed_forward(x_values)

df = pd.read_csv('Diabetes.csv')

df.drop(df.columns[[0]], axis= 1 , inplace= True)

y = df.iloc[0:904, 8].values

y=y.reshape(-1,1)

X = df.iloc[0:904,0:8].values

x_train, x_test, y_train, y_test =
train_test_split(X,y,stratify = y,test_size = 0.3,random_state =
42)

scaler = StandardScaler()

scaler.fit(x_train)

x_train = scaler.transform(x_train)

x_test = scaler.transform(x_test)

inputSize = X.shape[1] # Количество входных сигналов равно
количеству признаков задачи

```

```

hiddenSizes = 12 # Число нейронов скрытого слоя

outputSize = 1 # Количество выходных сигналов равно
количеству классов задачи

iterations = 50000

learning_rate = 0.0001

net = MLGS(inputSize, outputSize, learning_rate,
hiddenSizes)

start_learning=time.time()

# Обучение
for i in range(iterations):

    net.train(x_train, y_train)

end_learning=time.time()

time_total=end_learning-start_learning

# Считаем ошибку на обучающей выборке
y_pred_test = net.predict(x_test)

sum2=0

for i in range(0,len(y_pred_test)):

    if y_pred_test[i]>0.5:

        y_pred_test[i]=1

    else:

        y_pred_test[i]=0

    if y_pred_test[i]==y_test[i]:

        sum2=sum2+1

print("Результат")

print('Время')

print(time_total)

print(f'Верных выводов сети {sum2} из
{y_pred_test.shape[0]}')

print(f'Процент верных выводов {sum2/y_pred_test.shape[0]}')

