

РЕФЕРАТ

Пояснительная записка 85 листов, 25 рисунков, 6 таблиц, 15 источников, 1 приложение.

ИСКУССТВЕННЫЕ НЕЙРОННЫЕ СЕТИ, ГЕНЕТИЧЕСКИЙ АЛГОРИТМ, ПЕРСЕПТРОН, ФОРЕКС, ФИНАНСОВЫЙ РЫНОК, ARTIFICAL NEURON NETWORK, GENETIC ALGORITHM, PERCEPTRON, FOREX

Объектом исследования являются искусственные нейронные сети.

Предметом исследования является процесс поведенческого анализа рынка Форекс с использованием нейронных сетей.

Цель данной работы состояла в разработке подсистемы анализа финансовых временных рядов на основе нейронных сетей различных структур, обученных генетическим алгоритмом.

В результате проделанной работы была разработана и обучена нейронная сеть, которая помогает принимать решения о покупке или продаже соответствующей валюты на основании цен валютных пар за прошедший период времени.

ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ

ИНС — Искусственная Нейронная Сеть

ВР – Временной Ряд

ГА – Генетический Алгоритм

ЭА – Эволюционный Алгоритм

БНС – Биологическая Нейронная Сеть

НС – Нервная Система

ФА – Функция Активации

ГАВК – Генетический алгоритм с вещественным кодированием

ГАДК – Генетический алгоритм с двоичным кодированием

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	7
1.1 Биологическая нейронная сеть	7
1.2 Искусственный нейрон.....	9
1.3 Архитектура нейронных сетей	13
1.4 Обучение ИНС.....	17
1.5 Генетический алгоритм	20
2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ	24
2.1 Реализация нейронной сети	24
2.1.1 Класс Neuron.....	25
2.1.2 Класс NeuronNet.....	27
2.2 Подготовка входных данных	28
2.2.1 Нормализация и распределение.....	30
2.2.2 Формирование исходных файлов.....	34
2.3 Реализация генетического алгоритма	38
2.3.1 Класс Chromosome	38
2.3.2 Класс GA.....	39
2.4 Реализация алгоритма обучения.....	41
3 ИССЛЕДОВАНИЕ И КОРРЕКТИРОВКА СИСТЕМЫ.....	46
3.1 Обучение ИНС разных структур	46
3.1.1 Многослойный персептрон с одним скрытым слоем.....	46
3.1.2 Многослойный персептрон с двумя скрытыми слоями.....	48
3.1.3 Многослойный персептрон с тремя скрытыми слоями	50
3.2 Моделирование работы ИНС на тестовой выборке	52
3.3 Симуляция торгов на рынке Форекс.....	55
ЗАКЛЮЧЕНИЕ	61
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	63
ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ	65

ВВЕДЕНИЕ

Тема искусственного интеллекта, машинного обучения, а также эволюционных алгоритмов невероятно популярна в настоящее время. Рост интереса к данным областям подтверждается статистикой количества соответствующих поисковых запросов, предоставленной сервисом «Google Trends».

Искусственная нейронная сеть (ИНС), как одна из главных технологий моделирования интеллекта с помощью компьютерных алгоритмов, позволяет решать обширный круг задач, таких как распознавание образов, анализ временных рядов (ВР), обработка сигналов и др.

Модель ИНС пытается повторить структура мозга, способ обработки информации которого, в корне отличается от методов, применяемых обычными компьютерами. Мозг, в свою очередь, представляет собой очень сложный, нелинейный, параллельный компьютер, который управляет своими структурными компонентами – нейронами.

При этом нейронные сети не программируются в привычном смысле этого слова, а обучаются. Одним из главных преимуществ использования ИНС является их возможность получать обоснованный результат на основании данных, которые не встречались в процессе обучения [12].

Непосредственно процесс обучения нейронной сети с математической точки зрения представляет собой многопараметрическую задачу нелинейной оптимизации.

Генетический алгоритм (ГА) является разновидностью эволюционных вычислений, с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как наследование, мутации, отбор и кроссинговер [13].

Типичным примером временных рядов, анализ которых является одной из задач искусственных нейронных сетей, можно назвать биржевые курсы на финансовых рынках, то есть цены денежной единицы одной страны,

выраженные в денежной единице другой страны. При анализе финансовых курсов пытаются определить тренд, то есть тенденцию направления развития.

Рынок Форекс, что в переводе с английского означает международный обмен – это рынок межбанковского обмена валюты по свободным ценам, котировкам, значение которых формируется без ограничений продавцом или покупателем [14].

В данной работе предлагается реализовать искусственную нейронную сеть, используемую для прогнозирования тенденции изменения курса американского доллара относительно российского рубля и дальнейшей торговли на рынке Форекс на основании полученных прогнозов. Для обучения нейронной сети предлагается использовать генетический алгоритм.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Искусственные нейронные сети – технология, которая объединяет в себе обширное множество дисциплин, таких как нейрофизиология, статистика, математика, физика, а также различные области информационных технологий. Применение ИНС возможно в следующих направлениях: распознавание образов, анализ временных рядов, моделирование, обработка сигналов и др. [1].

В связи с тем, что модель ИНС построена по принципу организации и функционирования биологических нейронных сетей, в частности мозга человека, то необходимо подробно рассмотреть ее прототип – биологическую нейронную сеть (БНС).

1.1 Биологическая нейронная сеть

Биологическая нейронная сеть – это совокупность нейронов головного и спинного мозга центральной нервной системы и ганглия периферической нервной системы, которые связаны или функционально объединены в нервной системе (НС) и выполняют специфические физиологические функции.

НС человека можно рассматривать, как трехступенчатую, где центром является мозг, представленный сетью нейронов. Первой ступенью являются рецепторы, которые преобразовывают сигналы из окружающей среды в электрические импульсы. Эффекторы преобразуют электрические сигналы, сгенерированные мозгом, в выходные сигналы и являются третьей ступенью нервной системы человека. Мозг же непосредственно анализирует информацию, полученную от рецепторов, и передает результаты ее обработки, соответствующие решения, на эффекторы [1].

На рисунке 1 изображена простейшая диаграмма, описывающая нервную систему, где стрелки, направленные слева направо, обозначают прямую передачу сигналов информации в систему, а стрелки, направленные справа налево, – ответную реакцию системы.



Рисунок 1 - Блочная диаграмма НС человека

Нейрон – это структурно-функциональная единица нервной системы. Клетка нейрона имеет сложное строение и содержит ядро, тело клетки и отростки (дендриты и аксоны). Нейрон обрабатывает, хранит и передает информацию с помощью электрических и химических сигналов.

Дендрит – это короткий разветвленный отросток нейрона. Он получает информацию через синапсы от аксонов других нейронов и передает ее непосредственно телу нейрона.

Аксон – это длинный неветвящийся отросток нейрона. Его основная функция – передача информации от тела нейрона к другим клеткам НС или мышечным волокнам.

Каждый нейрон состоит из одного аксона и нескольких дендритов.

Синапсы – это элементарные структурные и функциональные единицы, которые передают импульсы между нейронами. Передача сигналов происходит с помощью сигнальных веществ, медиаторов, которые выделяются в пространство между пресинаптической и постсинаптической мембранами синапса. Затем вещество диффундирует и воздействует на рецепторы постсинаптической мембраны, тем самым передавая сигнал соседней клетке [8].

В традиционных описаниях нейронной организации синапс представляют простым соединением, которое может передавать возбуждение или торможение (но не то и другое одновременно) между нейронами.

На рисунке 2 схематично изображено строение нейрона.

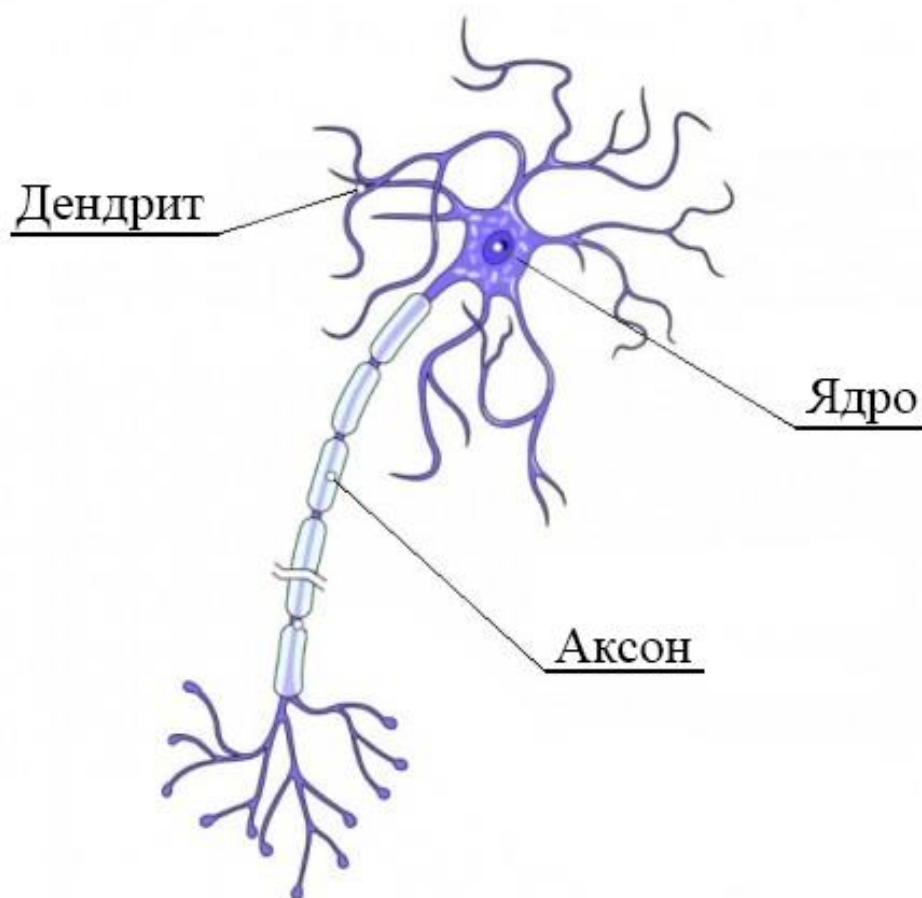


Рисунок 2 - Клетка нейрона

1.2 Искусственный нейрон

Искусственный нейрон представляет собой единицу обработки информации в нейронной сети. На рисунке 3 изображена модель нейрона, лежащего в основе ИНС. В этой модели можно выделить три основных элемента:

1. *Набор синапсов* или *связей*, каждый из которых характеризуется своим весом или силой. Таким образом, сигнал x_j на входе синапса j , связанного с нейроном k , изображенном на схеме, умножается на вес w_{kj} . В дальнейшем в данной работе, при указании индексов весов, первый индекс будет указывать на номер рассматриваемого нейрона, а второй на номер входного синапса этого нейрона. Стоит

обратить внимание, что в отличие от синапсов мозга синаптический вес искусственного нейрона может принимать не только положительные значения, но и отрицательные.

2. *Сумматор* – ядро нейрона. Он складывает входные сигналы, взвешенные относительно соответствующих синапсов нейрона. Фактически, на выходе сумматора получается результат линейной комбинации входов нейрона, где коэффициентами являются веса связей.
3. *Функция активации* (ФА), также называемая функцией сжатия, которая предназначена для ограничения амплитуды выходного сигнала нейрона. Обычно в результате нормализации значение выхода нейрона лежит в интервале $[0;1]$ или $[-1;1]$.

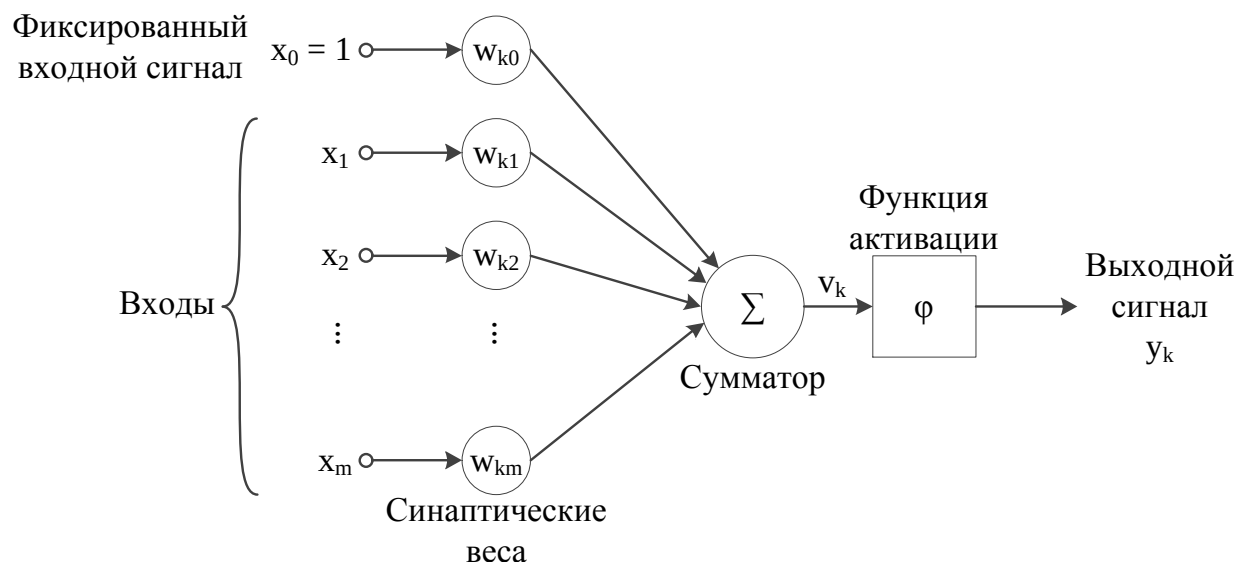


Рисунок 3 - Нелинейная модель нейрона

В модель нейрона, изображенного на рисунке, включен входной сигнал фиксированной величины, равной единице, а также синаптический вес, используемые в качестве порогового элемента. В данном случае значение веса w_{k0} позволяет сдвигать выходное значение нейрона относительно начала координат, то есть фактически обеспечивает эффект аффинного преобразования.

Математическое представление функционирования нейрона k представлено в формулах 1 и 2:

$$v_k = \sum_{j=0}^m w_{kj} x_j \quad (1)$$

$$y_k = \varphi(v_k) \quad (2)$$

где $x_0, x_1, \dots, x_m$ – входные сигналы;

$w_{k0}, w_{k1}, \dots, w_{km}$ – синаптические веса нейрона k ;

v_k – потенциал активации;

φ – функция активации;

y_k – выходной сигнал нейрона.

Функция активации, представленная в формуле 2 как $\varphi(v)$, определяет выходной сигнал нейрона в зависимости от индуцированного локального поля или потенциала активации. Функции активации можно разделить на три основных типа:

1. *Функция единичного скачка*, или *пороговая функция* описывается формулой 3. В соответствии с данным типом нейрон принимает на выходе значение 0, если индуцированное локальное поле этого нейрона отрицательно и 1 в обратном случае.

$$\varphi(v) = \begin{cases} 1, & \text{если } v \geq 0; \\ 0, & \text{если } v < 0; \end{cases} \quad (3)$$

2. *Кусочно-линейная функция* описывается формулой 4. Данный тип можно рассматривать как аппроксимацию нелинейного усилителя.

$$\varphi(v) = \begin{cases} 1, & v \geq \frac{1}{2}; \\ |v|, & \frac{1}{2} > v > \frac{1}{2}; \\ 0, & v \leq \frac{1}{2}; \end{cases} \quad (4)$$

3. *Сигмоидальная функция*, график которой похож на букву S, может описываться формулой 5 (логистическая функция). Данный тип

является самым распространенным и поддерживает баланс между линейным и нелинейным поведением.

$$\varphi(v) = \frac{1}{1+e^{-v}} \quad (5)$$

Различия между всеми типами ФА хорошо видны на рисунке 4. Под буквой «а» изображен график пороговой активационной функции, букве «б» соответствует график кусочно-линейной ФА, буква «в» - сигмоидальная активационная функция.

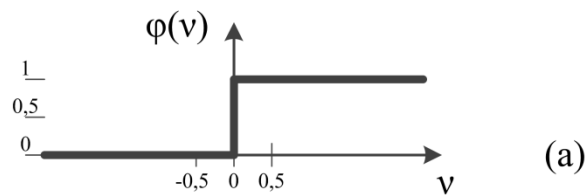
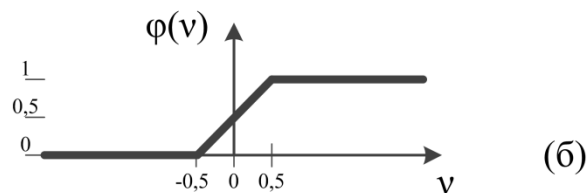
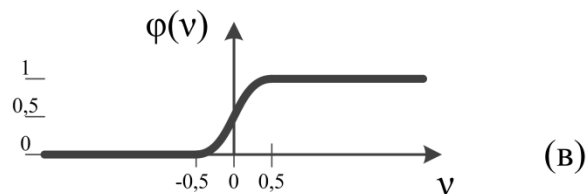


Рисунок 4 - Виды ФА

В формулах 3 – 5 область значений функции активации ограничена отрезком $[0;1]$, однако возможны ситуации, когда выходное значение нейрона должно находиться в диапазоне $[-1;1]$. В этих случаях активационная функция должна быть симметрична относительно начала координат.

1.3 Архитектура нейронных сетей

В общем случае можно выделить три фундаментальных класса нейронных сетей: однослойные сети прямого распространения, многослойные сети прямого распространения и рекуррентные сети.

В многослойной нейронной сети нейроны располагаются по слоям. Если информация внутри сети последовательно передается от первых слоев к последующим, но не наоборот, то такая сеть называется ацикличной или сетью прямого распространения.

В простейшем случае в ИНС существует входной слой нейронов, которые не выполняют никаких вычислений, и выходной слой нейронов, информация на который передается с узлов источника. На рисунке 5 изображена структура такой сети, в каждом из слоев которой находится по три нейрона. Данная ИНС и все сети с аналогичной архитектурой называются однослойными персептронам.

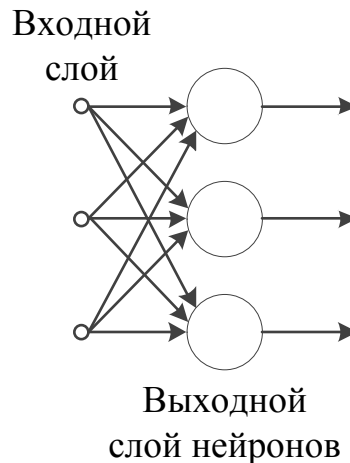


Рисунок 5 - Однослойная сеть прямого распространения

Однослойный персептрон предназначен для классификации линейно-разделимых сигналов и также называется персептроном Розенблатта. Если ИНС данного типа содержит только один нейрон, то такая сеть ограничена решением задачи разделения только двух классов. При увеличении размерности вычислительного слоя персептрона, включая в него несколько

нейронов, можно решать задачи классификации числа классов большего, чем два.

