****МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИРОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

**Факультет (институт, филиал) №3 «Системы управления, информатика и электроэнергетика» Кафедра 304 \_\_\_\_**

**Направление подготовки 230100 Группа 30-410Б\_\_\_\_\_\_**

**Квалификация (степень) бакалавр \_\_\_**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**БАКАЛАВРА**

На тему: Разработка программного обеспечения оптимизации гиперпараметров искусственной нейронной сети\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Автор квалификационной работы\_\_\_Кофман\_Марк\_Сергеевич\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_)

(Фамилия, Имя, Отчество)

Руководитель\_\_\_\_доц.каф.304, к.т.н, \_Чебатко\_Марина\_Игоревна\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_)

(Фамилия, Имя, Отчество)

Консультант\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_)

(Фамилия, Имя, Отчество)

Консультант\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_)

(Фамилия, Имя, Отчество)

Рецензент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_)

(Фамилия, Имя, Отчество)

**К з а щ и т е д о п у с т и т ь**

Зав. кафедрой\_\_№304, д.т.н., проф. Брехов О.М.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(\_\_\_\_\_\_\_\_\_\_\_)

(Фамилия, И, О)

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017\_\_г.

Москва 2017\_г.

Оглавление

[**ВВЕДЕНИЕ** 4](#_Toc483419770)

[**Глава 1.** **ОБЗОР СУЩЕСТВУЮЩИХ АЛГОРИТМОВ ОПТИМИЗАЦИИ ТОПОЛОГИИ НЕЙРОННЫХ СЕТЕЙ** 7](#_Toc483419771)

[**1.1** **Имитация отжига** 7](#_Toc483419772)

[**1.2** **Генетические алгоритмы** 8](#_Toc483419773)

[**1.3** **Поиск с запретами** 8](#_Toc483419774)

[**1.4** **Муравьиный алгоритм** 9](#_Toc483419775)

[**1.5** **«Жадный» адаптивный метод случайного поиска** 9](#_Toc483419776)

[**1.6** **Метод роя частиц** 10](#_Toc483419777)

[**Глава 2.** **СРАВНЕНИЕ АЛГОРИТМОВ ОПТИМИЗАЦИИ СЕТИ И ВЫБОР МЕТОДА ОПТИМИЗАЦИИ** 11](#_Toc483419778)

[**2.1** **Классификация по используемым подходам** 11](#_Toc483419779)

[**2.2** **Сравнение по производительности** 11](#_Toc483419780)

[**2.3** **Сравнение достоинств и недостатков** 14](#_Toc483419781)

[**2.4** **Анализ выбранного метода оптимизации** 15](#_Toc483419782)

[**2.4.1** **Прямое и косвенное кодирование** 16](#_Toc483419783)

[**2.4.2** **NEAT** 18](#_Toc483419784)

[**2.4.3** **Параметры генетических алгоритмов** 19](#_Toc483419785)

[**2.4.4** **Проблема переобучения** 22](#_Toc483419786)

[**Глава 3.** **АНАЛИЗ ДОСТУПНЫХ БИБЛИОТЕК ПО НЕЙРОННЫМ СЕТЯМ ДЛЯ ЯЗЫКА JAVA** 24](#_Toc483419787)

[**3.1** **Encog** 24](#_Toc483419788)

[**3.2** **Neuroph** 25](#_Toc483419789)

[**3.3** **Сравнение API библиотек на практике** 25](#_Toc483419790)

[**3.3.1** **Основная часть программы** 27](#_Toc483419791)

[**3.3.2** **Функция приспособленности** 28](#_Toc483419792)

[**3.3.3** **Функция вычисления ошибки функционирования** 29](#_Toc483419793)

[**3.3.4** **Сравнение функционирования реализаций** 31](#_Toc483419794)

[**3.3.5** **Изменение структуры сети** 33](#_Toc483419795)

[**Глава 4.** **ОБЗОР СУЩЕСТВУЮЩИХ ГИПЕРПАРАМЕТРОВ НЕЙРОННЫХ СЕТЕЙ И ВЫБОР ГИПЕРПАРАМЕТРОВ ДЛЯ ОПТИМИЗАЦИИ** 35](#_Toc483419796)

[**4.1** **Топология сети** 36](#_Toc483419797)

[**4.1.1** **Число скрытых слоев** 36](#_Toc483419798)

[**4.1.2** **Число нейронов в скрытых слоях** 36](#_Toc483419799)

[**4.1.3** **Связи между нейронами** 37](#_Toc483419800)

[**4.2** **Параметры активационной функции** 38](#_Toc483419801)

[**4.3** **Коэффициент скорости обучения** 39](#_Toc483419802)

[**4.4** **Момент** 39](#_Toc483419803)

[**4.5** **Диапазон случайной инициализации весов** 40](#_Toc483419804)

[**Глава 5.** **РАЗРАБОТКА АЛГОРИТМА ОПТИМИЗАЦИИ** 41](#_Toc483419805)

[**5.1** **Кодирование топологии сети** 42](#_Toc483419806)

[**5.2** **Алгоритмы мутаций** 44](#_Toc483419807)

[**5.2.1** **Мутации слоев** 45](#_Toc483419808)

[**5.2.2** **Мутации нейронов в слоях** 47](#_Toc483419809)

[**5.2.3** **Мутации связей между нейронами** 48](#_Toc483419810)

[**5.3** **Алгоритм скрещивания** 49](#_Toc483419811)

[**5.4** **Алгоритм вычисления функции приспособленности** 51](#_Toc483419812)

[**Глава 6.** **ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА** 53](#_Toc483419813)

[**Глава 7.** **ОТЛАДКА И ТЕСТИРОВАНИЕ ПРОГРАММЫ** 57](#_Toc483419814)

[**ЗАКЛЮЧЕНИЕ** 63](#_Toc483419815)

[**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ** 65](#_Toc483419816)

[**ПРИЛОЖЕНИЕ А** 68](#_Toc483419817)

[**ПРИЛОЖЕНИЕ Б** 75](#_Toc483419818)

[**ПРИЛОЖЕНИЕ В** 92](#_Toc483419819)

**ВВЕДЕНИЕ**

Использование искусственных нейронных сетей позволяет достичь высокой эффективности решения для многих проблем. Но несмотря на то, что для нейронных сетей существует множество разработанных моделей обучения, параметры которых оптимально подобраны под определенный класс задач, у них также есть большое число гиперпараметров, существенно влияющих на производительность и потому требующих оптимизации.

Ранее процесс подбора гиперпараметров проходил вручную путем проведения небольшого числа испытаний из-за низких вычислительных мощностей или же основывался на опыте экспертов. В настоящее время данный процесс можно автоматизировать и существенно упростить. В связи с этим актуальной представляется задача разработки программного обеспечения, способного оптимизировать гиперпараметры некоторой подающейся на вход нейронной сети, улучшив ее производительность и по возможности упростив ее структуру.

Объектом исследования данной работы, таким образом, являются гиперпараметры нейронных сетей, а предметом ее исследования – способы их оптимизации. Цель работы заключается в создании конечного программного продукта, принимающего на вход некоторую нейронную сеть и возвращающего пользователю ее улучшенную версию с оптимизированными гиперпараметрами и меньшей ошибкой функционирования.

Для достижения указанной цели необходимыми этапами представляются следующие задачи:

1. Рассмотреть существующие метаэвристики, применяющиеся в процессах оптимизации;
2. Определить среди рассмотренных метаэвристик наиболее подходящую для задачи оптимизации гиперпараметров нейронных сетей;
3. Выбрать подходящие программные средства для создания программного продукта на языке Java в IDE NetBeans;
4. Определиться с тем, какие именно из существующих гиперпараметров будет оптимизировать данное программное обеспечение;
5. Разработать алгоритм оптимизации, на основе которого будет создано программное обеспечение;
6. Создать функционирующее программное обеспечение;
7. Проверить, действительно ли созданное программное обеспечение эффективно оптимизирует гиперпараметры нейронных сетей.

При успешном выполнении всех поставленных задач результатом будет являться законченный программный продукт с возможностью последующей модернизации, то есть добавления иных оптимизируемых гиперпараметров.

В соответствии с заданием, работа состоит из введения, семи глав и заключения.

Введение отражает актуальность и значимость работы, определяет объект, предмет, цель, задачи и методы исследования.

В первой главе рассмотрены некоторые из существующих метаэвристик, использующихся для задач оптимизации. Во второй главе производится сравнение данных метаэвристик, выбор наиболее подходящей из них для задачи исследования и ее более подробный обзор. Третья глава посвящена анализу программных средств с открытым исходным кодом, позволяющих создавать, изменять и проводить глубокую настройку гиперпараметров искусственных нейронных сетей на языке Java в IDE NetBeans. В четвертой главе производится отбор для дальнейшей оптимизации посредством разрабатываемого программного обеспечения некоторых из гиперпараметров нейронных сетей и приводятся соображения по способам их оптимизации. Пятая глава содержит основные разработанные алгоритмы, использующиеся в программном обеспечении. Шестая глава описывает структуру кода программного обеспечения, а именно разработанные классы, поля данных, методы и использующиеся форматы файлов. В седьмой главе созданное программное обеспечение проходит проверку на эффективность оптимизации на нейронной сети, распознающей монохромные образы, а разработанный способ оптимизации также сравнивается с другим существующим способом.

В заключении работы содержатся итоги исследования и выводы по рассмотренной теме, а также приводятся соображения по дальнейшей доработке созданного программного обеспечения.

1. **ОБЗОР СУЩЕСТВУЮЩИХ АЛГОРИТМОВ ОПТИМИЗАЦИИ ТОПОЛОГИИ НЕЙРОННЫХ СЕТЕЙ**

Рассматриваемые ниже алгоритмы оптимизации применимы не только к оптимизации нейронных сетей, но и к любым задачам оптимизации и поиска решений для широкого круга задач. Эти алгоритмы принадлежат области, называемой «метаэвристикой». Метаэвристика – общий подход к решению широкого класса задач поиска и оптимизации, чаще всего не имеющий строгого обоснования, но дающий хорошие практические результаты [5]. Значительная часть метаэвристик представляет собой гибрид алгоритмов случайного поиска, обеспечивающего определенный разброс решений, и локальной оптимизации, обеспечивающей определенное качество полученного решения [4].

Терминология, используемая при описании метаэвристик, специфична и связана с процессами, вдохновившими создание каждой данной метаэвристики. Так, одни метаэвристики основаны на поведении животных, другие – на биологических концептах, третьи – на производственных процессах и так далее.

Следует также отметить, что представленные ниже алгоритмы представляют лишь часть из существующих на данный момент.

* 1. **Имитация отжига**

Метод имитации отжига (Simulated annealing, далее – SA) основан на физическом процессе отжига твердых тел. Отжиг представляет собой процесс получения низких энергетических состояний твердого тела при термообработке. Процесс отжига начинается с расплавления твердого вещества с помощью термической обработки. Затем температура снижается, что приводит тело в состояние с минимальной энергией. В процессе охлаждения тела возможно кратковременное повышение энергетического уровня, что при реализации процесса позволяет избежать локальных минимумов [6].

Основной параметр – температура, величина, влияющая на вероятность перехода системы в состояние с большей энергией.

* 1. **Генетические алгоритмы**

Генетические алгоритмы (Genetic algorithms, далее – GA) имитируют процессы передачи определенных свойств живых организмов от предков к потомкам. Свойства кодируются в хромосомах – векторах, содержащих численные значения свойств. Хромосомы принадлежат особям, вместе составляющим популяцию. Над особями внутри популяции (то есть, над их хромосомами) определены операции селекции, скрещивания и мутации.

Селекция – это процесс отбора ограниченного числа особей из популяции. Как правило, селекция производится в соответствии с функцией приспособленности (fitness-function), вычисляемой для каждой особи и зависящей от поставленной перед алгоритмом задачи.

Скрещивание необходимо, чтобы найти более эффективные решения и поддерживать разнообразие. Например, новая особь (ребенок) может быть получено путем объединения двух отдельных особей (родители).

Мутация также поддерживает разнообразие в популяции, внося случайные изменения в хромосомы отдельных особей.

* 1. **Поиск с запретами**

Основная идея метода поиска с запретами (Tabu search, далее – TS) состоит в наложении ограничений на поиск при помощи списка запретов (tabu list), избегая посещения одних и тех же решений и, таким образом, не останавливаясь в локальных минимумах. На каждой итерации генерируется полная окрестность текущего решения, из которой исключаются записи из списка запретов, а затем выбирается лучшее решение [2].

Список запретов хранит информацию об истории поиска в виде фиксированного числа посещенных решений. Решение может быть временно выведено из списка запретов, если оно соответствует некоторому уровню стремления (aspiration level), что в некоторых случаях позволяет переместить область поиска в новый регион.

* 1. **Муравьиный алгоритм**

Муравьиный алгоритм (Ant colony optimization, далее – ACO) использует модель поведения муравьёв, ищущих пути от колонии к источнику питания. Изначально решение достигается случайным образом, путям к решению присваивается максимальное значение параметра уровня феромона.

Феромон – параметр пути к решению, определяющий интенсивность, с которой «муравьи» используют данный путь. Со временем уровень феромона падает, а возрастает он только если «муравьи», пройдя по данному пути, добрались до решения. Таким образом, на более коротких путях к решению будет больший уровень феромона. Падение уровня феромона позволяет избежать локальных минимумов.

* 1. **«Жадный» адаптивный метод случайного поиска**

«Жадный» адаптивный метод случайного поиска (Greedy randomized adaptive search procedure, далее – GRASP) генерирует несколько решений с различными начальными параметрами, среди которых жадным алгоритмом отбираются лучшие, а затем производятся дальнейшие улучшения через локальный поиск. Полученные решения на текущей итерации служат основой для генерации решений на следующей.

* 1. **Метод роя частиц**

В методе роя частиц (Particle swarm optimization, далее – PSO), основанном на социальном поведении толпы, агентами являются частицы в пространстве параметров задачи оптимизации. В каждый момент времени (на каждой итерации) частицы имеют в этом пространстве некоторое положение и вектор скорости. Для каждого положения частицы вычисляется соответствующее значение целевой функции, и на этой основе по определенным правилам частица меняет свое положение и скорость в пространстве поиска [3].

1. **СРАВНЕНИЕ АЛГОРИТМОВ ОПТИМИЗАЦИИ СЕТИ И ВЫБОР МЕТОДА ОПТИМИЗАЦИИ**
   1. **Классификация по используемым подходам**

Одна из возможных классификаций приведенных выше алгоритмов [24] приведена в таблице 2.1.

Таблица 2.1 Классификация алгоритмов оптимизации [24].

|  |  |
| --- | --- |
| Алгоритм | Классификация |
| SA | M/S-N/I |
| GA | M/S-N/P |
| TS | A/N/I-P |
| ACO | M/S-N/P |
| GRASP | M/S-N/I |
| PSO | M/S-N/P |

В приведенной таблице, символы в правой колонке представляют:

“A” – использование адаптивной памяти; “M” – без использования памяти; “N” – использование особой окрестности; “S” – использование случайной выборки; “I” – использование итеративного подхода; “P” – использование популяционного подхода.

* 1. **Сравнение по производительности**

В работе [20] было произведено сравнение работы метаэвристических алгоритмов на примере задачи Изменчивости Времени Реакции (Response Time Variability, далее – RTV).

Проблема RTV – комбинаторная задача оптимизации, которая возникает при необходимости свести к минимуму разницу во времени при распределении каких-либо ресурсов. Обобщенная задача RTV представляется следующим образом: существует некоторая последовательность длиной *D*, состоящая из *n* различных символов, в которую символ *i* (*i*=1,...,*n*) должен попасть заданное число *di* раз (то есть, число раз, сколько символ *i* должен получить ресурс). Оптимальное решение проблемы RTV – такая последовательность, которая сводит к минимуму изменчивость расстояний между любыми двумя последовательными копиями одного и того же символа. Другими словами, расстояние между любыми двумя последовательными копиями одного и того же символа должно быть настолько регулярным, насколько это возможно (в идеале, постоянным) [13].

Для проведения исследования были составлены выборки из 60 учебных экземпляров для тонкой настройки параметров алгоритмов и 740 тестовых экземпляров. Эти экземпляры были сгруппированы в четыре класса (от CAT1 до CAT4 с 15 учебными экземплярами и 185 тестовыми экземплярами в каждом классе) в зависимости от их размера. Эти экземпляры были получены с использованием случайных значений *D* и *n*. Для всех экземпляров и для каждого символа *i*=1,...,*n*, случайная величина *di* (число копий символа *i*) составляет от 1 до ⌈(*D*-*n*+1)/2.5⌉ таким образом, что .

Все алгоритмы были написаны и запускались под Java 2 Platform Standard Edition (J2SE) версии 1.4.2.14.

В таблицах 2.2 и 2.3 приведены средние значения решений для задачи PTV, полученные алгоритмами за 50 и 1000 итераций, соответственно. Для случаев, когда существовал больше чем один алгоритм, основанный на одной метаэвристике, показаны результаты лучшего. В исследовании было замечено, что 1000 секунд достаточно для сходимости всех алгоритмов.

Таблица 2.2 Результаты для задачи RTV за 50 итераций [20].

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритм | В среднем | CAT1 | CAT2 | CAT3 | CAT4 |
| SA | 50.87 | 10.26 | 21.67 | 44.57 | 126.98 |
| GA | 186.94 | 11.65 | 29.41 | 84.54 | 622.16 |
| TS | 210.47 | 10.26 | 22.56 | 73.26 | 735.78 |
| ACO | 1651.48 | 10.92 | 36.83 | 504.84 | 6053.31 |
| GRASP | 2308.69 | 13.00 | 60.45 | 210.93 | 8890.37 |
| PSO | 4625.54 | 16.42 | 51.34 | 610.34 | 17824.04 |

Таблица 2.3 Результаты для задачи RTV за 1000 итераций [20].

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритм | В среднем | CAT1 | CAT2 | CAT3 | CAT4 |
| SA | 50.75 | 10.26 | 21.67 | 44.55 | 126.54 |
| GA | 106.68 | 10.92 | 27.00 | 74.86 | 313.92 |
| TS | 78.62 | 10.24 | 21.16 | 48.12 | 234.96 |
| ACO | 1208.81 | 10.46 | 31.17 | 337.31 | 4456.32 |
| GRASP | 301.90 | 11.56 | 50.45 | 227.50 | 918.10 |
| PSO | 1537.34 | 14.35 | 46.55 | 143.96 | 5944.51 |

Можно отметить большие различия между работой различных метаэвристик: худший средний результат (полученный с помощью алгоритма PSO) после 1000 секунд примерно в 30 раз больше, чем лучший средний результат (полученный с помощью алгоритма SA). При наличии достаточного количества времени для сходимости алгоритмов наблюдается тенденция: итеративные метаэвристики (GRASP, TS и SA) работают лучше, чем популяционные (GA, ACO и PSO). В частности, SA и TS превосходят лучшую популяционную метаэвристику (GA) в среднем на 52,43%, и 26,30% соответственно. Такое превосходство в производительности не зависит от размера экземпляров (от CAT1 до CAT4).

Если сравнить результаты, полученные с помощью 50 итераций (таблица 2.2) и 1000 итераций (таблица 2.3), можно увидеть, что SA также является методом, с наилучшей сходимостью. Результаты SA после 1000 секунд улучшились только на 0,24% по сравнению с результатами за 50 секунд.

* 1. **Сравнение достоинств и недостатков**

Основными критериями, влияющими на применимость того или иного алгоритма, можно назвать его изученность (в том числе, существующие усовершенствования) и сложность начальной настройки, от используемой модели. Сравнительный анализ [1] приведен в таблице 2.4.

Таблица 2.4 Достоинства и недостатки алгоритмов оптимизации [1].

|  |  |  |
| --- | --- | --- |
| Алгоритм | Достоинства | Недостатки |
| SA | Широко изучен | Сложность настройки параметров,  преждевременная  сходимость |
| GA | Широко изучен | Сложность настройки параметров,  Относительно медленная  сходимость |
| TS | Широко изучен | Сложность настройки параметров,  преждевременная  сходимость |
| ACO | Легкость настройки параметров | Неопределенное время сходимости |
| GRASP | Легкость настройки параметров | Нет учета истории  решений из предыдущих итераций |
| PSO | Легкость настройки параметров | Относительно медленная сходимость |

Несмотря на очевидную сложность настройки параметров, работа с широко изученными алгоритмами представляется более предпочтительной в связи с существующей большой теоретической и практической базой. Поэтому, в дальнейшем обзор алгоритмов будет сосредоточен на SA, GA и TS.

В некоторой степени работа SA, и полностью работа GA, в отличие от TS, зависит от эвристических знаний о проблемной области [28]. В различных сравнительных исследованиях на нейронных сетях для трех рассматриваемых алгоритмов, GA и TS попеременно показывают лучшие результаты. SA также справляется с поставленными задачами, хотя и с меньшим успехом [15, 25, 28]. Однако, при исследовании на задаче RTV SA, напротив, показал лучший результат среди всех метаэвристических алгоритмов. Такие результаты можно объяснить теоремой «Об отсутствии бесплатных завтраков» (No free lunch theorem, далее – NFL, 1996).

Согласно теореме NFL, разработанной Д.Волпертом и В.Макреди, производительности любой пары алгоритмов оптимизации эквивалентны, когда рассматриваются среди всех возможных задач. Иными словами, если алгоритм хорошо работает на определенном классе задач, то он обязательно платит за это с ухудшение характеристик на оставшемся множестве проблем, где лучше справляется другой алгоритм из рассматриваемой пары.

Однако эта теорема не рассматривает алгоритмы, зависящие от предметных знаний в рабочей области. Вопрос успешности таких алгоритмов заключается только в их оптимальной настройке. Это, а также наличие хорошей теоретической и практической базы по задаче оптимизации нейронных сетей, позволяет сделать выбор в пользу метода GA.

* 1. **Анализ выбранного метода оптимизации**

Генетические алгоритмы были разработаны исследовательской группой Джона Холланда в 1960-е и 1970-е годы. Холланд первым предложил использовать селекцию, скрещивание и мутацию в изучении адаптивных и искусственных систем. С тех пор множество вариантов GA было разработано и применено к широкому спектру задач оптимизации, от окраски графа до распознавания образов; для дискретных (например, задача коммивояжера) и непрерывных систем (например, задача создания эффективной конструкции аэродинамического профиля в авиационно-космической технике) [23].