conf_matr=[[0,0],[0,0]]

```

```

for i in range(0,len(y_pred_test)):
    if (y_pred_test[i]==0 and y_test[i]==0):
        conf_matr[0][0]=conf_matr[0][0]+1
    if (y_pred_test[i]==1 and y_test[i]==1):
        conf_matr[1][1]=conf_matr[1][1]+1
    if (y_pred_test[i]==0 and y_test[i]==1):
        conf_matr[1][0]=conf_matr[1][0]+1
    if (y_pred_test[i]==1 and y_test[i]==0):
        conf_matr[0][1]=conf_matr[0][1]+1
a=conf_matr
VO=a[0][0]/(a[0][0]+a[
0][1])
VP=a[1][1]/(a[1][0]+a[1][1])
print("Матрица неточности")
print(conf_matr)
print("VO")
print(VO)
print("VP")
print(VP)

```

Приложение Б

(Обязательное)

Реализация генетических алгоритмов и способы управления генетическими
операциями

Презентация

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
на тему
РЕАЛИЗАЦИЯ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ И
СПОСОБЫ УПРАВЛЕНИЯ ГЕНЕТИЧЕСКИМИ
ОПЕРАЦИЯМИ

Автор работы: студент группы 22ВВИм1 Холодков Д.В.

Научный руководитель: к.т.н., доцент Слепцов Н.В.

Пенза, 2024

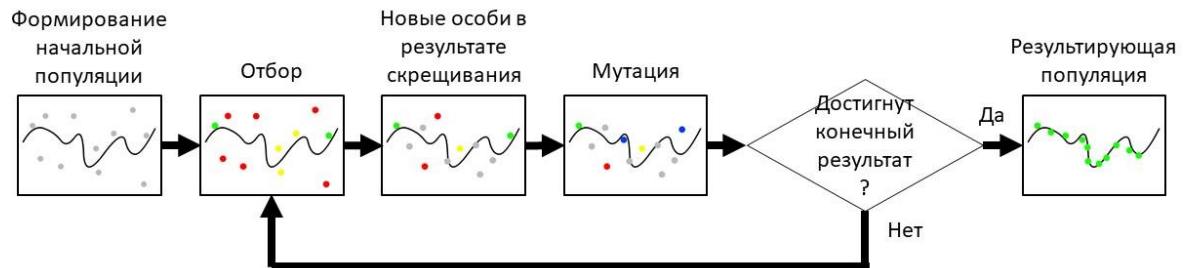
Цель и объект работы

Цель работы: реализация генетических алгоритмов в качестве метода оптимизации функции ошибки в процессе обучения искусственной нейронной сети для решения задачи классификации, а также анализ и сравнение данного метода с аналогом.

Объект работы: объектом работы является программа обучения искусственной нейронной сети с применением генетических алгоритмов в качестве метода оптимизации функции ошибки.

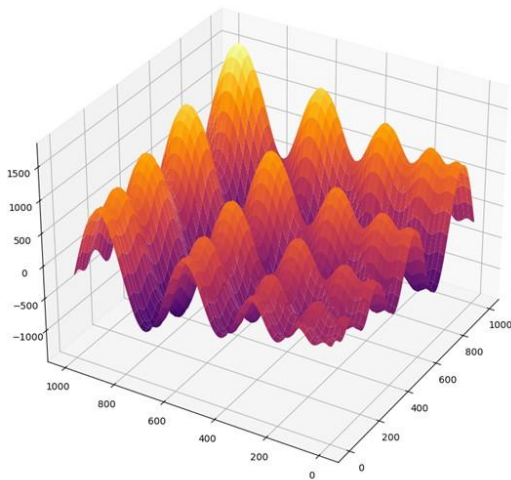
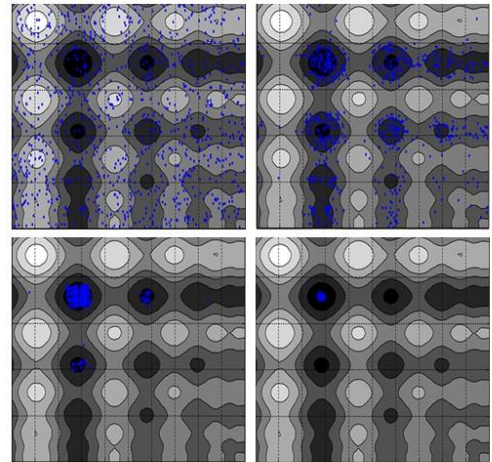
Предметная область

Генетические алгоритмы — это адаптивные методы поиска, которые в последнее время используются для решения задач оптимизации. Этот подход к решению задач оптимизации представляет собой разновидность эволюционных алгоритмов и основан на имитации отбора, скрещивания и мутации в популяции особей.



Функция Ханс-Пауля Швефеля

$$f(x_1, x_2) = -x_1 * \sin \sqrt{|x_1|} - x_2 * \sin \sqrt{|x_2|}$$



Исследованные методы

Отбор

- Метод рулетки
- Метод стохастической универсальной выборки
- Ранговый отбор
- Турнирный отбор
- Отбор устойчивого состояния

Скрещивание

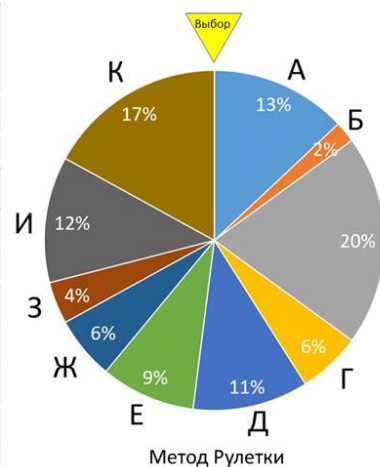
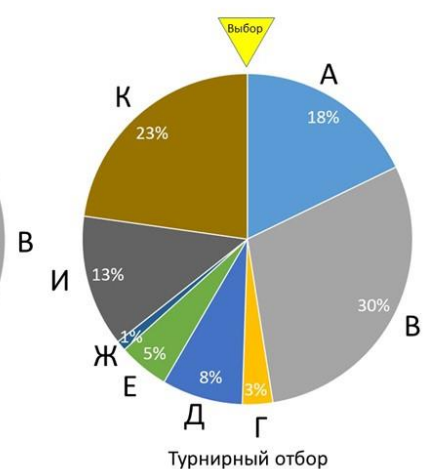
- Одноточечное скрещивание
- Двухточечное скрещивание
- Равномерное скрещивание

Мутация

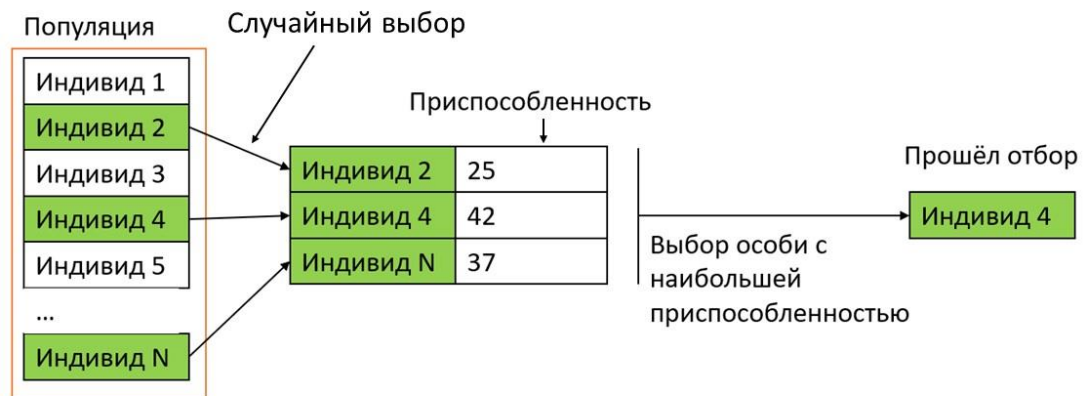
- Мутация инверсией
- Мутация обменом

Вероятность прохождения отбора для особи