Обычно, в качестве активационной функции однослойных персептронов, используют пороговую функцию, что, фактически, делает поведение нейрона линейным. Однако верным является утверждение, что устойчивое принятие решения однослойным персептроном не зависит от вида нелинейности нейрона. В связи с этим, можно утверждать, что данный вид ИНС способен выполнять классификацию образов для линейно-разделимых классов.

Следующий класс нейронных сетей прямого распространения характеризуется наличием одного или нескольких скрытых слоев, узлы которых называются скрытыми нейронами. Добавление скрытых слоев в архитектуру нейронной сети позволяет выделить статистики высокого порядка. Такие нейронные сети обычно называют многослойными персептронами.

Структура многослойной сети прямого распространения представлена на рисунке 6. Во входном слое изображенной ИНС расположено пять нейронов, в единственном скрытом слое – три нейрона, а в выходном – один. Такая ИНС называется сеть 5-3-1, а в общем случае сеть прямого распространения называют сетью $m-h_1-h_2-q$, где m – количество входов, h_1 – количество нейронов первого уровня, h_2 – количество нейронов второго уровня и q – количество выходов.

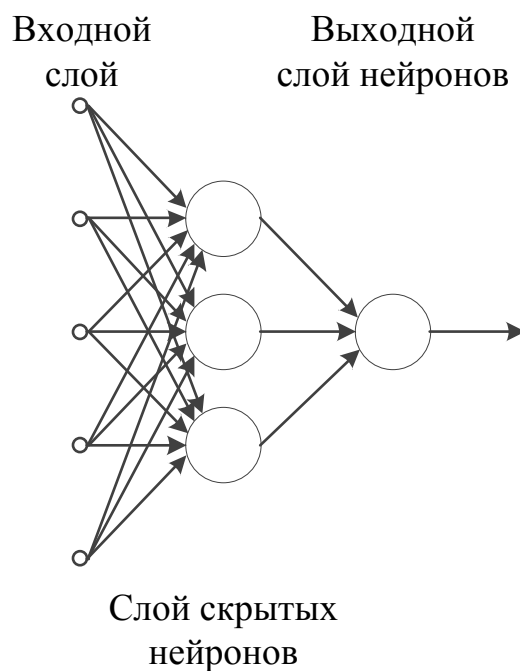


Рисунок 6 - Полносвязная сеть прямого распространения с одним скрытым слоем

Нейронная сеть, изображенная на рисунке 6, является полносвязной, так как все узлы конкретного слоя связаны синапсами со всеми узлами соседних слоев. Сети, для которых данное правило не выполняется, называются неполносвязными.

Многослойные персептроны имеют несколько отличительных признаков:

- Каждый нейрон имеет нелинейную функцию активации. При этом очень важно, чтобы данная функция была всюду дифференцируемой, то есть гладкой, в отличие от пороговой функции, используемой в персептроне Розенблатта. В связи с этим, самой популярной формой функции для многослойного персептрона является сигмоидальная. Наличие нелинейности играет очень важную роль, так как иначе такую ИНС можно будет привести к обычному однослойному персептрону. Наконец, использование логистической функции обусловлено биологическими факторами, так как в ней учитывается восстановительная фаза реального нейрона.

- Сеть содержит один или несколько скрытых слоев нейронов, которые не относятся к входному и выходному слоям. Наличие скрытых нейронов делает возможным обучение решению сложных задач, позволяя последовательно извлекать наиболее важные признаки их входного вектора.
- Сеть обладает высокой степенью связности, которая реализуется за счет большого количества синаптических соединений. Изменение уровня связности требует изменения множества синапсов или их весовых коэффициентов.

Комбинация всех этих качеств наряду со способностью к обучению на собственном опыте обеспечивает вычислительную мощность многослойного персептрона. Тем не менее, эти же качества приводят к неполноте современных знаний о поведении сетей такого рода, так как нелинейность и высокая связность существенно усложняют теоретический анализ многослойного персептрона. Кроме того, наличие скрытых слоев нейронов делает процесс обучения более сложным для визуализации.

Для рекуррентных нейронных сетей характерно наличие хотя бы одной обратной связи. В этом заключается их отличие от сетей прямого распространения. На рисунке 7 изображена ИНС со скрытым слоем, в котором один из нейронов направляет свой выходной сигнал на первый вход нейронной сети.

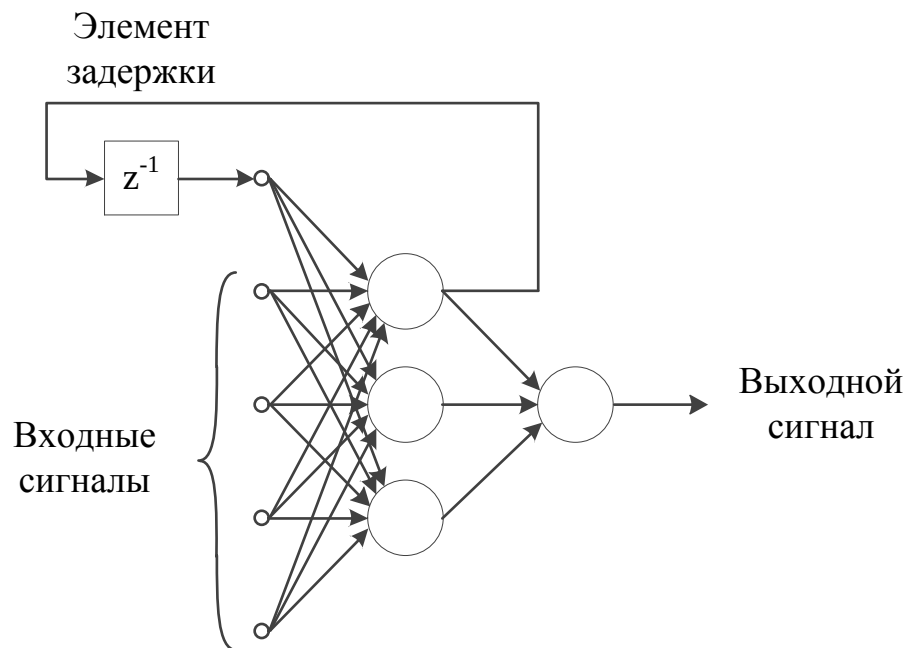


Рисунок 7 - Рекуррентная сеть со скрытым слоем нейронов

Наличие обратных связей оказывает непосредственное влияние на способность сетей к обучению и на их производительность. Кроме того, наличие обратной связи вызывает необходимость использования элементов единичной задержки, что приводит к нелинейному динамическому поведению сети, если в сети содержатся нелинейные нейроны.

1.4 Обучение ИНС

Главным свойством нейронных сетей является их способность обучаться на основе данных окружающей среды и в результате обучения повышать свою производительность. Процесс обучения происходит путем корректировки синаптических весов.

Обучение – это процесс, в котором свободные параметры нейронной сети настраиваются посредством моделирования среды, в которую эта сеть встроена. Тип обучения определяется способом подстройки этих параметров. [1]

В соответствии с данным определением, можно выделить три этапа процесса обучения:

1. В нейронную сеть поступают стимулы из окружающей среды.
2. В результате этого изменяются свободные параметры нейронной сети.
3. После изменения внутренней структуры нейронная сеть отвечает на возбуждения уже иным образом.

Список правил, указанных выше, решает проблему обучения и называется алгоритмом обучения. Не существует универсального алгоритма обучения для всех архитектур нейронных сетей, а каждая единица из их множества имеет собственные преимущества и недостатки.

Существует набор из пяти основных видов обучения:

1. на основе коррекции ошибок;
2. с использованием памяти;
3. Хеббовское обучение;
4. конкурентное обучение;
5. метод Больцмана.

Обучение, основанное на коррекции ошибок, реализует метод оптимальной фильтрации. Обучение на основе памяти предполагает явное использование обучающих данных. Метод Хебба и конкурентный подход к обучению основаны на нейробиологических принципах. В основу метода Больцмана положены идеи статической механики.

Можно выделить две парадигмы обучения нейронных сетей: обучение с учителем и без учителя.

Под наличием учителя можно понимать существование знаний об окружающей среде, которые можно представить в виде пар вход-выход. При этом сама среда неизвестна обучаемой нейронной сети. На рисунке 8 показана блочная диаграмма, иллюстрирующая эту форму обучения.

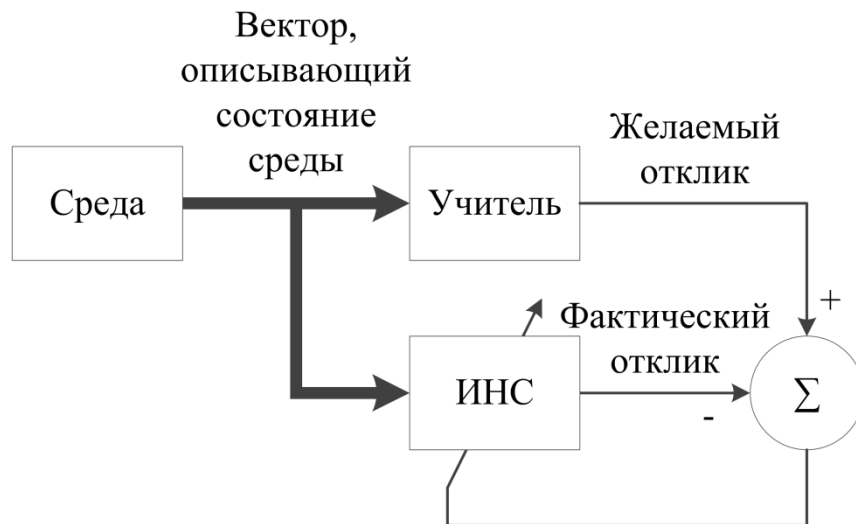


Рисунок 8 - Блочная диаграмма обучения с учителем

Учителю и обучаемой нейронной сети подается обучающий вектор из окружающей среды. На основе собственных знаний учитель может сформировать и передать обучаемой ИНС желаемый отклик, соответствующий данному входному вектору. Этот желаемый отклик представляет собой оптимальные действия, которые должна выполнить нейронная сеть. Параметры сети корректируются в зависимости от сигнала ошибки, который является разностью между желаемым результатом и фактическим откликом ИНС.

Альтернативная парадигма обучения без учителя подразумевает отсутствие внешнего корректора, контролирующего процесс настройки весовых коэффициентов. При использовании этого подхода не существует маркированных примеров, по которым проводится обучение сети, существует лишь независимая от задачи мера качества представления, которому должна научиться нейронная сеть, а свободные параметры сети оптимизируются по отношению к этой мере.

На рисунке 9 изображена блочная диаграмма обучения ИНС без учителя.



Рисунок 9 - Блочная диаграмма обучения без учителя

В данной работе для обучения нейронной сети будет использоваться генетический алгоритм, который относится к парадигме обучения с учителем.

1.5 Генетический алгоритм

Генетический алгоритм – это эволюционный алгоритм поиска решений в задачах оптимизации и адаптации. ГА относится к большому классу эволюционных алгоритмов (ЭА), в котором выделяются:

- генетический алгоритм;
- генетическое программирование;
- эволюционное программирование;
- дифференциальная эволюция;

В основе данных алгоритмов заложены те же процессы, что и в теории эволюции Ч. Дарвина.

Основными принципами алгоритма являются:

- Принцип естественного отбора, предложенный Ч. Дарвином в своей теории эволюции.
- Принцип скрещивания, открытый Г. Менделем в 1865г.
- Принцип мутации, открытый в 1900г. де Вре.

Описанные принципы являются основными в генетическом подходе, благодаря им происходит эволюция особей и поиск лучших решений.

Генетический алгоритм отличается от других методов оптимизации следующими особенностями:

- поиск решения осуществляется из нескольких позиций (особи в популяции);
- использование целевой функции в чистом виде, без использования её производной или другой дополнительной информации;
- применение вероятностных методов;
- устойчивость к попаданию в локальные экстремумы.

Генетический алгоритм широко распространен и успешно используется в качестве основного или вспомогательного метода в различных классах задач, таких как:

- задачи оптимизации;
- настройка и обучение ИНС;
- фильтрация данных;
- распознавание образов;
- аппроксимация функций;
- задача коммивояжера;
- составление расписаний.

Так как основная идея ГА взята из генетики, то и терминология, используемая при описании метода, соответствует данной области науки. Ниже представлены основные термины, используемые при описании генетических алгоритмов:

- *Эпоха* или *поколение* – один полный цикл работы алгоритма.
- *Популяция* – совокупность особей.
- *Особь* – набор хромосом. Каждая особь представляет собой вариант решения задачи.
- *Хромосома* – вектор, совокупность генов.
- *Ген* – любое число или закодированная двоичным кодом информация, минимальная единица хромосомы.

- *Аллель* – совокупность генов.
- *Кроссинговер* или *скрещивание* – обмен генетическим материалом двух или более хромосом.
- *Мутация* – случайное или направленное изменение одного или нескольких генов в хромосоме.
- *Селекция* – выбор особей для скрещивания.
- *Редукция* – удаление менее приспособленных особей.
- *Приспособленность* – степень приспособленности особи.
- *Локус* – позиция гена в хромосоме.

Иерархия объектов генетического алгоритма представлена на рисунке 10.

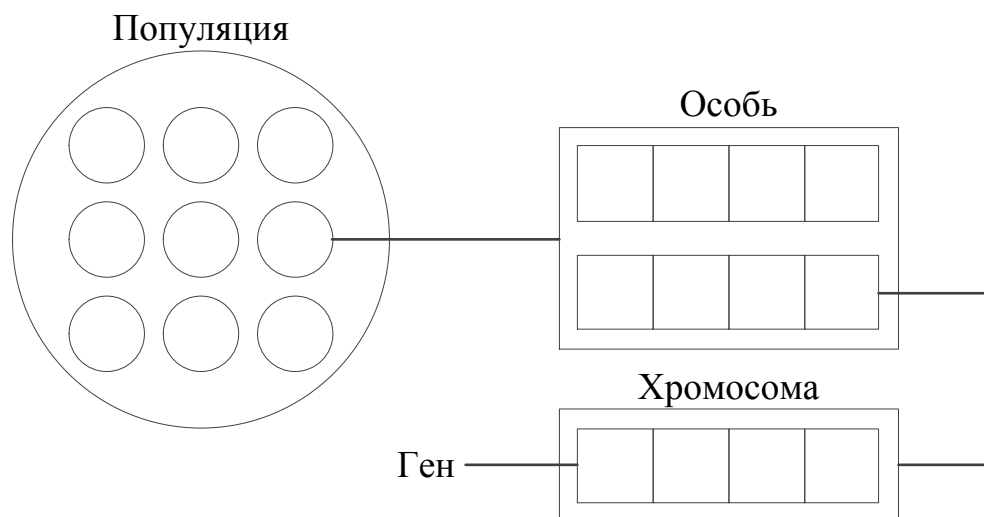


Рисунок 10. Структура популяции

Популяция представляет собой набор особей или потенциальных решений искомой задачи. Каждая особь состоит из набора хромосом. В свою очередь, хромосома представляет собой совокупность генов. В генах закодирована генетическая информация.

Классический генетический алгоритм состоит из следующей последовательности этапов:

1. генерация начальной популяции;
2. оценка приспособленности каждой особи в популяции;

3. отбор родителей для процесса скрещивания;
4. скрещивание пар родителей;
5. мутация новых особей;
6. увеличение размера популяции за счет порожденных потомков;
7. выбор особей и сокращение популяции для следующего поколения;
8. проверка условия завершения алгоритма;
9. выбор наилучшей особи;
10. результат.

Все ГА делятся на два вида:

- Генетический алгоритм с вещественным кодированием (ГАВК), оперирует с числами, что несколько снижает вычислительную нагрузку за счет отсутствия преобразования данных.
- Генетический алгоритм с двоичным кодированием (ГАДК), оперирует с двоичными хромосомами.

2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

В ходе анализа предметной области, было решено разделить обучение нейронной сети на три части.

В первой части будет производиться обучение нескольких ИНС разной структуры генетическим алгоритмом на данных, соответствующих курсам валют за 2010 год. Этот этап позволит отобрать ряд наиболее подходящих для решения поставленной задачи структур нейронной сети, а именно число скрытых слоев ИНС, а также количество нейронов в них.

Второй этап будет заключаться в отборе лучшей структуры из полученных в первой части, путем моделирования работы ИНС на данных, соответствующих курсам валют за 2011 год.

На третьем шаге будет произведена симуляция торгов на рынке Форекс, с использованием данных курсов валют за 2012 год. В результате этого этапа будет проанализирована потенциальная прибыль, полученная за счет предсказаний курса, сделанных при помощи разработанной и обученной на первых двух шагах нейронной сети.

2.1 Реализация нейронной сети

Для реализации нейронной сети был выбран язык программирования C++. С одной стороны данный язык обладает высокой вычислительной производительностью, позволяет быстро и гибко работать с памятью, поддерживает работу с указателями. С другой стороны он в полной мере является объектно-ориентированным, что позволяет использовать преимущества этой парадигмы программирования.

В качестве среды разработки был выбран продукт Visual Studio от Microsoft. Эта среда содержит все необходимые инструменты для разработки

компиляции и отладки программ, а также позволяет настроить работу с локальными и сетевыми репозиториями систем контроля версий.

Исходя из рисунка 3 очевидно, что для создания нейронной сети необходимо реализовать следующие классы:

- Input – вещественное значение входа нейрона или нейронной сети.
- Synapse – вход, соответствующий синапсу, и вес связи.
- Neuron – непосредственно нейрон, содержащий в себе набор синапсов, один выход, а также функции для преобразования сигналов входных связей в выходной.
- NeuronNet – набор входов, и нескольких слоев нейронов, последний из которых является выходным.
- Output – вещественное значение выхода нейрона.

Классы Input и Output содержат в себе только одно публичное поле типа double. Кроме того, в самом начале разработки стало очевидно, что имеет смысл использовать один класс, как для входов, так и для выходов. Это позволит использовать указатель на выход конкретного нейрона в качестве входа для синапса другого. В связи с этим, типы данных Input и Output были сделаны синонимами при помощи директивы typedef.