GA, используемые для оптимизации нейронных сетей, относятся к подходу в машинном обучении, называемом «нейроэволюцией». Существует большое количество нейроэволюционных алгоритмов, которые делятся на две группы: алгоритмы, которые производят эволюцию весов при заданной топологии сети, и алгоритмы, которые помимо эволюции весов также производят эволюцию топологии сети. Сети, в которых производится эволюция как весов, так и топологии, называются TWEANNs (Topology & Weight Evolving Artificial Neural Networks). В данной работе предлагается расширить область применения нейроэволюционных алгоритмов на гиперпараметры нейронных сетей, включающих, среди прочего, и топологию. Далее GA будут рассматриваться применительно к оптимизации топологии нейронных сетей, как наиболее сложного в оптимизации и комплексного гиперпараметра.

* + 1. **Прямое и косвенное кодирование**

В первую очередь при выборе метода нейроэволюции необходимо определиться с методом генетического представления – схемы кодирования и соответствующего декодирования. Генетическое представление определяет класс сетей, которые могут быть построены с помощью выбранного метода. В настоящее время выделяют два основных класса генетического представления: прямое кодирование (direct encoding) и косвенное кодирование (indirect encoding) [8].

В **прямом кодировании** используется понятие хромосомы – линейного представления сети с явным указанием нейронов, весов и связей сети. Благодаря явному представлению, между структурными элементами (фенотипом) и соответствующими им участками хромосом (генотипом) всегда можно построить взаимно-однозначное соответствие [8].

Среди плюсов этого представления можно выделить его простоту и интуитивность, а также возможность применения к хромосомам генетических операторов (например, скрещивания и мутации). Из наиболее очевидных минусов выделяется рост генотипа при увеличении числа нейронов и связей в сети и вызываемую с этим низкую эффективность алгоритма за счет увеличения пространства поиска [8].

Из наиболее очевидных минусов такой схемы кодирования можно отметить рост размеров генотипа при увеличении количества нейронов и связей сети и, как следствие, низкую эффективность за счет значительного увеличения пространства поиска. Однако, для прямого кодирования предложены методики, целью которых является устранение данных недостатков.

**Косвенное кодирование** предполагает, что в генотипе кодируется не сам фенотип, а правила его построения (то есть, некая программа). Обычно, во время операции декодирования генотипа данные правила применяются рекурсивно и в зависимости от текущего контекста, в итоге образуя нейронную сеть, соответствующую фенотипу [8].

В случае с косвенным кодированием, генетическое представление и пространство поиска GA сужаются, но пропадает возможность отследить зависимость изменений фенотипа от изменений в генотипе. Кроме того, существуют сложности с подбором генетических операторов, сходимостью и производительностью [8].

Для поставленной задачи прямое кодирование – в силу своей простоты и изученности – представляется лучшим выбором.

* + 1. **NEAT**

Одной из наиболее потенциально успешных попыток избавиться от недостатков прямого кодирования с сохранением всех его достоинств является предложенный в 2002 году метод, под названием NEAT – Neural Evolution through Augmenting Topologies [26].

В своих исследованиях авторы выделили ряд ключевых проблем, свойственных прямому кодированию в частности и нейроэволюции вообще [8]. Эти проблемы:

* Конкурирующие представления (Competing Conventions) – один и тот же фенотип (топологически) сети может быть по-разному представлен в генотипе даже в рамках одного способа кодирования;
* Незащищенные инновации – при нейроэволюции инновации (т.е. изменения структуры сети) производятся добавлением или удалением нейронов и/или групп нейронов. Зачастую, добавление новой структуры в сеть ведет к тому, что значение её функции приспособленности снижается. Например, добавление нового нейрона вносит нелинейность в линейный процесс, что приводит к снижению значения функции приспособленности до тех пор, пока вес добавленного нейрона не оптимизируются;
* Начальные размер и топологические инновации – во многих методиках нейроэволюции начальная популяция является набором случайных топологий. Помимо того, что приходится тратить определенное время на отсеивание изначально нежизнеспособных сетей (например, таких, у которых ни один вход не соединен ни с одним выходом), такие популяции имеют тенденцию к преждевременной сходимости к решениям, размер которых неоптимален (т.е. слишком велик).

Предложенное авторами методики решение базируется на биологическом понятии гомологичных генов (алеллей – генов, расположенные в одинаковых участках парных хромосом и определяющих направление развития одного и того же признака), а также на существовании в природе процесса синапсиса – выравнивания гомологичных генов перед скрещиванием. В методике предполагается, что два гена (у двух разных особей) являются гомологичными, если они возникли в результате одной и той же мутации в прошлом. Другими словами, при каждой структурной мутации (добавление гена), новому гену присваивается уникальный маркер (innovation number), который затем не меняется в процессе эволюции [8].

Использование маркеров положено в основу решения всех трех описанных выше задач, за счет [8]:

* Выполнения скрещивания только между гомологичными генами;
* Защиты инноваций за счет того, что особи, имеющие близкие топологические структуры, отсеиваются, таким образом оставляя место для «новичков»;
* Минимизации размерности за счет последовательного роста от минимального размера.
  + 1. **Параметры генетических алгоритмов**

Кроме схемы кодирования, при составлении GA необходимо определиться с функцией приспособленности, начальной популяцией и другими параметрами. Обычно, **функция приспособленности** определяется через целевую функцию путем ее инвертирования. В случае с нейронными сетями целевую функцию представляется рациональным выразить через величину среднеквадратичной ошибки обученной сети на контрольном множестве.

**Начальная популяция** хромосом может формироваться как случайным образом, так и самонаводящимися способами [6]. Размер популяции следует выбирать пропорционально числу оптимизируемых параметров.

Для лучшего понимания механизмов GA следует подробнее рассмотреть этапы селекции, скрещивания и мутации.

На этапе **селекции** из всей популяции выбирается некоторая доля, над которой будут производиться остальные преобразования. Попадание особи в селективную долю зависит от значения ее функции приспособленности. Размер доли является одним из параметров GA и задается до начала функционирования алгоритма.

Обобщенная процедура отбора может быть реализована следующим образом:

1. Для каждой особи вычисляется ее функция приспособленности. Затем происходит нормализация значений функции для всех особей в популяции таким образом, чтобы сумма всех значений была равна 1.
2. Особи сортируются по убыванию значения нормализованной функции приспособленности.
3. Вычисляется накопленное нормализованное значение функции приспособленности каждой особи как сумма значения ее собственной нормализованной функции и значений функции всех особей в сортированном списке до нее. Накопленное нормализованное значение последней особи в списке, таким образом, должно быть равным 1.
4. Выбирается случайное число R от 0 до 1.
5. Выбирается первая особь, чье накопленное нормализованное значение функции приспособленности больше R.

**Скрещивание** в разных алгоритмах определяется по-разному, так как зависит от формата представляемых в хромосоме данных. Главное требование – чтобы потомок имел возможность унаследовать черты обоих родителей. Чтобы избежать возможного недостатка разнообразия среди потомков, к скрещиванию представляются не только особи, успешно прошедшие этап селекции, но и часть особей из остальной части популяции.

Для хромосом, кодирующих топологию нейронных сетей, особенно сложным представляется определение операции скрещивания. В связи с этим, в дальнейшем предполагается определить кодирование топологии в хромосомах таким образом, чтобы максимально упростить их скрещивание.

**Мутация** используется для поддержания генетического разнообразия в поколениях. Она изменяет одно или несколько значений генов в хромосоме. Таким образом возможен случайный переход к лучшему решению. Мутация происходит в процессе эволюции в соответствии с изначально задаваемым параметром вероятности мутации. Эта вероятность должна быть установлена достаточно низко, так как при высоких параметрах GA переходит в форму алгоритма случайного поиска.

Для хромосом, кодирующих топологию нейронных сетей, мутация представляет собой добавление или удаление связей и/или нейронов.

Еще одной проблемой, решаемой при разработке GA для оптимизации топологии нейронных сетей, представляется выбор в пользу редукции или наращивания размеров сети. Считается, что наиболее эффективным является поиск с последовательным увеличением размера нейронной сети – в этом случае пространство поиска сильно сужается. Однако, для сетей с уже определенной структурой применение такого подхода может отрицательно сказаться на нахождении оптимальной топологии сети.

Одним из способов решения этой задачи является введение **штрафной функции**, которая уменьшает значение функции приспособленности в зависимости от размера сети. Комбинируя редукцию и наращивание топологии в одном алгоритме предполагается достичь большей пластичности его работы для различных задач.

* + 1. **Проблема переобучения**

Одна из самых серьезных трудностей при построении нейроэволюционного алгоритма состоит в явлении переобучения [6, 12]. Нейронная сеть, сформированная в ходе работы алгоритма, должна быть способна обобщать результаты работы не только на обучающем множестве, но и на всем возможном пространстве данных. Однако зачастую возможен сценарий, при котором данная топология сети успешно работает только на тех данных, на которых была обучена, и для других данных оптимальной будет уже другая топология.

Одним из методов борьбы с переобучением является разбиение всех данных на два множества: обучающее, на котором происходит обучение сети и контрольное (тестовое), на котором проверяется качество распознавания сети. Предполагается разделять данные на эти множества в соотношении 70:30.

Самый простой метод предупреждения переобучения состоит в остановке обучения при возрастании ошибки на контрольном множестве, которая при «полезном» обучении должна убывать.

В ходе исследования в рамках данной работы был сделан выбор в пользу генетических алгоритмов среди прочих методов решения задачи оптимизации топологии нейронной сети.

В силу простоты и изученности, для кодирования топологии нейронной сети в хромосомах было выбрано прямое кодирование.

Функцию приспособленности, выражаемую через инвертированную целевую функцию, предполагается определить через величину среднеквадратичной ошибки обученной сети на контрольном множестве.

При инициализации работы алгоритма оптимизации среди прочих параметров предполагается определить размер популяции, размер селективной доли и параметр вероятности мутации.

Кодирование топологии в хромосомах предполагается определить таким образом, чтобы максимально упростить их скрещивание. Операцию мутации представить как добавление или удаление слоев, связей и нейронов.

При построении алгоритма также необходимо учитывать вероятность переобучения сети и избегать его путем разбиения обучающих данных на несколько множеств.

1. **АНАЛИЗ ДОСТУПНЫХ БИБЛИОТЕК ПО НЕЙРОННЫМ СЕТЯМ ДЛЯ ЯЗЫКА JAVA**

На данный момент существует широкий спектр API для создания и экспериментов с нейронными сетями. В процессе анализа из [29] были отобраны библиотеки, поддерживающие программирование на языке Java: Encog и Neuroph. Обе библиотеки распространяются по свободной лицензии (Free and Open-Source Software, FOSS) с открытым API.

* 1. **Encog**

API Encog включает в себя множество классов, реализующих алгоритмы распознавания, кластеризацию, генетические алгоритмы, скрытые марковские модели, байесовские сети и др. Среди поддерживаемых архитектур нейронных сетей – машина Больцмана, нейронные сети обратного распространения, рекуррентные сети Эльмана, рекуррентные сети Джордана, самоорганизующиеся карты Кохонена и т.д. Доступны следующие функции активации нейронов: биполярная, соревновательная, функция Элиотта, функция Гаусса, гиперболический тангенс, линейная, синусоидальная, сигмоида. Также, для Encog доступно несколько руководств по использованию [16, 17, 30].

Для создания сетей со свободной конфигурацией предусмотрен класс FreeformNetwork. Однако, при детальном анализе была выявлена сложность организации сети с помощью данного класса (так, например, в объекте FreeformNetwork невозможно напрямую обратиться к содержащимся в нем слоям или нейронам). Тем не менее, в пакете Encog можно легко создавать сети со стандартными архитектурами.

* 1. **Neuroph**

API Neuroph предоставляет библиотеку классов Java, реализующих основные понятия в терминологии нейронных сетей, такие как: искусственный нейрон, слой, соединение нейронов, вес, передаточные функции, обучение и т.д. Среди поддерживаемых нейросетевых архитектур – многослойный персептрон, нейронные сети обратного распространения, самоорганизующиеся карты Кохонена, сеть Хопфилда и т.д. Все эти классы могут быть расширены и настроены для создания пользовательских нейронных сетей и правила обучения [31].

Для создания сетей со свободной конфигурацией предусмотрен класс NeuralNetwork. Для детальной настройки архитектуры сети доступны статические функции класса ConnectionFactory, позволяющие соединять друг с другом отдельные нейроны, нейроны внутри одного слоя или целые слои.

Neuroph больше сфокусирован на архитектурах и параметрах нейронных сетей, чем Encog, который взамен предоставляет большой выбор методов обучения, инструментов анализа и прочих утилит.

* 1. **Сравнение API библиотек на практике**

Для того, чтобы понять на практике, какая из двух библиотек лучше подходит для разработки алгоритма оптимизации нейронных сетей методом GA, было решено создать с использованием API каждой из библиотек простую нейронную сеть для решения задачи XOR и определить для этой сети ее функцию приспособленности.

Суть задачи XOR для нейронной сети в следующем: дана логическая функция XOR (исключающее ИЛИ) – функция от двух аргументов, каждый из которых может принимать значение 0 или 1. Когда значения аргументов совпадают, функция принимает значение 0, иначе – 1.

Обозначим один вход сети через x1, а другой через x2, тогда все их возможные комбинации будут состоять из четырех точек на плоскости. Таблица 3.1 показывает требуемую связь между входами и выходом, где входные комбинации, которые должны давать нулевой выход, помечены A0 и A1, единичный выход – B0 и B1.

Таблица 3.1 Связь между входами и выходами в XOR-задаче.

|  |  |  |  |
| --- | --- | --- | --- |
| Точки | Значение x1 | Значение x­2 | Требуемый выход |
| A0 | 0 | 0 | 0 |
| B0 | 1 | 0 | 1 |
| A1 | 0 | 1 | 1 |
| B1 | 1 | 1 | 0 |

Реализовать функцию XOR на нейронной сети можно различными способами, и для текущей задачи была выбрана архитектура, представленная на рисунке 3.1.

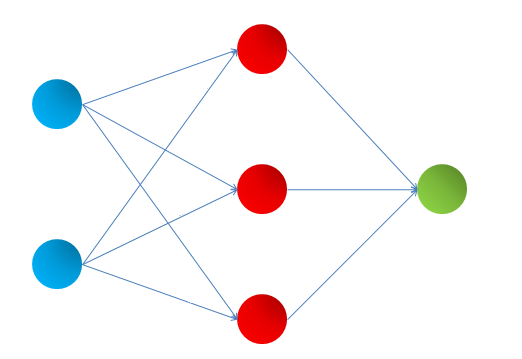


Рисунок 3.1 – Сеть для решения задачи XOR

Данные, представленные в таблице 3.1, на языке Java были помещены в двумерные массивы:

public static double XOR\_INPUT[][] = {{0.0,0.0},{1.0,0.0},{0.0,1.0},{1.0,1.0}};

public static double XOR\_IDEAL[][] = {{0.0},{1.0},{1.0},{0.0}};

Для реализации предотвращения переобучения сети была также создана контрольная выборка:

public static double CONTROL\_INPUT[][] = {{1.0,1.0},{1.0,0.0},{0.0,0.0},{0.0,1.0}};

public static double CONTROL\_IDEAL[][] = {{0.0},{1.0},{0.0},{1.0}};

Сеть обучалась с точностью (величиной ошибки) ε=0,001.

* + 1. **Основная часть программы**

1. Encog

Класс MLDataSet отвечает за данные для обучения/функционирования сети.

Функция ConnectLayers позволяет полносвязно соединить между собой слои сети типа FreeformNetwork, используя активационную функцию (в данном случае – сигмоида).

getFitness – пользовательская функция, вычисляющая функцию приспособленности сети.

MLDataSet trainingSet = new BasicMLDataSet(XOR\_INPUT, XOR\_IDEAL);

MLDataSet controlSet = new BasicMLDataSet(CONTROL\_INPUT,CONTROL\_IDEAL);

FreeformNetwork net = new FreeformNetwork();

FreeformLayer hidL = net.createLayer(3);

net.ConnectLayers(net.createInputLayer(2), hidL, new ActivationSigmoid());

net.ConnectLayers(hidL, net.createOutputLayer(1), new ActivationSigmoid());

double fitError = 0.001;

System.out.println(getFitness(net, trainingSet, controlSet, fitError));

1. Neuroph

Класс DataSet отвечает за данные для обучения сети. Объект данного класса состоит из нескольких столбцов, каждый из которых представляет собой один элемент обучающей выборки.

Класс данных для функционирования сети не предусмотрен. При функционировании, на вход сети подается только входной вектор. Это связано с отсутствием реализации функции вычисления ошибки функционирования сети, которая, однако, реализована в Encog (описана в п. 4.3.3).

Класс NeuralNetwork представляет сеть со свободной конфигурацией.

Функция addLayer позволяет добавить к сети слои с заданным числом нейронов с определенным типом, активационной (в данном случае – взвешенная сумма) и (функционально) передаточной функцией (в данном случае – сигмоида).

Статическая функция ConnectionFactory.fullConnect полносвязно соединяет между собой слои сети.

Функции setInputNeurons и setOutputNeurons определяют нейроны, отвечающие за входы и выходы сети соответственно.

getFitness – пользовательская функция, вычисляющая функцию приспособленности сети.

DataSet trainingSet = new DataSet(2,1);

for(int i=0; i<4; i++)

trainingSet.addRow(XOR\_INPUT[i], XOR\_IDEAL[i]);

NeuralNetwork net = new NeuralNetwork();

net.addLayer(0, new Layer(2,new NeuronProperties(InputNeuron.class, Linear.class)));

net.addLayer(1, new Layer(3,new NeuronProperties(Neuron.class, WeightedSum.class, Sigmoid.class)));

net.addLayer(2, new Layer(1,new NeuronProperties(Neuron.class, WeightedSum.class, Sigmoid.class)));

ConnectionFactory.fullConnect(net.getLayerAt(0), net.getLayerAt(1));

ConnectionFactory.fullConnect(net.getLayerAt(1), net.getLayerAt(2));

net.setInputNeurons(net.getLayerAt(0).getNeurons());

net.setOutputNeurons(net.getLayerAt(2).getNeurons());

double fitError = 0.001;

System.out.println(getFitness(net, trainingSet, CONTROL\_INPUT, CONTROL\_IDEAL, fitError));

* + 1. **Функция приспособленности**

1. Encog

Функция reset устанавливает случайные значения весов в сети.

Статическая функция EncogUtility.trainToError обучает сеть на некотором обучающем наборе данных до соответствия некоторой величине ошибки.

Функция calculateError включена в API библиотеки Encog и определяет ошибку сети на некотором наборе данных (описана в п. 4.3.3).

public static double getFitness(FreeformNetwork net, MLDataSet trainingSet,

MLDataSet controlSet, double fitError){

net.reset();

EncogUtility.trainToError(net,trainingSet,fitError);

return -1\*net.calculateError(controlSet);

}

1. Neuroph

Функция reset устанавливает случайные значения весов в сети.

Перед обучением сети функцией setLearningRule для нее устанавливается обучающая дисциплина (в данном случае – обратное распространение ошибки).

Функция learn обучает сеть на некотором обучающем наборе данных до соответствия некоторой величине ошибки.

Функция calculateError – пользовательская функция, вычисляющая ошибку функционирования сети на некотором наборе данных (аналогично подобной функции в API Encog).

public static double getFitness(NeuralNetwork net,DataSet trainingSet,

double[][] controlInput, double[][] controlIdeal,double fitError){

net.reset();

BackPropagation train = new BackPropagation();

net.setLearningRule(train);

train.learn(trainingSet,fitError);

return -1\*calculateError(net,controlInput,controlIdeal);

}

* + 1. **Функция вычисления ошибки функционирования**

Ошибка функционирования вычислялась методом среднеквадратичной ошибки (Mean squared error, MSE).

1. Encog

Функция API Encog calculateError может определять ошибку несколькими различными методами, но в для FreeformNetwork она вычислялась через MSE.

В процессе работы, функция calculateError вызывает статическую функцию EncogUtility.calculateRegressionError, перенаправляя ей входной набор данных.

Основная работа этой функции заключается в накоплении общей квадратичной ошибки для всего входного набора данных функцией errorCalculation.updateError и в вычислении результирующей ошибки функцией calculate, которая возвращает разный результат в зависимости от используемого метода вычисления ошибки (в данном случае, использовалась функция errorCalculation.calculateMSE).

Фрагмент кода EncogUtility.calculateRegressionError:

for (final MLDataPair pair : data) {

final MLData actual = method.compute(pair.getInput());

errorCalculation.updateError(actual.getData(),

pair.getIdeal().getData(), pair.getSignificance());

}

return errorCalculation.calculate();

Функция updateError:

public final void updateError(final double[] actual, final double[] ideal, final double significance) {

for (int i = 0; i < actual.length; i++) {

double delta = (ideal[i] - actual[i]) \* significance;

this.globalError += delta \* delta;

}

this.setSize += ideal.length;

}

Функция calculateMSE:

public final double calculateMSE() {

if (this.setSize == 0) {

return 0;

}

final double err = this.globalError / this.setSize;

return err;

}

1. Neuroph

Для вычисления ошибки функционирования в Neuroph была разработана пользовательская статическая функция, вычисляющая ошибку функционирования сети на входном множестве controlInput при известном эталонном выходном множестве controlIdeal.

Аналогично с подобной функцией в Encog, сначала вычисляется накопленная квадратичная ошибка на всем входном множестве, а затем вычисляется средняя ошибка, согласно методу MSE.

Функция setInput передает на вход сети один элемент из входной выборки.

Функция calculate производит расчет выходного вектора сети.

Функция getOutput возвращает выходной вектор сети.