Особь	Значение функции приспособленности
А	30
Б	5
В	46
Г	15
Д	27
Е	21
Ж	13
З	10
И	28
К	40



Турнирный отбор



Используемые инструменты



Обучающая выборка

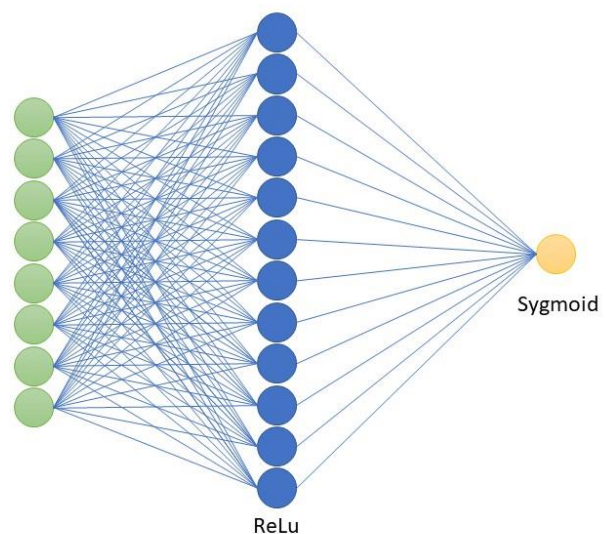
Задачей, для решения которой было решено обучать нейронную сеть, стала задача прогнозирования диабета по показателям здоровья пациентов.

Были найдены данные пациентов женского пола. Обучающая выборка содержит данные о 904 пациентах женского пола, обследованных на наличие диабета.

Беременностей	Глюкоза	Давление	Толщина_Кожи	Инсулин	ИМТ	DPF	Возраст	Результат
6.0	148.0	72.0	35.00	155.61	33.6	0.63	50.0	1.0
1.0	85.0	66.0	29.00	155.61	26.6	0.35	31.0	0.0
8.0	183.0	64.0	28.99	155.61	23.3	0.67	32.0	1.0
1.0	89.0	66.0	23.00	94.00	28.1	0.17	21.0	0.0
0.0	137.0	40.0	35.00	168.00	43.1	2.29	33.0	1.0

Структура нейронной сети

Входной слой: 8 нейронов
Скрытый слой: 12 нейронов
Выходной слой: 1 нейрон

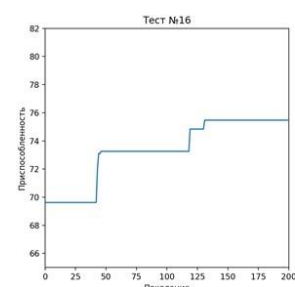
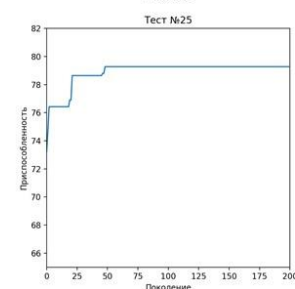


Результаты тестов ГА

№	Время, с	Результат, %	ВП, %	ВО, %	Поколение
16	1305	74.63	63.97	85.29	132
25	1415	79.41	83.82	75.0	49

№	Отбор	Скращивание	Мутация
16	МСУВ	ДС	МО
25	ОУС	ОС	МИ

Оператор	Функция	Время, с	Результат, %	Поколение
МИ	Средн	1377	75	89
МО	Средн	953	70	71



Результаты тестов ГС

№	Итераций	Время, с	Результат, %	ВП, %	ВО, %
1	100	0.54	63,97	59,81	68,31
2	500	3.08	69,49	68,01	71,01
3	1000	5.82	74,27	71,32	77,21
4	5000	27.90	74,26	72,06	76,47
5	10000	57.02	79,04	75	83,09
6	50000	289.98	77,57	75	80,15

Плюсы

- обширная область поиска
- устойчивость к локальным оптимумам
- имеется возможность распараллелить процесс
- не зависит от дифференцируемости функции
- область применения шире, чем у аналогичных методов

Минусы

- проблематично найти точный оптимум
- затратен по времени

Заключение

Все цели и задачи выпускной квалификационной работы были достигнуты. В рамках работы были проведены исследования генетических алгоритмов и способов управления генетическими операциями. Также выполнена реализация генетических алгоритмов в качестве метода оптимизации функции ошибки в процессе обучения искусственной нейронной сети.

В результате выполненной работы было доказано, что генетические алгоритмы – это конкурентноспособный инструмент, по эффективности обучения искусственной нейронной сети не уступающий градиентному спуску, но проигрывающий в плане времени обучения.

Спасибо за внимание!