Класс Synapse тоже фактически является структурой и содержит в себе два публичных поля: вещественный вес связи (double Weight) и указатель на вход синапса (Input* pInput).

2.1.1 Класс Neuron

Класс Neuron содержит два публичных поля: массив входных синапсов (vector<Synapse> InputSynapses) и выход (Output Output).

Для вызова пользователем открыты следующие функции:

- `void ProcessNeuron()`, непосредственно производит расчет выходного значения нейрона, последовательно вызывая закрытые функции сумматора `Adder()` и функции активации `ActivationFunction`.
- `void SetWeights(vector<double>* pWeights)`, вызывается для настройки весов синапсов нейрона во время обучения сети. В данной функции для всех элементов массива входных синапсов устанавливается значение поля `Weight`. Новое значение веса связи берется из вектора, переданного в данную функцию в качестве параметра.

Кроме того, для данного класса реализован конструктор, принимающий в качестве параметров указатель на массив указателей на входы, а также указатель на массив весов: `Neuron(vector<Input*>* pInputs, vector<double>* pWeights)`. Внутри данного конструктора для текущего нейрона создается массив синапсов, после чего, для каждого из них запоминается указатель на соответствующий вход и устанавливается значение веса. Остальные поля устанавливаются в значение по умолчанию.

В приватной секции класса находятся два вещественных поля: потенциал активации (`double ActivationPotential`) и выходное значение (`double OutputValue`).

Закрыты от пользователя следующие функции:

- `void Adder()`, линейная комбинация синапсов. Данная функция в цикле суммирует входы из массива `InputSynapses`, умноженные на соответствующие веса связей. Результат записывается в переменную `ActivationPotential`.
- `void ActivationFunction()`, функция активации нейрона. В данной функции в качестве сигмоиды используется гиперболический тангенс, который вычисляется для значения поля `ActivationPotential`, а результат записывается в поле `OutputValue` и

непосредственно записывается в объект класса Output данного нейрона.

2.1.2 Класс NeuronNet

Класс NeuronNet содержит следующие поля: массив входов ИНС (`vector<Input*> Inputs`), несколько массивов нейронов, количество которых зависит от конфигурации сети (для однослойного персептрона `vector<Neuron> FirstNeuronsLayer`), а также массив выходного слоя нейронов (`vector<Neuron> OutputNeurons`).

Для вызова пользователем доступны следующие функции:

- `void ProcessNet()`, непосредственно отвечает за работу нейронной сети, последовательно в циклах вызывая метод `ProcessNeuron()` для каждого нейрона первого и последующих, а в заключение выходного, слоев нейронов.
- `void SetInputs(vector<Input*>* pInputs)`, вызывается для установки входных значений ИНС. Так как в классе хранятся не значения входов, а указатели на них, то для установки новых значений входов достаточно заменить их по адресу на новые. Переконфигурация нейронной сети при этом не требуется.
- `void SetWeights(vector<vector<vector<double>>>*pWeightsLayers)`, используется для настройки весов нейронов во время обучения сети. Данная функция в нескольких циклах последовательно вызывает метод `SetWeight` для каждого нейрона скрытых и выходного слоев. На вход описываемого метода передается указатель на трехмерный массив, где третий уровень – веса для отдельных синапсов нейрона, второй уровень необходим для отделения весов разных нейронов друг от друга, а третий для разделения нейронов на слои.

Конструктор данного класса принимает на вход указатель на массив указателей на входы нейронной сети, а также указатель на трехмерный массив, аналогичный передаваемому в качестве параметра на вход в метод SetWeights: `NeuronNet(vector<Input*>* pInputs , vector<vector<vector<double>>>* pWeightsLayers`. Внутри конструктора запоминается указатель на массив входов, а также создаются все нейроны скрытых и выходного слоев, при чем количество создаваемых нейронов зависит от размера переданного на вход трехмерного массива.

Таким образом, в реализованной нейронной сети нет жесткой привязки к количеству нейронов ни в одном из ее слоев, а значит отсутствует необходимость перекомпиляции программы, в случае изменения ее структуры.

2.2 Подготовка входных данных

Очень важным и ответственным шагом для обучения и проверки любой нейронной сети является подготовка входных данных.

Для упрощения описания и понимания в данной работе будет использоваться понятие валютной пары. Валютная пара – отношение цен двух валют, входящих в данную пару, на валютном рынке. Первая валюта в паре называется базовой, а вторая котируемой [15].

Например, валютная пара USD/RUB показывает, какое количество российских рублей (RUB) можно приобрести за один доллар США (USD). В данной паре базовой является валюта USD. Решение о том, какая валюта является базовой, а какая – котируемой, принимается Международной организацией по стандартизации (ISO).

В таблице 1 представлено соответствие всех валют, используемых в данной работе, их краткому названию в валютных парах.

Таблица 1 - Соответствие валют краткому обозначению

Обозначение в валютной паре	Валюта
EUR	Доллар США
USD	Единая европейская валюта
RUB	Российский рубль
CHF	Швейцарский франк
JPY	Иена (денежная единица Японии)
BRENT	Эталонная марка нефти

Далее в данной части работы все операции произведенные операции с данными будут показаны для валютной пары USD/RUB, однако аналогичные действия проводились и для других валютных пар, используемых для обучения нейронной сети.

Финансовый временной ряд принято представлять в виде японских свечей, изображенном на рисунке. Для реальной торговли на рынке Форекс важно понимать термины «свеча» и «тень свечи». На рисунке 11 изображен график валютной пары USDRUB за сентябрь 2010 года в виде японских свечей.

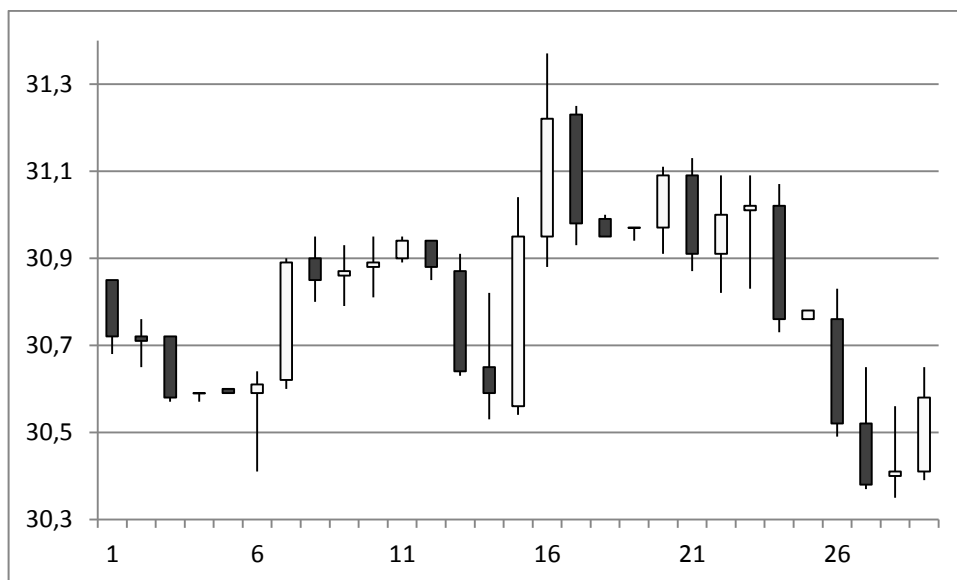


Рисунок 11 – Курс USD/RUB за сентябрь 2010 в японских свечах

Свеча состоит из тела черного либо белого цвета и фитиля, который чаще называют тенью. Верхняя и нижняя граница тени отображает минимум

и максимум цены за соответствующий период. Границы тела показывают цену открытия и закрытия торгов.

Белое тело означает общий рост цен, тогда нижняя граница будет соответствовать цене открытия, а верхняя – закрытия. Черное тело символизирует падение курса, в связи с этим верхняя граница соответствует цене открытия торгов, а нижняя – закрытия.

В данной работе было принято решение пренебречь колебаниями курсов в течение дня и оперировать только ценами закрытия интересующих валютных пар. На рисунке 12 изображен график динамики цены закрытия торгов пары USD/RUB за сентябрь 2010.

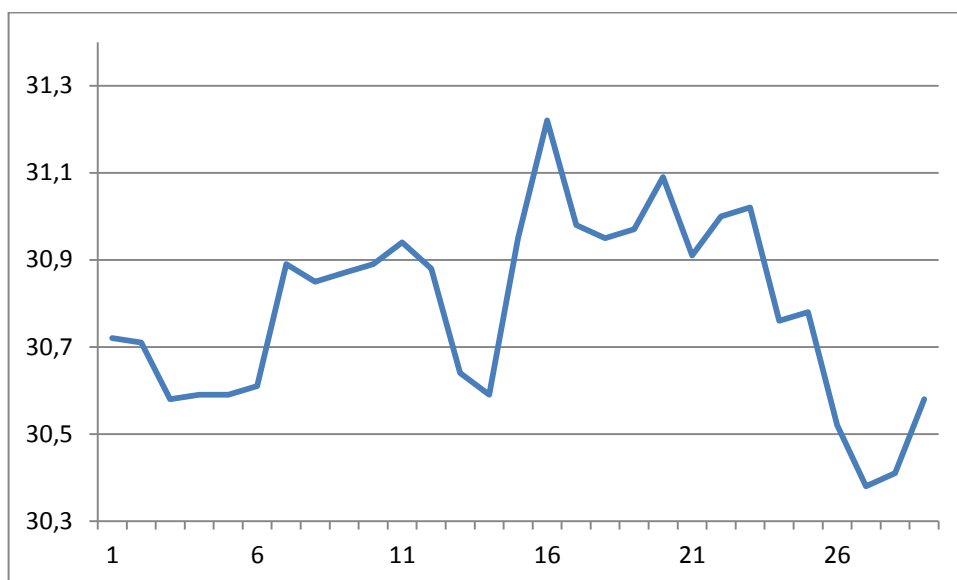


Рисунок 12 - Динамика цены закрытия USD/RUB за сентябрь 2010

2.2.1 Нормализация и распределение

В случае прогнозирования финансовых временных рядов можно выделить два параметра обучающей выборки: количество элементов и размер скользящего окна. От выбора этих параметров зависит противоречивость и повторяемость, для которых можно выделить следующие правила:

- противоречивость и повторяемость прямо пропорциональны количеству элементов обучающей выборки;

- противоречивость и повторяемость обратно пропорциональны размеру окна погружения во временной ряд;
- противоречивость и повторяемость обратно пропорциональны количеству возможных значений, подаваемых на вход ИНС.

Из указанных выше правил видно, что факторы влияют на противоречивость и повторяемость с одинаковым направлением. Необходимо найти такие параметры окна, количества элементов и классов входных значений, при которых противоречивость данных минимальна, а повторяемость максимальна.

Из графика, изображенного на рисунке, видно, что в сентябре 2010 за один доллар США можно было купить около тридцати российских рублей. На момент написания данной работы, курс доллара США вырос до пятидесяти шести рублей. Очевидно, что сеть, обученная на значениях пары USDRUB за 2010 год, скорее всего не сможет предсказать тренд данной пары для входных данных 2016 года.

В связи с этим, в первую очередь было решено использовать приращения курсов валют, а не соответствующие цены закрытия, так как это позволит использовать сеть, обученную на данных конкретного года для других временных промежутков, где курс доллара США относительно российского рубля значительно изменился, а диапазон колебаний сохранен.

На рисунке 13 изображен график, значения которого соответствуют приращениям валютной пары USD/RUB за полный 2010 год.

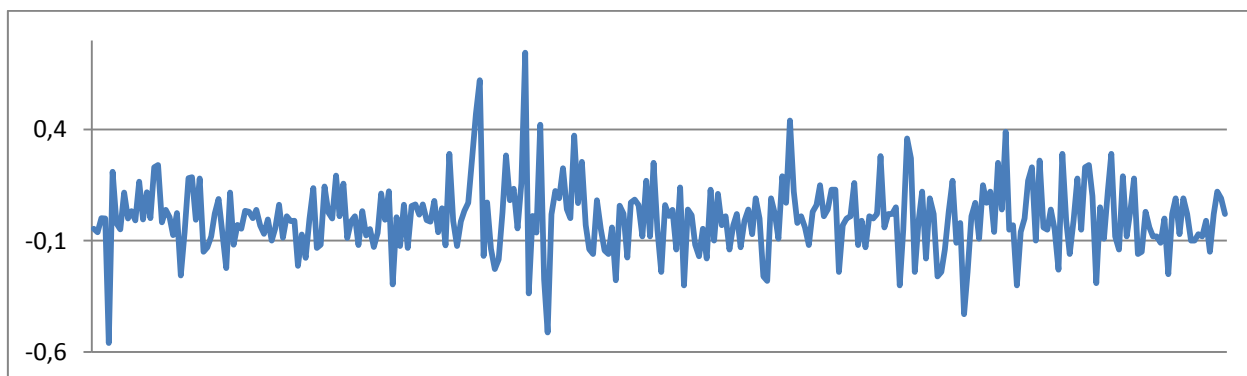


Рисунок 13 - График приращения USD/RUB за 2010 год

На рисунке видно, что большинство значений сконцентрировано в диапазоне $[-0,2;0,2]$ и лишь некоторые приращения выходят из данного отрезка. Кроме того, в связи с особенностями реализованной нейронной сети, значение входов и выходов может находиться в интервале $[-1;1]$, а значит, что данная выборка не позволяет охватить весь возможный диапазон значений входных нейронов.

Тем не менее, после получения соответствующих значений приращений было произведено тестовое обучение нескольких нейронных сетей на этих данных, после чего стало понятно, что обучающая выборка слишком противоречива.

Если разделить все входные значения обучающей выборки на условные интервалы диапазоном по одной десятой ($[-1;-0,9)$, $[-0,9;-0,8)$ и т.д.) и отобразить количество значений приращений в каждом из полученных интервалов на графике, то можно увидеть, что распределение входных данных крайне неравномерное.

На рисунке 14 изображена диаграмма частоты попадания приращения пары USD/RUB в диапазоны, описанные выше.

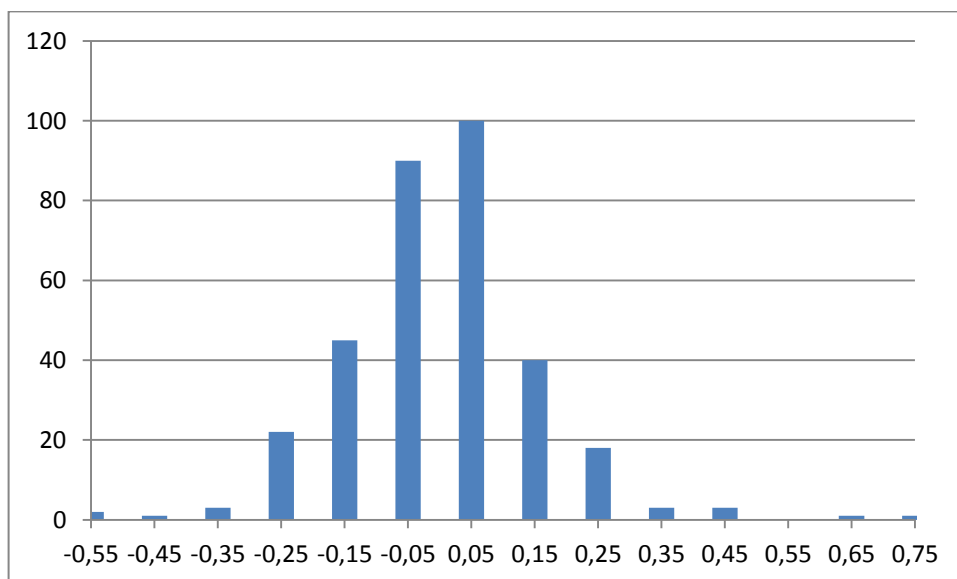


Рисунок 14 – Диаграмма частоты попадания приращения USD/RUB в диапазон

В результате вывода о большой противоречивости, было решено равномерно разделить все возможные входные значения на несколько классов.

Для решения задачи равномерного распределения входных данных по классам, которые должны нести одинаковую информационную нагрузку, все значения приращений были отсортированы по возрастанию, и поделены на 11 равных частей. Каждому классу было присвоено значение от -1 до 1 с шагом 0,2 для передачи на вход нейронной сети. Таким образом, получилось 11 классов, число приращений для которых изображено на диаграмме в рисунке 15.

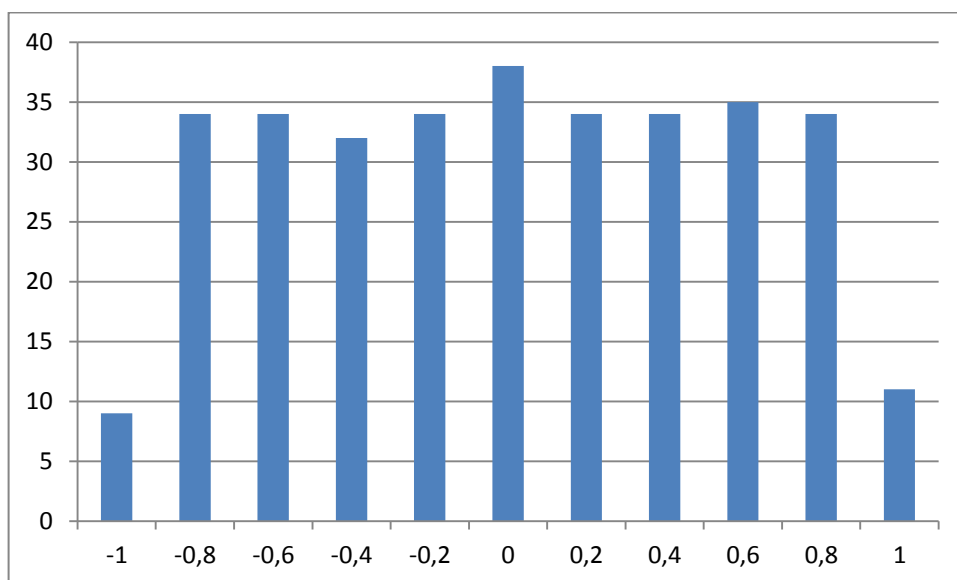


Рисунок 15 - Распределение приращений по классам

Из эвристических соображений было решено сдвинуть классы так, чтобы отрицательные приращения находились в классах со значениями от -1 до 0 и наоборот. Кроме того, было намеренно решено оставить для крайних классов меньшее число значений, так как максимальные по модулю приращения в обучающей выборке не являются предельными для других временных промежутков.