Math.pow – функция возведения в степень.

public static double calculateError(NeuralNetwork net, double[][] controlInput, double[][] controlIdeal) {

double error=0;

for(int i=0; i<controlInput.length; i++){

net.setInput(controlInput[i]);

net.calculate();

for(int j=0; j<controlIdeal[0].length; j++){

double delta = controlIdeal[i][j] - net.getOutput()[j];

error += Math.pow(delta,2);

}

}

return error/controlIdeal.length;

}

* + 1. **Сравнение функционирования реализаций**

С помощью отладчика были отслежены этапы вычисления ошибки в обеих API (выходное значение сети для каждого элемента входного множества). Значения представлены в таблице 3.2.

Таблица 3.2 Выходные значения сети и MSE для двух API.

|  |  |  |
| --- | --- | --- |
| Эталон | Encog | Neuroph |
| 0 | 0.0261 | 0.0421 |
| 1 | 0.9889 | 0.9509 |
| 0 | 0.0103 | 0.0435 |
| 1 | 0.9603 | 0.9564 |
| MSE | 0.0006 | 0.0019 |

Проанализировав таблицу 3.2, можно сделать вывод о том, что при решении данной задачи оба API показывают сравнительно одинаковые результаты.

По завершению функционирования программ, в консоль были выведены следующие записи:

1. Encog

Beginning training...

Iteration #1 Error:34,859721% Target Error: 0,100000%

Iteration #2 Error:31,451479% Target Error: 0,100000%

Iteration #3 Error:27,859738% Target Error: 0,100000%

Iteration #4 Error:25,381611% Target Error: 0,100000%

Iteration #5 Error:25,609355% Target Error: 0,100000%

Iteration #6 Error:25,373412% Target Error: 0,100000%

Iteration #7 Error:25,066964% Target Error: 0,100000%

Iteration #8 Error:25,366392% Target Error: 0,100000%

Iteration #9 Error:25,093458% Target Error: 0,100000%

Iteration #10 Error:25,048605% Target Error: 0,100000%

Iteration #11 Error:25,073907% Target Error: 0,100000%

Iteration #12 Error:24,998326% Target Error: 0,100000%

Iteration #13 Error:24,991779% Target Error: 0,100000%

Iteration #14 Error:24,955910% Target Error: 0,100000%

Iteration #15 Error:24,916651% Target Error: 0,100000%

Iteration #16 Error:24,882470% Target Error: 0,100000%

Iteration #17 Error:24,837190% Target Error: 0,100000%

Iteration #18 Error:24,793119% Target Error: 0,100000%

Iteration #19 Error:24,756450% Target Error: 0,100000%

Iteration #20 Error:24,758587% Target Error: 0,100000%

Iteration #21 Error:24,731683% Target Error: 0,100000%

Iteration #22 Error:24,693043% Target Error: 0,100000%

Iteration #23 Error:24,653403% Target Error: 0,100000%

Iteration #24 Error:24,608278% Target Error: 0,100000%

Iteration #25 Error:24,538675% Target Error: 0,100000%

Iteration #26 Error:24,447640% Target Error: 0,100000%

Iteration #27 Error:24,327788% Target Error: 0,100000%

Iteration #28 Error:24,169906% Target Error: 0,100000%

Iteration #29 Error:24,001194% Target Error: 0,100000%

Iteration #30 Error:23,757193% Target Error: 0,100000%

Iteration #31 Error:23,429482% Target Error: 0,100000%

Iteration #32 Error:23,044495% Target Error: 0,100000%

Iteration #33 Error:22,512774% Target Error: 0,100000%

Iteration #34 Error:21,878929% Target Error: 0,100000%

Iteration #35 Error:21,037288% Target Error: 0,100000%

Iteration #36 Error:19,995521% Target Error: 0,100000%

Iteration #37 Error:18,726264% Target Error: 0,100000%

Iteration #38 Error:17,211631% Target Error: 0,100000%

Iteration #39 Error:15,416086% Target Error: 0,100000%

Iteration #40 Error:13,451055% Target Error: 0,100000%

Iteration #41 Error:11,813896% Target Error: 0,100000%

Iteration #42 Error:9,598312% Target Error: 0,100000%

Iteration #43 Error:7,825145% Target Error: 0,100000%

Iteration #44 Error:5,909835% Target Error: 0,100000%

Iteration #45 Error:4,185429% Target Error: 0,100000%

Iteration #46 Error:2,662614% Target Error: 0,100000%

Iteration #47 Error:1,721158% Target Error: 0,100000%

Iteration #48 Error:0,853372% Target Error: 0,100000%

Iteration #49 Error:2,542489% Target Error: 0,100000%

Iteration #50 Error:0,961227% Target Error: 0,100000%

Iteration #51 Error:0,623372% Target Error: 0,100000%

Iteration #52 Error:0,383361% Target Error: 0,100000%

Iteration #53 Error:0,223346% Target Error: 0,100000%

Iteration #54 Error:0,115161% Target Error: 0,100000%

Iteration #55 Error:0,052906% Target Error: 0,100000%

-6.368011851861356E-4

1. Neuroph

-0.0019113032724244838

Видно, что Encog представляет подробную информацию об обучении сети. Однако, это может стать помехой при анализе выводимых данных от других функций, так как информация, выводимая API Encog, занимает большой объем.

* + 1. **Изменение структуры сети**

Попытка изменить структуру сети (а именно удалить связь между одним из нейронов скрытого слоя и нейроном выходного слоя, как на рисунке 3.2) в Encog не удалась. Во всем многообразии функций API не было найдено функции, позволяющей удалить связь между двумя нейронами в сети типа FreeformNetwork.

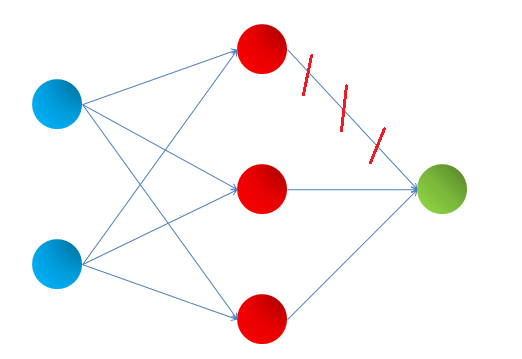


Рисунок 3.2 – Сеть для решения задачи XOR с удаленной связью

В случае с API Neuroph, функция removeOutputConnectionTo позволяет удалить связь между конкретными нейронами.

Такая сеть по-прежнему будет способна решить XOR-задачу, и в пакете Neuroph успешно ее решает:

Функция приспособленности до удаления связи:

-0.001811299482848926

Функция приспособленности после удаления связи:

-0.0018112912797263261

При решении задачи XOR среди результатов для API Encog и Neuroph не было замечено существенных различий. При этом, API Encog выводит подробную статистику при каждом обучении сети. В некоторых случаях это может быть полезно, однако при реализации GA может помешать обработке данных от пользовательских функций.

Было также выяснено, что из-за широкой специализации в Encog существенно усложнен процесс создания сети со свободной архитектурой по сравнению с Neuroph, API которой был специально разработан чтобы поддерживать подобные задачи.

Таким образом, для реализации GA с целью оптимизации топологии нейронной сети очевидно больше подходит библиотека Neuroph.

1. **ОБЗОР СУЩЕСТВУЮЩИХ ГИПЕРПАРАМЕТРОВ НЕЙРОННЫХ СЕТЕЙ И ВЫБОР ГИПЕРПАРАМЕТРОВ ДЛЯ ОПТИМИЗАЦИИ**

**Гиперпараметр** некоторого алгоритма обучения 'A' определяется как переменная величина, которая должна быть определена до фактического применения 'A' к данным и оставаться постоянной на протяжении всего применения 'A', и которая не выбирается непосредственно самим алгоритмом обучения [14, 11].

Гиперпараметры нейронных сетей могут определять либо процесс обучения – например, коэффициент скорости обучения (learning rate), – либо архитектуру сети – например, количество скрытых слоев. Нейронные сети обычно содержат более десяти гиперпараметров, что делает их оптимизацию сложной задачей [18].

В связи с эти большинство опубликованных работ по нейронным сетям используют стандартные эвристики, такие как ручной или случайный поиск, а иногда и вовсе не оптимизируют гиперпараметры. Также, одной из причин того, что оптимизации гиперпараметров в этих работах не было уделено должное внимание, является высокая временная сложность данной задачи в процессе функционирования сети. Однако, эффективная оптимизация может позволить найти такие гиперпараметры, что суммарное время их поиска и дальнейшего функционирования будет меньше, чем общее время функционирования сети без оптимизации [18].

В [19] гиперпараметры оптимизировали путем изменения одного гиперпараметра за раз при сохранении остальных гиперпараметров фиксированными. Оказалось, что оптимизация каждого гиперпараметра по отдельности может привести к локально-оптимальному выбору гиперпараметров. Глобально-оптимальный выбор зависит от подбора всех гиперпараметров. Таким образом, рациональнее будет оптимизировать все гиперпараметры единовременно.

Теоретически, можно включить подбор гиперпараметров в алгоритм обучения более высокого уровня (алгоритм ‘B’ подбирает гиперпараметры алгоритма ‘A’). Тогда, если алгоритм ‘B’ не имеет собственных гиперпараметров, то композиция ‘B’ над ‘A’ может быть названа «чистым» алгоритмом обучения.

Так или иначе, численные значения гиперпараметров должны быть подобраны вручную или настроены с помощью алгоритма перед началом обучения. Для любого гиперпараметра, который влияет на эффективность обучения, имеет смысл выбирать его значение на основе данных вне выборки [11].

Далее будут рассмотрены некоторые из существующих гиперпараметров для сетей с иерархической структурой связей, обучающихся по методу обратного распространения ошибки.

* 1. **Топология сети**
     1. **Число скрытых слоев**

Для многих задач наличие даже одного скрытого слоя уже дает хорошие результаты. Фактически было доказано, что многослойный персептрон с одним скрытым слоем может моделировать даже самые сложные функции, если у него достаточно нейронов. Долгое время это удерживало исследователей от исследования более глубоких сетей. Однако, сети с бо́льшим числом скрытых слоев имеют гораздо более высокую эффективность: они могут моделировать сложные функции, используя экспоненциально меньшее количество нейронов, чем неглубокие сети, что значительно ускоряет их обучение [14].

* + 1. **Число нейронов в скрытых слоях**

Большое число нейронов в скрытых слоях может серьезно улучшить способность сети к обучению. Большие, чем оптимальные значения, как правило, не сильно ухудшают производительность обобщения, но требуют пропорционального увеличения вычислений (в при масштабировании всех слоев одновременно в полносвязной архитектуре). Использование одинакового числа нейронов для всех скрытых слоев обычно работает лучше или как минимум не хуже слоев с различным количеством нейронов (например, для случаев уменьшения размера (пирамидально) или увеличения размера (перевернутая пирамида)). Кроме того, обычно положительный эффект дает использование первого скрытого слоя большего по размеру, чем входной [11].

* + 1. **Связи между нейронами**

Не всегда имеет смысл использовать полносвязные нейронные сети. Число соединений (активных весов) в сети может быть как уменьшено, так и увеличено. Уменьшение числа соединений между нейронами иногда называют «обрезкой» сети.

На рис. 4.1 показано, сколько связей может быть удалено перед тем, как производительность сети будет существенно ухудшена на примере простой сети для распознавания символов алфавита [21].

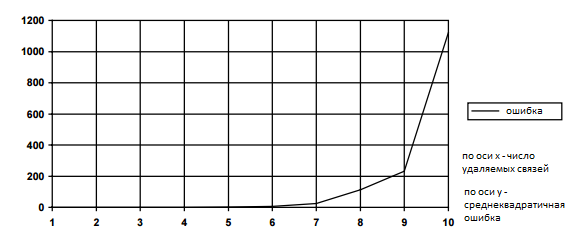


Рисунок 4.1 – Эффект удаления связей для нейронов в одном слое

Значимое ухудшение наблюдается только после удаления более 200 связей.

* 1. **Параметры активационной функции**

Активационная функция является основным вычислительным элементом нейронной сети и играет важную роль в достижении наилучшей производительности вычислений. Активационная функция варьирует входные значения нейрона в пределах от 0 до 1 или от 1 до -1 в зависимости от типа используемой функции.

Наиболее распространенным типом передаточной функции активации является сигмоида. Сигмоида – это дифференцируемая функция, определяемая как: , где y - значение сигмоиды, k - коэффициент крутизны сигмоиды, а x - входные значения. Единица в знаменателе добавляется как пороговая величина, чтобы функция принимала значения от 0 до 1.

В исследовании [27] изучалось влияние значения коэффициента крутизны сигмоиды k на функционирование многослойной нейронной сети обратного распространения ошибки.

Всего было использовано 10 сигмоид с различными коэффициентами для нейронов выходного и скрытых слоев. Коэффициент крутизны варьировался от 0.025 до 1. Использовались пять различных архитектур сетей с шестью наборами данных.

Для всех архитектур и наборов данных обучение с коэффициентами, отличными от единицы, как правило, имело лучшие результаты по сравнению со стандартным единичным коэффициентом. Результаты исследования показали, что различные архитектуры нейронных сетей влияют на эффективность обучения значительно меньше, чем выбор оптимального коэффициента крутизны.

В данной работе представляется возможным также оптимизировать коэффициент крутизны линейной функции , использующейся в нейронах входного слоя, таким образом вместе с коэффициентами сигмоидальных функций нейронов выходного и скрытых слоев задействовав в оптимизации коэффициенты активационных функций всех нейронов в сети.

* 1. **Коэффициент скорости обучения**

В методе обратного распространения ошибки для коррекции весов сети используется дельта-правило, подразумевающее значение коррекции: , где – входные значения, – коэффициент скорости обучения, – разница между желаемым и реальным выходными значениями [6].

Выбор скорости обучения значительно влияет на эффективность обучения: при больших значениях скорость обучения растет, но увеличивается и величина ошибки обучения. Для нейронных сетей со входами или входами, ограниченными на [0,1], η обычно устанавливается между 10-6 и 1 [11].

* 1. **Момент**

Одной из возможных модификаций метода обратного распространения ошибки является добавление момента (или импульса) к функции коррекции весовых коэффициентов: .

Составляющая момента зависит от коррекции на предыдущем шаге, придавая инерционности процессу обучения. Чем больше значение момента, тем больше новая коррекция веса зависит от предыдущей.

Введение момента уменьшает колебания вычислений, поддерживая одно и то же направление подстройки весов. Однако, при слишком больших значениях момента, появляется возможность «проскочить» оптимальное значение функции ошибки [7].

* 1. **Диапазон случайной инициализации весов**

Веса должны быть инициализированы тщательно, чтобы не допустить симметрии среди нейронов одного слоя. Здесь под симметрией понимается ситуация, когда скрытые нейроны одного и того же слоя используют одни и те же входные и выходные веса, вычисляя один и тот же результат и, следовательно, работая впустую.

Данное требование нарушения симметрии не касается выходных весов (для нейронов в выходном слое), которые поэтому могут быть инициализированы нулевыми значениями.

В качестве гиперпараметра можно ввести масштабный коэффициент r для диапазона инициализации. Эта идея основана на том, что нейроны с бо́льшим числом входов должны иметь меньшие веса.

Если взять за диапазон инициализации интервал и ввести обозначение числа выходов нейрона , то для определения величины можно выделить несколько методов [11]:

* для нейронов в общем случае;
* для нейронов с функцией активации гиперболического тангенса;
* для нейронов с сигмоидальной функцией активации.

Данный гиперпараметр, однако, согласно [11] не нуждается в оптимизации при условии использования вышеописанных формул.

1. **РАЗРАБОТКА АЛГОРИТМА ОПТИМИЗАЦИИ**

Алгоритм оптимизации разрабатывался в соответствии с общими принципами построения генетических алгоритмов [8].

Обобщенный алгоритм:

1. В программу вводится структурная схема нейронной сети.
2. Задаются параметры, необходимые для функционирования алгоритма:
   * Допустимая ошибка обучения;
   * Максимальное число итераций обучения;
   * Размер популяции N;
   * Число поколений NG.
3. Создается первая популяция размером N, содержащая начальную нейронную сеть и N-1 нейронных сетей (далее – особей), полученных в результате применения операции мутации к начальной топологии нейронной сети.
4. Для особей в поколении считаются значения функции приспособленности.
5. Особи сортируются по вычисленным значениям функции приспособленности.
6. Создается новое поколение размером N, куда без изменений переводится особь с лучшим значением функции приспособленности.
7. Оставшиеся N-1 особей в случайном порядке попарно скрещиваются, полученные в результате скрещивания потомки проходят через операцию мутации и после переходят в новое поколение.
8. Пункты 4-7 повторяются, пока не будет достигнуто заданное число поколений NG.
9. Так как после прохождения 5 пункта в поколении NG особи уже отсортированы, первая по порядку особь в данном поколении будет являться оптимизированной версией начальной нейронной сети.

Структурная схема обобщенного алгоритма оптимизации изображена на рисунке 5.1.

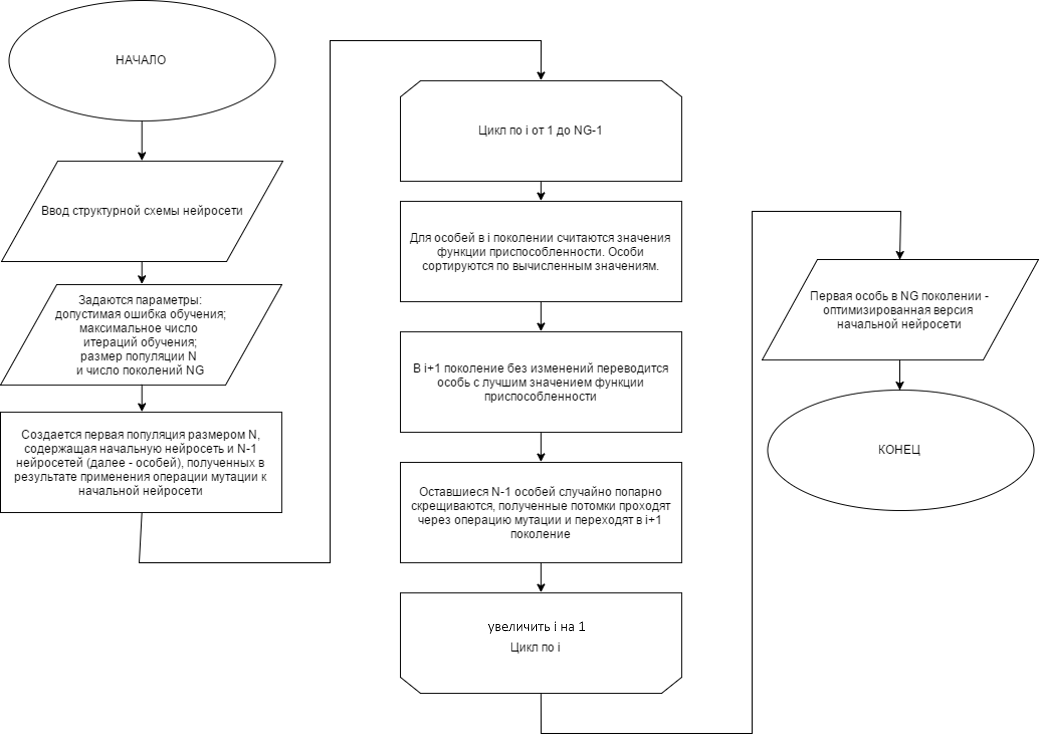


Рисунок 5.1 – Структурная схема обобщенного алгоритма оптимизации

При более детальной разработке алгоритма необходимо определиться со способом кодирования топологии нейронной сети внутри программы, а затем, основываясь на ней, с алгоритмами мутаций, скрещивания и вычисления функции приспособленности.

* 1. **Кодирование топологии сети**

Согласно заданию, максимально возможное число скрытых слоев в оптимизируемых сетях – 10, а максимально возможное число нейронов в слоях – 30. Отталкиваясь от этого, был предложен способ кодирования топологии сети в трехмерном массиве.

В языке Java многомерные массивы реализованы как массивы ссылок: все индексы, кроме последнего, содержат массивы ссылок на объекты по следующему индексу [9].

Топология будет закодирована в логическом массиве вида: boolean[10][31][31]. Первый индекс массива соответствует номеру слоя сети, второй индекс – номеру нейрона в данном слое, третий индекс – связям данного нейрона с нейронами из предыдущего слоя. Второй и третий индекс больше максимального числа нейронов в слое на единицу в связи с необходимостью кодирования «использования» слоя и «использования» нейрона в слое. Если слой или нейрон закодированы как не использующиеся, они не будут учитываться при преобразовании генотипа в фенотип – то есть, при построении нейронной сети по закодированной в массиве топологии.

Элементы топологии кодируются в массиве следующим образом:

* Слой n: boolean[n][0][0]. Если значение ячейки равно единице, слой используется, иначе – не используется.
* Нейрон m в слое n: boolean[n][m][0]. Если значение ячейки равно единице, нейрон используется, иначе – не используется.
* Связь нейрона m в слое n с нейроном k в слое n-1: boolean[n][m][k]. Если значение ячейки равно единице, связь существует, иначе – нет.

Так как последний индекс указывает на наличие связи с предыдущем слоем, в массиве кодируются данные только о скрытых и выходном слоях. Информацию о размере входного слоя предполагается получать из информации о размере входного вектора обучающей выборки.

Пример кодирования топологии для некоторой нейронной сети представлен на рисунке 5.2.

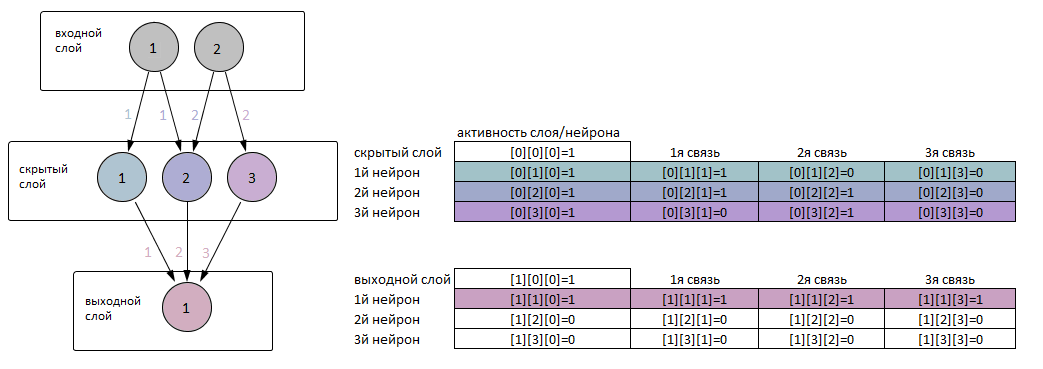


Рисунок 5.2 – Пример кодирования топологии

Размерность массива в примере была сокращена до boolean[2][4][4]. Различными цветами на рисунке отмечены отдельные нейроны и их связи с нейронами предыдущего слоя.

* 1. **Алгоритмы мутаций**

Все оптимизируемые гиперпараметры проходят через операцию мутации на каждой итерации алгоритма. Эти гиперпараметры можно разделить на две группы: численные значения (то есть параметры активационных функций, коэффициент скорости обучения и момент) и гиперпараметры, относящиеся к топологии сети.

Численные значения проходят через простой алгоритм мутации:

1. Случайным образом выбирается, будет ли значение увеличиваться или уменьшаться на некоторую величину δ.
2. Если после увеличения/уменьшения значения на δ это значение выйдет за допустимые границы, производится обратная операция (уменьшение/увеличение соответственно).
3. Производится корректировка значения на δ в соответствии с определенным сценарием.

Структурная схема алгоритма мутации численных значений представлена на рисунке 5.3

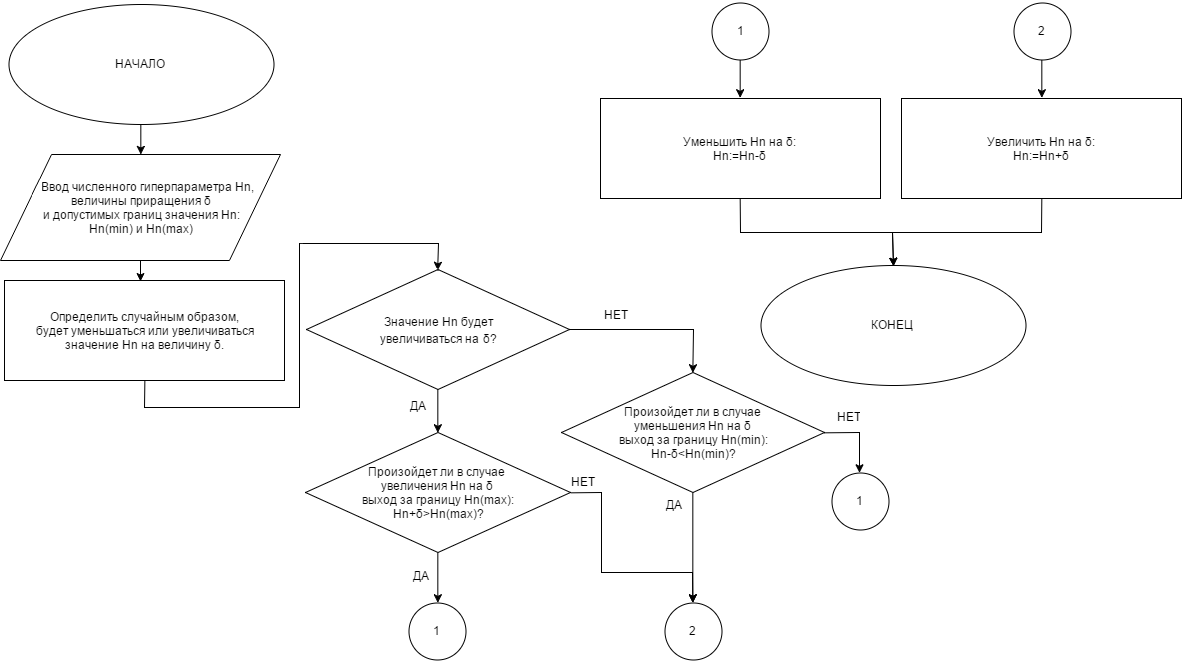


Рисунок 5.3 – Структурная схема алгоритма мутации численных значений

Гиперпараметры, относящиеся к топологии сети, мутируют сложнее в связи с их зависимостью от описанного выше метода кодирования топологии. Можно выделить мутации слоев, нейронов в слоях и связей между нейронами. Каждая из мутаций увеличивает или уменьшает число соответствующих элементов в топологии сети, таким образом делая возможным как редукцию, так и наращивание топологии.

* + 1. **Мутации слоев**

Увеличение и уменьшение числа слоев с помощью операции мутации представлено четырьмя различными алгоритмами. Все они применяются к скрытым слоям, не затрагивая выходной слой нейронной сети.

Число скрытых слоев может быть увеличено двумя способами. В первом, «копировании» (рис. 5.4), нейроны и их связи с предыдущим и следующим слоем полностью копируются и образуют новый слой. Во втором способе, «разделении» (рис. 5.5), половина нейронов остается в одном слое, а вторая вместе со связями образует новый.

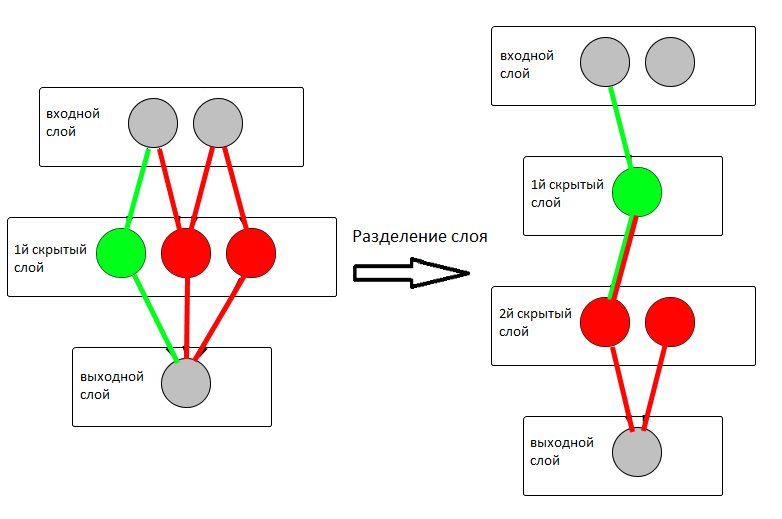


Рисунок 5.4 – Мутация слоев – разделение слоя

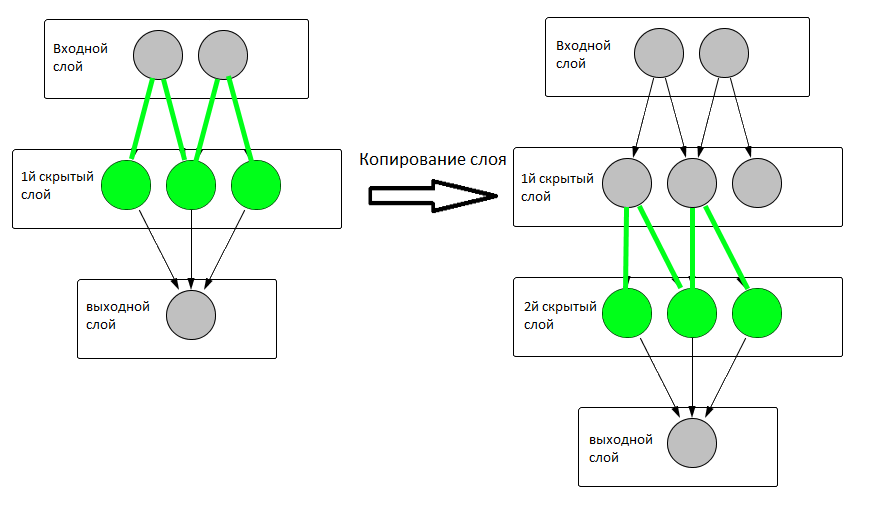


Рисунок 5.5 – Мутация слоев – копирование слоя

Аналогично увеличению, существует два способа уменьшить число слоев. В одном из них, «свертке» слоев (рис. 5.6), нейроны «сворачиваются» в связи: связь образуется от нейрона предыдущего слоя к следующему, если она проходит через нейрон удаляемого слоя. Во втором, «слиянии» слоев (рис. 5.7), половина нейронов удаляемого слоя вместе со связями переходят в предыдущий слой.

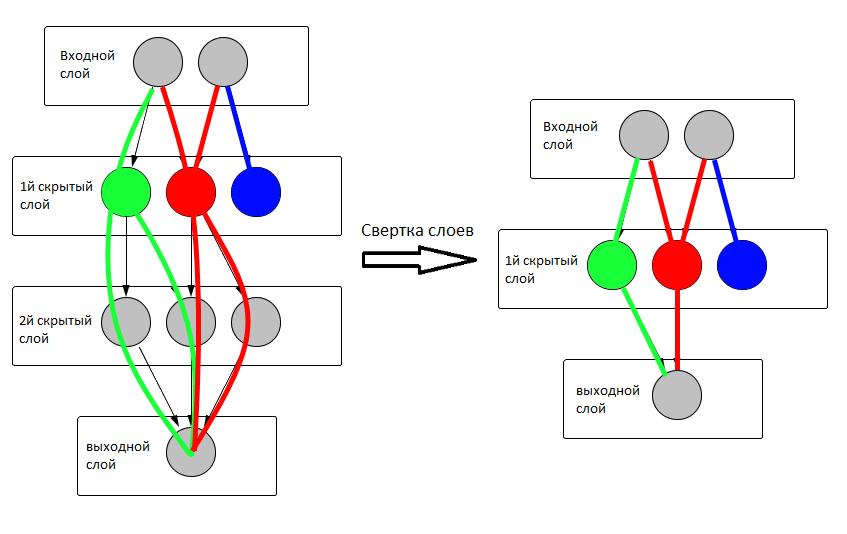


Рисунок 5.6 – Мутация слоев – свертка слоев

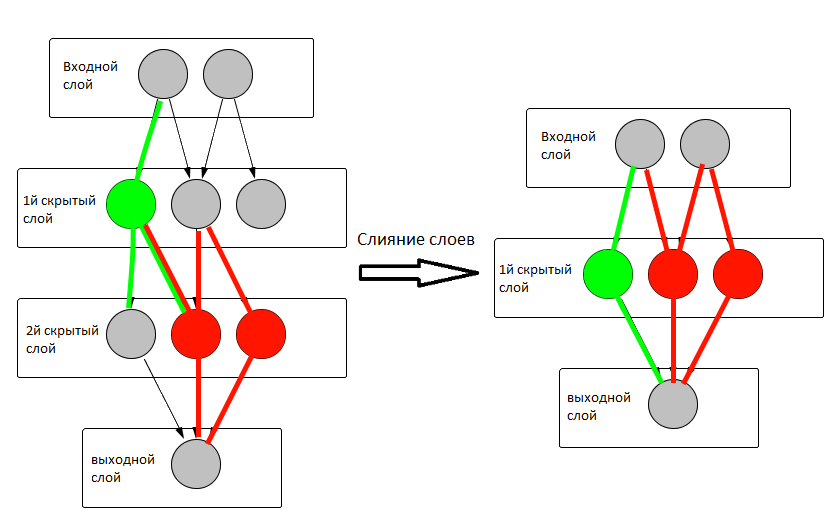


Рисунок 5.7 – Мутация слоев – слияние слоев

Представленные способы позволяют в равной степени комбинировать редукцию и наращивание сети.

* + 1. **Мутации нейронов в слоях**

Последовательное добавление и удаление различных нейронов изображено на рисунке 5.8.

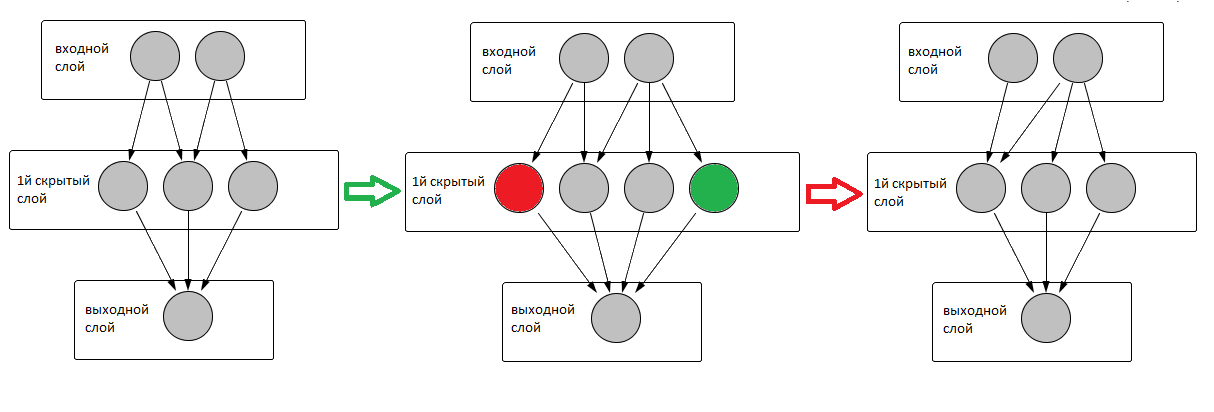


Рисунок 5.8 – Мутация нейронов в слоях – добавление и удаление нейронов

В любой скрытый слой сети может быть случайным образом добавлен нейрон вместе со случайными связями с нейронами из предыдущего и следующего слоя, если нейронов в этом слое меньше 30. Если это не приведет к образованию слоя без нейронов, случайный нейрон может быть аналогично удален.

* + 1. **Мутации связей между нейронами**

Последовательное создание и удаление связей между различными нейронами показано на рисунке 5.9.

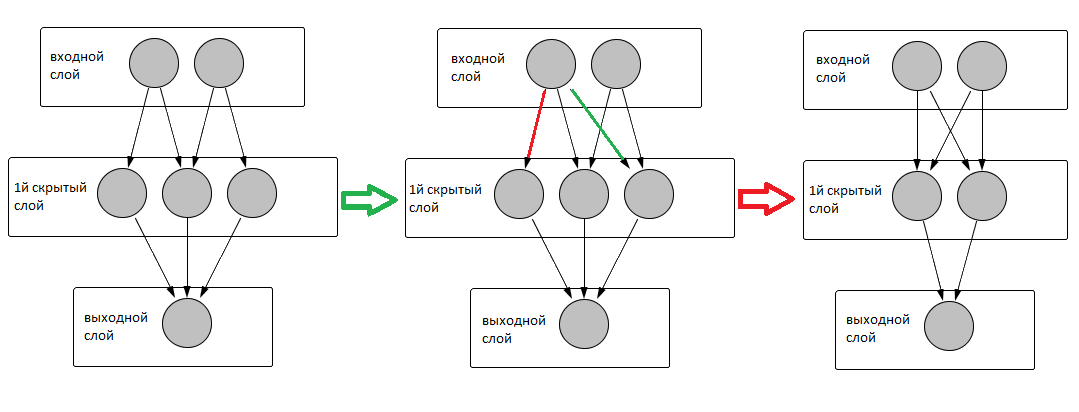


Рисунок 5.9 – Мутация связей между нейронами – создание и удаление связей

Между любыми двумя нейронами в соседних слоях может быть случайным образом создана или удалена связь (если это не приведет к полному отсутствию связей между слоями).

* 1. **Алгоритм скрещивания**

В операции скрещивания принимают участие два генома нейронных сетей-особей. В результате данной операции появляется третья особь – «потомок», геном которого имеет общие черты с геномами обоих особей-«родителей».

Для гиперпараметров – численных значений геном потомка получает значение, равное среднему арифметическому от численных значений родителей: .

Структурная схема алгоритма скрещивания гиперпараметров, относящихся к топологии сети, представлена на рисунке 5.10.

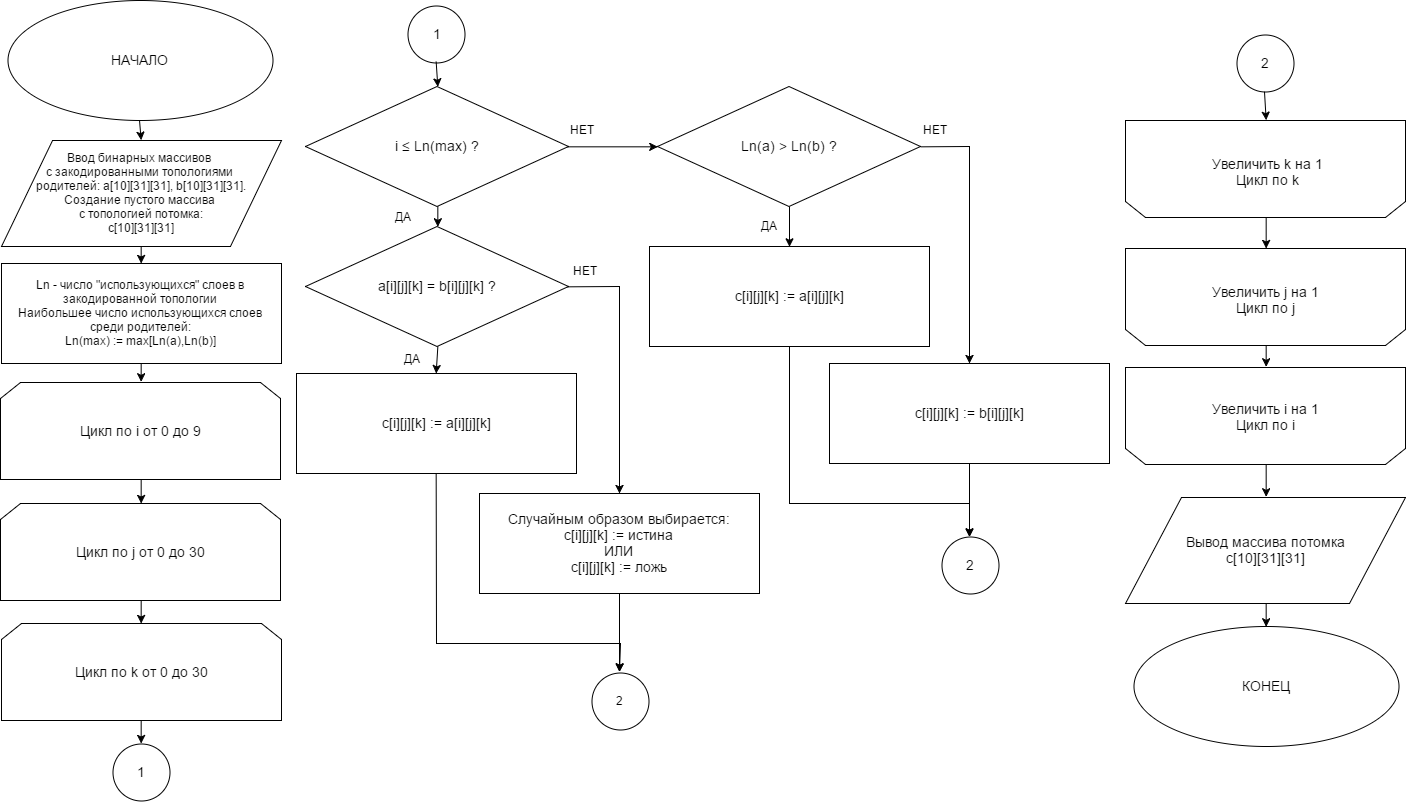


Рисунок 5.10 – Структурная схема алгоритма скрещивания гиперпараметров, относящихся к топологии сети

Для гиперпараметров, относящихся к топологии сети, алгоритм зависит от метода кодирования топологии:

1. Если значения в соответствующих ячейках логического массива, кодирующего топологию, у обоих родителей совпадают – в соответствующую ячейку массива потомка записывается то же значение.
2. Иначе, в соответствующую ячейку потомка заносится случайное логическое значение.
3. Пункты 1-2 повторяются для наибольшего числа «использующихся» слоев среди обоих родителей.
4. Значения для оставшихся ячеек копируются от родителя с наибольшим числом «использующихся» слоев.
   1. **Алгоритм вычисления функции приспособленности**

Структурная схема алгоритма вычисления функции приспособленности представлена на рис. 5.11

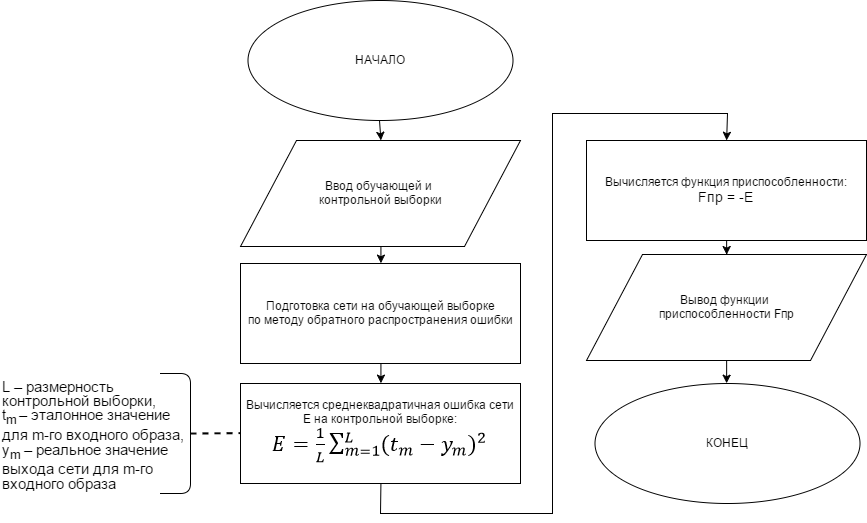


Рисунок 5.11 – Структурная схема алгоритма вычисления функции приспособленности

Алгоритм вычисления значения функции приспособленности нейронной сети заключается в следующем:

1. Сеть проходит подготовку на обучающей выборке по методу обратного распространения ошибки;
2. Вычисляется среднеквадратичная ошибка сети на контрольной выборке по формуле: , где – размерность контрольной выборки, – эталонное значение для -го входного образа, – реальное значение выхода сети для -го входного образа;
3. Значение функции приспособленности выражается как вычисленная среднеквадратичная ошибка, взятая со знаком минус.
4. **ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА**

Структура классов программной реализации алгоритма оптимизации гиперпараметров нейронных сетей представлена в таблице 6.1.