В таблице 2 представлено соответствие интервалов значений приращений значениям полученных классов для каждой валютной пары, используемой в обучении.

Таблица 2 - Соответствие интервалов валютных приращений классам

	USD/RUB	USD/JPY	USD/CHF	EUR/USD	USD/BRENT
-1	$(-\infty; -0,28]$	$(-\infty; -0,09]$	$(-\infty; -0,0075]$	$(-\infty; -0,0125]$	$(-\infty; -2]$
-0,8	$(-0,28; -0,14)$	$(-0,09; -0,5]$	$(-0,0075; -0,0045]$	$(-0,0125; -0,0075]$	$(-2; -1]$
-0,6	$(-0,14; -0,09]$	$(-0,5; -0,3]$	$(-0,0045; -0,00225]$	$(-0,0075; -0,0042]$	$(-1; -0,6]$
-0,4	$(-0,09; -0,05]$	$(-0,3; -0,2]$	$(-0,00225; -0,0012]$	$(-0,0042; -0,0015)$	$(-0,6; -0,26]$
-0,2	$(-0,05; -0,01)$	$(-0,2; -0,1)$	$(-0,0012; -0,0004)$	$(-0,0015; -0,0005)$	$(-0,26; -0,065]$
0	$(-0,01; 0,01]$	$(-0,1; -0,03)$	$(-0,0004; 0,0004]$	$(-0,0005; 0,0005]$	$(-0,065; 0,065)$
0,2	$(0,01; 0,04)$	$(-0,03; 0,04)$	$(0,0004; 0,002]$	$(0,0005; 0,002)$	$(0,065; 0,3]$
0,4	$(0,04; 0,08]$	$(0,04; 0,17)$	$(0,002; 0,0045)$	$(0,002; 0,004]$	$(0,3; 0,75)$
0,6	$(0,08; 0,14]$	$(0,17; 0,33)$	$(0,0045; 0,0075)$	$(0,004; 0,008]$	$(0,75; 1,2)$
0,8	$(0,14; 0,29)$	$(0,33; 0,53)$	$(0,0075; 0,0150)$	$(0,008; 0,0150)$	$(1,2; 2)$
1	$(0,29; +\infty)$	$(0,53; +\infty)$	$(0,0150; +\infty)$	$(0,0150; +\infty)$	$(2; +\infty)$

2.2.2 Формирование исходных файлов

Эмпирическим путем было решено подавать на вход нейронной сети тридцать пять нормированных соответствующим образом значений, распределенных между пятью валютными парами следующим образом:

- 15 значений USD/RUB;
- 5 значений USD/JPY;
- 5 значений USD/CHF;
- 5 значений EUR/USD;
- 5 значений USD/BRENT.

На выходе ИНС было решено сформировать пять нейронов, в активное положение которых соответственно означает по порядку:

- сильное падение;
- падение;
- рост;
- сильный рост;
- отсутствие изменений.

В данном случае падением является изменение валютной пары USD/RUB в сторону уменьшения, то есть укрепление российского рубля. И наоборот: рост – изменение пары USD/RUB в сторону увеличения, то есть укрепление доллара США.

В качестве «сильного» изменения было решено использовать значения приращений, превышающих по модулю 0,2.

В связи с описанными выше решениями, структура обучаемых ИНС выглядит так, как изображено на рисунке 16.

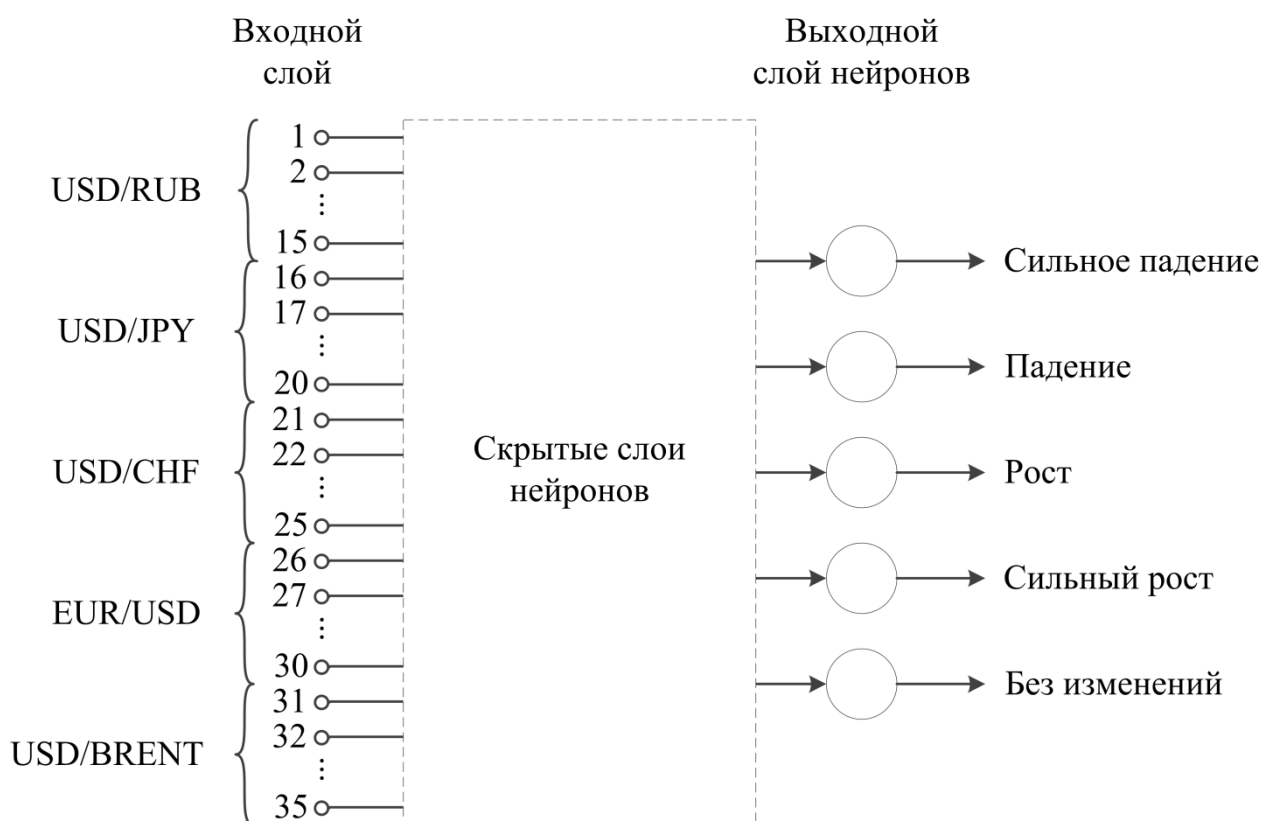


Рисунок 16 - Структура обучаемой ИНС в общем виде

В процессе реализации нейронной сети было решено, что файл, предназначенный для установки значений входов ИНС должен иметь следующую структуру: на первой строке должно быть указано целое число N (в описанном случае – 35), соответствующее количеству входов, далее N строк с вещественными значениями соответствующих входов.

Для генерации входных файлов была написана программа, которая из файла, содержащего значения валютных пар за необходимый период,

получает значения цен закрытия и сохраняет их в массив. Затем для данного массива получаются значения приращений, и, наконец, значения, нормализованные описанным в предыдущей части образом.

В листинге представлен фрагмент функции, непосредственно записывающий необходимые значения в файл.

```
for (int i = 14; i < UsdRub.size() - 1; i++)
{
    char intStr[5] = { 0 };
    _itoa(i - 13, intStr, 10);
    string fileName = intStr;
    fileName += ".txt";
    cout << fileName << endl;
    f = fopen(fileName.c_str(), "w");
    fprintf(f, "35\n");

    for (int j = 14; j >= 0; j--)
    {
        fprintf(f, "%lf ", UsdRub[i - j]);
    }
    fprintf(f, "\n");

    for (int j = 4; j >= 0; j--)
    {
        fprintf(f, "%lf ", UsdJpy[i - j]);
    }
    fprintf(f, "\n");

    for (int j = 4; j >= 0; j--)
    {
        fprintf(f, "%lf ", UsdChf[i - j]);
    }
    fprintf(f, "\n");

    for (int j = 4; j >= 0; j--)
    {
        fprintf(f, "%lf ", EurUsd[i - j]);
    }
    fprintf(f, "\n");

    for (int j = 4; j >= 0; j--)
    {
        fprintf(f, "%lf ", Brent[i - j]);
    }
    fprintf(f, "\n");

    fclose(f);
}
```

Из листинга видно, что в цикле, начиная с пятнадцатого элемента массива, соответствующего паре USD/RUB, до его окончания, производятся следующие действия:

- Создается и открывается текстовый файл на запись.
- В созданный файл записывается пятнадцать нормализованных значений приращений пары USD/RUB, начиная с элемента под индексом « $i - 14$ » и заканчивая индексом « i », то есть в сторону увеличения индекса. Фактически, в файл записываются необходимые значения для пятнадцати предыдущих перед « i » дней.
- В созданный файл записывается пять нормализованных значений приращений пары USD/JPY, начиная с элемента под индексом « $i - 4$ » и заканчивая индексом « i », то есть в сторону увеличения индекса. Фактически, в файл записываются необходимые значения для пятнадцати предыдущих перед « i » дней.
- Аналогичные действия для пяти нормализованных значений приращений валютных пар USD/CHF.
- Аналогичные действия для пяти нормализованных значений приращений валютных пар EUR/USD.
- Аналогичные действия для пяти нормализованных значений приращений валютных пар USD/BRENT.

В результате генерации было получено 285 текстовых файлов, соответствующих периоду 2010 года, 271 текстовый файл, описывающих 2012 год и 277 текстовых файлов из периода 2012 года. Соответственно файлы 2010 года будут использоваться для обучения нейронных сетей разной структуры, на основании прогнозов для файлов 2011 года будет выбираться наилучшая из обученных и отобранных на первом этапе сетей, а входные файлы за 2012 год будут использоваться для моделирования торговли на рынке Форекс.

2.3 Реализация генетического алгоритма

Для обучения нейронной сети использовалась подсистема, реализованная Алатовым Евгением в работе «Подсистема обучения нейронной сетей с использованием генетических алгоритмов».

Генетический алгоритм, как и нейронная сеть, написан на языке C++, поэтому готовые исходные файлы его описания и реализации были просто добавлены в проект.

Так как в данной работе ГА, разработанный Алатовым Евгением, используется в качестве интерфейса, то будем рассматривать только публичные поля и методы реализованных им классов.

2.3.1 Класс Chromosome

Класс Chromosome описывает одну особь в популяции и содержит следующие публичные поля:

- alleles – массив типа double, реализующий понятие набора генов из генетического алгоритма. В случае с нейронной сетью, данный массив будет содержать в себе весовые коэффициенты синапсов ИНС.
- fitness – поле типа double для хранения значения пригодности данной особи.
- metrics – массив типа double, который используется при конкуренции двух особей за право стать родителем. Содержит информацию об ответах системы.
- chance – поле типа double, содержащее размер сектора на колесе рулетки, в операции селекции.
- percent, countTrue – поля типа double, которые хранят информацию о пригодности особи.

2.3.2 Класс GA

Класс GA является основным классом генетического алгоритма, непосредственно реализующим его работу. Класс поддерживает обучение и тестирование нейронной сети. Все поля класса являются закрытыми, однако имеют соответствующие геттеры и сеттеры, а значит их можно рассматривать как публичные. Класс GA содержит следующие поля:

- Population, OldPopulation, CrossingPopulation – вещественные массивы типа double. В поле Population записываются особи текущей эпохи. В OldPopulation хранятся особи родителей. CrossingPopulation – промежуточная популяция скрещивания.
- BestPopulation – объект типа Chromosome, хранит информацию о лучшей особи.
- MaxPopulation, AllelesSize. MaxPopulation – задаёт размер популяции. AllelesSize – задает общее количество весов ИНС.
- RTTypeSelectrionPercent – вероятность выбора метода рулетки в качестве оператора селекции. Значение 1 – RTTypeSelectrionPercent является вероятностью выбора метода турнира.
- TruncationStep, ElitePercent. TruncationStep – пороговое значение (в процентах) лучших особей в методе редукции с усечением. ElitePercent – процент элитных особей в элитарном методе отбора.
- MutatePercent, MutateStep, MutatePercentNewGenom. MutatePercent – вероятность мутации, в процентах. MutateStep – значение приращения (в процентах) оператора мутации. MutatePercentNewGenom – вероятность случайного гена в операторе мутации, выраженная в процентах

- MaxIteration, Epoch, ResultPercent – количество эпох, текущая эпоха и требуемый процент точности. Используются для определения завершения алгоритма
- CrossObject – объект типа CrossT.

Вспомогательный класс CrossT реализует механизм выбора стратегии скрещивания. Содержит вероятности выбора того или иного метода скрещивания.

Операторы генетического алгоритма представлены следующими функциями.

GenerateStartPopulation() – в данной функции реализован оператор генерации начальной популяции. В создаваемую популяцию подмешивается лучшая особь предыдущего запуска алгоритма.

Fitness() – в данной функции происходит инициализация ИНС и разделения программы на многопоточное выполнение функции FitnessForOne().

FitnessForOne() – многопоточная функция расчета значения приспособленности отдельной особи. В данной функции происходит инициализация весов и запуск ИНС.

SelectNewPopulation() – функция, реализующая оператор редукции.

CreateNewPopulation() – функция создания нового поколения популяции. Происходит подготовка данных для многопоточных операций скрещивания.

Crossing() – функция реализует оператор селекции ГА.

RouletteSelection() – реализация селекции методом рулетки.

GenerateLikelihoods() – определение размера сектора для каждой особи на колесе рулетки.

TournamentSelection() – реализация турнирного метода селекции.

ThreadCross() – Многопоточная функция, реализующая оператор скрещивания ГА.

Mutate() – функция реализует оператор мутации.

Run() – основная функция, отвечающая за исполнение последовательностей операторов.

GetTrend() – данная функция служит для проверки ответа сети с эталонным значением, после чего высчитывается значение приспособленности данной ИНС(особи).

StartLearning() – функция инициализации, настройки и запуска обучения нейронных сетей с помощью ГА.

RealTimeLearnTest() – функция обучения и тестирования ИНС.

2.4 Реализация алгоритма обучения

Программа обучения сводится к объединению классов генетического алгоритма и нейронной сети, и их последовательному вызову методов друг друга. Так как генетический алгоритм, фактически, является учителем в процессе обучения, то класс ГА является ведущим относительно класса NeuronNet.

В данной работе было принято решение использовать пакетный принцип обучения, когда нейронная сеть выполняется для всего набора входных файлов на текущем наборе весов.

Таким образом, обучение нейронной сети можно свести к следующим этапам:

1. Генерация начальной популяции, где каждая особь представляет набор весов нейронной сети.
2. Выполнение ИНС с весами каждой особи текущей популяции на всех входных файлах обучающей выборки.
3. Оценка результатов выполнения нейронной сети.
4. Получение новой популяции и переход ко второму этапу.

На рисунке 17 изображена схема алгоритма главной функции программы, обучения.

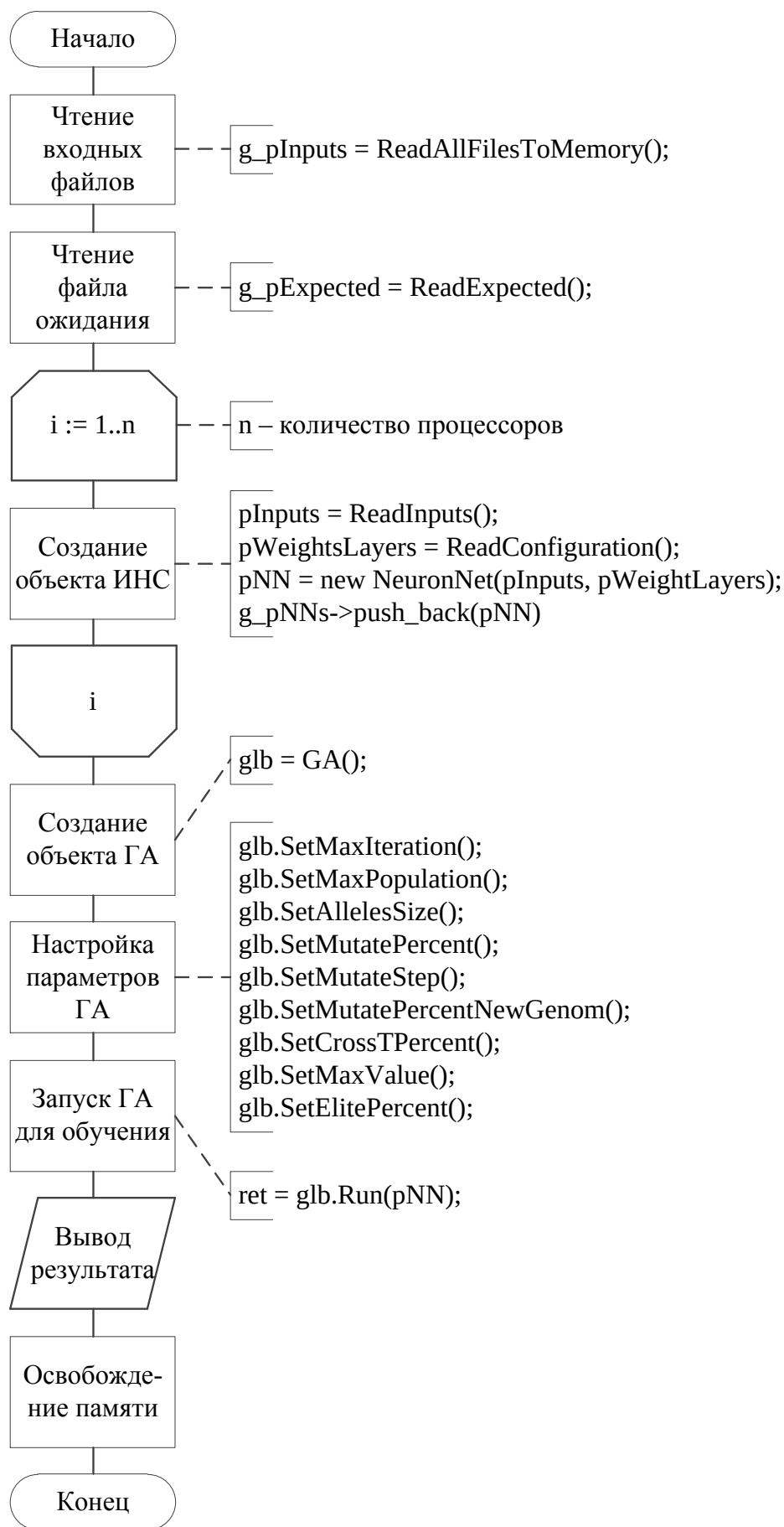


Рисунок 17 – Схема алгоритма главной функции программы

Из блок-схемы видно, что в первую очередь в память читаются все входные файлы и файл ожидания. Стоит уточнить, что в файле ожидания последовательно хранятся номера выходных нейронов, которые должны быть активными в результате работы нейронной сети на первом, втором и последующих файлах входной выборки.