Таблица 6.1 Классы программной реализации алгоритма.

|  |  |
| --- | --- |
| Имя класса | Комментарий |
| ANNOptimizer | *Статический* главный класс программы |
| CompData | Класс данных для обучения и проверки функционирования сети |
| Func | *Статический* класс с методами, не включенными в другие классы |
| Gene | Класс генотипа особи генетического алгоритма |
| Individ | Класс особи генетического алгоритма |
| Mutation | *Статический* класс мутаций генетического алгоритма |
| Population | Класс популяции генетического алгоритма |

Главный класс включает только метод main. В этом методе производится импорт файлов нейронных сетей (\*.nnet), контрольных и обучающих выборок (\*.dset); выбираются величины допустимой ошибки обучения (fitError), максимального числа итераций обучения (maxIter), размера популяции (PopSize) и числа поколений (GNum); производится оптимизация гиперпараметров импортированной нейронной сети и вывод сопутствующей информации.

Поля данных нестатических классов описаны в таблице 6.2, методы классов – в Приложении А.

Таблица 6.2 Поля данных нестатических классов программной реализации алгоритма.

|  |  |  |  |
| --- | --- | --- | --- |
| Класс | Поля данных класса (закрытые) | Тип | Комментарий |
| CompData | trainigSet | DataSet | Обучающая выборка в формате, предоставляемом API Neuroph |
|  | controlSet | DataSet | Контрольная выборка в формате, предоставляемом API Neuroph |
|  | fitError | Вещественное | Допустимая ошибка обучения |
|  | maxIter | Целочисленное | Максимальное число итераций обучения |
| Gene | MAX\_LAYERS | Целочисленное | Константа максимального числа слоев в сети (10) |
|  | MAX\_NEUR\_IN\_LAY | Целочисленное | Константа максимального числа нейронов в слое (30) |
|  | Connections | Логический трехмерный массив | Закодированная топология сети |
|  | InputNum | Целочисленное | Число нейронов во входном слое сети |
|  | OutputNum | Целочисленное | Число нейронов в выходном слое сети |
|  | LearRate | Вещественное | Гиперпараметр – коэффициент скорости обучения сети |
|  | Momentum | Вещественное | Гиперпараметр – момент |
|  | LinearSlope | Вещественное | Гиперпараметр – коэффициент крутизны линейной функции |
|  | SigmoidSlope | Вещественное | Гиперпараметр – коэффициент крутизны сигмоиды |
| Individ | g | Gene | Генотип особи |
|  | Fit | Вещественное | Значение функции приспособленности особи |
| Population | G | Одномерный массив типа Gene | Массив с генотипами особей, представленных в популяции |
|  | CD | CompData | Данные для обучения и проверки функционирования сетей в популяции |
|  | Ind | Одномерный массив типа Individ | Массив с особями, представленными в популяции |

Мутации гиперпараметров, относящихся к топологии сети, осуществляются с различными вероятностями, подобранными опытным путем:

1. Мутация связей – 46%
2. Мутация нейронов – 46%
3. Мутации слоев (разделение, копирование, свертка и слияние) – по 2% каждая.

Мутации численных гиперпараметров осуществляются безусловно.

Разработчики API Neuroph также предоставляют редактор нейронных сетей Neuroph Studio [32]. Данный редактор позволяет создавать и детально настраивать различные типы нейронных сетей с возможностью их дальнейшего сохранения в формате \*.nnet. API Neuroph, в свою очередь, имеет набор функций для чтения и записи в этом формате.

Однако, при создании программной реализации алгоритма оптимизации было выяснено, что данные функции работают в одностороннем порядке: Neuroph Studio распознает файлы, созданные через программу, но программа не читает файлы, созданные в Neuroph Studio. Это связано с отличиями структуры файлов \*.nnet в Neuroph Studio и в API Neuroph, не устраненными в текущей версии API.

Вместо того, чтобы создавать отдельный программный продукт, преобразующий одну структуру файла в другую, было решено написать функцию, сохраняющую полносвязную нейронную сеть с настраиваемыми числом слоев и размерами входного и выходного векторов. Более подробно данная функция описана в Приложении А (класс Func, метод GenerateNet).

Еще один использующийся в программе формат файлов \*.dset, представляющий файл выборки, был разработан в процессе создания программной реализации алгоритма оптимизации. Файлы этого формата можно создавать и открывать с помощью стандартного текстового редактора. Для чтения файла и преобразования в формат DataSet, использующийся в API Neuroph, в программу был добавлен метод GetData (см. Приложение А, класс Func). Обобщенная структура файла \*.dset приведена в таблице 6.3. В случае записи вещественных значений следует использовать разделитель «точка». Пример содержимого файла \*.dset для обучающей выборки к задаче XOR:

2 1 4

0.0 0.0 0.0

1.0 0.0 1.0

0.0 1.0 1.0

1.0 1.0 0.0

Таблица 6.3 Обобщенная структура файла «\*.dset».

|  |  |
| --- | --- |
| Порядок числа в файле | Комментарий |
| 1 | Inp, Размер входного вектора сети |
| 2 | Out, Размер выходного вектора сети |
| 3 | N, число элементов в выборке |
|  | Вектор входных значений элемента выборки. Считываются только первые N векторов. |
|  | Вектор эталонных выходных значений элемента выборки. Считываются только первые N векторов. |

Чтение файла, составленного не в полном соответствии с таблицей 6.3, вызовет ошибку в программе.

Код всей программы на языке Java приведен в Приложении Б.

1. **ОТЛАДКА И ТЕСТИРОВАНИЕ ПРОГРАММЫ**

Для проверки работы программного обеспечения оптимизации гиперпараметров нейронных сетей были созданы обучающая и контрольная (с добавленными шумами) выборки для задачи распознавания монохромных образов – римских цифр. Выборки представлены в таблице 7.1.

Таблица 7.1 Выборки для проверки работы программы.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Римская цифра | Изображение | Двоичный входной вектор обучающей выборки | Искаженное изображение | Двоичный входной вектор контрольной выборки | Двоичный выходной вектор |
| I |  | 0 1 0 0 1 0 0 1 0 |  | 0 1 0 0 0 0 0 1 0 | 1 0 0 0 0 0 0 |
| V |  | 1 0 1 1 0 1 0 1 0 |  | 1 0 1 1 0 0 0 1 0 | 0 1 0 0 0 0 0 |
| X |  | 1 0 1 0 1 0 1 0 1 |  | 1 0 1 0 1 1 1 0 1 | 0 0 1 0 0 0 0 |
| L |  | 1 0 0 1 0 0 1 1 1 |  | 1 0 0 1 0 0 0 1 1 | 0 0 0 1 0 0 0 |
| C |  | 1 1 1 1 0 0 1 1 1 |  | 1 1 1 1 0 0 1 1 0 | 0 0 0 0 1 0 0 |
| D |  | 1 1 0 1 0 1 1 1 0 |  | 1 1 0 1 1 1 1 1 0 | 0 0 0 0 0 1 0 |
| M |  | 1 0 1 1 1 1 1 0 1 |  | 1 0 1 1 1 1 0 0 1 | 0 0 0 0 0 0 1 |

Выборки были занесены в файлы RomTr.dset и RomCr.dset соответственно.

В качестве оптимизируемой нейронной сети была выбрана полносвязная сеть с одним скрытым слоем из 8 нейронов. Это число нейронов определялось по эвристическому правилу геометрической пирамиды (geometric pyramid rule [22]). как , где m – число нейронов в выходном слое (7 согласно размеру выходного вектора), n – число нейронов во входном слое (9 согласно размеру входного вектора). Сеть была занесена в файл ROM.nnet. Структура оптимизируемой сети представлена на рисунке 7.1.

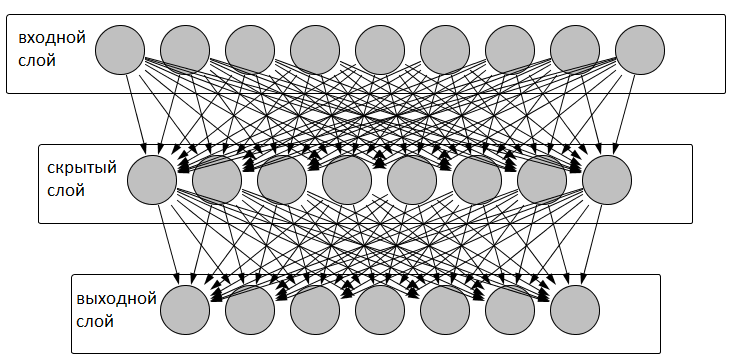


Рисунок 7.1 – Оптимизируемая нейронная сеть

Число особей в популяции выбиралось согласно [10] выбиралось по формуле: , где – вероятность получения в следующем поколении особей с новыми генами, – число элементов в генотипе. Так как основная часть генотипа представлена в виде логического массива boolean[10][31][31], L=10\*31\*31=9610. С выбранной вероятностью P=0.9, N≈17. Число поколений GNum было определено экспериментальным путем как GNum=5. Сеть обучалась до уровня допустимой ошибки ε=0.01.

Лучший результат в процессе оптимизации показала сеть со структурой, представленной на рисунке 7.2.

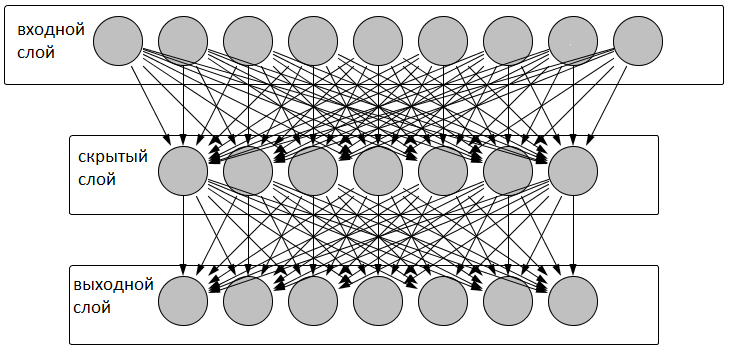


Рисунок 7.2 – Оптимизированная нейронная сеть, первое использование программы

Значения остальных оптимизированных гиперпараметров:

1. коэффициент скорости обучения: 0.12
2. Момент: 0.05
3. Коэффициент крутизны линейной функции: 0.7
4. Коэффициент крутизны сигмоиды: 0.9

Консольный вывод в процессе работы программного обеспечения представлен в Приложении В. В процессе изучения консольного вывода было выяснено, что, несмотря на то, что на первое место в следующее поколение всегда переносится одна особь с лучшим значением функции приспособленности, при повторном обучении и вычислении функции приспособленности данной особи она может показать худшие результаты.

Так, исходя из консольного вывода, неоптимизированная сеть показала ошибку на контрольной выборке E≈0.355, а оптимизированная – E≈0.381. В процессе экспериментов с программой была сделана попытка запустить процесс оптимизации над уже оптимизированной сетью (консольный вывод представлен в Приложении В).

Полученная в результате повторно оптимизированная сеть показала лучшие результаты по сравнению с неоптимизированной и оптимизированной единожды сетями. Структура повторно оптимизированной сети представлена на рисунке 7.3. Ошибка на контрольной выборке для повторно оптимизированной сети составляет E≈0.30.

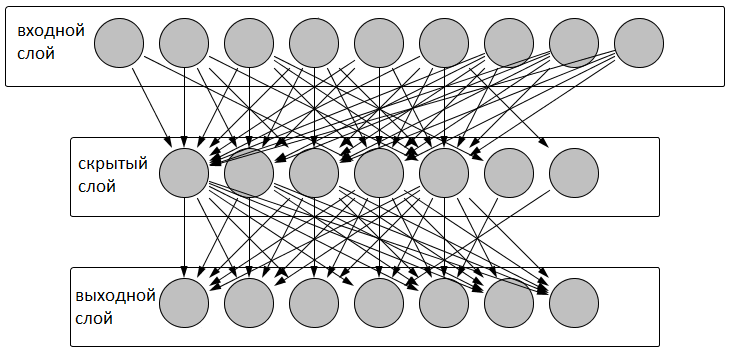


Рисунок 7.3 – Оптимизированная нейронная сеть, повторное использование программы

Значения остальных оптимизированных гиперпараметров:

1. коэффициент скорости обучения: 0.16
2. Момент: 0.28
3. Коэффициент крутизны линейной функции: 0.925
4. Коэффициент крутизны сигмоиды: 0.725

График зависимости величины ошибки от числа итераций обучения для оптимизируемой, оптимизированной на первом использовании программы и оптимизированной повторно сетей изображен на рисунке 7.4.

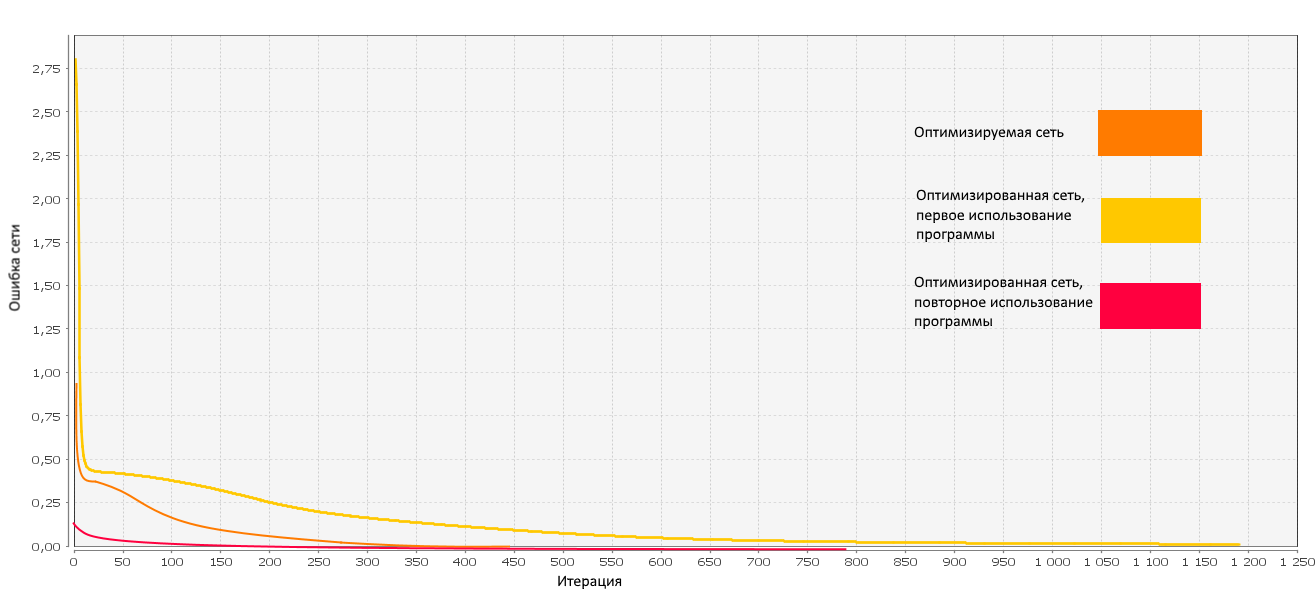


Рисунок 7.4 – Сравнительный график зависимости ошибки сети от числа итераций обучения

Также, для сравнения с другими существующими алгоритмами оптимизации гиперпараметров, задача распознавания монохромных образов (табл. 7.1) была испытана на алгоритме NEAT в среде Encog Workbench [30].

В качестве исходных параметров были заданы размер популяции N=17 и уровень допустимой ошибки ε=0.01. График зависимости ошибки сети от числа итераций обучения для алгоритма NEAT представлен на рисунке 7.5.

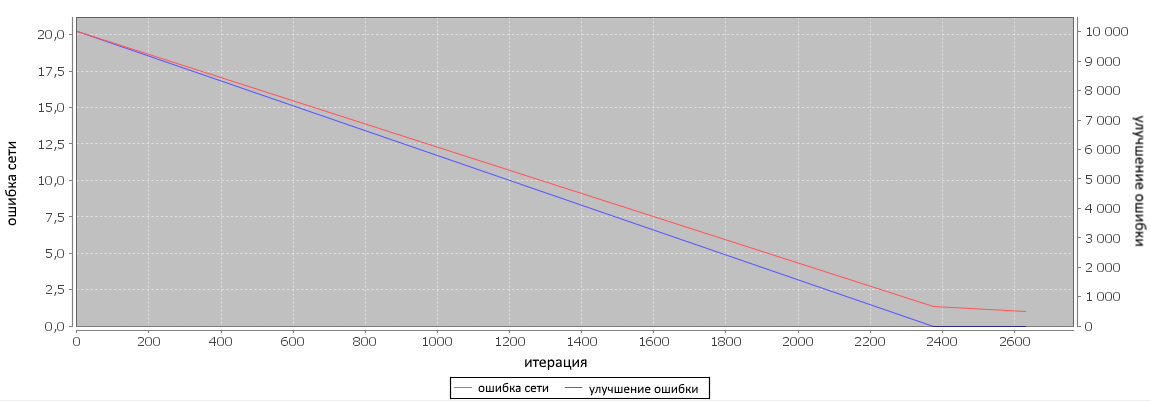


Рисунок 7.5 – График зависимости ошибки сети от числа итераций обучения для алгоритма NEAT

Если сравнить графики на рисунках 7.4 и 7.5, можно отметить, что алгоритм NEAT обучает сеть за большее число итераций. Структура сети, оптимизированной алгоритмом NEAT, представлена на рисунке 7.6.

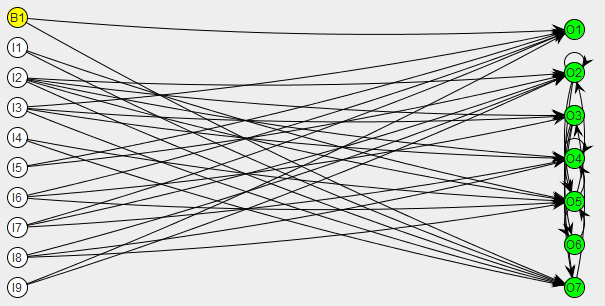


Рисунок 7.6 – Структура сети, оптимизированной алгоритмом NEAT

На рисунке 7.6 отмечены: I1-I9 – входные нейроны; O1-O7 – выходные нейроны, B1 – нейрон смещения (bias). Исходная сеть состояла только из нейронов входного и выходного слоя, не связанных между собой. Все образованные связи и нейрон смещения – результат функционирования алгоритма. Специфическими особенностями сети, оптимизированной алгоритмом NEAT, являются связи между нейронами выходного слоя и отсутствие скрытых слоев.

Хотя оба способа оптимизации основаны на генетических алгоритмах, программное обеспечение, описанное в данной работе, в отличие от NEAT обладает способностью к редукции структуры сети, а также оптимизирует не только топологию, но и другие важные гиперпараметры, такие как коэффициент скорости обучения, момент и параметры активационных функций. Еще одно отличие заключается в том, что связи в оптимизируемой сети могут быть образованы только между нейронами в соседних слоях, тогда как NEAT может создать связь между любыми двумя нейронами или даже от нейрона к самому себе.

**ЗАКЛЮЧЕНИЕ**

Подводя итоги данной работы, следует отметить, что выбранный метод оптимизации гиперпараметров, а именно использование генетических алгоритмов, отлично подошел для достижения поставленной цели.

Для этого метода был разработан способ кодирования топологии сети, а также предложены алгоритмы операций мутации, скрещивания и вычисления функции приспособленности, а также общий алгоритм функционирования. Для реализации данного алгоритма в разработанном программном обеспечении было решено задействовать API Neuroph. Среди предложенных к оптимизации гиперпараметров были выбраны топология сети (число скрытых слоев и нейронов в них, архитектура связей между нейронами), параметры активационной функции, коэффициент скорости обучения и момент.

Созданное программное обеспечение, хотя и не с первого раза, смогло уменьшить ошибку функционирования нейронной сети на контрольном множестве приблизительно на 20%, при этом упростив ее структуру. Однако, оптимизация производилась лишь на небольшом множестве из всех возможных гиперпараметров и только для узкого класса нейронных сетей.

Кроме того, при создании программного обеспечения средствами API Neuroph и NeurophStudio возникли сложности, связанные с отличиями в структуре файлов, хранящих данные о нейронных сетях, частично устраненные путем создания функции, сохраняющей полносвязную сеть в необходимом формате.

Научная новизна данной работы состоит в разработке нового алгоритма оптимизации гиперпараметров нейронных сетей, подразумевающего в том числе возможность как увеличения, так и редукции структуры сети. Практическая значимость работы заключается в возможности использования разрабатываемого программного обеспечения для оптимизации уже существующих, но показывающих недостаточно эффективную производительность в решении соответствующих задач нейронных сетей. После возможной доработки функций и создания пользовательского интерфейса данное программное обеспечение может стать полезным инструментом в комплексе с редактором нейронных сетей NeurophStudio.

В дальнейшем представляется возможным ввести коэффициент коррекции функции приспособленности в зависимости от размера сети, что увеличит шанс на одновременное уменьшение ошибки функционирования и размера сети в процессе оптимизации. Также возможно введение других оптимизируемых гиперпараметров, изменение алгоритма в целях обеспечения поддержки других классов нейронных сетей и добавление тонких настроек для оптимизируемых гиперпараметров, таких как выбор гиперпараметров из списка и задание границ их изменения.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. **Алсагарова Е.В.** Метаэвристические методы оптимизации в экономике // БГУ: ЭКОНОМИКА. ПРАВО. МЕНЕДЖМЕНТ. - 2014 г.
2. **Боровых Н.И. и Красоткина О.В.** Применение алгоритма поиска с запретами в задаче автоматизированного составления оптимального штатного расписания // Известия Тульского государственного университета. Технические науки. - 2013 г.
3. **Карпенко А.П. и Селиверстов Е.Ю.** Обзор методов роя частиц для задачи глобальной оптимизации (Particle Swarm Optimization) // Наука и образование: научное издание МГТУ им. Н.Э. Баумана. - 2009 г.
4. **Кочетов Ю.А.** Методы локального поиска для дискретных задач размещения. - Новосибирск : Институт Математики им. С.Л.Соболева, 2009 г.
5. **Кузнецов В.А. и Сеничев К.Н.** Метаэвристики для решения комбинаторных задач // Вестник Череповецкого государственного университета. - 2012 г.
6. **Осовский С.** Нейронные сети для обработки информации. - Москва : Финансы и статистика, 2002 г.
7. **BaseGroup™ Labs** Алгоритм обратного распространения ошибки. - BaseGroup™ Labs, 2010 г.
8. **Morincer** Обзор методов эволюции нейронных сетей // Geektimes. - 02.12.2010 г. (проверено 22.05.2017) - https://geektimes.ru/post/84015.
9. Массивы. Часть 3 – многомерные массивы. // Pro Java. - 18.05.2015 г. (проверено 22.05.2017) - http://pr0java.blogspot.fr/2015/05/3.html.
10. **Митин А.** Выбор размера популяции для генетического алгоритма // Хабрахабр. - TechMedia, 04.03.2011 г. (проверено 22.05.2017) - https://habrahabr.ru/post/114910/.
11. **Bengio Y.** Practical Recommendations for Gradient-Based Training of Deep Architectures ver. 2. - Canada : Universite de Montreal, 2012.
12. **C.M. Bishop** Neural Networks and Pattern Recognition. - Oxford Press., 1995.
13. **Corominas A., Garcia-Villoria A. and Pastor R.** The Response Time Variability Problem: A Review. - IOC : 3rd International Conference on Industrial Engineering and Industrial Management, 2009.
14. **Geron Aurelien** Hands-On Machine Learning with Scikit-Learn and TensorFlow - O’Reilly Media, Inc., 2017.
15. **Ghosh R. [et al.]** Comparative analysis of Genetic algorithm, Simulated annealing and Cutting Angle method for artificial neural networks. - University of Ballarat : School of InformationTechnology and Mathematical Sciences, 2005.
16. **Heaton J.** Encog 3.3: Development Guide. - St.Louis : Heaton Research, Inc., 2014.
17. **Heaton J.** Programming Neural Networks with Encog3 in Java. - St.Louis : Heaton Research, Inc., 2011.
18. **Lee Ji Young and Dernoncourt F.** Optimizing Neural Network Hyperparameters with Gaussian Processes for Dialog Act Classification. - MIT CSAIL Cambridge, 2016.
19. **Lee Ji Young and Dernoncourt F.** Sequential short-text classification with recurrent and convolutional neural networks in Human Language Technologies. - NAACL HLT : The Conference of the North American Chapter of the Association for Computational Linguistics, 2016.
20. **M. Pandian** Meta-Heuristics Optimization Algorithms in Engineering, Business, Economics, and Finance. - IGI Global. - Vol. 2012.
21. **MacLeod C. and Maxwell G. M.** Incremental Evolution in ANNs: Neural Nets which Grow. - The Robert Gordon University Aberdeen : School of Electronic and Electrical Engineering, 2001.
22. **Masters T.** Practical Neural Network Recipies in C++. - San Diego : Academic Press, Inc., 1993.
23. **Mitchell M.** Genetic Algorithms: An Overview. - Santa Fe Institute, 1995.
24. **Pirim H., Bayraktar E. and Eksioglu B.** Tabu Search: A Comparative Study. - Mississippi State University : Industrial and System Engineering Department, 2008.
25. **Said G., Mahmoud A. and El-Horbaty M.** A Comparative Study of Meta-heuristic Algorithms for Solving Quadratic Assignment Problem // International Journal of Advanced Computer Science and Applications. - 2014.
26. **Stanley Kenneth O. and Miikkulainen R.** Evolving Neural Networks through Augmenting Topologies // The MIT Press Journals. - 2002.
27. **Sulaiman M. [et al.]** Improved Water Level Forecasting Performance by Using Optimal Steepness Coefficients in an Artificial Neural Network. - Springer Science+Business Media B.V., 2011.
28. **Youssef H., Sait S. and Adiche H.** Evolutionary algorithms, simulated annealing and tabu search: a comparative study. - KFUPM : Department of Computer Engineering, 2000.
29. Comparison of Neural Network Simulators // CCNLab. - 11.10.2014. (проверено 22.05.2017) - http://grey.colorado.edu/emergent/index.php/Comparison\_of\_Neural\_Network\_Simulators.
30. Encog Core 3.3.0 API // Encog. - 2014. (проверено 22.05.2017) - http://heatonresearch-site.s3-website-us-east-1.amazonaws.com/javadoc/encog-3.3/overview-summary.html.
31. Neuroph Core 2.9 API // Neuroph. - 2014. (проверено 22.05.2017) - http://neuroph.sourceforge.net/javadoc/overview-summary.html.
32. Neuroph Download // Neuroph. – 2014. (проверено 22.05.2017) - http://neuroph.sourceforge.net/download.html.

# **ПРИЛОЖЕНИЕ А**

Таблица А.1 Методы классов программной реализации алгоритма.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс | Методы | Тип возвращаемого значения | Входные параметры | Комментарий |
| CompData | CompData | – | DataSet trainingSet | Конструктор класса |
| DataSet controlSet |
| Вещественное fitError |
| Целочисленное maxIter |
|  | getTrainingSet | DataSet | – | Акцессоры, возвращают соответствующие значения полей данных |
| getControlSet | DataSet |
| getFitError | Вещественное |
| getMaxIter | Целочисленное |
|  | setTrainingSet | Пустотное (void) | DataSet trainingSet | Акцессоры, устанавливают соответствующие значения полей данных |
| setControlSet | DataSet controlSet |
| setFitError | Вещественное fitError |
| setMaxIter | Целочисленное maxIter |
| Func | rnd (открытый) | Целочисленное | Целочисленное min | Возвращает случайное число в пределах [min, max] |
| Целочисленное max |
|  | calculateError (открытый) | Вещественное | NeuralNetwork net | Вычисляет ошибку функционирования сети net на контрольной выборке controlSet |
| DataSet controlSet |
|  | GetData (открытый) | DataSet | Строковое filePath | Преобразует данные из файла по адресу filePath в выборку |

Таблица А.1 (продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс | Методы | Тип возвращаемого значения | Входные параметры | Комментарий |
| Func | GenerateNet (открытый) | Пустотное (void) | DataSet trainingSet | Сохраняет полносвязную сеть, способную обучаться на выборке trainingSet, содержащую hidLayNum скрытых слоев с NeurInLay нейронов в каждом. Сеть сохраняется по адресу filePath в формате, предоставляемом API Neuroph |
| Целочисленное hidLayNum |
| Целочисленное NeurInLay |
| Строковое filePath |
| Gene | Gene | – | Gene G | Конструктор класса, создает новый объект класса на основе существующего объекта G |
|  | Gene | – | Логический трехмерный массив Connections | Конструктор класса, получает значения полей данных InputNum и OutputNum из trainingSet, поля LearRate, Momentum, LinearSlope и SigmoidSlope задаются по умолчанию (0.2, 0, 1 и 1 соответственно) |
| DataSet trainingSet |

Таблица А.1 (продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс | Методы | Тип возвращаемого значения | Входные параметры | Комментарий |
| Gene | Gene | – | Логический трехмерный массив Connections | Конструктор класса |
| Целочисленное InputNum |
| Целочисленное OutputNum |
| Вещественное LearRate |
| Вещественное Momentum |
| Вещественное LinearSlope |
| Вещественное SigmoidSlope |
|  | getConnections | Логический трехмерный массив Connections | – | Акцессоры, возвращают соответствующие значения полей данных |
| getInputNum | Целочисленное |
| getOutputNum | Целочисленное |
| getLearRate | Вещественное |
| getMomentum | Вещественное |
| getLinearSlope | Вещественное |
| getSigmoidSlope | Вещественное |
| getMAX\_LAYERS | Целочисленное |
| getMAX\_NEUR\_IN\_LAY | Целочисленное |

Таблица А.1 (продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс | Методы | Тип возвращаемого значения | Входные параметры | Комментарий |
| Gene | setConnections | Пустотное (void) | Логический трехмерный массив Connections | Акцессоры, устанавливают соответствующие значения полей данных |
| setInputNum | Целочисленное InputNum |
| setOutputNum | Целочисленное OutputNum |
| setLearRate | Вещественное LearRate |
| setMomentum | Вещественное Momentum |
| setLinearSlope | Вещественное LinearSlope |
| setSigmoidSlope | Вещественное SigmoidSlope |
|  | isConnDeletable (открытый) | Логическое | Целочисленное LayerNum | Если из слоя LayerNum можно удалить связь, возвращает логическое значение «истина» |
|  | isNeurDeletable (открытый) | Логическое | Целочисленное LayerNum | Если из слоя LayerNum можно удалить нейрон, возвращает логическое значение «истина» |
|  | isLaySplitable (открытый) | Логическое | Целочисленное LayerNum | Если слой LayerNum можно разбить на два слоя, возвращает логическое значение «истина» |
|  | getLayerCount (открытый) | Целочисленное | – | Возвращает число «использующихся» слоев |
|  | getLastActNeur | Целочисленное | Целочисленное LayerNum | Возвращает номер последнего «использующегося» нейрона в слое LayerNum |
|  | Crossover (открытый статический) | Gene | Gene a | Скрещивает генотипы родителей a и b и возвращает генотип потомка |
| Gene b |

Таблица А.1 (продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс | Методы | Тип возвращаемого значения | Входные параметры | Комментарий |
| Gene | Mutate (открытый) | Пустотное (void) | – | Осуществляет мутации над генотипом в соответствии с определенными вероятностями. |
|  | TransformFromNet  (открытый статический) | Gene | NeuralNetwork net | Переводит сеть net из формата, предоставляемого API Neuroph, в программное представление генотипа. |
|  | TransformToNet (открытый) | NeuralNetwork | – | Переводит программное представление генотипа в формат, предоставляемый API Neuroph |
| Individ | Individ | – | Gene g | Конструктор класса |
| Вещественное fit |
|  | getG | Gene | – | Акцессоры, возвращают соответствующие значения полей данных |
| getFit | Вещественное |
|  | setG | – | Gene g | Акцессоры, устанавливают соответствующие значения полей данных |
| setFit | Вещественное fit |
|  | Sort (открытый статический) | Пустотное (void) | Одномерный массив Ind типа Individ | Осуществляет сортировку массива Ind по убыванию |
| Mutation | ConnInv (открытый статический) | Пустотное (void) | Gene g | Меняет значение случайной связи между двумя нейронами в случайном слое на обратное. Если имеется менее двух «использующихся» соединений в слое, устанавливает значение случайного соединения в положение «истина» |
|  | NeurInv (открытый статический) | Пустотное (void) | Gene g | Меняет значение случайного нейрона в случайном скрытом слое на обратное. Если имеется менее двух «использующихся» нейронов в слое, устанавливает значение случайного нейрона в положение «истина» |
|  | LayerCopy (открытый статический) | Пустотное (void) | Gene g | Копирует случайный скрытый слой, если для копии имеется свободное место |
|  | LayerSplit (открытый статический) | Пустотное (void) | Gene g | Разделяет случайный скрытый слой на два, если для этого имеется свободное место |

Таблица А.1 (продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс | Методы | Тип возвращаемого значения | Входные параметры | Комментарий |
| Mutation | LayerConvlt (открытый статический) | Пустотное (void) | Gene g | «Сворачивает» два слоя в один. Связь образуется от нейрона предыдущего слоя к следующему, если она проходит через нейрон удаляемого слоя |
|  | LayerMerge (открытый статический) | Пустотное (void) | Gene g | Производит «слияние» двух слоев в один. Половина нейронов удаляемого слоя вместе со связями переходит в предыдущий слой |
|  | MomentChange (открытый статический) | Пустотное (void) | Gene g | Мутация численного значения момента |
|  | LRChange (открытый статический) | Пустотное (void) | Gene g | Мутация численного значения коэффициента скорости обучения |
|  | LSChange (открытый статический) | Пустотное (void) | Gene g | Мутация численного значения коэффициента крутизны линейной функции |
|  | SSChange (открытый статический) | Пустотное (void) | Gene g | Мутация численного значений коэффициента крутизны сигмоиды |
| Population | Population | – | Одномерный массив G типа Gene | Конструктор класса |
| CompData CD |
|  | Population | – | Gene Init | Конструктор класса. Инициализирует массив с генотипами особей, представленных в популяции, включая в него особь Init и Size-1 особей, полученных после прохождения Init операции мутации. |
| Целочисленное Size |
| CompData CD |

Таблица А.1 (продолжение)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Класс | Методы | Тип возвращаемого значения | Входные параметры | Комментарий |
| Population | getG | Одномерный массив типа Gene | – | Акцессоры, возвращают соответствующие значения полей данных |
| getCD | CompData |
| getInd | Одномерный массив типа Individ |
|  | setG | Пустотное (void) | Одномерный массив G типа Gene | Акцессоры, устанавливают соответствующие значения полей данных |
| setCD | CompData CD |
| setInd | Одномерный массив Ind типа Individ |
|  | getFitness (открытый) | Вещественное | Целочисленное Num | Вычисляет функцию приспособленности для Num особи в популяции |
|  | InitFit (закрытый) | Пустотное (void) | – | Осуществляет нормализацию и сортировку значений функции приспособленности для всех особей популяции, после чего вычисляет накопленное нормализованное значение их функции приспособленности |
|  | GetParent (закрытый) | Gene | – | Акцессоры, устанавливают соответствующие значения полей данных |
|  | NewGen (закрытый) | Одномерный массив типа Gene | – | Инициализирует новое поколение на основе текущего |
|  | Iterate (открытый) | Пустотное (void) | – | Производит одну итерацию генетического алгоритма |
|  | PrintGene (открытый) | Пустотное (void) | Целочисленное Num | Выводит в консоль информацию о структуре части генома, отвечающей за топологию Num особи в поколении |

# **ПРИЛОЖЕНИЕ Б**

**ANNOptimizer.java:**

package ann.optimizer;

import org.neuroph.core.NeuralNetwork;

import org.neuroph.core.data.DataSet;

import org.neuroph.core.exceptions.NeurophException;

public class ANNOptimizer {

public static void main(String[] args) {

String file = "NNETS/2/P3\_N5.nnet";

DataSet trainingSet = Func.GetData("RomTr.dset");

DataSet controlSet = Func.GetData("RomCr.dset");

double fitError = 0.01;

int maxIter = 100000;

CompData CD = new CompData(trainingSet, controlSet, fitError, maxIter);

int PopSize = 5; int GNum = 2;

try {

NeuralNetwork.createFromFile(file);

}

catch(NeurophException ne) {

System.err.println(ne);

System.exit(1);

}

Gene g = new Gene(Gene.TransformFromNet(NeuralNetwork.createFromFile(file)));

Population pop = new Population(g,PopSize,CD);

int i=0;

for(; i<GNum; i++) {

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

pop.getG()[j].TransformToNet().save("NNETS/P"+i+"\_N"+j+".nnet");

System.out.print("Pop #"+i+"\n");

pop.Iterate();

}

System.out.print("Pop #"+i+"\n");

for(int j=0; j<PopSize; j++) {

pop.getFitness(j);

pop.getG()[j].TransformToNet().save("NNETS/P"+i+"\_N"+j+".nnet");

}

}

}

**CompData.java:**

package ann.optimizer;

import org.neuroph.core.data.DataSet;

public class CompData {

private DataSet trainingSet;

private DataSet controlSet;

private double fitError;

private int maxIter;

public CompData(DataSet trainingSet, DataSet controlSet, double fitError, int maxIter) {

this.trainingSet = trainingSet;

this.controlSet = controlSet;

this.fitError = fitError;

this.maxIter=maxIter;

}

/\*\*

\* @return the trainingSet

\*/

public DataSet getTrainingSet() {

return trainingSet;

}

/\*\*

\* @param trainingSet the trainingSet to set

\*/

public void setTrainingSet(DataSet trainingSet) {

this.trainingSet = trainingSet;

}

/\*\*

\* @return the controlSet

\*/

public DataSet getControlSet() {

return controlSet;

}

/\*\*

\* @param controlSet the controlSet to set

\*/

public void setControlSet(DataSet controlSet) {

this.controlSet = controlSet;

}

/\*\*

\* @return the fitError

\*/

public double getFitError() {

return fitError;

}

/\*\*

\* @param fitError the fitError to set

\*/

public void setFitError(double fitError) {

this.fitError = fitError;

}

/\*\*

\* @return the maxIter

\*/

public int getMaxIter() {

return maxIter;

}

/\*\*

\* @param MaxIter the maxIter to set

\*/

public void setMaxIter(int MaxIter) {

this.maxIter = MaxIter;

}

}

**Func.java:**

package ann.optimizer;

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Locale;

import java.util.Scanner;

import org.neuroph.core.Layer;

import org.neuroph.core.NeuralNetwork;

import org.neuroph.core.Neuron;

import org.neuroph.core.data.DataSet;

import org.neuroph.core.input.WeightedSum;

import org.neuroph.core.transfer.Linear;

import org.neuroph.core.transfer.Sigmoid;

import org.neuroph.nnet.comp.neuron.InputNeuron;

import org.neuroph.util.ConnectionFactory;

import org.neuroph.util.NeuronProperties;

public class Func {

public static int rnd(int min, int max)

{

max -= min;

return (int) (Math.random() \* ++max) + min;

}

public static double calculateError(NeuralNetwork net, DataSet controlSet) {

double error = 0;

for (int i = 0; i < controlSet.size(); i++) {

net.setInput(controlSet.getRowAt(i).getInput());

net.calculate();

for (int j = 0; j < controlSet.getRowAt(i).getDesiredOutput().length; j++) {

double delta = controlSet.getRowAt(i).getDesiredOutput()[j] - net.getOutput()[j];

error += Math.pow(delta, 2);

}

}

System.out.print("Err: "+error/controlSet.size()+"\n");

return error / controlSet.size();

}

public static DataSet GetData(String filePath) {

try {

Scanner sc = new Scanner(new File(filePath));

int inp=sc.nextInt(), out=sc.nextInt();

DataSet Data = new DataSet(inp,out);

int rows=sc.nextInt();

sc.useLocale(Locale.US);

for(int i=0; i<rows; i++) {

double InpSet[] = new double[inp];

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

InpSet[j]=sc.nextDouble();

double OutSet[] = new double[out];

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

OutSet[j]=sc.nextDouble();

Data.addRow(InpSet, OutSet);

}

return Data;

} catch (FileNotFoundException ex) {

System.err.println(ex);

System.exit(1);

}

return null;

}

public static void GenerateNet(DataSet trainingSet, int hidLayNum, int NeurInLay, String filePath) {

NeuralNetwork net = new NeuralNetwork();

net.addLayer(0, new Layer(trainingSet.getInputSize(),new NeuronProperties(InputNeuron.class, Linear.class)));

net.setInputNeurons(net.getLayerAt(0).getNeurons());

int i=1;

for(; i<hidLayNum+1; i++)

net.addLayer(i,new Layer(NeurInLay,new NeuronProperties(Neuron.class,WeightedSum.class,Sigmoid.class)));

net.addLayer(i,new Layer(trainingSet.getOutputSize(),new NeuronProperties(Neuron.class,WeightedSum.class,Sigmoid.class)));

net.setOutputNeurons(net.getLayerAt(i).getNeurons());

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

ConnectionFactory.fullConnect(net.getLayerAt(j),net.getLayerAt(j+1));

net.save(filePath);

}

}

**Gene.java:**

package ann.optimizer;

import org.neuroph.core.Layer;

import org.neuroph.core.NeuralNetwork;

import org.neuroph.core.Neuron;

import org.neuroph.core.data.DataSet;

import org.neuroph.core.input.\*;

import org.neuroph.core.transfer.\*;

import org.neuroph.nnet.comp.neuron.InputNeuron;

import org.neuroph.util.ConnectionFactory;

import org.neuroph.util.NeuronProperties;

public class Gene {

private static final int MAX\_LAYERS = 10;

private static final int MAX\_NEUR\_IN\_LAY = 30;

private boolean[][][] Connections;

private int InputNum;

private int OutputNum;

private double LearRate;

private double Momentum;

//private Class<? extends TransferFunction> FTrans;

//private Class<? extends InputFunction> FInp;

private double LinearSlope;

private double SigmoidSlope;

Gene(Gene G) {

this.Connections = new boolean[MAX\_LAYERS][1+MAX\_NEUR\_IN\_LAY][1+MAX\_NEUR\_IN\_LAY];

for(int i=0; i<MAX\_LAYERS; i++)

for(int j=0; j<1+MAX\_NEUR\_IN\_LAY; j++)

for(int k=0; k<1+MAX\_NEUR\_IN\_LAY; k++)

this.Connections[i][j][k]=G.getConnections()[i][j][k];

this.InputNum = G.getInputNum();

this.OutputNum = G.getOutputNum();

this.LearRate=0.2; this.Momentum=0.0;

this.LinearSlope=1d; this.SigmoidSlope=1d;

}

Gene(boolean[][][] Connections, int InputNum, int OutputNum,

double LearRate, double Momentum,

double LinearSlope,

double SigmoidSlope) {

this.Connections = new boolean[MAX\_LAYERS][1+MAX\_NEUR\_IN\_LAY][1+MAX\_NEUR\_IN\_LAY];

for(int i=0; i<MAX\_LAYERS; i++)

this.Connections[i]=Connections[i];

this.InputNum = InputNum;

this.OutputNum = OutputNum;

this.LearRate=LearRate; this.Momentum=Momentum;

this.LinearSlope=LinearSlope; this.SigmoidSlope=SigmoidSlope;

}

Gene(boolean[][][] Connections, DataSet trainingSet) {

this.Connections = new boolean[MAX\_LAYERS][1+MAX\_NEUR\_IN\_LAY][1+MAX\_NEUR\_IN\_LAY];

for(int i=0; i<MAX\_LAYERS; i++)

this.Connections[i]=Connections[i];

InputNum = trainingSet.getInputSize();

OutputNum = trainingSet.getOutputSize();

this.LearRate=0.2; this.Momentum=0.0;

this.LinearSlope=1d; this.SigmoidSlope=1d;

}

public boolean isConnDeletable(int LayerNum) {

int ActiveConn=0;

for(int i=0; i<getMAX\_NEUR\_IN\_LAY(); i++)

for(int j=0; j<getMAX\_NEUR\_IN\_LAY(); j++)

if(LayerNum > 0) {

//for(int k=0; k<MaxNeurInLay; k++) {

if( getConnections()[LayerNum] [1+i][0]&

getConnections()[LayerNum] [1+i][1+j]&

getConnections()[LayerNum-1] [1+j/\*k\*/][0]==true)