Внутри функции `ReadAllInputFilesToMemory()` в цикле от 1 до `INPUT_FILES_NUMBER` включительно, в данном случае до 285, формируется имя входного файла и, при помощи функции `ReadInputs()`, производится непосредственно его чтение.

Результатом чтения отдельного файла является указатель на массив входов нейронной сети. Для структуры ИНС, изображенной на рисунке, такой вектор будет содержать 35 элементов класса `Input`. Входы последовательно считываются из файла, имя которого передано во входном параметре функции.

Далее, функция `ReadAllInputFilesToMemory()` добавляет полученные указатели в массив и возвращает его адрес в главную функцию, которая сохраняет его в глобальную переменную `g_pInputs`. Таким образом, все функции и методы программы имеют доступ к указателю на двумерный массив указателей. Первый уровень массива соответствует входному файлу, а второй – соответствующим значениям входов нейронной сети. Для структуры ИНС изображенной на рисунке, массив будет иметь 285 указателей на вектора с 35 значениями каждый.

Функция `ReadExpectedFile()` схожим образом формирует массив ожидаемых результатов работы нейронной сети, указатель на который сохраняется в глобальную переменную `g_pExpected`.

Далее в функции получается значение количества процессоров в системе, на которой запущена программа и, в цикле, для каждого процессора создается свой экземпляр нейронной сети.

Для создания объекта ИНС используется конструктор с параметрами, описанный в части 2.1.2 данной работы. Этот конструктор принимает на вход два указателя: на массив входов и конфигурации сети, соответственно.

Для получения первого параметра вызывается уже знакомая функция `ReadInputs()`. Второй параметр получается из файла конфигурации при помощи функции `ReadConfiguration()`.

В функции `ReadConfiguration()` из файла читается число нейронов для входного, скрытых и выходного слоев, а дальше, соответственно считываются веса соответствующих синапсов нейронов. В результате чтения формируется трехмерный массив, структура которого описана в пункте 2.1.2. Третий уровень полученного массива – веса для отдельных синапсов нейрона, второй уровень необходим для отделения весов разных нейронов друг от друга, а третий для разделения нейронов на слои.

Указатель на массив конфигурации возвращается в главную функцию, а затем, вместе с массивом входов, передается на вход в конструктор класса `NeuronNet`. Все созданные объекты класса сохраняются в массив, указатель на который сохранен в глобальной переменной `g_pNNs`.

После создания экземпляров нейронной сети по количеству процессоров, вызывается непосредственно функция, ответственная за обучение, `StartLearning()`.

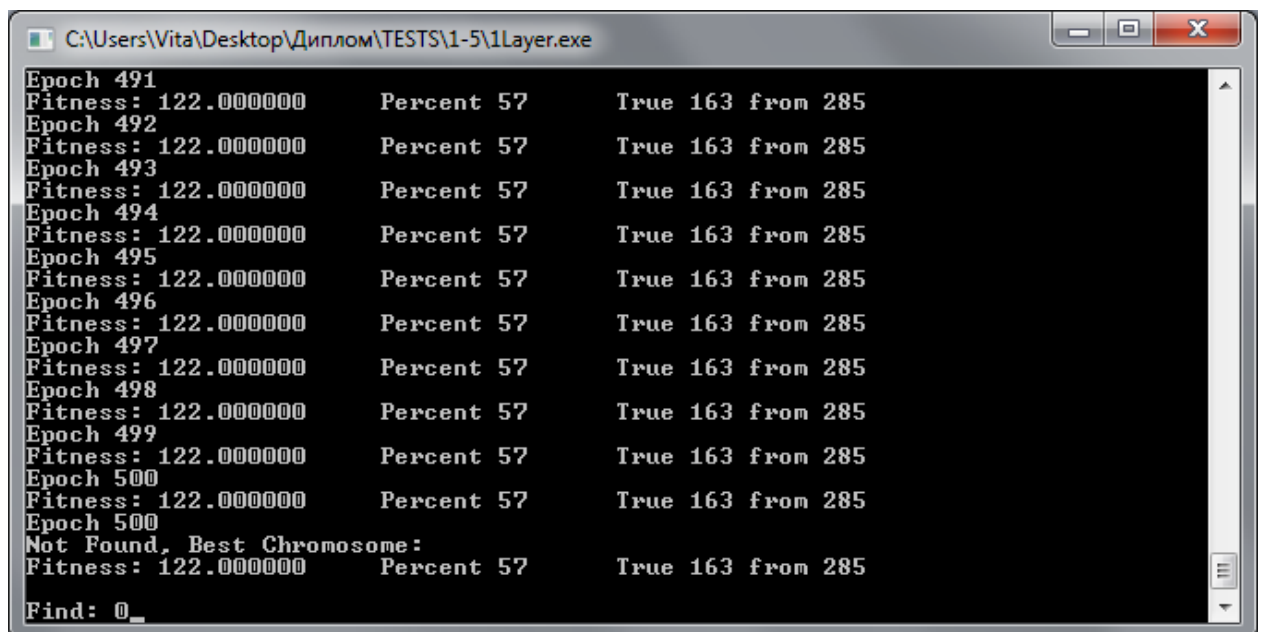
В функции начала обучения создается объект класса `GA`, затем вычисляется необходимое число аллелей, то есть весов синапсов, и соответствующим образом настраиваются параметры генетического алгоритма: максимальная итерация, максимальная популяция, число аллелей, процент мутации и другие.

После настройки вызывается метод `Run` для созданного объекта генетического алгоритма, который непосредственно стартует цикл обучения, по завершении которого, в консоль выводится результат лучшей нейронной сети, полученной в результате обучения.

В качестве результата выводятся следующие значения:

- значение фитнес-функции для лучшей ИНС;
- процент правильно активированных нейронов для обучающей выборки;
- количество обучающих наборов из всей выборки, для которых нейронная сеть показала правильный результат.

В ходе обучения, по окончании каждой эпохи, на экран выводится информация о результатах лучшей на данный момент особи. На рисунке 18 изображена консоль после окончания обучения одной из нейронных сетей.



```
C:\Users\Vita\Desktop\Диплом\TESTS\1-5\1Layer.exe
Epoch 491
Fitness: 122.000000      Percent 57      True 163 from 285
Epoch 492
Fitness: 122.000000      Percent 57      True 163 from 285
Epoch 493
Fitness: 122.000000      Percent 57      True 163 from 285
Epoch 494
Fitness: 122.000000      Percent 57      True 163 from 285
Epoch 495
Fitness: 122.000000      Percent 57      True 163 from 285
Epoch 496
Fitness: 122.000000      Percent 57      True 163 from 285
Epoch 497
Fitness: 122.000000      Percent 57      True 163 from 285
Epoch 498
Fitness: 122.000000      Percent 57      True 163 from 285
Epoch 499
Fitness: 122.000000      Percent 57      True 163 from 285
Epoch 500
Fitness: 122.000000      Percent 57      True 163 from 285
Epoch 500
Not Found, Best Chromosome:
Fitness: 122.000000      Percent 57      True 163 from 285
Find: 0_
```

Рисунок 18 - Консоль программы после обучения

3 ИССЛЕДОВАНИЕ И КОРРЕКТИРОВКА СИСТЕМЫ

Как было сказано в части 2, работу было решено разделить на три этапа:

1. Обучение нескольких ИНС разной структуры генетическим алгоритмом.
2. Отбор лучшей структуры из полученных в первой части путем моделирования работы сети на тестовой выборке.
3. Симуляция торгов на рынке Форекс.

3.1 Обучение ИНС разных структур

Исходя из теории, был сделан вывод, что однослойный персептрон Розенблатта не подходит для решения задачи прогнозирования финансовых рядов, так как может классифицировать только линейно-разделимые классы. В связи с этим, было решено обучать многослойные персептроны с различным количеством скрытых слоев и нейронов в них.

3.1.1 Многослойный персептрон с одним скрытым слоем

В ходе обучения многослойного персептрона с одним скрытым слоем, было обнаружено, что три из пяти нейронов, фактически, «отмирают», то есть не являются активными в результате ни одного из наборов входных значений обучающей выборки. В связи с этим, в таблице представлен дополнительный столбец, где указан процент верно угаданных направлений. Таким образом, если, исходя из файла ExpectedOutputs.txt, для очередного входного файла ожидается активным выходной нейрон «сильного» приращения, а в результате работы сети стал активным нейрон с тем же направлением изменения, но не силы, то считаем такой нейрон верным.

В таблице 3 представлено соответствие структуры персептрона с одним скрытым слоем, результатам его обучения на данных фондового рынка за 2010 год.

Таблица 3 - Результаты обучения ИНС с одним скрытым слоем

Структура нейронной сети	Количество верных значений выходных нейронов	Процент верных активных выходных нейронов	Процент верно предсказанных направлений
35-5-5	163	58,84%	65,34%
35-6-5	161	58,12%	65,34%
35-7-5	158	57,03%	64,62%
35-8-5	161	58,12%	67,50%
35-9-5	159	57,40%	65,34%
35-10-5	145	52,34%	61,01%
35-13-5	151	54,51%	62,09%
35-15-5	153	55,23%	62,81%
35-20-5	160	57,76%	66,42%
35-25-5	141	50,90%	57,76%
35-30-5	129	46,57%	54,51%
35-35-5	120	43,32%	50,90%
35-40-5	117	42,23%	51,98%
35-45-5	107	38,62%	48,73%
35-50-5	117	42,23%	51,62%

Если учесть, что в выборке присутствует 18 дней, для которых на выходе ИНС ожидается активным нейрон «без изменения», то можно считать процент угаданных направлений выше указанного в таблице.

На рисунке 19 изображен график зависимости процентов верно указанных направлений от структуры ИНС.

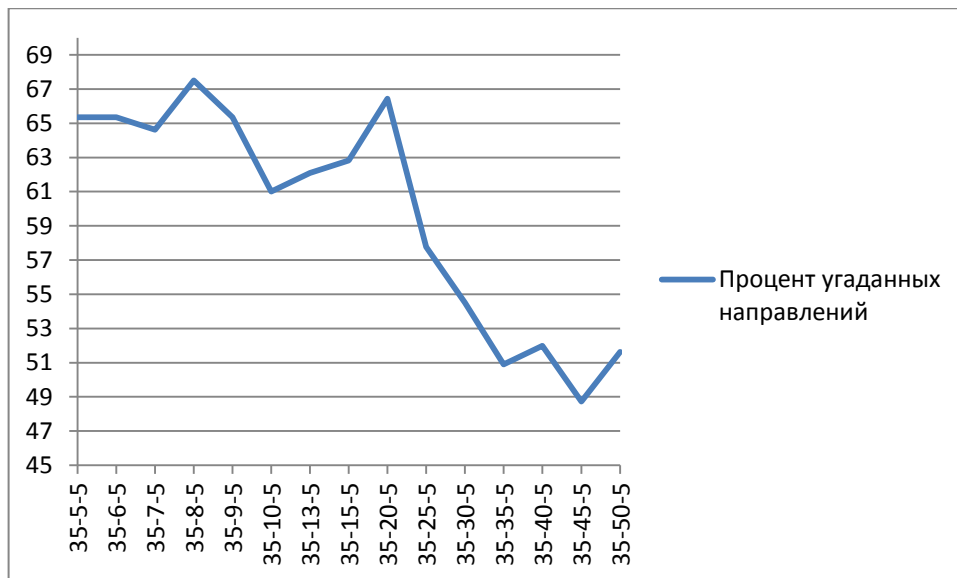


Рисунок 19 - График соответствия угаданных направлений структуре ИНС

Из рисунка 19, как и из таблицы 3, видно, что при увеличении количества нейронов в скрытом слое нейронной сети, процент угаданных направлений падает. Вероятнее всего, для структуры с большим количеством нейронов требуется большее количество эпох обучения.

Очевидно, что существует линейная зависимость времени обучения от количества нейронов, как и зависимость времени обучения от количества эпох генетического алгоритма. Таким образом, если для нейронной сети с большим числом вычислительных узлов, предоставлять большее число циклов обучения генетическим алгоритмом, то непосредственно время обучения в худшем случае будет увеличено в квадрат.

В связи со сделанным выше выводом, было решено в дальнейшем не оперировать массивными нейронными сетями, в связи с ограниченной вычислительной мощностью.

3.1.2 Многослойный персептрон с двумя скрытыми слоями

В ходе обучения было выявлено, что утверждение про отмирание трех выходных нейронов, сделанное в пункте 3.1.1 справедливо и для нейронных сетей с двумя скрытыми слоями.

В таблице 4 представлено соответствие структуры персептрона с двумя скрытыми слоями, результатам его обучения на данных фондового рынка за 2010 год.

Таблица 4 - Результаты обучения ИНС с двумя скрытыми слоями

Структура нейронной сети	Количество верных значений выходных нейронов	Процент верных активных выходных нейронов	Процент верно предсказанных направлений
35-5-5-5	161	58,12%	66,06%
35-5-10-5	145	52,34%	59,20%
35-5-15-5	134	48,37%	56,67%
35-5-20-5	142	51,26%	61,73%
35-10-5-5	156	56,31%	64,98%
35-10-10-5	151	54,51%	61,37%
35-10-15-5	115	41,51%	50,18%
35-10-20-5	146	52,70%	59,92%
35-15-5-5	162	58,48%	65,70%
35-15-10-5	105	37,90%	45,12%
35-15-15-5	136	49,09%	57,40%
35-15-20-5	141	50,90%	58,12%
35-20-5-5	166	59,92%	67,14%
35-20-10-5	159	57,40%	66,78%
35-20-15-5	144	51,98%	60,28%
35-20-20-5	145	52,34%	61,73%

На рисунке 20 изображен график зависимости процентов верно указанных направлений от структуры ИНС.

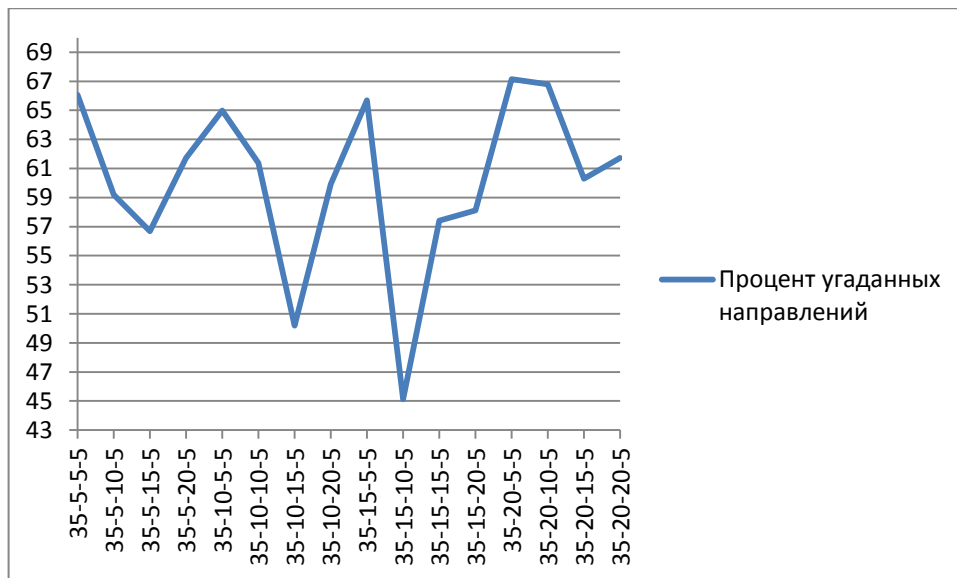


Рисунок 20 – График соответствия угаданных направлений структуре ИНС

Можно сделать промежуточный вывод, что в целом нейронные сети с двумя скрытыми слоями показали схожие результаты с ИНС с одним скрытым слоем. При чем, если результаты лучших экземпляров для угаданных направлений практически одинаковые, то процент точных предсказаний у сетей с двумя скрытыми слоями лучше.

Кроме того, исходя из рисунка, можно сказать, что тенденция ухудшения процента предсказаний при увеличении числа нейронов, для персептронов с двумя скрытыми слоями не повторяется, так как и лучший и худший результат при обучении сетей данной структуры, показали ИНС с суммарным количеством нейронов в скрытых слоях, равным 25.

С другой стороны, можно заметить, что лучшие значения процента угаданных направлений соответствуют сетям с пятью нейронами в последнем скрытом слое, вне зависимости от первого.

3.1.3 Многослойный персептрон с тремя скрытыми слоями

В ходе обучения персептронов с тремя скрытыми слоями, было выявлено, что отмирание трех выходных нейронов для сетей данной

архитектуры прекратилось. Тем не менее, продолжим обращать внимание на такой показатель, как процент верно указанных направлений.

В таблице 5 представлено соответствие структуры персептрона с тремя скрытыми слоями, результатам его обучения на данных фондового рынка за 2010 год.

Таблица 5 - Результаты обучения ИНС с тремя скрытыми слоями

Структура нейронной сети	Количество верных значений выходных нейронов	Процент верных активных выходных нейронов	Процент верно предсказанных направлений
35-5-5-5	178	59,63%	65,93%
35-5-10-5-5	131	43,33%	48,52%
35-5-15-5-5	160	54,81%	59,26%
35-6-18-3-5	202	69,63%	74,81%
35-5-1-5-5	170	57,04%	62,96%
35-2-5-5-5	198	66,67%	73,33%
35-7-1-3-5	166	56,30%	61,48%
35-20-10-5-5	139	44,07%	51,48%
35-20-20-20-5	127	41,85%	47,04%
35-4-4-4-5	164	55,56%	60,74%
35-7-5-3-5	170	55,56%	62,96%
35-12-4-3-5	199	68,52%	73,70%
35-8-1-3-5	162	54,81%	60,00%
35-7-3-4-5	171	55,93%	63,33%
35-6-5-6-5	152	51,11%	56,30%

На рисунке 21 изображен график зависимости процентов верно указанных направлений от структуры ИНС.