ActiveConn++;

//}

}

else {

if( getConnections()[LayerNum][1+i][0]&

getConnections()[LayerNum][1+i][1+j]==true)

ActiveConn++;

}

return ActiveConn>1;

}

public boolean isNeurDeletable(int LayerNum) {

int ActiveNeur = 0;

for(int i=0; i<getLastActNeur(LayerNum); i++)

if(getConnections()[LayerNum][1+i][0]==true)

ActiveNeur++;

return ActiveNeur>1;

}

public boolean isLaySplitable (int LayerNum) {

int ActiveNeur = 0;

for(int i=getLastActNeur(LayerNum)/2; i<getLastActNeur(LayerNum); i++)

if(getConnections()[LayerNum][1+i][0])

ActiveNeur++;

return ActiveNeur>1;

}

public int getLayerCount() {

int LayerCount = 0;

for(int i=0; i<getMAX\_LAYERS(); i++)

if(getConnections()[i][0][0]==true)

LayerCount++;

return LayerCount;

}

public int getLastActNeur(int LayerNum) {

int LastNeur = 0;

for(int i=1; i<1+getMAX\_NEUR\_IN\_LAY(); i++)

if(getConnections()[LayerNum][i][0] == true)

LastNeur = i;

return LastNeur;

}

public static Gene Crossover(Gene a, Gene b){

boolean[][][] Conn\_c = new boolean[Gene.getMAX\_LAYERS()][1+Gene.getMAX\_NEUR\_IN\_LAY()][1+Gene.getMAX\_NEUR\_IN\_LAY()];

for(int i=0; i<Gene.getMAX\_LAYERS(); i++)

for(int j=0; j<Gene.getMAX\_NEUR\_IN\_LAY()+1; j++)

for(int k=0; k<Gene.getMAX\_NEUR\_IN\_LAY()+1; k++)

if(i!=(a.getLayerCount()>b.getLayerCount()?

a.getLayerCount():

b.getLayerCount())) {

if(a.getConnections()[i][j][k]==b.getConnections()[i][j][k])

Conn\_c[i][j][k]=a.getConnections()[i][j][k];

else {

int rnd = Func.rnd(0, 1);

Conn\_c[i][j][k]=(boolean)(rnd==1);

}

}

else {

if(a.getLayerCount()>b.getLayerCount())

Conn\_c[i][j][k]=a.getConnections()[i][j][k];

else

Conn\_c[i][j][k]=b.getConnections()[i][j][k];

}

double retLSlope=(a.getLinearSlope()+b.getLinearSlope())/2.0;

double retSSlope=(a.getSigmoidSlope()+b.getSigmoidSlope())/2.0;

return new Gene(Conn\_c,a.getInputNum(),a.getOutputNum(),

(a.getLearRate()+b.getLearRate())/2,

(a.getMomentum()+b.getMomentum())/2,

retLSlope, retSSlope);

}

public void Mutate() {

int rnd = Func.rnd(1, 100);

if(rnd>0 && rnd<=46)

Mutation.ConnInv(this);

if(rnd>46 && rnd<=92)

Mutation.NeurInv(this);

if(rnd>92 && rnd<=94)

Mutation.LayerCopy(this);

if(rnd>94 && rnd<=96)

Mutation.LayerSplit(this);

if(rnd>96 && rnd<=98)

Mutation.LayerConvlt(this);

if(rnd>98 && rnd<=100)

Mutation.LayerMerge(this);

Mutation.LRChange(this);

Mutation.MomentChange(this);

Mutation.LSChange(this);

Mutation.SSChange(this);

}

public static Gene TransformFromNet(NeuralNetwork net) {

boolean[][][] Conn = new boolean[getMAX\_LAYERS()][1+getMAX\_NEUR\_IN\_LAY()][1+getMAX\_NEUR\_IN\_LAY()];

int IN = net.getLayerAt(0).getNeuronsCount();

int ON = net.getLayerAt(net.getLayersCount()-1).getNeuronsCount();

for(int i=0; i<net.getLayersCount()-1; i++) {

Conn[i][0][0] = true;

for(int j=0; j<net.getLayerAt(i+1).getNeuronsCount(); j++) {

Conn[i][1+j][0] = true;

for(int k=0; k<net.getLayerAt(i).getNeuronsCount(); k++)

if(net.getLayerAt(i+1).getNeuronAt(j).

hasInputConnectionFrom(net.getLayerAt(i).getNeuronAt(k)))

Conn[i][1+j][1+k] = true;

}

}

Gene g = new Gene(Conn,IN,ON,0.2,0.0,1.0,1.0);

return g;

}

public NeuralNetwork TransformToNet() {

NeuralNetwork net = new NeuralNetwork();

net.addLayer(0, new Layer(getInputNum(),new NeuronProperties(InputNeuron.class)));

Linear L = new Linear(this.LinearSlope);

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

net.getLayerAt(0).getNeuronAt(n).setTransferFunction(L);

net.setInputNeurons(net.getLayerAt(0).getNeurons());

int i=1;

Sigmoid S = new Sigmoid(this.SigmoidSlope);

for(; i<getLayerCount(); i++) {

net.addLayer(i,new Layer(getMAX\_NEUR\_IN\_LAY(),new NeuronProperties(Neuron.class)));

for(int n=0; n<getMAX\_NEUR\_IN\_LAY(); n++)

net.getLayerAt(i).getNeuronAt(n).setTransferFunction(S);

}

net.addLayer(i,new Layer(getOutputNum(),new NeuronProperties(Neuron.class)));

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

net.getLayerAt(i).getNeuronAt(n).setTransferFunction(S);

net.setOutputNeurons(net.getLayerAt(i).getNeurons());

int LayerNum=0;

for(; LayerNum<getLayerCount()-1; LayerNum++) {

for(i=0; i<getMAX\_NEUR\_IN\_LAY(); i++)

for(int j=0; j<getMAX\_NEUR\_IN\_LAY(); j++)

if(LayerNum > 0) {

if( getConnections()[LayerNum] [1+i][0]&

getConnections()[LayerNum] [1+i][1+j]&

getConnections()[LayerNum-1] [1+j][0]==true)

ConnectionFactory.createConnection(

net.getLayerAt(LayerNum).getNeuronAt(j),

net.getLayerAt(LayerNum+1).getNeuronAt(i));

}

else {

if( getConnections()[LayerNum][1+i][0]&

getConnections()[LayerNum][1+i][1+j] &

(j<InputNum) ==true) {

ConnectionFactory.createConnection(

net.getLayerAt(LayerNum).getNeuronAt(j),

net.getLayerAt(LayerNum+1).getNeuronAt(i));

}

}

}

for(i=0; i<getMAX\_NEUR\_IN\_LAY(); i++)

for(int j=0; j<getMAX\_NEUR\_IN\_LAY(); j++)

if( getConnections()[LayerNum] [1+i][0]&

getConnections()[LayerNum] [1+i][1+j] &

getConnections()[LayerNum-1][1+j][0] &

(i<OutputNum) ==true)

ConnectionFactory.createConnection(

net.getLayerAt(LayerNum).getNeuronAt(j),

net.getLayerAt(LayerNum+1).getNeuronAt(i));

for(i=0; i<getLayerCount()-1; i++)

for(int j=getMAX\_NEUR\_IN\_LAY()-1; j>=0; j--)

//if(getConnections()[i][1+j][0] == false)

if(!net.getLayerAt(i+1).getNeuronAt(j).hasInputConnections() ||

net.getLayerAt(i+1).getNeuronAt(j).getOutConnections().isEmpty())

net.getLayerAt(i+1).removeNeuronAt(j);

return net;

}

/\*\*

\* @return the Connections

\*/

public boolean[][][] getConnections() {

return Connections;

}

/\*\*

\* @param Connections the Connections to set

\*/

public void setConnections(boolean[][][] Connections) {

this.Connections = Connections;

}

/\*\*

\* Get the value of InputNum

\*

\* @return the value of InputNum

\*/

public int getInputNum() {

return InputNum;

}

/\*\*

\* Set the value of InputNum

\*

\* @param InputNum new value of InputNum

\*/

public void setInputNum(int InputNum) {

this.InputNum = InputNum;

}

/\*\*

\* Get the value of OutputNum

\*

\* @return the value of OutputNum

\*/

public int getOutputNum() {

return OutputNum;

}

/\*\*

\* Set the value of OutputNum

\*

\* @param OutputNum new value of OutputNum

\*/

public void setOutputNum(int OutputNum) {

this.OutputNum = OutputNum;

}

/\*\*

\* Get the value of LearRate

\*

\* @return the LearRate

\*/

public double getLearRate() {

return LearRate;

}

/\*\*

\* Set the value of LearRate

\*

\* @param LearRate the LearRate to set

\*/

public void setLearRate(double LearRate) {

this.LearRate = LearRate;

}

/\*\*

\* Get the value of Momentum

\*

\* @return the Momentum

\*/

public double getMomentum() {

return Momentum;

}

/\*\*

\* Set the value of Momentum

\*

\* @param Momentum the Momentum to set

\*/

public void setMomentum(double Momentum) {

this.Momentum = Momentum;

}

/\*\*

\* @return the LinearSlope

\*/

public double getLinearSlope() {

return LinearSlope;

}

/\*\*

\* @param LinearSlope the LinearSlope to set

\*/

public void setLinearSlope(double LinearSlope) {

this.LinearSlope = LinearSlope;

}

/\*\*

\* @return the SigmoidSlope

\*/

public double getSigmoidSlope() {

return SigmoidSlope;

}

/\*\*

\* @param SigmoidSlope the SigmoidSlope to set

\*/

public void setSigmoidSlope(double SigmoidSlope) {

this.SigmoidSlope = SigmoidSlope;

}

/\*\*

\* @return the MAX\_LAYERS

\*/

public static int getMAX\_LAYERS() {

return MAX\_LAYERS;

}

/\*\*

\* @return the MAX\_NEUR\_IN\_LAY

\*/

public static int getMAX\_NEUR\_IN\_LAY() {

return MAX\_NEUR\_IN\_LAY;

}

}

**Individ.java:**

package ann.optimizer;

import java.util.Arrays;

public class Individ {

private Gene g;

private double fit;

public Individ(Gene g, double fit) {

this.g = g;

this.fit = fit;

}

/\*\*

\* @return the g

\*/

public Gene getG() {

return g;

}

/\*\*

\* @param g the Gene to set

\*/

public void setG(Gene g) {

this.g = g;

}

/\*\*

\* @return the fit

\*/

public double getFit() {

return fit;

}

/\*\*

\* @param fit the fitness to set

\*/

public void setFit(double fit) {

this.fit = fit;

}

public static void Sort(Individ[] Ind){

Arrays.sort(Ind, (Individ o1, Individ o2) -> {

if (o1.fit > o2.fit) return -1;

if (o1.fit < o2.fit) return 1;

return 0;

});

}

}

**Mutation.java:**

package ann.optimizer;

/\*\*

\* Contains static functions which implement mutations of Gene.

\* @author M. Kofman

\*/

public class Mutation {

/\*\*

\* Inverse value of random connection between two neurons.

\* If there are less than two connections with value 1,

\* it set random connection to 1.

\* @param g Gene to mutate

\*/

public static void ConnInv(Gene g){

System.out.println("MUT:ConnInv");

/\*random layer\*/

int rnd = Func.rnd(0, g.getLayerCount()-1);

/\*random neuron\*/

int rnd2;

if(g.getLastActNeur(rnd)<30)

rnd2 = Func.rnd(0, g.getLastActNeur(rnd)+1);

else

rnd2 = Func.rnd(0, g.getLastActNeur(rnd));

/\*random connection\*/

int rnd3;

if(rnd>0)

//if(g.getLastActNeur(rnd-1)<30)

// rnd3 = Func.rnd(0, g.getLastActNeur(rnd-1)+1);

//else

rnd3 = Func.rnd(0, g.getLastActNeur(rnd-1));

else

rnd3 = Func.rnd(0,g.getInputNum()-1);

if(g.isConnDeletable(rnd))

g.getConnections()[rnd][1+rnd2][1+rnd3]^=true;

else

g.getConnections()[rnd][1+g.getLastActNeur(rnd)+1][1+rnd3]=true;//g.getConnections()[rnd][1+rnd2][1+rnd3]=true;

}

/\*\*

\* Inverse state of neuron in random hidden layer.

\* If there are less than two neurons with value 1,

\* it set random neuron to 1.

\* @param g Gene to mutate

\*/

public static void NeurInv(Gene g) {

System.out.println("MUT:NeurInv");

int rnd = Func.rnd(0, g.getLayerCount()-2);

if(g.getLayerCount()>1){

/\*random hidden layer\*/

if(Func.rnd(1, 10)<6 && g.isNeurDeletable(rnd)) {

int rnd2;

/\*random neuron \*/

if(g.getLastActNeur(rnd)<30)

rnd2 = Func.rnd(0, g.getLastActNeur(rnd)+1);

else

rnd2 = Func.rnd(0, g.getLastActNeur(rnd));

g.getConnections()[rnd][1+rnd2][0]^=true;

}

else {

if(g.getLastActNeur(rnd)<30) {

int neur=g.getLastActNeur(rnd)+1;

g.getConnections()[rnd][1+neur][0]=true;

int rnd2;

if(rnd>0)

rnd2 = Func.rnd(0, g.getLastActNeur(rnd-1));

else

rnd2 = Func.rnd(0, g.getInputNum()-1);

int rnd3 = Func.rnd(0, g.getLastActNeur(rnd+1));

g.getConnections()[rnd][1+neur][1+rnd2]=true;

g.getConnections()[rnd+1][1+rnd3][1+neur]=true;

}

}

}

}

/\*\*

\* Copy-past one of the hidden layers if there are free place for it.

\* @param g Gene to mutate

\*/

public static void LayerCopy(Gene g) {

System.out.println("MUT:LayerCopy");

if(g.getLayerCount()<10){

/\*random hidden layer\*/

int rnd = Func.rnd(0, g.getLayerCount()-2);

/\*copying layers\*/

for(int i=g.getLayerCount()-1; i>=rnd; i--)

g.getConnections()[i+1]=g.getConnections()[i];

}

}

/\*\*

\* Split one of the hidden layers for two layers

\* if there are free place for it.

\* @param g Gene to mutate

\*/

public static void LayerSplit(Gene g) {

System.out.println("MUT:LayerSplit");

if(g.getLayerCount()<10){

/\*random hidden layer\*/

int rnd = Func.rnd(0, g.getLayerCount()-2);

if(g.isLaySplitable(rnd)){

/\*copying layers\*/

for(int i=g.getLayerCount()-1; i>rnd; i--)

for(int j=0; j<1+Gene.getMAX\_NEUR\_IN\_LAY(); j++)

g.getConnections()[i+1][j]=g.getConnections()[i][j];

/\*free the layer\*/

for(int i=0; i<Gene.getMAX\_NEUR\_IN\_LAY(); i++)

for(int j=0; j<Gene.getMAX\_NEUR\_IN\_LAY()+1; j++)

g.getConnections()[rnd+1][1+i][j] = false;

/\*copying half of the layer\*/

for(int i=g.getLastActNeur(rnd)/2, k=0;

i<g.getLastActNeur(rnd); i++, k++)

for(int j=0; j<1+Gene.getMAX\_NEUR\_IN\_LAY(); j++) {

g.getConnections()[rnd+1][1+k][j]=

g.getConnections()[rnd]

[1+i][j];

g.getConnections()[rnd]

[1+i][j] = false;

}

}

}

}

/\*\*

\* Convolve two layers into one. "Throws" connections from previos layer to

\* next if they are passing through current layer.

\* @param g Gene to mutate

\*/

public static void LayerConvlt(Gene g) {

System.out.println("MUT:LayerConvlt");

if(g.getLayerCount()>2){

/\*random hidden & not previos to output or next to input layer\*/

int rnd = Func.rnd(1, g.getLayerCount()-2);

/\*save new layer configuration\*/

boolean tempLayer [][] =

new boolean[1+Gene.getMAX\_NEUR\_IN\_LAY()][1+Gene.getMAX\_NEUR\_IN\_LAY()];

tempLayer[0][0] = true;

for(int i=0; i<g.getLastActNeur(rnd+1); i++)

tempLayer[1+i][0] = g.getConnections()[rnd+1][1+i][0];

for(int i=0; i<g.getLastActNeur(rnd+1); i++)

for(int j=0; j<g.getLastActNeur(rnd); j++)

for(int k=0; k<g.getLastActNeur(rnd-1); k++)

if(g.getConnections()[rnd+1][1+i][1+j] &

g.getConnections()[rnd][1+j][1+k])

tempLayer[1+i][1+k] = true;

/\*write new layer and shift the rest\*/

g.getConnections()[rnd] = tempLayer;

for(int i=rnd+1; i<g.getLayerCount()-1; i++)

g.getConnections()[i] = g.getConnections()[i+1];

/\*clean last layer\*/

for(int i=Gene.getMAX\_NEUR\_IN\_LAY(); i>=0; i--)

for(int j=Gene.getMAX\_NEUR\_IN\_LAY(); j>=0; j--)

g.getConnections()[g.getLayerCount()-1][i][j] = false;

}

}

/\*\*

\* Merge two layers into one. One half of neurons in one layer becomes other

\* half in other layer.

\* @param g Gene to mutate

\*/

public static void LayerMerge(Gene g) {

System.out.println("MUT:LayerMerge");

if(g.getLayerCount()>2){

/\*random hidden & not previos to output or next to input layer\*/

int rnd = Func.rnd(1, g.getLayerCount()-2);

/\*save new layer configuration\*/

boolean tempLayer [][] =

new boolean[1+Gene.getMAX\_NEUR\_IN\_LAY()][1+Gene.getMAX\_NEUR\_IN\_LAY()];

tempLayer[0][0] = true;

int k=0;

for(int i=0; i<g.getLastActNeur(rnd)/2 && k<Gene.getMAX\_NEUR\_IN\_LAY(); i++, k++)

for(int j=0; j<1+Gene.getMAX\_NEUR\_IN\_LAY(); j++)

tempLayer[1+i][j] = g.getConnections()[rnd][1+i][j];

for(int i=g.getLastActNeur(rnd+1)/2; i<g.getLastActNeur(rnd+1) && k<Gene.getMAX\_NEUR\_IN\_LAY(); i++, k++)

for(int j=0; j<1+Gene.getMAX\_NEUR\_IN\_LAY(); j++)

tempLayer[1+i][j] = g.getConnections()[rnd+1][1+i][j];

/\*write new layer and shift the rest\*/

g.getConnections()[rnd] = tempLayer;

for(int i=rnd+1; i<g.getLayerCount()-1; i++)

g.getConnections()[i] = g.getConnections()[i+1];

/\*clean last layer\*/

for(int i=Gene.getMAX\_NEUR\_IN\_LAY(); i>=0; i--)

for(int j=Gene.getMAX\_NEUR\_IN\_LAY(); j>=0; j--)

g.getConnections()[g.getLayerCount()-1][i][j] = false;

}

}

public static void MomentChange(Gene g) {

int rnd = Func.rnd(0, 1);

if(rnd==0) {

if(g.getMomentum()<0.9)

g.setMomentum(g.getMomentum()+0.1);

else

g.setMomentum(g.getMomentum()-0.1);

}

else {

if(g.getMomentum()>=0.1)

g.setMomentum(g.getMomentum()-0.1);

else

g.setMomentum(g.getMomentum()+0.1);

}

}

public static void LRChange(Gene g) {

int rnd = Func.rnd(0, 1);

if(rnd==0) {

if(g.getLearRate()<0.95)

g.setLearRate(g.getLearRate()+0.05);

else

g.setLearRate(g.getLearRate()-0.05);

}

else {

if(g.getLearRate()>=0.05)

g.setLearRate(g.getLearRate()-0.05);

else

g.setLearRate(g.getLearRate()+0.05);

}

}

public static void LSChange(Gene g) {

int rnd = Func.rnd(0, 1);

if(rnd==0) {

if(g.getLinearSlope()<0.9)

g.setLinearSlope(g.getLinearSlope()+0.1);

else

g.setLinearSlope(g.getLinearSlope()-0.1);

}

else {

if(g.getLinearSlope()>=0.1)

g.setLinearSlope(g.getLinearSlope()-0.1);

else

g.setLinearSlope(g.getLinearSlope()+0.1);

}

}

public static void SSChange(Gene g) {

int rnd = Func.rnd(0, 1);

if(rnd==0) {

if(g.getSigmoidSlope()<0.9)

g.setSigmoidSlope(g.getSigmoidSlope()+0.1);

else

g.setSigmoidSlope(g.getSigmoidSlope()-0.1);

}

else {

if(g.getSigmoidSlope()>=0.1)

g.setSigmoidSlope(g.getSigmoidSlope()-0.1);

else

g.setSigmoidSlope(g.getSigmoidSlope()+0.1);

}

}

}

**Population.java:**

package ann.optimizer;

import org.neuroph.core.NeuralNetwork;

//import org.neuroph.nnet.learning.BackPropagation;

import org.neuroph.nnet.learning.MomentumBackpropagation;

public class Population {

private Gene[] G;

private CompData CD;

private Individ[] Ind;

Population(Gene[] G, CompData CD) {

this.G=G;

this.CD=CD;

}

Population(Gene Init, int Size, CompData CD){

G = new Gene[Size];

G[0] = new Gene(Init);

for(int i=1; i<Size; i++) {

G[i] = new Gene(Init);

G[i].Mutate();

}

this.CD=CD;

}

public double getFitness(int Num){

NeuralNetwork net = this.getG()[Num].TransformToNet();

net.reset();

//BackPropagation train = new BackPropagation();

MomentumBackpropagation train = new MomentumBackpropagation();

net.setLearningRule(train);

train.setLearningRate(this.getG()[Num].getLearRate());

train.setMomentum(this.getG()[Num].getMomentum());

train.learn(getCD().getTrainingSet(),getCD().getFitError(),getCD().getMaxIter());

if(train.getCurrentIteration()==getCD().getMaxIter())