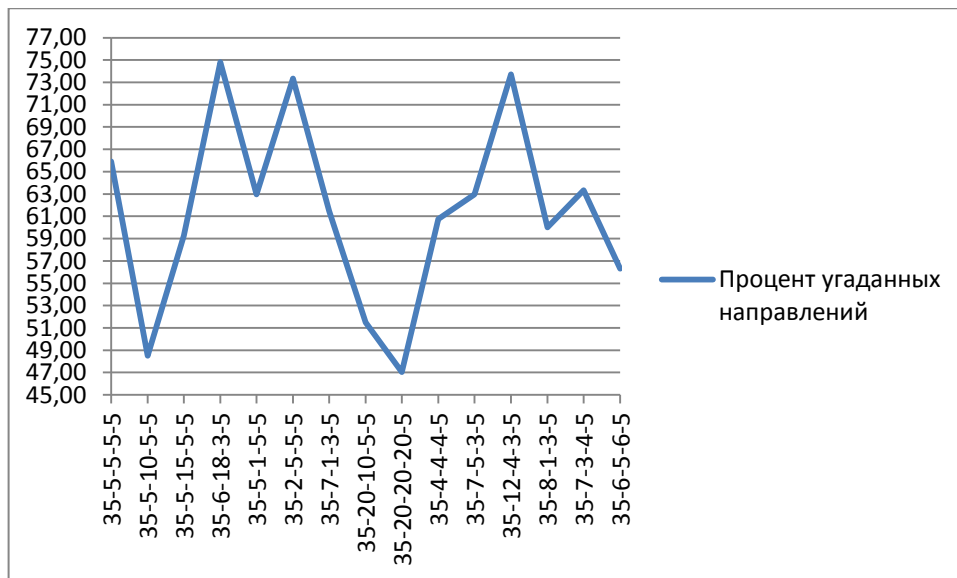


Рисунок 21 – График соответствия угаданных направлений структуре ИНС

Из таблицы 5 и рисунка 21 видно, что лучшие из обученных структур ИНС с тремя скрытыми слоями, показали значительно более хороший результат, нежели нейронные структуры с меньшим количеством скрытых слоев.

Стоит отметить, что улучшение процента верных активных нейронов на выходе, а также факт прекращения отмирания нейронов были замечены практически сразу после начала обучения персептронов с тремя скрытыми слоями. В связи с этим, было решено сделать упор на обучение сетей именно этой архитектуры.

В общей сложности было обучено более пятидесяти персептронов с тремя скрытыми слоями, результаты лишь пятнадцати из которых были отображены в таблице и на рисунке. Стоит отметить, что в таблицу были добавлены результаты, как для самых хорошо обученных ИНС, так и для плохих экземпляров.

3.2 Моделирование работы ИНС на тестовой выборке

Как уже было сказано в пункте 2, второй этап будет заключаться в отборе лучшей нейронной сети из обученных на первом этапе. Отбор будет

производиться путем моделирования работы сети на данных, соответствующих торгам за 2011 год.

Для дальнейшего анализа было решено взять по три лучших структуры из сетей с одним, двумя и тремя скрытыми слоями. Для оценки качества будем учитывать сумму процента верных активных нейронов и процента верно угаданных направлений. Исходя из таблиц, на данном этапе будет проводиться моделирование работы для следующих структур:

- 35-5-5, сумма 124,18%;
- 35-8-5, сумма 125,62%;
- 35-20-5, сумма 124,18%;
- 35-5-5-5, сумма 124,18%;
- 35-15-5-5, сумма 124,18%;
- 35-20-5-5, сумма 127,06%;
- 35-6-18-3-5, сумма 144,44%;
- 35-2-5-5-5, сумма 140,00%;
- 35-12-4-3-5, сумма 142,22%;

Кроме того, возьмем по одной нейронной сети, для которых разница между процентом верных активных нейронов и процента верно угаданных направлений минимальна. Таким образом, для моделирования на втором этапе добавляются еще три сети:

- 35-13-5, разница 7,58%;
- 35-10-10-5, разница 6,86%;
- 35-8-1-3-5, разница 5,19%;

Результаты моделирования работы отобранных сетей представлены в таблице 6.

Таблица 6 - Результаты моделирования работы сетей на данных торгов за 2011 год

Структура нейронной сети	Количество верных значений выходных нейронов	Процент верных активных выходных нейронов	Процент верно предсказанных направлений
35-5-5	110	40,59%	50,55%
35-8-5	110	40,59%	50,55%
35-20-5	111	40,95%	51,29%
35-5-5-5	116	42,80%	53,87%
35-15-5-5	97	35,79%	45,75%
35-20-5-5	110	40,59%	53,50%
35-6-18-3-5	106	39,25%	49,25%
35-2-5-5-5	108	40,00%	52,59%
35-12-4-3-5	106	39,25%	51,29%
35-13-5	111	40,95%	53,87%
35-10-10-5	102	37,63%	48,70%
35-8-1-3-5	98	36,29%	50,00%

На рисунке 22 изображен график соответствия угаданных направлений структурам нейронных сетей для 2010 года, данные которого являлись обучающей выборкой, и 2011 года, на котором проводилось моделирование работы сети.

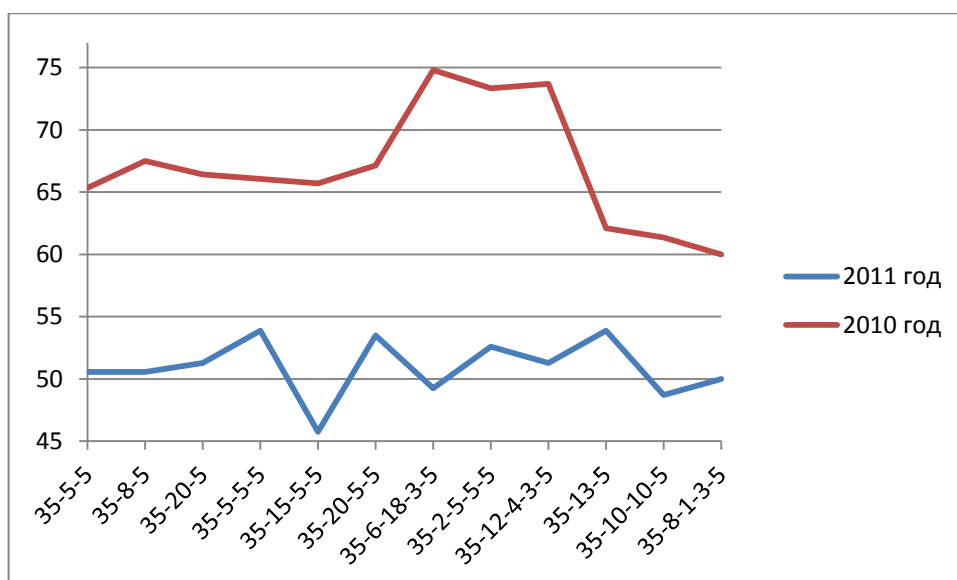


Рисунок 22 - График соответствия угаданных направлений структуре ИНС

На графике очевидно сильное падение угаданного процента для тестового периода, в сравнении со значениями, показанными для обучающей выборки. Тем не менее, большинство обученных ИНС дает результаты выше 50%.

Лучший результат для выборки 2011 года показала сеть со структурой 35-13-5. Кроме того, лучшим оказался показатель данной сети не только для процента верно предсказанных направлений, но и в наименьшей разнице между процентами обучающей и тестовой выборки.

Сильное падение результатов у сетей с тремя скрытыми слоями можно объяснить их переобученностью, то есть данные ИНС, наряду с факторами, значительными для данной задачи в целом, учитывают черты, характерные лишь для обучающей выборки.

3.3 Симуляция торгов на рынке Форекс

Безусловной целью попытки предсказания трендов фондового рынка являются торги. Так как обученная нейронная сеть предсказывает направление изменения цены валютной пары USD/RUB, то, очевидно, торги будут производиться именно для этой пары.

Так как национальной валютой РФ является российский рубль, то будем пытаться увеличить количество номинала этой валюты за счет своевременной покупки и продажи долларов США.

Исходя из таблицы и рисунка, был сделан выбор сети для симуляции торгов. Такой сетью стала ИНС с одним скрытым слоем и структурой 35-13-5. В первую очередь смоделируем работу данной сети на входных файлах, соответствующих 2012 году. Таким образом, получим файл со значениями выходных нейронов для каждого торгового дня финальной выборки.

Для симуляции торгов была реализована программа, которая соотносит предсказание сети и текущую цену доллара на рынке и совершает покупку или продажу таковых.

Так как нам заранее известно, что из пяти нейронов сети с одним скрытым слоем, только 2 остаются активными, то в данной программе не будем обращать внимание на три отмерших нейрона.

Было решено во время торгов принять следующую стратегию:

- если ИНС предсказывает рост доллара США, то необходимо совершить покупку на N российских рублей;
- если ИНС предсказывает рост российского рубля, то необходимо продать все купленные до этого доллары США.

Очевидно, что количество прибыли или убытков прямо пропорционально сумме, участвующей в торгах. В связи с этим, число N было принято равным одной тысяче. Кроме того, стоит отметить, что было принято решение не ограничивать бюджет определенным начальным количеством рублей.

На рисунке 23 изображен алгоритм программы, симулирующей торги на фондовом рынке, основываясь на предсказаниях нейронной сети.

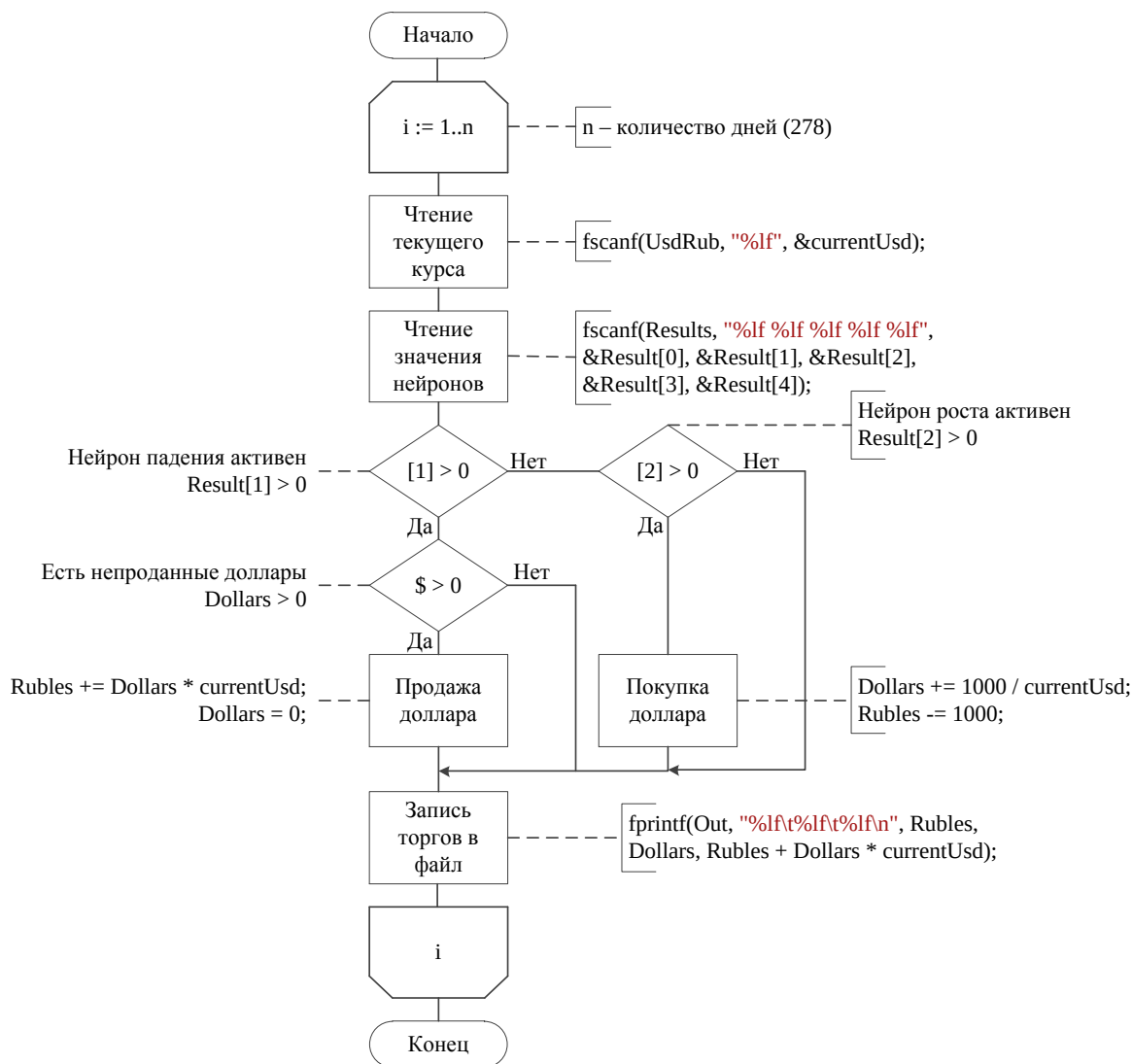


Рисунок 23 - Схема программы симуляции торгов

Для удобства оценки результатов торговли, введем понятие потенциальных рублей. Так как в момент покупки долларов, текущее значение рублей уменьшается на значение N , то график будет постоянно прыгать. Тем не менее, фактически данные рубли все еще принадлежат нам, просто они сконвертированы в доллары. Сумму текущего количества рублей и долларов, переведенных в рубли по текущему курсу, будем называть потенциальной.

На рисунке 24 изображен график потенциальных рублей, полученный в ходе симуляции торгов валютной парой USD/RUB на рынке Форекс.

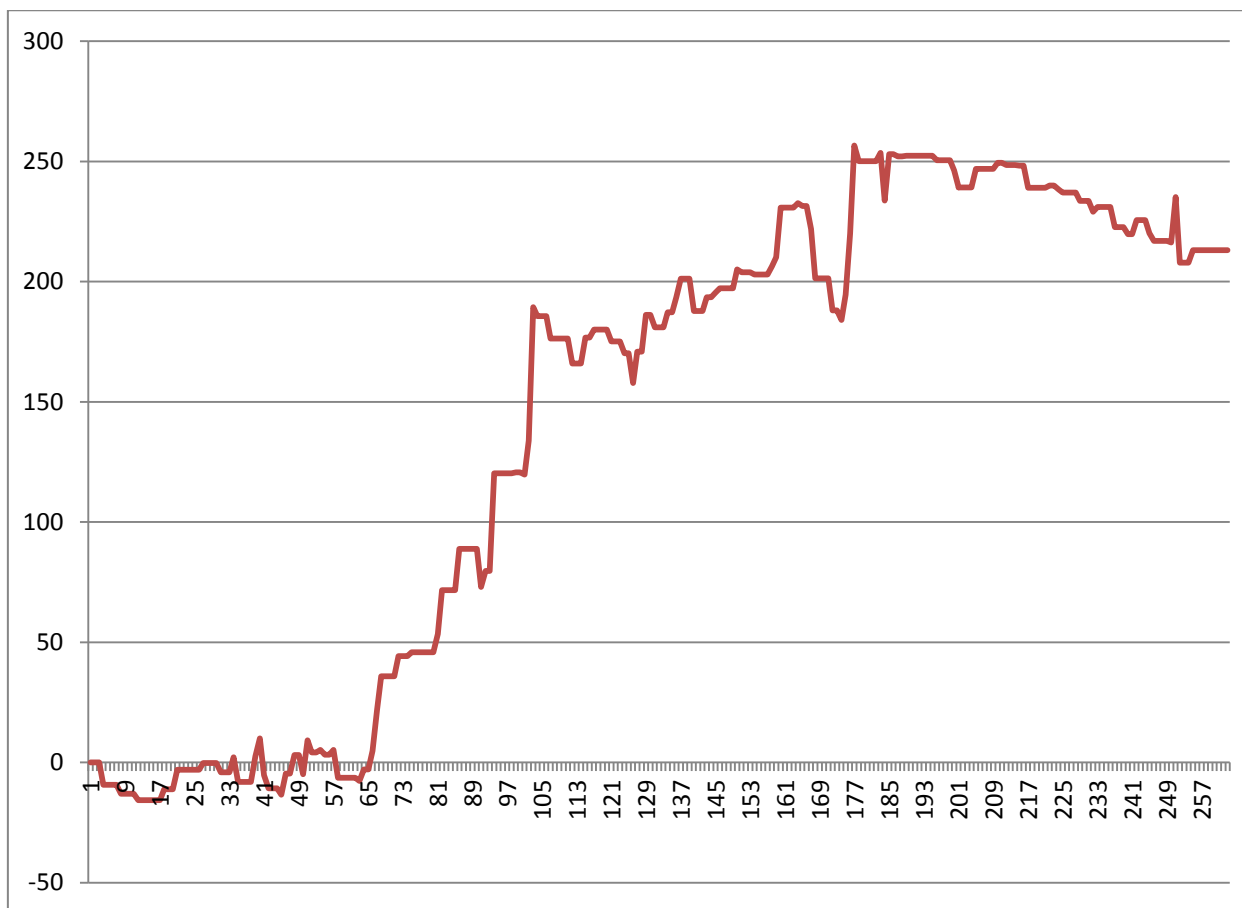


Рисунок 24 - График симуляции торгов за 2012 год

На рисунке видно, что в первые дни торгов прогнозы были преимущественно неудачными, тем не менее, график не ушел сильно вниз и, если бы мы завершили торги, вернув все доллары в рубли, то потеряли бы менее десяти рублей, что составляет около одного процента от вложенной суммы.

Начиная с 65 дня, график вышел из минуса и начал быстро расти. Пик прибыли пришелся на 177 день торгов и составлял 253 российских рубля, что составляет 25,3% от начальной суммы вложений и 15,06% от максимальной суммы вложений, которая составила 3812 рублей.

График прибыли корректно рассматривать в связке с графиком курса пары USD/RUB за 2012 год, который изображен на рисунке 25. При сравнении графиков курса USD/RUB и прибыли видно, что отрицательная прибыль в первые два месяца торгов в основном связана с общим падением рубля.