System.out.println("Maximum of iterations reached!");

System.out.print("Net #"+Num+

" LR: "+String.format(java.util.Locale.ENGLISH,

"%(.2f",getG()[Num].getLearRate())+

" Mmntm: "+String.format(java.util.Locale.ENGLISH,

"%(.2f",getG()[Num].getMomentum())+

" LS: "+getG()[Num].getLinearSlope()+

" SS: "+getG()[Num].getSigmoidSlope()+" ");

return -1\*Func.calculateError(net,getCD().getControlSet());

}

private void InitFit() {

/\* The fitness function is evaluated for each individual,

\* providing fitness values, which are then normalized

\*/

setInd(new Individ[this.getG().length]);

double sumFit=0;

for(int i=0; i<this.getG().length; i++){

getInd()[i] = new Individ(this.getG()[i],this.getFitness(i));

sumFit+=getInd()[i].getFit();

}

for(int i=0; i<getInd().length; i++)

getInd()[i].setFit(getInd()[i].getFit()/sumFit);

/\* The population is sorted by descending fitness values\*/

Individ.Sort(getInd());

/\* Accumulated normalized fitness values are computed\*/

for(int i=getInd().length-1; i>=0; i--) {

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

getInd()[i].setFit(getInd()[i].getFit()+getInd()[j].getFit());

}

}

private Gene GetParent() {

/\* A random number rnd between 0 and 1 is chosen\*/

double rnd = Math.random();

/\* The selected individual is the first one whose

\* accumulated normalized value is greater than R

\*/

int i=0;

for(; i<getInd().length; i++)

if(getInd()[i].getFit()>rnd)

break;

return getInd()[i].getG();

}

private Gene[] NewGen() {

Gene[] NG = this.getG();

InitFit();

NG[0] = getInd()[getInd().length-1].getG();

for(int i=1; i<this.getG().length; i++){

NG[i] = Gene.Crossover(GetParent(), GetParent());

NG[i].Mutate();

}

return NG;

}

public void Iterate() {

this.setG(NewGen());

}

public void PrintGene(int Num) {

for(int i=0; i<Gene.getMAX\_LAYERS(); i++) {

System.out.print("L# "+i+" = "+(getG()[Num].getConnections()[i][0][0] ?1:0)+"\n");

for(int j=1; j<Gene.getMAX\_NEUR\_IN\_LAY()+1; j++) {

System.out.print("\tN# "+j+" = "+(getG()[Num].getConnections()[i][j][0] ?1:0)+"\n\t\t");

for(int k=1; k<Gene.getMAX\_NEUR\_IN\_LAY()+1; k++){

System.out.print((getG()[Num].getConnections()[i][j][k] ?1:0)+" ");

}

System.out.print("\n");

}

}

}

/\*\*

\* @return the G

\*/

public Gene[] getG() {

return G;

}

/\*\*

\* @param G the G to set

\*/

public void setG(Gene[] G) {

this.G = G;

}

/\*\*

\* @return the CD

\*/

public CompData getCD() {

return CD;

}

/\*\*

\* @param CD the CD to set

\*/

public void setCD(CompData CD) {

this.CD = CD;

}

/\*\*

\* @return the Ind

\*/

public Individ[] getInd() {

return Ind;

}

/\*\*

\* @param Ind the Ind to set

\*/

public void setInd(Individ[] Ind) {

this.Ind = Ind;

}

}

# **ПРИЛОЖЕНИЕ В**

**Консольный вывод при первом использовании программы оптимизации:**

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:LayerSplit

MUT:NeurInv

MUT:LayerConvlt

MUT:ConnInv

MUT:ConnInv

Pop #0

Net #0 LR: 0.20 Mmntm: 0.00 LS: 1.0 SS: 1.0 Err: 0.35472388019528595

Net #1 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.5370826186417089

Net #2 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.5539020462120823

Net #3 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.4102007203599252

Net #4 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.6784103169643687

Net #5 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.5289744381694497

Net #6 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.2683489117500107

Net #7 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.29784398219965563

Net #8 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.4476525833872258

Net #9 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.39589811972746297

Net #10 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.4682084878853305

Net #11 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.47272287266816243

Maximum of iterations reached!

Net #12 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 1.1071868976908539

Net #13 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.7539839145511895

Net #14 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.6559834140412736

Net #15 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.5344820805313125

Net #16 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.4860296948678258

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:ConnInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:ConnInv

Pop #1

Net #0 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.3307887227645651

Maximum of iterations reached!

Net #1 LR: 0.15 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 1.4690441230282272

Net #2 LR: 0.20 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 0.45791126129912224

Net #3 LR: 0.20 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 0.6853824928846232

Net #4 LR: 0.25 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 0.508150580763582

Net #5 LR: 0.15 Mmntm: 0.00 LS: 0.8 SS: 0.8 Err: 0.4917141526680705

Net #6 LR: 0.25 Mmntm: 0.00 LS: 0.8 SS: 0.8 Err: 0.47301527055079645

Net #7 LR: 0.25 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 0.49852283081437015

Net #8 LR: 0.18 Mmntm: 0.15 LS: 0.85 SS: 0.85 Err: 0.8228422995370416

Net #9 LR: 0.10 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 0.5199568939723884

Net #10 LR: 0.10 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 0.3680757761353668

Net #11 LR: 0.18 Mmntm: 0.15 LS: 0.85 SS: 0.85 Err: 0.34964595993272957

Maximum of iterations reached!

Net #12 LR: 0.15 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 0.8157661301180302

Net #13 LR: 0.15 Mmntm: 0.00 LS: 0.8 SS: 0.8 Err: 0.9827408075202689

Maximum of iterations reached!

Net #14 LR: 0.15 Mmntm: 0.00 LS: 0.8 SS: 0.8 Err: 0.9564952536411232

Net #15 LR: 0.15 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 0.5843985390579293

Net #16 LR: 0.15 Mmntm: 0.00 LS: 0.8 SS: 0.8 Err: 1.002284186855223

MUT:ConnInv

MUT:ConnInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:ConnInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:ConnInv

MUT:NeurInv

Pop #2

Net #0 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.5692333276207234

Maximum of iterations reached!

Net #1 LR: 0.20 Mmntm: 0.10 LS: 0.9 SS: 0.7000000000000001 Err: 0.9353035579116505

Maximum of iterations reached!

Net #2 LR: 0.20 Mmntm: 0.20 LS: 0.9 SS: 0.7000000000000001 Err: 1.0903579691363101

Net #3 LR: 0.20 Mmntm: 0.30 LS: 0.7000000000000001 SS: 0.7000000000000001 Err: 0.6127381146995533

Net #4 LR: 0.25 Mmntm: 0.10 LS: 0.7000000000000001 SS: 0.7000000000000001 Err: 0.7733540600769345

Maximum of iterations reached!

Net #5 LR: 0.10 Mmntm: 0.10 LS: 0.7000000000000001 SS: 0.7000000000000001 Err: 1.6201602868451788

Maximum of iterations reached!

Net #6 LR: 0.15 Mmntm: 0.20 LS: 0.7000000000000001 SS: 0.7000000000000001 Err: 0.8147508941054455

Net #7 LR: 0.08 Mmntm: 0.00 LS: 0.7000000000000001 SS: 0.7000000000000001 Err: 0.7049945288609462

Net #8 LR: 0.13 Mmntm: 0.20 LS: 0.9 SS: 0.7000000000000001 Err: 0.7003485362452311

Maximum of iterations reached!

Net #9 LR: 0.08 Mmntm: 0.30 LS: 0.9 SS: 0.7000000000000001 Err: 0.9251634292487537

Net #10 LR: 0.20 Mmntm: 0.10 LS: 0.7000000000000001 SS: 0.9 Err: 0.6003550820741822

Maximum of iterations reached!

Net #11 LR: 0.10 Mmntm: 0.00 LS: 0.7000000000000001 SS: 0.9 Err: 1.3942240051662484

Net #12 LR: 0.20 Mmntm: 0.20 LS: 0.9 SS: 0.7000000000000001 Err: 0.7161467706024504

Net #13 LR: 0.20 Mmntm: 0.10 LS: 0.9 SS: 0.7000000000000001 Err: 0.3493933005748492

Net #14 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.36369587800958103

Net #15 LR: 0.25 Mmntm: 0.00 LS: 0.9 SS: 0.9 Err: 0.2584204233263879

Maximum of iterations reached!

Net #16 LR: 0.23 Mmntm: 0.00 LS: 0.7000000000000001 SS: 0.7000000000000001 Err: 0.6679934756593068

MUT:ConnInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:ConnInv

MUT:ConnInv

MUT:NeurInv

MUT:LayerCopy

MUT:ConnInv

MUT:ConnInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

Pop #3

Net #0 LR: 0.25 Mmntm: 0.00 LS: 0.9 SS: 0.9 Err: 0.7760644349944777

Maximum of iterations reached!

Net #1 LR: 0.23 Mmntm: 0.20 LS: 0.6000000000000001 SS: 0.6000000000000001 Err: 1.253998746071698

Maximum of iterations reached!

Net #2 LR: 0.25 Mmntm: 0.25 LS: 0.8 SS: 0.6000000000000001 Err: 1.1344119969192923

Net #3 LR: 0.15 Mmntm: 0.15 LS: 0.9 SS: 0.6000000000000001 Err: 0.5903421216452178

Maximum of iterations reached!

Net #4 LR: 0.15 Mmntm: 0.10 LS: 0.8 SS: 0.8 Err: 1.4974677650607444

Maximum of iterations reached!

Net #5 LR: 0.21 Mmntm: 0.10 LS: 0.6000000000000001 SS: 0.9 Err: 1.4274594331029504

Maximum of iterations reached!

Net #6 LR: 0.03 Mmntm: 0.25 LS: 0.7000000000000001 SS: 0.8 Err: 0.915528511380436

Maximum of iterations reached!

Net #7 LR: 0.15 Mmntm: 0.15 LS: 0.8 SS: 0.7000000000000001 Err: 1.4274683035893148

Maximum of iterations reached!

Net #8 LR: 0.19 Mmntm: 0.15 LS: 0.8 SS: 0.8 Err: 0.7404114401602444

Net #9 LR: 0.28 Mmntm: 0.10 LS: 0.9 SS: 0.7000000000000001 Err: 0.5282160698436402

Maximum of iterations reached!

Net #10 LR: 0.13 Mmntm: 0.20 LS: 0.7000000000000001 SS: 0.7000000000000001 Err: 1.75

Maximum of iterations reached!

Net #11 LR: 0.10 Mmntm: 0.20 LS: 0.9 SS: 0.8 Err: 1.75

Maximum of iterations reached!

Net #12 LR: 0.21 Mmntm: 0.15 LS: 0.6000000000000001 SS: 0.6000000000000001 Err: 0.8922172188550469

Net #13 LR: 0.19 Mmntm: 0.15 LS: 0.9 SS: 0.6000000000000001 Err: 0.25585164959430234

Maximum of iterations reached!

Net #14 LR: 0.15 Mmntm: 0.35 LS: 0.8 SS: 0.8 Err: 0.9523236568863515

Maximum of iterations reached!

Net #15 LR: 0.06 Mmntm: 0.20 LS: 0.7000000000000001 SS: 0.9 Err: 1.4107763256802155

Net #16 LR: 0.19 Mmntm: 0.05 LS: 0.6000000000000001 SS: 0.8 Err: 1.0744674410710902

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

Pop #4

Net #0 LR: 0.19 Mmntm: 0.15 LS: 0.9 SS: 0.6000000000000001 Err: 1.4244090221920087

Maximum of iterations reached!

Net #1 LR: 0.16 Mmntm: 0.28 LS: 0.9 SS: 0.65 Err: 1.1013379222266049

Maximum of iterations reached!

Net #2 LR: 0.22 Mmntm: 0.20 LS: 0.6000000000000001 SS: 0.85 Err: 1.4863725267413934

Maximum of iterations reached!

Net #3 LR: 0.14 Mmntm: 0.25 LS: 0.6000000000000001 SS: 0.8 Err: 1.053383811518783

Maximum of iterations reached!

Net #4 LR: 0.18 Mmntm: 0.28 LS: 0.65 SS: 0.9500000000000001 Err: 1.4434692435075813

Maximum of iterations reached!

Net #5 LR: 0.14 Mmntm: 0.30 LS: 0.8 SS: 0.7000000000000001 Err: 1.75

Maximum of iterations reached!

Net #6 LR: 0.14 Mmntm: 0.25 LS: 0.8 SS: 0.85 Err: 0.8442753352524459

Net #7 LR: 0.12 Mmntm: 0.05 LS: 0.7000000000000001 SS: 0.9 Err: 0.30856855792113486

Maximum of iterations reached!

Net #8 LR: 0.23 Mmntm: 0.15 LS: 0.6000000000000001 SS: 0.6000000000000001 Err: 1.3295140398455847

Maximum of iterations reached!

Net #9 LR: 0.24 Mmntm: 0.13 LS: 0.65 SS: 0.5500000000000002 Err: 0.619878422215074

Maximum of iterations reached!

Net #10 LR: 0.23 Mmntm: 0.13 LS: 0.7500000000000001 SS: 0.8 Err: 1.75

Maximum of iterations reached!

Net #11 LR: 0.09 Mmntm: 0.05 LS: 0.85 SS: 0.65 Err: 1.5193152933134002

Maximum of iterations reached!

Net #12 LR: 0.24 Mmntm: 0.25 LS: 0.8 SS: 0.65 Err: 1.75

Maximum of iterations reached!

Net #13 LR: 0.20 Mmntm: 0.05 LS: 0.7500000000000001 SS: 0.5500000000000002 Err: 1.4231445188057152

Maximum of iterations reached!

Net #14 LR: 0.11 Mmntm: 0.30 LS: 0.85 SS: 0.8 Err: 1.43514155271282

Maximum of iterations reached!

Net #15 LR: 0.13 Mmntm: 0.30 LS: 0.9 SS: 0.7500000000000001 Err: 1.5000001557155467

Maximum of iterations reached!

Net #16 LR: 0.13 Mmntm: 0.35 LS: 0.6000000000000001 SS: 0.6000000000000001 Err: 1.578425744266049

MUT:ConnInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:ConnInv

MUT:ConnInv

MUT:ConnInv

MUT:LayerConvlt

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:ConnInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:ConnInv

Pop #5

Net #0 LR: 0.12 Mmntm: 0.05 LS: 0.7000000000000001 SS: 0.9 Err: 0.3812106755755935

Maximum of iterations reached!

Net #1 LR: 0.13 Mmntm: 0.31 LS: 0.675 SS: 0.8750000000000001 Err: 0.9455346488771291

Maximum of iterations reached!

Net #2 LR: 0.22 Mmntm: 0.08 LS: 0.925 SS: 0.7500000000000001 Err: 1.6428571601534572

Maximum of iterations reached!

Net #3 LR: 0.23 Mmntm: 0.26 LS: 0.7250000000000001 SS: 0.7000000000000001 Err: 0.6847026021942596

Maximum of iterations reached!

Net #4 LR: 0.13 Mmntm: 0.18 LS: 0.9249999999999999 SS: 0.8250000000000001 Err: 1.5357142857161707

Maximum of iterations reached!

Net #5 LR: 0.23 Mmntm: 0.31 LS: 0.6750000000000002 SS: 0.65 Err: 1.5357142866393834

Maximum of iterations reached!

Net #6 LR: 0.10 Mmntm: 0.19 LS: 0.9500000000000001 SS: 0.5750000000000001 Err: 1.536297684801866

Maximum of iterations reached!

Net #7 LR: 0.08 Mmntm: 0.18 LS: 0.9249999999999999 SS: 0.9249999999999999 Err: 1.1139739937129691

Maximum of iterations reached!

Net #8 LR: 0.13 Mmntm: 0.34 LS: 0.775 SS: 0.8 Err: 1.5000001931315297

Maximum of iterations reached!

Net #9 LR: 0.13 Mmntm: 0.35 LS: 0.85 SS: 0.7000000000000001 Err: 1.785416878254958

Maximum of iterations reached!

Net #10 LR: 0.28 Mmntm: 0.24 LS: 0.5750000000000001 SS: 0.6000000000000001 Err: 1.75

Maximum of iterations reached!

Net #11 LR: 0.15 Mmntm: 0.08 LS: 0.85 SS: 0.6250000000000001 Err: 0.9590807723409097

Maximum of iterations reached!

Net #12 LR: 0.23 Mmntm: 0.20 LS: 0.8 SS: 0.725 Err: 1.535714285830499

Maximum of iterations reached!

Net #13 LR: 0.22 Mmntm: 0.33 LS: 0.7500000000000001 SS: 0.5500000000000002 Err: 1.395101072879532

Maximum of iterations reached!

Net #14 LR: 0.12 Mmntm: 0.08 LS: 0.7250000000000001 SS: 0.7500000000000001 Err: 1.4349402204622452

Maximum of iterations reached!

Net #15 LR: 0.23 Mmntm: 0.35 LS: 0.6000000000000001 SS: 0.675 Err: 1.6071428572266508

Maximum of iterations reached!

Net #16 LR: 0.13 Mmntm: 0.35 LS: 0.5000000000000001 SS: 0.5000000000000001 Err: 0.9953988888959545

**Консольный вывод при повторном использовании программы оптимизации:**

MUT:NeurInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:ConnInv

MUT:LayerSplit

MUT:ConnInv

Pop #0

Net #0 LR: 0.20 Mmntm: 0.00 LS: 1.0 SS: 1.0 Err: 0.3161540999746858

Net #1 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.265621229009215

Net #2 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.9593809893939286

Net #3 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.5996032087959942

Net #4 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 1.014284399180638

Net #5 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.6685056746259994

Maximum of iterations reached!

Net #6 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.8871000431553998

Maximum of iterations reached!

Net #7 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 1.2188834360537444

Net #8 LR: 0.15 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.7497301847529603

MUT:ConnInv

MUT:NeurInv

MUT:NeurInv

MUT:ConnInv

MUT:ConnInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

Pop #1

Net #0 LR: 0.25 Mmntm: 0.10 LS: 0.9 SS: 0.9 Err: 0.7021804595711911

Maximum of iterations reached!

Net #1 LR: 0.30 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 0.9518651343076081

Maximum of iterations reached!

Net #2 LR: 0.15 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 0.9161621039963996

Maximum of iterations reached!

Net #3 LR: 0.20 Mmntm: 0.00 LS: 0.8 SS: 0.8 Err: 0.9392095848584413

Net #4 LR: 0.10 Mmntm: 0.00 LS: 0.8 SS: 0.8 Err: 0.7923720863680971

Net #5 LR: 0.30 Mmntm: 0.20 LS: 0.8 SS: 0.8 Err: 0.7834585818206048

Maximum of iterations reached!

Net #6 LR: 0.18 Mmntm: 0.15 LS: 0.85 SS: 0.85 Err: 1.1651065483456036

Net #7 LR: 0.13 Mmntm: 0.15 LS: 0.85 SS: 0.85 Err: 0.8074519005514356

Net #8 LR: 0.25 Mmntm: 0.00 LS: 0.8 SS: 0.8 Err: 0.5391798089304166

MUT:NeurInv

MUT:ConnInv

MUT:ConnInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:NeurInv

MUT:LayerConvlt

Pop #2

Net #0 LR: 0.25 Mmntm: 0.00 LS: 0.8 SS: 0.8 Err: 1.1348762350852135

Net #1 LR: 0.14 Mmntm: 0.02 LS: 0.775 SS: 0.775 Err: 1.083080875053487

Maximum of iterations reached!

Net #2 LR: 0.15 Mmntm: 0.25 LS: 0.7500000000000001 SS: 0.7500000000000001 Err: 1.7093817426573974

Maximum of iterations reached!

Net #3 LR: 0.09 Mmntm: 0.18 LS: 0.725 SS: 0.725 Err: 1.7300052496885294

Maximum of iterations reached!

Net #4 LR: 0.29 Mmntm: 0.08 LS: 0.9249999999999999 SS: 0.9249999999999999 Err: 0.5841783468357528

Net #5 LR: 0.16 Mmntm: 0.28 LS: 0.9249999999999999 SS: 0.725 Err: 0.30315228205631894

Net #6 LR: 0.23 Mmntm: 0.15 LS: 0.9500000000000001 SS: 0.9500000000000001 Err: 0.8217072666396764

Net #7 LR: 0.26 Mmntm: 0.28 LS: 0.9249999999999999 SS: 0.725 Err: 0.7642490248551658

Net #8 LR: 0.25 Mmntm: 0.00 LS: 0.7000000000000001 SS: 0.9 Err: 0.9265356599778409

MUT:ConnInv

MUT:NeurInv

MUT:NeurInv

MUT:ConnInv

MUT:NeurInv

MUT:ConnInv

MUT:ConnInv

MUT:ConnInv

Pop #3

Net #0 LR: 0.16 Mmntm: 0.28 LS: 0.9249999999999999 SS: 0.725 Err: 0.30632139228240735

Net #1 LR: 0.19 Mmntm: 0.31 LS: 0.8375 SS: 0.7375 Err: 0.7176021357600904

Maximum of iterations reached!

Net #2 LR: 0.14 Mmntm: 0.30 LS: 0.7500000000000001 SS: 0.7500000000000001 Err: 1.425004167628683

Net #3 LR: 0.14 Mmntm: 0.10 LS: 0.7500000000000001 SS: 0.9500000000000001 Err: 0.6336474908661834

Net #4 LR: 0.31 Mmntm: 0.04 LS: 0.9125 SS: 0.9125 Err: 0.640275431806608

Net #5 LR: 0.29 Mmntm: 0.18 LS: 0.775 SS: 0.775 Err: 0.4217972051140707

Net #6 LR: 0.14 Mmntm: 0.11 LS: 0.8375 SS: 0.9375 Err: 0.8018883556031646

Maximum of iterations reached!

Net #7 LR: 0.32 Mmntm: 0.14 LS: 0.7125 SS: 0.8125 Err: 0.85107603454416

Maximum of iterations reached!

Net #8 LR: 0.22 Mmntm: 0.19 LS: 0.8624999999999999 SS: 0.6625 Err: 0.3909789361396538