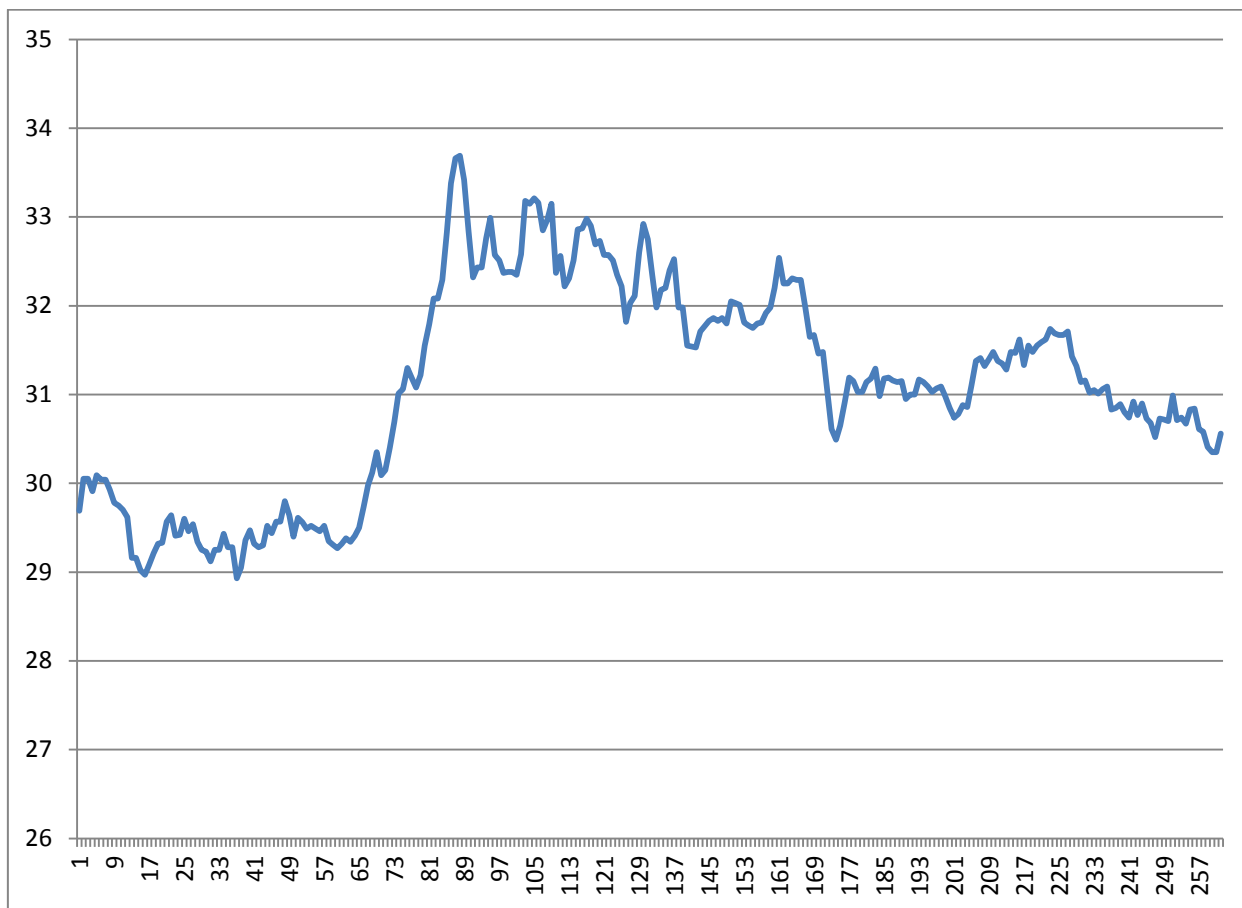


Рисунок 25 - График курса валютной пары USD/RUB за 2012 год

Далее, за счет быстрого роста цены на доллар США, который ИНС удалось предсказать, прибыль с торгов значительно выросла. В заключающей трети года удалось не потерять, а местами даже увеличить прибыль, несмотря на общее падение доллара относительно рубля.

Стоит отметить, что помимо представленной, была произведена симуляция торгов для бюджета, ограниченного одной тысячей рублей. В данном случае стратегия поведения на рынке сохранялась, но добавлялись следующие правила:

- если в наличии меньше тысячи рублей, и предсказано укрепление доллара, то покупка производится на все имеющиеся средства;
- если в наличии больше тысячи рублей, и предсказано укрепление доллара, то покупка производится на одну тысячу рублей;

- если в наличии нет рублей, то покупка долларов не производится вне зависимости от прогноза.

В результате симуляции с ограничением бюджета было выявлено, что такая стратегия приносит меньше прибыли в процентах относительно максимально затраченной на торги суммы.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы была разработана и обучена нейронная сеть, прогнозирующая направление изменения цены валютной пары USD/RUB, основываясь на колебаниях цены данной валютной пары за предыдущие пятнадцать дней, а так же колебаниях пар USD/CHF, USD/JPY, EUR/USD и USD/BRENT.

Обучение данной нейронной сети производилось при помощи генетического алгоритма, и было разделено на три этапа:

1. обучение нескольких ИНС разной структуры на данных, соответствующих курсам валют за 2010 год;
2. моделирование работы ИНС на данных, соответствующих курсам валют за 2011 год;
3. симуляция торгов на рынке Форекс, с использованием данных курсов валют за 2012 год.

Полученная в результате обучения сеть показала хороший результат во время симуляции торгов, выраженный в потенциальной прибыли, составившей 15,06% от вложенной в торги суммы.

В общей сложности, за время исследования было обучено более ста нейронных сетей, которые отличались друг от друга следующими параметрами:

- количеством входов;
- количеством выходных нейронов;
- количеством скрытых слоев нейронов;
- количеством нейронов в каждом из скрытых слоев;
- наличием порогового элемента во входном слое;
- количеством валютных пар во входных данных;
- способом нормализации входных данных и другими.

В ходе исследования различных структур нейронных сетей, отличающихся количеством скрытых слоев и количеством нейронов в них, не было выявлено какой-либо существенной корреляции между структурой ИНС и ее способностью к обучению и предсказанию.

Данная работа является результатом двухгодичного курса обучения магистра и отражает уровень подготовки в рамках заданной темы.

В качестве направлений дальнейших исследований можно выделить:

- реализация прочих способов обучения ИНС;
- реализация структуры ИНС с обратной связью;
- реализация неполносвязной структуры ИНС.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Хайкин С. Нейронные сети: полный курс = Neural Networks: A Comprehensive Foundation. 2-е изд. — М.: Вильямс, 2006. — 1104 с.
2. Вороновский Г. К., Махотило К. В., Петрашев С. Н., Сергеев С. А. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности. — Харьков: Основа, 1997. — 112 с.
3. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечёткие системы. // Д. Рутковская, пер. с польского И. Д. Рудинского. -М.: Горячая линия–Телеком, 2007. — 452 с.
4. Тадеусевич Рышард, Боровик Барбара, Гончаж Томаш, Леппер Бартош. Элементарное введение в технологию нейронных сетей с примерами программ / Перевод И. Д. Рудинского. — М.: Горячая линия — Телеком, 2011. — 408 с.
5. Каллан Р. Основные концепции нейронных сетей = The Essence of Neural Networks First Edition. — М.: Вильямс, 2001. — 288 с.
6. Горбань А.Н. Обучение нейронных сетей. — М.: СССР-США СП «Параграф», 1990. — 160 с.
7. Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика. — М.: Горячая линия - Телеком, 2001. — 382 с.
8. Физиология человека : учеб. / под ред. Н. А. Агаджаняна и В. И. Циркина. СПб.: СОТИС, 1998. —526 с.
9. Ясницкий Л. Н. Введение в искусственный интеллект. — М.: Издат. центр «Академия», 2005. — 176 с.
10. Орлов Ю., Осминин. Нестационарные временные ряды. Методы прогнозирования с примерами анализа финансовых и сырьевых рынков// М.: ИД «Либроком», 2011. — 384 с.

11. Гладков Л.А., Курейчик В.В., Курейчик В.М. Генетические алгоритмы// Под ред. В.М. Курейчика. — 2-е изд., испр. и доп. — М.: ФИЗМАТЛИТ, 2006. — 320 с.
12. Искусственная нейронная сеть.
URL: https://ru.wikipedia.org/wiki/Искусственная_нейронная_сеть
(дата обращения: 31.05.2017)
13. Генетический алгоритм.
URL: https://ru.wikipedia.org/wiki/Генетический_алгоритм
(дата обращения: 31.05.2017)
14. Форекс.
URL: <https://ru.wikipedia.org/wiki/Форекс>
(дата обращения: 31.05.2017)
15. Валютные пары.
URL: https://ru.wikipedia.org/wiki/Валютные_пары
(дата обращения: 31.05.2017)

ИСХОДНЫЙ КОД ПРОГРАММЫ
ПРИЛОЖЕНИЕ А
(обязательное)

Листинг 1. Файл Headers.h

```
#define _CRT_SECURE_NO_WARNINGS

//Подключение всех заголовочных файлов
#include <iostream>
#include <thread>
#include <mutex>
#include <time.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <Windows.h>
#include <vector>
#include <string>
#include <algorithm>
#include <random>
#include <cstdlib>
#include <sstream>
#include <cstring>
using namespace std;

#define INPUTS_NUMBER 35
#define INPUT_FILES_NUMBER 285
#define ALL_INPUT_FILES_NUMBER 285
#define TEST_FILES_NUMBER 271

#include "NN\Input.h"
#include "NN\Synapse.h"
#include "NN\Neuron.h"
#include "NN\NeuronNet.h"

#include "GA\Chromosome.h"
#include "GA\GA.h"

vector<vector<double>>*> ProcessNetForAllInputFiles (NeuronNet*
pNN);
vector<vector<double>>*> ProcessNetForAllInputVectors (NeuronNet*
pNN);
int GetTrend(vector<vector<double>>*> pResults);
void WriteConfiguration(const char* ConfigurationFileName,
NeuronNet* pNN, Chromosome* best);
vector<vector<vector<double>>>*> ReadConfiguration(const char*
ConfigurationFileName);
extern vector<vector<Input*>>*> g_pInputs;
extern vector <Input*>*> g_pExpected;
extern vector <NeuronNet*>*> g_pNNs;
```

Листинг 2. Файл Input.h

```
//Класс для описания входов ИНС
class Input
{
public:
    //Значение входа
    double Value;
};

//Имя Output будет совпадать с Input
typedef Input Output;
```

Листинг 3. Файл Synapse.h

```
//Класс описывающий синапсы (связи) ИНС
class Synapse
{
public:
    //Вес связи
    double Weight;
    //Вход синапса
    Input* pInput;
};
```

Листинг 4. Файл Neuron.h

```
//Класс описывающий нейрон ИНС
class Neuron
{
public:
    Neuron();
    Neuron(vector<Input*>* pInputs, vector<double>* pWeights);
    ~Neuron(){};

    //Функция расчета выходного значения нейрона
    void ProcessNeuron();

    //Функция установки весов синапсов
    void SetWeights(vector<double>* pWeights);

    //Выход
    Output Output;
    vector<Synapse> InputSynapses;
private:
    //Массив входных синапсов (связей)
```

```

//Индукцированное локальное поле
//или потенциал активации
double ActivationPotential;
//Выходное значение
double OutputValue;

//Функция сумматор входных синапсов
void Adder();
//Функция активации
void ActivationFunction();
};

```

Листинг 5. Файл NeuronNet.h

```

//Класс описывающий ИНС
class NeuronNet
{
public:
    NeuronNet(vector<Input*>* pInputs,
vector<vector<vector<double>>>* pWeightsLayers);
    ~NeuronNet() {};

    //Массив входов ИНС
    vector<Input*> Inputs;

    //Первый слой нейронов
    vector<Neuron> FirstNeuronsLayer;

    //Второй слой нейронов
    vector<Neuron> SecondNeuronsLayer;

    //Третий слой нейронов
    vector<Neuron> ThirdNeuronsLayer;

    //Выходной слой нейронов
    vector<Neuron> OutputNeurons;

    //Функция начала работа ИНС
    void ProcessNet();

    //Функция установки новых весов для ИНС
    void SetWeights(vector<vector<vector<double>>>*
pWeightsLayers);

    //Функция установки новых значений входов для ИНС
    void SetInputs(vector<Input*>* pInputs);
};

```

Листинг 6. Файл Neuron.cpp

```
//Реализация функций класса Neuron
#include "../Headers.h"

Neuron::Neuron()
{
    //TODO:
    //Сделать возможность для разных нейронов
    //делать разное количество синапсов
    this->OutputValue = 0;
    this->ActivationPotential = 0;
}

//Конструктор
Neuron::Neuron(vector<Input*>* pInputs, vector<double>*
pWeights)
{
    this->InputSynapses = vector<Synapse>(pInputs->size());

    //Сохраняем соответствующие указатели на входы
    //для каждого синапса и устанавливаем веса
    for (int i = 0; i < pInputs->size(); i++)
    {
        this->InputSynapses[i].pInput = (*pInputs)[i];
        this->InputSynapses[i].Weight = (*pWeights)[i];
    }

    this->Output.Value = 0;
    this->OutputValue = 0;
    this->ActivationPotential = 0;
}

//Сумматор. Складывает входные сигналы,
//взвешенные относительно соответствующих синапсов нейрона
void Neuron::Adder()
{
    //На всякий случай обнулим значение активационного
    //потенциала
    this->ActivationPotential = 0;

    //Суммируем значение всех входных сигналов,
    //умноженных на соответствующие веса синапса
    for (int i = 0; i < this->InputSynapses.size(); i++)
    {
        this->ActivationPotential +=
            this->InputSynapses[i].Weight * this-
            >InputSynapses[i].pInput->Value;
    }
}

//Функция активации. Ограничивает амплитуду выходного сигнала
нейрона.
```

```

//Будем использовать сигмоидальную функцию.
void Neuron::ActivationFunction()
{
    //Гиперболический тангенс подойдет.
    this->OutputValue = tanh(this->ActivationPotential);
    this->Output.Value = this->OutputValue;
}

void Neuron::ProcessNeuron()
{
    this->Adder();
    this->ActivationFunction();
}

void Neuron::SetWeights(vector<double>* pWeights)
{
    //Сохраняем соответствующие веса для каждого синапса
    for (int i = 0; i < pWeights->size(); i++)
    {
        this->InputSynapses[i].Weight = (*pWeights)[i];
    }
}

```

Листинг 7. Файл NeuronNet.cpp

```

//Реализация функций класса NeuronNet
#include "../Headers.h"

//Конструктор
NeuronNet::NeuronNet(vector<Input*>* pInputs,
vector<vector<vector<double>>>* pWeightsLayers)
{
    //Инициализация вектора входов
    this->Inputs = *pInputs;
    vector<vector<vector<double>>> WeightsLayer =
*pWeightsLayers;

    //Инициализация вектора первого уровня нейронов
    //this->Neurons = vector<Neuron>();

    vector<Input*> FirstLayerOutputs;
    vector<Input*> SecondLayerOutputs;
    vector<Input*> ThirdLayerOutputs;

    //Добавляем нейроны первого слоя
    for (int i = 0; i < WeightsLayer[0].size(); i++)
    {
        Neuron CurrentNeuron = Neuron(&this->Inputs,
&WeightsLayer[0][i]);
        this->FirstNeuronsLayer.push_back(CurrentNeuron);
    }
}

```

```

        //Заполняем вектор выходов первого слоя
        for (int i = 0; i < this->FirstNeuronsLayer.size(); i++)
        {
            FirstLayerOutputs.push_back(&(this->FirstNeuronsLayer[i].Output));
        }

        //Добавляем нейроны второго слоя
        for (int i = 0; i < WeightsLayer[1].size(); i++)
        {
            Neuron CurrentNeuron = Neuron(&FirstLayerOutputs,
            &WeightsLayer[1][i]);
            this->SecondNeuronsLayer.push_back(CurrentNeuron);
        }

        //Заполняем вектор выходов второго слоя
        for (int i = 0; i < this->SecondNeuronsLayer.size(); i++)
        {
            SecondLayerOutputs.push_back(&(this->SecondNeuronsLayer[i].Output));
        }

        //Добавляем нейроны третьего слоя
        for (int i = 0; i < WeightsLayer[2].size(); i++)
        {
            Neuron CurrentNeuron = Neuron(&SecondLayerOutputs,
            &WeightsLayer[2][i]);
            this->ThirdNeuronsLayer.push_back(CurrentNeuron);
        }

        //Заполняем вектор выходов третьего слоя
        for (int i = 0; i < this->ThirdNeuronsLayer.size(); i++)
        {
            ThirdLayerOutputs.push_back(&(this->ThirdNeuronsLayer[i].Output));
        }

        //Инициализируем выходные нейроны с соответствующими связями
        for (int i = 0; i < WeightsLayer[3].size(); i++)
        {
            Neuron CurrentNeuron = Neuron(&ThirdLayerOutputs,
            &WeightsLayer[3][i]);
            this->OutputNeurons.push_back(CurrentNeuron);
        }
    }

void NeuronNet::ProcessNet()
{
    //Первый слой
    for (int i = 0; i < this->FirstNeuronsLayer.size(); i++)
    {
        this->FirstNeuronsLayer[i].ProcessNeuron();
    }
}

```

```

        //Второй слой
        for (int i = 0; i < this->SecondNeuronsLayer.size(); i++)
        {
            this->SecondNeuronsLayer[i].ProcessNeuron();
        }

        //Третий слой
        for (int i = 0; i < this->ThirdNeuronsLayer.size(); i++)
        {
            this->ThirdNeuronsLayer[i].ProcessNeuron();
        }

        for (int i = 0; i < this->OutputNeurons.size(); i++)
        {
            this->OutputNeurons[i].ProcessNeuron();
        }
    }

    void NeuronNet::SetWeights(vector<vector<vector<double>>>*>
    pWeightsLayers)
    {
        vector<vector<vector<double>>> WeightsLayer =
        *pWeightsLayers;

        //Нейроны первого слоя
        for (int i = 0; i < WeightsLayer[0].size(); i++)
        {
            this->FirstNeuronsLayer[i].SetWeights(&WeightsLayer[0][i]);
        }

        //Нейроны второго слоя
        for (int i = 0; i < WeightsLayer[1].size(); i++)
        {
            this->SecondNeuronsLayer[i].SetWeights(&WeightsLayer[1][i]);
        }

        //Нейроны третьего слоя
        for (int i = 0; i < WeightsLayer[2].size(); i++)
        {
            this->ThirdNeuronsLayer[i].SetWeights(&WeightsLayer[2][i]);
        }

        //Выходные нейроны
        for (int i = 0; i < WeightsLayer[3].size(); i++)
        {
            this->OutputNeurons[i].SetWeights(&WeightsLayer[3][i]);
        }
    }

```

```

void NeuronNet::SetInputs(vector<Input*>* pInputs)
{
    //Так как в нейронах хранится указатель на вход,
    //то достаточно поменять текущие значения в векторе входов
    for (int i = 0; i < pInputs->size(); i++)
    {
        this->Inputs[i]->Value = (*pInputs)[i]->Value;
    }
}

```

Листинг 8. Файл Main.cpp

```

#include "Headers.h"

vector<vector<Input*>>* g_pInputs;
vector<Input*>* g_pExpected;
vector<NeuronNet*>* g_pNNs;

//Функция чтения входных сигналов из файла
vector<Input*>* ReadInputs(const char* InputFileName)
{
    //Открываем файл
    FILE* InputFile;
    InputFile = fopen(InputFileName, "r");

    //Читаем количество входов
    int InputsNumber;
    fscanf_s(InputFile, "%d", &InputsNumber);

    vector<Input*>* pInputs = new vector<Input*>(InputsNumber);

    //Создаем вектор входов
    for (int i = 0; i < pInputs->size(); i++)
    {
        (*pInputs)[i] = new Input;
        double Value;
        fscanf_s(InputFile, "%lf", &Value);
        (*pInputs)[i]->Value = Value;
    }

    fclose(InputFile);

    return pInputs;
}

//Функция чтения весов синапсов из конфигурационного файла
vector<vector<vector<double>>>* ReadConfiguration(const char*
ConfigurationFileName)
{
    //Открываем файл
    FILE* InputFile;
    InputFile = fopen(ConfigurationFileName, "r");

```

```

if (InputFile == NULL)
    return NULL;

//Читаем количество входов
int InputsNumber;
fscanf_s(InputFile, "%d", &InputsNumber);

//Читаем количество нейронов первого уровня
int FirstNeuronsLayerNumber=0;
fscanf_s(InputFile, "%d", &FirstNeuronsLayerNumber);

//Читаем количество нейронов второго уровня
int SecondNeuronsLayerNumber=0;
fscanf_s(InputFile, "%d", &SecondNeuronsLayerNumber);

//Читаем количество нейронов третьего уровня
int ThirdNeuronsLayerNumber = 0;
fscanf_s(InputFile, "%d", &ThirdNeuronsLayerNumber);

//Читаем количество выходных нейронов
int OutputNeuronsNumber;
fscanf_s(InputFile, "%d", &OutputNeuronsNumber);

vector<vector<vector<double>>>* pWeightsLayers = new
vector<vector<vector<double>>>();

//Создаем вектор весов синапсов первого слоя нейронов
vector<vector<double>> Weights = vector<vector<double>>();
for (int i = 0; i < FirstNeuronsLayerNumber; i++)
{
    Weights.push_back(vector<double>(InputsNumber));
    for (int j = 0; j < InputsNumber; j++)
    {
        double Weight;
        fscanf_s(InputFile, "%lf", &Weight);
        Weights[i][j] = Weight;
    }
}
pWeightsLayers->push_back(Weights);

//Создаем вектор весов синапсов второго слоя нейронов
Weights = vector<vector<double>>();
for (int i = 0; i < SecondNeuronsLayerNumber; i++)
{
    Weights.push_back(vector<double>(FirstNeuronsLayerNumber));
    for (int j = 0; j < FirstNeuronsLayerNumber; j++)
    {
        double Weight;
        fscanf_s(InputFile, "%lf", &Weight);
        Weights[i][j] = Weight;
    }
}

```

```

pWeightsLayers->push_back(Weights);

//Создаем вектор весов синапсов третьего слоя нейронов
Weights = vector<vector<double>>>();
for (int i = 0; i < ThirdNeuronsLayerNumber; i++)
{

Weights.push_back(vector<double>(SecondNeuronsLayerNumber));
    for (int j = 0; j < SecondNeuronsLayerNumber; j++)
    {
        double Weight;
        fscanf_s(InputFile, "%lf", &Weight);
        Weights[i][j] = Weight;
    }
}
pWeightsLayers->push_back(Weights);

//Создаем вектор весов синапсов выходного нейрона
Weights = vector<vector<double>>>();
for (int i = 0; i < OutputNeuronsNumber; i++)
{

Weights.push_back(vector<double>(ThirdNeuronsLayerNumber));
    for (int j = 0; j < ThirdNeuronsLayerNumber; j++)
    {
        double Weight;
        fscanf_s(InputFile, "%lf", &Weight);
        Weights[i][j] = Weight;
    }
}
pWeightsLayers->push_back(Weights);

return pWeightsLayers;
}

//Функция записи весов синапсов
void WriteConfiguration(const char* ConfigurationFileName,
NeuronNet* pNN, Chromosome* best)
{
    FILE* pFile;
    pFile = fopen(ConfigurationFileName, "w");

    int FirstNeuronsLayerNumber = pNN->FirstNeuronsLayer.size();
    int SecondNeuronsLayerNumber = pNN->SecondNeuronsLayer.size();
    int ThirdNeuronsLayerNumber = pNN->ThirdNeuronsLayer.size();
    int InputsNumber = pNN->Inputs.size();
    int OutputNeuronsNumber = pNN->OutputNeurons.size();

    fprintf(pFile, "%d %d %d %d %d\n", InputsNumber,
FirstNeuronsLayerNumber, SecondNeuronsLayerNumber,
ThirdNeuronsLayerNumber, OutputNeuronsNumber);
}

```

```

int point = 0;

for (int i = 0; i < FirstNeuronsLayerNumber; i++)
{
    for (int j = 0; j < InputsNumber; j++)
    {
        fprintf(pFile, "%lf ", best->alleles[point]);
        point++;
    }
    fprintf(pFile, "\n");
}

fprintf(pFile, "\n");

for (int i = 0; i < SecondNeuronsLayerNumber; i++)
{
    for (int j = 0; j < FirstNeuronsLayerNumber; j++)
    {
        fprintf(pFile, "%lf ", best->alleles[point]);
        point++;
    }
    fprintf(pFile, "\n");
}

fprintf(pFile, "\n");

for (int i = 0; i < ThirdNeuronsLayerNumber; i++)
{
    for (int j = 0; j < SecondNeuronsLayerNumber; j++)
    {
        fprintf(pFile, "%lf ", best->alleles[point]);
        point++;
    }
    fprintf(pFile, "\n");
}

fprintf(pFile, "\n");

for (int i = 0; i < OutputNeuronsNumber; i++)
{
    for (int j = 0; j < ThirdNeuronsLayerNumber; j++)
    {
        fprintf(pFile, "%lf ", best->alleles[point]);
        point++;
    }
    fprintf(pFile, "\n");
}

fclose(pFile);
}

```

//Функция запуска сети для входного файла

```

void ProcessNetForInputFile(NeuronNet* pNN, const char*
InputFileName)
{
    vector <Input*>* pInputs;
    pInputs = ReadInputs(InputFileName);

    pNN->SetInputs(pInputs);

    pNN->ProcessNet();
    delete(pInputs);
}

//Функция запуска сети для входного вектора
void ProcessNetForInputVector(NeuronNet* pNN, vector <Input*>*
pInputs)
{
    pNN->SetInputs(pInputs);

    pNN->ProcessNet();
}

//Функция запуска сети для всех входных файлов
vector<vector<double>>* ProcessNetForAllInputFiles(NeuronNet*
pNN)
{
    FILE* f;
    f = fopen("2-5-5_2011_2.txt", "w");
    vector<vector<double>>* pResults = new
vector<vector<double>>();

    for (int i = 1; i <= TEST_FILES_NUMBER; i++)
    {
        char intStr[5] = { 0 };
        _itoa(i, intStr, 10);
        string fileName = intStr;
        fileName = "Files\\Distributed 2011\\" + fileName +
"_2011_distr.txt";
        ProcessNetForInputFile(pNN, fileName.c_str());

        vector<double> Result = vector<double>(5);
        Result[0] = pNN->OutputNeurons[0].Output.Value;
        Result[1] = pNN->OutputNeurons[1].Output.Value;
        Result[2] = pNN->OutputNeurons[2].Output.Value;
        Result[3] = pNN->OutputNeurons[3].Output.Value;
        Result[4] = pNN->OutputNeurons[4].Output.Value;
        pResults->push_back(Result);

        fprintf(f, "%lf\t%lf\t%lf\t%lf\t%lf\n", Result[0],
Result[1], Result[2], Result[3], Result[4]);
    }
    fclose(f);

    return pResults;
}

```

```

}

//Функция запуска сети для всех векторов входов
vector<vector<double>>*> ProcessNetForAllInputVectors (NeuronNet*
pNN)
{
    vector<vector<double>>*> pResults = new
vector<vector<double>>*>();

    for (int i = 0; i < g_pInputs->size(); i++)
    {
        ProcessNetForInputVector (pNN, (*g_pInputs)[i]);

        vector<double> Result = vector<double>(5);
        Result[0] = pNN->OutputNeurons[0].Output.Value;
        Result[1] = pNN->OutputNeurons[1].Output.Value;
        Result[2] = pNN->OutputNeurons[2].Output.Value;
        Result[3] = pNN->OutputNeurons[3].Output.Value;
        Result[4] = pNN->OutputNeurons[4].Output.Value;
        pResults->push_back(Result);
    }

    return pResults;
}

vector<vector<Input*>>*> ReadAllInputFilesToMemory (bool Learn)
{
    vector<vector<Input*>>*> pInputs = new
vector<vector<Input*>>*>();

    int from;
    int to;

    if (Learn)
    {
        from = 1;
        to = INPUT_FILES_NUMBER;
    }
    else
    {
        from = INPUT_FILES_NUMBER;
        to = ALL_INPUT_FILES_NUMBER;
    }

    for (int i = from; i <= to; i++)
    {
        char intStr[5] = { 0 };
        _itoa(i, intStr, 10);
        string fileName = intStr;
        fileName = "Files\\Distributed\\" + fileName +
"_distr.txt";
        pInputs->push_back(ReadInputs(fileName.c_str()));
    }
}

```

```

        return pInputs;
    }

    //Функция чтения результатов
    vector<Input*>* ReadExpected(const char* InputFileName, bool
Learn)
    {
        //Открываем файл
        FILE* InputFile;
        InputFile = fopen(InputFileName, "r");

        vector<Input*>* pInputs = new
vector<Input*>(ALL_INPUT_FILES_NUMBER+1);
        vector<Input*>* pInput = new vector<Input*>(0);
        //Создаем вектор входов
        for (int i = 0; i < pInputs->size(); i++)
        {
            (*pInputs)[i] = new Input;
            double Value;
            fscanf_s(InputFile, "%lf", &Value);
            (*pInputs)[i]->Value = Value;
        }

        fclose(InputFile);
        if (!Learn)
        {
            pInput->insert(pInput->end(), pInputs-
>begin()+INPUT_FILES_NUMBER, pInputs->end());
            return pInput;
        }

        return pInputs;
    }

    int GetTrend(vector<vector<double>>* pResults)
    {
        double FastDown;
        double Down;
        double Up;
        double FastUp;
        double NoChange;
        double result;
        int trend=-1;
        int trend2 = -1;
        int fitness = 0;
        int trendCount = 0;

        for (int i = 0; i < pResults->size(); i++)
        {
            trend = -1;
            FastDown = (*pResults)[i][0];
            Down = (*pResults)[i][1];

```

```

Up = (*pResults)[i][2];
FastUp = (*pResults)[i][3];
NoChange = (*pResults)[i][4];

result = (*g_pExpected)[i]->Value;

/*if (result == 4)
{
    //fitness++;
}
else
{
    if ((FastDown > 0 || Down > 0) && (Up < 0 &&
FastUp < 0))
    {
        trend = 0;
        trend2 = 1;
    }

    else if ((FastUp > 0 || Up > 0) && (Down < 0 &&
FastDown < 0))
    {
        trend = 2;
        trend2 = 3;
    }
}

if (trend == result || trend2 == result)
    fitness++;*/

double trueValue = 0.0;
double falseValue = 0.0;
bool bTrend = false;

//Прогноз по 5 направлениям
if (FastDown > trueValue && Down <= falseValue && Up <=
falseValue && FastUp <= falseValue && NoChange <= falseValue)
{
    bTrend = true;
    trendCount++;
    trend = 0;
}

else if (Down > trueValue && FastDown <= falseValue &&
Up <= falseValue && FastUp <= falseValue && NoChange <=
falseValue)
{
    bTrend = true;
    trendCount++;
    trend = 1;
}

```

```

        else if (Up > trueValue && Down <= falseValue &&
FastDown <= falseValue && FastUp <= falseValue && NoChange <=
falseValue)
        {
            bTrend = true;
            trendCount++;
            trend = 2;
        }
        else if (FastUp > trueValue && Down <= falseValue && Up
<= falseValue && FastDown <= falseValue && NoChange <=
falseValue)
        {
            bTrend = true;
            trendCount++;
            trend = 3;
        }
        else if (NoChange > trueValue && FastUp <= falseValue
&& Down <= falseValue && Up <= falseValue && FastDown <=
falseValue)
        {
            bTrend = true;
            trendCount++;
            trend = 4;
        }

    if (trend == result)
    {
        if (!bTrend)
        {
            fitness = fitness;
        }
        else
        {
            fitness++;
        }
    }

    /*if (FastDown>0 && Down<0 && Up<0 && FastUp<0 &&
NoChange<0)
    {
        trend = 0;
    }

    else if (Down>0 && FastDown<0 && Up<0 && FastUp<0
&& NoChange<0)
    {
        trend = 1;
    }
    else if (Up>0 && Down<0 && FastDown<0 && FastUp<0
&& NoChange<0)

```

```

        {
            trend = 2;
        }
        else if (FastUp>0 && Down<0 && Up<0 && FastDown<0
&& NoChange<0)
        {
            trend = 3;
        }
        else if (NoChange > 0 && FastUp < 0 && Down < 0 &&
Up < 0 && FastDown < 0 )
        {
            trend = 4;
        }

        if (trend == result)
            fitness++;*/

    }

    return fitness;
}

void StartLearning(NeuronNet* pNN)
{
    Chromosome chromosome;
    GA glb = GA();

    int AllAlleles = pNN->Inputs.size() * pNN-
>FirstNeuronsLayer.size()
        + pNN->FirstNeuronsLayer.size() * pNN-
>SecondNeuronsLayer.size()
        + pNN->SecondNeuronsLayer.size() * pNN-
>ThirdNeuronsLayer.size()
        + pNN->ThirdNeuronsLayer.size() * pNN-
>OutputNeurons.size();

    glb.SetMaxIteration(INT_MAX);
    glb.SetMaxPopulation(40);
    glb.SetAllelesSize(AllAlleles);
    glb.SetMutatePercent(1);
    glb.SetMutateStep(10);
    glb.SetMutatePercentNewGenom(100);
    glb.SetCrossTPercent(10, 20, 10, 10, 50);
    glb.SetMaxValue(1);
    glb.SetElitePercent(60);

    glb.SetPrecision(0.01);
    glb.SetResult(135.5);

    int ret = glb.Run(pNN);

```

```

printf("Epoch %d\n", glb.GetEpoch());

if (ret == -1)
{
    printf("Not Found, Best Chromosome:\n");
    chromosome = glb.GetChromosome(0);
}
else
{
    chromosome = glb.GetChromosome(ret);
}

printf("Fitness: %f", chromosome.fitness);
printf("\tPercent %d", chromosome.percent);
printf("\tTrue %d from %d\n", chromosome.countTrue,
INPUT_FILES_NUMBER);

int find = 0;

for (int i = 0; i < glb.Population.size(); i++)
{
    if (glb.Population[i].fitness == 0)
        find++;
}

printf("\nFind: %d", find);

_getch();
}

void StartValidate()
{
    g_pInputs = ReadAllInputFilesToMemory(false);

    g_pExpected =
ReadExpected("Files\\ExpectedOutputs.txt", false);

    vector<vector<vector<double>>>* pWeightsLayers;

    pWeightsLayers =
ReadConfiguration("Files\\OutputWeights.txt");
    vector<vector<double>>* pResults;

    (*g_pNNs)[0]->SetWeights(pWeightsLayers);

    pResults = ProcessNetForAllInputVectors((*g_pNNs)[0]);
    int trend = GetTrend(pResults);
    int countTrue = trend;
    int fitness = abs(trend - (ALL_INPUT_FILES_NUMBER-
INPUT_FILES_NUMBER));
    int percent = (double)trend / (ALL_INPUT_FILES_NUMBER -
INPUT_FILES_NUMBER) * 100;

```

```

        printf("Fitness: %f", fitness);
        printf("\tPercent %d", percent);
        printf("\tTrue %d from %d\n", countTrue,
(ALL_INPUT_FILES_NUMBER - INPUT_FILES_NUMBER));
    }

int main()
{
    g_pInputs = ReadAllInputFilesToMemory(true);

    g_pExpected =
ReadExpected("Files\\ExpectedOutputs.txt", true);

    SYSTEM_INFO siSysInfo;
    GetSystemInfo(&siSysInfo);

    g_pNNs = new vector<NeuronNet*>;

    vector<void*>* needToFreeInputs = new vector<void *>;
    vector<void*> needToFreeWeights;

    for (int i = 0; i < 4; i++)
    {
        vector<Input*>* pInputs;
        pInputs = ReadInputs("Files\\Input.txt");

        vector<vector<vector<double>>>* pWeightsLayers;
        pWeightsLayers =
ReadConfiguration("Files\\Configuration.txt");

        NeuronNet* pNN = new NeuronNet(pInputs,
pWeightsLayers);
        g_pNNs->push_back(pNN);
        needToFreeInputs->push_back(pInputs);
        needToFreeWeights.push_back(pWeightsLayers);
    }

    //Начало обучения
    StartLearning((*g_pNNs)[0]);

    //ТЕСТ НА 2011
    vector<Input*>* pInputs;
    pInputs = ReadInputs("Files\\Input.txt");

    vector<vector<vector<double>>>* pWeightsLayers;
    pWeightsLayers =
ReadConfiguration("Files\\OutputWeights.txt");

    NeuronNet* pNN = new NeuronNet(pInputs, pWeightsLayers);
    ProcessNetForAllInputFiles(pNN);

    //Тестирование

```

```

//StartValidate();
_getch();
//Освобождение памяти
for (int i = 0; i < pInputs->size(); i++)
{
delete((*pInputs)[i]);
}

delete(g_pExpected);

for (int i = 0; i < g_pInputs->size(); i++)
{
for (int j = 0; j < (*g_pInputs)[i]->size(); j++)
{
delete((*g_pInputs)[i])[j]);
}
}
delete(g_pInputs);

for (int i = 0; i < g_pNNs->size(); i++)
{
delete((*g_pNNs)[i]);
}
delete(g_pNNs);

return 0;
}

```